Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Рачунарска техника и софтверско инжењерство
Ниво студија: Основне академске студије
Предмет: Програмирање система који раде у реалном времену
Број индекса: 578/2016

Саво Г. Вуковић

Реализација вођења евиденције о пацијентима унутар информационог система за здравство

Дипломски рад

Комисија за преглед и одбрану:

1. **Доц. др Владимир Миловановић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

Циљ завршног рада је реализација система за обраду медицинских података ради евидентирања информација о пацијентима приликом пријема у неку од здравствених установа. Корисници ће бити у могућности да манипулишу са подацима који су везани за одређеног пацијента а сами подаци ће бити део једног централног система у виду веб апликације. Главни акценат ће бити стављен на скалирању датих података, обради података у случају грешке и претрази.

Препоручена литература:

[1] Бранивоје Тимотић, Миомир Јањић: Примарна здравствена заштита, ELIT MEDICA, Београд 2004

[2] Vlad Mihalcea: High Performance Java Persistance, Vlad Mihalcea 2006

[3] Ranga Rao Karanam: Learn Spring 5, Packt Publishing, Mumbai 2017

Крагујевац, јул 2020. Ментор: Доц. др Владимир Миловановић

Садржај

1. Резиме

Систем за вођење евиденције о пацијентима биће реализован у виду веб сервиса као централни систем намењен да подржи интеракцију између корисника ради размењивања информација зарад боље организације здравствене установе. Корисници, који су у овом случају здравствени радници, биће у могућности да манипулишу са подацима везаним за пацијенте у оквиру једне здравствене установе у виду додавања нових пацијената, додавања информација које су везане за саме пацијенте, као и претрага информација које су доступне здравственом раднику. Функционисање система биће детаљно објашњено, док ће приступ прикупљања података у виду складиштења информација на удаљени систем бити поређено са традицијалном методом прикупљања података када су у питању здравствени системи.

Кључне речи: Веб сервиси, прикупљање података, здравстени систем, сигурност, претрага, манипулисање са великом количином података

2. Abstract

System for managing patient data will be realized in the form of a web service as a centralized system intended to supports the interaction between users and the actual system itself to exchange information for the better organization of health institutions. Users, in this case, the medical staff, will have an option to manipulate with patient data in terms of adding new patients, adding new information related to patients as well as searching for information that’s available to the medical worker. The functioning of the system will be explained in detail, while the approach of gathering data in a remote system will be compared with the traditional method of data collection when it comes to health systems.

Keywords: Web services, gathering data, healthcare system, security, searching, HTTP protocol, manipulating with large data.

3. Увод

Напредак технологије и увођење информационих система у здравствене установе отворили су пут за иновације везане за стварање концепта електронског медицинског картона, којим је омогућено да сви подаци о пацијенту буду у електронској форми. Овакав начин чувања информација пружа могућност за значајно побољшање квалитета медицинских услуга и повећање саме ефективности медицинске праксе. Здравствена установа би увођењем оваквог система скратила административни посао на најмањи могући ниво, са потпуно тачним увидом о броју пацијента у самој установи као и о пруженим услугама од стране самог медицинског особља. Поред тога што што се административни део посла смањује на минимум, предности самог система се могу уочити и у другим ставкама:

- Замењује се традиционални начин чувања података у медицинским картонима који често могу бити недовршени и нечитљиви.
- Повећава се сигурност прикупљених података
- Онемогућава се појава дуплираних података
- Смањује се ризик за могућност медицинске грешке
- Доступност евиденције о пруженим услугама

Сам рад ће бити фокусиран на реализацији система за манипулисање оваким типом података где ће посебан акценат бити стављен на ауторизацију, аутентификацију, складиштење самих података о пацијентима, могућност брзе и лаке промене података у најкраћем могућем временском року, претраживање података, као и потпуни увид у податке ради брже анализе рада дате здравствене установе и њених медицинских радника. Систем ће садржати подршку за већи број корисника и пацијената а због дизајна самог система биће прилагодив за било које окружење где се води евиденција о пацијентима од стране запослених медицинских радника.

4. Електронски здравствени систем у Србији

Електронски здравствени систем или е-здравство, представља примену информационо-комуникационих технологија за потребе грађана, пацијената, здравствених стручњака и здравствених установа. Систем као такав представља информациону и управљачку компоненту која подржава процесе здравствене службе у целини. Имплементација таквог ситема омогућава да се прате подаци везани за одређеног пацијента у дужем временском периоду, где се смањује могућност грешке код самих медицинских радника јер су све информације доступне у кратком временском периоду. Србија се налази на почетку развоја е-здравства и преласку са администрације засноване на папиру и картонима на компјутерску обраду података. Процедуре о употреби тако чуваних података треба да буду у складу са правним оквирима е-здравства. Све дијагнозе пацијената из самог система представљају поверљиве податке, и као тавки морају бити тајна и не смеју бити доступни административном особљу нити било којим приватним здравственим установама, чиме се остварује заштита података пацијената.

Србија би увођењем оваквог система имала комплетан увид о здравственом стању својим пацијената, статистици преписаних лекова и терапија, ефикасношћу својих медицинских радника као и функционисању целокупних здравствених установа, што би омогућило ефикаснију набавку медицинских препарата, опреме и лекова, и тиме допринело бољем функционисању самог здравственог система у нашој земљи. Проблематика увида у медицинску документацију, великог броја пацијената у чекаоницама и константних грешака приликом преписивања терапија свела би се на минимум што би утицало на укупно задовољство по питању здравства у целој земљи, јер би тада медицинско особље заслужно за лечење одређених пацијената имало бољи увид у то шта треба урадити за добробит пацијента због бољег организовања саме историје лечења пацијента.

Јединствени централни систем за чување свих података о дијагнозама приликом лечења одређених пацијената, могао би се искористити за стварање целокупне слике о општем здравственом стању наше земље. Коришћењем оваквог система, пацијенти би имали могућност да без проблема добију информације о њиховом лечењу и у случају да желе да промене примарно место лечења, имали би могућност да код новог лекара са собом имају целокупну историју лечења.

5. Радни оквир Спринг (енг. *Spring Framework*)

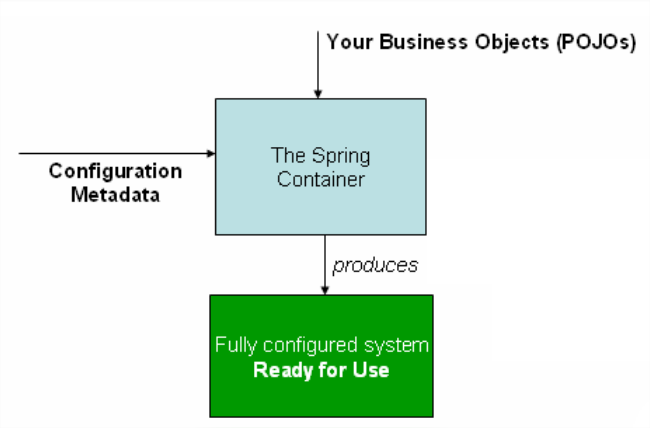
Спринг оквир представља најпопуларнији програмски оквир за развијање пословних апликација у програмском језику Јава. Његова главна сврха је да води рачуна о свим техничким везама које су потребне да би се повезали различити делови апликације, то омогућава програмерима да се фокусирају на свој део посла, писање пословне логике. Једна од главних функционалности програмског оквира Спринг је убризгавање зависности (енг. *Dependency Injection*), познатија као инверзија контроле (енг. *Inversion of control).*

5.1 Убризгавање зависности

Убризгавање зависности је процес у којем објекти дефинишу своје зависности (односно објекте са којима функционишу) само путем аргумената конструктора, аргумената фабричке методе или својства која се постављају на инстанцу објекта након што је објекат конструисан или враћен из фабричке методе. Контејнер затим убризгава дефинисане зависности када креира такозвана зрна (енг. *Beans*). У Спринг оквиру, објекте који представљају кичму наше апликације и који су под контролом контејнера инверзије контроле, називају се зрна. Зрно представља објекат који је инстанциран, склопљен и под контролом контејнера инверзије контроле.

Пакети *org.springframework.beans* и *org.springframework.context* представљају основе контејнера инверзије контроле код програмског оквира Спринг. Важно је напоменути да у поменутим пакетима постоји имплементација интерфејса *BeanFactory,* који омогућава напредни конфигурациони механизам за манипулисање било којим типом објекта и такође пружа конфигурациони оквир и основну функционалност.

Интерфејс *ApplicationContext* представља посебни тип интерфејса *BeanFactory* и фактички представља Спрингов контејнер инверзије контроле заслужан за инстанцирање, конфигурисање и склапање зрна. Контејнер добија своје инструкције од објекта које инстанцира, конфигурише и склапа тако што чита конфигурационе метаподатке. Метаподаци су доступни путем Јава анотација, Јава изворног кода или датотека са екстензијом *XML*.



Слика 1. Спрингов контејнер инверзије контроле

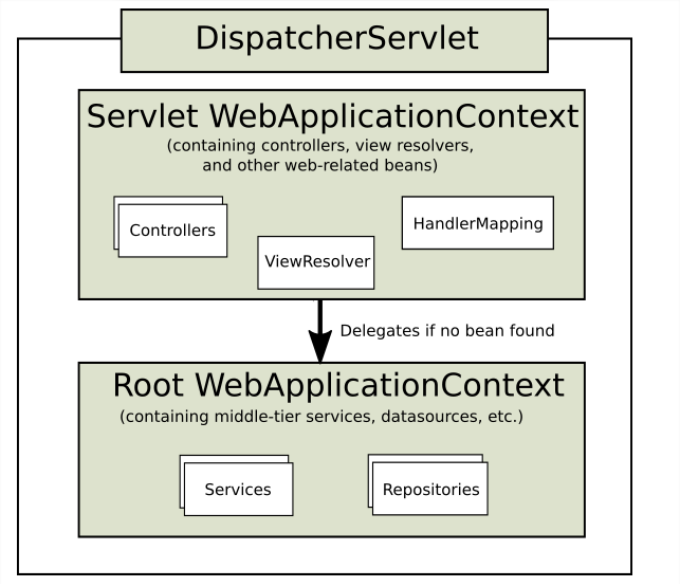
5.2 Радни оквир Спринг Веб (енг. *Spring Web MVC*)

Оригинални Спринг веб оквир изграђен је на сервлет АПИ-ју (апликациони програмски интерфејс) и укључен је у Спрингов оквир од самог почетка. Формални назив „*Spring Web MVC*" потиче од његовог изворног модула (*spring-webmvc*), али је познатији и као „*Spring MVC*". Оквир је дизајниран око централне компоненте која носи назив *DispatcherServlet,* која шаље захтеве својим обрађивачима са подесивим мапирањем и подршком за руковање са датотекама. Подразумевани руковалац захтева се заснива на анотацијама *@Controller* и *@RequestMapping*, нудећи широк спектар флексибилних метода руковања захтева.

5.2.1 Централни диспечер (енг. *DispatcherServlet*)

Спрингов веб оквир, као и многи други популарни веб оквири, дизајниран је око централног контролера који пружа одређени алгоритам за процесуирање захтева од стране клијента, док се стварна логика везана са сам захтев процесуира у делегираним компонентама. Овај модел обрађивања захтева је веома флексибилан и подржава разне моделе приликом дизајнирања веб апликације.

Централни диспечер захтева инстанцу објекат типа *WebApplicationContext* (који представља екстензију обичног *ApplicationContext* објекта) за сопствену конфигурацију, овде можемо видети да диспечер преко веб контекста има везу са контекстом апликације, која има везу са самим сервлетом где је апликација покренута. За многе апликације један веб контекст је сасвим довољан али је такође могуће имати хијерархију контекста са једним централним контекстом који је доступан свим диспечерима који су инстанцирани. Централни контекст најчешће садржи инфраструктуру свих зрна, као што су зрна заслужна за приступ централној бази података или зрна класа потребна за ауторизацију одређених корисника.



Слика 2. Изглед хијерархије диспечер сервлета

5.2.2 Означени контролери (енг. *Annotated Controllers*)

Спрингов веб оквир омогућава модел програмирања заснован на означавању, где компоненте као што су *@Controller* и *@RestController* користе анотације да означе мапирање одређеног захтева, начин изгледа захтева и још много тога. Означени контролери имају флексибилне описе метода који омогућавају лако мапирање захтева и не захтебају наслеђивање посебних класа нити имплементацију посебних интерфејса. На следећој слици приказан је пример означеног контролера:

```
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

Слика 3. Пример означеног контролера

5.2.3 Мапирање захтева (енг. *Request Mapping)*

Коришћењем анотицаије (ознаке) *@RequestMapping* могуће је мапирати захтев који долази од стране централног контролера на методе обрађивача захтева делегираног контролера. Постоје разне методе мапирања захтева по путањи, *HTTP* методи, параметрима захтева, медијским типовима и заглављу самог захтева. Постоје следеће варијанте ознаке *@RequestMapping*, које фактички мапирају одређене *HTTP* методе са методом обрађивача:

- *@GetMapping* – приликом слања *HTTP GET* захтева
- *@PostMapping* – приликом слања *HTTP POST* захтева
- *@PutMapping* – приликом слања *HTTP PUT* захтева
- *@DeleteMapping* – приликом слања *HTTP DELETE* захтева
- *@PatchMapping* – приликом слања *HTTP PATCH* захтева

Наведене ознаке су такозване прилагођене анотације, свака метода контролера се мора мапирати на одређену *HTTP* методу, насупрот анотације за мапирање захтева (енг. *request mapping*) која опслужује све *HTTP* методе. Треба напоменути да је ово још један начин мапирања датих метода с обзиром да сама анотација пружа могућност декларисања *HTTP* методе помоћу аргумента *method.*

5.2.4 Методе руковања (енг. *Handler Methods)*

Методе са ознаком *@RequestMapping* имају веома флексибилан начин декларисања аргумената методе и повратних вредности. У следећој листи биће приказан један део подршке, када су у питању аргументи методе приликом мапирања одређеног контролера:

- Подршка за приступ параметрима УРЛ-а приликом слања *HTTP GET* позива коришћењем анотације *@PathVariable*:

```
@GetMapping("/owners/{ownerId}/pets/{petId}")
public Pet findPet(@PathVariable Long ownerId, @PathVariable Long petId) {
    // ...
}
```

Слика 4. Пример коришћења анотације @PathVariable

- Подршка за приступ телу *HTTP* позива коришћењем анотације *@RequestBody*:

```
@PostMapping("/accounts")
public void handle(@RequestBody Account account) {
    // ...
}
```

Слика 5. Пример коришћења анотације @RequestBody.

- Подршка за приступ поглављу *HTTP* позива помоћу анотације *@RequestHeader*:

```
@GetMapping("/demo")
public void handle(
        @RequestHeader("Accept-Encoding") String encoding,
        @RequestHeader("Keep-Alive") long keepAlive) {
    //...
}
```

Слика 6. Пример коришћења анотације @RequestHeader

- Подршка за приступ вредностима колачића помоћу анотације *@CookieValue:*

```
@GetMapping("/demo")
public void handle(@CookieValue("JSESSIONID") String cookie) {
    //...
}
```

Слика 7. Приступ колачићу помоћу анотације @CookieValue

- Подршка за приступ вредностима сесије помоћу анотације *@SessionAttribute*:

```
@RequestMapping("/")
public String handle(@SessionAttribute User user) {
    // ...
}
```

Слика 8. Приступ вредностима сесије помоћу анотације @SessionAttribute

На следећој слици можемо видети пример потписа методе обрађивача заслужне за конекцију са датим системом у коме се захтев мапира на УРЛ са вредношћу */connect*, сама метода је мапирана да опслужује само *HTTP POST* позиве са телом позива у формату *application/json.* У самом потпису методе можемо видети да је тело *HTTP* позива десеријализовано у објекат типа *LoginRequest* што нам омогућава манипулацију са датим подацима ради аутентификације корисника за приступ апликацији:

```
@RequestMapping(value = "/connect", method = RequestMethod.POST, consumes = APPLICATION_JSON_VALUE, produces = APPLICATION_JSON_VALUE)
public BaseResponse login(@Valid @RequestBody LoginRequest request) {
```

Слика 8. Пример потписа методе заслужне за конекцију са системом

5.3 Радни оквир Спринг Подаци (енг. *Spring Data JPA*)

Дати оквир пружа подршку спремишта за Јава перзистентни АПИ и олакшава развој апликација којима је потребан приступ ЈПА изворима података. Циљ апстракције само оквира је да значајно смањи количину основног кода потребног за имплементацију слоја за приступ подацима. Централни интерфејс Спринг података је *Repository*, потребно је пружити ентитет класе као и идентификатор самог ентитета као аргументе типа интефејса, овај интерфејс делује првенствено као маркерски интерфејс за хватање типова са којима треба радити и као помоћ при откривању интерфејса који дати интерфејс проширују. У датом раду, спремишта углавном имплементирају такозвани *CrudRepository* који пружа основне функционалности за манипулисање са датим подацима. На следећој слици је дат пример поменутог интерфејса:

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // … more functionality omitted.
}
```

Слика 9. Интерфејс *CrudRepository*

5.3.1 Јава перзистентни АПИ (енг. *Java Persistence API)*

По спецификацији, Јава перзистентни АПИ описује управљање релационим подацима у корпоративним Јава апликацијама у којим Јава објекти настављају да постоје иако је процес апликације, који их је направио, угашен. Спецификација сама по себи није алат или оквир, него дефинише скуп концепата који се могу применити било којим алатом или оквиром. Иако се релационо мапирање у почетку базирало на оквиру Хибернације, од тада је еволурирало тако да сада подршава и нерелационе базе података.

5.3.2 Функционалност динамичког креирања упита

Механизам креирања упита уграђен у инфраструктуру оквира Спринг података омогућава креирање ограничавајућих упита над ентитетима спремишта. Прво се дати префикси, дефинисани у самом интерфејсу који наслеђује један од типова *Repository* интерфејса, уклањају, затим започиње рашчлањивање остатака ради креирања датог упита. На следећој слици дат је конкретан пример функционисања датог механизма:

```
1  package com.patienthealthcare.dao;
2
3  import java.util.List;
4
5  import org.springframework.data.repository.CrudRepository;
6
7  import com.patienthealthcare.entities.Patient;
8  import com.patienthealthcare.entities.PatientEntry;
9
10 public interface EntryRepository extends CrudRepository<PatientEntry, Long> {
11
12     List<PatientEntry> findByPatient(Patient patient);
13 }
```

Слика 10. Пример дефинисања динамичких упита

Дат упит састављен је од кључне речи „*find*“ која говори механизму да треба направити упит за претрагу који није везан за мењање самих података, потом иде кључна реч „*by*“ после које иде ентитет по којем треба да се врши сама претрага. С обзиром да је ентитет прегледа пацијента мапиран тако да у самом ентитету постоји страни кључ повезан са табелом пацијента, механизам ће генерисати следећи упит:

```
SELECT entry.* FROM PH_PATIENT_ENTRY entry WHERE entry.PATIENT_ID = ?
```

Слика 11. Пример упита генерисаног од стране Спринг механизма

Механизам подржава сортирање, основне логичке изразе као и могућност на лимитирања количине података који враћа дати упит и самим тим, пружа нам могућност за фино подешавање функционалности наше апликације када је у питању слој апликације заслужан за приступ самој бази података.

5.3.3 Коришћење именованих упита (енг. *Named Queries)*

Коришћење именованих упита за декларисање упита за ентитете је ваљан приступ и добро функционише за мањи број упита. Како су сами упити везани за Јава методе које се извршавају, заправо се може директно повезати упит са методом преко анотације *@Query*, уместо да се додају у класи ентитета. Овај приступ ослобађа класу ентитета од специфичних информација везаних за интерфејс спремишта. На следећој слици је дат пример именованог упита:

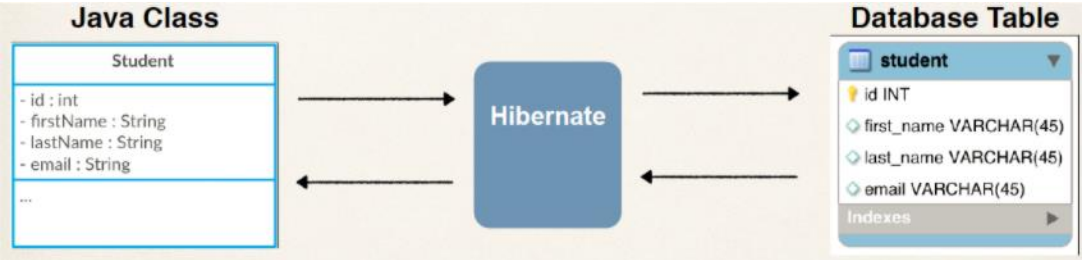
```
14  @Repository
15  public interface UserRepository extends CrudRepository<User, Long> {
16
17      @Modifying
18      @Query(value = "update User user set user.healthInstitution.institutionId = :hiId, user.jobTitle = :jobTitle where user.userId = :userId")
19      void assignUser(@Param("hiId") Long instId, @Param("jobTitle") String title, @Param("userId") Long userId);
20
```

Слика 12. Коришћење именованих упита у спремишту корисника

Овде можемо видети како спремиште корисника наслеђује интерфејс *CrudRepository* и тиме бива декларисано као спремиште Спринг података. На методи *assignUser* налази се поменута анотација са именованим упитом, такође се може приметити да се у потпису метода користи анотација *@Param* која заправо повезује податке из именованог упита са подацима у потпису методе.

5.4 Мапирање помоћу оквира Хибернације

Оквир Хибернација (енг. *Hibernate)* је објектно-релационо решење за мапирање у Јава окружењу. Објектно-релационо мапирање или скраћено ОРМ (енг. *ORM),* је техника програмирања за мапирање објеката апликације у табеле релационе базе података. Хибернација је ОРМ алат заснован на програмском језику Јава који пружа подршку за такву функционалност и то представља једну од кључних функционалности самог оквира. Оквир представља имплементацију Јава перзистентног АПИ-а.



Слика 13. Апстракција функционисања оквира Хибернација

На следећој слици приказан је пример мапирања здравствене институције из Јава објекта у табелу под именом „PH_HEALTH_INSTITUTION“:

```
13  @Entity
14  @Table(name = "PH_HEALTH_INSTITUTION")
15  public class HealthInstitution {
16
17      @Column(name = "ADDRESS", nullable = false)
18      private String address;
19
20      @Id
21      @GeneratedValue(strategy = GenerationType.IDENTITY)
22      private Long institutionId;
23
24      @Column(name = "NAME", unique = true, nullable = false)
25      private String name;
26
27      @Column(name = "PHONE_NUMBER", nullable = false)
28      private String phoneNumber;
29
30      @OneToMany(mappedBy = "healthInstitution")
31      private List<User> users;
32
```

Слика 14. Мапирање здравствене организације

Приликом мапирања датог ентитета здравствене организације можемо приметити коришћење одређених анотација:

- Анотација @Column, користи се за спецификацију одређених вредности мапиране колоне као што су имена, јединственост и да ли нека вредност може да буде празна или не.

- Анотација @Id у комбинацији са анотацијом @GeneratedValue, представља мапирање примарног кључа у датој табели и стратегију креирања нових вредности у датој табели (стратегија приликом мапирања овог ентитета аналогна је додавању AUTO_INCREMENT атрибута приликом креирања примарног кључа)

- Анотација @OneToMany представља бидирекционо мапирање са ентитетом корисника и практично значи да више корисника (здравствених радника), може да припада одређеној здравственој установи. Аргумент *mappedBy* се поставља код неприпадајуће стране приликом мапирања ентитета и представља инфомацију да је страни кључ код ентитета корисника.

У једној здравственој институцији може да ради одређен број здравствених радника, на следећој слици је приказано мапирање ентитета корисника:

```java
@Entity
@Table(name = "PH_USER")
public class User {

    @Column(name = "EMAIL", unique = true)
    private String email;

    @Column(name = "ENABLED")
    private Boolean enabled;

    @Column(name = "FIRSTNAME", nullable = false)
    private String fistName;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "INSTITUTION_ID")
    private HealthInstitution healthInstitution;

    @Column(name = "JOB_TITLE", nullable = false)
    private String jobTitle;

    @Column(name = "LASTNAME", nullable = false)
    private String lastName;

    @Column(name = "PASSWORD", nullable = false)
    private String password;

    @OneToMany(mappedBy = "user")
    private List<Patient> patients;

    @Column(name = "UID", unique = true, nullable = false)
    private Long uid;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "USER_ID")
    private Long userId;

    @OneToMany(mappedBy = "user", cascade = { DETACH, MERGE, PERSIST, REFRESH }, orphanRemoval = true)
    private List<UserRole> userRoles;

    @Column(name = "USERNAME", unique = true, nullable = false)
    private String username;
```

Слика 15. Мапирање ентитета корисника

Енитет корисника поред основних информација као што су корисничко име, лозинка, име, презиме, статус, пословна титула и ЈМБГ, садржи поље за бидирекционо мапирање са ентитетом здравствене установе. Овде можемо видети да један корисник може да има више сигурносних улога као и пацијената. Када су у питању пацијенти и сигурносне улоге, ентитет корисника представља страну која не садржи стране кључеве за асоцијацију датих ентитета тј. ентитет корисника није припадајућа страна.

```java
@Entity
@Table(name = "PH_USER_ROLES")
public class UserRole {

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ROLE_ID", nullable = false)
    private Role role;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_ID", nullable = false)
    private User user;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "USER_ROLE_ID")
    private Long userRoleId;

```

Слика 16. Мапирање ентитета сигурносних улога и корисника

Претходна слика представља табелу везе између корисника и сигурносних улога, како више корисника може да има одређену сигурносну улогу и више сигурносних улога може да припада кориснику, ова табле практично представља *@ManyToMany* анотацију реализовану помоћу две бидирекционе асоцијације са два ентитета.

```java
@Entity
@Table(name = "PH_ROLE")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ROLE_ID")
    private Long roleId;

    @Column(name = "ROLE_NAME")
    private String roleName;

    public Long getRoleId() {
        return roleId;
    }

```

Слика 17. Мапирање ентитета сигурносне улоге

Како свака здравствена установа мора да има медицинско особље, то особље је задужено за старање о пацијентима. Пацијент поред основних информација садржи инфомацију о крвној групи, на које лекове је дати пацијент алергичан (ова информација је веома важна приликом преписивања одређених лекова јер спречавају појаву нежељених дејства), ЈМБГ и ЛБО. Мапирање датог ентитета дато је на следећој слици:

```java
@Entity
@Table(name = "PH_PATIENT")
public class Patient {

    @OneToMany(mappedBy = "patient", orphanRemoval = true, cascade = CascadeType.ALL)
    private List<PatientAllergies> allergiesList;

    @Column(name = "BLOOD_TYPE")
    private String bloodType;

    @Column(name = "DATE_OF_BIRTH")
    private Date dateOfBirth;

    @OneToMany(mappedBy = "patient", orphanRemoval = true, cascade = CascadeType.ALL)
    private List<PatientEntry> entries;

    @Column(name = "FIRSTNAME")
    private String firstName;

    @Column(name = "GENDER")
    private String gender;

    @Column(name = "LASTNAME")
    private String lastName;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PATIENT_ID")
    private Long patientId;

    @Column(name = "PHONE_NUMBER")
    private String phoneNumberString;

    @Column(name = "SECURITY_NUMBER")
    private String securityNumber;

    @NaturalId
    private String uid;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "USER_ID")
    private User user;
```

Слика 18. Мапирање ентитета пацијента

Ентитет пацијента има бидирекционо мапирање са ентитетом алергија пацијента, бидирекционо мапирање са ентитетом корисника и бидирекционо мапирање са ентитетом доласка пацијента (аналогно картону пацијента).

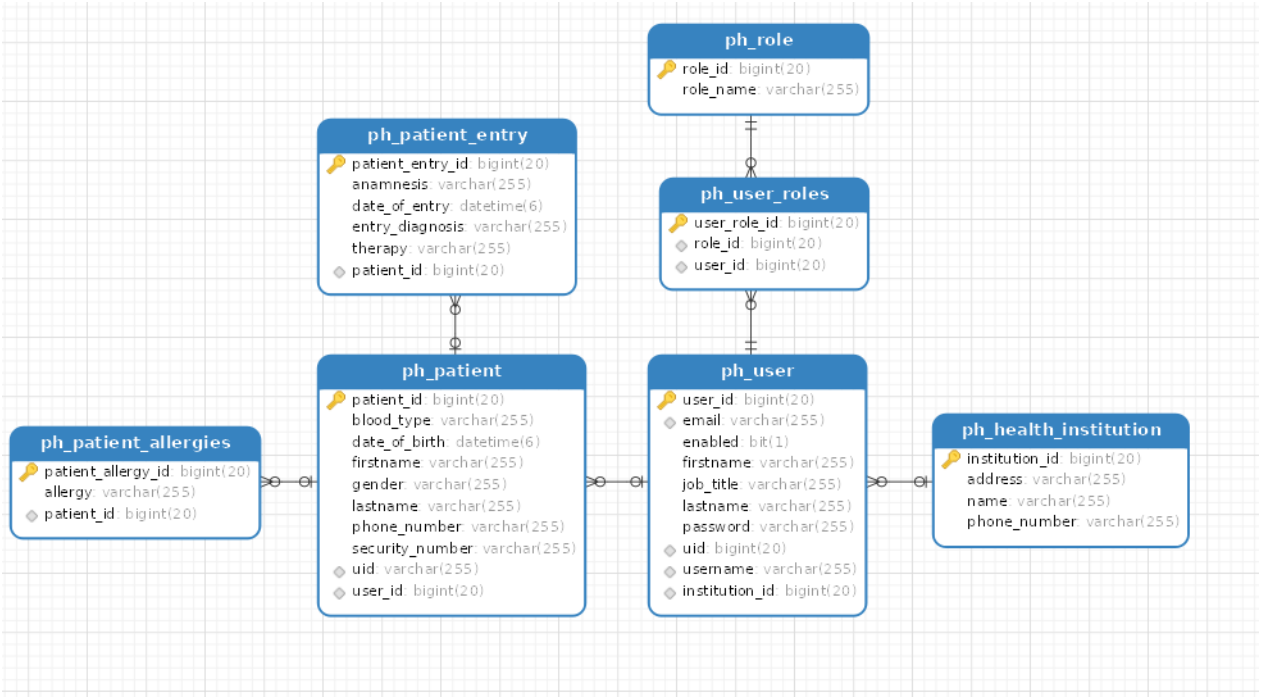
```java
@Entity
@Table(name = "PH_PATIENT_ENTRY")
public class PatientEntry {

    @Column(name = "ANAMNESIS")
    private String anamnesis;

    @Column(name = "DATE_OF_ENTRY", nullable = false)
    private Date dateOfEntry;

    @Column(name = "ENTRY_DIAGNOSIS")
    private String diagnosis;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PATIENT_ENTRY_ID")
    private Long entryId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "PATIENT_ID")
    private Patient patient;

    @Column(name = "THERAPY")
    private String therapy;

```

Слика 19. Мапирање ентитета картона пацијента (доласка пацијента)

```java
@Entity
@Table(name = "PH_PATIENT_ALLERGIES")
public class PatientAllergies {

    @Column(name = "ALLERGY")
    private String allergy;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "PATIENT_ID")
    private Patient patient;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "PATIENT_ALLERGY_ID")
    private Long patientAllergiesId;

```

Слика 20. Мапирање ентитета алергија пацијента

Приликом покретања апликације уколико је уколико је у конфигурационој датотеци својству *spring.jpa.hibernate.ddl-auto* додељена вредност *update*, оквир Спринг података ће покупити вредност датог својства и проследити оквиру Хибернације и то ће омогућити генерисање шеме наше базе података на основу метаподатака добијених приликом мапирања ентитета Јава објеката. База података ће на основу података добијених из мапираних енитета имати следећи изглед:



Слика 21. Физичка шема базе података

Ова функционалност омогућава лакше покретање саме апликације без обзира на то ко је клијент и да ли је неко стручан задужен за подешавање окружења за нормално функционисање. У конфигурационој датотеци постоје следећа својства која омогућавају фино подешавање окружења у којем функционише дата апликација:

- *spring.datasource.driver-class-name*=org.mariadb.jdbc.Driver, драјвер заслужан за приступање нашој бази података.

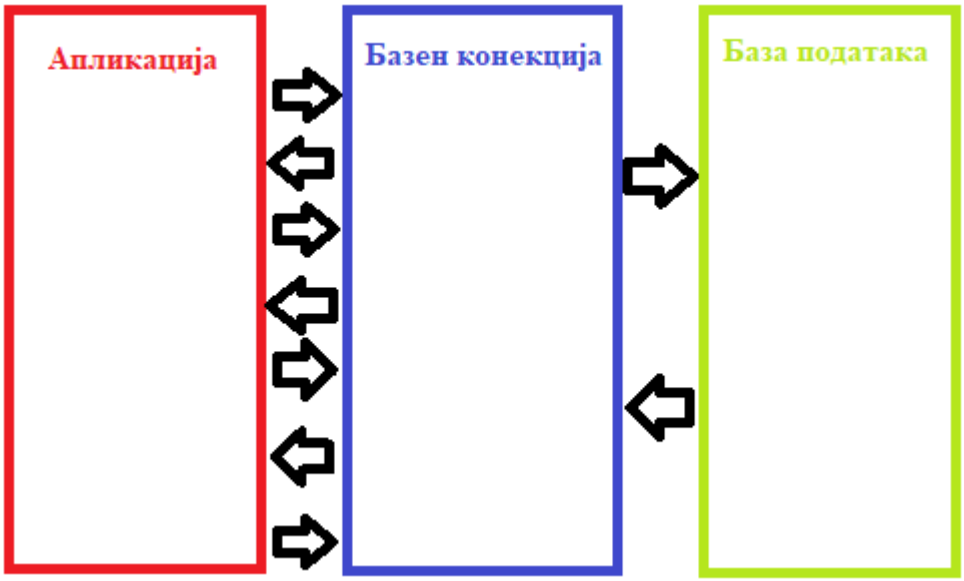
- *spring.datasource.url*=jdbc:mariadb://localhost:3306/patient_healthcare, јединствена путања до наше базе података.

- *spring.datasource.username*=savo, кориснички налог заслужан за приступ бази података.

- *spring.datasource.password*=шифра!, корисничка лозинка потребна за приступ бази података.

5.5 Базен конекција ХикариЦП (енг. *Connection Pool „HikariCP“)*

Базен конекција је техника која се користи за побољшавање перформанси у апликацијама са динамичким садржајем вођеним базом података. Отварање и затварање веза са одређеном базом података представља велики проблем у систему са великим бројем корисника. Базени конекција (енг. *Connection Pools*), су у основи кеш меморија отворених веза према бази података, једном када се отвори веза према бази података и када се та веза врати, уместо да се затвори, она се директно враћа у базен конекција за поновно коришћење. Уколико корисник захтева опет везу према бази података, оквир проверава да ли је одређена веза доступна и враћа кориснику ту већ постојећу везу уместо успостављања нове.



Слика 22. Пример функционисања базена конекција

„HikariCP“ је веома брз и лаган базен конекција базиран на програмском језику Јава. Имплементација датог оквира је веома мала и отвореног кода док је сама функционалност базена високо оптимизована. Коришћењем базена конекција добијамо могућност да лимитирамо број конекција према бази података, да приликом наглог повећања захтеба према апликацији лимитирањем броја конекција форсирамо да апликација застане док конекција не буде доступна. Сам оквир долази са подразумеваним подешавањима које можемо подесити преко одређених параметара у *application.properties* конфигурационом фајлу:

- *spring.datasource.hikari.connectionTimeout*=30000, максимално дозвољено време држања конекције.

- *spring.datasource.hikari.idleTimeout*=600000, максимално дозвољено у којој конекција може бити у базену конекција а да није активна.

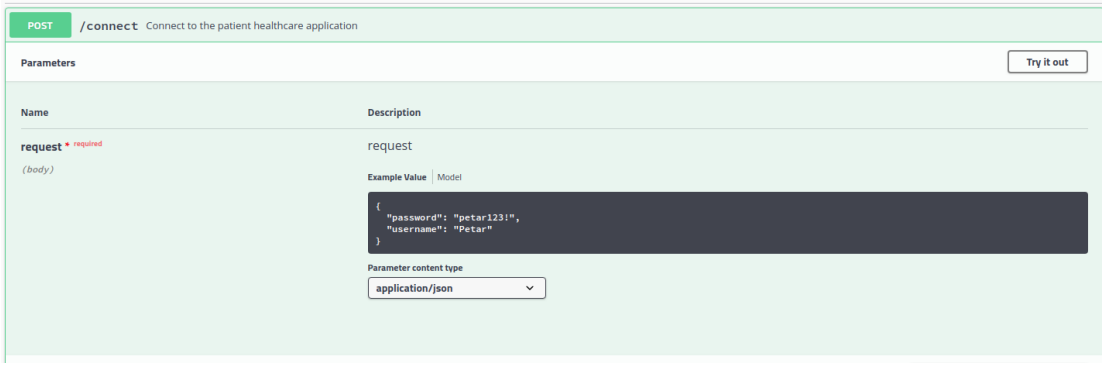
- *spring.datasource.hikari.maxLifetime*=1800000, максимално дозвољено време трајања конекција у датом базену конекција.

6. Реализација апликације за вођење евиденције

Веб сервис апликација за реализацију вођења евиденције о пацијентима намењена је да буде смештена на неком удаљеном серверу, са основном наменом да пружи подршку за интеракцију између здравстених организација и саме апликације. Здравственим радницима би се онда пружила могућност за олакшано обављање администрације а купцима апликације би се пружила могућност да имају потпуну евиденцију о ефикасности својих организација.

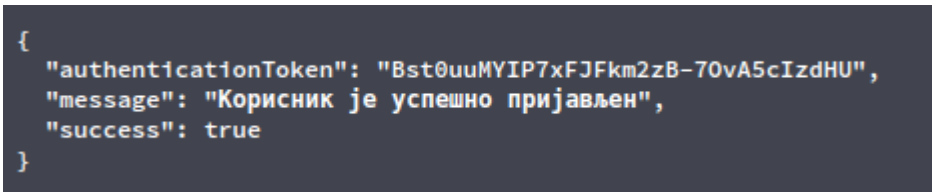
6.1 Аутентификација

Аутентификација корисника реализована је помоћу коришћења аутентификационих токена, генерисаних од стране саме апликације. Уколико корисник (здравствени радник) жели да приступи одређеним ресурсима наше апликације он мора пошаље *HTTP* захтев са заглављем „X-PH-API-Key“, где дато заглавље садржи вредност аутентификационог токена. Када корисник жели да се аутентификује ради добијања аутентификационог токена, он мора да приступи следећем ресурсу и да унесе одговарајућу лозинку и корисничко име:



Слика 23. Крајња тачка обрађивача за конекцију са апликацијом

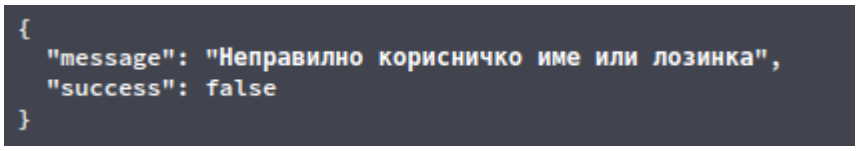
Као што се може приметити, крајња тачка не захтева аутентификациони токен и практично представља једини део апликације, поред наравно крајње тачке за одјаву са апликације, који нема укључену сигурност за верификовање аутентификованих корисника. Аутентификација функционише тако што се од корисничког имена и лозинке формира одређени токен, који се затим прослеђује одређеном објекту који је одговоран за аутентификацију корисника, корисничка лозинка се енкриптује регистрованим зрном за енкрипцију и такве информације се пореде са информацијама које се налазе у самој бази података. Уколико је корисник успешно повезан са апликацијом добиће повратну поруку, која у зависности са подразумеваним језиком апликације, изгледа као на следећој слици:



Слика 24. Повратна порука успешно аутентификованог корисника

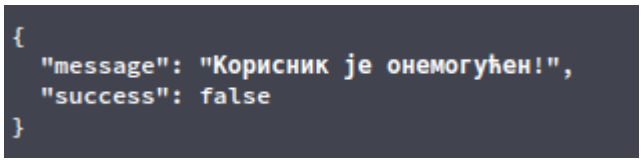
У зависности од тога које информације је корисник унео приликом слања захтева за аутентификацију, повратна порука може имати следећи облик:

- Погрешно унети креденцијали:



Слика 25. Повратна порука за неуспешно повезивање

- Повратна порука да је корисник онемогућен од стране администратора:

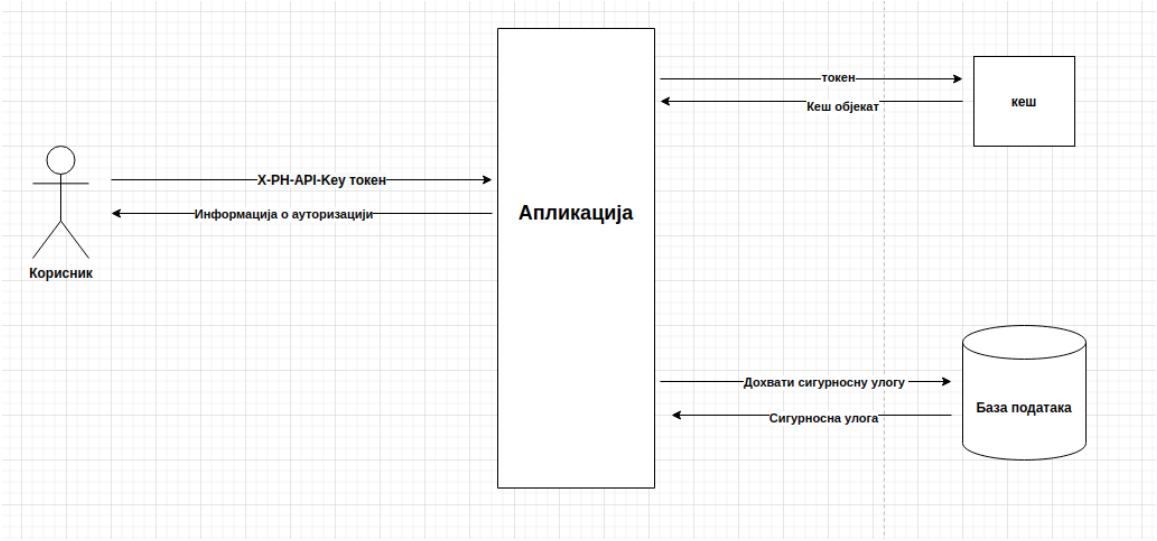


Слика 26. Повратна порука да је корисник онемогућен

Једна од главних функционалности када је у питању повезивање са апликацијом је да приликом успешног повезивања, аутентификациони токен од одређеног корисника бива смештен у конфигурисани кеш. Механизам је имплементиран тако да побољша перформасе самог система и смањи приступ самој бази података тиме што ће аутентификационе токене стављати у дати кеш објекат, где се трајање токена поклапа са његовим веком трајања у самом кешу. Када се аутентификациони токен не искористи у унапред задатом временском интервалу, он се аутоматски избацује из кеша и самим тим тај корисник више није аутентификован. Приликом аутентификације, корисник се извлачи из базе података и од информација као што су његов идентификациони број, кориснички налог, име и презиме, се прави посебан кеш објекат, који се затим ставља у поменути кеш. На овај начим можемо унапред знати који корисник извршава позиве ка нашој апликацији, на основу датог токена који је убачен у привремени кеш приликом аутентификације.

6.2 Ауторизација

Ауторизација је процес који се дешава када се корисников идентитет успешно аутентификује од стране апликације, што доводи до тога да корисник добија приступ ресурсима, додуше, само ресурсима за које има приступ. Ауторизација у апликацији је реализована тако што је направљен објекат (зрно), које пресреће било који захтев, а да није у питању повезивање или одјава, и извлачи кеш објекат повезан са датим аутентификационим токеном у заглављу захтева. Затим, на основу токена извлаче се корисничке информације на основу којих се доноси одлука да ли тај специфичан корисник има приступ нашој апликацији.



Слика 27. Приказ процеса ауторизације корисника

Као што је приказано на претходној слици, за сваког корисника је повезана одређена сигурносна улога. Апликација подржава два типа сигурносне улоге, са темељом за олакшано додавање нових сигурносних улога по потреби, на основу којих се врши ауторизација датих корисника. Подржане сигурносне улоге су:

- *Role Admin*, представља администратора апликације који има врховну функцију одговорну за нормално функционисање саме апликације. Админ корисник има могућност додавања нових здравствених радника, додавање нових институција, додавање нових корисника, мењање пословних титула здравствених радника као и онемогућавање корисника.

- *Role User*, представља здравственог радника на нашој апликацији, здравствени раник има функционалност да види кориснике којима је он изабрани здравствени радник, могућност да види све здравствене раднике у одређеној институцији, да додаје пацијенте, додаје алергије на медицинска средства одређеном пацијенту, могућност да додаје долазак одређеног пацијента тј прегледе, види прегледе везане за одређеног пацијента и много тога још.

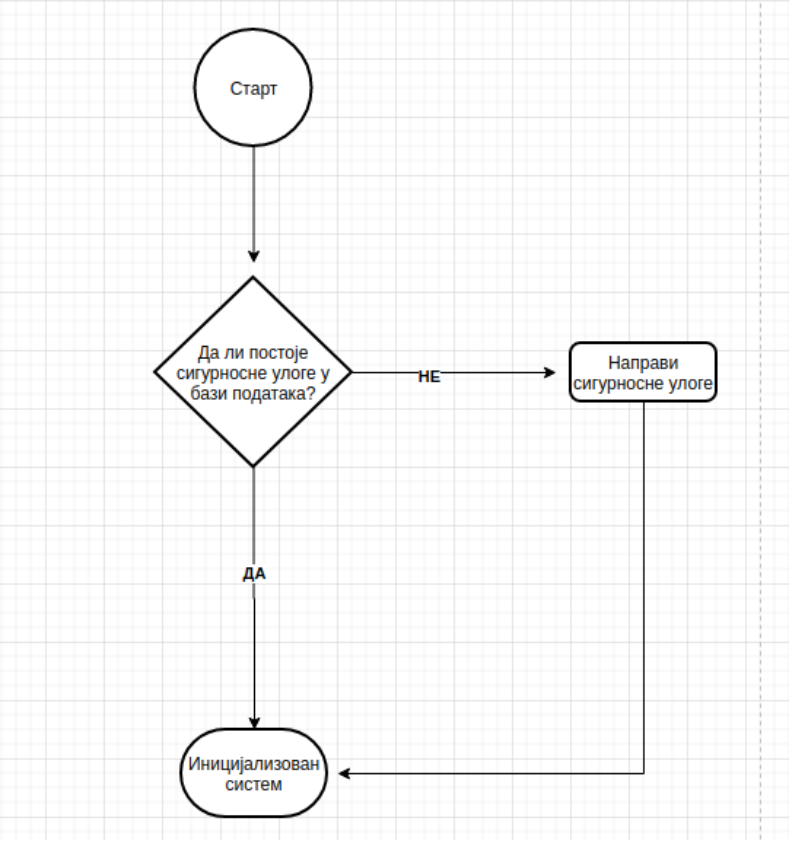
Уколико здравствени радник нема привилегију за обављање одређене акције, као што је на пример додавање здравствене установе, следећа повратна порука ће бити враћена од стране нашег система:


```
{
  "message": "Немате привилегије за обављање дате акције!",
  "success": false
}
```


Слика 28. Повратна порука за неуспешну ауторизацију

6.3 Иницијализација система

Приликом сваког покретања апликације, ради лакшег подешавања саме инстанце, аутоматски се покреће метода *afterPropertiesSet()* која се налази у класи *Setup*. Дата метода се покреће после иницијализације свих зрна у апликацији од стране *BeanFactory* објекта и олакшава подешавање инстанце за лакше коришћење од стране самих корисника. На следећој слици приказан је дијаграм процеса иницијализације:



Слика 29. Процес иницијализације система

6.4 Подршка за језике у апликацији

Апликација је моделирана тако да може да се примени на било који здравствени ниво (примарни, секундарни или терцијарни) у држави Републици Србији, али због свог дизајна може се лако применити на било који здравствени систем, са малим изменама, у већини земаља у свету. Једна од функционалности која подржава ову констатацију је управо подршка за више језика. На следећој слици, приказана је конфигурациона датотека везана за саму апликацију:

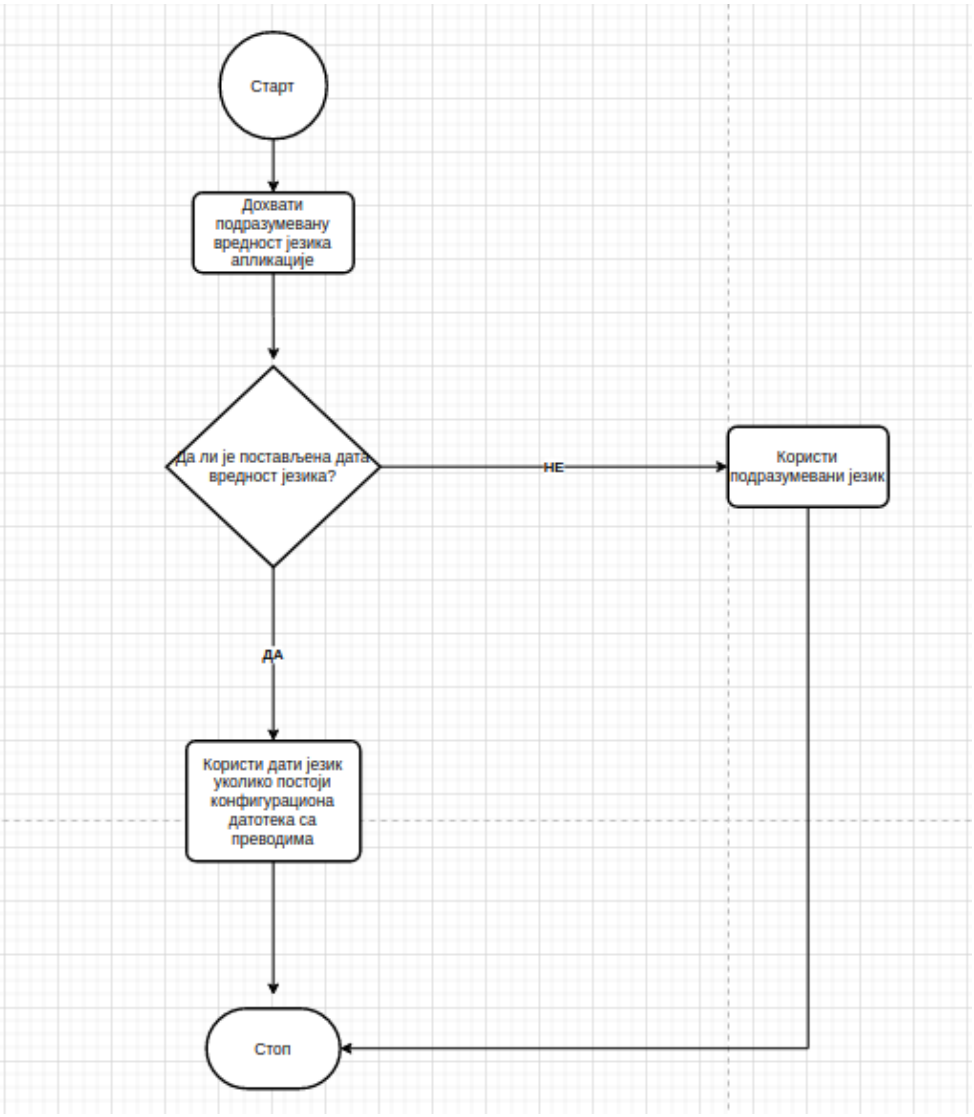
```
1  ### Data source properties ###
2  spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
3  spring.datasource.url=jdbc:mariadb://localhost:3306/patient_healthcare
4  spring.datasource.username=savo
5  spring.datasource.password=savacar123!
6  spring.jpa.hibernate.ddl-auto=update
7
8  ### Cache configuration ###
9  spring.cache.jcache.config=classpath:cache.config/ehcache.xml
10
11 ### Application properties ###
12 server.servlet.context-path=/patient-healthcare
13
14 # Application language property
15 # Default value: RS (Serbia)
16 ph.preferred.language=rs
17
18 # Maximum input allergies size
19 patient.allergies.maximum.input.length=5
20
21 # Therapy expiration threshold
22 # Default value: 14 (days)
23 patient.therapy.threshold=14
```

Слика 30. Конфигурациона датотека

Својство које је задужено за језик апликације приказано је на линији 16. Може се видети да је подразумевана вредност самог својства *RS*, што представља код државе. У апликацији, додата је подразумевана подршка за два језика (енглески и српски), док је омогућено да се дода још и подршка за друге језике додавањем конфигурационе датотеке за поруке у следећем формату:

- *messages_*<име кода државе>, ову датотеку је потребно ставити у јавни директоријум *resources* и променити циљано својство да има следећу форму *ph.preferred.language*=<име кода државе>.

Механизам за одређивање подразумеваног језика за апликацију представљен је следећим дијаграмом:



Слика 31. Процес иницијализације подразумеваног језика

Конфигурациона датотека заслужна за све преводе на апликацији садржи својства, која се мапирају у одређену повратну поруку на основу акције које је извршио дати корисник. Пример формата неколико таквих својства дат је на следећој слици::

```
1  BadCredentialsMessage=Неправилно корисничко име или лозинка
2  SuccessfulLogin=Корисник је успешно пријављен
3  SuccessfulLogout=Корисник је успешно одјављен
4  UnsuccessfulLogout=Није пронађен активан корисник са датим токеном
```

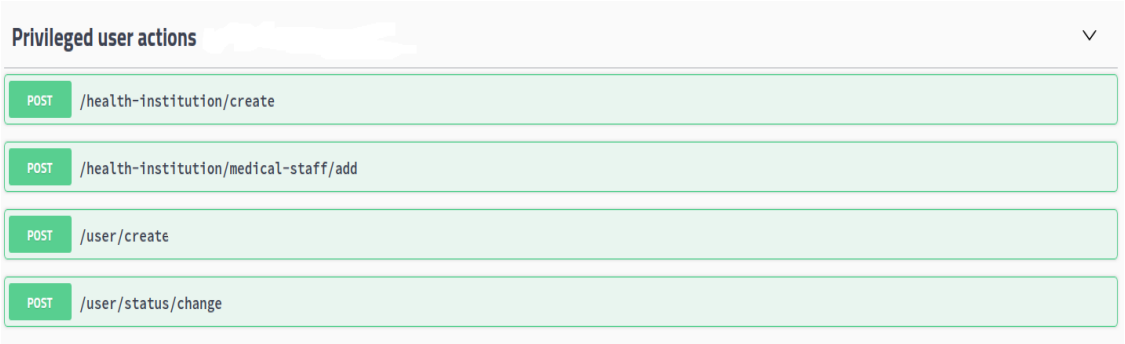
Слика 32. Пример својства у конфигурационој датотеци за језике

6.5 Акције везане за административног корисника

Апликација је базирана на систему где један корисник, по могућству их може бити и више, има право на извршавање акција које га чине привилегованим. Приликом стартовања апликације сваки пут се покреће иницијализација самог система, ова иницијализација проверава да ли је ово први пут покретање апликације, приликом чега ће се креирати сигурносне улоге а и административни корисник. Административни корисник има право на извршавање следећих привилегованих акција:

- Креирање здравствених институција
- Креирање корисника
- Додељивање здравствене институције као и пословне титуле кориснику
- Онемогућавања и омогућавање корисника

На следећој слици су приказане документоване крајње тачке за чије извршавање има право само административни корисник:

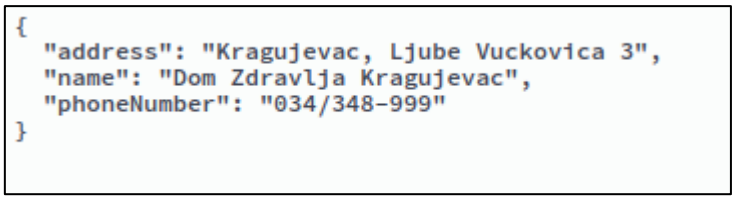


Слика 33. Документоване крајње тачке привилегованих акција

Код документованих крајњих тачака можемо видети УРЛ где корисник мора послати *HTTP* захтев као и *HTTP* методу која иде са самим захтевом.

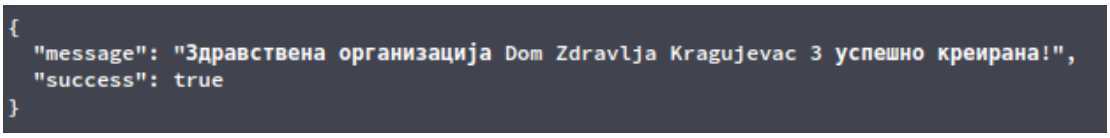
6.5.1 Привилегована акција креирања здравствене институције

Као што се може и претпоставити, креирање здравствене институције у систему захтева привилегије које поседује само административни корисник. Уколико административни корисник жели да направи одређену здравствену институцију, мораће да пошаље *HTTP POST* захтев на одговарајући УРЛ са карактеристичном структуром у телу самог позива. Пример слања захтева за креирање здравствене институције дат је на следећој слици:



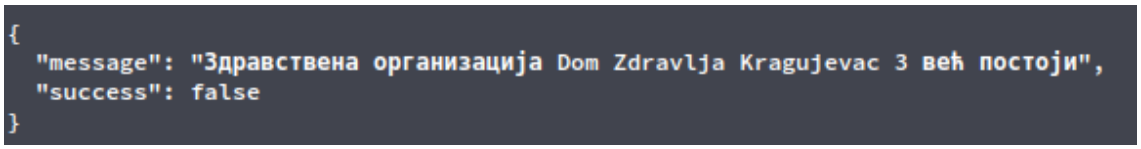
Слика 34 Пример захтева за креирање здравствене организације

Поред информација о адреси, имену и броју телефона саме институције, у заглављу *HTTP* позива на одређени УРЛ, потребно је ставити вредност аутентификационог токена приликом повезивања са апликацијом као административни корисник. Име здравствене организације мора да буде јединствено и потребно је послати све параметре у телу позива. Уколико је успешно креирана здравствена организација, повратна порука ће изгледати као што је приказано на следећој слици:



Слика 35. Пример успешног креирања здравствене организације

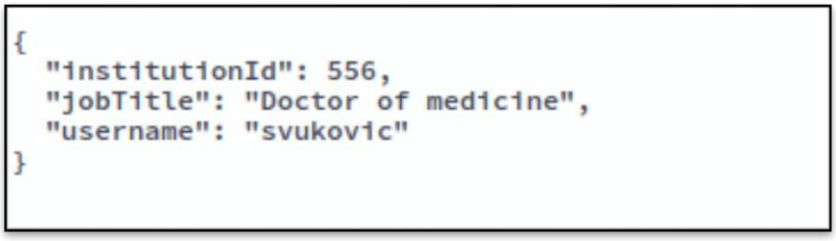
Додуше уколико је наведено име здравствене организације која већ постоји у датом систему, повратна порука ће изгледати као на следећој слици:



Слика 36. Пример неуспешног креирања здравствене организације

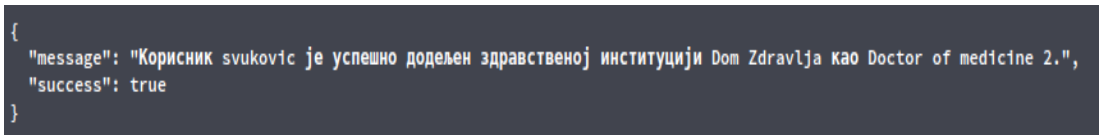
6.5.2 Додељивања здравстеног радника одређеној здравственој институцији

Уобичајено је да здравствени радник промени радно место или бива унапређен на постојећем. Ова акција омогућава да административни корисник наведе здравствену институцију преко јединственог идентификатора, јединствено корисничко име здравственог радника и будућу титулу на послу која ће бити додељена том здравственом раднику у наведеној здравственој институцији. Уколико административни корисник жели успешно да кориснику додели одређену титулу у датој организацији, пример захтева дат је на следећој слици:



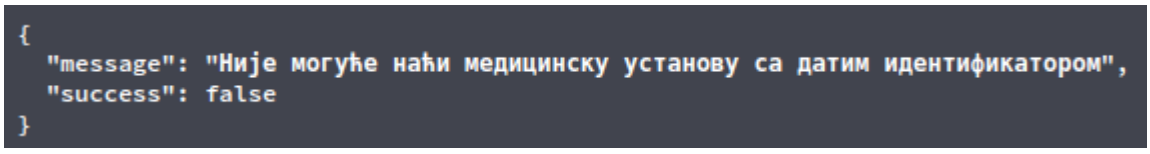
Слика 37. Пример захтева за додељивање здравствених радника здравственој институцији са одређеном титулом

Уколико је захтев за промену здравствене институције био успешан, повратна порука као на следећој слици ће бити прикана:

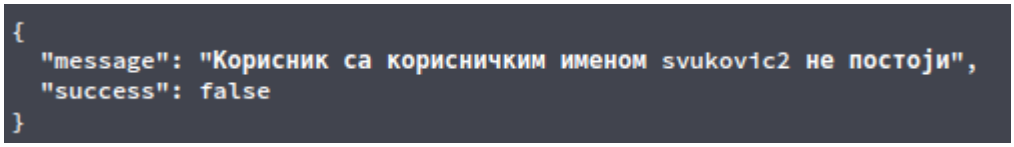


Слика 38. Порука успешне акције додељивање корисника

Неуспешно додељивање корисника се може догодити уколико административни радник унесе јединствени идентификатор који није повезан ни за једну здравствену институцију или уколико административни радник унесе корисничко име које не постоји у нашој бази података. Примери неуспешних порука приликом додељивања корисника здравственој организацији, приказани су на следећим сликама:



Слика 39. Неуспешна порука приликом прослеђивања погрешног идентификатора



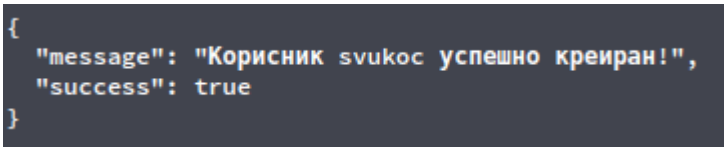
Слика 40. Неуспешна порука прилико прослеђивања погрешног корисничког имена

6.5.3 Привилегована акција креирања здравствених радника

Главна ставка у нашој апликацији, поред пацијената, су заправо здравствени радници. Здравствени радник манипулише са информацијама везаним за одређеног пацијента. Административни радник има привилегију креирања корисника, пример захтева на одређени УРЛ, дат је на следећој слици:

Слика 41. Пример захтева за креирање корисника

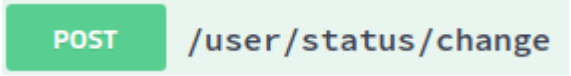
Уколико корисничко име, ЈМБГ (UID) и идентификатор здравствене установе садрже вредности које нису исправне, повратна порука ће бити потпуно идентична порукама са већ приказаних слика. Успешно креирање корисника генерише следећу поруку:



Слика 42. Порука за успешно креирање корисника

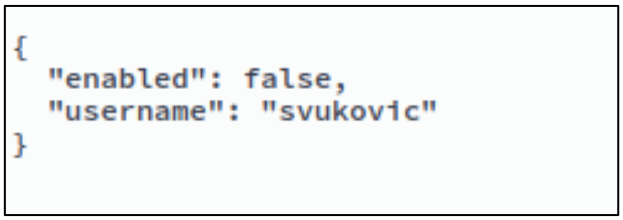
6.5.4 Привилегована акција промене статуса корисника

Уколико административни радник, из неког разлога, жели да онемогући кориснику приступ апликацији потребно је послати два параметра на одговарајући УРЛ који је дат на следећој слици:



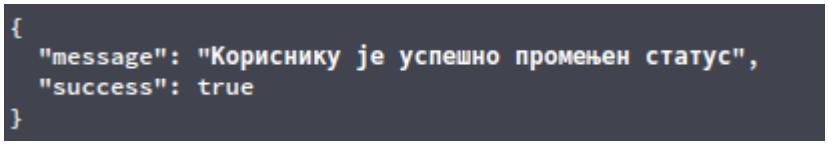
Слика 43. Крајња тачка за промену статуса корисника

У телу позива потребно је поред корисничког имена, навести да ли корисник треба да буде омогућен или не, пример захтева је дат на следећој слици:



Слика 44. Пример захтева за промену статуса кориснику

Уколико дати корисник не постоји, одговарајућа повратна порука ће бити приказана док уколико је успешно онемогућен корисник, одговор ће бити следећи:



Слика 45. Пример успешне промене статуса кориснику

6.6 Акције везане за здравственог радника

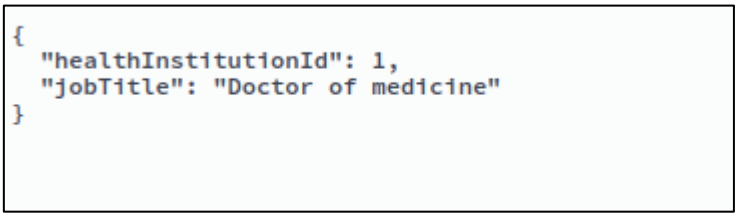
Док административни корисник има функцију додавања потребних ствари за нормално функционисање апликације, корисник тј. здравствени радник, манипулише са подацима који су уско везани за саме пацијенте. Уколико је корисник здравствени радник он има могућност за обављање следећих акција:

- Дохватање свих здравствених радника у одређеној здравственој институцији
- Дохватање свих пацијента у одређеној здравственој институцији
- Додавање алергија пацијенту
- Креирање пацијента
- Додавање евиденције о прегледу пацијента
- Претрага свих евиденција доласка одређеног пацијента
- Промена корисничке имејл адресе
- Преглед свих пацијента који припадају одређеном здравственом раднику.
- Додељивање примарног лекара пацијенту

Овим методама, здравственом раднику се омогућава нормално функционисање кад је у питању администрација везана за пацијенте.

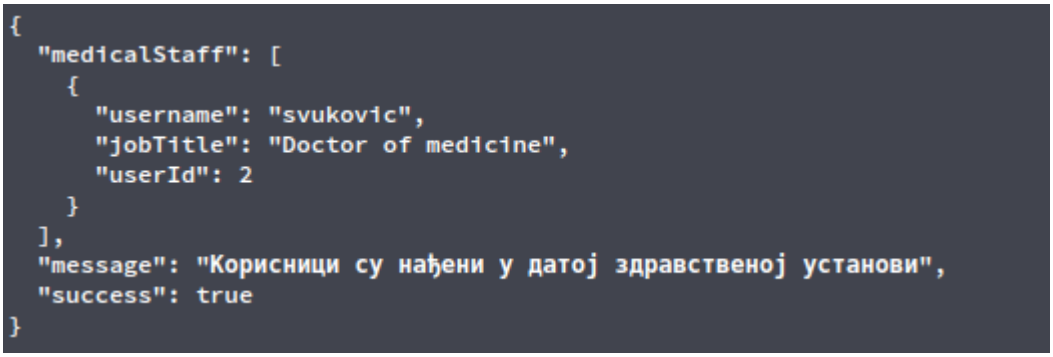
6.6.1 Претрага медицинских радника у здравственој институцији

Једна веома корисна метода за здравствене раднике је претрага одређених здравствених радника у институцији (у којој је запослен дати корисник), по струци. Пример једног таквог захтева дат је на следећој слици:



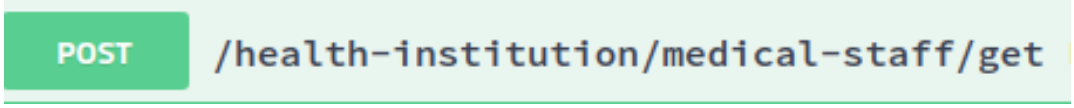
Слика 46. Пример захтева за претрагу особља

Овде здравствени радник има право да врши претрагу само у здравственој институцији којој припада, уколико пошаље идентификатор неке друге здравствене институције, одговарајућа порука ће бити враћена. Пример успешног завршетка акције дат је на следећој слици:



Слика 47. Пример успешне претраге особља у здравственој институцији

На основу ове операције, корисник може видети тачно колико особља са одређеном струком има у здравственој установи. Ова акција може бити корисна за онај тип медицинског особља који је одговоран за примање нових лекара или који организује рад самих здравствених радника. Потребно је само послати *HTTP POST* позив на следећи УРЛ заједно са аутентификационим заглављем:



Слика 48. УРЛ за дохватање медицинског особља

6.6.2 Претрага регистрованих пацијената у одређеној здравственој установи

Приликом несреће, одређени пацијенти са специфичном крвом групом, могу бити у проблему уколико нема одговарајућих крвних јединица у тренутној здравственој установи. Ова метода омогућава претрагу свих пацијената, у одређеној институцији, на основу које се могу видети следеће информације:


```
{
  "count": 1,
  "patients": [
    {
      "bloodType": "AB+",
      "dateOfBirth": "1970-01-19T12:54:03.682+00:00",
      "firstName": "Savo",
      "lastName": "Vukovic",
      "phoneNumber": "+318600000",
      "securityNumber": "2123314541",
      "uid": "020299629911"
    }
  ],
  "success": true
}
```


Слика 49. Повратн порука дохватања пацијената у одређеној установи

Као што можемо видети, осим основних информација везаним за датог пацијента, можемо видети информације као што су крвна група, старост, ЈМБГ и остало. Ово може бити веома корисно када су у питању специфичне ситуације где су ове информације преко потребне. Потребно је само послати јединствени идентификатор на следећи УРЛ:

```
POST  /health-institution/patients/get
```

Слика 50. Крајња тачка за дохватање пацијената

Уколико здравствена институција са датим идентификатором не постоји, биће враћена одговарајућа порука приказана на једној од претходних слика.

6.6.3 Акције везане за манипулацију података везаних за пацијенте

Централни део апликације представљају информације добијене од самих пацијената приликом пријема у једну од здравствених институција. Без обзира који је здравствени ниво у питању, пацијента је потребно првобитно додати у здравствену организацију (уколико већ не постоји), затим на основу приче са пацијентом, додати алергије на лекове које ће помоћи приликом писања дијагнозе, и на крају, извршити преглед чије резултате такође треба сачувати у нашу базу података. Медицински радник прво мора да омогући пацијенту да изабере одређеног изабраног лекара и на основу тога да додели пацијенту тог медицинског радника.

У следећој листи приказане су основне опције које има здравствени радник приликом рада са пацијентима:

- Додавање пацијента
- Додељивање здравственог радника пацијенту (лекара опште праксе или лекара специјалисту)
- Претрага свих пацијената који припадају одређеном лекару
- Претрага пацијентовог досијеа заједно са додавањем нових података у сам досије.
- Додељивање алергија на лекове одређеном пацијенту (веома важна ставка када се пацијенту врши дијагностика)

На следећој слици приказани су документоване крајње тачке које омогућавају обављање наведених функција од стране здравствених радника:

Слика 51. Комплетна листа доступних акција здравствених радника

6.6.4 Додавање алергија на лекове одређеном пацијенту

Алергија на лекове је неприродна реакција имуног система на одређене лекове. Било какав лек (било без рецепта, на рецепт или биљни лек) може довести до алергијске реакције, а најчешћи симптоми су копривњача, осип или температура. Алергија на лекове може довести до компликација попут алергијских реакција, отока и астме, који с друге стране могу да делују на функционисање неколико система органа. Зато је веома важно обавестити здравствене раднике о поменутим подацима како би се обезбедило одговарајуће лечење. Здравственом раднику је управо због тога обезбеђена функција самог система за додавање алергија одређеном пацијенту. Уколико лекар жели да дода алергију одређеном пацијенту, потребно ја да *HTTP POST* позивом, са одговарајућим параметрима у телу захтева, интерагује са одређеном крајњом тачком нашег контролера. На следећој слици, приказан је пример тела захтева за ову крајњу тачку:

```
{
  "allergies": [
    "Penicillin",
    "Ibuprofen"
  ],
  "patientUid": "0202991120026"
}
```

Слика 52. Пример тела позива за додавање алергија пацијенту

Уколико јединствени матични број пацијента није нађен у бази података, апликација ће вратити следећу повратну поруку:

```
{
  "message": "Није могуће наћи пацијента са датим идентификатором",
  "success": false
}
```

Слика 53. Пример повратне информације система приликом слања лоше вредности јединственог матичног броја грађана

Када су алергије успешно додате одређеном пацијенту, апликација враћа следећу поруку о успешно извршетку акције:

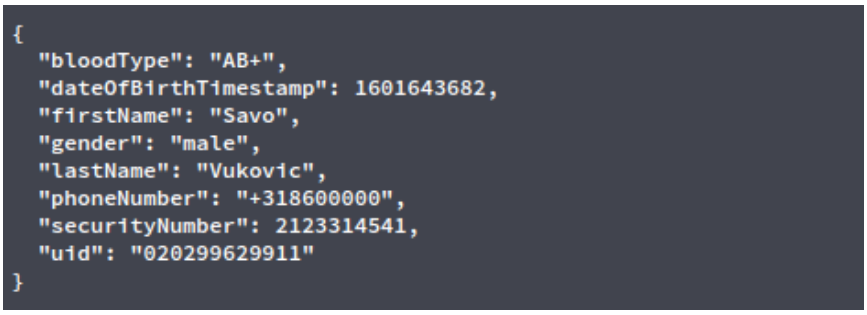
```
{
  "message": "Алергије [Penicillin, Ibuprofen] успешно додате пацијенту са ЈМБГ 020299629911",
  "success": true
}
```

Слика 54. Порука успешно додатих алергија пацијенту

На овај начин олакшан је посао здравственом раднику, јер се подаци, везани за тог одређеног пацијента, чувају чак иако пацијент промени здравствену установу или изабраног лекара и на тај начин смањује се вероватноћа за прављење грешака приликом писања одређене терапије.

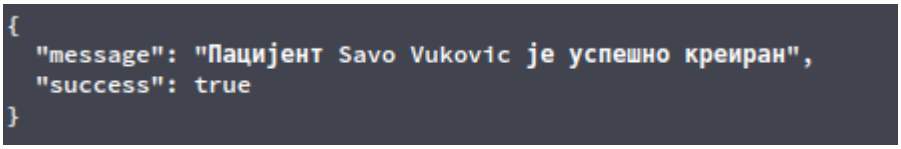
6.6.5 Додавање нових пацијената

Уколико пацијент први пут долази у одређену здравствену установу, а да притом није регистрован од стране неке друге установе у нашем систему, потребно је додати датог пацијента у нашу апликацију. За додавање нових пацијената одговоран је један од здравствених радника у тој установи и ту је веома важно да се тачне информације добију од пацијента ради неке будуће претраге. Одговарајућим *HTTP POST* позивом на крајњу тачку контролера, са одређеном формом тела захтева, пацијент се додаје у наш систем. Пример захтева дат је на следећој слици:



Слика 55. Пример захтева за додавање пацијента

Уколико су све информације везане за тело позива одговарајуће, апликација ће вратити следећу поруку:

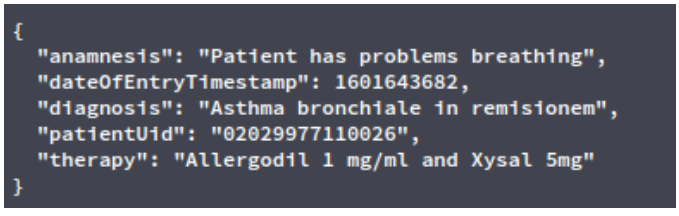


Слика 56. Порука за успешно додавање пацијента

Веома је важно да приликом додавања пацијента параметри као што су ЈМБГ, ЛБО и крвна група буду валидни како систем не би вратио поруку о неуспешном додавању пацијента.

6.6.6 Манипулација са картоном пацијента

Уколико пацијент дође у нашу здравствену установу, ради прегледа због одређених тегоба, потребно је прегледати датог пацијента, направити одређену дијагнозу и на основу те дијагнозе преписати одговарајућу терапију. Како би се сачували подаци о доласку самог пацијента, не користи се традиционални начин писања у картон, већ се слањем на крајњу тачку */patients/entry/add,* шаље одговарајући захтев чији је пример дат на следећој слици:



Слика 57. Пример тела захтева за додавање у картон

Уколико су сви параметри у телу позива валидни, апликација ће вратити поруку да је у електронски картон додат нови податак о датом пацијенту. У другом случају уколико корисник, у овом случају здравствени радник, жели да види цео картон за одређеног пацијента, потребно је да пошаље *HTTP POST* захтев на следећи УРЛ */patient/entry/view* где се у телу захтева наводи ЈМБГ пацијента, уколико постоје пацијенти чији је изабрани лекар тренутно повезани корисник, систем ће вратити следеће:


```
{
  "count": 1,
  "patientEntries": [
    {
      "expired": true,
      "anamnesis": "Patient has problems breathing",
      "diagnosis": "Asthma bronchiale in remisionem",
      "therapy": "Allergodil 1 mg/ml and Xysal 5mg"
    }
  ],
  "message": "Успешно су нађени подаци за датог пацијента!",
  "success": true
}
```


Слика 58. Пример повратне информације приликом претраге пацијената

У одговору, приликом слања захтева за преглед целог картона одређеног пацијента, можемо видети параметар *expired*. Овај параметар се динамички одређује тако што се из конфигурационе датотеке (параметар *patient.therapy.treshold*), извлачи вредност подразумеваног броја дата трајања одређене терапије, на основу које се конфигурише дати параметар.

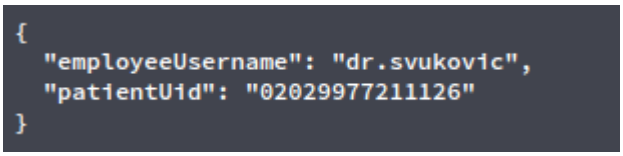
6.6.7 Претрага пацијената по изабраном лекару и додељивање изабраног лекара пацијенту

Уколико здравствени радник жели да види списак свих пацијената којима је он изабрани лекар, потребно је слањем одговарајућем *HTTP POST* захтева на УРЛ */user/patients/view,* у тело захтева пошаље параметар свој корисничког налога. Ако кориснички налог постоји и ако здравствени радник има пацијенте који су га изабрали као свог лекара, следећи одговор ће се вратити од апликације:


```
{
  "count": 1,
  "patients": [
    {
      "bloodType": "AB+",
      "dateOfBirth": "1970-01-19T12:54:03.682+00:00",
      "firstName": "Savo",
      "lastName": "Vukovic",
      "phoneNumber": "+318600000",
      "securityNumber": "2123314541",
      "uid": "020299629911"
    }
  ],
  "success": true
}
```

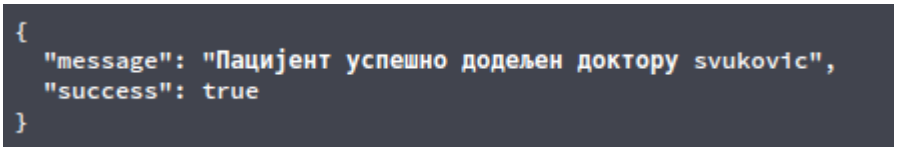

Слика 59. Пример одговора приликом успешне претраге пацијента по изабраном лекару

У другом случају уколико корисник жели да одређеног пацијента додели здравственом раднику потребно је послати следећи захтев, са телом позива у датој форми:



Слика 60. Пример тела позива за додељивање изавраног лекара

Уколико корисничко име постоји у систему и ЈМБГ пацијента јесте јединствено, апликација ће вратити повратну поруку о успешном додељивању корисника одређеном пацијенту.



Слика 61. Пример успешног додељивања изабраног лекара

На овај начин сви подаци у систему су повезани, пацијенти припадају одређеном здравственом раднику и тиме припадају здравственој институцији којој припада одређени изабрани лекар, са опцијом да промене изабраног лекара а самим ти и здравствену установу без беспотребног пребацивања података и додатне администрације. Сви подаци о доласку пацијента на преглед су сачувани у нашој бази података тако да иако пацијент промени здравствену институцију, нови изабрани лекар има могућност да претражује претходне прегледе пацијента. Здравственом раднику је омогућено да види све преписане терапије, прегледе, дијагнозе, анамнезе као и основне информације о самом пацијенту. Омогућено је да се види крвна група као и алергије уско повезане са самим пацијентом, тако да је смањена могућност грешке приликом преписивања терапије и самим тим побољшана ефикасност самог радника. Здравствена институција има комплетан увид у рад својих радника на основу кога може да предузме одређене кораке у смислу администрације и начину комуникације са самим радницима. Пацијентима је олакшан одабир места и методе лечења одабиром разлилитих изабраних лекара и могућству лаког преласка из одређене здравствене институције у другу.

7. Закључак

Централни систем за вођење евиденције о пацијентима, помогао би нашој држави да има комплетан увид у функционисање здравственог система. Подаци, добијени од стране саме апликације, могу бити корисни за моделирање епидемија, зараза као и општед здравственог стања у нашој земљи. Систем би омогућио лакше вођење администрације од стране здравствених радника, олакшао би се посао запослених и побољшала ефективност рада. Архитектура самог информационог система нуди висок ниво бележења акција, што представља добар темељ за додавање подршке слања таквих информација у специјализован систем облака за даљу обраду. Потпуни увид у инфраструктуру здравствених организација омогућио би увођење система за заказивање прегледа, систем за боље вођење радника као и увид у то колико на одређеном временском периоду одређена здравствена институција има пацијената. Модел апликације би се затим могао применити и на образовање у школама као и на другим институцијама у нашој држави, што би омогућило боље функционисање, када су у питању административни послови, у целој земљи. С обзиром да је дат систем реализован као веб сервис, нуди се могућност имплементације посебног корисничког интерфејса за целокупно здравство у целој држави као и имплементирање појединачних корисничких интерфејса за одређене здравствене институције. Сигурност система, валидација као и могућност скалирања података, омогућава нашој држави да се потпуно ослони на дати систем ради одржавања и складиштења осетљивих података. Подршка за више језика као и лако подизање информационог система у новом окружењу, нуди могућност уштеде времена и уштеду средстава када су у питању људи одговорни за одржавање здравственог система. При поређењу са традиционалним начином вођења евиденције, информациони систем поред тога што достиже далеко веће перформансе када је у питању складиштење података, смањује могућност за грешку, дуплирање медицинских картона, нуди могућност за претрагу података и самим тим смањује време за извршавање датих послова.

8. Литература

[1] Бранивоје Тимотић, Миомир Јањић: Примарна здравствена заштита, ELIT MEDICA, Београд 2004

[2] Vlad Mihalcea: High Performance Java Persistance, Vlad Mihalcea 2006

[3] Ranga Rao Karanam: Learn Spring 5, Packt Publishing, Mumbai 2017

[4] Sujoy Archarya: Test-Driven Development with Mockito, Packt Publishing, Mumbai 2013

[5] Cristian Bauer, Gaving King, Gary Gregory: Java persistence with Hibernate Second Edition, Manning Publications Co. 2016

[6] Joshua Bloch: Effective Java Third Edition, Pearson Education Inc. 2018

[7] Joshua Bloch: Effective Java Second Edition, Sun Microsystems Inc. 2008

[8] Tomcy John: Spring Security 5 for Reactive Applicationns, Packt Publishing, Mumbai 2018

[9] Vlad Mihalcea: Transactions and Concurrency Control, Vlad Mihalcea 2020

[10] The Spring Framework Documentation Refference. [Приступљено - 02.09.2020.] https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/