Univerza v Ljubljani

Fakulteta za gradbeništvo in *geodezijo*

Jamova 2
1000 Ljubljana, Slovenija
telefon: (01) 47 68 500
faks: (01) 42 50 681
fgg@fgg.uni-lj.si

VISOKOŠOLSKI ŠTUDIJ
GRADBENIŠTVA
KONSTRUKCIJSKA SMER

Kandidat:

TOMAŽ LIPUŠČEK

APLIKACIJA ZA ZAJEMANJE PODATKOV O KONSTRUKCIJSKIH SKLOPIH OBSTOJEČIH STAVB ZA PAMETNE TELEFONE

Diplomska naloga št.: 492/KS

AN APPLICATION FOR GATHERING DATA ON CONSTRUCTION ELEMENTS OF EXISTING BUILDINGS FOR SMART PHONES

Graduation thesis No.: 492/KS

Mentor:
doc. dr. Vlado Stankovski

Predsednik komisije:
doc. dr. Tomo Cerovšek

Ljubljana, 11. 10. 2013

STRAN ZA POPRAVKE

Stran z napako	Vrstica z napako	Namesto	Naj bo

IZJAVA O AVTORSTVU

Podpisani Tomaž Lipušček izjavljam, da sem avtor diplomskega dela z naslovom »APLIKACIJA ZA ZAJEMANJE PODATKOV O KONSTRUKCIJSKIH SKLOPIH OBSTOJEČIH STAVB ZA PAMETNE TELEFONE«.

Izjavljam, da je elektronska različica v vsem enaka tiskani različici.

Izjavljam, da dovoljujem objavo elektronske različice v repozitoriju UL FGG.

Tolmin, 4.10.2013

(podpis)

BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK

UDK:	**004:621.39:624.04(043.2)**
Avtor:	**Tomaž Lipušček**
Mentor:	**doc. dr. Vlado Stankovski**
Naslov:	**Aplikacija za zajemanje podatkov o konstrukcijskih sklopih obstoječih stavb za pametne telefone**
Tip dokumenta:	**Dipl. nal. – VSŠ**
Obseg in oprema:	**69 str., 3 pregl., 37 sl., 14 en.**
Ključne besede:	**Android, baza konstrukcijskih sklopov, toplotni prehod, JDBC, MySQL**

Izvleček

Stavbni fond v Sloveniji sestavljajo stavbe iz različnih časovnih obdobij. Posledično so energijski kazalniki stavb zelo različni. Če bi podatke o stanju stavb sistematično zbirali, bi nad stavbnim fondom imeli večji pregled in možnost nadaljnjih analiz.

Z množično uporabo pametnih telefonov in njihovo priročnostjo bi lastnike in upravljavce stanovanj lažje prepričali, da bi za svoje stanovanje s pomočjo mobilne aplikacije sami izpolnili potrebne podatke in se informirali o trenutnem energetskem stanju svojega objekta ter možnostih za njegovo izboljšavo.

Za diplomsko delo smo izdelali mobilno aplikacijo, kjer uporabnik vnese podatke o želenem objektu in po poenostavljeni metodologiji izračunov dobi informativne rezultate o njegovih toplotnih izgubah. Informacije o konstrukcijskih sklopih se pošljejo centralnemu strežniku, ki zbira podatke vseh stavb analiziranih s pomočjo aplikacije.

Aplikacija zajema informacije o lokaciji, merah objekta, režimu obratovanja in o površinah ter sestavah konstrukcijskih sklopov. Omogoča tudi shranjevanje in odpiranje obstoječih popisov stavb. Podatki se pred pošiljanjem shranjujejo v pametni napravi s pomočjo interne baze SQLite. Za pošiljanje podatkov do centralnega strežnika, kjer je nameščena baza MySQL, smo uporabili gonilnik JDBC in namestitev dodatne aplikacije za strežnik (ti. servleta).

Aplikacija je bila izdelana za mobilni operacijski sistem Android. Napisana je bila v programskem jeziku java v integriranem razvojnem okolju Eclipse s pomočjo Android Development Tools vključka.

Rezultat diplomskega dela je koristno orodje, ki bi pripomoglo k ozaveščanju in ukrepanju v smeri zmanjševanja porabe energentov v stavbah.

BIBLIOGRAPHIC-DOCUMENTALISTIC INFORMATION

UDC:	**004:621.39:624.04(043.2)**
Author:	**Tomaž Lipušček**
Supervisor:	**Assist. Prof. Vlado Stankovski, Ph.D.**
Title:	**An application for gathering data on construction elements of existing buildings for smart phones**
Document type:	**Graduation Thesis – Higher professional studies**
Notes:	**69 p., 3 tab., 37 fig., 14 eq.**
Key words:	**Android, database of structural elements, heat transfer, JDBC, MySQL**

Abstract

Housing stock in Slovenia consists of buildings from different periods of time. As a result, there are wide differences in building's energy indicators. If their condition data would be systematically collected, there would be greater possibility of housing stock review and further analysis.

Convenient usage of smart phone software could make it easier for housings owners and managers to fill the mobile application's form and inform them about the facility's current energy status and the possibilities for its improvement.

For our thesis we designed a mobile application that, using a simplified calculation methodology based on the information previously filled in by the user, provides him with indicative results about building's heat losses. Information of structural elements is sent to a central server that collects data from all buildings that have been analyzed with this application.

The Application collects information about the facility's location and dimensions, the temperature inside the building and surface areas with its structural elements. The existing inventory of buildings can be saved and opened to/from SD card. Before the data is sent, it is stored in the smart device using internal database SQLite. To send the data to a central server where MySQL database is installed, we used the JDBC driver and installation of a servlet application on server.

The Application was designed for Android mobile operating system. It was written in Java programming language in an integrated development environment Eclipse using Android Development Tools plug-in.

The result of thesis is a useful tool which could help raising awareness and take action towards the reduction of energy consumption in buildings.

ZAHVALA

Zahvalil bi se mentorju doc. dr. Vladu Stankovskemu za idejo, potrpežljivost in čas pri izdelavi mojega diplomskega dela.

Hvala tudi moji družini, ki mi je v času šolanja potrpežljivo stala ob strani. Brez njih tega diplomskega dela ne bi bilo.

Hvala.

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO GRAFIKONOV

KAZALO SLIK

1 UVOD

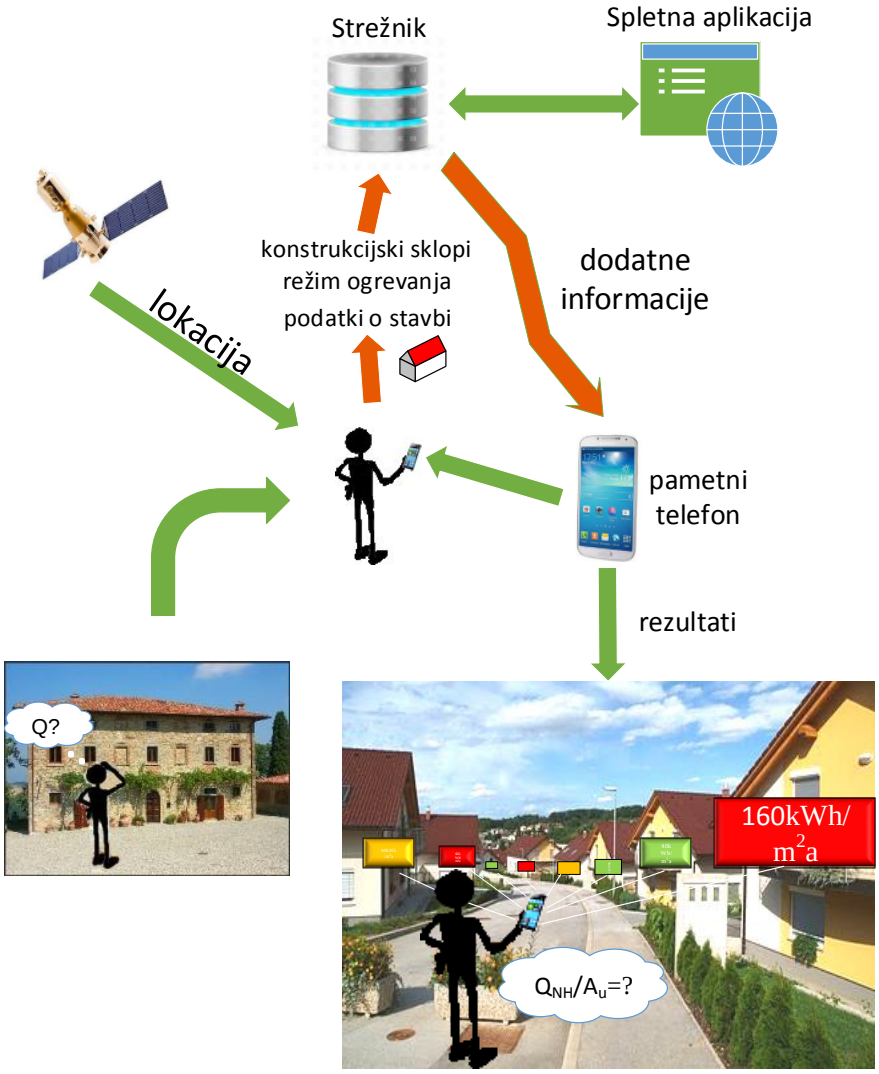
Z obsežnim trgom mobilne telefonije in uporabnostjo, ki jo pametni telefoni ponujajo, se odpirajo možnosti za njihovo izkoriščanje na področju gradbeništva. Po raziskavah mednarodne telekomunikacijske unije [1] je globalno v letu 2013 število uporabnikov mobilnih telefonov doseglo 6.8 milijard, kar ustreza 96 odstotkom vseh prebivalcev sveta. Število uporabnikov pametnih telefonov je lansko leto (2012) preseglo 1 milijardo. Glede na obete (po [2]) bo samo v letu 2013 prodana nova milijarda pametnih telefonov. Tudi v prihodnosti se predvideva povečan delež prodaje pametnih telefonov glede na t.i. klasične. S tem se povečuje število ljudi, ki bi izkoriščalo znanje gradbeništva s pomočjo aplikacij.

Z današnjimi potrebami po zeleni gradnji, je ena od možnosti aplikacija, ki bi služila določanju energetske učinkovitosti posameznih stanovanjskih enot, hiš, zgradb. Uporabnik pametnega telefona bi na enostaven način lahko zajemal njihove konstrukcijske sklope z vsebovanimi podatki ter za svoje stanovanje pregledal, koliko toplotnih izgub ima oziroma, s kakšnimi izboljšavami bi jih znižal. S pomočjo telefona, v katerega bi vnesli želene informacije o stavbi in njene konstrukcijske sklope, bi posredovali podatke oddaljeni bazi. Do te baze bi imela dostop spletna aplikacija z dodatnimi informacijami, ki bi izvedla račune in jih posredovala nazaj bazi oziroma uporabniku telefona. S sistematičnim zbiranjem podatkov bi dobili boljši pregled nad energetskim stanjem stavbnega fonda v Sloveniji. Podatke bi bilo možno uporabiti tudi za nadaljnje analize.

Pri realiziranju aplikacije smo uporabili podatke iz diplomske naloge Uporaba tehnologije OntoWiki pri tipizaciji stavb v Sloveniji [3], kjer je predstavljena spletna aplikacija s katero bi uporabniku na hiter in enostaven način pokazala energijske kazalnike njegove stavbe. V tej aplikaciji je vgrajena približna ocena na podlagi Evropskega projekta Inteligentna energija Evropa Tabula (IEE Tabula). Projekt IEE Tabula je zgradil enotno Evropsko strukturo tipologije stanovanjskih objektov. Države udeleženke so izdelale model za razvrščanje stavb s podobnimi energetskimi lastnostmi v posamezne razrede.

1.1 Analiza možnega poteka aplikacije na pametni napravi

Z namenom določitve nujnih nalog, ki jih mora aplikacija opraviti, je v nadaljevanju predstavljena okvirna analiza možnega poteka. Aplikacija bi morala zbirati informacije od uporabnika na preprost način. Z nekompliciranim uporabniškim vmesnikom bi uporabnik kar se da hitro izpolnil vse potrebne informacije.



Slika 1: Praktičen primer uporabe aplikacije na pametnem telefonu

Na sliki 1 je prikazan možen potek take aplikacije. Sprva bi aplikacija samodejno zaznala lokacijo uporabnika in ga locirala na zemljevidu. Z njegovo pomočjo ali preko iskalnika bi dobili stavbo, ki jo želimo obravnavati. Za zajem podatkov izbrane stavbe bi vključili 4 večje skupine podatkov:

- lokacijo
- splošen opis stavbe
- režim ogrevanja v stavbi
- konstrukcijski sklopi

Do zaključka popisa bi se podatki najprej zbirali v notranji bazi telefona. Omogočeno bi bilo shranjevanje začasnega ali končnega popisa stavbe na spominsko kartico v telefonu. To bi prišlo prav tudi v primeru, če v času popisa povezava z internetom ne bi bila mogoča. Ko bi bil popis stavbe popoln, bi poslali preko mobilne podatkovne povezave ali povezave preko Wi-Fija paket informacij oddaljenemu strežniku, ki bi shranil vse informacije o objektu.

Glede na to, da z zajemom konstrukcijskih sklopov lahko računamo tudi druge parametre, bi aplikaciji lahko dodali tudi informativni izračun toplotnega toka skozi stavbo *P* [W] in toplotne izgube skozi stavbo [W/K]. Uporabnik bi tako lahko preizkusil različne možnosti dodatnih izolacij oziroma drugih materialov v konstrukcijskem sklopu in videl kolikšne so toplotne izgube ter toplotni tok v stavbah pri različnih notranjih in zunanjih temperaturah.

1.2 Pridobivanje podatkov za energetske izkaznice na pametnih napravah

Pametni telefoni ponujajo dodatne senzorje, ki jih pri običajnih računalnikih ni, zato bi aplikacije, ki bi bile narejene za te naprave omogočale lažje in hitrejše popise za analizo stavb. Obstajajo namreč možnosti, da se s pomočjo teh senzorjev poenostavijo popisi in izračuni za izdajanje energetskih izkaznic, ki jih trenutno v večini še opravljajo programi na običajnih računalnikih. Aplikacija, ki smo jo razvili ponuja zasnovo tudi za te namene, vendar bi jo bilo potrebno dodatno razviti do njene uporabne vrednosti.

1.3 Stavbni fond v Evropi in Sloveniji ter možnosti za izboljšavo

V članicah EU se stavbni fond močno razlikuje glede starosti, tipa, lastništva in energijske učinkovitosti. Posledično so tudi konstrukcijski sklopi, ki obsegajo to področje sestavljeni iz širokega spektra materialov. Nujno bi bilo sistematično zbiranje podatkov o stavbnem fondu, ki bi vključeval podatke o konstrukcijskih sklopih teh objektov, saj bi s temi podatki lažje opredelili kam intenzivneje usmeriti sredstva za spodbujanje prenove.

Stavbni fond v Sloveniji je razmeroma star in grajen v glavnem pred časom uporabe večjih količin izolacije v stavbni ovoj. 38% stanovanjskih zgradb v Sloveniji je zgrajenih pred letom 1960 in le 20% je narejenih v zadnjih dvajsetih letih (1991 - 2010) [4]. Podatkov o številu naknadnih prenov, globin posegov ali posodobitev v bodoče ni veliko na razpolago. Večina virov naznanja, da se povprečno letno v EU prenavlja med 1% in 2% stavbnega fonda.

Spodbude pri sofinanciranju energetske prenove stavb (zlasti v obliki subvencij in davčnih olajšav) uporablja predvsem pri individualnih energijsko učinkovitih komponentah, kar vodi v 10% – 30% izboljšanje pri energetski porabi [5]. Potrebno bi bilo, da bi prihajalo do celostnih prenov z 50% - 80% izboljšanjem [6].

Uporaba aplikacij za evidentiranje stavbnega fonda s konstrukcijskimi sklopi, bi omogočile hitrejše analize stavbnega fonda, kar bi pomenilo tudi možnost hitrejših ukrepov za različne spodbude.

»Ta stran je namenoma prazna«

2 MOBILNI OPERACIJSKI SISTEMI NA PAMETNIH TELEFONIH

Da bi lahko razvili aplikacijo za pametni mobilni telefon s katero bi bilo možno zajemanje podatkov ter ocenitev energetskih izgub smo najprej morali preučiti tehnologije, ki se uporabljajo pri izdelavi aplikacij za pametne telefone. V nadaljevanju so predstavljeni mobilni operacijski sistemi (podrobneje platforma Android), na katerih bi aplikacija lahko delovala.

Sodobni mobilni operacijski sistemi združujejo funkcije operacijskih sistemov osebnih računalnikov z drugimi, ki vključujejo zaslon na dotik, prenosni telefon, tehnologijo Bluetooth, WiFi, GPS navigacijo, kamero, videokamero, prepoznavo glasu, predvajalnik glasbe in podobno.

Sistemi, ki v letu 2013 prevladujejo na pametnih telefonih so Googlov Android, Applov iOS, Windows Phone ter BlackBerry OS. Po prodaji in številu aplikacij za pametne telefone, platformi Android in iOS zavzemata največji delež; Android v letu 2013 z milijonom razpoložljivih aplikacij ter iOS z 900.000 aplikacijami [11].

Aplikacije so napisane za različne mobilne operacijske sisteme. Vsak ima svoj priporočljiv način izgradnje aplikacij, kot tudi programski jezik za njihovo izgradnjo. Za Android aplikacij se uporablja java, za iOS objektivni C ter za Windows Phone C# ali Visual Basic.

Ker največ ljudi uporablja Android in nudi obsežno podporo pri učenju programskega jezika, smo se za izvedbo aplikacije odločili zanj. V nadaljevanju so predstavljene tehnologije, ki se uporabljajo v mobilnem operacijskem sistemu Android. Poglavja so povzeta po [12].

2.1 Operacijski sistem Android

Android je bil primarno oblikovan za namene uporabe na mobilnih napravah, ki imajo zaslone na dotik in se uporabljajo kot pametni telefoni in tablični računalniki. Začet je bil s strani podjetja Android, Inc., ki ga je Google finančno podprl in kasneje kupil v letu 2005. Razkrit je bil leta 2007, skupaj z ustanovitvijo Open Handset Allience; konzorcij strojne opreme, programske opreme in telekomunikacijskih podjetij, namenjenih za pospeševanje odprtih standardov za mobilne naprave. Prvi telefon z operacijskim sistemom Android je bil prodan v letu 2008.

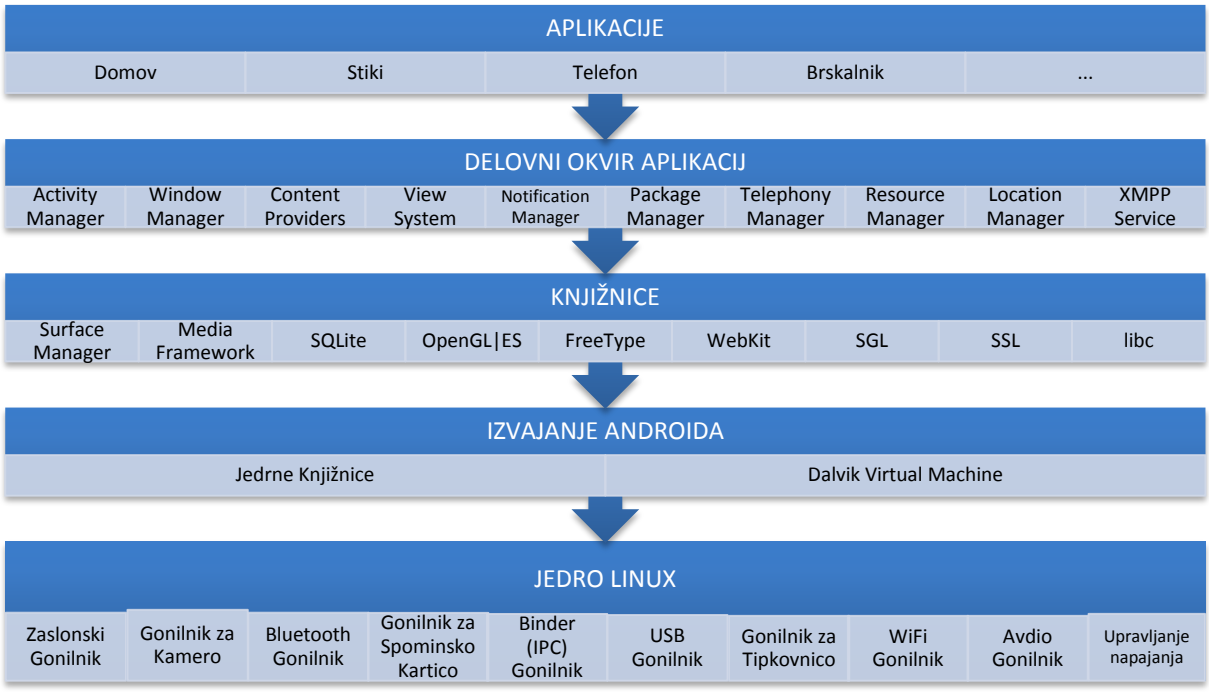
Priljubljenost mobilnega operacijskega sistema je povzročila, da je leta 2010 prehitel takrat najbolj razširjeno platformo Symbian. Postal je programska izbira za tehnološka podjetja, ki potrebujejo poceni, prilagodljiv in lahek operacijski sistem za visokotehnološke naprave, brez potrebe po razvijanju programske opreme od začetka. Kljub temu, da je bil prvotno namenjen predvsem telefonom in tabličnim računalnikom, se uporablja tudi za televizorje, igralne konzole, digitalne fotoaparate in druge elektronske naprave. Androidova odprtost je spodbudila veliko skupnost razvijalcev in navdušencev, da uporabijo odprto izvorno kodo kot temelj za druge usmerjene projekte, ki dodajajo nove funkcije naprednim uporabnikom ali prenašajo Android na naprave, ki so prvotno tekle na drugih operacijskih sistemih.

2.1.1 Arhitektura sistema Android

Android je sestavljen iz *jedra, ki temelji na Linuxovem jedru verzije 2.6 oziroma od različice Android 4.0 Ice Cream Sandwich dalje na jedru 3.x s programsko – strojno opremo (*middleware), knjižnicami in programskimi vmesniki napisanih v jeziku C. Programske aplikacije tečejo na *delovnem okvirju aplikacije, ki vključuje Java združljive knjižnice, katere temeljijo na *Apache Harmony standardu. Android uporablja *Dalvikov navidezni stroj, ki poganja Dalvik "dex-kodo", ki je navadno prevedena iz Java bitne kode. Glavna strojna platforma za Android je *ARM arhitektura.

Androidovo Linux jedro ima dodatne arhitekturne spremembe zunaj tipičnega jedra Linuxovega razvojnega cikla. Android privzeto nima *X Window Sistema, niti ne podpira celotnega nabora standardnih knjižnic GNU, kar otežuje premoščanje obstoječih aplikacij za Linux ali knjižnic na Android. Podpora za enostavne C in SDL aplikacije je omogočena z vgradnjo majhne Java knjižnice (*shima) in uporabo Java Native Vmesnika (*JNI).

Arhitektura operacijskega sistema s plastmi je predstavljena na sliki 2. Operacijski sistem Android si lahko predstavljamo kot programski skupek različnih plasti, kjer ima vsaka plast različne programske komponente. Skupaj tvorijo operacijski sistem, *middleware ter pomembnejše aplikacije. Vsaka plast zagotavlja v arhitekturi programa različne storitve plastem tik nad njo.



Slika 2: Diagram arhitekture operacijskega sistema Android

Medtem ko je večina aplikacij Android napisana v Javi, obstajajo številne razlike med Javinim in Androidovim aplikacijsko programerskim vmesnikom (Java *API, Android *API). Android namreč ne uporablja Javinega navideznega stroja, ampak *Dalvikov.

Android tudi ne uporablja Javinih uveljavljenih standardov, kot sta Java SE in ME. To preprečuje združljivost med Java aplikacijami napisanih na teh platformah in tistih za platformo Android. Android uporablja le jezikovno sintakso in semantiko Jave, vendar ne

* beseda je pojasnjena v poglavju TERMINOLOŠKI SLOVAR

zagotavlja polnega razreda knjižnic in aplikacijsko programskega vmesnika *(API) v paketu z Java SE ali ME.

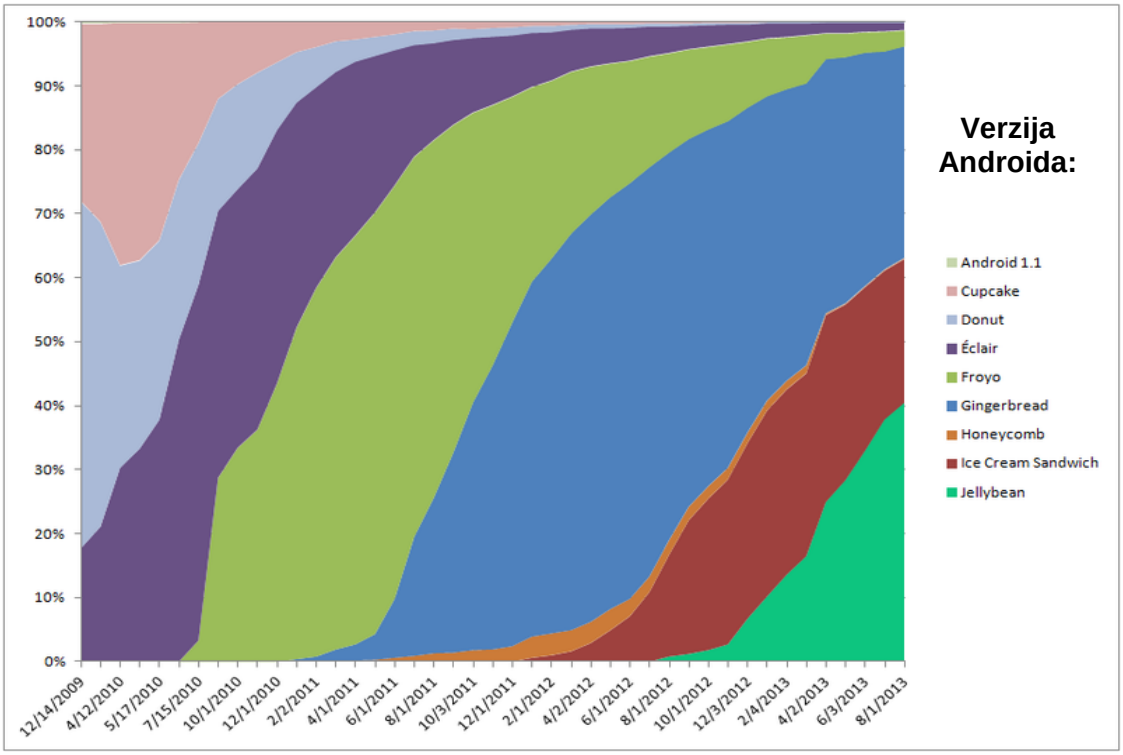
Aplikacije so pakirane v .apk formatu, ki vsebuje .dex datoteke (ustvarjene "Dalvik izvršljive" bitne kode), datoteke virov itd.

2.1.2 Upravljanje s pomnilnikom

Ker je Android baterijska naprava je, za razliko od namiznih operacijskih sistemov, ki so praviloma povezani z neomejenim napajanjem iz električnega omrežja, zasnovan tako, da upravlja s pomnilnikom (RAM), da bi bila poraba energije minimalna. Ko aplikacije niso več v uporabi, jih sistem samodejno naloži v notranji pomnilnik. Aplikacija je še vedno tehnično "odprta", a za delovanje ne porablja nobenih virov (npr. baterijske moči ali procesorske moči). Stoji v pripravljenosti, dokler ni ponovno uporabljena. To ima dvojno korist, povečanje splošne odzivnosti naprave Android, ker aplikacije ni potrebno zapreti in jo ponovno odpreti od začetka ter drugo, da aplikacije v ozadju ne izgubljajo energije po nepotrebnem. Android samodejno upravlja aplikacije, shranjene v pomnilniku. Ko zmanjka pomnilnika, začne sistem z zapiranjem aplikacij in procesov, ki so bili največ časa neaktivni.

2.1.3 Različne verzije operacijskega sistema Android

Delež, ki ga imajo naprave v posameznem letu, je predstavljen v grafikonu 3 [13]. Verzije operacijskega sistema Android se stalno posodabljajo. Zadnja izdaja je 4.3 Jellybean. Večje spremembe videza in knjižnic so bile narejene po verziji 2.3.4 Gingerbread tudi zaradi novejših vgrajenih komponent v Android napravah. Aplikacijo, ki je bila ustvarjena za to diplomsko delo, smo izvajali in testirali na verziji 2.3.4 Gingerbread, ki je nameščena v največjem deležu Android naprav. To je bil tudi razlog, da smo uporabili knjižnice za to verzijo.



Grafikon 1: Delež uporabe Android verzij skozi leta

* beseda je pojasnjena v poglavju TERMINOLOŠKI SLOVAR

2.1.4 Senzorji v napravah Android

Z namenom, da bi preučili možnosti za uporabo različnih senzorjev v sami aplikaciji smo izdelali pregled nad trenutnim stanjem.

Večina naprav Android ima vgrajene senzorje, ki zaznavajo gibanje, usmerjenost in različne okoljske razmere. Ti senzorji so sposobni zagotavljati neobdelane podatke visoke natančnosti. Koristni so pri uporabi v vseh vrst aplikacij. Z izdelavo aplikacije za spremljanje vremena lahko, na primer, uporabljamo podatke s senzorja zunanje temperature ali vlažnosti ali pri izdelavi aplikacije za potovanje uporabimo podatke s senzorja za magnetno polje in senzorja pospeška.

Senzorji, ki so trenutno v napravah Android in jih je možno uporabljati v aplikacijah, so predstavljeni preglednici 1. V napravah s prejšnjimi različicami operacijskega sistema je senzorjev manj.

Senzor	Uporaba
Senzor pospeška	Zaznava gibanja (stresanje, nagib)
Senzor zunanje temperature	Spremljanje temperature zraka
Senzor za gravitacijo	Zaznava gibanja (stresanje, nagib)
Žiroskop	Zaznava rotacije (zasuk, obračanje)
Senzor za zunanjo svetlobo	Nadzor svetlosti zaslona
Senzor za linearni pospešek	Spremljanje pospeška v eni osi
Senzor za magnetično polje	Izdelava kompasa
Senzor za orientacijo	Določitev položaja naprave
Senzor za pritisk	Spremljanje sprememb zračnega pritiska
Senzor za bližino	Položaj telefona med klicem
Senzor zunanje vlage	Spremljanje rosišča, absolutne in relativne vlažnosti
Senzor za vektorsko rotacijo	Zaznava gibanja in vrtenja
Senzor za temperaturo v telefonu	Spremljanje temperature

Preglednica 1: Senzorji prisotni v napravah z operacijskim sistemom Android 4.0.

Večina naprav nima vgrajenih vseh senzorjev, ki jih Android podpira. Bolj ali manj imajo vsi vgrajen senzor pospeška in magnetometer, le manjši delež pa tudi barometre in termometre. Poleg tega imajo nekatere več senzorjev istega tipa. Na primer, naprava ima lahko dva senzorja za gravitacijo, vsakega za drugo območje delovanja.

2.1.5 Android na drugih napravah

Uporabo naše aplikacije bi z nadaljnjim razvojem lahko razširili tudi na druge pametne naprave. V nadaljevanju so navedene naprave, kjer je podprt operacijski sistem Android.

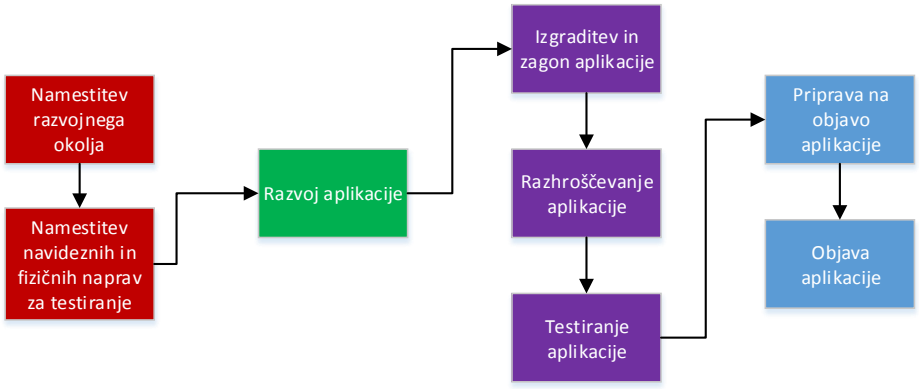
Odprt in prilagodljiv sistem Androida omogoča, da se uporablja v prenosnih računalnikih, netbookih, smartbookih, pametnih televizorjih (Google TV) in fotoaparatih. V bodoče napovedujejo razvijalci pri Googlu vključitev Androida v zapestne ure, slušalke, avto predvajalnike, ogledala, prenosne predvajalnike medijev ter zemeljske in "Voice over IP" telefone.

Slika 3: Pametna očala

Na sliki 3 so prikazana pametna očala (Google Glass), ki jih Google napoveduje za prodajo v letu 2014. Uporabnik na okvirju očal vgrajeno kamero z zaslonom, s katero upravlja funkcije Androida, kot je to mogoče na pametnih telefonih.

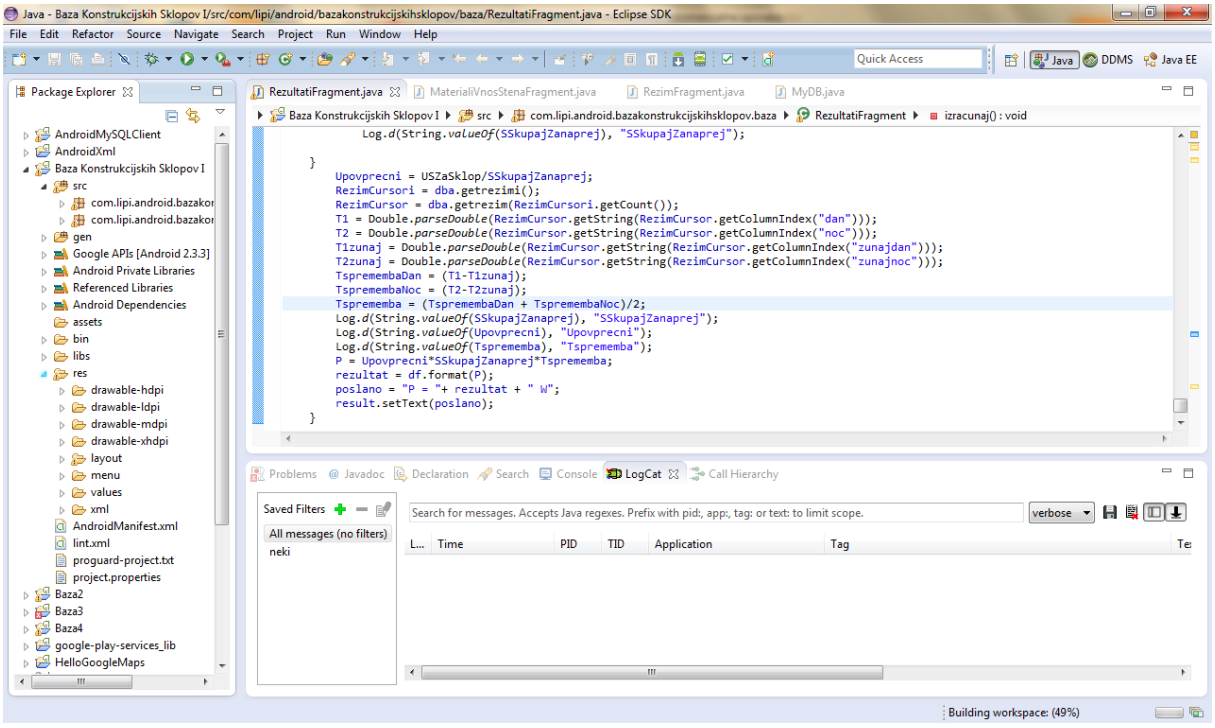
3 POTEK RAZVOJA APLIKACIJ ZA OS ANDROID

Razvoj aplikacije za OS Android poteka skozi več procesov, tehnologij in standardnih postopkov. Da bi bila izgradnja naše aplikacije mogoča jih je potrebno razumeti. Štirje glavni procesi so: namestitev, razvoj, razhroščevanje in testiranje ter objava (glej sliko 4). V naslednjem poglavju se predvsem osredotočamo na razlago najpomembnejšega, razvoja, v katerem bomo razložili način izgradnje aplikacije za pametni telefon.



Slika 4: Delovni potek od začetka izdelave do objave aplikacije

Najprej je potrebna namestitev ter nastavitev integriranega razvojnega okolja (IDE) in Android Software Development Kita (SDK). Omenjeni komplet za razvoj programske opreme, tj. Android SDK, vključuje celovit nabor orodij za razvoj. Ta so: razhroščevalnik, knjižnice, emulator, dokumentacijo, vzorčno kodo in primere.



Slika 5: Razvojno okolje Eclipse

Na sliki 5 je prikazano integrirano razvojno okolje Eclipse, ki smo ga v diplomskem delu uporabili skupaj z Android Development Tools vključkom (ADT). Razvoj je mogoč tudi z drugimi orodji kot so Android Studio, IntelliJ IDEA ali NetBeans. Za testiranje aplikacije lahko uporabljamo priložen emulator v sklopu Android navideznih naprav ali preko povezave USB na napravah z operacijskim sistemom Android.

Namestitvi sledi faza razvoja, kjer se nastavi in razvija aplikacija, ki vsebuje celotno izvorno kodo ter potrebne datoteke za delovanje. Za razvoj aplikacije smo uporabili programski jezik java z uporabo SDK knjižnic. Glede na to, da je Java objektni programski jezik, se tudi razvoj aplikacij za Android razvija na ta način. Objektno programiranje je paradigma za reševanje problemov. Koncepte se predstavi kot objekte, ki imajo podatkovna polja (atribute, ki predstavljajo objekt) in z njimi povezane postopke, znane kot metode. Objekti, ki so del razreda, komunicirajo med seboj in tvorijo aplikacijo. Objekt si lahko predstavljamo kot sestavo samostalnikov (npr. kot podatki v obliki stringov, spremenljivk ali številk) in glagolov (npr. kot akcije v obliki funkcij) [14].

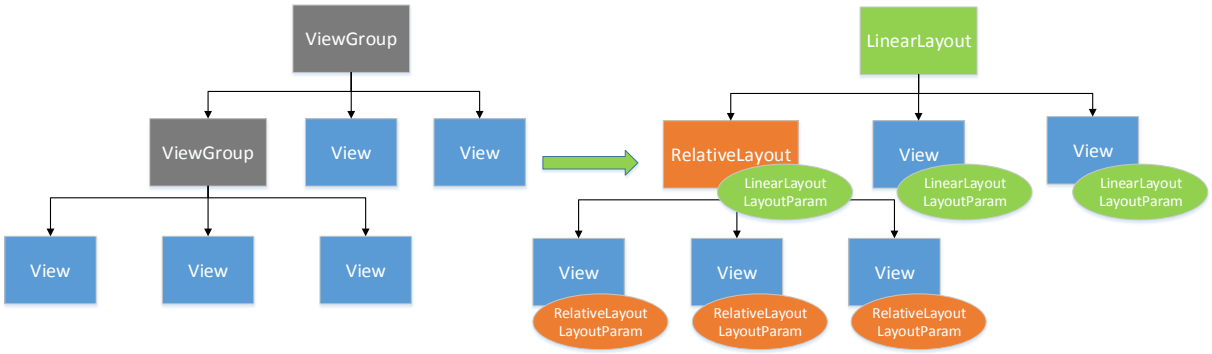
Iz izvorne kode in pripadajočih datotek je potrebna izgradnja .apk paketa, ki jo naprave lahko berejo. Pri tem svojo funkcijo opravijo orodja, ki se nahajajo v Android Software Development Kitu (SDK). Nazadnje testiramo aplikacijo z različnimi orodji, ki so prav tako vključene v SDK.

V kolikor želimo, da se aplikacija javno objavi, je potrebno prečiščenje oz. odstranitev odvečnih kod, podpisati aplikacijo ter zgraditi paket za objavo. Objava je mogoča preko Googlove strani za aplikacije (Google Play), spletne pošte ali drugih spletnih strani.

3.1 Videz aplikacij

3.1.1 Grafični uporabniški vmesnik:

Grafični uporabniški vmesnik (User Interface - UI) za aplikacije je zgrajen z uporabo hierarhije predmetov View in ViewGroup. Ta je prikazana na sliki 6. Predmeti View so navadno UI pripomočki, kot so gumbi ali besedilna polja. ViewGroup predmeti pa nevidni gradniki videza, ki določajo, kako se predmeti View obnašajo (npr. v mreži (GridView) ali v navpičnem seznamu (ListView)) [15].



Slika 6: Hierarhično izdelan grafični uporabniški vmesnik

Razpored predmetov View in ViewGroup določimo z uporabo tehnologije XML, kjer je mogoča enostavna izgradnja hierarhične strukture videza. Določitev UI videza je z uporabo tehnologije XML in ne izvorne kode predvsem uporabna, ker imamo lahko več različnih

videzov za različne velikosti zaslonov naprav. V primeru dveh verzij videza, lahko naročimo sistemu, da v primeru večjega zaslona uporabi eno in v primeru manjšega drugo.

3.1.2 Primeri pogostih Layoutov

Čeprav lahko s pomočjo razvejanja različnih vrst predmetov ViewGroup ustvarimo poljuben videz, je priporočljivo, da razvejanja niso preveč številčna in s preveč nivoji. Stremeti moramo k čim enostavnejši hierarhiji predmetov View in ViewGroup. Da bi bili izrisi hitrejši, je priporočljiva uporaba standardnih videzov, ki so vgrajeni v Android platformo. Najpogostejši so prikazani na sliki 7.

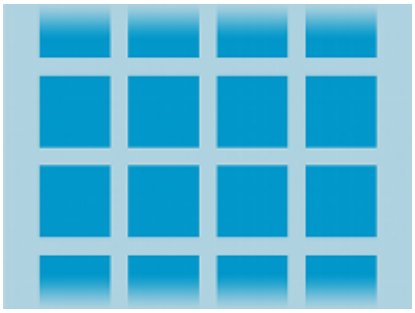
a.) relativni izgled (RelativeLayout)

b.) linearni izgled (LinearLayout)

c.) Spletni izgled (WebView)

Primera z Adapterji:

d.) Pogled za seznam (ListView)



e.) Mrežni pogled

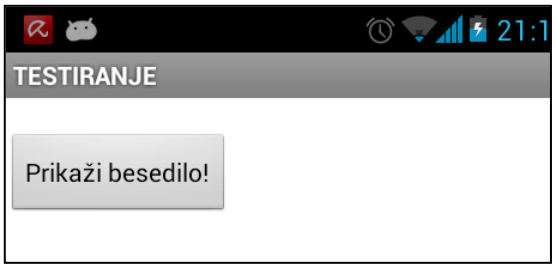
Slika 7: Primeri standardnih izgledov

3.1.3 Primer enostavne strukture XML za izgradnjo grafičnega uporabniškega vmesnika

Na sliki 8 je prikazan primer strukture XML s skupino pogleda (ViewGroup) kot LinearLayout. Gre za vrsto skupine pogleda, kjer so gradniki View nanizani horizontalno ali vertikalno en za drugim čez celoten zaslon. V nadaljevanju se bosta uporabila samo dva. Enega kot vrstico za tekst (gradnik TextView) in drugega za gumb (gradnik Button). Videz na napravi pa je prikazan na sliki 9.

```
<?xml version="1.0" encoding="utf-8"?> =>definiranje XMLja
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android =>…
                …=>postavitev skupine pogleda (GroupView) kot LinearLayout
                  android:layout_width="fill_parent"  =>razporeditev dolžine
          android:layout_height="fill_parent" => razporeditev višine
          android:orientation="vertical" >  =>orientacija LinearLayouta
   <TextView android:id="@+id/tekst"  =>ime gradnika
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"/>
   <Button android:id="@+id/gumb"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Prikaži besedilo!" />   =>besedilo, ki naj ga gumb vsebuje
</LinearLayout>
```

Slika 8: Struktura XML datoteke s podatki za določitev grafičnega uporabniškega vmesnika

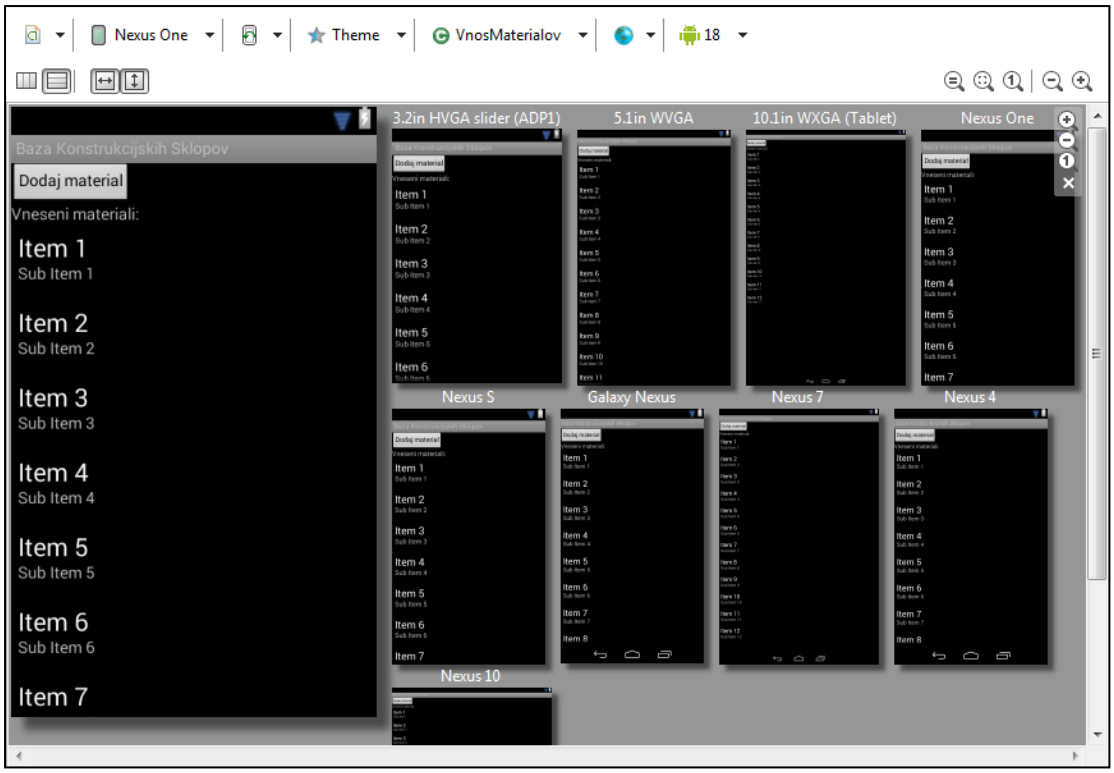


Slika 9: Videz izgrajenega primera XML

3.1.4 ADT grafični urejevalnik videza za Eclipse

Naše aplikacije lahko urejamo tudi z grafičnim orodjem WYSIWYG ("What You See Is What You Get"), ki je vključeno v Androidova razvojna orodja (Android Development Tools - ADT).

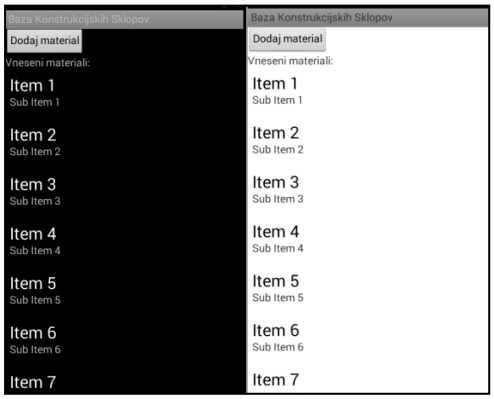
Ker se Android poganja na napravah z različnimi velikostmi zaslonov (od majhnih na telefonu do velikih TV zaslonov), je pri oblikovanju grafičnega vmesnika potrebno zagotoviti ustrezen zunanji videz čim večjemu številu naprav. Na sliki 10 je prikaz temu primerno v grafičnim urejevalniku programa ponujena izbira zunanjih videzov za različne dimenzije telefonov oziroma tablic.



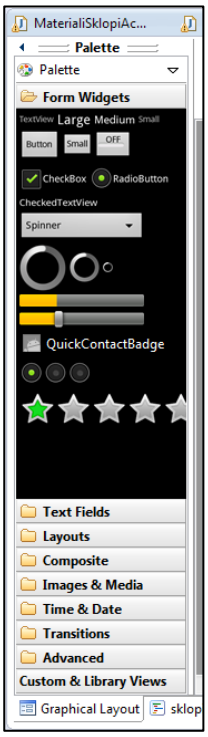
Slika 10: Videz na različnih merah zaslona

Na sliki 11 so prikazani nekateri pripomočki v grafičnem orodju. Poleg videza na različnih merah zaslona, lahko v zgornji opravilni vrstici izbiramo še med ležečim ali pokončnim položajem telefona, temo videza, videzom v drugih tujih jezikih ter izbiro verzije operacijskega sistema Android. Starejše verzije Androida ne podpirajo vseh gradnikov, ki jih iz orodne vrstici po principu "povleci in spusti" dodajamo pri izgradnji.

a.) Test videza v ležečem položaju telefona



b.) Standardni temi Holo in Holo.Light



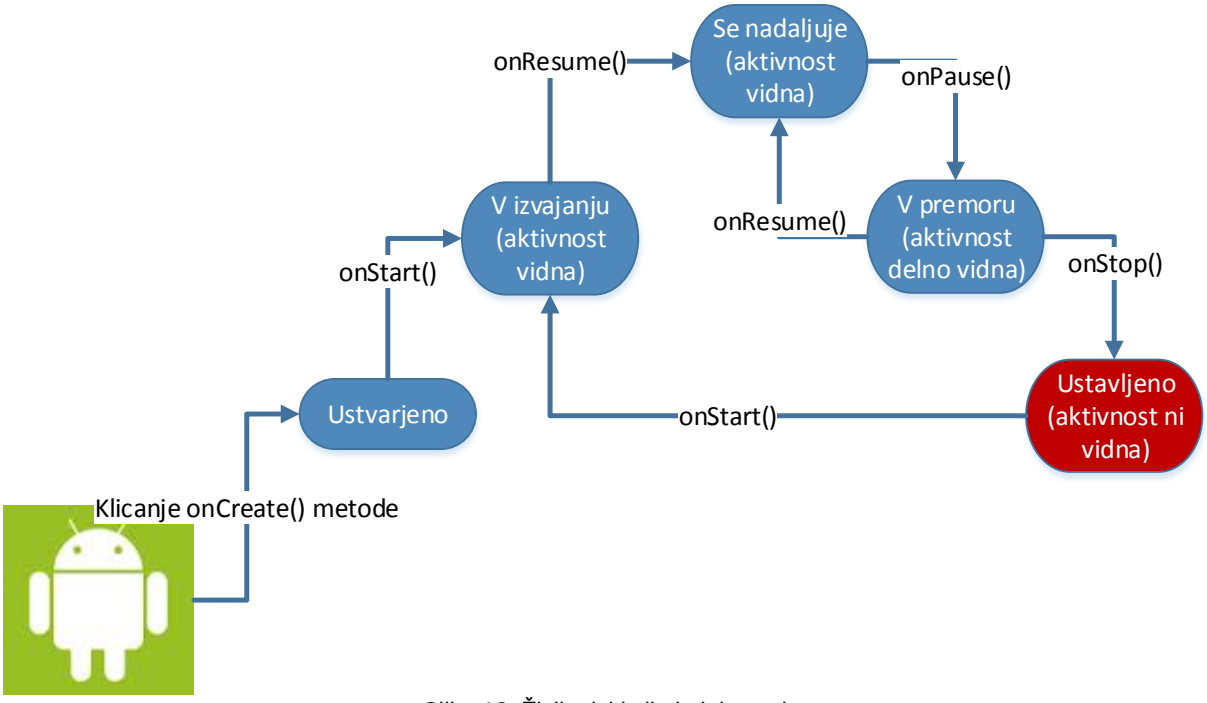
c.) Orodna vrstica z gradniki

Slika 11: Pripomočki v grafičnem orodju

3.2 Zgradba aplikacij

3.2.1 Aktivnosti

Aplikacije so sestavljene iz aktivnosti. Vsaka mora vsebovati minimalno eno aktivnost. Aktivnost zažene uporabniški vmesnik in s kodo nadzoruje gradnike in elemente na njem. Ena od aktivnosti v aplikaciji je navedena kot "glavna" in je aktivirana, ko se aplikacija prvič odpira. Vsaka aktivnost lahko nato zažene drugo aktivnost. Vsakič, ko se začne nova aktivnost, se prejšnja aktivnost ustavi, sistem pa ohranja aktivnost mirujoče v ozadju. S klicanjem aktivnosti, se ustvarja zaporedje plasti aktivnosti; prva je spodaj, zadnja klicana pa na vrhu zaslona. S klikom gumba nazaj, se aktivnost, s katero smo pravkar komunicirali, uniči in kliče aktivnost pred njo. Med ustvarjanjem in zapiranjem aktivnosti potekajo različne faze, ki so prikazane na sliki 12. Android stremi k skrbnemu ravnanju s spominom, zato je nad fazami potreben nadzor. V nekaterih primerih te faze sam nadzoruje (izklopi) [16].



Slika 12: Življenjski cikel aktivnosti

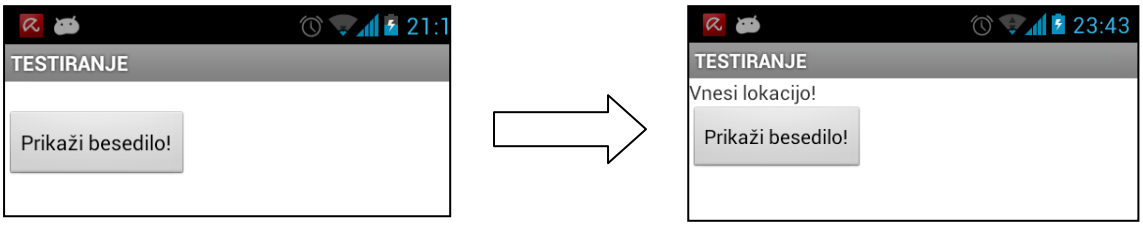
OnCreate() metoda je vključena v vsako aktivnost, saj je potrebna za njeno izgradnjo. V okviru njene uveljavljenosti, so potrebne tudi nekatere inicializacije komponent za našo aktivnost. Ena od njih je setContentView(), ki definira grafično podobo uporabniškega vmesnika. Ta je določena ločeno od aktivnosti in se jo ustvarja s pomočjo XML datotek.

Namera (intent) se uporabi za zagon aktivnosti in za komunikacijo med različnimi deli sistema Android. Aplikacija lahko prenaša ali sprejema namere. S prenašanjem namere se pošilja sporočilo Androidu, da se bo nekaj zgodilo. Ta namera lahko sporoča Androidu, da se želi začeti nova aktivnost znotraj aplikacije ali zagon druge aplikacije. Registriran mora biti tudi sprejemnik za namero, ki sporoči Androidu, kaj naj stori.

3.2.2 Primer enostavnega razreda (MainActivity.java)

V nadaljevanju bomo predstavili enostaven razred z namenom predstavitve osnovnih komponent, ki pogosto tvorijo razred. Pojasnilo za to kar vrstice kode ustvarjajo, je v tekstu nad njo (kodo) označeno z //. V spodnji razred vključimo videz, ki smo ga uporabili v poglavju 3.1.3 ter aktiviramo gradnika gumb (Button) in vrstico za besedilo (TextView), ki smo ga vključili v sam videz. V aplikaciji se ob kliku na gumb v vrstici za besedilo pojavi tekst "Vnesi lokacijo!". Videz na aplikaciji pred in po pritisku gumba je prikazan na sliki 13.

```
// Ime paketa v katerem gradimo aplikacijo
package com.example.testiranje;
// Vključitev različnih knjižnic
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
// Kakšne vrste aktivnosti ali dejavnosti razširi določen razred (datoteka java). V tem primeru je ime razreda MainActivity, ki je tudi začetni razred s katerim aplikacija starta – deklarirano v AndroidManifestu)
public class MainActivity extends Activity {
        // Deklaracija spremenljivke za gradnik Button (gradnik gumb)
        private Button knof;
        // Deklaracija spremenljivke za gradnik TextView (gradnik za besedilo)
        TextView mTextView;
// Start metode onCreate()
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
// Nastavi izgled uporabniškega vmesnika za to aktivnost
// Izgled je definiran v datoteki res/layout/main_activity.xml
 setContentView(R.layout.activity_main);
// Povezava grafičnega dela (gradnik Button) iz datoteke main_activity.xml s spremenljivko "knof" v temu razredu
 knof=(Button)findViewById(R.id.gumb);
// Registriranje poslušalca za dotik na gradnik "knof"
 knof.setOnClickListener(
      new View.OnClickListener()
       {
         // Ob kliku…
          public void onClick(View view)
         {
                // vzpostavi zvezo med grafičnim delom gradnika (TextView v main_activity.xml) in spremenljivko mTextView v temu razredu
                mTextView = (TextView) findViewById(R.id.text);
                //Ustvari spremenljivko "tekst". Spremenljivka vsebuje tekst "Vnesi lokacijo!"
                String text = "Vnesi lokacijo!";
                //v gradnik TextView ("mTextView") vnesi spremenljivko "text". Na zaslonu se nam ob kliku na gumb pojavi besedilo "Vnesi lokacijo!"
                mTextView.setText(text);
         }
         }
      });}}
```



Slika 13: Izgled našega primera pred in po kliku na gumb

3.2.3 AndroidManifest.xml

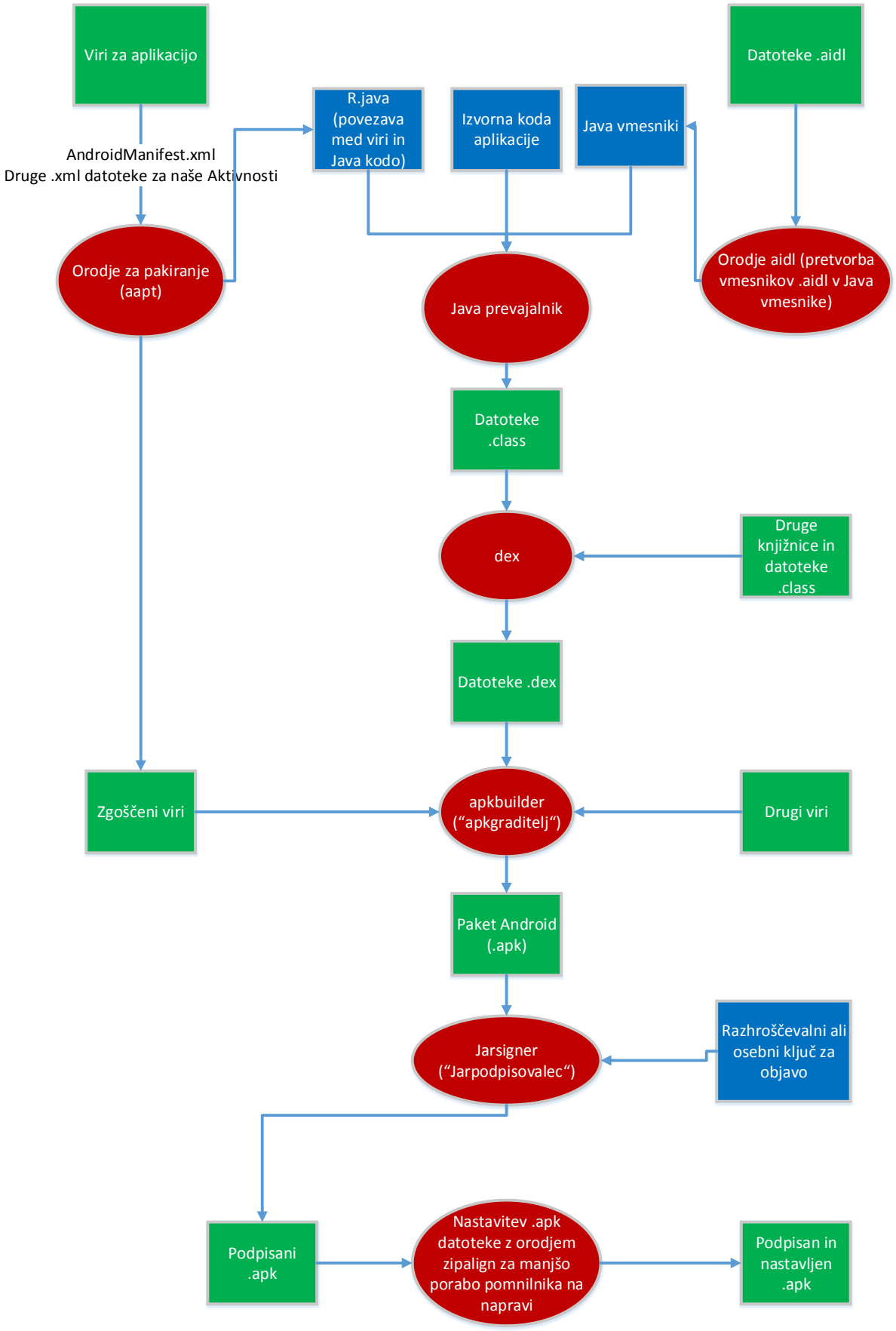
Vsaka aplikacija mora imeti AndroidManifest datoteko v korenskem imeniku. Manifest predstavi bistvene podatke aplikacije sistemu Android. Ti so nujno potrebni pred zagonom katerega koli dela kode za aplikacijo. Pomembnejše informacije, ki jih vsebuje so:

- ime paketa s katerim se aplikacija predstavi napravi z OS Android,
- razred s katerim se bo aplikacija začela,
- druge razrede (aktivnosti), ki se bodo v aplikaciji lahko izvajali (tudi servisi, ponudniki podatkov ipd.),
- dovoljenja, ki so potrebna, da bo aplikacija lahko opravljala svojo nalogo,
- minimalno verzijo Androidovega operacijskega sistema, ki je potrebna za delovanje aplikacije ter
- vključevanje možnosti Google Mapsa z Google Maps API ključem.

3.3 Izgradnja apk paketov

Projekti Android so pri izgradnji aplikacije in prenosa na napravo pakirani v .apk datoteko. Paket vsebuje vse potrebne informacije za zagon aplikacije na napravi ali emulatorju. Med te spadajo ustvarjene .dex datoteke (.class datoteke pretvorjene v Dalvik bitno kodo), binarna verzija datoteke AndroidManifest.xml, ustvarjeni zgoščeni viri (resources.arsc) in datoteke nezgoščenih virov za našo aplikacijo.

Na sliki 14 je podrobneje prikazan proces pakiranja .apk datotek. Ta je skrit in se samodejno izvede s pomočjo orodij, ki so vključene v Android Development Tools vključek (ADT). Izvorna koda aplikacije, Java vmesniki ter razred R.java, ki vsebuje informacije z datotek XML, so preko Java prevajalnika pretvorjene v datoteke .class, ki jih je mogoče pretvoriti v Dalvik "dex-kodo", ki jo Dalvikov navidezni stroj, lahko izvaja. V apk graditelju (apkbuilder) se združijo .dex datoteke, arsc viri, nezgoščeni in zgoščeni viri (AndroidManifest.xml), ki tvorijo paket .apk. Paketu je dodeljen samodejno zgeneriran razhroščevalni ključ. V primeru, da je aplikacija pripravljena za objavo, je potreben podpis .apk paketa z našim osebnim ključem. Orodje zipaplign poskrbi za manjšo porabo pomnilnika na napravi.



Slika 14: Izgradnja .apk paketa, ki zgosti vso ustvarjeno kodo in ostale vire v eno zagonsko datoteko. Proces je nam neviden in ga opravijo orodja iz ADT.

4 RAZVOJ BAZE KONSTRUKCIJSKIH SKLOPOV

V tem poglavju opisujemo programsko kodo, ki smo jo uporabili za izgradnjo naše aplikacije (tudi tu so s predznakom **//** podane dodatne obrazložitve kode). Poglavja so razdeljena glede na aktivnosti in posamezne zavihke za vnos podatkov v aplikaciji.

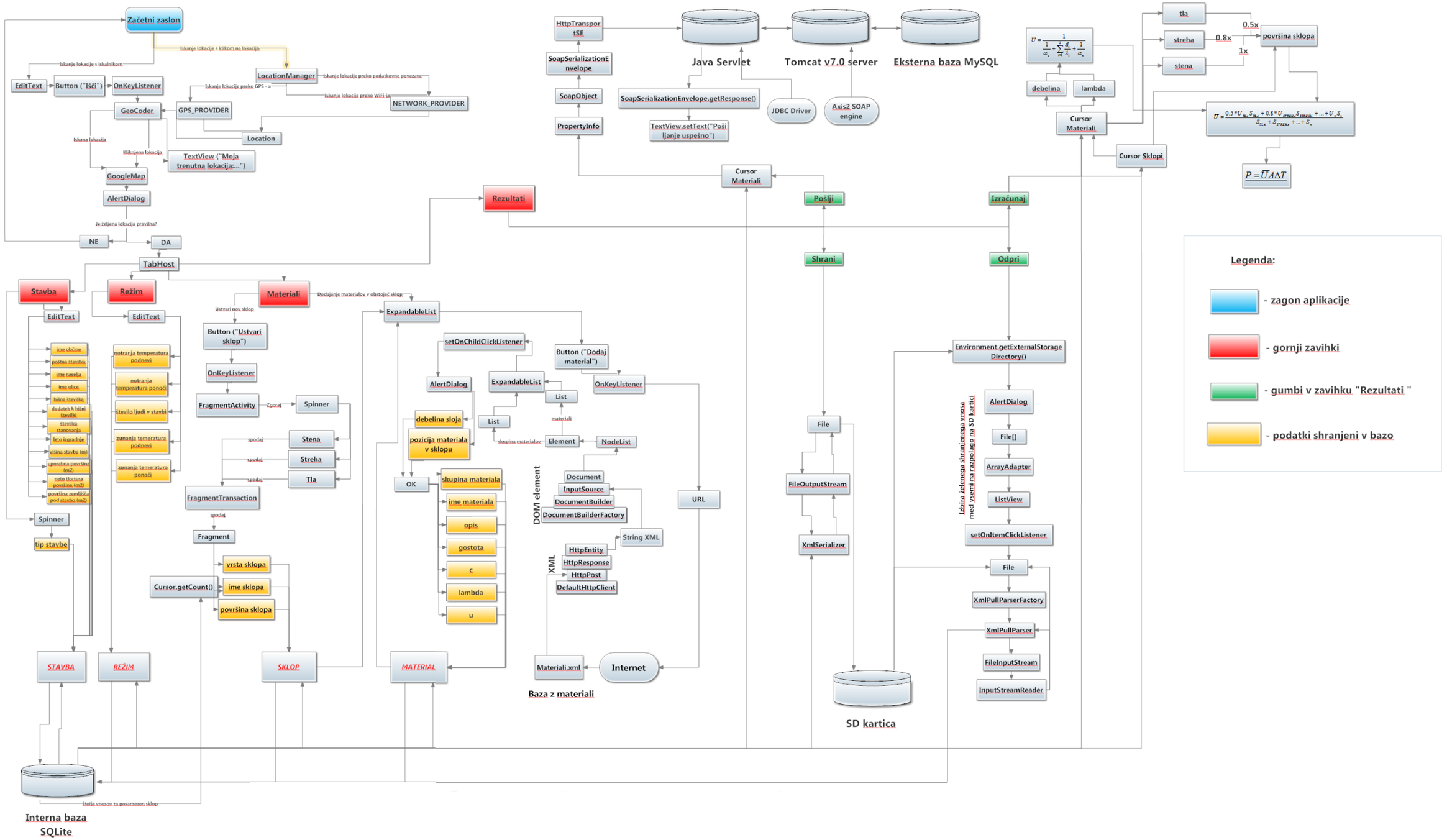
4.1 Potek aplikacije

V nadaljevanju je opisan potek aplikacije z namenom lažjega razumevanja kode, ki sledi v naslednjih poglavjih.

Prva aktivnost se začne z zemljevidom, kjer uporabnik najprej določi lokacijo svojega objekta. Z aktiviranjem GPS sprejemnika zazna aplikacija njegovo lokacijo. Prikaže jo na zemljevidu, v vrstici nad njim pa izpiše naslov. V kolikor vzpostavitve povezave s pomočjo GPS ni, je možno približno lokacijo zaznati preko internetne povezave, preko Wi-Fi – ja ali podatkovne povezave mobilnega operaterja. Če želi uporabnik obravnavati objekt, ki ni v njegovi okolici, je v zgornjem delu zaslona iskalnik katere koli lokacije v Sloveniji. Zemljevid na zaslonu se locira na željen naslov. Z dotikom na objekt, se odpre novo okence, kjer se potrdi ali zavrne obravnavanje dotaknjenega naslova.

Ko je lokacija potrjena, se odpre nova aktivnost TabHost (glej sliko 15). Gre za novo okno, ki vsebuje štiri zavihke: stavba, režim, materiali ter rezultati. Pod zavihkom "Stavba", izpolni uporabnik 13 okenc z želenimi podatki o manjkajočih podatkih o naslovu stavbe, tipu stavbe, merah in površinah stavbe. Ko je zavihek izpolnjen, se podatki z zavihka že shranijo v notranji bazi. V zavihku "Režim" uporabnik izpolni še štiri dodatna polja, ki se nanašajo na notranjo in zunanjo temperaturo v in izven objekta ter prazno polje o številu ljudi, ki prebivajo v objektu. Ko izpolni tudi ta zavihek, se ponovno podatki shranijo v notranjo bazo. V zavihku "Materiali" izpolni uporabnik vse podatke, ki se nanašajo na konstrukcijske sklope v objektu. Najprej doda vse sklope, ki so potrebni za obravnavo. Za vsak sklop izpolni ime, vrsto (tla, streha ali stena) ter površino tega sklopa. Na sklop se vežejo materiali, ki jih uporabnik doda naknadno, za vsakega posebej. Baza s temi podatki se nahaja na strežniku in se do nje dostopa preko interneta. Vsak material vsebuje podatke o gostoti, specifični toploti, toplotni prevodnosti ter difuzijski upornosti vodni pari. S klikanjem na seznam doda materiale za vsak sklop. Za vsakega se odpre še dodatno okno, kjer se izpolni debelina sloja ter lega materiala v sklopu. Tudi tu se izpolnjene informacije samodejno shranjujejo v interni bazi.

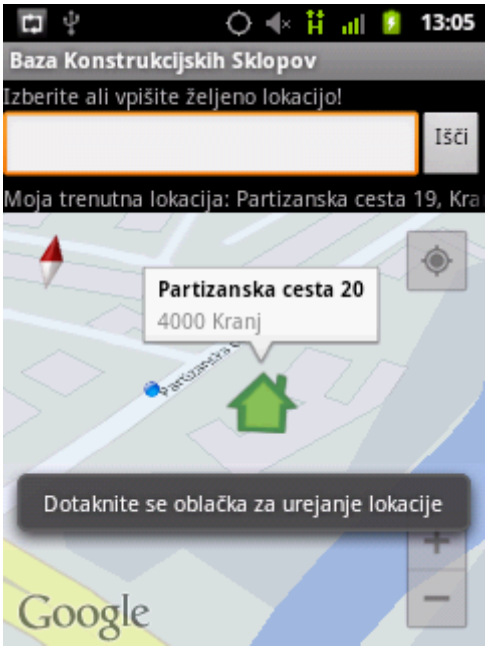
Pod zavihkom "Rezultati" ima uporabnik štiri opcije. S klikom na gumb "Pošlji" pošljemo podatke o stavbi na oddaljeni strežnik, kjer je nameščena baza MySQL, ki shranjuje velike količine vnešenih konstrukcijskih sklopov s pripetimi informacijami poslanih od uporabnikov. Gumba "Shrani" in "Odpri" sta namenjena odpiranju in shranjevanju vseh vnesenih podatkov na SD kartico. Ti se shranijo v datoteko XML in jih je mogoče naknadno urejati z osebnim računalnikom. S klikom na gumb "Izračunaj" pokliče uporabnik iz notranje baze vse potrebne podatke, ki so potrebni za algoritme. Iz tega sledi izračun toplotnih izgub ter toplotnega toka skozi stavbo. Potek aplikacije ter njeni gradniki so predstavljeni na sliki 15.



Slika 15: Potek aplikacije ter njeni gradniki

4.2 Začetni zaslon (določitev lokacije)

Aplikacija sama ugotovi uporabnikovo trenutno lokacijo (glej sliko 17). Ta se izpiše in pojavi na interaktivnem zemljevidu. V kolikor želi obravnavati želeni objekt, se ga na zemljevidu dotakne. Če želeni objekt ni v njegovi bližji okolici, ga lahko poišče z vnosom naslova v iskalnik nad zemljevidom. Ko je lokacija potrjena, sledi nova aktivnost s štirimi zavihki. Uporabljen je zemljevid Google Maps, ki ga je mogoče enostavno vgraditi v aplikacije.



Slika 17: Samodejna zaznava naše lokacije, s klikom na objekt dobimo želeni naslov

Slika 16: Uporaba iskalnika za želeno lokacijo

4.2.1 Vključitev Google Maps Android API v2 v aplikacijo

Knjižnico zagotavlja com.google.android.gms.maps.MapFragment in MapView, ki ju je potrebno vključiti v naš razred. Poleg tega je potrebna namestitev iz Android SDK Managerja "Google Play services" ter podpora za Google programski vmesnik (Google API).

4.2.1.1 Dovoljenja v AndroidManifestu

V datoteki AndroidManifest.xml moramo urediti dovoljenja ter vnesti Google Maps API ključ. Urejena dovoljenja so nujna, da želeni senzorji za določeno aplikacijo lahko delujejo.

```
//Poleg dovoljenja za internet

<uses-permission android:name="android.permission.INTERNET" />

//in dovoljenja za uporabo GPS - a ter določitve približne lokacije preko podatkovne povezave

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>,

//je potrebno dovoljenje za uporabo Google Mapsa:

<permission

        android:name="com.example.mapdemo.permission.MAPS_RECEIVE"

        android:protectionLevel="signature"/>

<uses-feature

    android:glEsVersion="0x00020000"

    android:required="true"/>

<uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>

<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>

//Poleg tega v manifest vnesemo Google Maps API ključ, ki ga dobimo z registracijo in prijavo
aplikacije na Googlovi strani (https://code.google.com/apis/console/)

<meta-data

    android:name="com.google.android.maps.v2.API_KEY"

    android:value="AIzaSyCdSWnkRj-AosrRTdsGpQowefFFDSnv-CAo"/>
```

4.2.1.2 Vključitev v razred

Kodo, prikazano spodaj, je potrebno vključiti v razred, kjer želimo pripomoček uporabiti. Sprva je potrebna inicializacija spremenljivke. Imena spremenljivk določamo sami. Zasebne (private) spremenljivke delujejo le v razredu v katerem jo uporabljamo. Z deklaracijo spremenljivk, spremenljivkam določimo dodatne informacije.

```
//Inicializacija spremenljivke

private GoogleMap mMap;

//Deklaracija spremenljivke

mMap =
((SupportMapFragment)getSupportFragmentManager().findFragmentById(R.id.map1)).getMap();

mMap.setMyLocationEnabled(true);
```

4.2.1.3 Vključitev v videz aplikacije

Google Maps se v videz aplikacije vključi kot gradnik fragment. Za razred v katerem ga obravnavamo, je potrebno, da razširja aktivnost fragmenta – extends FragmentActivity. V nadaljevanju je podan fragment, ki ga moramo vključiti v datoteko za videz uporabniškega vmesnika.

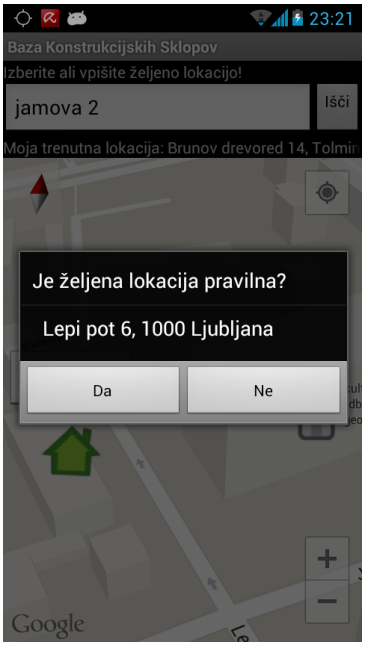
```
<fragment
        android:id="@+id/map1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        class="com.google.android.gms.maps.SupportMapFragment"/>
```

4.2.2 Določitev naše lokacije

Določitev trenutne lokacije poteka preko GPS satelitov oziroma koordinatnih točk, ki jih telefon sprejme od njih. GPS naprava ne zazna satelitov, če smo v zaprtem prostoru, zato je takrat boljše določanje lokacije z internetno povezavo preko Wi-Fi – ja. V aplikacijo je priporočljivo dodati poslušanje obeh senzorjev. Spodaj je predstavljena vzpostavitev povezave preko GPS-a ter vnos koordinat v kamero, ki bo prikazovala našo lokacijo.

```
//Inicializacija LocationManagerja

LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// Poslušanje, ko se lokacija spremeni (posodobitev lokacije)

LocationListener locationListener = new LocationListener() {

   public void onLocationChanged(Location gpsLocation) {

   Location gpsLocation =
requestUpdatesFromProvider(LocationManager.GPS_PROVIDER,R.string.not_support_gps);

//Nastavitev načina pogleda in centriranje kamere na mapi na lokacijo, kjer se mi nahajamo

final CameraPosition moje = new CameraPosition.Builder().target(new
LatLng(gpsLocation.getLatitude(),gpsLocation.getLongitude()))

                .zoom(18.5f)

                .tilt(50)

                .build();

mMap.animateCamera(CameraUpdateFactory.newCameraPosition(moje));

 } };

// Poslušanje za spremembo lokacije z LocationManagerjem

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
```

Ko je naša pozicija na zemljevidu prikazana in objekt izbran, se ga dotaknemo. Aplikacija kliče gradnik AlertDialog, v katerem potrdimo ali zavrnemo izbrano lokacijo. Primer je prikazan na sliki 18.



Slika 18: Potrditev lokacije

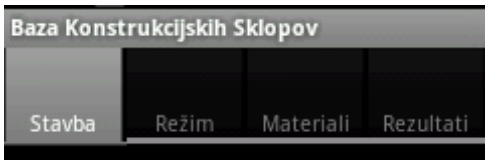
4.2.3 Klicanje namere

Namera je klicana z namenom, da se odpre nova aktivnost. Če želimo, da se v novo aktivnost prenesejo podatki iz prejšnje, jih moramo v namero vključiti. Iz aktivnosti zemljevida prenesemo v novo aktivnost naslov izbrane lokacije. Z novo aktivnostjo sledi tudi odpiranje novega grafičnega uporabniškega vmesnika.

```
//Razrez naslova, ki jih Google posreduje v enem stavku (na poštno številko, ime naselja, ime ulice in hišno številko)

String NaseljePostna = Naselje.substring(0,Naselje.indexOf(" "));

String NaseljeBesedilo = Naselje.substring(Naselje.indexOf(" ")+1,Naselje.length());

String stevilka = Ulica.substring(Ulica.trim().lastIndexOf(" ")+1,Ulica.trim().length());

String besedilo = Ulica.substring(0,Ulica.lastIndexOf(" ")+1);

//Definiranje namere

Intent myIntent = new Intent(this, MainActivity.class);

//V nameri (intentu) so shranjeni podatki iz naše lokacije

myIntent.putExtra(Constants.STAVBA_POSTNA, NaseljePostna);

myIntent.putExtra(Constants.STAVBA_NASELJE, NaseljeBesedilo);

myIntent.putExtra(Constants.STAVBA_ULICA, besedilo);

myIntent.putExtra(Constants.STAVBA_HISNA, stevilka);

//Zagon namere

this.startActivity(myIntent);
```

4.3 Uporabniški vmesnik z zavihki

Namera kliče aktivnost TabHost, ki omogoča zavihke na vrhnjem delu zaslona. V naši aplikaciji jih imamo štiri. Trije so namenjena vnosu podatkov, eden pa rezultatom oziroma shranjevanju, odpiranju ter pošiljanju podatkov na strežnik. Videz zavihkov je prikazan na sliki 19.



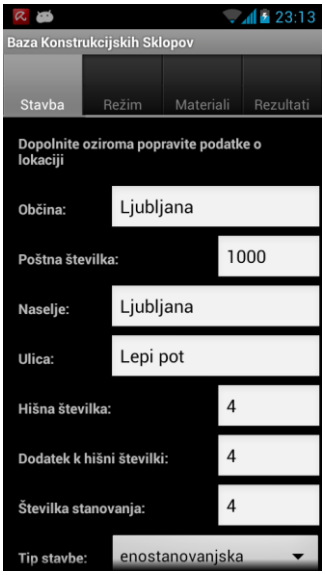
Slika 19: Zavihki v našem programu

```
//Razred (v našem primeru MainActivity.class) razširjamo s TabActivitijem
public class MainActivity extends TabActivity {
…
//Definiranje TabHosta
TabHost tabHost = getTabHost();
//Definirati je potrebno vsak zavihek
TabSpec stavbaspec = tabHost.newTabSpec("Stavba");
    stavbaspec.setIndicator("Stavba");
// Definicija namere, ki zažene razred StavbaFragment
    Intent stavbaIntent = new Intent(this, StavbaFragment.class);
    stavbaspec.setContent(stavbaIntent);
TabSpec pregledspec = tabHost.newTabSpec("Režim");
    pregledspec.setIndicator("Režim");
//Definicija namere, ki zažene razred RezimFragment
    Intent pregledIntent = new Intent(this, RezimFragment.class);
    pregledspec.setContent(pregledIntent);
TabSpec materialispec = tabHost.newTabSpec("Materiali");
    materialispec.setIndicator("Materiali");
       // Definicija namere, ki zažene razred MaterialiSklopiActivity
    Intent materialiIntent = new Intent(this, MaterialiSklopiActivity.class);
    materialispec.setContent(materialiIntent);
TabSpec rezultatispec = tabHost.newTabSpec("Rezultati");
```

```
        rezultatispec.setIndicator("Rezultati");
//Definicija namere, ki zažene razred RezultatiFragment
        Intent rezultatiIntent = new Intent(this, RezultatiFragment.class);
        rezultatispec.setContent(rezultatiIntent);
//Povezava (registracija) vseh zavihkov v celoto
tabHost.addTab(stavbaspec);
tabHost.addTab(pregledspec);
tabHost.addTab(materialispec);
tabHost.addTab(rezultatispec);
}
```

4.4 Zavihek "Stavba"

V zavihek stavba uporabnik vnese osnovne informacije o stavbi. To so manjkajoči osnovni podatki o naslovu, mere ter površine, ki jih objekt zavzema. Njegov videz je prikazan na sliki 20.



Slika 20: Izgled zavihka "Stavba" v aplikaciji

Podatki iz prejšnje aktivnosti, kot so poštna številka, ime naselja, ime ulice in hišna številka so že vneseni iz prejšnje aktivnosti. Za vsak podatek je bil uporabljen gradnik EditText. Za izbiro tipa zgradbe, kjer so že vnaprej določene mogoče izbire, pa gradnik Spinner. Podatki, ki jih zavihek shrani so:

- ime občine
- poštna številka
- ime naselja
- ime ulice

- hišna številka
- dodatek k hišni številki
- številka stanovanja
- tip stavbe
- leto izgradnje
- višina stavbe (m)
- uporabna površina (m^2)
- neto tlorisna površina (m^2)
- površina zemljišča pod stavbo (m^2)

4.4.1 Gradnik EditText

Z gradnikom EditText najlažje dobivamo podatke od uporabnika. Gre za poljubno velik prazen prostor v katerega zapiše uporabnik želene vrednosti.

```
//Definiranje gradnika EditText:

private EditText ime_obcine;

//Povezava z gradnikom, ki smo ga registrirali pod imenom "imeObcineEditText" in ga vnesli v XML datoteko za izgled (stavba_layout.xml)

ime_obcine = (EditText) findViewById(R.id.imeObcineEditText);

//Zajem naših vnesenih podatkov (imena občine) v EditTextu in kreiranje stringa s temi podatki

private String Obcina = ime_obcine.getText().toString();
```

4.4.1.1 Vnos gradnika v XML datoteko za videz

Poleg nastavitev kako naj se gradnik obnaša v razredu, ga je potrebno dodati tudi v datoteko XML, ki ureja grafični del zavihka "Stavba".

```
<EditText

        android:id="@+id/ imeObcineEditText "

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:ems="10"

        android:inputType="textCapWords" />
```

4.4.2 Gradnik Spinner

Gradnik Spinner uporabljamo, kadar želimo, da ima uporabnik že vnaprej določene mogoče izbire. V našem primeru gre za izbiro tipa zgradbe, kjer uporabniku omogočajo izbrati 10 tipov.

```
//Definiranje gradnika Spinner

private Spinner tip_zgradbe;
```

//Povezava z gradnikom, ki smo ga registrirali pod imenom "tipZgradbeSpinner" in ga vnesli v XML datoteko za izgled (stavba_layout.xml)

```
tip_zgradbe = (Spinner) findViewById(R.id.tipZgradbeSpinner);
```

// Kreiranje stringa s podatki iz izbire (tipa zgradbe) v gradniku Spinner:

```
String TipStavbe = spinner1.getSelectedItem().toString();
```

4.4.2.1 Vnos gradnika v XML datoteko za videz

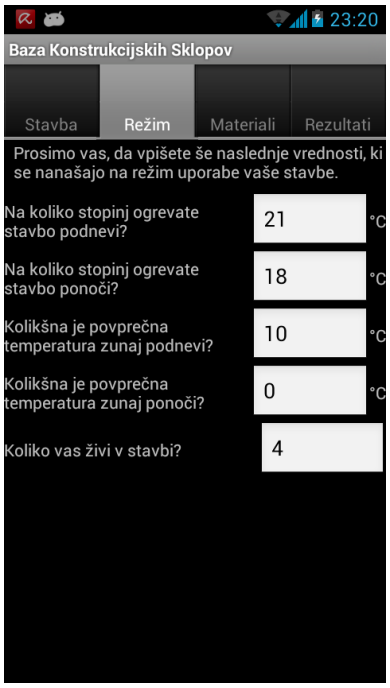
Gradnik Spinner na podoben način kot EditText vnesemo v grafični del uporabniškega vmesnika. Izbire, ki so uporabniku omogočene, so določene v datoteki strings.xml.

```
<Spinner
        android:id="@+id/ tipZgradbeSpinner"
        android:layout_width="2dp"
        android:layout_height="match_parent"
        android:layout_weight="0"
         //Število in imena opcij v gradniku Spinner določimo v datoteki     res/values/strings.xml
        android:entries="@array/tip_stavbe"
        android:prompt="@string/izberitetip" >
    </Spinner>
```

Vrednosti (tipi stavb) iz datoteke strings.xml:

```
<string-array name="tip_stavbe">
    <item>enostanovanjska</item>
    <item>večstanovanjska</item>
    <item>upravna in pisarniška stavba</item>
    <item>stavba za izobraževanje</item>
    <item>stavba za zdravstvo</item>
    <item>gostinska stavba</item>
    <item>športna stavba</item>
    <item>trgovska stavba</item>
    <item>stavba za storitvene dejavnosti</item>
    <item>stavba za druge namene</item>
 </string-array>
```

4.5 Zavihek "Režim"



Slika 21: Zavihek "Režim"

Na sliki 21 je prikazan zavihek "Režim", kjer izpolnimo podatke, ki se nanašajo na temperaturo v objektu in izven njega. Na razpolago imamo 5 gradnikov EditText v katere vnesemo naslednje podatke:

- temperaturo, na katero ogrevamo stavbo podnevi
- temperaturo, na katero ogrevamo stavbo ponoči
- povprečno temperaturo zunaj podnevi
- povprečno temperaturo zunaj ponoči
- število ljudi, ki živijo v stavbi

Potek vnosa v razred in grafični uporabniški vmesnik poteka na enak način kot v poglavju 4.4.

4.6 Zavihek "Materiali"

Pod zavihek "Materiali" vnašamo konstrukcijske sklope z njihovimi značilnostmi. Najprej je potrebno ustvariti sklope, čemur je namenjen gumb "Ustvari sklope" (gradnik Button) (glej sliko 22). Ta sproži novo namero in aktivnost, kjer sklope definiramo. Ti se samodejno vnašajo v gradnik ExpandableList prejšnje aktivnosti. Zanj smo se odločili, ker pregledno razporedi glavne skupine (konstrukcijske sklope) in podskupine (materiale) v seznam, ki ga je s klikom mogoče razširiti. Prikazan je na sliki 23.

Slika 22: Zavihek "Materiali" brez vnesenih sklopov

Slika 23: Zavihek "Materiali" z vnesenimi sklopi v gradniku ExpandableList

Na desni strani gradnika ExpandableList je pri vsakem dodanem konstrukcijskem sklopu vgrajen gumb (gradnik Button), ki nas poveže z datoteko XML, ki je locirana na oddaljenem strežniku. V tej se nahajajo vsi materiali, ki jih je mogoče vgraditi v sklop. Za vsakega so definirane tudi njegove karakteristike. Te se skupaj z imenom materiala, debelino sloja in pozicije v sklopu prenese v uporabnikovo napravo k želenemu sklopu.

4.6.1 Ustvarjanje gumbov s povezavo na drugo aktivnost

V nadaljevanju je pojasnjeno, kako nastavimo gumb, da opravi želeno nalogo. V tem primeru se s pomočjo namere odpre nova aktivnost, kjer vnašamo podatke za nov sklop.

```
//Definiranje gumba

private Button mButton;

//Povezava z gradnikom, ki smo ga registrirali pod imenom "button1", ki smo ga vnesli v datoteko za layout (sklopi_list.xml)

mButton = (Button) findViewById(R.id.button1);

//Poslušanje na "klik"

mButton.setOnClickListener(new View.OnClickListener() {

                    public void onClick(View view) {

                    //Akcija ob kliku (zagon razreda MaterialiVnosActivity)
```

```
                Intent i = new Intent(this, MaterialiVnosActivity.class);
                startActivityForResult(i, ACTIVITY_CREATE);
                }
        });
```

4.6.2 Ustvarjanje sklopa

Preverili smo možnosti, kako bi na najenostavnejši način vnašali podatke o sklopih. Ker je površina na pametnih mobilnih napravah omejena, je bilo za vnos le-teh potrebna nova aktivnost z grafičnim vmesnikom (glej sliko 25). Ta nova aktivnost je sestavljena iz dveh okvirjev. Zgornji okvir določa, katero vrsto sklopa želimo ustvariti. Vanj je vključen po meri narejen gradnik Spinner, ki se od standardnega razlikuje po tem, da ima več možnosti za videz in druge nastavitve. Prikazan je na sliki 24.

Slika 24: Po meri narejen gradnik Spinner

V spodnjem okvirju definiramo površino in ime, ki ga aplikacija samodejno generira. V kolikor želimo drugačno poimenovanje, ga lahko spremenimo. Spodnji okvir, glede na izbiro v zgornjem, odpira različne razrede. Sklopi in njihovi podatki se shranijo v bazo SQLite.

Slika 25: Vnos novega konstrukcijskega sklopa

```
//Definiranje spremenljivk

public static Spinner mSpinner;

//MaterialiSpinnerVrstaSklopa je po meri (custom) narejen adapter in samostojen razred
(MaterialiSpinnerVrstaSklopa.class)

private ArrayAdapter<MaterialiSpinnerVrstaSklopa> spinnerArrayAdapter;

private long pos;
```

```
FragmentTransaction ft;

mSpinner = (Spinner)findViewById(R.id.spinner1);

spinnerArrayAdapter = new ArrayAdapter<MaterialiSpinnerVrstaSklopa>(this,
R.layout.spinner_entry, new MaterialiSpinnerVrstaSklopa[] {

                    new MaterialiSpinnerVrstaSklopa( 1, "Streha"),

                    new MaterialiSpinnerVrstaSklopa( 2, "Stena"),

                    new MaterialiSpinnerVrstaSklopa( 3, "Tla"),

                        });

//Registriranje adapterja s Spinnerjem

mSpinner.setAdapter(spinnerArrayAdapter);

mSpinner.setOnItemSelectedListener(new OnItemSelectedListener() {

        public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

                pos = spinnerArrayAdapter.getItemId(position);

                ft = fm.beginTransaction();

                if (pos == 0){

                //Če je pozicija pri Spinnerju na poziciji 0, se v spodnjem fragmentu odpre
                MaterialiVnosStrehaFragment.class

                ft.replace(R.id.fragment_spodaj, new MaterialiVnosStrehaFragment());

                }

                if (pos == 1){

                //Če je pozicija pri Spinnerju na poziciji 1, se v spodnjem fragmentu odpre
                MaterialiVnosStenaFragment.class

                ft.replace(R.id.fragment_spodaj, new MaterialiVnosStenaFragment());

                }

                if (pos == 2){

                //Če je pozicija pri Spinnerju na poziciji 2, se v spodnjem fragmentu odpre
                MaterialiVnosTlaFragment.class

                ft.replace(R.id.fragment_spodaj, new MaterialiVnosTlaFragment());

                }

                ft.commit();

                }
```

4.6.2.1 Vnos okvirjev v XML datoteko za videz

V grafičnem delu ustvarimo dva okvirja. Zgornji obstaja ločen v svojem XML zapisu.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   //zgornji del
   <include
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      layout="@layout/materiali_glavna_tab" />
   //spodnji del
   <FrameLayout
      android:id="@+id/fragment_spodaj"
      android:layout_width="match_parent"
      android:layout_height="match_parent" >
   </FrameLayout>
</LinearLayout>
```

4.6.3 Baza SQLite

SQLite je sistem za upravljanje relacijske baze podatkov, ki je shranjen v majhni (približno 350 KB veliki) C programirani knjižnici. V nasprotju z drugimi sistemi za upravljanje baz podatkov, SQLite ni ločen od aplikacije, ampak sestavni del. Programi uporabljajo SQLite s klicanjem preprostih funkcij z zmanjšanim dostopnim časom do baze podatkov.

4.6.3.1 Ustvarjanje baze

Potrebno je ustvariti nov razred MyDBhelper, ki ga razširjamo z SQLiteOpenHelperjem.

```
//Definiranje spremenljivk
public static final String TABLE_NAME_SKLOPI="tabela_sklopi";
public static final String SKLOP_ID= "_id";
```

```
public static final String SKLOP_NAME="title";
public static final String SKLOP_VRSTA="vrsta";
public static final String SKLOP_POVRSINA="povrsina";
private SQLiteDatabase db;
public class MyDBhelper extends SQLiteOpenHelper {
private static final String CREATE_TABLE_SKLOPI = "create table "
                        + TABLE_NAME_SKLOPI + " ("
                        //ID vsake vrstice baza sama ustvari
                        + SKLOP_ID + " integer primary key autoincrement, "
                        + SKLOP_NAME + ", "
                        + SKLOP_VRSTA + ", "
                        + SKLOP_POVRSINA + ");";
                @Override
                public void onCreate(SQLiteDatabase db) {
                try {
                //Ustvarjanje tabele "TABLE_NAME_SKLOPI"
                db.execSQL(CREATE_TABLE_SKLOPI);
                } catch (SQLiteException ex) {
                //Javljanje napake v kolikor pride do nje
                Log.v("Create table exception", ex.getMessage());
                }
        }
}
```

4.6.3.2 Vnašanje podatkov v bazo

V nadaljevanju je določen način kako naj se podatki iz poglavja 4.6.3.1 vnašajo v izdelano tabelo. Vsakič, ko uporabnik shrani sklop, se njegove vrednosti shranjujejo v gradnik ContentValues. V rdeče prikazanih stringih so vnesene vrednosti iz sklopa.

```
public long insertsklop (String title,String vrsta,String povrsina) {
        try{
        // V gradnik ContentValues se shranjujejo stringi title, vrsta ter povrsina
```

```
ContentValues newTaskValue = new ContentValues();
newTaskValue.put(SKLOP_NAME,title);
newTaskValue.put(SKLOP_VRSTA,vrsta);
newTaskValue.put(SKLOP_POVRSINA,povrsina);
//Vrednost newTaskValue se shrani v tabelo TABLE_NAME_SKLOPI
return db.insert(TABLE_NAME_SKLOPI, null, newTaskValue);
}
catch(SQLiteException ex) {
        //Javljanje napake v kolikor pride do nje
        Log.v("Insert into database exception caught",
                ex.getMessage());
                return -1;
                }
}
```

4.6.4 Meni ExpandableList s podanimi materiali

Ko vnesemo konstrukcijske sklope, lahko začnemo z dodajanjem materialov v posamezen sklop (klik na gumb "Dodaj material"). Sledi povezava na strežnik in branje datoteke XML v kateri so definirani materiali. V preglednici 2 in 3 so podani materiali, ki jih je možno dodajati v sklope. Z vsakim dodanim materialom so v bazo dodane naslednje lastnosti materiala:

- ime materiala
- gostota - ρ
- specifična toplota - c
- toplotna prevodnost - λ
- difuzijska upornost vodni pari - μ
- dodatni opis
- debelina sloja (mm)
- pozicija materiala v sklopu

Na sliki 26 je prikazano, kako so materiali s svojimi značilnostmi podani v datoteki XML. Videz teh vrednosti pa na gradniku ExpandableList na sliki 27.

Zidovi	Malte	Naravni kamen in zemlja	Polnila	Betoni
Polna opeka	Apnena malta	Granit	Pesek	Beton s kamnitimi agregati
Mrežasta opeka	Podaljšana apnena malta	Gnajs	Gramoz	Keramzitni beton
Porozna opeka	Cementna malta	Gosti apnenec	Zdrobljena opeka	Parjeni beton
Klinker opeka	Cementni estrih	Dolomit	Zdrobljena pluta	Beton iz opečnega drobirja
Bloki iz elektrofilskega pepela	Pigmentna fasadna malta	Marmor	Perlit	Beton iz žlindre
Silikatna opeka	Cementna malta s sintetičnimi dodatki	Peščenec	Keramzit	Celični beton
Porolit	Mavčna in apnena malta	Amorfni apnenec	Oblanci	
Žlindrin termoblok	Lahka mavčna malta	Pesek	Nasuta zemlja	
Bloki iz porobetona	Perlitna malta	Drobni gramoz		
Bloki iz celičastega betona	Toplotna izolacijska malta	Zaraščeno zemljišče		
Bloki iz lahkega betona	Mavčna malta na trstiki	Humus		
Bloki iz celičastega betona	Mavčna malta na rabic mreži			
Bloki iz betona				
Votlasta opeka				
Klinker opeka				

Preglednica 2: Možni materiali za vnos v konstrukcijski sklop (1)

Materiali za obloge	Kovine	Toplotni izolatorji
Azbest cementne plošče	Jeklo	Steklena volna
Mavčne kartonske plošče	Lito jeklo	Kamena volna
Polne mavčne plošče	Aluminjasta folija	Steklena pena
Mavčne plošče	Bakrena folija	Pluta
Klinker ploščice	Svinec	Plošče iz prešite trstike
Ploščice z opeke	Cink	Plošče iz stiskane slame
Fasadne plošče		Brizgani azbest
Keramične ploščice		Lesni beton
Keramični mozaik		Sintetične plošče iz večplastnega poliestra
Steklen mozaik		Plošče z akrilne smole
Linolej		PVMD in PVC plošče
Guma		Polistirenske plošče
Vnaprej izdelani betonski elementi		Polistiren
Lahki betonski elementi		Fenolne plošče
Plošče iz gostih apnencev		Poliuretanske plošče
Plošče iz peščenjaka		PVC plošče
Okensko steklo		Urea plošče
Armirano steklo		Ekstrudirani polistiren
Votli stekleni bloki		Ovčja volna
Plošče iz dolomita		Kokosova vlakna
Plošče iz marmorja		Vlaknaste lesne plošče
Hrastov les		Toplotnoizolacijski ometi
Smrekin les		Celulozna vlakna
Borov les		Bombaž
Panelne plošče		Perlitne plošče
Vezane plošče		Penjeno steklo
Iverne plošče		Poliuretanska pena
Plošče iz lesne volne		Perlitno nasutje
Papirnate tapete		
Bitumen		
Asfalt		
Bitumenska lepenka		
PVC		
Vinil azbestne plošče		
Preproge		
Deske za tla		
Parket		
Trde plošče iz lesnih vlaken		
Polietilenska folija		
PVC folija		
Bitumenski trak		
Bitumenski trakovi		
Strešna lepenka		
Večkratni bitumenski premaz		
Večplastna bitumenska hidroizolacija		
Večplastna bitumenska hidroizolacija na perforirani lepenki		
PVC strešni trakovi		
PIB (polizobuti) trakovi		
CR (kloropren-kavčuk) trakovi		
CMS (klorosulfidni polietilen) trakovi		
EPDM (etilen-propilen-kavčuk) trakovi		
Strešniki		
Akrilne plošče		

Preglednica 3: Možni materiali za vnos v konstrukcijski sklop (2)

```
<menu>
 <item>
  <skupina>
   Zidovi
  </skupina>
  <material>
   <name>
    Polna opeka
   </name>
   <opis>
    izvotljenost do 15%; gostota skupaj z votlinami
   </opis>
   <gostota>
    1800
   </gostota>
   <c>
    920
   </c>
   <lambda>
    0.76
   </lambda>
   <u>
    12
   </u>
  </material>
  <material>
   <name>
    Zlindrin termoblok
   </name>
   <opis/>
```

Slika 26: XML z materiali in njihovimi karakteristikami

4.6.4.1 Pridobitev XML podatkov iz oddaljenega strežnika do naše naprave

Spodaj prikazujemo način, kako pridobiti v pomnilnik naprave podatke iz datoteke XML preko URL - ja s pomočjo http zahtevka. Z njegovo pomočjo bomo v nadaljevanju razčlenili podatke in jih projicirali na gradnik ExpandableList.

```
// Inicializacija stringa, ki ga določimo kot prazno vrednost

String xml = null;

// Default HttpClient

DefaultHttpClient httpClient = new DefaultHttpClient();

//Naslov, kjer se nahaja želeni XML

HttpPost httpPost = new HttpPost("http://baza-sklopov.com/materiali.xml");

HttpResponse httpResponse = httpClient.execute(httpPost);

HttpEntity httpEntity = httpResponse.getEntity();

xml = EntityUtils.toString(httpEntity);
```

4.6.4.2 Document Object Model (DOM):

Ko pridobimo XML datoteko, je za nadaljnje upravljanje z informacijami v zapisu najlažja razčlenitev XML - ja in njegova pretvorba v element DOM. Mogoče ga je ustvariti z razčlenjevalnikom (parserjem) ali ročno. Slabost DOM izvedbe je velika poraba spomina, saj običajno zahteva, da se cel dokument naloži v pomnilnik in izdela celotno drevo elementov, preden je do njega dovoljen dostop. To ni v prid Android naprav, ker so zelo omejene pri porabi spomina.

```
Document doc = null;
        //Izgradnja gradnika DocumentBuilderFactory
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder db = dbf.newDocumentBuilder();
            InputSource is = new InputSource();
            //spremenljivka xml s prejšnje strani
            is.setCharacterStream(new StringReader(xml));
            //Členitev podatkov s pomočjo razčlenjevalnika
            doc = db.parse(is);
            //V primeru napak…
            } catch (ParserConfigurationException e) {
                Log.e("Error: ", e.getMessage());
                return null;
            } catch (SAXException e) {
                Log.e("Error: ", e.getMessage());
            return null;
            } catch (IOException e) {
                Log.e("Error: ", e.getMessage());
            return null;
            }
```

4.6.4.3 Node Element

Iz dokumenta DOM moramo dobiti vsak element (node) z njegovo vrednostjo, da jih lahko v nadaljevanju razdelimo po skupinah in podskupinah v drugih gradnikih. V našem primeru je to razporeditev v gradnik ArrayLists. Vnašanje vanj je predstavljeno v naslednjem poglavju.

```
public String getValue(int j, Element e, String str) {
            NodeList n = e.getElementsByTagName(str);
            return this.getElementValue(n.item(j));
        }
```

4.6.4.4 Pretvorba v ArrayLists z uporabo zank

Da se pride do vseh elementov v strukturi XML je potrebna zanka oz. več podzank za vse podrejene elemente. Na liki 27 je predstavljen gradnik ArrayLists, ki je potreben, da ExpandableList pravilno deluje.

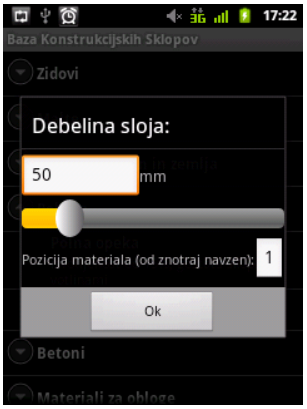
Slika 27: Podatki iz datoteke XML vnesene v gradnik ExpandableList

Spodaj je prikazana vrhnja, glavna zanka za razvrstitev materialov v skupine, podzanka pa zbere materiale in njihove karakteristike.

```
//Definiranje spremenljivk (v navednicah morajo biti imena node elementov)
static final String KEY_ITEM = "item";
static final String KEY_SKUPINA = "skupina";
static final String KEY_MATERIAL = "material";
static final String KEY_ID = "id";
static final String KEY_NAME = "name";
static final String KEY_OPIS = "opis";
static final String KEY_GOSTOTA = "gostota";
static final String KEY_C = "c";
static final String KEY_LAMBDA = "lambda";
static final String KEY_U = "u";
//Definiranje seznama in podseznama za skupino materialov oz materialov samih
List<Map<String, String>> groupData = new ArrayList<Map<String, String>>();
List<List<Map<String, String>>> childData = new ArrayList<List<Map<String, String>>>();
NodeList nl = doc.getElementsByTagName(KEY_ITEM);
//prva zanka (skupina materialov)
for (int i = 0; i < nl.getLength(); i++) {
```

```
            Map<String, String> map = new HashMap<String, String>();
            Element e = (Element) nl.item(i);
            map.put(KEY_SKUPINA, getValue(0, e, KEY_SKUPINA));
            groupData.add(map);
            List<Map<String, String>> children = new ArrayList<Map<String,
            String>>();
            //pod-zanka (materiali)
            for (int k = 0; k < e.getElementsByTagName(KEY_MATERIAL)
                        .getLength(); k++) {
                Map<String, String> podmap = new HashMap<String, String>();
                podmap.put(KEY_ID, getValue(k, e, KEY_ID));
                podmap.put(KEY_NAME, getValue(k, e, KEY_NAME));
                podmap.put(KEY_OPIS, getValue(k, e, KEY_OPIS));
                podmap.put(KEY_GOSTOTA, getValue(k, e, KEY_GOSTOTA));
                podmap.put(KEY_LAMBDA, getValue(k, e, KEY_LAMBDA));
                podmap.put(KEY_U, getValue(k, e, KEY_U));
                children.add(podmap);
            }
            childData.add(children);
        }
```

4.6.6 Gradnik AlertDialog

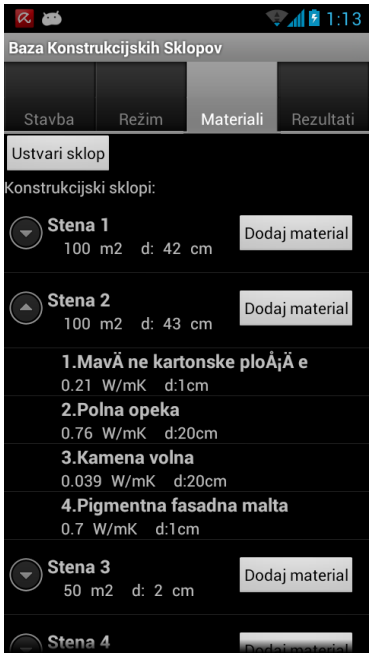


Slika 28: Gradnik AlertDialog, kjer izberemo debelino sloja izbranega materiala in njegovo pozicijo v sklopu

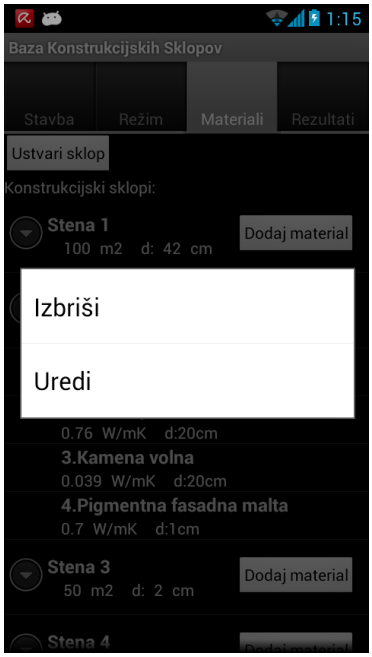
Z izbiro materiala se na zaslonu pojavi pop-up okno – gradnik AlertDialog, ki je prikazan na sliki 28. Vanj vnesemo debelino sloja materiala in pozicijo v sklopu. Tako dobimo razvrstitev materialov za vsak sklop.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);

LayoutInflater inflater = this.getLayoutInflater();

//Definiramo kateri layout naj se v AlertDialogu uporabi

View v = inflater.inflate(R.layout.materiali_izbira_debeline, null);

…

builder.create();

builder.show();
```

Ko so vneseni vsi materiali za konstrukcijski sklop s svojimi debelinami, se vrnemo na zavihek "Materiali". V gradniku ExpandableList so pod vsakim sklopom razvrščeni materiali, ki smo jih dodali. Primer je prikazan na sliki 29. Po želji lahko materiale in sklope tudi urejamo in brišemo, kot je prikazano je na sliki 30.



Slika 29: Sestava sklopov z dodanimi materiali



Slika 30: Urejanje že vnesenih sklopov ali materialov

4.7 Zavihek "Rezultati"

V zavihku "Rezultati" imamo 4 možnosti:

- vnesene informacije za objekt lahko shranimo na SD kartico,
- z SD kartice odpiramo že popisane objekte oziroma njihove delne vnose,

- pošljemo informacije o popisanem objektu v bazo MySQL, ki se nahaja na oddaljenem strežniku ter

- izračun toplotnih izgub ter toplotnega toka skozi vnesene konstrukcijske sklope

4.7.1 Shranjevanje podatkov v obliki XML na SD kartico

Podatki se na SD kartico shranjujejo v mapi "Baza Konstrukcijskih Sklopov" in sicer v XML obliki. V AndroidManifestu moramo vnesti dovoljenje za dostop do SD kartice. Določiti moramo s kakšnim imenom se bo datoteka shranjevala. Zapis vnesenega objekta je shranjen z naslednjo obliko "poštna številka,naselje,ime ulice,hišna številka.xml". Podatke iz našega popisa aplikacija dobi iz notranje baze SQLite in jih s pomočjo XmlSerializerja beleži na ustvarjeno datoteko po ustreznem zaporedju. V primeru uspešne shranitve nas program obvesti s tekstom "Podatki iz … so bili shranjeni na SD kartico!". Vnosov iz same aplikacije ni mogoče brisati. Brisanje je mogoče z drugimi dostopi do SD kartice.

```
//Definiranje datoteke (mape, imena in končnice), ki bo shranjena na SD kartico

File newxmlfile = new File(Environment.getExternalStorageDirectory()+"/Baza Konstrukcijskih Sklopov/"+imeObjekta+".xml");

    try{

        newxmlfile.createNewFile();

      //V primeru napake (neizpolnitve ukaza)

    }catch(IOException e){

        Log.e("IOException", "exception in createNewFile() method");

    }

    //Povežemo novo datoteko z FileOutputStream - om

    FileOutputStream fileos = null;

    try{

        fileos = new FileOutputStream(newxmlfile);

    //V primeru napake (neizpolnitve ukaza)

    }catch(FileNotFoundException e){

        Log.e("FileNotFoundException", "can't create FileOutputStream");

    }

    //Za pisanje v oblki XML najprej ustvarimo XmlSerializer

    XmlSerializer serializer = Xml.newSerializer();

    try {
```

```
 //nastavimo FileOutputStream kot output za serializer z uporabo UTF-8 kodiranja (z
 drugimi – ANSI, Unicode prihaja do težav s šumniki)

serializer.setOutput(fileos, "UTF-8");

// Zapis <?xml izjave

serializer.startDocument(null, Boolean.valueOf(true));

//Nastavitev možnosti funkcij

 serializer.setFeature("http://xmlpull.org/v1/doc/features.html#indent-output", true);

 //začetek XMLja z <root>

serializer.startTag(null, "root");

 //začetek splošnih podatkov o stavbi z <stavba-objekt>

serializer.startTag(null, "stavba-objekt");

 //začetek za vnos imena občine z začetnim indikatorjem <obcina>

serializer.startTag(null, "obcina");

 //vnos teksta – v mojem primeru podatki o stavbi prihajajo iz baze SQLite

 serializer.text(String.valueOf(podatkiStavbe.getString(podatkiStavbe.getColumnIndex
 ("obcina"))));

 //označitev konca vnosa imena občine z </obcina>

serializer.endTag(null, "obcina");

 //začetek za vnos poštne številke z začetnim indikatorjem <postna>

serializer.startTag(null, "postna");

 // vnos teksta – poštne številke

 serializer.text(String.valueOf(podatkiStavbe.getString(podatkiStavbe.getColumnIndex
 ("postna"))));

 //označitev konca vnosa številke poštne z </postna>

 serializer.endTag(null, "postna");

 …

 //označitev konca vnosov o splošnih podatkih stavbe, ki ga zaključimo z indikatorjem
 </stavba-objekt>

 serializer.endTag(null, "stavba-objekt");

 //Cursor z naborom vseh sklopov v bazi

 mSklopiCursor = dba.getsklopi();
```

```
//Dokler obstaja sklop, za vsakega izbranega ponavljaj

while (mSklopiCursor.moveToNext()) {

        try{

        //začetek podatkov o sklopih z indikatorjem <sklop>

serializer.startTag(null, "sklop");

        //ID številka vsakega sklopa iz baze

        Integer steviloSklopa =
        mSklopiCursor.getInt(mSklopiCursor.getColumnIndex("_id"));

        //Cursor z naborom vseh materialov v konstrukcijskem sklopu z določenim
        IDjem

        mSklopiCursorMateriali = dba.getsklopiVSE(steviloSklopa);

mSklopiCursorMateriali.moveToPosition(-1);

        //Dokler obstaja sklop z ID – jem steviloSklopa ponavljaj

while (mSklopiCursorMateriali.moveToNext()) {

try{

        //začetek za vnos poštne številke z začetnim indikatorjem <postna>

serializer.startTag(null, "vnosmaterial");

// <id>

serializer.startTag(null, "id");

        // vnos (tekst) IDja vstavljenega materiala

        serializer.text(String.valueOf(mSklopiCursorMateriali.getString(mSklopiCurs
        orMateriali.getColumnIndex("_id"))));

        // </id>

        serializer.endTag(null, "id");

        // ID sklopa <IDsklopa>

        serializer.startTag(null, "IDsklopa");

        // vrednost ID sklopa

        serializer.text(String.valueOf(mSklopiCursorMateriali.getString(mSklopiCurs
        orMateriali.getColumnIndex("IDsklopa"))));

        // </IDsklopa>

        serializer.endTag(null, "IDsklopa");
```

```
                // <imeSklopa>

                serializer.startTag(null, "imeSklopa");

  // Ime sklopa v katerem se material nahaja. Navadno kot Streha 1,2.../Stena 1,2... ali
  Tla 1,2…
  serializer.text(String.valueOf(mSklopiCursorMateriali.getString(mSklopiCursorMater
  iali.getColumnIndex("title"))));

  //</imeSklopa>

  serializer.endTag(null, "imeSklopa");

  …

  serializer.startTag(null, "u");

  serializer.text(String.valueOf(mSklopiCursorMateriali.getString(mSklopiCursorMater
  iali.getColumnIndex("u"))));

serializer.endTag(null, "u");

  // konec vnosa informacij za trenutni obdelani material </vnosmaterial>

  serializer.endTag(null, "vnosmaterial");

  }

  …

  }

  //Konec podatkov o sklopih.

  serializer.endTag(null, "sklop");

  }

  //Zaključek XML drevesa

  serializer.endTag(null, "root");

serializer.endDocument();

  //Zapis xml podatkov v FileOutputStream

serializer.flush();

//Zaključimo z zapiranjem "file streama"

fileos.close();

  //Prikaz sporočila v gradniku TextView, da so bili podatki uspešno vneseni na SD
  kartico

result.setText("Podatki iz " + imeObjekta + " so bili shranjeni na SD kartico!");

  …}
```

4.7.2 Odpiranje podatkov s SD kartice

4.7.2.1 Iskanje razpoložljivih datotek na SD kartici in vnos razpoložljivih datotek v gradnik AlertDialog

Odpiranje datoteke se najprej začne z iskanjem razpoložljivih shranjenih vnosov v mapi "Baza Konstrukcijskih Sklopov" na SD kartici. Temu sledi vnos razpoložljivih datotek v gradnik AlertDialog. Videz je prikazan na sliki 31.

Slika 31: Vnos razpoložljivih predhodno shranjenih vnosov na SD kartici v gradnik AlertDialog

Koda je prikazana spodaj.

```
try{    dir = new File(Environment.getExternalStorageDirectory() + "/Baza Konstrukcijskih
        Sklopov");

        }

        //V primeru napak…

        catch (Exception e){

       Log.e("Exception", e.getMessage());

        }

        //Dodajanje vsakega vnosa v skupino

        File[] files = dir.listFiles(new XmlFileFilter());

        //Ustvarjanje ArrayLista

        ArrayList<String> fileList = new ArrayList<String>();

        for (File f : files) {

            String fileIME = f.getName();

            fileList.add(fileIME);
```

```
}
// Definiranje gradnika AlertDialog
coco = new AlertDialog.Builder(mContextOdpri);
//Vgrajevanje custom layouta v AlertDialog
LayoutInflater inflater = this.getLayoutInflater();
final View v = inflater.inflate(R.layout.open_file, null);
//Vnašanje naslova v layout
coco.setView(v).setTitle("Izberite podatke iz željene lokacije");
//Registriranje gradnika ListViewa z komponento ListViewa v layoutu
ListView mainListView = (ListView) v.findViewById(R.id.listView1);
//Vgraditev ArrayLista v ArrayAdapter
ArrayAdapter<String> openAdapter = new ArrayAdapter<String>(this,
R.layout.materiali_vrstica, R.id.tvGroupName, fileList);
mainListView.setAdapter(openAdapter);
//Kreiranje AlertDialoga
AlertDialog alert = coco.create();
//Kaj se zgodi, ko kliknemo na določeno izbiro na ListViewu
mainListView.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                        long arg3) {
        …
        //Iz izbire razberemo ime (se uporabi pri parseXML();)
        imeOpenDatoteke = (String) arg0.getItemAtPosition(arg2);
        …
        //Ko je izbran določen objekt, sledi skok v private void parseXML()
        parseXML();
        //Ko gre skozi void parseXML() sledi zapiranje AlertDialoga
        alert.dismiss();
        } });
alert.show();
```

4.7.2.2 Odpiranje XML datoteke in razbiranje podatkov iz XML strukture

V tem razdelku sledi prikaz kode, s katero aplikacija s pomočjo gradnika XmlPullParser bere podatke iz izbranega dokumenta XML. Prva zanka poišče podatke o stavbi, druga pa še njene konstrukcijske sklope in materiale.

```
private void parseXML() throws XmlPullParserException, IOException {

    try{

    //Odpiranje prej izbranega objekta

    File file = new File(Environment.getExternalStorageDirectory()

    + "/Baza Konstrukcijskih Sklopov/" + imeOpenDatoteke);

    //Nastavitve za xml parsing

    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();

    factory.setNamespaceAware(true);

    XmlPullParser xpp = factory.newPullParser();

    //Ustvari vhodni tok, s katerega bo stream reader bral.

    FileInputStream fis = new FileInputStream(file);

    //Nastavitve vhoda za parser z uprabo InputStreamReaderja

    xpp.setInput(new InputStreamReader(fis));

    //Tip dogodka

    int eventType = xpp.getEventType();

    //Dokler eventType "ni enak" koncu XML dokumenta izvajaj…

    while (eventType != XmlPullParser.END_DOCUMENT) {
```

//Ko gre bralnik skozi vsako drevesno strukturo v XML – ju, vzame iz strukture: če je eventType "start tag" - <…> in je pri tem definiran kot "občina". Najprej iščemo podatke, ki se bodo shranili v bazi SQLite pod tabelo o **podatkih stavbe** (brez sklopov in vgrajenih materialov).

```
    if ((eventType == XmlPullParser.START_TAG) &&(xpp.getName().equals("obcina"))){

            //Vzemi ven takoj naslednji tekst po "start tagu" in ga definiraj kot String
            STAVBA_OBCINA

            STAVBA_OBCINA = xpp.nextText();

            //Pojdi na naslednji tag

            xpp.nextTag();

            //Definiraj naslednji tekst kot String STAVBA_POSTNA
```

```
STAVBA_POSTNA = xpp.nextText();

//Pojdi na naslednji tag

xpp.nextTag();

…ponavljamo enako dokler ne pridemo do zadnjega podatka o številu ljudi v
gospodinjstvu

}
```

//V strukturi poiščemo še kje je "start tag" z vsebovanim "_id". Podobno kot prej dobimo vse naslednje podatke. V tem primeru se bodo informacije shranjevale v tabeli o **konstrukcijskih sklopih in vgrajenih materialih**.

```
if ((eventType == XmlPullParser.START_TAG) &&(xpp.getName().equals("_id"))){

KEY_ID = xpp.nextText();

xpp.nextTag();

SKLOP_ID_MATERIALI = xpp.nextText();

xpp.nextTag();

SKLOP_NAME = xpp.nextText();

xpp.nextTag();

…

//Na enak način kot prej dobimo vse želene podatke

}
```

4.7.2.3 Vnašanje prebranih informacij iz XML – ja v bazo SQLite

Da bi aplikacija v nadaljevanju uporabila shranjene podatke pravilno, jih je potrebno iz datoteke XML nazaj shraniti v bazo SQLite.

```
//Vnos sklopa, če ta že ni vnešen

Cursor mCursor = dba.getsklopiVSE(Integer.valueOf(SKLOP_ID_MATERIALI));

//Šteje vnose v Cursorju

Integer stev = mCursor.getCount();

//Če še ni nobenega…

if (stev.equals(0)) {

        //…vstavitev v bazo – podatki o sklopu

        dba.insertsklop(SKLOP_NAME, SKLOP_VRSTA, SKLOP_POVRSINA,
        STAVBA_OBCINA, STAVBA_POSTNA, STAVBA_NASELJE, STAVBA_ULICA,
        STAVBA_HISNA, STAVBA_DODATEK_HISNI, STAVBA_STEVILKA_STANOVANJA,
        STAVBA_TIP_STAVBE, STAVBA_LETO_IZGRADNJE, STAVBA_VISINA,
```

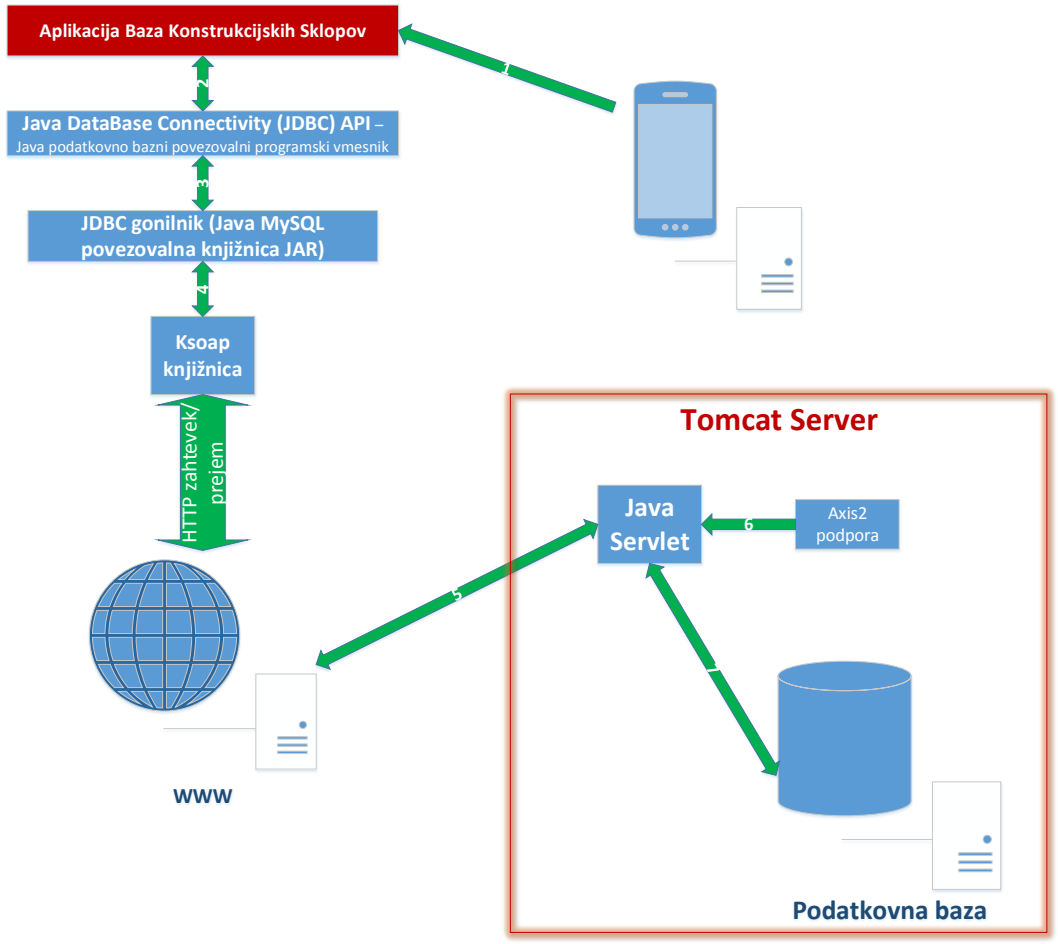
```
STAVBA_UPORABNA_POVRSINA, STAVBA_NETO_TLORIS,
STAVBA_POVRSINA_POD_STAVBO, REZIM_DAN, REZIM_NOC,REZIM_LJUDI);

}

//Vstavitev v bazo – podatki o stavbi

dba.insertstavba(STAVBA_OBCINA,STAVBA_POSTNA,STAVBA_NASELJE,STAVBA_
ULICA,STAVBA_HISNA,STAVBA_DODATEK_HISNI,STAVBA_STEVILKA_STANOV
ANJA,STAVBA_TIP_STAVBE,STAVBA_LETO_IZGRADNJE,STAVBA_VISINA,STAV
BA_UPORABNA_POVRSINA,STAVBA_NETO_TLORIS,
STAVBA_POVRSINA_POD_STAVBO);

//Vstavitev v bazo – podatki o režimu

dba.insertrezim(REZIM_DAN,REZIM_NOC,REZIM_LJUDI);

//Vstavitev v bazo – podatki o vsakem materialu

dba.insertmaterial(STAVBA_OBCINA,STAVBA_POSTNA,STAVBA_NASELJE,STAVBA
_ULICA,STAVBA_HISNA,STAVBA_DODATEK_HISNI,STAVBA_STEVILKA_STANO
VANJA,STAVBA_TIP_STAVBE,STAVBA_LETO_IZGRADNJE,STAVBA_VISINA,STA
VBA_UPORABNA_POVRSINA,STAVBA_NETO_TLORIS,STAVBA_POVRSINA_POD_
STAVBO,REZIM_DAN,REZIM_NOC,REZIM_LJUDI,Integer.parseInt(SKLOP_ID_MATE
RIALI),SKLOP_NAME,SKLOP_VRSTA,SKLOP_POVRSINA,KEY_MATERIAL,KEY_ID
_ZAPOREDNA,Integer.parseInt(KEY_DEBELINA),KEY_NAME,KEY_OPIS,KEY_GOST
OTA,KEY_C,KEY_LAMBDA,KEY_U);

}

eventType = xpp.next();

}

//Če je vse OK se izpiše tekst "Podatki so bili odprti!"

result.setText("Podatki so bili odprti!");}
```

Ko so vsi podatki pretvorjeni in preneseni v bazo SQLite, je v vsakem okvirju potrebna še posodobitev EditTextov in drugih gradnikov, da se na njih pojavi shranjena vsebina.

4.7.3 Pošiljanje podatkov na strežnik v bazo MySQL

Na sliki 32 je prikazan potek podatkov od pametne naprave do baze MySQL, kot v primeru naše aplikacije. Da je to mogoče, je potrebna postavitev in uporaba komponent, ki so predstavljene v nadaljevanju. Najpomembnejše so:

- podatkovna baza MySQL,
- Java web service (Java servlet) ter
- pošiljka iz Android aplikacije



Slika 32: Tok poteka podatkov od aplikacije do podatkovne baze

V aplikaciji potrebujemo gonilnik JDBC (Java Database Connectivity) pri katerem gre za Java osnovano podatkovno sprejemno tehnologijo, ki definira kako se uporabnik lahko poveže z podatkovno bazo. Razpolaga z metodami za poizvedovanje (query) in posodabljanje podatkov v bazi (update). Poleg tega mora biti nameščena tudi ksoap2 knjižnica, ki omogoča pošiljanje podatkov servletu, ki je lociran na oddaljenem strežniku.

Java servlet je razred programskega jezika java, ki se uporablja za razširitev zmogljivosti strežnika. Čeprav se lahko strežnik odzove na vse vrste zahtev, se pogosto uporablja za razširitev aplikacij, ki jih gosti spletni strežnik. Predstavljamo si jih lahko kot javanske programe, ki tečejo na strežnikih namesto v spletnih brskalnikih.

Da na strežniku lahko uporabimo servlet, je potrebna namestitev Tomcata, ki poskrbi za razvojno okolje in nastavitve strežnika. Apache Tomcat je odprtokodni spletni strežnik in ponudnik servletov. Potrebna je tudi nastavitev Apache Axis2 SOAP engina, ki spletnim aplikacijam omogoča prenos podatkov.

S HTTP pošiljko iz pametne naprave servlet sprejme poslane podatke. Ta jih preoblikuje in pošlje podatkovni bazi MySQL, ki je nameščena na istem strežniku. MySQL je sistem za upravljanje relacijskih baz podatkov (*RDBMS). V kolikor je pošiljka uspešna, servlet lahko pošlje povratno informacijo, da je bilo pošiljanje uspešno. V našem primeru pošlje sporočilo "Podatki uspešno poslani!", kot je prikazano na sliki 32.

* beseda je pojasnjena v poglavju TERMINOLOŠKI SLOVAR

Slika 33: Potrditev uspešno poslanih podatkov na oddaljen strežnik

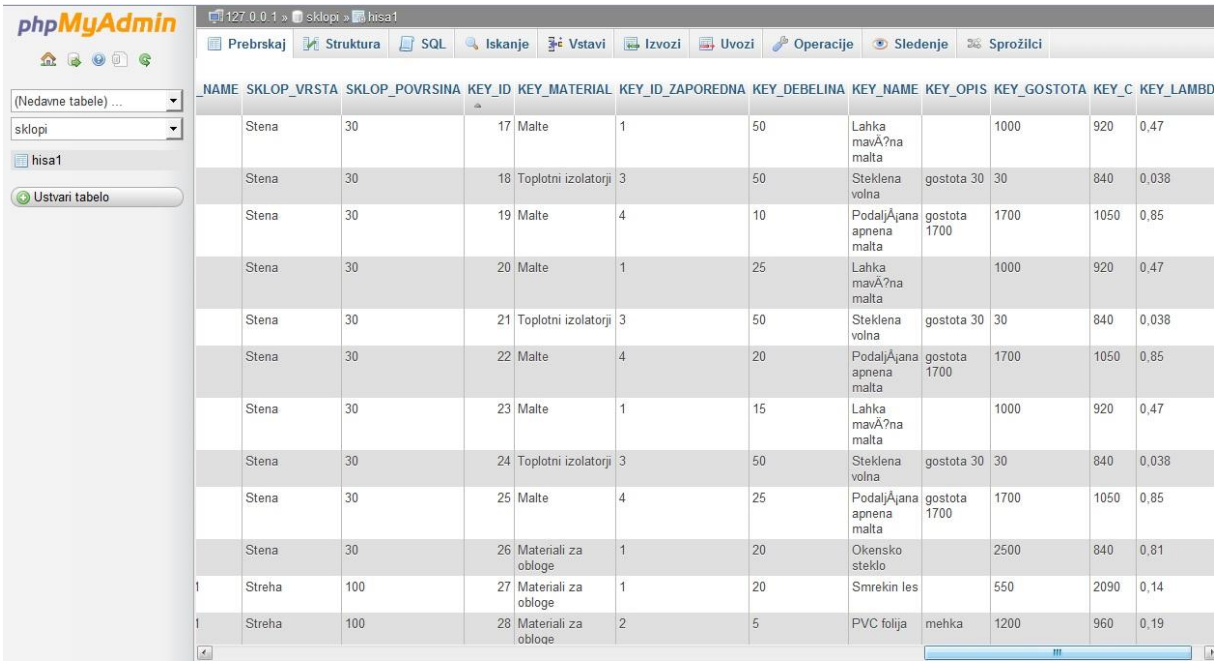
V primeru internet povezave preko Wi-Fi – ja, je pošiljanje potekalo uspešno. Zaradi neznanih razlogov pošiljanje preko mobilne podatkovne povezave ne uspe. MySQL z navideznim strežnikom smo testirali s pomočjo programa XAMPP, kot je prikazano na sliki 34. Servlet pa smo lahko namestili s pomočjo Eclipsa.

4.7.3.1 Pošiljka iz naprave Android

V nadaljevanju prikazujemo izrez programske kode, s katero omogočamo pošiljanje paketa na oddaljen javin servlet z vsemi podatki, ki smo jih zbrali v aplikaciji.

```
private final String NAMESPACE = "http://ws.apache.org/axis2";
private final String URL = "http://192.168.1.103:8080/InsertToUsers/services/Users?wsdl";
private final String SOAP_ACTION = "http://ws.login.com/axis2/insertData";
private final String METHOD_NAME = "insertData";
…
SklopiCursor = dba.getmateriali();
while (SklopiCursor.moveToNext()) {
        SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
        try {
        PropertyInfo _idProp =new PropertyInfo();
        //Definiranje imena spremenljivke v servletu
        _idProp.setName("_id");
        //Določanje vrednosti spremenljivke "_id"
        _idProp.setValue(SklopiCursor.getString(SklopiCursor.getColumnIndex("_id")));
```

```
        //Definiranje tipa spremenljivke
        _idProp.setType(String.class);
        //Prenos informacij na request
        request.addProperty(_idProp);
        PropertyInfo obcinaProp =new PropertyInfo();
        //Definiranje imena spremenljivke v servletu
        obcinaProp.setName("obcina");
        //Določanje vrednosti spremenljivke "obcina"
        obcinaProp.setValue(SklopiCursor.getString(SklopiCursor.getColumnIndex("obcina")));
        //Definiranje tipa spremenljivke
        obcinaProp.setType(String.class);
        //Prenos informacij na request
        request.addProperty(obcinaProp);
        …sledijo še vsi ostali podatki
//Definiranje "kuverte"
SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
//V paket vnesemo prej definiran request
envelope.setOutputSoapObject(request);
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);
androidHttpTransport.call(SOAP_ACTION, envelope);
SoapPrimitive response = (SoapPrimitive)envelope.getResponse();
result.setText(response.toString());
poslano = response.toString();
}
//Če je pošiljanje neuspešno
catch(Exception e){
result.setText("Pošiljanje neuspešno");
}
}
```



_NAME	SKLOP_VRSTA	SKLOP_POVRSINA	KEY_ID	KEY_MATERIAL	KEY_ID_ZAPOREDNA	KEY_DEBELINA	KEY_NAME	KEY_OPIS	KEY_GOSTOTA	KEY_C	KEY_LAMBD
	Stena	30	17	Malte	1	50	Lahka mavÂ?na malta		1000	920	0,47
	Stena	30	18	Toplotni izolatorji	3	50	Steklena volna	gostota 30	30	840	0,038
	Stena	30	19	Malte	4	10	PodaljÅ¡ana apnena malta	gostota 1700	1700	1050	0,85
	Stena	30	20	Malte	1	25	Lahka mavÂ?na malta		1000	920	0,47
	Stena	30	21	Toplotni izolatorji	3	50	Steklena volna	gostota 30	30	840	0,038
	Stena	30	22	Malte	4	20	PodaljÅ¡ana apnena malta	gostota 1700	1700	1050	0,85
	Stena	30	23	Malte	1	15	Lahka mavÂ?na malta		1000	920	0,47
	Stena	30	24	Toplotni izolatorji	3	50	Steklena volna	gostota 30	30	840	0,038
	Stena	30	25	Malte	4	25	PodaljÅ¡ana apnena malta	gostota 1700	1700	1050	0,85
	Stena	30	26	Materiali za obloge	1	20	Okensko steklo		2500	840	0,81
1	Streha	100	27	Materiali za obloge	1	20	Smrekin les		550	2090	0,14
1	Streha	100	28	Materiali za obloge	2	5	PVC folija	mehka	1200	960	0,19

Slika 34: Prejeti sklopi z materiali v bazi MySQL na strežniku

4.7.4 Izračun toplotnih izgub in toplotnega toka skozi stavbo

Izračuni v tem zavihku niso vezani na tipizacijo objektov. Izračun poteka na osnovi osnovne enačbe prevajanja toplote $\frac{P}{A} = \frac{\lambda(T_1 - T_2)}{L}$ z namenom ilustracije možnosti, ki jih lahko ponuja taka aplikacija. V prihodnje bi te izračune lahko izboljšali z upoštevanjem katere od novejših metodologij.

V nadaljevanju je pojasnjena teorija prevajanja toplote z namenom razumevanja enačb, ki smo jih v aplikaciji uporabili. Aplikacija mora izračunati toplotne izgube ($L_D + L_S = \overline{U}A$), celotni toplotni tok skozi objekt ($P = \overline{U}A\Delta T$) ter ločeno toplotni tok skozi objekt podnevi ($P = \overline{U}A\Delta T_{podnevi}$) in toplotni tok skozi objekt ponoči ($P = \overline{U}A\Delta T_{ponoci}$). Primer uspešno dobljenih rezultatov v aplikaciji je prikazan na sliki 36.

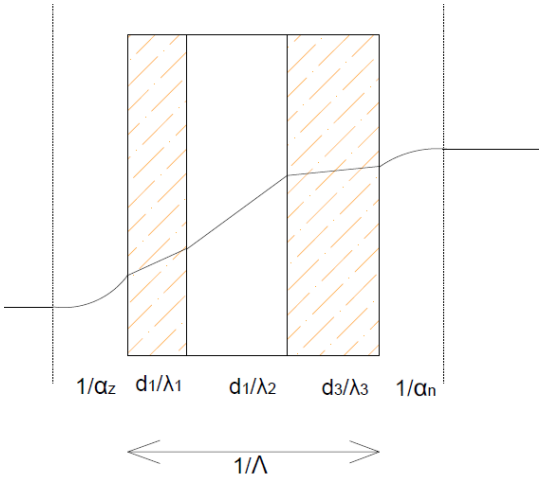
4.7.4.1 Toplota

Ker so vsi materiali iz konstrukcijskega ovoja stavb v večji ali manjši meri toplotni prevodniki, prehaja toplota v zimskem času navzven, poleti pa navznoter. Prehajanje toplote skozi plašč nastane zaradi prevajanja toplote skozi material in zaradi konvekcije zraka. Izgube, ki jih povzroča prevajanje skozi material, lahko dokaj natančno določamo. Izgube s konvekcijo (prezračevanje, netesnost oken in vrat, vpliv vetra ipd.) pa le približno. V tej diplomski nalogi oziroma s to aplikacijo se osredotočamo le na prvo.

4.7.4.1.1 Toplotni tok skozi večplastni sistem; koeficient prehoda U

Slika 35 ponazarja sistem N vzporednih plasti, različnih debelin in lastnosti materiala, ki so z obeh straneh obdani z zrakom temperature T_z na zunanji strani in T_n na notranji strani.

Zunanja temperatura stene v splošnem ni enaka temperaturi zunanjega zraka daleč od stene zaradi vrste pojavov, ki ob steni nastopajo. Takšni značilni pojavi so: konvekcija toplote, sevanje, vpadli solarni energijski tok in prenos latentne toplote (izparevanje, kondenzacija in zmrzovanje vodne pare). Iz navedenih razlogov je potrebno razlikovati temperaturo zunanje stene T_{sz} od temperature zunanjega zraka daleč od stene, T_z, pri čemer torej velja $T_{sz} \neq T_z$. V splošnem za notranjo steno velja podobno razmišljanje, temperatura notranje stene T_{sn} ni enaka temperaturi zraka v notranjosti, T_n, daleč od mejne plasti [17].



Slika 35: Toplotna prehodnost skozi različne plasti sklopa

Osnovna enačba prevajanja toplote (brez snovnih tokov, to je t.im. kondukcija toplote):

$\frac{P}{A} = \frac{\lambda(T_1 - T_2)}{L}$, kjer so

P [W] …toplotni tok (fizikalna količina, ki pove koliko toplote v časovni enoti preteče med dvema telesoma v toplotnem stiku).

Skladno z 2. zakonom termodinamike teče toplotni tok spontano vedno le v smeri od telesa z višjo temperaturo k telesu z nižjo [18].

$P = \frac{dQ}{dt} = \frac{\lambda A \Delta T}{L}$ [W], kjer je

A [m^2] …obravnavana površina (prečni presek),

λ [W/mK] …koeficient toplotne prevodnosti dane snovi,

$(T_1 - T_2)$ [K] …temperaturna razlika med notranjo in zunanjo temperaturo,

L [m]…dolžina (širina) materiala skozi katerega obravnavamo prehod toplote.

$U = \frac{\lambda}{L} = \frac{1}{R_{Tcel}}$ [W/m^2K], kjer je

U…celotna toplotna prehodnost, ki upošteva prehod toplote skozi sistem ravnih sten in vključuje prevajanje, konvekcijo in sevanje.

Toplotni tok, ob upoštevanju pojavov na mejnih plasteh, ki teče skozi več plasti:

$$P = UA\Delta T$$

Celotni upor toplotnega pretoka R_{Tcel}, skozi sistem zaporedno postavljenih elementov, vključno z obema mejnima plastema:

$$R_{Tcel} = R_{SZ} + \sum_i R_{Ti} + R_{SN}$$

$$R_{Tcel} = \frac{1}{\alpha_z} + \frac{1}{\Lambda} + \frac{1}{\alpha_n}$$, kjer je

Λ …upor toplotnega pretoka večslojnega elementa.

$$\frac{1}{\Lambda} = \sum_i \frac{d_i}{\lambda_i}$$

1/Λ je vrednost prevodnosti toplotnega pretoka. Podaja toplotno karakteristiko sestave medtem ko koeficient *U*, toplotna prehodnost [W/m^{2}K] podaja toplotno karakteristiko elementa upoštevaje še mejni plasti zraka, katerih koeficient prestopa toplote je α_z in α_n.

Na osnovi zapisanega sledi, da je toplotna prehodnost*, U*, sistema na sliki 35, podana z

$$U = \frac{1}{\frac{1}{\alpha_z} + \sum_{i=1}^{3} \frac{d_i}{\lambda_i} + \frac{1}{\alpha_n}}$$

4.7.4.1.2 Popravni koeficienti za toplotni prehod strehe in tal

Celotno zgradbo tvori več zidov, oken, tla in streha, zato izračunamo povprečno vrednost *U*:

$$\overline{U} = \frac{U_1 S_1 + U_2 S_2 + ... + U_n S_n}{S_1 + S_2 + ... + S_n}$$, kjer je

U_n…toplotna prehodnost posameznega sklopa [W/m^{2}K] ter

S_n…površina obravnavanega sklopa.

Upoštevamo, da so toplotne izgube skozi streho zaradi obsevanja manjše, enako velja za dele stavb, ki so v stiku s tlemi, ki imajo navadno višjo T od ozračja, ali pa s pomožnimi prostori (klet, skladišča, …). V diplomi upoštevam empirične faktorje po [19]. Pri strehah je upoštevano korigiranje koeficienta prehodnosti s faktorjem 0.8, pri tleh pa z 0.5. Gre za poenostavitve, saj je za natančnejše modele potrebno natančnejša določitev začetnih in robnih pogojev, poleg tega pa gre tudi za zahtevnejši matematični model. V modelu tudi ne upoštevam oblike stavbe, ki vpliva na toplotne izgube stavbe.

$$\overline{U} = \frac{0.5 * U_{TLA} S_{TLA} + 0.8 * U_{STREHA} S_{STREHA} + ... + U_n S_n}{S_{TLA} + S_{STREHA} + ... + S_n}$$

Toplotne izgube skozi objekt:

$$L_D + L_S = \overline{U} A$$

Celotni toplotni tok skozi objekt:

$$P = \overline{U} A \Delta T$$

4.7.4.1 Uporaba enačb za prehod toplote v programski kodi

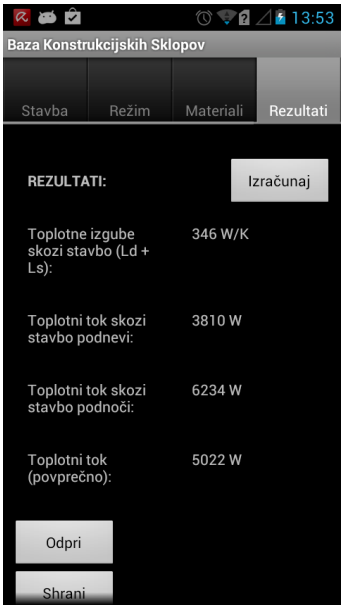
V nadaljevanju je prikazano kako uporabljamo enačbe za izračun toplotne prehodnosti v programski kodi. Potrebno je dobiti podatke iz notranje baze ter jih po pravilnem zaporedju klicati in obravnavati s pomočjo zank in pogojnih stavkov.

```
//Najprej moramo dobiti sklope iz notranje baze SQLite
mSklopiCursor = dba.getsklopi();
mSklopiCursor.moveToPrevious();
    //Dokler obstajajo sklopi ponavljaj
    while (mSklopiCursor.moveToNext()) {
        //Dobi id številko sklopa
        Integer steviloSklopa = mSklopiCursor.getInt(mSklopiCursor.getColumnIndex("_id"));
        //Dobi iz baze le materiale za sklop, ki ima zgoraj dobljen id.
        mSklopiCursorMateriali = dba.getsklopiVSE(steviloSklopa);
        //Štartaj s seštevanjem R od tu
        double R = 0;
        mSklopiCursorMateriali.moveToPrevious();
                //Ponavljaj dokler obstajajo materiali za določen id sklopa
                while (mSklopiCursorMateriali.moveToNext()) {
                    //Dobi debelino za ta material
                    double debelina =
                    Double.parseDouble(mSklopiCursorMateriali.getString(mSklopiCursorMateri
                    ali.getColumnIndex("debelina")));
                    //Dobi lambdo za ta material
```

```
            double lambda =
            Double.parseDouble(mSklopiCursorMateriali.getString(mSklopiCursorMateri
            ali.getColumnIndex("lambda")));

            //Seštej vse toplotne upore R od vsakega materiala (za vsak posamezen sklop)
            in jih seštevaj

            R = R + (debelina/1000/lambda);

}

//Izračunaj toplotni prehod U vsakega sklopa

U = 1/(0.04+R+0.13);

//Dobi površino sklopa

double S =
Double.parseDouble(mSklopiCursor.getString(mSklopiCursor.getColumnIndex("povr
sina")));

//Če so vrsta obravnavanega sklopa "Tla"…

If
(mSklopiCursor.getString(mSklopiCursor.getColumnIndex("vrsta")).equals("Tla")){

            //…potem množi vrednost U krat 0.5.

            USSklop = 0.5 * U * S;

            }

//Če je vrsta obravnavanega sklopa "Streha"…

if
(mSklopiCursor.getString(mSklopiCursor.getColumnIndex("vrsta")).equals("Streha"))
{

//…potem množi vrednost U krat 0.5.

USSklop = 0.8 * U * S;

}

//Če je vrsta obravnavanega sklopa "Stena"…

if
(mSklopiCursor.getString(mSklopiCursor.getColumnIndex("vrsta")).equals("Stena"))
{

//…potem ni potrebnih popravnih faktorjev

USSklop = U * S;

}

//Seštevek vseh USSklop (x*U*S) na objektu
```

```
            USZaSklop = USZaSklop + USSklop;
            //Seštevek vseh površin sklopov (S)
            SSkupaj = SSkupaj + S;
        }

//Izračun $\overline{U}$
Upovprecni = USZaSklop/ SSkupaj;
//Dobi zadnje vnesene podatke za režim stavbe
RezimCursori = dba.getrezimi();
RezimCursor = dba.getrezim(RezimCursori.getCount());
    //Dobi dnevne in nočne temperature, znotraj in izven objekta
    T1 = Double.parseDouble(RezimCursor.getString(RezimCursor.getColumnIndex("dan")));
    T2 = Double.parseDouble(RezimCursor.getString(RezimCursor.getColumnIndex("noc")));
    T1zunaj =
    Double.parseDouble(RezimCursor.getString(RezimCursor.getColumnIndex("zunajdan")));
    T2zunaj =
    Double.parseDouble(RezimCursor.getString(RezimCursor.getColumnIndex("zunajnoc")));
    TspremembaDan = (T1-T1zunaj);
    TspremembaNoc = (T2-T2zunaj);
    //Dobimo povprečno spremembo temperature podnevi in ponoči
    Tsprememba = (TspremembaDan + TspremembaNoc)/2;
    // Izračun toplotnega toka skozi objekt
    P = Upovprecni* SSkupaj *Tsprememba;
    // Izračun toplotnih izgub skozi objekt
    LdLs = Upovprecni*SSkupajZanaprej;
    // Izračun toplotnega toka skozi objekt podnevi
    Ppodnevi = Upovprecni*SSkupajZanaprej*TspremembaDan;
    // Izračun toplotnega toka skozi objekt ponoči
    Pponoci = Upovprecni*SSkupajZanaprej*TspremembaNoc;
    //Na koliko decimalnih mest naj bo rezultat
```

```
            rezultat = df.format(P);
            //Rezultati se izpišejo v gradniku TextView
            result.setText("P = "+ rezultat + " W");
            result1.setText(" " + LdLs + " W/K");
            result2.setText(" " + Ppodnevi + " W");
            result3.setText(" " + Pponoci + " W");
    }
```



Slika 36: Videz končnega izračuna toplotne prehodnosti in toplotnega toka v aplikaciji

4.8 Povzetek izdelane in uporabljene kode

Z uporabljeno kodo v poglavju 4 je bila mogoča izdelava aplikacije, ki smo si jo zamislili za diplomsko delo. Povzetek, kaj smo z uporabljeno kodo ustvarili, je predstavljen nadaljevanju.

Z izdelano aplikacijo lahko uporabnik pošlje štiri večje sklope podatkov strežniku: lokacijo, splošen opis stavbe, režim ogrevanja v stavbi ter konstrukcijske sklope.

1. Lokacija: Aplikacija sama ugotovi uporabnikovo trenutno lokacijo. Ta se izpiše in pojavi na interaktivnem zemljevidu. V kolikor želi obravnavati želeni objekt, se ga na zemljevidu dotakne. Če želeni objekt ni v njegovi bližji okolici, ga lahko poišče z vnosom naslova v iskalnik nad zemljevidom. Ko je lokacija potrjena, sledi nova aktivnost s štirimi zavihki.

2. Splošen opis stavbe: Pod prvi zavihek vnese osnovne informacije o stavbi. Te vključujejo naslov (samodejni vnos), tip stavbe, leto izgradnje, višino stavbe, uporabno površino, neto tlorisno površino ter površino zemljišča pod stavbo.

3. Režim ogrevanja stavbe: Drugi zavihek vsebuje podatke o tem, na koliko stopinj ogrevamo stavbo podnevi, na koliko ponoči ter koliko sta dnevni zunanja in nočna temperatura.

4. Konstrukcijski sklopi: V tretji zavihek se vnašajo konstrukcijski sklopi z vsebovanimi materiali. Najprej uporabnik določi vsak sklop posebej, njegovo vrsto (tla, stena ali streha) ter površino, ki jo sklop zaseda. Sledi vnos materialov za vsak sklop. Ti so podani na oddaljenem strežniku. V kolikor je baza posodobljena, se seznam samodejno posodobi. V bazi materialov se skupaj z vsakim materialom nahajajo njihove karakteristike: gostota - ρ, specifična toplota – *c*, toplotna prevodnost - λ, difuzijska upornost vodni pari - μ ter dodatni opis. Skupaj z izbranim materialom se v aplikacijo prenesejo tudi te vrednosti. Za vsak material uporabnik vnese še debelino sloja ter njegovo pozicijo v sklopu. Tako so za celoten sloj stavbe vneseni vsi sklopi z nanizanimi materiali.

Zadnji zavihek je namenjen pošiljanju podatkov v oddaljeno bazo, shranjevanju vnesenih vnosov ter odpiranju že obstoječih. V ta zavihek sta vnesena tudi informativni izračuni toplotnih izgub skozi objekt in toplotni tok skozi objekt. Ti podatki niso vezani na tipizacijo objektov. Gre za približen izračun na podlagi osnovne enačbe za prehod toplote skozi snov. Pri tem ne upošteva odprtin (oken) in njenih sončnih dobitkov, izgub s konvekcijo, npr. prezračevanja, netesnosti odprtin (okna, vrata), vplivov vetra, toplotnih mostov itd. Prav tako ni mogoč vnos ogrevalnih sistemov. Gre za zasnovo izračunov, ki bi z nadaljnjim razvojem aplikacije, opravljali enakovredno nalogo kot jih opravljajo sedanji programi za osebne računalnike za namen izračunov energetskih izkaznic.

5 ZAKLJUČEK

Namen diplomske naloge je bil izdelava aplikacije za pametno mobilno napravo, ki bi zbirala podatke o določenem objektu in omogočala shranjevanje le-teh na centralni strežnik. Na strežniku bi se zbirale obsežne zbirke podatkov, s katerimi bi bile mogoče nadaljnje raziskave stavbnega fonda. Mobilni operacijski sistem, za katerega je bila aplikacija izdelana, je bil Googlov Android. Izbrali smo ga zaradi velike podpore pri učenju programskega jezika in zaradi njegove velike popularnosti ter obsežne uporabe.

5.1 Uporabnost aplikacije

Kot navaja evropsko poročilo Finančne podpore za energetsko učinkovitost 2013 v stavbah [10], so ena od ovir za hitrejše urejanje sprememb in finančnih spodbud na področju energetsko učinkovitih stavb velike razlike v stavbnem fondu članic . Primanjkujejo tudi podatki o naknadnih prenovah, globinah posegov in posodobitvah v preteklosti. Če bi podatke o stanju stavb zbirali bolj sistematično, bi z večjo preglednostjo lažje usmerjali finančne spodbude za njihovo prenovo.

S tipizacijo stavb (npr. EU projekta Tabula) ne bi potrebovali tolikšnega števila strokovnjakov za analizo stavb. Vsak uporabnik ali lastnik stanovanja bi za svoj objekt vnesel osnovne podatke in jih s pošiljanjem v bazo po našem postopku nazaj prejel v obliki analize stavbe in možnih ukrepov za izboljšave. Podatki bi se ohranili v bazi na strežniku in tako služili nadaljnjim analizam. Z množično uporabo pametnih telefonov in njihovo priročnostjo bi lastnike in upravljavce stanovanj lažje prepričali, da bi za svoje stanovanje izpolnili potrebne podatke in se informirali o trenutnem energetskem stanju objekta ter možnostih za njegovo izboljšavo.

5.2 Možnosti za izboljšavo aplikacije

Vsi izračuni so narejeni na podlagi enostavnih formul, vendar bi jih v prihodnje lahko izboljšali z upoštevanjem katere od novejših metodologij. Najpomembnejši vnosi, ki bi jih morali dodati pri vključevanju v račun energetske izgube so sončni dobitki (predvsem za transparentne površine), izmenjava zraka ter toplotni mostovi.

V aplikaciji kot posredniku informacij, smo imeli največ težav pri pošiljanju paketa na strežnik. V primeru, ko je mobilna naprava povezana z Wi-Fijem, se paket brez težav pošlje do servlet aplikacije in naprej v bazo MySQL. Pri mobilni podatkovni povezavi z internetom pa paket ne prispe do baze. Pri testiranju aplikacije na novejših platformah Androida smo ugotovili, da aplikacija javi napako pri pošiljanju na strežnik. To se zgodi tudi, ko želimo v konstrukcijski sklop dodati materiale s svojimi lastnostmi. Tu poskuša aplikacija prejeti informacije iz XML datoteke na oddaljenem strežniku. Tudi sicer se je način prejemanja informacij iz oddaljenega strežnika slabo izkazal, saj v primeru, ko imamo opraviti z obsežno bazo materialov, traja samo razvrščanje materialov v gradnik ExpandableList predolgo. Bolje bi bilo vnaprejšnje kopiranje datoteke v napravo. Datoteko bi s strežnika ponovno prekopirali šele, ko bi obstajala novejša baza materialov.

5.3 Predviden nadaljnji razvoj

Pri nadaljnjem razvoju bi se morali predvsem odločiti o smeri razvoja aplikacije. Svoj namen aplikacija dosega v projektu tipizacije stavb. Pametna naprava služi le kot posrednik informacij na strežnik, od koder spletna aplikacija, v povezavi z EU projektom Tabula, prevzema podatke o stavbi, opravlja tipizacijo in jih uvršča v razrede.

Pri računanju energetskih izkaznic pa se mobilna naprava zaradi majhnega zaslona in tipkovnice izkaže kot neprimerna. Zaradi nujnega velikega števila vnosov različnih podatkov je uporaba mobilne naprave vprašljiva, saj je ročno vnašanje podatkov v pametne naprave dolgotrajnejše v primerjavi s tipkovnico na osebnih računalnikih. Nadaljnje razvijanje aplikacije bi bilo smiselno v povezavi z osebnim računalnikom.

Pametna naprava se zaradi svoje mobilnosti izkaže za primerno predvsem za delo na terenu. Zaradi senzorjev, ki jih te naprave ponujajo, bi lahko še olajšale postopke pri vnašanju podatkov. Ena od možnosti nadaljnje uporabe je merjenje razdalj za računanje površin stavbe pri njeni analizi. Z uporabo kamere ali senzorja pospeška je mogoče izmeriti želene razdalje, kote in podobno. V te namene že obstaja kopica narejenih in objavljenih aplikacij, njihovo tehnologijo pa bi bilo mogoče vgraditi tudi v našo aplikacijo.

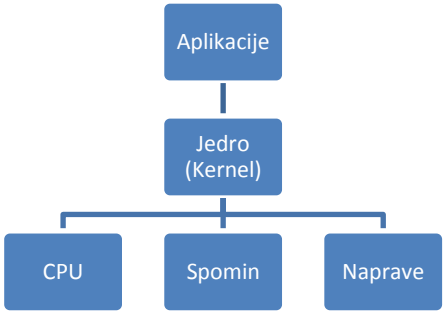
V nadaljnji razvoj aplikacije bi lahko še vključili:

- prikaz porabe za vsako stavbo na določenem radiju ter njeno projiciranje na vgrajenem zemljevidu pod pogojem, da bi bil dostop do baz informacij o energetskih porabah stavb in pridobivanje teh odprt,
- samodejno predlaganje različnih izvedb izolacij ter
- z uporabo googlovga vmesnika in zaznavo naše lokacije, bi aplikacija lahko ponudila seznam imen vseh najbližjih izvajalcev toplotne sanacije objektov v okolici.

Ker je bil cilj delati na uporabnosti in enostavni uporabi aplikacije se samemu videzu nismo posvečali. Ker je tako kot pri spletnih straneh, ko velikokrat pritegne predvsem njen videz, bi veljalo pri naši aplikaciji nadaljevati tudi na tem področju.

TERMINOLOŠKI SLOVAR

Jedro (kernel) – glavna komponenta računalniškega operacijskega sistema, most med programsko in strojno opremo (Wikipedia, Kernel (computing)). Povezava med jedrom, strojno opremo in programsko opremo je prikazano na sliki 37.



Slika 37: Jedro, most med strojno in programsko opremo

Middleware – programska oprema, ki omogoča storitve aplikacijam poleg tistih, ki so mogoče iz opreacijskega sistema (Wikipedia, Middleware).

Knjižnica – zbirka implementacij vedenja, napisanih v programskem jeziku, ki ima dobro opredeljen vmesnik, s katerim se to obnašanje uveljavlja (Wikipedia, Library (computing)).

Delovni okvir aplikacije (application framework) – sestavljen je iz okvira programske opreme, ki jih razvijalci programske opreme uporabljajo za implementacijo standardne strukture določene aplikacije (Wikipedia, Application Framework).

Apache Harmony – je bila odprtokodna, zastonjska implementacija Jave, ki jih je Apache Software Foundation razvila (Wikipedia, Apache Harmony).

ARM arhitektura – družina računalniških procesorjev (Wikipedia, ARM architecture).

Dalvik virtual machine – procesni navidezni stroj (VM) na operacijskem sistemu Google Android. Programska oprema, na kateri tečejo aplikacije za naprave Android (Wikipedia, Dalvik Virtual Machine).

Just-in-time (JIT) – je metoda za izboljšanje runtime delovanja računalniških programov, ki temeljijo na byte kodi (virtualna strojna koda) (Wikipedia, Just-in-time compilation).

String – zaporedje znakov, pogosto se uporabi kot niz bajtov (ali besed) (Wikipedia, String (computing)).

X Window Sistem – sistem za računalniško programsko opremo in omrežni protokol, ki predstavlja podlago za grafični uporabniški vmesnik (GUI) in bogato vhodno zmogljivost za povezane računalnike (Wikipedia, X Window System).

Shim – majhna knjižnica, ki transparentno prestreže aplikacijski programerski vmesnik (API) in spremeni želene parametre, obravnava samo operacijo ali preusmeri delovanje drugam (Wikipedia, Shim (computing)).

API – aplikacijsko programski vmesnik; določa kako določene programske komponente komunicirajo med seboj (Wikipedia, Aplication programming interface).

JNI – Java Native Interface je programski okvir, ki omogoča Java kodi, ki teče v Java Virtualnem Stroju (JVM) da kliče in je klicana od aplikacij (specifični programi za strojno opremo in platformo operacijskega sistema) in knjižnic napisanih v drugih jezikih kot so C, C + + (Wikipedia, Java Native Interface).

Extensible Markup Language (XML) - je označevalni jezik, ki določa vrsto pravil za kodiranje dokumentov v formatu, ki je človeško in strojno berljiv. Popularnost XML-ja odlikuje preprostost, splošnosti in uporabnost. Gre za tekstovni format podatkov z močno podporo preko Unicoda za različne svetovne jezike (Wikipedia, XML).

RDBMS – kratica za relational database management system. Tip sistema za upravljanje baz podatkov, ki shranjuje podatke v obliki povezanih tabel (Wikipedia, Relational database management system).

HTTP – kratica za HyperText Transfer Protocol. Je glavna metoda za prenos informacij na spletu. Protokol je prvotno namenjen objavljanju in prejemanju HTML strani (Wikipedia, HTTP).

VIRI

[1] Mobile-Cellular Subscriptions. 2013. ICT Facts and Figures. Ženeva, International Telecommunication Union: str. 1.

http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2013.pdf (Pridobljeno 10. 9. 2013.)

[2] van der Meulen, R., Rivera, J. 2013. Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012. Gartner announcements (13. feb. 2013).

http://www.gartner.com/newsroom/id/2335616 (Pridobljeno 10. 9. 2013.)

[3] Stegnar, G. 2012. Uporaba tehnologije ontowiki pri tipizaciji stavb v Sloveniji. Diplomska naloga. Ljubljana, Univerza v Ljubljani, Fakulteta za gradbeništvo in geodezijo (samozaložba G. Stegnar): 136 str.

[4] Slika 3: Final energy consumption, EU-27, 2010. 2012. Consumption of energy. Luxembourg, Eurostat.

http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Consumption_of_energy (Pridobljeno 11. 9. 2013.)

[5] Graf 2: Household energy consumption in the EU- 27. 2007. Odyssee-Mure: str. 1.

http://www.odyssee-indicators.org/reports/household/households.pdf (Pridobljeno 11. 9. 2013.)

[6] Slika PG4-4: Delež porabe energije v gospodinjstvih v letu 2009. 2011. Poraba energije in goriv v gospodinjstvih. Ljubljana, Statistični urad RS.

http://kazalci.arso.gov.si/?data=indicator&ind_id=350 (Pridobljeno 12. 9. 2013.)

[7] Grafikon 1: Končna poraba energije po namenih, gospodinjstva, Slovenija, 2011. 2012. Poraba energije in goriv v gospodinjstvih. Ljubljana, Statistični urad RS.

http://www.stat.si/novica_prikazi.aspx?id=5027 (Pridobljeno 12. 9. 2013.)

[8] Age categorisation of housing stock in Europe. 2011. Europe's buildings under the microscope. Bruselj, European Climate Foundation: str. 9.

http://www.europeanclimate.org/documents/LR_%20CbC_study.pdf (Pridobljeno 12. 9. 2013.)

[9] Mobilising investment in energy efficiency: economic instruments for low-energy buildings (Insights paper). 2012. Paris, OECD/IEA: 156 str.

http://www.iea.org/publications/insights/Mobilising_investment_EE_FINAL.pdf (Pridobljeno 12. 9. 2013.)

[10] Financial support for energy efficiency in buildings. 2013. Bruselj, European Commision: 30 str.

http://ec.europa.eu/energy/efficiency/buildings/doc/swd_2013_143_accomp_report_financing_ee_buildings.pdf (Pridobljeno 12. 9. 2013.)

[11] List of mobile software distribution platforms. 2013.

http://en.wikipedia.org/wiki/List_of_mobile_software_distribution_platforms (Pridobljeno 12. 9. 2013.)

[12] Android (operating system). 2013.

https://en.wikipedia.org/wiki/Android_(operating_system) (Pridobljeno 12. 9. 2013.)

[13] Android version history. 2013.

http:***//***en.wikipedia.org/wiki/Android_version_history (Pridobljeno 12. 9. 2013.)

[14] Object-oriented programming. 2013.

http:***//***en.wikipedia.org/wiki/Object-oriented_programming (Pridobljeno 12. 9. 2013.)

[15] Building a Simple User Interface. 2013.

http://developer.android.com/training/basics/firstapp/building-ui.html (Pridobljeno 12. 9. 2013.)

[16] Starting an Activity. 2013.

http://developer.android.com/training/basics/activity-lifecycle/starting.html (Pridobljeno 12. 9. 2013.)

[17] Cvikl, B. 2002. Gradbena fizika. Maribor, Univerza v Mariboru, Fakulteta za gradbeništvo: 122 str.

http://rcum.uni-mb.si/~pinteric/data/gf-knjiga.pdf (Pridobljeno 12. 9. 2013.)

[18] Toplotni tok. 2013.
http://sl.wikipedia.org/wiki/Toplotni_tok (Pridobljeno 12. 9. 2013.)

[19] Arčon, D. 2010. Gradbena fizika. Ljubljana, Univerza v Ljubljani, Fakulteta za matematiko in fiziko: 27 str.
http://www-f5.ijs.si/catalog/datoteke/lekcija3-20100331115020.pdf (Pridobljeno 12. 9. 2013.)

9.1 Ostali viri

Kernel (computing). 2013.
http://en.wikipedia.org/wiki/Kernel_(computing) (Pridobljeno 12. 9. 2013.)

Middleware. 2013.
http://en.wikipedia.org/wiki/Middleware (Pridobljeno 10. 9. 2013.)

Library (computing). 2013.
http://en.wikipedia.org/wiki/Library_(computing)
Application framework. 2013.
http://en.wikipedia.org/wiki/Application_framework (Pridobljeno 10. 9. 2013.)

Apache Harmony. 2013.
http://en.wikipedia.org/wiki/Apache_Harmony (Pridobljeno 10. 9. 2013.)

ARM architecture. 2013.
http://en.wikipedia.org/wiki/ARM_architecture (Pridobljeno 10. 9. 2013.)

Dalvik Virtual Machine. 2013.
http://en.wikipedia.org/wiki/Dalvik_virtual_machine (Pridobljeno 10. 9. 2013.)

Just-in-time compilation. 2013.
http://en.wikipedia.org/wiki/Just-in-time_compilation (Pridobljeno 10. 9. 2013.)

String (computer science). 2013.
http://en.wikipedia.org/wiki/String_(computer_science) (Pridobljeno 10. 9. 2013.)

X Window System. 2013.
http://en.wikipedia.org/wiki/X_Window_System (Pridobljeno 10. 9. 2013.)

Shim (computing). 2013.
http://en.wikipedia.org/wiki/Shim_(computing) (Pridobljeno 10. 9. 2013.)

Application programming interface. 2013.
http://en.wikipedia.org/wiki/API (Pridobljeno 10. 9. 2013.)

Java Native Interface. 2013.
http://en.wikipedia.org/wiki/JNI (Pridobljeno 10. 9. 2013.)

MySQL. 2013.
http://en.wikipedia.org/wiki/MySQL (Pridobljeno 10. 9. 2013.)

Java Servlet. 2013.
http://en.wikipedia.org/wiki/Java_servlet (Pridobljeno 10. 9. 2013.)

Sensors Overview. 2013.
http://developer.android.com/guide/topics/sensors/sensors_overview.html (Pridobljeno 12. 9. 2013.)

Introduction. 2013.
http://developer.android.com/tools/workflow/index.html (Pridobljeno 12. 9. 2013.)

Document Object Model. 2013.
http://en.wikipedia.org/wiki/Document_Object_Model (Pridobljeno 12. 9. 2013.)

XML. 2013.
http://en.wikipedia.org/wiki/Extensible_Markup_Language (Pridobljeno 12. 9. 2013.)

Relational database management system. 2013.

http://en.wikipedia.org/wiki/Relational_database_management_system (Pridobljeno 12. 9. 2013.)

HTTP. 2013.

http://sl.wikipedia.org/wiki/HTTP (Pridobljeno 12. 9. 2013.)

Felker, D, Dobbs, J. 2011. Android Application Development For Dummies. Indianapolis, Wiley Publishing, Inc.: 388 str.

Android Developers. 2013.

http://developer.android.com/training/index.html (Pridobljeno 12. 9. 2013.)

Stackoverflow. 2013.

http://stackoverflow.com (Pridobljeno 12. 9. 2013.)

Code On Cloud. 2013.

http://codeoncloud.blogspot.com (Pridobljeno 12. 9. 2013.)