

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Problema Transplantului de Rinichi
propusă de

Florentin Emanuel Olariu

Sesiunea: 07, 2020

Coordonator științific

Lect. Dr. Florentin Emanuel Olariu

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Problema Transplantului de Rinichi

Ionuț Dima

Sesiunea: 07, 2020

Coordonator științific

Drd. Colab. Florin Olariu

Cuprins

Introducere	3
Informații generale	3
Tema	3
Motivație	4
Gradul de noutate	5
Capitolul 1 – Descrierea problemei	7
Capitolul 2 – Descrierea soluției	9
Prezentarea aplicației	9
Arhitectura Sistemului	10
Controller	11
Service	12
Repository	12
Model	12
Arhitectura bazei de date	14
Modul de funcționare al aplicației	15
Algoritmi dezvoltati	25
Algoritmul de inserare al intervalelor	25
Algoritmul de spargere al intervalelor	26
Concluziile lucrării	27
Dezvoltare ulterioară	28
Anexe	28
Anexa 1 - Java	28
Anexa 2 - Spring framework	28
Anexa 3 - Beneficiile utilizării framework-ului Spring	29
Anexa 4 - Spring Boot	30
Anexa 5 - Maven	30
Anexa 6 - Orm, JPA, Hibernate	31
Anexa 7 - Adnotari	32
Anexa 8 - JWT	33
Bibliografie	33

Introducere

Informații generale¹

Problema Transplantului de Rinichi sau de organe în general a apărut atunci când transplanturile de organe au devenit posibile datorită evoluției medicinei. Încă de la început găsirea unui donator într-un timp scurt era destul de complicată, multe din acele proceduri de găsire a unui donator compatibil aplicându-se și acum. Și anume: așteptarea pe o listă lungă de așteptare după un donator decedat de cele mai multe ori, iar această așteptare poate duce la probleme mult mai grave ale sănătății.

Dezavantajele metodei clasice de cautare a donatorilor:

1. Imposibilitatea centralizării situației pe o întreagă țară sau stat.
2. Nu pot fi aplicați algoritmi prea complexi de cautare a unui pacient, atâta timp cât datele nu ajung să fie preluate de către un computer.

Tema

Tema aleasă este o platformă creată pentru întreaga regiune a României care să fie ușor și intuitiv de utilizat pentru utilizatorii de orice vârstă și cu o interfață cât mai simplă, astfel toți cei care intră în această aplicație să poată găsi ușor butoanele de înregistrare și de logare.

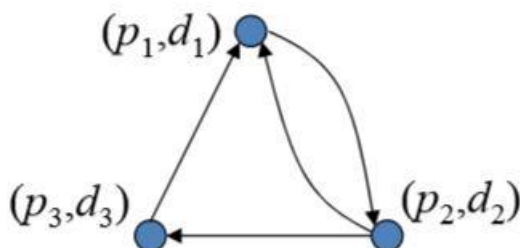
În spate, pe partea de backend proiectul dorește implementarea unui algoritm deja existent și elaborat de o universitate din United Kingdom, Glasgow, cea care a elaborat "Kidney Exchange Problem" folosind în final rezolvarea unei probleme liniare.

Rezolvarea acestei probleme propusă de cei de la Universitatea Glasgow constă în conceptul folosirii unui graf oricentrat care poate ajunge doar până la maxim 3 cicluri, pentru că având un număr mai mare de etape, ar deveni inefficient. Această rezolvare constă în preluarea de date dintr-un mediu de stocare și crearea unui digraf orientat care are ca noduri n perechi pacient–donator incompatibile $\{(p_i, d_i) : 1 \leq i \leq n\}$ și k donatori altruști $\{d_{n+i} : 1 \leq i \leq k\}$. Donatorii altruști sunt acei donatori care vor să ajute din proprie inițiativă, fără a dori să ajute vreun anumit pacient și fără a avea vreun interes.

Donatorii altruști d_{n+i} li se vor atribui pacienți fictivi p_{n+i} care sunt compatibili cu orice donator d_j where $1 \leq j \leq n$. Vom modela problema transplantului de rinichi având un Digraf orientat $D = (V, A)$, unde $V = \{v_1, \dots, v_{n+k}\}$ și v_i corespunde perechii (p_i, d_i) ($1 \leq i \leq n+k$).

În plus $(v_i, v_j) \in A$ dacă și numai dacă d_i este compatibil cu p_j .
 Pentru graful D vom cauta strict ciclurile de maxim 2 sau 3 arce, deoarece tot ce trece de aceasta limită există riscul major în care un pacient să nu mai poată dona un rinichi, tot acel ciclu să nu mai existe.

Aici este un exemplu de un digraf care are un ciclu de doi intern și un ciclu de 3 extern:



1. Fie o listă cu toate ciclurile de lungime 2 sau 3 din Digraful D cu C_1, C_2, \dots, C_m , unde, ciclurile de lungime 2 sunt notate C_1, \dots, C_{n_2} , ciclurile de lungime 3 sunt $C_{n_2+1}, \dots, C_{n_2+n_3}$, și ciclurile de lungime 3 având un ciclu de lungime 2 în interior sunt $C_{n_2+1}, \dots, C_{n_2+n_3}$ (deci $m = n_2 + n_3$);

2. Fie x un vector binar de lungime $(m + k) \times 1$ variabilele x_1, x_2, \dots, x_{m+k} , pentru $1 \leq i \leq m$, $x_i = 1$ dacă și numai dacă C_i devine soluție optimă, și $1 \leq i \leq k$, $x_{m+i} = 1$ dacă și numai dacă donatorii altruiști d_{n+i} nu au pereche;

3. Fie A o matrice de lungime $(n + 2k) \times (m + k)$ $\{-1, 0, 1\}$ -fiind valorile matricii, pentru început având numai valori de 0 și după cum urmează:

(a) pentru $1 \leq i \leq n$ și $1 \leq j \leq m$, $A_{i,j} = 1$ dacă și numai dacă C_j conține d_i ;

(b) pentru fiecare i ($1 \leq i \leq k$), în liniile $n + 2i - 1$ și $n + 2i$:

i. pentru $1 \leq j \leq m$, $A_{n+2i-1,j} = 1$ dacă și numai dacă C_j conține d_{n+i} , și pentru $1 \leq j \leq k$, $A_{n+2i-1,m+j} = 1$ dacă și numai dacă $i = j$;

ii. pentru $1 \leq j \leq m$, $A_{n+2i,j} = -1$ dacă și numai dacă C_j conține d_{n+i} , și pentru $1 \leq j \leq k$, $A_{n+2i,m+j} = -1$ dacă și numai dacă $i = j$;

4. Fie b un vector de lungime $(n + 2k) \times 1$ unde :

(a) pentru fiecare i ($1 \leq i \leq n$), $b_i = 1$;

(b) pentru fiecare i ($1 \leq i \leq k$) $b_{n+2i-1} = 1$ și $b_{n+2i} = -1$;

5. Fie c un vector de lungime $1 \times (m + k)$ un vector al coeficienților criteriilor ciclurilor, poate fi considerat un criteriu lungimea fiecărui ciclu;

6. Se calculează $\max(cx)$ astfel încât $Ax \leq b$.

Capitolul 1 – Descrierea problemei

Motivație

Motivatia crearii acestui proiect a venit în faptul ca nu exista încă o astfel de platforma pusă în funcțiune pe regiunea Româiei. Acest proiect poate ajuta oameni, chiar putem spune ca poate ajunge sa salveze vietii daca va reusi sa gasească soluții în situații critice. Prin crearea lui poti ajunge sa ajuti o multime de oameni.

Gradul de noutate

Gradul de noutate este unul destul de mare avand in vedere gradul la care medicina de dezvoltă, si este evident ca nu peste mulți ani vom ajunge sa folosim aplicatii de genul asta din plin pentru ieficientizarea sistemului medical.

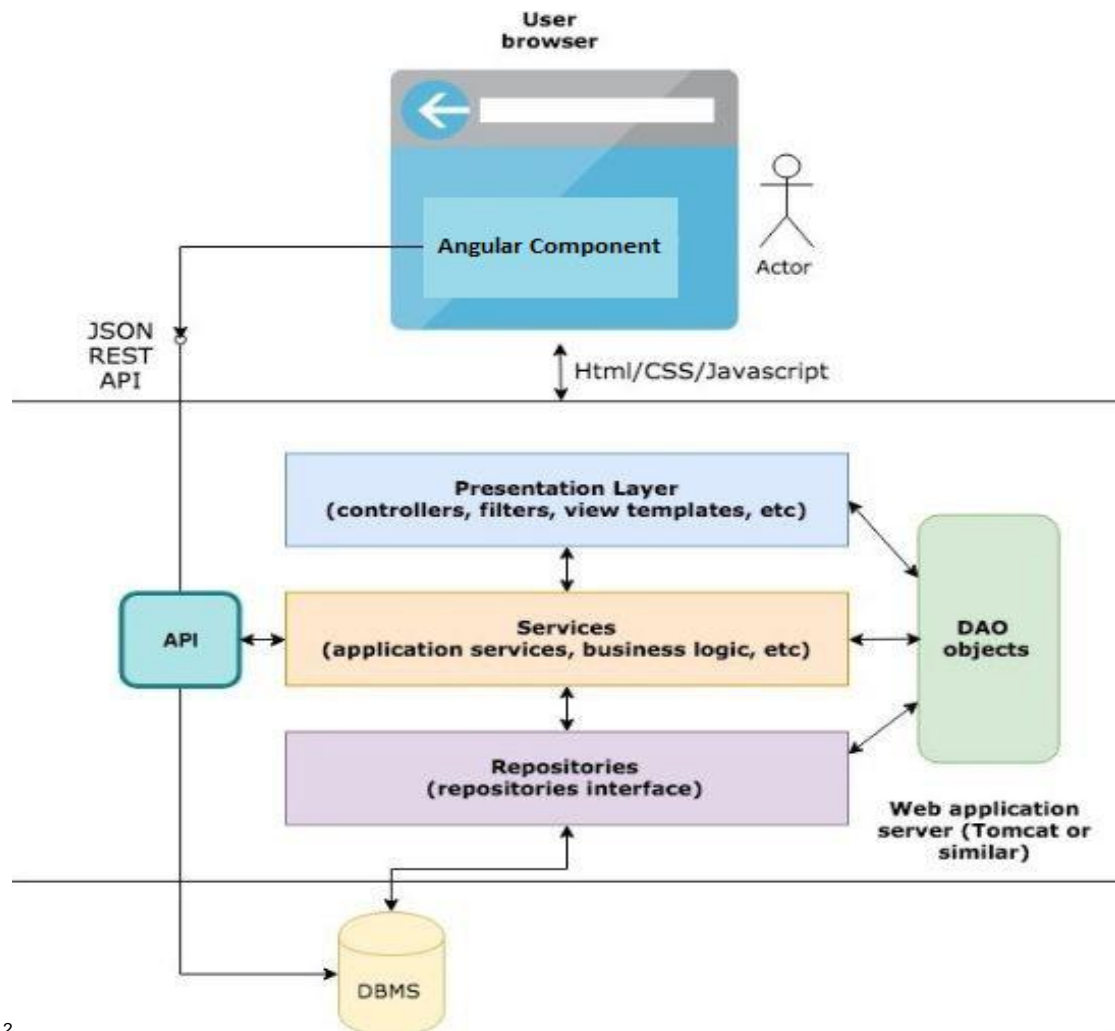
Capitolul 2 – Descrierea soluției

Prezentarea aplicației

Această lucrare are ca scop crearea unei aplicații web având numele ”Problema Transplantului de Rinichi” pentru a deservi bolnavilor de rinichi și donatorilor săi de pe teritoriul României. Ca scop principal atât pacienți care au nevoie de un rinichi, cât și donatorii dispuși să ofere un rinichi unui pacient se pot înscrie pe această aplicație și vor intra într-un proces de creare a perechilor pacient-donator compatibile urmărindu-se creșterea ratei de transplant de rinichi la un număr cât mai mare. Pe întreg parcursul dezvoltării aplicației s-a urmărit simplificarea și ușurarea utilizării aplicației de către utilizatori astfel punându-se la dispoziție butoane accesibile pentru toate operațiunile existente utilizatorului. În prezent utilizatorul are posibilitatea înregistrării ca pacient sau ca donator, dar și a autentificării folosind doar o simplă pagină care nu necesită un reîncărcarea ei la fiecare apăsare de buton, ci doar un simplu click pe secțiunea care îi interesează. Implementarea acestor obiective a fost posibilă prin integrarea conceptelor și tehnologiilor prezentate la sfârșitul documentației în partea de Anexe. Prin urmare în ceea ce privește partea de server aplicația utilizează Spring ca framework peste limbajul Java, a fost ales acest framework pentru realizarea configurărilor într-un mod rapid și ușor și pentru lăsarea în spate a detaliilor de care nu a fost nevoie, Maven(anexa 7) pentru realizarea aducerii dependințelor externe într-un mod rapid și ușor, iar conceptul JWT(anexa 11) pentru autorizarea interacțiunilor și pentru autentificare. Pe partea de client, aplicația folosește Angular ca librărie peste JavaScript având multe componente din librăria Angular Material. Sistemul de baze de date este manageriat de către H2 Database Engine.

Arhitectura Sistemului

Aplicația este alcătuită din două servere, unul pentru frontend altul pentru backend. Serverul de backend are o arhitectură de tip REST, acesta expune un endpoint care este folosit de serverul de frontend specific suportului oferit de limbajul TypeScript prin framework-ul Angular. Flow-ul prin care trebuie să treacă un request pentru a fi procesat de serverul de backend este descris în diagrama de mai jos.



2

2

<https://www.google.ro/search?q=spring+angular+architecture>

În continuare este prezentat în detaliu procesare unui requeste de la client, aceasta se face în felul următor:

1. Serverul de frontend trimite la un endpoint un request, acesta fiind preluat de un controller;
2. Un anumit controller aparținând server-ului de backend apelează unul sau mai multe servicii cu scopul de a procesa informația primită.
3. Serviciul sau serviciile apelate de catre controller vor folosi Repository-urile corelate cu tabelele existente din baza de date pentru a modifica, scrie sau citi informatii din baza de date respectand o anumite logica.
4. Rolul principal al unui Repository este acela de a sustine legatura dintre tabelele din baza de date si modelele existente si create in mediul de lucru, in cazul de fata in limbajul Java.
5. Transferul de informații dintre browser si serverul de backend se face prin intermediul unor obiecte de tip JSON care au un model atât pe serverul de backend cat si pe serverul de frontend.

Controller

Controller-ul primește date de la serverul de frontend printr-un apel de tip Http Client în format șir de string-uri ce sunt trimise unui endpoint expus de catre controller, acesta preia acele date și manipuland obiectele definite în categoria "model" obține informațiile, le proceseaza si apoi trimite raspunsul în format Json.

Orice obiect/clasa definită în limbajul Java care are adnotarea @Controller este luată în considerare de framework-ul Spring ca fiind un controller. Un Controller decide ce fel de cerere HTTP accepta(GET, POST, PUT, DELETE).

Legatura dintre controller și un URL este facuta prin adnotarea @RequestMapping. Aceasta este folosită în cele mai multe cazuri la nivelul clasei și contribuie la uniunea dintre clasa și calea unde vor fi trimise apelurile.

În imaginea de mai jos este prezentat un exemplu din aplicație care cuprinde o metodă, cea de autentificare, în care adnotarea @RequestMapping face legatura dintre metoda createAuthenticationToken și calea "/authenticate" a serverului de backend, calea fiind dată ca prim parametru. Cel de-al doilea parametru este "method" care specifica ce fel de cerere acceptă, în cazul de față este vorba despre "POST", deoarece acceptă datele de autentificare ale user-ului(username, password).

```
@RequestMapping(value = "/authenticate", method = RequestMethod.POST)
public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequestLogin authenticationRequest)
    throws Exception {

    User user = userRepository.findByUsernameAndPassword(authenticationRequest.getUsername(), authenticationRequest.getPassword()).orElse(null);

    if(user != null){
        authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

        final UserDetails userDetails = jwtInMemoryUserDetailsService
            .loadUserByUsername(authenticationRequest.getUsername());

        final String token = jwtTokenUtil.generateToken(userDetails);

        return ResponseEntity.ok(new JwtResponseLogin(token));
    }
```

Service

O clasa adnotată cu `@Service` reprezintă un serviciu ce deservește unui controller și are rolul de a manipula și prelucra datele din baza de date prin intermediul Repository-urilor. Acesta aduce informațiile stocate în baza de date, le aplică o anumită logică, sunt salvate dacă este cazul, iar mai apoi rezultatul procesării este întors controller-ului. Fiecare serviciu primește de asemenea informații de la controller care le poate salva în baza de date sau doar le poate verifica dacă există. Odată pusă aceasta adnotare Spring-ul știe că acea clasă se va ocupa cu funcționalitățile despre mai sus. Un exemplu al unui serviciu este:

```
@Service
public class JwtUserDetailsService implements UserDetailsService {

    @Autowired
    WebSecurityConfig webSecurityConfig;

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        com.kidneyExchange.Entity.User user = userRepository.findByUsername(username).orElse( () -> null);

        if (user != null) {
            return new User(user.getUsername(), webSecurityConfig.encodePassword(user.getPassword()),
                new ArrayList<>());
        } else {
            throw new UsernameNotFoundException("User not found with username: " + username);
        }
    }
}
```

Repository

O clasa adnotată cu `@Repository`, este interpretată de către Spring ca o interfață între un model definit în cod și un tabel din baza de date. Definirea unui Repository este efectuată prin extinderea clasei JPA Repository existentă în Spring care ne permite crearea automată de query-uri urmărind modelul CRUD și care ne oferă posibilitatea salvării, modificării sau ștergerii datelor dintr-o tabelă. Mai jos aveți un exemplu unei astfel de clase definite ca și interfață:

```
public interface UserRepository extends JpaRepository<User, Integer> {

    Optional<User> findByUsernameAndPassword(String username, String password);

    Optional<User> findByUsernameAndEmail(String username, String email);

    Optional<User> findByUsername(String username);
}
```

Model

Un model este o reprezentare a unui obiect în codul nostru. Acesta constituie o reprezentare a unei stări din aplicația noastră. Un exemplu de obiect definit în aplicația

prezentată este modelul ”Donor”, având ca atribute: numele, tipul de sange, e-mail-ul, orașul și data înregistrării. Dar pe lângă aceste atribute, un model poate avea definite anumite funcții care oferă posibilitatea manipulării atributelor.

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Donor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @NotBlank
    private String firstName;

    @NotBlank
    private String lastName;

    @NotBlank
    private String bloodType;

    private String gene;

    private String email;

    private String city;

    private Date arrivingTime;

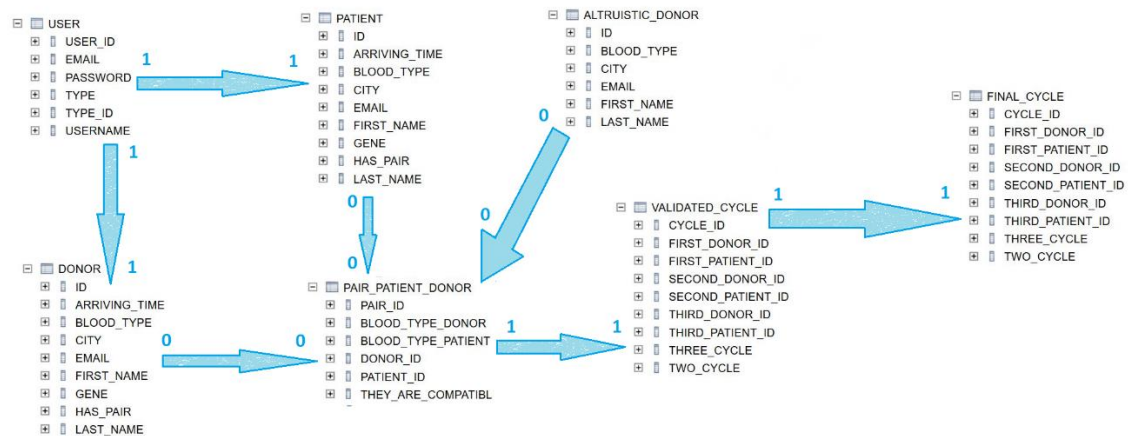
    private Boolean hasPair = false;

    public Donor(String firstName, String lastName, String email, String city, String bloodType,
        Date arrivingTime, Boolean hasAPair) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.city = city;
        this.bloodType = bloodType;
        this.arrivingTime = arrivingTime;
        this.hasPair = hasAPair;
    }
}
```

În exemplul prezentat mai sus întâlnim adnotarea `@Entity` prezentă înaintea antetului clasei `Donor`, care pentru Spring semnifică faptul că aceasta clasă este în strânsă legătură cu tabelul ”donor” din baza de date, iar atributele prezente în această clasă vor fi automat interpretate ca fiind coloanele acelui tabel. În cele mai multe cazuri se poate pune adnotarea `@Column` înaintea fiecărui atribut, dar nu este strict necesar acest lucru. De asemenea poate fi omisă și adnotarea `@Table`, fiind scrisă necesară când intervin anumite contrangeri între entități.

În locul adnotării `@Column` poate fi pusă adnotarea `@Id` care înștiințează Spring-ul că acel atribut va fi considerat cheie primară și va ști că acea coloană va conține valori unice. De asemenea, dacă va fi considerat un `Id` de tip `Integer`, se va folosi adnotarea `@GeneratedValue` care va ajuta la generarea unei chei unice de tipul atribului asignat.

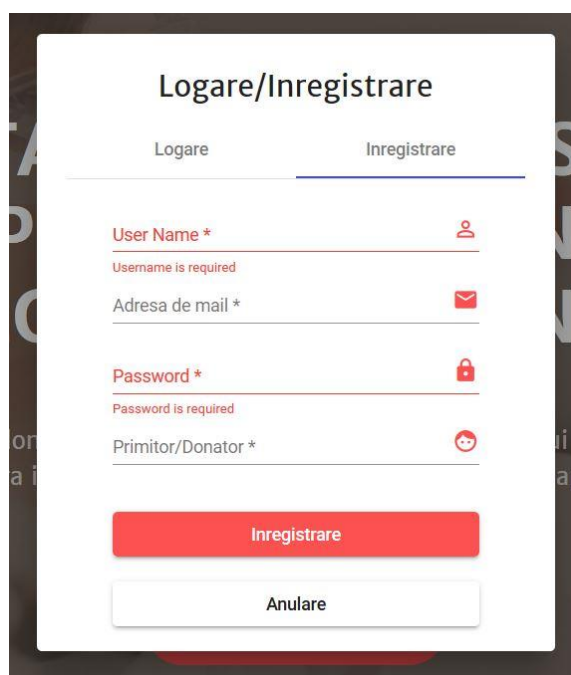
Arhitectura bazei de date



- ✓ Tabela "User" conține informații despre user, precum id-ul unic, username, password, e-mail-ul și tipul de user (pacient/donator).
- ✓ Tabela "Patient" reține informațiile despre pacienții înscrși, aceasta conține un id unic, o data în care s-a înscris pe platformă.
- ✓ Tabela "Donor" conține aceleași field-uri ca cea de pacienți, incluzând și un field de tip boolean "Has Pair" care în funcție de valoare ne spune dacă donatorul are deja un pacient sau nu.
- ✓ Tabela "Pair Patient Donor" va reține datele atât venite de la serverul de pe frontend, dar și cele obținute în urma procesării și creării în funcție de criteriile de compatibilitate și în funcție de perioada în care s-au înscris pe platformă.
- ✓ Tabela "Altruistic Donor" conține acei donatori care sunt în plus și care s-au înscris fără a avea un pacient predefinit, ci doar ca să ajute alți oameni. Aceștia li se va asigna un pacient fictiv și vor fi salvați în tabela perechilor.
- ✓ Tabela "Validated Cycle" conține acele tabele în urma creării unui Digraf Orientat cu care are arcurile trasate în funcție de compatibilitatea între perechile pacienți și donatori. Se vor cauta și salva doar arcurile de maxim lungime 3, conform algoritmului implementat.
- ✓ Tabela "Final Cycle" va reține datele ce vor fi salvate în urma terminării rularii algoritmului și găsirii soluției optime, acolo fiind ciclurile din graf care au fost cel mai favorabil de ales folosind algoritmul ce va fi prezentat în acestă lucrare.

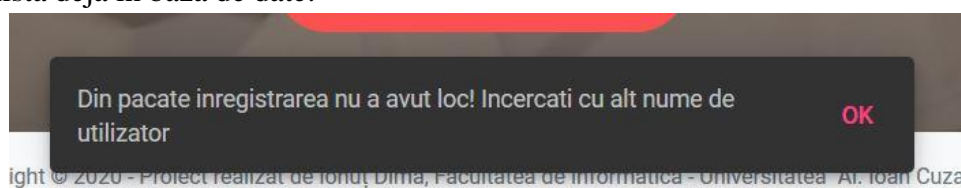
Modul de funcționare al aplicației

Pentru început orice user nou, care dorește să își creeze un cont pe platforma noastră are nevoie să se înregistreze la noi, acest lucru este posibil apasand butonul "Conectare" din dreapta sus, positionat in baza de meniu a aplicației. Odata apasat acel buton se v-a deschide o fereastră de dialog care are doua taburi, în tabul denumit "Înregistrare" va putea introduse datele necesare logării. Aceste date sunt verificate initial pe partea de front-end inainte de a fi trimise la backend. Mai jos puteți vedea interfata mai jos:

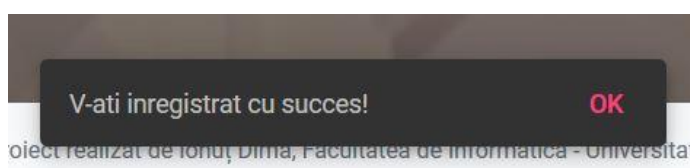


Pentru fiecare request catre serverul de backend, în urma verificării cu baza de date catre front-end va ajunge un raspuns în format JSON care îl va semnala frontend-ului ce mesaj de înștiințare să afișeze.

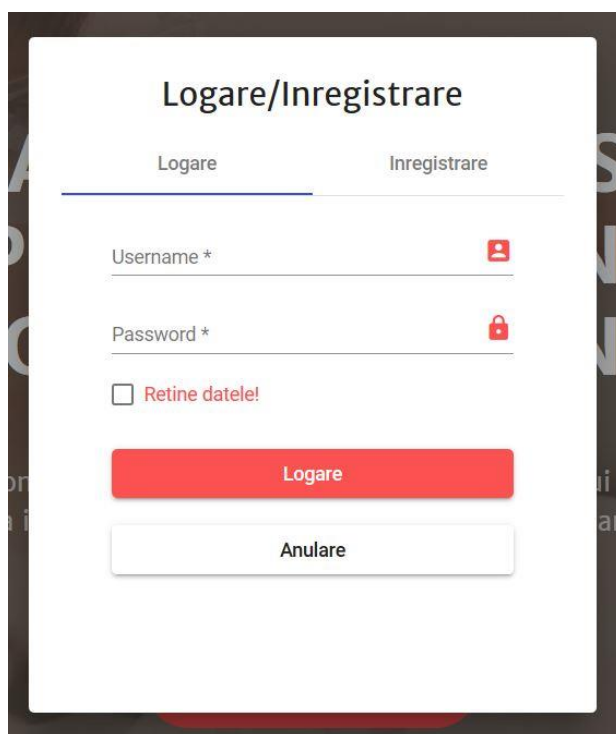
Acest mesaj este afișat în caz ca nu s-a putut efectua înregistrarea sau numele de utilizator exista deja în baza de date:



Acest mesaj este afișat atunci când totul a fost în regulă și user-ul a fost înregistrat cu succes:



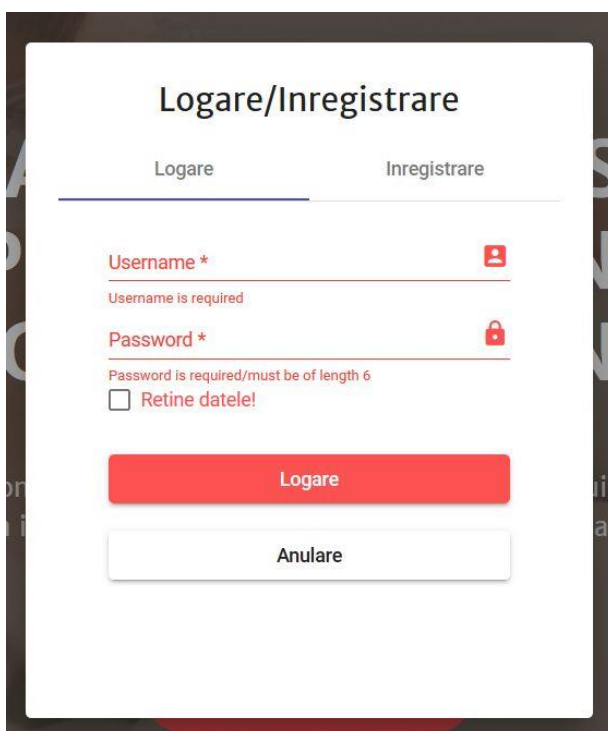
Logarea pe acesta platformă se va face apăsând pe butonul din dreapta sus, denumit "Conectare" care va fi în meniul de sus, dreapta sus, va deschide o pagina de dialog care va conține 2 tab-uri, primul va conține chestionarul de logare.



The screenshot shows a dialog box titled "Logare/Inregistrare". It has two tabs: "Logare" (selected) and "Inregistrare". The "Logare" tab contains the following elements:

- A "Username *" input field with a red user icon on the right.
- A "Password *" input field with a red lock icon on the right.
- A checkbox labeled "Retine datele!" in red text.
- A red button labeled "Logare".
- A white button labeled "Anulare".

Acest formular la randul lui conține anumite verificări anterioare trimeriei datelor la backend.



This screenshot shows the same "Logare/Inregistrare" dialog box, but with validation errors displayed below the input fields:

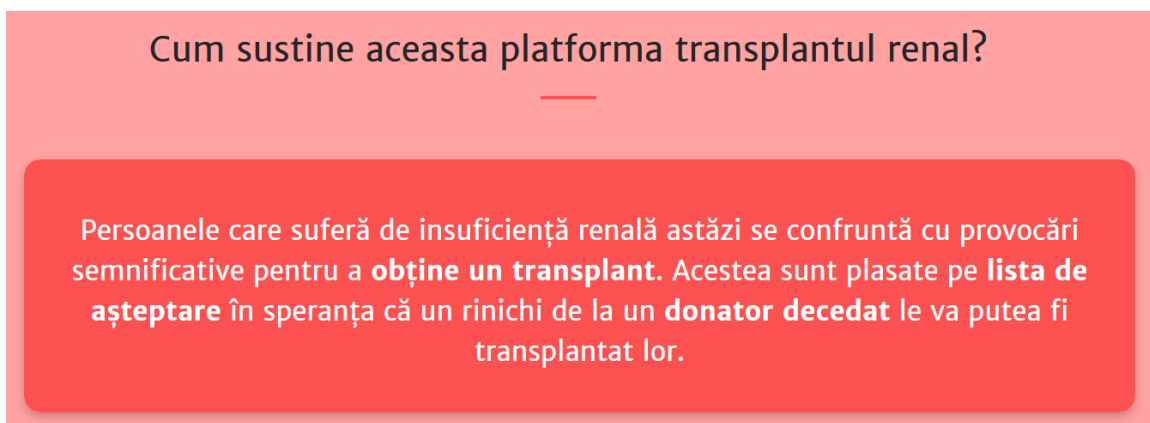
- Below the "Username *" field, the text "Username is required" is shown in red.
- Below the "Password *" field, the text "Password is required/must be of length 6" is shown in red.
- The "Retine datele!" checkbox and the "Logare" and "Anulare" buttons remain visible at the bottom.

Una din primele secțiuni ale interfeței este reprezentat de pagina unde utilizatorii pot afla mai multe despre motivul și scopul acestei aplicații, pagina de informații, aici sunt prezente 3 carduri rotative ce au pe una din fețe întrebarea și pe spate răspunsul și explicațiile acelei întrebări.

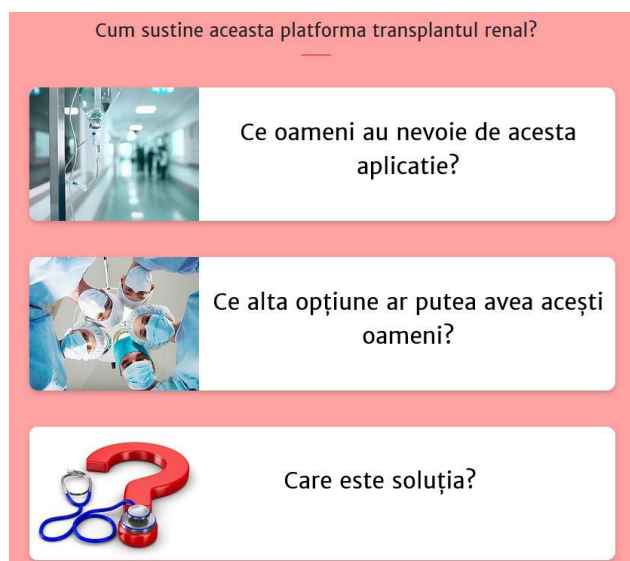
Aceasta este fața unui card:



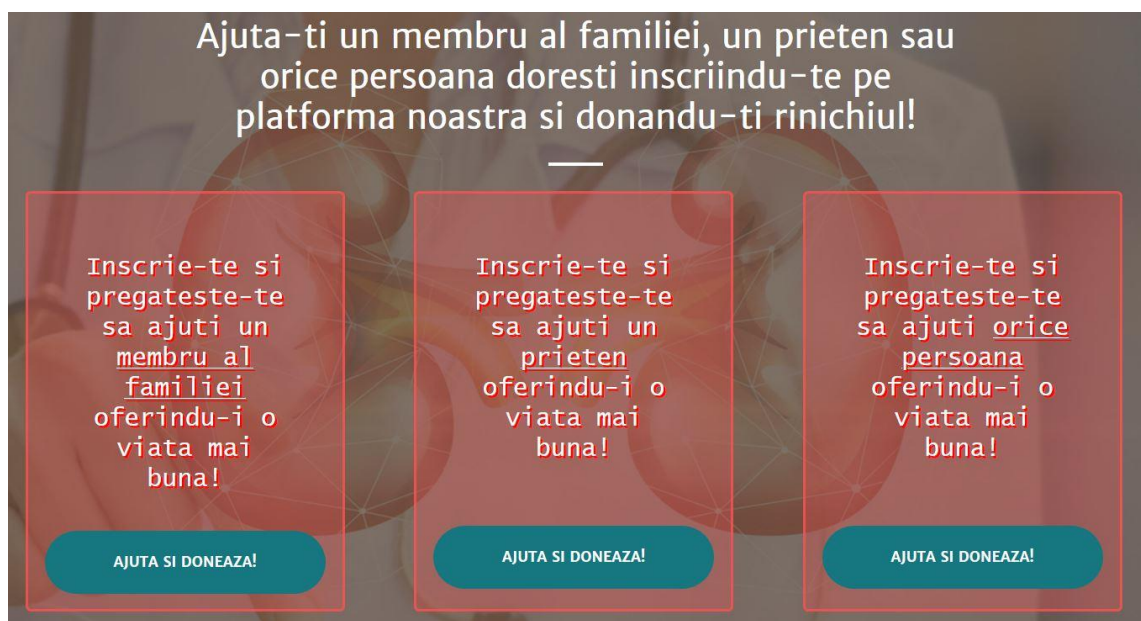
Aceasta este spatele cardului:



Cele 3 carduri rotative sunt:



În secțiunea următoare, unde un utilizator binevoitor se poate înscrie ca și donator din mai multe perspective, ca și membru de familie al unui pacient, ca și prieten al unui pacient sau ca și donator altruist:

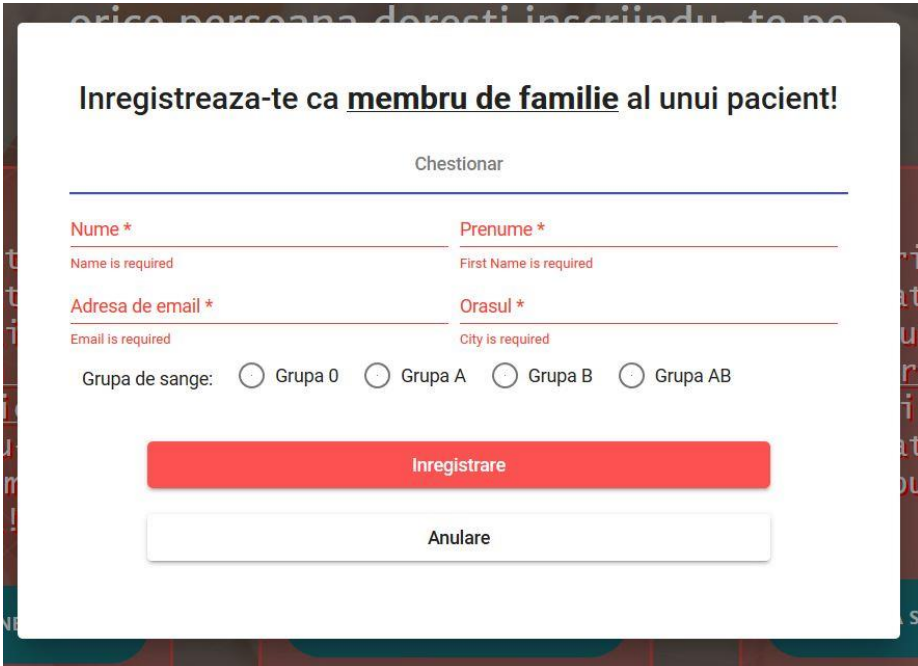


În această pagină există trei carduri, fiecare având cate un buton prin care utilizatorul se poate înscrie, butonul "Ajuta si Doneaza!", la apăsarea acestuia se va deschide o pagina de dialog ce conține un formular care odată completat corect, donatorul va fi înscris pe aceasta platformă, iar la următoarea rulare a algoritmului, acesta va fi luat inclus în procesarea datelor.

The image shows a registration form titled "Inregistreaza-te ca membru de familie al unui pacient!". The form is contained within a white box with a thin border. At the top of the form is a light blue header with the word "Chestionar". Below the header are four input fields: "Nume *" and "Prenume *" on the first line, and "Adresa de email *" and "Orasul *" on the second line. Below these fields is a section for blood group selection, labeled "Grupa de sange:", with four radio button options: "Grupa O", "Grupa A", "Grupa B", and "Grupa AB". At the bottom of the form are two buttons: a red "Inregistrare" button and a white "Anulare" button with a grey border.

Aceast chestionar cuprinde cateva câmpuri obligatorii și un field cu un buton de tip "radio" care permite alegerea unei singure grupe de sânge din cele patru opțiuni puse la dispoziție. La acționarea butonului "Inregistrare" datele sunt preluate și trimise backend-ului, unde daca trec de validări, vor fi inserate in baza de date.

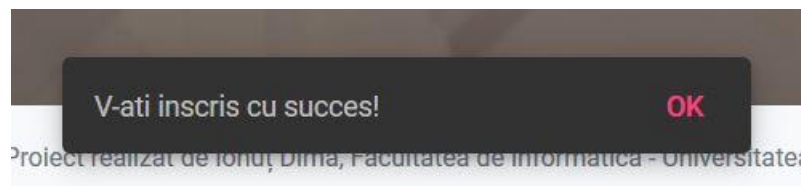
În ceea ce privește validarea field-urilor, există câte un mesaj separat pentru fiecare dată completată:



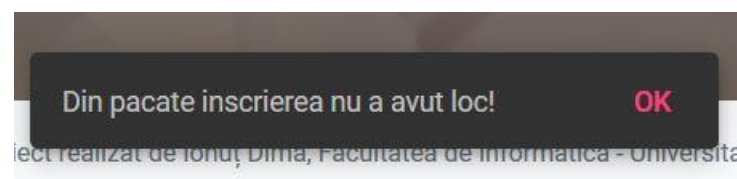
The image shows a registration form titled "Inregistreaza-te ca membru de familie al unui pacient!". Below the title is a section labeled "Chestionar". The form contains several input fields: "Nume *" (Name) with a red error message "Name is required", "Prenume *" (First Name) with a red error message "First Name is required", "Adresa de email *" (Email) with a red error message "Email is required", and "Orasul *" (City) with a red error message "City is required". There is also a radio button group for "Grupa de sange:" (Blood group) with options "Grupa 0", "Grupa A", "Grupa B", and "Grupa AB". At the bottom of the form are two buttons: a red "Inregistrare" button and a white "Anulare" button.

După completarea tuturor informațiilor în acele câmpuri și după validarea lor în partea de backend, după primirea răspunsului de la backend, serverul de frontend va afișa următoarele mesaje în funcție de nereușită sau succes:

Dacă logarea s-a efectuat cu succes:



Dacă logarea NU s-a efectuat cu succes:



În secțiunea unde un pacient se poate înscrie există două posibilitati, prima fiind cea în care un pacient se poate înscrie împreună cu donatorul sau, mai ales dacă nu sunt compatibili, dar chiar și având deja un donator compatibil ar putea oferi ajutorul unei alte perechi donator-pacient care este posibil să nu fie compatibilă, iar prin interschimbarea pacientului și donatorului de la o pereche la alta este posibil ca ambele perechi să poată ajunge să fie compatibile.

Această pagină conține două carduri, iar fiecare dintre cele două joacă un rol important, prima a fost explicată mai sus, iar cel de-al doilea card permite înscrierea unui simplu pacient care încă nu și-a găsit un donator, dar care va avea șanse mari să-l găsească.

Medicul te-a diagnosticat cu **insuficienta renala**? Din fericire, există o soluție mai bună decât listele de așteptare interminabile, înscrie-te pe platforma noastră și într-o perioadă destul de scurtă este posibil să găsești un donator disponibil!

Ai deja o persoană disponibilă să îți doneze rinichiul său, dar nu sunteți compatibili? Inscriți-vă împreună și sansele să găsești un donator compatibil vor crește!

ÎNSCRIȚI-VĂ ÎMPREUNĂ!

Nu ai găsit încă nici un donator disponibil, care să te ajute în drumul spre o viață mai calitativă? Nu îți pierde speranța și înscrie-te pe platforma noastră mărindu-ți sansele spre găsirea unui donator compatibil!

ÎNSCRIE-TE ȘI MĂREȘTE-ȚI ȘANSELE!

La apăsarea primului buton, prin primul rând se va deschide un dialog care va conține două tab-uri, unul pentru înscrierea Primitorului și cel de-al doilea pentru înscrierea donatorului. Aceștia vor fi inserați în baza de date ca fiind o pereche de tip donator-pacient, inserările se vor face în trei tabele, în tabelele "Patient", "Donor" și "Pair Patient Donor".

Înscrie-te împreună cu donatorul tău!

Primitor	Donator
Nume *	Prenume *
Adresa de email *	Orasul *
Grupa de sange: <input type="radio"/> Grupa 0 <input type="radio"/> Grupa A <input type="radio"/> Grupa B <input type="radio"/> Grupa AB	
Inregistreaza si donatorul	
Anulare	

Următorul tab conține campurile pentru înscrierea donatorului:

The image shows a registration form titled "Inscriete împreuna cu donatorul tau!". The form is divided into two columns: "Primitor" (Patient) and "Donator" (Donor). The "Primitor" column has fields for "Nume *" (Name) and "Adresa de email *" (Email address). The "Donator" column has fields for "Prenume *" (First Name) and "Orasul *" (City). Below these fields, there is a section for "Grupa de sange:" (Blood group) with four radio button options: "Grupa 0", "Grupa A", "Grupa B", and "Grupa AB". At the bottom of the form, there are two buttons: a red button labeled "Inregistrare pereche pacient + donator" and a white button labeled "Anulare". The form is set against a background with a repeating pattern of text.

Inscriete împreuna cu donatorul tau!

Primitor	Donator
Nume *	Prenume *
Adresa de email *	Orasul *
Grupa de sange: <input type="radio"/> Grupa 0 <input type="radio"/> Grupa A <input type="radio"/> Grupa B <input type="radio"/> Grupa AB	
Inregistrare pereche pacient + donator	
Anulare	

SCRIȚI-VĂ ÎMPREUNA! ÎNSCRIE-TE SI MAREȘTE-ȚI ȘAN

Pentru fiecare formular în parte se aplică cateva restricții pana când datele să fie trimise la server:

This image shows the same registration form as above, but with red validation messages displayed below the input fields. The messages are: "Name is required" under the "Nume *" field, "First Name is required" under the "Prenume *" field, "Email is required" under the "Adresa de email *" field, and "City is required" under the "Orasul *" field. The rest of the form, including the blood group options and buttons, remains the same.

Inscriete împreuna cu donatorul tau!

Primitor	Donator
Nume * Name is required	Prenume * First Name is required
Adresa de email * Email is required	Orasul * City is required
Grupa de sange: <input type="radio"/> Grupa 0 <input type="radio"/> Grupa A <input type="radio"/> Grupa B <input type="radio"/> Grupa AB	
Inregistrare pereche pacient + donator	
Anulare	

SCRIȚI-VĂ ÎMPREUNA! ÎNSCRIE-TE SI MAREȘTE-ȚI ȘAN

În ceea ce privește încrierea unui singur pacient folosind cardul al doilea, aici se completează un campurile, iar când acestea sunt introduse corect, datele ajung in baza de date.

Inregistreaza-te ca primitor!

Chestionar

Nume *

Prenume *

Adresa de email *

Orasul *

Grupa de sange: ☐ Grupa 0 ☐ Grupa A ☐ Grupa B ☐ Grupa AB

Inregistrare pacient

Anulare

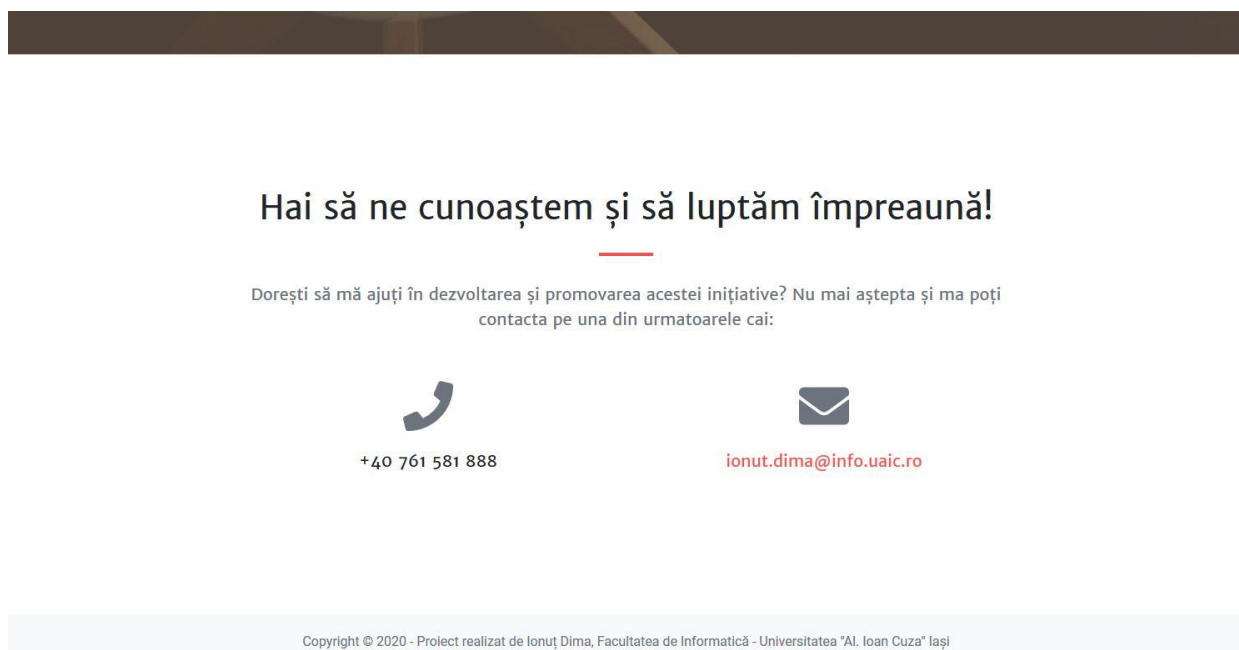
SCRIȚI-VĂ ÎMPREUNA!

ÎNSCRIE-TE SI MARESTE-ȚI ȘAN

În ceea ce privește secțiunea ”Despre noi” am ținut să menționăm câteva lucruri despre autorii acestei lucrări, aici sunt prezente 3 carduri pe orizontală unde sunt încadrate 3 paragrafe cu câteva detalii despre autori.



Iar la finalul aplicației avem o pagină cu câteva detalii de contact ale autorilor:



Este de menționat că această platformă este responsive și își modifică dimensiunile în funcție de dimensiunile dispozitivului de pe care este accesată.

Algoritmi dezvoltăți

Pregătirea datelor pentru algoritm și elaborarea algoritmului

Dupa cum am prezentat pana acum aplicația ofera defirite moduri pentru utilizatorii săi de a se înscrie și de a face parte dintr-un sistem de calcul care ar putea duce la gasirea unui perechi de tip(donator-pacient) care inițial nu erau compatibili, la o pereche compatibilă.

Datele stocate în tabelele "Donator", "Pacient" sunt manipulate astfel:

1. Se creează în memorie cate o lista cu fiecare entitate din cele doua, aceasta este deja de la creare setata în ordine descrescătoare în funcție de data la care s-a facut înregistrarea în tabela.
2. Ținând cont de criteriul de compatibilitate stabilit mai sus, cel cu grupele de sange, se vor forma perechi de tipul pacient-donator, ținand cont de ordinea și probabilitatile de a dona sau de a primi în funcție de grupa de sange. Astfel având prioritate grupele de sange care găsesc mai greu donatori.
3. Dupa terminare acestui proces, perechile ce raman incompatibile, impreuna cu donatorii care raman fara un pacient(numarul de donatori fiind mai mare ca cel de primitori), acestora din urma adaugandu-le un pacient fictiv vor forma un graf orientat.
4. Graf-ul orientat va avea ca noduri perechile incompatibile ramase, si va fi arc de la un nodul A la nodul B, daca si numai daca donatorul de la nodul A poate dona pacientului de la nodul B.
5. Dupa trasarea arcurilor vom începe prin cautarea ciclurilor din graf de lungime doi și trei, odată gasite, acestea sunt salvate în tabela "Validated Cycle".
6. Conform algoritmului descris de cei de la Uuniversitatea Glasgow, cu ajutorul careia vom continua prin creerea a unei matrici C care sa stocheze situatia ciclurilor și a doi vectori B si C, ca mai apoi sa putem calcula $\max cx$ în funcție de Ax mai mic sau egal decat B.
7. Cu ajutorul unui solver pentru probleme liniare și avand datele stogate în matrice si în vectori am contruit o problema liniara pe care acel solver stie sa o rezolve.
8. În final valorile de x care vor avea valoare 1 vor reprezenta ciclurile optime din graf.

Mai jos veti putea vedea cum arata fisierul problemei liniare pe care solverul integrat n proiect reuseste să-l rezolve.


```

/* Objective function */
max: +3 x1 +3 x2 +2 x3 +3 x4 +3 x5 +3 x6 +3 x7 +3 x8 +2 x9 +3 x10 +3 x11 +3 x12 +2 x13 +3 x14 +2 x15 +3 x16
+2 x17 +3 x18 +3 x19 +3 x20 +3 x21 +3 x22 +3 x23 +3 x24 +3 x25 +2 x26 +3 x27 +3 x28 +2 x29 +3 x30 +3 x31
+3 x32 +2 x33 +3 x34 +3 x35 +3 x36 +3 x37 +3 x38 +2 x39 +3 x40 +3 x41 +3 x42 +2 x43 +3 x44 +3 x45 +3 x46
+3 x47 +2 x48 +3 x49 +2 x50 +3 x51 +3 x52 +2 x53 +3 x54 +3 x55 +2 x56 +3 x57 +2 x58 +3 x59 +3 x60 +3 x61
+3 x62 +3 x63 +3 x64 +3 x65 +2 x66 +3 x67 +3 x68 +2 x69 +3 x70 +3 x71 +3 x72 +3 x73 +2 x74 +3 x75 +3 x76
+3 x77 +3 x78 +3 x79 +3 x80 +3 x81 +3 x82 +2 x83 +3 x84 +3 x85 +3 x86 +2 x87 +3 x88 +3 x89 +3 x90 +3 x91
+2 x92 +3 x93 +3 x94 +2 x95 +3 x96 +3 x97 +2 x98 +3 x99 +3 x100 +2 x101 +3 x102 +3 x103 +3 x104 +3 x105
+2 x106 +3 x107 +3 x108 +3 x109 +3 x110 +2 x111 +3 x112 +3 x113 +3 x114 +3 x115 +2 x116 +3 x117 +3 x118
+3 x119 +2 x120 +3 x121 +3 x122 +3 x123 +2 x124 +3 x125 +3 x126 +3 x127 +3 x128 +2 x129 +3 x130 +3 x131
+3 x132 +3 x133 +2 x134 +3 x135 +3 x136 +3 x137 +2 x138 +3 x139 +3 x140 +3 x141 +3 x142 +3 x143 +3 x144
+3 x145 +3 x146;

/* Constraints */
+x1 +x2 +x3 +x4 +x5 +x6 +x7 +x8 +x9 +x10 +x11 +x12 +x13 +x14 +x18 +x21 +x45 +x50 +x51 +x52 +x59 +x62
+x70 +x75 +x84 +x87 +x88 +x89 +x90 +x91 +x107 +x112 +x121 +x124 +x125 +x126 +x127 +x128 +x139 <= 1;
+x1 +x2 +x15 +x16 +x17 +x18 +x19 +x20 +x21 +x22 +x23 +x24 +x31 +x32 +x66 +x67 +x68 +x76 +x88 +x92 +x93
+x94 +x102 +x117 +x125 +x130 +x135 +x140 <= 1;
+x25 +x26 +x27 +x77 +x95 +x96 +x97 +x141 <= 1;
+x28 +x29 +x30 +x78 +x98 +x99 +x100 +x142 <= 1;
+x19 +x22 +x31 +x32 +x33 +x34 +x35 +x36 +x37 +x38 +x39 +x40 +x41 +x42 +x43 +x44 +x46 +x53 +x54 +x55 +x60
+x63 +x71 +x79 +x85 +x101 +x102 +x103 +x104 +x105 +x108 +x113 +x122 +x129 +x130 +x131 +x132 +x133 +x143 <= 1;
+x10 +x40 +x45 +x46 +x47 +x48 +x49 +x80 +x106 +x107 +x108 +x109 +x110 +x144 <= 1;
+x3 +x4 +x5 +x6 +x11 +x33 +x34 +x35 +x36 +x41 +x50 +x51 +x52 +x53 +x54 +x55 +x56 +x57 +x58 +x59 +x60
+x61 +x62 +x63 +x64 +x65 +x69 +x70 +x71 +x72 +x73 +x81 +x89 +x103 +x111 +x112 +x113 +x114 +x115 +x118
+x126 +x131 +x136 +x145 <= 1;
+x6 +x7 +x8 +x15 +x16 +x20 +x23 +x25 +x28 +x36 +x37 +x38 +x47 +x51 +x54 +x56 +x57 +x61 +x64 +x66 +x67
+x68 +x69 +x70 +x71 +x72 +x73 +x74 +x75 +x76 +x77 +x78 +x79 +x80 +x81 +x82 +x83 +x84 +x85 +x86 +x90
+x93 +x96 +x99 +x104 +x109 +x114 +x116 +x117 +x118 +x119 +x123 +x127 +x132 +x134 +x135 +x136 +x137 +x146
+x147 <= 1;
-x6 -x7 -x8 -x15 -x16 -x20 -x23 -x25 -x28 -x36 -x37 -x38 -x47 -x51 -x54 -x56 -x57 -x61 -x64 -x66 -x67
-x68 -x69 -x70 -x71 -x72 -x73 -x74 -x75 -x76 -x77 -x78 -x79 -x80 -x81 -x82 -x83 -x84 -x85 -x86 -x90
-x93 -x96 -x99 -x104 -x109 -x114 -x116 -x117 -x118 -x119 -x123 -x127 -x132 -x134 -x135 -x136 -x137 -x146
-x147 <= -1;
+x1 +x4 +x7 +x9 +x10 +x11 +x12 +x14 +x16 +x17 +x18 +x19 +x20 +x24 +x25 +x26 +x27 +x28 +x29 +x30 +x31
+x34 +x37 +x39 +x40 +x41 +x42 +x44 +x45 +x46 +x47 +x48 +x49 +x52 +x55 +x57 +x58 +x59 +x60 +x61 +x65
+x67 +x72 +x74 +x75 +x76 +x77 +x78 +x79 +x80 +x81 +x82 +x86 +x87 +x88 +x89 +x90 +x91 +x92 +x93 +x94
+x95 +x96 +x97 +x98 +x99 +x100 +x101 +x102 +x103 +x104 +x105 +x106 +x107 +x108 +x109 +x110 +x111 +x112

```

Concluziile lucrării

Aplicația Problema Transplantului de rinichi a fost proiectată să deservească pacienților care au nevoie de un transplant de rinichi, și pentru ai ajuta să-și găsească cat mai repede un donator viu dispus să doneze, dar mai ales să interschimbe cat mai multe perechi donator-pacient, astfel numărul de transplanturi să crească și numărul de cicluri să scadă.

Dezvoltare ulterioară

În viitor aplicația va putea avea îmbunătățiri majore pe următoarele planuri:

1. Adăugarea de noi constrângeri la rezultatul actual al algoritmului.
2. Îmbunătățirea interfeței cu utilizatorul.
3. Adăugarea unei noi pagini de profil care să arate noi și existente date despre acesta.

Anexe

Anexa 1 - Java

Java este un limbaj de programare orientat-obiect de nivel înalt dezvoltat inițial de compania Sun Microsystems lansat pentru prima oară în anul 1995. Ultima versiune de Java Standard Edition este Java SE 10 lansat în 2018, cea mai populară.

Un program ce este format din cod Java poate fi rulat pe orice tip de mașina indiferent de sistemul de operare. Cele mai importante caracteristici ale acestui limbaj sunt⁸:

- orientat-obiect – orice este scris în Java este un obiect
- poate rula pe orice mașina indiferent de sistemul de operare
- sigur – are o buna documentație și asigură diverse mecanisme de securitate
- se poate lucra pe mai multe fire de execuție

Anexa 2 - Spring framework

Un framework este o structură conceptuală destinată să fie folosită ca și suport sau ghid pentru construirea unor materiale care extind structura respectivă în ceva folositor.

În domeniul IT un framework este, de obicei, o structură stratificată ce indică tipul programelor care pot fi construite și cum ar trebui să acestea să interacționeze⁹.

Spring este cel mai popular framework pentru dezvoltarea aplicațiilor în Java. Milioane de dezvoltatori folosesc Spring Framework pentru a crea cod reutilizabil, ușor de testat și cu o performanță ridicată. Un element cheie al Spring-ului este suportul infrastructural la nivelul aplicației ceea ce înseamnă că framework-ul se concentrează pe “sanitizarea” aplicațiilor, astfel încât echipele să se poată concentra pe logică afacerii la nivelul respective, fără legături inutile cu medii specifice de livrare.

Spring framework este o platformă Java care a apărut pentru prima dată sub licența Apache 2.0 în anul 2003, dezvoltat de către Rob Johnson. Versiunea de bază a frameworkului are o dimensiune scăzută de doar 2MB. Funcționalitățile de bază ale acestui

⁸ https://www.tutorialspoint.com/java/java_overview.htm

⁹ <https://whatistechtarget.com/definition/framework>

framework pot fi folosite în dezvoltarea tuturor aplicațiilor Java, dar există și extensii ale acestuia construite special pentru dezvoltarea aplicațiilor web în platformă Java EE¹⁰. Fiind open-source¹¹, Spring se bucură de o comunitate vastă și activă care oferă suport bazat pe o varietate de cazuri reale, ceea ce a ajutat framework-ul să evolueze cu succes într-o perioadă lungă de timp.

Framework-ul este împărțit pe module. Aplicațiile pot alege doar modulele de care au nevoie. La baza stau modulele containerului principal (core) ce include un model de configurație și un mecanism de injectare a dependentelor. Pe lângă acestea, Spring oferă suport pentru diferite arhitecturi de aplicații printre cum ar fi mesagerie, date tranzacționale și persistarea datelor și web. Include de asemenea un framework web bazat pe servlet-uri numit Spring MVC.

Spring continuă să inoveze și să evolueze. În afară de Spring Framework există și alte proiecte precum Spring Boot, Spring Security, Spring Dată, Spring Cloud, Spring Batch.

Anexa 3 - Dependency Injection și Inversion of Control

Injectarea dependentelor, ce face parte din conceptul mai vast de inversiune a controlului, sta la baza Spring-ului. Principiul de design numit “inversiune a controlului” este folosit pentru a inversa diferite tipuri de control într-o proiectare orientata-obiect pentru a obține un cuplaj slab între elemente. În contextul curent, controlul înseamnă orice responsabilitate adițională pe care o clasă o are, pe lângă responsabilitatea de bază a acesteia, cum ar fi controlul asupra pașilor de execuție a unei aplicații sau controlul asupra pașilor de execuție urmați când se creează un obiect.[\[1\]](#)¹² Injectarea dependentelor nu este nimic altceva decât un caz concret al inversiunii controlului.

Când o aplicație Java complexă este scrisă, clasele aplicației trebuie să fie cât mai independente pentru a crește posibilitatea refolosirii acestora și pentru a fi posibilă testarea pe unități. Injectarea dependentelor ajută la îmbinarea claselor, acestea rămânând totuși independente.

Anexa 4 - Beneficiile utilizării framework-ului Spring

Cele mai importante beneficii pe care le aduce framework-ul Spring și motivele pentru care acesta ar trebui ales pentru implementarea unei aplicații sunt:

- Spring asigură diverse mecanisme ce ajută la implementarea aplicațiilor complexe într-un mod simplu și elegant
- Spring este organizat în stil modular, există o imensitate de clase și pachete în acest framework dar noi putem să le importăm doar pe cele de care avem nevoie
- Codul este ușor de testat
- În Spring putem asocia ușor erorile sau excepțiile provenite de la alte tehnologii

¹⁰https://www.tutorialspoint.com/spring/spring_overview.htm

¹¹ open-source = codul sursă original este accesibil oricui și poate fi redistribuit sau modificat

¹²<https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html>

- Dependency Injection și Inversion of Control
- Oferă o interfață pentru managementul tranzacțiilor de date¹³

Anexa 3 - Spring Boot

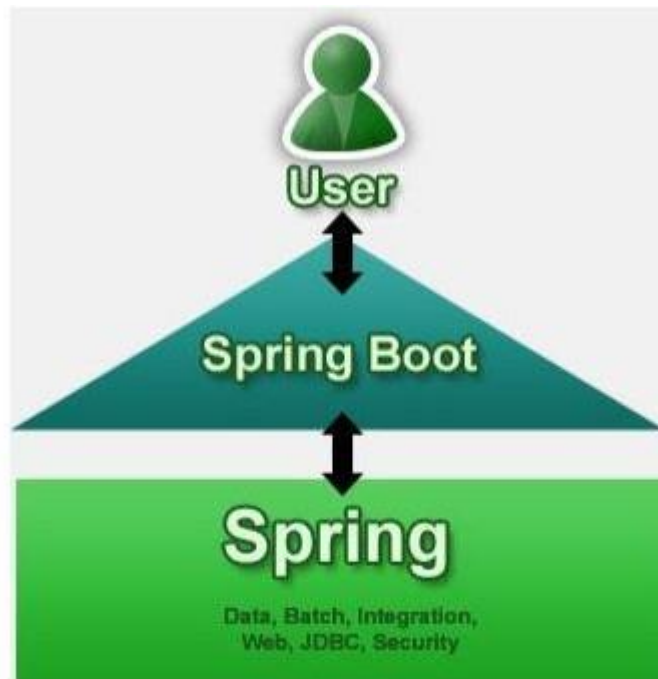


Fig 3 Spring boot

Spring Boot este o versiune superioara a Spring-ului care vine cu noi mecanisme care sa ajute la configurarea mai ușoara a unui sistem. Asa cum se vede și în Fig 3 , Spring Boot nu este decât o versiune superioara Spring care are ca obiective principale următoarele¹⁴:

- Să asigure o ușoara dezvoltare a proiectelor abia începute, neavând nevoie de toate cunoștințele necesare din Spring, Spring Boot-ul făcând multa magie în spate pentru programator.
- Sa elimine partea greoaie de configurări
- Să aducă o varietate de mecanisme care sa ușureze procesul de development

Anexa 4 - Maven

“Maven” este considerat ca fiind un instrument pentru gestionarea proiectelor software (“Project management tool”) și include facilități necesare dezvoltării de aplicații mari, la care contribuie mai multe persoane, care refolosesc părți din alte proiecte software, care pot genera rapoarte și site-uri Web. Versiunile Maven folosite în prezent sunt 2.2 și 3.0 și folosesc tot fișiere XML pentru descrierea operațiilor și dependentelor.

¹³https://www.tutorialspoint.com/spring/spring_overview.htm

¹⁴<https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone>

Anexa 5 - Orm, JPA, Hibernate

ORM sau Object Relațional Mapping este un procedeu de transformare a obiectelor în tipuri relaționale și viceversa.. Cea mai importanta problema pe care ORM-ul o rezolva este aceea de a realiza o corespondență între un obiect și datele din baza de date.

Avantajele unei implementări ORM iau această formă în Hibernate¹⁵:

- Relația obiect java – tabel în baza de date se face pe bază de fișiere XML
- O interfață simplă pentru stocarea și preluarea datelor în/din baza de date
- Dacă se produce o schimbare în tabele, o simplă modificare în fișierele XML este tot ce trebuie • Hibernate nu are nevoie de un server al aplicației pentru a funcționa
- Minimizează accesul la baza de date prin strategii de preluare a datelor JPA, sau Java Persistence API, este o colecție de clase și metode care au ca scop stocarea și preluarea de date dintr-o bază de date. Deoarece JPA este open-source, diferite companii au furnizat produse folosindu-se de funcționalitățile oferite de JPA, una dintre ele fiind Hibernate.

Anexa 6 - Arhitectura ORM

Asa cum se vede în Fig 5, salvarea datelor în baze de date relaționale se face prin 3 pași¹⁶, acești pași sunt:

Pasul 1: numit și Object data, folosește clase POJO, interfețe și alte clase.

Faza 2: numit și pasul de Mapping care realizează asocierea între obiecte Java și tabelele din baza de date.

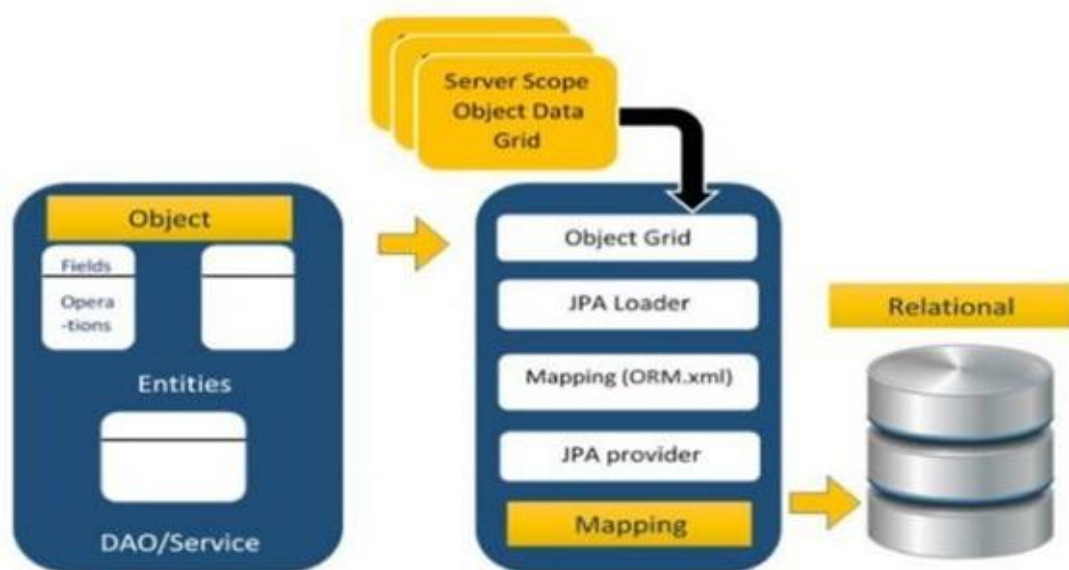


Fig 5 Arhitectura ORM

¹⁵ https://www.tutorialspoint.com/hibernate/hibernate_overview.htm

¹⁶ https://www.tutorialspoint.com/jpa/jpa_orm_components.htm

Pasul 3: Conține datele relaționale care sunt conectate în mod logic de business. Datele sunt stocate fizic în baza atunci când se face commit pe partea de business.

Anexa 7 - Adnotari

În general, fișierele XML sunt folosite pentru a configura o componentă specifică său. În cazul arhitecturii ORM, trebuie să păstrăm fișierul ORM.xml, ce definește relațiile între obiectele Java și obiectele relaționale, undeva în framework. Asta înseamnă că atunci când acest fișier xml este scris, trebuie mereu comparate numele atributelor din clasele POJO cu etichetele scrise în xml pentru a se asigura corespondența între cele două.

Adnotările sunt soluția pentru problema descrisă mai sus. În definiția clasei se poate scrie partea de configurare folosind doar adnotări. Adnotările sunt folosite pentru clase, proprietăți și metode, acestea începând cu simbolul “@”. Toate adnotările sunt definite în pachetul javax.persistence¹⁷.

Exemple de adnotări:

- @Entity – specifica ca o clasă este o entitate (tabel)
- @Id – apare atunci când se dorește ca un atribut din entitate sa fie cheia primara pentru tabelul din baza de date
- @GeneratedValue – este folosit pentru a specifica cum trebuie sa se genereze valorile pentru cheile primare ce sunt adnotate cu @Id
- @Column – indica ca un anumit atribut al entității este o coloana în baza de date
- @UniqueConstraint – este folosit pentru a declara o constrângere
- @OneToOne – este folosit pentru a declara o relație între doua tabele

Anexa 8 - JWT

JWT sau JSON Web Token este un standard ce definește o cale compactă și independentă pentru a transmite, în siguranță, informații între diferite componente sub forma unui obiect JSON. Aceste informații sunt de încredere deoarece pot fi verificate datorită semnăturii digitale. În aplicația în discuție, JWT sunt semnate cu chei publice și private utilizant algoritmul de criptare RSA.

Cele mai importante scenarii în care JSON Web Token este folosit sunt:

- Autorizare – Este cel mai comun scenariu de folosire a JWT. Din moment ce utilizatorul este conectat, orice cerere ulterioară va conține un JWT, astfel permițând utilizatorului să acceseze diferite rute, servicii și resurse care sunt disponibile cu acel token.
- Schimb de informații – JSON Web Tokens sunt o modalitate bună de a transmite informații în siguranță între diferite componente. Deoarece JWT poate fi semnat, putem fi siguri de identitatea utilizatorului. Adițional, datorită faptului că semnătura

¹⁷ https://www.tutorialspoint.com/jpa/jpa_orm_components.htm

este calculată folosind antetul și sarcină utilă (payload), se poate verifica faptul că nu a fost manipulat conținutul informației.

Structura unui JSON Web Token implică 3 componente separate printr-un punct. Acestea sunt: antetul (header), sarcină utilă (payload) și semnătura (signature). După cum se poate observa și în Fig 6 de mai jos, antetul este format din două părți: tipul token-ului (adică JWT) și algoritmul de hashing folosit (în cazul nostru RSA).

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Fig 6 JWT Header

Payload-ul conține solicitările (claims). Acestea sunt despre o entitate (de obicei utilizatorul) și datele adiționale. Există trei tipuri de solicitări:

- Înregistrate – Sunt un set de solicitări predefinite care nu sunt obligatorii, dar sunt recomandate deoarece oferă funcționalități de identificare a emitentului, data de expirare a tokenului și multe altele
 - Publice – pot fi definite liber, dar trebuie avut în vedere faptul că există posibilitatea apariției coliziunilor între acestea.
 - Private – solicitări personalizate create pentru a împărtăși informații între componentele care își exprimă acordul pentru folosirea lor. Fig 6 JWT Header 17
- Structura unui payload este descrisă în Fig 7 de mai jos.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Fig 7 structura unui payload

Pentru a genera semnătura avem nevoie de antetul criptat, payload-ul criptat, un secret și de algoritmul specificat în header. Semnătură este folosită să pentru a verifica faptul că mesajul nu a fost schimbat în timpul schimbului de informații și, în cazul în care token-urile sunt semnate cu chei private, pentru a valida identitatea expeditorului.

Mod de funcționare:

Când un utilizator se autentifica folosind credentialele sale, un JSON Web Token va fi returnat. Oricând utilizatorul va dori accesarea unei rute sau unei resurse protejate, el va trebuie să trimită în cerere (request) JWT-ul returnat anterior. De obicei, JWT-ul este trimis în antetul de autorizare folosind schema "Bearer", cum se poate observa și în Fig 8.

```
Authorization: Bearer <token>
```

Fig 8 schema "Bearer" în antetul de autorizare

Dacă se încearcă accesarea unei rute sau resurse protejate, serverul verifica existent unui JWT valid în antetul de autorizare. Dacă acesta există și este valid, utilizatorului i se permite accesarea resursei protejate.

După cum se observă și în Fig 9 pașii de execuție pentru obținerea și utilizarea JWT-urilor sunt:

- 1. Aplicația cere autorizare de la serverul competent
- 2. Când autorizarea este acordată, serverul de autorizare întoarce un token de acces
- 3. Aplicația folosește token-ul de acces pentru a accesa resurse protejate



Fig 9 flow pentru obtinerea si folosirea JWT

Avantajele utilizării JWT-urilor în detrimentul altor tehnologii asemănătoare precum SWT (Simple Web Tokens) sau SAML (Security Assertion Markup Language Tokens) sunt¹⁸ :

- JWT este o alegere mai bună decât SAML în contextul folosirii mediilor HTML și HTTP deoarece este mai compact. JSON folosit de JWT este mai puțin prolix decât XML folosit de SAML, iar când este criptat dimensiunea lui scade.
- Din punct de vedere al securității, SWT pot fi semnate simetric de un secret împărțit de mai multe componente folosind algoritmul HMAC. JWT și SAML pot folosi perechi de chei publice-private conform certificatului X.509 pentru a semnatura. Deoarece semnarea fișierelor XML folosite de SAML fără a introduce breșe de securitate este foarte dificilă comparată cu semnarea obiectelor JSON folosite de JWT, cel din urmă este clar o opțiune mai rapidă, sigură și mai ușor de perceput.
- Tehnici de parsare ale unui obiect JSON sunt prezente în majoritatea limbajelor de programare deoarece se mapează direct la un obiect. XML nu beneficiază de asemenea funcționalitate.
- Folosit la scară mare pe internet ceea ce evidențiază dificultatea redusă de procesare a JWT pe platforme multiple, în special pe dispozitivele mobile.
- Folosit la scară mare pe internet ceea ce evidențiază dificultatea redusă de procesare a JWT pe platforme multiple, în special pe dispozitivele mobile.

¹⁸ <https://jwt.io/introduction/>

Bibliografie

- <https://www.maxcdn.com/one/visual-glossary/web-application/>
- https://www.tutorialspoint.com/java/java_overview.htm
- <https://whatis.techtarget.com/definition/framework>
- https://www.tutorialspoint.com/spring/spring_overview.htm
- <https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html>
- <http://www.tutorialsteacher.com/ioc/inversion-of-control>
- <https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone>
- https://www.tutorialspoint.com/jpa/jpa_orm_components.htm
- https://www.tutorialspoint.com/hibernate/orm_overview.htm
- https://www.tutorialspoint.com/hibernate/hibernate_overview.htm
- <https://jwt.io/introduction/>
- <https://en.wikipedia.org/wiki/HTML>
- https://en.wikipedia.org/wiki/Cascading_Style_Sheets
- <https://www.javainuse.com/>
- <https://medium.com>