Social Monitor - Platformă Integrată pentru Managementul Cazurilor Sociale

Proiectarea Interefelor utilizator

Autori: Mihaila Claudiu, Monoranu Tudor, Rafa Ioana, Vadean Catalin

Grupa: 30243

Specializarea: Calculatoare

Facultatea de Automatică
și Calculatoare

19 ianuarie 2026

Cuprins

1 Introducere

1.1 Contextul Proiectului

În contextul actual al digitalizării serviciilor publice, gestionarea eficientă a resurselor umane și materiale pentru asistența socială reprezintă o provocare majoră. Persoanele vulnerabile (vârstnici, persoane cu dizabilități, sinistrați) depind adesea de o rețea complexă de asistenți sociali și voluntari. Lipsa unei coordonări centralizate duce la întârzieri în livrarea ajutoarelor și la o monitorizare precară a stării beneficiarilor.

Platforma **Social Monitor** a fost dezvoltată pentru a răspunde acestor nevoi, oferind o soluție software robustă care facilitează comunicarea în timp real între coordonatori (administratori) și agenții de teren (voluntari).

1.2 Obiectivele Sistemului

Obiectivul principal este crearea unui ecosistem digital care să permită:

- **Centralizarea datelor:** Stocarea securizată a informațiilor despre beneficiari și istoricul intervențiilor.
- **Dispecerizare eficientă:** Alocarea dinamică a sarcinilor către voluntari în funcție de disponibilitate.
- **Monitorizare în timp real:** Urmărirea statusului intervențiilor (Preluată $\rightarrow$ În Curs $\rightarrow$ Finalizată).
- **Feedback și Evaluare:** Implementarea unui mecanism de recenzie bidirecțional (Voluntar-Beneficiar).

2 Arhitectura Sistemului

Sistemul urmează o arhitectură de tip **Client-Server** (Monolithic Repository), separând clar responsabilitățile între interfața utilizator și logica de afaceri.

2.1 Stiva Tehnologică (Tech Stack)

Aplicația este construită folosind tehnologii moderne, specifice ecosistemului JavaScript:

1. **Frontend (Client):**
   - **React.js**: Bibliotecă pentru construirea interfețelor reactive.
   - **CSS3**: Pentru stilizarea componentelor (fișier dedicat `App.css`).
   - **Fetch API**: Pentru comunicarea asincronă cu serverul.
2. **Backend (Server):**
   - **Node.js**: Mediu de execuție pentru JavaScript pe server.
   - **Express.js**: Framework web minimalist pentru gestionarea rutelor API.
3. **Persistența Datelor:**

- **SQLite**: Bază de date relațională, aleasă pentru portabilitate și performanță ridicată în scenarii de tip embedded.
- **sqlite3 / sqlite**: Drivere pentru interacțiunea Node.js cu baza de date.

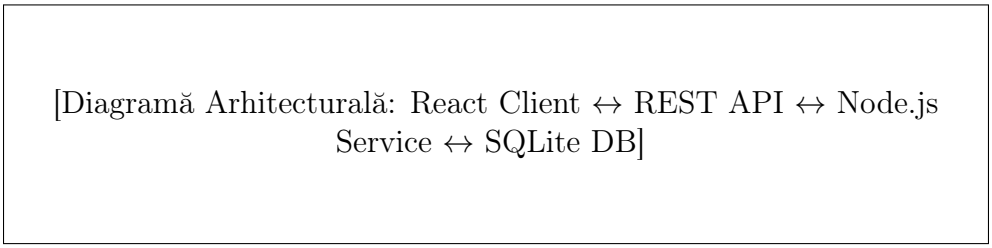

Figura 1: Schema bloc a arhitecturii aplicației

3 Proiectarea Bazei de Date

Modelul de date este unul relațional, fiind proiectat pentru a asigura integritatea referențială între entități. Structura bazei de date este definită în fișierul `backend/db.js` și constă în următoarele tabele principale.

3.1 Schema Relațională

Mai jos este prezentat codul SQL utilizat pentru inițializarea tabelelor critice ale sistemului. Se observă utilizarea cheilor străine (*Foreign Keys*) pentru a lega intervențiile de beneficiari și voluntari.

```
-- Tabela pentru voluntari
CREATE TABLE IF NOT EXISTS voluntari (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nume TEXT NOT NULL UNIQUE
);

-- Tabela pentru beneficiari
CREATE TABLE IF NOT EXISTS beneficiari (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  nume TEXT NOT NULL,
  categorie TEXT NOT NULL -- ex: Pensionar, Handicap
);

-- Tabela centrala: Interventii
CREATE TABLE IF NOT EXISTS interventii (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  tip TEXT NOT NULL,
  status TEXT NOT NULL,
  descriere TEXT,
  beneficiarId INTEGER NOT NULL,
  voluntarId INTEGER,
  createdAt TEXT NOT NULL,
  -- Constrangeri de integritate
  FOREIGN KEY(beneficiarId) REFERENCES beneficiari(id) ON DELETE
   RESTRICT,
  FOREIGN KEY(voluntarId) REFERENCES voluntari(id) ON DELETE SET NULL
);
```

Listing 1: Definirea schemei bazei de date în backend/db.js

3.2 Entități Secundare

Pe lângă entitățile principale, sistemul gestionează și:

- **Urgențe:** Stochează alertele emise de voluntari din teren.
- **Vizite:** Gestionează calendarul de vizite programate.
- **Evaluări:** Tabelele `evaluari_voluntari` și `evaluari_beneficiari` permit stocarea scorurilor (1-10) și a observațiilor calitative.

4 Implementare Backend (API REST)

Serverul expune un API RESTful care permite clientului să efectueze operații CRUD (Create, Read, Update, Delete). Implementarea se regăsește în fișierul `backend/server.js`.

4.1 Gestionarea Intervențiilor

Unul dintre cele mai complexe endpoint-uri este cel de preluare a intervențiilor (`GET /api/interventii`). Acesta realizează un *JOIN* între tabele pentru a furniza frontend-ului toate detaliile necesare (nume beneficiar, nume voluntar) într-un singur răspuns JSON.

```
app.get("/api/interventii", async (req, res) => {
  try {
    const db = await getDb();
    // Suporta filtrare optionala pentru vizualizarea voluntarului
    const voluntarId = req.query.voluntarId ? Number(req.query.voluntarId) : null;

    const sql = `
      SELECT i.id, i.tip, i.status, i.descriere, i.createdAt,
             b.id as beneficiarId, b.nume as beneficiar, b.categorie as categorie,
             v.id as voluntarId, v.nume as voluntar
      FROM interventii i
      JOIN beneficiari b ON b.id = i.beneficiarId
      LEFT JOIN voluntari v ON v.id = i.voluntarId
      ${voluntarId ? "WHERE i.voluntarId = ?" : ""}
      ORDER BY i.id DESC
    `;
    const rows = voluntarId ? await db.all(sql, [voluntarId]) : await db.all(sql);
    res.json(rows);
  } catch (e) {
    console.error(e);
    res.status(500).json({ message: "Eroare la interventii" });
  }
});
```

Listing 2: Endpoint-ul GET pentru listarea intervențiilor

4.2 Mecanismul de Atribuire

Atribuirea unei sarcini către un voluntar este o operațiune atomică ce actualizează starea intervenției din "Neatribuită" în "Preluată".

```
app.post("/api/atribuiri", async (req, res) => {
  try {
    const interventieId = Number(req.body?.interventieId);
    const voluntarId = Number(req.body?.voluntarId);

    // Validare input
    if (!interventieId || !voluntarId) {
      return res.status(400).json({ message: "ID-uri obligatorii" });
    }

    const db = await getDb();
    // Actualizare relatie si status
    await db.run("UPDATE interventii SET voluntarId=?, status=? WHERE id
   =?", [
      voluntarId,
      "Preluata",
      interventieId,
    ]);

    res.json({ ok: true });
  } catch (e) {
    res.status(500).json({ message: "Eroare la atribuire" });
  }
});
```

Listing 3: Logica de atribuire (POST /api/atribuiri)

5 Implementare Frontend

Interfața utilizator este modulară, folosind componente reutilizabile și un sistem de routing intern bazat pe starea aplicației.

5.1 Gestionarea Stării Globale și Rutare

În fișierul `src/App.jsx`, starea aplicației este controlată prin Hook-ul `useState`. Aici se definește logica de afișare condiționată în funcție de rolul utilizatorului (Administrator sau Voluntar).

```
function App() {
  const [role, setRole] = useState("admin"); // admin | voluntar
  const [page, setPage] = useState("interventii");
  const [voluntari, setVoluntari] = useState([]);

  // Functie de securitate pe frontend (ascunde meniuri)
  function canSee(p) {
    if (role === "admin") return true;
    // Voluntarii nu au acces la atribuire sau gestionare voluntari
    return !["atribuire", "voluntari"].includes(p);
  }

```

```
  // ... (cod pentru randarea Sidebar-ului si a Content-ului)
}
```

Listing 4: Logica de roluri din App.jsx

5.2 Componenta Intervenții

Componenta `Interventii.jsx` este inima operațională a aplicației. Aceasta implementează o mașină de stări simplă pentru ciclul de viață al unei sarcini.

Funcția `nextStatus` determină automat următoarea stare validă a unei intervenții:

```
const STATUSURI = ["Preluata", "In curs", "Finalizata"];

function nextStatus(current) {
  const idx = STATUSURI.indexOf(current);
  if (idx === -1) return "Preluata";
  // Cicleaza prin statusuri
  return STATUSURI[(idx + 1) % STATUSURI.length];
}

async function updateStatus(id, currentStatus) {
  const ns = nextStatus(currentStatus);
  await apiSend(`/interventii/${id}/status`, "PATCH", { status: ns });
  showToast("Status actualizat!");
  loadAll();
}
```

Listing 5: Logica de tranzitie a stărilor (State Machine)

5.3 Componenta de Evaluare

Sistemul de evaluare (`Evaluare.jsx`) permite colectarea de metrici de performanță. Interfața se adaptează dinamic:

- Dacă utilizatorul este **Admin**, vede un formular pentru a evalua un Voluntar.
- Dacă utilizatorul este **Voluntar**, vede un formular pentru a evalua starea Beneficiarului.

6 Ghid de Utilizare și Scenarii

6.1 Scenariul 1: Fluxul Complet al unei Intervenții

1. **Creare:** Administratorul accesează pagina *Intervenții*, selectează tipul (ex: "Livrare alimente") și beneficiarul, apoi apasă "Creează". Intervenția apare cu status "Neatribuită".
2. **Atribuire:** Administratorul navighează la pagina *Atribuire cazuri*. Selectează intervenția creată și voluntarul "Popescu Ion". Confirmă prin "Asignează".
3. **Execuție:** Voluntarul "Popescu Ion" se autentifică. În lista sa de *Intervenții*, vede sarcina cu status "Preluată".

4. **Finalizare:** Când ajunge la beneficiar, voluntarul apasă "Actualizează status" → "În curs". După livrare, apasă din nou → "Finalizată".

6.2 Scenariul 2: Raportarea unei Urgențe

În cazul unei situații critice la domiciliul beneficiarului, voluntarul accesează pagina *Urgență*. Completează descrierea situației în caseta de text și apasă "Trimite". Alerta este salvată instantaneu în baza de date și devine vizibilă administratorilor.

7 Concluzii

Proiectul **Social Monitor** demonstrează o implementare funcțională și scalabilă a unui sistem de management pentru asistența socială. Arhitectura modulară permite extinderea facilă cu noi funcționalități (ex: hărți GPS, notificări Push), iar structura codului respectă standardele industriei pentru claritate și mentenanță.