

CAPITOLUL 4

4.1 GENERAREA BAZEI DE DATE

Un pas important în implementarea aplicației web este posibilitatea stocării datelor într-un mod persistent și sigur. Astfel, am utilizat o bază de date *Oracle Express*, pe care am creat-o în cadrul unui container *Docker* cu ajutorul unei imagini specifice, asigurând o gestionare mai ușoară și o instalare mai rapidă. În *Figura 4.1* este prezentată comanda de creare și rulare a containerului specific bazei de date utilizată.

Odată creată baza de date de tip CDB, am creat în cadrul acesteia o nouă bază de date de tip PDB, pe care am folosit-o pentru a stoca datele utilizate de aplicația web. Scriptul de creare a bazei de date PDB este prezentat în *Figura 4.2*.

```
docker run -d --name orcllex
-p 1521:1521 -p 5500:5500
-e ORACLE_PWD=sys
-v C:\Users\Diana\opt\oracle\oradata:/opt/oracle/oradata
container-registry.oracle.com/database/express:21.3.0-xe
```

Figura 4.1 – Comanda de creare și rulare container *Docker*

```
CREATE PLUGGABLE DATABASE app_pdb
ADMIN USER pdb_admin
IDENTIFIED BY admin
FILE_NAME_CONVERT = ('/opt/oracle/oradata/XE/', '/opt/oracle/oradata/XE/app_pdb/');
```

Figura 4.2 – Comanda de creare a PDB *app_pdb*

În cadrul PDB-ului am creat mai departe tabelele necesare, specificate în diagrama entitate-asociere. Astfel o primă tabelă creată este tabela *Institutii_inrolate*, ce a fost creată conform scriptului din *Figura 4.3*. Astfel, sunt prezentate principalele caracteristici ale coloanelor tablei, dar și constrângerile setate.

```
CREATE TABLE institutii_inrolate (
  cui          VARCHAR2(10) PRIMARY KEY,
  denumire     VARCHAR2(128) NOT NULL,
  telefon      VARCHAR2(10) NOT NULL,
  email_public VARCHAR2(64) NOT NULL,
  adresa       VARCHAR2(64) NOT NULL,
  link_site_oficial VARCHAR2(64),
  localitate   VARCHAR2(32) NOT NULL,
  procent_bonificare NUMBER(3) NOT NULL,
  procent_penalizare NUMBER(3) NOT NULL,
  id_cerere    NUMBER NOT NULL,
  cod_judet    VARCHAR2(2) NOT NULL,
  FOREIGN KEY ( id_cerere )
    REFERENCES cereri_inrolare ( id ),
  FOREIGN KEY ( cod_judet )
    REFERENCES judete ( cod )
    ON DELETE CASCADE,
  CONSTRAINT institutii_publice_con_unique_email_public UNIQUE ( email_public )
    USING INDEX enable,
  CONSTRAINT institutii_publice_con_unique_email_admin UNIQUE ( email_admin )
    USING INDEX enable,
  CONSTRAINT institutii_publice_con_unique_iban UNIQUE ( cont_iban )
    USING INDEX enable
);
```

Figura 4.3 – Script creare tabelă *Institutii_inrolate*

O altă tabelă creată este tabela pentru *Elemente_platite*, a cărei script de creare este vizibil în *Figura 4.4*. Pentru a permite autoincrementarea valorii cheiei primare, am folosit secvențe, iar în cazul acestei tabele, înainte de inserarea unei înregistrări în tabelă se verifică ca doar una dintre coloanele *id_impozit_anuntat*, *id_impozit* și *id_taxa* să aibă o valoare diferită de *null*, conform *triggerului* prezentat în *Figura 4.5*.

```
CREATE TABLE elemente_platite (
    id                NUMBER PRIMARY KEY,
    id_plata          NUMBER NOT NULL,
    id_impozit_anuntat NUMBER,
    id_impozit         NUMBER,
    id_taxa            NUMBER,
    FOREIGN KEY ( id_plata )
        REFERENCES plati ( id ),
    FOREIGN KEY ( id_impozit_anuntat )
        REFERENCES impozite_anuntate ( id ),
    FOREIGN KEY ( id_impozit )
        REFERENCES impozite ( id ),
    FOREIGN KEY ( id_taxa )
        REFERENCES taxe ( id )
);
```

Figura 4.4 – Script creare tabelă Elemente_platite

```
create or replace TRIGGER bi_elemente_platite
BEFORE INSERT ON elemente_platite
FOR EACH ROW
DECLARE
    field_count NUMBER := 0;
BEGIN
    IF :NEW.id IS NULL THEN
        SELECT elemente_platite_id_seq.nextval INTO :NEW.id FROM dual;
    END IF;

    IF COALESCE(:NEW.id_impozit_anuntat, :NEW.id_impozit, :NEW.id_taxa) IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'One field among id_impozit_anuntat, id_impozit,
        and id_taxa should have a value and be not null.');
```

```
END IF;

    field_count := field_count + CASE WHEN :NEW.id_impozit_anuntat IS NOT NULL THEN 1 ELSE 0 END;
    field_count := field_count + CASE WHEN :NEW.id_impozit IS NOT NULL THEN 1 ELSE 0 END;
    field_count := field_count + CASE WHEN :NEW.id_taxa IS NOT NULL THEN 1 ELSE 0 END;

    IF field_count <> 1 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Only one field among id_impozit_anuntat, id_impozit,
        and id_taxa should have a value and be not null.');
```

```
END IF;

END;
```

Figura 4.5 – Script trigger inserare înregistrare în Elemente_platite

Pentru a stabili conexiunea cu baza de date, am utilizat ORM-ul sequelize și am creat o instanță precum este prezentat în *Figura 4.6*. Ulterior am construit modele ce replică tabelele din baza de date, folosind sequelize pentru a eficientiza lucrul cu structurile stabilite ale tabelor. Două dintre modelele create sunt pentru Utilizator și respectiv pentru Utilizator_cu_cont, prezentate în Anexa 4.B și Anexa 4.A.

```
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize({
  dialect: 'oracle',
  host: 'localhost',
  port: 1521,
  database: 'app_pdb',
  username: 'PDB_ADMIN',
  password: 'admin',
  dialectOptions: {
    charset: 'AL32UTF8',
    collate: 'AL32UTF8_BIN',
  }
});

sequelize.sync().then(() => {
  console.log("All models were synchronized successfully!");
})

module.exports = sequelize;
```

Figura 4.6 – Instanță *sequelize*

4.2 ÎNROLARE INSTITUȚIE

În cadrul aplicației, instituțiile publice colectoare de taxe se pot înrola pentru a putea colecta impozitele și taxele locale direct de la utilizatori (persoane fizice) prin intermediul plăților online. Astfel, pentru a putea fi vizibile în listele de selecție specifice formularelor de plată puse la dispoziția utilizatorilor, instituțiile trebuie să treacă printr-un proces de înrolare.

Primul pas din cadrul procesului de înrolare a unei instituții este accesarea paginii de înrolare. Din cadrul paginii de înregistrare prezentată în *Figura 4.7*, prin apăsarea butonului **Înrolează-te** se va deschide o fereastră modală, prin intermediul căreia este cerută adresa de mail, așa cum este exemplificat în *Figura 4.8*.

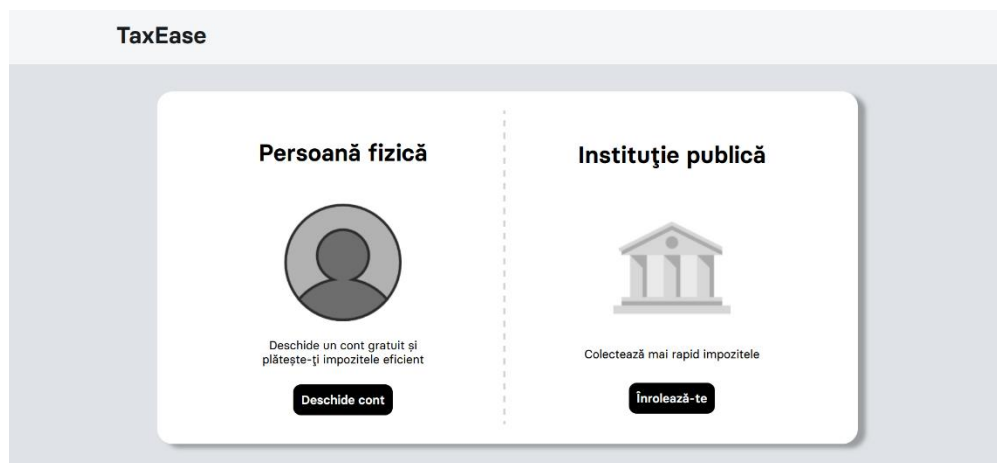


Figura 4.7 – Pagina de înregistrare



Figura 4.8 – Fereastra modală pentru introducerea adresei de email

În baza de date sunt stocate în cadrul tabelii *Institutii* adresele de mail oficiale ale persoanelor delegate (administratorilor) ale instituțiilor publice colectoare de taxe din România. Astfel, pentru a preveni înrolările neautorizate este verificată existența în baza de date a adresei furnizate în câmpul de *input* din cadrul ferestrei modale. În *Figura 4.9* este prezentată funcția pentru verificarea adresei de email furnizate, ce înglobează fluxul logicii generale. Corpul API-ului specific aducerii adreselor de mail autorizate din baza de date este prezentat mai detaliat în *Figura 4.10*.

```
const handleRequestLinkClick = async () => {
  if (mail === '') {
    setErrorMessage('Introduceți o adresă de mail.');
```

```
    setIsEmailValid(false);
    return;
  }

  const response = await fetch('http://localhost:8080/api/get-mailuri-admini');
  const data = await response.json();

  const emailAddresses = data.map((item) => item.email_admin);

  if (!emailAddresses.includes(mail)) {
    setIsEmailValid(false);
    setErrorMessage(
      'Adresa de email nu este recunoscută ca fiind autorizată. Accesul către pagina de înrolare este nepermis.'
    );
    setMail('');
    return;
  } else {
    setIsEmailValid(true);
    try {
      await fetch('http://localhost:8080/api/cerereLink', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ userEmail: mail }),
      });
      setIsRequestSent(true);
    } catch (err) {
      console.error(err);
    }
  }
};
```

Figura 4.9 – Funcția principală de verificare a adresei de mail furnizate pentru a accesa pagina de înrolare

```

router.route('/get-mailuri-admini').get(async (req, res) => {
  try {
    const mailuriAdmini = await Institutie.findAll({
      attributes: ['email_admin'],
    });
    res.status(200).json(mailuriAdmini);
  } catch (error) {
    console.log(error)
    res.status(500).json(error);
  }
})

```

Figura 4.10 – Metoda GET pentru adresele de email autorizate

În cazul în care adresa de *email* furnizată este una autorizată, va fi trimis un *email* ce are în conținutul său un *link* de access pentru pagina de înrolare. Corpul mailului este construit cu ajutorul bibliotecii *mailgen*, iar mailul este creat cu ajutorul bibliotecii *nodemailer*, așa cum este prezentat în *Figura 4.11* și *Figura 4.12*.

```

let transporter = nodemailer.createTransport(config);

let MailGenerator = new Mailgen({
  theme: 'salted',
  product: {
    name: "TaxEase",
    link: 'http://localhost:3000/'
  }
})

let response = {
  body: {
    title: 'Cerere înrolare',
    intro: 'Ne bucurăm pentru interesul acordat!',
    action: {
      instructions: 'Pentru a începe procesul de înrolare în aplicație vă rugăm accesați butonul de mai jos:',
      button: {
        color: '#02755D',
        text: 'Înrolează-te',
        link: 'http://localhost:3000/inrolare?userEmail=${userEmail}'
      }
    }
  }
}

let mail = MailGenerator.generate(response)

```

Figura 4.11 – Construire corp mail cu link către pagina de înrolare

```

let mail = MailGenerator.generate(response)

let message = {
  from: EMAIL,
  to: userEmail,
  subject: "Link pagina înrolare TaxEase",
  html: mail
}

transporter.sendMail(message).then(() => {
  return res.status(201).json({
    msg: "you should receive an email"
  })
}).catch(error => {
  console.error(error);
  return res.status(500).json({ error })
})

```

Figura 4.12 – Generare și trimitere mail cu link către pagina de înrolare

Odată accesată pagina de înrolare, aceasta este formată din două secțiuni principale, conform *Figurii 4.13*. Prima este secțiunea de descărcare, prin intermediul căreia, apăsând butonul **Descarcă** se va descărca formularul de tip cerere în format PDF. Funcția de descărcare a cererii este prezentată în *Figura 4.14*.

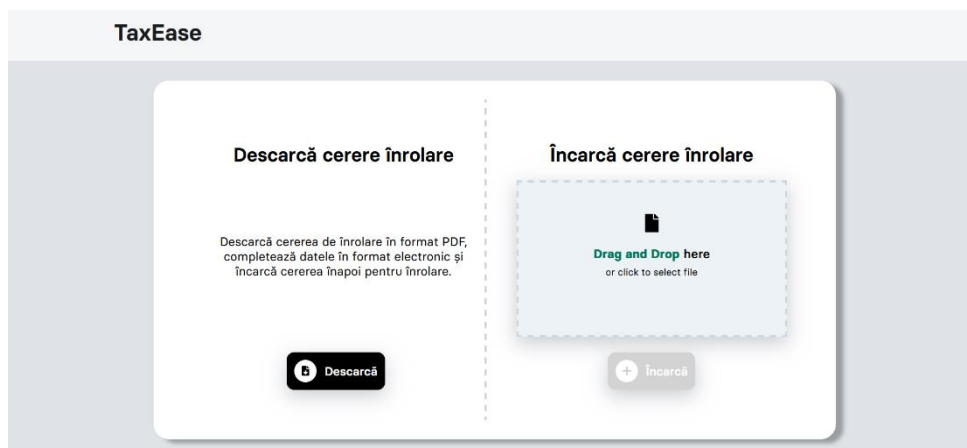


Figura 4.13 – Pagina de înrolare

```
const downloadCerere = () => {  
  const cerere = new URL('../samples/FormularCerereÎnrolare.pdf', import.meta.url).href  
  fetch(cerere)  
    .then(response => response.blob())  
    .then(blob => {  
      saveAs(blob, 'formularCerereNecompletat.pdf')  
    });  
}
```

Figura 4.14 – Funcția de descărcare a cererii de înrolare

A doua secțiune este secțiunea de încărcare, unde se va încărca formularul, odată ce a fost completat. Formatul acceptat este PDF, iar după încărcare, datele de interes sunt automat extrase și validate direct de către sistem. Astfel, dacă datele esențiale nu lipsesc și sunt considerate valide, instituția va fi adăugată în baza de date și ulterior va fi vizibilă în listele prezentate utilizatorilor în cadrul aplicației *web*. În *Figura 4.15* este prezentată funcția de extragere a textului din documentul PDF încărcat, în cadrul căreia a fost folosită biblioteca *pdf-parse*. Parsarea efectivă a textului este prezentat în *Anexa 4.C*.

Funcția principală de inserare a unei noi instituții în baza de date este prezentată în *Figura 4.16*. Astfel, odată cu înrolarea unei noi instituții, se ține cont de legăturile cu celelalte tabele. Vor fi automat inserate în baza de date atât cererea tip încărcată, cât și impozitele și taxele pe care instituția le poate colecta.

```

app.post("/extract-text",
  (req, res, next) => {
    upload.single('newFile')(req, res, function (err) {
      if (err) {
        console.error(err);
        return res.status(500).json({ error: 'Failed to upload newFile in file' });
      }
      next();
    });
  },
  (req, res) => {
    console.log('req.file', req.file)
    console.log('req.file.path', req.file.path)
    if (!req.file || !req.file.path) {
      res.status(400);
      res.end();
      return
    }
    pdfParser(req.file.path).then(result => {
      res.send(result.text);
    })
  })
)

```

Figura 4.15 – Funcția de extragere a textului din documentul PDF încărcat

```

const insertDataToInstitutiipublice = async (data, formData, taxData) => {
  try {
    data.forEach((value, key) => {
      formData.append(key, value);
    })
    const pdfResponse = await axios.post('http://localhost:8080/api/addCerere', formData)
    const institutieResponse = await axios.post('http://localhost:8080/api/inrolare-institutie', data);
    for (const element of taxData) {
      if (element.denumire.startsWith('I')) {
        const impozitResponse = await axios.post('http://localhost:8080/api/addImpozit', {
          cui: data.get('CUI:'),
          data: element
        });
      } else if (element.denumire.startsWith('T')) {
        const taxResponse = await axios.post('http://localhost:8080/api/addTaxa', {
          cui: data.get('CUI:'),
          data: element
        });
      }
    }
  } catch (error) {
    console.error(error);
  }
};

```

Figura 4.16 – Funcția principală de inserare a unei noi instituții în baza de date

Algoritmul din spatele extragerii datelor esențiale din cadrul formularului este prezentat în *Anexa 4.C*. De asemenea, API-urile specifice folosite pentru a înregistra în baza de date formularul de cerere, instituția, impozitele și taxele specific instituției sunt expuse mai în detaliu în *Figura 4.17*, *Figura 4.18*, *Figura 4.19* și respectiv *Figura 4.20*.

```

router.route("/addImpozit").post(async (req, res) => {
  try {
    const { cui, data } = req.body;

    const impozit = await Impozit.create({
      nume: removeDiacritics(data.denumire),
      cod_cont: data.cod_cont,
      cont_iban: data.cont_iban,
      cui_institutie: cui
    });

    res.status(200).json(impozit);
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'Failed to insert data into impozite table.' });
  }
});

```

Figura 4.17 – Metoda POST de inserare a unui impozit în baza de date

În cadrul metodei de inserare a unei cereri de înrolare în baza de date, am folosit biblioteca *multer* pentru stoca cererea și în formatul său inițial, într-un fișier prestabilit. De asemenea, pentru a transforma cererea din format PDF în format BLOB, specific câmpului *afferent* din tabela *Cereri_inrolare* am convertit documentul într-un format de tip *buffer*, folosind biblioteca *fs*.

```

1  const storage = multer.diskStorage({
2    destination: 'C:\\Users\\Diana\\opt\\oracle\\oradata\\cereri-inrolare',
3    filename: function (req, file, cb) {
4      cb(null, Date.now() + '-' + file.originalname);
5    }
6  });
7
8  const upload = multer({ storage: storage });
9
10
11  router.post('/addCerere',
12    (req, res, next) => {
13      upload.single('newFile')(req, res, function (err) {
14        if (err) {
15          console.error(err);
16          return res.status(500).json({ error: 'Failed to upload newFile in file' });
17        }
18        next();
19      });
20    },
21    async (req, res) => {
22      try {
23        const data = req.body;
24        console.log('data ', data);
25        const file = req.file;
26        console.log('file=', file);
27        const fileData = fs.readFileSync(file.path);
28
29        const cerereInrolare = await CerereInrolare.create({
30          cui_institutie: data['CUI:'],
31          data_incarcare: new Date(),
32          pdf: fileData
33        });
34
35        console.log('cerereInrolare=', cerereInrolare);
36        res.status(200).json(cerereInrolare);
37      } catch (error) {
38        console.error(error);
39        res.status(500).json({ success: false, message: 'Failed to insert data into cereri_inrolare table.' });
40      }
41    })

```

Figura 4.18 – Metoda POST de inserare a cererii de înrolare în baza de date

```

router.route('/addTaxa').post(async (req, res) => {
  try {
    const { cui, data } = req.body;

    const taxa = await Taxa.create({
      nume: removeDiacritics(data.denumire),
      cod_cont: data.cod_cont,
      cont_iban: data.cont_iban,
      suma: data.suma,
      cui_institutie: cui
    });

    res.status(200).json(taxa);
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'Failed to insert data into taxa table.' });
  }
})

```

Figura 4.19 – Metoda POST de inserare unei taxe în baza de date


```

router.post('/inrolare-institutie', upload.none(), async (req, res) => {
  try {
    const data = req.body;
    console.log('data-', data);

    const id_cerere = await axios.get('http://localhost:8080/api/get-iu-cerere/${data['CUI:']}');
    if (id_cerere) {
      console.log('Id cerere returnat cu succes')
    } else {
      res.status(404).json({ success: false, message: 'Id cerere not found.' });
    }

    const cod_judet = await axios.get('http://localhost:8080/api/get-cod-judet/${data['ul:']}');
    if (cod_judet) {
      console.log('Cod judet returnat cu succes')
    } else {
      res.status(404).json({ success: false, message: 'Cod Judet not found.' });
    }

    const institutieResponse = await InstitutieInrolata.create({
      cui: data['CUI:'],
      denumire: removeDiacritics(data['iei:']),
      telefon: data['Telefon de contact:'],
      email_public: data['Adresa de mail publică\\x83:'],
      adresa: removeDiacritics(data['Adresa:']),
      link_site_oficial: data['Link site:'],
      localitate: data['Localitatea:'],
      procent_bonificare: data['bonificare:'],
      procent_penalizare: data['penalizare:'],
      id_cerere: id_cerere.data.id_cerere,
      cod_judet: cod_judet.data.judetCod
    });

    console.log(institutieResponse);
    res.status(200).json(institutieResponse);
  } catch (error) {
    console.error(error);
    res.status(500).json({ success: false, message: 'Failed to insert data into Institutii_inrolate table.' });
  }
});

```

Figura 4.20 – Metoda POST de inserare a unei instituții în baza de date

Astfel, după o înrolare reușită, instituția va putea fi vizibilă în formularele principale destinate utilizatorilor, în cadrul paginii *web*. Unul dintre locurile în care sunt prezentate detalii despre instituțiile înrolate, este reprezentat de pagina instituțiilor. Prin selectarea județului de interes, vor fi afișate alăturat, precum în *Figura 4.21*, instituțiile ce aparțin respectivului județ, dintre cele înrolate.

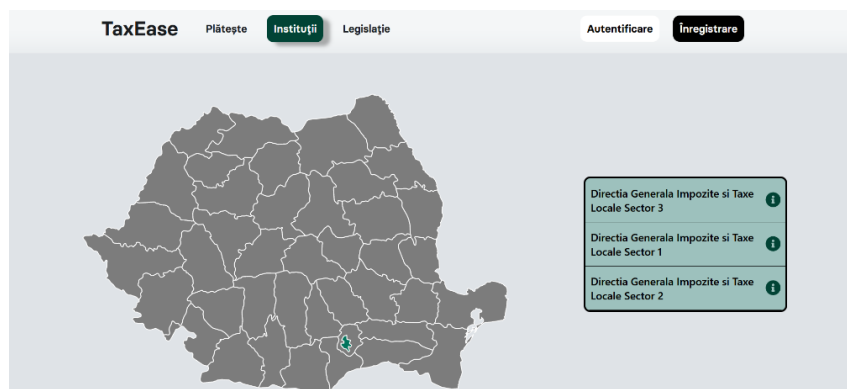


Figura 4.21 – Pagina instituțiilor

De asemenea, prin apăsarea butonului de tip *icon* aferent numelui instituției sau instituțiilor dorite, din lista afișată, este permisă vizualizarea mai multor detalii specifice, precum este reprezentat în *Figura 4.22*. Funcțiile specifice rerandării județului ales, dar și cele specifice afișării detaliilor despre instituția selectată, sunt prezentate în *Figura 4.23* și *Figura 4.24* respectiv.

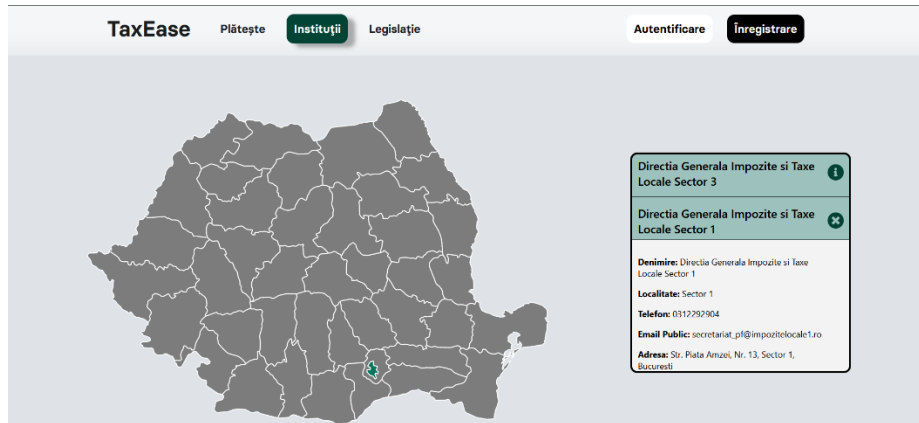


Figura 4.22 – Pagina instituțiilor – afișare detalii instituție selectată

```
useEffect(() => {
  const paths = document.querySelectorAll("path");
  paths.forEach((path) => {
    path.removeEventListener("click", handleClick);
    path.addEventListener("click", () => handleClick(path));
  });
}, [handleClick]);

useEffect(() => {
  const paths = document.querySelectorAll("path");
  paths.forEach((path) => {
    const id = path.getAttribute("id");
    if (id === selectedDistrict) {
      path.setAttribute("fill", "#02755D");
    } else {
      path.setAttribute("fill", "");
    }
  });
}, [selectedDistrict]);
```

Figura 4.23 – Funcții rerandare județ selectat

```
const [selectedDistrict, setSelectedDistrict] = useState();
const [institutionsInDistrict, setInstitutionsInDistrict] = useState([]);
const [showMoreInfo, setShowMoreInfo] = useState(false);

const handleClick = useCallback((element) => {
  console.log("Clicked on district:", element.getAttribute('name'));
  const districtId = element.getAttribute("id");
  setSelectedDistrict(districtId);
  fetchInstitutionsFromDatabase(districtId);
}, []);

const fetchInstitutionsFromDatabase = async (districtId) => {
  try {
    const response = await fetch(`http://localhost:8080/api/getInstitutiile/${districtId}`);
    if (response.ok) {
      const institutions = await response.json();
      setInstitutionsInDistrict(institutions);
    } else {
      console.log("Error fetching institutions:", response.status);
    }
  } catch (error) {
    console.log("Error fetching institutions:", error);
  }
};

const handleInfoClick = useCallback(
  (institutionName) => {
    setShowMoreInfo((prevState) => ({
      ...prevState,
      [institutionName]: !prevState[institutionName],
    }));
  },
  []
);
```

Figura 4.24 – Funcții specifice afișării informațiilor despre instituții

4.3 FUNCȚIONALITĂȚI PENTRU UTILIZATORII CU CONT

În cadrul aplicației, utilizatorii ce optează pentru crearea unui cont vor beneficia de câteva funcționalități specifice, precum: vizualizarea unui istoric al plăților efectuate, programarea unor plăți viitoare și vizualizarea obligațiilor de plată avute.

Pentru a crea un cont, utilizatorii trebuie să acceseze pagina aferentă deschiderii unui cont, unde vor completa datele personale prin intermediul unui formular împărțit în trei părți: date personale, date cont și date card. Deoarece părțile formularului sunt accesibile consecutiv, pentru a accesa următoarea parte a formularului trebuie completate toate câmpurile solicitate. Altfel, vor fi afișate atenționări, precum este prezentat în *Figura 4.25*.

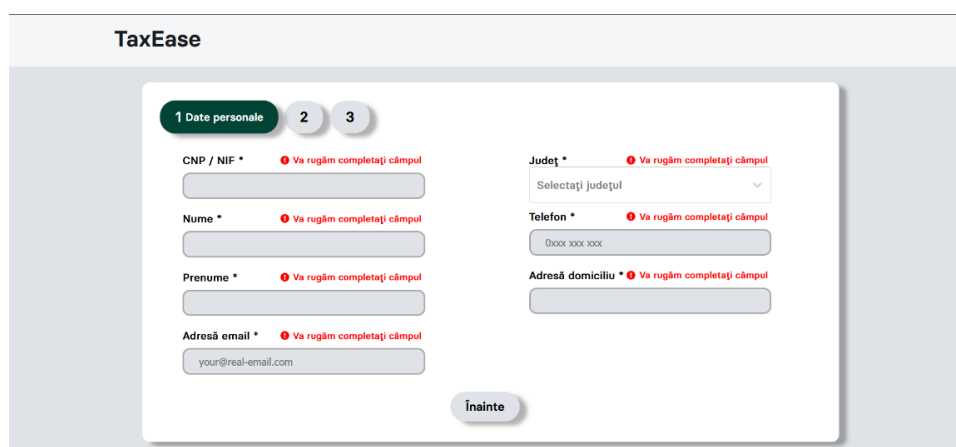
The image shows a web form titled "TaxEase" for creating an account. It has three steps: "1 Date personale", "2", and "3". The first step is active. The form contains several input fields: "CNP / NIF", "Nume", "Prenume", "Adresă email", "Județ", "Telefon", and "Adresă domiciliu". Each field has a red error message above it: "Va rugăm completați câmpul". The "Adresă email" field has the placeholder "your@real-email.com". At the bottom right, there is a button labeled "Înainte".

Figura 4.25 – Pagină creare cont -activare atenționări câmpuri necompletate

```
const handleSubmit = (e) => {
  e.preventDefault();
  const errors = {};

  inputFields.forEach((field) => {
    if (formData[field.id].trim() === '') {
      errors[field.id] = 'Va rugăm completați câmpul';
    } else if (field.id === 'telefon') {
      if (!/^[0-9\s]+$/.test(formData[field.id])) {
        errors[field.id] = 'Vă rugăm introduceți doar cifre';
      } else if (!/^0\d{3} \d{3} \d{3}$/.test(formData[field.id])) {
        errors[field.id] = 'Respectați formatul (0xxx xxx xxx)';
      }
    } else if (field.id === 'tin' && !/^\d+$/.test(formData[field.id])) {
      errors[field.id] = 'Vă rugăm introduceți doar cifre';
    } else if (field.id === 'tin' && formData[field.id].length !== 13) {
      errors[field.id] = 'Vă rugăm introduceți 13 cifre.';
    }
  });

  setFormErrors(errors);
  console.log(Object.keys(errors).length, errors);
  if (Object.keys(errors).length === 0) {
    navigate('/creare-cont/date-cont');
  }
};
```

Figura 4.26 – Funcție *submit* verificare câmpuri formular date personale

De asemenea, în momentul apăsării pe butonul Înainte, sunt verificate și validate datele introduse în câmpurile formularului, cu ajutorul funcției din *Figura 4.26*.

La finalul ultimei părți din cadrul formularului, partea de introducere a datelor specifice cardului bancar, dacă nu lipsesc date și toate datele introduse sunt valide, se va genera un cod *random*, care va fi ulterior trimis prin mail utilizatorului, pentru validarea adresei de mail. Astfel, o pagină modală va fi afișată pentru introducerea codului de verificare, conform *Figurii 4.27*, iar în cazul în care codul introdus se potrivește cu cel generat și trimis, utilizatorul va fi inserat în baza de date. Funcția specifică generării codului de confirmare este prezentată în *Figura 4.28*.

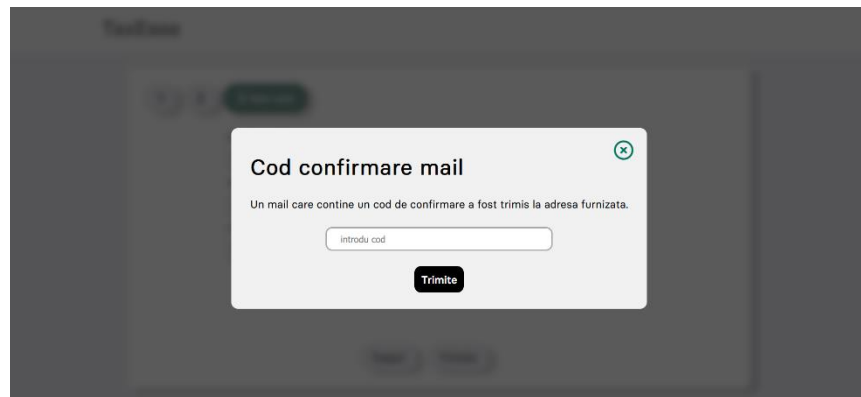


Figura 4.27 – Pagină modală – introducere cod validare email

```
function generearecodConfirmare() {
  const min = 1000;
  const max = 9999;

  const randomCode = Math.floor(Math.random() * (max - min + 1)) + min;
  return randomCode.toString();
}

const handleSubmit = async (e) => {
  e.preventDefault();

  let newErrors = {};

  for (const key in formData) {
    if (formData[key].trim() === '') {
      newErrors[key] = 'Va rugăm completați câmpul';
    } else if (key === 'numarCard') {
      if (!/^[0-9\s]+$/.test(formData[key])) {
        newErrors[key] = 'Vă rugăm introduceți doar cifre';
      } else if (!/^\d{4} \d{4} \d{4} \d{4}$/.test(formData[key])) {
        newErrors[key] = 'Vă rugăm respectați formatul (xxxx xxxx xxxx xxxx)';
      }
    } else if (key === 'cvv' && !/^\d+$/.test(formData[key])) {
      newErrors[key] = 'Vă rugăm introduceți doar cifre';
    } else if (key === 'cvv' && formData[key].length !== 3) {
      newErrors[key] = 'Vă rugăm introduceți 3 cifre.';
    }
  }

  setErrors(newErrors);

  if (Object.keys(newErrors).length === 0) {
    const randomCode = generearecodConfirmare();
    setGeneratedCode(randomCode);
    toggleModal();
  }
};
```

Figura 4.28 – Funcțiile de *submit* verificare câmpuri formular date card și de generare cod confirmare adresă de email

La apăsarea butonului trimite din cadrul paginii modale, codul introdus este verificat, iar dacă acesta se potrivește cu cel generat, va fi apelat API-ul prezentat în *Figura 4.29*, de creare a unui nou utilizator cu cont. Pentru a insera în tabela *Utilizatori* data de naștere și genul persoanei, acestea sunt mai întâi extrase din codul de identificare furnizat (CNP/NIF), conform *Figurii 4.30* și *Figurii 4.31* respectiv.

```
const handleRequestLinkClick = async () => {
  if (cod === '') {
    setErrorMessage('Introduceți codul de confirmare.');
```

```
    setIsCodValid(false);
    return;
  } else if (cod === generatedCode) {
    setIsCodValid(true);
    console.log(generatedCode, 'vs', cod);
    try {
      const response = await fetch('http://localhost:8080/api/create-account', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData),
      });
      setIsRequestSent(true);
    } catch (err) {
      console.error(err);
    }
  } else {
    console.log('generated:', generatedCode, 'vs', 'cod', cod);
    setErrorMessage('Cod introdus invalid.');
```

```
    setIsCodValid(false);
    return;
  }
};
```

Figura 4.29 – Funcție verificare cod validare email

```
function extractDataNastere(tin) {
  const firstDigit = parseInt(tin.charAt(0));
  const twoLastDigitsOfYear = tin.substr(1, 2);
  const month = tin.substr(3, 2);
  const dayOfMonth = tin.substr(5, 2);
  let yearPrefix;
  let dataNastere = null;

  if (firstDigit === 1 || firstDigit === 2) {
    yearPrefix = '19';
  } else if (firstDigit === 3 || firstDigit === 4) {
    yearPrefix = '18';
  } else if (firstDigit === 5 || firstDigit === 6) {
    yearPrefix = '20';
  }

  if (yearPrefix) {
    const year = parseInt(yearPrefix + twoLastDigitsOfYear);
    dataNastere = new Date(year, month - 1, dayOfMonth);
  }

  return dataNastere;
}
```

Figura 4.30 – Funcție extragere data naștere

```
function extractGen(tin) {
  const firstDigit = parseInt(tin.charAt(0));

  if (firstDigit === 1 || firstDigit === 3 || firstDigit === 5 || firstDigit === 7) {
    return 'M';
  } else if (firstDigit === 2 || firstDigit === 4 || firstDigit === 6 || firstDigit === 8) {
    return 'F';
  }
  return null;
}
```

Figura 4.31 – Funcție extragere gen

În cadrul înregistrării în baza de date a unui nou utilizator cu cont, parola furnizată în cadrul formularului de înregistrare este codificată pentru a menține un nivel de securitate al datelor, folosind *bcrypt*, pentru a transforma parola din *plain text* într-un format *hash*, greu de decifrat. Astfel, în *Figura 4.32* este prezentat API-ul de inserare a unui nou utilizator cu cont.

```
router.post('/add-utilizatorCuCont', async (req, res) => {
  try {
    const { id_utilizator, username, password } = req.body;
    const hashedPassword = await bcrypt.hash(password, saltRounds);

    const utilizatorCuCont = await UtilizatorCuCont.create({
      id_utilizator,
      username,
      password: hashedPassword
    });

    res.status(200).json({ message: 'UtilizatorCuCont created successfully.\n' + utilizatorCuCont });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Failed to create UtilizatorCuCont.' });
  }
});
```

Figura 4.32 – Metoda POST de inserare a unui utilizator cu cont

Astfel, în cadrul logării se verifică existența utilizatorului în baza de date după username, iar mai apoi după parolă, aplicându-se metoda inversă a criptării, prezentată în *Figura 4.33*.

```
router.route('/get-utilizatorCuCont').get(async (req, res) => {
  try {
    const { username, password } = req.query;

    const utilizatorCuCont = await UtilizatorCuCont.findOne({
      where: {
        username: username
      }
    });

    if (utilizatorCuCont) {
      const isPasswordMatch = bcrypt.compare(password, utilizatorCuCont.password);

      if (isPasswordMatch) {
        res.status(200).json(utilizatorCuCont);
      } else {
        res.status(401).json({ error: 'Invalid username or password' });
      }
    } else {
      res.status(401).json({ error: 'Invalid username or password' });
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: 'Failed to retrieve UtilizatorCuCont' });
  }
});
```

Figura 4.33 – Metoda GET verificare existență utilizator cu cont după parolă

În cazul în care atât *username*-ul, cât și parola sunt corecte, atunci utilizatorul poate intra în contul său din cadrul aplicației. O sesiune va fi creată, folosind biblioteca *express-session*, conform *Figurii 4.34* și va exista atâta timp cât butonul de *Logout* nu va fi accesat. De asemenea, sesiunea va expira singură după o perioadă prestabilită de o oră.

```
app.use(session({
  key: "userId",
  secret: "mySecret",
  resave: false,
  saveUninitialized: false,
  cookie: {
    maxAge: 60 * 60 * 1000,
  },
}));
```

Figura 4.34 – Creare sesiune

În cadrul sesiunii va fi păstrat ID-ul utilizatorului logat, putând fi folosit în cadrul tuturor paginilor destinate utilizatorilor cu cont, pentru aducerea datelor necesare din baza de date. Funcțiile specifice acțiunii de *login* sunt prezentate în cadrul *Figurii 4.35*.

```
app.get("/login", (req, res) => {
  if (req.session.user) {
    res.send({ loggedIn: true, user: req.session.user });
  } else {
    res.send({ loggedIn: false });
  }
});

app.post("/login", (req, res) => {
  const userId = req.body;
  req.session.user = userId;
  req.session.save((error) => {
    if (error) {
      console.error('Error saving session:', error);
    }
    res.send("Login successful");
  });
});
```

Figura 4.35 – Metoda GET și POST specific acțiunii de login

Odată logat, utilizatorul este redirectionat către pagina sa personală, în cadrul căreia poate vizualiza obligațiile fiscale pe care le are de plătit către instituțiile de care aparține. Astfel, pentru fiecare instituție vor fi afișate detalii despre impozitele locale datorate, dar și suma totală a acestora, folosind funcția prezentată în *Figura 4.36*, oferind utilizatorului posibilitatea de a le achita direct din această pagină, apăsând pe butonul **Plătește**.

```
const getTotalSuma = (cui) => {
  console.log('impoziteAnuntateData', impoziteAnuntateData);
  const subtotal = impoziteAnuntateData
    .filter((impozit) => impozit.cui_institutie === cui)
    .reduce((total, impozit) => total + parseFloat(impozit.suma), 0);
  return subtotal;
};
```

Figura 4.36 – Metodă calculare valoare totală impozite datorate pentru o instituție

În cazul în care utilizatorul dorește să realizeze plata impozitelor datorate pentru o instituție, prin apăsarea butonului Plătește va fi redirecționat către portalul de plată, conform funcției din *Figura 4.37*. Pagina portalului de plată a fost implementată cu ajutorul *stripe*, iar pentru elementele de plătit afișate în cadrul portalului de plată, sunt aplicate după caz atât penalități de întârziere cu plata impozitelor, cât și bonificații, precum în *Figura 4.39*, aferente plății integrale până la data de 31 martie a aceluiași an în care impozitele au fost anunțate. Verificarea acestor condiții este prezentată în *Figura 4.38*.

```
const navigateToPayPortal = (impozitIds, cui) => {
  const items = impozitIds.map(id => ({ id, quantity: 1 }));
  const metadata = {
    cod_identificare: codIdentificare,
    cui_institutie: cui,
    id_card_bancar: idCard,
    items: JSON.stringify(items),
  }
  console.log('metadata:', metadata)

  axios.post("http://localhost:8080/create-checkout-session", {
    items: items,
    metadata: metadata,
  }, {
    headers: {
      "Content-Type": "application/json",
    },
  })
    .then((res) => {
      if (res.status === 200) return res.data;
      return Promise.reject(res.data);
    })
    .then(({ url, session_id }) => {
      localStorage.setItem("session_id", session_id);
      window.location = url;
    })
    .catch((error) => {
      console.error(error);
    });
};
```

Figura 4.37 – Metodă navigare către portalul de plată

```
if (impozitAnuntat.suma <= 50) {
  if (paymentDate < new Date(paymentDate.getFullYear(), 2, 31) && paymentDate.getFullYear() === new Date(impozitAnuntat.data_emitere).getFullYear()) {
    bonificatie = { suma: (procBonificare / 100) * impozitAnuntat.suma, nume: 'Bonificatie: ' + nume }
  } else {
    const nr_luni_deposite = getMonthDifference(new Date(paymentDate.getFullYear(), 2, 31), new Date(impozitAnuntat.data_emitere))
    penalitati = { suma: (procPenalizare / 100) * nr_luni_deposite * impozitAnuntat.suma, nume: 'Penalitati: ' + nume }
  }
} else {
  if (paymentDate < new Date(paymentDate.getFullYear(), 2, 31)) {
    if (paymentDate.getFullYear() === new Date(impozitAnuntat.data_emitere).getFullYear()) {
      bonificatie = { suma: (procBonificare / 100) * impozitAnuntat.suma, nume: 'Bonificatie: ' + nume }
    } else {
      const nr_luni_deposite = getMonthDifference(new Date(impozitAnuntat.data_emitere), new Date(paymentDate.getFullYear(), 8, 30))
      penalitati = { suma: (procPenalizare / 100) * nr_luni_deposite * impozitAnuntat.suma, nume: 'Penalitati: ' + nume }
    }
  } else {
    if (paymentDate > new Date(paymentDate.getFullYear(), 8, 30)) {
      const nr_luni_deposite = getMonthDifference(new Date(paymentDate.getFullYear(), 8, 30), new Date(impozitAnuntat.data_emitere))
      penalitati = { suma: (procPenalizare / 100) * nr_luni_deposite * impozitAnuntat.suma, nume: 'Penalitati: ' + nume }
    }
  }
}
```

Figura 4.38 – Calcularea bonificației și a penalităților

Impozitele afișate în lista elementelor de plată din cadrul portalului de plată au fost formate conform celor prezentate în *Figura 4.40*. De asemenea, crearea unei sesiuni noi de tip *checkout* specifică *stripe*, este prezentată în *Figura 4.41*. Astfel, după ce plata a fost făcută cu succes, utilizatorul este automat redirecționat înapoi către pagina personală.

Figura 4.39 – Exemplu portal plată

```
const applyDiscount = paymentDate.getFullYear() === new Date(impozitAnuntat.data_emitere).getFullYear() &&
  paymentDate < new Date(paymentDate.getFullYear(), 2, 31) ? 1 : 0

return {
  price_data: {
    currency: "ron",
    product_data: {
      name: applyDiscount === 1 ? nume + ` * Bonificație de ${procBonificare}% aplicată` : nume
    },
    unit_amount: applyDiscount === 1 ? unitAmountInBani - ((procBonificare / 100) * impozitAnuntat.suma) * 100 : unitAmountInBani
  },
  quantity: item.quantity,
};
```

Figura 4.40 – Formatare elemente plată

```
const session = await stripe.checkout.sessions.create({
  payment_method_types: ["card"],
  mode: "payment",
  line_items: allItems,
  success_url: `http://localhost:3000/dashboard`,
  cancel_url: `http://localhost:3000/dashboard`,
  metadata: metadata,
  customer_email: email,
});
```

Figura 4.41 – Creare sesiune de checkout

Tot în cadrul paginii personale, utilizatorul cu cont poate vizualiza atât sub formă de grafic de tip linie, cât și sub formă de grafic de tip doughnut, plățile efectuate. Pentru realizarea graficelor am utilizat biblioteca *chart.js*. Astfel, am folosit componentele *Line* și respectiv *Doughnut* din cadrul acesteia. Funcțiile specifice creării celor două grafice sunt reprezentate în *Figura 4.42* și *Figura 4.43*.

```

const generateChartData = () => {
  if (!formValues.institutie || !formValues.impozit || !impoziteAnuntateData.length) {
    setChartData(null);
    return;
  }

  const impozitId = impoziteData.find((impozit) => impozit.nume === formValues.impozit)?.id;
  const institutieCui = institutiiData.find((institutie) => institutie.denumire === formValues.institutie)?.cui;

  const filteredData = impoziteAnuntateData.filter((elem) => {
    return elem.id_impozit === impozitId && elem.cui_institutie === institutieCui;
  });
  filteredData.sort((a, b) => {
    const dateA = new Date(a.data_emitere);
    const dateB = new Date(b.data_emitere);
    return dateA - dateB;
  });

  const chartLabels = filteredData.map((elem) => new Date(elem.data_emitere).getFullYear());
  const chartValues = filteredData.map((elem) => elem.suma);

  const chartData = {
    labels: chartLabels,
    datasets: [
      {
        label: `${formValues.impozit}`,
        data: chartValues,
        backgroundColor: '#014335',
        borderColor: 'black',
        borderWidth: 1,
      }
    ]
  };

  setChartData(chartData);
};

```

Figura 4.42 – Metoda generare date *line chart*

```

const data = elemPlatite.reduce(
  (acc, elem) => {
    const impozitAnuntat = impoziteAnuntateData.find(
      (ia) => ia.id === elem.id_impozit_anuntat
    );
    const impozit = impoziteData.find((i) => i.id === impozitAnuntat.id_impozit);

    acc.labels.push(impozit.nume);
    acc.values.push(impozitAnuntat.suma);

    return acc;
  },
  { labels: [], values: [] }
);

const chartData = {
  labels: data.labels,
  datasets: [
    {
      data: data.values,
      backgroundColor: [
        '#FF6384', '#36A2EB', '#FFCE56', '#8bc34a', '#9c27b0',
        '#ff5722', '#009688', '#607d8b', '#795548', '#ff9800',
      ],
      hoverBackgroundColor: [
        '#FF6384', '#36A2EB', '#FFCE56', '#8bc34a', '#9c27b0',
        '#ff5722', '#009688', '#607d8b', '#795548', '#ff9800',
      ],
    }
  ],
};

setChartData(chartData);

```

Figura 4.43 – Metoda generare date *doughnut chart*

O altă funcționalitate specifică utilizatorilor cu cont este încărcarea unei poze de profil și schimbarea pozei *default* cu cea dorită. În baza de date, în cadrul tabelului Utilizatori_cu_cont este stocată imaginea de profil sub formă de BLOB, așa cum este prezentat în *Figura 4.44*. Astfel, pentru a putea seta o imagine de profil *default*, am creat un director cu calea către fișierul comun containerului *docker* și sistemului local, în care am stocat poza efectivă. Comenzile specifice sunt prezentate în *Figura 4.45*.

```
CREATE TABLE utilizatori_cu_cont (
  id_utilizator NUMBER PRIMARY KEY,
  username      VARCHAR2(32) NOT NULL,
  password      VARCHAR2(64) NOT NULL,
  poza_profil   BLOB,
  FOREIGN KEY ( id_utilizator )
    REFERENCES utilizatori ( id )
);
```

Figura 4.44 – Script creare table Utilizatori_cu_cont

```
CREATE OR REPLACE DIRECTORY CERERI_DIR AS '/opt/oracle/oradata/cereri-inrolare';
ALTER TABLE utilizatori_cu_cont
MODIFY poza_profil DEFAULT BFILENAME('IMAGE_DIR', 'user.png') ;
commit;
```

Figura 4.45 – Script creare director și adăugare valoare *default* pentru câmpul poza_profil

Astfel, prin apăsarea butonului **Modifică** specific câmpului de poză profil, precum este exemplificat și în Figura 4.46, acesta se va transforma într-un buton de tip *submit*, conform funcției reprezentate în Figura 4.47 și, de asemenea va apărea un *form* de încărcare a pozei dorite. Astfel, în momentul salvării noii poze de profil va fi acționată funcția din Figura 4.48.

The screenshot shows a user profile interface with the following fields and buttons:

- CNP / NIF : 1760809403022
- Nume : Popescu
- Prenume : Gigel
- Email : yalih93175@ratedane.com (with a 'Modifică' button)
- Telefon : 0731292855 (with a 'Modifică' button)
- Adresă : str. Zen nr. 13 (with a 'Modifică' button)
- Poză profil : No file selected. (with a 'Save' button)

Figura 4.46 – Exemplu acționare buton modificare poză profil

```
const renderActionButton = (fieldName) => {
  if (fieldName === editingField) {
    if (fieldName === 'poza_profil') {
      return (
        <>
        <input type="file" accept="image/*" onChange={handleImageChange} />
        <button className="active" onClick={handleSubmit}>Save</button>
        </>
      );
    } else {
      return <button className="active" onClick={handleSubmit}>Save</button>;
    }
  } else {
    return <button onClick={() => handleEditClick(fieldName)}>Modifică</button>;
  }
};
```

Figura 4.47– Funcție randare buton modificare/salvare

```

const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    if (selectedImage && editingField === 'poza_profil') {
      const formData = new FormData();
      formData.append('poza_profil', selectedImage);

      const response = await axios.put(`http://localhost:8080/api/update-pozaprofil/${userData.id}`, formData);
      setPozaProfil(response.data.pozaprofil);
      setEditingField(null);
    } else {
      const updatedUtilizator = await fetch(`http://localhost:8080/api/update-utilizator/${userData.id}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({ field: editingField, value: editedUserData[editingField] })
      });

      setEditingField(null);
    }
  } catch (error) {
    console.error(error);
  }
};

```

Figura 4.48 – Funcție salvare poza profil încărcată

Metoda specifică schimbării pozei de profil este detaliată în *Figura 4.49*. Am folosit de asemenea, biblioteca *fs* pentru a transforma poza într-un format compatibil BLOB și biblioteca *multer* pentru a stoca într-un director indicat poza încărcată de către utilizator.

```

router.put('/update-pozaprofil/:id_utilizator', checkId,
  (req, res, next) => {
    upload.single('poza_profil')(req, res, function (err) {
      if (err) {
        console.error(err);
        return res.status(500).json({ error: 'Failed to upload poza_profil in file' });
      }
      next();
    });
  },
  async (req, res) => {
    try {
      const { id_utilizator } = req.params;
      const { path } = req.file;
      const fileData = fs.readFileSync(path);
      const updatedUtilizatorCuCont = await UtilizatorCuCont.update(
        { poza_profil: fileData },
        { where: { id_utilizator } }
      );

      if (updatedUtilizatorCuCont[0]) {
        const utilizatorUpdatat = await UtilizatorCuCont.findById(id_utilizator);
        res.status(200).json(utilizatorUpdatat);
      } else {
        res.status(404).json({ error: 'UtilizatorCuCont with id ${id} not found!' });
      }
    } catch (error) {
      console.error(error);
      res.status(500).json({ error: 'Failed to update poza_profil' });
    }
  }
);

```

Figura 4.49 – Metoda PUT pentru schimbarea pozei de profil în baza de date

Nu în ultimul rând, pentru a realiza operația de *logout*, utilizatorul va trebui să apese pe butonul de **Logout** din bara paginii personale. În momentul accesării butonului, acesta va ieși din cont, va fi redirecționat către pagina principală, iar sesiunea sa este distrusă, așa cum reiese din funcția prezentată în *Anexa 4.D*.

4.4 Funcționalități specifice utilizatorilor fără cont

În cadrul aplicației mai există și utilizatori care nu optează pentru crearea unui cont și care implicit nu pot beneficia de funcționalitățile specifice paginii personale. Cu toate acestea, funcționalitatea principală oferită utilizatorilor fără cont este posibilitatea efectuării plăților obligațiilor fiscale, cu condiția ca cuantumul acestora să fie știut în prealabil. Astfel, pentru a efectua o plată utilizatorii trebuie să completeze formularul specific, prezentat în *Figura 4.50*.

Figura 4.50 – Pagină efectuare plată pentru utilizatorii fără cont

În cadrul formularului am folosit trei componente de tip *Select* pentru a putea selecta județul, instituția și respectiv taxa sau impozitul dorite. În *Figura 4.51* este reprezentată componenta *Select* pentru instituții. Astfel pentru a selecta instituția de interes, sunt încărcate din baza de date drept opțiuni instituțiile înrolate specifice județului ales, conform funcției prezentate în *Figura 4.52*.

```
<Select
  id="institutie"
  placeholder="Selectați instituția"
  value={formValues['institutie']} ? { value: formValues['institutie'], label: formValues['institutie'] } : null
  onChange={(selectedOption) =>
    |   handleSelectChange(selectedOption, 'institutie')
    | }
  options={dropdownOptionsInstitutie}
  filterOption={({ label }, inputValue) =>
    |   label.toLowerCase().includes(inputValue.toLowerCase())
    | }
  isSearchable
  required
/>
```

Figura 4.51 – Componenta *Select* pentru instituții înrolate

```

const fetchDropdownOptionsInstitutie = async (selectedJudet) => {
  try {
    const response = await fetch(
      `http://localhost:8080/api/getInstitutii/${selectedJudet.cod}`
    );
    const data = await response.json();
    const dropdownOptionsFormatted = data.map((option) => ({
      value: option.denumire,
      label: option.denumire,
      cui: option.cui,
    }));
    setDropdownOptionsInstitutie(dropdownOptionsFormatted);
    setFormValues((prevValues) => ({
      ...prevValues,
      institutie: '',
      taxa: '',
    }));
  } catch (error) {
    console.error('Error fetching institutii options:', error);
  }
};

```

Figura 4.52 – Metodă aducere din baza de date a instituțiilor specific județului selectat

Astfel, pentru a naviga spre portalul de plată, utilizatorul trebuie să completeze toate câmpurile cerute în cadrul formularului și să acceseze butonul **Plătește**. Mai departe, în *Anexa 4.E* este prezentată funcția specifică butonului, ce conduce spre portalul de plată.

În cazul în care plata a fost realizată cu succes, utilizatorul este redirecționat înapoi către pagina formularului de plată, iar funcția prezentată în *Figura 4.53* este executată. Metoda de tip POST din cadrul funcției este detaliată în *Anexa 4.F*, fiind metoda prin cadrul căreia sunt introduse detaliile noii plăți în baza de date.

```

useEffect(() => {
  const session_id = localStorage.getItem("session_id");
  console.log('session_ID:', session_id)

  if (session_id) {
    axios.post("http://localhost:8080/payment-success-plata-neautenticata", {
      session_id: session_id,
    }, {
      headers: {
        "Content-Type": "application/json",
      },
    })
    .then((res) => {
      if (res.status === 200) {
        console.log("Payment successful!");

        if (res.data.clearLocalStorage) {
          localStorage.clear();
          console.log("localStorage cleared successfully");
        }
        else {
          console.error("Payment failed!");
        }
      }
    })
    .catch((error) => {
      console.error(error);
    });
  }
  localStorage.clear()
}, []);

```

Figura 4.53 – Funcție rerandare pagină plată

CONCLUZII ȘI CERCETĂRI VIITOARE

Structura lucrării, împărțită în patru capitole, prezintă un parcurs coerent și complet al procesului de dezvoltare al aplicației *web*, abordând analiza domeniului, proiectarea și implementarea sistemului informatic. Fiecare capitol are rolul său bine definit în documentarea și ilustrarea procesului de creare a aplicației.

În primul capitol, am realizat o analiză aprofundată a domeniului de interes, oferind o perspectivă amplă asupra subiectului și prezentând contextul general în care acesta se încadrează. Am evidențiat, de asemenea, problemele și provocările în cadrul acestui domeniu, demonstrând astfel relevanța și necesitatea dezvoltării aplicației *web*.

În cel de-al doilea capitol, am focalizat atenția asupra funcționalităților principale ale sistemului informatic. Am detaliat fiecare funcționalitate în parte, evidențiind modul în care aceasta contribuie la atingerea obiectivelor propuse. De asemenea, am creat diagrame specifice, precum diagrama generală a cazurilor de utilizare, diagrama de clase, diagrame de activitate și diagrame de stare, pentru a ilustra interacțiunile și relațiile dintre diferitele componente ale sistemului.

Capitolul trei a avut ca obiectiv proiectarea propriu-zisă a sistemului informatic. Am abordat aspectele legate de machetele principale de intrare-ieșire, creând astfel o reprezentare vizuală a interfeței aplicației. De asemenea, am proiectat ierarhia paginilor *web*, asigurând o navigabilitate intuitivă și eficientă prin aplicație. Nu în ultimul rând, am detaliat structura și *designul* bazei de date, asigurând faptul că aceasta este optimizată pentru stocarea și gestionarea informațiilor relevante.

În capitolul patru, am trecut la etapa de implementare a aplicației propuse. Am pus în practică toate conceptele și deciziile luate în fazele anterioare, elaborând algoritmi și programe specifice pentru fiecare funcționalitate. Am detaliat pașii necesari pentru utilizarea aplicației, oferind astfel un ghid de utilizare.

Prin urmare, prin structura sa bine definită, lucrarea acoperă în mod exhaustiv întregul proces de dezvoltare a aplicației *web* de achitare și administrare a impozitelor și taxelor locale. Fiecare capitol aduce contribuții specifice și completează povestea creării aplicației, evidențiind atât aspectele teoretice, cât și cele practice ale acestui proiect.

În urma dezvoltării și implementării aplicației *web* de achitare și administrare a impozitelor și taxelor locale, s-a avut în vedere eficientizarea procesului de plată a obligațiilor fiscale ale persoanelor fizice, contribuabililor, într-o societate în care digitalizarea devine din ce în ce mai pronunțată. Astfel, sistemele fiscale trebuie să se adapteze la aceste schimbări și să ofere soluții moderne și accesibile pentru îndeplinirea obligațiilor fiscale.

Aplicația *web* dezvoltată își propune să simplifice și automatizeze procesul de achitare a impozitelor și taxelor locale, oferind contribuabililor o modalitate eficientă și ușor accesibilă de a-și îndeplini obligațiile fiscale. Prin intermediul platformei online, utilizatorii pot accesa și gestiona impozitele și taxele într-un mod simplu și comod, evitând necesitatea deplasării fizice la ghișeele administrative.

Unul dintre obiectivele urmărite a fost reducerea erorilor și timpului de procesare prin automatizarea procesului de înregistrare și procesare a plăților. Astfel, utilizatorii beneficiază de un sistem precis și rapid, eliminând riscul de erori umane și asigurând o gestionare eficientă a plăților.

De asemenea, accesibilitatea a fost un aspect important luat în considerare. Aplicația *web* poate fi utilizată oricând și de oriunde, prin intermediul dispozitivelor conectate la internet. Astfel, contribuabilii au posibilitatea de a verifica situația impozitelor și taxelor și de a efectua plăți în mod convenabil și accesibil.

În ceea ce privește cercetările viitoare, există mai multe direcții de dezvoltare și îmbunătățire a aplicației. Integrarea cu alte sisteme și platforme relevante, cum ar fi sistemele de evidență fiscală sau portalurile guvernamentale, ar spori interconectivitatea și ar permite o gestionare mai eficientă a informațiilor.

Dezvoltarea funcționalităților de comunicare și suport pentru contribuabili ar oferi utilizatorilor posibilitatea de a adresa întrebări, de a solicita clarificări sau de a obține asistență în legătură cu aspecte specifice legate de impozite și taxe.

În plus, adăugarea funcționalităților de raportare și analiză ar oferi utilizatorilor informații detaliate despre plățile efectuate, tendințele fiscale și alte date relevante, sprijinind procesul de luare a deciziilor informate și optimizarea proceselor fiscale.

Astfel, dezvoltarea aplicației *web* de achitare și administrare a impozitelor și taxelor locale reprezintă doar începutul unei evoluții continue, în concordanță cu tendințele digitale și nevoile contribuabililor, în vederea asigurării unui sistem fiscal modern, eficient și accesibil pentru toți utilizatorii.

ANEXE

Anexa 4.

Codul sursă al aplicației informatice (fragmente)

A. Model sequelize creat pentru Utilizator_cu_cont

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');
const Utilizator = require('../utilizator');

const UtilizatorCuCont = sequelize.define(
  'UtilizatorCuCont',
  {
    id_utilizator: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      references: {
        model: Utilizator,
        key: 'id'
      },
      field: 'ID_UTILIZATOR'
    },
    username: {
      type: DataTypes.STRING(32),
      allowNull: false,
      field: 'USERNAME'
    },
    password: {
      type: DataTypes.STRING(60),
      allowNull: false,
      field: 'PASSWORD'
    },
    poza_profil: {
      type: DataTypes.BLOB,
      field: 'POZA_PROFIL'
    }
  }, {
    tableName: 'UTILIZATORI_CU_CONT',
    timestamps: false, // Disable timestamps (createdAt, updatedAt)
    freezeTableName: true
  });

UtilizatorCuCont.belongsTo(Utilizator, {
  foreignKey: 'id_utilizator',
  onDelete: 'CASCADE',
  onUpdate: 'CASCADE'
});

module.exports = UtilizatorCuCont;
```

B. Model sequelize pentru Utilizator

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const Utilizator = sequelize.define(
  'Utilizator',
  {
    id: {
      type: DataTypes.INTEGER,
      primaryKey: true,
      autoIncrement: true,
      field: 'ID'
    },
    cod_identificare: {
      type: DataTypes.STRING(13),
      allowNull: false,
      unique: 'utilizatori_con_unique_cod_identificare',
      field: 'COD_IDENTIFICARE'
    },
    nume: {
      type: DataTypes.STRING(32),
      allowNull: false,
      field: 'NUME'
    },
    prenume: {
      type: DataTypes.STRING(32),
      allowNull: false,
      field: 'PRENUME'
    },
    data_nastere: {
      type: DataTypes.DATE,
      allowNull: false,
      field: 'DATA_NASTERE'
    },
    email: {
      type: DataTypes.STRING(64),
      allowNull: false,
      unique: 'utilizatori_con_unique_email',
      validate: {
        isEmail: true, //checks for email format
        isEmail: {
          msg: "Formatul acceptat este de email! ex: foo@bar.com"
        }
      },
      field: 'EMAIL'
    },
    telefon: {
      type: DataTypes.STRING(10),
      allowNull: true,
      validate: {
        isPhoneNumberFormat(value) {
          if (value === null) {
            return;
          }

          const phoneRegex = /^\\d{10}$/;
          if (!phoneRegex.test(value)) {
            throw new Error('Invalid telephone number format.');
          }
        }
      },
      field: 'TELEFON'
    }
  },
  {
    tableName: 'utilizatori'
  }
);
```

```

    },
    field: 'TELEFON'
  },
  adresa: {
    type: DataTypes.STRING(64),
    allowNull: true,
    field: 'ADRESA'
  },
  gen: {
    type: DataTypes.STRING(1),
    allowNull: true,
    field: 'GEN'
  }
}, {
  tableName: 'UTILIZATORI',
  timestamps: false,
  freezeTableName: true
});

module.exports = Utilizator;

```

C. Funcție pentru parsarea textului și extragerea datelor de interes

```

const extractData = () => {
  return new Promise((resolve, reject) => {
    const formData1 = new FormData();
    const file = files[0]
    formData1.append('newFile', file)
    console.log(formData1)

    axios.post('http://localhost:8080/extract-text', formData1)
      .then((res) => {
        let ok = true;
        const keywords = ['iei:', 'ul:', 'Localitatea:', 'CUI:', 'Adresa:', 'Telefon de contact:',
          'Adresa de mail publică:', 'Link site:', 'bonificare:', 'penalizare:', 'Lista impozite și taxe:'];
        const extractedData = new FormData();
        const taxData = [];

        let startExtraction = false;
        const lines = res.data.split('\n');

        lines.forEach(line => {
          if (startExtraction) {
            const taxValues = line.split(', ');

            if (taxValues.length >= 3) {
              const taxEntry = {
                cont_iban: taxValues[0].trim() || null,
                cod_cont: taxValues[1].trim() || null,
                denumire: taxValues[2].trim() || null,
                suma: taxValues[3] ? taxValues[3].trim() : null,
              };

              taxData.push(taxEntry);
            }
          }

          if (line.includes(keywords[8])) {
            startExtraction = true;
          }
        });
      })
  });
}

```

```

    });

    let listaImpoziteAdded = false;

    keywords.forEach(element => {
      const institutionNameRegex = new RegExp(`${element}(.)`, 'u');
      const match = res.data.match(institutionNameRegex);
      if (match) {
        const matchedPhrase = match[1];
        if (element === 'Lista impozite și taxe:' && !listaImpoziteAdded) {
          extractedData.append(element, JSON.stringify(taxData));
          listaImpoziteAdded = true;
        } else {
          extractedData.append(element, matchedPhrase.trim());
        }
      } else if (['iei:', 'ul:', 'Localitatea:', 'CUI:', 'Adresa:', 'Adresa de mail publică:',
'bonificare:', 'penalizare:'].includes(element)) {
        setError('Formularul trimis nu este complet');
        ok = false;
        return;
      } else {
        console.log("No match found");
        ok = false;
      }
    });

    console.log(extractedData);

    if (ok) {
      insertDataToInstitutiiPublice(extractedData, formData1, taxData);
      toggleModal();
      setFiles([]);
      setDisabled(true);
      resolve();
    } else {
      reject(new Error('Extraction failed'));
    }

    console.log('Tax Data:', taxData);
  })
  .catch((err) => {
    reject(err);
  });
});
};

```

D. Funcția specifică operației de logout

```
app.post('/logout', (req, res) => {
  req.session.destroy((error) => {
    if (error) {
      console.error('Error destroying session:', error);
      res.status(500).json({ error: 'Failed to destroy session' });
    } else {
      console.log('Session destroyed!');
      res.clearCookie('userId', {
        domain: 'localhost',
        path: '/',
        httpOnly: true,
      });
      res.sendStatus(200);
    }
  });
});
```

E. Funcția ce conduce către portalul de plată

```
const handleSubmit = (e) => {
  e.preventDefault();
  const errors = {};

  inputFields.forEach((field) => {
    if (formValues[field.id].trim() === '') {
      errors[field.id] = 'Va rugăm completați câmpul';
    }
  });

  setFormErrors(errors);
  if (Object.keys(errors).length === 0) {

    const cui = dropdownOptionsInstitutie.find((option) => option.value === formValues['institutie']).cui
    let id_taxa = null
    let nume_taxa = null
    let id_impozit = null
    let nume_impozit = null

    const selectedOption = dropdownOptionsTaxa.find(
      (option) => option.value === formValues['taxa']
    );

    if (selectedOption && selectedOption.suma !== null) {
      id_taxa = selectedOption.id;
      nume_taxa = formValues['taxa']
    } else {
      id_impozit = selectedOption.id;
      nume_impozit = formValues['taxa']
    }

    const metadata = {
      cod_identificare_platitor: formValues.codPlatitor,
      cod_identificare_beneficiar: formValues.codBeneficiar,
      suma: formValues.suma,
      cui_institutie: cui,
      id_taxa: id_taxa,
      nume_taxa: nume_taxa,
      id_impozit: id_impozit,
```

```

        nume_impozit: nume_impozit,
        nume: formValues.nume,
        prenume: formValues.prenume,
        email: formValues.email,
    }
    const item = { id: id_taxa !== null ? id_taxa : id_impozit, quantity: 1 }
    axios.post("http://localhost:8080/create-checkout-session-plata-neautenticata", {
        items: item,
        metadata: metadata,
    }, {
        headers: {
            "Content-Type": "application/json",
        },
    })
    .then((res) => {
        if (res.status === 200) return res.data;
        return Promise.reject(res.data);
    })
    .then(({ url, session_id }) => {
        localStorage.setItem("session_id", session_id);
        window.location = url;
    })
    .catch((error) => {
        console.error(error);
    });
}
};

```

F. Metoda POST specifică unei plăți realizate cu succes

```

app.post("/payment-success-plata-neautenticata", async (req, res) => {
    try {
        const session_id = req.body.session_id;
        const session = await stripe.checkout.sessions.retrieve(session_id);

        const metadata = session.metadata;

        const bodyUtilizator = {
            cod_identificare: metadata.cod_identificare_platitor,
            nume: metadata.nume,
            prenume: metadata.prenume,
            data_nastere: extractDataNastere(metadata.cod_identificare_platitor),
            email: metadata.email,
            telefon: null,
            adresa: null,
            gen: extractGen(metadata.cod_identificare_platitor),
        }

        const utilizatorResponse = await axios.post('http://localhost:8080/api/add-utilizator', bodyUtilizator)
        const id_utilizator = utilizatorResponse.data.id;

        const bodyPlata = {
            data_efectuaire: new Date(),
            suma: metadata.suma,
            cod_identificare_beneficiar: metadata.cod_identificare_beneficiar,
            cui_institutie: metadata.cui_institutie,
            id_card_bancar: null,
            id_utilizator: id_utilizator
        }
    }
}

```

```

const plataResponse = await axios.post('http://localhost:8080/api/add-plata', bodyPlata)
const id_plata = plataResponse.data.id;

const bodyElem = {
  id_plata: id_plata,
  id_impozit_anuntat: null,
  id_impozit: metadata.id_impozit !== '' ? metadata.id_impozit : null,
  id_taxa: metadata.id_taxa !== '' ? metadata.id_taxa : null,
}
console.log('bodyElem:', bodyElem)

const elemResponse = await axios.post('http://localhost:8080/api/add-element-plata', bodyElem)

res.status(200).json({ clearLocalStorage: true });
} catch (e) {
  res.status(500).json({ error: e.message });
}
});

```