

**Семинарски рад**

Предмет:

**Електронско пословање**

Тема:

**Веб апликација за анализу игара**

|  |  |
| --- | --- |
| Професор:  Душан Бараћ | Студент:  Петар Јеремић 2015/0093 |
|  |  |

Београд, 2020. Године

Contents

[1. Увод 1](#_Toc50370046)

[2. Коришћене технологије 2](#_Toc50370047)

[Spring data 2](#_Toc50370048)

[Spring security 2](#_Toc50370049)

[Spring mvc 3](#_Toc50370050)

[Spring container 4](#_Toc50370051)

[JWT token 5](#_Toc50370052)

[Angular.js 6](#_Toc50370053)

[Liquibase 7](#_Toc50370054)

[Mapstruct 7](#_Toc50370055)

[**Git** 7](#_Toc50370056)

[**Rest API** 8](#_Toc50370057)

[3. Вербални опис 9](#_Toc50370058)

[4. Случајеви коришћења 10](#_Toc50370059)

[4.1 СК4: Случај коришћења – Пријављивање корисника 11](#_Toc50370060)

[3. Анализа 11](#_Toc50370061)

[3.1 Дијаграми секвенци 11](#_Toc50370062)

[3.1.4 ДС4: Дијаграми секвенци случаја коришћења – Пријављивање корисника 11](#_Toc50370063)

[3.3 Концептуални модел 12](#_Toc50370064)

[6 Пројектовање 13](#_Toc50370065)

[6.1 Архитектура система 14](#_Toc50370066)

[6.2 Пројектовање корисничког интерфејса 14](#_Toc50370067)

[6.2.1 Пројектовање веб страница 14](#_Toc50370068)

[6.5 Пројектовање апликационе логике 19](#_Toc50370069)

[6.5.1 Део за комуникацију са клијентима 19](#_Toc50370070)

[6.5.2 Део који садржи пословну логику 19](#_Toc50370071)

[6.5.3 Део који комуницира са складиштем података 19](#_Toc50370072)

[6.6 Пројектовање складишта података 20](#_Toc50370073)

[6.6.1 Табела user 20](#_Toc50370074)

[6.6.2 Табела user\_following 21](#_Toc50370075)

[6.6.3 Табела game 21](#_Toc50370076)

[6.6.4 Табела game\_score 21](#_Toc50370077)

[6.6.5 Табела game\_session 21](#_Toc50370078)

[6.6.6 Табела game\_session\_player 21](#_Toc50370079)

[6.6.7 Табела game\_session\_player\_strategy 22](#_Toc50370080)

[6.6.8 Табела verification\_status 22](#_Toc50370081)

[6.6.9 Табела rank 22](#_Toc50370082)

[6.6.10 Табела player\_configuration 22](#_Toc50370083)

[6.6.11 Табела payoff 22](#_Toc50370084)

[6.6.12 Табела strategy 23](#_Toc50370085)

[23](#_Toc50370086)

[7 Закључак 23](#_Toc50370087)

[9 Литература 23](#_Toc50370088)

# Увод

Софтверски инжењеринг представља сваку врсту креације помоћу давања наредби рачунару и као такав постоји од првих дана програмирања као признатог појма. Почетком, програми су били једноставна рачунања која користе предност рачунара који је бржи од човека. Међутим, још увек, машина ради једино оно што јој човек каже. Као и сваки нови појам, програмирање је прво било заступљено у војсци и за научне потребе, али данас можемо да видимо програмирање за потребе индустрије, забаве, али и свакодневних активности. Како су детаљи све већи, системи који се праве постају све сложенији, не само због захтева који су све већи и компликованији, већ и због броја корисника и све веће одвојености инжењера и корисника. Због тога, инжењеров задатак је да колико је могуће, свој програм направи на такав начин да случајеви коришћења буду јасни и недвосмислени и да, у најбољем случају, корисник може да га користи без читања упутсва тј. да сви аспекти програма буду интуитивно јасни[[1]](#footnote-1). Поред тога, корисник очекује да од апликације добије одговоре тренутно и ако чека неки одговор, да може да ради нешто друго у апликацији док чека. Све ово и још много тога покушаћемо да објаснимо кроз пример веб апликације која користи клијент-сервер архитектуру[[2]](#footnote-2). Кроз ову апликацију покушаћемо да објаснимо и са којим проблемима програмер може да се сусретне, шта се подразумева да његова апликација има и који су неки од најпрепорученијих алата да му олакшају да задовољи све те захтеве. Управо прво почињемо са представљањем тих алата.

# Коришћене технологије

У овом поглављу објаснићемо неке од основних концепта технологија које су коришћене у развоју софтвера. Као оквир за прављење сервера коришћен је Spring[[3]](#footnote-3) са неким од великог броја модула које он пружа. Неки од њих су:

* Spring data
* Spring security
* Spring mvc
* Spring container

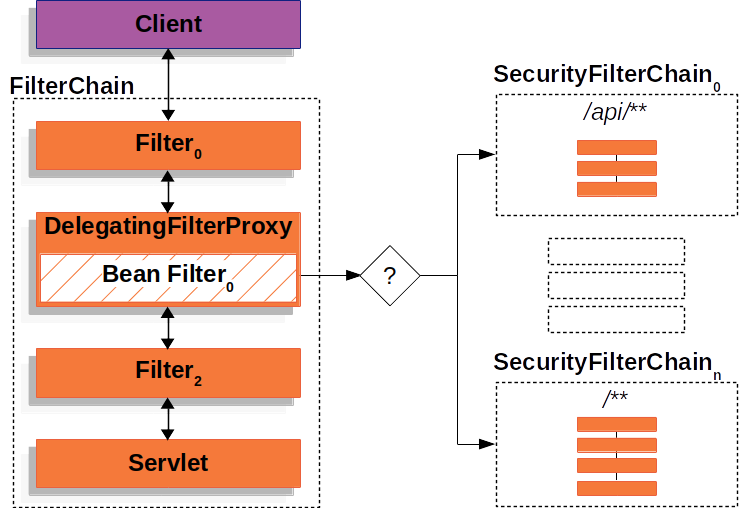
Да бисмо користили све могућности које нам овај оквир пружа, морамо конфигурисати наш пројекат на прави начин. Већина програмера почетника у овом кораку често греши тако да је у помоћ притекао Spring boot[[4]](#footnote-4) који управо упрошћава коришћење апликација вођених овим оквиром. Укратко ћемо објаснити сваки од наведених делова који су коришћени.

## Spring data

Spring data нам омогућава ефективну комуникацију са изабраном базом података. Како релационе базе података имају другачију логику представљања и повезивања података од програма писаних у објектно орјентисаном језику, потребно је након извлачења из базе пребацити из релационог у објектни модел података. Тај посао може да буде доста напоран и монотон тако да нам Spring data омогућава коришћење објектно релационог мапера[[5]](#footnote-5) који тај посао ради за нас, на нама је да само поставимо везе у објектом моделу. Постоји више објекто релационих мапера које можемо користити[[6]](#footnote-6). У овом раду, коришћен је Hibernate.

## Spring security

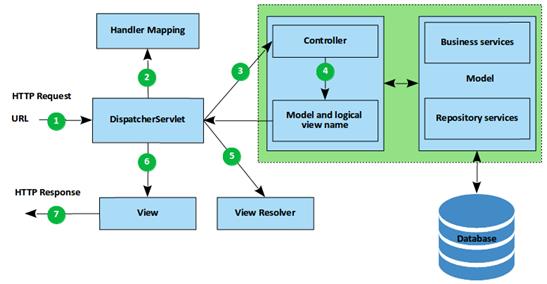
Како су захтеви све већи и системи све сложенији, све више ствари може да пође по злу. Један од највећих изазова су сигурносни проблеми. Spring security омогућава прагматичан приступ решавању сигурносних проблема као што су малициозни напади на систем и аутентификација и ауторизација корисника. Функционише тако што клијентов захтев полази кроз одређене филтере који уграђени али и дефинисани од стране инжињера. Сваки филтер може да онемогући даљу обраду захтева.



Слика Spring security филтери

## Spring mvc

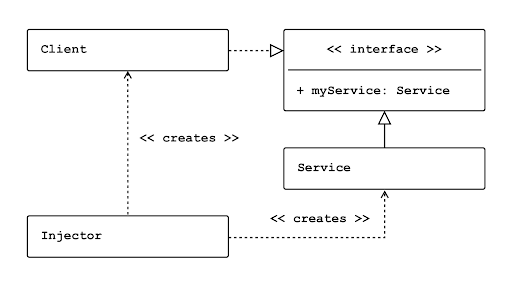
Spring mvc представља имплементацију Model–view–controller[[7]](#footnote-7) патерна и његовог коришћења у веб апликацијама. Сваки захтев долази на главни контролер који накнадно прослеђује захтев другим контролерима у зависности од тога шта се налази у путањи. Какав ће бити одговор зависи од разрешавања захтева.



Слика Spring mvc

## Spring container

Spring container је део који програмерима олакшава проблем инстанцирања објеката имплементирајући Dependency injection[[8]](#footnote-8) технику. Овом техником, клијент који позива интерфејс независтан је од његове имплементације тј. задужење инстанцирања није на клијенту. Ово омогућава промену имплементације без знања клијента.



Слика Dependency injection

## JWT token[[9]](#footnote-9)

За прављење клијент-сервер апликације у доста ситуација клијентски и серверски програм немају толико сличности у виду технологија које користе тако да сигурносни проблеми ту постају сложенији. Spring security даје два начина за аутентификацију: базична аутентификација и аутентификација преко форме. У коришћеној верзији у овом пројекту, само укључивање Spring security модула подразумева аутоматско генерисање форме и аутентификацију преко форме. Ако се ништа не конфигурише, сервер прави корисника са генерисаном шифром. Како у овом пројекту за клијентски програм нећемо користити програмски језик Јава, потребан нам је други вид аутентификације. JWT token представља енкодирани стринг који има три дела.

Први део садржи следеће информације:

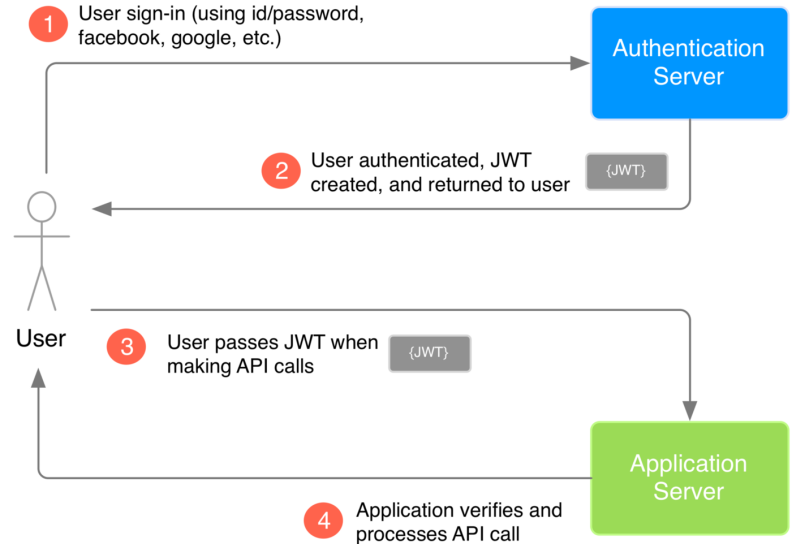
* алгоритам коришћем за енкодирање токена
* тип токена

Други део:

* изадвача токена
* онога коме је токен намењен
* време издавања
* време истека важења токена
* коме је издат токен

Трећи део садржи верификациони потпис.

Следећа слика приказује ток коришћења токена.



Слика Jwt Token

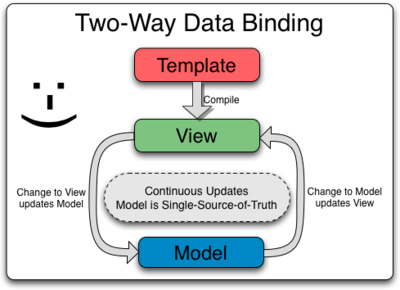
## Angular.js

Као што смо већ напоменули, клијентски и серверки програм ће бити потпуно одвојени. То заправо значи да можемо да направимо нпр. други серверски програм и да га употпуности спојимо са клијентским програмом са 0% мењања клијентског програма. За развој клијентског програма користићемо један од најчешће коришћених javascript оквира Angular.js. Неки од других оквира који се користе су React.js, Vue.js. Једна од главних предности оквира који ћемо користити је што он користи typescript који је заправо модификовани javascript који садржи какву такву проверу типова за време компајлирања. Ово је лабаво решење јер се typescript изворни код компајлира у javascript изворни код и потом се извршава као javascript скрипта. Проблем у овоме је, иако вам компајлер јавља грешку, и поред тих грешака, скрипта ће се покренути и извршити. Међутим, великом броју програмера и таква провера типова олакшава развој.



Слика Typescript u javascript

Angular.js представља апликацију као једну веб страницу која динамички мења садржај тако да корисник добија само илузију да мења странице. Он, заправо, мења само садржај једне једине странице. Предност коришћења оквира се огледа у томе што умногоме упрошћава динамичку интеракцију са корисником у зависности од тога које захтеве корисник прави. Један пример тога је што страница мења свој садржај ако смо неку колекцију података везали за неки елемент у ДОМ-у[[10]](#footnote-10) тј. садржај странице се мења ако се промени садржај колекције. Још једна предност је што та повезаност не мора да буде једносмерна. Ако корисник промени стање елемента, може се променити стање неког модела података.



Слика Data binding Angular.js

## Liquibase

Омогућава: верзионисање базе података

Кроз развијање апликације често се мења модел података. Не тако ретко, пројектант заборави да дода неко поље у објекат или да дода неко структурно ограничење нпр ограничење јединствености колоне . Ове промене, поготово ако се пројекат ради у тиму, морају да се доносе са посебном пажњом јер често може доћи до несинхронизованих база (када се базе код једног члана разликују од база код другог члана) што може створити проблеме у будућности које је тешко открити. Liquibase омогућава верзионисање базе и на тај начин чува стање не само података у бази већ и саме њене шеме. Овим софтвером смо омогућили такозвани „Save game“ у игри развоја наше базе података.

Овај софтвер функционише тако што се одређени sql ili xml фајлови енкодирају у базу и промене базе које се налазе у њима се валидирају сваки пут када се овај софтвер покрене. Ако не постоји конзистентност, програм ће бацити грешку.

## Mapstruct

Омогућава: пребацивање из једне врсте модела објеката у друге

У комуникацији између клијента и сервера, врло је битно да се на клијент шаљу само оне информације које су клијенту потребне и обратно. Да би сервер то обезбедило, мора се направити одређено мапирање из објеката који се добијају из базе у објекте који се преносе преко мреже. Mapstruct аутоматизује посао пребацивања из једних у друге објекте. Функционише тако што програмер само дефинише објекте који су потребни за комуникацију и мапере који ће вршити мапирање. Mapstruct обезбеђује имплементацију мапирања.

**Git**

Омогућава: верзионисање кода

У једном од претходних поглавља, поменут је Liquibase као софтвер за верзионисање базе података. Најмање подједнаке важности али вероватно и важније је верзионисање кода. Код је чисто издавање наредби машини и управо ту највише ствари може да пође по злу (поготово ако се се ради у тиму и пројектант нема потпуну контролу над системом који развија). Git представља софтвер за верзионисање кода али и много више од тога. Чак и ако један програмер ради на систему, увек је добра могућност снимити тренутно стање знајући да тада све ради па, ако у даљем развоју нешто пође по злу, увек може да се врати на то стање. Овај софтвер представља још већег савезника ако се ради у тиму јер тада, иако су програмери радили на једном истом фајлу, тај фајл лако може да се интегрише преко овог софтвера.

**Rest API**

Representational state transfer (REST) представља стил комуникације порграма на мрежи. Овај стил је потребан из истог разлога зашто су протоколи потребни комуникацији. Са њима се лакше споразумевају енитети у мрежи тј. лакше интерпретирају податке које су примили и наредбе које су им послате. Сваки сервер добија захтев за приступ неком веб ресурсу који може бити неки документ али и акцију или ентитет. Тај ресурс мора бити доступан преко URI-а и сервер као одговор на приступ том ресурсу шаље као одговор неки документ у формату html, xml, json или неком другом. Једна од најзначајнијих предности овог стила је интероперабилност. (разни клијенти могу лако да комуницирају са сервером који је вођен овим стилом)

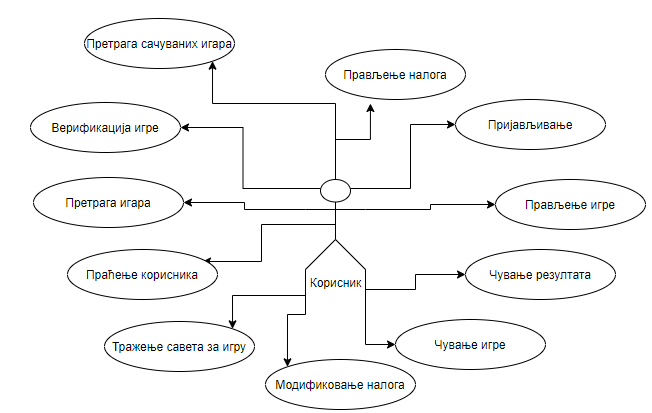
У следећим поглављима описаћемо у детаље изглед ситема који се пројектује и све његове функционалности.

# Вербални опис

Софтвер се користи за прављење играра за анализу теорије игара као и изучавање тих игара али и интеракција са корисницима који имају иста интересовања. Корисник прави налог и помоћу тог налога користи софтвер. Када направи налог, пребачен је на главну страницу која садржи мени помоћу ког може да:

* Иде на свој профил где може да мења податке о свом налогу
* Одабере опцију прављења нове игре и затим направи нову игру коју може да игра као и остали корисници
* Игра игру, сачува је и пошаље резултат на сервер. Ако је игра верификована, повећава му се искуство што може да доведе до промоције тј повећања ранга што узрокује додатне привилегије
* Верификује игру. Верификација зависи од ранга који корисник има
* Гледа своје сачуване игре и поново их игра
* Добија аналитичке савете за игру коју игра
* Прати друге кориснике

# Случајеви коришћења



Слика Модел случајева коришћења

На основу вербалног описа идентификовани су следећи случајеви коришћења који се могу видети на Слици 1:

1. Прављење налога
2. Модификовање налога
3. Прављење игре
4. Пријављивање
5. Чување резултата
6. Чување игре
7. Тражење савета за игру
8. Праћење корисника
9. Претрага игара
10. Претрага сачуваних игара
11. Верификовање игре

## 4.1 СК4: Случај коришћења – Пријављивање корисника

**Назив СК**

Пријављивање корисника

**Актори СК**

Корисник

**Учесници СК**

Корисник и систем (програм)

**Предуслов**: Систем је укључен и приказује форму за пријављивање.

**Основни сценарио СК**

1. Корисник **уноси** вредностпо којој се пријављује. (АПУСО)
2. Корисник **позива** систем да се пријави. (АПСО)
3. Систем **тражи** налог по задатој вредности. (СО)
4. Систем приказује кориснику поруку: “Успешно сте се пријавили”. (ИА)

Алтернативна сценарија

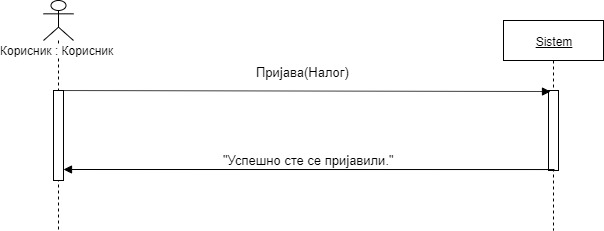
4.1 Уколико систем не може да нађе налог он приказује кориснику поруку: “Пријављивање није успело”. (ИА)

# Анализа

## 3.1 Дијаграми секвенци

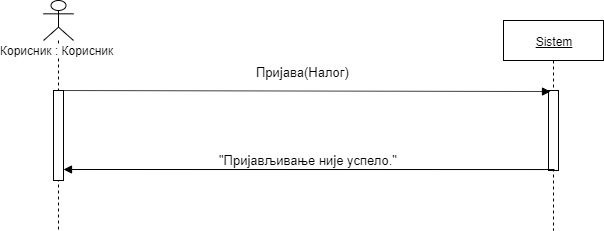
### 3.1.4 ДС4: Дијаграми секвенци случаја коришћења – Пријављивање корисника

1. Корисник **позива** систем да се пријави. (АПСО)
2. Систем приказује кориснику поруку: “Успешно сте се пријавили”. (ИА)



***Алтернативна сценарија***

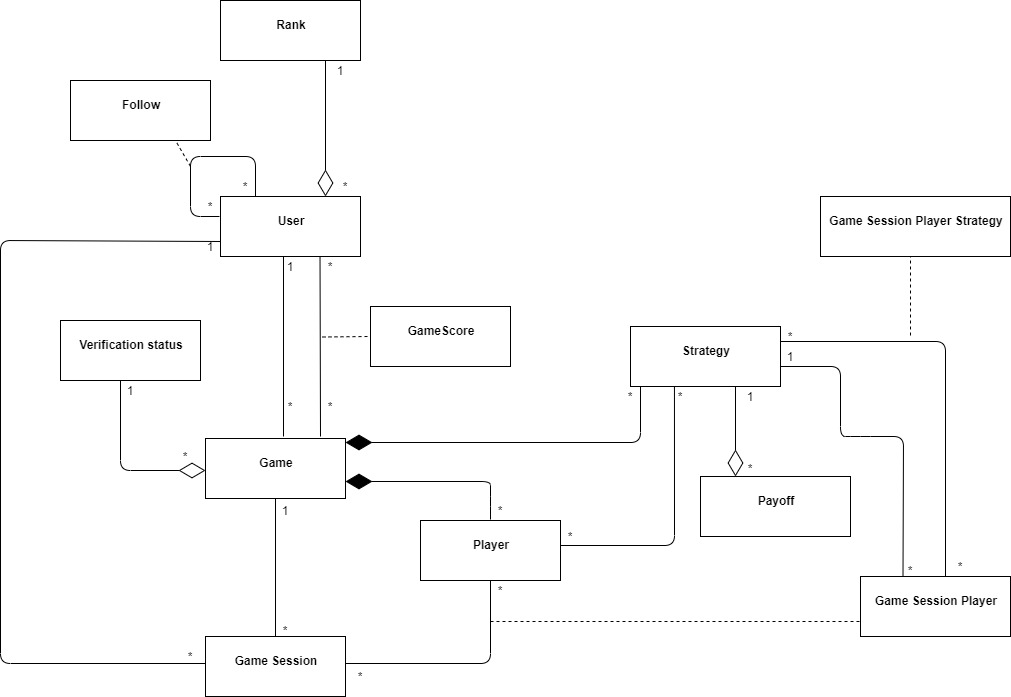
* 1. Уколико систем не може да нађе налог он приказује кориснику поруку: “Пријављивање није успело”. (ИА)



Са наведених секвенцних дијаграма уочава се још 1 системска операција коју треба пројектовати:

1. сигнал Пријава(Налог);

## 3.3 Концептуални модел



Слика Концептуални модел

# Пројектовање

У овој фази биће објашњена и направљена физичка структура и понашање система тј. Архитектура система. Ова фаза одлучује природу нашег система и његову будућност што значи да, иако се систем пројектује тако да испуњава захтеве, то не значи да је систем добро пројектован. Како се то може проверити? Ако надоградња и одржавање система у будућности захтевају све веће и веће трошкове до тог момента када је јефтиније прављење нове архитектуре, систем није добро пројектован.

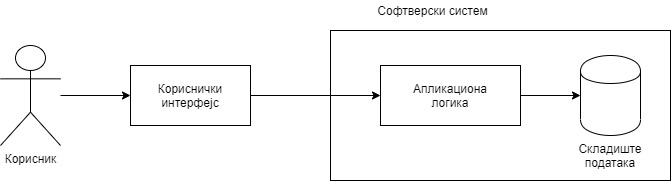
## Архитектура система

У овом раду архитектура која ће се користити се назива Клијент-Сервер архитектура. Шта то заправо значи? Цео систем се састоји од два дела од којег један део (Сервер) прима захтеве од другог дела (Клијент), обрађује их и шаље клијенту одговарајуће одговоре.

Архитектура система који се пројектује у овом раду садржи три главна нивоа:

* Кориснички интерфејс
* Апликациона логика
* Складиште података

Који од ових нивоа се налази на клијентској страни а који на серверској страни? Није увек могуће повући линију али се тежи ка томе да апликациона логика и складиште података буду на серверској страни тј. да клијент буде растерећен. Растерећеност клијента се огледа у томе да троши минималну количину ресурса а да при томе сви његови захтеви буду задовољени.



Слика тронивојска архитектура

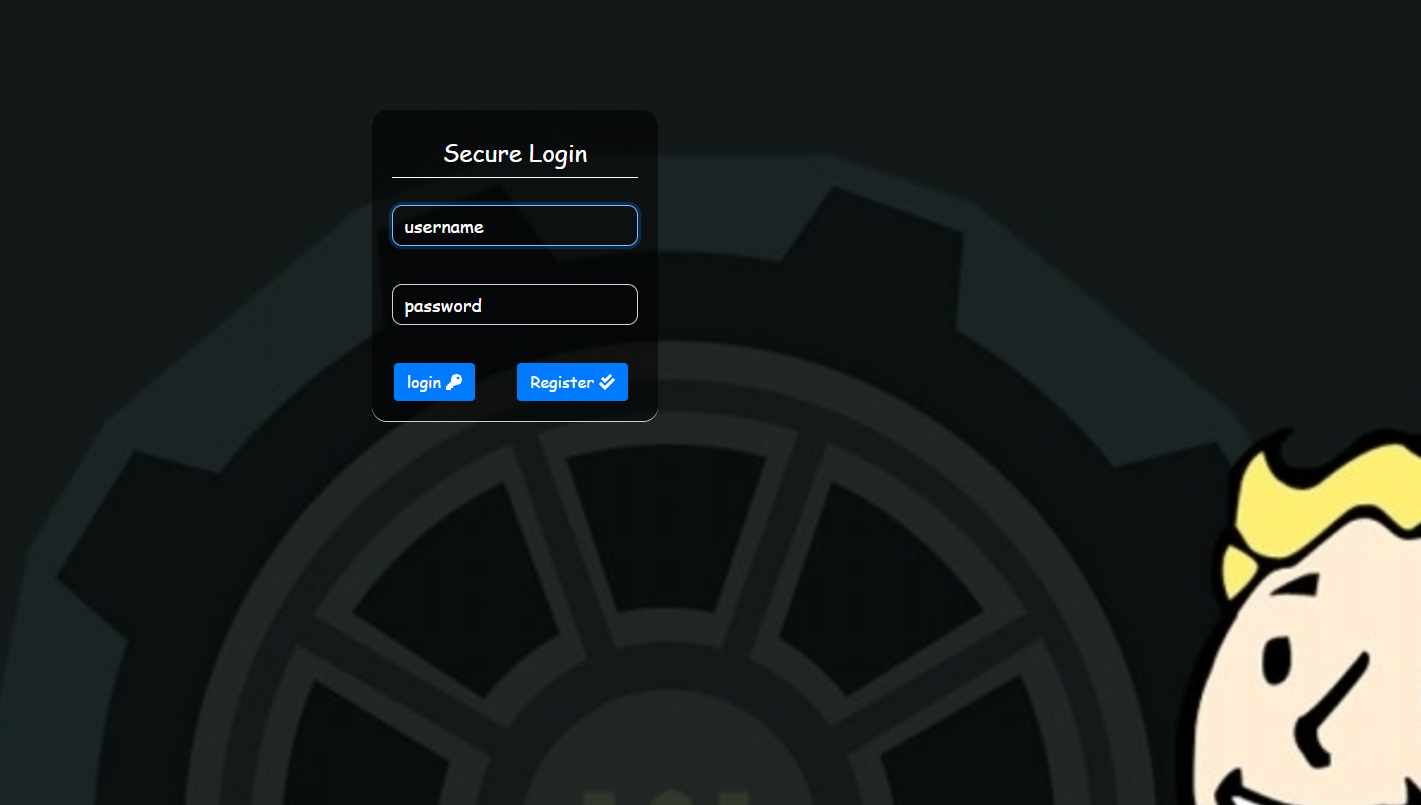
## Пројектовање корисничког интерфејса

Кориснички нитерфејс представља начин на који ће корисник комуницирати са софтверским системом и реализовати своје захтеве. У овом случају, корисник ће своје захтеве слати уз помоћ дугмића на веб страници.

### Пројектовање веб страница

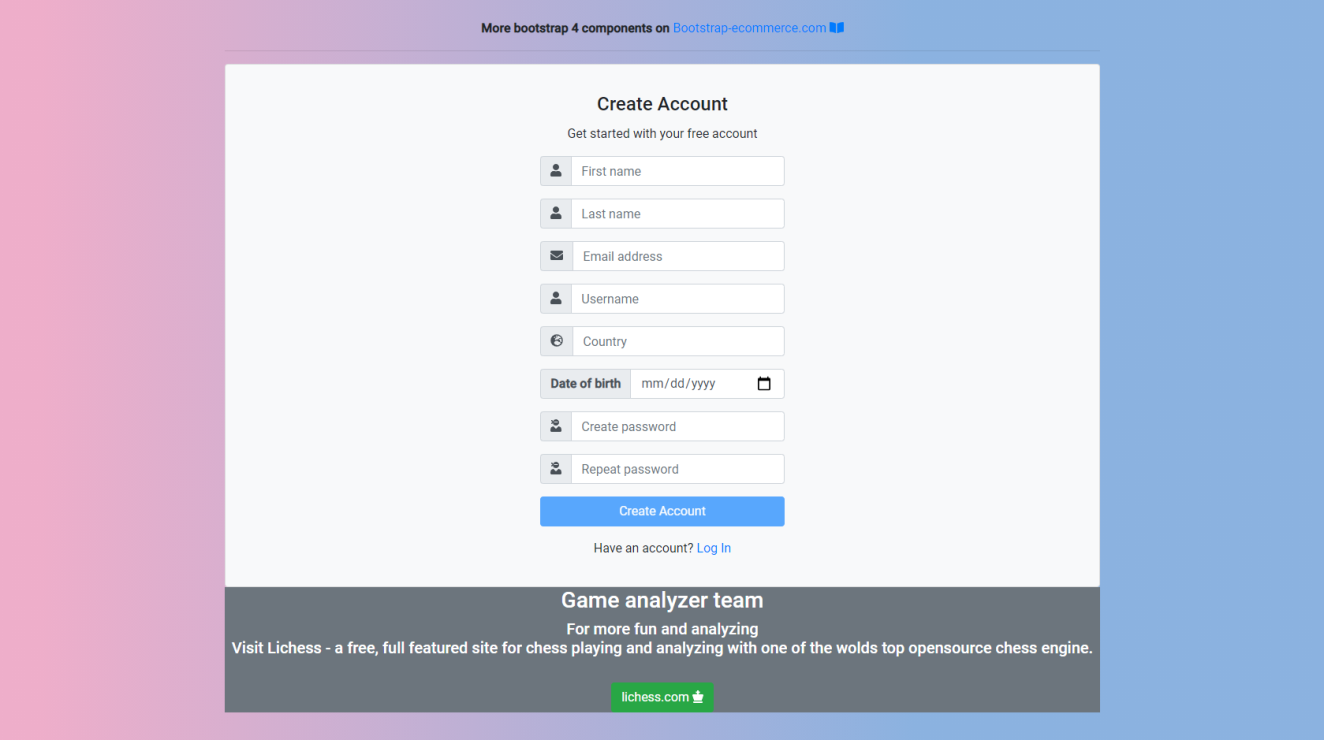
Као што је већ наведено, веб страница је оно што корисник види и оно чиме корисник управља. Да би покрио све случајеве коришћења, пројектант мора да има у виду то да је кориснику врло битно време за које ће да савлада коришћење софтвера тако да пројектант мора да направи визуелни приказ који је логичан и, у идеалном случају, корисник ни не мора да гледа упутсво за коришћење да би могао да користи софтвер.

Сваки клијент почиње са веб страном за пријављивање и тек пошто се пријави може користити остатак софтвера. Ако клијент нема налог, може започети прављење налога преко ове форме:



Слика Веб страна за пријаљивање - почетна форма

Веб страна за креирање налога изгледа овако:



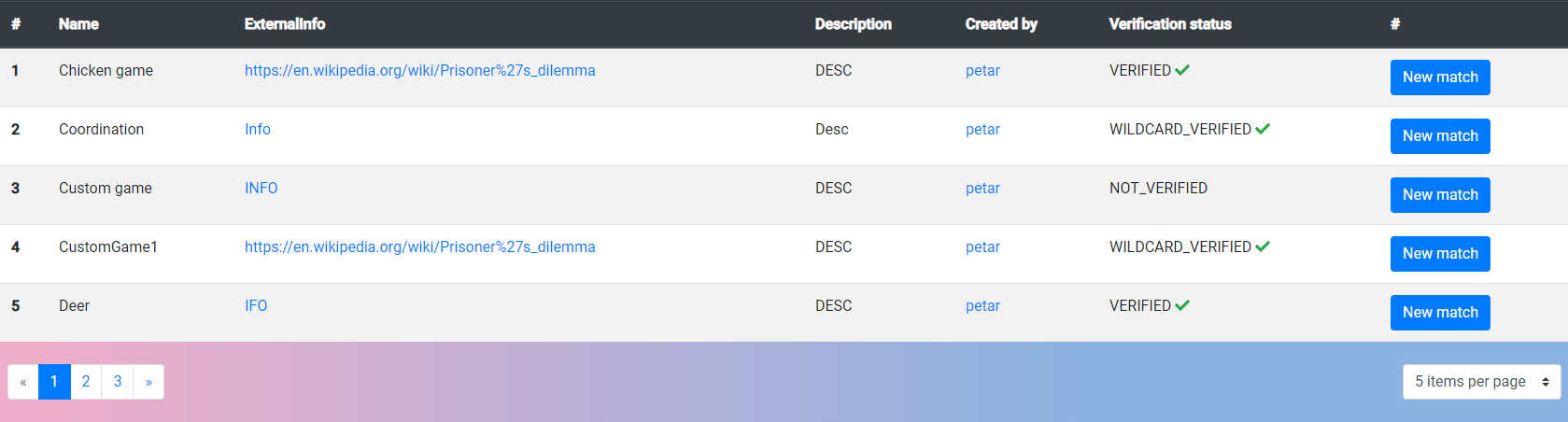
Слика Веб страна за креирање налога

Након што се корисник успешно пријави, може користити све функционалности софтвера. Прва ствар коју ће видети после пријављивања то је хоме веб страна:



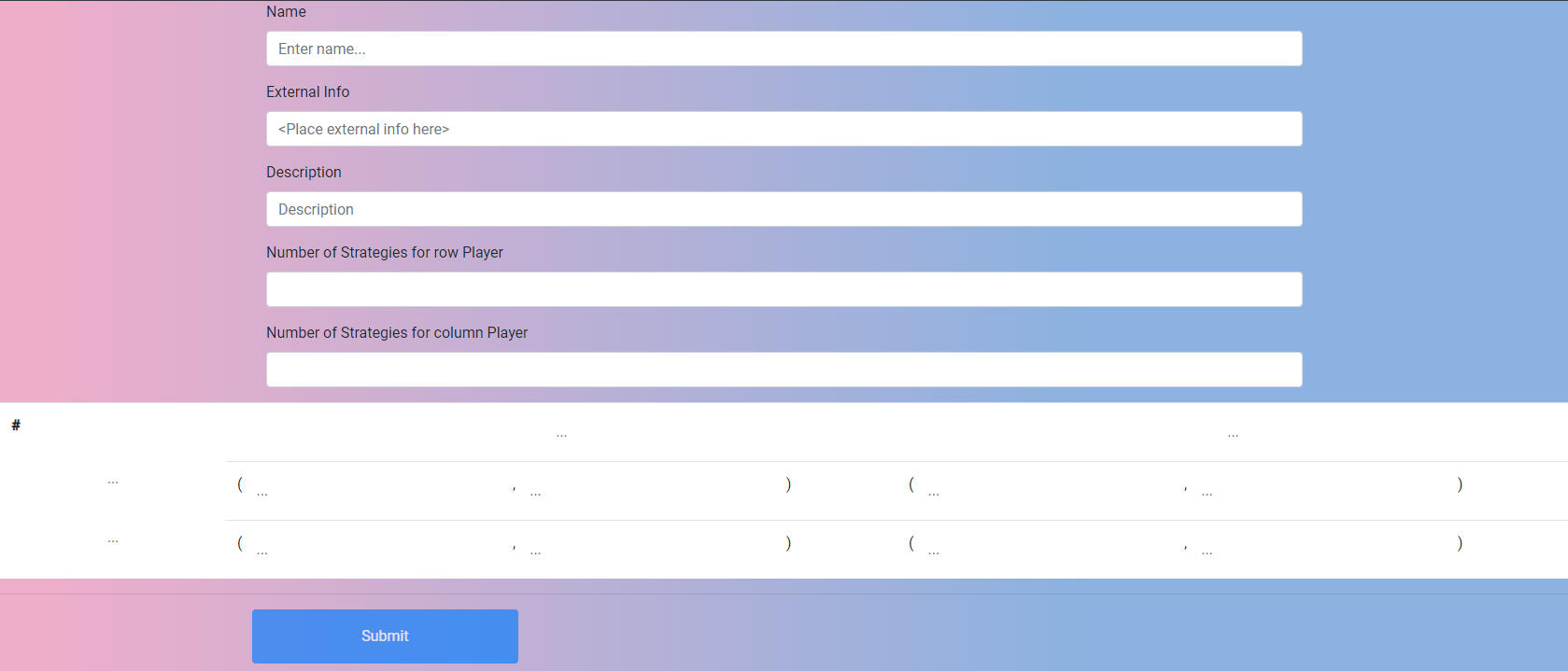
Слика Главна Веб страна клијента

Веб страна за претрагу игара:



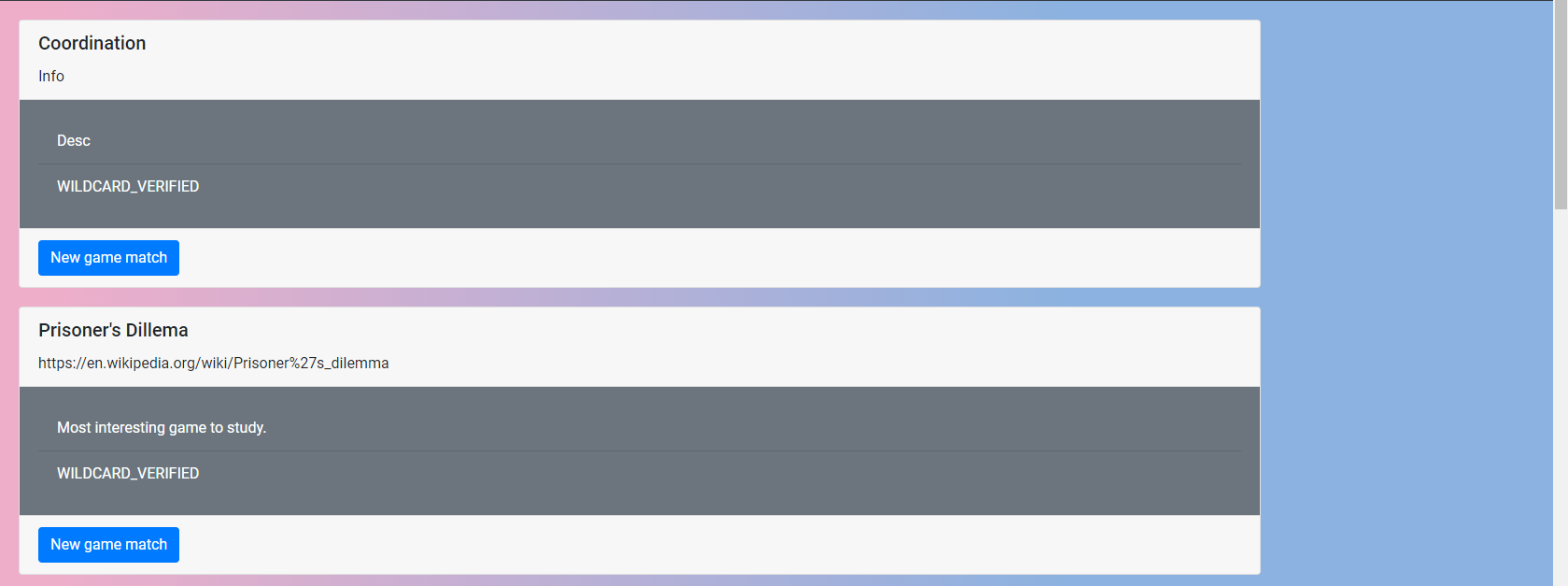
Слика Веб страна за претрагу свих игара

Веб страна за креирање игара:



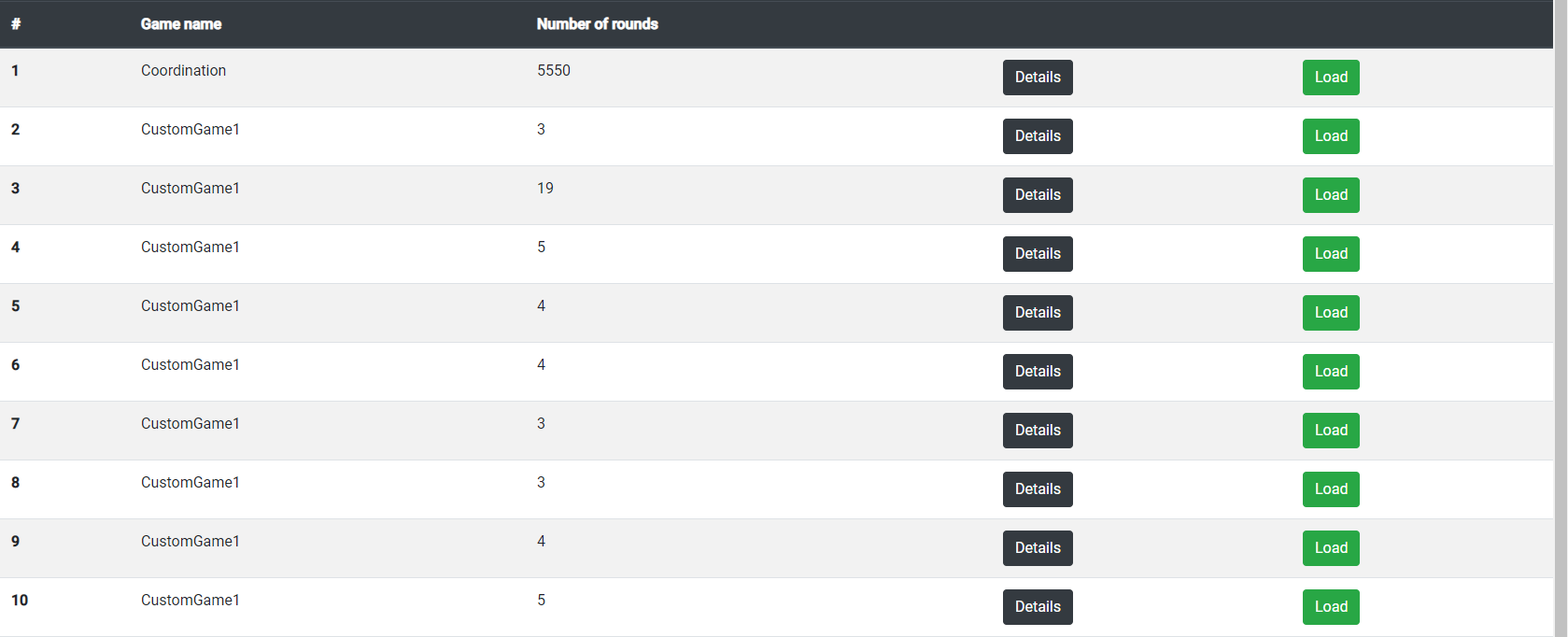
Слика Веб страна за креирање игара

Веб страна за претрагу корисникових игара:



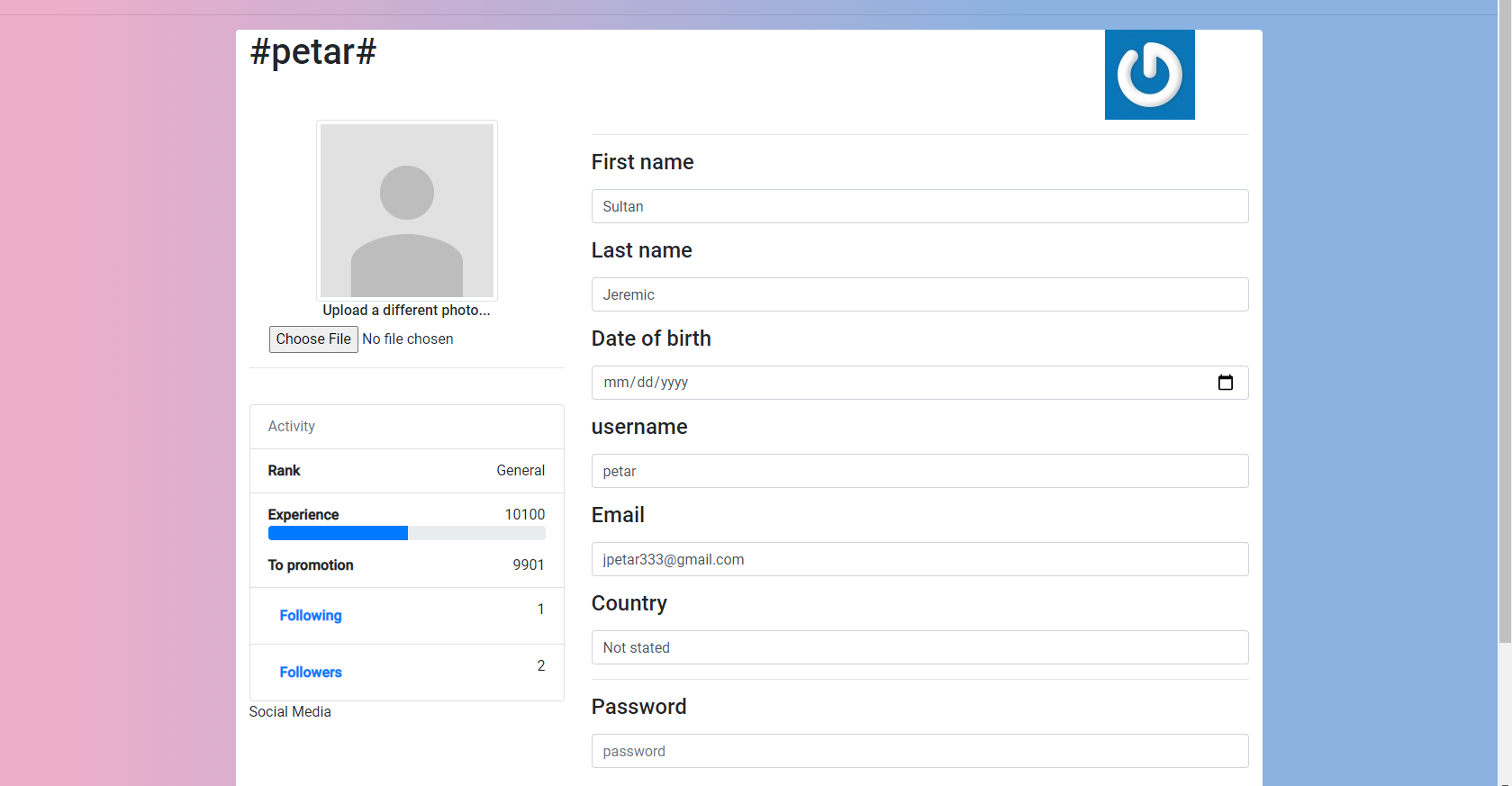
Слика Веб страна за игре корисника

Веб страна за претрагу свих сачуваних игара корисника:



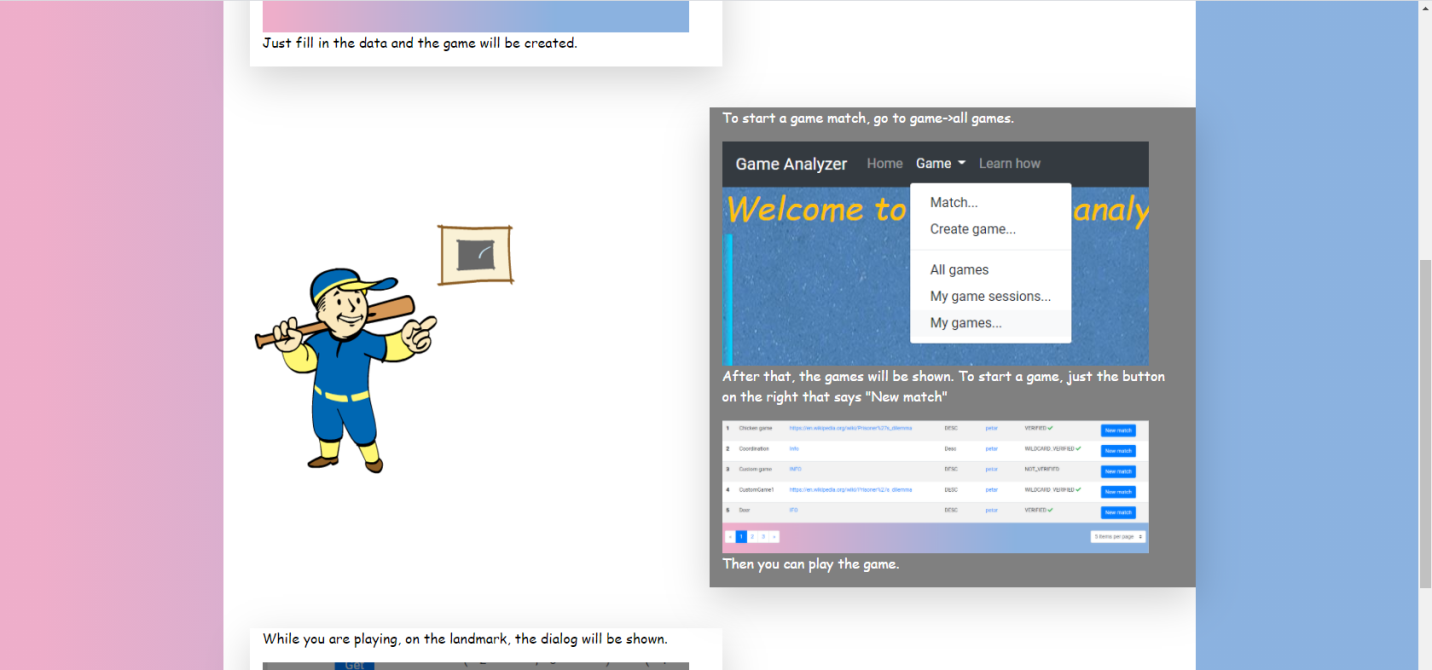
Слика Веб страна за сачуване игре корисника

Веб страна за профил корисника:



Слика Веб страна за профил корисника

Веб страна за инструкције:



Слика Веб страна за инструкције

## 6.5 Пројектовање апликационе логике

Апликациона логика има задужење да омогући све сервисе који ће постићи правилно коришћење система. У пројектовање апликационе логике спада пројектовање:

* Део за комуникацију са клијентима(контролери)
* Део који комуницира са складиштем података(репозиторијум интерфејси)
* Део који садржи пословну логику(сервиси)

### 6.5.1 Део за комуникацију са клијентима

Комуникација представља омогућавање двосмерног канала за проток података између сервера и клијента. Клијент шаље путању ресурса којем хоће да приступи, док сервер прима тај захтев и шаље му одоговарајући одговор.

### 6.5.2 Део који садржи пословну логику

Након примања захтева, сервер обично треба да одради одређени низ наредби који се од њега очекује. Те наредбе могу бити извлачење података, неко израчунавање или нешто слично. За то су задужени сервиси који обично иду раме уз раме са контролером и контролер позива те сервисе. Након позива. Севис враћа контролеру податке које контролер шаље клијенту.

### 6.5.3 Део који комуницира са складиштем података

Репозиторијуми се обично везују за објекат из концептуалног модела али то не мора нужно да значи. Шта је свакако битно, добар систем дозвољава једна врата комуникације са другим системом. Како су контролери били врата за комуникацију са клијентом, репозиторијуми су врата за комуникацију са системом за управљање базом података.

## 6.6 Пројектовање складишта података

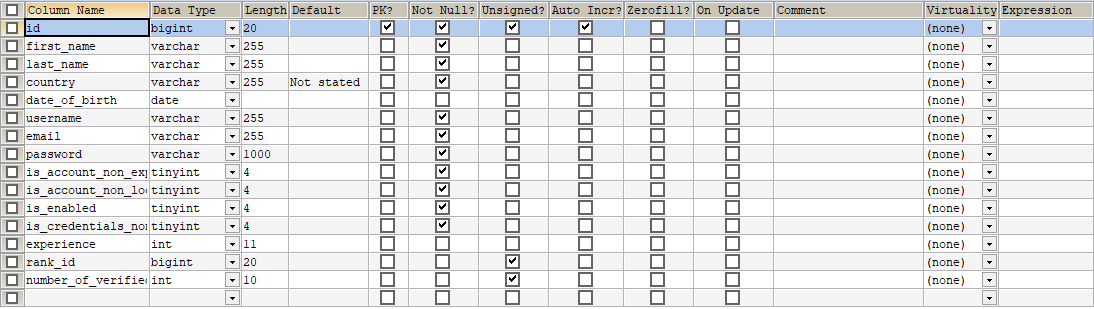
На основу објетног модела и класа које њега представљају, можемо обезбедити перзистентност на више начина од који су два најчешћа:

* Чување у фајловима
* Коришћење базе података

Бољи начин(који и даље има мане) за чување података на основу концептуалних класа је коришћење базе података, поготово релационе базе података. Да бисмо користили базу података, морамо имати приступ систему за управљање базом података тј. сервер нашег софтвера ће се понашати као клијент према систему за управљање базом података. У овом примеру, за перзистентност података је коришћена релациона база података преко приступа систему за управљање базом података „MySql“. Релационе базе података садрже одређене табеле које представљају податке и везе између њих. На основу тих веза се врши објекто релационо мапирање које омогућава нашем систему, који је развијан по принципима објектно орјентисаног програмирања, да податке из те базе корсти ефективно.

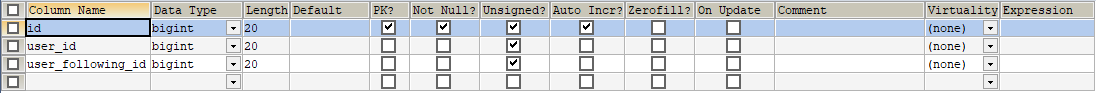
Даље следи приказ сваке од табела у нашем софтверском систему.

### 6.6.1 Табела user



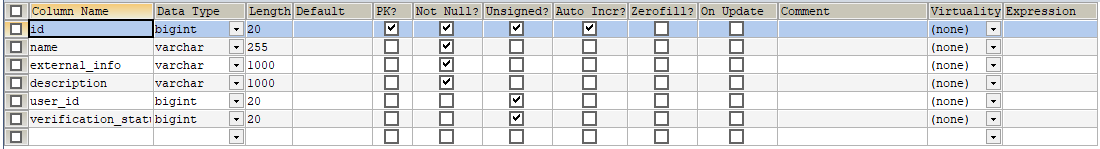
Слика Tabela user

### 6.6.2 Табела user\_following



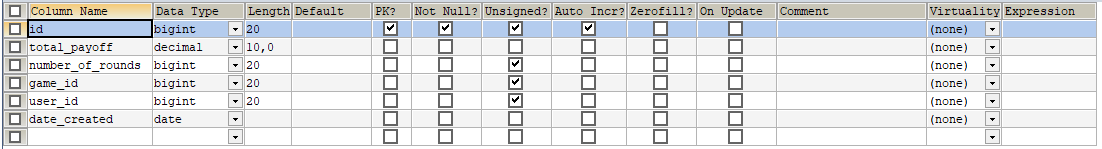
Слика Tabela user\_following

### 6.6.3 Табела game



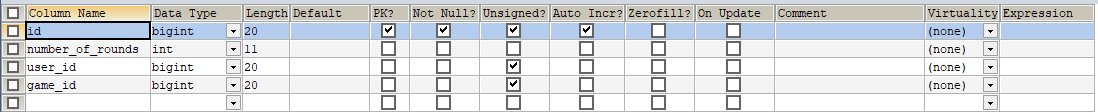
Слика Tabela game

### 6.6.4 Табела game\_score



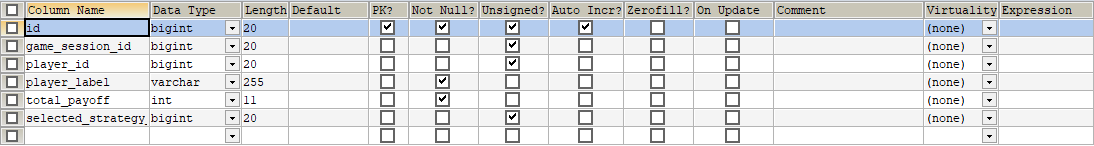
Слика Tabela game\_score

### 6.6.5 Табела game\_session



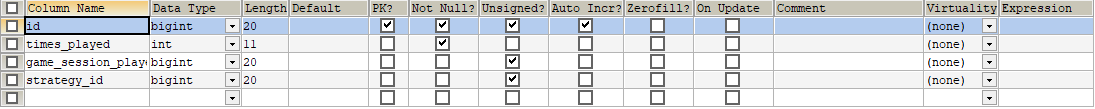
Слика Tabela game\_session

### 6.6.6 Табела game\_session\_player



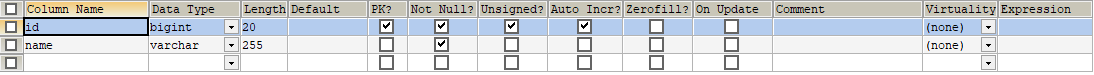
Слика Tabela game\_session\_player

### 6.6.7 Табела game\_session\_player\_strategy



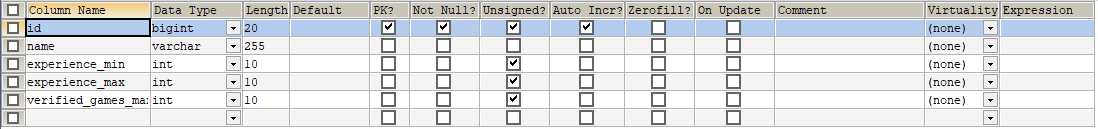
Слика Tabela game\_session\_player\_strategy

### 6.6.8 Табела verification\_status



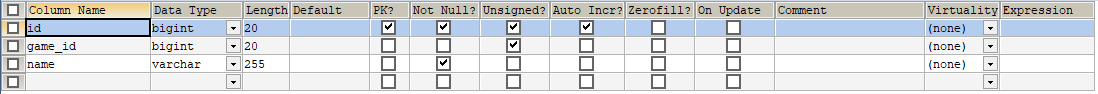
Слика Tabela verification\_status

### 6.6.9 Табела rank



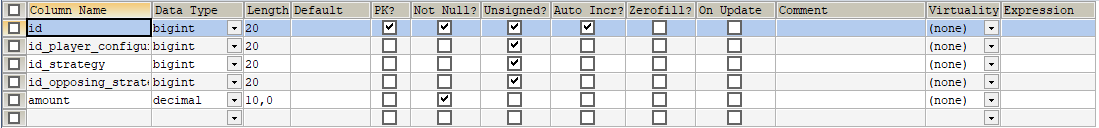
Слика Tabela rank

### 6.6.10 Табела player\_configuration



Слика Tabela player\_configuration

### 6.6.11 Табела payoff



Слика Tabela payoff

### 6.6.12 Табела strategy

### 

Слика Tabela strategy

# Закључак

У овом раду показани су неки од основних принципа поројектовања софтвера. Основни принцип пројектовања софтвера јесте тај да се софтвер мора састојати од независних компонената како промена једне не би утицала на промену друге компоненте и тако се узроковали проблеми при унапређењу и одржавању софтвера до тог тренутка када би ономе ко одржава и унапређује било јефтиније да прави нову архитекуру.

Први корак пројектовања се односио на скупљање случајева коришћења од којих сваки мора да буде независтан. Случај коришћења се односи на начин интеракције корисника са системом.

Други корак се односи на анализу и детаљно развијање случајева коришћења које открива структуру и понашање система. Ово је кључан део за следећи корак тј. само пројектовање.

При пројекотвању, коришћена је клијент-сервер, тронивојска архитектура која пројектанту даје јасан увид у то како да почне да организује структуру свог кода.

Остатак пројектовања је ствар пројектанта тј. његовок стила и преференција што значи да пре овог корака, сваки пројектант би требало да зна све о систему, а у идеалном случају, сваки корисник би интуитивно знао како да га користи.

# 9 Литература

[1] Материјали са сајта <https://www.elab.fon.bg.ac.rs/> приступљено август 2020

[2] Сервиси сајта <https://www.draw.io/> приступљено август 2020

[3] <https://stackoverflow.com/> приступљено август 2020

[4] <https://csharpindepth.com/Articles/Singleton> приступљено август 2020

1. https://en.wikipedia.org/wiki/Poka-yoke [↑](#footnote-ref-1)
2. https://en.wikipedia.org/wiki/Client%E2%80%93server\_model [↑](#footnote-ref-2)
3. https://spring.io/ [↑](#footnote-ref-3)
4. https://spring.io/projects/spring-boot [↑](#footnote-ref-4)
5. https://en.wikipedia.org/wiki/Object-relational\_mapping [↑](#footnote-ref-5)
6. https://en.wikipedia.org/wiki/List\_of\_object-relational\_mapping\_software [↑](#footnote-ref-6)
7. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller [↑](#footnote-ref-7)
8. https://en.wikipedia.org/wiki/Dependency\_injection [↑](#footnote-ref-8)
9. https://jwt.io/ [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/Document\_Object\_Model [↑](#footnote-ref-10)