



**Universitatea Tehnică „Gheorghe Asachi” din  
IAȘI**



**Facultatea de Automatică și Calculatoare**

**Domeniul: *Calculatoare și Tehnologia Informației***

**Specializarea: *Calculatoare/Tehnologia Informației***

## **EcoConnect**

### **Aplicație web de eco-gestiune**

**Lucrare de Licență**

Absolvent  
Vasile-Cosmin RESMERITĂ

Coordonator științific  
Ş.l. dr. Iulian PETRILA

**Iași, 2024**

**DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII**  
**PROIECTULUI DE DIPLOMĂ**

Subsemnatul Resmerită Vasile-Cosmin,

legitimat cu CI seria NZ nr. 181090, CNP 5010927271694

autorul lucrării EcoConnect – Aplicație de eco-gestiune

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii Calculatoare și tehnologia informației, specializarea Tehnologia informației organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași,

sesiunea iulie a anului universitar 2023-2024, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

**Data**

**Semnătura**



# EcoConnect – aplicație de eco-gestiune

Resmeriță Vasile-Cosmin

## Rezumat

Lucrarea își propune dezvoltarea unei aplicații web, în care să faciliteze procesul de reciclare și să promoveze un comportament mai ecologic atât în rândul utilizatorilor, cât și al firmelor de acest tip. Aplicația este concepută pentru a oferi utilizatorilor o experiență ușoară și eficientă în acest proces, furnizând informații detaliate despre modul corect de reciclare. Inițial, aplicația va dispune de două tipuri de conturi distințe: unul pentru persoane fizice și unul pentru persoane juridice, care reprezintă firmele de reciclare. Fiecare entitate introduce date personale și informații de localizare relevante. Atunci când un utilizator dorește să recicleze un produs sau o cantitate specifică de produse similare (de exemplu, plastic sau metal) creează un request, completând anumite căsuțe, pe care le poate edita sau șterge până la trimiterea requestului. Utilizatorul poate vedea toate cererile realizate de el.

Aplicația va direcționa cererile utilizatorului la cele mai apropiate firme specializate în reciclarea respectivului tip de material pe o rază definită, facilitând astfel procesul de reciclare și promovând colaborarea între comunitatea de utilizatori și industria reciclarii. Utilizatorul firmei observă pe hartă că există o cerere în apropiere, apasă pe pin-ul respectiv, apoi vede toate detaliile cererii, unde poate accepta. Firmele pot observa toate requesturile acceptate, iar după colectare acestea trebuie să finalizeze cererea pentru a adapta statusul acesteia. Aceasta poate observa și detaliile firmelor din raza de acțiune, cât și cererile care sunt deja acceptate sau în curs de procesare.

Backendul este realizat în IDE-ul IntelliJ folosind framework-ul SpringBoot cu dependențele necesare în Maven. Backendul comunică cu baza de date PostgreSQL cu ajutorul Hibernate. Pentru frontend s-a utilizat IDE-ul Visual Studio Code utilizând limbajul de programare React.

În frontend se utilizează un API de la Google Maps pentru a extrage coordonatele geografice ale utilizatorilor, pentru realizarea distanței dintre două puncte, algoritm creat cu ajutorul formulei Haversine.

Câteva din avantajele folosirii acestei aplicații sunt: toate informațiile esențiale despre firmele de reciclare, tipurile de materiale reciclate și locațiile centrelor sunt adunate într-un singur loc; fiind o rază mică de acoperire a preluării deșeurilor, firmele de reciclare pot planifica rute mai eficiente pentru colectare, economisind timp și resurse; în plus o interfață cât mai prietenoasă și minimalistă.

# Cuprins

<b>Introducere.....</b>	<b>1</b>
<b>Capitolul 1. Fundamentarea teoretică și tehnologiile utilizate.....</b>	<b>3</b>
<b>1.1. Prezentare a companiilor care au abordat aceeași tema.....</b>	<b>4</b>
<b>1.2. Tehnologii utilizate.....</b>	<b>6</b>
<b>1.2.1. Algoritmi utilizați.....</b>	<b>6</b>
<b>1.2.2. Google Maps API.....</b>	<b>10</b>
<b>1.2.3. Hibernate.....</b>	<b>11</b>
<b>Capitol 2. Proiectarea aplicației.....</b>	<b>13</b>
<b>2.1. Prezentarea aplicației.....</b>	<b>13</b>
<b>2.2. Resurse Software.....</b>	<b>14</b>
<b>2.2.1. Baza de Date PostgreSQL.....</b>	<b>14</b>
<b>2.2.2. Backend SpringBoot, IntelliJ.....</b>	<b>15</b>
<b>2.2.3. Frontend React, Visual StudioCode.....</b>	<b>17</b>
<b>2.2.4. Comunicare server-client.....</b>	<b>19</b>
<b>2.2.4.1. Protocol HTTP.....</b>	<b>19</b>
<b>2.2.4.2. Configurare CORS.....</b>	<b>20</b>
<b>Capitolul 3. Implementarea aplicației.....</b>	<b>21</b>
<b>3.1. Descrierea generală a aplicației.....</b>	<b>21</b>
<b>3.2. Funcționalitatea aplicației.....</b>	<b>22</b>
<b>Capitolul 4. Testarea aplicației și completări viitoare ale aplicației.....</b>	<b>35</b>
<b>4.1. Testarea aplicației.....</b>	<b>35</b>
<b>4.1.1. Testarea aplicației cu firme reale din Iași.....</b>	<b>35</b>
<b>4.2. Completări viitoare ale aplicației.....</b>	<b>36</b>
<b>4.2.1. Comunicarea între client și companie.....</b>	<b>36</b>
<b>4.2.2. Securizarea aplicației.....</b>	<b>37</b>
<b>Concluzie.....</b>	<b>38</b>
<b>Bibliografie.....</b>	<b>39</b>
<b>Anexe.....</b>	<b>40</b>
<b>Anexa1 – SpringBoot.....</b>	<b>40</b>
<b>Anexa2 – React.....</b>	<b>48</b>

## Introducere

Într-o lume în care resursele naturale sunt tot mai limitate, iar impactul activităților umane asupra mediului devine din ce în ce mai vizibil, importanța reciclării nu a fost niciodată mai mare. Reciclarea este esențială pentru conservarea resurselor, reducerea poluării și protejarea ecosistemelor fragile de pe întreaga planetă. Fiecare gest mic, fiecare decizie conștientă de a recicla, contribuie la un viitor mai verde și mai sustenabil.

Reciclarea nu este un concept nou; de fapt, are rădăcini adânci în istoria omenirii. Primele dovezi ale reciclării datează din perioada preistorică, când oamenii reutilizau uneltele și materialele pentru a maximiza utilizarea resurselor limitate. În Evul Mediu, fierul și alte metale erau adesea topite și refolosite. Revoluția Industrială a marcat o schimbare semnificativă, odată cu creșterea producției și a consumului, generând nevoie de gestionare a deșeurilor. În secolul XX, reciclarea a început să capete o importanță deosebită, în special în timpul și după cele două războaie mondiale, când resursele erau extrem de limitate și reutilizarea materialelor a devenit crucială. În anii '70, mișcarea ecologistă a adus reciclarea în prim-planul conștiinței publice, iar de atunci, numeroase guverne și organizații au implementat programe de reciclare pe scară largă.

În discursul său istoric de Ziua Pământului, pe 22 aprilie 1970, Gaylord Nelson a subliniat necesitatea de a aborda problemele de mediu în contextul mai larg al inegalităților sociale și economice. El a declarat: „Pământul nu ne-a fost dat de la părinții noștri, ci împrumutat de la copiii noștri.” Nelson a vorbit despre importanța unei noi etici americane care să pună accentul pe demnitatea umană și bunăstarea, mai degrabă decât pe o tehnologie nesfârșită care produce mai multe deșeuri și poluare. El a cerut o revoluție în politicile naționale pentru a proteja mediul și pentru a asigura un viitor de calitate pentru toți.[1]

Un alt moment istoric important în conștientizarea ecologică este discursul legendar al lui Chief Seattle, care a oferit o profundă lecție de viață americanilor cu privire la respectul pentru natură și interconectivitatea tuturor formelor de viață. În discursul său din 1854, Chief Seattle a spus: „Pământul nu aparține omului; omul aparține Pământului.” El a subliniat că tot ceea ce se întâmplă cu Pământul afectează inevitabil oamenii, deoarece toate lucrurile sunt legate între ele. Chief Seattle a îndemnat la respectarea naturii și la înțelegerea faptului că distrugerea mediului va duce la propria noastră distrugere. Aceste cuvinte rezonează și astăzi, amintindu-ne de responsabilitatea noastră de a proteja planeta.[2]

În momentul de față, reciclarea se află la un punct de cotitură. Deși multe comunități și țări au făcut pași importanți în adoptarea unor practici de reciclare mai eficiente, există încă provocări majore. Gradul de conștientizare este variabil, infrastructura de reciclare este adesea insuficientă, iar lipsa de informații precise poate duce la confuzii și la practici incorecte. În plus, problema gestionării deșeurilor este accentuată de creșterea constantă a consumului și a deșeurilor generate. Reciclarea se concentrează în principal pe reducerea depozitelor de gunoi și a emisiilor de gaze cu efect de seră, contribuind la campania de protejare a mediului de schimbările climatice. Dar aceasta ar trebui implementată energetic, cu infrastructuri bine stabilite și educație riguroasă pentru ca materialele să fie sortate și procesate eficient.

Reciclarea a fost abordată prin diverse inițiative și tehnologii de-a lungul timpului. Această tranziție de la metodele vechi, care în cele mai multe cazuri erau corelate cu metodologii manuale și logistica dificilă, a adus creșterea aplicațiilor mobile și a platformelor digitale. Abordând deficiențele acestor aplicații, EcoConnect se bazează în continuare pe o rețea de companii de reciclare, prin care utilizatorii își pot trimite cererile direct din aplicație, venind cea mai apropiată firmă de reciclare pentru a prelua deșeurile. Procesul asigură reciclarea corectă și eficientă a deșeurilor de către firmele din domeniu. Cu ajutorul tehnologiei de geolocalizare, EcoConnect poate identifica cea mai apropiată companie specializată în tipul de material menționat de utilizator pe o distanță definită în funcție de oraș. Acest lucru aduce mult mai multă optimizare în procesul de reciclare și eliberează utilizatorii de a depune mult timp și efort. Aplicația implementează o tehnologie rezonabilă pentru gestionarea deșeurilor, care rezolvă problemele contemporane și contribuie la atingerea unui loc curat și durabil.

## Capitolul 1. Fundamentarea teoretică și tehnologiile utilizate

Gestionarea adecvată a deșeurilor care pot fi reciclate a devenit o preocupare mondială în contextul actual. Creșterea populației și consumul de resurse sunt responsabile de producerea unor cantități mari de deșeuri, prin urmare sunt necesare soluții durabile în acest domeniu. Fiind o parte importantă a sustenabilității, reciclarea reduce impactul asupra mediului prin reutilizarea materialelor și reducerea resurselor naturale. Cu toate acestea, punerea în aplicare și practica bine informată a reciclării corecte nu au fost încă pe deplin realizate.

O economie circulară se referă la o economie care este de natură regenerabilă. Acest lucru asigură reutilizarea, reprocesarea și reciclarea materialelor și produselor în comparație cu economia liniară tradițională bazată pe un model „a lua, a face, a arunca”. Acest lucru asigură promovarea unui sistem în buclă închisă, minimizând risipa și menținând resursele în funcțiune pe termen nelimitat.

În ceea ce privește reciclarea, în UE s-au înregistrat în medie 249 kg de deșeuri reciclate per persoană, o scădere față de media din 2021. Statele cu cele mai mari cantități de deșeuri reciclate per persoană au fost Austria, Danemarca și Germania, în timp ce cele mai mici cantități au fost înregistrate în România, Malta și Grecia. [3] România este la coada Europei când vine vorba de reciclare.

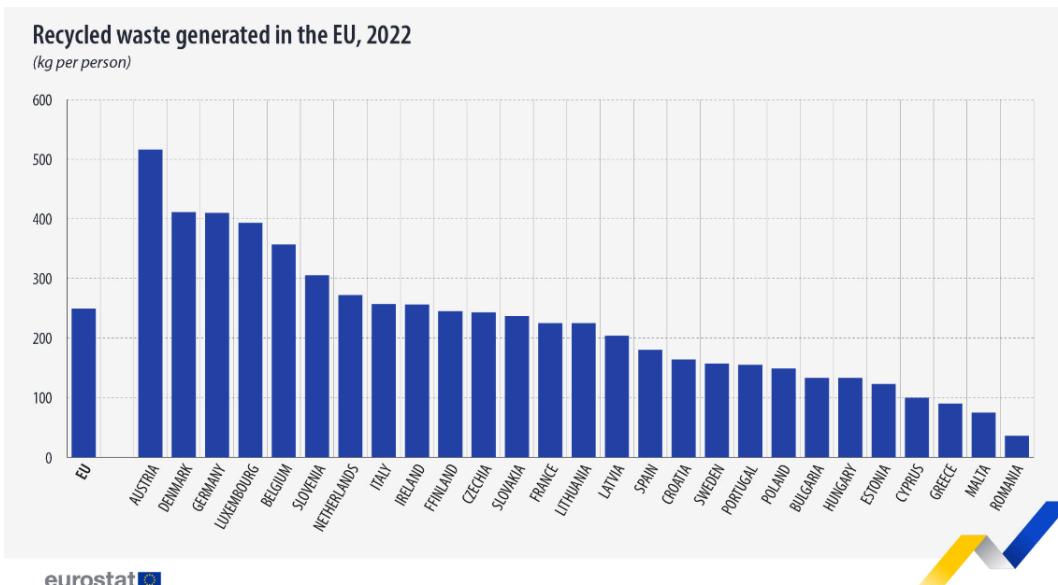


Figura 1.1. Grafic UE<sup>1</sup>

<sup>1</sup> Source dataset: [env\\_wasmun](#)

Cu alte cuvinte, colectarea selectivă este un sistem în care deșeurile sunt sortate la sursă în funcție de materialul din care provin, pentru a fi separate eficient în vederea reciclării ulterioare. Mai exact, depinde în principal de dorința și de conștientizarea gospodăriilor și a întreprinderilor de a coopera, astfel încât deșeurile să poată fi prelucrate cu ușurință în instalațiile de reciclare. Instalațiile de sortare utilizează senzori optici, raze X și alte tehnologii avansate în clasificarea materialelor reciclabile. În acest fel, sortarea devine mai precisă și accelerează procesarea acestor materiale pentru a fi preluate în mod corespunzător. Reciclarea chimică descompune materialele până la cele mai de bază componente chimice, care sunt apoi folosite pentru a fabrica noi materiale. Acest lucru se dovedește benefic pentru materialele care sunt dificil de reciclat prin metode convenționale. Reciclarea mecanică constă în prelucrarea materialului în vederea reutilizării într-un mod pragmatic. Metodele de reciclare mecanică implică mărunțirea, topirea și modelarea materialelor în produse noi.

Aplicațiile mobile și platformele digitale vor fi instrumente esențiale pentru a face reciclarea mai eficientă și mai disponibilă. Aceste tehnologii vor permite accesul la informații referitoare la reciclare, la locațiile punctelor de colectare și la serviciile de colectare la domiciliu; exemple notabile sunt aplicațiile Recapp dezvoltate de Veolia și Recycle Nation. Astfel de tehnologii contribuie la colectarea și prelucrarea eficientă a materialelor reciclabile, reducând astfel impactul negativ asupra mediului. Eficiența centrului de reciclare este, prin urmare, crescută; există o procesare a deșeurilor într-o perioadă scurtă de timp; și, cu ajutorul aplicațiilor de eco-gestiune a deșeurilor, a roboților, se poate produce o reducere a costurilor. Automatizarea permite ca deșeurile să fie manipulate la viteze mari, cu volume mari și rate de reciclare îmbunătățite.

## 1.1. Prezentare a companiilor care au abordat aceeași tema

Realizăm că 80% dintre consumatorii din întreaga lume doresc să ajute la salvarea planetei pentru generațiile viitoare. În această lume cu o conștientizare ecologică crescută, reciclarea contribuie semnificativ la păstrarea resurselor naturale și prevenirea poluării. Există mai multe aplicații și platforme care tratează subiectul reciclării și fac întregul proces foarte ușor de utilizat.

Printre acestea se numără Recapp și Recycle Nation. Grupul Veolia [4] se angajează să devină o întreprindere de referință în transformarea ecologică. Veolia Group este o companie multinațională franceză specializată în managementul resurselor și a serviciilor de mediu, având activități în domenii precum gestionarea apei, gestionarea deșeurilor și energia. Veolia oferă soluții pentru gestionarea eficientă a apei, inclusiv tratarea apei potabile, tratarea apelor uzate și managementul rețelelor de apă. Compania oferă soluții energetice durabile, inclusiv producerea de energie regenerabilă, gestionarea eficientă a energiei și reducerea emisiilor de carbon. Veolia este implicată în toate aspectele gestionării deșeurilor, de la colectare și sortare până la reciclare și eliminare sigură. Compania se concentreză pe promovarea economiei circulare, transformând deșeurile în resurse valoroase prin reciclare și reutilizare.

În acest context, a dezvoltat Recapp[5], care oferă servicii de colectare a deșeurilor reciclabile pentru gospodăriile din Emiratele Arabe Unite. Făcând acest lucru, Recapp garantează

că utilizatorii nu vor fi lăsați complet pe cont propriu să transporte deșeurile posibile până la punctele de colectare și vor primi recompense pentru reciclarea regulată. Introducerea acestei infrastructuri avansate de colectare a deșeurilor și instituirea unui sistem de depunere care recompensează reciclarea decurge promovarea economiei circulare și modelele de consum sustenabile care reduc stresul asupra mediului.[6] În plus, Recapp oferă o platformă digitală ușor de utilizat care permite utilizatorilor să se înscrie și să programeze colectarea deșeurilor. Utilizatorii pot urmări activitățile de reciclare prin intermediul aplicației și pot vedea impactul pozitiv pe care îl au asupra mediului. Sistemul de recompense este conceput pentru a motiva utilizatorii să recicleze în mod constant, oferindu-le puncte care pot fi răscumpărate pentru diferite beneficii sau reduceri la produse ecologice. Recapp nu numai că simplifică procesul de reciclare, dar și încurajează un comportament responsabil față de mediu. Avantajul esențial al acestei soluții este comoditatea pentru utilizatori.



Frigura 1.2 Image Recapp<sup>2</sup>

Recycle Nation este o aplicație populară din SUA care se preocupă de educarea utilizatorilor, având un motor de căutare pentru punctele de reciclare. Aplicația oferă instrucțiuni pas cu pas despre cum să reciclezi diferite materiale și să faci înregistrări personale cu privire la activitățile lor de reciclare. Prin intermediul widget-ului interactiv și al instrumentelor API, RecycleNation ajută întreprinderile, consumatorii, entitățile guvernamentale locale, statale și federale, precum și organizațiile non-profit să își îmbunătățească obiectivele și inițiativele de sustenabilitate și de economie circulară. API-ul RecycleNation oferă opțiuni infinite de personalizare pentru a adăuga o căutare personalizată de reciclare pe site-urile web partenere.[7]. Recycle Nation pune un accent puternic pe educație și conștientizare, oferind articole informative și ghiduri pentru a ajuta utilizatorii să înțeleagă importanța reciclării și să adopte

---

<sup>2</sup> Image Recapp <https://www.gorecapp.com/>

practici sustenabile. Aplicația oferă și o secțiune de știri și actualizări legate de reciclare, unde utilizatorii pot afla despre noile tehnologii, legislații și inițiative în domeniul reciclării. De asemenea, Recycle Nation colaborează cu diverse organizații și companii pentru a promova evenimente și campanii de reciclare la nivel național. Prin furnizarea acestor resurse valoroase, Recycle Nation ajută utilizatorii să devină mai conștienți de impactul deșeurilor și îi motivează să ia măsuri concrete pentru a contribui la un mediu mai curat și mai sănătos.[8]



Figura 1.3 Logo Recycle Nation<sup>3</sup>

Aceste platforme nu doar că facilitează procesul de reciclare, dar și educă și motivează utilizatorii să adopte practici ecologice. Ambele platforme subliniază importanța adoptării unui comportament responsabil față de mediu și demonstrează că, prin tehnologie și educație, este posibil să se creeze un impact pozitiv semnificativ asupra mediului.

## 1.2. Tehnologii utilizate

### 1.2.1. Algoritmi utilizați

#### Algoritm pentru calcularea distanței dintre două puncte pe suprafața Pământului

Pământul este rotund, este mare, așa că pe distanțe scurte putem să îl considerăm plat. Cu toate acestea, în timp ce circumferința Pământului este de aproximativ 40.000 de kilometri, formula de calcul a distanței dintre două puncte prezintă erori semnificative la distanțe mai mari de aproximativ 20 de kilometri. Prin urmare, atunci când se calculează distanțe pe o sferă, este necesar să se țină cont de geometria sferei, adică de studiul formei acesteia pe suprafața sferei. Geometria sferică se ocupă de trigonometria sferică. Se ocupă de relațiile dintre funcțiile trigonometrice pentru calcularea laturilor și unghiurilor poligoanelor sferice. Aceste poligoane sferice sunt definite de o serie de cercuri mari care se intersecțează pe sferă.

#### Formula Haversine

Formula Haversine este utilizată pentru a calcula distanța dintre două puncte de pe suprafața Pământului, considerând Pământul ca o sferă. Aceasta este ideală pentru calcularea distanțelor mari și este relativ simplă din punct de vedere matematic. Această expresie este importantă pentru utilizarea în navigație. Expresiile trigonometrice suplimentare sunt versina, haversina, coversina, hacoversina, exsecanta și excosecanta. Toate acestea pot fi exprimate pur și simplu în termeni de funcții trigonometrice mai cunoscute. De exemplu:

---

<sup>3</sup> Logo Recycle Nation <https://recyclenation.com/>

$$\text{haversine}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

Figura 1.4 Haversine<sup>4</sup>

Funcțiile de mai sus datează dintr-o perioadă în care nu existau calculatoare sau procesoare puternice. Pe atunci, oamenii calculau manual unghiurile și direcțiile folosind tabele logaritmice, iar evaluarea fiecărei funcții menționate necesita mult efort. Haversina unghiului central se calculează cu următoarea formulă:

$$\left(\frac{d}{r}\right) = \text{haversine}(\Phi_2 - \Phi_1) + \cos(\Phi_1)\cos(\Phi_2)\text{haversine}(\lambda_2 - \lambda_1)$$

Figura 1.5 Central angle<sup>5</sup>

unde r este raza Pământului egală cu 6371 km.

Formula Haversine este o modalitate foarte precisă de a calcula distanțele dintre două puncte de pe suprafața unei sfere, folosind latitudinea și longitudinea celor două puncte. Formula este o reformulare a legii sferice a cosinusurilor, dar formularea în termeni de haversine este mai utilă pentru unghiuri și distanțe mici. În zilele noastre nu se mai folosesc aceste funcții, se folosesc funcțiile trigonometrice, spre exemplu: [9]

$$d = 2r \arcsin \left( \sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \right)$$

Figura 1.6 Haversine Formula<sup>6</sup>

- $\phi_1$  - latitudinea primului punct
- $\phi_2$  - latitudinea punctului al doilea
- $\lambda_2$  - longitudinea punctului al doilea
- $\lambda_1$  - longitudinea primului punct
- r - raza Pământului (6371 km)

### Formula Vincenty

Formulele lui Vincenty sunt două metode denumite după Thaddeus Vincenty, un geodez american de origine poloneză. Au fost dezvoltate două formule pentru calcularea distanțelor

---

<sup>4</sup> Figura 1.4 Haversine [Haversine Formula](#)

<sup>5</sup> Figura 1.5 Central angle [Haversine Formula](#)

<sup>6</sup> Figura 1.6 Haversine Formula <https://github.com/DaniilSydorenko/haversine-geolocation/issues/1>

geodezice între o pereche de puncte de latitudine/longitudine pe un model elipsoidal al Pământului (sferă oblică). Formulele Vincenty includ două metode: directă și indirectă. Cea directă calculează coordonatele punctului final, iar cea indirectă calculează distanța dintre 2 puncte.

Această geometrie elipsoidală este următorul pas și este probabil cea mai bună geometrie pentru a modela Pământul după ce s-a lucrat cu sferă folosind formula Haversine. Spre deosebire de metoda Haversine, care calculează distanțele pe o suprafață sferică, aceste formule sunt metode iterative care presupun că Pământul este un elipsoid. Formula se bazează pe iterații și include semiaxa mare a elipsoidului (a), aplatizarea (f),  $\Phi_1$ ,  $\Phi_2$  latitudinile punctelor, L1, L2 longitudinile și transformă coordonatele latitudinale. Vincenty folosește metode inverse pentru a rezolva distanța prin calcularea mai multor variabile intermediare. [10]

$$U_1 = \arctan((1-f)\tan\Phi_1),$$

$$U_2 = \arctan((1-f)\tan\Phi_2) \text{ reduce latitudinea pe sferă auxiliară}$$

$\sigma$  - distanța unghiulară dintre puncte

$\sigma_1$  - separația unghiulară dintre punct și ecuator

$\sigma_m$  - separație unghiulară între punctul median al dreptei și ecuator

$$\sin \sigma = \sqrt{(\cos U_2 \sin \lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos \lambda)^2}$$

$$\cos \sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos \lambda$$

$$\sigma = \arcsin(\sin \sigma)$$

$$\sin \alpha = \frac{\cos U_1 \cos U_2 \sin \lambda}{\sin \sigma} \quad [2]$$

$$\cos^2 \alpha = 1 - \sin^2 \alpha$$

$$\cos(2\sigma_m) = \cos \sigma - \frac{2 \sin U_1 \sin U_2}{\cos^2 \alpha} = \cos \sigma - \frac{2 \sin U_1 \sin U_2}{1 - \sin^2 \alpha} \quad [3]$$

$$C = \frac{f}{16} \cos^2 \alpha [4 + f(4 - 3 \cos^2 \alpha)]$$

$$\lambda = L + (1 - C) f \sin \alpha \{ \sigma + C \sin \sigma [\cos(2\sigma_m) + C \cos \sigma (-1 + 2 \cos^2(2\sigma_m))] \}$$

Figura 1.7 Vincenty Formula<sup>7</sup>

În acest caz, este esențial să observăm că algoritmul Vincenty este mai precis decât algoritmul Haversine, deoarece în timp ce algoritmul Vincenty presupune un Pământ nesferic și utilizează o formulă mult mai complicată, algoritmul Haversine rămâne o metodă valoroasă și eficientă de calcul al distanței între două puncte ale Pământului. În mod semnificativ, principala diferență între algoritmii Haversine și Vincenty constă în complexitatea lor computațională: Algoritmul Haversine este relativ simplu și eficient, ceea ce îl face să fie alegerea potrivită pentru lucrările care presupun ca treaba să fie făcută rapid pe seturi mari de date. Pe de altă parte, algoritmul Vincenty este mult mai complicat și mai intensiv din punct de vedere computațional, fiind foarte potrivit pentru aplicații care necesită o precizie mai mare. Cu alte cuvinte, distanța dintre două puncte de pe Pământ poate fi găsită prin algoritmii Haversine și Vincenty: algoritmul

<sup>7</sup> Figura 1.7 Vincenty Formula [Vincenty's formulae](#)

Vincenty este exact, dar costisitor din punct de vedere computațional; în schimb, algoritmul Haversine este simplu și foarte practic. [11]

### Algoritmi pentru criptarea parolei

Criptarea este o parte foarte importantă a securității informaticii moderne. Scopul criptării este de a proteja confidențialitatea datelor, asigurându-se că acestea nu pot fi înțelese de persoane neautorizate chiar dacă sunt interceptate.

AES este aplicat pe scară largă în întreaga lume pentru protejarea datelor sensibile datorită rezistenței și eficienței sale. AES este un algoritm de criptare simetrică și un cifru bloc. Criptarea simetrică înseamnă că utilizează aceeași cheie pentru a cripta și decripta datele. Destinatarul și expeditorul trebuie să cunoască, să folosească aceeași cheie secretă de criptare. Acest lucru face ca AES să fie diferit de algoritmii asimetrici, în care se folosesc chei diferite pentru criptarea și decriptarea datelor. [12]

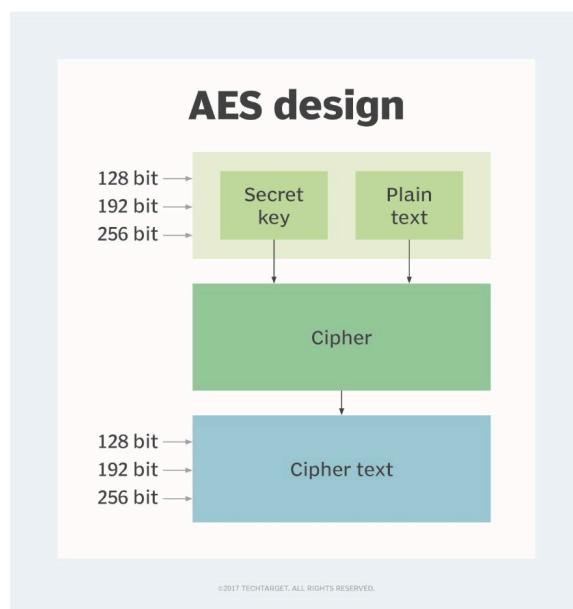


Figura 1.8 AES<sup>8</sup>

SHA-256 este o funcție de hash criptografică care produce o valoare de 256 de biți. În forma criptată, datele au dimensiuni nelimitate și au adesea aceeași lungime ca datele necriptate. În schimb, hashing-ul transformă datele de dimensiuni arbitrară în date de dimensiuni fixe. Verificarea parolelor este esențială pentru securitatea informațiilor în aplicațiile moderne, iar utilizarea hash-urilor criptografice joacă un rol crucial în acest proces. Stocarea parolelor utilizatorilor în formă de text simplu reprezintă un risc major de securitate, deoarece un atacator care obține acces la aceste fișiere ar descoperi imediat toate parolele neprotejate. În schimb, este mult mai sigur să stocăm valorile hash ale parolelor. Atunci când un utilizator introduce o parolă, se calculează valoarea hash a acesteia, care este apoi comparată cu hash-urile stocate în baza de

<sup>8</sup> Figura 1.8 AES [AES uses 128-, 192- or 256-bit keys to encrypt and decrypt data.](#)

date. Dacă valoarea hash calculată se potrivește cu una dintre valorile hash stocate, parola este considerată validă și utilizatorul primește accesul. [13]

### 1.2.2. Google Maps API

Google Maps este un serviciu web care oferă informații detaliate despre regiuni geografice și situri din întreaga lume. Oferă imagini din satelit, fotografii aeriene, hărți stradale, condiții în timp real și planificarea traseelor pentru călătorii pe jos, cu mașina, cu bicicleta, cu avionul sau cu mijloacele de transport în comun. Hărțile oferă o vedere cuprinzătoare și detaliată a lumii, caracteristicile includ vederi din satelit, hărți ale terenului și hărți stradale. Funcționalitate "Street View" permite utilizatorilor să vizualizeze și să navigheze prin imagini la 360 de grade la nivel de stradă, este utilă pentru explorarea virtuală a orașelor și a cartierelor.

API-ul JavaScript Maps vă permite să personalizați hărțile cu conținut și imagini proprii pentru a le afișa pe paginile web și pe dispozitivele mobile. Aceasta oferă patru tipuri de hărți de bază (rutieră, satelit, hibrid și teren) pe care pot fi personalizate folosind straturi și stiluri, controale și evenimente, precum și diverse servicii și biblioteci.[14]

JavaScript Maps API acceptă crearea unor markeri personalizați și afișarea lor pe hartă. Cu aceste markere se pot extrage cu ajutorul Geocoding API, un API tot a Google Maps, coordonatele punctului( latitudinea și longitudinea) și conversia coordonatelor într-o adresă lizibile de către om.

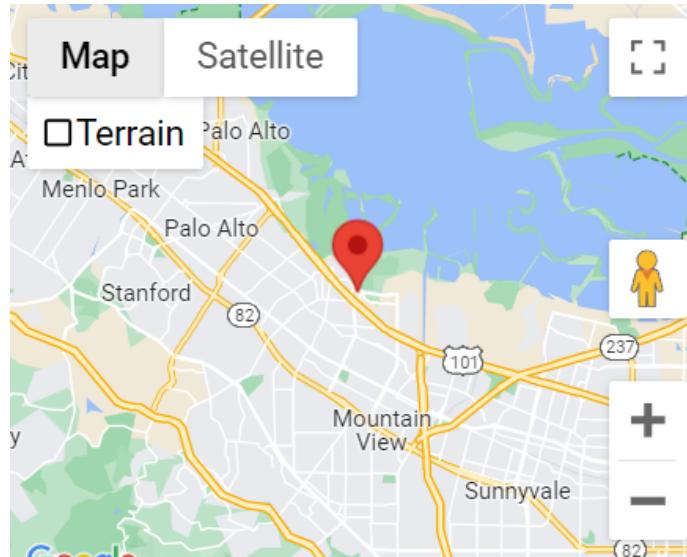


Figura 1.9. Google Maps<sup>9</sup>

---

<sup>9</sup> Figura 1.9. [Google Maps](#)

### 1.2.3. Hibernate

Hibernate este un cadru Java care simplifică dezvoltarea aplicațiilor Java care interacționează cu bazele de date. Este un instrument ORM (Object-Relational Mapping) cu sursă deschisă ușor de utilizat. Hibernate implementează specificația Java Persistence API (JPA) pentru persistența datelor. Java Persistence API (JPA) este o caracteristică Java care oferă anumite funcționalități și standarde pentru instrumentele ORM. Instrumentele ORM simplifică crearea, editarea și accesul la date, gestionând operațiile CRUD într-un mod fiabil și atomic. ORM este o tehnică de programare care mapează obiectele cu datele stocate într-o bază de date. Aceasta este realizată prin utilizarea fișierelor de configurare XML sau a anotărilor Java. [15]

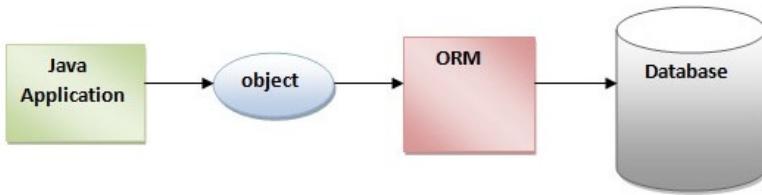


Figura 1.10. Hibernate Diagram<sup>10</sup>

ORM reprezintă un mecanism de conversie automată a datelor între modelul obiect al aplicației Java și modelul relațional al bazei de date. Hibernate este un exemplu cunoscut de ORM pentru Java.

<sup>10</sup> Figura 1.10. <https://www.javatpoint.com/hibernate-tutorial>

## Capitol 2. Proiectarea aplicației

### 2.1. Prezentarea aplicației

Reciclarea a fost abordată prin diverse inițiative și tehnologii de-a lungul timpului. Această tranziție de la metodele vechi, care în cele mai multe cazuri erau corelate cu metodologii manuale și logistica dificilă, a adus creșterea aplicațiilor mobile și a platformelor digitale. Lucrarea își propune să faciliteze procesul de reciclare și să promoveze un comportament mai ecologic atât în rândul utilizatorilor, cât și al firmelor de acest tip. Aplicația este concepută pentru a oferi utilizatorilor o experiență ușoară și eficientă în acest proces, furnizând informații detaliate despre modul corect de reciclare.



# EcoConnect

. Logo EcoConnect<sup>11</sup>

Utilizatorii beneficiază de o interfață intuitivă, care le permite să navigheze ușor printre funcționalitățile aplicației. Aceasta include o hartă integrată Google Maps care afișează toate companiile de reciclare din zona lor, pe o rază definită de utilizator. Utilizatorii pot filtra companiile după tipul de material pe care acestea sunt specializate, asigurându-se astfel că aleg cea mai potrivită opțiune pentru nevoile lor de reciclare. EcoConnect permite utilizatorilor să facă cereri (request-uri) pentru colectarea deșeurilor. Fiecare request este înregistrat într-un istoric, unde utilizatorii pot vedea statusul cererii, data la care a fost creată, tipul de material reciclat și alte detalii relevante. Această funcționalitate oferă transparență și permite utilizatorilor să urmărească progresul fiecărei cereri.

Pe lângă reciclarea utilizatorilor, lucrarea optimizează rutele de colectare a deșeurilor pentru companii, atribuindu-le cele mai apropiate cereri din zona lor. Aceasta nu doar că reduce emisiile de gaze și aglomerația din oraș, dar și îmbunătățește eficiența operațională a companiilor de reciclare. Prin reducerea distanțelor parcuse și a timpului necesar pentru colectare, aplicația contribuie la un mediu mai curat și la o economie mai sustenabilă. EcoConnect colaborează îndeaproape cu companiile de reciclare pentru a asigura că cererile utilizatorilor sunt procesate rapid și eficient. Companiile primesc cererile în timp real despre noile requesturi ale utilizatorilor și pot gestiona cererile, să le respingă sau să le accepte, iar apoi după ce s-a finalizat preluarea deșeurilor, compania are un buton de "Finish" în care actualizează statusul requestului. Aceasta permite o comunicare eficientă și o cooperare strânsă între utilizatori și furnizorii de servicii de reciclare.

---

<sup>11</sup> Logo EcoConnect

Prin implementarea acestor funcționalități, EcoConnect își propune să transforme procesul de reciclare într-o activitate accesibilă și eficientă pentru toți utilizatorii. Aplicația va reduce barierele în calea reciclării, oferind soluții convenabile și motivationale. EcoConnect nu doar că va facilita reciclarea, dar va și inspira un nou nivel de implicare comunitară în practicile sustenabile. Utilizatorii vor fi motivați să participe activ în procesul de reciclare, simțindu-se parte dintr-o mișcare mai mare de protejare a mediului. În acest fel, aplicația va juca un rol esențial în promovarea economiei circulare și în construirea unui viitor mai verde pentru toți.

Aplicația este structurată în mai multe componente interconectate. Principalele componente ale arhitecturii sunt:

- Backend:** Implementat în IntelliJ cu Spring Boot, gestionează logica aplicației și comunicarea cu baza de date.
- Frontend:** Implementat în Visual Studio Code dezvoltat cu React, oferă interfața utilizatorului și interacționează cu backend-ul prin HTTP folosind API-uri.
- Baza de date:** PostgreSQL, folosit pentru stocarea și gestionarea datelor aplicației.



Figura 2.1 Arhitectura aplicației<sup>12</sup>

Figura 2.1. ilustrează un flux de lucru tipic pentru o aplicație web full-stack, evidențiind interacțiunile dintre frontend și backend și, în cele din urmă, cu baza de date.

## 2.2. Resurse Software

### 2.2.1. Baza de Date PostgreSQL

PostgreSQL este o bază de date open source care are o reputație solidă pentru fiabilitatea, flexibilitatea și sprijinul acordat standardelor tehnice deschise. Spre deosebire de alte RDMBS (Relational database management system), PostgreSQL acceptă atât tipuri de date non-relaționale, cât și relaționale. Acest lucru îl face să fie una dintre cele mai conforme, stabile și mature baze de date relaționale disponibile în prezent. PostgreSQL oferă suport pentru un vast set de funcționalități și extensii, fiind o alegere ideală pentru aplicații complexe.[16]

Ca și relații între tabele:

- company - general\_user → One to One
- person - general\_user → One to One
- company - request → One to Many

<sup>12</sup> Figura 2.1 Arhitectura aplicației

- person - request → One to Many
- material - company → material\_company Many to Many
- company - material → material\_company Many to Many
- material - request → One to Many
- request - company → Many to One
- request - person → Many to One
- request - material → Many to One

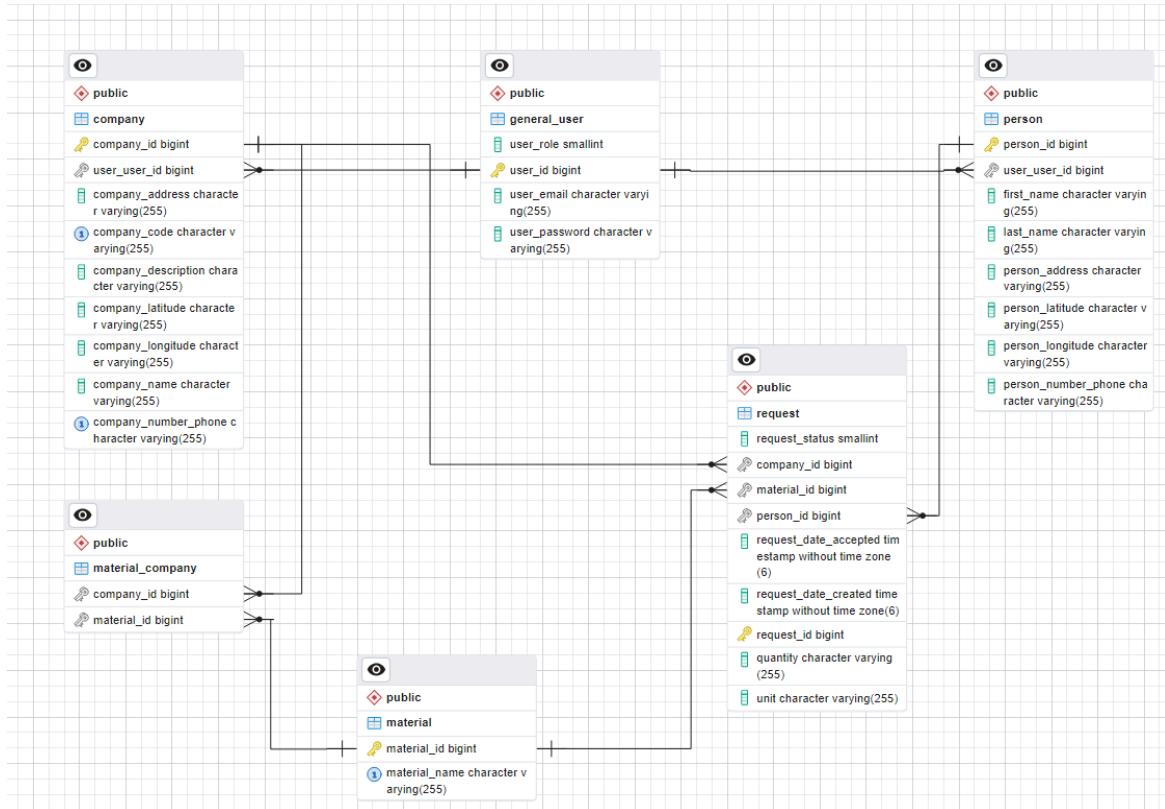


Figura 2.2 Diagrama ER<sup>13</sup>

## 2.2.2. Backend SpringBoot, IntelliJ

Spring Boot este un framework open-source bazat pe Spring, destinat dezvoltării rapide și eficiente a aplicațiilor Java. Creat pentru a simplifica procesul de dezvoltare și configurare a aplicațiilor, Spring Boot oferă o serie de caracteristici care reduc semnificativ cantitatea de cod necesar și timpul de dezvoltare, permitând dezvoltatorilor să se concentreze mai mult pe logică și mai puțin pe configurarea infrastructurii. [17]

```

@Table(name = "company")
public class Company {

```

<sup>13</sup> Figura 2.2 Diagrama ER

```

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "company_id")
private Long companyId;

```

În SpringBoot este totul configurat automat, pentru a utiliza anumite funcții, trebuie să utilizăm configurația corectă. Dacă dorim să utilizăm Hibernate (ORM), trebuie doar să adăugăm o anotare `@Table` deasupra clasei de model/entitate și o anotare `@Column` care să mapeze la tabelul și coloana bazei de date.

```

@RestController
@RequestMapping(value = "/api/users")

```

De exemplu, aceste anotări sunt și pentru primirea sau transmiterea informațiilor din interfața web cu ajutorul endPoint-urilor, folosind anotărilor `@RestController` și `@RequestMapping`.

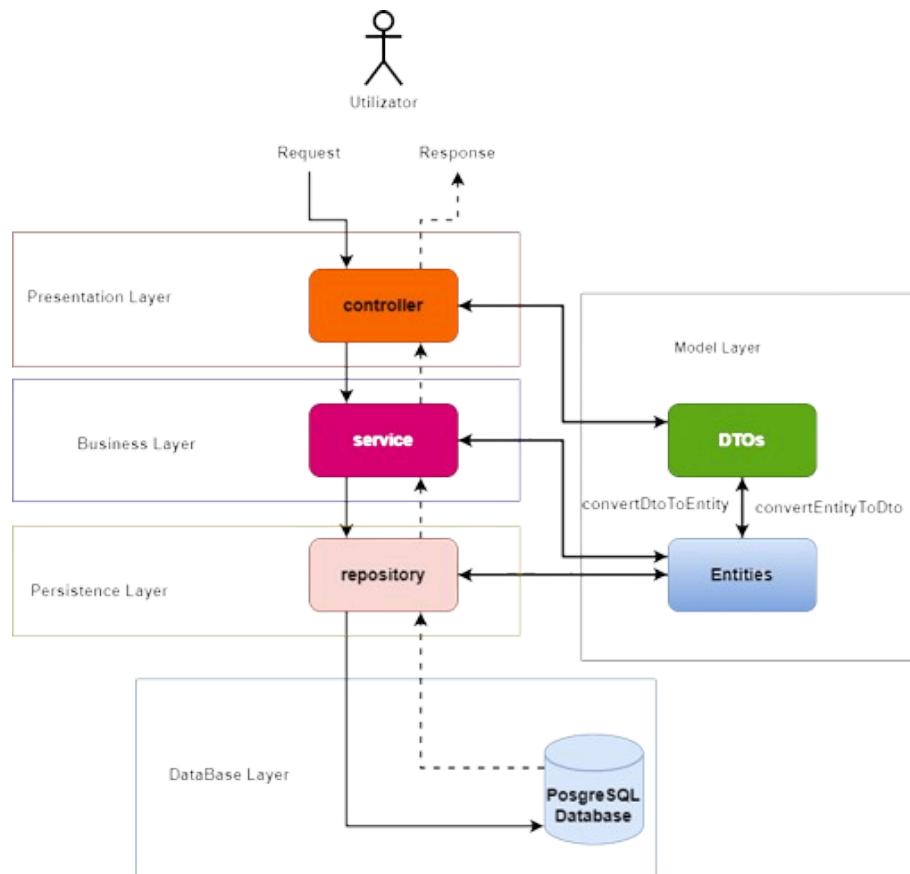


Figura 2.3. Arhitectură Backend<sup>14</sup>

<sup>14</sup> Figura 2.3. Arhitectură Backend

Figura 2.5 ilustrează arhitectura proiectului, evidențiind relațiile dintre componente, precum DTOs, controller, service, repository, Entities. DTO-urile sunt folosite pentru a transfera datele între client și controller, facilitând astfel schimbul de informații. Controller-ul primește cererile HTTP de la client, utilizează DTO-urile pentru a gestiona datele și apelează metodele definite în interfața Service pentru a implementa logica. Service definește și implementează funcțiile oferind funcționalitățile necesare. Service interacționează cu Repository pentru operațiunile CRUD (Create, Read, Update, Delete) asupra entităților, facilitând astfel accesul la baza de date, dar și cu entitățile direct în funcțiile de salvare. Entitățile reprezintă structurile de date din baza de date și sunt mapate prin intermediul Repository pentru a fi manipulate în cadrul aplicației.

### 2.2.3. Frontend React, Visual StudioCode

În prezent, cadrele și bibliotecile front-end devin o parte esențială a dezvoltării web moderne. React.js este o bibliotecă front-end care a devenit treptat cadrul de referință pentru dezvoltarea web contemporană în cadrul mediului JavaScript. În loc să se ocupe de întreaga interfață cu utilizatorul ca de o singură unitate, React.js stimulează dezvoltatorii să împartă aceste interfețe complexe în componente individuale reutilizabile care alcătuiesc elementele de bază ale întregii interfețe. Astfel, cadrul ReactJS combină rapiditatea și eficiența JavaScript cu o metodă mai eficientă de manipulare a DOM pentru a afișa paginile web mai rapid și a crea aplicații web foarte dinamice și receptive.[18]

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
```

- useState este un hook introdus în React care permite componentelor funcționale să gestioneze starea.
- useEffect este un hook, care permite executarea de efecte secundare în componente funcționale. Efectele secundare includ lucruri precum manipularea DOM-ului, apeluri API, subscripții și alte operațiuni care nu sunt direct legate de redarea componentelor.
- useNavigate este un hook din biblioteca “react-router-dom”, care este utilizată pentru gestionarea rutelor în aplicațiile React. Din această librărie putem importa mai multe elemente pentru navigare, precum Link, Router, Route, Routes.

**Material UI (MUI)** este o bibliotecă de componente React care implementează principiile de design Material Design, dezvoltate de Google. Material UI este utilizat pentru a construi interfețe de utilizator moderne, elegante și consistente.

```

import {AppBar, Toolbar, Container, TextField, Typography, Box, Button,
IconButton, Table, TableBody, TableCell, TableContainer, TableHead, TableRow,
Paper, Dialog, DialogActions, DialogContent, DialogTitle} from '@mui/material';
import AccountCircleIcon from '@mui/icons-material/AccountCircle';

```

În codul de mai sus, putem observa cum importăm anumite elemente din două pachete principale ale Material UI: “@mui/material” și “@mui/icons-material”. Toate aceste elemente pot fi reutilizabile ori de câte ori este nevoie.

```

import { GoogleMap, Marker, useJsApiLoader } from
'@react-google-maps/api';

```

“@react-google-maps/api” este o bibliotecă care facilitează integrarea Google Maps în aplicațiile React. Aceasta oferă componente și hook-uri pentru a utiliza și personaliza hărțile Google Maps, oferind dezvoltatorilor o modalitate simplă și eficientă de a încorpora funcționalități de hartă în aplicațiile lor.

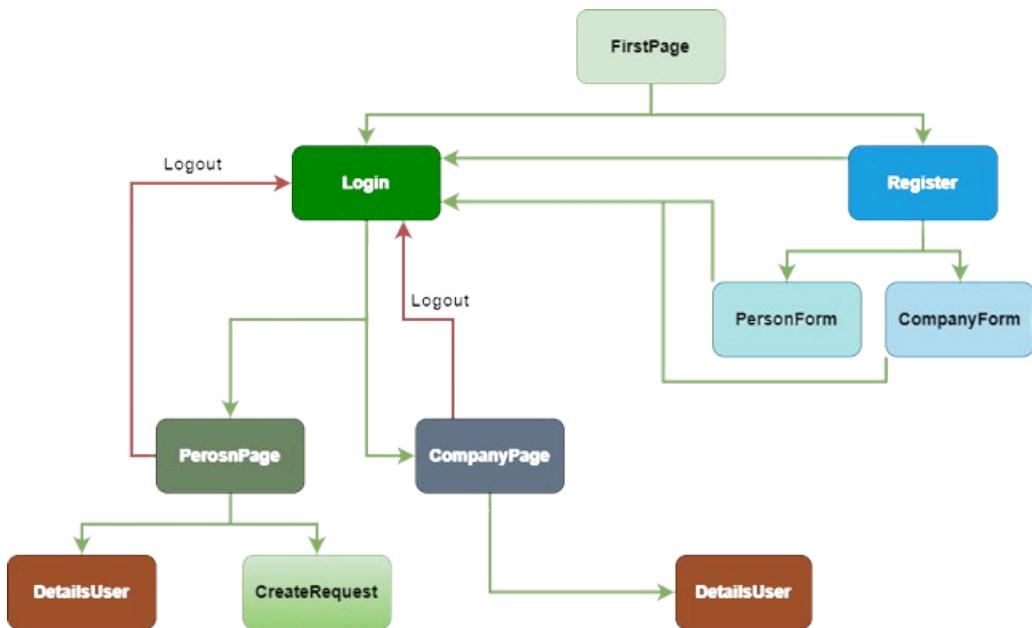


Figura 2.4. Arhitectură Interfață<sup>15</sup>

Prima pagină a aplicației, denumită FirstPage, este punctul de acces pentru utilizatori, prezentând două butoane: "Login" și "Register". Utilizatorii care nu dețin un cont, apasă pe buton de înregistrare, fiind direcționați către pagina de Register, unde li se oferă posibilitatea de a selecta tipul de entitate pe care doresc să o înregistreze: persoană fizică sau persoană juridică,

<sup>15</sup> 2.4. Arhitectură Interfață

care reprezintă compania. Această alegere determină setul de informații necesare care trebuie completate pentru înregistrare. După finalizarea acestui proces, utilizatorii sunt redirectionați către pagina de Login, unde trebuie să introducă e-mailul și parola stabilite în timpul înregistrării.

Accesul în aplicație este diferit în funcție de tipul de entitate aleasă. Pe PersonPage, utilizatorii observă în bara de navigare un icon prin care pot accesa pagina DetailsUser și un buton "Logout" care îi redirecționează către pagina de Login. De asemenea, pentru a crea noi cereri, există un buton "Create Request" care direcționează utilizatorii către pagina CreateRequest. În contrast, pe CompanyPage, utilizatorii găsesc în bara de navigare un buton "Logout" și un icon pentru accesarea informațiilor companiei DetailsUser.

Aceasta este structura aplicației, care este concepută pentru a fi ușor de folosit, facilitând navigarea și interacțiunea eficientă cu utilizatorii. Designul intuitiv al interfeței permite accesul rapid la funcționalitățile esențiale, îmbunătățind experiența generală a utilizatorului și optimizând procesul de administrare a conturilor, fie că este vorba de persoane fizice sau juridice.

## 2.2.4. Comunicare server-client

### 2.2.4.1. Protocol HTTP

**Protocolul HTTP** (Hypertext Transfer Protocol) este un protocol de comunicație utilizat pentru a transfera date pe web. Este baza oricărei comunicări dintre un client (de obicei un browser web) și un server web.[19] Conexiunea între backend și frontend implică, de obicei, comunicarea prin intermediul API-urilor (Application Programming Interfaces). Acestea permit transferul de date și funcționalități între server și client.

#### Backend

- **Serverul:** rulează aplicația backend.
- **API-ul:** expune endpoint-uri la care frontend-ul poate trimite requesturi HTTP pentru a obține sau trimite date.

```
@RequestMapping(value = "/api/users")
```

```
@PostMapping(value = "/login")
```

#### Frontend

- **Clientul:** aplicația frontend trimite requesturi HTTP către server pentru a interacționa cu backend-ul.
- **Axios/Fetch:** metode folosite în JavaScript pentru a trimite requesturi HTTP asincron către server fără a reîncărca pagina.

```
const response = await fetch('http://localhost:8082/api/materials');
```

```

const API_URL = 'http://localhost:8082/api/requests';
export const getAllRequestsByUserId = async (userId) => {
  return await axios.get(` ${API_URL}/person/${userId}/requests`);
};

```

#### 2.2.4.2. Configurare CORS

Cross-Origin Resource Sharing (CORS) este un mecanism bazat pe antetul HTTP care permite resurselor web dintr-un domeniu să fie accesate de pagini web dintr-un alt domeniu.[20]

```

@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
            .allowedOrigins("http://localhost:3000")
            .allowedMethods("GET", "POST", "PUT", "DELETE", "PATCH",
    "OPTIONS")
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
    }
}

```

De exemplu, frontend-ul React rulează pe “<http://localhost:3000>”, iar backend-ul Spring Boot pe “<http://localhost:8082>”. Pentru a permite comunicarea între aceste două domenii, se configurează CORS în backend.

## Capitolul 3. Implementarea aplicației

### 3.1. Descrierea generală a aplicației

Aplicația web EcoConnect a fost dezvoltată în mediul de dezvoltare IntelliJ IDEA, folosind framework-ul Spring pentru partea de backend și Visual Studio Code pentru partea de frontend, realizată cu React. Implementarea backend-ului a fost realizată cu ajutorul limbajului de programare Java, folosind framework-ul Spring pentru a gestiona logica aplicației și interacțiunea cu baza de date. Pentru frontend, React a fost ales datorită flexibilității și eficienței sale în construirea interfețelor utilizator.

Scopul implementării este de a crea o aplicație care să ofere celor două tipuri de utilizatori, informații actualizate și relevante despre reciclare. În acest sens, sunt folosite mai multe API-uri pentru a obține informații despre anumite date, precum informații despre datele companiilor, tipurile de material pe care le reciclează, filtrările companiilor după material sau raza și multe altele.

La finalul implementării se așteaptă ca aplicația să funcționeze corect într-un procent foarte mare, pentru a se trece la partea de testare și de rezolvare a unor eventuale bug-uri și probleme întâlnite la utilizarea ei.

Procesul de dezvoltare al aplicațiilor web este un proces creativ de rezolvare a problemelor și, prin urmare, există o probabilitate mare să apară bug-uri (erori). Orice aplicație dezvoltată înseamnă rezolvarea unor nevoi sau cerințe și de aceea fiecare programator întâlnește astfel de probleme în procesul de dezvoltare.

Aplicația EcoConnect a întâmpinat o serie de probleme și dificultăți pe parcursul dezvoltării acesteia, care au fost însă rezolvate. Un obstacol major a fost integrarea API-ului Google Maps. Inițial, markerele nu apăreau pe hartă și coordonatele nu erau preluate corect. Această problemă a fost rezolvată prin:

- Debugging pentru a identifica eventualele erori de cod care împiedicau afișarea markerelor, utilizând console.log pentru a verifica datele și pașii intermediari.
- Actualizarea documentației și implementarea unor exemple practice de utilizare a API-ului Google Maps, asigurându-se că toate permisiunile și cheile API necesare sunt corect configurate.

Aceste soluții au permis depășirea problemelor întâmpinate și asigurarea funcționării corecte a aplicației „EcoConnect”, pregătind-o pentru faza de testare și pentru lansarea finală.

În zilele noastre, este din ce în ce mai dificil pentru aplicații să vină cu idei inovatoare. Multe dintre aplicațiile nou lansate doar completează alte aplicații din aceeași categorie, iar numărul lor crește în fiecare zi.

Aplicația EcoConnect încearcă să aducă o îmbunătățire la aplicațiile de ecologizare și reciclare, asigurând utilizatorul că nu va fi lăsat să transporte singur deșeurile, venind cea mai apropiată firmă de pe o rază bine stabilită. Ca și îmbunătățire este combinarea ideilor celor două firme mari Recapp și Recycle Nation, una care asigură transportul deșeurilor prin cererile utilizatorului, respectiv cealaltă care are un motor de căutare și afișează cele mai apropiate stații de colectare.

O altă idee, de a atrage mai mulți utilizatori, a fost de a face interfața cât mai minimalistă și ușor de înțeles posibil. Utilizarea elementelor de design simple, utilizarea unei palete de culori limitate și plasarea cea mai precisă a informațiilor pe ecran, fac ca utilizarea aplicației să fie plăcută și fără probleme.

## 3.2. Funcționalitatea aplicației

Algoritmul pentru calcularea distanței între 2 puncte pe suprafața Pământului este implementată pe baza formulei Haversine. Formula Haversine este o metodă destul de precisă pentru calculul distanței dintre două puncte de pe suprafața unui sferoid bazându-se pe latitudinile și longitudinile lor.

```
@Override
public Double CalculateDistanceBetweenTwoPoints(Double lat1, Double lon1,
Double lat2, Double lon2) {
    Double earthRadius = 6371.0;
    double dLatitude = Math.toRadians(lat2-lat1);
    double dLongitude = Math.toRadians(lon2-lon1);
    double firstPointLat = Math.toRadians(lat1);
    double secondPointLat = Math.toRadians(lat2);

    double rez = Math.pow(Math.sin(dLatitude/2), 2) +
    Math.pow(Math.sin(dLongitude/2), 2)*
    Math.cos(firstPointLat)*Math.cos(secondPointLat);
    double result = 2*Math.asin(Math.sqrt(rez));
    return earthRadius * result;
}
```

Acesta este algoritmul pentru calcularea distanței între două puncte pe suprafața Pământului, unde raza pământului este setată la 6371 km, fiind raza medie a pământului. Diferențele între latitudini și longitudini, cât și cele două latitudini se transformă în radiani, pentru a putea folosi funcțiile trigonometrice.

Funcționalitățile de bază ale aplicației propuse sunt disponibile odată cu prima pagină a aplicației (Figura 3.1.), unde apare logo-ul aplicației și două butoane, unul de Login și unul de Register. Depinde dacă utilizatorul are sau nu cont în acel moment.

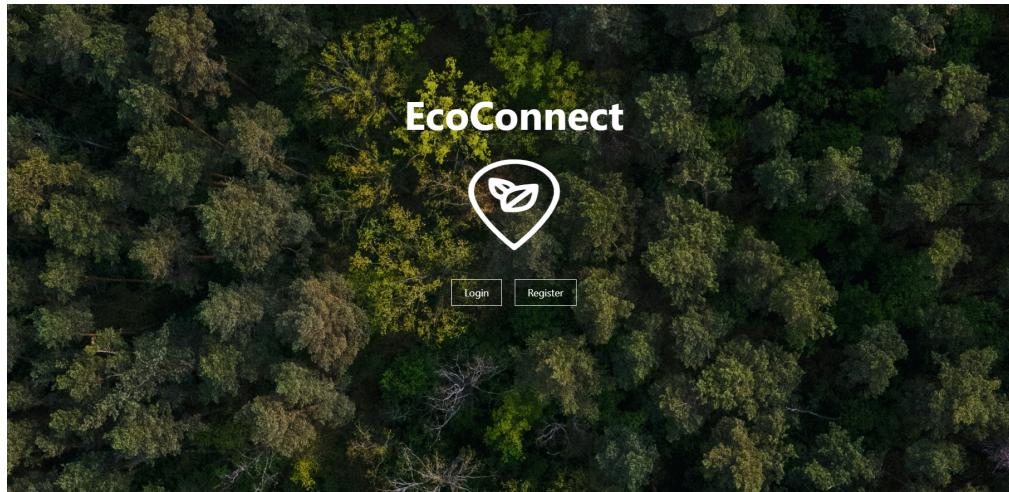


Figura 3.1. Prima Pagină<sup>16</sup>

În etapa de înregistrare, utilizatorul trebuie să aleagă tipul de entitate pe care dorește să o reprezinte: persoană fizică sau persoană juridică. Pentru fiecare entitate, câmpurile sunt diferite și necesare toate, pentru a crea cont.

A screenshot of a "Sign Up" registration form. The form is set against a background image of a forest. At the top, the word "Sign Up" is centered in a green, bold, sans-serif font. Below it is a horizontal line. Underneath the line, there are two radio buttons: one labeled "Person" and another labeled "Company". The "Person" button is checked. The form consists of six input fields, each with a placeholder text and a small info icon to its right. The fields are: "First Name", "Last Name", "Email", "Password", "Phone Number", and "Building".

Figura 3.2. Pagina Register Person\_1<sup>17</sup>

---

<sup>16</sup> Figura 3.1. Logo-ul aplicației

<sup>17</sup> Figura 3.2. Pagina Register Person\_1

Parola care este adăugată în câmpul corespunzător este criptată și apoi introdusă în baza de date.

```

private static void setKey(String myKey) {
    MessageDigest sha;
    try {
        key = myKey.getBytes(StandardCharsets.UTF_8);
        sha = MessageDigest.getInstance("SHA-256");
        key = sha.digest(key);
        key = Arrays.copyOf(key, 16);
        secretKey = new SecretKeySpec(key, "AES");
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
}

public static String encrypt(String strToEncrypt, String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);

        return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF_8)));
    } catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}

```

Acesta este algoritmul de criptare simetric AES folosind hashing SHA-256. Convertim cheia într-un array de octeți. Cheia este introdusă într-un hash folosind SHA-256, rezultând un array de 32 de octeți. Se iau doar primii 16 octeți și se creează cheia secretă specificând algoritmul care o să fie utilizat. Se creează obiectul cipher care folosește algoritmul AES în modul ECB(Electronic CodeBook) cu padding PKCS5. Rezultatul criptării este codificat în Base64 și returnat ca sir de caractere.

```

private boolean validate(UserLoginDTO userLoginDTO, String email, String password) {

```

```

String encryptPwd = CryptPassword.encrypt(userLoginDTO.getPassword(),
MessageContent.SECRET_KEY);

return userLoginDTO.getEmail().equals(email) &&
encryptPwd.equals(password);
}

```

Această funcție verifică dacă emailul și parola coincid cu cele furnizate.

user_role	user_id	user_email	user_password
smallint	[PK] bigint	character varying (255)	character varying (255)
0	1	ionpopescu@gmail.com	LGEc/o9cQrJxuz7Uu5wl3A==

Figura 3.3. Parola criptată<sup>18</sup>

În figura 3.3. se observă parola criptată care este stocată în baza de date.

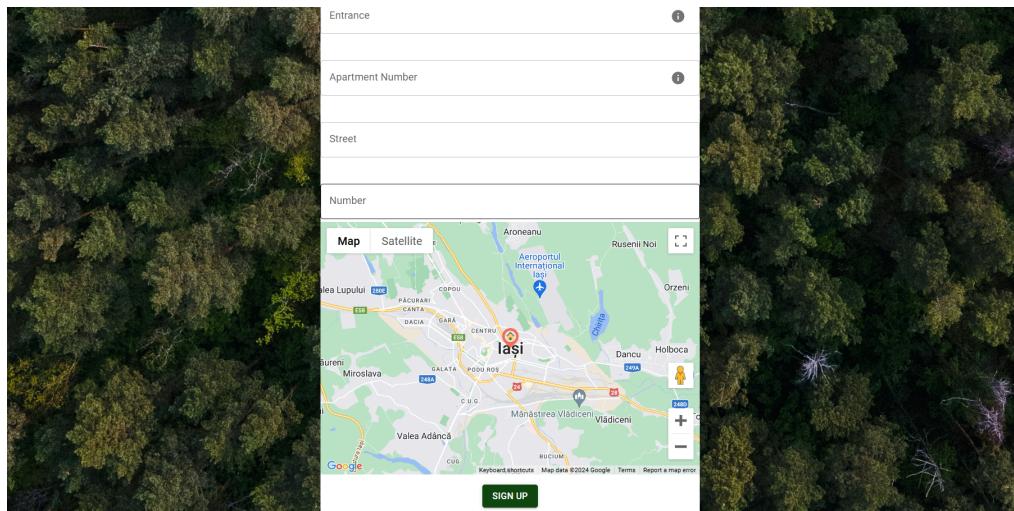


Figura 3.4. Pagina Register Person\_2<sup>19</sup>

Câmpurile Street și Number se completează singure după ce se selectează locația persoanei sau companiei, punând pinul exact pe gospodăria sau punctul de lucru al acestuia. Așa se extrage și latitudinea și longitudinea cu ajutorul acestei selecții.

---

<sup>18</sup> Figura 3.3. Parola criptată

<sup>19</sup> Figura 3.4. Pagina Register Person\_2

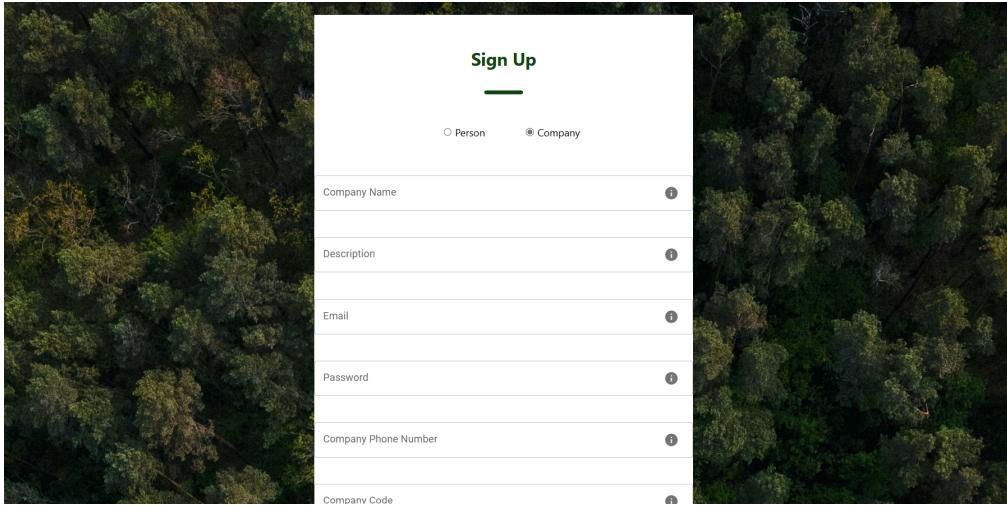


Figura 3.5. Pagina Register Company\_1<sup>20</sup>

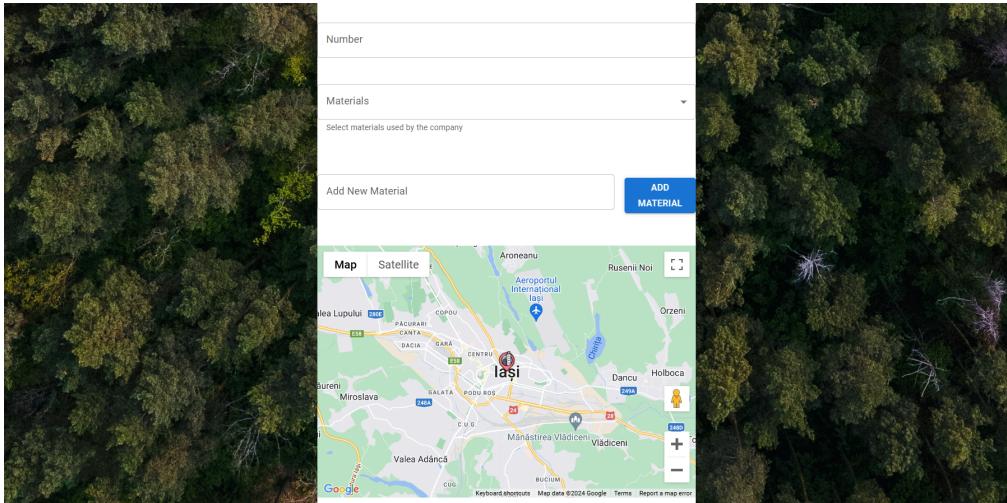


Figura 3.6. Pagina Register Company\_2<sup>21</sup>

La Company Register este aproximativ aceeași idee ca la Person Register, doar că sunt alte câmpuri de completat. Companiile sunt rugate să aleagă tipurile de material pe care sunt specializate pentru reciclare, iar dacă nu există un tip de material îl pot introduce. La toate câmpurile pe care trebuie completeate, există un icon info care prezintă un exemplu cum ar putea utilizatorul completa acel câmp.

```
const personSchema = yup.object().shape({
```

<sup>20</sup> Figura 3.5. Pagina Register Company\_1

<sup>21</sup> Figura 3.6. Pagina Register Company\_2

```
firstName: yup.string().matches(/[A-Z][a-z]*$/,'Invalid name,  
[A-Z][a-z]').required('First name is required'),  
  
lastName: yup.string().matches(/[/A-Z][a-z]*$/,'Invalid name,  
[A-Z][a-z]').required('Last name is required'),  
  
email: yup.string().email(/([a-zA-Z0-9]+@[a-zA-Z]+\.[a-zA-Z]+)*$/,'Invalid  
email address').required('Email is required'),  
  
password: yup.string().min(8,'Password must be at least 8  
characters').required('Password is required'),  
  
numberPhone: yup.string().matches(/^0[7]([0-9]{8})$/,'Invalid phone  
number').required('Phone number is required')
```

Fiecare câmp are o restricție sau un format anume, precum firstName, lastName trebuie să înceapă cu literă mare, la email trebuie să prezinte un format bine stabilit (“@”, “.com”). Parola trebuie să conțină minim 8 caractere, numărul de telefon trebuie să înceapă cu “07” continuat de 8 numere.

```
@NotNull(message = "Required")  
 @Size(max = 50, message = "Email provided does not adhere to the specified  
 field validation rules")  
 @Pattern(regexp = "([a-zA-Z0-9]+@[a-zA-Z]+\.[a-zA-Z]+)", message = "Email  
 provided does not adhere to the specified field validation rules")  
 private String email;
```

Aceste constrângeri sunt realizate în backend prima dată și folosim funcții de validare ale câmpurilor completate.

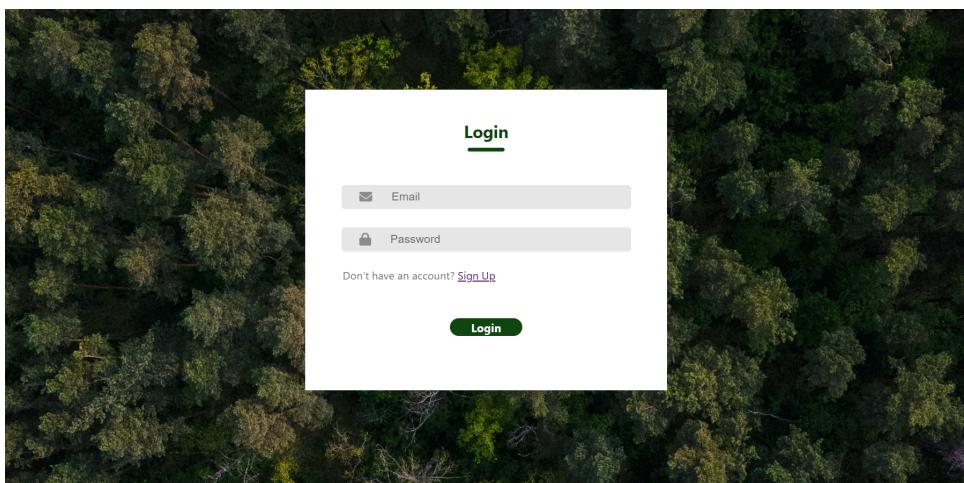


Figura 3.7. Pagina Login<sup>22</sup>

---

<sup>22</sup> Figura 3.7. Pagina Login

În ecranul de înregistrare (Figura 3.7.) utilizatorul trebuie să completeze e-mailul și parola pentru a reuși autentificarea. În funcție de rolul utilizatorului, acesta se loghează pe pagina corespunzătoare.

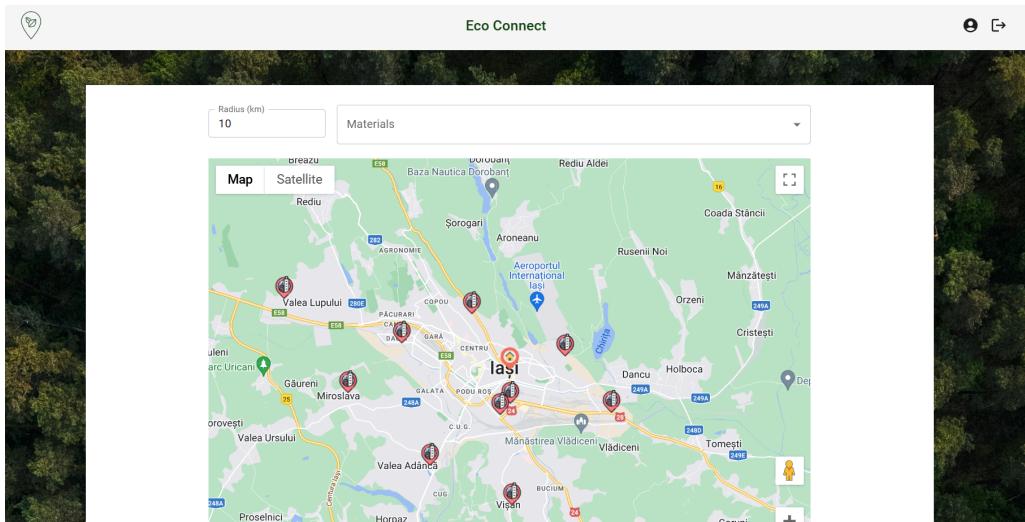


Figura 3.8. Pagina Person<sup>23</sup>

În figura 3.8. este pagina principală a utilizatorului persoană unde poate filtra companiile după raza de acțiune pe care o dorește și după materialele specializate ale companiilor. Aceasta este funcția prin care se calculează distanța care ajută utilizatorul să filtreze companiile după rază.

```
@Override
public List<Company> getAllNearbyCompanyRadius(Double latitude, Double longitude, Double radius) {
    List<Company> nearbyCompany = new ArrayList<>();
    for(Company company : companyRepository.findAll()) {
        Double latitudeCompany = Double.parseDouble(company.getLatitude());
        Double longitudeCompany = Double.parseDouble(company.getLongitude());
        if(CalculateDistanceBetweenTwoPoints(latitude, longitude, latitudeCompany, longitudeCompany) < radius &&
           CalculateDistanceBetweenTwoPoints(latitude, longitude, latitudeCompany, longitudeCompany) !=0) {
            nearbyCompany.add(company);
        }
    }
}
```

<sup>23</sup> Figura 3.8. Pagina Person

```
        }

        return nearbyCompany;
    }
}
```

Totodată, utilizatorul când apasă pe un pinul unei companii, poate observa anumite detalii relevante despre acea companie, aceste informații sunt puse pe un InfoWindow.

```
{selectedCompany && (
    <InfoWindow
        position={{
            lat: parseFloat(selectedCompany.latitude),
            lng: parseFloat(selectedCompany.longitude)
        }}
        onCloseClick={() => setSelectedCompany(null)}>

        <div>
            <h2>{selectedCompany.companyName}</h2>
            <p><strong>Materials:</strong> {selectedCompany.materialList ? selectedCompany.materialList.join(', ') : 'N/A'}</p>
            <p><strong>Email:</strong> {selectedCompany.email || 'N/A'}</p>
            <p><strong>Phone:</strong> {selectedCompany.companyNumberPhone || 'N/A'}</p>
            <p><strong>Address:</strong> {selectedCompany.address || 'N/A'}</p>
        </div>
    </InfoWindow>
}
```

În bara aplicației sunt două icon-uri, unul de delegare care redirecționează pe pagina de Login și un icon pentru profile care navighează pe pagina de DetailsUser, de unde sunt prezentate informațiile necesare utilizatorului și un buton de delete unde se poate șterge contul.(Figura 3.9.)

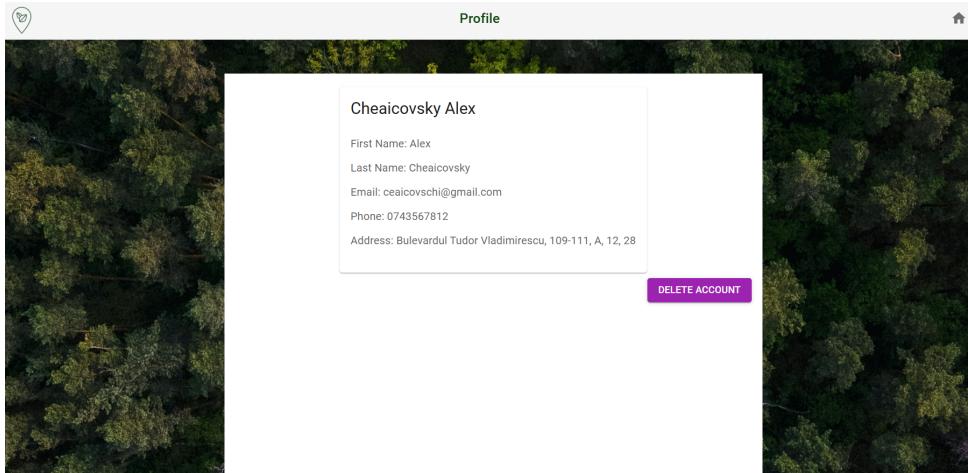


Figura 3.9. Pagina DetailsUser<sup>24</sup>

Tot pe pagina principală, utilizatorul poate observa un istoric al requesturilor făcute și statusul fiecărui request, pe lângă aceasta se prezintă mai multe detalii relevante precum, data creării requestului, tipul de material cantitatea și unitatea în care este măsurată cantitatea. Se poate observa din figura următoare, că fiecare cerere se poate șterge prin butonul Delete. Există butonul “Create request”, destul de intuitiv, unde se poate crea un request.

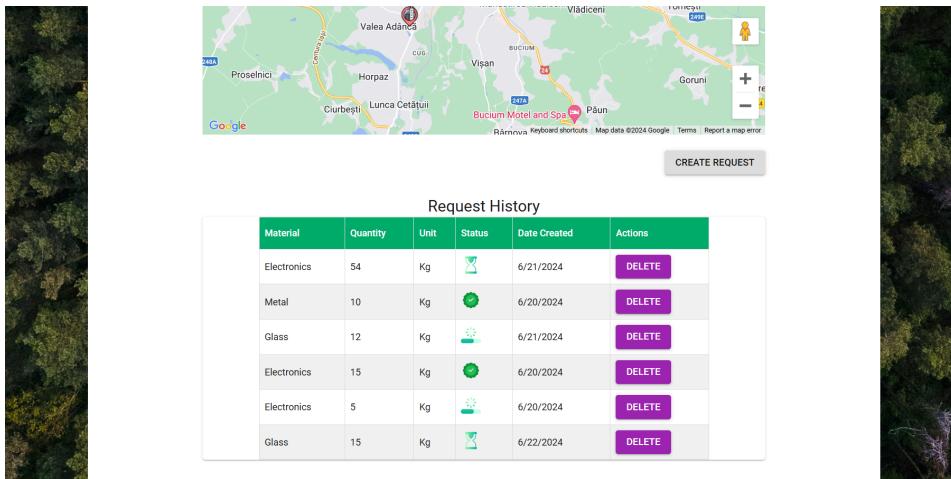


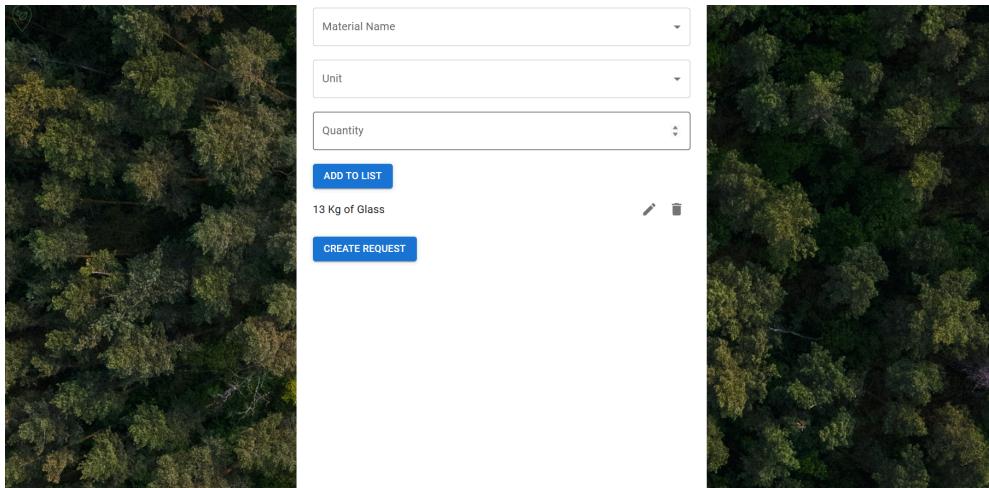
Figura 3.10. Pagina RequestHistory<sup>25</sup>

Pe pagina de CreateRequest, utilizatorul alege tipul de material pe care îl dorește reciclat, cantitatea și unitatea în care s-a măsurat; acesta îl bagă într-o listă, pentru a nu face mai multe

<sup>24</sup> Figura 3.9. Pagina DetailsUser

<sup>25</sup> Figura 3.10. Pagina RequestHistory

requesturi, mai bine le bagă într-un request pe toate, cu opțiunea de a șterge cererea, sau de a o edita. Prezentat toate aceste funcționalități în figura 3.10.



The screenshot shows a web application interface for creating a request. On the left is a large image of a forest. On the right is a form with the following fields:

- Material Name dropdown
- Unit dropdown
- Quantity dropdown
- A list entry: "13 Kg of Glass" with edit and delete icons
- Buttons: "ADD TO LIST" (blue), "CREATE REQUEST" (blue)

Figura 3.11. Pagina CreateRequest<sup>26</sup>

Pe pagina CompanyPage, după ce se loghează utilizatorul cu e-mailul firmei și parola corespunzătoare, apare harta cu pinurile companiilor din apropiere și requesturile persoanelor din apropiere, această rază este o distanță relativă de 3 km, pentru a putea fi mult mai bine optimizat aceste cereri. Totodată apar și requesturile acceptate, care după ce sunt finalizate, actualizează statusul din Processing în Completed prin butonul “Finish”

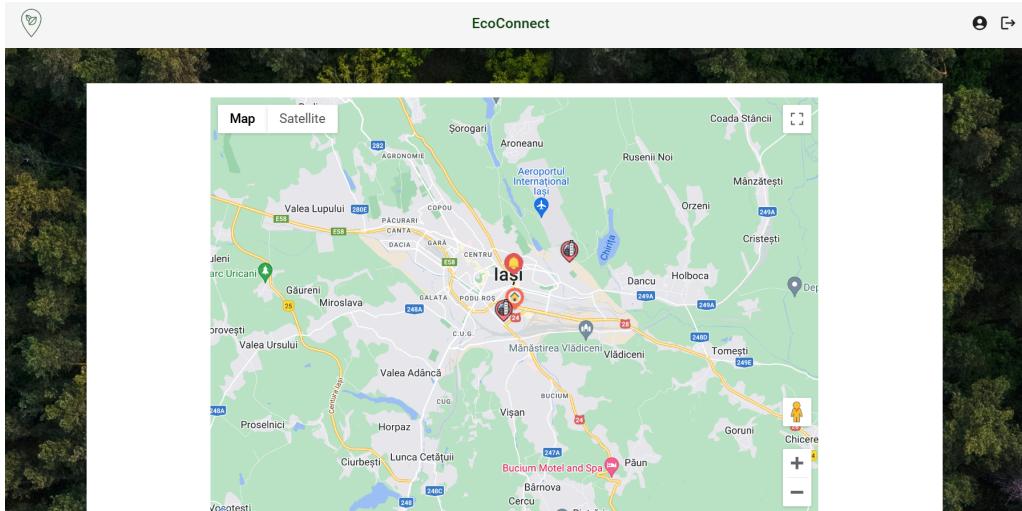
Requests Accepted:				
Material	Quantity	Unit	Status	Action
Electronics	5	Kg		<a href="#">FINISH</a>
Metal	10	Kg		
Electronics	15	Kg		

Figura 3.12. Pagina CompanyPage RequestsAccepted<sup>27</sup>

---

<sup>26</sup> Figura 3.11. Pagina CreateRequest

<sup>27</sup> Figura 3.12. Pagina CompanyPage RequestsAccepted

Figura 3.13. Pagina CompanyPage<sup>28</sup>

Utilizatorul trebuie să dea click pe pin-ul cu clopoțel, deoarece acolo sunt cererile de reciclare pentru un anumit material pe care este specializată firma logată. La fiecare request, acesta trebuie să programeze data și ora de sosire a firmei pentru preluarea deșeurilor.(Figura 3.12.) Pe pinurile companiilor, utilizatorul poate observa detalii ale firmelor concurente, tipurile de materiale pe care le reciclează fiecare.(Figura 3.13.) Aceasta este funcția de click pe markere.

```
const handleMarkerClick = (location) => {
    setSelectedUser(null);
    setSelectedCompanyDetails(null);
    setSelectedUserRequests([]);
    setRequestsOnHold([]);
    if (location.role === 'Person') {
        setSelectedUser(location);
        getAllRequestsByUserId(location.id).then(response => {
            setSelectedUserRequests(response.data);
        }).catch(error => {
            console.error('Error fetching user requests:', error);
        });
        getAllRequestsOnHold(location.id).then(response => {

```

<sup>28</sup> Figura 3.13. Pagina CompanyPage

```

        setRequestsOnHold(response.data);
    }).catch(error => {
        console.error('Error fetching requests on hold:', error);
    });
} else if (location.role === 'Company') {
    setSelectedCompanyDetails(location);
}
});

```

Codul sursă pentru selectarea pin-urilor și extragerea informațiilor

În codul de mai sus este prezentată funcția în care se poate selecta pinul și a extrage informațiile necesare.

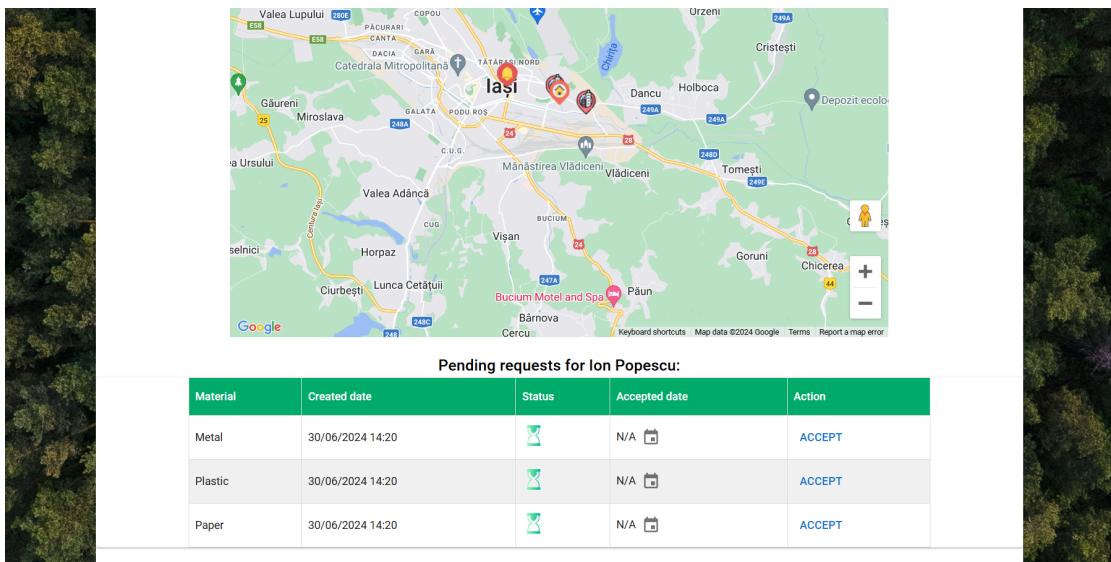


Figura 3.14. Accept Request<sup>29</sup>

<sup>29</sup> Figura 3.14. Accept Request

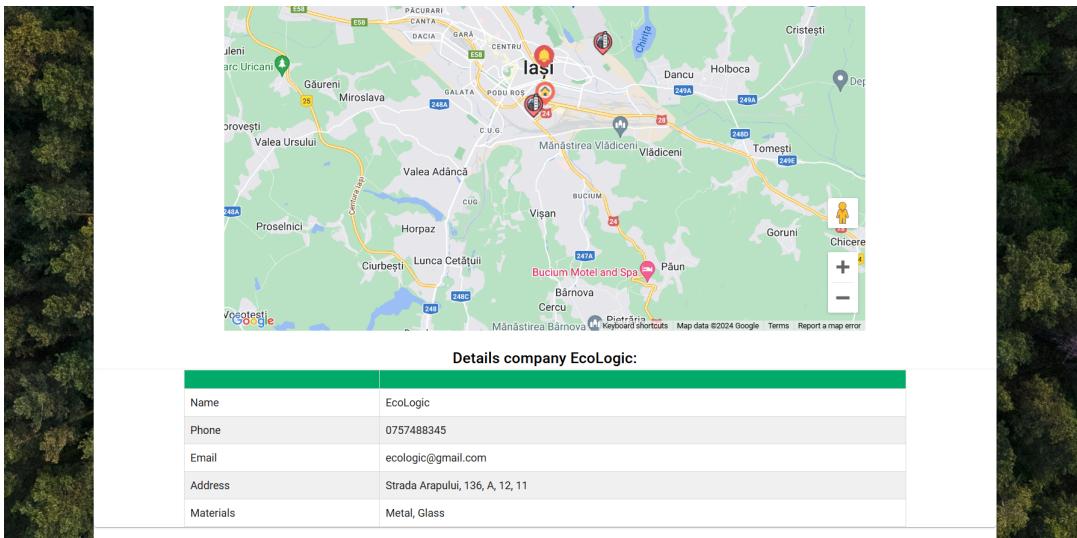


Figura 3.15. Detalii altă companie<sup>30</sup>

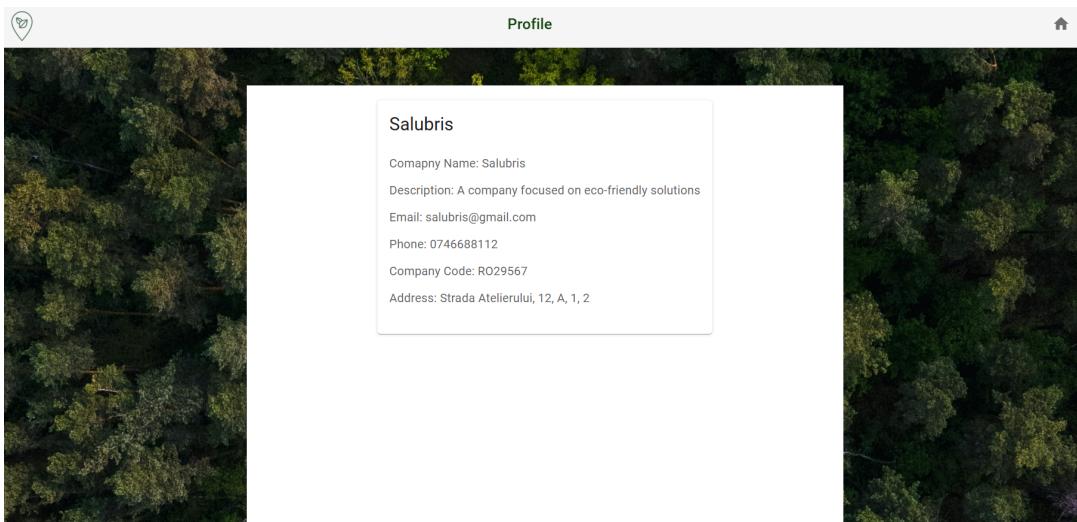


Figura 3.16. Detalii companie profile<sup>31</sup>

În bara aplicației, lângă icon-ul de logout este și icon-ul de profile pentru a putea vedea detaliile utilizatorului logat.

<sup>30</sup> Figura 3.15. Detalii altă companie

<sup>31</sup> Figura 3.16. Detalii companie profile

## Capitolul 4. Testarea aplicației și completări viitoare ale aplicației

### 4.1. Testarea aplicației

Succesul unei aplicații web este dat și de testare, prin care se evaluatează calitatea produsului respectiv. În urma testării unei aplicații se pot identifica probleme și defecte iar apoi se va încerca eliminarea acestora. Testarea nu poate demonstra cu certitudine de 100% că aplicația funcționează corect în orice condiții, dar poate demonstra că aceasta nu funcționează corect în anumite condiții, urmând ca aceste situații să fie analizate. După implementarea aplicației au fost făcute o serie de teste precum:

- Testarea funcționalității - se verifică dacă partea de backend a aplicației nu are probleme, și toate obiectivele aplicației funcționează la parametri normali.
- Testarea interfeței utilizator - se analizează elementele de frontend ale aplicației (design-ul), afișarea și aspectul lor în ecran trebuind să îndeplinească planurile proiectării aplicației.
- Testarea compatibilității - aplicația se va testa pe mai multe browsere diferite. Aplicația ar trebui să fie versatilă și potrivită pentru diferite dimensiuni de ecran, așa că este rulată cu diverse rezoluții.

De asemenea înaintea implementării aplicației s-a avut în vedere o testare pre-dezvoltare analizând nevoile utilizatorilor și alte aplicații din aceleași categorii și stabilind ce plusuri pot fi aduse în aplicația propusă.

#### 4.1.1. Testarea aplicației cu firme reale din Iași

Pentru a se realiza această testare, s-au luat informațiile publice de pe internet despre firmele respective și s-au introdus în aplicație. Această testare(Figura 4.1.) are o vedere în ansamblu a reciclării orașului nostru.

Primul pas în realizarea acestei testări a fost colectarea informațiilor publice disponibile despre firmele de reciclare din oraș. Aceste informații au inclus:

- Date de contact ale firmelor
- Tipurile de materiale reciclate
- Locațiile centrelor de reciclare

Rezultatele testării oferă o bază solidă pentru a înțelege situația actuală a reciclării în oraș și pentru a lua măsuri concrete de îmbunătățire. Ca principală problemă pentru conceptul aplicației

este numărul mic de firme. Extinderea rețelei de centre de reciclare reprezintă deschiderea de noi centre în zonele identificate ca având nevoie de servicii suplimentare.

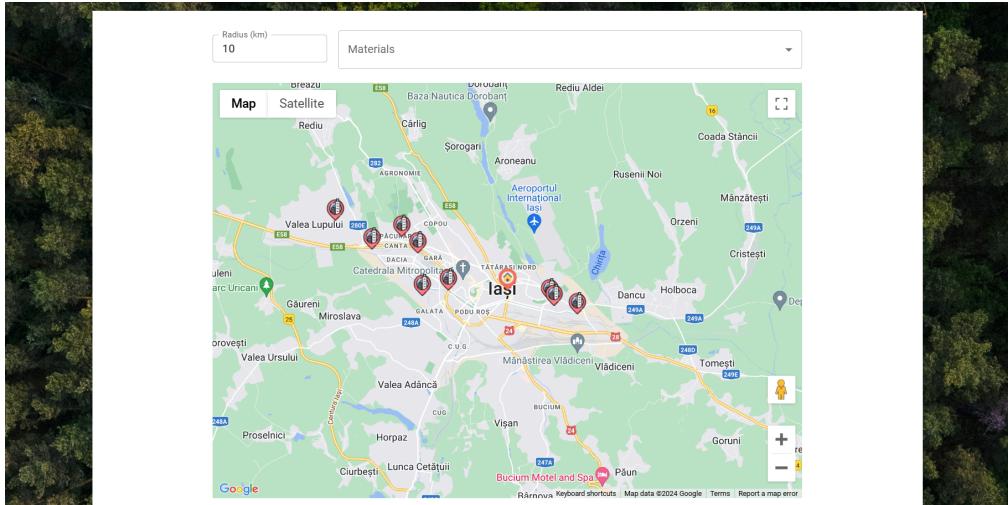


Figura 4.1. Firme reale de reciclare din Iași<sup>32</sup>

## 4.2. Completări viitoare ale aplicației

Pentru a continua să inspire și să motiveze utilizatorii, aplicația trebuie să adauge anumite completări, idei inovatoare. Aceste completări viitoare au scopul de a transforma aplicația într-o platformă și mai robustă și interactivă, oferind utilizatorilor instrumentele necesare pentru a face alegeri ecologice informate și pentru a se conecta cu o comunitate dedicată protejării mediului.

### 4.2.1. Comunicarea între client și companie

Comunicarea între client și companie trebuie să fie făcută pentru a realiza o comunitate placută. Această comunicare ar putea fi făcută prin următoarele idei:

- o secțiune de știri despre ecologizare și modul de a recicla pentru a proteja mediul și a ajuta utilizatorii să conștientizeze importanța reciclării și a salvării planetei. O serie de articole educative prin care se învăță cum se reciclează corect.
- să fie implementat un sistem de recompense, să fie un clasament pentru a motiva utilizatorii să recicleze cât mai mult și să promovăm aplicația. Acest clasament să existe și pentru companii pentru a se îmbunătății calitatea relației client-companie.
- toate evenimentele cererii să fie comunicate prin email, utilizatorii să primească mailuri când se acceptă și când se finalizează preluarea deșeurilor. Desigur și companiile să primească mailuri când apare o nouă cerere pe raza lor de acțiune.

<sup>32</sup> Figura 4.1. Firme reale de reciclare din Iași

- introducerea unor notificări și mementouri care să amintească utilizatorilor să colecteze și să depoziteze selectiv deșeurile.
- generarea de statistici și rapoarte personalizate privind cantitatea de deșeuri reciclate de fiecare utilizator, precum și impactul pozitiv asupra mediului.
- oferirea de informații despre evenimentele și campaniile ecologice care au loc în apropierea utilizatorului, cum ar fi acțiuni de curățenie, plantări de copaci

#### **4.2.2. Securizarea aplicației**

În era digitală contemporană, securitatea este o preocupare primordială atât pentru indivizi, cât și pentru organizații. Amenințările cibernetice devin tot mai sofisticate și frecvente, companiile mari investesc foarte mulți bani în acest domeniu, deoarece ar exista pierderi financiare foarte mari sau furtul de informații financiare. Din cauza importanței datelor personale și câte lucruri rele se pot întâmpla cu acestea, cum ar fi furtul de identitate și frauda, este foarte importantă securizarea acestor date. Câteva idei pentru a realiza securitatea aplicației:

- autentificarea utilizatorilor să fie securizată cu ajutorul unor token-uri JWS
- când se creează un cont, să fie o verificare prin email sau un mesaj pe telefon, generând un cod unic pe care utilizatorul trebuie să îl introducă în aplicație.

Fiind un domeniu foarte vast, există multe concepte de securitate a aplicației. Acestea sunt doar câteva concept.

## Concluzie

Aplicația "EcoConnect" a fost creată în conformitate cu planul de proiectare stabilit inițial, respectând tema și obiectivele propuse. Prin atingerea tuturor acestor obiective, aplicația se dovedește a fi o metodă de încredere a ecologizării, sprijinind atât companiile, cât și utilizatorii să protejeze mediul înconjurător și să cultive dragostea pentru natură cu doar un click distanță.

După cum a fost prezentat și în lucrare, ideea aplicației este ca utilizatorul să poată trimite cereri la cele mai apropiate firme de reciclare specializate pe respectivul material, iar acestea să asigure utilizatorul că toată metoda de reciclare o să se ducă la bun sfârsit. Acest tip de aplicație asigură o data, clienții că nu trebuie să transporte ei deșeurile la anumite centre de colectare, și companiile prin garanția că raza pe care trebuie să o acopere este mică, asta însemnând îmbunătățirea eficienței gestionării deșeurilor.

Beneficiile utilizatorilor prin utilizarea aplicației, aceștia nu mai trebuie să se deplaseze până la centrele de reciclare. Tot ce trebuie să facă este să trimită o cerere prin intermediul aplicației, iar firma de reciclare va veni să preia deșeurile de la gospodăria acestuia. Eliminarea necesității de a transporta personal deșeurile reduce semnificativ timpul investit în procesul de reciclare.

Beneficiile companiilor prin utilizarea aplicației, acestea pot optimiza rutele de colectare datorită localizării precise a cererilor, reducând astfel distanțele parcuse și economisind resurse. Companiile pot asigura utilizatorii că materialele vor fi reciclate corespunzător, crescând astfel încrederea în procesul de reciclare. Prin afilierea cu aplicația, companiile de reciclare își pot crește vizibilitatea și baza de clienți, atrăgând mai multe cereri de servicii.

## Bibliografie

- [1] Sen. Gaylord Nelson's Earth Day Speech  
<https://doorcountypulse.com/sen-gaylord-nelsons-earth-day-speech/>
- [2] Chief Seattle's Letter <https://www.csun.edu/~vcpsy00h/seattle.htm>
- [3] Eurostat European Commission, "Waste recycling statistics in the European Union"  
<https://ec.europa.eu/eurostat/en/web/products-eurostat-news/w/ddn-20240208-2>
- [4] Veolia Group. „About Veolia.” <https://www.veolia.com/en>
- [5] Recapp by Veolia. „How Recapp Works.” <https://www.gorecapp.com/>
- [6] Technical Review Middle East <https://technicalreviewmiddleeast.com/manufacturing>
- [7] RecycleNation: Search. Find. Recycle <https://recyclenation.com/>
- [8] Recycle Nation, Articles <https://recyclenation.com/blog/>
- [9] Distance on a sphere: The Haversine Formula [Esri Community](#)
- [10] Distance on an ellipsoid: Vincenty's Formulae [Esri Community](#)
- [11] Medium, Comparing the Haversine and Vincenty Algorithms for Calculating Great-Circle  
<https://medium.com/@herihermawan/comparing-the-haversine-and-vincenty-algorithms-for-calculating-great-circle-distance-5a2165857666>
- [12] TechTarget, Advanced Encryption Standard  
<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>
- [13] SHA-256 Algorithm - N-able <https://www.n-able.com/it/blog/sha-256-encryption>
- [14] Google, Google Maps, Maps JavaScript API  
<https://developers.google.com/maps/documentation/javascript/overview>
- [15] Hibernate JavatPoint <https://www.javatpoint.com/hibernate-tutorial>
- [16] IBM, What is PostgreSQL? <https://www.ibm.com/topics/postgresql>
- [17] IBM, What is Java SpringBoot? <https://www.ibm.com/topics/java-spring-boot>
- [18] HubSpot Blog, What is React.js? <https://blog.hubspot.com/website/react-js>
- [19] MDN Web Docs, HTTP <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- [20] MDN Web Docs, CORS <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

## Anexe

### Anexa1 - SpringBoot

```

Controller

@RestController
@CrossOrigin
@RequestMapping(value = "/api/companies")
public class CompanyController {

    private Validator validator;

    public CompanyController() {
        ValidatorFactory validatorFactory =
Validation.buildDefaultValidatorFactory();
        validator = validatorFactory.getValidator(); }

    @PostMapping
    public ResponseEntity saveCompany(@RequestBody @Valid CompanyDTO
companyDTO, BindingResult bindingResult) {
        User user= new User();
        DtoToEntity convertor = new DtoToEntity();
        Company company =
convertor.convertorCompanyDtoToEntity(companyDTO);
        List<Material> materials = new ArrayList<>();

        Map<String, String> validations = checkValidations(companyDTO);
        checkCompanyCodeUnicity(companyDTO, validations);
        checkPhoneNumberUnicity(companyDTO, validations);
        checkEmailUnicity(companyDTO, validations);
        if (!validations.isEmpty()) {
            return new ResponseEntity<>(validations,
HttpStatus.BAD_REQUEST); }
        processMaterialList(companyDTO, materials);
        materialService.saveAllMaterial(materials);
        company.setMaterialList(materials);
        setUserDetails(companyDTO, user);
        company.setUser(user);
        User userFromDatabase =userService.saveUser(user);
        companyService.saveCompany(company);
    }
}

```

```

validations.put("user_id", userFromDatabase.getUserId().toString());
    return new ResponseEntity<>(validations, HttpStatus.OK);
}

//compania după materialele pe care e specializată
@GetMapping(value = "/{id}/company-materials")
public ResponseEntity<?> getCompanyByMaterialName(@PathVariable Long id) {
    Company company = companyService.getCompanyById(id);
    if (company == null) {
        Map<String, String> companyNullMessage = new
HashMap<>();
        companyNullMessage.put("message", "Company not found");
        return new ResponseEntity<?>(companyNullMessage,
HttpStatus.NOT_FOUND);
    } else {
        List<Long> materialIds =
companyService.getMaterialIdsByCompanyId(id);
        return new ResponseEntity<?>(materialIds,
HttpStatus.OK);
    }
    ///valdăm constrângările pe care le avem în companyDTO
    private Map<String, String> checkValidations(CompanyDTO companyDTO) {
        Set<ConstraintViolation<CompanyDTO>> violations =
validator.validate(companyDTO);

        Map<String, String> validations = new HashMap<>();
        for (ConstraintViolation<CompanyDTO> violation : violations) {
            String propertyPath =
violation.getPropertyPath().toString();
            String message = violation.getMessage();
            if (propertyPath.startsWith("materialName[")) {
                propertyPath = "materialName";
            }
            validations.put(propertyPath, message);
        }
        return validations;
    }
    @CrossOrigin(origins = "http://localhost:3000")
    @RequestMapping(value = "/api/materials")
    @RestController
    public MaterialController() {

```

```

    ValidatorFactory validatorFactory =
Validation.buildDefaultValidatorFactory();

    validator = validatorFactory.getValidator();}

@PostMapping

public ResponseEntity saveMaterial(@RequestBody @Valid MaterialDTO
materialDTO, BindingResult bindingResult) {

    DtoToEntity convertor = new DtoToEntity();

    Material material =
convertor.convertorMaterialDtoToMaterialEntity(materialDTO);

    Map<String, String> validations = checkValidations(materialDTO);
    checkMaterialUnicity(materialDTO, validations);

    if (!validations.isEmpty()) {

        return new ResponseEntity<>(validations, HttpStatus.BAD_REQUEST);}

    materialService.saveMaterial(material);

    validations.put("Results", "The material will be created with succes.");
    return new ResponseEntity<>(validations, HttpStatus.OK);}

@GetMapping(value = "/{id}")

public ResponseEntity getMaterialById(@PathVariable Long id){

    Material material = materialService.getMaterialById(id);

    if(material == null){

        return new ResponseEntity("Material not found.", HttpStatus.NOT_FOUND);
    }else{

        return new ResponseEntity(material, HttpStatus.OK);
    }
}

@RestController

@CrossOrigin(origins = "http://localhost:3000")

@RequestMapping(value = "/api/requests")

public class RequestController {

    @GetMapping(value = "/person/{id}/requests-onhold")

    public ResponseEntity getAllByUserIdAndStatusOnHold(@PathVariable Long id) {

        List<RequestListDTO> requestListDTOS = new ArrayList<>();
        DtoToEntity convertor = new DtoToEntity();

        Person person = personService.getByUserId(id);

        if (person == null) {
            Map<String, String> personNullMessage = new HashMap<>();
            personNullMessage.put("message",
MessageContent.USER_NOT_FOUND);
        }
    }
}

```

```

        return new ResponseEntity<>(personNullMessage,
HttpStatus.NOT_FOUND) ;
    }

    return verifyRequestsOnHold(id, requestListDTOS, convertor) ;
}

///verific requesturile care au statusul onhold

private ResponseEntity verifyRequestsOnHold(Long id, List<RequestListDTO>
requestListDTOS, DtoToEntity convertor) {

    List<Request> requestList =
requestService.getAllByPersonUserIdAndStatus(id, Status.ON_HOLD) ;

    if (requestList == null || requestList.size() == 0) {
        return new ResponseEntity<>(Collections.emptyList(),
HttpStatus.OK) ;
    } else {
        for (Request request : requestList) {
            RequestListDTO requestListDTO =
convertor.convertorRequestListEntityToRequestListDTO(request) ;
            requestListDTOS.add(requestListDTO) ;
        }
        return new ResponseEntity<>(requestListDTOS, HttpStatus.OK) ;
    }
}

///actualizeaza requestul in processing din baza de date

private ResponseEntity updateRequestAcceptedMethod(RequestAcceptedDTO
requestAcceptedDto) {

    Request requestFromDb =
requestService.getRequestById(requestAcceptedDto.getId()) ;

    if (requestFromDb == null) {
        Map<String, String> requestNullMessage = new HashMap<>() ;
        requestNullMessage.put("message",
MessageContent.REQUEST_NOT_FOUND) ;
        return new ResponseEntity<>(requestNullMessage,
HttpStatus.NOT_FOUND) ;
    } else {
        requestFromDb.setStatus(requestAcceptedDto.getStatus()) ;
        requestService.saveRequest(requestFromDb) ;
    }
    Map<String, String> requestUpdatedMessage = new HashMap<>() ;
    requestUpdatedMessage.put("message",MessageContent.REQUEST_UPDATED) ;
    return new ResponseEntity<>(requestUpdatedMessage, HttpStatus.OK) ;
}

```

```

@RestController
@RequestMapping(value = "/api/users")
@CrossOrigin(origins = "http://localhost:3000")
public class UserController {
    @PostMapping(value = "/login")
    public ResponseEntity login(@RequestBody(required = false) UserLoginDTO
        userLoginDTO) {
        if (checkLoginFields(userLoginDTO))
            return new ResponseEntity<>(MessageContent.LOGIN_ERROR,
        HttpStatus.BAD_REQUEST);
        User user = userService.getByEmail(userLoginDTO.getEmail());
        return checkUserExistence(userLoginDTO, user);
    }

    @GetMapping(value = "/nearby-users-for-company/{id}")
    public ResponseEntity getNearbyUsersForCompany(@PathVariable Long id) {
        Company company = companyService.getById(id);
        if (checkCompanyExist(company)) return new ResponseEntity<>("User not
        found.", HttpStatus.NOT_FOUND);
        Double companyLatitude = Double.parseDouble(company.getLatitude());
        Double companyLongitude = Double.parseDouble(company.getLongitude());
        List<PersonNearbyDTO> personNearbyDTOS = setNearbyPerson(companyLatitude,
        companyLongitude);
        List<CompanyDetailPracticeDTO> companyDetailPracticeDTOS =
        setNearbyCompanies(companyLatitude, companyLongitude);
        List<Object> combinedList = new ArrayList<>();
        combinedList.addAll(companyDetailPracticeDTOS);
        combinedList.addAll(personNearbyDTOS);
        return new ResponseEntity<>(combinedList, HttpStatus.OK);
    }
    ///luam cele mai apropiate persoanele care au statusul onhold
    private List<PersonNearbyDTO> setNearbyPerson(Double latitude, Double
        longitude) {
        List<Person> nearbyPerson =
        distanceBetweenUsers.getAllNearbyPersonWithStatusOnHold(latitude, longitude);
        List<PersonNearbyDTO> personNearbyDTOList = new ArrayList<>();
        DtoToEntity convertor = new DtoToEntity();
        for (Person person : nearbyPerson) {
            PersonNearbyDTO personNearbyDTO =
            convertor.convertorPersonEntityToPersonNearbyDTO(person);
        }
    }
}

```

```
User user = userService.getByPersonId(person.getPersonId());
List<String> materials = new ArrayList<>();
for (Request request :
requestService.getAllByPersonId(person.getPersonId())) {
    Material material = request.getMaterial();
    if (!materials.contains(material.getMaterialName())) {
        materials.add(material.getMaterialName());}}
personNearbyDTO.setRequestsMaterialsPerson(materials);
personNearbyDTO.setId(user.getUserId());
personNearbyDTOList.add(personNearbyDTO);}
return personNearbyDTOList;
}
```

### Repository

```
@Repository
public interface CompanyRepository extends JpaRepository<Company, Long> {
    Company findByCompanyId(Long companyId);
    Company findByCompanyCode(String companyCode);
    Company findByCompanyNumberPhone(String companyNumberPhone);
    Company findByUserUserId(Long userId);
    @Query("select c.id from Company c join c.materialList ml where
ml.materialName = :materialName")
    List<Long> findByMaterialName(String materialName);
    @Query("select m.id from Material m join m.companyList cl where cl.id =
:companyId")
    List<Long> findMaterialIdsByCompanyId(Long companyId);
}
```

### ServiceImplement

```
@Service
public class CompanyServiceImplement implements CompanyService {
    @Autowired
    private CompanyRepository companyRepository;
    @Override
    public Company getByUserId(Long userId) {
        return companyRepository.findByUserUserId(userId);}
    @Override
    public Company getCompanyById(Long companyId) {
```

```

        return companyRepository.findByCompanyId(companyId) ;}

@Override
public Company getCompanyByCompanyCode(String companyCode) {
    return companyRepository.findByCompanyCode(companyCode) ;}

@Override
public Company getCompanyByNumberPhone(String companyNumberPhone) {
    return companyRepository.findByCompanyNumberPhone(companyNumberPhone) ;}

@Override
public Company saveCompany(Company company) {
    return companyRepository.save(company) ;}

@Override
public List<Long> getCompanyByMaterialName(String materialName) {
    return companyRepository.findByMaterialName(materialName) ;}

@Override
public List<Long> getMaterialIdsByCompanyId(Long companyId) {
    return companyRepository.findMaterialIdsByCompanyId(companyId) ;}

}

Service

@Service
public interface CompanyService {
    Company getCompanyById(Long companyId) ;
    Company getByUserId(Long userId) ;
    Company getCompanyByCompanyCode(String companyCode) ;
    Company getCompanyByNumberPhone(String companyNumberPhone) ;
    Company saveCompany(Company company) ;
    List<Long> getCompanyByMaterialName(String materialName) ;
    List<Long> getMaterialIdsByCompanyId(Long companyId) ;
}

Entity

@Table(name = "company")
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "company_id")
}

```

```

private Long companyId;
@Column(name = "company_name", nullable = false)
private String companyName;
@Column(name = "company_description", nullable = false)
private String description;
@Column(name = "company_address", nullable = false)
private String companyAddress;
@Column(name = "company_number_phone", unique = true, nullable = false)
private String companyNumberPhone;
@Column(name = "company_code", unique = true, nullable = false)
private String companyCode;
@Column(name = "company_latitude", nullable = false)
private String latitude;
@Column(name = "company_longitude", nullable = false)
private String longitude;
@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
@JoinTable(
    name = "material_company",
    joinColumns = @JoinColumn(name = "company_id"),
    inverseJoinColumns = @JoinColumn(name = "material_id")
)
private List<Material> materialList = new ArrayList<>();
@OneToMany(mappedBy = "company", fetch = FetchType.LAZY)
private List<Request> requestList;

```

**DTO**

```

public class PersonDTO {
    @NotNull(message = "Required")
    @Size(max = 30, message = "First name provided does not adhere to the
specified field validation rules")
    @Pattern(message = "First name provided does not adhere to the specified
field validation rules", regexp = "[A-Z][a-z]*$")
    private String firstName;

    @NotNull(message = "Required")
    @Size(max = 30, message = "Last name provided does not adhere to the
specified field validation rules")
    @Pattern(message = "Last name provided does not adhere to the specified
field validation rules", regexp = "[A-Z][a-z]*$")

```

```

private String lastName;

@NotNull(message = "Required")
@Size(max = 50, message = "Email provided does not adhere to the specified
field validation rules")
@Pattern(regexp = "^( [a-zA-Z0-9]+@[a-z]+\\.[a-z]+)$", message = "Email
provided does not adhere to the specified field validation rules")

private String email;

@NotNull(message = "Required")
@Size(min = 8, max = 50, message = "Password provided does not adhere to
the specified field validation rules")
private String password;

```

## Anexa2 - React

```

Service.js
//functii cu API-urile apelate
import axios from 'axios';
const API_BASE_URL = 'http://localhost:8082/api';
const fetchRequestByCompanyId = (userId) => {
    return axios.get(`${API_BASE_URL}/users/company-id/${userId}`);
};

const fetchAllMaterials = () => {
    return axios.get(`${API_BASE_URL}/materials`);
};

const fetchCompaniesByMaterial = (material) => {
    return axios.get(`${API_BASE_URL}/companies/material/${material}`);
};

const fetchRequestsAccepted = (userId) => {
    return axios.get(`${API_BASE_URL}/companies/${userId}/accepted-request`);
};

```

PersonPage.js

```

useEffect(() => {
    if (user && user.id) {
        fetchRequestByPersonId(user.id)
            .then(response => {
                if (response.status === 200) {
                    setRequests(response.data);
                }
            })
    }
});

```

```

        console.log('Fetched requests:', response.data);
    })
    .catch(error => console.error('Failed to fetch requests:',
error));
}

if (user.latitude && user.longitude) {
    setCenter({
        lat: parseFloat(user.latitude),
        lng: parseFloat(user.longitude)
    });
}

fetchNearbyCompaniesRadius(user.id, radius)
    .then(response => {
        if (response.status === 200) {
            setCompanies(response.data);
        }
    })
    .catch(error => console.error('Failed to fetch nearby
companies:', error));
}

const handleRadiusChange = (event) => {
    setRadius(event.target.value);
};

const handleMaterialChange = (event) => {
    const materialName = event.target.value;
    setSelectedMaterial(materialName);

    if (materialName === '') {
        fetchNearbyCompaniesRadius(user.id, radius)
            .then(response => {
                if (response.status === 200) {
                    setCompanies(response.data);
                    console.log('Fetched companies:', response.data);
                } else {
                    console.error('Failed to fetch nearby companies:',
response.status, response.statusText);
                }
            })
            .catch(error => console.error('Failed to fetch nearby
companies:', error));
    } else {
        fetchCompaniesByMaterial(materialName)
            .then(response => {
                if (response.status === 200) {

```

```

        const companyIds = response.data;
        console.log("Company IDs:", companyIds);
        return Promise.all(companyIds.map(id =>
fetchCompanyId(id)));
    } else {
        throw new Error('Failed to fetch companies by
material');
})
.then(responses => {
    const companiesData = responses.map(response =>
response.data);
    console.log("Fetched companies by material:",
companiesData);
    setCompanies(companiesData);
})
.catch(error => console.error('Failed to fetch companies by
material:', error));
}
};


```

**CompanyPage.js**

```

const fetchCompanyDetails = async (companyId) => {
try {
    const companyData = await fetchCompanyId(companyId);
    if (companyData && companyData.data.latitude &&
companyData.data.longitude) {
        const newCenter = {
            lat: parseFloat(companyData.data.latitude),
            lng: parseFloat(companyData.data.longitude),
        };
        setCenter(newCenter);
        setCompanyData(companyData.data);
        await getMaterialsByCompany(companyId);
    }
} catch (error) {
    console.error('Error fetching company data:', error);
}
};

const getMaterialsByCompany = async (companyId) => {
try {
    const response = await fetchMaterialsByCompany(companyId);

```

```
const materialIds = response.data;
const materialNamesMap = {};
await Promise.all(materialIds.map(async (materialId) => {
  const materialResponse = await fetchMaterialById(materialId);
  materialNamesMap[materialId] = materialResponse.data.materialName;
}));
setSpecializedMaterials(Object.values(materialNamesMap));
setMaterialNames(materialNamesMap);
console.log(specializedMaterials)
} catch (error) {
  console.error('Error fetching company materials:', error);
}
};

useEffect(() => {
  console.log('Specialized Materials Updated:', specializedMaterials);
}, [specializedMaterials]);
const fetchNearbyUsers = async (userId) => {
try {
  const response = await getNearbyUsersForCompany(userId);
  const locations = response.data;
  console.log(locations)
  setNearbyLocations(locations);
  let allFilteredRequests = [];
  for (const location of locations) {

    if (location.role === "Person") {
      try {
        const res = await getAllRequestsOnHold(location.id);
        const requestsOnHold = res.data;
        const filteredRequests = requestsOnHold.filter(request =>
          specializedMaterials.includes(request.materialName)
        );
        allFilteredRequests = [...allFilteredRequests,
...filteredRequests];
      } catch (error) {
        console.error('Error fetching requests on hold:', error);
      }
    }
  }
  setFilterReq(allFilteredRequests);
  console.log("Filtered Requests:", allFilteredRequests);
} catch (error) {
```

```

        console.error('Error fetching nearby locations:', error);
    }
};


```

Functia folosita pentru extragerea numarului si numele strazii

```

const getAddressFromCoordinates = async (lat, lng) => {
    try {
        const response = await
fetch(`https://maps.googleapis.com/maps/api/geocode/json?latlng=${lat},${lng}&
key=${apiKey}`);
        const data = await response.json();
        if (data.status === 'OK') {
            const addressComponents = data.results[0].address_components;
            const address = {
                street: '',
                number: ''
            };
            addressComponents.forEach(component => {
                if (component.types.includes('route')) {
                    address.street = component.long_name;
                }
                if (component.types.includes('street_number')) {
                    address.number = component.long_name;
                }
            });
            setFormData(prevState => ({
                ...prevState,
                ...address,
            }));
        } else {
            throw new Error(`Geocoding API error: ${data.status}`);
        }
    } catch (error) {
        setErrorMessage(`Unable to fetch address from the selected coordinates.
Error: ${error.message}`);
    };
}


```