O imagine care conține text, Font, captură de ecran, siglă

Descriere generată automat

**Proiect Inginerie Software. Clinica medicala**

Toma-Nan George-Alexandru

Grupa 30238

**Cuprins**

1. **Introducere**
2. **Descriere**
3. **Functionalitati**
4. **Limbaje de programare**
5. **Framework-uri**
6. **Diagrame**
7. **Design Pattern-uri**
8. **Cod**
9. **Ghid rulare**

**Introducere**

Această aplicație este un sistem de gestionare a programărilor medicale, destinat pentru utilizarea de către utilizatori (pacienți) și medici, care permite programarea întâlnirilor între aceștia. Aplicația oferă utilizatorilor posibilitatea de a-și crea conturi, de a se conecta și de a programa întâlniri cu medici compatibili, pe baza bolii pe care o au. De asemenea, administratorii pot gestiona compatibilitățile dintre medici și pacienți, pot crea și edita informațiile despre pacienti si doctori și pot vizualiza programările existente.

**Descriere**

Aplicația este un sistem bazat pe platforma web, care permite gestionarea și programarea întâlnirilor între pacienți și medici. Acesta are mai multe componente, inclusiv:

* **Pagina de înregistrare**: Permite utilizatorilor să își creeze conturi, completând detalii precum email, nume, tipul bolii și vârsta.
* **Pagina de login**: Permite utilizatorilor autentificarea în aplicație.
* **Pagina de programări**: Permite utilizatorilor să vizualizeze programările disponibile și să facă programări cu medici compatibili.
* **Pagina de administrare**: Permite administratorilor să vizualizeze, să editeze și să adauge informații despre medici și pacienți, inclusiv compatibilitățile între aceștia.

**Functionalitati**

Aplicația oferă următoarele funcționalități:

* **Crearea unui cont pentru utilizatori**: Permite pacienților să se înregistreze și să creeze un cont.
* **Autentificare utilizatori**: Permite pacienților să se logheze în aplicație folosind emailul și parola.
* **Programări**: Permite pacienților să vizualizeze medicii disponibili și să își facă programări în funcție de disponibilitatea acestora și compatibilitatea cu boala pacientului.
* **Gestionarea compatibilităților**: Permite administratorilor să adauge sau să modifice compatibilitățile dintre pacienți și medici pe baza bolii pacientului și specializării medicului.
* **Administrare utilizatori și medici**: Permite administratorilor să adauge, editeze și șteargă medici și pacienți.

**Limbaje de programare**

Proiectul utilizează următoarele limbaje de programare:

* **JavaScript** (pentru partea de front-end)
* **TypeScript** (pentru Angular, framework-ul de front-end)
* **Java** (pentru partea de back-end)
* **SQL** (pentru gestionarea bazei de date)

**Framework-uri**

Aplicația folosește următoarele frameworkuri și biblioteci:

* **Angular**: Framework de front-end folosit pentru crearea interfeței utilizatorului.
* **Spring Boot**: Framework de back-end folosit pentru crearea API-urilor și gestionarea logicii aplicației.
* **Hibernate**: Folosit pentru gestionarea bazei de date și maparea obiect-relatională (ORM).
* **MySQL**: Baza de date utilizată pentru stocarea datelor.

**Diagrame**

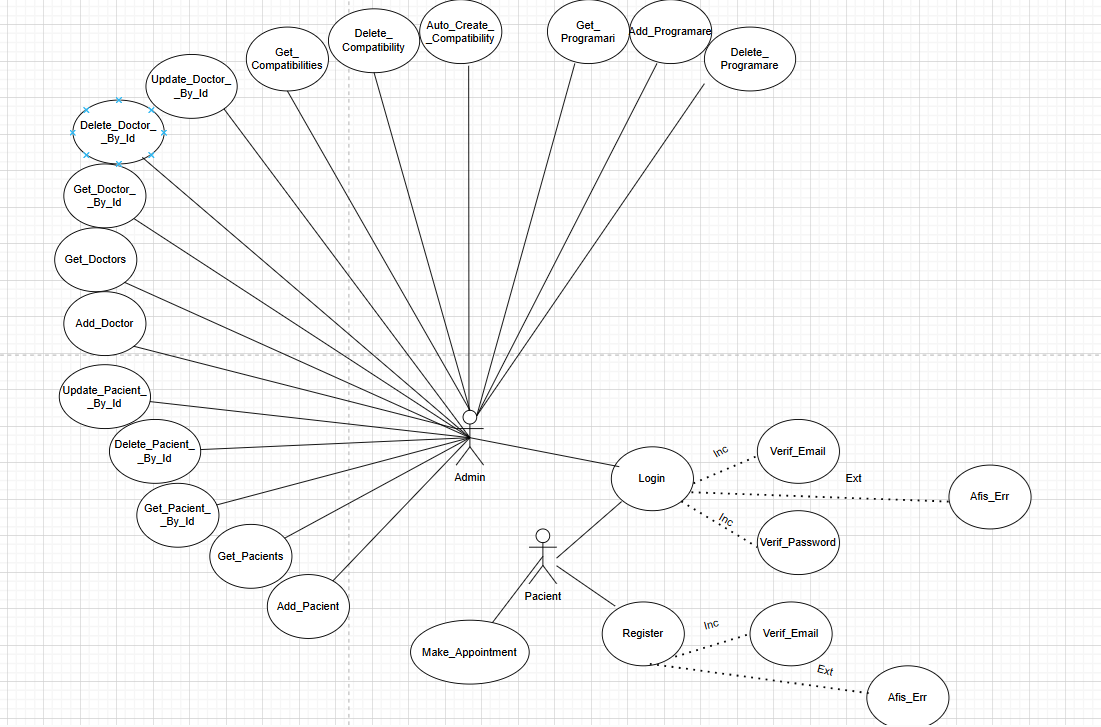
**Diagrama use-case:**

Diagrama Use Case descrie interacțiunile utilizatorilor cu aplicația. Aceasta include actorii (utilizatori, administratori) și cazurile de utilizare, cum ar fi:

* Înregistrarea unui utilizator
* Autentificarea utilizatorilor
* Autentificarea administratorilor
* Vizualizarea programărilor
* Adăugarea și editarea medicilor și pacienților de către administrator

**Diagrama secventa**

O imagine care conține text, captură de ecran, chitanță, diagramă

Descriere generată automat

Diagrama de secvență descrie procesul de autentificare a unui utilizator într-o aplicație. Aceasta evidențiază interacțiunile dintre actorul **User**, **Controller**, **Authentication Service** și **Database** în cadrul unei cereri de login. Această diagramă de secvență ilustrează procesul tipic de autentificare utilizând un username și o parolă. Dacă parola este validă, utilizatorului i se furnizează un **JWT Token** care îi permite să acceseze resursele protejate ale aplicației. Dacă parola este incorectă, utilizatorul primește un mesaj de eroare și nu poate accesa resursele aplicației.

Această abordare a autentificării este frecvent folosită în aplicațiile moderne pentru a asigura securitatea prin tokenuri și pentru a permite un flux de autentificare fără a necesita gestionarea directă a sesiunilor de către server.

**Diagrama clase**

Diagrama de clase prezintă structura claselor utilizate în aplicație și relațiile dintre ele, cum ar fi:

* User (cu atribute ca id, email, nume, etc.)
* Doctor (cu atribute ca id, specializare, ora\_start, ora\_finish, etc.)
* Programari (care face legătura între User și Doctor).

**Diagrama baza de date**

Diagrama bazei de date prezintă structura tabelelor din MySQL, relațiile dintre ele, și câmpurile fiecărei tabele. De exemplu:

* **Tabela users**: conține informațiile despre utilizatori.
* **Tabela doctors**: conține informațiile despre medici.
* **Tabela compatibilities**: leagă utilizatorii de medici pe baza bolii.
* **Tabela programari**: leagă utilizatorii de programările lor.

Fiecare tabela are un id, cele 2 tabele pentru user(pacienti) si doctori au informatii specifice, tabela compatibilities are un user si un doctor pentru a face legatura intre ei, iar tabela programari mai are campurile zi\_lucru si ora, zi\_lucru reprezentand ziua in care lucreaza doctorul, iar ora fiind ora programarii care trebuie sa fie intre ora de inceput si ora de start a doctorului.

**Design Pattern-uri**

**Design pattern-urile utilizate în acest proiect includ:**

* **Singleton**: Folosit pentru gestionarea instanței unei clase, cum ar fi UserService, care gestionează utilizatorul curent.
* **Factory**: Folosit pentru crearea obiectelor de tip Doctor, Programare, etc., în backend.
* **Observer**: Folosit în Angular pentru a urmări modificările datelor și a actualiza UI-ul în funcție de acestea.
* **MVC**: Folosit pentru a separa aplicațiile în trei componente principale:

**Model**: Reprezintă logica de afaceri a aplicației și datele.

**View**: Reprezintă interfața utilizatorului, adică paginile HTML și componentele Angular care sunt responsabile pentru a prezenta datele utilizatorului.

**Controller**: Este componenta care intermediază între Model și View. El primește cereri din partea utilizatorului și acționează asupra Modelului pentru a modifica datele sau pentru a returna datele solicitate către View.

* **Design Pattern: DAO (Data Access Object):** Este un design pattern care separă logica de acces la date de logica de afaceri a aplicației. DAO oferă o interfață pentru a accesa datele dintr-o bază de date sau orice altă sursă de date, permițând astfel modificarea sursei de date fără a afecta restul aplicației. Acesta ajută la centralizarea accesului la date și oferă o modalitate mai clară și mai eficientă de a lucra cu datele persistente.

**Unde sunt folosite Design Pattern:**

* **Singleton**: Utilizat în gestionarea autentificării utilizatorilor (ex. UserService).
* **Factory**: Crearea obiectelor precum Programari, Doctor și User în backend.
* **Observer**: Folosit în Angular pentru a gestiona fluxul de date asincron (ex. gestionarea răspunsurilor API și actualizarea UI-ului).
* **MVC**:

**Model:** Reprezentat de entitățile **User**, **Doctor**, **Programari**, etc. Aceste entități sunt gestionate și manipulate de servicii precum **ProgramariService**, **UserService**.

**View:** Toate fișierele de tip HTML care definesc interfața utilizatorului (de exemplu, **register-page.component.html** și **login-page.component.html**).

**Controller:** Componentele Angular care gestionează logica de interacțiune între utilizator și date (de exemplu, **RegisterPageComponent**, **LoginPageComponent**).

* **Design Pattern: DAO (Data Access Object):** Pattern-ul DAO este implementat prin **Repository**-urile Spring, care sunt responsabile pentru gestionarea entităților în baza de date. De exemplu:

**UserRepository** este folosit pentru a accesa și manipula datele despre utilizatori.

**DoctorRepository** se ocupă de gestionarea informațiilor despre doctori.

**ProgramariRepository** este utilizat pentru gestionarea programărilor.

**Cod**

În această secțiune vor fi prezentate fragmente de cod importante, cum ar fi:

* Informatiile despre un pacient:  
  **O imagine care conține text, captură de ecran, Font

  Descriere generată automat**
* Metodele folosite pentru pacienti:

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat  
O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

O imagine care conține text, captură de ecran, Font

Descriere generată automat

* Codul pentru a face legatura cu baza de date

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

* Cod pentru crearea unui utilizator in front-end:

createUser(user: User): Observable<User> {

return this.http.post<User>(`${this.apiUrl}/add`, user);

}

**Ghid rulare**

Pentru a rula aplicația local, urmează pașii următori:

1. Instalarea dependințelor:

 Începe prin a instala Node.js și Angular CLI.

* Rulează npm install pentru a instala pachetele necesare.

1. Pornirea aplicației frontend:

* Mergi la directorul aplicației frontend și rulează ng serve.

1. Pornirea aplicației backend:

 Mergi la directorul aplicației backend și rulează aplicatia.

1. Accesarea aplicației:

* Deschide un browser și accesează http://localhost:4200 pentru aplicația frontend.