Насловна

Неки папир официјални

Садржај

[1. Увод 1](#_Toc141458320)

[2. Стање у области 2](#_Toc141458321)

[2.1 BetterHelp 2](#_Toc141458322)

[2.2 Апликација Центар срце 2](#_Toc141458323)

[2.3 Wysa 4](#_Toc141458324)

[2.4 Пресек стања 5](#_Toc141458325)

[3. Коришћене технике и технологије 6](#_Toc141458326)

[3.1 Angular 6](#_Toc141458327)

[3.2 REST 7](#_Toc141458328)

[3.3 Spring 7](#_Toc141458329)

[3.4 Знање 9](#_Toc141458330)

[3.4.1 Системи базирани на знању 9](#_Toc141458331)

[3.4.2 Експертски системи базирани на правилима 10](#_Toc141458332)

[3.4.3 Разрешавање конфликата 11](#_Toc141458333)

[3.4.4 RETE алгоритам 11](#_Toc141458334)

[3.4.5 Уланчавање 11](#_Toc141458335)

[3.5 Drools 12](#_Toc141458336)

[3.5.1 Drools правила 12](#_Toc141458337)

[3.5.2 Drools упит 13](#_Toc141458338)

[3.5.3 Rule template 13](#_Toc141458339)

[4. Спецификација 14](#_Toc141458340)

[4.1 Дијаграм случајева коришћења 14](#_Toc141458341)

[4.2 Дијаграм класа 15](#_Toc141458342)

[4.3 Дијаграм секвенци 17](#_Toc141458343)

[5. Имплементација система 18](#_Toc141458344)

[5.1 Клијентска апликација 18](#_Toc141458345)

[5.2 Серверска апликација 19](#_Toc141458346)

[5.3 Drools апликација 22](#_Toc141458347)

[6. Приказ имплементираног система 23](#_Toc141458348)

[7. Закључак 24](#_Toc141458349)

[Литература 25](#_Toc141458350)

[Биографија 26](#_Toc141458351)

# Увод

Модеран начин живота намеће све више обавеза и захтева. Стрес, нервоза, временски рокови и јурњава су описи већина данашњих занимања. Овакво окружење и брз начин живота утичу на физичко и ментално здравље. Буди се свест о важности менталног здравља и популаризације истог. На интернету се може пронаћи све више едукација које се баве психологијом, међуљудским односима, комуникацијом. Међутим, и поред експанзије садржаја многима је одлазак на психотерапију и даље табу тема, и даље је присутан недостатак подршке околине за такву одлуку. Такође, много људи се суочава са анксиозности, социјалном анксиозности и паничним нападима. Постојање експертског система који помаже при дијагностици наведених менталних стања би пружио корисницима први сусрет са психотерапијом и притом омогућила да попуњавањем упитника добију резултат о свом менталном стању и на тај начин започну рад на себи.

Тема рада је

Циљ рада је

Теоретски концепти проблема који је решаван

Појмови који се први пут срећу

Структура по поглављима

# Стање у области

Дефиниција здравља према Светској здравственој организацији (*World Health Organization - WHO*) гласи *Здравље представља комплексно физичко, ментално и социјално благостање, а не само одсуство болести и онеспособљености*. Наведена дефиниција јасно истиче важност менталног здравља као неизоставни део општег здравља.

На основу системске анализе опште популације у већини европских земаља показано је да ће 27% одрасле популације старости 18-65 година некада током свог живота имати ментални поремећај (то укључује поремећаје у вези са злоупотребом супстанци, психозе, депресије, анксиозност и поремећаје исхране); за жене ризик од оболевања износи око 33,2%, а за мушкарце 21,7%. Код 32% опште популације се очекује да ће имати један ментални поремећај, код 18% се могу очекивати два, а код 14% и три ментална поремећаја током живота [1].

Начин на који би човек могао да умањи могућност оболевања од менталних поремећаја је превенција и ментална хигијена. Данас постоје апликације које могу пружити подршку. У наставку су анализиране апликација Центар срце, *BetterHelp* и *Wysa*.

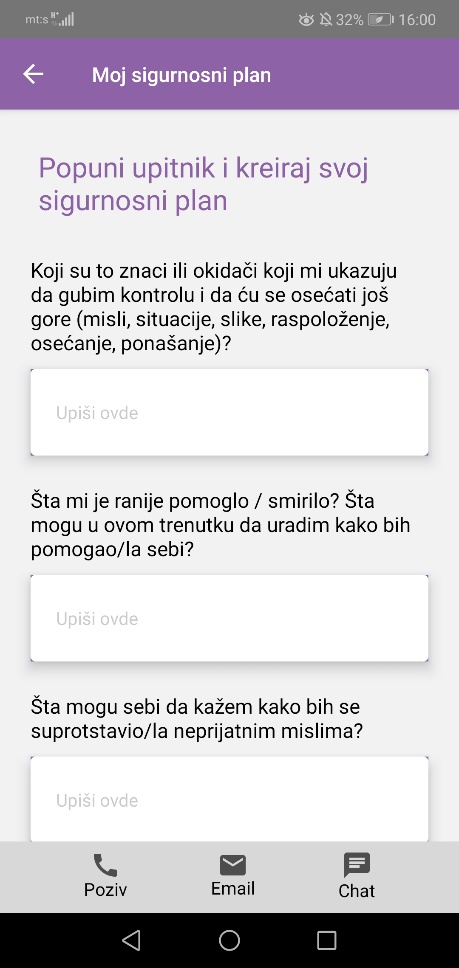
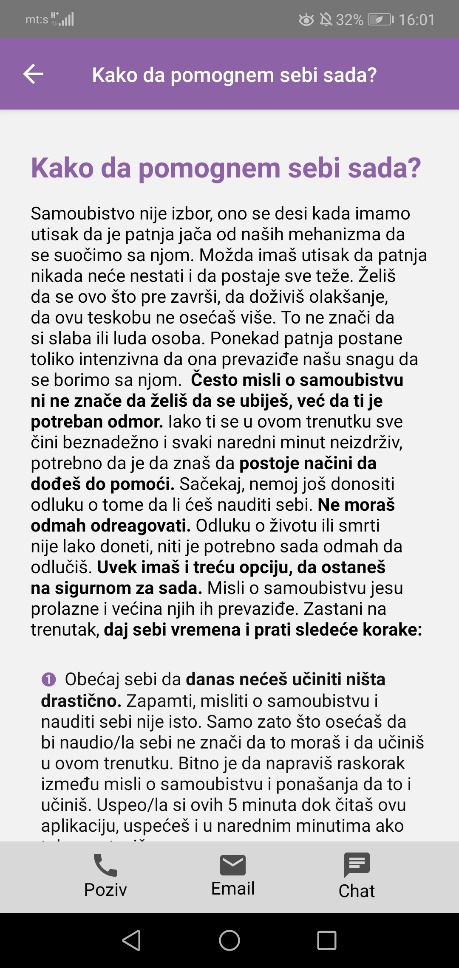
## BetterHelp

BetterHelp је данас најпопуларнија апликација која пружа повезивање корисника са психолозима широм света. Приликом инсталације апликације се попуњава упитник који покрива питања о жељеној терапији (индивидуална, партнерска или за децу), личним подацима, љубавном статусу, религији, религијским навикама, духовности, разлозима за терапију, очекивањима од терапеута, претходне терапије, коришћењу лекова, тренутном стању по питању неких болова или тренутној анксиозности, депресији, финансијском статусу, навикама за спавање, преференције везане за терапеута. Комуникација са терапеутом се одвија путем аудио, видео и текстуалних порука унутар чета док се уживо сесија у трајању од 30-45 минута може одвијате путем телефонског и видео позива или чета. Коришћење апликације се плаћа 65 долара недељно односно 260 долара месечно. У ову цену су укључене 4 сесије и подршка унутар чета. Ово је значајно повољније за америчке стандарде где је цена једне сесије 150-180 долара. У случају да корисник није у могућности да плати сесију има могућност да се обрати за финансијску помоћ.

## Апликација Центар срце

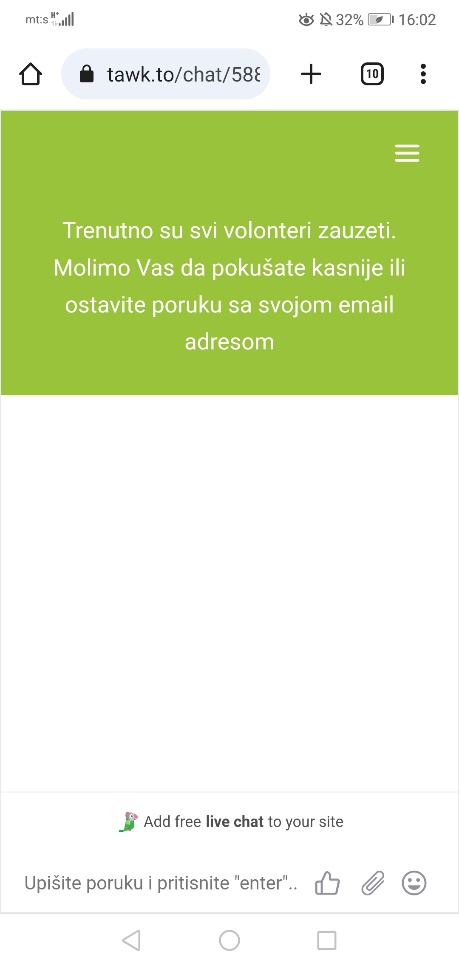
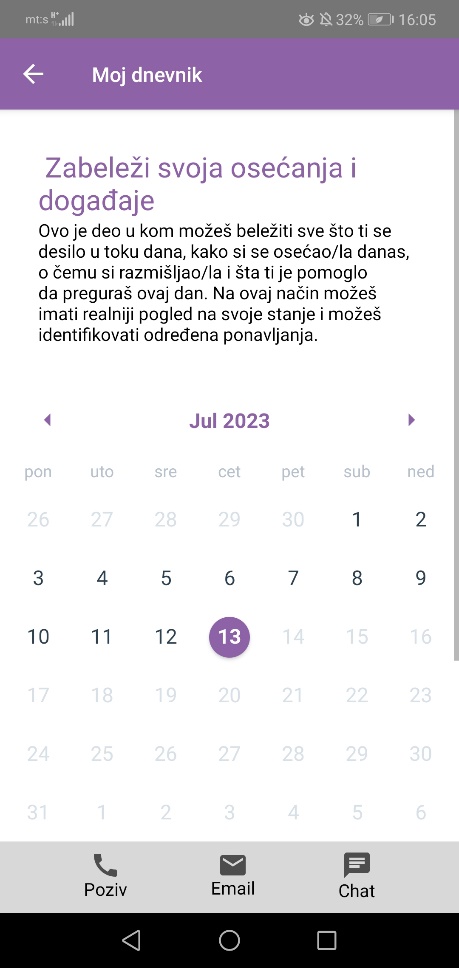
Апликација Центар срце је мобилна апликација за подршку људима у тренуцима када им је јако тешко и када се чини да нема излаза. Подршка је упућена свима који имају суицидне мисли и намере.

Након инсталације, апликација се може одмах користити без остављања података. Регистрација није потребна тако да је анонимност загарантована. На почетној страници се налази мени којим је омогућена навигација кроз апликацију. У апликацији се налазе текстови и опште информације о суициду као и текстуални водич „Како да помогнем себи сада?“ (Слика 2) који кроз текстуални опис корака пружа прву помоћ у безизлазним тренуцима. Одељак „Мој сигурносни план“ је упитник који наводи корисника да одговорима на питања направи свој план са акцијама које може предузети (Слика 1). Доступан је и дневник за евидентирање осећања и стања (Слика 3). Корисник има могућност да ступи у контакт са волонтерима преко позива, мејла или чета (Слика 4).



Слика Сигурносни план

Слика Водич за помоћ



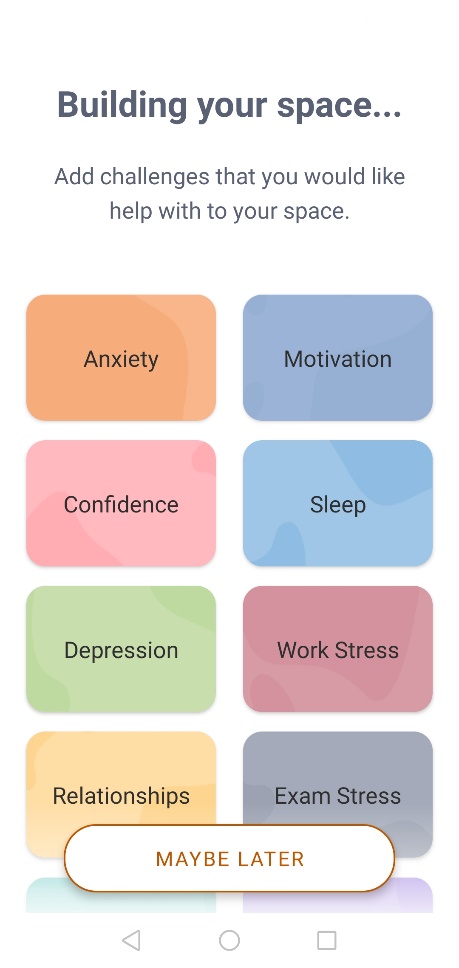
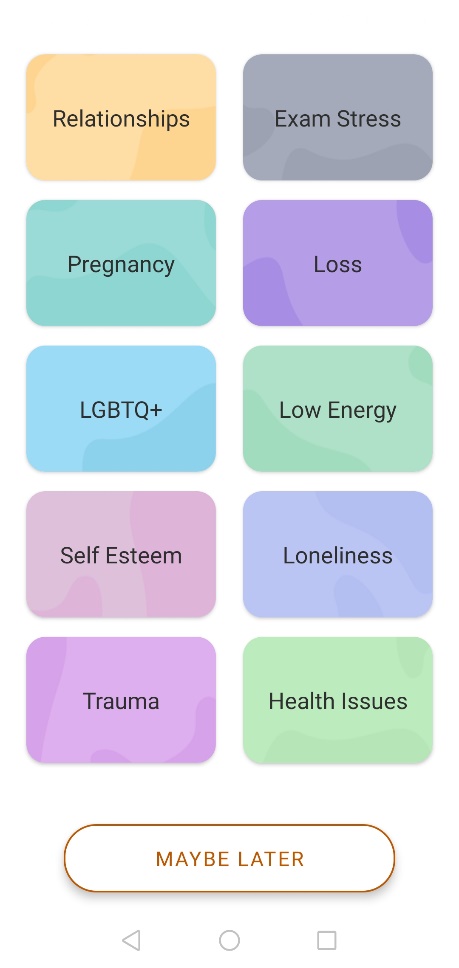
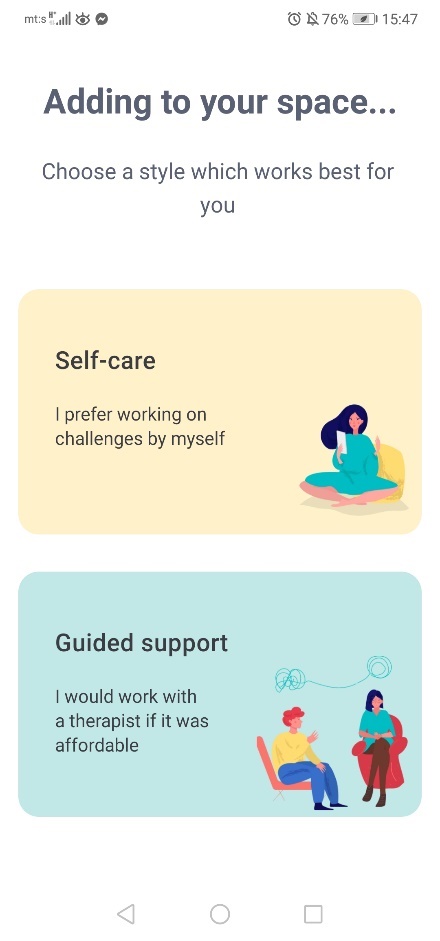
Слика Дневник

Слика Контакт са волонтерима

## Wysa

*Wysa* апликација је заснована на вештачкој интелигенцији која пружа подршку и смернице корисницима. Корисник се дописује са вештачком интелигенцијом која користи когнитивне бихејвиоралне технике засноване на доказима, технике дијалектичког понашања, медитацију, технике дисања, јогу [2]. Клинички је доказано да *Wysa* вештачка интелигенција успоставља терапијски однос еквивалентан људском терапеуту у току прве недеље [2]. Апликација пружа низ алата и функција, укључујући праћење расположења, технике медитације и опуштања и постављање циљева. За свакодневни стрес, *Wysa* нуди специфичне програме самопомоћи кроз више од 150 вежби заснованих на доказима, укључујући материјале за анксиозност, стрес, депресију, проблеме са спавањем (Слика 5, Слика 6).

Након инсталације апликације није потребна регистрација нити остављање личних података, довољно је само унети надимак. Након уноса неког имена корисник бира области на којима жели да ради (Слика 5, Слика 6), а затим бира и да ли жели да у његов рад буде укључен терапеут или ће самостално (Слика 7).



Слика Одабир начина рада на себи

Слика Области за рад

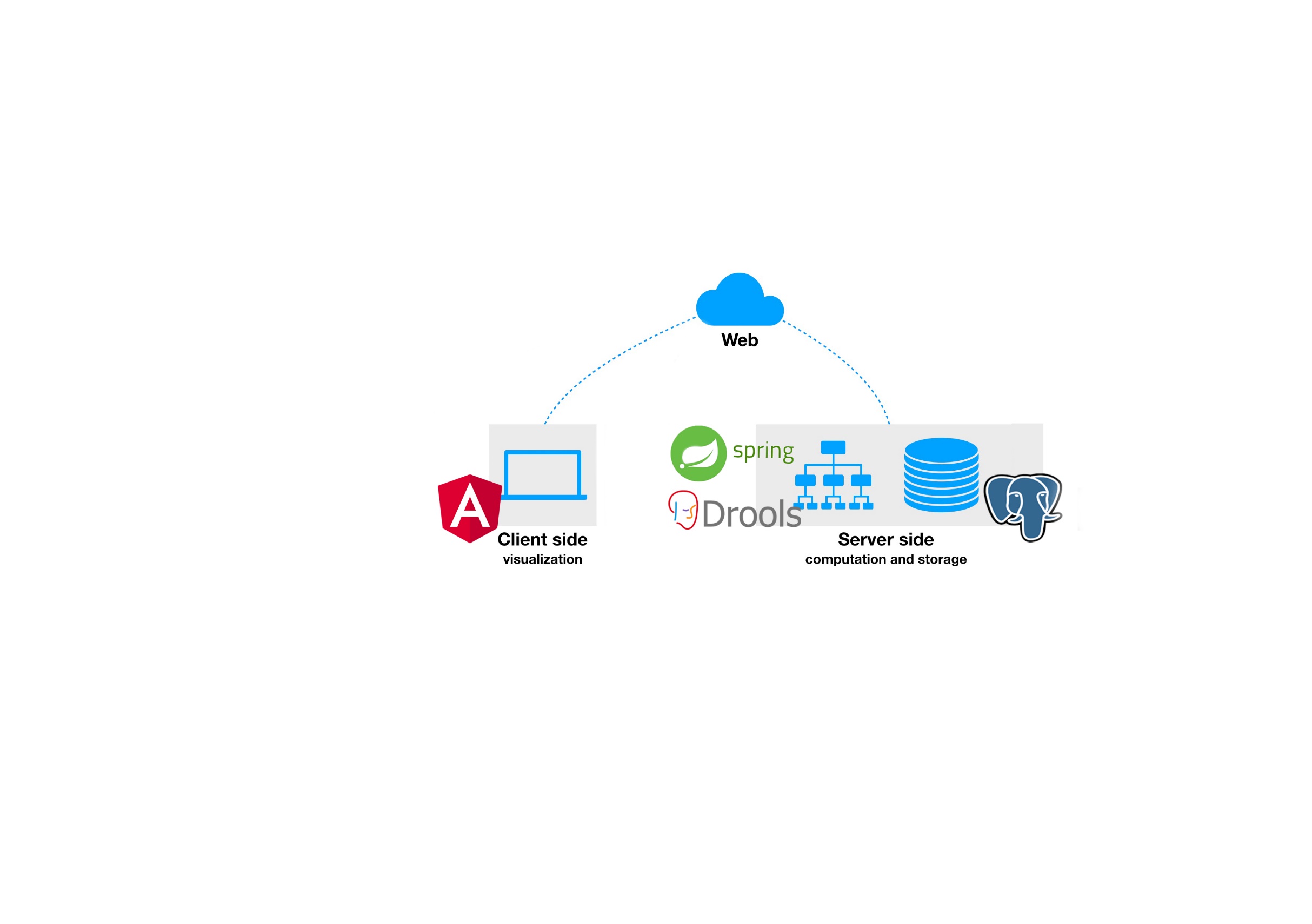
Слика Области за рад

## Пресек стања

Добре стране описаних апликацију су што повезују кориснике са психологом уз загарантовану анонимност или омогућавају коришћење у потпуности без остављања приватних података. Такође, апликације обезбеђују доста материјала у текстуалној форми и видео снимака које могу бити први корак у раду на себи и/или самопомоћи. Корисник може да оствари интерактивност преко чета или разговора, добија обавештења и упутства о темама од интереса.

Кориснику је дата могућност да самостално бира области којима жели да се бави попут анксиозности, мотивације, стреса, депресије, траума, веза што доводи корисника у ситуацију да сам себи одреди поље рада. Како корисник није стручан, може доћи у ситуацију да избегава област која би управо била његово поље рада. Апликација која је описана у раду се бави дијагностиковањем менталних стања тако да апликација упућује корисника на области којима би могао да се бави и да у свој рад укључи психолога..

# Коришћене технике и технологије

Систем чине клијентска апликација, серверска апликација, систем базиран на правилима и база података. Клијентска апликација је развијена у *Angular*-у [3], а серверски део је развијен у *Spring*-у [4] уз употребу *REST* [5] сервиса. Подаци се складиште у *PostgresSQL* бази података. Систем базиран на правилима је имплементиран у *Drools*-у [7]. На Слика 8 је приказана архитектура система.

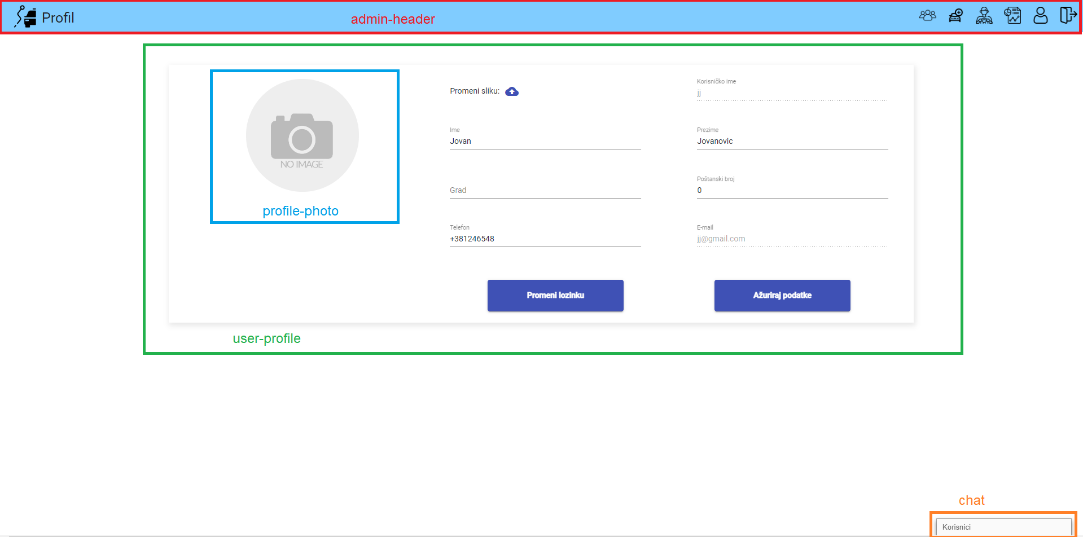
Слика Архитектура апликације

## Angular

*Angular* је радни оквир за дизајн апликација и развојна платформа за креирање апликација на једној страни (*Single-Page)* заснован на *HTML*-у и *TypeScript*-у [3].

Основни градивни елеменат апликације су компоненте. Компонента обједињује приказ у *HTML*-у и апликативну логику у *TypeScript*-у. Компоненте су повезане тако да чине стабло. Угњеждавањем компоненти постиже се читљивост кода и избегавање понављања кода (Слика 9).

Скуп компоненти који гради логичку целину представља модул. Свака *Angular* апликација мора се састојати од барем једног модула који мора садржати коренску компоненту док се остале компоненте угњеждавају унутар коренске. У *TypeScript* класи која представља модул наводе се све компоненте и сервиси који припадају том модулу, компоненте које се извозе у друге модуле и модули који се увозе у дати модул.

За комуникацију са серверском апликацијом користе се сервиси, односно *TypeScript* класе које су анотиране са *@Injectable* анотацијом. Инјектор је задужен за инстанцирање сервиса, управљање његовим животним циклусом и прибављање по потреби. У сервисима се налази логика коју користи више компоненти. Сервиси се могу користити и за комуникацију између две удаљене компоненте.

Слика Приказ уоквирених угњеждених компоненти

## REST

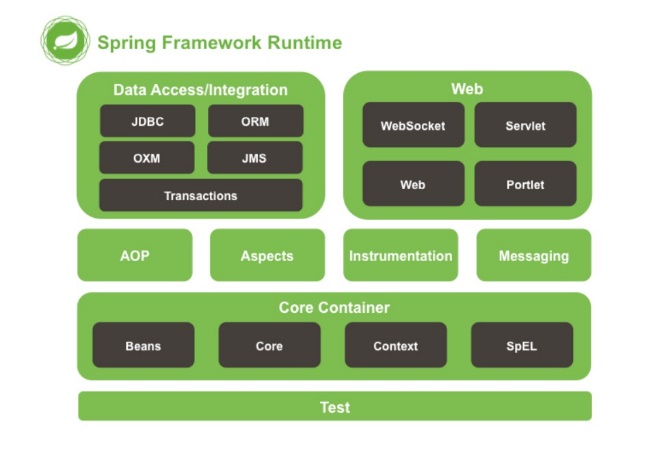
*REST* (*Representational state transfer*) је архитектонски стил који се користи на веб сервисима. *Rest API*, интерфејс за програмирање апликације, је скуп правила која дефинишу како апликације или уређаји могу да се повежу и међусобно комуницирају и размењују податке. Др Роја Филдинг је први пут дефинисао *REST* *API* у својој докторској дисертацији 2000. године [6]. Флексибилност је основна предност *REST* архитектуре. *REST* *API* најчешће комуницира путем *HTTP* захтева и податке може размењивати у различитим форматима од којих је *JSON* најпопуларнији. *REST* дефинише шест ограничења чијом применом се постижу скалабилност, могућност измене, једноставност, преносивост:

* Клијент-сервер архитектура
* Непостојање стања
* Кеш меморија
* Униформни интерфејс
* Слојевити систем
* Код на захтев (опционо)

## Spring

*Spring* је радни оквир за развој апликација у *Java* програмском језику. *Spring* омогућује креирање *Java EE* апликација на једноставнији начин јер обезбеђује инфраструктуру тако да програмер може да се фокусира на доменске проблеме. Прва верзија *Spring* радног оквира публикована је 2003. године.

Предност *Spring*-а је што користи обичне класе - *Plain old Java objects*(*POJO*) које су анотиране анотацијама којима се дефинише како *Spring* управља објектима тих класа. *Spring* обезбеђује повезивање објеката, брине о животном циклусу објеката и добавља тражене објекте на захтев пратећи *Dependency Injection (DI)* механизам. Такође, у *Spring-у* је подржано декларативно и аспектно-оријентисано програмирање.

*Spring* се састоји од функција организованих у 20 модула. Модули су груписани у *Core Container*, *Web*, *Data Access/Integration*, *Aspect Oriented Programming (AOP)*, *Aspects*, *Instrumentation* и *Test* (Слика 10).

Слика Spring модули [4]

*Core Container* је основни модул који обезбеђује фундаменталне делове оквира попут *Inversion of Control* и *Dependency Injection* механизама. *Beans* модул садржи фабрику објеката која омогућава одвајање конфигурације и спецификације зависности од стварне програмске логике уместо употребе *Singletone* шаблона. Модул *Context* се ослања на *Core* и *Beans* модуле. Модул *Context* наслеђује функције *Beans* модула и додаје подршку за пропагацију догађаја, учитавање конфигурације апликације и свих објеката. Централна тачка *Context* модула је *ApplicationContext* интерфејс. *Expression language* модул пружа језик за писање упита и манипулацију графом објеката током времена извршавања (*run-time*).

*Data Access/Integration* модул се састоји из подмодула *JDBC, ORM, OXM, JMS* и *Transactions*. *JDBC* модул апстрахује везу са базом података и уклања потребу за заморним *JDBC* програмирањем и парсирањем грешака специфичних за добављање података из базе података. *ORM* модул обезбеђује објектно-релационо мапирање чија конкретна имплементација може бити Open*JPA*, *Hibernate* [8].Сви оквири за објектно-релационо-мапирање могу се користити у комбинацији са функцијама које нуди *Spring* попут декларативног управљања трансакцијама.

*Web* модул пружа подршку за различите аспекте развоја веб апликација попут инјекције објеката у сервлете, постављање фајлова на сервер. Модул *Web-Servlet* садржи *Spring*-ову имплементацију *MVC* обрасца за веб апликације. Коришћена је подршка за *REST* јер је клијентски део имплементиран помоћу *Angular*-a.

*Test* модул омогућује тестирање засновано на *Junit* и *TestNG* библиотекама. Омогућује спровођење јединичног и интеграционог тестирања. За интеграционо тестирање омогућује учитавање објеката у *Spring application context.* Такође, садржи и имплементацију *mock* објеката који се користе за јединичне тестове и изолацију дела апликације који се тестира.

## Знање

Знање је резултат анализе информација које се доводе у контекст. Знање представља информацију коју људи користе за решавање проблема. По Трипаритној дефиницији знање је образложено истинито уверење ако је исказ тачан, истинит и оправдан. Компоненте знања су:

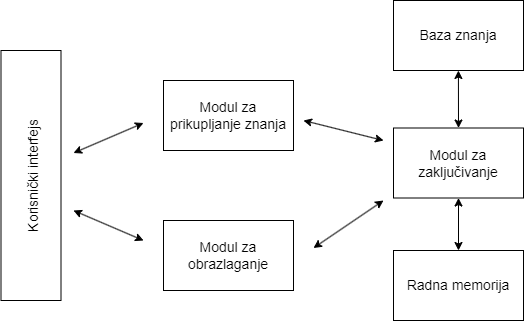
* Чињеница – ствар која се зна или је доказано да је она истинита
* Концепт – апстракција или уопштење искуства
* Процедура – низ акција које се спроводе по одређеном редоследу и начину
* Модел – репрезентација система или сложеног ентитета
* Хеуристика – искуствено базирана техника за решавање проблема
* Пример – део нечега што се узима да се покаже карактер целине

### Системи базирани на знању

Системи базирани на знању се ослањају на знање које је формално исказано и складиштено у систему тако да систем може да доноси одлуке на основу тог знања. Експертски системи представљају специјализоване системе базиране на знању јер интегришу доменско знање експерта у базу знања која се користи за закључивање.

Експертски системи су применљиви на уске области знања односно домене знања. Разлози за прављење експертских система јесу чување знања експерта, основа за обуку почетницима, обезбеђивање уједначености одлука и омогућавање доступности знања експерта.

Системи базирани на знању се састоје из базе знања, модула за закључивање, радне меморије, модула за резоновање, модула за прикупљање знања и корисничког интерфејса (Слика 11).



Слика Структура система базираног на знању

### Експертски системи базирани на правилима

Експертски системи базирани на правилима су системи чија база знања садржи доменско знање стручњака представљено у форми *if-then* правила. Тврдње у *if-then* форми називају се продукциона правила. У експертским системима базираним на правилима знање стручњака је формулисано продукционим правилима која се примењују на чињенице из радне меморије у циљу доношења закључака.

Претпоставке које је потребно да буду задовољене се пишу у *if* делу правила, односно у левој страни правила. Потребно је да су задовољене све претпоставке како би се извршила акција или донео закључак. У случају да само једна претпоставка није задовољена, акција се неће извршити. Над претпоставкама се могу писати логички оператори попут: *and, or, not*. Оператор *not* се тумачи као да не постоји чињеница у меморији који задовољава услов. Заграде се користите за груписање. Претпоставка се састоје из објекта, вредности и оператора. У примеру *The temperature is 36 degrees -* > *the tempеrаture* је објекат, *36 degress* је вредност и *is* је оператор. Акције/закључци се пишу у десној страни правила, након *then* кључне речи. Десна страна правила представља закључак до ког је дошао експерт сагледавањем свих чињеница. Акција/закључак представља предвиђени корак за решавање проблема.

Пре почетка резоновања продукциона правила попуњавају базу знања, а чињенице се иницијализују у радој меморији. Након покретања модула за закључивање упарују се чињенице из радне меморије са патернима који су дефинисани у претпоставкама у циљу проналаска правила која су задовољена. Да би правило било задовољено лева страна правила мора да се слаже са чињеницама из радне меморије. Модул за закључивање одређује правила чији су предуслови задовољени. Задовољена правила се смештају у агенду која се налази унутар модула за закључивање. У једном тренутку само једно правило може да се активира. У случају да се у агенди нађе више правила примењује се неки од поступака за разрешавање конфликта. Након извршавања десне стране правила, правило се уклања из агенде. Извршавање правила може довести до промена у радној меморији додавањем нових чињеница или изменом и уклањањем постојећих тако да се врши поново упаривање и додавање нових задовољених правила у агенду и избацивање оних која више не важе. Модул за закључивање завршава са радом у тренутку када у агенди више нема задовољених правила или експлицитном *stop* командом.

### Разрешавање конфликата

Сет правила која су задовољена се назива конфликт сет (*conflict-set*). Разрешавање конфликта (*conflict resolution*) представља метод одабира правила које ће се извршити из конфликт сета.

Стратегије за одабир правила:

1. Одабир правила на основу приоритета – најчешће редослед правила у бази знања одређује и приоритет правила. Приоритет се може и експлицитно дефинисати.
2. Одабир правила на основу позиције правила – бира прво правило које дефинисано у тексту
3. Одабир најспецифичнијег правила – правило је специфичније ако његово задовољење захтева испуњење више предуслова
4. Одабир на основу мета-правила – односно на основу правила о правилима. Већи приоритет има правило који је писао експерт него правило које је писао почетник
5. Одабир на основу најсвежијих података – одабрати правило на основу чињеница које су најскорије додате

### RETE алгоритам

*RETE* алгоритам је развијен крајем седамдесетих година двадесетог века захваљујући др *Charles L. Forgy. RETE* алгоритам је алгоритам који проверава да ли чињенице задовољавају правила у систему правила (*rule-engine).* Правила су скуп претпоставки и акција. Чињеница је вектор својстава.

*RETE* на латинском значи мрежа. *RETE* мрежу дефинишу усмерени ациклични графови који представљају сетове правила. Сваки чвор у мрежи одговара услову или делу услова из неке претпоставке у левој страни правила. Путања од коренског чвора до крајњег чвора представља леву стару правила. Сваки чвор има своју меморију у којој се налазе све чињенице које задовољавају дати услов. За сваку чињеницу креира се елемент радне меморије (*Working Element Memory - WME*) који представља ту чињеницу. Свака чињеница улази у *RETE* мрежу кроз коренски чвор мреже. Коренски чвор прослеђује чињеницу чворовима наследницима. На тај начин чињеница пролази кроз неку од грана стабла док не стигне до неког од крајњих чворова.

### Уланчавање

Два основна механизма за закључивање су уланчавање унапред и уланчавање уназад.

Уланчавање унапред (*forward chaining*) је резоновање које је вођено подацима (*data-driven*) од чињеница ка закључку. На основу чињеница које задовољавају претпоставке се активирају правила чијим извршењем се потенцијално мења радна меморија или доноси закључак. Промена радне меморије доводи до поновне евалуације правила. Стање радне меморије је познато, али коначни закључак, ако се до њега дође, није познат.

Уланчавање уназад (*backward chaining*) је резоновање вођено упитом (*query-driven*) од закључка ка чињеницама. Знамо шта желимо да докажемо, али не знамо да ли ћемо то доказати са чињеницама које имамо у радној меморији. Полази се од хипотезе односно упита и прво се траже правила чија је десна страна хипотеза. У случају да таква правила не постоје, хипотеза је оборена. У случају да таква правила постоје, проверава се да ли чињенице задовољавају леве стране. Ако чињенице задовољавају леве стране, хипотеза је доказана, а ако не задовољавају онда лева страна правила постаје хипотеза и циклус се понавља рекурзивно.

## Drools

*Drools* је део *KIE* (*Knowledge Is Everything*) пројекта који нуди алате за решења за аутоматизацију пословања и управљање. *Drools* представља систем за управљање пословним правилима користећи механизам за закључивање заснованим на уланчавању унапред и уназад. Подржава и комплексно процесирање догађаја (*Complex Event Processing* - *CEP*). *Drools* представља основни градивни елемент за развој експертских система. Имплементира проширен *RЕТЕ* алгоритам.

### Drools правила

Правила се пишу у .drl фајловима. Структура фајла обухвата назив пакета, импорт секцију, декларацију глобалних варијабли, типова и догађаја, правила и упите. У (Листинг 1) је приказана структура фајла и пример правила. Називи правила морају бити јединствени. У *when* секцији се пишу услови помоћу *DRL* језика. *When* секција представља леву страну правила (*Left Hand Side - LHS)*. Потребно је да чињенице испуњавају услове из *when* дела како би се извршило правило. У *then* секцији, односно у десној страни правила (*Right Hand Side - RHS)* се пишу последице активирања правила.

За свако правило се могу дефинисати и атрибути који омогућавају филтрирање правила. Евалуација атрибута се ради након што је задовољен *when* део правила. *Salience* атрибут дефинише приоритет, *no-loop* атрибутом се спречава бесконачна петља где правило само себе активира док *look-on-active* спречава да се правило поново изврши за исту чињеницу. Правила се могу груписати помоћу *agenda-group* атрибута.

Листинг Структура .drl фајла и пример правила

*// naziv paketa*

*// importi*

*// globalne promenljive*

rule "naziv pravila"  
*// atributi: sailence, no-loop, agenda-group*when   
 *// Left Hand Side*then  
 *// Right Hand Side*end

У *LHS* делу правила могу се користити оператори, *Java* изрази попут *and, or, not, collect, contains, memberOf, from, accumulate, exists, forAll*. У *RHS* делу правила се пишу акције које могу оперисати над радном меморијом попут *insert, modify, update* и *delete*. Помоћу наведених кључних речи могуће је додавати нове чињенице у радну меморију, мењати и брисати постојеће.

### Drools упит

Упит (*query*) представља једноставан начин за претрагу радне меморије. Упити се састоје само из *LHS* дела и могу да приме параметре. Погодни су за писање рекурзивних упита односно за уланчавање уназад. Упит се састоје из кључне речи *query* након које се пише назив упита и опциони параметри које прима сам упит. Након тога пише се *LHS* што је приказано на Листинг 2*.*

Листинг Пример упита

query “naziv upita” (Tip parametar)  
 Left Hand Side  
end

### Rule template

*Rule template* представљају начин за генерисање *DRL* правила у времену извршавања користећи *template* фајлове и податке који се могу представити табеларно. Основна предност *rule template*-а је да се може користити један *rule template* фајл за различите сетове података и обрнуто. Текстуални фајлови у којима се пишу темплејти имају .*drt (Drools rule template)* екстензију и кључне речи којима се означавају различите секције темплејта и променљиве.

На Листинг 3 је приказана структура темплејта. Након заглавља темплејта наводе имена променљивих која се налазе и у подацима која ће се интегрисати у генерисана правила. Празан ред означава крај заглавља. Након импорт секције и декларација наводи се кључна реч *template* и јединствено име темплејта. Унутар темплејта дефинише се правило које ће бити генерисано са различитим подацима. Јединственост назива правила постиже се бројачем који је додат на крају назива правила.

template header  
// nazivi promenljivih  
  
package diagnostics; // naziv paketa  
  
// import sekcija, definisanje tipova i funkcija  
  
template "templ-1" // početak templejta

rule "nazi\_pravila\_@{row.rowNumber}"   
  
when  
 // LHS  
then

// RHS  
end  
  
end template // kraj templejta

Листинг Структура темплејта

# Спецификација

Систем за дијагностику менталног здравља пацијената је веб апликација која је замишљена као помоћно средство у раду клијената и психолога. У овом поглављу систем је представљен различитим *UML*(*Unified Modeling Language*) [9] дијаграмима.

## Дијаграм случајева коришћења

У комуникацији наручиоца софтвера и тима који развија софтвер важну улогу има презентација захтева. Технику за прикупљање, анализу и бележење захтева базирану на случајевима коришћења (*use case*) осмислио је и Иван Јакобсон 1994. године [10]. Дијаграми случајева коришћења се користе за описивање скупа радњи (случајева коришћења) које софтвер треба да изврши у сарадњи са једним или више учесника.

Случај коришћења представља функционалност или сервис која се пружа учеснику система. Сваки случај коришћења треба да пружи видљив и вредан резултат учесницима [10].

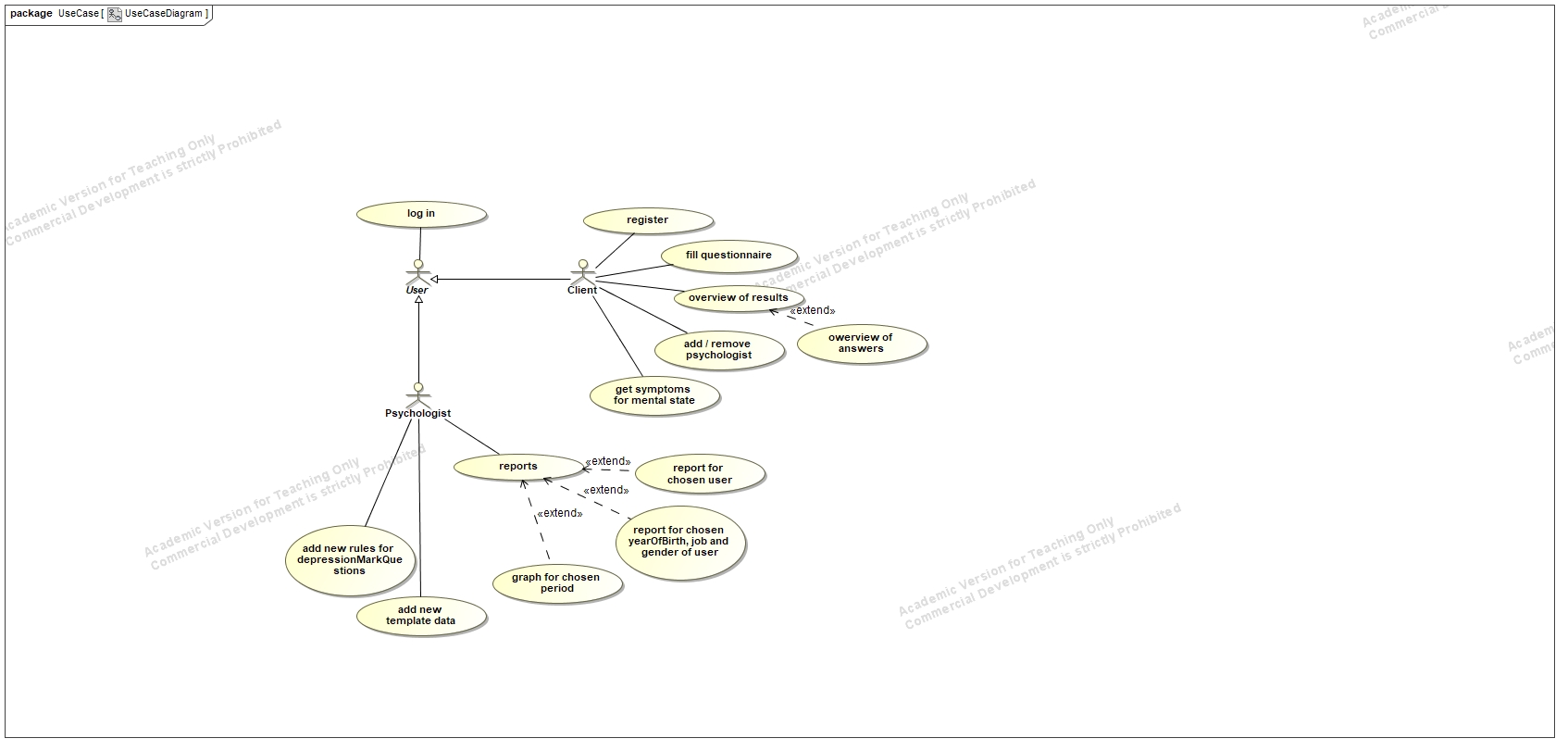
Учесник (actor) представља улогу у систему. Улоге се додељују корисницима система и једном кориснику се може доделити више улога. На Слика 12 приказан је дијаграм случајева коришћења за систем описан у раду .

Апликација подржава два типа корисника Клијента и Психолога.

Клијент је корисник који се може регистровати на систем и након регистрације може да се пријави. Клијент има могућност да попуни упитник за дијагностиковање, додаје и уклање психологе којима дозвољава приступ својим одговорима. Такође, има преглед својих дијагностикованих стања и одговора на питања. Клијент може да за одабрано ментално стање добије листу свих стања која су симптоми одабраном стању.

Психолог има могућност да се пријави и додаје нове упитнике и правила за обраду тих упитника као и увид у извештаје. Психолог може да прегледа резултате за кориснике који су му претходно одобрили приступ, добије извештај на основу године рођење, професије и пола клијената као и графички приказ дијагностикованих стања у временском опсегу.

Обрада одговора на упитник који попуњава корисник и упит за добијање симптома за одабрано стање је реализована коришћењем система базираних на правилима која се ослањају на доменско знање експерта.



Слика Дијаграм случајева коришћења

## Дијаграм класа

Дијаграм класа се користи за скицирање структуре пословног домена за који се имплементира софтвер, комуникацију идеја унутар тима, спецификацију имплементације, генерисање кода и документовање решења зависно у којој се фази развоја налази софтвер [10].

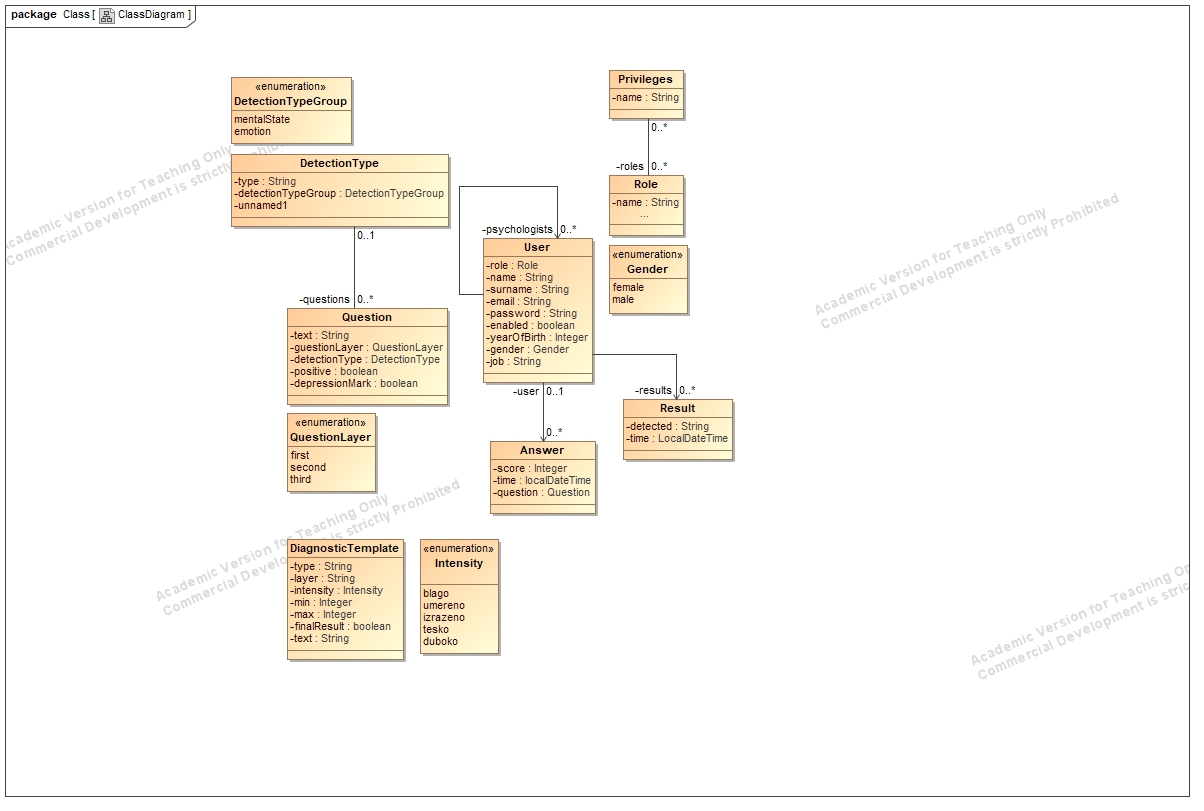
Описани систем се састоји из осам класа и четири енумерације. Набројане су класе:

* User
* Role
* Privileges
* Question
* DetectionType
* Answer
* Result
* DiagnosticTemplate

Класе *User, Role* и *Privileges* служе за пријаву и ауторизацију корисника као и бележење основних података о кориснику. *DetectionType* класа представља ментално стање које систем може дијагностиковати. Класама *Question, Answer* и *Result* описана су питања, одговори које даје корисник и резултат дијагностиковања на основу одговора. *DiagnosticTemplate* класом описан је темплејт за дијагностику.

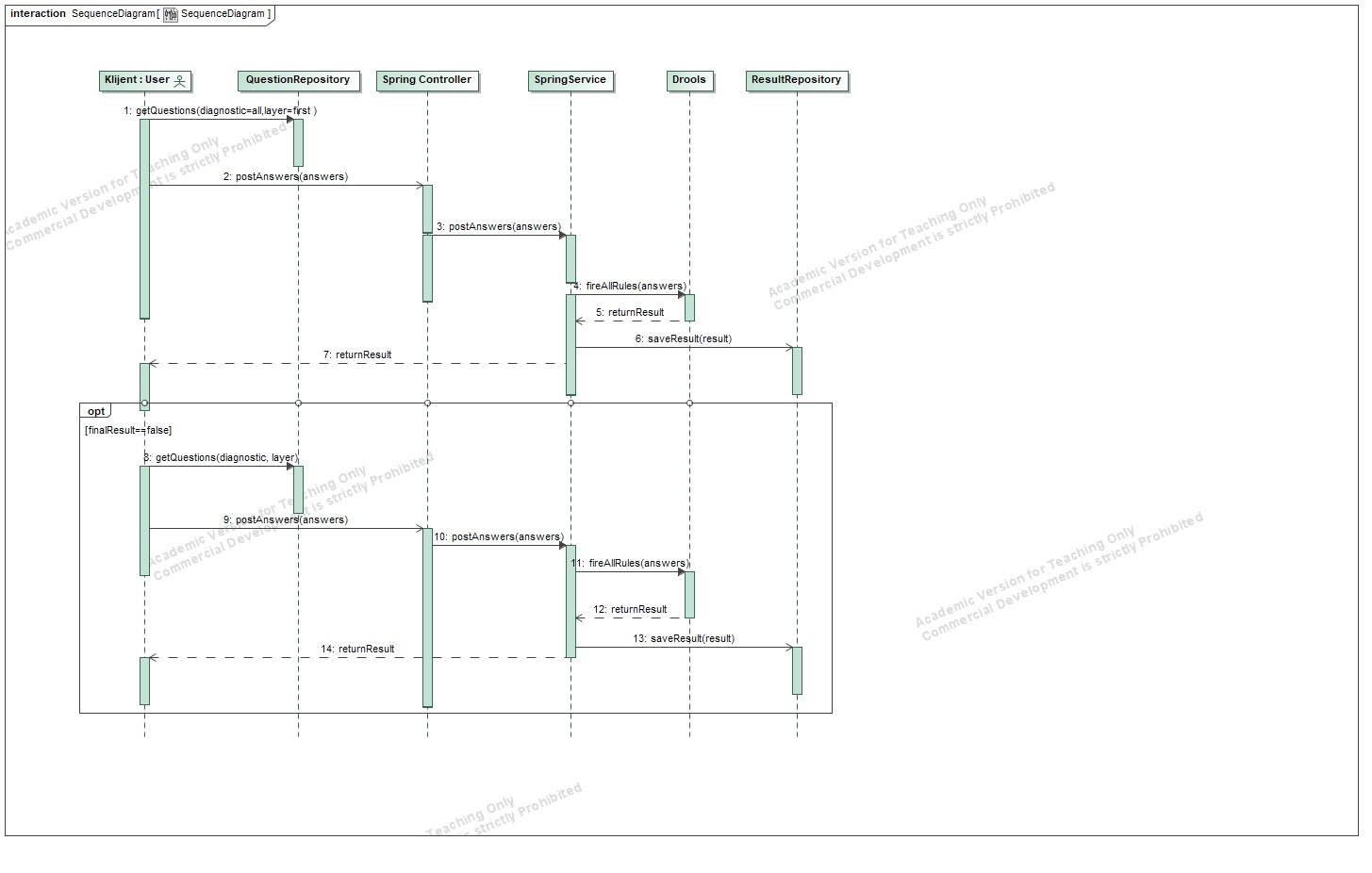
Поред класа, систем чине и четири енумерације:

* DetectionTypeGroup
* QuestionLayer
* Intensity
* Gender

*DetectionTypeGroup* енумерација има вредности *mentalState* и *emotion* и тако одваја стања од емоција с обзиром да је дијагностиковање стања подсистем. *QuestionLayer* има вредности *first*, *second* и *third* којим се описује редослед питања у упитнику, то јест подела питања по нивоима. *Intensity* означава ниво озбиљности дијагностикованог стања. *Gender* представља пол.

Слика Дијагам класа

## Дијаграм секвенци

Дијаграм секвенци приказује комуникацију између учесника у систему и елемената система у времену [10]. Дијаграм секвенци описује интеракцију фокусирајући се на редослед порука које се размењују. На Слика 14 је приказан дијаграм секвенце за клијента приликом попуњавања упитника. Прва акција је добијање првог сета питања. На основу одговора се добија резултат о дијагностиковању стања. У случају да клијент има услове за даља испитивања јер су ментална стања међусобно условљена добија нови сет питања.

Слика Дијаграм секвенци

Немој одустати ! Бог ће ти дати снагу која ти је потребна да издржиш !

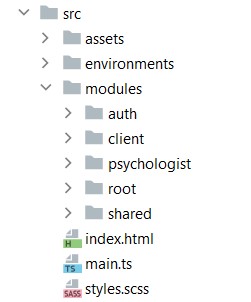
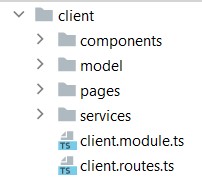
# Имплементација система

У овом поглављу биће описано како је имплементиран систем то јест детаљнији приказ сваког дела апликације. Апликација за дијагностиковање менталних стања се састоји из четири дела: клијентски део писан у *Angular*-у, серверски део, *PostgresSQL* база података и систем базиран на правилима имплементиран помоћу *Drools* алата.

Изворни код апликације се налази на GitHub репозиторијуму **111111** .

## Клијентска апликација

Клијентска апликација је имплементирана коришћењем *Angular* радног оквира. Изворни код апликације има структуру као на Слика 15. У *assets* фолдеру се налази статички садржај односно све слике и иконице које се користе за дизајн. У *Environments* фолдеру је дефинисана константа која представља адресу на којој се налази серверска апликација. Имплементирана је лења евалуација модула. Апликација се састоји из пет модула.



Слика Структура модула

Слика Структура клијентске апликације

Унутар сваког модула се налазе фолдери (Слика 16):

* *components* – унутар *components* фолдера налазе компоненте које су део неке од страница
* *pages* – унутар *pages* фолдера налазе се компоненте на које усмерава рутер
* *models* – у *models* фолдеру су дефиниције свих интерфејса који моделују податке које се размењују између клијентског и серверског дела. На основу интерфејса се аналогно формирају *DTO*(*Data Transfer Object*) класе на серверском делу
* *services* –унутар *services* фолдера се налазе све *TypeScript* класе које комуницирају са серверским делом апликације
* *naziv.module.ts* - *TypeScript* класа унутар које су декларисане све компоненте модула и сви модули које се увозе (импортују) у дати модул
* *naziv.routes.ts* - *TypeScript* класа унутар које дефинисане руте до компоненти из *pages*

*Root* модул је коренски модул апликације у ком се налази коренска компонента (RootComponent). У *root* модулу су смештена заглавља за клијента и психолога као и страница на коју се усмерава корисник уколико покуша да приступи путањи која не постоји. *TitleService* класа обезбеђује одговарајуће наслове зависно од путање на коју се усмерава корисник. У класи *AppModules* су дефинисане све компоненте *root* модула и увезени су сви модули које овај модул користи. У *AppRoutingModule* класи је дефинисана лења евалуација модула. На основу путање дефинисано је који модул је потребно учитати.

*Shared* модул садржи компоненте које су заједничке за више модула, а то су компоненте за преглед већ попуњеног упитника. Овде су дефинисани и *pipes* који омогућавају униформан приказ за датуме и енумерације. У *interceptors* фолдеру је дефинисан *interceptor* који обрађује сваки захтев који се упућује ка серверској апликацији тако што уграђује токен за ауторизацију у сваки захтев.

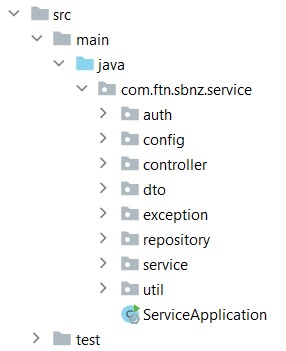
У *аuth* модулу се налазе компоненте за аутентификацију корисника. То су компонента за пријаву корисника коју може користити било који корисник, компонента за регистрацију новог корисника коју користе корисници који желе да се региструју и користе апликацију.

У *client* модулу се налазе све компоненте које су доступне клијенту. То су компоненте за попуњавање упитника, преглед добијене дијагнозе и преглед одговора на питања, преглед симптома за одабрано ментално стање. Такође, ту су и компоненте за приказ психолога који имају приступ подацима клијента и компоненте за додавање у уклањање психолога.

У *psychologist* модулу налазе се компоненте којима може да приступа психолог. Психолог може да додаје нова питања за упитник и дефинише правила за обраду тих питања. Такође може да креира правила на основу питања која су означена као маркери за стање депресије. Психологу су доступна три типа извештаја. Први тип се односи на преглед дијагностикованог стања и одговора за одабраног корисника за кога има приступ. Други извештај приказује резултате упита на основу године рођења, професије и пола корисника. Трећи извештај представља график на ком се може пратити одабрано ментално стање у одабраном временском опсегу.

### Серверска апликација

Серверска апликација је имплементирана коришћењем *Spring* радног оквира. Клијентска и серверска апликација комуницирају по принципу захтева и одговора. Сервер комуницира и са апликацијом која представља модел, системом базираним на знању и базом података. Изворни код апликације је структуриран у фолдере: *auth*, *config*, *controller*, *dto*, *exception*, *repository*, *service* и *util* Слика 17. У *auth* фолдеру су смештене класе које се баве ауторизацијом и токенима. *Config* фолдер садржи класе које се баве конфигурацијом. *Util* фолдер садржи класу за рад са *JSON* веб токенима. У *exception* фолдеру су класе које моделују изузетке које се могу јавити у апликацији. *Data Transfer Objects* класе које моделују податке за размену са клијентском апликацијом се налазе у *dto* фолдеру.



Слика Структура серверског дела

*Hibernate* је коришћен за објектно-релационо мапирање. Интерфејси репозиторијума који су смештени у *repository* фолдеру наслеђују и проширују *JpaRepository* интерфејс. Репозиторијуми који постоје у апликацији су:

* UserRepository
* RoleRepository
* QuestionRepository
* AnswerRepository
* ResultRepository
* DetectionTypeRepository
* DiagnosticTypeRepository
* DepressionMarkResultRepository

Методе репозиторијума се позивају у класама сервиса. Пример репозиторијума је приказан на Листинг 4.

Листинг Пример репозиторијума - DetectionTypeRepository

@Repository  
public interface DetectionTypeRepository extends JpaRepository<DetectionType, Long> {  
  
 DetectionType getDetectionTypeByType(String type);  
 DetectionType getDetectionTypeById(Long id);  
 List<DetectionType> getDetectionTypeByDetectionGroup(DetectionTypeGroup group);  
}

Сервиси су смештени у *service* фолдеру. Сервиси су имплемнтирани као интерфијси који имају своје импламентације. У класама сервиса је имплементирана пословна логика система. Сервиси који постоје у апликацији су:

* UserService
* QuestionService
* AnswerService
* ResultService
* DetectionTypeService
* DiagnosticService
* DiagnosticTemplateService
* PsychologistTemplateService
* ResultService

Листинг Пример сервиса - Diagnostic service

public interface DiagnosticService {  
 ArrayList<String> getAllSymptoms(String diagnostic);  
 ResultDTO getDiagnostics(List<AnswerDTO> answers, String loggedInUser) throws IOException;  
 List<DepressionMarkDTO> getResultForDepressionMark(String userEmail);  
 String makeNewRules(RuleDTO ruleDTO) throws IOException;  
 void addTemplateRules(List<TemplateParamDTO> templateParamDTOs) throws IOException;  
}

Пример интерфејса једног од сервиса дат је на Листинг 5.

Клијентска и серверска апликација комуницирају тако што клијентски део шаље захтеве серверу, а сервер одговара на добијене захтеве. Захтеви клијентске апликације покрећу функције које су смештене у класама које су анотиране *@RestController* анотацијом. Контролери који постоје у систему су:

* *AuthController*
* *Usercontroller*
* *QuestionController*
* *DiagnosticController*
* *DetectionTypeController*
* *ReportContoller*
* *PsychologistController*

Како су модел, кјар и сервис посебни *maven* пројекти, потребно је у *pom.xml* сервис пројкта додати одговарајуће зависности и повезати ове пројекте. У Листинг 6 приказане су зависности које су додате у *pom.xml* фајл како би се повезали модел и кјар пројекат са сервис пројектом.

<dependency>  
 <groupId>com.ftn.sbnz</groupId>  
 <artifactId>kjar</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>  
  
<dependency>  
 <groupId>com.ftn.sbnz</groupId>  
 <artifactId>model</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

Листинг Зависности модел и кјар пројектс у pom.xml фајлу

За инстанцирање сесије која омогућава додавања чињеница у базу знања и покретање модула за закључивање је потребно добавити инстанцу *KieContainer-*a. Зато је у *main* класу потребно додати код из Листинг 7.

Листинг Код који омогућава инстанцирање KieContainer-a

@Bean  
public KieContainer kieContainer() {  
 KieServices ks = KieServices.Factory.*get*();  
 KieContainer kContainer = ks  
 .newKieContainer(ks.newReleaseId("com.ftn.sbnz", "kjar", "0.0.1-SNAPSHOT"));  
 KieScanner kScanner = ks.newKieScanner(kContainer);  
 kScanner.start(1000);  
 return kContainer;  
}

## Drools апликација

Систем базиран на правилима који доноси одлуке на основу if-then правила имплементиран је као засебни *maven* пројекат и користи *Drools* алат.

# Приказ имплементираног система

# Закључак

# Литература

1. “ПРОГРАМ О ЗАШТИТИ МЕНТАЛНОГ ЗДРАВЉА У РЕПУБЛИЦИ СРБИЈИ ЗА ПЕРИОД 2019-2026. ГОДИНЕ,” n.d.

1. <https://www.wysa.com>
2. <https://angular.io/docs>
3. <https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html>
4. <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>
5. <https://www.ibm.com/topics/rest-apis>
6. <https://docs.jboss.org/drools/release/7.35.0.Final/drools-docs/html_single/>
7. <https://hibernate.org/>
8. <https://www.uml-diagrams.org/>
9. Milosavljević, Gordana. *Uvod u modelovanje softvera*. Novi Sad: Fakultet tehničkih nauka, 2020.

# Биографија

Милица Петровић је рођена 21. априла 2000. године у Сремској Митровици. Завршила је Основну школу „Бранко Радичевић“ у Шиду. Гимназију „Јован Јовановић Змај“ завршила је у Новом Саду. Године 2019. уписала је Факултет техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије. Све испите полаже и студије завршава у року, 2023. године.