

**Šolski center Novo mesto**

**Srednja elektro šola in tehniška gimnazija**

**Šegova ulica 112**

**8000 Novo mesto**

## **IZDELAVA IGER – SPLETNI KVIZ**

(maturitetna seminarska naloga)

Predmet: računalništvo

Avtor: Tomaž Černe, T4b

Mentor: dr. Albert Zorko

Novo mesto, april 2022

# POVZETEK

V seminarski nalogi so opisane teoretične osnove spletnih tehnologij, kot so HTML, CSS, JavaScript in PHP. S pomočjo teh jezikov sem ustvaril svojo lastno aplikacijo – spletni kviz. Pri načrtovanju in izgradnji aplikacije se je potrebno ravnati po natančno začrtanih korakih. Na samem začetku moramo imeti idejo in jasne cilje, kakšno aplikacijo sploh želimo imeti. Prvi segment spletne strani je naslovna stran, s pomočjo katere lahko uporabnik dostopa do vseh funkcionalnosti aplikacije. Ključni element v ozadju aplikacije je baza podatkov, načrtovana z ER modelom in realizirana z SQL ukazi. Uporabniki se s svojimi podatki prijavijo v spletno aplikacijo preko obrazcev. Podatki se preverjajo tako na strani odjemalca, kot tudi na strežniku. Beleženje podatkov o uporabniku se izvaja s pomočjo sej in piškotkov. Delovanje aplikacije je pogojeno z nenehno komunikacijo s podatkovno bazo. Programski jezik PHP omogoča povezavo z bazo podatkov, iz katere z SQL poizvedbami pridobimo določene podatke, ki jih nato prikažemo na spletni strani. Med igranjem kviza poteka dinamična komunikacija s podatkovno bazo v jeziku JavaScript s tehnologijo AJAX. Igralec kviza pri vsakem vprašanju izbira med štirimi možnimi odgovori, na voljo pa ima tudi tri oblike pomoči. Med igro se sproti prikazujejo rezultati, ki se hkrati zapisujejo v bazo, na koncu pa se izračuna uspešnost.

**Ključne besede:** aplikacija, spletna stran, podatkovna baza, kviz, igra, HTML, CSS, JavaScript, PHP, AJAX, SQL

# ABSTRACT

In this paper, some theoretical basics of web technologies such as HTML, CSS, JavaScript and PHP are described. Using these languages, I managed to create my own app - an online quiz. When designing and building the app, a precise set of steps is to be followed. At the very beginning, we need an idea and clear goals what kind of app we want to have. The first segment of the website is the front page, which allows users to access all the functionalities of the application. The key element in the background is the database, designed with an ER model and implemented with SQL commands. Users log in to the web app with their details via forms. The data is checked both client-side and server-side. User's data is tracked using sessions and cookies. The app's operation depends on constant communication with the database. The PHP programming language allows us to connect to a database, from which SQL queries are used to retrieve certain data, which is eventually displayed on the web page. During the quiz, the app is dynamically communicating with the database through JavaScript using AJAX technology. For each question, there are four possible answers that player can choose and he has also three types of help. During the game, the results are displayed continuously and recorded in the database. At the end of the game, the performance is calculated.

**Keywords:** app, website, database, quiz, game, HTML, CSS, JavaScript, PHP, AJAX, SQL

# KAZALA

## KAZALO VSEBINE

1	Uvod.....	1
2	Spletne tehnologije.....	2
2.1	HTML.....	2
2.2	CSS.....	3
2.3	JavaScript.....	4
2.4	PHP.....	5
3	Izdelava spletne aplikacije .....	6
3.1	Ideja.....	6
3.2	Zasnova.....	6
3.3	Baza podatkov.....	8
3.3.1	Načrtovanje baze z ER modelom.....	8
3.3.2	Postavitev PB z jezikom SQL.....	12
3.4	Sistem za prijavo in registracijo uporabnikov.....	14
3.4.1	Obrazci .....	15
3.4.2	Preverjaje vnesenih podatkov .....	16
3.4.3	Procesiranje podatkov, seje in piškotki .....	19
3.4.4	Odjava .....	20
3.5	Predstavitev podatkov iz podatkovne baze .....	21
3.5.1	Povezovanje programa PHP z bazo podatkov.....	21
3.5.2	Prikaz podatkov uporabniku.....	24
3.6	Delovanje in logika kviza kot igre.....	25
3.6.1	Odobritev vstopa v kviz.....	25
3.6.2	Dinamično komuniciranje s PB s tehnologijo AJAX .....	26
3.6.3	Proces izbire odgovora .....	27
3.6.4	Realizacija oblik pomoči.....	28
3.6.5	Beleženje in končni izračun rezultatov .....	29
3.7	Testiranje delovanja aplikacije.....	31
4	Zaključek .....	32
5	Stvarno kazalo.....	33
6	Zahvala.....	34

## KAZALO SLIK

Slika 1: primer HTML dokumenta .....	2
Slika 2: primer CSS kode.....	3
Slika 3: primer programa v jeziku JavaScript.....	4
Slika 4: primer kode PHP , vgrajene v HTML .....	5
Slika 5: glava naslovne strani index.php.....	7
Slika 6: izgled naslovne strani na različnih zaslonih .....	7
Slika 7: CSS koda za prilagoditev na velikost zaslona .....	8
Slika 8: začetna skica entitetnega modela.....	9
Slika 9: ER diagram .....	12
Slika 10: diagram toka podatkov za sistem prijave uporabnikov.....	14
Slika 11: obrazec za prijavo.....	15
Slika 12: HTML koda obrazca za prijavo .....	16
Slika 13: skica postopka preverjanja podatkov .....	17
Slika 14: sprotno preverjanje pri vnosu gesla .....	17
Slika 15: JavaScript funkcija za preverjanje gesla .....	18
Slika 16: del kode PHP za preverjanje prijave .....	19
Slika 17: primer PHP kode za povezavo s podatkovno bazo.....	23
Slika 18: funkcija za prikaz podatkov v HTML obliki .....	24
Slika 19: primer prikaza podatkov iz baze .....	25
Slika 20: primer vstopnega obvestila .....	26
Slika 21: del funkcije v JavaScript za prikaz novega vprašanja s tehnologijo AJAX .	27
Slika 22: prikaz vprašanja pri reševanju kviza .....	27
Slika 23: koda JavaScript za preverjanje pravilnosti izbranega odgovora .....	28
Slika 24: funkcija za pomoč polovička .....	29

# 1 UVOD

Spletne aplikacije je dandanes moč zaslediti praktično povsod. Postale so pomembne in razširjene na različnih področjih našega življenja od poslovanja pa vse do zabavnih vsebin. Posledično so tudi vse bolj kompleksne za načrtovanje, predvsem pa je ključno dati poudarek na varnost ter prijaznost aplikacije do uporabnikov. V tej seminarski nalogi nameravam izdelati aplikacijo, ki bo delovala kot igra, natančneje spletni kviz. S tem bom na nek način združil različna znanja in tehnologije s področja razvoja spletnih aplikacij ter izdelave iger. Problem oz. cilj, ki sem si ga zastavil, je izdelati aplikacijo, ki bo vizualno prijetna, funkcionalna, varna in uporabnikom zanimiva. Eden izmed glavnih ciljev je tudi, da bo aplikacija interaktivna in dinamična. Uporabnik ne bo vključen le v reševanje kvizov, pač pa bo imel tudi možnost ustvarjanja svoje vsebine, katero bodo lahko reševali ostali uporabniki. Da bo dosežena takšna funkcionalnost, je potrebno izvesti določene korake, ki zahtevajo poglobljena znanja različnih jezikov in tehnologij. To so načrtovanje in postavitve podatkovne baze (SQL), izdelava uporabniškega vmesnika oz. ogrodja in vizualizacije spletne strani (HTML, CSS), realizacija sistema za prijavo in registracijo uporabnikov (PHP, JavaScript), interakcija aplikacije s podatkovno bazo (PHP) in logična zasnova kviza kot igre (JavaScript, AJAX). Zaradi obsežnosti projekta, bom v nalogi predstavil samo nekatere najpomembnejše in najzanimivejše postopke pri razvoju spletnih aplikacij. Osredotočiti se nameravam bolj na temo podatkovnih baz in na delovanje igre, v manjšem obsegu pa na ogrodje in izgled spletne strani. Na kratko bom predstavil tudi teoretične osnove uporabljenih tehnologij, vendar pa za razumevanje naloge priporočam vsaj nekaj osnovnega predznanja s tega področja. Spletna stran bo predvidoma do januarja 2023 dostopna vsem uporabnikom na svetovnem spletu, in sicer na spletnem naslovu <https://kviz-znam.si>.

## 2 SPLETNE TEHNOLOGIJE

Pri razvoju spletne aplikacije sem uporabljal različne spletne tehnologije. Preden se lotimo načrtovanja, je zaželeno imeti nekaj teoretičnega predznanja o delovanju teh tehnologij, da bomo lahko v nadaljevanju razumeli, kako stvari delujejo v praksi.

### 2.1 HTML

HTML (angl. Hyper Text Markup Language) je jezik, ki se uporablja za izdelovanje spletnih strani. HTML ni programski jezik, temveč je opisni jezik. Predstavlja osnovo, ogrodje in zgradbo spletnega dokumenta. Deluje tako, da s pomočjo značk spletnemu brskalniku podajamo navodila, v kakšnem zaporedju in v kakšni strukturi naj naše značke prikaže. Vsak HTML dokument je zato sestavljen iz značk in besedila. Spletni strani pravimo tudi dokument, saj mora biti strukturiran po določenih pravilih, da ga brskalnik lahko pravilno prebere. Datoteko, kjer zapisujemo HTML kodo, shranimo s končnico ».html«. (1) (2)

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Prazen kviz</title>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" type="text/css" href="sporocilo.css?verzija=1">
8  </head>
9  <body>
10     <div id="polje">
11         <div id="naslov">
12             Ta kviz trenutno še nima nobenega dodanega vprašanja!
13             Prosimo, poizkusite kasneje ali pa si oglejte ostale kvize.
14         </div>
15         <a href="kvizi.php">nadaljuj</a>
16     </div>
17
18 </body>
19 </html>
```

Slika 1: primer HTML dokumenta

## 2.2 CSS

CSS (angl. Cascading Style Sheets) je slogovni jezik, ki se uporablja za prezentacijo spletnih strani. Z njim lahko določamo oblikovne lastnosti spletnih elementov, ki so definirani v HTML dokumentu. Brskalniku pove, kako naj te elemente prikaže. Določamo lahko barve, velikosti, odmike, poravnave, obrobe, pozicije in še mnogo drugih atributov. S pomočjo CSS-ja lahko izgled spletne strani prilagodimo za različne velikosti zaslona. Na ozkih zaslonih, kot jih imajo npr. pametni telefoni, je zelo pomembno, da je velikost in pozicija elementov ustrezno prilagojena. CSS shranimo s končnico ».css« ločeno in neodvisno od HTML datoteke. Posledično je vzdrževanje spletne strani enostavnejše, saj lahko enake stilske predloge uporabljamo tudi za več različnih dokumentov. (1) (3)

```
1  *{
2  |   font-family: Arial, Helvetica, sans-serif;
3  |   box-sizing: border-box;
4  | }
5  #naslovna_vrstica{
6  |   position: fixed;
7  |   top: 0;
8  |   left: 0;
9  |   width: 100%;
10 |   height: 80px;
11 |   background-color: ■ rgb(40, 97, 218);
12 | }
13 #meni{
14 |   position: fixed;
15 |   top: 0;
16 |   left: 0;
17 |   width: 350px;
18 | }
```

Slika 2: primer CSS kode



## 2.3 JAVASCRIPT

JavaScript je objektni skriptni programski jezik, ki ga je leta 1995 razvilo podjetje Netscape. Sprva se je imenoval LiveScript, iz tržnih razlogov pa so ga kmalu preimenovali v JavaScript. Izvaja se na uporabnikovem računalniku, kar prinaša mnoge prednosti pri izdelavi spletnih strani. JavaScript lahko sodeluje s HTML kodo in s tem poživi stran z dinamičnim izvajanjem, tako npr. spletne strani ni potrebno ponovno osveževati, če želimo spremeniti barvo ozadja, spremeniti temo, velikost pisave ipd. Uporablja se za ustvarjanje dinamičnih spletnih strani. Program se vgradi ali pa vključi v HTML, da opravlja naloge, ki niso mogoče samo s statično stranjo. Na primer odpiranje novih oken, preverjanje pravilnosti vnesenih podatkov, enostavni izračuni... Jezik je bil sicer razvit neodvisno od programskega jezika Java, vendar si z njim deli številne lastnosti in strukture. Datoteko shranimo s končnico ».js«. (4)

```
1  function potrditev(){
2      let dovoli = true;
3      let tab = document.getElementsByClassName("vnos");
4      for(let i=0; i<tab.length; i++){
5          let input = tab[i].firstElementChild;
6          preveri(input);
7          if(input.nextElementSibling.innerHTML != ""){
8              dovoli = false;
9          }
10     }
11     return dovoli;
12 }
13 function napaka(input, txt){
14     let x = input.nextElementSibling;
15     x.innerHTML = txt;
16 }
17 function pokaziGeslo(box){
18     let input = box.parentElement.parentElement.firstElementChild;
19     if(box.checked == true){
20         input.type = "text";
21     }
22     else{
23         input.type = "password";
24     }
25 }
```

Slika 3: primer programa v jeziku JavaScript

## 2.4 PHP

PHP (angl. PHP Hypertext Preprocessor) je programski jezik, ki spletnim razvijalcem omogoča ustvarjanje dinamičnih spletnih aplikacij, za katere je značilna konstantna izmenjava podatkov med spletno stranjo in podatkovno bazo. Skladnja jezika PHP je podobna drugim programskim jezikom, kot sta npr. Perl in Java. Programsko kodo v PHP-ju lahko vgradimo neposredno v kodo HTML. Izvaja se na strani strežnika, ki podatke obdela in jih vrne odjemalcu (uporabniku) v obliki HTML kode. Na strani odjemalca se koda procesira običajno, kot bi šlo za statičen HTML dokument. PHP uporabljamo predvsem takrat, ko želimo na spletni strani prikazati podatke, ki jih bomo ob uporabnikovi zahtevi pridobili iz podatkovne baze, ki se nahaja na strežniku oz. ko želimo, da uporabniki preko spletnih obrazcev vnesejo podatke, ki jih nato shranimo v bazo. Datoteke imajo končnico ».php«. (5) (6)

```
66      <div id="up_meni" >
67          <button id="gumb_up_meni">
68              <i class="fa fa-user-circle"></i>
69              <span id="napis"> <?php echo $napis?> </span>
70              <i class="fa fa-caret-down" id="puscica"></i>
71          </button>
72          <div id="up_spustni_seznam">
73              <?php
74                  for($i=0; $i<count($povezava); $i++){
75                      echo $povezava[$i];
76                  }
77              ?>
78          </div>
79      </div>
```

Slika 4: primer kode PHP, vgrajene v HTML

## 3 IZDELAVA SPLETNE APLIKACIJE

V tem segmentu seminarske naloge bom opisal postopek, skozi katerega sem šel, da sem lahko naredil svojo spletno igro. Predstavil bom tudi najpomembnejše teoretične osnove ter razložil nekatere zanimivejše izseke iz programske kode. Celoten projekt z vsemi datotekami je na voljo preko povezav, ki jih najdete v prilogah.

### 3.1 IDEJA

Vsak projekt se začne z idejo. Pred samim načrtovanjem aplikacije si je potrebno zastaviti jasne cilje in pričakovanja, kako bo aplikacija delovala. Sam sem si kot osnovno idejo izbral kviz, ki ga rešujejo uporabniki. Dobil sem tudi idejo, da bo igra še bolj zanimiva, če bo imel vsak igralec tudi možnost ustvarjanja svojih lastnih kvizov. Pravila kviza so sledeča: kviz vsebuje vprašanja, katerih število ni vnaprej določeno, torej lahko ustvarjalec v kviz doda poljubno število vprašanj. Pri vsakem vprašanju so možni natanko štirje odgovori, od katerih je pravilen le eden. Popestritev v kvizu predstavljajo trije »zasilni izhodi« oz. oblike pomoči, ki jih igralec lahko enkrat uporabi tekom reševanja kviza. Polovička je pomoč, ki naključno izloči dva napačna odgovora. Glas ljudstva igralcu omogoči vpogled v deleže posameznih odgovorov, ki so jih izbrali ostali igralci. Tretja pomoč se imenuje Joker; igralec izbere dva odgovora, Joker pa mu nato izloči enega izmed teh dveh, ki je zagotovo napačen. Pri vsakem kvizu se beležijo rezultati, ki so na voljo avtorju kviza. Igralec lahko spremlja tudi vse svoje rezultate, ki jih je dosegel pri različnih kvizih.

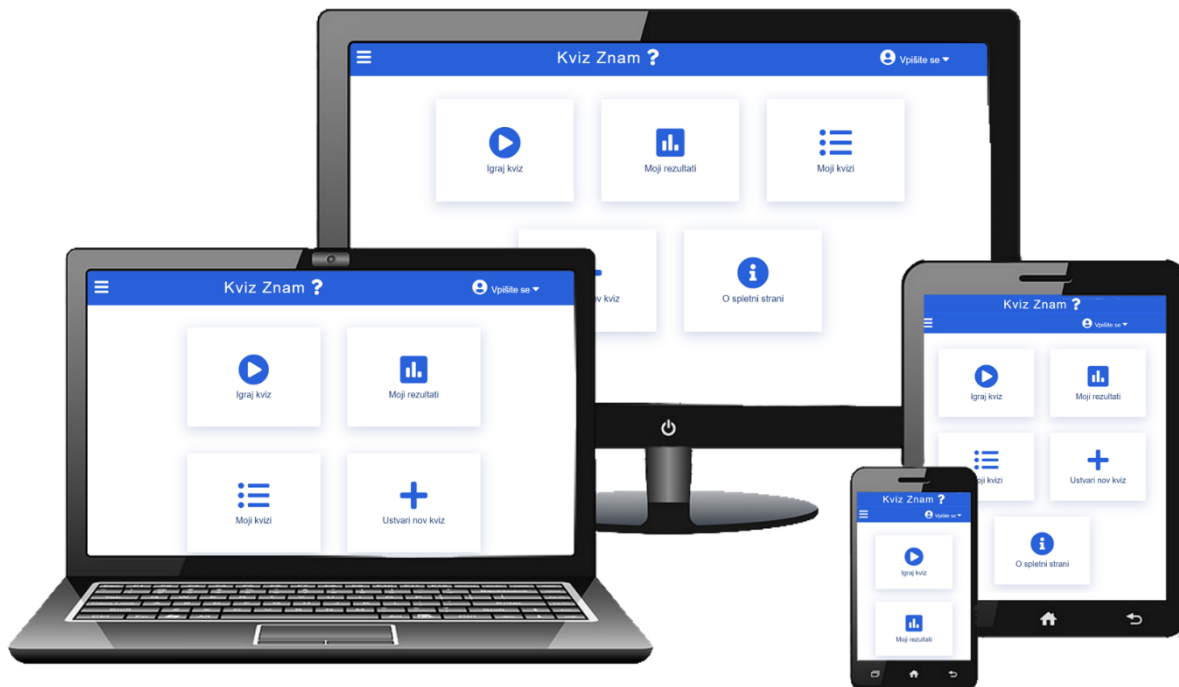
### 3.2 ZASNOVA

Na samem začetku potrebujemo naslovno stran. Vsak uporabnik, ki bo obiskal naše spletno mesto, se bo najprej znašel na naslovni strani, od koder se bo nato preusmeril na poljubne podstrani oz. funkcije, ki jih aplikacija ponuja. Za naslovno stran je pomembno, da je enostavna, pregledna in vizualno privlačna. Shranimo jo pod imenom »index«. V glavi dokumenta običajno dodamo t. i. meta oznake, to so podatki o spletni strani, ki so dosegljivi brskalniku, kot npr. opis, ključne besede, avtor... (glej Slika 5)

```
36 <!DOCTYPE html>
37 <html>
38 <head>
39     <title>Kviz Znam</title>
40     <meta charset="UTF-8">
41     <meta name="viewport" content="width=device-width, initial-scale=1.0">
42     <meta name="description" content="Zanimiva spletna igra, v kateri lahko rešujete
43     | najrazličnejše kvize, ustvarjate svoje, spremljate rezultate...">
44     <meta name="keywords" content="kviz, znam, kviz znam, igra" >
45     <meta name="author" content="Tomaž Černe">
46     <link rel="stylesheet" type="text/css" href="naslovnica.css?verzija=9">
47     <link rel="stylesheet" href="fontawesome-free-5.15.4-web/css/all.css">
48
49 </head>
```

Slika 5: glava naslovne strani index.php

Moja naslovna stran ima zgoraj naslovno vrstico, v kateri sta dva spustna menija. Na levem meniju se nahajajo povezave na določene podstrani. Na desni strani se nahaja uporabniški meni, preko katerega se lahko uporabnik prijavi v sistem in spremlja svoje uporabniške podatke. Zelo pomembno je, da izgled naslovnice prilagodimo različnim napravam, iz katerih bodo uporabniki dostopali na našo spletno stran (Slika 6).



Slika 6: izgled naslovne strani na različnih zaslonih

Prilagoditve glede na velikost zaslona sem določil s posebnim delom CSS-kode, v katerem sem definiral, do katere širine zaslona se bo izvajala prilagoditev. Če bo velikost zaslona manjša od te meje, se bodo dodatno upoštevale tudi lastnosti v tem segmentu kode. Običajno gre za spreminjanje velikosti, pozicije ter odmika določenih elementov (Slika 7).

```
135  @media only screen and (max-width: 700px) {
136      #naslovna_vrstica{
137          height: 100px;
138      }
139      #meni, #up_meni{
140          top: 50px;
141      }
142      #gumb_up_meni{
143          padding: 5px 20px;
144          font-size: 25px;
145          min-height: 50px;
146      }
```

Slika 7: CSS koda za prilagoditev na velikost zaslona

### 3.3 BAZA PODATKOV

Podatkovna baza je zbirka med seboj pomensko povezanih podatkov, ki so shranjeni v računalniškem sistemu, dostop do njih je centraliziran in omogočen s pomočjo sistema za upravljanje podatkovnih baz (SUPB). Baza podatkov je eden ključnih gradnikov vsake spletne aplikacije. Brez nje uporabniki ne bi imeli možnosti trajnega shranjevanja podatkov. V moji aplikaciji se bodo v bazo shranjevali vsi podatki o uporabnikih, njihovi kvizi, vprašanja, rezultati itd. (7)

#### 3.3.1 NAČRTOVANJE BAZE Z ER MODELOM

Načrtovanja podatkovne baze se je možno lotiti na več različnih načinov. Obstajajo namreč najrazličnejši postopki, s katerimi dosežemo, da so podatki predstavljeni v obliki, ki je primerna podlaga za izgradnjo podatkovne baze. Eden izmed najbolj uveljavljenih je postopek oblikovanja entitetnega modela, katerega končni rezultat je ER-diagram. V nadaljevanju bodo predstavljene vse faze tega postopka.

### 3.3.1.1 OPREDELITEV ENTITET

Entiteta je objekt iz realnega sveta, o katerem zbiramo, obdelujemo in hranimo podatke in informacije. Lahko ga enolično identificiramo, izoliramo od okolice in opišemo. Podobne entitete se nahajajo v skupni množici, katero predstavlja entitetni tip. (8)

Za podatkovni model mojega spletnega kviza sem določil naslednje tipe entitet: UPORABNIKI, KVIZI, VPRASANJA, ODGOVORI in REZULTATI

### 3.3.1.2 DOLOČITEV PRIMARNIH KLJUČEV

Primarni ključ je izbrani atribut ali njihova kombinacija, ki enolično identificira natanko en primerek entitete. Z njim si pomagamo pri iskanju podatkov in povezovanju različnih relacij. (9)

Vsem prej določenim entitetam je potrebno določiti pripadajoče primarne ključe, kar je najlažje storiti tako, da vsaka entiteta prejme svojo šifro oz. id:

UPORABNIKI(id\_uporabnika\*)

KVIZI(id\_kviza\*)

VPRASANJA(id\_vprasanja\*)

ODGOVORI(id\_odgovora\*)

REZULTATI(id\_rezultata\*)

### 3.3.1.3 ZAČETNA SKICA ENTITETNEGA MODELA

Ko smo določili entitete in primarne ključe, pride na vrsto prva vizualna predstavitev našega podatkovnega modela. V začetni skici (Slika 8) je potrebno opredeliti vse povezave med izbranimi entitetnimi tipi.



Slika 8: začetna skica entitetnega modela

Povezave med entitetami morajo biti smiselne in logično dobro opredeljene. V mojem primeru ima vsak uporabnik lahko shranjene svoje kvize. Hkrati se zanj beležijo tudi rezultati, doseženi pri vseh kvizih, ki jih je igral. Zabeleženi so tudi vsi odgovori, ki jih je uporabnik izbral skozi reševanje kvizov. Vsak kviz ima svoja vprašanja in zabeležene rezultate. Pri vsakem vprašanju se shranjujejo tudi izbrani odgovori (s strani vseh uporabnikov).

#### 3.3.1.4 OPREDELITEV ATRIBUTOV

Atribut opisuje lastnost neke entitete ali razmerja med entitetami. Vrednost atributa je podatek določenega podatkovnega tipa (niz, število, ...) in ga je možno prikazati v neki vidni ali slišni obliki (zaslonski prikaz, natisnjena oblika, zvok ...). Atribut, ki se lahko pojavi tudi brez vrednosti (NULL), se imenuje opcijski atribut. Atributom, ki imajo eno možno vrednost, pravimo enovrednostni atributi, tisti, ki lahko zavzamejo več vrednosti, pa so večvrednostni atributi. (10)

Vsaki entiteti bo potrebno poleg primarnega ključa dodeliti tudi ostale attribute, ki bodo ustrezali posameznemu tipu entitete. V mojem primeru sem določil naslednje attribute:

UPORABNIKI(id\_uporabnika\*, ime, priimek, uporabnisko\_ime, geslo, datum\_registracije)

KVIZI(id\_kviza\*, id\_uporabnika, naslov\_kviza, opis, geslo\_kviza, datum\_kviza, enkrat)

VPRASANJA(id\_vprasanja\*, id\_kviza, vprasanje, odgovor\_A, odgovor\_B, odgovor\_C, odgovor\_D, pravilen, datum\_vprasanja)

ODGOVORI(id\_odgovora\*, id\_uporabnika, id\_vprasanja, odgovor, datum\_odgovora)

REZULTATI(id\_rezultata\*, id\_uporabnika, id\_kviza, st\_pravilnih, st\_vprasanj, datum\_rezultata)

### 3.3.1.5 ZAPIS RELACIJSKE SCHEME

Entitete skupaj s pripadajočimi atributi povežemo v relacijsko shemo. Vsakemu atributu določimo podatkovni tip (niz – A, število – N, datum – D). Ker v podatkovni bazi nimamo neomejenega prostora, je dobro, da določimo tudi omejitev velikosti podatka za posamezni atribut. Relacija se z ostalimi povezuje tako, da ji kot atribut dodamo primarni ključ relacije, s katero jo želimo povezati. Ta atribut se imenuje tuji ključ. (11)

UPORABNIKI(id\_uporabnika\*: N, ime: A(30), priimek: A(30), uporabnisko\_ime: A(30), geslo: A(32), datum\_registracije: D)

KVIZI(id\_kviza\*: N, id\_uporabnika: N → UPORABNIKI, naslov\_kviza: A(50), opis: A(300), geslo\_kviza: A(32), datum\_kviza: D, enkrat: N(1))

VPRASANJA(id\_vprasanja\*: N, id\_kviza: N → KVIZI, vprasanje: A(200), odgovor\_A: A(50), odgovor\_B: A(50), odgovor\_C: A(50), odgovor\_D: A(50), pravilen: A(1), datum\_vprasanja: D)

ODGOVORI(id\_odgovora\*: N, id\_uporabnika: N → UPORABNIKI, id\_vprasanja: N → VPRASANJA, odgovor: A(1), datum\_odgovora: D)

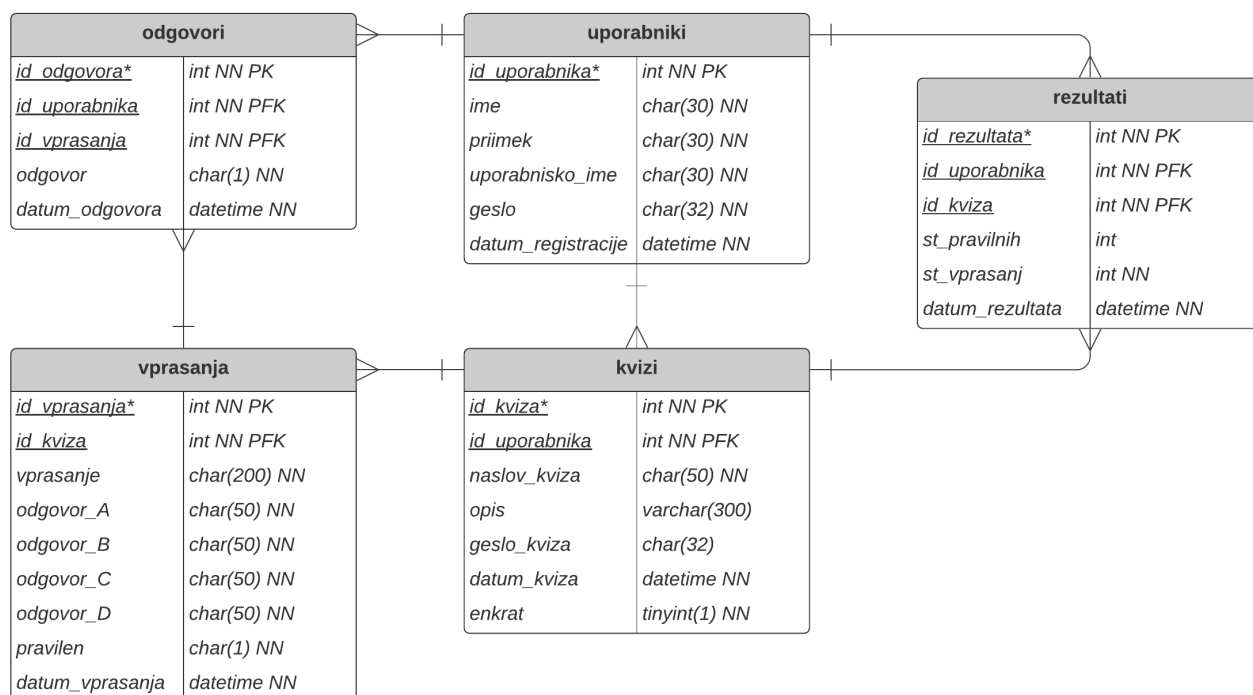
REZULTATI(id\_rezultata\*: N, id\_uporabnika: N → UPORABNIKI, id\_kviza: N → KVIZI, st\_pravilnih: N, st\_vprasanj: N, datum\_rezultata: D)

### 3.3.1.6 IZDELAVA ER-DIAGRAMA

S pomočjo ER (angl. Entity Relationship) diagrama grafično ponazorimo vse relacije iz relacijske sheme in povezave med njimi. Diagram, pridobljen skozi vse faze načrtovanja, na koncu v grobem predstavlja logično zasnovo in strukturo podatkovne baze. (12)

Za izdelavo ER-diagrama obstaja več standardov. Na mojem primeru (Slika 9) lahko vidite enega izmed možnih standardov za načrtovanje ER-diagrama.





Slika 9: ER diagram

### 3.3.2 POSTAVITEV PB Z JEZIKOM SQL

SQL (angl. Structured Query Language) je standardiziran jezik, namenjen upravljanju z relacijskimi podatkovnimi bazami ter izvajanju različnih operacij nad podatki, ki se v bazi nahajajo. Uporabljajo ga tako administratorji podatkovnih baz, kot tudi programerji in razvijalci, podatkovni analitiki ipd. S pomočjo SQL-a lahko ustvarimo podatkovno bazo, ustvarimo in spreminjamo tabele, vnašamo in posodabljammo podatke ter dostopamo do določene množice iskanih podatkov v podatkovni bazi.

(13)

#### 3.3.2.1 USTVARJANJE NOVE PODATKOVNE BAZE

Podatkovno bazo je najprej potrebno ustvariti. To naredimo precej enostavno z ukazom CREATE DATABASE, ki mu dodamo še ime naše podatkovne baze.

V mojem primeru sem bazo ustvaril takole:

```
CREATE DATABASE kviznam_baza;
```

### 3.3.2.2 DODAJANJE TABEL

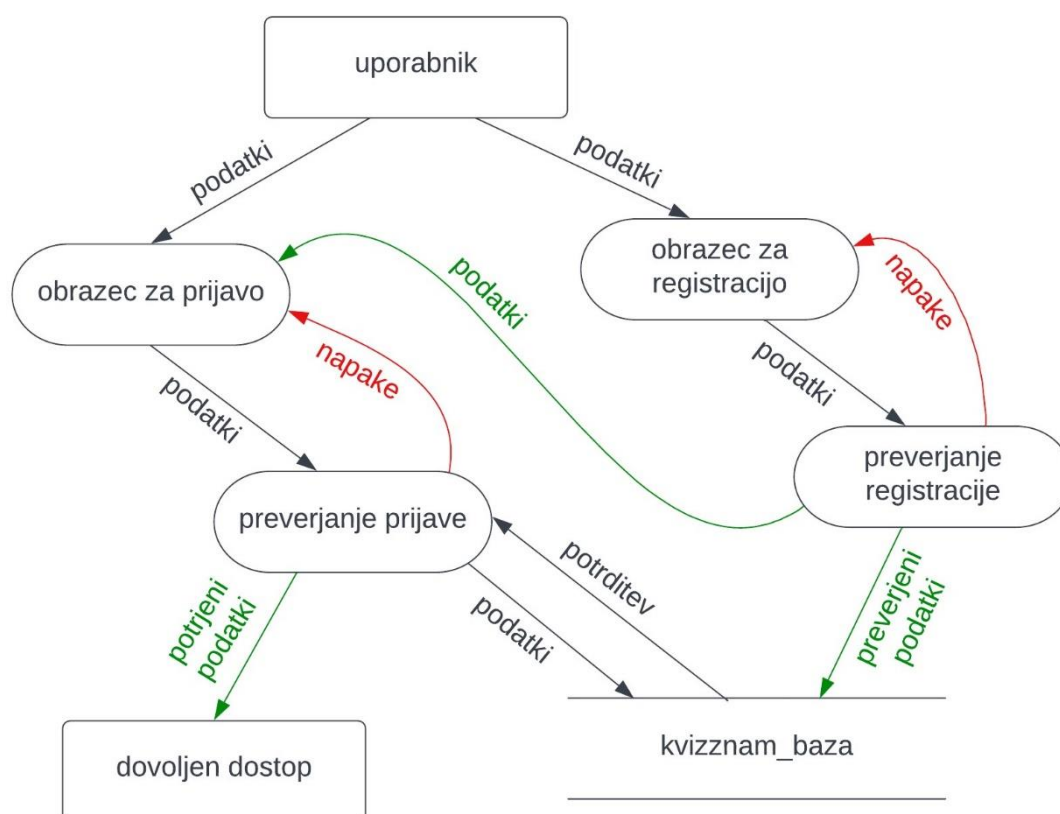
Ko smo ustvarili podatkovno bazo, je potrebno definirati tabele oz. relacije med podatki. Pri tem se strogo ravnamo po predhodno narejenem načrtu (ER diagramu). Spodaj sta navedena primera, kako sem ustvaril tabeli »uporabniki« in »kvizi«. SQL ukaze za vse ostale tabele pa lahko najdete v prilogah.

```
CREATE TABLE uporabniki(  
  id_uporabnika int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  ime char(30) NOT NULL,  
  priimek char(30) NOT NULL,  
  uporabnisko_ime char(30) NOT NULL,  
  geslo char(32) NOT NULL,  
  datum_registracije datetime NOT NULL);
```

```
CREATE TABLE kvizi(  
  id_kviza int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  id_uporabnika int NOT NULL,  
  naslov_kviza char(50) NOT NULL,  
  opis varchar(300),  
  geslo_kviza char(32),  
  datum_kviza datetime NOT NULL,  
  enkrat tinyint(1) NOT NULL,  
  FOREIGN KEY (id_uporabnika)  
  REFERENCES uporabniki(id_uporabnika));
```

## 3.4 SISTEM ZA PRIJAVO IN REGISTRACIJO UPORABNIKOV

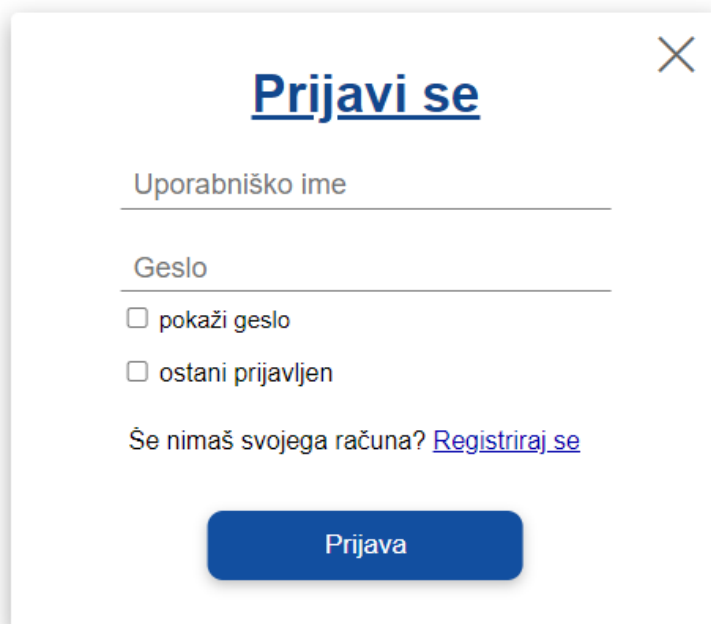
Ker je zasnova moje aplikacije taka, da želimo vsakemu uporabniku omogočiti ustvarjanje lastnih kvizov in vpogled v svoje rezultate, mora vsak uporabnik obvezno imeti svoj uporabniški račun, s katerim se prijavi v aplikacijo. Brez predhodne prijave aplikacija iz varnostnih razlogov neidentificiranemu uporabniku ne sme dovoliti dostopa do podatkov, temveč ga preusmeri na obrazec za prijavo. Če uporabnik še nima svojega računa, se lahko vpiše oziroma registrira v sistem preko obrazca za registracijo. Ob potrditvi obrazca mora aplikacija preveriti, ali so vneseni podatki kompletni in v dovoljeni obliki. Šele nato se podatki trajno shranijo v podatkovno bazo, pri tem procesu pa je izjemnega pomena varnost oz. zaščita občutljivih osebnih podatkov pred morebitnimi napadi, zlorabami, vdori... Celoten postopek je grafično prikazan v obliki diagrama toka podatkov (Slika 10).



Slika 10: diagram toka podatkov za sistem prijave uporabnikov

### 3.4.1 OBRAZCI

Uporabniki bodo pri procesu registracije in kasneje prijave vnašali svoje podatke v temu namenjen prostor na spletni strani, ki ga imenujemo obrazec (angl. form). Obrazec definiramo v jeziku HTML, pri tem pa moramo jasno določiti, kam pošiljamo podatke (action), kateri podatek v določenem vnosnem polju zahtevamo od uporabnika in v kakšni obliki se mora ta podatek nahajati. Uporabljajo se različne oblike polj, kot so npr. polja za vnos besedila, vnos gesel, izbiranje, označevanje itd. Ko uporabnik vnese vse podatke ter pritisne na gumb za potrditev (angl. submit), se podatki pošljejo na strežnik, kjer jih nato s pomočjo programa (PHP) obdelamo. Pri pošiljanju podatkov imamo na voljo dve metodi: »get« ali »post«. Pri metodi get se podatki pripnejo k URL naslovu, kamor jih pošiljamo. Če uporabimo metodo post pa se vrednosti pošljejo preko HTTP glave dokumenta. V mojem primeru, ko pošiljamo občutljive podatke, sem raje uporabil metodo post. Izgled obrazca lahko seveda prilagajamo z dodatno CSS kodo. Pri oblikovanju moramo paziti, da je obrazec karseda enostaven in razumljiv, sicer imajo lahko nekateri uporabniki med vnašanjem podatkov težave. (1) (14)



The image shows a login form titled "Prijavi se" (Log in) in blue text. The form is enclosed in a white box with a grey border and a close button (X) in the top right corner. It contains two input fields: "Uporabniško ime" (Username) and "Geslo" (Password). Below the password field, there are two checkboxes: "pokaži geslo" (show password) and "ostani prijavljen" (stay logged in). At the bottom, there is a link "Še nimaš svojega računa? Registriraj se" (Don't have your account? Register) and a blue button labeled "Prijava" (Login).

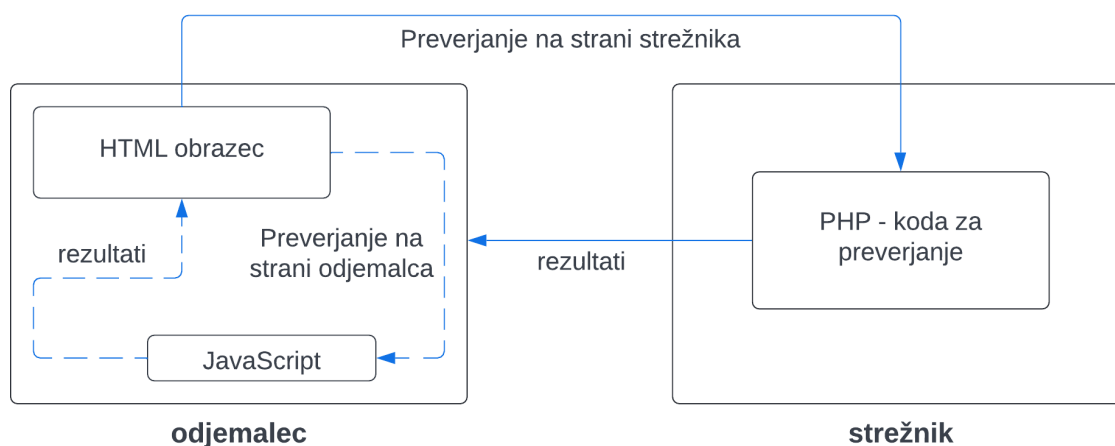
Slika 11: obrazec za prijavo

```
<form id="prijava" action="<?php echo $action ?>" method="post" onsubmit="return potrditev()">
  <div id="naslov">
    Prijavi se
  </div>
  <div class="vnos">
    <input type="text" id="uporabnisko_ime" name="uporabnisko_ime" placeholder="Uporabniško ime"
      | value="<?php echo vrni("uporabnisko_ime"); ?>" onkeyup="preveri(this)">
    <div class="napaka"> <?php echo napaka("uporabnisko_ime"); ?> </div>
  </div>
  <div class="vnos">
    <input type="password" id="geslo" name="geslo" placeholder="Geslo"
      | value="<?php echo vrni("geslo"); ?>" onkeyup="preveri(this)">
    <div class="napaka"> <?php echo napaka("geslo"); ?> </div>
    <div class="pokazi">
      <input type="checkbox" id="pg1" onclick="pokaziGeslo(this)">
      <label for="pg1"> pokaži geslo </label>
    </div>
  </div>
  <div id="ostani">
    <input type="checkbox" id="ostani_prijavljen" name="ostani_prijavljen" value="true">
    <label for="ostani_prijavljen"> ostani prijavljen</label>
  </div>
  <div id="preusmeritev">
    Še nimaš svojega računa? <a href="<?php echo $reg_url ?>"> Registriraj se </a>
  </div>
  <div id="splosna_napaka">
    <?php echo napaka("splosna");?>
  </div>
  <button id="submit" type="submit"> Prijava </button>
</form>
```

Slika 12: HTML koda obrazca za prijavo

### 3.4.2 PREVERJAJE VNESENIH PODATKOV

Preden gredo podatki v nadaljnjo obdelavo, jih je potrebno obvezno v celoti preveriti, ali so ustrezni. Nikoli se ne smemo zanašati le na to, da bo uporabnik vse podatke vnesel pravilno in v celoti. Uporabniki se lahko pogosto zmotijo, še bolj problematični pa so tisti, ki želijo namenoma zaobiti sistem preverjanja. V podatkovni bazi ne želimo imeti nepopolnih podatkov, saj lahko kasneje v fazi predstavitve podatkov, ki niso v taki obliki, kot bi pričakovali, pride do resnih težav pri delovanju aplikacije. Če pri preverjanju odkrijemo napake, jih je potrebno na razumljiv način sporočiti uporabniku ter mu omogočiti, da določene podatke vnese ponovno. Pri zasnovi moje aplikacije sem dal velik poudarek varnosti, zato poteka preverjanje v dveh fazah, na strani odjemalca in strežnika. (15)



Slika 13: skica postopka preverjanja podatkov

### 3.4.2.1 PREVERJANJE NA STRANI ODJEMALCA

Prva faza preverjanja poteka na strani uporabnika, ki v obrazec vnaša podatke. Realizirana je v jeziku JavaScript in se izvaja direktno na uporabnikovem računalniku. Prednost takega načina preverjanja je, da lahko odkrijemo morebitne napake še preden uporabnik obrazec potrdi in se podatki pošljejo na strežnik, posledično v takih primerih ne rabimo dodatno obremenjevati strežnika. Možno je tudi sprotno preverjanje, ki se izvaja že med samim vnašanjem podatkov in uporabnika lahko usmerja ter mu pomaga pravilno izpolniti obrazec (Slika 14). Glavna slabost preverjanja na strani odjemalca je, da lahko uporabnik v nastavitvah brskalnika onemogoči izvajanje jezika JavaScript, s tem zaobide preverjanje in brez težav nadaljuje s potrditvijo obrazca. Iz tega razloga lahko ta način preverjanja uporabimo le v kombinaciji s preverjanjem preko strežnika. (15)

Tukaj\_Ni\_številk!

Geslo mora vsebovati:

✓ male črke

✓ velike črke

✗ številke

✓ znake

☒ pokaži geslo

Slika 14: sprotno preverjanje pri vnosu gesla

```
function geslo(input, vnos){
  let male = false, velike = false, stevilke = false, znaki = false;
  for(let i=0; i<vnos.length; i++){
    let x = vnos.charAt(i);
    if(x.toUpperCase() != x.toLowerCase()){
      if(x == x.toUpperCase()){ velike = true; }
      else{ male = true; }
    }
    else if(x >='0' && x <='9'){ stevilke = true; }
    else{ znaki = true; }
  }
  if(!(male && velike && stevilke && znaki)){
    function vsebuje(pogoj){
      if(pogoj){ return '<span style="color: #00c853;"> &#x2713'; }
      return '<span> &#x2715';
    }
    let sp = "Geslo mora vsebovati:<br>";
    sp += vsebuje(male) + " male črke</span><br>";
    sp += vsebuje(velike) + " velike črke</span><br>";
    sp += vsebuje(stevilke) + " številke</span><br>";
    sp += vsebuje(znaki) + " znake</span>";
    napaka(input, sp);
    return false
  }
  return true;
}
```

Slika 15: JavaScript funkcija za preverjanje gesla

### 3.4.2.2 PREVERJANJE NA STREŽNIKU

Iz že prej omenjenih varnostnih razlogov je preverjanje na strani strežnika nujno. Uporabnik nima nikakršnega vpliva na delovanje programov, ki tečejo na strežniku, zato se ta način preverjanja zagotovo izvede. V primerjavi s preverjanjem na strani odjemalca poteka preverjanje na strežniku sicer počasneje, vendar učinkovito. S predhodnim preverjanjem na strani odjemalca v večini primerov že zagotovimo točne podatke, ki jih nato na strežniku le še potrdimo in nato nadaljujemo z nadaljnjimi postopki. S tem poskrbimo, da je strežnik čim bolj razbremenjen in lahko hitreje opravlja vse svoje naloge. Če uporabnik na svoji strani izklopi preverjanje in pošlje pomanjkljive podatke, bomo na strežniku zaznali morebitne napake in uporabnika o njih tudi obvestili. Programi na strežniku so lahko napisani v različnih programskih jezikih, v mojem primeru sem preverjanje realiziral v jeziku PHP (Slika 16). (15)

```
$podatki["uporabnisko_ime"] = trim($_POST["uporabnisko_ime"]);
$podatki["geslo"] = trim($_POST["geslo"]);

foreach($podatki as $input => $vnos){
    preveri($input, $vnos);
}

function preveri($input, $vnos){
    if(empty($vnos)){
        $_SESSION["pr_napake"][$input] = "Obvezno izpolnite to polje";
    }
}
```

Slika 16: del kode PHP za preverjanje prijave

### 3.4.3 PROCESIRANJE PODATKOV, SEJE IN PIŠKOTKI

Podatki, ki jih je uporabnik poslal preko obrazca, se pošljejo na strežnik. V programu PHP, ki teče na strežniku so ti podatki dosegljivi preko spremenljivk `$_GET[ime_spremenljivke]` ali `$_POST[ime_spremenljivke]`, odvisno od metode pošiljanja obrazca. Če na primer uporabljamo metodo POST in imamo v obrazcu vnosno polje `<input type="text" name="ime">`, bomo podatek, ki ga je uporabnik vnesel v to polje, pridobili preko spremenljivke `$_POST["ime"]`.

Pri registraciji uporabnika se podatki, potem ko smo jih dokončno preverili, zapišejo v podatkovno bazo. Več o povezovanju med kodo PHP in bazo podatkov boste izvedeli v poglavju 3.5. Ob uspešni registraciji pridobi vsak uporabnik svojo identifikacijsko številko (`ID_uporabnika`), ki je zapisana v bazi poleg ostalih njegovih podatkov. Nato se preusmeri na prijavo. Pri prijavi, ob predpostavki da je pravilno vnesel svoje uporabniško ime in geslo, se iz podatkovne baze pridobi njegov ID in se shrani v obliki seje. V času trajanja seje je uporabnik prijavljen na spletno stran.

Seja (angl. Session) lahko za čas obiska spletne strani hrani na strežniku poljubne podatke o uporabniku, na uporabnikovi strani pa se hrani samo identifikator seje. Navadno v sejah shranjujemo le manjše količine podatkov (npr. identifikatorje, kratke nize...). Seje imajo s strani strežnika le določen čas veljavnosti, največkrat pa jih uporabljamo za prijavo, nakupovalne košarice... V PHP uporabo seje najavimo z ukazom `session_start()`. Podatke seje beremo in pišemo preko spremenljivke `$_SESSION`. Seje uničimo z ukazom `session_destroy()`, posamezen ključ seje pa uničimo z ukazom `unset()`. (16)



V mojem primeru sem ob uspešni prijavi uporabnika ustvaril sejo, ki vsebuje le njegov ID, vsi ostali podatki o uporabniku pa so preko tega ID-ja dostopni v podatkovni bazi: `$_SESSION["id_uporabnika"] = $id_uporabnika;`

Pri prijavi ima uporabnik tudi možnost »ostani prijavljen«, kar pomeni da bi želel biti prijavljen dlje časa kot traja seja (ne le za čas enega obiska spletne strani). To mu seveda lahko omogočimo s pomočjo piškotkov. Piškotek (angl. cookie) je majhna datoteka, ki jo brskalnik shrani na uporabnikov računalnik. Piškotke uporabljamo za dolgoročno shranjevanje podatkov o uporabniku (dlje časa kot seja). V PHP piškotek ustvarimo s funkcijo `setcookie(ime, vrednost, veljavnost, pot, domena, varnost);` Vrednost piškotka dobimo z uporabo `$_COOKIE["ime"]`. Piškotke uničimo s `setcookie()`, kjer v parameter veljavnosti vpišemo čas, starejši od trenutnega. (17)

V mojem primeru sem ustvaril piškotek z veljavnostjo 60 dni, če je uporabnik v obrazcu označil, da želi ostati prijavljen:

```
if(isset($_POST["ostani_prijavljen"])){  
    setcookie("id_uporabnika", $id_uporabnika, time() + (86400  
        * 60), "/");  
}
```

### 3.4.4 ODJAVA

Ko se uporabnik s spletne strani odjavi, je potrebno poskrbeti, da se izbrišejo vse spremenljivke znotraj seje in tudi vsi piškotki. Nato uporabnika preusmerimo nazaj na naslovno stran. V mojem primeru je program za odjavo sledeč:

```
session_start();  
  
setcookie("id_uporabnika", $_SESSION["id_uporabnika"], time()  
    - 3600);  
  
session_destroy();  
  
header("location: ../");  
  
exit();
```

## 3.5 PREDSTAVITEV PODATKOV IZ PODATKOVNE BAZE

Uporabnik ima v aplikaciji pregled nad svojimi kvizi, vprašanji in rezultati. Podatki so za vsakega uporabnika različni in se seveda lahko sproti spreminjajo, npr. uporabnik dodaja nove kvize ali pa obstoječim kvizom dodaja nova vprašanja. Možno je tudi brisanje vprašanj ter kvizov. Vsi ti podatki se nahajajo v podatkovni bazi, zato se je potrebno z bazo najprej povezati, nato iz nje pridobiti ustrezne podatke in jih uporabniku predstaviti v čim bolj pregledni ter grafično privlačni obliki.

### 3.5.1 POVEZOVANJE PROGRAMA PHP Z BAZO PODATKOV

#### 3.5.1.1 VZPOSTAVITEV POVEZAVE

Najprej je potrebno ustvariti povezavo do baze podatkov. Shranili jo bomo v obliki nove spremenljivke. Pri definiranju povezave moramo navesti štiri podatke, in sicer ime strežnika, uporabniško ime, geslo za dostop do podatkovne baze ter ime baze.

Primer: `$povezava = new mysqli("localhost", $up_ime, $geslo, "kviznam_baza");`

#### 3.5.1.2 PREVERJANJE POVEZAVE

Ko smo ustvarili povezavo, je potrebno preveriti, ali le-ta deluje. To storimo z `if` stavkom, ki v primeru nedelujoče povezave uporabniku javi napako, v nasprotnem primeru pa nadaljujemo na naslednji korak.

```
if($povezava->connect_error){  
    //javi napako  
}  
else{  
    //nadaljuj postopek  
}
```

### 3.5.1.3 SQL PRIPRAVLJENI STAVKI

Pripravljeni stavki so eden varnejših načinov izvajanja SQL poizvedb, saj so zelo učinkoviti pri preprečevanju napadov z vstavljanjem (angl. SQL injection). Pripravljen stavek je na začetku sestavljen le iz ukazov in vrednosti, ki se med izvajanjem stavka nikoli ne spreminjajo. Vsi tisti podatki, ki so odvisni od uporabnika in se morajo uporabiti v poizvedbi, so stavku dodani kasneje in se imenujejo parametri. Za njih je značilno, da ne morejo spremeniti strukture prvotnega stavka, kar pomeni, da napadalec nima možnosti prirejanja SQL stavkov. Pripravljen stavek shranimo v obliki nove spremenljivke z ukazom `$stavek = $povezava->prepare()`, ki mu dodamo stavek SQL, v katerem na mesto, kjer pridejo parametri zapišemo vprašaj. Nato uporabimo ukaz `$stavek->bind_param()`, s katerim stavku pripnemo parametre, ki jim prej definiramo tudi podatkovni tip (i – celo število, d – realno število, s – niz). Stavek nato izvršimo z ukazom `$stavek->execute()`. (18)

Primer:

```
$stavek = $povezava->prepare("SELECT * FROM uporabniki WHERE  
id_uporabnika = ?");  
  
$stavek->bind_param("i", $_SESSION["id_uporabnika"]);  
  
$stavek->execute();
```

### 3.5.1.4 OBDELAVA REZULTATOV POIZVEDBE

Rezultate pridobimo tako, da ustvarimo novo spremenljivko z ukazom `$rezultat = $stavek->get_result()`. V naslednjem koraku preverimo, ali smo glede na našo poizvedbo iz podatkovne baze sploh dobili kakšen rezultat. To storimo z lastnostjo `$rezultat->num_rows`, ki nam pove število vrstic rezultata, ki smo ga s poizvedbo pridobili iz podatkovne baze. Nato se z while zanko sprehodimo čez vse vrstice našega rezultata. Podatke vsake vrstice lahko shranimo kot asociativno tabelo z ukazom `$vrstica = $rezultat->fetch_assoc()`.

### Primer:

```
$rezultat = $stavek->get_result();

if($rezultat->num_rows == 0){

    //ni rezultatov

}

else{

    while($vrstica = $rezultat->fetch_assoc()){

        tukaj so dostopni podatki trenutne vrstice npr.
        $vrstica["id_uporabnika"], $vrstica["ime"] ...

    }

}

require("../podatki/podatki.php");
$baza = new mysqli($p[0], $p[1], $p[2], $p[3]);
if($baza->connect_error){
    echo "<div id='napaka'> Povezava žal ni bila uspešna </div>";
}
else{
    $pridobi_podatke = $baza->prepare("SELECT id_vprasanja, vprasanje, odgovor_A, odgovor_B, odgovor_C,
    odgovor_D, pravilen FROM vprasanja WHERE id_kviza = ? ORDER BY datum_vprasanja");
    $pridobi_podatke->bind_param("i", $_GET["id_kviza"]);
    if($pridobi_podatke->execute()){
        $rezultat = $pridobi_podatke->get_result();
        if($rezultat->num_rows == 0){
            echo "<div id='brez'> Trenutno nimate nobenega vprašanja </div>";
        }
        else{
            $i=1;
            while($vrstica = $rezultat->fetch_assoc()){
                $vprasanje = '<span>' . $i . '. </span>' . $vrstica["vprasanje"];
                $odgovori["A"] = $vrstica["odgovor_A"];
                $odgovori["B"] = $vrstica["odgovor_B"];
                $odgovori["C"] = $vrstica["odgovor_C"];
                $odgovori["D"] = $vrstica["odgovor_D"];
                echo dodaj_vrstico($vrstica["id_vprasanja"], $vprasanje, $odgovori, $vrstica["pravilen"]);
                $i++;
            }
        }
    }
    $baza->close();
}
```

Slika 17: primer PHP kode za povezavo s podatkovno bazo

### 3.5.2 PRIKAZ PODATKOV UPORABNIKU

Za prikaz podatkov uporabniku sem ustvaril funkcijo, ki kot parametre sprejme vse potrebne podatke iz trenutne vrstice pridobljenih rezultatov in v obliki niza vrne HTML kodo, ki predstavlja izpis določene vrstice na uporabnikovem zaslonu. Ko uporabnik npr. želi urejati svoj kviz, se mu prikažejo vprašanja z odgovori, vsako vprašanje je pridobljeno preko SQL poizvedbe in se v kodo HTML vgradi preko funkcije `dodaj_vrstico()` (glej Slika 17).

```
function dodaj_vrstico($id_vprasanja, $vprasanje, $odgovori, $pravilen){
    $pn = pravilno_napacno($odgovori, $pravilen);
    $izbrisi = "";
    if(isset($_SESSION["vprasanja"])[ne_prikazuj]){
        $izbrisi = "&izbrisi=true";
    }
    $prikaz = ' <div class="kviz">
        <div class="vprasanje">
            ' . $vprasanje . '
        </div>
        <div class="odgovori">
            <div> ' . $pn["A"] . ' <span class="crka">A</span> ' . $odgovori["A"] . ' </div>
            <div> ' . $pn["B"] . ' <span class="crka">B</span> ' . $odgovori["B"] . ' </div>
            <div> ' . $pn["C"] . ' <span class="crka">C</span> ' . $odgovori["C"] . ' </div>
            <div> ' . $pn["D"] . ' <span class="crka">D</span> ' . $odgovori["D"] . ' </div>
        </div>
        <div class="moznosti">
            <a href="statistika.php?id_vprasanja=' . $id_vprasanja . '">
                <i class="fas fa-poll-h"></i> statistika</a>
            <a href="izbrisi_vprasanje.php?id_vprasanja=' . $id_vprasanja . $izbrisi . '">
                <i class="fas fa-trash-alt"></i> izbriši</a>
            </div>
        </div> ' ;
    return $prikaz;
}
```

Slika 18: funkcija za prikaz podatkov v HTML obliki

V mojem primeru (Slika 18) funkcija pridobi ID vprašanja, vprašanje, vse štiri odgovore in podatek o tem, kateri odgovor je pravilen. Nato oblikujemo strukturo prikaza s kodo HTML, ki jo shranimo kot spremenljivko (niz) in ji dodamo podatke, pridobljene iz podatkovne baze. Na koncu so dodane tudi povezave za brisanje oz. vpogled v statistiko določenega vprašanja. Povezavi dodamo parameter ID vprašanja, da lahko ob uporabnikovem kliku na povezavo aplikacija ugotovi, na katero vprašanje se to dejanje nanaša. Funkcija na koncu vrne vrednost spremenljivke, ki vsebuje HTML kodo za prikaz. Z nekaj dodatnega oblikovanja v jeziku CSS pridemo do končnega izgleda spletne strani, preko katere ima uporabnik vpogled v določene podatke iz podatkovne baze (Slika 19).

## Kviz znam

1. Znan pregovor pravi: kdor visoko leta, ...

- ✗ A daleč vidi
- ✗ B je pilot
- ✗ C dol ne gleda
- ✓ D nizko pade

statistika izbriši

2. Koliko dni ima prestopno leto?

- ✗ A 364
- ✗ B 365
- ✓ C 366
- ✗ D 367

statistika izbriši

+ dodaj vprašanje nazaj

Slika 19: primer prikaza podatkov iz baze

## 3.6 DELOVANJE IN LOGIKA KVIZA KOT IGRE

Uporabnik ima za reševanje na voljo različne kvize, ki so jih izdelali drugi uporabniki. Med vsemi kvizi izbere tistega, ki ga želi igrati, pri tem pa se začne odvijati postopek igranja kviza, kjer uporabnik po vrsti odgovarja na vsa vprašanja. Njegovi rezultati se sproti beležijo, na voljo pa so mu tudi tri oblike pomoči. Po zadnjem vprašanju se mu izračuna končni uspeh in kviz je zanj končan.

### 3.6.1 ODOBRITEV VSTOPA V KVIZ

Pred samim začetkom igranja kviza aplikacija preveri, ali je uporabnik sploh upravičen do igranja tega kviza. Če npr. uporabnik poskuša igrati svoj lastni kviz, ali pa želi ponovno igrati kviz, katerega je že reševal in večkratno reševanje ni omogočeno, ga aplikacija obvesti, da nima možnosti igranja tega kviza. Pred začetkom se uporabniku prikaže vstopno obvestilo, v katerem se nahaja opis kviza (dodan s strani avtorja kviza) in univerzalno sporočilo. Če je kviz zaklenjen, se spodaj pojavi polje, kamor mora uporabnik vnesti ustrezno geslo. Ko uporabnik potrdi vstopno obvestilo, mu je vstop v kviz odobren, takoj za tem pa se zanj začnejo beležiti rezultati, ki se shranijo v vsakem primeru, tudi če iz kakršnih koli razlogov predčasno prekine z reševanjem kviza.



**Kviz SQL**

Z vstopom v kviz se začnejo beležiti vaši rezultati. Če boste kviz zapustili predčasno, bodo vsi dotedanji rezultati dokončno shranjeni. Ne uporabljajte tipke "nazaj" in ne osvežujte spletnega mesta, sicer boste preusmerjeni izven kviza. Prosimo, igrajte pošteno! Želimo vam veliko uspeha pri reševanju kviza.

Geslo kviza

☐ pokaži geslo

nadaljuj

Slika 20: primer vstopnega obvestila

### 3.6.2 DINAMIČNO KOMUNICIRANJE S PB S TEHNOLOGIJO AJAX

Kviz od začetka do konca poteka na isti strani (igraj.php), kar pomeni da celoten proces menjave vprašanj, preverjanja pravilnosti podanih odgovorov in sprotne beleženja rezultatov teče brez osveževanja spletne strani. Tovrstno dinamičnost je možno doseči z jezikom JavaScript, vendar pa je sproti potrebna tudi komunikacija s podatkovno bazo. Iz tega razloga sem uporabil tudi tehnologijo AJAX.

AJAX (angl. Asynchronous JavaScript And XML) je skupina medsebojno povezanih spletnih razvojnih tehnik, uporabljenih za ustvarjanje interaktivnih spletnih aplikacij. S tehnologijo AJAX si lahko spletne aplikacije izmenjujejo podatke s strežnikom asinhrono v ozadju, brez potrebe po ponovnem nalaganju strani. Omogoča nam pridobivanje podatkov s strežnika tudi potem, ko je bila spletna stran že naložena, tekoče spreminjanje vsebine in pošiljanje podatkov v ozadju nazaj na strežnik. Funkcijo AJAX vključimo direktno v kodo JavaScript, s katero nato tudi prikažemo vse rezultate opravljenih poizvedb (glej Slika 21). (19)

```

let poizvedba = new XMLHttpRequest();
poizvedba.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        let r = document.getElementsByClassName("rezultat")[trenutno_vprasanje];
        if(this.responseText != false){
            document.getElementById("prikaz").innerHTML = this.responseText;
            document.getElementById("st").innerHTML = trenutno_vprasanje + 1 + ".";
            document.getElementById("pomoci").style.display = "flex";
            document.getElementById("nadaljuj").style.display = "none";
            r.style.color = "white";
            r.style.background = "#124fa0";
        }
        else{
            napaka();
            r.style.border = "solid red 2px";
        }
    }
};
poizvedba.open("POST", "poizvedbe/pridobi_vprasanje.php");
poizvedba.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
poizvedba.send("id_vprasanja=" + vprasanja[trenutno_vprasanje] );

```

Slika 21: del funkcije v JavaScript za prikaz novega vprašanja s tehnologijo AJAX

### 3.6.3 PROCES IZBIRE ODGOVORA

Pri vsakem vprašanju lahko uporabnik izbira med štirimi možnimi odgovori. Vsak izmed odgovorov se nahaja na svojem gumbu. Ko uporabnik klikne na enega izmed teh štirih gumbov, se pokliče funkcija, ki preveri, ali je izbran odgovor pravilen. Takoj, ko uporabnik pritisne na gumb, je njegova izbira odgovora dokončna, vsi ostali gumbi se do naslednjega vprašanja zaklenejo. Pri preverjanju pravilnosti odgovora sodeluje funkcija, realizirana s tehnologijo AJAX, ki se poveže z bazo podatkov. Če je izbran odgovor pravilen, se gumb obarva v zeleno, v nasprotnem primeru pa se izbrani odgovor obarva rdeče, pravilen odgovor pa zeleno.

1. Znan pregovor pravi: kdor visoko leta, ...

A daleč vidi	B je pilot
C dol ne gleda	D nizko pade

polovička   glas ljudstva   joker

1 2 3 4 5 6 7 8 9 10 11 12

Slika 22: prikaz vprašanja pri reševanju kviza



```
let izbira = element.id;
let resitev = pravilen();
for(let i=0; i<4; i++){
  document.getElementsByClassName("odgovor")[i].disabled = true;
}
document.getElementById("pomoci").style.display = "none";
document.getElementById("nadaljuj").style.display = "flex";
if(resitev == false){ napaka();}
else{
  let pravilno = (izbira==resitev);
  let r = document.getElementsByClassName("rezultat")[trenutno_vprasanje];
  if(pravilno){
    element.style.background = "#66ff66";
    element.style.color = "black";
    r.style.color = "black";
    r.style.background = "#66ff66";
    st_pravilnih++;
  }
  else{
    element.style.background = "red";
    document.getElementById(resitev).style.background = "#66ff66";
    document.getElementById(resitev).style.color = "black";
    r.style.color = "white";
    r.style.background = "red";
  }
  r.style.paddingTop = "6px";
  r.style.border = "transparent";
  zabelezi_odgovor(izbira, pravilno);
}
```

Slika 23: koda JavaScript za preverjanje pravilnosti izbranega odgovora

### 3.6.4 REALIZACIJA OBLIK POMOČI

Med igranjem kviza lahko igralec uporabi tri oblike pomoči, in sicer polovičko, glas ljudstva in joker. Vsako od teh treh pomoči lahko med samim kvizom uporabi samo enkrat pri kateremkoli vprašanju. Možno je tudi uporabiti dve ali pa kar vse tri pomoči pri istem vprašanju.

Ob pritisku na gumb polovička se izvede zanka, ki se ponovi dvakrat. Med vsako ponovitvijo zanke program naključno izbere enega izmed štirih odgovorov. Nato preveri, ali je izbrani odgovor napačen in ali je bil morda že predhodno izločen. Če je odgovor res napačen in ni bil predhodno izločen, ga program izloči, kar pomeni, da se gumb zaklene, besedilo odgovora pa se skrije. V nasprotnem primeru se zanka vrne na začetek in se ta del ponovno izvede. Polovička torej povzroči, da se izločita dva naključno izbrana napačna odgovora.

```
function polovicka(){
  let p = document.getElementById("polovicka");
  p.disabled = true;
  p.style.display = "none";
  for(let i=0; i<2; i++){
    let el = document.getElementsByClassName("odgovor")[nakljucno(0,3)];
    if(el.disabled == true || el.id == pravilen()){
      i--;
      continue;
    }
    else{
      el.disabled = true;
      el.firstChild.style.visibility = "hidden";
      el.style.background = "#124fa0";
    }
  }
}
```

Slika 24: funkcija za pomoč polovička

Glas ljudstva je pomoč, pri kateri lahko igralec vidi, kako so na določeno vprašanje odgovarjali vsi ostali, ki so do tistega trenutka reševali kviz. Ob pritisku na gumb, se izvede funkcija, ki iz podatkovne baze za vsak posamezen odgovor pridobi število igralcev, ki so izbrali ta odgovor. V naslednjem koraku se izračuna delež v odstotkih in se v obliki vrstičnega diagrama prikaže na zaslon. Igralec lahko torej vidi, koliko odstotkov vseh igralcev je izbralo odgovor A, B, C in D.

Če igralec izkoristi pomoč Joker, ima možnost, da izmed vseh 4 odgovorov oz. tistih, ki niso že od prej izločeni, poljubno izbere natanko dva odgovora. Med izbranimi odgovoroma nato program naključno izbere enega. Zatem preveri, ali je ta odgovor pravilen. Če pride do take ugotovitve, se postopek naključne izbire ponovi. V primeru, da je izbran odgovor napačen, ga izloči. Joker torej izmed dveh izbranih izloči en odgovor, ki je zagotovo napačen.

### 3.6.5 BELEŽENJE IN KONČNI IZRAČUN REZULTATOV

Igralec ima že med samim reševanjem kviza sproten pregled nad svojimi rezultati in nad številom vprašanj, ki so še pred njim. Rezultati so predstavljeni čisto spodaj pod vprašanjem in vsemi odgovori (glej Slika 22), in sicer v obliki kvadratkov. Vsako vprašanje ima svoj oštevilčen kvadrat, ki se po prikazu vprašanja obarva modro. Če igralec na vprašanje odgovori pravilno, se kvadratik obarva zeleno, sicer postane rdeč. S pomočjo tega prikaza lahko igralec vidi, kako je odgovarjal na vsa dozdajšnja vprašanja in koliko vprašanj ga še čaka do konca kviza.

Število kvadratkov je seveda odvisno od skupnega števila vprašanj, ki pa je za vsak kviz drugačno. Ob začetku igre mora program glede na število vprašanj ustrezno generirati in oštevilčiti kvadratke za prikaz rezultatov. To naredimo z zanko v jeziku PHP:

```
<div id="rezultati">

<?php
    for($i=0; $i<sizeof($vprasanja); $i++){
        echo '<div class="rezultat">' . ($i + 1) . '</div>';
    }
?>

</div>
```

Ko je igralec končal z reševanjem kviza se mu prikažejo končni rezultati, torej koliko pravih odgovorov glede na skupno število vprašanj je dosegel. Izračuna se mu tudi uspešnost v odstotkih. Vsi rezultati se seveda shranijo v podatkovno bazo, kar pomeni da ima uporabnik vedno pregled nad doseženim rezultatom. Na koncu le še potrdi rezultat in zapusti kviz.

Končni rezultati se prikažejo s pomočjo naslednje JavaScript kode:

```
document.getElementById("pravih").innerHTML = st_pravih;

document.getElementById("skupno").innerHTML =
vprasanja.length;

document.getElementById("uspešnost").innerHTML =
Math.round((100.0*st_pravih)/vprasanja.length) + "%";
```

### **3.7 TESTIRANJE DELOVANJA APLIKACIJE**

Razvoj aplikacije je potekal postopno med decembrom 2021 in aprilom 2022. Najbolj obsežno je bilo sprotno testiranje delovanja med samim razvojem aplikacije, ki je bilo izvedeno v brskalniku Google Chrome, ki velja za enega najbolj priljubljenih spletnih brskalnikov. Da lahko z gotovostjo trdim, da aplikacija deluje pravilno in po pričakovanjih, sem delovanje testiral tudi v nekaj ostalih spletnih brskalnikih, kot so Mozilla Firefox, Opera in Microsoft Edge. Ugotovil sem, da je funkcionalnost aplikacije v vseh testiranih brskalnikih brezhibna in da ni večjih odstopanj kar se tiče samega izgleda spletne strani ter pozicije posameznih elementov. Spletna stran je bila testirana tudi s strani uporabnikov, s pomočjo katerih so bile odpravljene manjše pomanjkljivosti, odkrite v kodi.

## 4 ZAKLJUČEK

V sklepnem delu seminarske naloge ugotavljam, da mi je uspelo doseči, mogoče celo preseči, zastavljene cilje. Pri načrtovanju in izgradnji aplikacije sem spoznal, da je postopek razvoja spletne aplikacije zelo zapleten in dolgotrajen. Da lahko na koncu pridemo do delujoče aplikacije, je potrebno narediti natančne načrte, s pomočjo katerih lahko nato postopoma gradimo celoten sistem. Med razvojem aplikacije sem naletel na nemalo težav, ki sem jih reševal sproti z nadaljnjim raziskovanjem. Pri tem projektu sem uspel nadgraditi svoje znanje programiranja in pridobiti ogromno novih znaj s področja spletnih tehnologij. Moram pa priznati, da brez osnovnega predznanja, discipliniranosti pri delu in zanimanja za to temo izvedba tako zahtevnega projekta ne bi bila mogoča. Hkrati bi želel opozoriti, da se, zlasti pri tako obsežnem projektu, vedno lahko pojavijo raznorazne napake oz. pomanjkljivosti. Do enakih rešitev bi se zagotovo dalo priti tudi na drugačne načine, morda celo boljše in lažje kot sem jih uporabil sam. Možna je tudi nadgradnja in nadaljnji razvoj aplikacije, kateri bi se lahko dodajalo še številne nove funkcionalnosti. Snov seminarske naloge obsega le del celotnega projekta. Pri razlagi sem se potrudil, da izpostavim najzanimivejše in najpomembnejše stvari ter jih razložim na karseda enostaven način. Upam, da je seminarska naloga bralcu dobro razumljiva, zanimiva in seveda zanj tudi koristna.

## 5 STVARNO KAZALO

<b>A</b>		<b>K</b>	
AJAX	4, 5, 26, 27	ključ	9, 11, 19, 35
aplikacija	1, 6, 14, 24, 25, 31	koda	5, 8, 16, 28
atribut	9, 10, 11, 35	kviz	4, 1, 6, 10, 24, 25, 29, 30, 37
<b>B</b>		<b>O</b>	
baza	8, 12, 21, 37	obrazec	5, 14, 15, 17
brskalnik	2, 20		
<b>C</b>		<b>P</b>	
CSS	4, 5, 1, 3, 8, 15, 24, 35	PHP	4, 5, 1, 5, 15, 18, 19, 20, 21, 23, 30, 35, 36
<b>E</b>		piškotek	20
entiteta	9, 35	pomoč	5, 6, 29, 34
ER	4, 5, 8, 11, 13, 35	povezava	21, 22
		preverjanje	5, 4, 16, 17, 18, 19, 28
		program	20, 28, 29, 30
<b>F</b>		<b>R</b>	
funkcija	5, 18, 24, 27, 29	rezultati	6, 8, 10, 21, 25, 29, 30
<b>G</b>		<b>S</b>	
geslo	10, 11, 13, 19, 21, 25	seja	20
<b>H</b>		spletna stran	26
HTML	4, 5, 1, 2, 3, 4, 5, 15, 16, 24, 35	SQL	4, 1, 12, 13, 22, 24, 35, 37
		strežnik	15, 17, 18, 19, 26
<b>I</b>		<b>T</b>	
ID	19, 20, 24	testiranje	31
igralec	6, 28, 29, 30		
<b>J</b>		<b>U</b>	
JavaScript	4, 5, 4, 17, 18, 26, 27, 28, 30, 35	uporabnik	6, 7, 10, 14, 15, 16, 17, 18, 19, 20, 21, 24, 25, 27, 30

## 6 ZAHVALA

Na tej točki bi se želel zahvaliti svojemu mentorju, prof. dr. Albertu Zorku, za vso podporo, pomoč in usmeritev pri izdelavi te seminarske naloge. Zahvaljujem se mu tudi za korektno predavano snov iz računalništva in informatike skozi vsa štiri leta, s katero sem pridobil številna znanja, brez katerih izvedba takega projekta ne bi bila mogoča.

Prav tako se zahvaljujem vsem, ki so sodelovali pri testiranju delovanja moje aplikacije oz. so na kakršenkoli drug način prispevali k izvedbi projekta in oblikovanja končnega izdelka. Iskrena zahvala gre tudi moji družini, ki mi je tekom celotnega srednješolskega izobraževanja predstavljala podporo in mi stala ob strani tudi v najtežjih trenutkih.

Tomaž Černe



# VIRI IN LITERATURA

1. Krebelj, Peter. *HTML in CSS za začetnike*. Ljubljana : Atelje Doria, 2015. pp. 19, 70-80, 98.
2. HTML Tutorial. *Tutorialspoint*. [Online] 2012. [Cited: marec 28, 2022.] <https://www.tutorialspoint.com/html/index.htm>.
3. W3C. HTML & CSS - W3C. *W3.org*. [Online] 2008. [Cited: marec 30, 2022.] <https://www.w3.org/standards/webdesign/htmlcss>.
4. JavaScript. *eNSA*. [Online] [Cited: marec 30, 2022.] <https://nsa-splet.si/js/js.php>.
5. Bilke, Petra. *Spoznajmo PHP in MySQL*. Nova Gorica : Flamingo Založba d.o.o., 2002. pp. 14, 44.
6. PHP Tutorial - Tutorialspoint. *Tutorialspoint*. [Online] 2019. [Cited: marec 30, 2022.] <https://www.tutorialspoint.com/php/index.htm>.
7. Mohorič, Tomaž. *Podatkovne baze*. Ljubljana : BI-TIM d.o.o., 2002. pp. 13-15.
8. Entiteta in entitetni tip. *colos.fri.uni-lj.si*. [Online] [Cited: november 27, 2021.] [http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/PODAT\\_PLAT\\_RAZ\\_PO/entiteta\\_in\\_entitetni\\_tip.html](http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/PODAT_PLAT_RAZ_PO/entiteta_in_entitetni_tip.html)
9. Primarni ključ. *egradivo.ecnm.si*. [Online] [Cited: november 27, 2021.] [http://egradivo.ecnm.si/BAZE/primarni\\_klju.html](http://egradivo.ecnm.si/BAZE/primarni_klju.html).
10. Atribut, tip podatkovnega elementa. *colos.fri.uni-lj.si*. [Online] [Cited: november 27, 2021.] [http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/PODAT\\_PLAT\\_RAZ\\_PO/atribut\\_tip\\_podatkovnega\\_elementa.html](http://colos.fri.uni-lj.si/eri/RACUNALNISTVO/PODAT_PLAT_RAZ_PO/atribut_tip_podatkovnega_elementa.html).
11. What is a Foreign Key? - Definition from Techopedia. *Techopedia.com*. [Online] 2019. [Cited: november 27, 2021.] <https://www.techopedia.com/definition/7272/foreign-key>.
12. Entity Relationship Diagram (ERD) - What is an ER Diagram? *Smartdraw.com*. [Online] 2019. [Cited: november 27, 2021.] <https://www.smartdraw.com/entity-relationship-diagram/>.
13. Loshin, Peter. What is Structured Query Language (SQL)? *techtargget.com*. [Online] februar 2022. [Cited: april 8, 2022.] <https://www.techtargget.com/searchdatamanagement/definition/SQL>.
14. Obrazci v HTML. *nsa-splet.si*. [Online] [Cited: april 9, 2022.] <https://nsa-splet.si/html/zgledi/html-zgledi-13-obrazci.php>.



15. Input Validation: Client-Side & Server-Side Cybersecurity Deterrent. *SecureCoding*. [Online] avgust 22, 2021. [Cited: april 10, 2022.] <https://www.securecoding.com/blog/input-validation>.
16. Seje. *e-računalništvo*. [Online] [Cited: april 10, 2022.] <https://gradiva.txt.si/index.php/seje>.
17. PHP: Piškotki. *nsa-splet.si*. [Online] [Cited: april 10, 2022.] <https://nsa-splet.si/php/zgledi/php-zgledi-24-piskotki.php>.
18. PHP MySQL Prepared Statements. *w3schools.com*. [Online] [Cited: april 11, 2022.] [https://www.w3schools.com/php/php\\_mysql\\_prepared\\_statements.asp](https://www.w3schools.com/php/php_mysql_prepared_statements.asp).
19. Garrett, Jesse James. Ajax: A New Approach to Web Applications. *immagic.com*. [Online] februar 18, 2005. [Cited: april 11, 2022.] <https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf>.

# PRILOGE

Dostop do vseh datotek, ki sestavljajo spletno stran kviz-znam.si, je možen preko spodnje povezave:

<https://github.com/tomazcerne/kviz-znam>

Tekstovne datoteke z SQL ukazi, s katerimi sem vzpostavil podatkovno bazo, so dostopne na naslednji povezavi:

[https://github.com/tomazcerne/kvizznam\\_baza](https://github.com/tomazcerne/kvizznam_baza)

Seminarska naloga (ta dokument) se v elektronski obliki nahaja na tej povezavi:

<https://github.com/tomazcerne/Seminarska-naloga>