Pametna zgrada

Stanić, Mato

SV VEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Diplomski studij

PAMETNA ZGRADA

Diplomski rad

Mato Stanić

Osijek, 2016.

SADRŽAJ

1. UVOD

Pamenta zgrada (engl. Smart building) je prostorija za život koja ima neki automatizirani i centralizirani sustav za upravljanje grijanjem, ventilacijom, rasvjetom i ostalim sustavima u zgradi. Cilj ovakvog upravljanja je povećati ugodnost boravka stanara u pojedini prostorijama, kao i povećati efikasnost i iskoristivost energije u samoj zgradi. Samim time to znači smanjenje troškova života.

Upravljanje pametnim zgradama, odnosno automatizacija zgrada, je primjer distribuiranih računalnih sustava. Tu se misli na mrežu elektroničkih uređaja koji su raspoređeni po zgradi, a imaju mogućnost međusobne komunikacije. Tu možemo uračunati sve sustave koji su dizajnirani kako bi nadgledali i kontrolirali mehaničke, sigurnosne, svjetlosne (svjetla u slučaju nužde) i vlage zraka. [1]

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je bio pružiti pregled već postojećih rješenja za nadzor i upravljanje sustavima u pametnim zgradama, te izraditi maketu zgrade i jednostavan računalni sustav za nadzor i upravljanje sustavima unutar.

2. POSTOJEĆA RJEŠENJA

2.1. Siemens – Pametni gradovi [2]

Pamentni gradovi (engl. *Smart cities*) je rješenje za upravljanje pametnim zgradama u većem mjerilu. Kada to kažemo mislimo da će više različitih zgrada biti povezano na jedan središnji sustav za upravljanje.

Osnovna odlika ovog sustava je upravljanje strujnom mrežom. Sustav će biti u mogućnosti napraviti rebalans potrošnje električne energije, tako što će proučavati uzorke ponašanja stanara u pojedinim zgrada i bilježiti potrošnju za svaku zgradu. Te će u svakom trenutku moći procijeniti koliko je energije potrebno preusmjeriti na koju zgradu.

Siemens je već razvio ovakav sustav te je on poznat pod imenom „Desigo CC“. To je prvi sustav koji omogućuje povezivanje više različitih zgrada i sustava za regulaciju u jednu cjelinu. S obzirom da su sustavi za zaštitu od požara, upravljanje ventilacijom, temperaturom, rasvjetom, video nadzor sada objedinjeni u jednu cjelinu, operaterima je znatno olakšan posao. Sva mjerenja i podatke imaju dostupne na jednom mjestu u stvarnom vremenu. Moraju se upoznati samo s jednom računalnim rješenjem za upravljanje svim sustavima u zgradi.

Glavna odlika ovog sustava je komunikacija sustava s električnim automobilima. Sustav električne automobile više ne gleda samo kao još jednog potrošača koji se priključio na električnu mrežu. Kao primjer je navedno sljedeće. Ujutro kada automobili dođu na parking ispred ureda i priključe se na mrežu, oni trebaju punjenje svojih baterija, ali to ne znači da sustav mora odmah isporučiti energiju. Automobil se može puniti povremeno tijekom dana kada u sustavu bude višak energije, bitno je samo da se napune do kraja radnog vremena. Tijekom radnog vremena taj isti automobil može poslužiti i kao izvor električne energije. Recimo na primjer da vrijeme iznenada postane oblačno i smanji se količina električne energije koju fotonaponske ćelije na krovovima zgrada mogu isporučiti. U tom slučaju će sustav uzeti tu energiju od automobila kako bi kompenzirao gubitke.

2.2. Intel – Pametne zgrade i Internet objekata [3]

Intel-ovo rješenje za pametne zgrade poznato pod nazivom Pametne zgrada i Internet objekata (engl. *Smart building with Internet of Things Technologies*).

Kako navode na svojim stranicama pametna zgrada je kao i pametna kuća, odnosno prostor u kojem centralni računalni sustav povećava efikasnost, ugodnost boravljenja, te sigurnost skupljanjem podataka različitih senzor i analiziranjem istih.

Mogućnosti koje njihov sustav ima možemo podijeliti u dvije kategorije: upravljanje i nadzor sustavima izvan zgrade, te upravljanje i nadzor sustavima u zgradi.

Kao primjere prvog upravljanja navode navigaciju na parkiralištu zgrade. Prilikom dolaska na parkiralište zgrade sustav detektira pametni telefon korisnika ili registracijske oznake automobila, te korisnika putem aplikacije obavješava i preusmjerava do prvog slobodnog parkinga. Ukoliko je riječ o električnom automobilu preusmjerit će ga do prve slobodne stanice za punjenje.

Drugi primjer za upravljanje sustavima izvan zgrade je upravljanje sustav za navodnjavanje. Sustav treba omogućiti optimalnu količinu vode za biljke koje se nalaze u okolini zgrade. Analizom trenutne vremenske prognoze i one za sljedećih par sati, sustav može odlučiti hoće li biti potrebno upaliti navodnjavanje ili ne.

Druga kategorija se odnosi na upravljanje unutar zgrade. Prilikom ulaska korisnika u zgradu, sigurnosne kamere bilježe njegov dolazak, te automatski provjeravanju njegove dokumente u bazi, ovlasti i prava pristupa koje korisnik ima. Putem aplikacije sustav obavještava korisnika o rasporedu događaja za trenutni dan, te koje su prostorije slobodne za rad. Ukoliko korisnik ima zakazan sastanak negdje u zgradi, aplikacija mu prikazuje vrijeme i lokaciju istog, te najbrži put kojim može stići do navedenog odredišta.

Sustav upravlja grijanjem, ventilacijom i rasvjetom unutar cijele zgrade, a svaki korisnik ima mogućnost ocjenjivanja koliko se ugodno osjeća u pojedinoj prostoriji. Ocjenjivanje se vrši na način da se odabere jedna od 3 ponuđene opcije: temperatura je previsoka, temperatura je zadovoljavajuća ili temperatura je preniska. Sustav na osnovu ovih ocjena može prilagoditi temperaturu kako bi što bolje odgovarala željama svih korisnika.

2.3. Esight Energy [4]

eSight Energy su razvili računalno rješenje pod nazivom „Energy Management Software“. Korištenjem njihovog rješenja omogućuje korisniku nadzor, upravljanje i smanjenje potrošnje energije.

Sustav omogućuje vizualizaciju prikupljenih podataka korištenjem velikog broja alata za analizu istih (slika 1.1)

Sl. 2.1 Primjer vizualizacije podataka

Neki od mogućih prikaza podataka su tablični ili grafički prikaz potrošnje energije u nekom vremenskom periodu, što nam omogućuje brzo i jednostavno uočavanje trendova ponašanja.

Analiza električne energije, kako oni navode, omogućuje analizu podataka vezanih za potrošnju električne energije, kao što su: najveća potrošnja, faktor snage[1], faktor opterećenja[2].

Korisnik ima mogućnost postavljanja alarma, gdje će zadati najveću željenu potrošnju, a sustav će ga obavijestiti ako se ta vrijednost prekorači. Ima mogućnost unosa vlastitih cijena energije, te se na osnovu tih podataka i podataka o stvarnoj potrošnji u prošlosti (analiza trendova) vrši analiza i procjena budžeta potrebnog za nadolazeće razdoblje.

Prema nekim procjenama korištenje ovog sustava omogućava uštedu do 30% troškova energije.

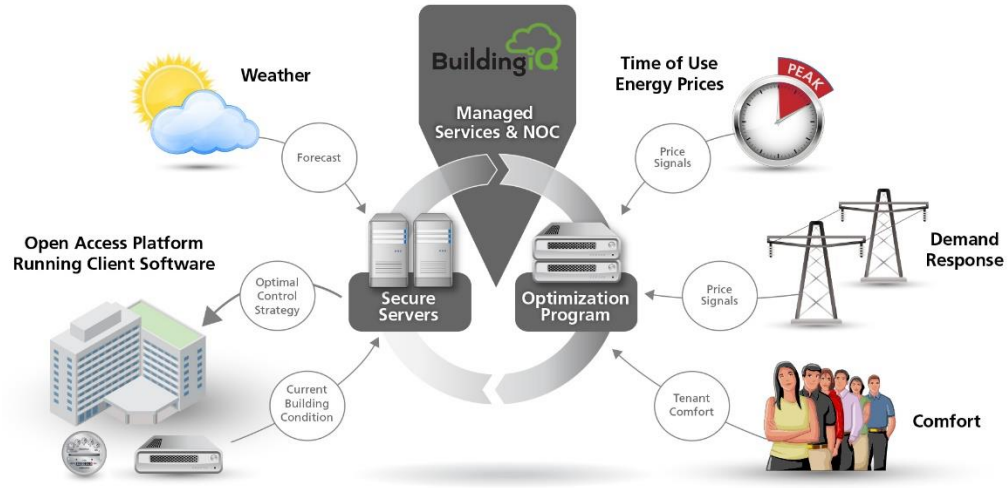
2.4. BuldingIQ [5]

Napredni sustava za upravljanje grijanjem, ventilacijom i klimatizacijom[3]. BuildingIQ je razvio računalno rješenje pod nazivom „Predictive Energy Optimization“. Njihovo rješenje radi sa svim napoznatijim (komercijalno dostupnim) sustavima za automatizaciju zgrade (BAS – engl.*building automation system*)

[1] Odnos između radne i prividne snage ([19], stranica 2)
[2] Faktor se opterećenja izračunava kao omjer ostvarene potrošnje energije i energije koja se mogla ostvariti da je elektroenergetski sustav bio opterećen u cijelom vremenskom intervalu maksimalnim opterećenjem. Faktor opterećenja može poslužiti kao pokazatelj iskoristivosti postrojenja elektroenergetskog sustava. ([20], stranica 9)
[3] HVAC (engl. *Heating, ventilation and air conditioning*)

Sustav konstantno provjerava vremenske uvjete izvan zgrade, broj osoba u zgradi kao i cijenu energetskih resursa. U ovisnosti od tim parametrima radi kalkulacije i procjene kako bi na što efikasniji način podesio parametre za reguliranje temperature, klimatizacije i ventilacije za naredna 24 sata. Samo male promjene temperature ili tlaka za HVAC mogu dovesti do velikih ušteda.

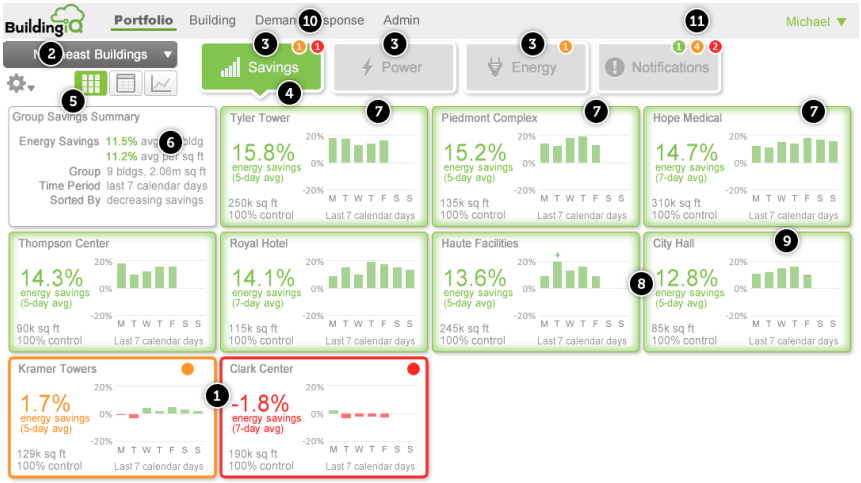
Slika 2.2 nam prikazuje grafički prikaz kako cijeli sustav funkcionira.



Sl. 2.2 Grafički prikaz sustava

Prema nekim procjenama standardna ušteda energije se kreće od 10 do 25%, iako su zabilježeni slučajevi kada je ta ušteda bila čak 40%.

Kako bi se omogućilo što jednostavnije upravljanje zgradama, razvijeno je grafičko sučelje gdje se korisniku prikazuje status pojedine zgrade, kao i ukupna ušteda ili gubitak za grupu zgrada. Slika 2.3 prikazuje primjer takvog sučelja.



Sl. 2.3 Primjer grafičkog sučelja

Brojevi na slici predstavljaju sljedeće stvari:

(1) Pojedine zgrade i njihove obavijesti su prikazane različitim bojama kako bi se korisniku skrenula pozornost na neka parametre koje treba provjeriti

(2) Ime trenutne grupe zgrada koje se prikazuju na sučelju

(3) Podaci se mogu filtrirati po tri različite kategorije

(4) Grafički prikaz uštede ili gubitka za zadnjih 7 kalendarskih dana. Podatak se može prikazati u obliku postotka koji će prikazivati odstupanje od neke referentne veličine (prosječna potrošnja) ili kao postotak uštede u nekoj valuti, npr. eurima.

(5) Tri različita načina prikaza podataka:

   (a) U obliku kvadrata – svaki kvadrat predstavlja jednu zgradu i prikazuje podatke za nju

   (b) Tablični prikaz – detaljniji prikaz podataka

   (c) Grafički prikaz – detaljniji prikaz podataka, mogućnost prikazivanja podataka više različitih zgrada na istom grafu

(6) Ukupni prikaz uštede za odabranu grupu zgrada

(7) Pojedinačni prikaz uštede za svaku zgradu

(8) Prosječna ušteda za zadnjih 7 kalendarskih dana

(9) Prikaz ušteda na dnevnoj bazi za zadnjih 7 dana

(10) Obavijesti se generiraju u trenutku kada se pojavio nekakav problem ili događaj koji zahtjeva procjenu i akciju korisnika

(11) Obavijesti su podijeljene na 3 različite razine: informacije – zeleno, upozorenja – narančasto, kritične pogreške – crveno

3. KORIŠTENE TEHNOLOGIJE

3.1. Java

Java je objektno orijentirani programski jezik koji su razvili James Gosling, Patrick Naughton i drugi inženjeri u tvrtci Sun Microsystems. Razvoj je počeo 1991. godine, a prva verzija je objavljena u studenom 1995.

Velika prednost u odnosu na većinu dotadašnjih programskih jezika je to što se programi pisani u Javi mogu izvoditi bez preinaka na svim operativnim sustavima za koje postoje Java Virtualni Strojevi (engl. *JVM - Java Virtual Machine*). Java je jedan od najkorištenijih programskih jezika. Procjene i izvješća o broju korisnika kreću se od gotovo 7 do preko 10 milijuna [6].

Kako bi se kôd koji napišu programeri (datoteke s ekstenzijom *.java*) mogao izvršiti, potrebno ga je prvo prevesti u format zvan *bytcode*. Prevođenje pretvara *.java* datoteke u *.class* datoteke. Takve datoteke se mogu izvršavati unutar Java Virtualnih Strojeva.

Ukoliko se Java koristi za razvoj mobilnih aplikacija za Android operacijski sustav, onda se koristi ART virtualni stroj. ART je zamijenio Dalvik koji se koristio prije njega, ali je ART poboljšao performanse aplikacije. Java kôd je moguće pisati u bilo kojem tekstualnom uredniku, tako na primjer na Windows operacijskom sustavu se može koristiti *Notepad* ili *Notepad++*, dok je na UNIX baziranim operacijskim sustavima moguće koristiti *vi* ili *nano.*Za prevođenje su potrebni alati i biblioteke koje se nalaze u Java razvojnom sučelju (engl. *JDK – Java Development Kit*). Neovisnost Jave o operacijskom sustavu, odnosno platformi na kojoj se izvodi program, proizlazi iz činjenice da Java program nema uvid pod kojim se operacijskim sustavom izvodi. Program se izvršava unutar Java izvršnog okruženja (engl. *JRE – Java Runtime Environment*).

U ovom diplomskom radu Java programski jezik je korišten za razvoj internet aplikacije, Android aplikacije i aplikacije koja je pokrenuta na Raspberry Pi uređaju.

3.2. Internet stranica

3.2.1. Apache Shiro

Apache Shiro je snažan i lagan za korištenje Java sigurnosti okvir. Koji nam omogućava autentifikaciju, autorizaciju, kriptografiju i upravljanje sesijama. Naziv Shiro potječe od japanske riječi koja znači dvorac. Cijeli okvir je dizajniran kako bi se na što lakši način omogućilo

osposobljavanje sigurnosti na internet stranicama, a kako bi se opet zadržala što bolja sigurnost sustava [7].

```
       protected AuthenticationInfo
doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {

        UsernamePasswordToken upat = (UsernamePasswordToken) token;
        AppUser appUser = appUserManager.getByUsername(upat.getUsername());
        if (appUser != null && appUser.isActive()) {
            AuthenticationInfo authInfo = new
SimpleAuthenticationInfo(appUser, appUser.getPassword(), getName());
            if (simpleCredentialsMatcher.doCredentialsMatch(token,
authInfo)) {
                log.debug("Successfull login: " + appUser.getUsername());
                return authInfo;
            }
        }
        throw new AuthenticationException("Invalid username/password
combination!");
    }
```

Sl. 3.1 Autentifikacija korisnika

```
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
principals) {

        Set<String> roles = new HashSet<>();
        Set<org.apache.shiro.authz.Permission> permissions = new
HashSet<>();

        Collection<AppUser> principalsList =
principals.byType(AppUser.class);
        for (AppUser appUser : principalsList) {
            Role role = appUser.getRole();
            if (role != null) {
                roles.add(role.getName());
                for (Permission permission : role.getPermissions()) {
                    permissions.add(new
WildcardPermission(permission.getValue()));
                }
            }
        }

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo(roles);
        info.setRoles(roles);
        info.setObjectPermissions(permissions);
        return info;
    }
```

Sl. 3.2 Autorizacija korisnika

Slika 3.1 i 3.2 prikazuju način na koji je moguće postaviti autorizaciju i autentifikaciju korisnika prilikom prijave na internet stranicu.

3.2.2. HTML5

HTML je akronim za *Hyper Text Markup Language*. To je jezik kojim opisujemo internet stranicu, odnosno služi nam za izradu stranice. HTML ne možemo nazvati programskim jezikom, to je više opisni jezik koji nam služi za opisivanje strukture i izgleda internet dokumenta. On daje upute internet pregledniku na koji način treba prikazati neku stranicu [8].

HTML5 je prva nova revizija standarda od HTML 4.01, koji je izdan 1999. Nastao je u suradnji sa WHATWG (engl. *Web Hypertext Application Technology Working Group*) i W3C (engl. *World Wide Web Consortium*).

Svaki HTML dokument sastoji se od osnovnih građevnih blokova, HTML elemenata. Na samom početku svakog HTML dokumenta preporučljivo je postaviti DTD (engl. *Document Type Declaration,* slika 3.3)

```
<!DOCTYPE html>
```

Sl. 3.3 HTML5 deklaracija (DTD)

Nakon toga slijedi element <*html*> kojim se označava početak HTML dokumenta. Unutar <*html*> dokumenta se mogu nalaziti <*head*> i <*body*> elementi. Unutar takve deklaracije podržani su svi atributi i elementi iz prethodnih inačica HTML-a, kao i neki novi element koje omogućava HTML5. Prema [9], HTML5 standard omogućuje korištenje novih elemenata za glazbu, video i prikaz grafičkih elemenata. HTML5 nam donosi brojne nove mogućnosti koje HTML 4.01 i XHTML 1.x nisu imali, kao što je mogućnost reprodukcije videa na stranici bez korištenja Adobe Flash-a ili Microsoftovog Silverlighta [10]. Neki od novododanih elemenata su <*footer*>, <*nav*> te <*header*>, oni nam omogućuju zamjenu nekih generičkih blokova, primjerice <*div*> *i* <*span*>. Popis nekih novododanih elemenata koje podržava HTML5 prikazan je tablicom 3.1.

Tab 3.1 *Novi HTML5 elementi*

<article>	članak
<audio>	standard za sviranje audio datoteka
<aside>	sadržaj koji se nalazi pored sadržaja stranice
<canvas>	područje unutar koje je moguće crtati grafike
<details>	dodatne podatke koje korisnik može prikazati ili skriti

<dialog>	blok ili prozor sa dijalogom
<figure>	samostalni sadržaj (ilustracije, dijagrami,slike...)
<footer>	donji dio dokumenta ili dijela stranice
<header>	gornji dio dokumenta ili dijela stranice
<mark>	označeni tekst
<nav>	poveznice navigacije
<progress>	napredak zadatka
<section>	dio dokumenta
<video>	standard za prikazivanje video datoteka

U kombinaciji sa CSS3, HTML5 je vrlo moćan alat koji nam omogućuje izradu takozvanih odzivnih (engl. *responsive*) stranica. Ovo je jedna od velikih prednosti HTML5 u odnosu na njegove prethodnike s obzirom da je zabilježena sve veća posjećenost internet stranica s mobilnih uređaja, odnosno pametnih telefona (engl. *smartphone*).

3.2.3. CSS3

CSS, akronim za *Cascading Style Sheets*, je stilski jezik korišten za formatiranje i opisivanje internetskih stranica. Prvotno su u HTML ubacivani elementi za definiciju prezentacije, na primjer <*font*> element. Kako je došlo do razvoja interneta i pojave sve većeg broja internet stranica, brzo je uočena potraba za stilskim jezikom koji će HTML osloboditi od takvih elemenata, odnosno trebalo je smisliti način kako će se odraditi oblikovanje sadržaja internet stranica, u tu svrhu danas koristimo CSS. Drugim riječima, stil definira kako prikazati HTML elemente, CSS-om uređujemo sam izgled i raspored stranice [11].

„CSS3“ predstavlja treću inačicu ovog jezika. Također za razliku od svojih prethodnika, gdje su sve specifikacije bile definirane u jednu cjelinu, CSS3 je podijeljen u više manjih dokumenata koji se nazivaju moduli. S obzirom na ovakvu podjelu različiti moduli se nalaze u različitim fazama razvoja. Možemo izdvojiti neke od češće korištenih modula kao što su: odabirnici (engl. *selectors*), animacije (engl. *animations*), pozadine i obrubi (engl. *background and borders*), transformacije (engl. *transforms*), prijelazi (engl. *transitions*), itd [12].

CSS3 omogućava dodavanje dinamičkih i dekorativnih svojstava HTML stranici. Podjela na module nam omogućuje lakše dodavanje novih funkcionalnosti, što je prije zahtjevalo pisanje kompliciranih kodova. Također nam omogućuje korištenje efekata kao što su gradijent, zaobljeni rubovi, animacije i razni prijelazi koji su prije zahtjevali korištenje drugih jezika i raznih dodataka kao što su JavaScript, jQuery.

3.2.4. Ajax, JavaScript, jQuery

JavaScript je skriptni programski jezik, koji se izvršava u internet pregledniku na strani korisnika. Napravljen je da bude sličan Javi, zbog lakšeg korištenja, ali nije objektno orijentiran kao Java. JavaScript u kombinaciji s Ajax (engl. *Asynchronous JavaScript and XML*) tehnikom omogućuje internet stranicama komunikaciju sa serverskim programom, što čini web aplikaciju interaktivnijom i lakšom za korištenje [13]. Kod pisan JavaScript jezikom naziva se skripta.

jQuery je mala JavaScript biblioteka koja nam omogućuje odabiranje, pretragu i upravljanje elementima HTML dokumenta. Također nam daje mogućnost upravljanje događajima (kao što je prelazak miša preko elementa), animiranje i korištenje Ajax-a kako bi se ostvarila komunikacija sa serverskim programom.

Slika 3.4 nam prikazuje primjer jQuery koda za upravljanje događajem prelazak miša preko elementa u tablici.

```
$( "td" ).hover(
  function() {
    $( this ).addClass( "hover" );
  }, function() {
    $( this ).removeClass( "hover" );
  }
);
```

Sl. 3.4 Primjer jQuery koda

Ajax, akronim za *Asynchronous JavaScript and XML*, odnosno asinkroni JavaScript i XML. Koristeći Ajax za uspostavljanje komunikacije sa serverom, JavaScript šalje zahtjev poslužitelju, interpretira vraćene rezultate i u ovisnosti od rezultat odrađuje neku radnju. Na ovaj način korisniku nije vidljiva komunikacija s poslužiteljem.

```
$.ajax(url, {
	data: {
		roomToChange: this.id,
		state: checkState
	},
	success: function (data, textStatus, jqXHR) {
		if (data === "SUCCESS") {
			if ($('#successMessage').is(":visible")) {
				return;
			}
			$('#successMessage').show();
			setTimeout(function () {
				$('#successMessage').hide();
			}, 3000);
		}
		else if (data === "ERROR") {
			if ($('#errorMessage').is(":visible")) {
				return;
			}
			$('#errorMessage').show();
			setTimeout(function () {
				checkBoxId.prop("checked",
!checkBoxId.prop("checked"));
				$('#errorMessage').hide();
			}, 3000);
		}
	}
});
```

Sl. 3.5 Primjer Ajax koda

Slika 3.5 nam prikazuje jednostavan primjer jQuery *ajax()* metode. Prije slanja zahtjeva na server dohvaćeni su željeni podaci, te su isti predani *ajax()* metodi, uz dodatak adrese na koju se podaci šalju. U slučaju uspješnog ili neuspješnog izvršavanja zadatka, korisniku će se ispisati odgovarajuća poruka.

3.2.5. Thymeleaf

Thymeleaf je dodatak za upravljanje XML, XHTML i HTML5 predlošcima prilikom razvoja Java aplikacija. Može se koristiti prilikom razvoja Java aplikacija koje će raditi u internet okruženju, ali i van njega. Ima bolje razrađenu podršku za XHTML/HTML5 predloške koji će se koristiti unutar internet aplikacije, ali ima mogućnost obrađivanja bilo kojeg XML predloška čak i ako se ne koristi unutar internet okruženja. Ovaj dodatak nam omogućuje potpunu integraciju Spring okvira (*Spring Framework*) [14].

```
<table>
  <thead>
    <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="prod : ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price,1,2)}">0.99</td>
    </tr>
  </tbody>
</table>
```

Sl. 3.6 Primjer Thymleaf predloška

Slika 3.6 nam prikazuje primjer Thymeleaf predloška koji će nam prikazati HTML5 tablicu. Redovi te tablice sadrže elemente iz *List<Product>* varijable koja je nazvana *allProducts*.

3.3. Android aplikacija

3.3.1. Android operacijski sustav

Google Android je prvi operacijski sustav za mobilne uređaje (mobilni telefoni, tableti, netbook računala, Google TV). Prva stabilna verzija ovog operacijskog sustava se pojavila u devetom mjesecu 2008. godine. Verzije 1.0 i 1.1 nisu imale kodno ime pod kojim su izdane. Prva verzija koja ima kodno ime je Android 1.5 odnosno *„Cupcake“*. Zadnja stabilna verzija koju su objavili je Android 6.0 (*„Marshmallow“*) [15].

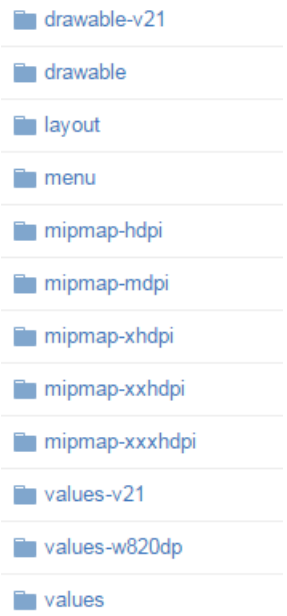
Android platforma je prilagođena za uporabu na uređajima s većim zaslonima poput pametnih telefona koji rabe 2D grafičku knjižicu ili 3D grafičku knjižicu temeljenu na OpenGL ES 2.0 specifikaciji. Omogućuje reproduciranje velikog broja audio i video formata. Iako je većina aplikacija pisana u Java programskom jeziku, na Android uređajima ne postoji Java Virtualni Stroj, te oni koriste ART Virtualni Stroj što je već spomenuto u poglavlju 3.1.

3.3.2. Android aplikacije

Android aplikacije su pisane Java programskim jezikom. Kôd se pomoću Android razvojnog okruženja (engl. *SDK – Software Development kit*) prevodi u Android paket (engl. *APK – Android package*). Paket u sebi sadrži sve dodatne biblioteke koje su potrebne za uspješno instaliranje aplikacije na uređaj. Prema [16], android aplikacija se sastoji od upravitelj aktivnostima (engl. *Activity manager*), upravitelj obavijestima (engl. *Notification manager*), prikaza (engl. *Views*), upravitelj resursima (engl. *Resource managers*), pružatelji usluga (engl. *content providers*).

Aktivnost predstavlja jedan prikaz zaslona sa korisničkim sučeljem. Iako funkcioniraju zajedno za stvaranje cjelovite aplikacije, aktivnosti su međusobno nezavisne.

Osim kompomenti koje su gore navedene, svaka aplikacija sadrži manifest i resurse. Resursi mogu biti animacije, slike, audio datoteke, izgled korisničkog sučelja (engl. *layout*). Slika 3.7 prikazuje moguće resurse koji se mogu nalaziti u android aplikaciji, odnosno direktorije u koje ti resursi trebaju biti raspoređeni.



Sl. 3.7 Mogući resursi Android aplikacije

Manifest datoteka sadrži neke osnovne podatke o aplikaciji. Kao što su ime same aplikacije, popis komponenti, te dopuštenja (engl. *Permission*) koja aplikacija mora zatrašiti od korisnika, a on može prihvatiti ili odbiti. Tu su definirana i svojstva pojedine aktivnosti, kao što su tema i naziv. Slika 3.8 prikazuje izgleda manifesta jednostavne aplikacije gdje aplikacija traži dopuštenje za čitanje kontakata iz imenika [17].

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.basiccontactables"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <!-- Min/target SDK versions (<uses-sdk>) managed by build.gradle -->
    <permission android:name="android"></permission>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.Sample" >
        <activity
            android:name="com.example.android.basiccontactables.MainActivity"
            android:label="@string/app_name"
            android:launchMode="singleTop">
            <meta-data
                android:name="android.app.searchable"
                android:resource="@xml/searchable" />
            <intent-filter>
                <action android:name="android.intent.action.SEARCH" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Sl. 3.8 Jednostavni manifest

4. RAČUNALNI SUSTAV ZA NADZOR I UPRAVLJANJE

U sklopu ovog diplomskog rada razvijen je jednostavan računalni sustav koji će korisniku omogućiti nadzor i upravljanje pojedinim stanovima unutar zgrade. Napravljena je jednostavna maketa zgrade koja će biti prikazana nešto kasnije u ovom radu. Programski dio je podijeljen na 3 dijela: aplikacija koja se nalazi na serveru, aplikacija za android pametne telefone uz pomoć koje se vrši kontrola, te aplikacija koja je instalirana na Raspberry Pi uređaj i obavlja kontrolu unutar zgrade.

4.1. Početni zahtjevi

Web aplikacija se nalazi na serveru s operacijskim sustavom Ubuntu 15.10. Prije samog prebacivanja aplikacije na server i pokretanja, bilo je potrebno zadovoljiti nekoliko početnih uvjeta:

- Intstalirati Oracle Java 8
- Instalirati Apache Tomcat 8
- Instalirati PostgreSQL bazu podataka

4.1.1. Instalacija Oracle Java 8

Kako bi instalirali Oracle Java 8 na server potrebno je pokrenuti sljedeće naredbe.

```
$ sudo add-apt-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java8-installer
```

Sl. 4.1 Instalacija Oracle Java 8

Nakon dovršene instalacije potrebno je potvrditi da je ista uspješno provedena do kraja.

```
root@Smartbuilding:~$ java -version

java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Sl. 4.2 Potvrda instalacije

4.1.2. Instalacija Apache Tomcat 8

Prije pokretanja instalacije za Apache Tomcat 8, potrebno je promijeniti trenutni direktorij na serveru.

```
root@Smartbuilding:~$ cd /opt
```

Sl. 4.3 Promjena direktorija

Zatim trebamo dohvatiti .tar.gz datoteku sa sljedećom naredbom.

```
root@Smartbuilding:~$ wget http://ftp.carnet.hr/misc/apache/tomcat/tomcat-8/v8.0.33/bin/apache-tomcat-8.0.33.tar.gz
```

Sl. 4.5 Preuzimanje datoteke Apache Tomcat 8.0.33

```
root@Smartbuilding:~$ tar -zxvf apache-tomcat-8.0.33.tar.gz
```

Sl. 4.4 Raspakiravanje datoteke

Nakon što smo preuzeli datoteku istu je potrebno raspakirati u trenutni direktorij.

Pokretanje prethodne naredbe napravit će novi direktorij pod imenom „apache-tomcat-8.0.33“. Kako bi si olakšali posao, potrebno je napraviti poveznicu između dva direktorija. U našem slučaju će to biti između „/opt/tomcat/“ i „/opt/apache-tomcat-8.0.33“.

```
root@Smartbuilding:~$ ln -s /opt/apache-tomcat-8.0.33 /opt/tomcat
```

Sl. 4.6 Stvaranje veze između direktorija

4.1.3. Instalacija PostgreSQL

Instalaciju PostgreSQL baze podataka možemo napraviti sa sljedećom naredbom.

```
root@Smartbuilding:~$ sudo apt-get install postgresql-client
```

Sl. 4.7 Instaliranje PostgreSQL baze

Nakon dovršetka instalacije, potrebno je napraviti novog korisnika i novu bazu podataka.

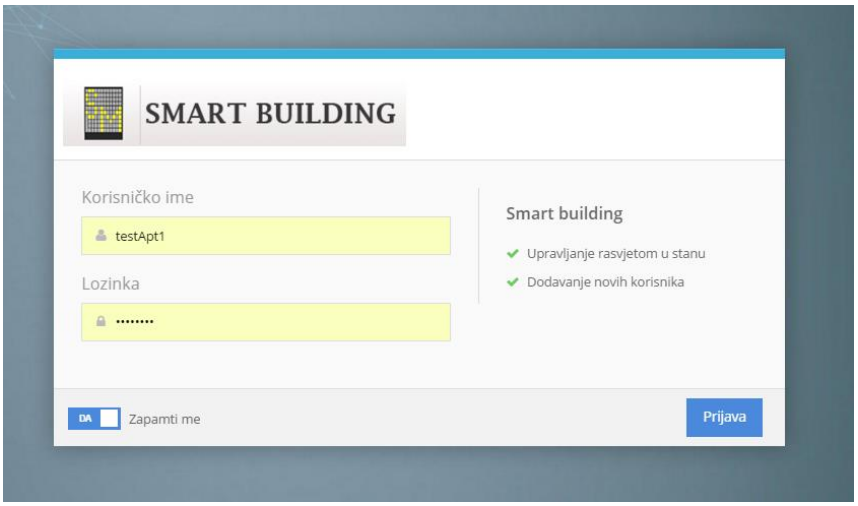
```
postgres=# CREATE USER m2stanic WITH PASSWORD 'm2stanic';

CREATE ROLE

postgres=# CREATE DATABASE smartbuilding WITH OWNER m2stanic;

CREATE DATABASE
```

Sl. 4.8 Kreiranje korisnika i baze

Pokretanjem ovih naredbi smo završili s početnim zahtjevima koje je potrebno zadovoljiti prije pokretanja aplikacije.

4.2. Internet aplikacija

Web aplikacija se sastoji od 2 dijela. Prvi dio je informacijski dio kojem mogu pristupiti svi korisnici. Drugi dio aplikacije je administracijski dio. Za pristup ovom dijelu stranice potrebna je prijava korisnika (slika 4.9).



Sl. 4.9 Prijava korisnika

4.2.1. Informacijski dio

Slika 4.10 prikazuje java kontroler koji je zadužen za prikazivanje odgovarajućih stranica na javnom dijelu aplikacije. *„PublicController“* se nalazi u paketu *„hr.m2stanic.smartbuilding.web“*. *„@RequestMapping({„/“})“* anotacija označava da će ovdje završiti svi korisnički zahtjevi koji su upućeni na URL „http://89.107.57.144:8080/smartbuilding/“

```
@Slf4j
@Controller
@RequestMapping({"/"})
public class PublicController {
```

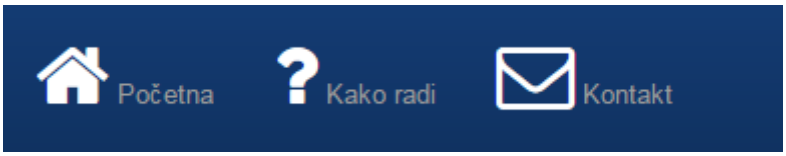
Sl. 4.10 Javni kontroler

Slika 4.11 prikazuje metodu *„indexPage“*. U ovisnosti sto se proslijedi kao vrijednost *MenuOption* parametra kontroler će korisnika preusmjeriti na odgovarajuću stranicu. U slučaju da ništa nije prosljeđeno za vrijednost ovog parametra otvorit će se početna stranica.

```
    @RequestMapping("/")
    public String indexPage(Model model, HttpServletRequest request ,
RedirectAttributes ra, @RequestParam(required = false) MenuOption
menuOption ) {
        if(menuOption == null || menuOption.equals(MenuOption.HOME))
            return "redirect:/home";
        else if(menuOption.equals(MenuOption.HOW))
            return "redirect:/how";
        else if(menuOption.equals(MenuOption.CONTACT))
            return "redirect:/kontakt";

        return "public/home";
    }
```

Sl. 4.11 indexPage metoda

Na javnom dijelu aplikacije su dostupne 3 stranice. Korisnik pristupa dostupnim stranicama preko navigacije koja je prikazana na vrhu stranice (slika 4.12)



Sl. 4.12 Navigacija

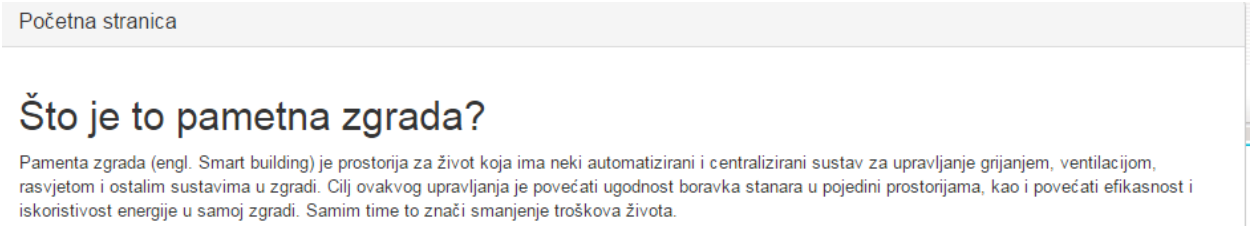
Slika 4.13 prikazuje HTML kod koji je potrebno napisati kako bi se navedena navigacija prikazala korisniku.

```
<ul class="nav navbar-nav">
      <li>
            <a th:href="@{/(menuOption='HOME')}">
                  <i class="fa fa-home fa-3x" aria-hidden="true"
style="color:white"></i>
                  Početna
            </a>
      </li>
      <li>
            <a th:href="@{/(menuOption='HOW')}">
                  <i class="fa fa-question fa-3x" aria-hidden="true"
style="color:white"></i>
                  Kako radi
            </a>
      </li>
      <li>
            <a th:href="@{/(menuOption='CONTACT')}">
                  <i class="fa fa-envelope-o fa-3x" aria-hidden="true"
style="color:white"></i>
                  Kontakt
            </a>
      </li>
</ul>
```

Sl. 4.13 HTML kôd za navigaciju

Kao što vidimo iz slike 4.13 korišteno je Thymeleaf okruženje (*„th:href“*) kako bi se ostvarila komunikacija između sučelja koje je vidljivo korisniku i serverske strane aplikacije. U ovom slučaju Java kontroleru (slika 4.11) je prosljeđena vrijednost parametra *„menuOption“*.

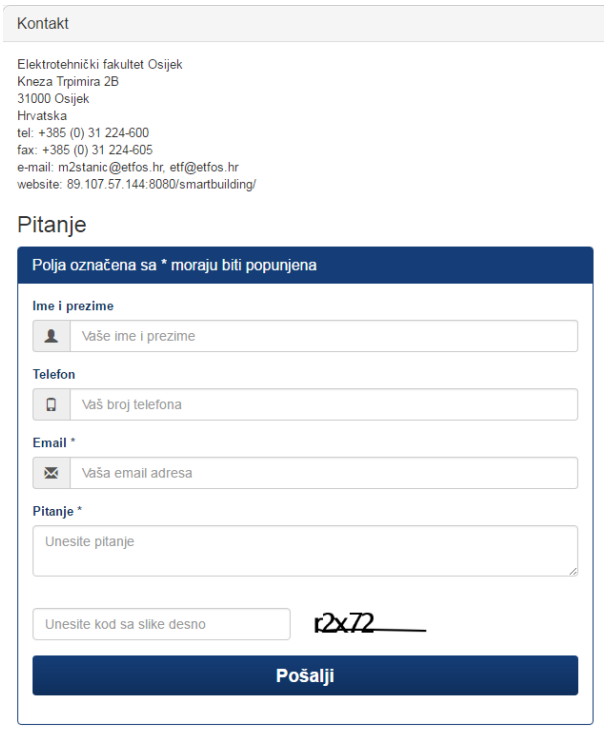
Prva stranica kojoj korisnik može pristupiti je početna stranica, gdje je ukratko opisano što je to pametna zgrada (slika 4.14).



Sl. 4.14 Početna stranica

Druga stranica je opis kako sama aplikacija radi, te na koji način je ostvarena komunikacija između Android aplikacije i RaspberryPi aplikacije.

Treća stranica je kontakt forma, gdje korisnici mogu poslati e-mail administratoru stranice. Slika 4.15 prikazuje izgleda kontakt forme, a slika 4.16 prikazuje dio kôda Java kontrolera koji je zadužen sa slanje e-mail obavijesti administratoru stranice. Za slanje email-a je korišten *„JavaMail API“*



Sl. 4.16 Kontakt forma

```
            final MimeMessage mimeMessage = this.mailSender.createMimeMessage();
            final MimeMessageHelper message =
                        new MimeMessageHelper(mimeMessage, true, "UTF-8"); //
true = multipart
            message.setSubject("Pitanje korisnika sa smartbuilding");
            InternetAddress fromAddress = new InternetAddress(emailSenderAddress,
emailSenderName);
            message.setFrom(fromAddress);
            message.setTo(questionRecipients.split(","));
            SpringWebContext ctx = new SpringWebContext(request, response,
request.getServletContext(), Locale.getDefault(), new HashMap<>(),
applicationContext);
            ctx.setVariable("fullName", questionForm.fullName);
            ctx.setVariable("phoneNo", questionForm.phoneNo);
            ctx.setVariable("email", questionForm.email);
            ctx.setVariable("question", questionForm.question);
            final String htmlContent = templateEngine.process("email/user-
question", ctx);
            message.setText(htmlContent, true); // true = isHtml
            mailSender.send(mimeMessage);
            ra.addFlashAttribute("flashMsg", "Dobili smo Vaše pitanje. Odgovor će
biti poslan na email adresu koju ste upisali. Hvala!");
            return "redirect:/kontakt";
```

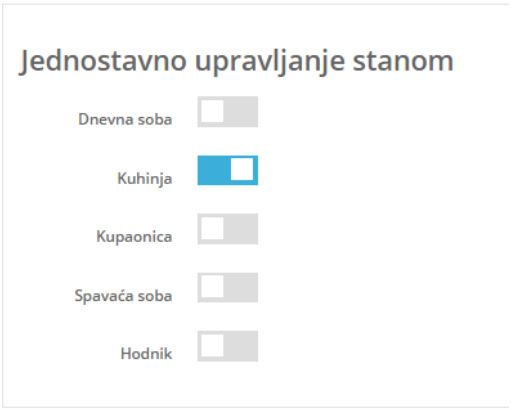
Sl. 4.15 Slanje email-a

4.2.2. Administracijski dio

Pristup stranicama na administracijskom dijelu je onemogućen ukoliko se korisnik nije prijavio, ili je pri tome korištena netočna kombinacija korisničkog imena i lozinke. Forma za prijavu korisnika je prikazana slikom 4.9.

U svrhu ograničenja pristupa administracijskim stranicama korišten je Apache Shiro (poglavlje 3.2.1).

Nakon pristupa administracijskom dijelu korisnik ima mogućnost kontrole rasvjete u stanu. Postoje dva različita načina kontrole. Prvi je trenutna kontrola, gdje promjenom stanja prekidača se pala ili gase svjetla (slika 4.17).



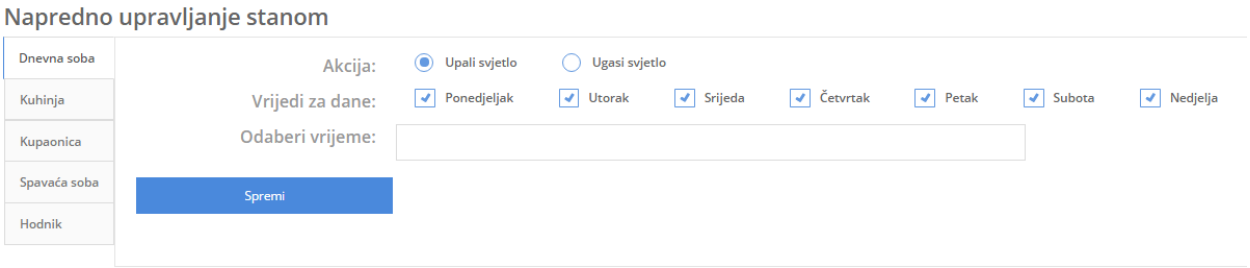
Sl. 4.17 Jednostavno upravljanje

Promjenom vrijednosti pojedinog prekidača poziva se Ajax metoda koja služi za komunikaciju s kontrolerom koji odrađuje željeni zadatak, slika 4.18.

```
$.ajax(url, {
      data: {
            roomToChange: this.id,
            state: checkState
      },
      success: function (data, textStatus, jqXHR) {
            if (data === "SUCCESS") {
                  if ($('#successMessage').is(":visible")) {
                        return;
                  }
                  $('#successMessage').show();
                  setTimeout(function () {
                        $('#successMessage').hide();
                  }, 3000);
            }
            else if (data === "ERROR") {
                  if ($('#errorMessage').is(":visible")) {
                        return;
                  }
                  $('#errorMessage').show();
                  setTimeout(function () {
                        checkBoxId.prop("checked",
!checkBoxId.prop("checked"));
                        $('#errorMessage').hide();
                  }, 3000);
            }
      }
});
```

Sl. 4.18 Ajax metoda

Drugi način je automatizirano upravljanje. Gdje se mogu zadati novi automatski zadatci. Korisnik bira za koju prostoriju će zadatak vrijediti, koja akcija će se napravit (paljenje ili gašenje svjetla), za koje dane vrijedi, te u koje vrijeme će se izvršiti (slika 4.18).



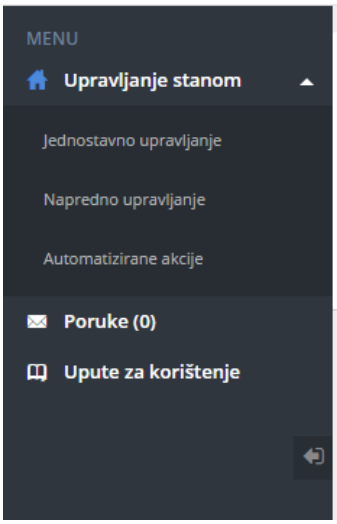
Sl. 4.19 Napredno upravljanje

Na kontroleru zaduženom za dodavanje novih automatiziranih zadataka postoji provjera kako bi se onemogućilo dodavanje istih akcija više puta (slika 4.20)

```
List<ApartmentCronJob> apartmentCronJobs =
apartmentManager.getApartmentCronJobsForRoom(apartment, roomToChange);
            boolean cronJobEqual = false;
            boolean dayListSameSize = false;

            for (ApartmentCronJob apartmentCronJob : apartmentCronJobs) {
                cronJobEqual = false;
                dayListSameSize = false;
                if(apartmentCronJob.getDays().size() == checkedDays.size())
                    dayListSameSize = true;
                else
                    dayListSameSize = false;

                if(dayListSameSize){
                    if(apartmentCronJob.getAction().equals(action) &&
apartmentCronJob.getTime().equals(time) &&
apartmentCronJob.getDays().containsAll(checkedDays)) {
                        cronJobEqual = true;
                        break;
                    }
                }
                else{
                    cronJobEqual = false;
                }
            }
            if(cronJobEqual)
                return "CRONEXISTS";
            else {
                ApartmentCronJob apartmentCronJob = new ApartmentCronJob();
                apartmentCronJob.setApartment(apartment);
                apartmentCronJob.setRoom(roomToChange);
                apartmentCronJob.setAction(action);
                apartmentCronJob.setDays(checkedDays);
                apartmentCronJob.setTime(time);
                apartmentManager.save(apartmentCronJob);
                return "SUCCESS";
            }
```

Sl. 4.20 Provjera automatiziranih akcija

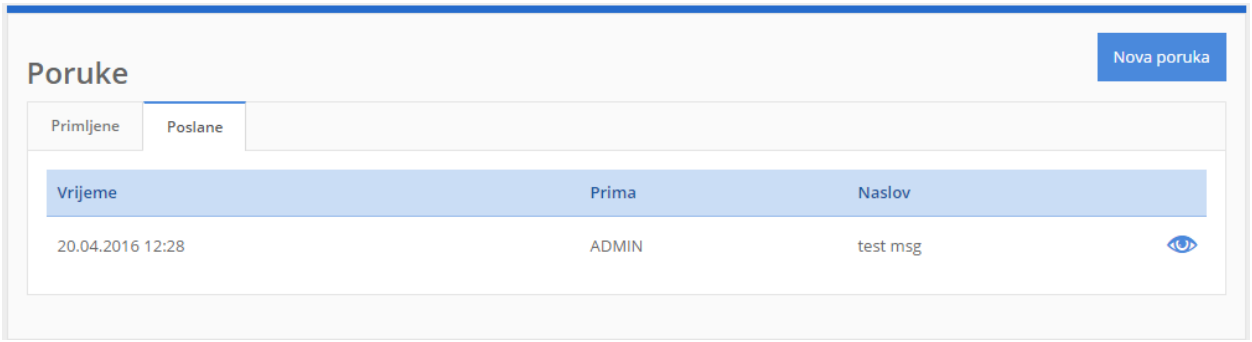
Nakon dodavanja automatiziranih akcija iste će korisniku biti vidljive u tablici koja je dostupna na stranici pod nazivom *„Automatizirane akcije“*. Navedenoj stranici se može pristupiti iz navigacije, koja se nalazi s lijeve strane stranice, slika 4.21. Prilikom pregleda liste kreiranih akcija, korisniku je omogućeno brisanje svake akcije pojedinačno.



Sl. 4.21 Navigacija

Ukoliko je prijavljeni korisnik administrator stranice, prva stranica koja će mu se učitati nakon prijave je pregled aktivnosti korisnika aplikacije. U ovom pregledu administrator može vidjeti svaki stan u zgradi i korisnike koji imaju otvoren profil za upravljanje stanom, te vrijeme njihove zadnje prijave na stranici.

U navigaciji administratorske stranice dostupna je još jedna opcija, a to su *„Poruke“*. Ovdje je omogućena komunikacija između administratora aplikacije i ostalih korisnika, slika 4.22.



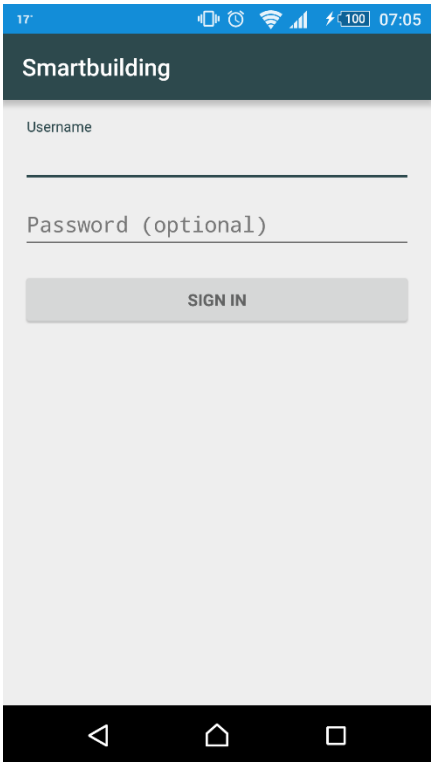
Sl. 4.22 Poruke

4.3. Android aplikacija

U ovom poglavlju je dan pregled korištenih kodova prilikom izrade android aplikacije uz koje su još priložene i slike krajnjeg izgleda aplikacije. Android aplikacija pokriva funkcionalnosti jednostavno upravljanje rasvjetom, dodavanje automatiziranih zadataka, pregled istih i njihovo brisanje.

4.3.1. Realizacija Android aplikacije

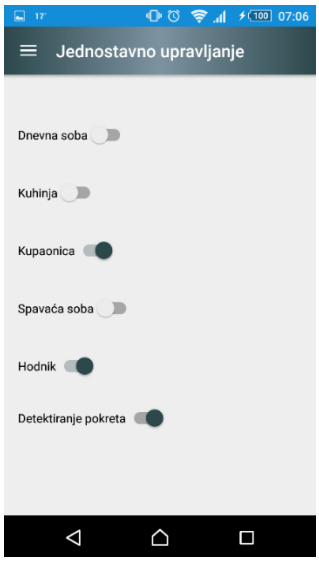
Pristup aplikaciji nije moguć ukoliko korisnik već ne posjeduje korisničko ime i lozinku. Za prijavu se koriste isti podaci s kojima se pristupa Internet aplikaciji.



Sl. 4.23 Prijava korisnika

Slika 4.23 prikazuje izgled aktivnosti za prijavu korisnika na Android aplikaciji. Ova aktivnost ima jako jednostavan dizajn, sastoji se samo od dva polja u koja korisnik treba upisati svoje korisničko ime i lozinku, te se pritiskom na gumb *"Sign In"* ostvaruje veza s poslužiteljem, te se od njega prima odgovarajući odgovor. Kako Android ne sadrži metode kojima bi se moglo odraditi upravljanje sesijama (engl. *Session*), ovaj dio funkcionalnosti je napravljen korištenjem dijeljenih postavki (engl. *Shared Preferences*).

U slučaju uspješne prijave, aplikacija s poslužitelja kao odgovor vraća JSON (engl. *JavaScript Object Notation*) tip podatka u kojem se nalaze podaci o korisniku, kao što su njegovo ime, prezime, stan u kojem živi i sl. Korisnik je zatim preusmjeren na glavnu aktivnost koja je u ovom slučaju *"SimpleLayoutActivity"*.



Sl. 4.24 Jednostavno upravljanje

Slika 4.24 prikazuje izgled glavne aktivnosti gdje korisnik ima mogućnost jednostavnog upravljanja stanom. Kako bi se omogućila ova funckionalnost korišteni su prekidači (engl. *Toggle switch*) koji imaju dva moguća stanja: upaljeno i ugašeno (engl. *On/off*).

```
View.OnClickListener clicks = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int id = v.getId();
            Switch sw = (Switch) findViewById(id);
            switch (id) {
                case R.id.switch1:
                    ApartmentManager.changeRoomState(sw,
getApplicationContext(), apartmentId, "living_room", sw.isChecked());
                    break;
                                        ⋮
                case 6:
                    int thumbColor;
                    if(sw.isChecked()){
                        thumbColor = Color.argb(255, 45, 73, 77);
                        sw.setChecked(true);
                    }else{
                        thumbColor = Color.argb(255, 236, 236, 236);
                        sw.setChecked(false);
                    }
                    sw.getThumbDrawable().setColorFilter(thumbColor,
PorterDuff.Mode.MULTIPLY);
                    ApartmentManager.changeRoomState(sw,
getApplicationContext(), apartmentId, "motionDetection", sw.isChecked());
                    break;
            }
        }
```

Sl. 4.25 Kôd za slušač (engl. ClickListener)

Slika 4.25 prikazuje dio koda koji predstavlja slušač (engl. *Listener*) koji reagira na promjenu vrijednosti pojedinog prekidača iz glavne aktivnosti. U ovisnosti koji prekidač je aktiviran zahtjev se šalje na poslužiteljsku stranu, te se vriši obrada podataka.

Korisniku je dostupan i drugi način upravljanja, koji je prikazan slikom 4.26.

Sl. 4.26 Napredno upravljanje

Ovdje ima mogućnost dodavanja automatiziranih akcija koje će se izvršavati u odabrano vrijeme. Korisnik može izabrati za koju prostoriju vrijedi određena akcija, što se treba napraviti (upaliti ili ugasiti svjetlo), za koje dane u tjednu vrijedi, i u koje vrijeme se ista treba izvršiti.

4.3.2. Komunikacija s poslužiteljem

Kako bi se ostvarila komunikacija s poslužiteljem korišteni su asinkroni zadaci (engl. *AsyncTask*) [18]. Oni nam omogućuju povezivanje na Internet i slanje nekakvih podataka prema poslužitelju.

```
List<NameValuePair> postParameters = new ArrayList<>();
postParameters.add(new BasicNameValuePair("apartmentId",
String.valueOf(apartmentId)));
postParameters.add(new BasicNameValuePair("roomToChange", roomToChange));
postParameters.add(new BasicNameValuePair("state", String.valueOf(state)));


httppost.setEntity(new UrlEncodedFormEntity(postParameters));

// Execute HTTP Post Request
HttpResponse response = httpclient.execute(httppost);
statusCode = response.getStatusLine().getStatusCode();
if (statusCode == HttpStatus.SC_OK)
{
      HttpEntity entity = response.getEntity();
      InputStream is = entity.getContent();
      apartmentLayout = ParseResponse.iStream_to_String(is);

}
else{
      return false;
}
```

Sl. 4.27 Kôd za komunikaciju s poslužiteljem

Slika 4.27 nam prikazuje jedan od primjera *AsyncTask-a* gdje se prema aplikaciji koja se nalazi na poslužitelju šalju podaci o kojem je stanu riječ (*apartmentId*), za koju prostoriju vrijedi odabrana akcija (*roomToChange*), te koja akcija se treba izvršiti (*state*). Kao odgovor sa poslužitelja dolazi nam HttpStatus kod, ako je akcija uspješno izvršena poslužitelj vraća *HttpStatus.OK*, te trenutno stanje prostorija unutar stana (odnosno koja svjetla su upaljena, a koja ugašena). Odgovor sa poslužitelja stiže u *JSON* obliku, slika 4.28a i 4.28b.

```
a)    @RequestMapping(value = "/apartmentLayout/editSimple", method =
RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE)
      @ResponseBody

b)    return new ResponseEntity<>(apartmentLayout, HttpStatus.OK);
```

Sl. 4.28 Odgovor sa poslužitelja

Svi ostali zahtjevi koji se šalju prema poslužitelju su napravljeni na sličan način. U svim slučajevim je korišten *AsyncTask*, samo je razlika u parametrima koji se šalju, te na koji URL se šalje zahtjev.

4.4. RaspberryPi aplikacija

U ovom poglavlju dan je pregled korištenih kodova prilikom izrade aplikacije koja je pokrenuta na RaspberryPi uređaju. Te pregled početnih zahtjeva koje je potrebno zadovoljiti kako bi se aplikacija mogla pokrenuti.

4.4.1. Početni zahtjevi

Prije prvog pokretanja aplikacije potrebno je generirati SSH ključa za uređaj na kojem će se pokrenuti aplikacija. To možemo učiniti pokretanjem naredbe koja je prikazana slikom 4.29.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Sl. 4.29 Generiranje SSH ključa

Zatim je potrebno pokrenuti sljedeću naredbu kako bi se SSH ključ kopirao na poslužitelj gdje se nalazi internet aplikacija i baza.

```
ssh-copy-id root@89.107.57.144
```

Sl. 4.30 Kopiranje SSH ključa

Aplikaciji koja se pokreće na RaspberryPi uređaju potrebno je omogućiti povezivanje na bazu koja se nalazi na poslužitelje, te je s toga prije samog pokretanja aplikacije potrebno otvoriti ssh tunel prema poslužitelju. To možemo učiniti na sljedeći način.

```
ssh -L9995:localhost:5432 root@89.107.57.144
```

Sl. 4.31 Otvaranje SSH tunela

Ukoliko su svi prijašnji koraci uspješno izvedeni, možemo pokrenuti aplikaciju.

```
sudo java -jar Smartbuilding-1.0.jar
```

Sl. 4.32 Pokretanje aplikacije

4.4.2. Realizacija RaspberryPi aplikacije

Pokratanjem aplikacije korištenjem naredbe prikazane slikom 4.32, pokreće se *main* metoda. Prilikom prvog pokretanja aplikacije sva se svjetla u maketi ugasi, odnosno vrijednost svih GPIO (engl. *General-purpose input/output*) pinova koji se koriste se postavlja na logičku nulu, slika 4.33.

```
private static void initPinStates(HashMap<Long, HashMap<String, Pin>>
apartmentGpios){
	final GpioController gpio = GpioFactory.getInstance();
	for (HashMap<String, Pin> stringPinHashMap : apartmentGpios.values())
{
		for (Pin pin : stringPinHashMap.values()) {
			final GpioPinDigitalOutput p =
gpio.provisionDigitalOutputPin(pin, "init", PinState.LOW);
			gpio.unprovisionPin(p);
		}
	}
	gpio.shutdown();
}
```

Sl. 4.33 Postavljanje pinova na logičku nulu

Nakon toga se pokreće metoda tipa *Runnable*, kojoj je definirano periodičko izvođenje svake sekunde. Na ovaj način se svake sekunde vrši provjera stanja, koja svjetla trebaju biti upaljena, a koja ugašena u svim stanovima, slika 4.34.

```
Runnable apartments = new Runnable() {
	@Override
	public void run() {
		MakeConnection.getApartments(apartmentGpios);
	}
};

ScheduledExecutorService executorApartment =
Executors.newScheduledThreadPool(1);
executorApartment.scheduleAtFixedRate(apartments, 0, 1, TimeUnit.SECONDS);
```

Sl. 4.34 Pokretanje Runnable metode

Kao što vidimo iz slike 4.34 prilikom svakog pozivanja *Runnable* metode poziva se metoda *getApartments* iz *MakeConnection* klase. Unutar nje se uspostavlja veza s bazom koja se nalazi na poslužitelju.

Slika 4.35 nam prikazuje dio kôda za uspotavljanje veze s bazom.

```
Class.forName("org.postgresql.Driver");
c = DriverManager

      .getConnection("jdbc:postgresql://127.0.0.1:9995/smartbuilding",
"m2stanic", "m2stanic");
c.setAutoCommit(false);

stmt = c.createStatement();

String select = " select * from apartment layout;";
```

Sl. 4.35 Uspostavljanje veze s bazom

Nakon što se dohvate podaci za sve stanove i stanja nihovih soba, odnosno koja svjetla su upaljena a koja ugašena, poziva se metoda *processRoom* iz klase *ControlGpio*.

```
private static void processRoom(Pin pinToChange, boolean action){
      final GpioController gpio = GpioFactory.getInstance();
      PinState wantedState = null;
      //to get room and action we need
      if (action)
            wantedState = PinState.HIGH;
      else
            wantedState = PinState.LOW;

      GpioPinDigitalOutput p = gpio.provisionDigitalOutputPin(pinToChange);
      PinState currentPinState = p.getState();
      System.out.println("MANUAL | pin: " + pinToChange.getName() + ",
current state: " + currentPinState.getName() + ", wanted state: " +
wantedState.getName());

      if(!currentPinState.equals(wantedState))
            p.setState(wantedState);

      gpio.shutdown();
      gpio.unprovisionPin(p);
}
```

Sl. 4.36 processRoom metoda

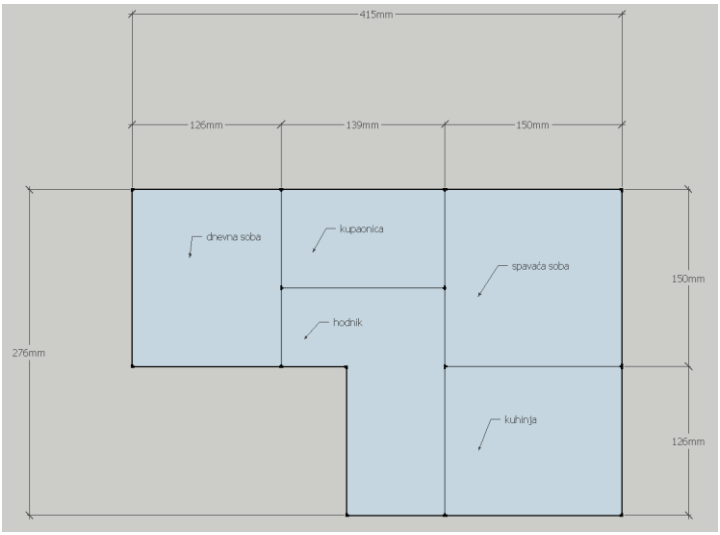
U ovoj metodi se za svaku prostoriju provjerava željeno stanje svjetla, te se ono uspoređuje s trenutnim stanjem. Ako su ova dva stanja jednaka, neće se dogoditi nikakva promjena. Ukoliko se stanja razlikuju GPIO pin će se postaviti na željeno stanje.

Za ovu aplikaciju nije razvijen nikakav oblik grafičkog sučelja s obzirom da aplikacija ne zahtjeva nikakav oblik direktne interakcije korisnika s aplikacijom. Svi podaci se dohvaćaju iz baze koja se nalazi na poslužitelju na način prikazan slikom 4.35.

5. IZRADA MAKETE

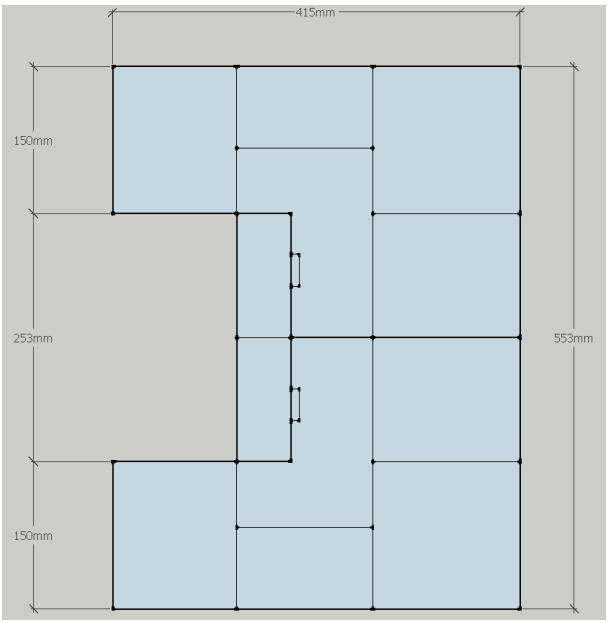
Za potrebe ovog diplomskog rada napravljena je i maketa, kako bi se na praktičan način mogao demonstrirati sustav koji je razvijen. Maketa je izrađena u mjerilu 1:65.

Maketa zgrade se sastoji od 3 stana, svaki od njih ima 5 prostorija. Dimenzije stana i raspored prostorija su prikazane slikom 5.1.



Sl. 5.1 Dimenzije stana

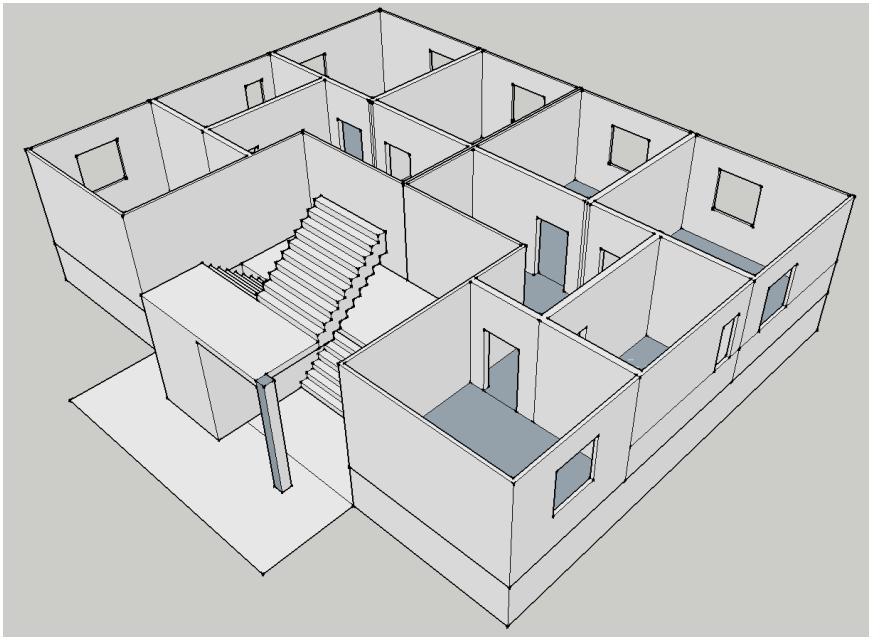
Dimenzije makete i izgled prvog kata su prikazani slikom 5.2.



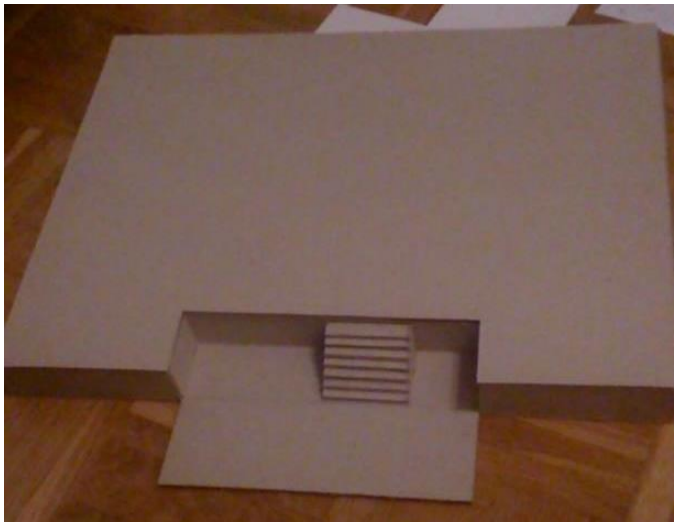
Sl. 5.2 Dimenzije prvog kata

U stanovima koji se nalaze na prvom katu omogućena je samo jednostavno i napredno upravljanje svjetlima unutar stana. Dok je u stanu na drugom katu implementirana i opcija detekcije pokreta. U tu svrhu je korišten senzor *HC-SR501*.

Slika 5.3 predstavlja 3D model prvog kata zgrade.



Sl. 5.3 3D model



Sl. 5.4 Početak izrade makete

Sl. 5.5 Prvi kat

Na slici 5.6 vidimo gotovu maketu s vidljivim senzorom za detekciju pokreta koji se koristi u stanu na drugom katu.

Sl. 5.6 Dovršena maketa

6. ZAKLJUČAK

Glavni cilj ovog rada bilo je proučiti postojeća rješenja za upravljanje sustavim unutar zgrade. Odnosno proučiti što je to pametna zgrada, te koje karakteristike mora imati neka zgrada da bi se mogla svrstati u ovu kategoriju. Također je trebalo razviti neki oblik jednostavnog programskog rješenja. U tu svrhu napravljena je internet stranica, mobilna (Android) aplikacija, te aplikacija za RaspberryPi.

Prije same analize i objašnjenja pojedinih kodova i na koji način su realizirane navedene aplikacije, dan je kratak osvrt na tehnologije koje su korištene. Te su tako objašnjeni HTML, CSS, JavaScript, jQuery i Ajax. Također je objašnjena i Java, kao i neke njene biblioteke koje su korištene prilikom izrade programskog rješenja. Prilikom izrade programskog rješenja pojavili su se neki problem, na poslužitelju nije bilo moguće otvoriti portove na kojima se nalazi baza podataka, i na taj način omogućiti vanjski pristup bazi, te se iz tog razloga prilikom pokretanja aplikacije na RaspberryPi uređaju prvo mora kreirati SSH tunel prema poslužitelju.

Ovaj sustav je moguće dodatno proširivati i povećavati broj funkcionalnosti, pri tome treba voditi brigu o broju preostalih GPIO pinova na RaspberryPi. Ukoliko postoji potreba za dodatnim pinovima moguće je dodati Arduino uređaj i ostvariti serijsku komunikaciju između Arduina i RaspberryPi. Ovim rješenjem bi se smanjila pouzdanost cijelog sustava, jer bi bila dodana još jedna točka u kojoj bi moglo doći do kvara.

LITERATURA

[1] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Building_automation. [Pokušaj pristupa Siječanj 2016].

[2] »Siemens,« [Mrežno]. Available: http://www.siemens.com/innovation/en/home/pictures-of-the-future/infrastructure-and-finance/smart-cities-smart-buildings.html. [Pokušaj pristupa Siječanj 2016].

[3] »Intel,« [Mrežno]. Available: http://www.intel.com/content/www/us/en/internet-of-things/smart-buildings.html. [Pokušaj pristupa Siječanj 2016].

[4] »eSight energy,« [Mrežno]. Available: http://www.esightenergy.com/uk/. [Pokušaj pristupa Siječanj 2016].

[5] »BuildingIQ,« [Mrežno]. Available: https://www.buildingiq.com/. [Pokušaj pristupa Siječanj 2016].

[6] »Wikipedia,« [Mrežno]. Available: https://hr.wikipedia.org/wiki/Java_(programski_jezik). [Pokušaj pristupa Ožujak 2016].

[7] »Apache Shiro,« [Mrežno]. Available: http://shiro.apache.org/. [Pokušaj pristupa Travanj 2016].

[8] »Fakultet elektrotehnike i računarstva,« [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/2._HTML.pdf. [Pokušaj pristupa Ožujak 2016].

[9] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/HTML5. [Pokušaj pristupa Travanj 2016].

[10] »Wikipedia,« [Mrežno]. Available: https://hr.wikipedia.org/wiki/HTML. [Pokušaj pristupa Travanj 2016].

[11] »Wikipedia,« [Mrežno]. Available: https://hr.wikipedia.org/wiki/CSS. [Pokušaj pristupa Travanj 2016].

[12] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets#CSS_3. [Pokušaj pristupa Svibanj 2016].

[13] »Wikipedia,« [Mrežno]. Available: https://hr.wikipedia.org/wiki/JavaScript. [Pokušaj pristupa Svibanj 2016].

[14] »Thymeleaf,« [Mrežno]. Available: http://www.thymeleaf.org/index.html. [Pokušaj pristupa Lipanj 2016].

[15] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Android_version_history. [Pokušaj pristupa Svibanj 2016].

[16] »Cprogramming,« [Mrežno]. Available: http://www.cprogramming.com/android/android_getting_started.html. [Pokušaj pristupa Lipanj 2016].

[17] »BasicContactables,« Google, [Mrežno]. Available: https://developer.android.com/samples/BasicContactables/AndroidManifest.html. [Pokušaj pristupa Svibanj 2016].

[18] »AsyncTask,« Google, [Mrežno]. Available: https://developer.android.com/reference/android/os/AsyncTask.html. [Pokušaj pristupa Lipanj 2016].

[19] »Fakultet elektrotehnike i računarstva,« [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/PREDAVANJE_4w.pdf. [Pokušaj pristupa Siječanj 2016].

[20] »IEEE,« [Mrežno]. Available: http://www.ieee.hr/_download/repository/ZR09MMedjugorac.pdf. [Pokušaj pristupa Siječanj 2016].

SAŽETAK

Cilj diplomskog rada je izrada makete zgrade i stvaranje jednostavnog računalnog sustava za nadzor i upravljanje sustavima unutar zgrade. Opisana su trenutna postojeća rješenja, kao što su Esight Energy i BuildingIQ. Praktični dio rada se sastoji od internetske stranice, Android aplikacije, RaspberryPi aplikacije te same makete. Prije analize, obrađene su tehnologije potrebne za realizaciju pojedine komponente praktičnog dijela rada. Od svih obrađenih tehnologija, najveću primjenu je imao programski jezik Java. Prilikom analize realizacije praktičnog dijela rada, navedeni su početni zahtjevi, te su posebno analizirane relizacije internetske aplikacije (stranice), Android aplikacije i RaspberryPi aplikacije. Priloženi su odgovarajući kodovi i slike koje opisuju. Tijekom obrade stvaranja makete priloženi su odgovarajući 3D modeli i fotografije krajnje makete.

Ključne riječi: Android, internetska stranica, RaspberryPi, pametna zgrada, maketa

ABSTRACT

Smart building

The point of the this thesis was the production of a model building and the creation of a simple computer system for controling and managing the systems within the building. Existing solutions, such as EsightEnergy and BuildingIQ, were described. Practical part of the thesis consists of a web-page, Android application, RaspberryPi application and the model building. Before the analysis, technologies for the individual components of the practical part were covered. Of all the covered technologies, Java programming language had the largest application. While analyzing the practical part, the inital requirements were given, along with specific analysis of the Internet application, Android application and RaspberryPi application. Appropriate codes were given along with screenshots they describe. While covering the creation of the model building, corresponding 3D models and pictures of the finished product were attached.

Key words: Android, web-page, RaspberryPi, smart-building, model

ŽIVOTOPIS

Mato Stanić je rođen 17.2.1992. u Odžaku, BiH. Početkom 2010. godine završava III. Gimnaziju u Osijeku, nakon čega se iste te godine upisuje na Elektrotehnički fakultet u Osijeku. Sudjelovao je na nekoliko međunarodnih projekata. U razdoblju od rujna 2008. do svibnja 2009. sudjeluje na međunarodnom projektu „Upoznajmo Kopački Rit i Neusiedler See“. U svibnju 2009. godine sudjeluje na projektu Europski parlament mladih Hrvatske, trodnevna simulacija rada europskog parlamenta. U kolovozu 2012. je bio član projekta EUROCAMP 2012 s temom „Sudjelovanje mladih u mjesnoj politici“. 2013 godine završava preddiplomski studij računarstva, te stječe akademski naziv Prvostupnik inženjer računarstva.