Sistem informatic de gestiune a unui complex imobiliar

Viziteu Mihai-Cezar

ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI
Facultatea de Contabilitate si Informatica de Gestiune

Coordonator științific: Asistent univ. drd. Laura-Eugenia-Lavinia Barna

Rezumat: Lucrarea de față își propune să prezinte un sistem (software) de gestiune pentru un complex imobiliar, axat în special pe încheierea de contracte de închiriere, folosind pentru dezvoltarea acestuia next.js și susținut de MongoDB pe post de bază de date. Cuprinde funcționalități de gestiune pentru apartamente, locuri de parcare, clienți și contracte. Am observat că multe documente referitoare la activitatea unui complex imobiliar se fac manual, lucru care poate determina apariția erorilor umane în respectivele documente. Acest software face parte dintr-un efort de a digitaliza și automatiza activitatea în birou, având ca focus principal generarea de contracte. Pe viitor, sistemul va include și calculul cheltuielilor, distribuirea facturilor și gestionarea defecțiunilor cu ajutorul unui sistem de semnalare a acestora (sistem de tickets).

Cuvinte-cheie: imobiliare, baze de date, gestiune, contracte, typescript, digitalizare

Abstract: This paper aims to present a management system (software) for a real estate complex, focused specifically on lease contract management, developed using next.js and supported by MongoDB as a database. It includes management functionalities for apartments, parking spots, clients, and contracts. I have noticed that many documents related to the activity of a real estate complex are manually generated, which can lead to human errors in these documents. This software is part of an effort to digitalize and automate office activities, with the primary focus being the generation of contracts. In the future, the system will also include expense calculations, invoice distribution, and defect management with the help of a ticketing system.

Keywords: real estate, databases, bookkeeping, contracts, typescript, digitalization

1. Motivația alegerii

Ideea acestei lucrări mi-a fost propusă inițial când făceam poze la un complex imobiliar. Întâmplarea face ca generarea contractelor pe baza unor șabloane (template-uri) să fie similară cu un alt program pe care l-am făcut cu un an în urmă pentru a centraliza feedback-ul semestrial pe clasă[1]. Specificații mai precise am primit pe 1 ianuarie 2023, moment în care am început să mă documentez în legătură cu ce tehnologii aș putea folosi pentru a ajunge la un rezultat. De asemenea, fiind primul program scris de mine în javascript/typescript, am considerat că este o oportunitate de a învăța tehnologii noi.

2. Descriere

Scopul sistemului realizat de mine este de a digitaliza o parte din gestionarea complexului. Acesta permite adăugarea de apartamente și locuri de parcare, și încheierea contractelor pe baza acestora, utilizând o combinație de Next.js pentru Backend și Frontend pentru interfața cu utilizatorul și MongoDB pentru baza de date, unde sunt stocate efectiv datele introduse in interfață. Deoarece Javascript nu include verificări asupra tipurilor de date, am decis să folosesc Typescript, care este „transpiled”[2], adică după un set de verificări asupra tipurilor de date, compilatorul typescript îl transformă în javascript. Printre avantajele acestei alegeri se numără eliminarea posibilității anumitor probleme cauzate de amestecarea variabilelor de tipuri diferite (exemplu: 1 + „1” = 2, „1” + 1 = 11) și obținerea unei mai bune integrări cu mediul de dezvoltare.

[1] https://github.com/Skiuileuf/ProcesareFeedback

[2] EN: transpiled – procedeul prin care cod sursă într-un limbaj de programare este transformat în cod sursă în alt limbaj de programare (exemplu – Typescript -> Javascript)

3. Proiectarea aplicației (backend si frontend)

3.1. Arhitectura aplicației (Model-View-Controller)

Aplicația este bazată pe un model (en. pattern) numit Model-View-Controller[3], care *„este utilizat de obicei pentru dezvoltarea de interfețe grafice care separă logica din spate în trei elemente interconectate”*.

Cele trei elemente sunt:

- ***Model (Modelul)*** – Structura datelor si datele din aplicație;
- ***View (Vizualizarea)*** – Reprezentarea grafică a datelor din aplicație, atât simplă (grafice, tabele), cat și compusă (pagini web);
- ***Controller (Controlorul)*** – Piesa de legătură între Model și View, conține logica care leagă acțiunile utilizatorului din View cu efectele asupra datelor din Model;

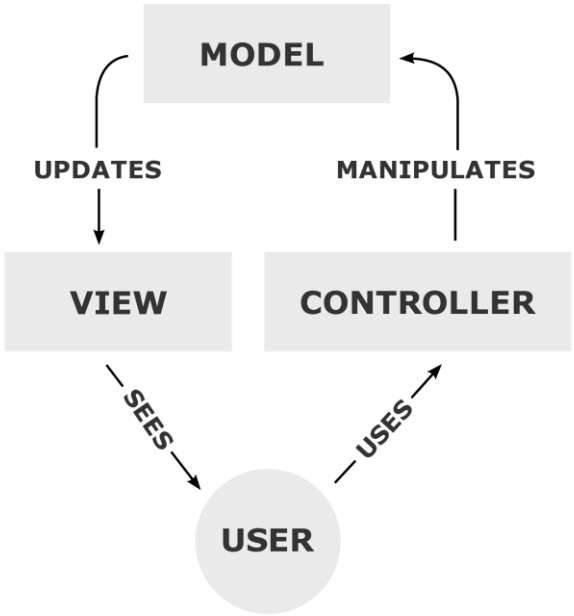

Figura 1. Diagrama interacțiunilor posibile în cadrul patternului Model-View-Controller

Sursa: RegisFrey – Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10298177

Astfel, aplicația se împarte în 3 componente mari care, de principiu, pot să fie desinestătătoare (oricare dintre aceste componente poate fi înlocuită cu alta cât timp menține un API[4] comun).

Atât backend-ul, cât și frontend-ul sunt scrise cu framework-ul Next.js utilizând limbajul typescript. Acest lucru îmi permite să mențin definiții comune ale tipurilor de date între componentele aplicației.

3.2. Backend (Controller)

Backend (*„partea din spate”)* este componenta aplicației care se ocupă cu menținerea unui REST API[5], gestionarea controller-elor și a structurii modelului de date și servirea către utilizator a frontendului.

Un REST API se bazează pe conceptul de resurse (date specifice puse la dispoziția altor aplicații sau servicii web), care sunt accesate prin intermediul unor adrese unice numite URI[6]. Utilizarea acestuia se face prin trimiterea de cereri HTTP, care pot fi de tipul GET (pentru a

[3] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
[4] Application Programming Interface – interfata prin care doua (sau mai multe) programe comunica
[5] Representational State Transfer – interfața care utilizează protocolul HTTP pentru a comunica cu alte aplicații.
[6] Uniform Resource Identifier – Identificator uniform al resursei

prelua date), POST (pentru a adăuga date), PUT (pentru a actualiza date) sau DELETE (pentru a șterge date). Aceste cereri se suprapun cu operațiile CRUD – Create, Read, Update, Delete.

Aplicația prevede URI pentru fiecare tip de resursă importantă (apartament, loc de parcare, client, contract). Acestea sunt accesibile pe ruta „[domeniu]/api/[nume resursă în engleză]” (exemplu: http://localhost:3000/api/apartments).

Tabelul 1

Metode HTTP disponibile în cazul rutei /api/apartments

Metoda	URI	Descriere	Request Body	Response Body
GET	/api/apartments	Obține toate apartamentele		Listă cu toate apartamentele
POST	/api/apartments	Adaugă un apartament nou	Obiect care reprezintă un apartament	Obiect care reprezintă apartamentul creat (și care conține _id)
PUT	/api/apartments	Actualizează toate apartamentele în masă	Operațiunea nu este permisă	
DELETE	/api/apartments	Șterge toate apartamentele	Operațiunea nu este permisă	
GET	/api/apartments/[id]	Obține un apartament după id		Obiect care reprezintă un apartament
POST	/api/apartments/[id]	Adaugă un apartament după id	Operațiunea nu este permisă (id-ul este dat de baza de date, care se asigură că este unic)	
PUT	/api/apartments/[id]	Actualizează un apartament după id	Obiect care reprezintă un apartament	Obiect care reprezintă apartamentul cu datele actualizate
DELETE	/api/apartments/[id]	Șterge un apartament după id		Obiect care reprezintă apartamentul care a fost șters

Sursa: Prelucrarea autorului

Datele trimise prin intermediul unui REST API ajung la controllerul corespunzător tipului de resursă. De exemplu, datele din Tabelul 1 sunt trimise către apartmentController.ts. Acesta preia datele din solicitare (*en. request*), realizează operațiuni asupra modelului și întoarce un răspuns (*en. response*).

```
export async function getAllApartments(req: NextApiRequest, res: NextApiResponse) {
    try {
        const filter = req.body.filter || {};
        const projection = req.body.projection || {};
        //Get all apartments from the database
        const apartments = await ApartmentModel.find(filter, projection);
        //If there are no apartments, return an empty array
        if (!apartments) return res.status(204).json({ success: true, data: {} });
        //Return the apartments
        res.status(200).json(apartments);
    } catch (error) {
        res.status(400).json(error);
    }
}
```

Figura 2. Funcția din apartmentController care returnează toate apartamentele

Sursa: Prelucrarea autorului

```
78
79  export async function deleteClient(req: NextApiRequest, res: NextApiResponse) {
80      try {
81          const id = req.query.id;
82          //If there is no id, return the Bad Request status code
83          if (!id) return res.status(400).json({ message: "Id is required" });
84          //Find the client by id
85          const client = await ClientModel.findOne({ _id: id });
86          //If there is no client, return the Not Found status code
87          if (!client) return res.status(404).json({ message: "Client not found" });
88          //Delete the client
89          const result = await client.remove();
90          //Return the client
91          res.status(200).json(result)
92      } catch (error) {
93          res.status(400).json(error)
94      }
95  }
```

Figura 3. Funcția din clientController care șterge un client

Sursa: Prelucrarea autorului

Structura modelului de date este dată de clase aflate în folderul model. Având o bază de date NoSQL, bazată pe documente, definiția unui obiect poate conține și vectori de elemente.

```
39
40  interface Apartment extends Base<string> { };
41
42  class Apartment {
43      @prop()
44      public _id!: string;
45
46      @prop()
47      apartmentNumber?: string;
48
49      @prop()
50      blockNumber?: string;
51
52      @prop()
53      floorNumber?: number;
54
55      @prop()
56      cadastralNumber?: string;
57
58      @prop()
59      interiorSurfaceArea?: number;
60
61      @prop()
62      balconySurfaceArea?: number;
63
64      @prop({type: () => [String] })
65      media?: string[];
66
67      @prop({type: () => [InventoryItem] })
68      inventory?: InventoryItem[];
69
70      @prop({type: () => [DamageItem]})
71      damages?: DamageItem[];
```

Figura 4. Modelul unui apartament

Sursa: Prelucrarea autorului

3.3. Frontend (View)

Frontend-ul ("partea din față") este interfața grafică cu care interacționează utilizatorul. Acesta este servită de backend din folderul „pages" și realizată folosind librăria React. Pe deasupra, proiectul folosește librăria ant.design pentru componente „gata-făcute" și care au un design comun. Frontend-ul comunică cu backend-ul prin intermediul REST API-ului menționat mai sus.

Aplicația este sub forma unui SPA[7]. Avantajul este că toate paginile sunt încărcate inițial, acest lucru făcând navigarea ulterioară foarte fluidă.

[7] Single Page Application

Frameworkul utilizat (Next.js) prevede trei tipuri de randare a unei pagini:

- ***Server-Side Rendering*** – folosit pentru date care se schimbă rar, pagina vine cu conținut de la server;
- ***Static Site Generation*** – folosit pentru date care nu se schimbă niciodată și nu necesită un server;
- ***Client-Side Rendering*** – folosit pentru date care se schimbă des, conținutul paginii este determinat pe client.

Ultima dintre cele enumerate mai sus este metoda utilizata în cadrul acestei aplicații. Informațiile din pagini sunt completate folosind SWR. SWR (*Stale While Revalidate*) este o tehnică de a obține date din backend și de a păstra un cache. Aceasta presupune ca mai întâi, să se folosească datele locale (din cache), apoi se face un request pentru a obține datele noi.

O pagina in React este formată din componente – fragmente de cod care își stochează propria stare și apeluri de funcții. Practic, o pagina conține componente care își ajustează conținutul în funcție de stare. Componentele transmit date *„în jos”* (spre copii) și apeluri de funcții *„în sus”* (spre părinți). Acestea sunt scrise într-o combinație de HTML si Typescript numita Typescript Extended (.tsx).

```
pages > apartments > index.tsx > ApartmentsPage
20  export default function ApartmentsPage() {
21      const router = useRouter();
22      //data: Apartment[], error: any, isLoading: bool, mutate: KeyedMutator<Apartment[]>
23      const { data: apartments, error, isLoading, mutate }: SWRResponse<Apartment[]> = useSWR('/api/apartments');
24      const developer = useContext(DeveloperContext);
25
26      if (isLoading) return <Loader />;
27      if (error || !apartments) return <Error errorMessage={error} />
28      return (
29          <>
30              <PageComponent>
31                  <Space size='middle' style={{
32                      paddingBottom: 24,
33                  }}>
34                      <h2>Complexul contine {apartments.length} apartamente.</h2>
35                      <ApartmentModal mutate={mutate} />
36                  </Space>
37
38                  <Table dataSource={apartments} size="middle" rowKey="_id">
39                      <Column title="Numar apartament" dataIndex="_id" render={(text, record: Apartment) => (<Link href={`/apartments/${record._id}`}>{record.apartmentNumber}</Link>)} />
40                      <Column title="Numar bloc" responsive={['md']} dataIndex="blockNumber"
41                          //charCodeAt is used to get the ascii value of the first letter
42                          sorter={(a: Apartment, b: Apartment) => (a.blockNumber || " ").charCodeAt(0) - (b.blockNumber || " ").charCodeAt(0)}
43                          //Get all distinct values from the dataSource
44                          filters={
45                              (apartments.map((apartment: Apartment) => (apartment.blockNumber!))).filter(
46                                  function (elem: string, index: number, self: string[]) {
47                                      return index === self.indexOf(elem);
48                                  }).map((blockNumber: string) => ({ text: blockNumber, value: blockNumber }))}
49                          onFilter={(value: any, record: Apartment) => { return record.blockNumber?.indexOf(value) === 0 }}
50                          filterMode="menu" />
51                      <Column title="Etaj" responsive={['lg']} dataIndex="floorNumber" />
52                      <Column title="Numar cadastral" responsive={['lg']} dataIndex="cadastralNumber"
53                          sorter={(a: Apartment, b: Apartment) => String(a.cadastralNumber).length - String(b.cadastralNumber).length} />
54                      <Column title="Suprafata interioara (m&#xB2;)" responsive={['xl']} dataIndex="interiorSurfaceArea" />
55                      <Column title="Suprafata balcon (m&#xB2;)" responsive={['xl']} dataIndex="balconySurfaceArea" />
56                      {/* <Column title="Media" responsive={['xl']} render={(text, record: IApartment) => (<p>{record.media.length}</p>)} />*/}
57                      <Column title="Obiecte de inventar" responsive={['xxl']} render={(text, record: Apartment) => (<p>{record.inventory?.length}</p>)} />
58                      <Column title="Daune" responsive={['xxl']} render={(text, record: Apartment) => (<p>{record.damages?.length}</p>)} />
```

Figura 5. Fragment din codul care generează pagina cu toate apartamentele

Sursa: Prelucrarea autorului

```
export default function CompanyInput({ isForeign }: CompanyInputProps) {
    return (
        <>
            <Divider orientation="left" orientationMargin="0">Persoana Juridica</Divider>
            <AddressInput isForeign={isForeign} text={"Adresa Companie"}/>

            <Divider orientation="left" orientationMargin="0">Detalii Persoana Juridica</Divider>
            <Form.Item label="Titlu Reprezentant (Imputernicit, Administrator etc.)" name="representantTitle">
                <Input />
            </Form.Item>
            <Form.Item label="Tara" name="country">
                <Input />
            </Form.Item>
            <Form.Item label="Denumire Firma" name="companyName">
                <Input />
            </Form.Item>
            <Form.Item label="Inmatriculata La" name="registeredBy">
                <Input />
            </Form.Item>
            <Form.Item label="Numar de ordine la Registrul Comertului (J)" name="tradeRegisterRegistrationNumber">
                <Input />
            </Form.Item>
            <Form.Item label="CUI / CIF" name="identificationCode">
                <Input />
            </Form.Item>
        </>
    );
}
```

Figura 6. Componenta care declara formularul cu datele unei firme

Sursa: Prelucrarea autorului

4. Proiectarea bazei de date (Model)

Aplicația folosește o bază de date MongoDB. Aceasta se axează pe conceptul de documente în locul înregistrărilor (records). Documentele pot conține, pe lângă tipuri de date simple (șiruri de caractere, numere, valori de tip boolean) și vectori, obiecte și referințe către alte documente. Mai multe documente formează o colecție, iar fiecare document respectă o schemă, care este partea din modelul de date definită de backend.

În MongoDB, relațiile intre documente sunt diferite de cele din SQL. Astfel, o relație 1:1 se reprezintă prin stocarea unui document ca subdocument al celuilalt sau o referința către un document (folosind câmpul _id). În cazul 1:n, un document stochează un vector cu n documente sau referințe. În cazul m:n, se stochează vectori cu referințele în toate documentele implicate.

Documentele folosesc pe post de cheie primară câmpul _id. Acesta este un șir hexadecimal generat în funcție de timpul la care a fost adăugat documentul și este făcut să fie unic. (de exemplu, un _id poate arăta așa: 63f7ce2b0f162c919876f7f9). Referințele conțin id-uri pentru documente din alte colecții.

Un driver MongoDB are posibilitatea să *„populeze”* câmpurile cu referințe, transformându-le în datele la care se face referire.

```
_id: 56
tenants: Array
  0: "63f7ce350f162c919876f7ff"
  1: "63f7ce370f162c919876f802"
parkingSpaces: Array
rentAmount: 99900
__v: 3
apartment: "f3"
rentStartDate: 2023-03-09T12:52:24.062+00:00
rentLength: 12
empowered: "Popescu Ion"
```


Figura 7. Documentul care reprezintă un contract. Arată referințele către chiriași și apartament.
Sursa: Prelucrarea autorului

Printre avantajele utilizării unei baze de date cu documente se numără viteza, scalabilitatea, cât și flexibilitatea în ceea ce privește datele care pot fi stocate.

Mențiuni importante referitoare la cum sunt stocate anumite date sunt prezentate mai jos:

- Valoarea chiriei este stocată în cea mai mică subdiviziune posibilă – eurocenți. Asta evită situații care apar din cauza modului în care sunt stocate numerele cu virgula (0.1 + 0.2 = 0.3000000004).
- Sistem universal de stocare al adreselor, care funcționează pentru orice tara. (bazat pe standardul xAL)

5. Utilizarea aplicației

Aplicația este gândita să fie intuitivă de folosit, având o interfață ușor de utilizat și oferind o serie de funcții care să faciliteze gestionarea mult mai eficientă a complexului imobiliar.

5.1. Operații cu apartamente

Aplicația permite vizualizarea tuturor apartamentelor, adăugarea, editarea și obținerea datelor unui apartament și a contractului asociat (dacă există).

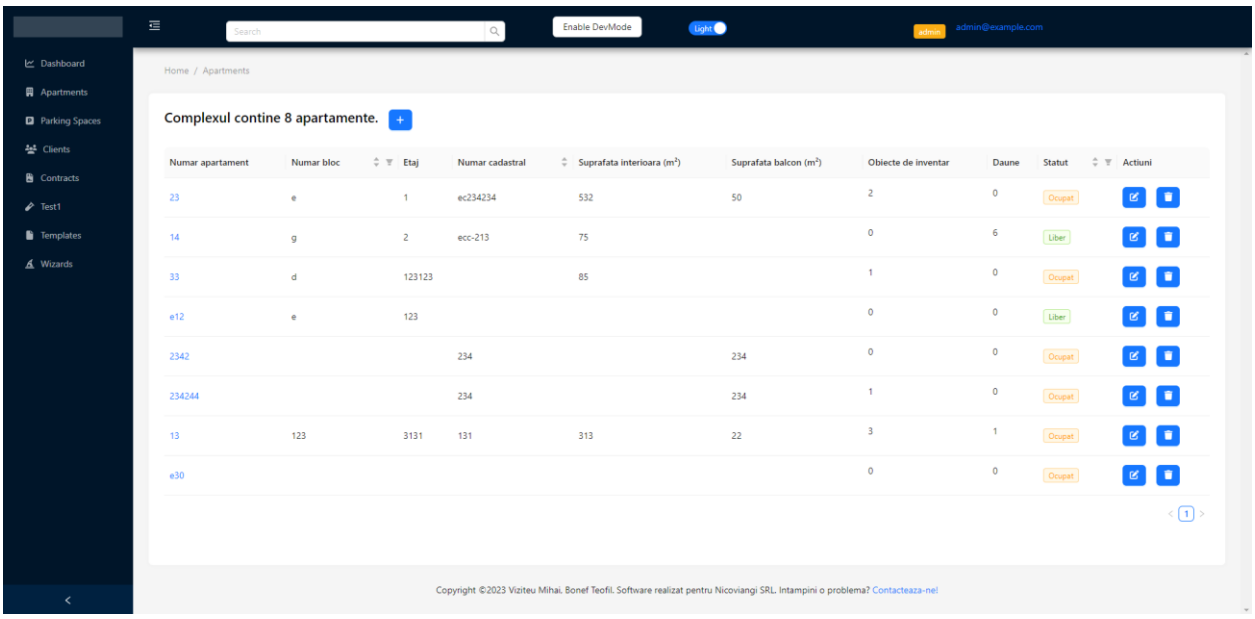

Figura 8. Pagina care conține o listă cu toate apartamentele
Sursa: Prelucrarea autorului

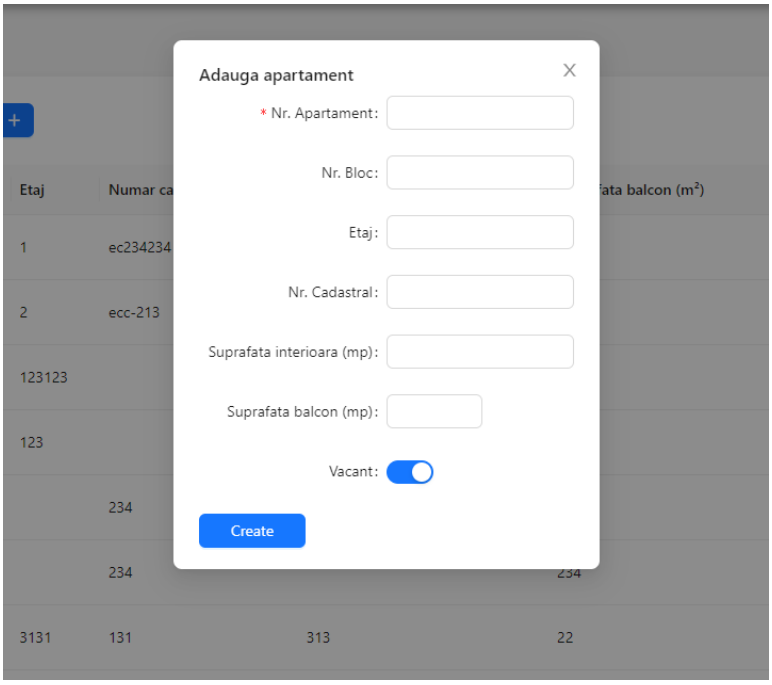

Figura 9. Formularul de adăugare al unui apartament nou
Sursa: Prelucrarea autorului

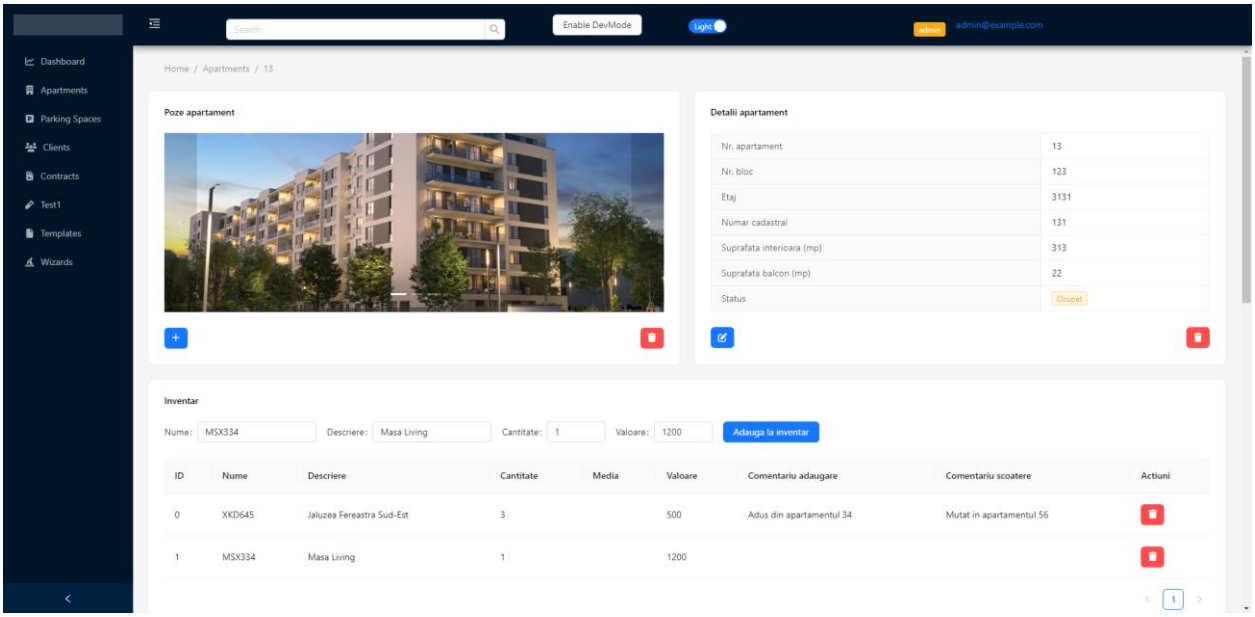

Figura 10. Pagina unui apartament
Sursa: Prelucrarea autorului

5.2. Operații cu clienți

Aplicația permite vizualizarea tuturor clienților, adăugarea, editarea și obținerea datelor unui client

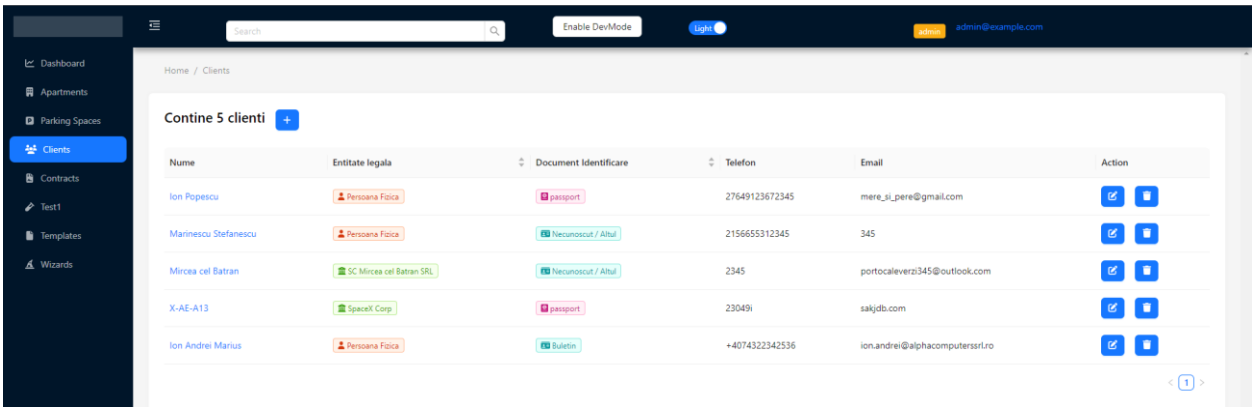

Figura 11. Pagina care conține o listă cu toți clienții
Sursa: Prelucrarea autorului

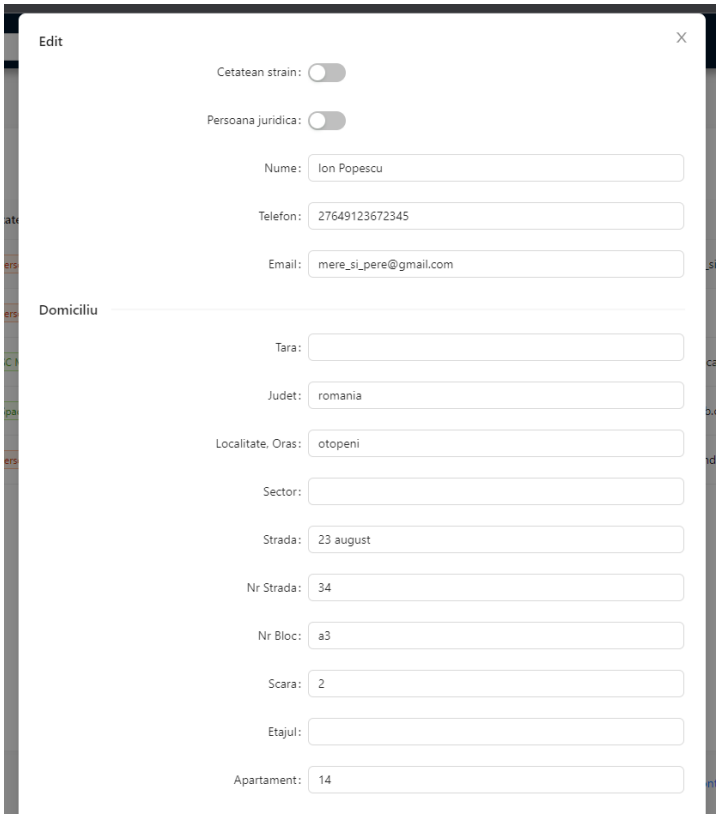

Figura 12. Formularul de adăugare al unui client nou
Sursa: Prelucrarea autorului

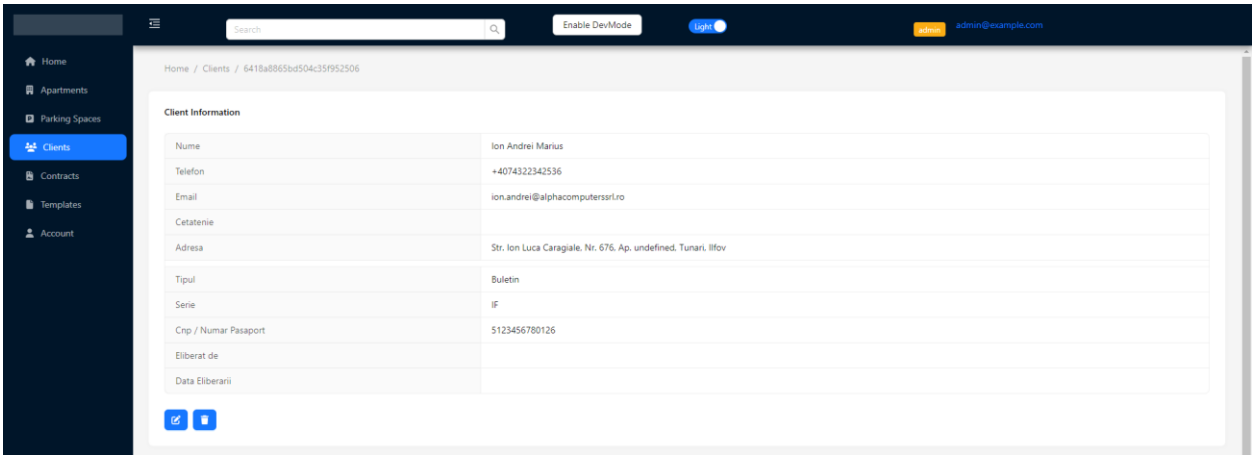

Figura 13. Pagina cu datele unui client
Sursa: Prelucrarea autorului

5.3. Operații cu contracte

Aplicația permite vizualizarea tuturor contractelor, adăugarea, editarea, obținerea datelor unui contract, cât și descărcarea acestuia ca document word.

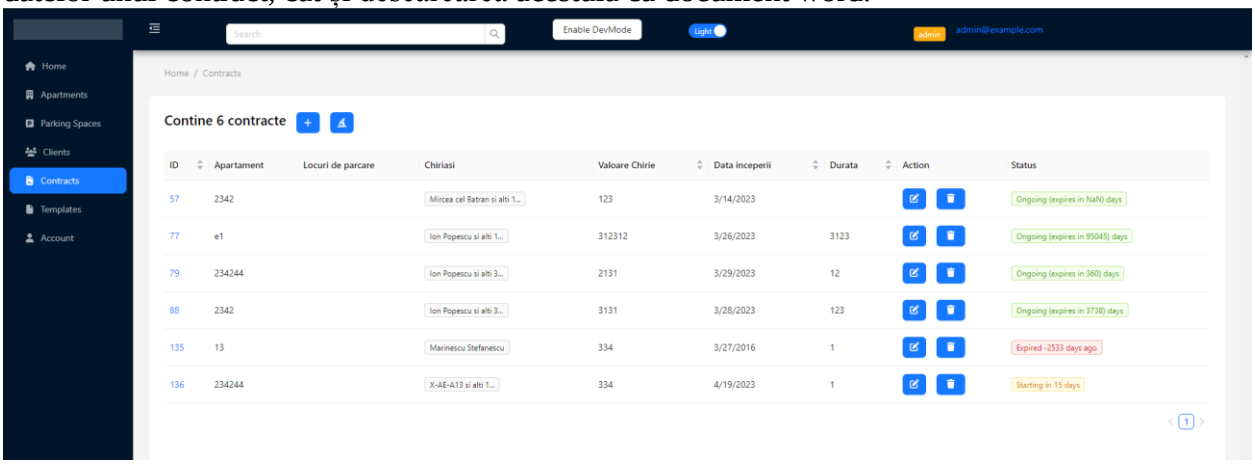

Figura 14. Pagina care conține o listă cu toate contractele
Sursa: Prelucrarea autorului

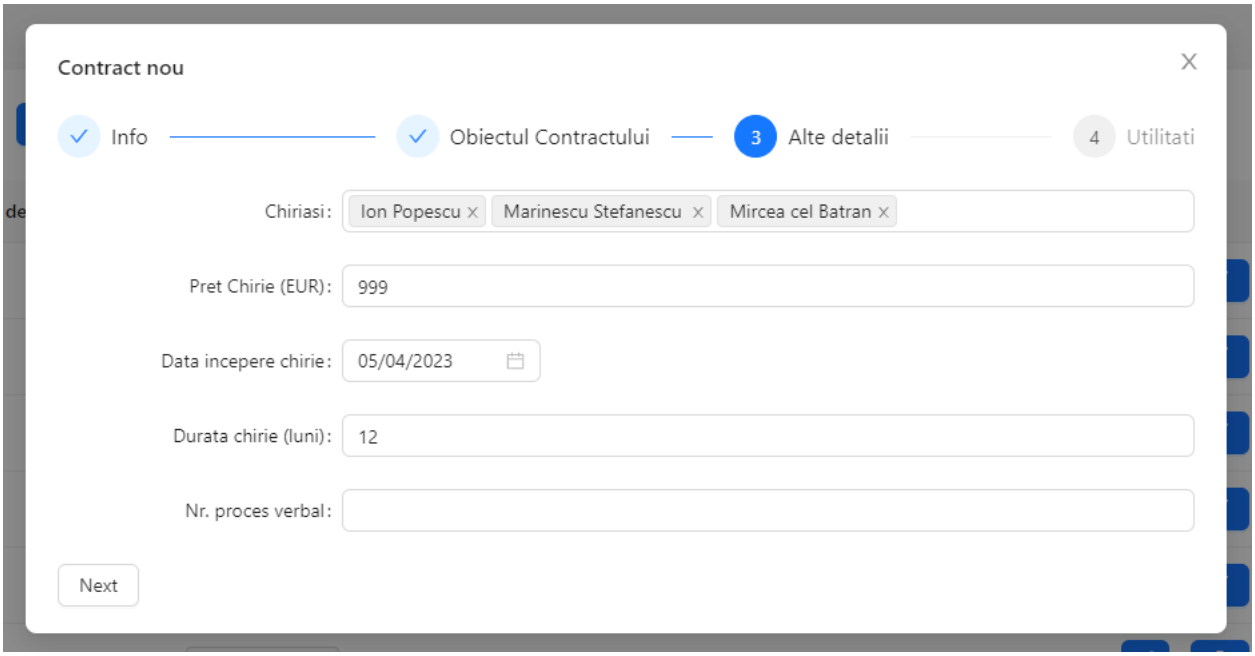

Figura 15. Asistentul pentru adăugarea unui contract nou
Sursa: Prelucrarea autorului

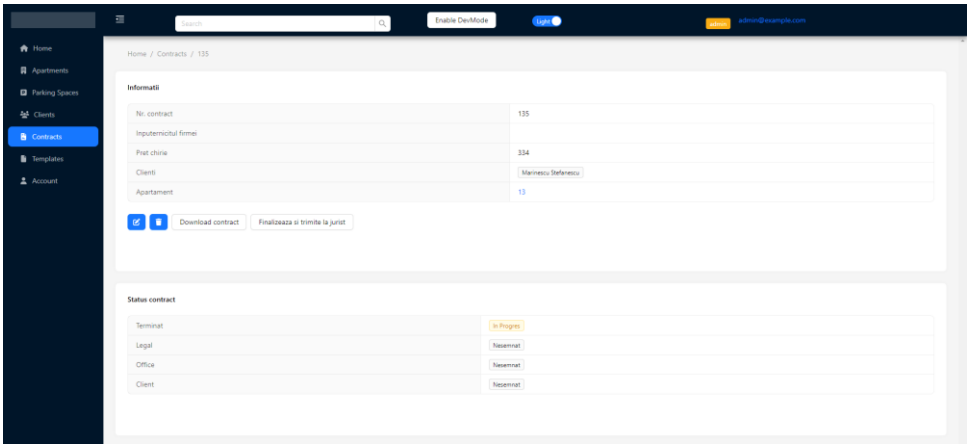

Figura 16. Pagina cu detaliile unui contract
Sursa: Prelucrarea autorului

Avantaje ale utilizării acestei aplicații sunt ușurința cu care se pot adăuga date. Orice entitate gestionată de aplicație poate fi creată, modificată sau ștearsa prin doar câteva apăsări de mouse. Informația din aplicație se actualizează rapid datorita tehnologiilor folosite. Comunicarea cu clienții este foarte importantă – astfel, pentru a păstra o buna legătura cu aceștia, este foarte ușoara adăugarea lor în baza de date. Generarea de contracte este un alt plus al acestei aplicații. Aceasta preia datele direct și le introduce într-un template. Designul flexibil al bazei de date face ca orice noua facilitate să poată fi dezvoltată si adăugată rapid.

În momentul de față nu este gata de folosit într-un mediu real, dar posibilități de dezvoltare viitoare includ următoarele:

- un sistem de tipul „My Apartment”, în care chiriașii își pot vizualiza cheltuielile, suprafața apartamentului și alte date similare, pot descărca o copie a contractului semnat, își pot obține si plăti facturile;
- realizarea de rapoarte lunare pe baza datelor din complex (rata ocupării apartamentelor, încasări etc.);

- construirea de aplicații mobile sau adaptarea paginilor web pentru a funcționa pe ecrane înguste;
- finalizarea sistemului de semnare a contractelor – un contract este trimis intre mai multe departamente, apoi la final către client care îl poate citi si semna;
- sistem de raportare al defecțiunilor – in cazul in care ceva se defectează (ex. Se arde un bec pe casa scării), clienții afectați pot lăsa un ticket iar echipa de mentenanță a complexului va rezolva;
- integrare cu sistemele contabile pentru contabilizarea facturilor și a chiriilor in cadrul firmei care deține complexul;
- separarea interfeței grafice pe secțiuni specifice domeniului utilizatorului care o accesează (de exemplu, contabilitatea nu poate să adauge apartamente, mentenanța poate doar sa vadă dacă există daune).

6. Bibliografie

1. Mozilla, "Web JavaScript Documentation," https://developer.mozilla.org/en-US/docs/Web/JavaScript.
2. TypeScript, "TypeScript Documentation," https://www.typescriptlang.org/docs/.
3. Skiuileuf, "ProcesareFeedback," https://github.com/Skiuileuf/ProcesareFeedback.
4. Wikipedia, "Model-view-controller," https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.
5. RegisFrey, Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10298177 (pentru Figura 1)
6. REST API Tutorial, "HTTP Status Codes," https://www.restapitutorial.com/httpstatuscodes.html.
7. MongoDB, "MongoDB Documentation," https://www.mongodb.com/docs/.
8. Mongoose, "Mongoose Documentation," https://mongoosejs.com/docs/.
9. Next.js, "Next.js Documentation," https://nextjs.org/docs.
10. React, "React Documentation," https://react.dev/reference/react.
11. Ant Design, "Ant Design Components Overview," https://ant.design/components/overview/.
12. OASIS Committee, "Customer Information Quality (CIQ) Specification," https://www.oasis-open.org/committees/ciq/download.html.