IZVEDBA ISPITNOG SUSTAVA POMOĆU ASP.NET MVC WEB APLIKACIJE I ANDROID MOBILNE APLIKACIJE

Ferenc, Ivan

VISOKA ŠKOLA ZA PRIMJENJENO RAČUNARSTVO

ZAVRŠNI RAD

IZVEDBA ISPITNOG SUSTAVA POMOĆU ASP.NET MVC WEB APLIKACIJE I ANDROID MOBILNE APLIKACIJE

Ivan Ferenc

Koprivnica, svibanj 2015.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Koprivnici, 05. svibnja 2015.

Predgovor

Zahvaljujem se svim kolegama i prijateljima koji su me bodrili i podržavali tijekom cijelog studija na Visokoj školi za primijenjeno računarstvo te prilikom izrade ovog rada. Najveće hvala ide mojoj obitelji koja mi je bila najveća potpora tijekom cijelog studija.

Posebna zahvala ide mom mentoru Bojanu Fulanoviću koji mi je svojim savjetima i komentarima pomogao prilikom izrade ovog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi.

Sažetak

Glavna ideja za ovaj završni rad mi je bila izraditi aplikaciju za ispitni sustav. Krajnji rezultat je web aplikacija koja s jedne strane omogućuje kreiranje ispitnog sadržaja (novih ispita, pitanja i odgovora), a s druge strane rješavanju ispita registriranim korisnicima, te android mobilna aplikacija koja je preko web servisa povezana sa istom bazom podataka kao i web aplikacija i koja također omogućuje korisnicima polaganje ispita. Nadalje, obje aplikacije imaju mogućnost registracije novih korisnika, te uvid u dosad polagane ispite korisnicima koji su prijavljeni u ispitni sustav.

Ključne riječi: ispitni sustav, ispitni sadržaj, web aplikacija, mobilna aplikacija, zajednička baza podataka, web servis, registracija i prijava korisnika, sigurnost.

Summary

Main idea for my thesis was to create application that will represent exam system. Final result is web application which on the one hand provides ability to create exam content (new exams, exam questions and question answers), and on the other hand provides ability for registered users to take exams. Other part of result is android mobile application which is connected through web service with same database as web application and also provides ability for registered users to take exams. Both applications have ability to register new users and providing insight in taken exams for logged in users.

Keywords: exam system, exam content, web application, mobile application, database, web service, registration, login, security.

Sadržaj

1 Uvod

Želja mi je bila izraditi aplikaciju univerzalnog ispitnog sustava koja, uz samo polaganje ispita, vlasniku sustava omogućava kreiranje ispitnog sadržaja te uvid u statističke podatke vezane za pojedini ispit i za pojedinog polaznika.

Prilikom dizajniranja samog sustava, odlučio sam se za web aplikaciju i mobilnu aplikaciju koje bi koristile zajedničku bazu podataka.

Web aplikacija bi vlasniku sustava pružala pristup administratorskom sučelju iz kojeg bi mogao u potpunosti upravljati sadržajem ispitnog sustava. To bi uključivalo kreiranje ispita i ispitnih opcija kao što su broj pitanja u ispitu, negativni bodovi, dostupno vrijeme rješavanja, potreban postotak za prolaz, te kreiranje pitanja i odgovora. Polaznici sustava imali bi mogućnost registracije ili prijave u sustav, te pristup rješavanju ispita ili uvid u već polagane ispite.

Mobilna aplikacija bi bila namijenjena za android operativni sustav, korisnicima bi omogućavala registraciju ili prijavu u sustav, te polaganje ispita ili pregled prijašnjih rezultata ispita. Problem je bio korištenje zajedničke baze podataka koju koristi i web aplikacija, pa sam u dizajn sustava uveo web servis preko kojeg će mobilna aplikacija pristupati zajedničkim ispitnim i korisničkim podacima.

U nastavku slijedi detaljniji opis arhitekture ispitnog sustava i tehnologija korištenih pri izradi koje su podijeljene u 4 većih cjelina:

- baza podataka
- web aplikacija
- web servis
- android mobilna aplikacija

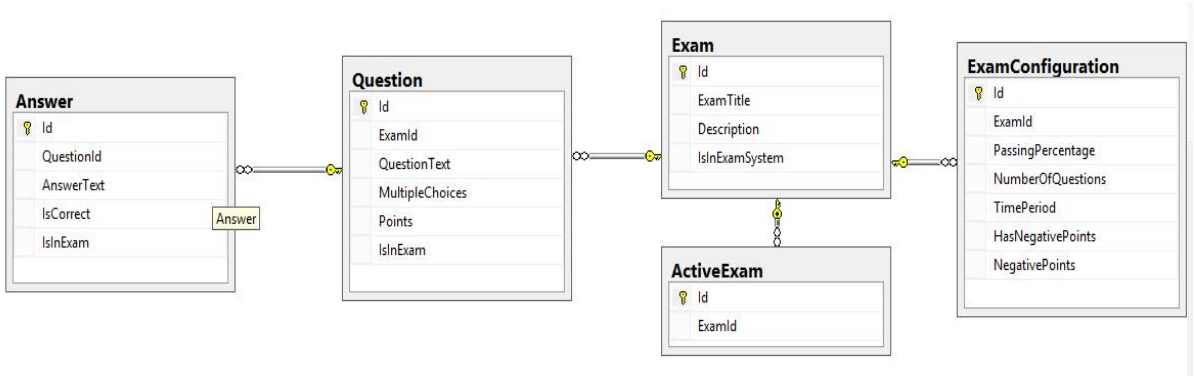
2 Baza podataka

U dizajniranju arhitekture ispitnog sustava krenuo sam od baze podataka kao središnjeg zajedničkog dijela s kojim će i web i mobilna aplikacija razmjenjivati podatke. Sustav treba pohranjivati različite vrste podataka, pa sam ih odlučio podijeliti u nekoliko logičkih cjelina:

- ispitni podaci
- korisnički podaci
- zapisi o polaganim ispitima

2.1 Ispitni podaci

U ovu skupinu spadaju podaci vezani za same ispite u ispitnom sustavu. Njih sam podijelio u nekoliko manjih logičkih cjelina, entiteta, kao što je prikazano na slici:



Slika 1 - Entiteti ispitnih podataka

2.1.1 Entitet Exam

Entitet *Exam* predstavlja sam ispit u ispitnom sustavu i opisan je sljedećim atributima:

- *Id* – primarni ključ
- *ExamTitle* – naslov ispita
- *Description* – opis ispita
- *IsInExamSystem* – da li je ispit trenutno aktivan u ispitnom sustavu

2.1.2 Entitet ActiveExam

Entitet *ActiveExam* predstavlja trenutno aktivni ispit u ispitnom sustavu kojem korisnici mogu pristupiti, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *ExamId* – strani ključ, veza sa entitetom Exam

2.1.3 Entitet ExamConfiguration

Entitet *ExamConfiguration* predstavlja ispitne opcije nekog ispita i opisan je slijedećim atributima:

- *Id* – primarni ključ
- *ExamId* – strani ključ, veza sa entitetom Exam
- *PassingPercentage* – postotak potreban za prolazak ispita
- *NumberOfQuestions* – broj pitanja u ispitu
- *TimePeriod* – raspoloživo vrijeme za rješavanje ispita
- *HasNegativePoints* – da li ispit ima negativne bodove
- *NegativePoints* – vrijednost negativnih bodova u ispitu

2.1.4 Entitet Question

Entitet *Question* predstavlja pitanje u ispitu, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *ExamId* – strani ključ, veza sa entitetom Exam
- *QuestionText* – tekst pitanja
- *MultipleChoices* – da li pitanje ima više točnih odgovora
- *Points* – bodovna vrijednost pitanja
- *IsInExam* – da li je trenutno u ispitu

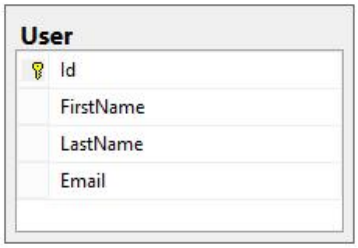
2.1.5 Entitet Answer

Entitet *Answer* predstavlja odgovor u nekom pitanju, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *QuestionId* – strani ključ, veza sa entitetom Question
- *AnswerText* – tekst odgovora
- *IsCorrect* – da li je odgovor točan
- *IsInExam* – da li je odgovor trenutno u ispitu

2.2 Korisnički podaci

U ovu skupinu spadaju podaci vezani za registrirane korisnike. Na sljedećoj slici je prikazan entitet registriranog korisnika:



Slika 2 - Entitet korisničkih podataka

Sa ostalim korisničkim podacima potrebnima za prijavu u sustav kao što je na primjer lozinka, podacima o vanjskim računima za prijavu kao što su na primjer Google, Facebook, upravlja ASP.NET Identity System koji će detaljnije biti objašnjen u sklopu sljedeće cjeline vezane za web aplikaciju.

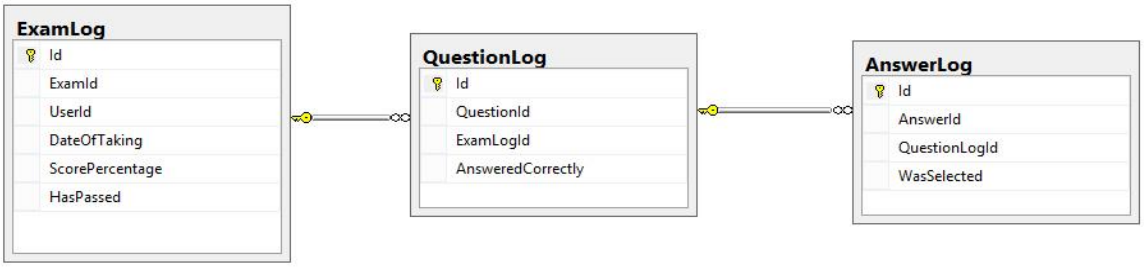
2.2.1 Entitet User

Entitet *User* predstavlja registriranog korisnika ispitnog sustava, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *FirstName* – korisničko ime
- *LastName* – korisničko prezime
- *Email* – korisnička email adresa, jedinstvena vrijednost (engl. unique)

2.3 Zapisi o polaganim ispitima

U ovu skupinu spadaju podaci vezani za zapise o riješenim ispitima i njihovim rezultatima. Podijeljeni su na nekoliko manjih logičkih cjelina, kao što je prikazano na sljedećoj slici:



Slika 3 - Entiteti zapisa o polaganim ispitima

2.3.1 Entitet ExamLog

Entitet *ExamLog* predstavlja riješeni ispit nekog korisnika, a opisan je sljedećim atributima:

- *Id* – primarni ključ
- *ExamId* – strani ključ, veza sa entitetom Exam
- *UserId* – strani ključ, veza sa entitetom User
- *DateOfTaking* – datum i vrijeme polaganja ispita
- *ScorePercentage* – ostvaren rezultat
- *HasPassed* – da li je ispit uspješno položen

2.3.2 Entitet QuestionLog

Entitet *QuestionLog* predstavlja zapis o pitanju riješenog ispita, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *QuestionId* – strani ključ, veza sa entitetom Question
- *ExamLogId* –strani ključ, veza sa entitetom ExamLog
- *AnsweredCorrectly* – da li je pitanje točno odgovoreno

2.3.3 Entitet AnswerLog

Entitet *AnswerLog* predstavlja zapis o odgovoru pitanja riješenog ispita, a opisuju ga sljedeći atributi:

- *Id* – primarni ključ
- *AnswerId* – strani ključ, veza sa entitetom Answer
- *QuestionLogId* – strani ključ, veza sa entitetom QuestionLog
- *WasSelected* – da li je bio označen kao točan odgovor prilikom rješavanja ispita

3 Web aplikacija

Nakon dizajna baze podataka, na red je došao dizajn web aplikacije. Aplikacija je morala odgovarati slijedećim zahtjevima:

- mogućnost kreiranja ispitnog sadržaja (ispita, ispitnih opcija, pitanja, odgovora)
- mogućnost uređivanja ispitnog sadržaja (ispita, ispitnih opcija, pitanja, odgovora)
- mogućnost prijave i registracije novih korisnika u sustav
- mogućnost korisniku da pregleda povijest polaganih ispita i njihovih rezultata
- mogućnost uvida vlasniku aplikacije u statističke podatke vezane za pojedini ispit
- mogućnost uvida vlasniku aplikacije u ispitne rezultate pojedinog polaznika

Aplikaciju sam odlučio izraditi pomoću ASP.NET MVC 5 Frameworka.

3.1 ASP.NET MVC Framework

Ovaj programski okvir je ekstenzija Microsoftovog .NET Frameworka, točnije njegove web sekcije ASP.NET. Temelji se na *MVC* obrascu dizajniranja arhitekture aplikacija.

3.1.1 MVC obrazac dizajniranja

MVC (*Model-View-Controller*) obrazac dizajniranja se već godinama koristi u računarstvu, a prvi puta se pojavio 1979. godine pod imenom *Thing-Model-View-Editor*. Pruža dobre i elegantne načine za razdvajanje poslova unutar same aplikacije (engl. *separation of concerns*), kao što su na primjer odvajanje prezentacije podataka i pristupa podacima. To dodaje malo povećanje kompleksnosti samog dizajna aplikacije, no to je zanemarivo u usporedbi sa krajnjim koristima ovog dizajna: dijelovi aplikacije nisu usko povezani, lakše se testiraju, a i u potpunosti mijenjaju bez da uzrokuju grešku u radu ostalih dijelova aplikacije.

MVC razdvaja korisničko sučelje (engl. *UI, user interface*) aplikacije na tri glavna dijela:

- Model – klase koje opisuju podatke koji se koriste u aplikaciji i klase koje sadrže poslovnu logiku vezanu za te podatke
- View – definira kako će korisničko sučelje biti prikazano korisniku, omogućuje komunikaciju s korisnikom
- Controller – klase koje upravljaju sa korisničkom komunikacijom, upravljaju tokom podataka, delegiraju posao sloju Model koji priprema podatke koji će u sloju View biti prikazani korisniku

3.1.2 Kratka povijest ASP.NET MVC Frameworka

3.1.2.1 ASP.NET MVC 1

U veljači 2007. godine Scott Guthrie je skicirao jezgru ASP.NET MVC. U početku je to bila jednostavna aplikacija koja se sastojala od nekoliko stotina linija koda. U listopadu 2007. godine pokazao ju je grupi programera i pitao ih za njihovo mišljenje. Aplikacija je odmah postala hit, te se puno ljudi uključilo u razvoj prototipa, kodnog imena Scalene. Eilon Lipton je razvojnom timu poslao prvu verziju prototipa, te su on i Scott Guthrie često razmjenjivali verzije prototipa, dijelova koda, ideja. Rezultat njihovog rada je objavljen 13. ožujka 2009. godine u obliku MVC 1.0.

3.1.2.2 ASP.NET MVC 2

ASP.NET MVC 2 službeno je objavljen u ožujku 2010. godine. Neke od njegovih glavnih značajki su bile:

- UI „pomagači“ (engl. *helpers*) sa automatskim kreiranjem predložaka temeljeni na podatkovnom modelu (engl. *scaffolding*) namijenjeni za CRUD[1] operacije nad podatkovnim modelom
- validacija podatkovnog modela na klijentskoj i serverskoj strani bazirana na atributima
- HTML[2] „pomagači“ (engl. *helpers*) za renderiranje HTML kontrola i povezivanje sa vrijednostima podatkovnog modela pomoću lambda izraza
- poboljšani razvojni alati za Visual Studio

Također je sadržavao mnogo API[3] poboljšanja i „profesionalnih“ značajka koje su bile uvedene kao rezultat prikupljanja povratnih informacija od strane programera koji su razvijali aplikacije pomoću prijašnje verzije programskog okvira. Neke od njih su:

- podrška za dijeljenje većih aplikacija na manje dijelove (engl. *area*)
- podrška za asinkrone kontrolere (engl. *Asynchronous controllers*)
- podrška za renderiranje jednog dijela stranice (engl. *Html.RenderAction*)

[1] CRUD – Create Read Update Delete
[2] HTML – HyperText Markup Language
[3] API – Application Programming Interface

3.1.2.3 ASP. NET MVC 3

Ova verzija je izašla samo 10 mjeseci nakon prijašnje verzije, a neke od važnijih značajki su bile:

- Razor mehanizam
- podrška za *.NET* podatkovne anotacije (engl. *Data Annotations*)
- poboljšana validacija podatkovnih modela
- poboljšana podrška za JavaScript , jQuery[4] validacija

Razor je bio prvo veće poboljšanje za renderiranje HTML-a nakon izlaska prve inačice programskog okvira. Do tada se za renderiranje HTML-a koristila sintaksa iz Web Formi (engl. *Web Forms*) koja je bila dizajnirana kao podrška za editiranje HTML kontrola pomoću grafičkog editora. Razor je bio dizajniran specifično za generiranje HTML koda u prezentacijskom sloju. Njegova sintaksa je lakša za korištenje i nije slična sintaksi korištenoj u Web Formama koja podsjeća na XML[5] sintaksu.

3.1.2.4 ASP.NET MVC 4

Ova verzija programskog okvira se fokusirala na naprednije scenarije korištenja, a neke od važnijih značajki su bile:

- ASP.NET Web API (objašnjeno kasnije)
- poboljšanja standardnih projektnih predložaka
- mobilni projektni predložak koristeći jQuery Mobile – umjesto razvijanja aplikacije za svaki mobilni operativni sustav, ovaj framework baziran na HTML5 omogućuje razvoj jedne aplikacije koja će se moći pokretat na bilo kojoj mobilnoj ili desktop platformi
- podrška zadacima (engl. *Task*, predstavlja asinkronu operaciju) za asinkrone kontrolere
- grupiranje i minimiziranje (engl. *bundling and minification*) – programski okvir koji je uključen u ASP.NET 4.5, sastoji se od dvije tehnike pomoću kojih se može poboljšati vrijeme odaziva na HTTP zahtjev tako što se smanjuje broj zahtjeva prema serveru i veličina datoteka u odgovoru (JavaScript, CSS[6])

[4] jQuery – JavaScript library
[5] XML - EXtensible Markup Language
[6] CSS – Cascading Style Sheet

3.1.2.5 ASP.NET MVC 5

Trenutna verzija programskog okvira u vrijeme pisanja ovog završnog rada, izašla u sklopu Visual Studia 2013 u listopadu 2013. godine. Glavni fokus je bio na inicijativi nazvanoj „One ASP.NET“ i poboljšanju osnovnih značajka svih ASP.NET programskih okvira. Neke od značajka su:

- One ASP.NET
- ASP.NET Identity (objašnjeno kasnije)
- Bootstrap predlošci (objašnjeno kasnije)
- Preusmjeravanje pomoću atributa (engl. *Attribute Routing*) – nova opcija specificiranja ruta pomoću anotacija koje se stavljaju na klase kontrolera ili njihove „akcije“
- Autentikacijski filteri (engl. *Authentication filters*) – omogućuju zabranu neovlaštenog pristupa određenim kontrolerima ili nekoj njihovoj „akciji“

„One ASP.NET“ polazi od toga da su prijašnje verzije MVC programskih okvira kod samog kreiranja projekata nudile previše opcija koje su često zbunjivale korisnike, te se nakon odabira vrste projekta teško bilo prebaciti na neki drugi izbor. U MVC 5 verziji programskog okvira nema više tog problema jer postoji samo jedan tip projekta, Web aplikacija. Podrške za razne programske okvire se lako mogu dodati kroz NuGet menadžer programskih paketa (engl. *NuGet package manager*).

NuGet menadžer programskih paketa je vezan za Microsoft razvojnu platformu. Prvi puta se pojavio 2010. godine, a danas korisnicima pruža veliki izbor raznih alata i servisa. Distribuira se kao ekstenzija Visual Studia. NuGet je zapravo jedan veliki repozitorij biblioteka vezanih za .NET programski paket. U današnje vrijeme postao je jedan od glavnih distribucijskih kanala tih biblioteka u obliku NuGet paketa.

3.2 View sloj ispitnog sustava

Korisničko sučelje web aplikacije sam podijelio na dvije veće cjeline: administratorsko sučelje gdje administrator ima mogućnost upravljanja svim ispitnim sadržajem i uvid u razne statističke podatke te korisničko sučelje gdje registrirani korisnik ima mogućnost pristupa rješavanju ispita i uvid u prijašnje ispite koje je polagao. Grafički izgled korisničkog sučelja temelji se na Bootstrap programskom okviru.

3.2.1 Bootstrap programski okvir

U MVC 5 verziji programskog okvira, projektni predlošci koriste popularni Bootstrap programski okvir. Bootstrap su razvili Mark Otto i Jacob Thornton u Twitteru, a glavni cilj im je bio potaknuti konzistentnost izgleda. Prije pojave Bootstrapa, za razvoj korisničkih sučelja su korišteni razne biblioteke što je dovelo do nedosljednosti izgleda korisničkih sučelja i otežalo njihova održavanja. Trenutno je aktualna Bootstrap 3 verzija programskog okvira objavljena u kolovozu 2013. godine.

3.2.1.1 Bootstrap 3

Bootstrap 3 verzija se temelji na prilagodljivom dizajnu web stranica koji posjetiteljima omogućuje lagan pregled sadržaja na raznim uređajima, od mobitela do desktop računala (engl. *RWD, Responsive Web Design*). Također se temelji na pristupu „Prvo mobilno“ (engl. *Mobile First*) što znači da je sve prilagođeno najmanjim veličinama ekrana i onda se povećava prema određenom omjeru za veće veličine ekrana.

3.2.1.2 Grid sistem

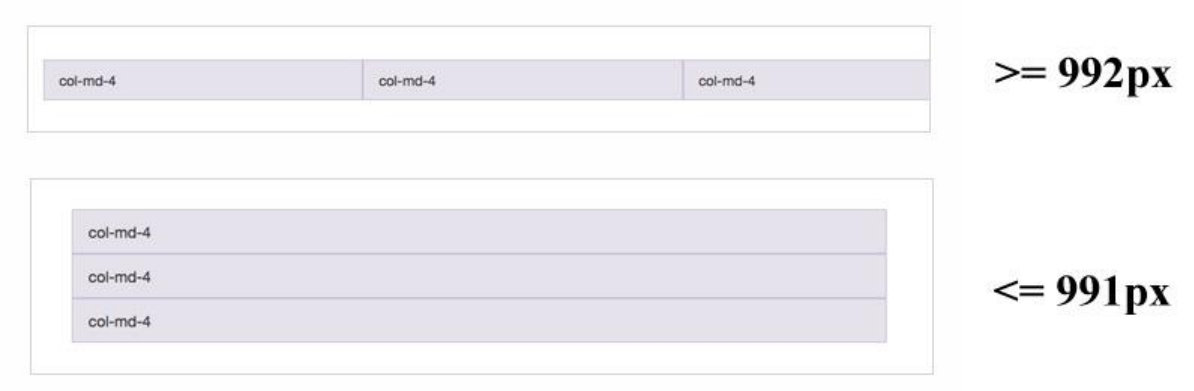
Nova verzija Bootstrapa dolazi sa Grid sustavom kojeg je moguće konfigurirati na 4 različita načina pomoću određene CSS klase:

- uređaji s vrlo malom veličinom ekrana ~ mobiteli (< 768px) ~ col-xs
- uređaji sa malom veličinom ekrana ~ tableti (>= 768px) ~ col-sm
- uređaji sa srednjom veličinom ekrana ~ desktop (>= 992px) ~ col-md
- uređaji sa velikom veličinom ekrana ~ desktop (>= 1200px) ~ col-lg

Za korištenje Grid sustava potreban je element sa klasom „container“ unutar kojeg se nalazi element sa klasom „row“, u kojeg se onda dodaju stupci. Stupci koji imaju klasu sa sufiksom *„xs“* će uvijek biti poredani horizontalno, bez obzira na veličinu ekrana uređaja na kojem se prikazuju. No, ako stupci na primjer imaju klasu sa sufiksom „md“ i prikazuju se na ekranu čija rezolucija prikaza je manja od 992px, oni će biti prikazani jedan ispod drugog.

```
<div class="container">
    <div class="row">
        <div class="col-md-4">col-md-4</div>
        <div class="col-md-4">col-md-4</div>
        <div class="col-md-4">col-md-4</div>
    </div>
</div>
```

Kod 1 - Korištenje Bootstrap Grid sustava



Slika 4 - Primjer Bootstrap Grid sustava

3.2.1.3 Korištenje Bootstrapa u web aplikaciji

Od ostalih komponenata Bootstrapa koristio sam još klase *„glyphicon“* i *„nav“* kod kreiranja navigacije.

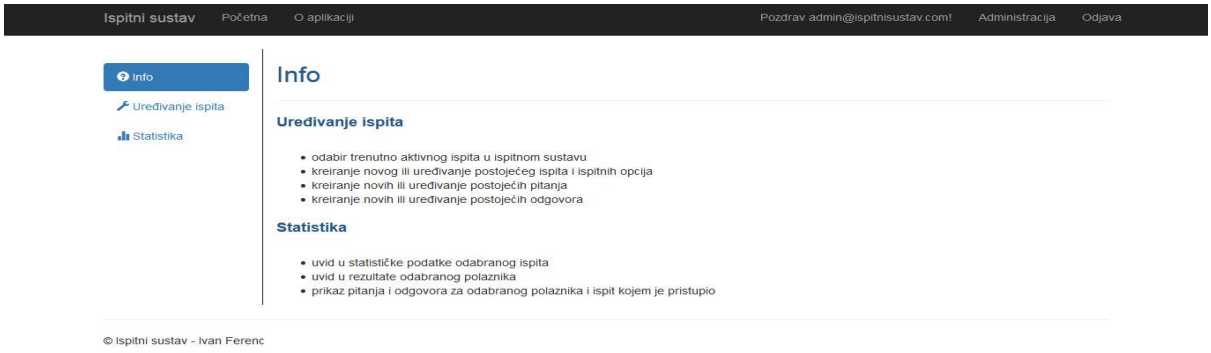
```
<ul class="nav nav-pills nav-stacked">
    <li id="nav_item_1">
        <a href="@Url.Action("Index", "Admin")">
            <span class="glyphicon glyphicon-question-sign"></span> Info
        </a>
    </li>
    <li id="nav_item_2">
        <a href="@Url.Action("Exams", "ExamManagement")">
            <span class="glyphicon glyphicon-wrench"></span> Uređivanje ispita
        </a>
    </li>
    <li id="nav_item_3">
        <a href="@Url.Action("Exams", "Statistics")">
            <span class="glyphicon glyphicon-stats"></span> Statistika
        </a>
    </li>
</ul>
```

Kod 2 - Primjer korištenja Bootstrap komponenti

Bootstrap je dobro prihvaćen od strane web programera, što je utjecalo na pojavljivanje velikog broja raznih Bootstrap predložaka. Unatoč tome što su neki predlošci besplatni, kod razvoja svoje aplikacije koristio sam standardnu bootstrap temu koja je aktivna odmah nakon kreiranja novog web projekta pomoću Visual Studia 2013.

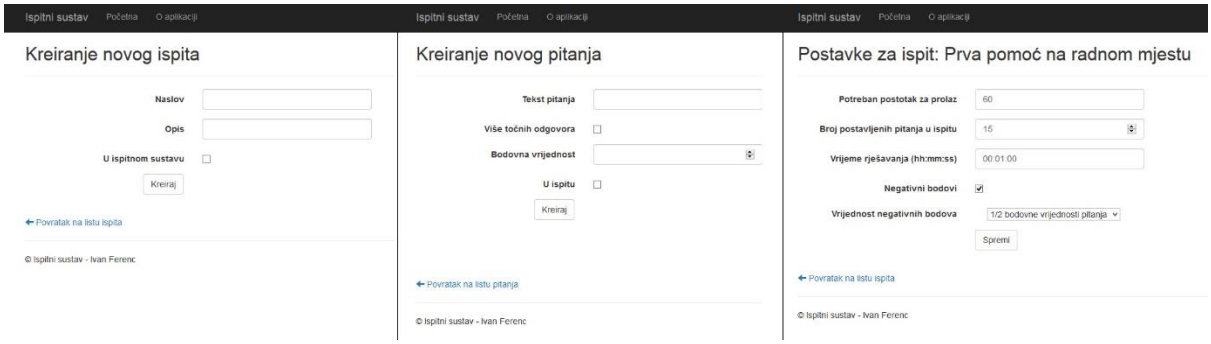
3.2.2 Administrativno korisničko sučelje – uređivanje ispita

Početnu stranicu ovog dijela korisničkog sučelja dizajnirao sam tako da prvenstveno informira administratora ispitnog sustava o mogućnostima koje su mu na raspolaganju i omogućuje laku navigaciju kroz te mogućnosti. Sljedeća slika prikazuje izgled početne stranice:

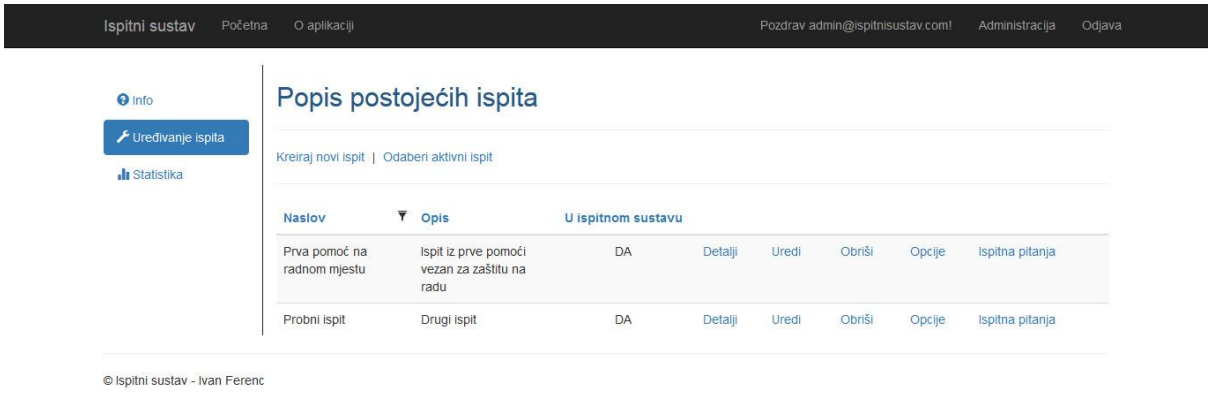


Slika 5- Administracijsko korisničko sučelje

Odabirom opcije „Uređivanje ispita“ prelazi se na stranicu koja omogućuje upravljanje ispitnim sadržajem poput kreiranja novog ispita, odabira aktivnog ispita, pregleda ili uređivanje detalja pojedinog ispita, brisanja, uređivanja ispitnih opcija ili uređivanja ispitnih pitanja, kao što je prikazano na slijedećim slikama:

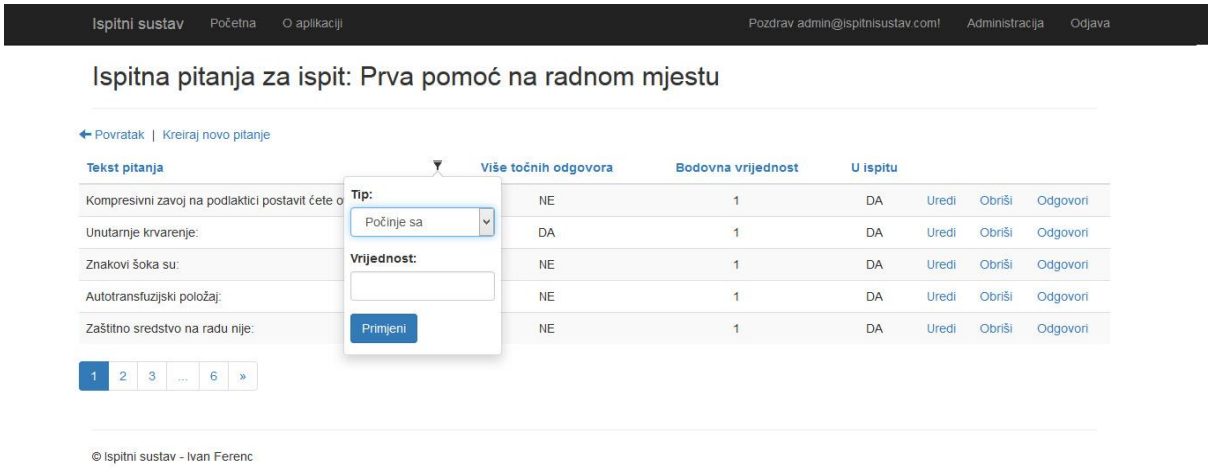


Slika 6 - Prikaz sučelja za kreiranje ispitnog sadržaja



Slika 7 - Korisničko sučelje upravljanja ispitnim sadržajem

Odabirom opcije „Ispitna pitanja“ za neki ispit sa liste prikazuje se korisničko sučelje koje omogućuje kreiranje ili uređivanje pitanja i odgovora pojedinog ispita, uz mogućnost pretraživanja liste pitanja temeljeno na tekstu pitanja.



Slika 8 - Korisničko sučelje za upravljanje ispitnim pitanjima i odgovorima

3.2.3 Grid.Mvc

Grid.Mvc se lako može dodati u postojeći projekt pomoću NuGet menadžera programskih paketa i dodaje funkcionalnost kreiranja GridView kontrola u ASP.NET MVC Web aplikaciji. To je komponenta koja omogućuje jednostavnu konstrukciju HTML tablica za prikazivanje, filtriranje i sortiranje podataka, te prelamanje njihovog prikaza preko više stranica (engl. *pagination*).

Grid.Mvc koristi jQuery kako bi omogućio filtriranje na klijentskoj strani, te Bootstrap programski okvir za grafičko uređenje prikaza.

Zbog navedenih funkcionalnosti filtriranja, sortiranja, pa i pretraživanja kolekcije podataka, odlučio sam ga koristiti prilikom izrade Web aplikacije. Prvenstveno sam htio pojednostaviti izgled grafičkog sučelja kod kreiranja novih ispitnih pitanja, te olakšati pretraživanje veće kolekcije ispitnih pitanja tako što sam omogućio sortiranje, ali i pretraživanje kolekcije prema određenoj vrijednosti. Kasnije sam se odlučio da ovakav način prikaza podataka koristim i za druge podatkovne kolekcije kao što su podaci o ispitima, odgovorima, te razni drugi statistički podaci.

Samo korištenje Grid.Mvc komponente je vrlo jednostavno, što se vidi na slijedećem primjeru:

```
@Html.Grid(Model).SetLanguage("hr").Columns(columns =>
    {
        columns.Add(c => c.QuestionText).Titled("Tekst
pitanja").Filterable(true).Css("question-style");
        columns.Add(c => c.MultipleChoices).Titled("Više točnih odgovora")
            .RenderValueAs(c =>
Html.CustomRenderingColumn(c.MultipleChoices)).Css("center-style");
        columns.Add(c => c.Points).Titled("Bodovna vrijednost").Css("center-style");
        columns.Add(c => c.IsInExam).Titled("U ispitu").RenderValueAs(c =>
Html.CustomRenderingColumn(c.IsInExam))
            .Css("center-style");
        columns.Add().Encoded(false).Sanitized(false).RenderValueAs(c =>
            @Html.ActionLink("Uredi", "EditQuestion", new { id = c.Id }));
        columns.Add().Encoded(false).Sanitized(false).RenderValueAs(c =>
            @Html.ActionLink("Obriši", "DeleteQuestion", new { id = c.Id }));
        columns.Add().Encoded(false).Sanitized(false).RenderValueAs(c =>
             @Html.ActionLink("Odgovori", "Answers", new { examId = ViewBag.ExamId, id
= c.Id }));
    }).WithPaging(5).Sortable(true)
```

Kod 3 - Primjer korištenja Grid.Mvc

Grid metoda kao parametar prima kolekciju podataka (proslijeđen podatkovni model u View sloj), te nakon toga slijedi određivanje jezika prikaza pomoću metode *SetLanguage* i definiranje stupaca. Kod kreiranja stupaca tablice pomoću metode *Columns*, potrebno je odrediti naziv stupca pomoću metode *Titled*, da li je omogućeno filtriranje vrijednosti stupca pomoću metode *Filterable* te odrediti podatak koji se prikazuje i eventualno način njegovog prikazivanja što se definira pomoću metode *RenderValueAs*. Svakom stupcu moguće je dodijeliti i zasebnu CSS klasu pomoću metode *Css*. Nakon toga se pomoću metode *WithPaging* određuje broj redaka koji se prikazuju po stranici, te pomoću metode *Sortable* da li je moguće sortirati podatke u tablici po određenom stupcu.

3.2.4 Administrativno korisničko sučelje – Statistika

Ovaj dio korisničkog sučelja omogućuje prikaz statističkih podataka ispitnog sustava. Na početnoj stranici se nalazi lista ispita sa koje je moguće odabrati pregled statističkih podataka pojedinog ispita ili prikaz polaznika koji su pristupili tom ispitu. Također je moguće odabrati prikaz liste polaznika gdje je za svakog polaznika moguće pregledati listu ispita kojima je pristupio, rezultat koji je postigao, pitanja i odgovore koji su se pojavili u ispitu.

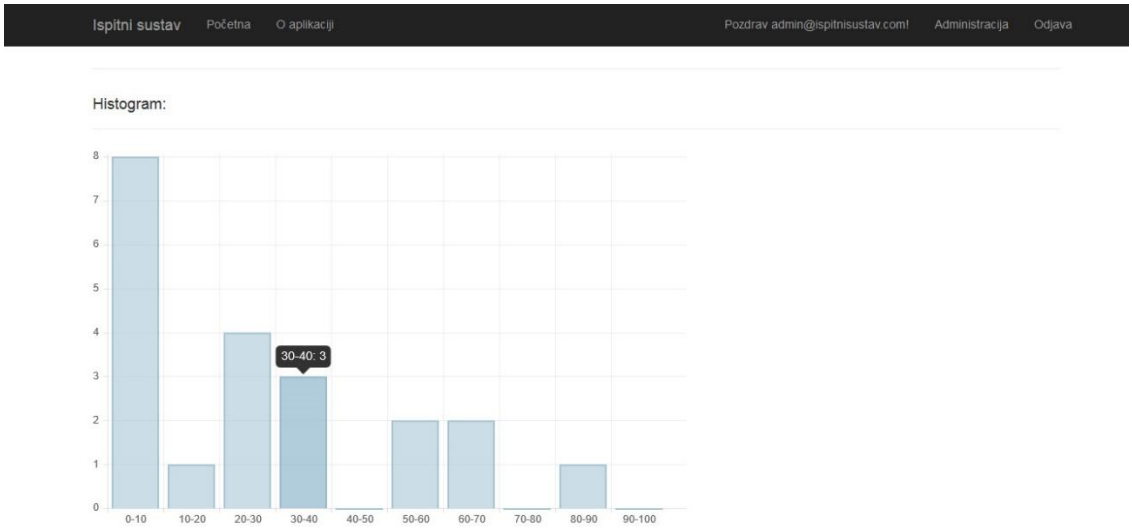
Odabirom opcije ispitne statistike prikazuju se statistički podaci vezani za prolaznost odabranog ispita. Rezultati su prezentirani u obliku nekoliko vrsta grafova:

- *Pie chart* – prikazuje omjer broja polaznika koji su položili i koji nisu položili odabrani ispit

- *Bar chart* – prikazuje broj polaznika razvrstanih u deset grupa prema ostvarenom rezultatu na ispitu
- *Line chart* – prikazuju standardnu normalnu distribuciju podataka za odabrani ispit

Na slijedećim slikama je prikazano korisničko sučelje sa nekim iscrtanim grafovima:

Slika 9 - Korisničko sučelje za prikaz statističkih podataka - Pie chart



Slika 10 - Korisničko sučelje za prikaz statističkih podataka - Bar chart

3.2.5 Chart.js

Za iscrtavanje grafova koristio sam Chart.js, JavaScript biblioteku koji omogućuje prikazivanje podataka na šest različitih načina, a za to koristi HTML5 *„canvas“* element. Pripremljeni podaci se vrlo lako mogu prikazati u obliku grafova:

3.2.5.1 Pie chart

```
var ctxPie = document.getElementById("pieChart").getContext("2d");

var pieData = [
    {
        value: failedExams,
        label: 'PALI',
        color: '#FF0000'
    },
    {
        value: data.GraphData.NumberOfPassedExams,
        label: 'PROŠLI',
        color: '#1D702F'
    }];

var pieChart = new Chart(ctxPie).Pie(pieData);
```

Kod 4 - Primjer korištenja Chart.js - Pie chart

Varijabla *„ctxPie“* predstavlja HTML5 canvas element gdje će se prikazati graf. Varijabla *„pieData“* predstavlja kolekciju podataka koju želimo prikazati i određene opcije za iscrtavanje tih podataka. Varijabla *„pieChart“* predstavlja novu instancu klase *„Chart“* koja kao parametar prima canvas element „*ctxPie*“ i pomoću metode „*Pie“* koja kao parametar prima „*pieData*“ prikazuje graf u zadanom elementu sa proslijeđenim podacima i zadanim opcijama.

3.2.5.2 Bar chart

```
var ctxBar = document.getElementById("barChart").getContext("2d");

var barData = {
        labels: ["0-10%", "10-20%", "20-30%", "30-40%", "40-50%", "50-60%", "60-70%",
"70-80%", "80-90%", "90-100%"],
        datasets: [
            {
                label: "Histogram",
                fillColor: "rgba(151,187,205,0.5)",
                strokeColor: "rgba(151,187,205,0.8)",
                highlightFill: "rgba(151,187,205,0.75)",
                highlightStroke: "rgba(151,187,205,1)",
                data: data.GraphData.HistogramData
            }
        ]
    };

var barChart = new Chart(ctxBar).Bar(barData);
```

Kod 5 - Primjer korištenja Chart.js - Bar chart

Na sličan način kao i kod prijašnje vrste grafa, podatke možemo prikazati korištenjem kontrole *Bar chart*. Razlika je jedino kod varijable „*barData*“ koja uz kolekciju podataka ima malo drugačije opcije, poput boje ispuna stupaca, boje linija obruba. Prikazuje se u odabranom elementu pomoću metode „*Bar*“.

3.2.5.3 *Line chart*

```
var ctxLine = document.getElementById("standardChart").getContext("2d");

var standardData = {
        labels: standardDistribution,
        datasets: [
            {
                label: "Dataset",
                fillColor: "rgba(151,187,205,0.2)",
                strokeColor: "rgba(151,187,205,1)",
                pointColor: "rgba(151,187,205,1)",
                pointStrokeColor: "#fff",
                pointHighlightFill: "#fff",
                pointHighlightStroke: "rgba(151,187,205,1)",
                data: density
            }
        ]
    };

var standardChart = new Chart(ctxLine).Line(standardData);
```

Kod 6 - Primjer korištenja Chart.js - Line chart

Opcije grafa se definiraju na sličan način kao i kod kontrole *Bar chart*, a u odabranom elementu se prikazuje pomoću metode „*Line*“.

3.2.6 Korisničko sučelje polaznika ispitnog sustava

Početna stranica ispitnog sustava korisnicima daje osnovne informacije o samoj aplikaciji i upute o rješavanju ispita. Da bi korisnici pristupili rješavanju ispita, moraju biti logirani u sustav. Sama prijava u sustav je moguća nakon uspješno izvršene registracije novog korisnika.

Nakon što se korisnik prijavio u sustav, ima mogućnost uređivanja svojih podataka za prijavu, pregledati rezultate ispita kojima je pristupio i pristupiti rješavanju trenutno aktivnog ispita u ispitnom sustavu.

Prije samog početka rješavanja ispita, korisniku se prikazuju ispitne opcije kako bi ga se informiralo o postotku potrebnom za prolaz ispita, broju pitanja u ispitu, dostupnom vremenu rješavanja i da li ima negativnih bodova u ispitu. Tijekom samog rješavanja ispita korisnik označava jedan ili više odgovora za koje misli da su točni. Ispit završava istekom dopuštenog vremena, ili nakon što korisnik odgovori na sva postavljena pitanja.

Prikaz pitanja i odgovora riješio sam tako da sam u samom html dokumentu definirao element prilagođenog tipa (engl. *type*) „*appliaction/html-template*“ kojeg internet preglednik neće prikazivati korisniku i dodijelio mu *id* pomoću kojeg ću ga kasnije moći dohvatiti pomoću

JavaScript-a. Unutar samog elementa su predefinirana mjesta gdje će se prikazivati informacije o samom pitanju i lista ponuđenih odgovora.

```
<script id="examTemplate" type="application/html-template">
    <div class="row margin-bottom-10">
        <div class="col-md-3 col-md-offset-1">
            TOČNIH ODGOVORA U PITANJU: {{CorrectAnswersNumber}}
        </div>
        <div class="col-md-2">PITANJE BROJ: {{QuestionNumber}}</div>
        <div class="col-md-2">UKUPNO PITANJA: {{TotalQuestions}}</div>
    </div>
    <div class="row">
        <div class="col-md-8 col-md-offset-1 bold-style blue-color">
            {{QuestionText}}
        </div>
    </div>
    <hr />
    <div class="row">
        <div class="col-md-offset-1 col-md-8">
            {{Answers}}
        </div>
    </div>
</script>
```

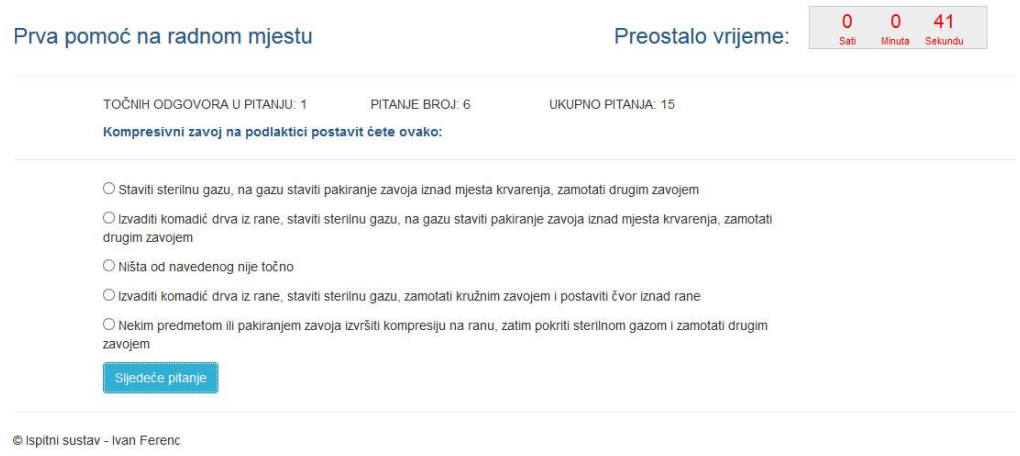
Kod 7 - Primjer definiranja predloška za prikaz pitanja i odgovora

Sami prikaz trenutnog pitanja riješio sam pomoću JavaScript-a, točnije pomoću jQuery biblioteke:

```
var template = $('#examTemplate').clone().html();
var html = template
    .replace('{{QuestionText}}', question.QuestionText)
    .replace('{{CorrectAnswersNumber}}', correctAnswersNumber)
    .replace('{{QuestionNumber}}', counter + 1)
    .replace('{{TotalQuestions}}', numberOfQuestions)
    .replace('{{Answers}}', answers.join(''));

$('#examContent').empty().html(html);
```

Kod 8 - Prikaz trenutnog pitanja i liste odgovora



Slika 11 - Korisničko sučelje prilikom rješavanja ispita

Nakon što je ispit završen, korisniku se prikazuje rezultat rješavanja ispita u obliku ostvarenog postotka.

3.2.7 jQuery Countown

Za prikaz preostalog dostupnog vremena za rješavanje ispita koristio sam jQuery dodatak (engl. *plugin*) pomoću kojeg se u *div* ili *span html* elementu prikazuje odbrojavanje preostalog vremena. Lokacija sa koje se navedena biblioteka i pripadajuća dokumentacija mogu preuzeti navedena je u popisu literature. Samo korištenje je vrlo jednostavno:

```
function startCountdown(remainingTime) {
    $('#remainingTime').countdown({ until: remainingTime,
        format: 'HMS', onExpiry: finishExam, onTick: highlightLast });
}

function highlightLast(periods) {
    if ($.countdown.periodsToSeconds(periods) === 60) {
        $('#remainingTime').addClass('highlight');
    }
}
```

Kod 9 - Primjer korištenja jQuery Countdown plugina

Metodom „*countown*“ se pokreće timer kojem se moraju odrediti neke postavke: „*until*“ određuje preostalo vrijeme, „*format*“ određuje način prikaza, „*onExpiry*“ poziva funkciju nakon što je odbrojavanje gotovo, „*onTick*“ poziva funkciju za svaki otkucaj timera (u ovom slučaju ta funkcija dodaje css klasu elementu gdje se prikazuje timer kako bi se promjenom boje prikaza upozorilo korisnika da mu je vrijeme za rješavanje ispita skoro isteklo).

3.2.8 ASP.NET Identity

Za prijavu korisnika u ispitni sustav i registraciju novih korisnika, koristio sam ASP.NET Identity sustav. Prije pojave ovog sustava, koristio se ASP.NET Membership sustav za validaciju i pohranu korisničkih podataka.

Sustavi za članstvo i autentičnost korisnika (engl. *membership and authentication systems*) su u MVC 5 radnom okviru bili u potpunosti iznova napravljeni kao dio ASP.NET Identity sustava. Novi sustav se u potpunosti maknuo od ograničenja starih sustava poput na primjer mogućnosti prijave u sustav samo sa registriranim korisničkim imenom i lozinkom. U današnje vrijeme ljudi često komuniciraju preko društvenih mreža kao što su Facebook, Twitter, pa su razvojni programeri htjeli omogućiti prijavu u sustav baziranu na preusmjeravanju prema vanjskim sustavima za provjeru autentičnosti kao što su na primjer Facebook, Twitter.

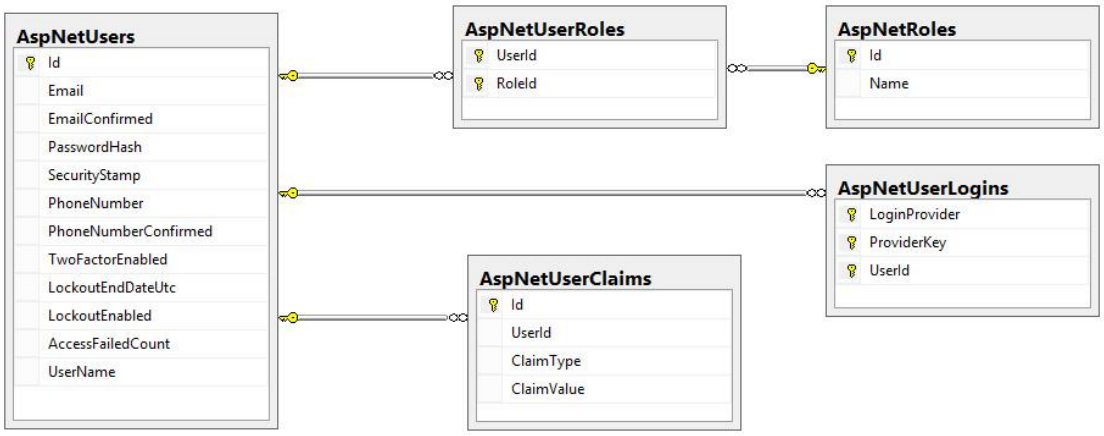
Uzimajući u obzir sve promjene nastale u razvoju web aplikacija, ASP.NET Identity je nastao temeljen na slijedećim ciljevima:

- One ASP.NET Identity sustav
    - ASP.NET Identity se može koristiti sa svim ASP.NET radnim okvirima, kao što su ASP.NET MVC, Web Forms, Web Pages, Web API i SignalR
    - ASP.NET Identity se može koristiti u razvoju web, mobilnih, desktop i hibridnih aplikacija
- lakoća dodavanja korisničkih podataka
    - kontrola nad schemom modela Korisnik (engl. *User*) i informacijama korisničkog profila omogućuje lako dodavanje dodatnih informacija kao što su na primjer datum rođenja, informacije o društvenim mrežama, adresa stanovanja, itd.
- Unit testiranje (engl. *testing*)
- upravljanje rolama (engl. *roles*)
    - moguće je omogućiti pristup pojedinim dijelovima aplikacije samo određenim rolama
    - omogućeno kreiranje rola i dodavanje korisnika u njih
- baziran na tvrdnjama (engl. *Claims based*)
    - ASP.NET Identity podržava provjeru autentičnosti baziranu na tvrdnjama, gdje je korisnički identitet predstavljen grupom tvrdnji što omogućuje mnogo informacija o korisničkom identitetu i članstvu
- prijava u sustav pomoću društvenih mreža
    - vrlo je lako dodati vanjske račune za prijavu kao što su Microsoft Account, Facebook, Twitter, Google, itd.
- OWIN[7] integracija
    - ASP.NET provjera autentičnosti se temelji na OWIN međuprogramu (engl. *middleware*) i nema nikakve ovisnosti prema *System.Web* imenskom prostoru (klase koje omogućuju komunikaciju između Internet preglednika i servera)
    - Za prijavu i odjavu u sustav aplikacija koristi OWIN *CookieAuthentication*
- NuGet programski paket

[7] OWIN - Open Web Interface for .NET

- ASP.NET Identity dolazi u obliku NuGet programskog paketa koji je instaliran u ASP.NET MVC, Web Forms i Web API predloške koji se nalaze unutar Visual Studia 2013

Na slijedećoj slici je prikaz ASP.NET Identity tablica u bazi podataka:



Slika 12 - Entiteti ASP.NET Identity

3.2.8.1 Konfiguriranje ASP.NET Identity u web aplikaciji Ispitnog sustava

ASP.NET Identity je spreman za korištenje odmah nakon kreiranja novog ASP.NET Web Application projekta unutar Visual Studia 2013. Za web aplikaciju Ispitnog sustava dodatno sam podesio validaciju lozinki tako da minimalna duljina bude 8 znakova i da lozinka sadržava barem jedan broj.

```
public static ApplicationUserManager
Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)
        {

            // Configure validation logic for passwords
            manager.PasswordValidator = new
IspitniSustav.CustomValidators.PasswordValidator
            {
                RequiredLength = 8,
                RequireNonLetterOrDigit = false,
                RequireDigit = true,
                RequireLowercase = false,
                RequireUppercase = false
            };


            return manager;
        }
    }
```

Kod 10 - Primjer podešavanja postavki za prilagođenu validaciju lozinki

Kod registracije novih korisnika, registracijski podatkovni model korisnika sam proširio sa dodatnim poljima, pa korisnik uz email adresu i lozinku, mora unijeti svoje ime i prezime.

Nadalje, omogućio sam prijavu u sustav pomoću vanjskih računa društvenih mreža Google i Facebook.

Za omogućavanje prijave pomoću Google računa, potrebno je u web aplikaciji unutar *Startup.Auth.cs* klase konfigurirati *ClientId* i *ClientSecret* vrijednosti.

```
app.UseGoogleAuthentication(new GoogleOAuth2AuthenticationOptions()
            {
                ClientId = "908241811824-
aceog4b24kpd2v3bt9i82mr0ngh4j8ch.apps.googleusercontent.com",
                ClientSecret = "ycLkFeDnl_Bjd6-exDGyebkn",
                CallbackPath = new PathString("/oauth2callback")
            });
```

Kod 11 - Primjer konfiguriranja prijave pomoću Google računa

Te vrijednosti moguće je dobiti na slijedećoj adresi: https://console.developers.google.com. Nakon prijave (ili eventualne registracije novog korisnika) pomoću postojećeg Google računa, prijavljeni korisnik ima između ostalog mogućnost kreiranja novog projekta ili uređenje nekog od postojećih projekata.

Nakon što sam se prijavio pomoću svojih korisničkih podataka, prvo sam kreirao novi projekt te otišao na podešavanje opcija *APIs & auth*. Pod opcijom *APIs* pronašao sam *Google+ API* i aktivirao ga (engl. *enable*). Zatim sam otišao na opciju *Credentials*. Tamo sam odabrao *Add new credentials* vezane za *OAuth 2.0 client ID*. Za tip aplikacije sam odabrao *Web application* te unio njenu URL[8] adresu. Nakon toga se pritiskom na gumb *Create* generira *ClientId* i *ClientSecret* vrijednosti koje sam iskoristio za konfiguriranje Google autentikacije u web aplikaciji Ispitnog sustava unutar klase *Startup.Auth.cs*.

Za omogućavanje prijave pomoću Facebook korisničkog računa, također je potrebno u web aplikaciji unutar *Startup.Auth.cs* klase konfigurirati *ClientId* i *ClientSecret* vrijednosti. Te vrijednosti moguće je dobiti na slijedećoj adresi: https://developers.facebook.com. Također je potrebno obaviti prijavu pomoću postojećeg Facebook računa ili registraciju novog korisničkog računa.

[8] URL – Uniform Resource Locator

Nakon što sam se prijavio pomoću svojih korisničkih podataka, odabrao sam opciju *My Apps* te nakon toga *Add a New App*. Kao platformu svoje aplikacije odabrao sam *Website* te nakon unosa naziva aplikacije kliknuo na *Create New Facebook App ID*. Nakon unosa URL adrese same web aplikacije Ispitnog sustava, moguće je odabirom nove dodane aplikacije iz *My Apps* izbornika saznati informacije vezane za *ClientId* i *ClientSecret*. Te vrijednosti sam iskoristio za konfiguriranje Facebook autentikacije u web aplikaciji Ispitnog sustava unutar klase *Startup.Auth.cs*.

```
var facebookAuth = new FacebookAuthenticationOptions();
            facebookAuth.Scope.Add("email");
            facebookAuth.AppId = "606976426113204";
            facebookAuth.AppSecret = "e6823a31e7a0a8ccaa07655d90c8764d";
            facebookAuth.Provider = new FacebookAuthenticationProvider()
            {
                OnAuthenticated = (context) =>
                {
                    context.Identity.AddClaim(new
System.Security.Claims.Claim("FacebookAccessToken", context.AccessToken));
                    foreach (var claim in context.User)
                    {
                        var claimType = string.Format("urn:facebook:{0}", claim.Key);
                        var claimValue = claim.Value.ToString();
                        if (!context.Identity.HasClaim(claimType, claimValue))
                            context.Identity.AddClaim(new
System.Security.Claims.Claim(claimType, claimValue, "XmlSchemaString", "Facebook"));

                    }
                    return Task.FromResult(0);
                }
            };
```

Kod 12 - Primjer konfiguriranja prijave pomoću Facebooka

3.2.8.2 OAuth 2.0

OAuth 2 je autorizacijski radni okvir koji omogućuje aplikacijama da dobe djelomičan pristup korisničkim računima na HTTP servisu kao što je na primjer Facebook. Radi tako da delegira provjeru autentičnosti korisnika servisu koji sadrži korisničke podatke i daje autorizaciju aplikaciji da pristupi korisničkim podacima.

OAuth definira četiri role:

- vlasnik resursa (engl. *resource owner*)
    - vlasnik resursa je korisnik koji autorizira aplikaciju da pristupi njegovom korisničkom računu
- poslužitelj resursa (engl. *resource server*)
    - server koji sadrži korisničke račune

- autorizacijski poslužitelj (engl. *authorization server*)
    - provjerava identitet korisnika te nakon toga izdaje tokene aplikaciji
- klijent (engl. *client*)
    - klijent je aplikacija koja želi pristup korisničkom računu, ali prvo mora biti autorizirana od strane korisnika, a ta autorizacija mora biti provjerena od strane autorizacijskog poslužitelja

Tijek interakcije između rola:

- aplikacija traži zahtjev za autorizacijom da pristupi servisu koji sadrži korisničke podatke
- ako korisnik odobri zahtjev za autorizacijom, aplikacija dobiva autorizacijsku dozvolu
- aplikacija šalje zahtjev za pristupnim tokenom autorizacijskom poslužitelju na temelju autentikacije vlastitog identiteta i autorizacijske dozvole
- ako je identitet aplikacije autentičan i ako je autorizacijska dozvola valjana, autorizacijski poslužitelj izdaje pristupni token aplikaciji i time je autorizacija završena
- aplikacija šalje zahtjev za resursom serveru koji sadrži resurse prezentirajući pristupni token za provjeru autentičnosti
- ako je pristupni token valjan, poslužitelj resursa šalje tražene resurse aplikaciji

3.3 Controller sloj Ispitnog sustava

Nakon kreiranja novog ASP.NET Web Application projekta u Visual Studiu 2013, automatski se kreiraju i određeni kontroleri, kao što su *HomeController*, *AccountController* i *ManageController*. *AccountController* i *ManageController* upravljaju poslovima vezanima za kreiranje i uređivanje korisničkih računa te prijavom korisnika u sustav, dok je *HomeController* zadužen za upravljanje poslovima vezanima za prikaz informacija na početnoj stranici web aplikacije Ispitnog sustava.

Za upravljanje drugim poslovima unutar aplikacije, kreirao sam vlastite kontrolere:

- *ExamController*
    - upravlja pokretanjem ispita i prikazivanjem rezultata polazniku ispita
- *AdminController*
    - upravlja prikazivanjem informacija na administracijskom korisničkom sučelju

- *ExamManagementController*
    - upravlja poslovima vezanima za ispite u Ispitnom sustavu
- *QuestionManagementController*
    - upravlja poslovima vezanima za pitanja i odgovore ispita
- *StatisticsController*
    - upravlja poslovima vezanima za prikaz statističkih podataka

3.3.1 Autorizacija

Određene dijelove web aplikacije sam želio zaštititi, to jest ograničiti im pristup. Polaganju ispita mogu pristupiti samo registrirani i prijavljeni korisnici, dok su je upravljanje ispitnim sadržajem i uvid u statističke podatke moguć uz administracijske ovlasti.

Ograničenje pristupa rješavanju ispita sam riješio tako da sam kontroleru *ExamController* koji upravlja poslovima vezanima za rješavanje ispita dodijelio atribut *Authorize*.

Ograničenje pristupa dijelovima aplikacije koji upravljaju ispitnim sadržajem i imaju uvid u statističke podatke riješio sam tako da sam atribut *Authorize* sa svojstvom „*Roles*“ dodijelio kontrolerima *AdminController*, *ExamManagementController*, *StatisticsController* i *QuestionManagementController*, što znači da samo korisnici kojima su dodijeljena određena prava imaju pravo pristupa.

ASP.NET MVC Framework omogućuje vrlo lagan način dodavanja dodatne logike obrade u životni ciklus MVC zahtjeva – odgovora (engl. *request response*). To se radi sa dodavanjem određenih atributa kontrolerima ili njihovim pojedinim metodama. Dodavanjem atributa „*Authorize“* na kontroler ili neku njegovu metodu, svaki puta kada se pozove neka metoda sa tim atributom, poziva se klasa *Attribute* koja implementira autorizacijski filter (engl. *authorization filter*) te se provjerava da li korisnik ima potrebna autorizacijska prava da pristupi traženim dijelovima i resursima aplikacije.

```
[Authorize(Roles = "Admin")]
    public class AdminController : Controller
    {
        //show administration user interface
        public ActionResult Index()
        {
            return View();
        }
    }
```

Kod 13 - Primjer korištenja autorizacijskog atributa

3.3.2 Usmjeravanje pomoću atributa (engl. *Attribute Routing*)

U ASP.NET MVC Frameworku „usmjeravanje“ označava način uparivanja traženog URI[9] sa odgovarajućom metodom kontrolera. MVC 5 verzija frameworka, uz stari konvencionalni način usmjeravanja, omogućuje i usmjeravanje pomoću atributa.

Prilikom izrade web aplikacije Ispitnog sustava, koristio sam oba načina definiranja ruta. U projektu, rute se definiraju u klasi *RouteConfig.cs*, a ako želimo koristiti usmjeravanje pomoću atributa, moramo ga omogućiti unutar ove klase.

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.MapMvcAttributeRoutes();
    }
}
```

Kod 14 - Primjer omogućavanja usmjeravanja pomoću atributa

Sam način definiranja nove rute pomoću atributa je prilično jednostavan, što se može vidjeti na slijedećem primjeru:

```
[Route("QuestionManagement/EditAnswer/{examId?}/{questionId?}/{id?}")]
public ActionResult EditAnswer(int? examId, int? questionId, int? id)
{
}
```

Kod 15 - Primjer definiranja nove rute pomoću atributa

3.4 Model sloj Ispitnog sustava

Ovaj sloj web aplikacije prvenstveno čine klase vezane za podatkovni model definiran u bazi podataka. Klase sa sufiksom *ViewModel* predstavljaju klase koje su prilagođene za prikaz određenih podataka korisnicima. Klase sa sufiksom *Action* predstavljaju klase koje sadrže rezultat uspješnosti upravljanja nad poslovnim modelima podataka.

[9] URI – Uniform Resource Identifier

Klase koje sadrže poslovnu logiku i metode za upravljanje podatkovnim modelom podijelio sam na tri dijela:

- klasa *ExamManager.cs*
- klasa *StatisticsManager.cs*
- klasa *Repository.cs*

3.4.1 Klasa ExamManager

Ova klasa obavlja slijedeće zadaće:

- priprema ispitnih pitanja i odgovora – „*random*“ poredak pojavljivanja pitanja u ispitu i „*random*“ poredak odgovora unutar svakog pitanja
- izračun ispitnog rezultata na temelju trenutnih ispitnih opcija
- slanje rezultata ispita polazniku putem elektroničke pošte

3.4.1.1 iTextSharp

Za generiranje PDF[10] dokumenta koji sadrži ispitne rezultate i koji se putem elektroničke pošte dostavlja polazniku ispita, koristio sam „*iTextSharp .NET PDF library*“. Ova biblioteka omogućuje kreiranje, uređivanje i pregledavanje PDF dokumenata. Primjer izgleda krajnjeg rezultata:

Slika 13 - Primjer izgleda generiranog PDF dokumenta

[10] PDF – Portable Document Format

3.4.2 Klasa StatisticsManager

Ova klasa obavlja slijedeće zadatke:

- izračun podataka za iscrtavanje grafa standardne distribucije
- izračun podataka za iscrtavanje grafa histograma

Kod prikazivanja podataka pomoću histograma, za što je korišten bar chart, ispitne rezultate sam podijelio u deset razreda u intervalu od 0 – 100%, te mjerio frekvenciju pojavljivanja rezultata u određenom razredu.

```
internal static int[] GetHistogramData(List<double> scores)
{
    bucketNumber = (max - min)/interval;
    histogramData = new int[bucketNumber];
    // increase initial value of each bucket(default is 0)
    for (int i = 0; i < scores.Count; i++)
    {
        if ((int) scores[i] != 100)
        {
            histogramData[(int) scores[i]/interval]++;
        }
        else
        {
            histogramData[interval - 1] ++;
        }
    }
    return histogramData;
}
```

Kod 16 - Primjer izračuna podataka za histogram

Kod izračuna podataka za standardnu distribuciju, prvo sam izračunao standardnu devijaciju.

```
private static void StandardDeviation(List<double> data)
{
    standardDeviation = 0;

    if (data.Count > 0)
    {
        average = data.Average();
        var sum = data.Sum(d => Math.Pow(d - average, 2));

        standardDeviation = Math.Sqrt(sum / (data.Count - 1));
     }
 }
```

Kod 17 - Primjer izračuna standardne devijacije

Z vrijednost koja pokazuje koliko neka vrijednost ima pomak od srednje vrijednosti, izračunao sam na slijedeći način:

```
public static double Z(double score)
{
    if (standardDeviation.Equals(0)) return 0;

    return Math.Round( ((score - average) / standardDeviation), 2);
}
```

Kod 18 - Primjer izračuna Z vrijednosti

Gustoću standardne normalne razdiobe za danu vrijednost računao sam na slijedeći način:

```
public static double StandardNormalPdf(double x)
{
    var exponent = -1 * (0.5 * Math.Pow(x, 2));
    var numerator = Math.Pow(Math.E, exponent);
    var denominator = Math.Sqrt(2 * Math.PI);

    return Math.Round((numerator / denominator), 2);
}
```

Kod 19 - Primjer izračuna gustoće standardne normalne razdiobe

3.4.3 Klasa Repository

Ova klasa je zadužena za sve poslove vezane za dohvaćanje, kreiranje, ažuriranje i brisanje podataka iz baze podataka. Za pristup bazi podataka koristio sam programski okvir *Entity Framework*.

3.4.3.1 Entity Framework

Prilikom izrade Ispitnog sustava koristio sam Entity Framework 6. Entity Framework je ORM[11] programski okvir koji omogućuje automatiziran način spremanja podataka poslovnog modela u relacijsku bazu podataka. Automatizira CRUD operacije nad podacima te time olakšava posao programerima koji ne moraju sami implementirati te operacije.

Kod pristupa radu sa Entity Frameworkom odlučio sam se za pristup „prvo kodiranje sa postojećom bazom podataka“ (engl. *Code First from database*) koji je na temelju entiteta u relacijskoj bazi podataka generirao klase modela u web aplikaciji.

Nakon kreiranja klasa modela, u samoj web aplikaciji su postojala dva različita konteksta podataka koji su trebali biti pohranjeni u jednoj bazi podataka: poslovni model podataka Ispitnog sustava i podaci o korisničkim računima vezani uz ASP.NET Identity.

Ova verzija Entity Frameworka omogućuje rad sa više različitih konteksti unutar jedne baze podataka. To je moguće izvesti korištenjem migracijskih naredbi. Migracije omogućuju kreiranje početne baze podataka, umetanje početnih vrijednosti u bazu podataka i promjene

[11] ORM – Object Relational Mapping

scheme baze. Migracijske naredbe se mogu proširiti sa dva svojstva: *ContextTypeName* i *MigrationsDirectory*, što omogućuje korištenje više podatkovnih konteksti u istoj bazi.

U web aplikaciji Ispitnog sustava kontekstima podataka sam dodijelio imena „*IdentityDb*“ i „*ExamSystemDB*“. Koristeći konzolu NuGet menadžera programskih paketa, sa slijedećim naredbama sam omogućio pohranu različitih konteksti podataka u istu bazu:

```
//omogućavanje migracija za oba konteksta

enable-migrations -ContextTypeName IdentityDb -MigrationsDirectory
DataContexts\IdentityMigrations

enable-migrations -ContextTypeName ExamSystemDB -MigrationsDirectory
DataContexts\ExamSystemMigrations

//kreiranje inicijalnih migracija za oba konteksta

add-migration -ConfigurationTypeName
IspitniSustav.DataContexts.IdentityMigrations.Configuration "InitialCreate"

add-migration -ConfigurationTypeName
IspitniSustav.DataContexts.ExamSystemMigrations.Configuration "InitialCreate"


//dodavanje migracijskog sadržaja u bazu

update-database -ConfigurationTypeName
IspitniSustav.DataContexts.IdentityMigrations.Configuration

update-database -ConfigurationTypeName
IspitniSustav.DataContexts.ExamSystemMigrations.Configuration
```

Kod 20 - Entity Framework - kreiranje inicijalnih migracija za različite podatkovne kontekste

Korištenjem migracija prilikom izrade web aplikacije, na jednostavan sam način, nakon promjene podatkovnih modela, mijenjao scheme tablica u relacijskoj bazi podataka.

4 Web servis

Web servis je komponenta Ispitnog sustava čiji zadatak je razmjena podataka sa bazom podataka i sa mobilnom aplikacijom. Izradio sam ga korištenjem ASP.NET Web API Frameworka.

4.1 ASP.NET Web API

ASP.NET MVC je bio dizajniran za stvaranje web stranica. Osnovna smjernica kod dizajniranja i razvijanja MVC radnog okvira je bila ta da web aplikacija odgovara na zahtjeve koji se šalju putem internet preglednika, a kao odgovor vraća HTML. No, MVC radni okvir je omogućio vrlo laganu kontrolu tipa odgovora, što je bilo korisno kod stvaranja servisnog sloja. Programeri su vrlo brzo shvatili da koristeći MVC radni okvir mogu kreirati web servise koji

vraćaju XML, JSON[12] (objašnjeno kasnije) i druge formate koji nisu HTML, što im je bilo jednostavnije nego korištenje drugih radnih okvira namijenjenih stvaranju web servisa, kao na primjer WCF[13].

Dolaskom MVC 4 verzije programskog okvira, pojavilo se bolje rješenje za kreiranje web servisa: ASP.NET Web API, programski okvir koji je programerima omogućio stvaranje HTTP servisa pomoću razvojnog stila već viđenog u ASP.NET MVC programskom okviru. To je omogućeno modificiranjem nekih ASP.NET MVC značajki da odgovaraju HTTP servisima i kreiranjem nekih novih značajki.

Neke Web API značajke koje su slične MVC značajkama:

- usmjeravanje (engl. *Routing*)
    - ASP.NET Web API koristi isti sustav usmjeravanja (engl. *routing system*) za preslikavanje URL u akciju kontrolera
    - kontekstualizira usmjeravanje prema HTTP servisima preslikavanjem HTTP metode (engl. *verbs, methods*) u pripadajuće akcije kontrolera, što slijedi RESTful dizajn servisa (servis implementira HTTP metode: *GET, PUT, POST, DELETE*)
- preslikavanje modela i validacija
    - kao što MVC programski okvir pojednostavljuje proces preslikavanja unosnih vrijednosti iz polja za unos, URL parametara, itd. u vrijednosti modela, tako i Web API automatski preslikava vrijednosti iz HTTP zahtjeva u vrijednosti modela
    - sustav preslikavanja uključuje validaciju temeljenu na atributima
- filteri
    - Web API pruža podršku korištenju nekim standardnim MVC filterima, kao što je na primjer „*Authorize"* atribut, optimiziran za web servis
- kreiranje Web API kontrolera pomoću postojećih predložaka

[12] JSON – JavaScript Object Notation
[13] WCF – Windows Communication Foundation

•

Web API je također dodao nekoliko koncepti i značajki specifičnih za razvoj HTTP web servisa:

- HTTP programski modul
    - Web API je optimiziran za rad sa HTTP zahtjevima i odgovorima
    - postoji strogo definiran model HTTP objekta
    - HTTP statusni kodovi i HTTP zaglavlje su lako dostupni
- poziv akcija kontrolera baziran na HTTP metodi
    - na primjer, GET zahtjev bi automatski bio preusmjeren prema akciji kontrolera naziva „*GetNekoIme“*
- Dogovaranje sadržaja (engl. *content negotiation*)
    - HTTP ima sustav za dogovaranje sadržaja pomoću kojeg internet preglednici i drugi HTTP klijenti indiciraju postavke formata odgovora, na temelju čega server šalje odgovor u preferiranom formatu
    - to znači da odgovori mogu biti tipa XML formata, JSON formata, ili nekog drugog formata koji najviše odgovara preferencijama klijenta koji je uputio zahtjev
    - omogućeno je dodavanje podrške za nove formate bez mijenjanja programskog koda unutar samog kontrolera
- konfiguracija servisa bazirana na kodu
    - za razliku od na primjer web servisa izrađenog pomoću WCF programskog okvira čija konfiguracija je kompleksna i nalazi se u konfiguracijskoj datoteci, Web API se u potpunosti konfigurira pomoću kôda

-

4.2 Korištenje Web API u Ispitnom sustavu

Postavke web servisa nalaze se u klasi *WebApiConfig.cs*.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

        //remove xml formatter so json formatter will be default
        var appXmlType =
config.Formatters.XmlFormatter.SupportedMediaTypes.FirstOrDefault(t => t.MediaType ==
"application/xml");
        config.Formatters.XmlFormatter.SupportedMediaTypes.Remove(appXmlType);

        // WebAPI when dealing with JSON & JavaScript!
        // Setup json serialization to serialize classes to camel (std. Json format)
        var formatter = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
        formatter.SerializerSettings.ContractResolver =
            new
Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

    }
}
```

Kod 21 - Primjer postavka web servisa

Uz definiranje standardnog načina preusmjeravanja, dodao sam još opciju koja određuje JSON kao standardni format odgovora.

Modelima podataka vezanima za web servis dodao sam sufiks *ServiceModel*:

- *ExamDataServiceModel*
  - model podataka vezan za sam ispit, ispitna pitanja i odgovore, ispitne opcije
- *ExamProcessedResultServiceModel*
  - model podataka vezan za rezultat polaganog ispita
- *LoginDataServiceModel*
  - model podataka vezan za korisničke podatke za prijavu u sustav
- *LoginResultServiceModel*
  - model podataka vezan za rezultat uspješnosti prijave korisnika u sustav
- *RegistrationDataServiceModel*
  - model podataka vezan za korisničke podatke za registraciju u sustav

- *RegistrationResultServiceModel*
    - model podataka vezan za rezultat uspješnosti registracije novog korisnika u sustav
- *TakenExamDataServiceModel*
    - model podataka vezan za ispite koje je korisnik polagao

Sam web servis implementiran je pomoću slijedećih kontrolera od kojih svaki nasljeđuje *ApiController*:

- *LoginServiceController*
    - podatke iz tijela zahtjeva preslikava u *LoginDataServiceModel*
    - na temelju tih podataka slijedi pokušaj prijave korisnika pomoću ASP.NET Identity sustava
    - vraća model podataka *LoginResultServiceModel*
- *RegistrationServiceController*
    - podatke iz tijela zahtjeva preslikava u *RegistrationDataServiceModel*
    - na temelju tih podataka slijedi pokušaj registracije novog korisnika pomoću ASP.NET Identity sustava
    - vraća model podataka *RegistrationResultServiceModel*
- *ExamServiceController*
    - iz tijela zahtjeva uzima korisničku email adresu na temelju koje se provjerava da li je korisnik registriran u Ispitnom sustavu
    - dohvaća podatke o aktivnom ispitu iz baze podataka i preslikava ih *ExamDataServiceModel*
    - vraća model podataka *ExamDataServiceModel*
- *ExamResultServiceController*
    - podatke iz tijela zahtjeva preslikava u *ExamServiceModel*
    - *ExamManager* obrađuje primljene ispitne podatke
    - ispitni rezultat se preslikava u ExamProcessedResultServiceModel model podataka koji se vraća u odgovoru
- *TakenExamsServiceController*
    - iz tijela zahtjeva uzima korisničku email adresu na temelju koje se iz baze podataka dohvaćaju svi ispiti koje je polaznik polagao
    - rezultat se preslikava u TakenExamDataServiceModel model podataka koji se vraća u odgovoru

5 Android mobilna aplikacija

Mobilna aplikacija Ispitnog sustava korisnicima omogućuje prijavu i registraciju u Ispitni sustav, te polaganje ispita i uvid pristupljenim ispitima. Sa bazom podataka ispitnog sustava komunicira pomoću web servisa, a njen dizajn se sastoji od tri sloja:

- sloj prezentacije
- sloj podataka
- sloj aplikacijske logike
- lokalna baza podataka

5.1 Komunikacija između slojeva – Otto library

Otto biblioteka služi za implementaciju komunikacijskog kanala, sabirnice. Omogućuje komunikaciju između razdvojenih dijelova aplikacije. Komunikacija je pokretana događajima (engl. *event*) gdje jedna strana objavljuje (engl. *publish*) da je došlo do nekog događaja, a druga strana se pretplatila (engl. *subscribe*) na taj događaj i reagira na objavu tog događaja.

5.1.1 Implementacija komunikacijskog kanala

Implementacija komunikacijskog kanala sadržana je u klasi *BusProvider.java*.

```
public final class BusProvider {

	//communication channel
	private final static Bus BUS = new Bus();

	private BusProvider() {}

	//get communication channel
	public static Bus getInstance() {
		return BUS;
	}
}
```

Kod 22 - Implementacija komunikacijskog kanala

```
public class Global extends Application {

	private Bus bus = BusProvider.getInstance();

	@Override
	public void onCreate() {

		//on application start, initialize services
		userManagementService = new UserManagementService(bus);
		//register services on communication channel
		bus.register(userManagementService);

	}

}
```

Kod 23 - Dohvaćanje instance komunikacijskog kanala i registracija na njega

Kod pokretanja aplikacije dohvaća se instanca komunikacijskog kanala na kojeg se registriraju pojedini dijelovi aplikacije. Također, kod pokretanja pojedinih aktivnosti (engl. *activity*) koje imaju svoj životni ciklus (engl. *lifecycle*) i komuniciraju putem komunikacijskog kanala, dolazi do registracije na taj kanal.

```
@Override
protected void onResume() {

	super.onResume();

	//register this activity on communication channel
	getBus().register(this);
}

@Override
protected void onPause() {

	super.onPause();

	//unregister from communication channel
	getBus().unregister(this);
}

private Bus getBus() {
	if (bus == null) {
		bus = BusProvider.getInstance();
	}
	return bus;
}
```

Kod 24 - Primjer registracije Android Activity na komunikacijski kanal

Sama komunikacija odvija se na način da jedna strana objavi određeni događaj preko komunikacijskog kanala (engl. *publish event*), dok se druga strana treba pretplatiti na taj događaj.

```
//send request trough communication channel to ExamManagementService for
requesting questions from web service
getBus().post(new GetExamEvent());
```

Kod 25 - Primjer objave događaja preko komunikacijskog kanala

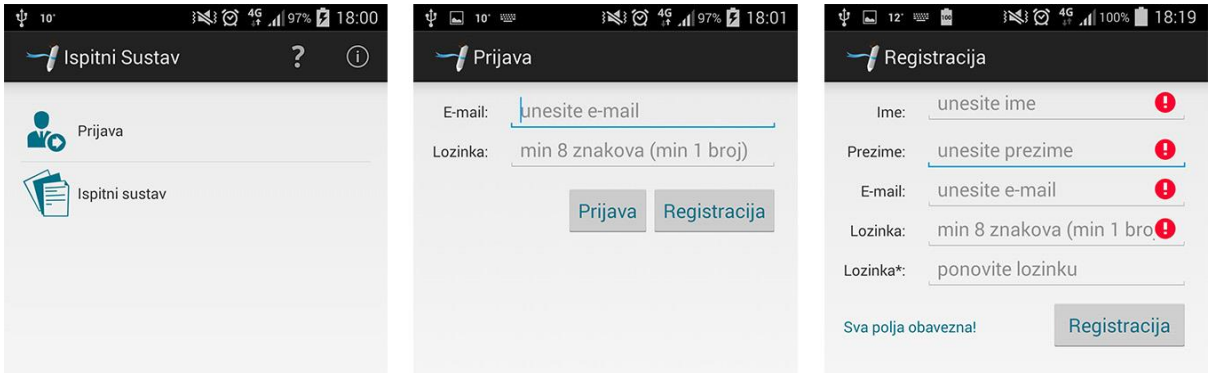
```
@Subscribe
public void onGetExamEvent(final GetExamEvent event) {


	//do something…
}
```

Kod 26 - Primjer pretplate na određeni događaj

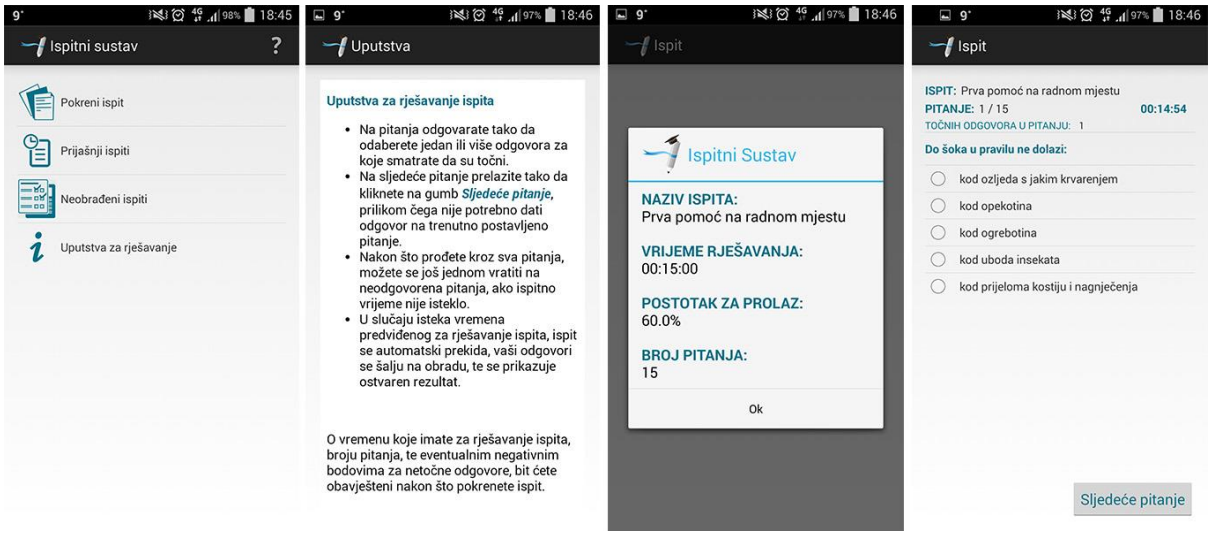
5.2 Sloj prezentacije

Zadaća ovog sloja je prezentacija informacija korisnicima mobilne aplikacije i komunikacija sa njima. Glavni izbornik nudi mogućnost prijave, to jest registracije u sustav, te pristup u ispitni sustav. Ispitnom sustavu mogu pristupiti samo prijavljeni korisnici, pa ako korisnik nije prijavljen u sustav, nudi mu se mogućnost prijave ili registracije novog korisnika.



Slika 14 - Prikaz glavnog izbornika mobilne aplikacije i korisničkog sučelja za prijavu i registraciju

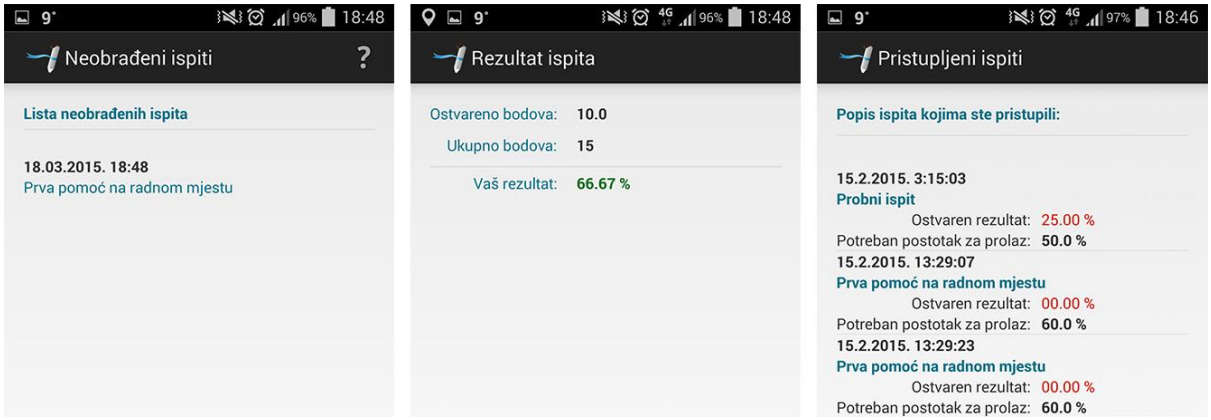
Sam Ispitni sustav korisniku nudi opcije pokretanja ispita, upute za rješavanje ispita, prikaz rezultata ispita kojima je korisnik već pristupio i listu neobrađenih ispita koja prikazuje ispite koji se nisu uspjeli poslati na obradu. Nakon pokretanja ispita, korisnik dobiva prikaz ispitnih opcija, nakon čega slijedi rješavanje ispita i prikaz rezultata.



Slika 15 - Prikaz izbornika Ispitnog sustava, uputstva rješavanja ispita, ispitnih opcija, ispita

Lista neobrađenih ispita prikazuje popis ispita za koje nisu dobiveni rezultati. U tom slučaju, podaci o ispitu se spremaju u lokalnu bazu podataka, te je korisniku omogućena opcija ponovnog slanja ispita na obradu. Sami rezultat ispita prikazuje osnovne informacije o broju

ostvarenih bodova i postotku uspješnosti, dok lista ispita kojima je korisnik pristupio prikazuje popis ispita sa ostvarenim rezultatima poredanih po datumu polaganja.



Slika 16 - Primjer korisničkog sučelja za prikaz neobrađenih ispita, rezultata ispita i liste pristupljenih ispita

5.3 Sloj podataka

Ovaj sloj predstavlja modele podataka koji služe kao prikaz informacija korisniku. Sami modeli su već opisani u poglavljima u kojima se govorilo o bazi podataka i web servisu. Podaci primljeni od web servisa se iz JSON formata konvertiraju u određene modele podataka, a određeni modeli podataka se konvertiraju u JSON format prije slanja web servisu.

5.3.1 JSON format

JSON format je lagan format za razmjenu podataka. S jedne strane je jednostavan za čitanje i pisanje ljudima, a s druge strane je jednostavan za generiranje i parsiranje strojevima. Baziran je na podskupu JavaScript programskog jezika. JSON je tekstualni format koji nije ovisan niti o jednom programskom jeziku, no koristi konvencije koje su poznate programerima C porodice programskih jezika kao što su: C, C++, C#, Java, JavaScript, Phyton, Perl i mnogi drugi. Te značajke čine JSON idealnim formatom za razmjenu podataka.

JSON je izgrađen na dvije strukture:

- kolekciji parova ime/vrijednost
    - u mnogim programskim jezicima to je izvedeno kao objekt, zapis, struktura, rječnik, hash tablica,…
- uređena lista vrijednosti
    - u mnogim programskim jezicima to je izvedeno kao polje, vektor, lista,…

U JSON formatu te strukture su predstavljene slijedećim oblicima:

- Objekt – neuređen set ime/vrijednost parova
  - objekt počinje lijevom vitičastom zagradom **{** i završava desnom vitičastom zagradom **}**
  - unutar zagrada nalaze se ime/vrijednost parovi
  - nakon svakog imena slijedi dvotočka **:** nakon koje slijedi vrijednost
  - ime/vrijednost parovi međusobno su odvojeni zarezom **,**
- Polje – uređena kolekcija vrijednosti
  - polje započinje lijevom uglatom zagradom **[** i završava desnom uglatom zagradom **]**
  - unutar zagrada nalaze se vrijednosti koje su međusobno odijeljene zarezom **,**
  - vrijednost može biti string unutar dvostrukih navodnika, broj, true, false, null, objekt ili polje

5.3.2 Jackson library

Ovu biblioteku sam koristio za automatsko konvertiranje JSON formata u odgovarajuće modele podataka i obrnuto. Da bi proces konvertiranja radio, potrebno je na „*getter*“ i „*setter*“ svojstva određene klase (modela) dodati *@JsonProperty* anotaciju određenog imena, što je prikazano u sljedećem primjeru.

```
@JsonProperty("answerId")
public int getAnswerId() {
	return answerId;
}

@JsonProperty("answerId")
public void setAnswerId(int answerId) {
	this.answerId = answerId;
}
```

Kod 27 - Primjer korištenja JsonProperty anotacija

5.4 Sloj aplikacijske logike

Ovaj sloj predstavlja sve operacije vezane za modele podataka. U njemu su implementirane metode čija je osnovna zadaća komunikacija i razmjena podataka sa web servisom, kao što je razmjena registracijskih podataka korisnika, podataka za prijavu u sustav, dohvaćanje ispitnih pitanja i dohvaćanje rezultata polaganog ispita.

5.4.1 Retrofit library

Retrofit library sam koristio za implementaciju Android REST[14] klijenta za komunikaciju sa web servisom Ispitnog sustava. Na slijedećem primjeru vezanom za dohvaćanje ispitnih pitanja sa web servisa je objašnjena sama implementacija Retrofit library-ja.

```
public interface GetExamInterface {
      @POST("/api/ExamService")
      void getExam(@Body String email, Callback<ExamDataResult> callback);
}
```

Kod 28 - Primjer implementacije Retrofit interface-a

Anotacija na interface metodi određuje kako će se HTTP zahtjev obraditi. Svaka metoda mora sadržavati HTTP anotaciju koja određuje vrstu HTTP zahtjeva i relativni URL. Postoje pet ugrađenih anotacija: *GET*, *POST*, *PUT*, *DELETE* i *HEAD*. Nadalje, relativni URL resursa kojem pristupamo na web servisu je naveden u anotaciji, kao što je vidljivo u navedenom primjeru.

U metodu HTTP zahtjeva moguće je dodati neki objekt podataka, a to se naznačuje sa *@Body* anotacijom.

Također je moguće odrediti da li će se metoda izvršavati sinkrono ili asinkrono. Za asinkrono izvršavanje, metoda kao zadnji parametar zahtjeva „*Callback*“, gdje je definiran i tip podataka koji se dohvaća sa web servisa. Konvertiranje podataka u JSON format i obrnuto se odvija automatski pomoću standardnog JSON konvertora, koji se također može definirati, kao što je vidljivo u slijedećem primjeru.

[14] REST – Representational State Transfer

```
ObjectMapper mapper = new ObjectMapper();

//configure rest adapter for communication with web service
RestAdapter restAdapter = new RestAdapter.Builder()
      .setEndpoint(URL_SERVICE)
      .setConverter(new JacksonConverter(mapper))
      .build();

//request questions from web service
GetExamInterface examInterface =
restAdapter.create(GetExamInterface.class);
examInterface.getExam(Global.getUserEmail(), new Callback<ExamDataResult>()
{

//get error from web service response (error in communication with web
service)
      @Override
      public void failure(RetrofitError error) {
            //log error
            Log.d("ERROR", error.getMessage());

            //send error message trough communication channel
            bus.post(new ServiceErrorEvent(error.getMessage()));
      }

      //get exam from web service response
      @Override
      public void success(ExamDataResult examData, Response response) {
            //send success message trough communication channel
            //create 'exam received event' with received questions

            if (examData.isSuccessful()) {
                  bus.post(new ExamReceivedEvent(examData.getExam()));
            }
            else {
                  bus.post(new
                  ServiceErrorEvent(examData.getResultMessage()));
                        }
            }

});
```

Kod 29 - Primjer korištenja Retrofit interface-a

Iz navedenog primjera je vidljivo da se prvo kreira nova instanca JSON konvertora. Zatim slijedi kreiranje instance *Retrofit adaptera* nad kojim se zovu metode *setEndpoint* koja kao parametar prima bazni URL web servisa, te *setConverter* koja kao parametar prima ranije kreiran JSON konvertor.

Nakon toga slijedi kreiranje instance ranije definiranog Retrofit interface-a pomoću definiranog *Retrofit adaptera*. Pozivom interface metode pokreće se definirani zahtjev prema web servisu. Na temelju dobivenog odgovora od strane web servisa određuje se daljnji tok događaja u mobilnoj aplikaciji.

5.5 Lokalna baza podataka

Lokalna baza podataka služi za pohranu riješenih ispita koji se radi eventualne pojave nekog problema nisu uspjeli poslati web servisu na obradu, te za njih nisu dobiveni rezultati. Ako dođe do neke greške prilikom dohvaćanja rezultata, korisnik dobiva obavijest o tome, te kasnije nakon otklanjanja problema (na primjer gubitka Internet konekcije) može ponovno pokušat poslati ispitne podatke na obradu. Nakon uspješnog dohvaćanja ispitnih rezultata, podaci se brišu iz lokalne baze podataka, dok se u protivnom korisnika obavještava da naknadna obrada nije uspjela.

Baza podataka je implementirana pomoću dvije klase: *ExamSystemDBOpenHelper.java* i *ExamSystemDataSource.java*.

ExamSystemDBOpenHelper klasa nasljeđuje *SQLiteOpenHelper* klasu i u njoj su definirane tablice podataka, te metode za inicijalnu kreaciju baze i tablica te za nadogradnju baze podataka na novu verziju.

ExamSystemDataSource klasa sadržava metode vezane za operacije nad podacima u bazi podataka, točnije pohranu, dohvaćanje i brisanje neobrađenih ispitnih podataka.

Zaključak

Glavni cilj ovog rada bio je izrada univerzalnog ispitnog sustava koji ima mogućnost testiranja polaznika sustava u više ispitnih područja. Krajnji rezultat je ispitni sustav koji s jedne strane pruža mogućnost kreiranja ispitnog sadržaja, a s druge strane pruža pristup testiranju znanja određenog područja.

Sam rad se sastoji od nekoliko komponenata, od kojih je web aplikacija izrađena u ASP.NET MVC tehnologiji jedna od važnijih. Pomoću nje administrator sustava može kreirati i uređivati ispitni sadržaj, te ima potpuni uvid u rezultate pojedinih ispita i pristupnika ispitima. S druge strane korisnici mogu pristupiti rješavanju ispitnog sadržaja, te im je omogućena prijava ili registracija u ispitni sustav. Svi podaci se pohranjuju u Microsoft SQL bazu podataka.

Jedna od komponenata ovog rada je i Android mobilna aplikacija koja također korisnicima pruža mogućnost prijave ili registracije u ispitni sustav, te pristup rješavanju ispitnog sadržaja. Komunikacija mobilne aplikacije sa bazom podataka izvedena je pomoću web servisa.

Sam web servis pruža mogućnost da se u budućnosti izrade aplikacije ispitnog sustava i na nekim drugim platformama kao što su na primjer Windows Phone ili iOS, koje će preko njega imati pristup zajedničkoj bazi podataka.

Sama sigurnost sustava je minimalna. Određeni dijelovi aplikacije zahtijevaju određeni autorizacijski nivo da bi im se moglo pristupiti. U budućnosti bi prvenstveno trebalo implementirati komunikaciju između pojedinih dijelova ispitnog sustava pomoću SSL (Secure Socket Layer).

Popis slika

Popis kôdova

Literatura

[1] WROX. *Professional ASP.NET MVC 5*. Jon Galloway, Brad Wilson, K. Scott Allen, David Matson, 2014.

[2] APRESS. *Beginning Android 4*. Grant Allen, 2012.

[3] http://getbootstrap.com/about/

[4] https://gridmvc.codeplex.com/documentation

[5] http://www.chartjs.org/docs/

[6] http://littledice.me/2014/03/19/calculating-probabilities-in-c/

[7] http://square.github.io/otto/

[8] http://square.github.io/retrofit/

[9] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

[10] http://www.codeproject.com/Articles/686994/Create-Read-Advance-PDF-Report-using-iTextSharp-in

[11] http://keith-wood.name/countdown.html

[12] http://json.org/