UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Proiect Editor interactiv de text

Inginerie Software

Sescu Diana
Stănuț Denisa-Alexandra
Grupa: 30234

Facultatea de Automatică
și Calculatoare

Cuprins

1 Introducere

1.1 Descrierea Proiectului

Acest proiect reprezintă o aplicație web de tip editor de texte colaborativ, care permite utilizatorilor să creeze, editeze și să partajeze documente în timp real. Aplicația oferă funcționalități avansate de formatare, gestionare a stilurilor personalizate, partajare în multiple moduri și colaborare simultană între utilizatori.

Aplicația a fost dezvoltată cu scopul de a demonstra utilizarea design pattern-urilor fundamentale în dezvoltarea software și de a oferi o experiență de editare colaborativă, similară aplicațiilor precum Google Docs.

1.2 Tehnologii și Limbaje Utilizate

Proiectul utilizează o arhitectură full-stack, împărțită în două componente principale:

1.2.1 Backend

- **Python 3.x** - Limbaj de programare principal
- **Django 5.2.8** - Framework web pentru REST API
- **Django Channels** - Pentru comunicare WebSocket în timp real
- **Django REST Framework** - Pentru construirea API-ului RESTful
- **SimpleJWT** - Autentificare și autorizare bazată pe token-uri JWT
- **SQLite** - Bază de date relațională
- **xhtml2pdf** - Generare documente PDF
- **SMTP (Gmail)** - Serviciu de trimitere email-uri

1.2.2 Frontend

- **JavaScript (ES6+)** - Limbaj de programare
- **React 18** - Library pentru construirea interfețelor
- **React Router** - Navigare între pagini
- **Axios** - Client HTTP pentru comunicare cu backend-ul
- **WebSocket API** - Comunicare în timp real
- **CSS3** - Stilizare interfață

1.2.3 Protocoale și Formate

- **WebSocket** - Comunicare bidirectională în timp real
- **REST API** - Arhitectură pentru servicii web
- **JWT (JSON Web Tokens)** - Autentificare și autorizare
- **JSON** - Format de schimb de date

2 Funcționalități Principale

2.1 Autentificare și Gestionare Utilizatori

Sistemul oferă un mecanism complet de **autentificare și gestionare a utilizatorilor.** La înregistrare, fiecare utilizator trebuie să furnizeze un username unic, o adresă de email validă și o parolă securizată. Parolele sunt stocate în baza de date, care este implicit în Django și oferă protecție împotriva atacurilor.

Autentificarea se realizează prin intermediul token-urilor **JWT (JSON Web Tokens)**, care permit o comunicare sigură între client și server. După autentificare, utilizatorul primește două token-uri: un access token cu durata de viață de 1 zi pentru operațiunile curente și un refresh token valabil 7 zile pentru reînnoirea automată a sesiunii.

La inregistrare fiecare utilizator primește automat o **culoare unică**, generată aleator. Această culoare este utilizată pentru a identifica vizual modificările și cursorul fiecărui utilizator, facilitând astfel colaborarea în timp real.

2.2 Gestionare Documente

Utilizatorii autentificați pot crea documente noi, specificând un titlu și conținutul inițial. Fiecare document este asociat cu utilizatorul care l-a creat (proprietarul) și este stocat în baza de date împreună cu data creării și ultima actualizare.

Sistemul oferă o funcție de auto-save care se declanșează automat după 1.5 secunde de inactivitate a utilizatorului. Pe langa auto-save utilizatorii pot salva si manual documentele.

Documentele pot fi listate, editate și șterse de către proprietarii lor. În plus, utilizatorii pot vizualiza și documentele la care au fost adăugați ca colaboratori, având astfel un singur loc de unde să acceseze toate documentele.

2.3 Editare și Formatare Text

Editorul oferă funcțiile de formatare text, similare celor din aplicațiile de procesare text clasice. Utilizatorii pot aplica stiluri de bază precum bold, italic și underline.

Sistemul permite personalizarea aspectului textului prin selectarea fontului, modificarea dimensiunii textului și schimbarea culorii.

O caracteristică importantă este suportul pentru undo și redo, care permite utilizatorilor să revină la versiuni anterioare ale documentului.

Editorul suportă și inserarea de link-uri hypertext, încărcarea și inserarea de imagini direct în document, și crearea de tabele cu număr configurabil de rânduri și coloane.

2.4 Stiluri Personalizate

Pentru a accelera procesul de formatare și a asigura consistența vizuală a documentelor, aplicația permite utilizatorilor să creeze și să salveze stiluri personalizate. Un stil personalizat poate include setări pentru font, dimensiune, culoare și diverse opțiuni de formatare (bold, italic, underline).

Odată create, aceste stiluri pot fi aplicate rapid asupra oricărui text selectat, eliminând necesitatea de a seta manual fiecare proprietate de formatare.

2.5 Partajare și Colaborare

Sistemul oferă multiple modalități de partajare a documentelor, fiecare adaptată unor scenarii diferite de utilizare.

Partajare prin link public: Fiecare document poate genera un link unic de partajare folosind un UUID. Acest link poate fi partajat cu oricine, iar persoanele care îl accesează pot vizualiza documentul fără a fi necesară autentificarea. Proprietarul documentului poate activa sau dezactiva editarea publică, controlând astfel dacă persoanele cu link-ul pot doar vizualiza sau pot și modifica documentul.

Partajare prin email: Utilizatorii pot trimite link-uri de partajare direct prin email către destinatari specifici.

Export PDF: Documentele pot fi descărcate în format PDF, păstrând formatarea și conținutul. Această funcționalitate folosește biblioteca xhtml2pdf pentru a converti conținutul HTML al documentului într-un fișier PDF.

Colaboratori dedicați: Proprietarul poate adăuga colaboratori specifici prin adresa lor de email. Colaboratorii primesc drepturi de editare pe document și sunt notificați automat prin email când sunt adăugați. Proprietarul poate gestiona lista de colaboratori, având posibilitatea să elimine accesul oricând.

2.6 Colaborare în Timp Real

Una dintre caracteristicile principale ale aplicației este editarea colaborativă în timp real. Când mai mulți utilizatori lucrează simultan pe același document, modificările fiecăruia sunt transmise celorlalți prin intermediul conexiunilor WebSocket in timp real.

Sistemul afișează cursorul fiecărui utilizator activ cu numele și culoarea sa unică. Această vizualizare în timp real îmbunătățește coordonarea între colaboratori.

Când un utilizator părăsește sesiunea de editare (închide pagina sau se deconectează), sistemul detectează automat acest eveniment și îl notifică pe ceilalți colaboratori, actualizând lista de utilizatori activi.

2.7 Sistem de Notificări

Aplicația implementează un sistem de notificări care informează utilizatorii despre modificările importante ale documentelor lor.

Pentru a evita spam-ul, sistemul nu trimite notificări pentru auto-salvările automate, care se întâmplă frecvent. În schimb, aceste evenimente sunt înregistrate doar în consolă.

Toate notificările prin email conțin un link direct către documentul modificat.

2.8 Diagrama Use Case

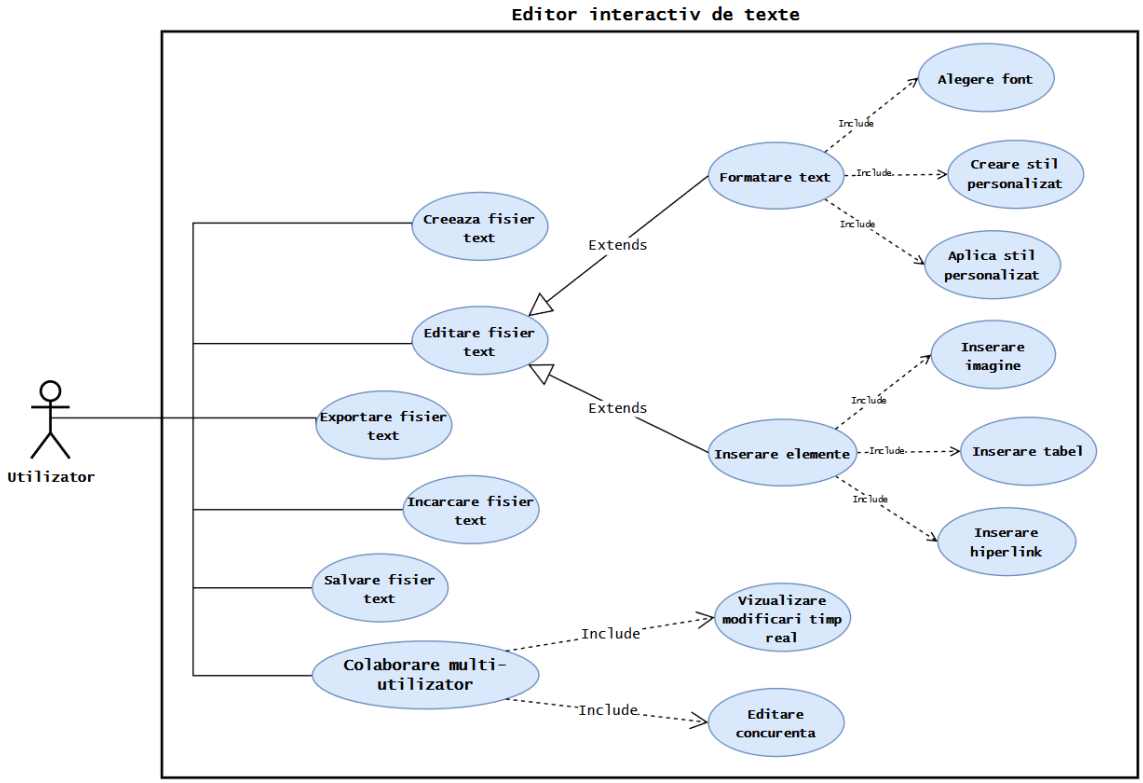

Figura 1: Diagrama Use Case - Funcționalități principale ale sistemului

2.9 Diagrame de Secvență

2.9.1 Autentificare

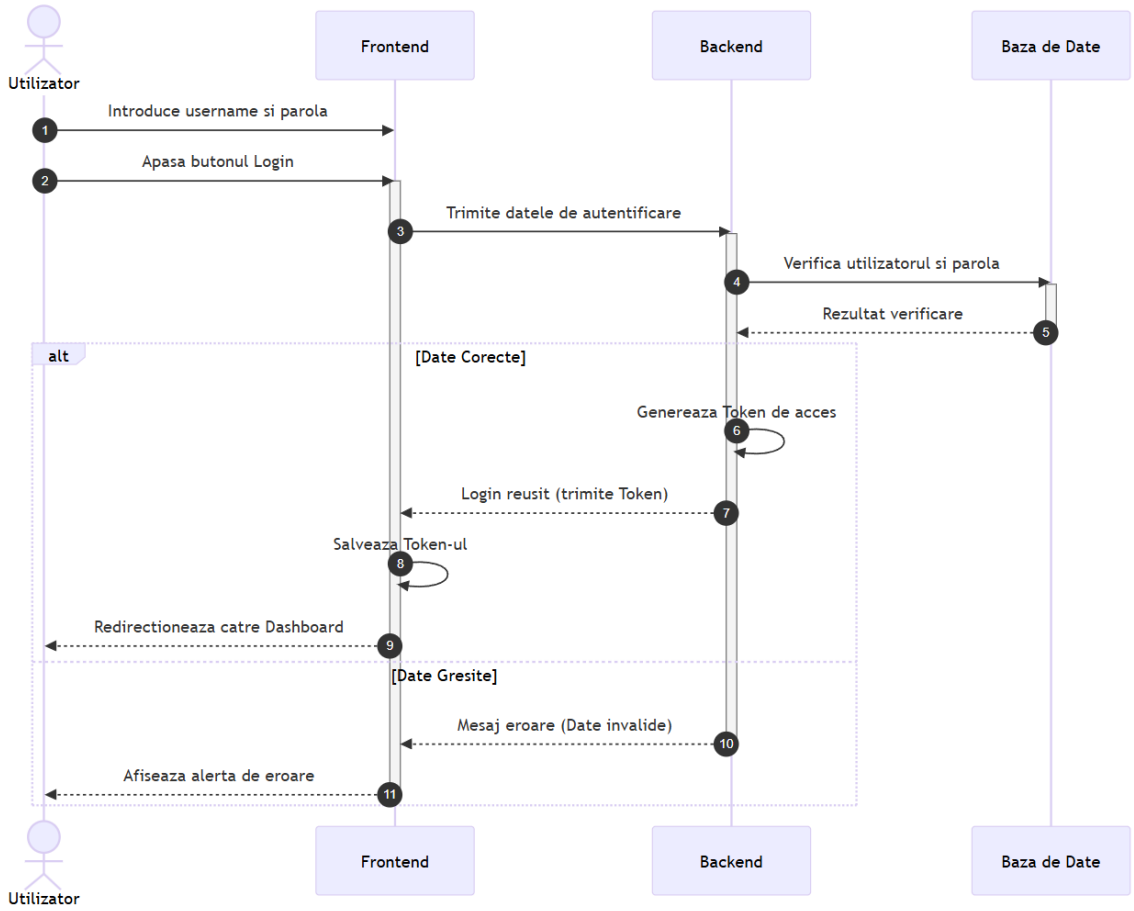

Figura 2: Diagrama de Secvență - Procesul de autentificare - Stanut Denisa

Această diagramă prezintă fluxul de autentificare al utilizatorului, de la introducerea credențialelor până la obținerea token-ului JWT și accesarea documentelor.

2.9.2 Salvare Document

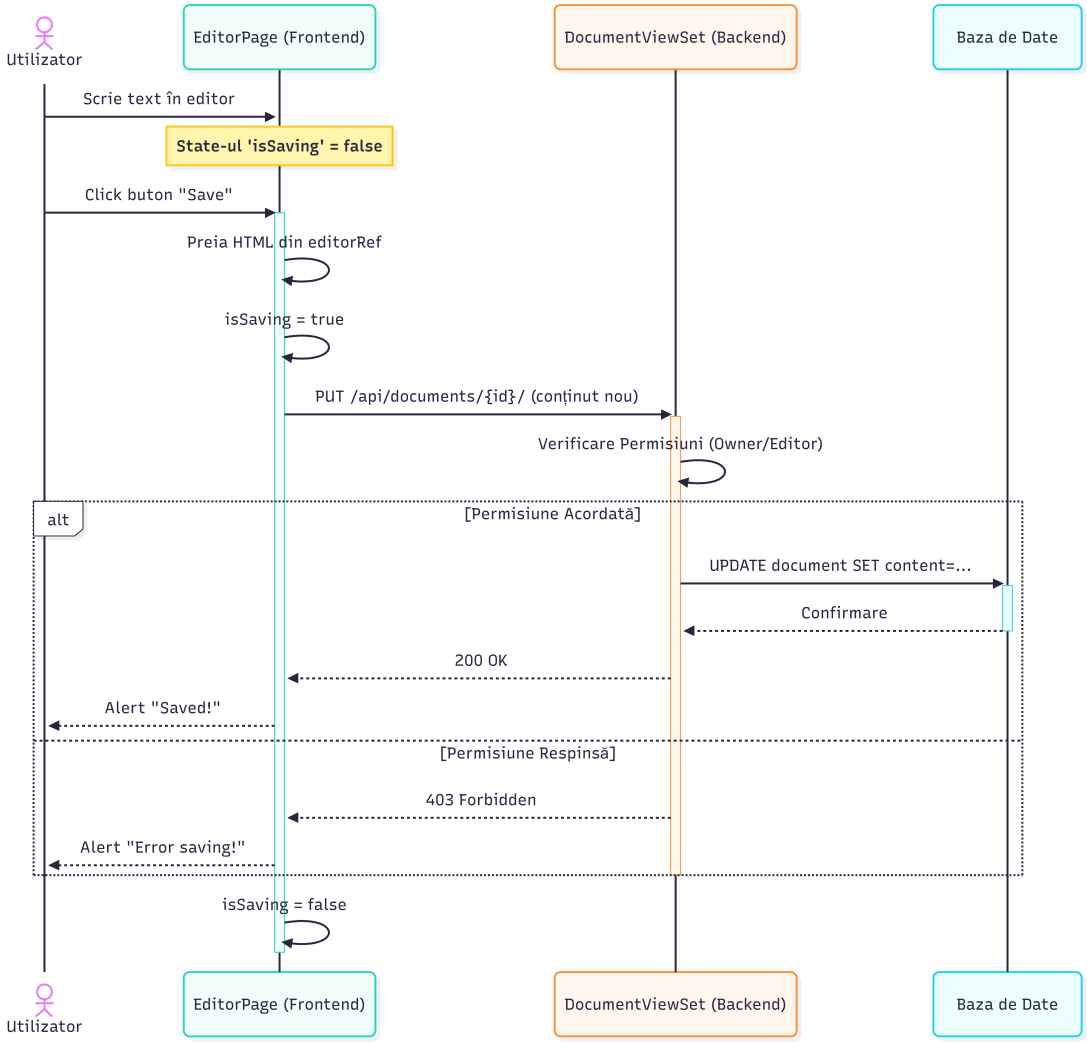

Figura 3: Diagrama de Secvență - Salvarea documentului - Sescu Diana

Fluxul de salvare a documentului include validarea permisiunilor, actualizarea în baza de date și notificarea observatorilor.

2.9.3 Autosave

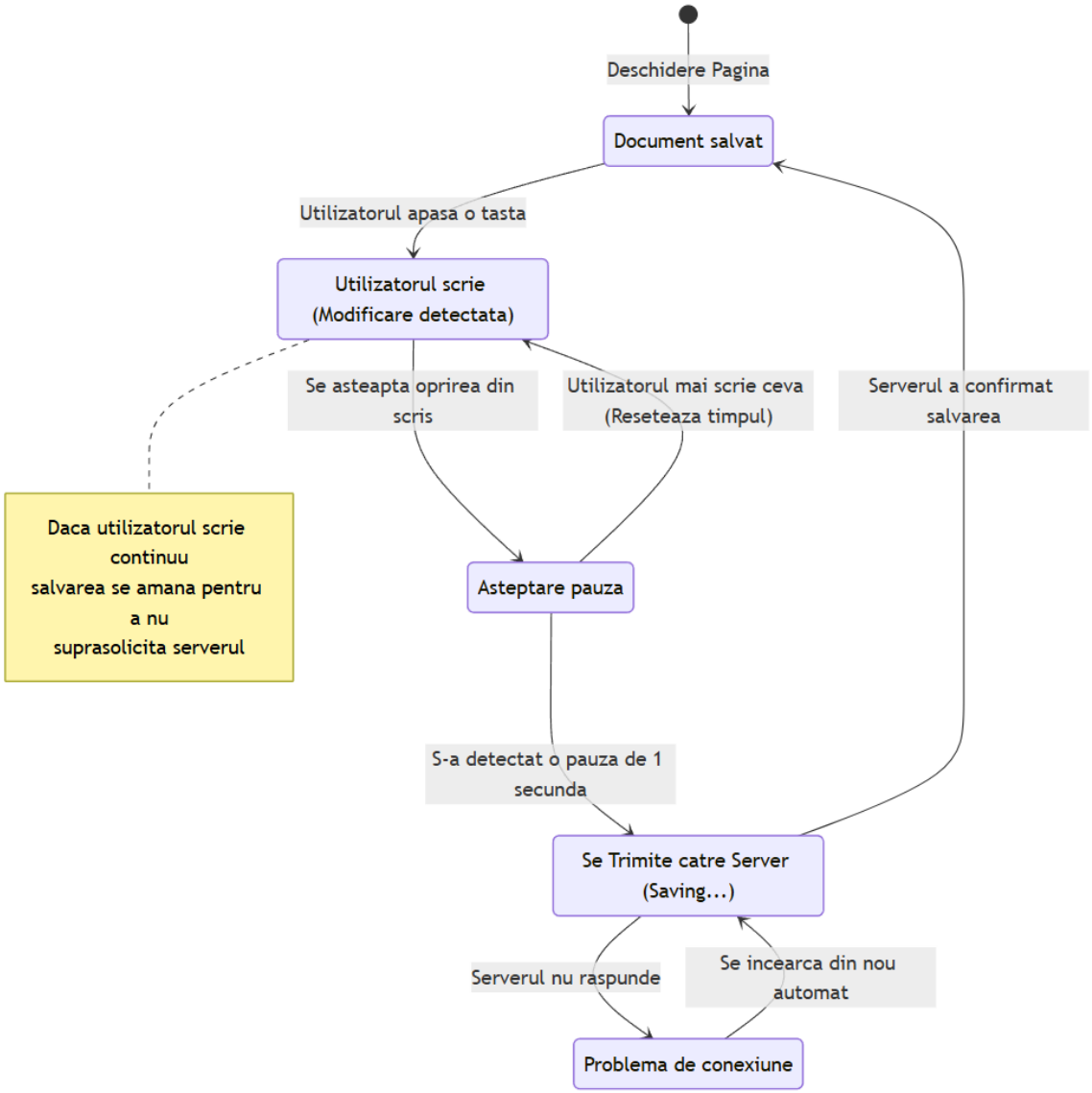

Figura 4: Diagrama de Stări - Procesul de auto-save - Stanut Denisa

Diagrama de stări pentru funcționalitatea de auto-salvare arată tranziția între stările de editare, așteptare și salvare automată.

2.10 Diagrame de Comunicare

2.10.1 Partajare prin Email

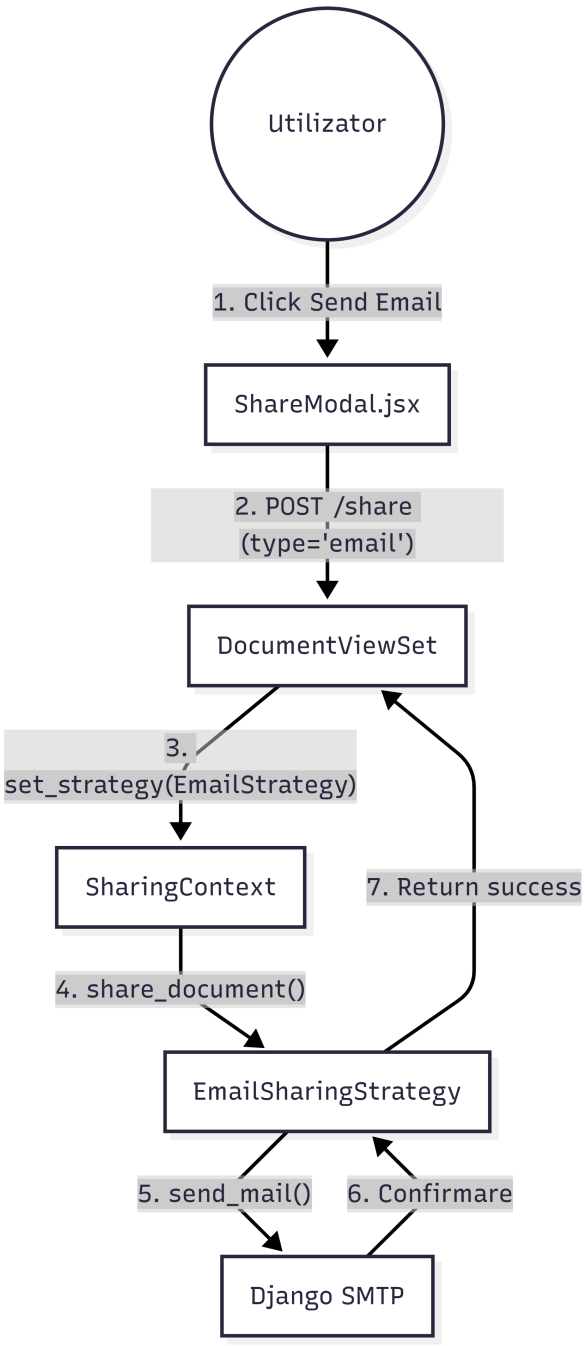

Figura 5: Diagrama de Comunicare - Partajare document prin email - Sescu Diana

Această diagramă ilustrează interacțiunile dintre componente în procesul de partajare a documentului prin email.

2.10.2 Export PDF

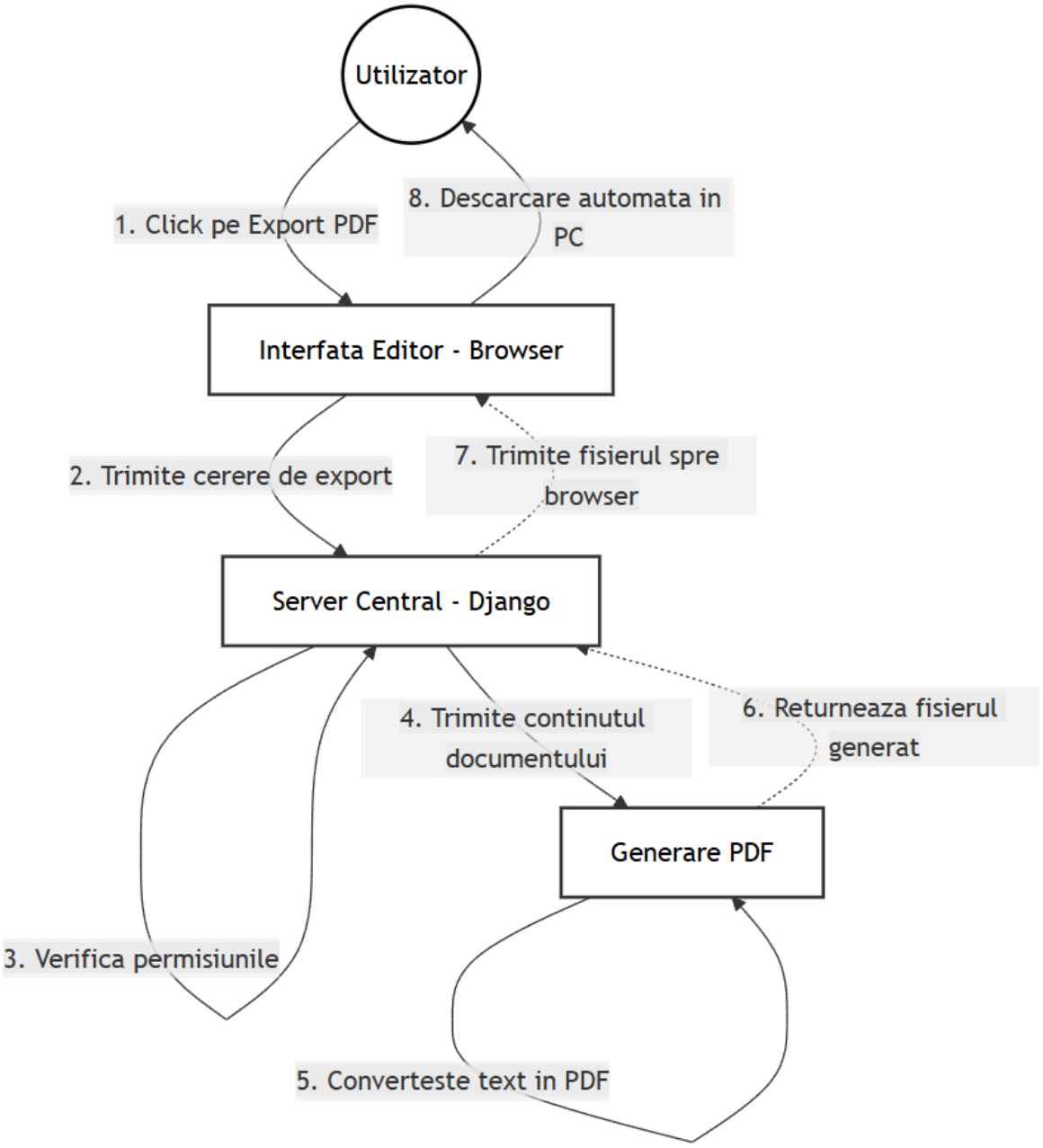

Figura 6: Diagrama de Comunicare - Export document în format PDF - Stanut Denisa

Fluxul de export PDF prezintă comunicarea între view, strategy pattern și biblioteca xhtml2pdf.

2.11 Diagramă de Activitate - Logica de Permisiuni

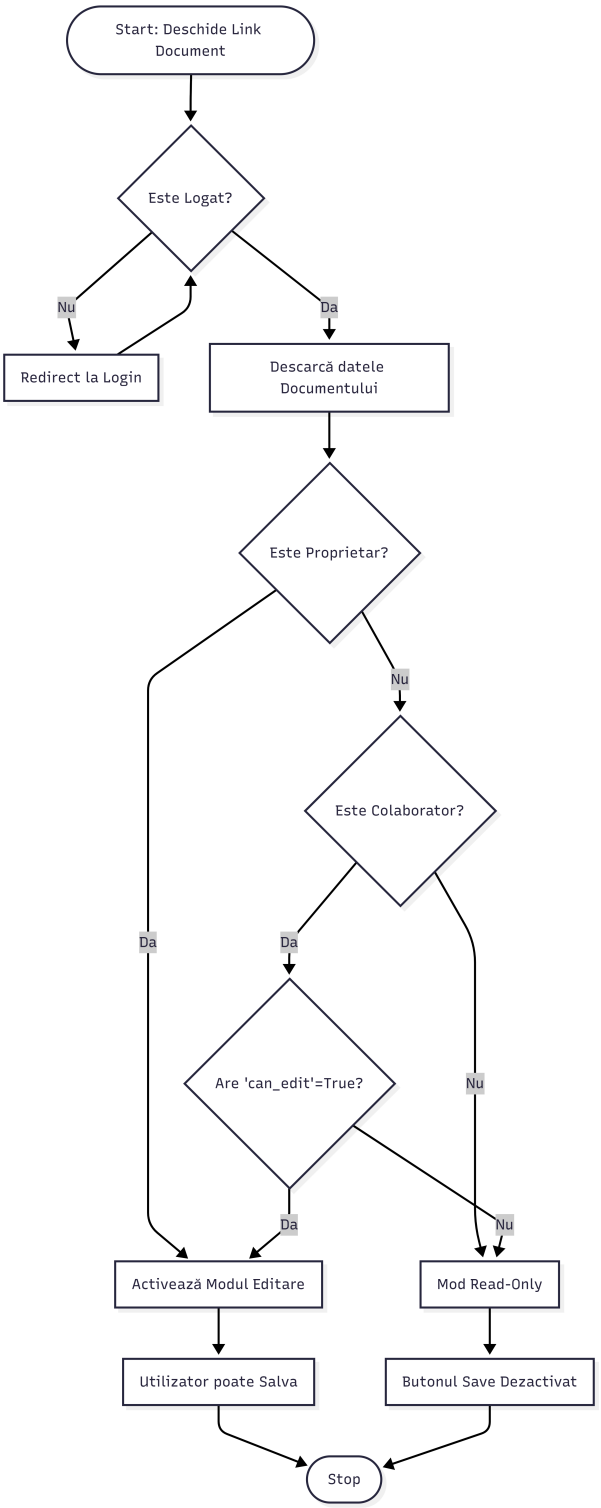

Figura 7: Diagrama de Activitate - Verificarea permisiunilor de editare - Sescu Diana

Această diagramă prezintă logica de verificare a permisiunilor pentru editarea documentelor, incluzând cazurile pentru proprietari, colaboratori și utilizatori cu link public.

3 Cerințe Non-Funcționale

3.1 Securitate

Securitatea este o prioritate în design-ul aplicației. Autentificarea se realizează prin token-uri JWT reducând riscul de utilizare neautorizată.

Parolele utilizatorilor nu sunt niciodată stocate în format plain text, fiind protejate prin algoritmul de hashing PBKDF2t, care este rezistent la atacuri.

Comunicarea WebSocket necesită un token JWT valid pentru autentificare, prevenind accesul neautorizat la fluxurile de date în timp real. Sistemul validează permisiunile utilizatorilor înaintea oricărei operațiuni pe documente, asigurându-se că doar utilizatorii autorizați pot vizualiza sau modifica conținutul.

3.2 Usability și Compatibilitate

Aplicația este responsive și funcționează corect pe diferite dimensiuni de ecran, de la desktop la tablete și mobile. Cursorii colaboratorilor sunt vizibili și ușor de identificat prin culoare și nume, facilitând înțelegerea modificărilor în timp real.

Din perspectiva compatibilității, aplicația funcționează pe toate browserele (Chrome, Firefox, Safari, Edge) și necesită Python 3.8+ pentru backend și React 18+ pentru frontend.

4 Design Patterns

4.1 Observer Pattern - Sistem de Notificări

4.1.1 Implementat de: Stanut Denisa

4.1.2 Problema Identificată

În contextul unei aplicații de editare colaborativă, era esențială existența unui mecanism prin care utilizatorii să fie notificați automat despre modificările importante ale documentelor lor. Provocarea constă în faptul că sistemul trebuie să suporte multiple tipuri de notificări fără ca logica de business a documentelor să fie strâns cuplată cu implementările specifice de notificare.

O abordare naivă ar fi condus la un cod plin de verificări condiționale pentru fiecare tip de notificare, făcând sistemul dificil de extins. De exemplu, dacă am dori să adăugăm notificări prin SMS în viitor, ar fi necesară modificarea codului existent al clasei Document, încălcând principiul Open/Closed.

Mai mult, diferite scenarii necesită notificări diferite: salvările manuale ar trebui să genereze email-uri către colaboratori, dar auto-salvările frecvente ar trebui să fie înregistrate doar în consolă pentru debugging.

4.1.3 Rezolvarea Problemei prin Observer Pattern

Observer Pattern oferă o soluție la aceasta problema prin stabilirea unei relații one-to-many între subject (documentul) și observatori (sistemele de notificare). Când documentul se modifică, acesta notifică automat toți observatorii săi, fără a cunoaște detaliile implementării lor.

Acest pattern aduce următoarele beneficii:

Decuplare totală: Clasa Document nu conține logică specifică de notificare. Ea doar informează observatorii că s-a produs o modificare, lăsând fiecărui observator libertatea de a decide cum să proceseze această informație.

Extensibilitate: Noi tipuri de observatori pot fi adăugati în viitor fără a modifica codul existent al documentului.

Separarea responsabilităților: Fiecare observator gestionează propriul tip de notificare în mod independent.

4.1.4 Implementare

Implementarea se bazează pe trei componente principale:

1. Interfața Observer (DocumentObserver)

Se definește o clasă abstractă. Metoda abstractă `update()` primește informații despre document, utilizatorul care a făcut modificarea, tipul modificării și detalii suplimentare.

2. Observatorii Concreți

EmailNotify implementează logica de trimitere email pentru notificări importante. Observatorul verifică tipul modificării și ignoră auto-salvările pentru a evita spam-ul. Obține lista de destinatari (proprietar și colaboratori, excluzându-l pe cel care a făcut modificarea) și construiește mesajul email cu informații relevante și link către document.

ConsoleNotifier este folosit pentru debugging și monitorizare, afișând în consolă toate modificările, inclusiv auto-salvările. Acest observator este util în timpul dezvoltării și pentru diagnosticarea problemelor în producție.

3. Subject-ul (DocumentObservable)

Clasa DocumentObservable gestionează colecția de observatori și oferă metode pentru:

- `attach_observer()`: adaugă un nou observator în listă
- `detach_observer()`: elimină un observator
- `notify_observers()`: iterează prin toți observatorii și apelează metoda lor `update()`
- `clear_observers()`

4. Integrarea în Model

Clasa Document moștenește DocumentObservable

5. Utilizarea în View-uri

În view-urile Django, observatorii sunt atașați dinamic în funcție de context. Pentru salvări manuale se atașează atât ConsoleNotifier cât și EmailNotify, în timp ce pentru auto-save se atașează doar ConsoleNotifier.

4.1.5 Avantaje și Beneficii

Observer pattern-ul aduce multiple avantaje practice în contextul acestei aplicații. În primul rând, sistemul devine extensibil: adăugarea unui nou tip de notificare necesită doar crearea unei noi clase care implementează interfața DocumentObserver, fără modificarea codului existent.

Modul de testare este îmbunătățit semnificativ deoarece fiecare observator poate fi testat independent.

Flexibilitatea runtime permite adaptarea comportamentului în funcție de context: pentru auto-save se evită spam-ul de email-uri, în timp ce pentru modificări importante se notifică toți colaboratorii.

4.2 Strategy Pattern - Sistem de Partajare

4.2.1 Implementat de: Sescu Diana

4.2.2 Problema Identificată

Aplicația necesită multiple modalități de partajare a documentelor: generare link public, trimitere prin email și export PDF. Fiecare metodă are algoritmi și cerințe diferite, iar combinarea tuturor acestora într-o singură metodă ar conduce la un cod complex, greu de întreținut.

Problema specifică constă în faptul că:

Logica de partajare prin link este simplă - generează un UUID și construiește un URL. Partajarea prin email necesită validare de email, construirea unui mesaj personalizat și integrare cu serviciul SMTP. Export-ul PDF implică transformarea HTML-ului documentului într-un format PDF, gestionarea resurselor embedded (imagini) și setarea header-elor HTTP corecte.

O implementare clasică cu if-else statements ar încălca principiul Open/Closed: adăugarea unei noi metode de partajare (de exemplu, export Word sau partajare prin API-uri externe) ar necesita modificarea codului existent, crescând riscul de introducere a bug-urilor.

Mai mult, testarea ar fi dificilă deoarece toate algoritmii ar fi într-o singură metodă, necesitând setup complex pentru fiecare caz de test.

4.2.3 Rezolvarea Problemei prin Strategy Pattern

Strategy Pattern rezolvă această problemă prin încapsularea fiecărui algoritm de partajare într-o clasă separată, permițând schimbarea dinamică a comportamentului la runtime fără modificarea contextului.

Acest pattern aduce următoarele beneficii:

Încapsulare: Fiecare algoritm de partajare este izolat în propria clasă, cu toate dependențele și logica necesară. Aceasta simplifică înțelegerea și modificarea fiecărei strategii independent.

Interschimbabilitate: Strategiile pot fi schimbate transparent la runtime în funcție de cerințele utilizatorului. Contextul nu trebuie să cunoască detaliile implementării, doar interfața comună.

Extensibilitate: Noi strategii pot fi adăugate prin simpla creare a unei clase noi care implementează interfața SharingStrategy, fără modificarea codului existent.

Testabilitate: Fiecare strategie poate fi testată izolat, cu mock-uri pentru dependențele externe (serviciu email, bibliotecă PDF).

4.2.4 Implementare

Implementarea Strategy Pattern pentru sistemul de partajare se bazează pe următoarea structură:

1. Interfața Strategy (SharingStrategy)

Se definește o clasă abstractă cu metoda `share()` care primește documentul, request-ul HTTP și parametri suplimentari. Această interfață garantează că toate strategiile concrete vor oferi același contract.

2. Strategiile Concrete

LinkSharingStrategy gestionează partajarea prin link public. Strategia verifică dacă documentul are deja un token de partajare (UUID) sau generează unul nou. Construiește apoi link-ul complet folosind domain-ul din request și returnează un dicționar cu link-ul generat.

EmailSharingStrategy implementează partajarea prin email. Validează adresa de email a destinatarului, generează link-ul de partajare, construiește mesajul email cu informații despre document și utilizatorul care partajează, și folosește serviciul SMTP pentru trimitere. Gestionează excepțiile și returnează status-ul operațiunii.

PDFSharingStrategy realizează export-ul în PDF. Transformă conținutul HTML al documentului într-un șablon complet cu stiluri CSS pentru formatare (fonturi, tabele, imagini). Folosește biblioteca xhtml2pdf pentru generarea PDF-ului, gestionează resursele embedded (imagini) prin funcția `fetch_resources`, și setează header-ele HTTP pentru download automat al fișierului.

3. Contextul (SharingContext)

Clasa SharingContext menține o referință către strategia curentă și oferă metode pentru:

- `set_strategy()`: setează strategia care va fi utilizată
- `share_document()`: execută strategia curentă, delegând operația de partajare

Contextul validează că o strategie a fost setată înainte de execuție, aruncând o excepție descriptivă dacă nu.

4. Utilizarea în View-uri

În view-ul Django, contextul este creat și strategia este selectată dinamic în funcție de parametrul `type` din request. Pentru `type='email'` se setează EmailSharingStrategy, pentru `type='pdf'` se setează PDFSharingStrategy, iar implicit se folosește LinkSharingStrategy.

După setarea strategiei, se apelează `share_document()` care execută algoritmul specific și returnează rezultatul către client.

4.2.5 Avantaje și Beneficii

Strategy Pattern-ul oferă multiple avantaje practice pentru sistemul de partajare. Extensibilitatea este evidentă: adăugarea unor noi metode de partajare (de exemplu, export în format Word sau integrare cu Google Drive) necesită doar crearea unei noi clase de strategie, fără a modifica codul existent.

Fiecare strategie poate fi testată complet izolat pentru dependențele externe (serviciul SMTP pentru EmailSharingStrategy, biblioteca xhtml2pdf pentru PDFSharingStrategy). Acest lucru simplifică semnificativ procesul de testare și debugging.

Separarea responsabilităților este clară: fiecare strategie gestionează propriile dependențe și logică de eroare. De exemplu, EmailSharingStrategy gestionează erorile SMTP, în timp ce PDFSharingStrategy gestionează erorile de conversie HTML-PDF.

Flexibilitatea runtime permite oferirea unor opțiuni diferite utilizatorilor în funcție de context. De exemplu, export-ul PDF poate fi oferit doar pentru documente finalizate, în timp ce partajarea prin link este disponibilă întotdeauna.

În plus, pattern-ul facilitează îmbunătățiri viitoare precum: adăugarea de strategii compuse (de exemplu, EmailWithPDFStrategy care trimite documentul ca attachment PDF), implementarea de strategii cu configurări diferite (de exemplu, PDFWithWatermarkStrategy pentru documente confidențiale), sau crearea de strategii adaptate pentru anumite tipuri de utilizatori (de exemplu, PremiumSharingStrategy cu opțiuni suplimentare).

5 Concluzii

Acest proiect demonstrează implementarea practică a design pattern-urilor fundamentale în contextul unei aplicații web moderne. Observer Pattern și Strategy Pattern au fost integrate în arhitectura aplicației, rezolvând probleme reale de extensibilitate și mentenabilitate.

Aplicația oferă o experiență de editare colaborativă, combinând funcționalități de formatare text cu capacități avansate de partajare și colaborare în timp real.

Tehnologiile alese (Django pentru backend și React pentru frontend) s-au dovedit potrivite pentru cerințele proiectului, oferind un echilibru bun între performanță, securitate și ușurință în dezvoltare.

Referințe

[1] Refactoring.Guru - *Design Patterns* - `https://refactoring.guru/design-patterns`

[2] Mermaid Editor - *Online Diagram Editor* - `https://mermaid.live/`

[3] Django Channels - *Channels Documentation* - `https://channels.readthedocs.io/`

[4] Meta Open Source - *React Documentation* - `https://react.dev/`

[5] Mozilla Developer Network - *WebSocket API* - `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket`