Cabinet Medical

Buzoi Bianca-Nicoleta – grupa 405

Scopul aplicatiei

Aplicatia prezentata in aceasta lucrare are rolul principal de a ajuta oamenii in crearea si managereriea programarilor la diferiti medici, in functie de programul lor de lucru si disponibilitate, la un cabinet medical din Romania. Mai mult decat atat, aplicatia isi propune sa incurajeze donarea de sange si sa vina in ajutorul oamenilor aflati in impact -oameni care au nevoie urgent de sange. Astfel, la nivelul aplicatiei pot fi adaugate diferitele cazuri urgente (oameni care au nevoie de sange dupa ce au suferit un accident rutier grav, se afla intr-o stare nefavorabila de sanatate, etc) astfel incat sa devina vizibile unui numar cat mai mare de persoane. Totodata, se pot face programari pentru a merge sa se doneze sange de catre clientii aplicatiei. In acest mod, se urmareste o cat mai buna raspandire a stirilor de acest fel si incurajarea populatiei sa mearga la donare.

Modelele aplicatiei

Aplicatia are la baza un numar de 7 modele:

1. **User** – retine informatiile esentiale despre un anumit client al aplicatiei:
    - Username
    - Firstname
    - Lastname
    - Email
    - Password
    - UserType (ADMIN/DOCTOR/PACIENT) – la nivelul aplicatiei userii pot avea unul dintre cele 3 roluri enumerate. Adminul este responsabil de crearea, adaugarea si updatarea informatiilor despre doctori la nivelul aplicatiei, precum si de adaugarea cazurilor pentru donatii de sange. Doctorul va ocupa un rol special (trimite retete catre pacient, vizualizeaza istoricul medical al lui, seteaza un diagnostic), dar functionalitatile sale vor fi introduse intr-o dezolvare ulterioara a aplicatiei. Pacientul are posibilitatea de a crea o programare atat pentru donatie, cat si pentru o consultatie, de a-si vizualiza proprile programari pe diferite momente de timp (trecut/viitor) si de a anula o programare la care, din diferite motive, nu mai poate ajunge.
    - LoggedStatus – retine daca un user este logat in aplicatie (=1) sau nu (=0)
2. **DoctorDetails** – retine informatiile/detalile despre un anumit doctor:
    - Specialization
    - Experience

- Description

3. **DoctorAvailableTime** – retine programul specific unui doctor pe zilele saptamanii. Clasa contine 7 variabile (corespunzatoare zilelor saptamanii) de tipul String, ce va lua valori de forma "HH-HH" , semnificand intervalul in care medicul se gaseste in clinica. In cazul in care acesta nu are program in ziua respectiva, programul ia forma "-".
4. **Doctor** - retine informatiile esentiale despre un anumit doctor din cadrul cabinetului medical:
    - Firstname
    - Lastname
    - Email
    - (One-to-one) DoctorDetails
    - (One-to-one) DoctorAvailableTime

5. **Appointment** – clasa ce mapeaza o programare la nivelul aplicatiei:
    - Data
    - Hour
    - Notes
    - (One-to-Many) Doctor
    - (One-to-Many) User
6. **RequestDonation** – retine informatiile despre un caz grav al unei persoane care are nevoie de sange:
    - Firstname
    - Lastname
    - Hospital – spitalul in care se afla pacientul aflat in stare grava
    - Description
    - BloodType
7. **DonationAppointment** – clasa ce mapeaza o programare la o donatie:
    - Data
    - Hour
    - (One-to-Many) User
    - (One-to-Many) RequestDonation

Functionalitatile principale ale aplicatiei

Modele enuntate mai sus sunt folosite pentru implementarea urmatoarelor mari functionalitati:

1. ***Inregistrarea user-ilor si logarea/delogarea lor de la nivelul aplicatiei (UserService)***
2. ***Adaugarea doctorilor, updatarea informatiilor despre acestia de catre Admin (DoctorService)***
3. ***Managerierea programarilor pentru o consultatie (AppointmentService)***
4. ***Adaugarea cazurilor pentru donatie de sange (RequestDonationService)***
5. ***Managerierea programarilor pentru o donatie (DonationAppointment Service)***

6. ***Notificarea utilizatorului in momentul in care realizeaza o programare (NotificationService) – notificarea se face intr-un mod simplu, prin afisarea unui mesaj in consola cu datele programarii***.

```
System.out.println("New appointment created:  " + app.getData().toString()
+ " - " + app.getHour() + " .Doctor"
        + app.getDoctor().getFirstName() + " " +
app.getDoctor().getLastName());
```

API-urile create

In cele ce urmeaza, se vor prezenta detaliat API-urile create, creionandu-se astfel modul de utilizare al aplicatiei. Prezentarea se va face urmarind functiile definite la nivelul fiecarui controller.

UserController

a) API: /register

```
@PostMapping("/register")
public ResponseEntity<UserDto> createUser(@Valid @RequestBody UserDto
userDto)
```

- Prin apelarea acestui API, se inregistreaza un nou user la nivelul aplicatiei. Pentru aceasta este nevoie de specificarea in Body a detaliilor despre un User(mapate in UserDto).

```
{
    "username": "Antonia12",
    "firstName": "Antonia",
    "lastName": "Ionescu",
    "email": "antonia_ionescu@yahoo.com",
    "password": "antopass234"
}
```

Exista anumite validari la nivelul clasei UserDto care trebuiesc satisfacute pentru a nu intoarce un cod de eroare:

- username/firstname/lastname – lungime > 4
- email – format clasic de email
- parola – lungime > 5

Functia din controller apeleaza functia create(UserDto user, UserType.Pacient) din Service, adaugand la tabela “users” un nou client cu rolul de Pacient.

b) API: /registerAdmin

```
@PostMapping("/registerAdmin")
public ResponseEntity<UserDto> createAdmin(@Valid @RequestBody UserDto
userDto)
```

- Similar cu API-ul anterior, acesta adauga in tabela “users” un utilizator cu rolul de Admin.

c) API: /{username}

```
@GetMapping("/{username}")
public ResponseEntity<UserDto> get(@PathVariable String username)
```

- Plecand de la string-ul specificat in parametru, aceasta reda informatiile despre utilizatorul cerut. In cazul in care acesta nu exista la baza de date, se intoarce un raspuns cu statusul “NO_CONTENT”.

d) API: /users

```
@GetMapping("/users")
public ResponseEntity<List<UserDto>> getAllUsers()
```

- Intoarce informatiile despre toti userii din aplicatie

e) API: /login

```
@PatchMapping("/login")
public String loginUser(@RequestParam String username, @RequestParam
String password, HttpServletResponse response)
```

- Prin acest API se face logarea la nivelul aplicatiei. Specificandu-se un username si un parola, controller-ul apeleaza o functie din service care verfica daca datele de conectare sunt corecte (exista un utilizator la nivelul bazei de date care are credentialele specificate). In caz afirmativ, variabila loggedStatus ia valoarea 1. Inainte de aceasta initilizare, este necesar sa ne asiguram ca niciun alt utilizator nu mai e logat (pentru a lucra cu API-urile urmatoare), de aceea se seteaza variabila loggedStatus = 0 pentru toti ceilalti useri.
- API-ul intoarce un mesaj de “login successful” sau de eroare in caz de esec, impreuna cu un exitcode corespunzator.
-

f) API: /logout

```
@PatchMapping("/logout")
public String logoutUser(@RequestParam String username,
HttpServletResponse response)
```

- Asigura delogarea din aplicatie, setand variabila loggedStatus=0 pentru utilizatorul cu username-ul specificat in request

DoctorController

a) API: /doctorCreate

```
@PostMapping("/doctor/create")
public String createDoctor(@Valid @RequestBody DoctorDto doctorDto,
HttpServletResponse response)
```

- Creeaza un obiect de tip doctor si il adauga in tabele Doctors. Request-ul trebuie sa aiba un body de forma:

```
{
    "firstName": "Mihaela",
    "lastName": "Todeoriu",
    "email": "mihaela_todeoriu@derma.com",
    "specialization": "medic endocrinolog",
    "description": "experienta in endocrinologie",
    "experience": 5,
    "sunday": "10-17",
    "monday": "10-17",
    "tuesday": "10-17",
    "wednesday": "10-15",
    "thursday": "17-19",
    "friday": "17-19",
    "saturday": "17-19"
}
```

Pentru crearea unui doctor este necesar sa fim logati cu un cont de admin. Astfel ca functia sin service-ul asociat, verifica acest fapt. In cazul in care nu sunt logati cu un cont de admin, request-ul intoarce un mesaj de eroare si crearea nu are loc:

```
response.setStatus(404);
return "Doctor cannot be created - you are not logged with
an admin account!";
```

Programul doctorului trebuie sa aiba un format bine-stabilit de tipul HH-HH sau -, aceasta conditie fiind pusa prin aplicarea unui validator:

```
  @Pattern(regexp="^(1[0-9]|20)-(1[0-9]|20)|-
",message="length must be 5, format HH-HH or -")
```

Odata cu crearea unui doctor la nivelul aplicatiei se creeaza si un nou user cu rolul de doctor, avand aceleasi date.

b) API: /doctors

```
@GetMapping("/doctors")
public List<DoctorDto> getAll()
```

- Intoarce lista tuturor doctorilor. Acest lucru se face printr-o interogare a tabelei “doctors”
-

c) API: /doctor/details

```
@GetMapping("/doctor/details")
public ResponseEntity<DoctorDto> getByName(@RequestParam String
firstname,
                                    @RequestParam String lastname)
```

- Intoarce informatiile despre doctorul ce are numele si prenumele specificat in request sau un raspuns de tip “NO_CONTENT” in caz de eroare

d) API: /doctor/update

```
@PatchMapping("/doctor/update")
public String updateDoctor(@Valid @RequestBody DoctorDto doctorDto,
HttpServletResponse response)
```

- Actualizeaza informatiile despre doctor, suprapunand datele cu cele aflate deja in baza de date. La fel ca primul API(crearea de doctori), updatarea se poate face doar de catre adminul aplicatiei, acest fapt verificandu-se la nivelul serviciului.

AppointmentController

a) API: /appointment/create

```
@PostMapping("/appointment/create")
public String createAppointment(@Valid @RequestBody AppointmentDto
appointmentDto
```

- Creeaza o programare la nivelul bazei de date. Body-ul request-ului trebuie sa contina informatii despre user, cat si despre doctor.

```
- {
-     "data": "2022-02-10",
-     "hour": "18",
-     "notes": "fara notite",
-     "firstNameDoctor": "Dana",
-     "lastNameDoctor": "Bratu",
-     "username": "Bianca11",
-     "firstNameUser": "Bianca",
-     "lastNameUser": "Buzoi"
- }
```

Crearea unei programari urmareste mai multi pasi:

- Ne asiguram ca suntem logati cu contul user-ului pentru care se doreste sa se faca programarea
- Ne asiguram ca medicul specificat exista la nivelul bazei de date
- Verificam disponibilitatea doctorului in ziua si ora precizata (se afla in programul sau de lucru in functie de ziua saptamanii si nu exista nicio alta programare la ora specificata)

b) API: /appointment/delete

```
@DeleteMapping("/deleteAppointment")
public String cleanupAppointments(@RequestParam int id ,
HttpServletResponse response)
```

- Utilizat pentru anularea unei programari, plecand de la id-ul acesteia. Schimbarea datei unei programari nu este posibila. Daca se doreste acest fapt, utilizatorul trebuie sa anuleze programarea actuala si sa creeze alta noua.

Celelate API-uri sunt de tip GET si afiseaza programarile in diferite formate: pe useri, pe doctori, pe momente de timp (trecut/viitor):

```
@GetMapping("/appointmentsUser")
public List<AppointmentDto> getByUser(@RequestParam String firstname,
                            @RequestParam String lastname)
@GetMapping("/appointmentsDoctor")
public List<AppointmentDto> getByDoctor(@RequestParam String
firstname,
                            @RequestParam String lastname)
@GetMapping("/futureAppointmentsUser")
public List<AppointmentDto> futureApp(@RequestParam String username)
@GetMapping("/passedAppointmentsUser")
public List<AppointmentDto> passedApp(@RequestParam String username)
```

RequestDonationController

a) API: /donationRequest/create

```
@PostMapping("/donationRequest/create")
public String createDonationRequest(@Valid @RequestBody
RequestDonationDto donationRequestDto, HttpServletResponse response)
```

- Adauga o cerere de donare in tabele “donationRequests”. Body-ul trebuie sa fie de forma:

```
-   {
-       "firstName": "Mihai",
-       "lastName": "Anton",
```

```
-       "hospital": "Spitalul Judetean Pitesti",
-       "description": "accident grav rutier",
-       "bloodType": "A",
-       "number": 5
-   }
```

Grupele de sange pot avea urmatoarele valori : A|B|0|AB, iar acest lucru este verificat prin intermediul unui validator aplicat pe variabila BloodType:

```
@Pattern(regexp="^A|B|0|AB",message="Group should be
A|B|0|AB")
```

b) API: /donationRequest

```
@GetMapping("/donationRequest")
public List<RequestDonationDto> getAll()
```

- Reda lista tututor cererilor pentru donare

DonationAppointmentController

a) API: /donationAppointment/create

```
@PostMapping("/donationAppointment/create")
public String createAppointment(@Valid @RequestBody DonationAppointmentDto
appointmentDto, HttpServletResponse response)
```

- Creeaza o programare pentru donatie la nivelul bazei de date. Body-ul request-ului trebuie sa contina informatii despre user, cat si despre cazul de donare.

```
-    {
-       "data": "2022-02-10",
-       "hour": "16",
-       "firstNameRequestPacient": "Ioana",
-       "lastNameRequestPacient": "Vasile",
-       "username": "Bianca11",
-       "firstNameUser": "Bianca",
-       "lastNameUser": "Buzoi"
-   }
-
```

Crearea unei programari urmareste mai multi pasi:

- Ne asiguram ca suntem logati cu contul user-ului pentru care se doreste sa se faca programarea
- Ne asiguram ca exista pacientul pentru care se doreste sa se doneze sange

- Verificam ca nu exista deja o programare de acelasi tip la aceeasi ora pentru a evita aglomeratia

b) API: /deleteDonationAppointment

```
@DeleteMapping("/deleteDonationAppointment")
public String cleanupDonationAppointments(@RequestParam int id ,
HttpServletResponse response)
```

- Anularea unei programari pentru donatie dupa id

c) API-uri pentru afisarea programarilor:

```
@GetMapping("/donationAppointments")
public List<DonationAppointmentDto> getAllDonationAppointments()

@GetMapping("/donationAppointmentsUser")
public List<DonationAppointmentDto>
getAllDonationAppointmentsByUser(@RequestParam String firstname,
                                 @RequestParam String lastname)
```

Pentru a opri introducerea de date necoerente si eronate in aplicatie au fost folosite urmatoarele validari (aplicate variabilelor tuturor claselor):

- @NotNull
- @Email
- @Pattern
- @NotEmpty
- @Size
- @Min
- @Max
- @DateTimeFormater

Testarea aplicatiei

Testarea aplicatiei s-a facut prin implementarea de teste unitare ce au ca referinta controller-le, servicile si validarile de la nivelul dto-urilor.

Exemplu testare in service:

```
@Test
@DisplayName("Create new pacient")
void create() {
    //arrange (given)
    UserDto userDto = aUserDto("Maria11","Maria","Creanga");
    UserType type = UserType.PACIENT;
    User user = aUser("Maria11","Maria","Creanga");
    User savedUser = aUser(1L);

    when(userMapper.mapToEntity(userDto)).thenReturn(user);
    when(userRepository.save(user)).thenReturn(savedUser);
    when(userMapper.mapToDto(savedUser)).thenReturn(userDto);

    //act (when)
    UserDto result = userService.create(userDto, type);

    //assert (then)
    assertThat(result).isNotNull();
    verify(userMapper, times(1)).mapToEntity(userDto);
    verify(userMapper, times(1)).mapToDto(savedUser);
    verify(userRepository, times(1)).save(user);

}
```

- Se testeaza crearea unui nou user la nivelul aplicatiei. Se verifica ca maparea rezultatului intors de functia create() din serviciu corespunde 1:1 cu cea definita de catre noi.
-

Exemplu testare de API:

```
@Test
@DisplayName("Testing creating a new doctor")
void test_createDoctor_happyFlow() throws Exception {
    //Arrange
    DoctorDto dto = DoctorDtoUtil.aDoctorDto("firstname1","lastname1");
    int resultCreation = 1;
    when(doctorService.create(any())).thenReturn((long) resultCreation);

    //Act
    MvcResult result = mockMvc.perform(post("/doctor/create")
```

```
                    .content(objectMapper.writeValueAsString(dto))
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isAccepted())
            .andReturn();

}
```

- Se verifica ca apeland API-ul ***/doctor/create*** specificand in Body un obiect de tip DoctorDto (valid), rezultatul este ACCEPTED

Exemplu testare validatori:

```
@Test
void test_request_whenFirstName_LastName_isInvalid() {
    request.setFirstName("fs");
    request.setLastName("ln");

    //Act
    Set<ConstraintViolation<DoctorDto>> violations =
validator.validate(request);

    //Assert
    assertThat(violations.size()).isEqualTo(2);
}
```

- Testeaza informatiile unui doctor. Numele si prenumele unui doctor au aplicate o restrictie de lungime minima.

Dezvoltari ulterioare ale aplicatiei

Dat fiind faptul ca aplicatia este la inceput de drum, aceasta poate fi imbunatatita astfel incat sa devina cat mai complexa si sa aduca unui posibil client cat mai multe beneficii. Printre dezoltarile sale ulterioare se numara:

- Implementarea functionalitatii prin care dupa logare, un doctor, poate trimite reteta online catre pacientul cu care acesta are programare.
- Introducerea unui numar de farmacii online si oferirea posibilitatii ca dupa primirea unei retete, pacientul sa poata gasi cea mai apropiata unitate care are pe stoc toate medicamentele prescrise.
- Integrarea unui sistem prin care se pot crea programari online pentru analize medicale si primirea rezultatelor/interpretarea lor direct pe platforma.