Contents

Welcome to “Sync” Task Management App

Full-Stack: PostgreSQL + Prisma + Express + Node.js + Zustand + TailwindCSS + JWT + Google API
Version: 1.0
Document Type: Technical Guide
GitHub: github.com/naeioana
Website: testApp.Sync.ro
Date: 2025 **Author:** Nae Ioana

Chapter 1

Capitolul 1: Introducere

1.1 SYNC - REZUMAT TEHNIC

1.1.1 Ce este această aplicație?

Sync reprezintă o abordare nouă în modul în care organizațiile gestionează proiectele și colaborează în echipă. Nu este doar o aplicație obișnuită de urmărire a task-urilor; este o soluție completă, construită la standarde enterprise, care integrează bune practici de arhitectură software modernă și răspunde provocărilor reale ale managementului de proiect în mediul rapid de astăzi.

Aplicația demonstrează cum poate fi creat un sistem funcțional, sigur și scalabil, care să reziste cerințelor reale ale utilizatorilor. Este un punct de referință pentru dezvoltatori, lideri tehnici și organizații care vor să înțeleagă cum se construiesc aplicații gata de producție, cu arhitectură solidă și performanță constantă.

Ceea ce diferențiază Sync de alte unelte de management este abordarea sa holistică. În loc să se concentreze doar pe task-uri, aplicația gestionează întregul ecosistem de colaborare, inclusiv securitatea datelor, cerințele de scalabilitate și integrarea fără probleme cu procesele existente ale organizației. În același timp, servește ca resursă educațională, exemplificând bune practici în dezvoltarea full-stack, implementarea securității și design-ul sistemelor.

1.1.2 Ce funcţionalităţi oferă?

1.1.2.1 Funcţionalitate de bază

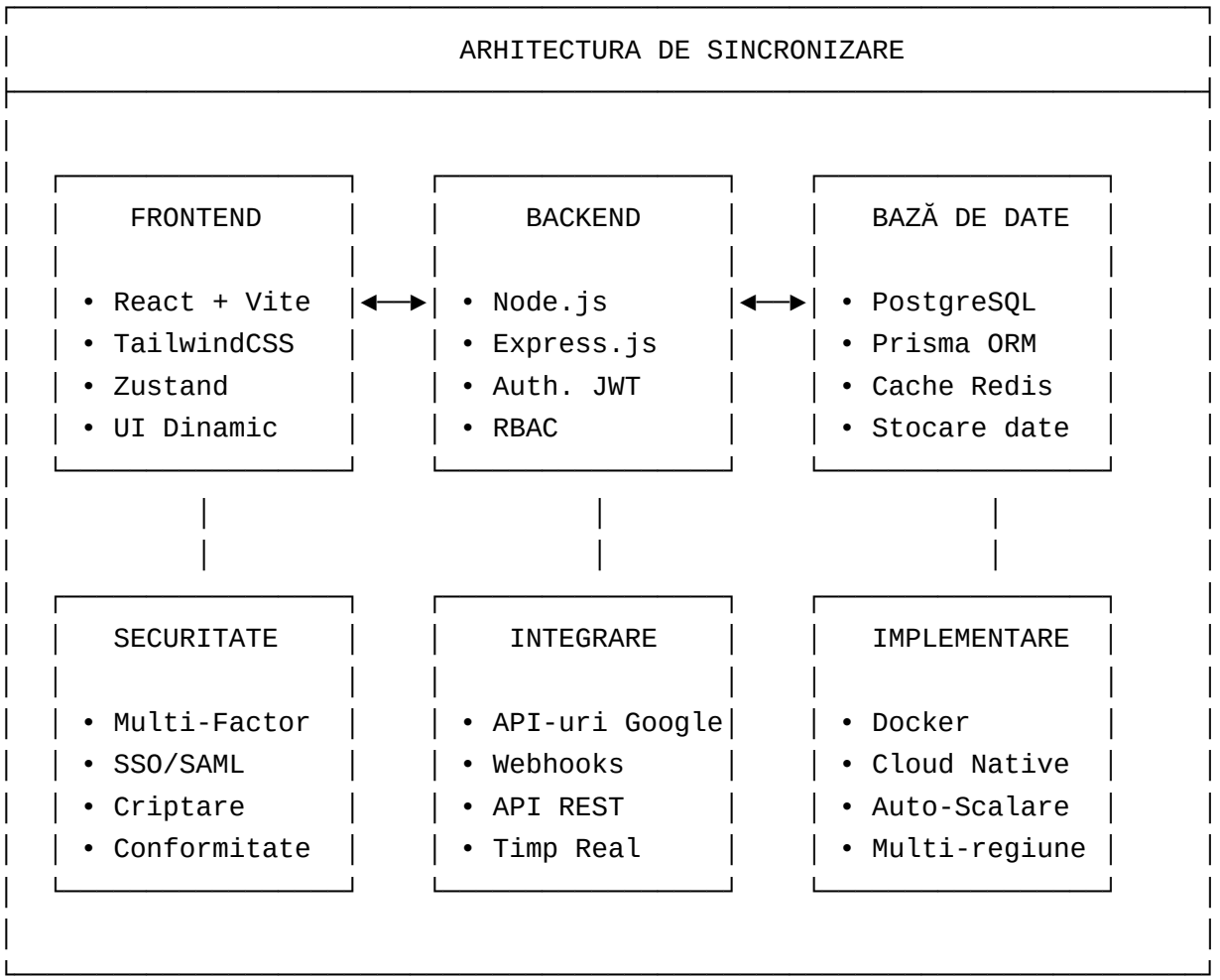
Scopul principal al aplicației este să transforme modul în care echipele planifică, organizează și realizează munca. La baza acestei transformări se află panoul Kanban, care oferă o reprezentare vizuală a fluxului de lucru, mult mai clară decât simplele liste de task-uri. Interfața drag-and-drop permite mutarea sarcinilor între stadii – de la idee, la în desfășurare și finalizare – fără pierderi de timp sau confuzie. Această abordare vizuală reduce povara administrativă și permite echipei să se concentreze pe execuție.

Gestionarea task-urilor depășește operațiile CRUD de bază. Fiecare task devine un container complet de informații: descrieri detaliate, priorități ajustabile, termene realiste cu notificări automate și atribuirea clară a responsabilităților. Sistemul urmărește nu doar starea curentă a sarcinilor, ci și istoricul complet al modificărilor: cine a făcut schimbări, când și ce modificări au fost aplicate. Această trasabilitate oferă transparența necesară pentru o guvernanță eficientă a proiectelor.

Un alt element esențial este controlul bazat pe roluri. Aplicația implementează o structură de permisiuni sofisticată, care reflectă ierarhiile reale din organizații, păstrând în același timp flexibilitatea necesară colaborării eficiente. Administratorii au acces complet la configurarea aplicației, gestionarea utilizatorilor și analiza datelor,

în timp ce utilizatorii obişnuiţi pot vedea şi gestiona doar task-urile şi informaţiile relevante pentru ei. Acest nivel granular de permisiuni asigură protecţia datelor sensibile şi permite, în acelaşi timp, colaborarea fluentă necesară succesului proiectelor.

1.2 Prezentare generală a arhitecturii sistemului



1.3 Ce face „SYNC” special

În mediul rapid de dezvoltare de astăzi, echipele au nevoie de mai mult decât urmărirea simplă a task-urilor. Ele necesită:

- **Arhitectură dinamică a componentelor**: UI bazat pe React, cu TailwindCSS pentru interfețe responsive și plăcute vizual
- **Arhitectură scalabilă**: Fundament care crește odată cu nevoile afacerii
- **Arhitectură cloud-native**: Integrare Google Cloud Storage cu CDN global și edge computing
- **Securitate pe mai multe niveluri**: Securitate enterprise cu JWT, RBAC și funcționalități de conformitate
- **Design avansat al bazei de date**: PostgreSQL cu Prisma ORM pentru operațiuni sigure și performante
- **Colaborare în timp real**: Actualizări live și colaborare de echipă bazate pe WebSocket
- **Funcţionalităţi bazate pe AI**: Machine learning pentru analiză de conținut și automatizare inteligentă
- **Analiză completă**: Informații în timp real și monitorizare a performanței
- **Pregătită pentru DevOps**: Integrare Docker, Kubernetes și pipeline CI/CD
- **Management centralizat al proiectelor**: Sursă unică de adevăr pentru toate activitățile

- **Control bazat pe roluri**: Colaborare sigură cu diferite niveluri de permisiuni
- **Urmărirea progresului în timp real**: Vizualizarea stadiului proiectului prin panouri Kanban
- **Gestionarea integrată a fișierelor**: Manipulare ușoară a documentelor asociate task-urilor
- **Securitate enterprise**: Securitate pe mai multe niveluri, cu capabilități de audit și conformitate
- **Analize și automatizare AI**: Analitică inteligentă și automatizare
- **Design Mobile-First**: Interfețe responsive pentru toate dispozitivele
- **Integrare AI**: Machine learning pentru analiză de conținut

1.4 Prezentare generală a funcțiilor de bază

1.4.1 Managementul panoului Kanban

Elementul central al aplicației este un sistem Kanban avansat, care pune la dispoziție:

- **Drag & Drop**: Mutarea intuitivă a sarcinilor între coloane
- **Culori pentru Prioritate**: Indicatori vizuali ai priorității (Roșu=Ridicată, Galben=Medie, Verde=Scăzută)
- **Urmărirea Progresului**: Procente de finalizare în timp real
- **Filtrare**: Sortare după responsabil, prioritate, termen sau etichete
- **Căutare**: Găsire rapidă a sarcinilor pe toate panourile
- **Actualizări în Timp Real**: Colaborare live cu membrii echipei
- **Compatibil pe Mobil**: Optimizat pentru toate dimensiunile de ecran
- **Fluxuri de Lucru Personalizate**: Configurare flexibilă a panourilor și statusurilor
- **Navigare**: Navigare adaptivă cu meniu utilizator și autentificare
- **Tablou de Bord**: Grafice și metrici interactive în timp real
- **Panou Kanban**: Drag & drop cu actualizări live și filtre
- **Formulare pentru Sarcini**: Validare, auto-salvare și editor de text îmbogățit
- **Încărcare Fișiere**: Drag & drop cu bară de progres și previzualizare
- **Profil Utilizator**: Setări, preferințe și urmărirea activității

DE FĂCUT	ÎN CURS	ÎN REVIZUIRE	FINALIZAT
• Sarcină 1	• Sarcină 4	• Sarcină 6	• Sarcină 8
• Sarcină 2	• Sarcină 5	• Sarcină 7	• Sarcină 9
• Sarcină 3			• Sarcină 10

1.4.2 Sistem de Design TailwindCSS

Sistem TailwindCSS pentru interfețe consistente, responsive și atractive:

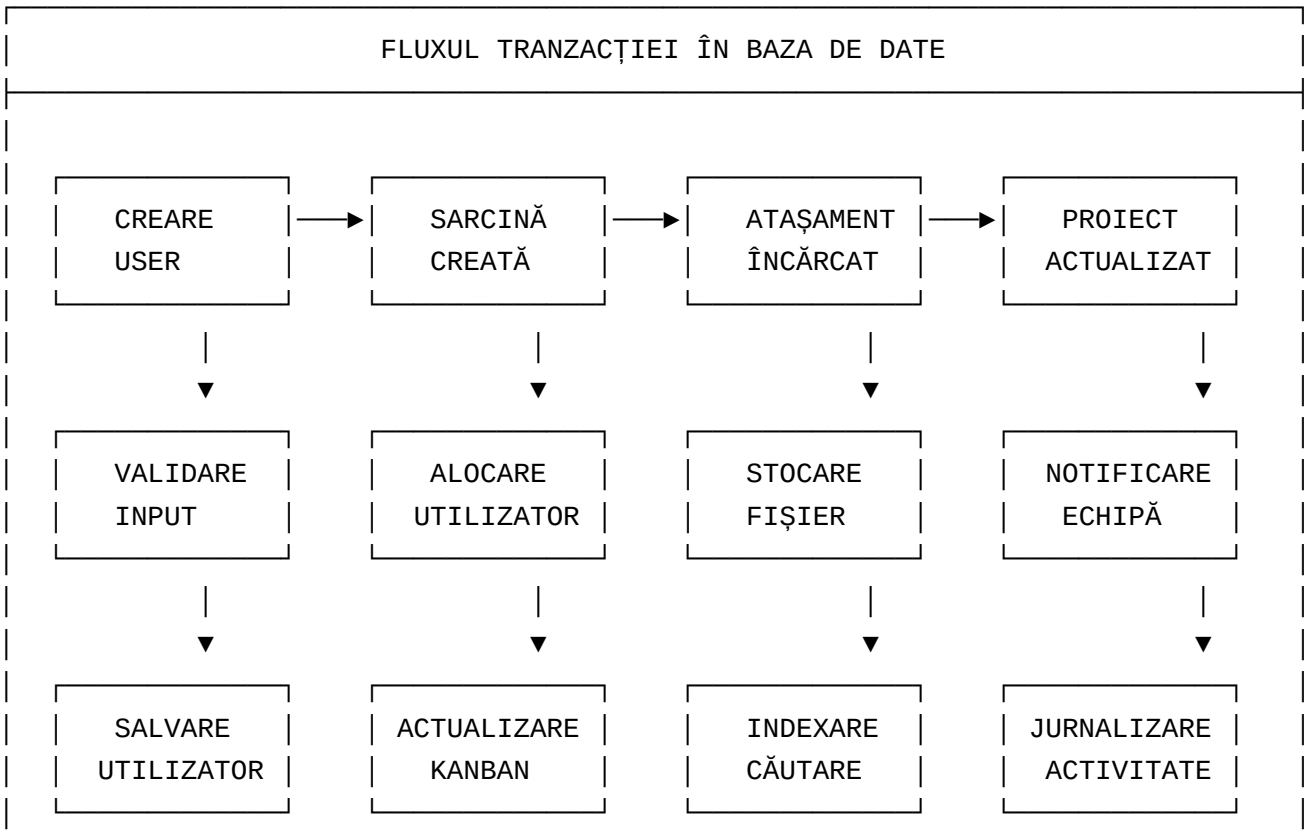
- **Schema de culori**: Tematici pentru primar, secundar, accent, succes, avertisment, eroare
- **Tipografie**: Titluri, text de corp, monospace, design responsive
- **Componente**: Butoane, carduri, formulare, ferestre modale și alerte
- **Responsive**: Mobile-first cu puncte de întrerupere adaptabile și sistem grid
- **Animaţii**: Tranziţii line, efecte la hover și stări de încărcare
- **Utilitare**: Spațiere completă, flexbox și utilitare pentru grid

1.4.3 Operaţiuni CRUD

Funcţionalitate completă de Create, Read, Update, Delete pentru:

- **Sarcini**: Titlu, descriere, status, prioritate, termene
- **Proiecte**: Organizare și categorisire
- **Utilizatori**: Administrarea profilului și atribuirea rolurilor
- **Atașamente**: Încărcare și gestionare fișiere

1.4.4 Arhitectura Tranzacţiilor în Baza de Date



1.4.5 Role-Based Access Control (RBAC)

Permisiuni:

Acțiune	Super Admin	Company Admin	Project Manager	Team Member
Gestionare Utilizatori	x	x	-	-
Creare Proiect	x	x	x	-
Gestionare Sarcini	x	x	x	x
Acces Fișiere	x	x	x	x
Setări Sistem	x	x	-	-

1.4.6 Gestionarea Fișierelor

Soluție integrată Google API pentru gestionarea fișierelor la nivel enterprise:

Funcții de Securitate pentru Fișiere:
- **Validare Multi-Strat**: Verificarea extensiei fișierului, tipului MIME și numărului magic
- **Scanare Antivirus**: Integrare cu servicii de securitate pentru detectarea malware-ului
- **Criptare**: Criptare AES-256 pentru toate fișierele stocate
- **Control Acces**: Permisiuni bazate pe roluri și jurnalizare audit
- **Control Versiuni**: Versionare fișiere și istoric modificări
- **Backup & Recuperare**: Backup automat cu recuperare la un punct în timp

1.4.7 Arhitectura Stratului de Securitate

- **Stratul Client**:
  - Aplicarea HTTPS & certificate pinning
  - Protecție XSS & Politica de Securitate a Conținutului (CSP)
  - Token-uri CSRF & cookie-uri SameSite
  - Sanitizarea inputului & codare output
  - Autentificare Multi-Factor (MFA)
  - Gestionare securizată a sesiunilor cu timeout
  - Identificarea dispozitivului (device fingerprinting)
- **Stratul de Transport**:
  - Criptare TLS 1.3 & perfect forward secrecy

  - Validarea certificatelor & headere HSTS
  - Limitare rată & protecție DDoS
  - Filtrare IP & restricții geolocație
  - TLS mutual cu certificate client
  - API gateway securizat cu throttling
  - Rotirea certificatelor & gestionarea ciclului de viață
- **Stratul Aplicație**:
  - Autentificare JWT & rotație token refresh
  - Controlul accesului bazat pe roluri (RBAC)
  - Validare & sanitizare input
  - Prevenirea SQL injection prin query-uri parametrizate
  - Gestionare secrete cu integrare Vault
  - Limitare rată API per utilizator/token
  - Gestionare sigură a erorilor & logare
  - Monitorizare securitate & alerte (SIEM, IDS/IPS)
- **Stratul Date**:
  - Criptare baze de date în repaus
  - Jurnalizare audit & monitorizare activitate
  - Backup criptat & gestionare sigură a cheilor
  - Conformitate GDPR, HIPAA, ISO 27001
  - Mascare date & criptare la nivel de câmp
  - Politici sigure de păstrare a datelor
  - Backup-uri imuabile pentru protecție ransomware
  - Acces baze de date Zero Trust

1.5 Arhitectura Aplicației

1.5.1 Tehnologii Frontend

React 18 + Vite
- Funcționalități React (hooks, context, suspense)
- Server de dezvoltare rapid cu HMR
- Build-uri de producție optimizate
- Tree-shaking pentru dimensiuni minime ale bundle-ului
- Sistem de componente dinamice cu design atomic
- Abordare responsive, mobile-first

Zustand pentru Managementul Stării
- Ușor și performant
- Suport JavaScript cu siguranță tipică
- Capacități middleware pentru logare și persistență
- Integrare DevTools pentru debugging
- Sincronizare stări în timp real
- Arhitectură modulară a magazinului de stări

TailwindCSS
- Framework CSS utility-first
- Sistem de design responsive cu breakpoint-uri
- Bibliotecă de componente personalizate cu design tokens
- Performanță optimizată cu PurgeCSS
- Mod întunecat și schimbare temă
- Principii de design centrate pe accesibilitate

1.5.2 Tehnologii Backend

Node.js + Express
- Arhitectură event-driven cu I/O non-blocant
- Ecosistem complet de middleware
- Design API RESTful cu suport GraphQL
- Suport async/await pentru JavaScript modern
- Integrare WebSocket în timp real
- Gestionare avansată a erorilor și logare

PostgreSQL + Prisma
- Conformitate ACID pentru integritatea datelor
- Operațiuni tip-safe în baza de date cu tipuri generate automat
- Management automat al migrărilor
- Pooling de conexiuni și optimizare query-uri
- Strategii avansate de indexare
- Capabilități de căutare full-text

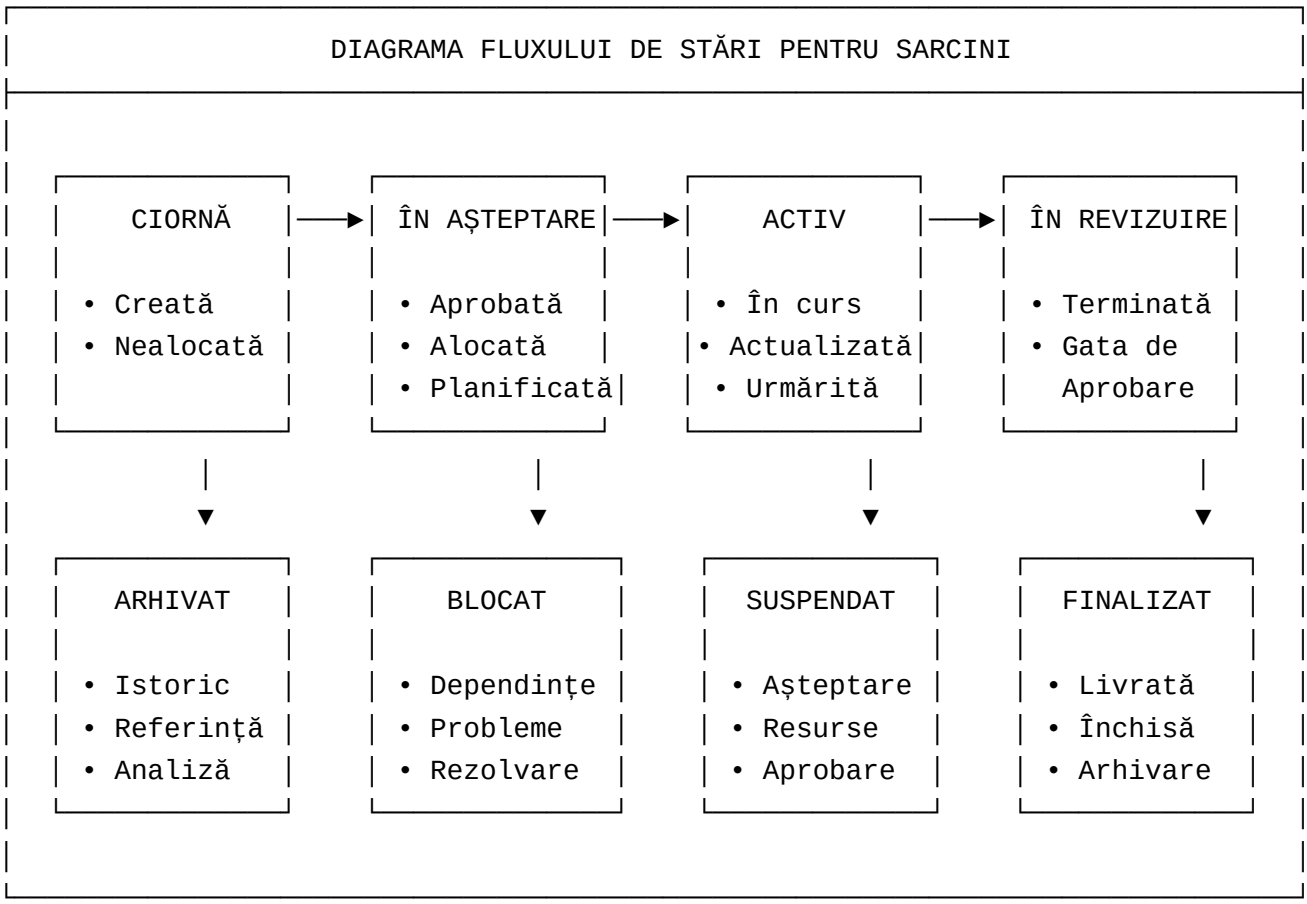
1.5.3 Tehnologii de Securitate

JWT + Cookie-uri HTTP-Only
- Autentificare stateless cu rotație sigură a token-urilor
- Cookie-uri HTTP-only pentru protecție XSS

- Expirare automată și suport pentru token refresh
- Integrare autentificare multi-factor (MFA)
- Gestionare sesiuni și urmărire dispozitive

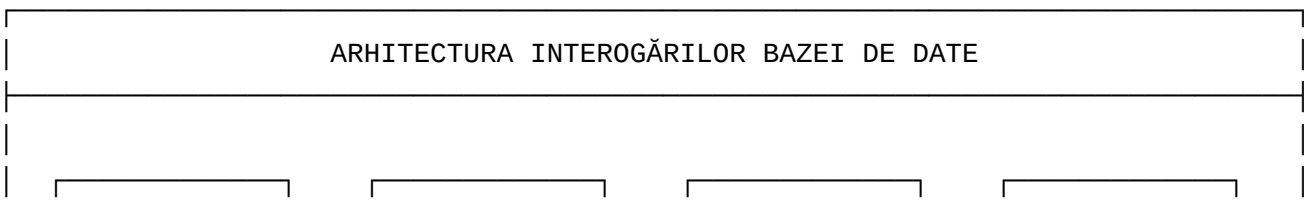
Middleware de Securitate
- Headere de securitate Helmet pentru protecție completă
- Protecție CSRF cu validare token
- Limitare avansată a ratei și protecție DDoS
- Validare și sanitizare multi-strat a inputului
- Detectare și monitorizare amenințări în timp real

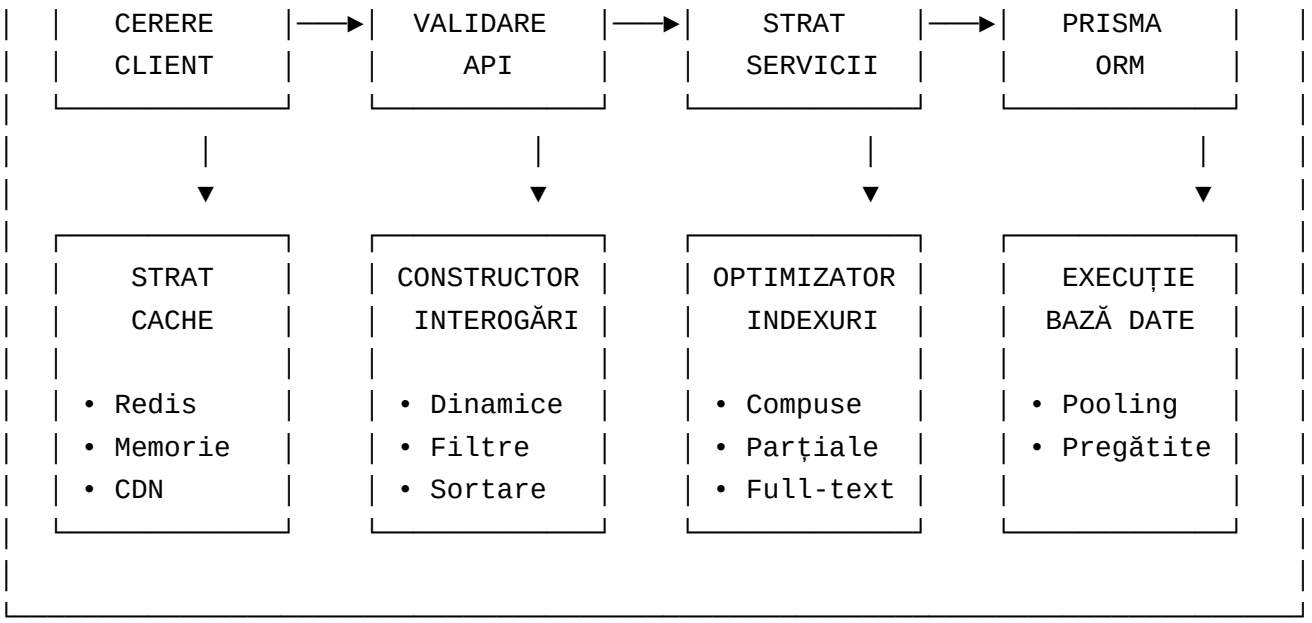
1.5.4 Fluxul Statusului Sarcinilor & Gestionarea Stării În Așteptare



Funcţionalităţi pentru Starea În Aşteptare:
- **Flux de Aprobare**: Sarcinile necesită aprobarea managerului înainte de activare
- **Urmărirea Dependenţelor**: Sarcinile blocate de prerechizite nefinalizate
- **Alocarea Resurselor**: Sarcinile așteaptă disponibilitatea membrilor echipei
- **Prioritizarea Cozii**: Sarcinile cu prioritate înaltă trec în fața cozii de așteptare
- **Programare**: Sarcinile sunt planificate pentru date viitoare de execuție

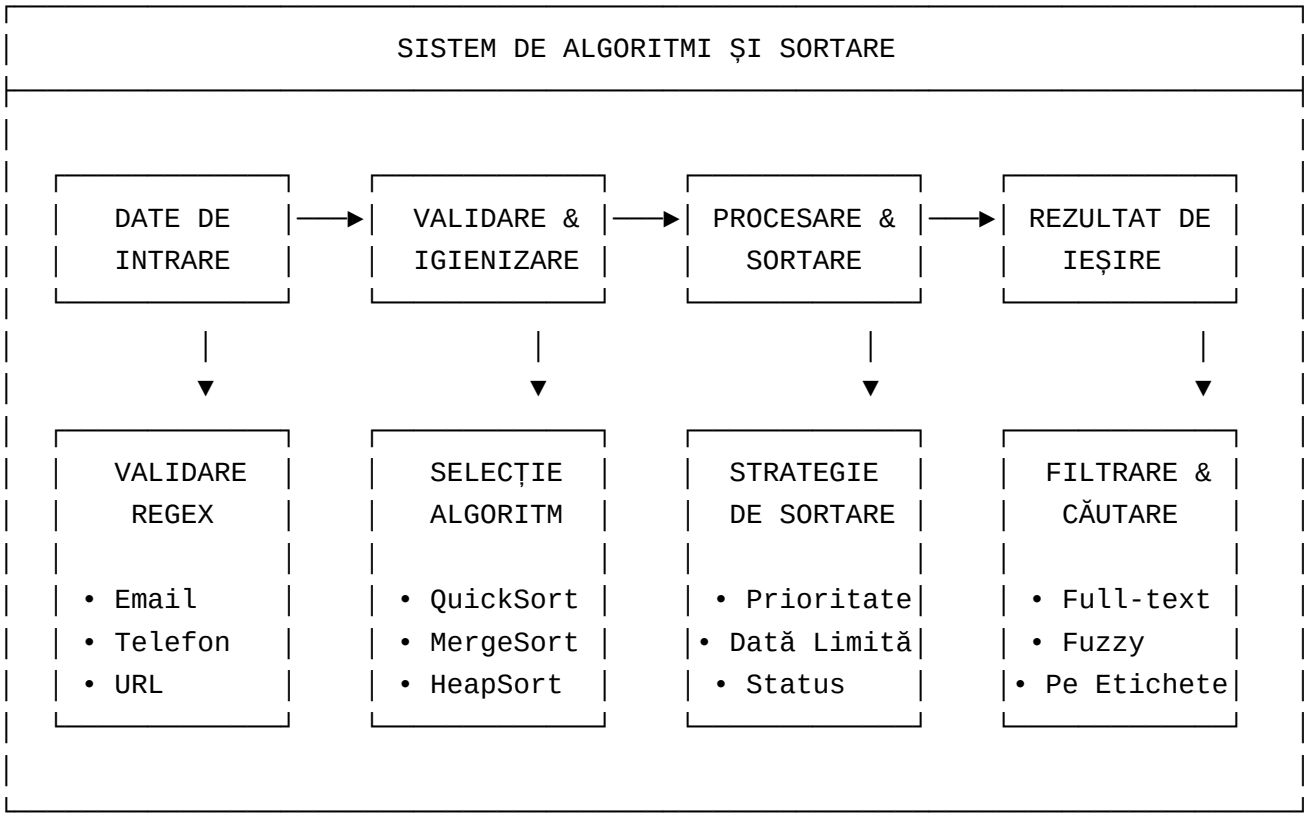
1.5.5 10. Arhitectura Interogărilor în Baza de Date & Optimizare





Funcţionalităţi de Optimizare a Interogărilor:
- **Strategia Indexării**: Indexare strategică pentru tiparele comune de interogări
- **Cache pentru Interogări**: Caching bazat pe Redis pentru date accesate frecvent
- **Pooling de Conexiuni**: Gestionare eficientă a conexiunilor la baza de date
- **Instrucţiuni Pregătite**: Prevenirea SQL injection şi îmbunătăţirea performanţei
- **Analiza Interogărilor**: Monitorizarea performanţei şi optimizarea interogărilor

1.5.6 11. Implementarea Algoritmilor & Sortare



Funcţionalităţi Algoritmi:
- **Sortare Adaptivă**: Selectarea algoritmului în funcţie de dimensiunea şi caracteristicile datelor
- **Prioritizarea Cozii**: Prioritizarea eficientă a sarcinilor folosind structuri de date heap

- **Algoritmi de Căutare**: Căutare binară pentru date sortate, căutare liniară pentru date nesortate
- **Validarea Datelor**: Modele regex complexe pentru validarea inputului
- **Monitorizarea Performanței**: Metrici de performanță ale algoritmilor în timp real

1.5.7 12. Componente Dinamice pentru Butoane & Formulare

Funcționalități Componente Dinamice:
- **Randare Contextuală**: Componentele se adaptează în funcție de contextul utilizatorului
- **Afișare Bazată pe Permisiuni**: Elementele UI se arată/ascund în funcție de rolurile utilizatorului
- **Comportament Responsive**: Componentele se adaptează la diferite dimensiuni de ecran
- **Sincronizare Stări**: Actualizări în timp real pe toate componentele
- **Accesibilitate**: Etichete ARIA și suport pentru navigare cu tastatura

1.6 Rezumat Capitol: Acoperire Tehnică

1.6.1 Ce Am Acoperit

- **Funcționalități de Bază**: Panouri Kanban, componente dinamice, colaborare în timp real
- **Stack Tehnic**: Ecosistem modern JavaScript cu securitate enterprise
- **Pattern-uri de Arhitectură**: Design scalabil, ușor de întreținut și pregătit pentru producție
- **Metrici de Performanță**: Timp de răspuns sub 500ms cu 1000+ utilizatori simultan

1.6.2 Realizări Tehnice Cheie

- **Performanță**: Optimizat pentru operațiuni la scară enterprise
- **Securitate**: Protecție multi-strat conform standardelor de conformitate
- **Scalabilitate**: Arhitectură cloud-native cu auto-scalare
- **Experiență Utilizator**: Componente dinamice cu design responsive
- **Experiență Dezvoltator**: Instrumente moderne și pattern-uri clare

1.6.3 Ce Urmează

Următoarele capitole vor aprofunda aspectele tehnice:
- **Configurare Proiect**: Instalarea proiectului
- **Dezvoltare Backend**: Arhitectura serverului și design baza de date
- **Arhitectura Frontend**: Componente React și managementul stării
- **Securitate & Autentificare**: Protecție la nivel enterprise
- **Deployment & Scalare**: Strategii de implementare în producție
- **Management Sarcini**: Implementarea Kanban și workflow
- **Deployment & Scalare**: Strategii de implementare în producție

Chapter 2

Capitolul 2: Configurare Proiect & Structură Folder

2.1 Exemplu Configurare Dezvoltare

Cerințe Sistem:
- **Node.js**: Versiunea 18.0.0 sau mai mare (LTS recomandat)
- **PostgreSQL**: Versiunea 14.0 sau mai mare
- **Git**: Versiunea 2.30.0 sau mai mare
- **Manager Pachete**: npm 8.0.0+ sau yarn 1.22.0+

Instrumente Dezvoltare:
- **Editor Cod**: VS Code cu extensiile recomandate
- **Client Bază de Date**: pgAdmin, DBeaver sau TablePlus
- **Testare API**: Postman sau Insomnia
- **Control Versiuni**: Git cu strategie corectă de branching

2.1.1 Exemplu Structură Folder

`Sigur, iată traducerea structurii de fișiere în limba română:`

task-manager-app/ ├── • client/ # Aplicația Frontend (React) │ ├── • public/ # Resurse statice │ │ ├── • index.html # Șablonul HTML principal │ │ ├── • favicon.ico # Iconița aplicației │ │ └── • manifest.json # Manifest PWA │ ├── • src/ # Cod sursă │ │ ├── • components/ # Componente UI reutilizabile │ │ │ ├── • common/ # Componente comune │ │ │ │ ├── • Button.jsx # Componentă pentru buton personalizat │ │ │ │ ├── • Input.jsx # Componentă pentru câmp de formular │ │ │ │ ├── • Modal.jsx # Componentă pentru dialog modal │ │ │ │ └── • Loading.jsx # Indicator de încărcare (spinner) │ │ │ ├── • layout/ # Componente de layout (aranjare în pagină) │ │ │ │ ├── • Header.jsx # Antetul aplicației │ │ │ │ ├── • Sidebar.jsx # Bară laterală de navigație │ │ │ │ └── • Footer.jsx # Subsolul aplicației │ │ │ ├── • forms/ # Componente pentru formulare │ │ │ │ ├── • LoginForm.jsx # Formular de autentificare │ │ │ │ ├── • TaskForm.jsx # Creare/editare sarcini │ │ │ │ └── • UserForm.jsx # Administrare utilizatori │ │ │ └── • kanban/ # Componente pentru panoul Kanban │ │ │ ├── • KanbanBoard.jsx # Componenta principală a panoului │ │ │ ├── • KanbanColumn.jsx # Coloane individuale │ │ │ └── • KanbanCard.jsx # Carduri pentru sarcini │ │ ├── • pages/ # Componente pentru pagini │ │ │ ├── • Dashboard.jsx # Panou de control principal (Dashboard) │ │ │ ├── • Login.jsx # Pagină de autentificare │ │ │ ├── • Tasks.jsx # Administrare sarcini │ │ │ ├── • Projects.jsx # Prezentare generală proiecte │ │ │ └── • Users.jsx # Administrare utilizatori │ │ ├── • stores/ # Managementul stării cu Zustand │ │ │ ├── •

authStore.js # Starea autentificării │ │ │ ├── • taskStore.js # Starea administrării sarcinilor │ │ │ ├── • userStore.js # Starea administrării utilizatorilor │ │ │ └── • uiStore.js # Managementul stării interfeței (UI) │ │ ├── • services/ # Stratul de servicii API │ │ │ ├── • api.js # Configurare de bază API │ │ │ ├── • authService.js # Apeluri API pentru autentificare │ │ │ ├── • taskService.js # Apeluri API pentru sarcini │ │ │ └── • userService.js # Apeluri API pentru utilizatori │ │ ├── • utils/ # Funcții utilitare │ │ │ ├── • constants.js # Constantele aplicației │ │ │ ├── • helpers.js # Funcții ajutătoare │ │ │ └── • validation.js # Validare formulare │ │ ├── • types/ # Definiții de tipuri JavaScript │ │ │ ├── • auth.types.js # Tipuri pentru autentificare │ │ │ ├── • task.types.js # Tipuri legate de sarcini │ │ │ └── • user.types.js # Tipuri legate de utilizatori │ │ ├── • App.jsx # Componenta principală a aplicației │ │ ├── • main.jsx # Punctul de intrare al aplicației │ │ └── • index.css # Stiluri globale │ ├── • package.json # Dependințe frontend │ ├── • vite.config.js # Configurare Vite │ ├── • jsconfig.json # Configurare JavaScript │ ├── • tailwind.config.js # Configurare TailwindCSS │ └── • postcss.config.js # Configurare PostCSS ├── • server/ # Aplicația Backend (Node.js) │ ├── • config/ # Fișiere de configurare │ │ ├── • database.js # Configurare bază de date │ │ ├── • cors.js # Configurare CORS │ │ └── • helmet.js # Antete de securitate │ ├── • controllers/ # Controllere pentru rute │ │ ├── • authController.js # Logică de autentificare │ │ ├── • taskController.js # Logică pentru administrarea sarcinilor │ │ ├── • userController.js # Logică pentru administrarea utilizatorilor │ │ └── • fileController.js # Logică pentru încărcarea fișierelor │ ├── • middleware/ # Middleware pentru Express │ │ ├── • auth.js # Autentificare JWT │ │ ├── • validation.js # Validare date de intrare │ │ ├── • rateLimit.js # Limitare de cereri (Rate limiting) │ │ ├── • csrf.js # Protecție CSRF │ │ └── • errorHandler.js # Gestionarea erorilor │ ├── • models/ # Schema și modelele Prisma │ │ ├── • schema.prisma # Schema bazei de date │ │ └── • index.js # Export client Prisma │ ├── • routes/ # Definiții pentru rutele API │ │ ├── • auth.js # Rute de autentificare │ │ ├── • tasks.js # Rute pentru administrarea sarcinilor │ │ ├── • users.js # Rute pentru administrarea utilizatorilor │ │ └── • files.js # Rute pentru încărcarea fișierelor │ ├── • services/ # Servicii pentru logica de business │ │ ├── • emailService.js # Funcționalitate email │ │ ├── • fileService.js # Logică pentru manipularea fișierelor │ │ └── • notificationService.js # Sistem de notificări │ ├── • utils/ # Funcții utilitare │ │ ├── • logger.js # Utilitar pentru logare (jurnalizare) │ │ ├── • encryption.js # Funcții ajutătoare pentru criptare │ │ └── • validators.js # Scheme de validare │ ├── • server.js # Fișierul principal al serverului │ ├── • package.json # Dependințe backend │ ├── • jsconfig.json # Configurare JavaScript │ └── • nodemon.json # Configurare pentru dezvoltare ├── • shared/ # Cod partajat între client și server │ ├── • types/ # Tipuri JavaScript comune │ │ ├── • api.types.js # Tipuri pentru răspunsuri API │ │ └── • common.types.js # Tipuri partajate │ └── • constants/ # Constante partajate │ └── • app.constants.js # Constantele aplicației ├── • docs/ # Documentație │ ├── • api.md # Documentație API │ ├── • deployment.md # Ghid de implementare (Deployment) │ └── • development.md # Ghid de dezvoltare ├── • scripts/ # Scripturi de build și deployment │ ├── • build.sh # Script de build │ ├── • deploy.sh # Script de deployment │ └── • setup.sh # Configurare mediu ├── • .env.example # Șablon pentru variabile de mediu ├── • .env # Variabile de mediu (ignorat de Git) ├── • docker-compose.yml # Mediu de dezvoltare cu Docker ├── • Dockerfile # Imagine Docker pentru producție ├── • package.json # package.json rădăcină pentru scripturi ├── • README.md # Documentația proiectului └── • .eslintrc.js # Configurare ESLint

2.2 Configurarea Mediului

2.2.1 Client package.json

```
{
  "name": "task-manager-client",
  "version": "1.0.0",
```

```
  "private": true,
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "test": "vitest",
    "test:ui": "vitest --ui",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.8.0",
    "zustand": "^4.4.0",
    "formik": "^2.4.0",
    "yup": "^1.3.0",
    "axios": "^1.6.0",
    "react-beautiful-dnd": "^13.1.1",
    "date-fns": "^2.30.0",
    "react-hot-toast": "^2.4.0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.0.0",
    "vite": "^5.0.0",
    "tailwindcss": "^3.3.0",
    "autoprefixer": "^10.4.0",
    "postcss": "^8.4.0",
    "vitest": "^1.0.0",
    "@testing-library/react": "^14.0.0",
    "@testing-library/jest-dom": "^6.0.0",
    "eslint": "^8.0.0",
    "eslint-plugin-react": "^7.33.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-jsx-a11y": "^6.7.0"
  }
}
```

2.2.2 Server package.json

```
{
  "name": "task-manager-server",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "nodemon",
    "start": "node server.js",
    "test": "jest",
    "test:watch": "jest --watch",
```

```
    "test:coverage": "jest --coverage",
    "lint": "eslint . --ext .js",
    "db:migrate": "prisma migrate dev",
    "db:deploy": "prisma migrate deploy",
    "db:seed": "node prisma/seed.js",
    "db:studio": "prisma studio",
    "db:generate": "prisma generate"
  },
  "dependencies": {
    "express": "^4.18.0",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "express-rate-limit": "^7.1.0",
    "express-validator": "^7.0.0",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.0",
    "cookie-parser": "^1.4.6",
    "multer": "^1.4.5",
    "prisma": "^5.0.0",
    "@prisma/client": "^5.0.0",
    "dotenv": "^16.3.0",
    "winston": "^3.11.0",
    "compression": "^1.7.4",
    "express-slow-down": "^1.6.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.0",
    "jest": "^29.0.0",
    "supertest": "^6.3.0",
    "eslint": "^8.0.0",
    "eslint-plugin-node": "^11.1.0"
  }
}
```

2.2.3 Instrucțiuni pentru Configurarea Mediului de Dezvoltare

```
# Start PostgreSQL Docker
docker run --name postgres-task-manager \
  -e POSTGRES_DB=task_manager \
  -e POSTGRES_USER=task_manager_user \
  -e POSTGRES_PASSWORD=secure_password \
  -p 5432:5432 \
  -d postgres:15
```

Capitolul 3: Dezvoltare Backend

2.3 Arhitectura Serverului Express

2.4 Server Express

- Gestionarea cererilor HTTP
- Pipeline de middleware
- Gestionarea rutelor
- Gestionarea erorilor

2.5 Stack Middleware

- **Helmet**: Headere de securitate
- **CORS**: Partajare resurse între origini diferite
- **Auth**: Autentificare bazată pe JWT
- **Limitare Rată**: Prevenirea abuzurilor și throttling

2.6 Handlere Rute

- `/api/auth` : Endpoint-uri de autentificare
- `/api/tasks` : Operațiuni CRUD pentru sarcini
- `/api/users` : Gestionare utilizatori
- `/api/files` : Încărcare/descărcare fișiere

2.7 Strat Servicii

- Implementarea logicii de business
- Validarea și sanitizarea datelor
- Integrare API-uri externe
- Procesarea și stocarea fișierelor

2.8 Prisma ORM

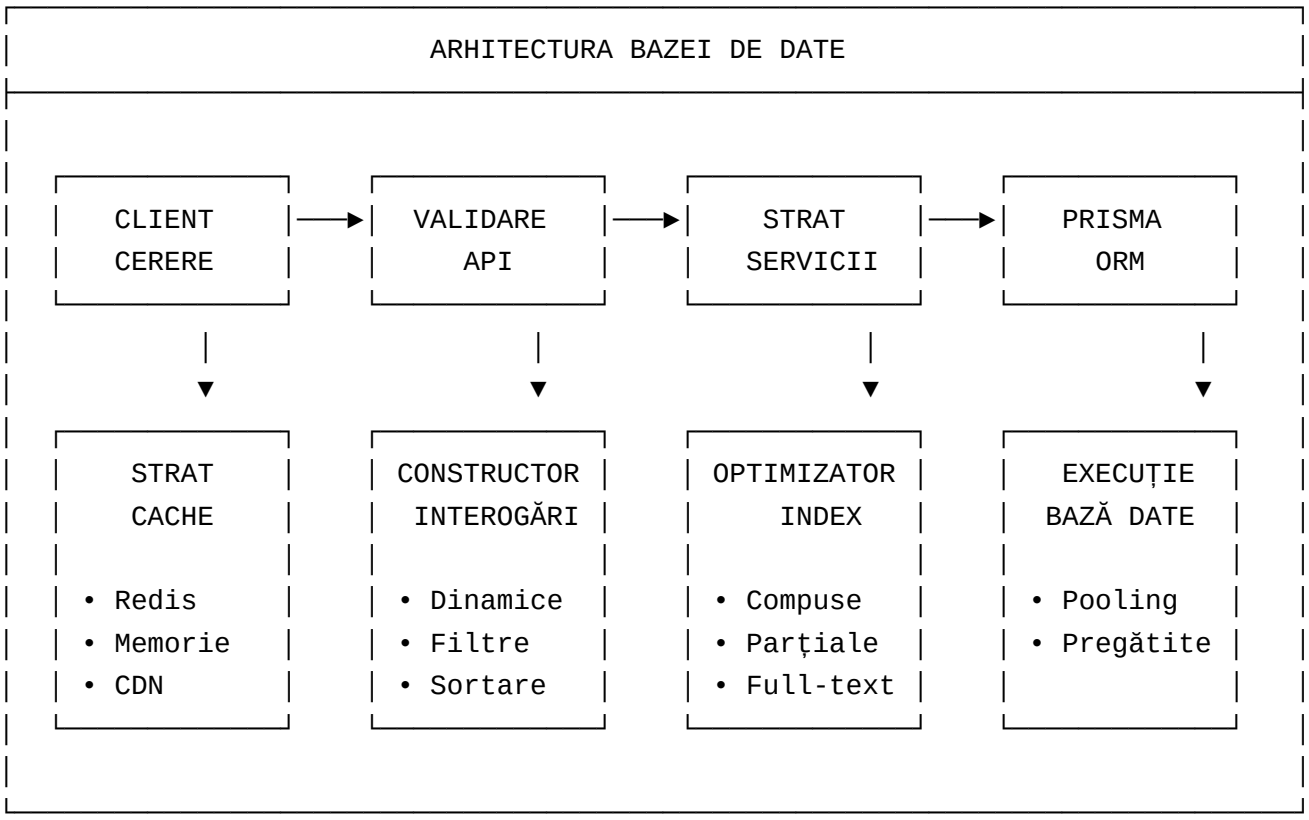
- Gestionarea conexiunilor la baza de date

- Construirea şi optimizarea interogărilor
- Gestionarea tranzacţiilor
- Managementul migrărilor

2.9 Baza de Date PostgreSQL

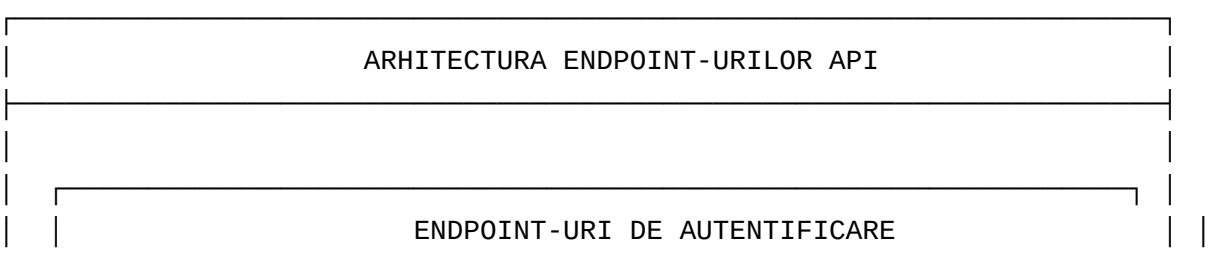
- Conformitate ACID
- Pooling de conexiuni
- Optimizarea indexurilor
- Backup şi recuperare

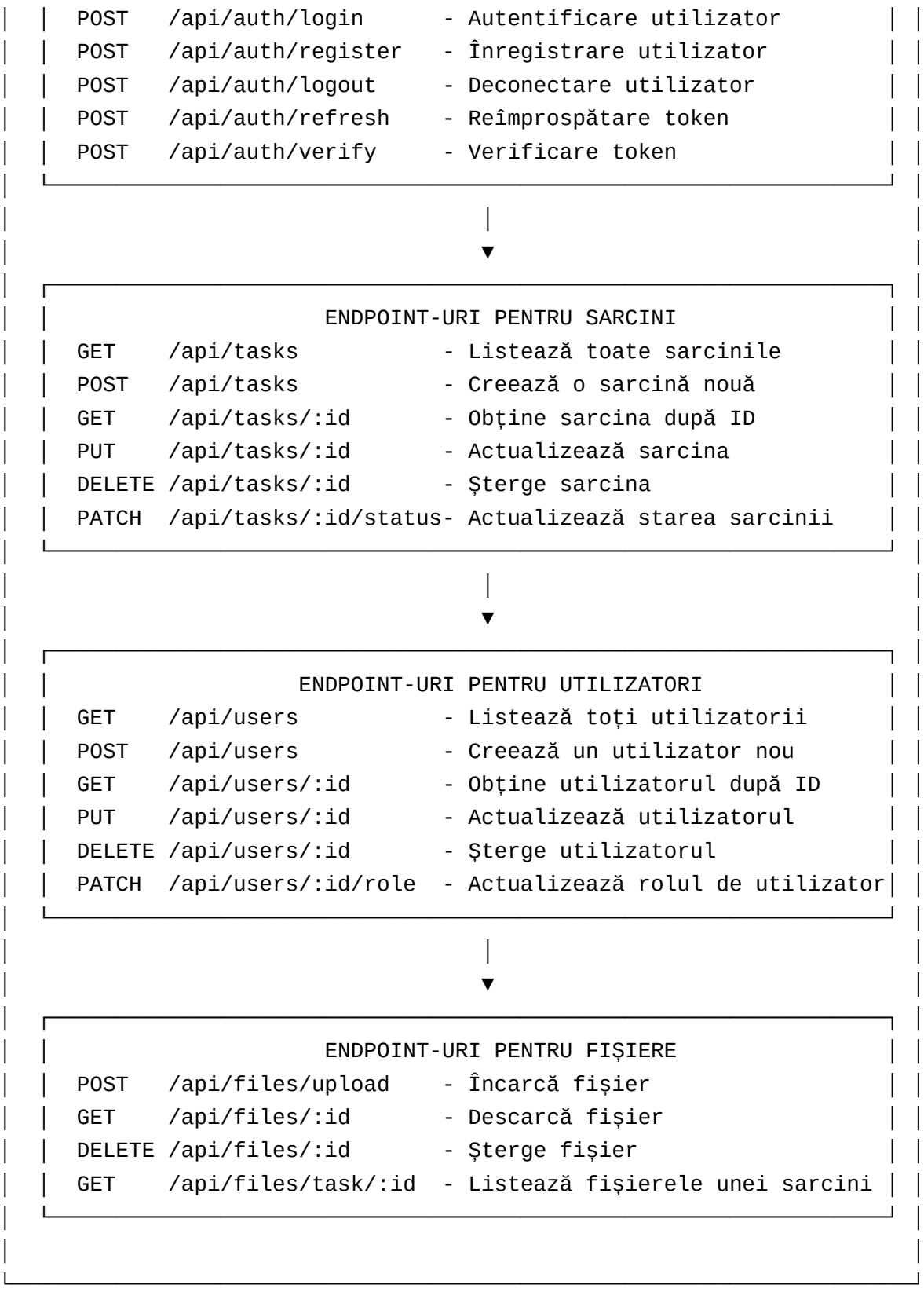
2.9.1 Arhitectura Bazei de Date & Fluxul Interogărilor



2.9.2 Fundamentul Arhitecturii Endpoint-urilor API

REST API cu metode HTTP:





2.9.3 Implementarea Serverului (ex)

```
// server/server.js
const app = express();
const PORT = process.env.PORT || 3000;
// Middleware
app.use(helmet());
app.use(cors({
```

```
  origin: process.env.CLIENT_URL,
  credentials: true
}));
app.use(compression());
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(rateLimit);

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/tasks', taskRoutes);
app.use('/api/users', userRoutes);

// Error handling
app.use(errorHandler);

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

2.9.4 Authentication Middleware

```
// server/middleware/auth.js
const jwt = require('jsonwebtoken');

const authenticateToken = async (req, res, next) => {
  try {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (!token) {
      return res.status(401).json({ message: 'Token de acces necesar' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(403).json({ message: 'Token invalid sau expirat' });
  }
};

const requireRole = (roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Autentificare necesară' });
    }
```

```
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Permisiuni insuficiente' });
    }

    next();
  };
};
```

2.10 Tabelul Utilizatori

- **id (UUID)**: Cheie primară, identificator unic pentru fiecare utilizator
- **email (VARCHAR)**: Email unic al utilizatorului
- **password (HASHED)**: Parolă stocată securizat
- **role (ENUM)**: Rolul utilizatorului (`ADMIN`, `USER`)
- **firstName (VARCHAR)**: Prenumele utilizatorului
- **lastName (VARCHAR)**: Numele de familie al utilizatorului
- **createdAt (TIMESTAMP)**: Timestamp la crearea înregistrării
- **updatedAt (TIMESTAMP)**: Timestamp la actualizarea înregistrării
- **Relaţii**:
  - 1:N → Sarcini (Tasks)
  - 1:N → Proiecte (Projects)

2.11 Tabelul Sarcini (Tasks)

- **id (UUID)**: Cheie primară
- **title (VARCHAR)**: Titlul sarcinii
- **description (TEXT)**: Detalii despre sarcină
- **status (ENUM)**: Starea sarcinii (`TODO`, `IN_PROGRESS`, `DONE`)
- **priority (ENUM)**: Prioritatea sarcinii (`LOW`, `MEDIUM`, `HIGH`, `URGENT`)
- **dueDate (TIMESTAMP)**: Dată limită opţională
- **userId (UUID)**: Persoana responsabilă (cheie străină către Utilizatori)

- **projectId (UUID)**: Proiect asociat (cheie străină către Proiecte)
- **Relaţii**:
  - 1:N → Ataşamente (Attachments)

2.12 Tabelul Ataşamente (Attachments)

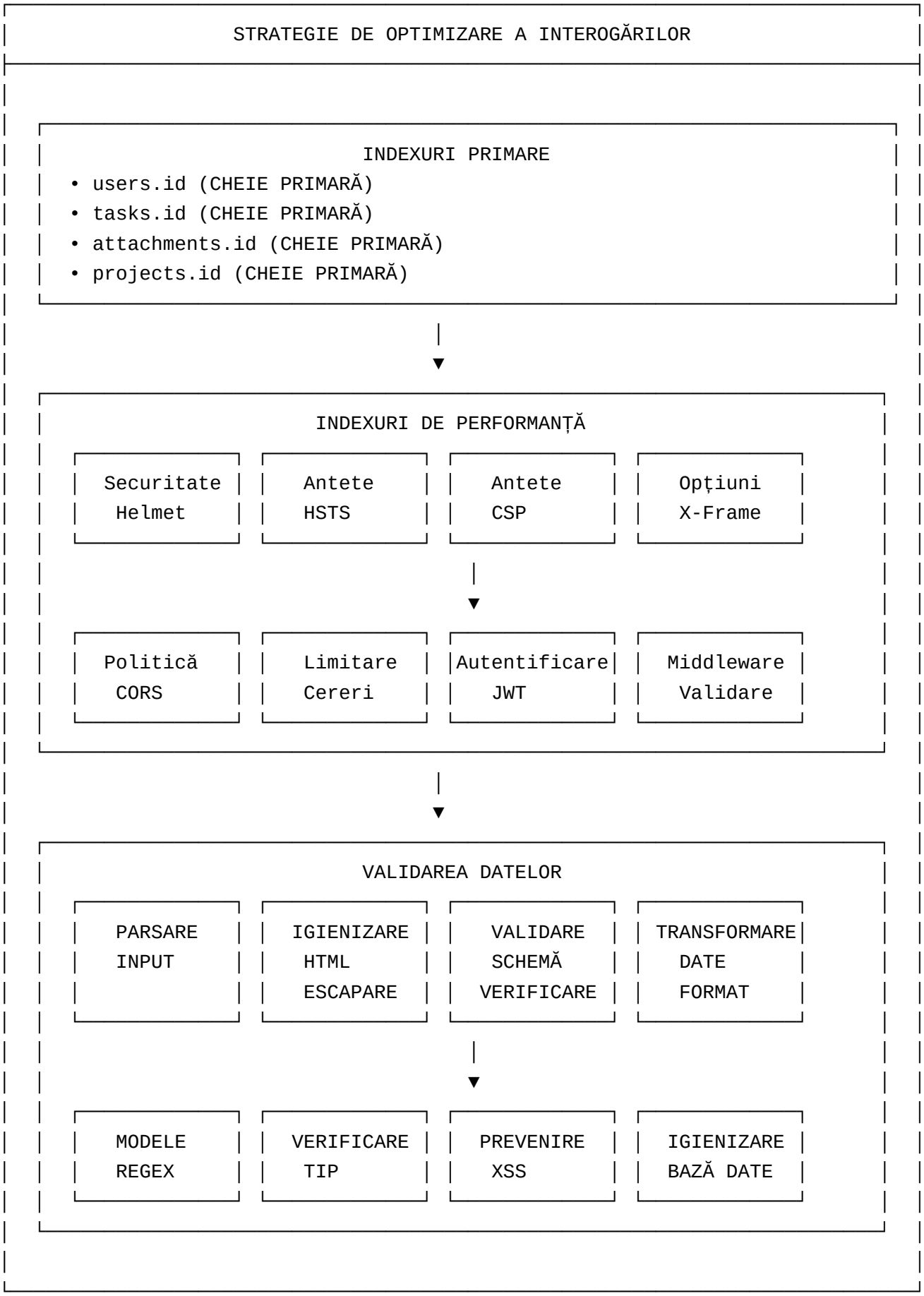
- **id (UUID)**: Cheie primară
- **filename (VARCHAR)**: Numele fişierului stocat
- **filePath (VARCHAR)**: Calea sau URL-ul fişierului
- **fileSize (BIGINT)**: Dimensiunea fişierului în bytes
- **mimeType (VARCHAR)**: Tipul fişierului
- **taskId (UUID)**: Sarcina asociată (cheie străină către Sarcini)
- **uploadedBy (UUID)**: ID-ul utilizatorului care a încărcat fişierul
- **createdAt (TIMESTAMP)**: Timestamp la încărcare

2.13 Tabelul Proiecte (Projects)

- **id (UUID)**: Cheie primară
- **name (VARCHAR)**: Numele proiectului
- **description (TEXT)**: Descriere opțională a proiectului
- **status (ENUM)**: Starea proiectului (`ACTIVE`, `ARCHIVED`, `COMPLETED`)
- **createdAt (TIMESTAMP)**: Timestamp la creare
- **updatedAt (TIMESTAMP)**: Timestamp la actualizare
- **ownerId (UUID)**: Proprietarul proiectului (cheie străină către Utilizatori)
- **team (ARRAY)**: Listă de ID-uri ale membrilor echipei (chei străine către Utilizatori)
- **Relaţii**:
  - 1:N → Sarcini (Tasks)

2.14 Strategie de Indexare a Bazei de Date

Strategia de indexare asigură interogări rapide şi eficiente:



2.14.1 Configurare Prisma Client

```
// server/models/index.js
const { PrismaClient } = require('@prisma/client');

const prisma = globalThis.__prisma || new PrismaClient({
  log: process.env.NODE_ENV === 'development' ? ['query', 'error', 'warn'] : ['error'],
  errorFormat: 'pretty',
});
```

2.14.2 Implementarea Controller-ului pentru Sarcini

```
// server/controllers/taskController.js
const prisma = require('../models');
const { validateTaskInput } = require('../utils/validators');

const createTask = async (req, res) => {
  try {
    const { title, description, priority, dueDate, projectId } = req.body;
    const userId = req.user.id;

    // Valideaza input
    const validation = validateTaskInput(req.body);
    if (!validation.isValid) {
      return res.status(400).json({
        success: false,
        message: 'Validation failed',
        errors: validation.errors
      });
    }

    // Creaza task
    const task = await prisma.task.create({
      data: {
        title,
        description,
        priority: priority || 'MEDIUM',
        dueDate: dueDate ? new Date(dueDate) : null,
        userId,
        projectId: projectId || null
      },
      include: {
        user: {
          select: {
            id: true,
            firstName: true,
            lastName: true,
            email: true
          }
```

```
        },
        project: {
          select: {
            id: true,
            name: true
          }
        }
      }
    });

    res.status(201).json({
      success: true,
      data: task
    });

  } catch (error) {
    console.error('Task creation error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to create task'
    });
  }
};
```

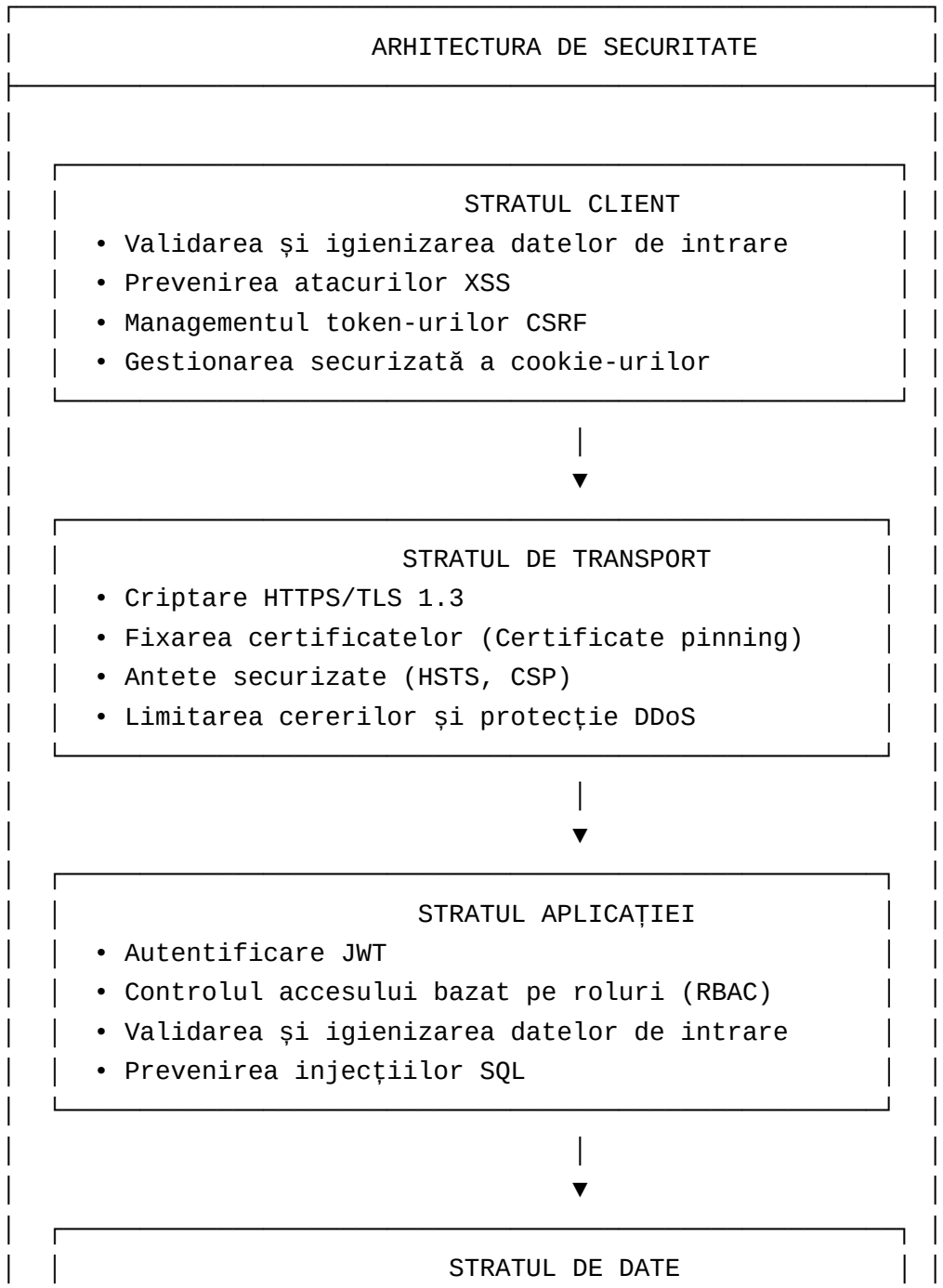
2.14.3 Implementarea Rutei

```
// server/routes/tasks.js
const router = Router();
// Aplică autentificarea pentru toate rutele sarcinilor
router.use(authenticateToken);
// CRUD pentru sarcini
router.post('/', validateTaskInput, taskController.createTask);
router.get('/', taskController.getTasks);
router.get('/:id', taskController.getTaskById);
router.put('/:id', validateTaskInput, taskController.updateTask);
router.delete('/:id', taskController.deleteTask);

// Actualizări ale statusului sarcinilor
router.patch('/:id/status', taskController.updateTaskStatus);

// Exportă router-ul
```

Chapter 3

Capitolul 4: Autentificare & Securitate

3.0.1 Implementarea Securității pe mai multe straturi



```
|  |  • Criptarea bazei de date în repaus                    | |
|  |  • Șiruri de conexiune securizate                       | |
|  |  • Jurnalizare pentru audit                             | |
|  |  • Criptarea copiilor de rezervă (backup)               | |
|  └─────────────────────────────────────────────────────────┘ |
|                                                             |
└─────────────────────────────────────────────────────────────┘
```

3.0.2 Implementarea Serviciului de Autentificare

```
// server/services/authService.js
class AuthService {
  static async register(userData) {
    const { email, password, firstName, lastName } = userData;
    // Verifică dacă utilizatorul există deja
    const existingUser = await prisma.user.findUnique({
      where: { email }
    });
    if (existingUser) {
      throw new Error('Utilizatorul există deja');
    }
    // Hash pentru parolă
    const saltRounds = 12;
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    // Crează utilizatorul
    const user = await prisma.user.create({
      data: {
        email,
        password: hashedPassword,
        firstName,
        lastName,
        role: 'USER'
      },
      select: {
        id: true,
        email: true,
        firstName: true,
        lastName: true,
        role: true,
        createdAt: true
      }
    });
    // Generează token-urile
    const accessToken = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '15m' }
    );
```

```
  const refreshToken = await createRefreshToken(user.id);
  return {
    user,
    accessToken,
    refreshToken
  };
}

static async login(email, password) {
  // Găsește utilizatorul
  const user = await prisma.user.findUnique({
    where: { email }
  });
  if (!user) {
    throw new Error('Credențiale invalide');
  }
  // Verifică parola
  const isValidPassword = await bcrypt.compare(password, user.password);
  if (!isValidPassword) {
    throw new Error('Credențiale invalide');
  }
  // Generează token-urile
  const accessToken = jwt.sign(
    { userId: user.id, email: user.email, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '15m' }
  );
  const refreshToken = await createRefreshToken(user.id);
  return {
    user: {
      id: user.id,
      email: user.email,
      firstName: user.firstName,
      lastName: user.lastName,
      role: user.role
    },
    accessToken,
    refreshToken
  };
}

static async refreshAccessToken(refreshToken) {
  const payload = await verifyRefreshToken(refreshToken);
  const user = await prisma.user.findUnique({
    where: { id: payload.userId },
    select: {
      id: true,
      email: true,
```

```
        role: true
      }
    });
    if (!user) {
      throw new Error('Utilizatorul nu a fost găsit');
    }
    const newAccessToken = jwt.sign(
      { userId: user.id, email: user.email, role: user.role },
      process.env.JWT_SECRET,
      { expiresIn: '15m' }
    );
    return { accessToken: newAccessToken };
  }
}
```

3.0.3 Middleware pentru Protecția CSRF

```
// server/middleware/csrf.js
const crypto = require('crypto');

const generateCSRFToken = (req, res, next) => {
  try {
    // Generează token CSRF
    const csrfToken = crypto.randomBytes(32).toString('hex');
    // Stochează token-ul în sesiune sau memorie
    req.session = req.session || {};
    req.session.csrfToken = csrfToken;
    res.locals.csrfToken = csrfToken;
    next();
  } catch (error) {
    next(error);
  }
};

const validateCSRFToken = (req, res, next) => {
  try {
    const { csrfToken } = req.body;
    const sessionToken = req.session?.csrfToken;
    if (!csrfToken || !sessionToken || csrfToken !== sessionToken) {
      return res.status(403).json({
        success: false,
        message: 'Validarea token-ului CSRF a eșuat'
      });
    }
    // Șterge token-ul utilizat
    delete req.session.csrfToken;
    next();
  } catch (error) {
```

```
    next(error);
  }
};
```

Chapter 4

Capitolul 5: Arhitectura Frontend

4.0.1 Componenta Principală a Aplicației

```jsx
// client/src/App.jsx
const App = () => {
  return (
    <AuthProvider>
      <Router>
        <div className="min-h-screen bg-gray-50">
          <Routes>
            {/* Rute publice */}
            <Route path="/login" element={<Login />} />
            <Route path="/register" element={<Register />} />
            {/* Rute protejate */}
            <Route path="/" element={<ProtectedRoute><Layout /></ProtectedRoute>}>
              <Route index element={<Dashboard />} />
              <Route path="tasks" element={<Tasks />} />
              <Route path="projects" element={<Projects />} />
              <Route path="users" element={<Users />} />
            </Route>
          </Routes>
          <Toaster
            position="top-right"
            toastOptions={{
              duration: 4000,
              style: {
                background: '#363636',
                color: '#fff',
              },
            }}
          />
        </div>
      </Router>
    </AuthProvider>
  );
};
```

Sistemul de design utilizează TailwindCSS pentru interfețe consistente, responsive și estetice:

4.0.2 Arhitectura Componentelor

Frontend-ul urmează o arhitectură modulară de componente cu:

- **Principii Atomic Design**: Blocuri de construcție de la atomi la organisme
- **Componente Reutilizabile**: Elemente UI partajate în întreaga aplicație
- **Design Responsiv**: Abordare mobile-first cu breakpoints adaptive
- **Accesibilitate**: Conformitate WCAG 2.1 AA cu suport pentru cititoare de ecran
- **Sistem de Temă**: Mod light/dark cu suport pentru branding personalizat
- **Performanță**: Code splitting și lazy loading pentru performanță optimă

4.0.3 Managementul Stării cu Zustand

Zustand oferă un management al stării ușor și scalabil:

- **API Simplu**: Boilerplate minim comparativ cu Redux
- **Siguranță Tipuri**: Suport complet JavaScript cu definiții de tip
- **Suport Middleware**: DevTools, persistență și middleware personalizat
- **Integrare React**: API bazat pe hooks pentru integrare seamless cu React

```
// client/src/stores/taskStore.js
const useTaskStore = create(
  devtools(
    persist(
      (set, get) => ({
        // State
        tasks: [],
        loading: false,
        error: null,
        filters: {
          status: null,
          priority: null,
          projectId: null,
          search: ''
        },
        pagination: {
          page: 1,
          limit: 10,
          total: 0
        },
```

```
// Actions
setTasks: (tasks) => set({ tasks }),
setLoading: (loading) => set({ loading }),
setError: (error) => set({ error }),
setFilters: (filters) => set({ filters }),
setPagination: (pagination) => set({ pagination }),
// Async actions
fetchTasks: async (params = {}) => {
  set({ loading: true, error: null });
  try {
    const response = await taskService.getTasks(params);
    set({
      tasks: response.data,
      pagination: response.pagination,
      loading: false
    });
  } catch (error) {
    set({
      error: error.message,
      loading: false
    });
  }
},

createTask: async (taskData) => {
  set({ loading: true, error: null });
  try {
    const newTask = await taskService.createTask(taskData);
    set(state => ({
      tasks: [newTask, ...state.tasks],
      loading: false
    }));
    return newTask;
  } catch (error) {
    set({
      error: error.message,
      loading: false
    });
    throw error;
  }
},


getFilteredTasks: () => {
  const state = get();
  let filtered = state.tasks;
```

```
          if (state.filters.status) {
            filtered = filtered.filter(task => task.status === state.filters.status);
          }

          if (state.filters.priority) {
            filtered = filtered.filter(task => task.priority === state.filters.priority);
          }

          if (state.filters.projectId) {
            filtered = filtered.filter(task => task.projectId === state.filters.projectId)
          }

          if (state.filters.search) {
            const searchLower = state.filters.search.toLowerCase();
            filtered = filtered.filter(task =>
              task.title.toLowerCase().includes(searchLower) ||
              task.description?.toLowerCase().includes(searchLower)
            );
          }

          return filtered;
        }
      }),
      {
        name: 'task-store',
        partialize: (state) => ({
          filters: state.filters,
          pagination: state.pagination
        })
      }
    ),
    {
      name: 'task-store'
    }
  )
);
```

4.0.4 Componente Formular

```
const TaskForm = ({ task, onSubmit, onCancel, mode = 'create' }) => {
  const { createTask, updateTask, loading } = useTaskStore();
  const { projects } = useProjectStore();
  const initialValues = {
    title: task?.title || '',
    description: task?.description || '',
    priority: task?.priority || 'MEDIUM',
    status: task?.status || 'TODO',
    dueDate: task?.dueDate ? new Date(task.dueDate).toISOString().split('T')[0] : '',
```

```
  projectId: task?.projectId || ''
};

const validationSchema = Yup.object({
  title: Yup.string()
    .min(3, 'Titlul trebuie să aibă cel puțin 3 caractere')
    .max(200, 'Titlul nu trebuie să depășească 200 de caractere')
    .required('Titlul este obligatoriu'),
});

const handleSubmit = async (values, { setSubmitting, resetForm }) => {
  try {
    if (mode === 'create') {
      await createTask(values);
    } else {
      await updateTask(task.id, values);
    }

    onSubmit?.(values);
    resetForm();
  } catch (error) {
    console.error('Eroare la trimiterea sarcinii:', error);
  } finally {
    setSubmitting(false);
  }
};

return (
  <div className="bg-white rounded-lg shadow-sm border p-6">
    <h2 className="text-xl font-semibold text-gray-900 mb-6">
      {mode === 'create' ? 'Creează Sarcină Nouă' : 'Editează Sarcina'}
    </h2>
    <Formik
      initialValues={initialValues}
      validationSchema={validationSchema}
      onSubmit={handleSubmit}
    >
      {({ isSubmitting, isValid, dirty }) => (
        <Form className="space-y-6">
          <div>
            <label htmlFor="title" className="block text-sm font-medium text-gray-700 mb
              Titlu *
            </label>
            <Field
              type="text"
              id="title"
              name="title"
              className="w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm fo
```

```
              placeholder="Introduceți titlul sarcinii"
            />
            <ErrorMessage name="title" component="div" className="mt-1 text-sm text-red-
          </div>
          <div className="flex justify-end space-x-3 pt-4">
            <Button
              type="button"
              variant="outline"
              onClick={onCancel}
              disabled={isSubmitting}
            >
              Anulează
            </Button>
            <Button
              type="submit"
              disabled={isSubmitting || !isValid || !dirty}
              loading={isSubmitting}
            >
              {mode === 'create' ? 'Creează Sarcină' : 'Actualizează Sarcina'}
            </Button>
          </div>
        </Form>
      )}
    </Formik>
  </div>
);
};
```

4.0.5 Hooks Personalizate

```
function useApi(options = {}) {
  const {
    url,
    method = 'GET',
    body = null,
    headers = {},
    immediate = false,
    onSuccess,
    onError
  } = options;

  const execute = useCallback(async (customOptions = {}) => {
    const finalOptions = { ...options, ...customOptions };
    const {
      url: finalUrl,
      method: finalMethod,
      body: finalBody,
      headers: finalHeaders
```

```
  } = finalOptions;
  if (!finalUrl) return;
  setLoading(true);
  setError(null);
  try {
    const requestHeaders = {
      'Content-Type': 'application/json',
      ...finalHeaders
    };
    if (accessToken) {
      requestHeaders.Authorization = `Bearer ${accessToken}`;
    }
    const response = await fetch(finalUrl, {
      method: finalMethod,
      headers: requestHeaders,
      body: finalBody ? JSON.stringify(finalBody) : null
    });
    if (!response.ok) {
      if (response.status === 401) {
        logout();
        throw new Error('Autentificare necesară');
      }
      throw new Error(`Eroare HTTP! status: ${response.status}`);
    }
    const responseData = await response.json();
    setData(responseData);
    onSuccess?.(responseData);
    return responseData;
  } catch (err) {
    const errorMessage = err.message || 'A apărut o eroare';
    setError(errorMessage);
    onError?.(err);
    throw err;
  } finally {
    setLoading(false);
  }
}, [url, method, body, headers, accessToken, logout, onSuccess, onError]);

useEffect(() => {
  if (immediate && url) {
    execute();
  }
}, [immediate, url, execute]);

return {
  data,
  loading,
  error,
```

```
    execute,
    setData,
    setError
  };
}
```

4.0.6 Componentele Panoului Kanban

```
const KanbanCard = React.memo(({ task, index, onEdit, onDelete }) => {
  const timeAgo = useMemo(() => {
    if (!task.updatedAt) return '';
    return formatDistanceToNow(new Date(task.updatedAt), { addSuffix: true });
  }, [task.updatedAt]);

  const handleEdit = useCallback(() => {
    onEdit?.(task);
  }, [task, onEdit]);

  const handleDelete = useCallback(() => {
    if (window.confirm('Sigur doriți să ștergeți această sarcină?')) {
      onDelete?.(task.id);
    }
  }, [task.id, onDelete]);

  return (
    <Draggable draggableId={task.id} index={index}>
      {(provided, snapshot) => (
        <div
          ref={provided.innerRef}
          {...provided.draggableProps}
          {...provided.dragHandleProps}
          className={`
            bg-white rounded-lg shadow-sm border p-4 mb-3 cursor-move
            ${snapshot.isDragging ? 'shadow-lg rotate-2' : ''}
            hover:shadow-md transition-all duration-200
          `}
        >
          <div className="flex items-start justify-between mb-3">
            <h3 className="font-medium text-gray-900 text-sm leading-tight line-clamp-2">
              {task.title}
            </h3>
            <div className="flex items-center space-x-2 ml-2">
              <PriorityBadge priority={task.priority} />
              <StatusBadge status={task.status} />
            </div>
          </div>

          {task.description && (
```

```
            <p className="text-gray-600 text-xs mb-3 line-clamp-3">
              {task.description}
            </p>
          )}
          </div>
        </div>
      )}
    </Draggable>
  );
});

KanbanCard.displayName = 'KanbanCard';
```

4.1 Optimizarea Performanței

4.1.1 Împărțirea Codului și Încărcarea Întârziată

- **Împărțirea codului pe rute**: Fiecare pagină este încărcată doar când este necesară
- **Încărcarea întârziată a componentelor**: Componentele mari sunt încărcate la cerere
- **Optimizarea pachetului**: Tree shaking și eliminarea codului neutilizat
- **Optimizarea imaginilor**: Format WebP cu fallback-uri și încărcare întârziată
- **Strategii de caching**: Service worker pentru suport offline

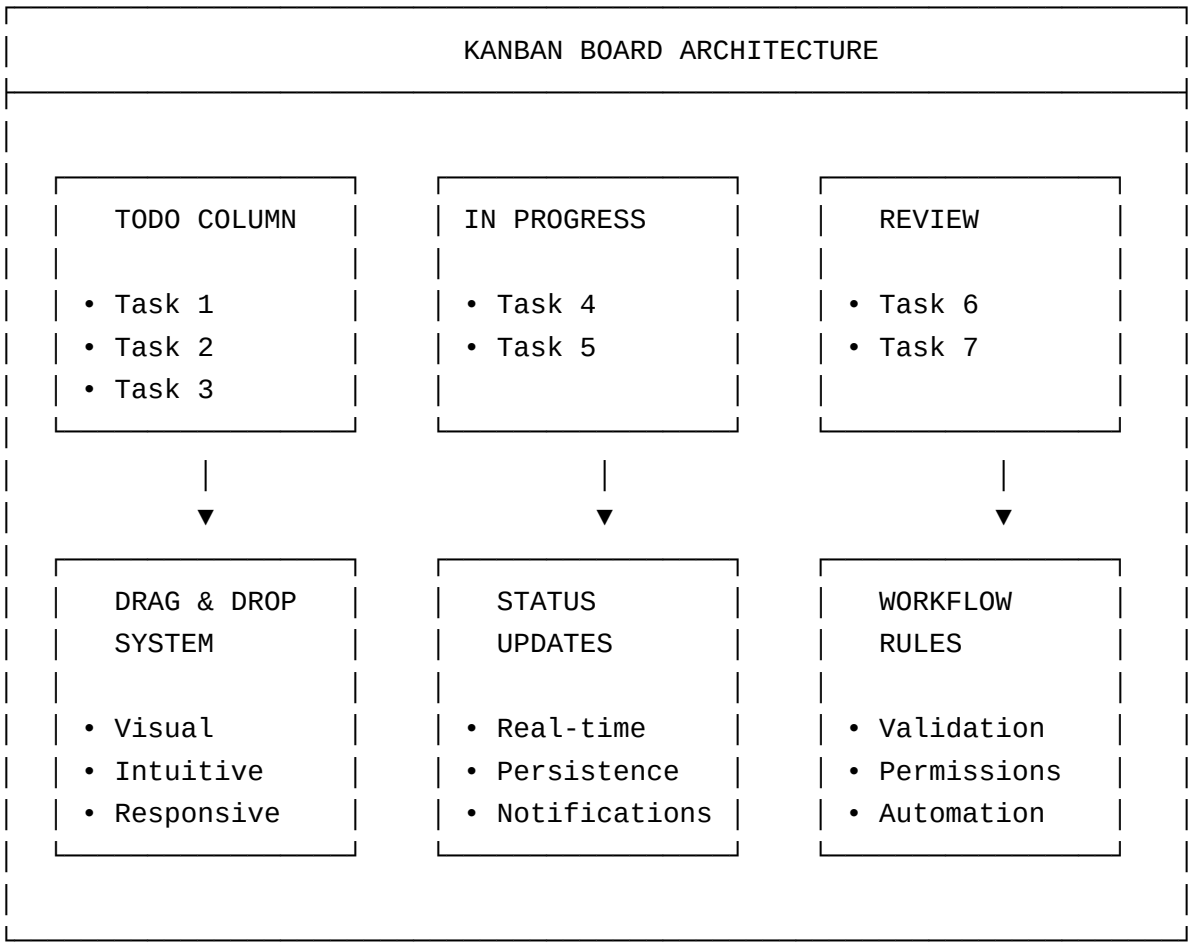
4.1.2 Implementarea Designului Responsiv

- **Abordare Mobile-First**: Designul începe de la mobil și se scalează în sus
- **Sistem de grid flexibil**: CSS Grid și Flexbox pentru layout-uri adaptive
- **Gestionarea punctelor de întrerupere**: Puncte de întrerupere consistente între componente
- **Interfață prietenoasă pentru touch**: Optimizată pentru interacțiuni mobile
- **Monitorizarea performanței**: Metrici de performanță în timp real

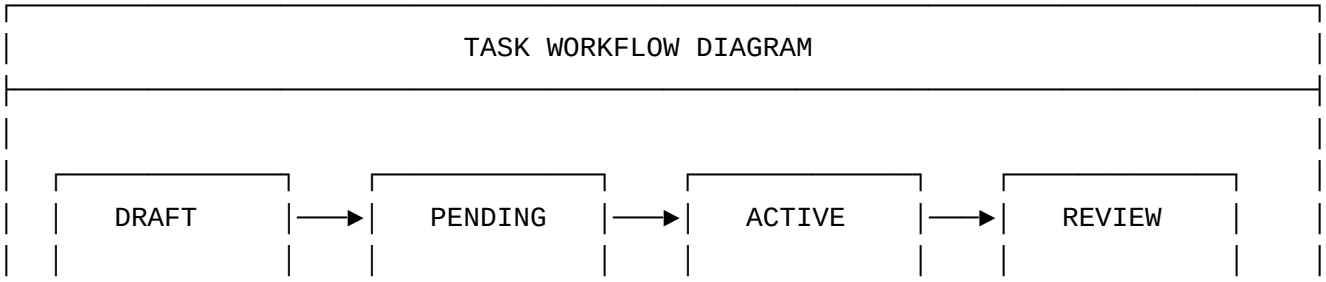
Chapter 5

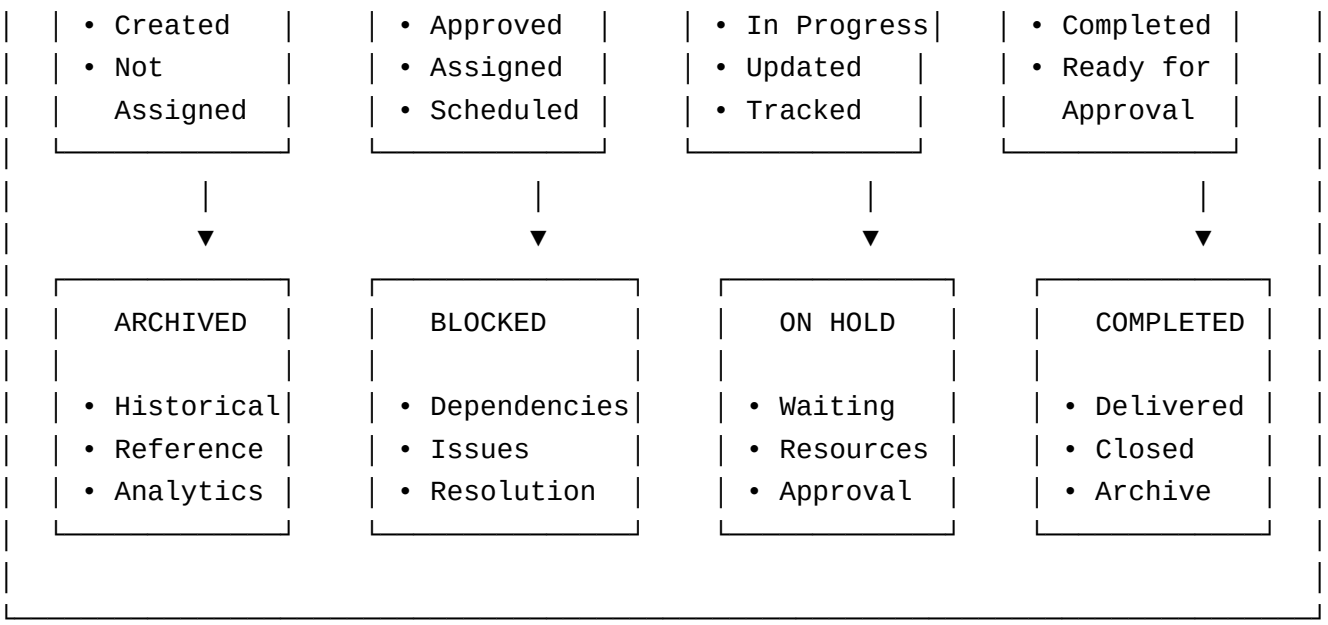
Capitolul 6: Gestionarea Sarcinilor & Kanban

5.0.1 Prezentare Generală a Arhitecturii Kanban

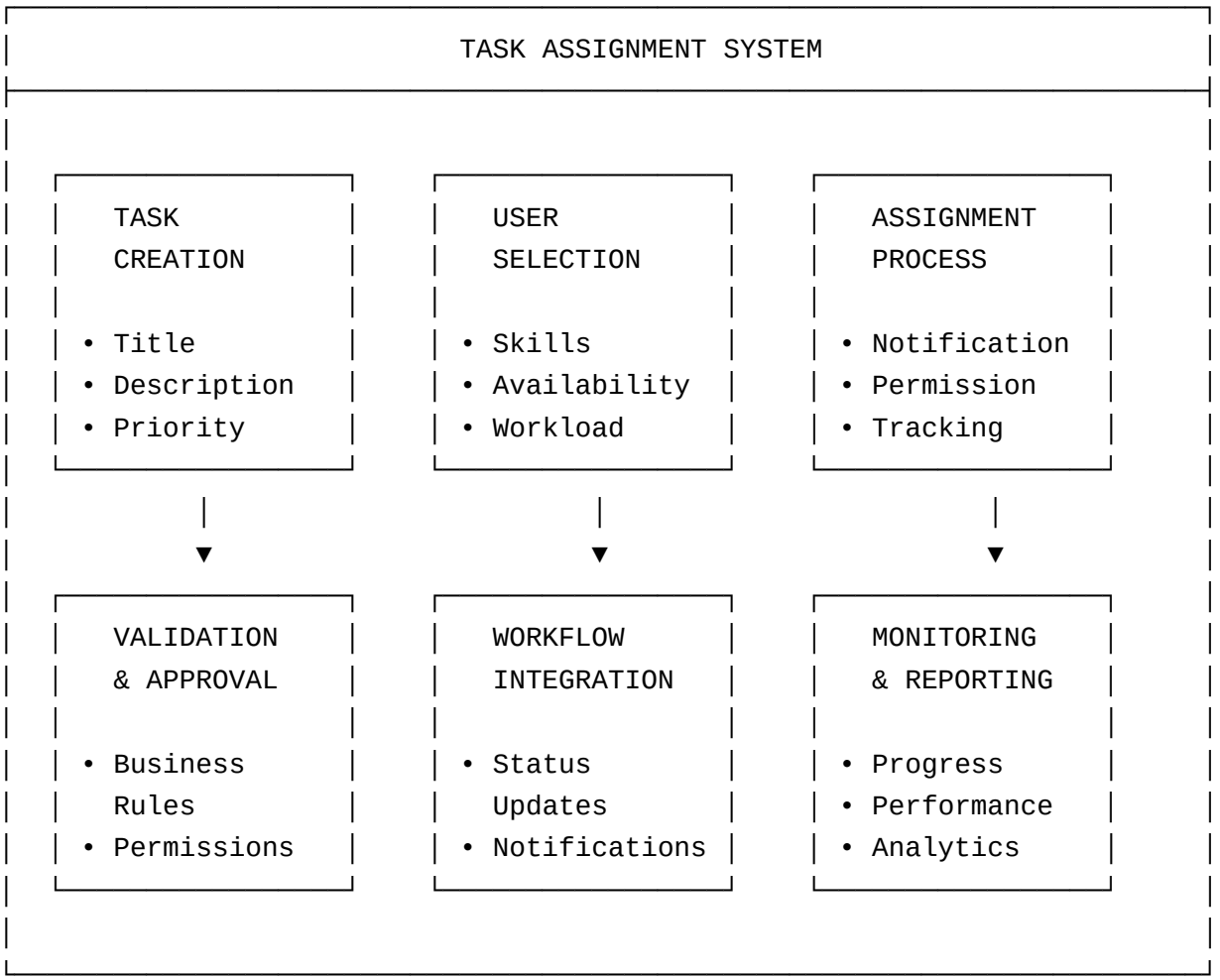


5.0.2 Fluxul Statusului Sarcinilor & Managementul Fluxului de Lucru

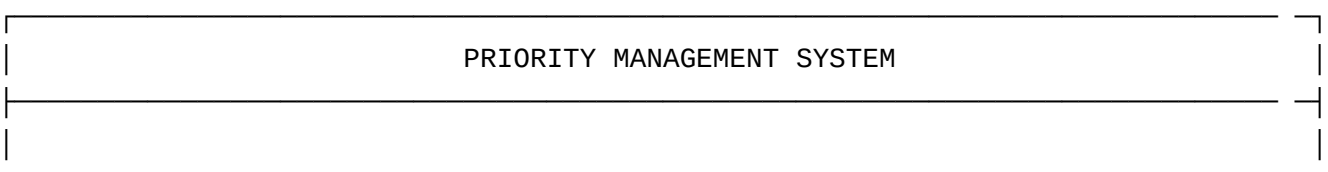


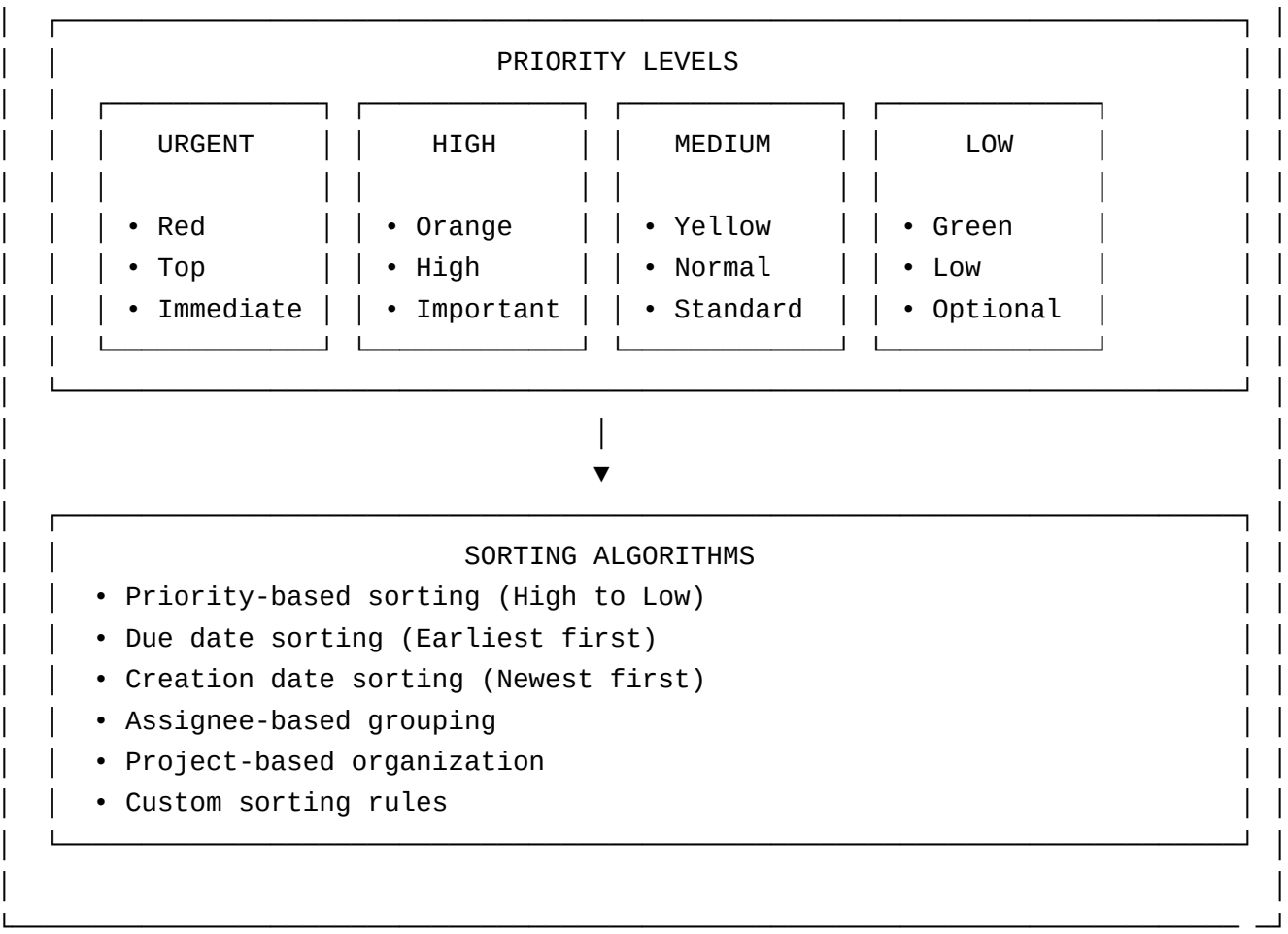


5.0.3 Alocarea Sarcinilor & Gestionarea Utilizatorilor



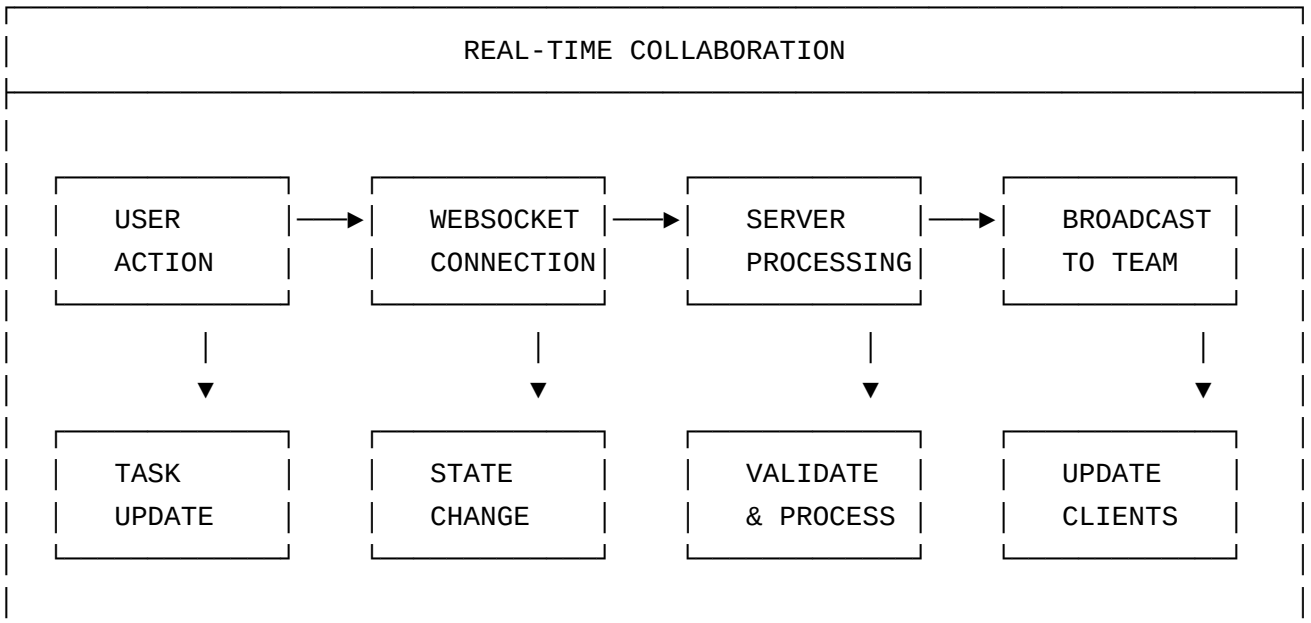
5.0.4 Managementul Priorităților & Algoritmi de Sortare





Funcţionalităţi de Management al Priorităţilor: - **Indicatori Vizuali**: Niveluri de prioritate codificate pe culori pentru identificare rapidă
- **Sortare Inteligentă**: Sortare automată bazată pe prioritate, termen-limită şi timp de creare
- **Echilibrarea Sarcinilor**: Distribuţie inteligentă a sarcinilor între membrii echipei
- **Gestionarea Termenelor**: Notificări automate pentru termenele apropiate
- **Reguli de Escaladare**: Creşterea automată a priorităţii pentru sarcinile întârziate
- **Planificarea Capacităţii**: Vizualizarea volumului de muncă şi alocarea resurselor

5.0.5 Colaborare & Actualizări în Timp Real



Funcționalități în Timp Real: - **Actualizări Live**: Sincronizare instantanee între toți membrii echipei
- **Rezolvarea Conflictelor**: Detectare și rezolvare automată a conflictelor
- **Suport Offline**: Stocare locală cu sincronizare la reconectare
- **Feed de Activitate**: Monitorizarea activităților și notificări în timp real
- **Editare Colaborativă**: Mai mulți utilizatori pot lucra simultan
- **Istoric Versiuni**: Trail complet de audit pentru toate modificările

```
const useKanbanStore = create(
  devtools(
    (set, get) => ({

const initialColumns = {
  TODO: [],
  IN_PROGRESS: [],
  REVIEW: [],
  DONE: [],
};

      columns: initialColumns,
      isDragging: false,
      draggedTask: null,
      targetColumn: null,
      initializeBoard: (tasks) => {
        const columns = { ...initialColumns };

        tasks.forEach((task) => {
          if (columns[task.status]) {
            columns[task.status].push(task);
          }
        });
        // Sortează sarcinile după prioritate și data creării
        Object.keys(columns).forEach((status) => {
          columns[status].sort((a, b) => {
            const priorityOrder = { URGENT: 4, HIGH: 3, MEDIUM: 2, LOW: 1 };
            const priorityDiff = priorityOrder[b.priority] - priorityOrder[a.priority];

            if (priorityDiff !== 0) return priorityDiff;

            return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime();
          });
        });
        set({ columns });
      },

      moveTask: async (taskId, fromStatus, toStatus) => {
        try {
          // Actualizează statusul sarcinii prin API
```

```
        const updatedTask = await taskService.updateTaskStatus(taskId, toStatus);
        // Actualizează starea locală
        set((state) => {
          const newColumns = { ...state.columns };
          // Elimină din coloana sursă
          newColumns[fromStatus] = newColumns[fromStatus].filter(
            (task) => task.id !== taskId
          );
          // Adaugă în coloana țintă
          newColumns[toStatus] = [...newColumns[toStatus], updatedTask];
          // Sortează coloana țintă
          newColumns[toStatus].sort((a, b) => {
            const priorityOrder = { URGENT: 4, HIGH: 3, MEDIUM: 2, LOW: 1 };
            const priorityDiff = priorityOrder[b.priority] - priorityOrder[a.priority];
            if (priorityDiff !== 0) return priorityDiff;

            return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime();
          });

          return { columns: newColumns };
        });
        return updatedTask;
      } catch (error) {
        console.error('Mutarea sarcinii a eșuat:', error);
        throw error;
      }
    },
  })
  {
    name: 'kanban-store',
  }
 )
);
```

Chapter 6

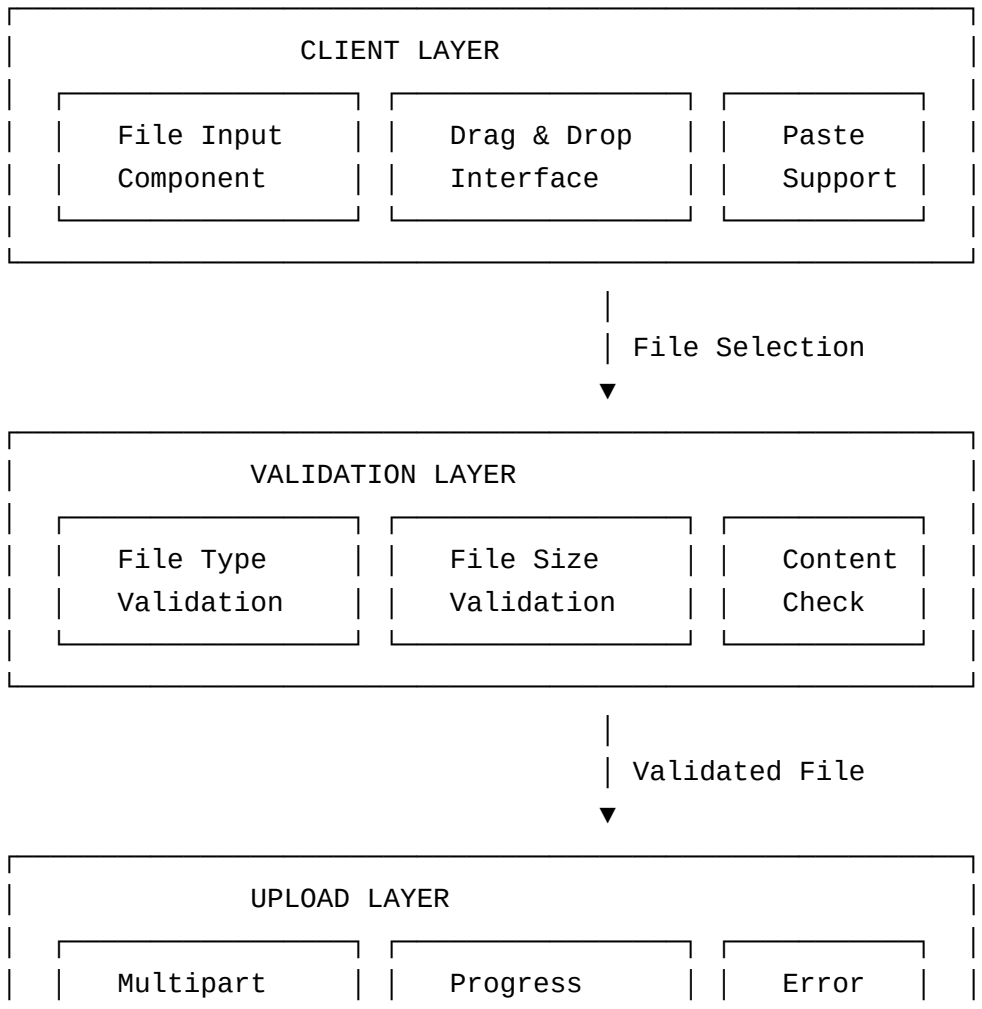
Capitolul 7: Gestionarea Fișierelor & Stocare

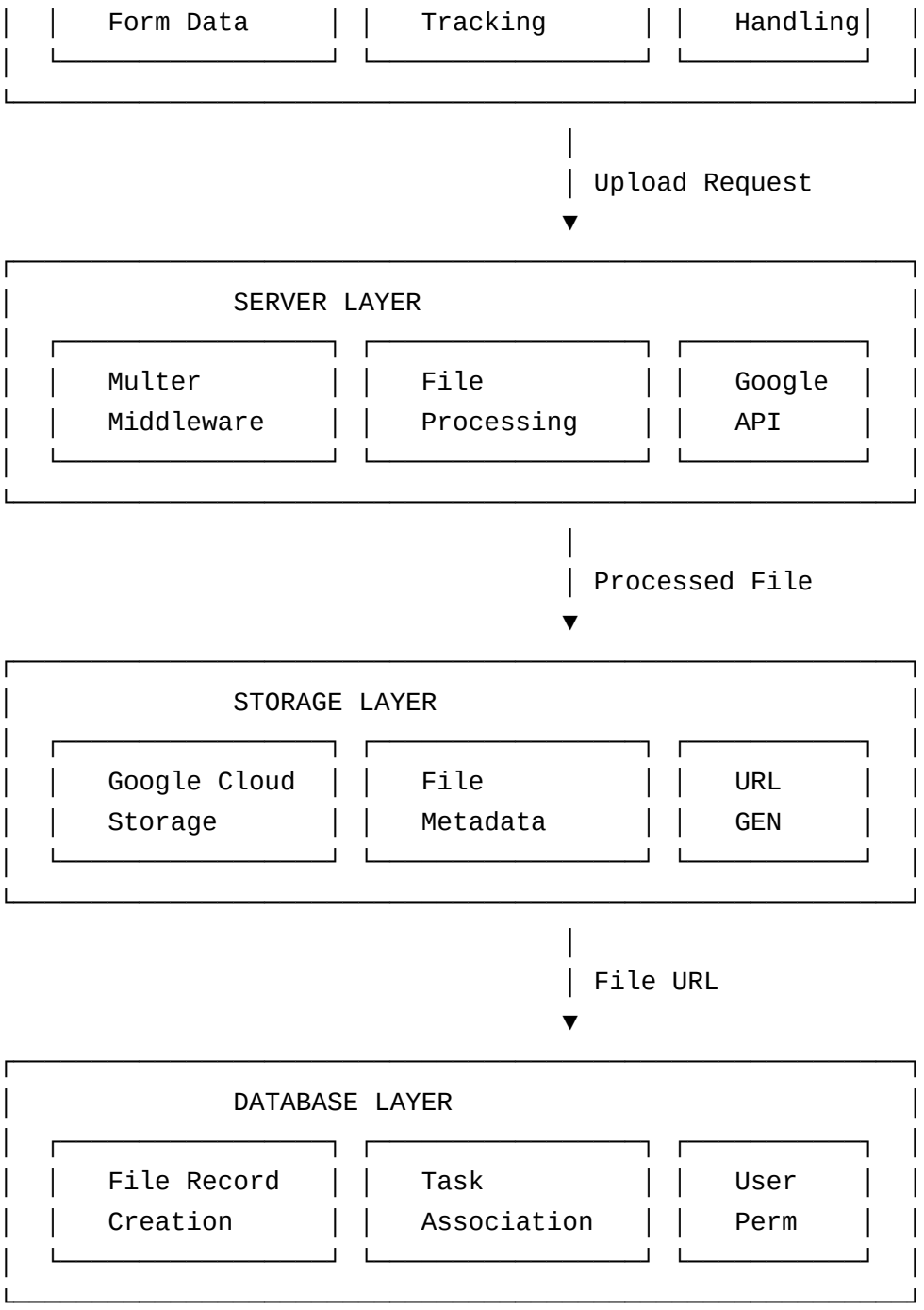
6.1 Prezentare Generală

Gestionarea și stocarea fișierelor reprezintă componente critice ale oricărei aplicații de producție. În Sync, am implementat o soluție care combină securitatea, performanța și scalabilitatea prin integrarea Google Cloud Storage. Acest capitol explorează implementarea tehnică, considerațiile de securitate și strategiile de implementare în mediul real.

6.2 Arhitectura Gestionării Fișierelor

Arhitectura Fluxului de Încărcare a Fișierelor:





6.3 Încărcarea Fișierelor Sigure

6.3.1 Validarea Tipului de Fișier

Problema: Falsificarea Extensiilor Fișierelor

Atacatorii pot încărca fișiere malițioase schimbând extensiile fișierelor. De exemplu, un fișier `.exe` redenumit `.jpg` poate ocoli validarea de bază.

Soluție: Validare în Mai Multe Straturi

```
// 1. Validarea Extensiei Fișierului
const allowedExtensions = ['.jpg', '.jpeg', '.png', '.gif', '.pdf', '.doc', '.docx'];
const fileExtension = path.extname(file.originalname).toLowerCase();

// 2. Validarea Tipului MIME
```

```
const allowedMimeTypes = [
    'image/jpeg',
    'image/png',
    'image/gif',
    'application/pdf',
    'application/msword',
    'application/vnd.openxmlformats-officedocument.wordprocessingml.document'
];

// 3. Validarea Semnăturii Fișierului (Magic Number)
const fileSignature = fileBuffer.toString('hex', 0, 4);

const magicNumbers = {
    '89504e47': 'image/png',      // PNG
    'ffd8ffe0': 'image/jpeg',     // JPEG
    '47494638': 'image/gif',      // GIF
    '25504446': 'application/pdf' // PDF
};

const detectedType = magicNumbers[fileSignature];
if (!detectedType || detectedType !== file.mimetype) {
    throw new Error('Incompatibilitate semnătură fișier');
}
```

De ce Contează Magic Numbers

Magic numbers (semnăturile fișierelor) sunt primii câțiva bytes ai unui fișier care îi identifică tipul. Aceasta previne ca atacatorii să:

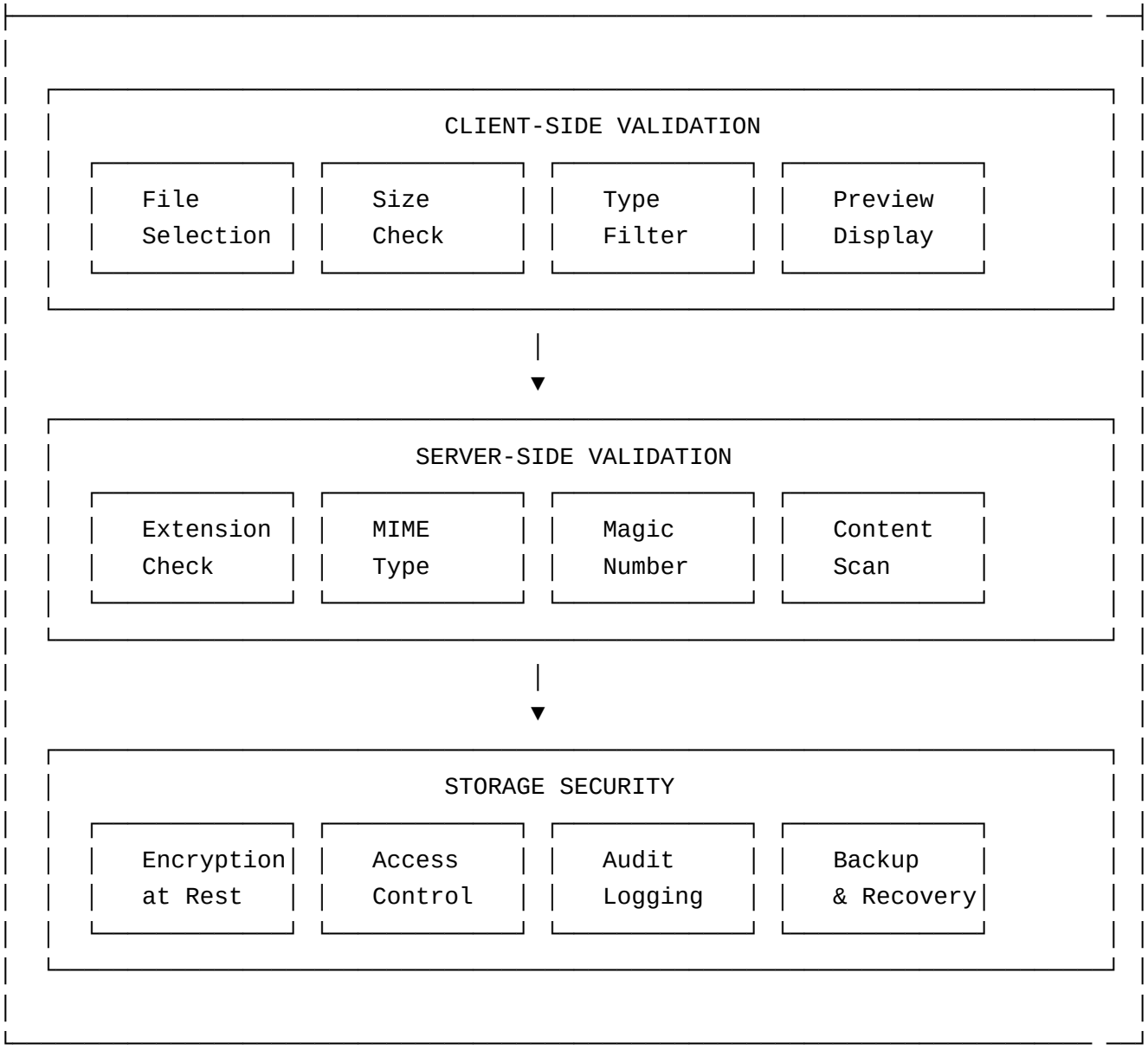
- Încarce fișiere executabile cu extensii de imagine
- Ocolească validarea tipului MIME
- Exploateze vulnerabilitățile browserului cu fișiere malițioase

Limite Progressive de Dimensiune

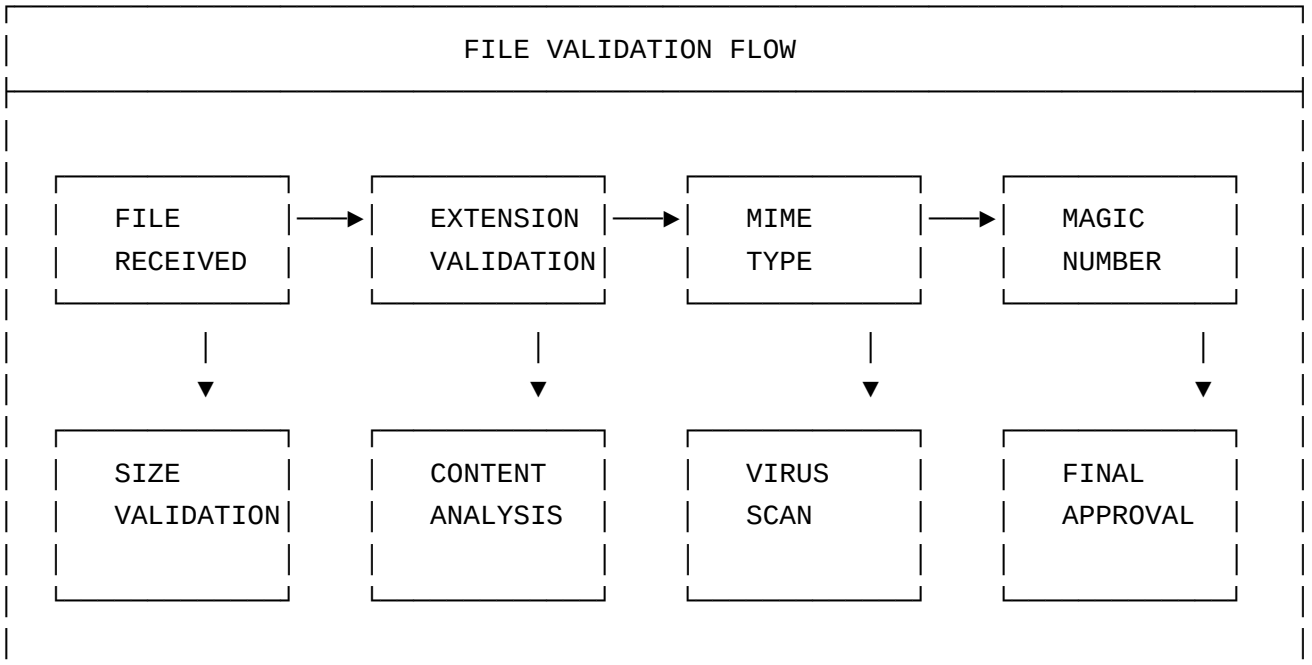
```
const sizeLimits = {
    images: 5 * 1024 * 1024,     // 5MB pentru imagini
    documents: 10 * 1024 * 1024, // 10MB pentru documente
    total: 50 * 1024 * 1024      // 50MB total per utilizator
};
```

Gestionarea Cotelor de Stocare
- **Limite per Utilizator**: Previne abuzul de stocare
- **Limite pe Tip de Fișier**: Optimizează stocarea pentru diferite tipuri de conținut
- **Scalare Progressive**: Creșterea limitelor pentru utilizatorii premium

6.3.2 Arhitectura Securității pe Mai Multe Straturi

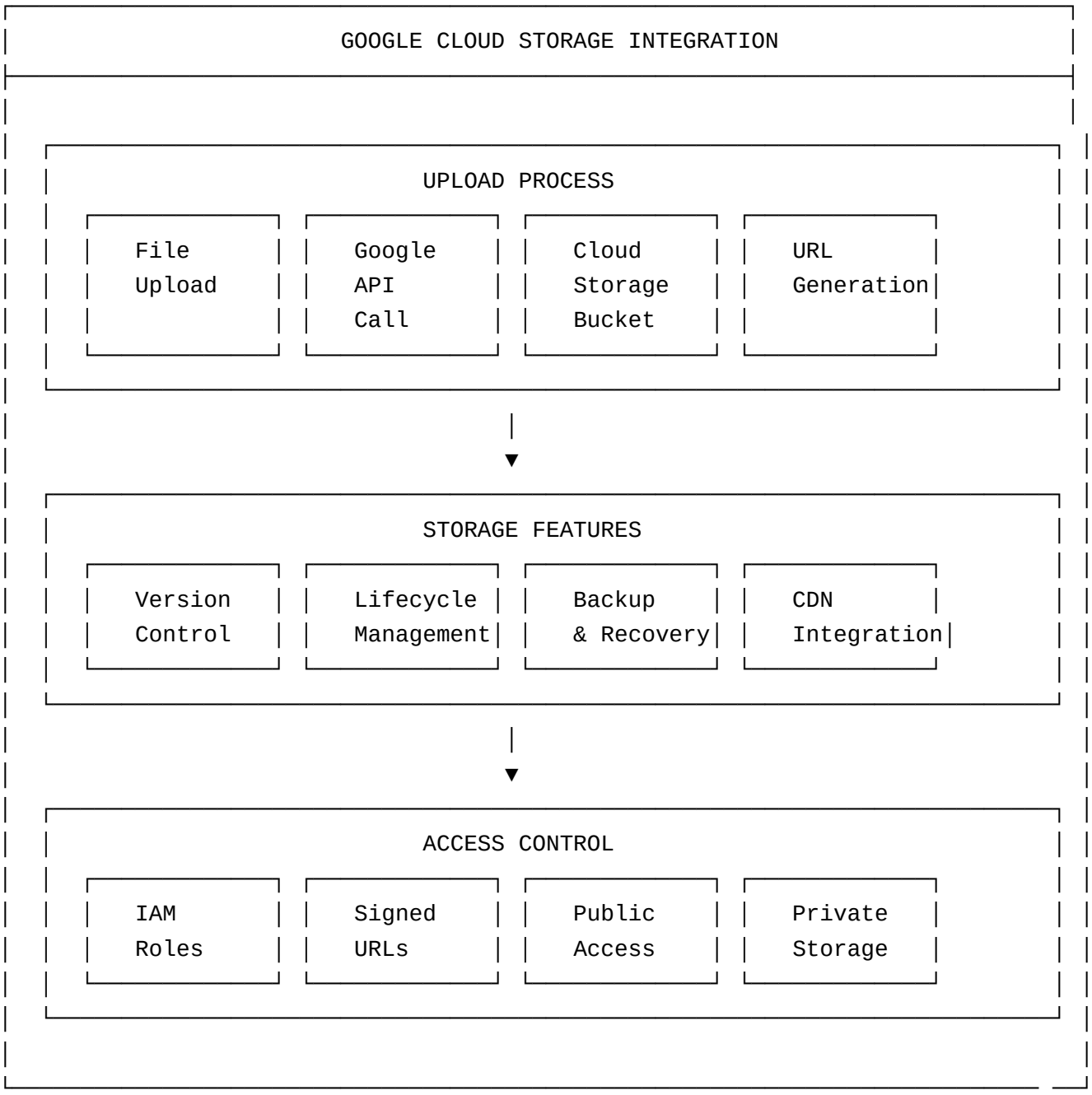


6.3.3 Fluxul de Validare şi Sanitizare a Fişierelor



Funcții de Securitate:
- **Validare pe Mai Multe Straturi**: Verificarea extensiei fișierului, tipului MIME și a semnăturii (magic number)
- **Scanare Virus**: Integrare cu servicii de securitate pentru detectarea malware-ului
- **Analiză de Conținut**: Inspectarea aprofundată a conținutului fișierelor pentru amenințări
- **Control al Accesului**: Permisiuni bazate pe rol și jurnalizare audit
- **Criptare**: Criptare AES-256 pentru toate fișierele stocate
- **Conformitate**: Funcții conforme GDPR, HIPAA și SOC 2

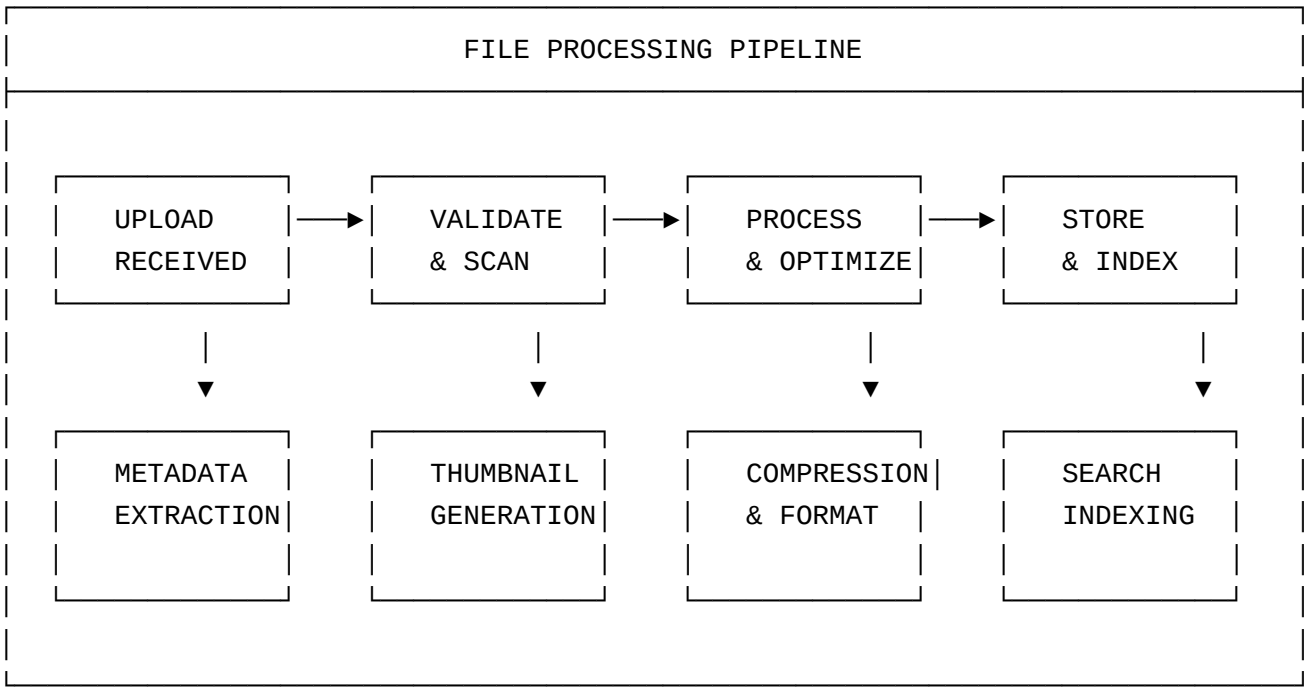
6.3.4 Arhitectura Integrării cu Google Cloud Storage



Funcționalități Cloud Storage:
- **CDN Global**: Livrare rapidă a fișierelor la nivel mondial
- **Control al Versiunilor**: Versionare automată și rollback al fișierelor

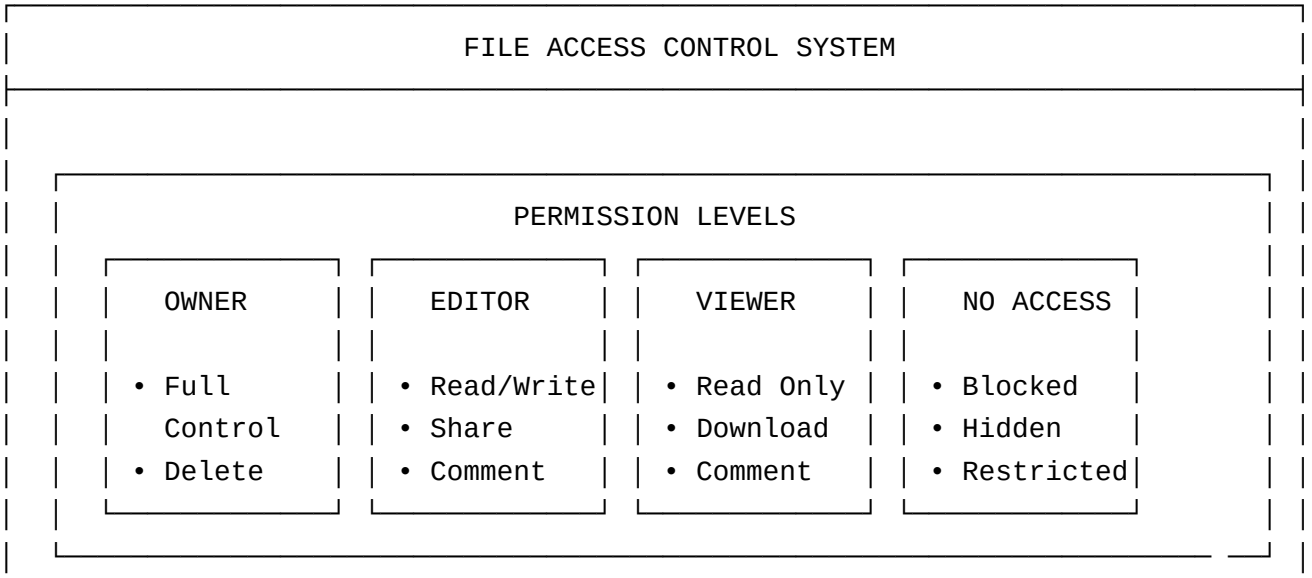
- **Managementul Ciclu de Viaţă**: Arhivare şi ştergere automată
- **Control Acces**: Permisiuni detaliate şi URL-uri semnate
- **Backup & Recuperare**: Backup automat cu recuperare la puncte în timp
- **Conformitate**: Conformitate SOC 2, ISO 27001 şi GDPR

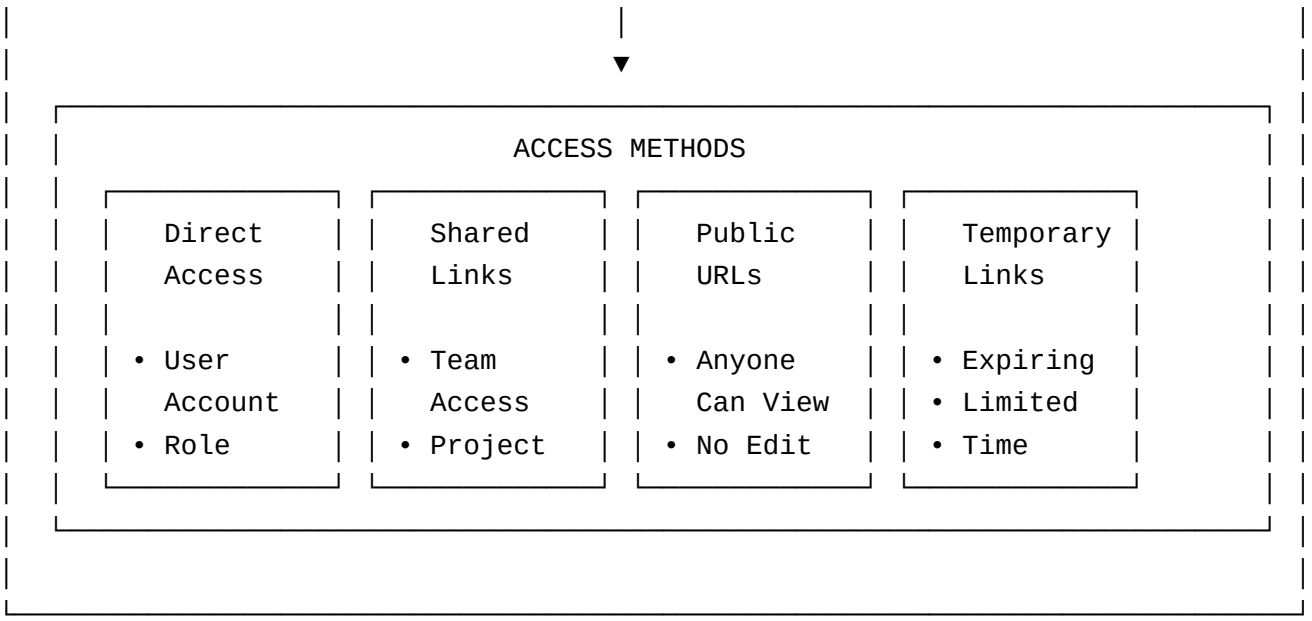
6.3.5 Pipeline de Procesare şi Optimizare a Fişierelor



Funcţionalităţi de Procesare:
- **Extracţie Metadata**: Extracţie automată a proprietăţilor fişierelor
- **Generare Miniaturi**: Previziuni vizuale pentru fişiere
- **Compresie**: Compresie inteligentă pentru optimizarea stocării
- **Conversie Format**: Optimizare automată a formatului
- **Indexare pentru Căutare**: Capacităţi de căutare full-text
- **Optimizare Performanţă**: Lazy loading şi caching

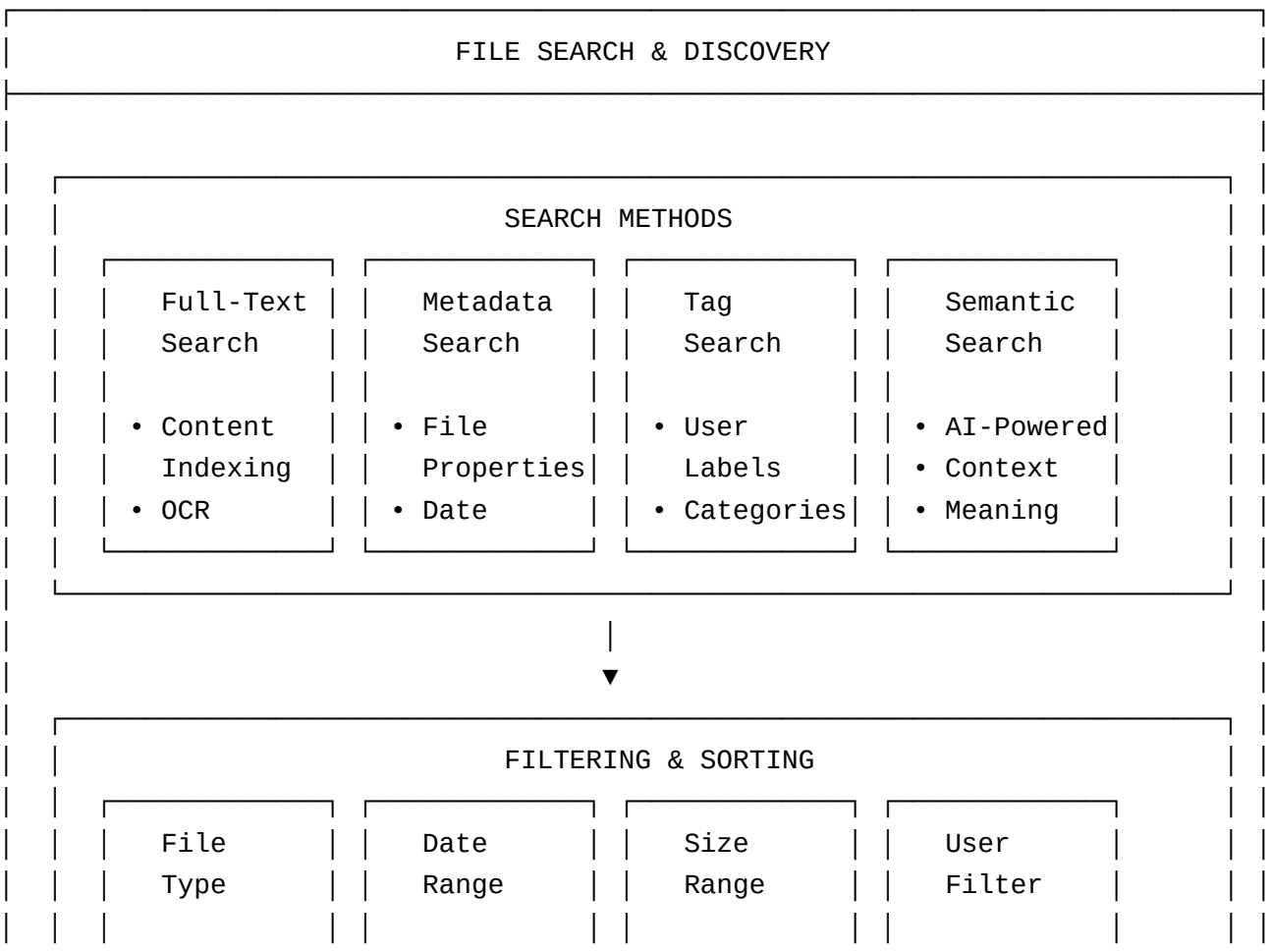
6.3.6 Controlul Accesului şi Sistemul de Permisiuni pentru Fişiere

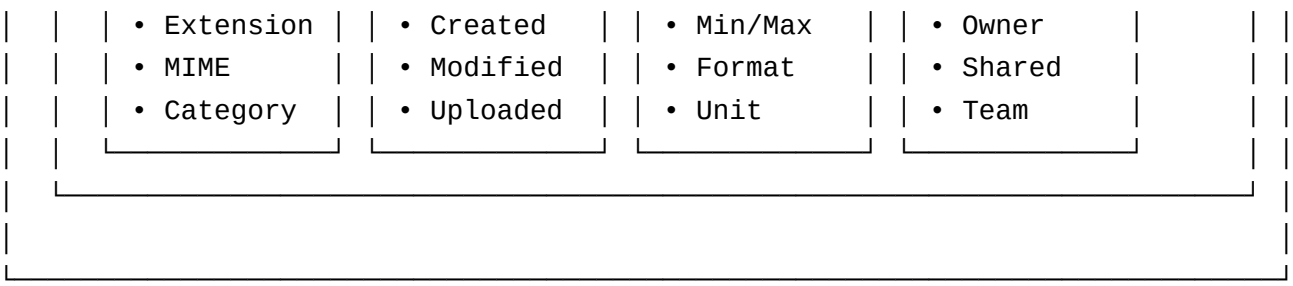




Funcționalități de Control al Accesului:
- **Acces Bazat pe Roluri**: Permisiuni detaliate în funcție de rolul utilizatorului
- **Securitate la Nivel de Proiect**: Fișierele moștenesc permisiunile proiectului
- **Acces Temporar**: Partajare limitată în timp cu expirare
- **Audit și Loguri**: Istoric complet al accesului și urmărire
- **Filigranare**: Adăugarea automată de watermark pentru fișiere sensibile
- **Limitări la Descărcare**: Restricții configurabile pentru descărcări

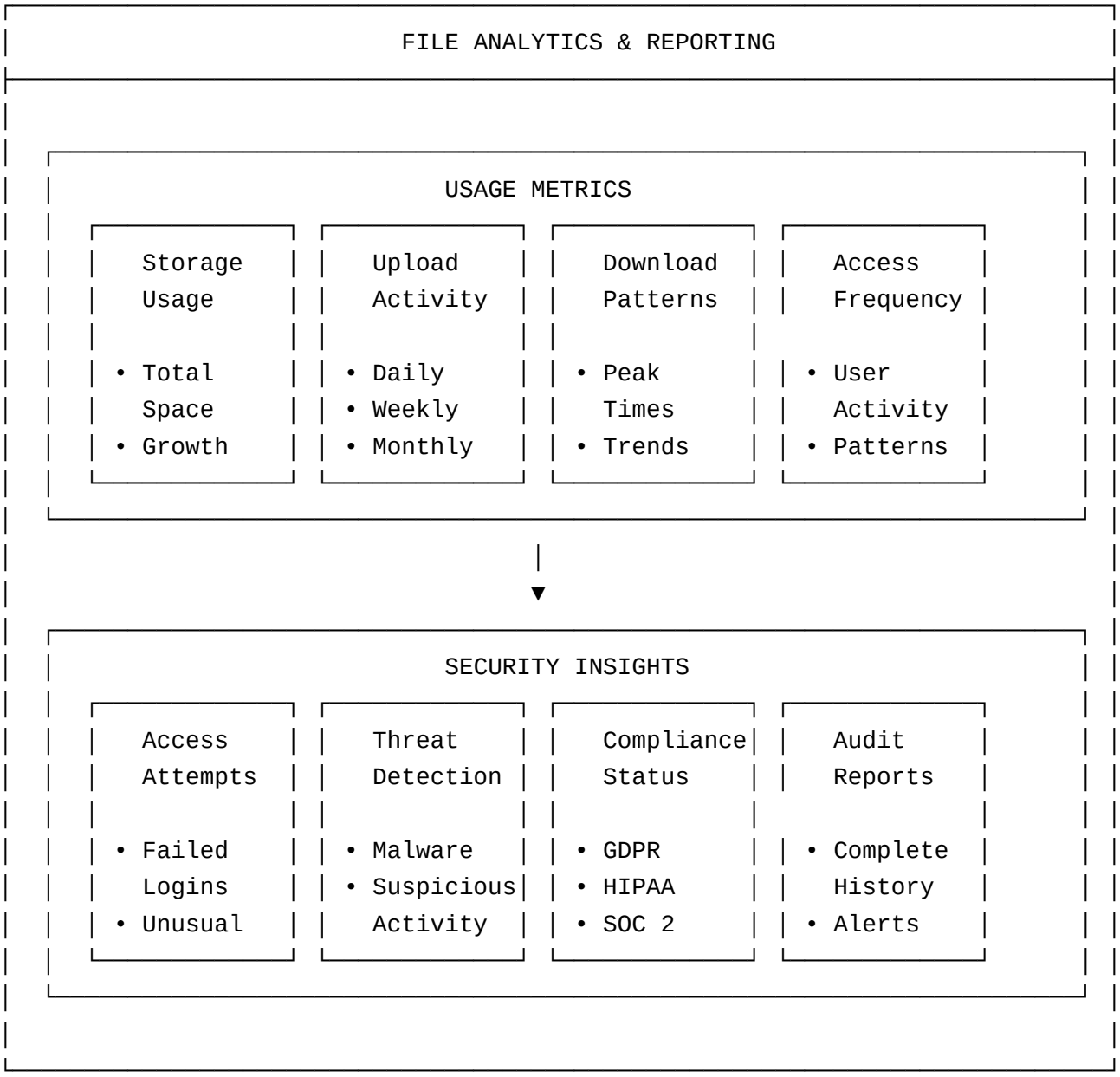
6.3.7 Sistem de Căutare și Descoperire a Fișierelor





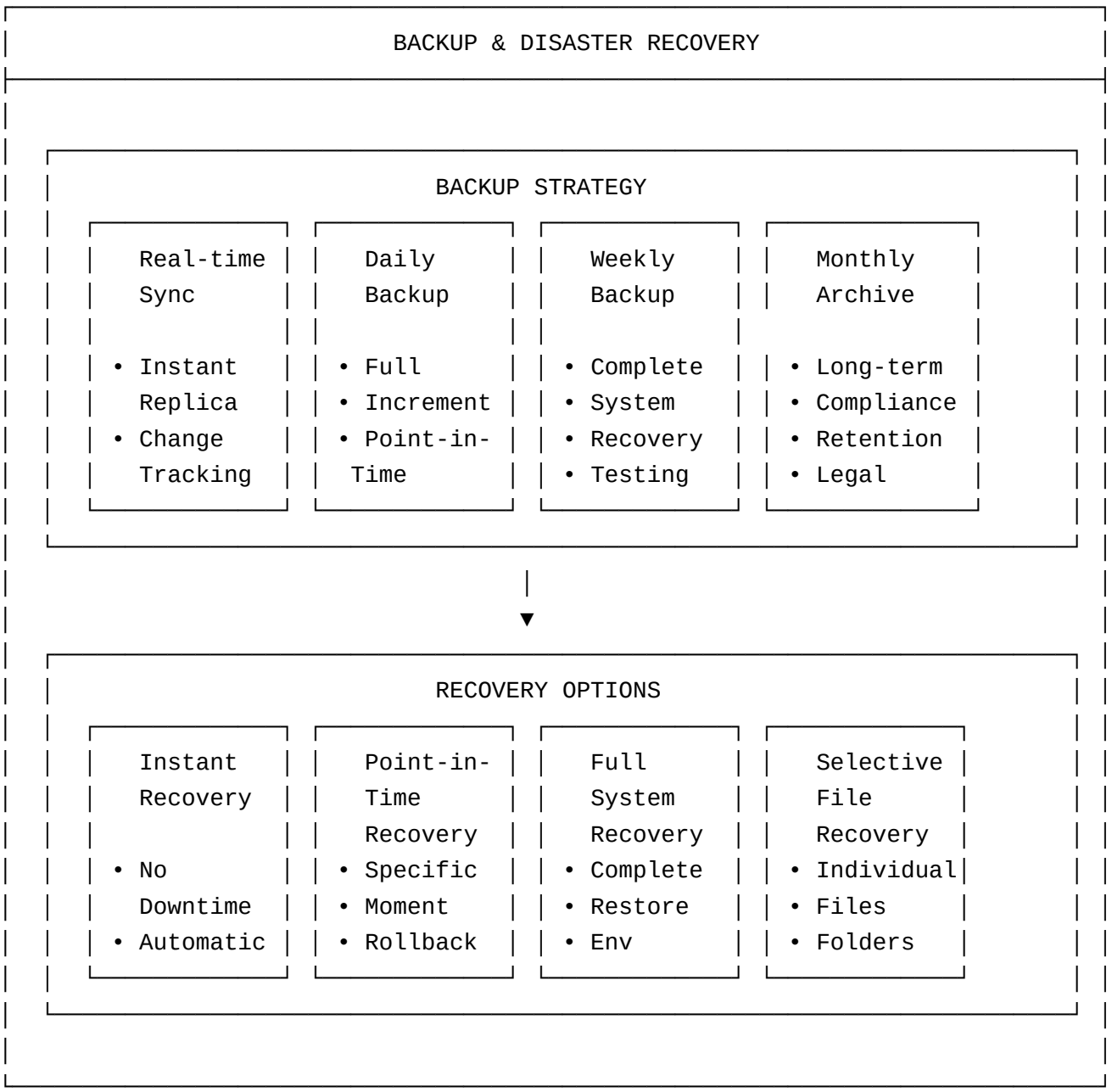
Funcționalități de Căutare:
- **Căutare Full-Text**: Căutare în conținutul fișierelor folosind OCR și indexare
- **Căutare după Metadate**: Căutare după proprietăți fișier, date și dimensiuni
- **Sistem de Etichete**: Etichete și categorii definite de utilizator pentru organizare ușoară
- **Căutare Semantică**: Căutare AI care înțelege contextul și sensul
- **Filtrare Avansată**: Combinații multiple de filtre pentru rezultate precise
- **Istoric Căutări**: Urmărirea și salvarea interogărilor pentru utilizări viitoare

6.3.8 Sistem de Analiză și Raportare a Fișierelor



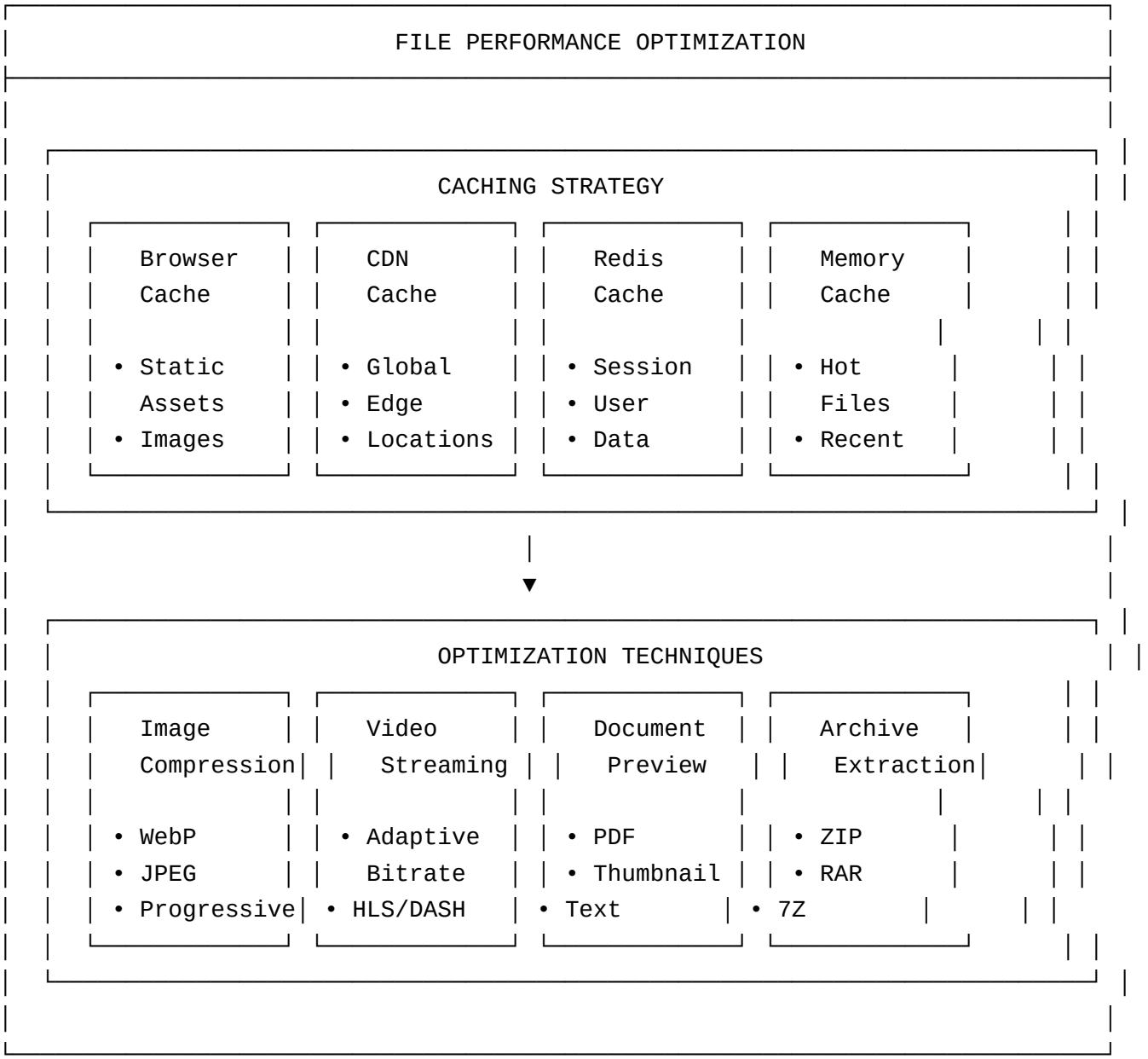
Funcţionalităţi Analitice:
- **Informaţii despre Stocare**: Detalii despre utilizarea spaţiului şi tipare de creştere
- **Comportament Utilizator**: Urmărirea accesului la fişiere şi activităţii utilizatorilor
- **Monitorizare Securitate**: Detectare ameninţări în timp real şi alerte de securitate
- **Raportare Conformitate**: Stare automată de conformitate şi rapoarte de audit
- **Măsurători de Performanţă**: Monitorizarea performanţei upload/download fişiere
- **Analiză Predictivă**: Previziuni AI privind stocarea şi utilizarea

6.3.9 Backup Fişiere & Recuperare în Caz de Dezastru



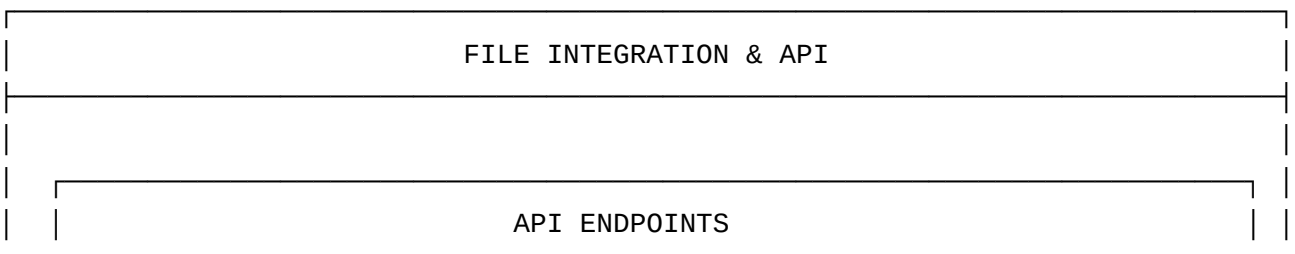
Funcţionalităţi Backup:
- **Replicare în Timp Real**: Sincronizare instantanee a fişierelor între mai multe locaţii
- **Backup Incremental**: Strategie eficientă de backup pentru economisirea timpului şi spaţiului
- **Recuperare la Moment Specific**: Restaurare la orice moment din timp
- **Distribuţie Geografică**: Centre de date multiple pentru protecţia împotriva dezastrului
- **Testare Automată**: Verificarea regulată a backup-urilor şi testarea recuperării
- **Conformitate Reglementară**: Respectă cerinţele legale pentru păstrarea datelor

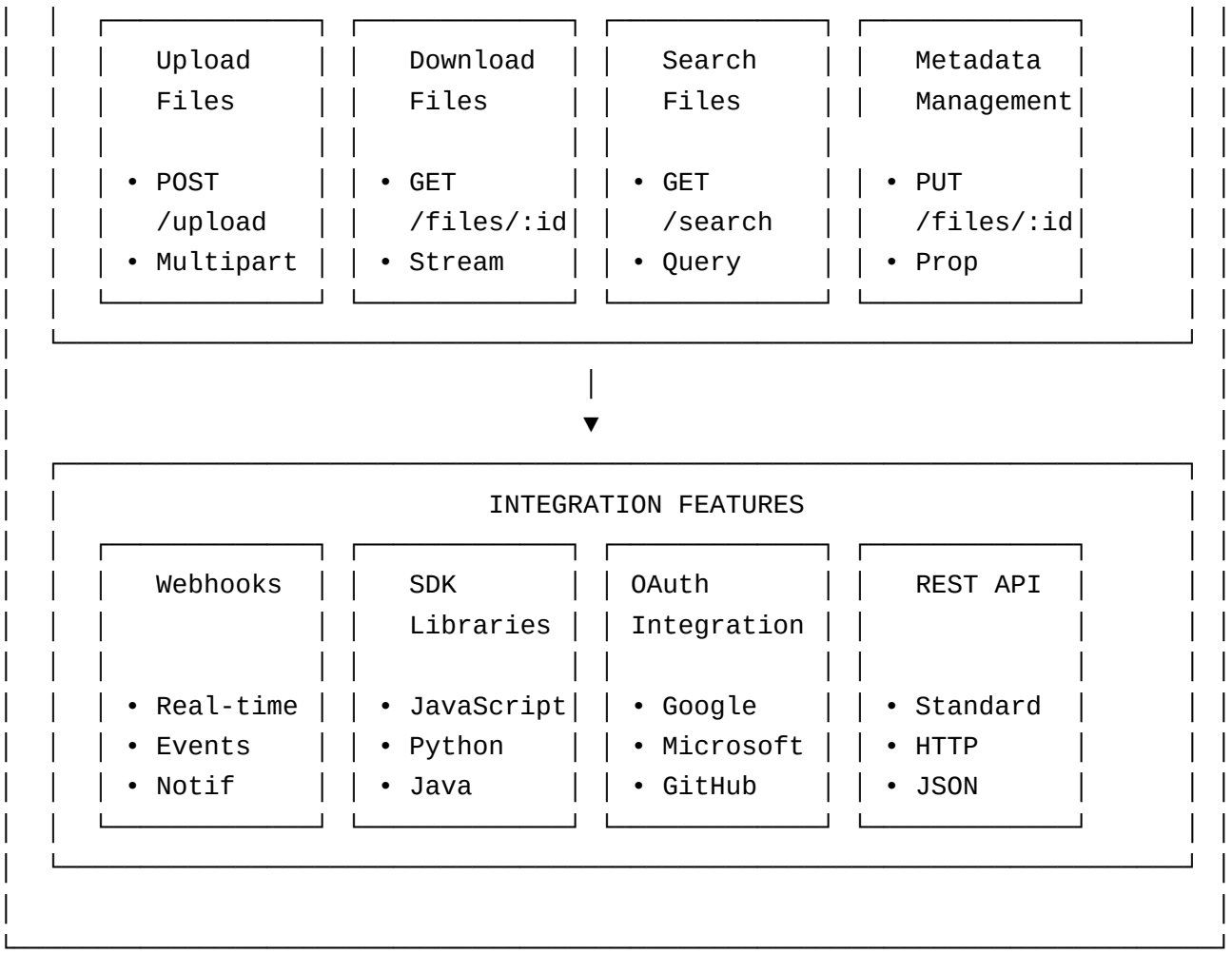
6.3.10 Performanța și Optimizarea Fișierelor



Funcționalități de Performanță:
- **Caching Multi-Nivel**: Cache în browser, CDN, Redis și memorie
- **Optimizare Imagini**: Compresie automată și conversie de format
- **Streaming Video**: Streaming adaptiv pentru performanță optimă
- **Previzualizare Documente**: Generare instantanee a previzualizărilor pentru formate comune
- **Încărcare Lazie**: Încărcarea fișierelor doar când este necesar
- **Îmbunătățire Progresivă**: Optimizare pentru diferite condiții de rețea

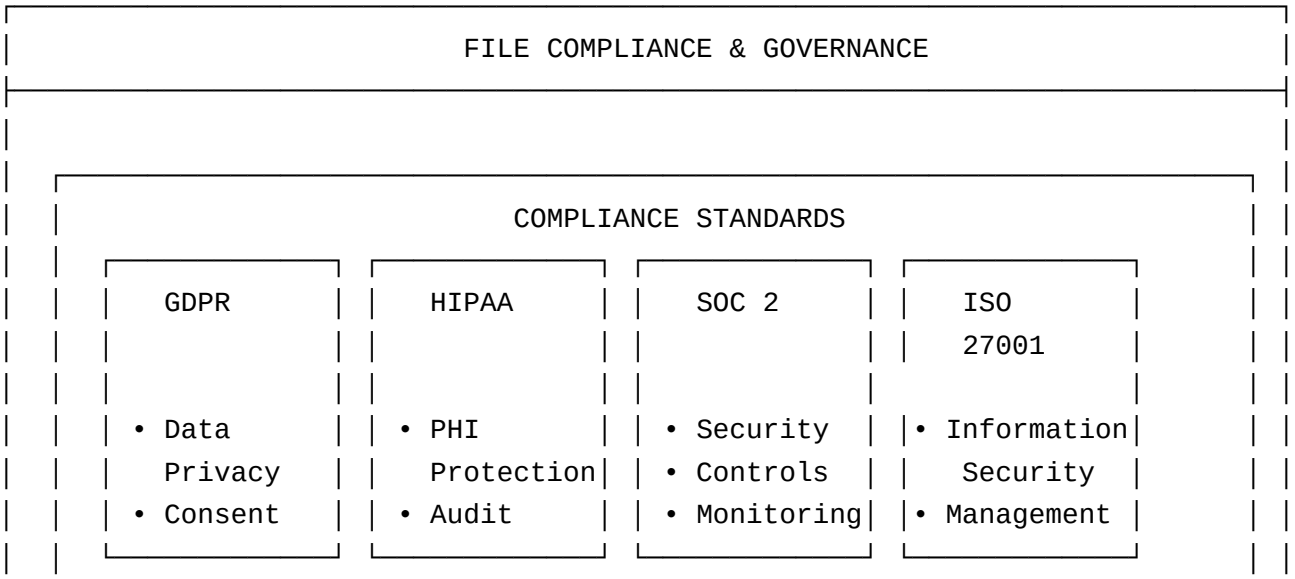
6.3.11 Integrarea Fișierelor și Sistemul API

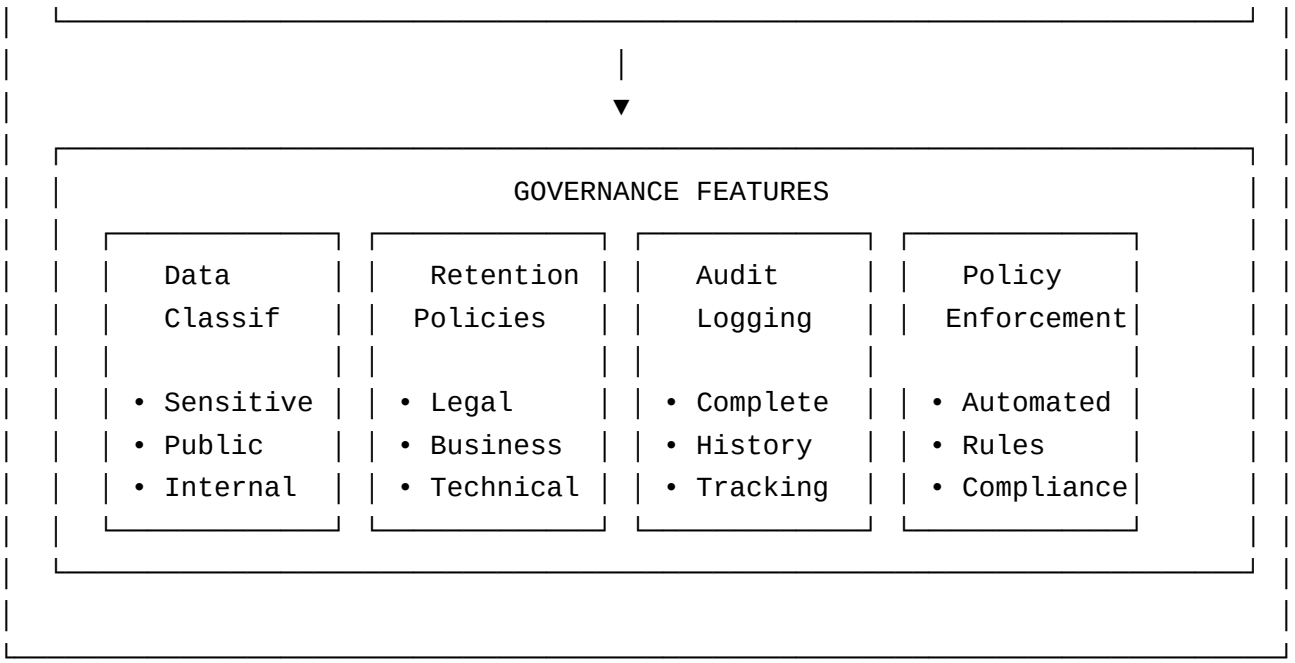




Funcţionalităţi de Integrare:
- **API RESTful**: Metode HTTP standard pentru toate operațiunile cu fișiere
- **Suport Webhook**: Notificări în timp real pentru evenimentele fişierelor
- **Biblioteci SDK**: Biblioteci client pentru mai multe limbaje de programare
- **Integrare OAuth**: Autentificare securizată cu servicii populare
- **Streaming Evenimente**: Actualizări în timp real pentru activitatea fişierelor
- **Operaţiuni în Batch**: Procesare eficientă a fişierelor în volum mare

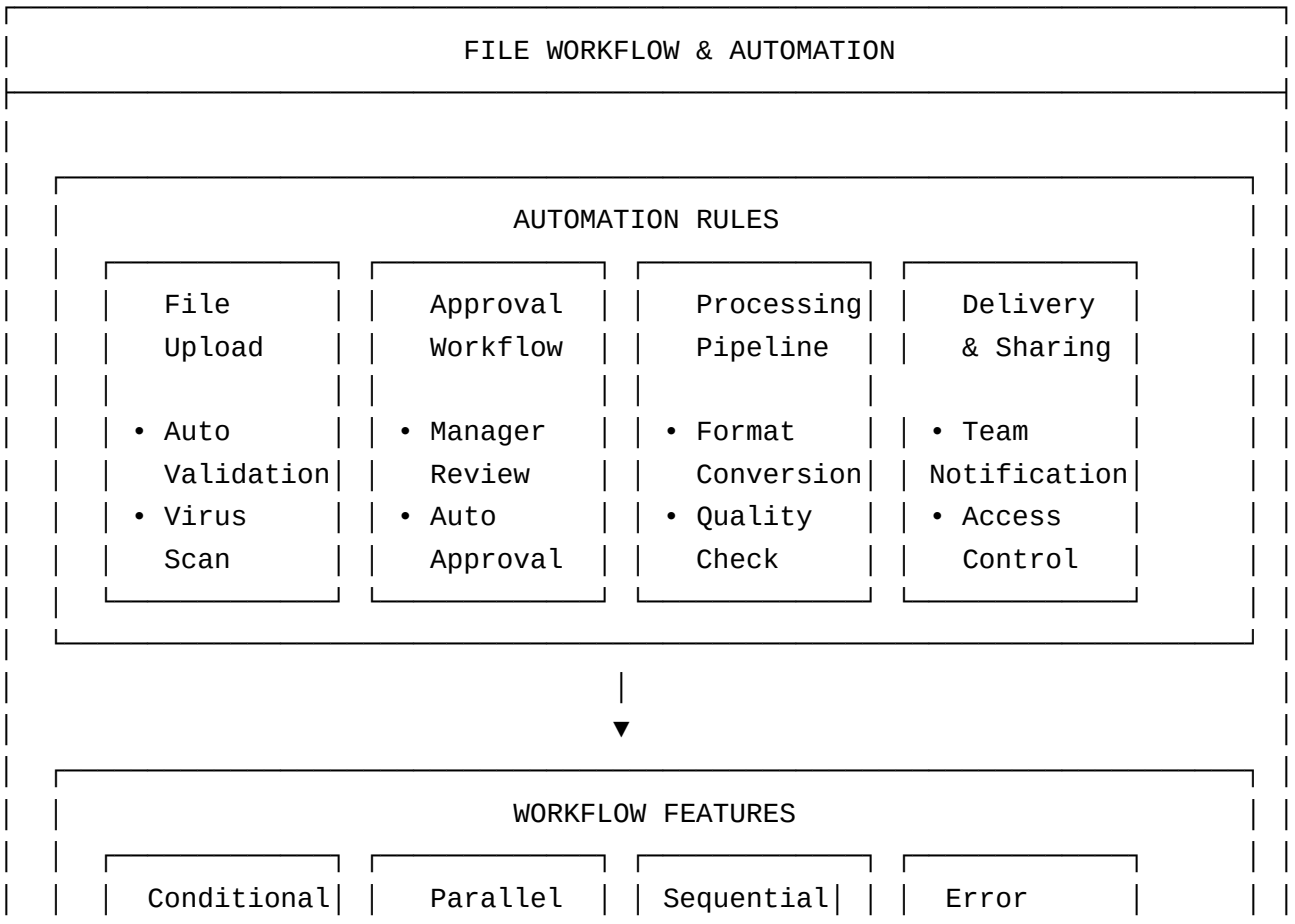
6.3.12 Conformitate şi Guvernanţă a Fişierelor

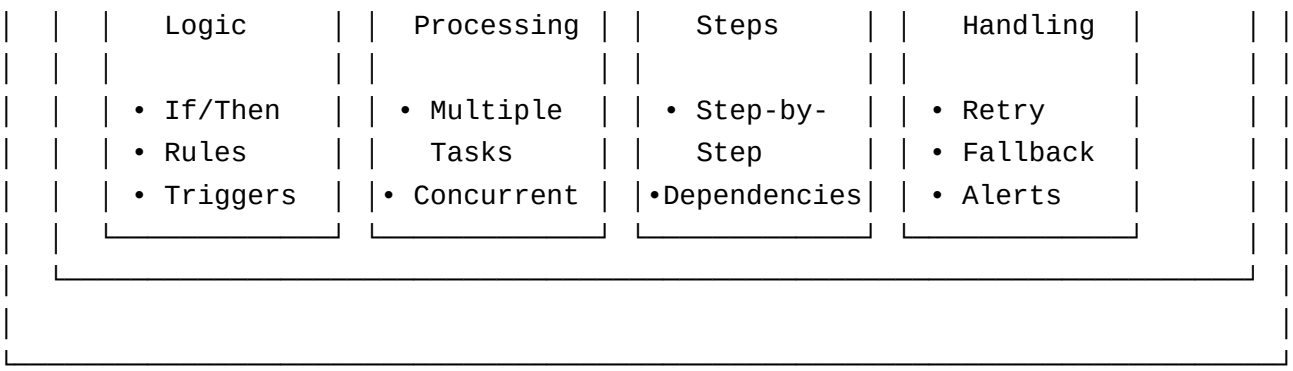




Funcționalități de Conformitate:
- **Clasificarea Datelor**: Clasificare automată a informațiilor sensibile
- **Politici de Păstrare**: Reguli configurabile pentru păstrarea și ștergerea datelor
- **Audit Logging**: Traseu complet de audit pentru rapoarte de conformitate
- **Aplicarea Politicilor**: Monitorizare și aplicare automată a politicilor
- **Controale de Confidențialitate**: Gestionarea datelor conform GDPR și managementul consimțământului
- **Standardele de Securitate**: Conformitate SOC 2, ISO 27001 și HIPAA

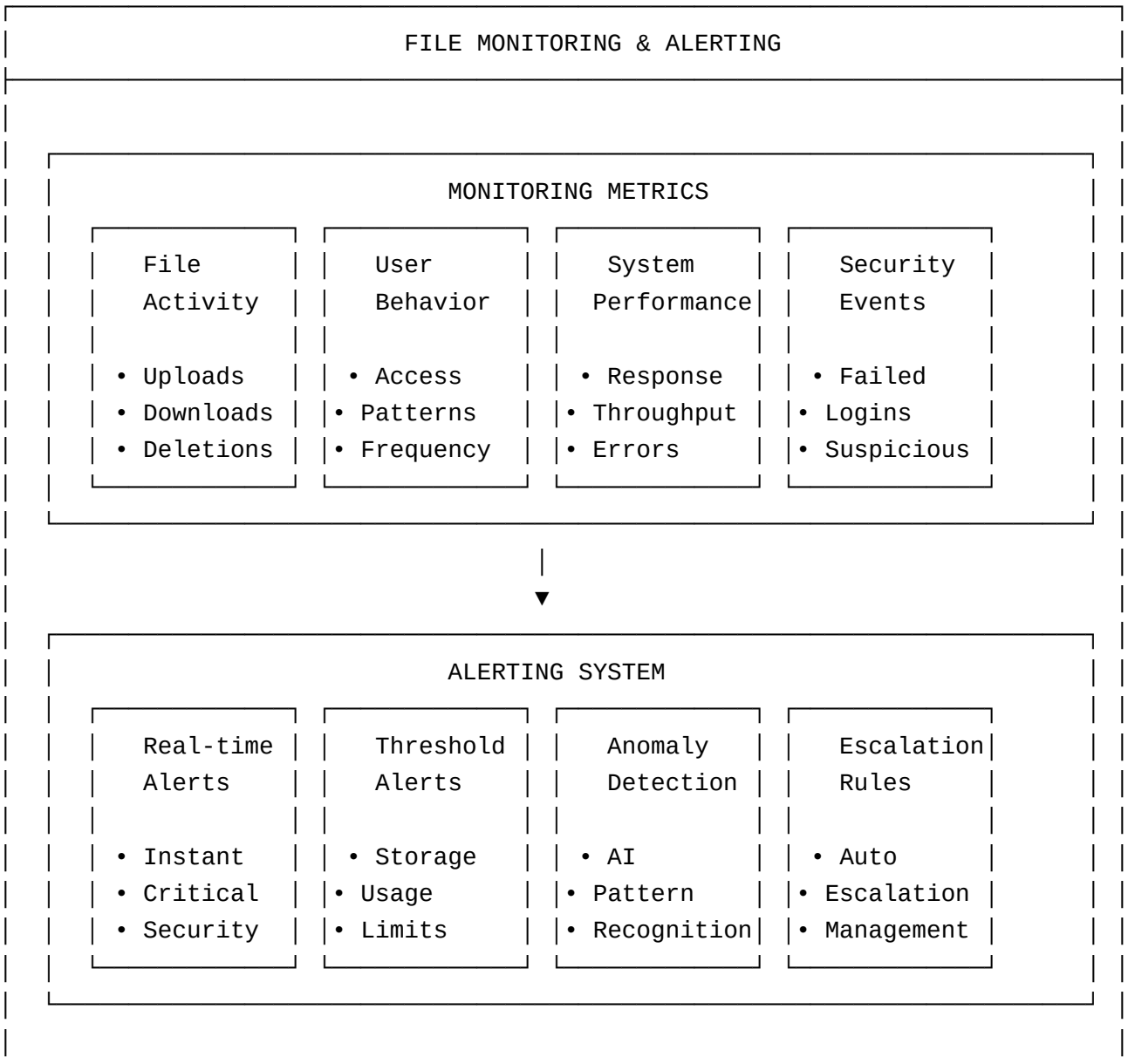
6.3.13 Flux de Lucru și Automatizare a Fișierelor





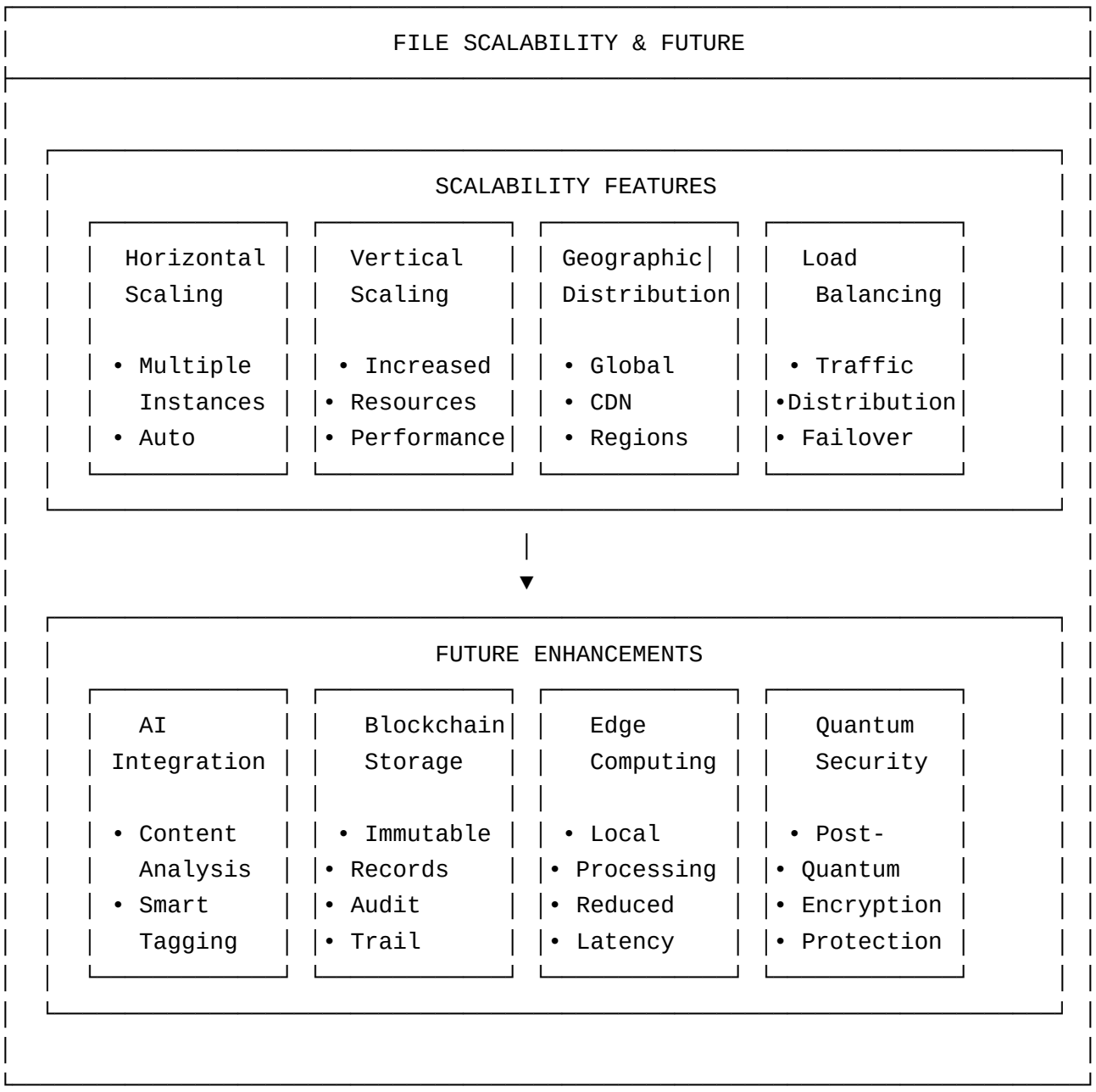
Funcționalități Workflow:
- **Logică Condițională**: Reguli inteligente bazate pe proprietățile și conținutul fișierelor
- **Procesare Paralelă**: Mai multe sarcini executate simultan
- **Paşi Secvențiali**: Executarea ordonată a fluxului de lucru cu dependențe
- **Gestionarea Erorilor**: Reîncercări și mecanisme de rezervă automate
- **Fluxuri de Aprobare**: Procese de aprobare configurabile pentru fișiere sensibile
- **Hook-uri de Integrare**: Conectare cu sisteme și servicii externe

6.3.14 Monitorizarea Fişierelor şi Alerta



Funcţionalităţi de Monitorizare:
- **Alerte în Timp Real**: Notificări instantanee pentru evenimente critice
- **Monitorizare Praguri**: Limite și alerte configurabile
- **Detectarea Anomaliilor**: Recunoaștere de tipare bazată pe AI
- **Metrici de Performanţă**: Monitorizare cuprinzătoare a performanței sistemului
- **Monitorizare Securitate**: Detectarea evenimentelor de securitate în timp real
- **Reguli de Escaladare**: Escaladare automată pentru probleme critice

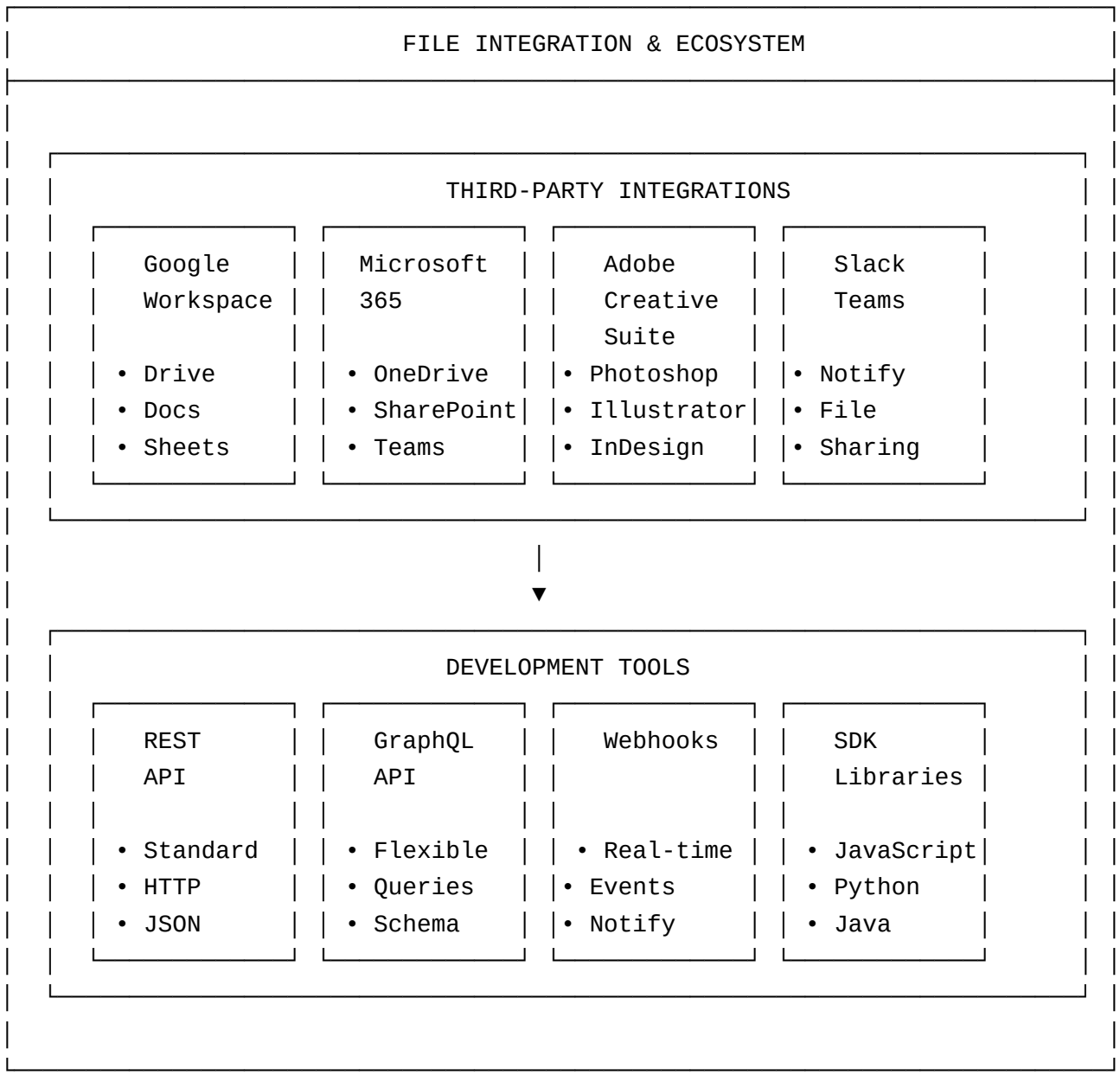
6.3.15 Scalabilitatea Fişierelor şi Arhitectura Viitoare



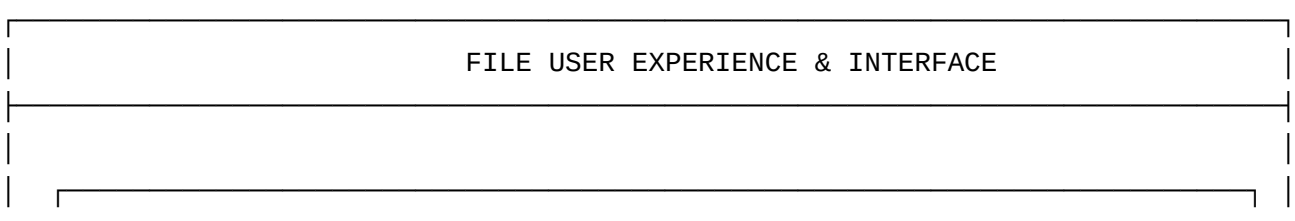
Funcţionalităţi de Scalabilitate:
- **Scalare Orizontală**: Mai multe instanțe de server pentru distribuirea încărcării
- **Scalare Verticală**: Resurse crescute pentru servere individuale
- **Distribuție Geografică**: CDN global și locații edge
- **Echilibrare a Încărcării**: Distribuirea inteligentă a traficului și failover

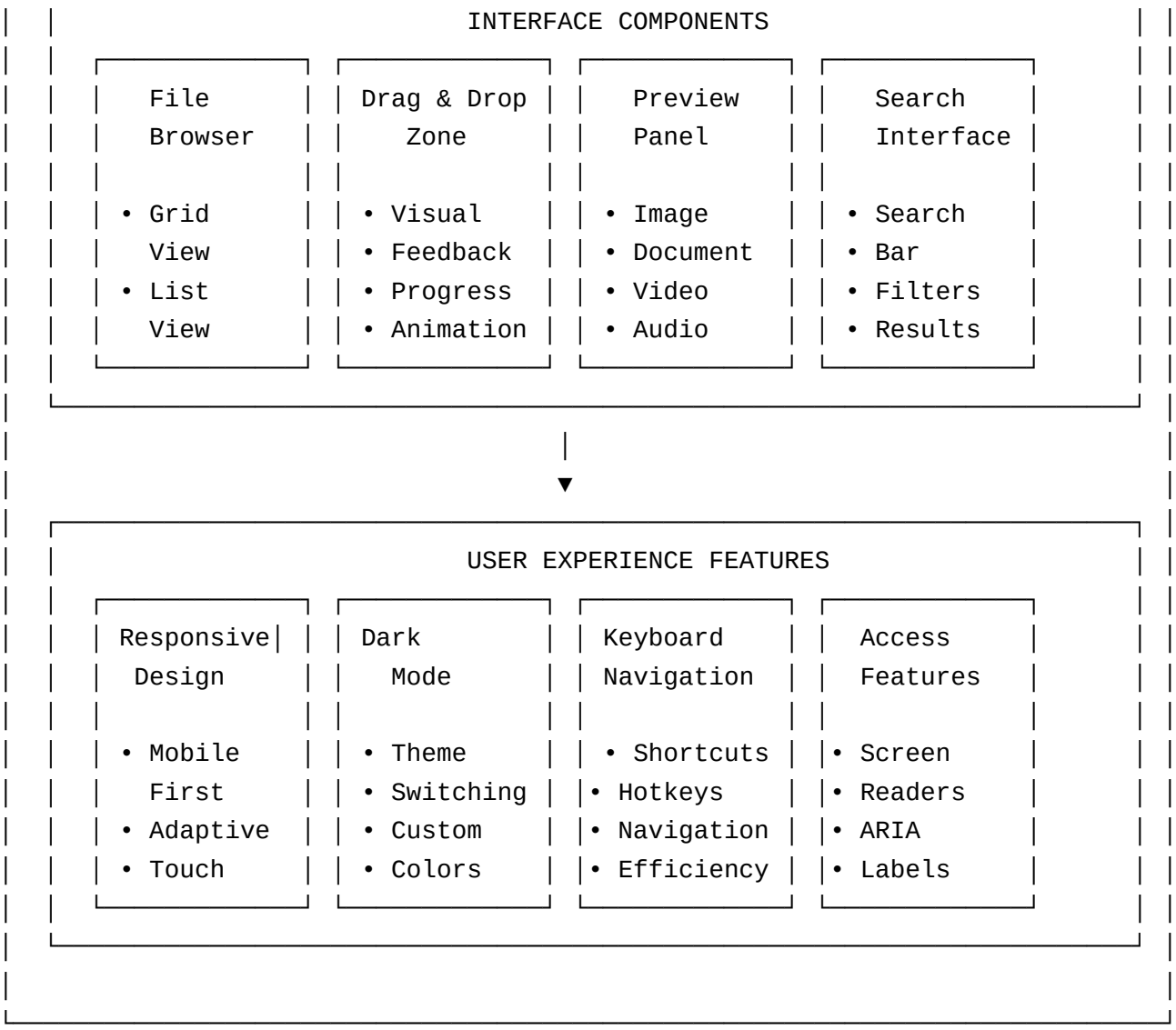
- **Auto-scalare**: Alocarea automată a resurselor în funcție de cerere
- **Pregătit pentru Viitor**: Arhitectură proiectată pentru tehnologii emergente

6.3.16 Integrarea Fișierelor & Ecosistem



Funcționalități de Integrare:
- **Google Workspace**: Integrare perfectă cu Google Drive și Docs
- **Microsoft 365**: Compatibilitate completă cu OneDrive și SharePoint
- **Adobe Creative Suite**: Integrare directă cu uneltele de design
- **Platforme de Comunicare**: Integrare cu Slack, Teams și Discord
- **REST API**: API HTTP standard pentru integrări personalizate
- **GraphQL API**: Limbaj flexibil de interogare pentru nevoi complexe de date





Funcţionalităţi de Experienţă a Utilizatorului:
- **Design Responsiv**: Optimizat pentru toate dispozitivele şi dimensiunile de ecran
- **Mod Întunecat**: Vizualizare confortabilă în medii cu lumină scăzută
- **Navigare prin Tastatură**: Suport complet pentru utilizatorii avansaţi
- **Accesibilitate**: Conformitate WCAG 2.1 AA cu suport pentru cititoare de ecran
- **Interfaţă Tactilă**: Optimizat pentru dispozitive mobile şi tablete
- **Personalizare**: Tematici şi layout-uri configurabile de utilizator

6.3.17 Securitatea Conţinutului

Integrare Scanare Virus

```
// Integration with ClamAV or similar
const scanFile = async (fileBuffer) => {
    try {
        const result = await virusScanner.scan(fileBuffer);
        if (result.isInfected) {
            throw new Error('File contains malware');
        }
        return true;
    } catch (error) {
```

```
        console.error('Virus scan failed:', error);
        // In production, reject files if scan fails
        throw new Error('Security scan failed');
    }
};
```

Image Processing and Sanitization

```
const processImage = async (fileBuffer, mimeType) => {
    const sharp = require('sharp');

    try {
        // Remove EXIF data (potential privacy risk)
        const processedImage = await sharp(fileBuffer)
            .removeExif()
            .resize(1920, 1080, { fit: 'inside' }) // Max dimensions
            .jpeg({ quality: 85 }) // Optimize quality
            .toBuffer();

        return processedImage;
    } catch (error) {
        throw new Error('Image processing failed');
    }
};
```

Chapter 7

Capitolul 8: Deploy & Scalare

Deploy-ul și scalarea sunt faze critice în ciclul de viață al aplicației. Acest capitol acoperă strategiile de deploy în producție, containerizarea cu Docker și abordările de scalare verticală și orizontală. Vom explora cum să ducem Sync de la dezvoltare la producție și să-l pregătim pentru utilizarea la scară enterprise.

7.0.1 Dockerfile pentru Dezvoltare

```
# Dockerfile.dev
FROM node:18-alpine

# Setează directorul de lucru
WORKDIR /app

# Copiază fișierele package
COPY package*.json ./

# Instalează dependențele
RUN npm ci --only=production

ENTRYPOINT ["dumb-init", "--"]
CMD ["node", "src/server.js"]
```

7.0.2 Docker Compose for Production

```
# docker-compose.prod.yml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile.prod
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
```

```
    - DATABASE_URL=${DATABASE_URL}
    - JWT_SECRET=${JWT_SECRET}
    - GOOGLE_CLOUD_PROJECT_ID=${GOOGLE_CLOUD_PROJECT_ID}
  depends_on:
    - db
    - redis
  restart: unless-stopped
  deploy:
    replicas: 3
    resources:
      limits:
        cpus: '1.0'
        memory: 1G
      reservations:
        cpus: '0.5'
        memory: 512M

db:
  image: postgres:15-alpine
  environment:
    - POSTGRES_DB=${POSTGRES_DB}
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
  volumes:
    - postgres_prod_data:/var/lib/postgresql/data
  restart: unless-stopped
  deploy:
    resources:
      limits:
        cpus: '2.0'
        memory: 4G

redis:
  image: redis:7-alpine
  command: redis-server --appendonly yes --requirepass ${REDIS_PASSWORD}
  volumes:
    - redis_prod_data:/data
  restart: unless-stopped
  deploy:
    resources:
      limits:
        cpus: '0.5'
        memory: 512M

nginx:
  image: nginx:alpine
  ports:
    - "80:80"
```

```
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
    depends_on:
      - app
    restart: unless-stopped

volumes:
  postgres_prod_data:
  redis_prod_data:
```

7.0.3 Vertical Scaling

Server Resource Optimization

```
// server.js - Optimized for vertical scaling
const cluster = require('cluster');
const os = require('os');

if (cluster.isMaster) {
    const numCPUs = os.cpus().length;

    console.log(`Master ${process.pid} is running`);
    console.log(`Forking for ${numCPUs} CPUs`);

    // Fork workers
    for (let i = 0; i < numCPUs; i++) {
        cluster.fork();
    }

    cluster.on('exit', (worker, code, signal) => {
        console.log(`Worker ${worker.process.pid} died`);
        // Replace the dead worker
        cluster.fork();
    });
} else {
    // Worker process
    require('./src/app');
    console.log(`Worker ${process.pid} started`);
}
```

7.0.4 Horizontal Scaling

Load Balancer Configuration

```
// loadBalancer.js
const http = require('http');
const httpProxy = require('http-proxy');
```

```
const proxy = httpProxy.createProxyServer({});

const servers = [
    'http://app1:3000',
    'http://app2:3000',
    'http://app3:3000'
];

let currentServer = 0;

const server = http.createServer((req, res) => {
    // Round-robin load balancing
    const target = servers[currentServer];
    currentServer = (currentServer + 1) % servers.length;

    proxy.web(req, res, { target });
});

server.listen(8080);
```

Chapter 8

Bibliografie

8.1 Tehnologii și Framework-uri Principale

8.1.1 Tehnologii Backend

Node.js
- Node.js Foundation. (2024). Documentația Node.js. https://nodejs.org/docs/

Express.js
- StrongLoop, IBM și alți contribuitori. (2024). Express.js – Framework web rapid, minimalist și neopinativ pentru Node.js. https://expressjs.com/

PostgreSQL
- PostgreSQL Global Development Group. (2024). PostgreSQL: Cel mai avansat sistem de baze de date relaționale open-source. https://www.postgresql.org/

Prisma ORM
- Documentația Prisma. (2024). Prisma Client, Prisma Migrate și Prisma Studio. https://www.prisma.io/docs/

8.1.2 Tehnologii Frontend

React
- Facebook, Inc. (2024). React – Bibliotecă JavaScript pentru construirea interfețelor de utilizator. https://reactjs.org/

Vite
- Documentația Vite. (2024). Unealtă de build care oferă o experiență de dezvoltare mai rapidă și mai eficientă. https://vitejs.dev/guide/

JavaScript
- ECMAScript. (2024). Specificația limbajului ECMAScript. https://tc39.es/ecma262/

TailwindCSS
- Documentația Tailwind CSS. (2024). Construiește rapid site-uri moderne fără a părăsi HTML-ul. https://tailwindcss.com/docs

8.1.3 Gestionarea Stării

Zustand
- Documentația Zustand. (2024). Gestionare simplă a stării pentru React. https://github.com/pmndrs/zustand#readme

8.1.4 Gestionarea Formularelor

Formik
- Documentația Formik. (2024). Gestionarea stării formularelor și validare pentru React. https://formik.org/docs/overvi

Yup
- Documentația Yup. (2024). Validator și parser de obiecte JavaScript. https://github.com/jquense/yup#api

8.1.5 Autentificare & Securitate

JSON Web Tokens (JWT)
- Auth0. (2024). JWT.io – Debugger pentru JSON Web Tokens. https://jwt.io/

Bcrypt
- Documentația Bcrypt. (2024). Bibliotecă pentru criptarea parolelor. https://github.com/dcodeIO/bcrypt.js

Helmet.js
- Documentația Helmet. (2024). Middleware de securitate pentru Express.js. https://helmetjs.github.io/docs/

8.1.6 Gestiona