Idejna razrada vođenja građevinskog dnevnika u elektronskom obliku na platformi Android

Dumanić, Daniela

Master's thesis / Diplomski rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:
University of Split, Faculty of Civil Engineering, Architecture and Geodesy / Sveučilište u Splitu, Fakultet građevinarstva, arhitekture i geodezije

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:123:867582

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

FCEAG Repository - Repository of the Faculty of Civil Engineering, Architecture and Geodesy, University of Split

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU
FAKULTET GRAĐEVINARSTVA ARHITEKTURE I GEODEZIJE

DIPLOMSKI RAD

Daniela Dumanić

Split, 2015.

SVEUČILIŠTE U SPLITU
FAKULTET GRAĐEVINARSTVA ARHITEKTURE I GEODEZIJE

Daniela Dumanić

Idejna razrada vođenja građevinskog dnevnika u elektronskom obliku na platformi Android

Diplomski rad

Split, 2015

SVEUČILIŠTE U SPLITU
FAKULTET GRAĐEVINARSTVA, ARHITEKTURE I GEODEZIJE

STUDIJ:	**DIPLOMSKI SVEUČILIŠNI STUDIJ GRAĐEVINARSTVA**
KANDIDAT:	Daniela Dumanić
BROJ INDEKSA:	466
KATEDRA:	**Katedra za organizaciju i ekonomiku građenja**
PREDMET:	Organizacija građenja

ZADATAK ZA DIPLOMSKI RAD

Tema:
Idejna razrada vođenja građevinskog dnevnika u elektronskom obliku na platformi Android

Opis zadatka:
Predmet diplomskog rada je razrada ideje vođenja građevinskog dnevnika na elektronskom uređaju na platformi Android kao buduće aplikacije.

U Splitu, 08.rujan 2015.

Voditelj Diplomskog rada:

Izv.prof.dr.sc. Nives Ostojić Škomrlj,dipl.ing.građ.

Predsjednik Povjerenstva
za završne i diplomske ispite:

Prof. dr. sc. Ivica Boko

Idejna razrada vođenja građevinskog dnevnika u elektronskom obliku na platformi Android

Sažetak:

Tema ovog rada je izrada ideje za buduću aplikaciju vođenja građevinskog dnevnika u elektronskom obliku. Prvo je izvedena sama teorija, kako teorija građevinskog dnevnika, tako i teorija izabrane programerske okoline Android Studio. Sama ideja je posebno razrađena, ispisana i predočena u ovom radu. Prikazano je sučelje idejne aplikacije, bitne značajke kao i neki važni kodovi koji su činili da se može ideja razvijati u pravom smislu.

Ključne riječi:

Građevinski dnevnik, Android Studio

Conceptual eraboration of a conctruction journal in electronic form on Android platform

Abstract:

The topic of this paper is the assembly of a future application for a construction journal in electronic form. Presented first is the theory, both the theory of the construction journal as well as the theory about the programming enviroment Android Studio. The idea itself is developed, written and presented in this paper. Showcased within are the interface, important features and key portions of code that ensured the idea could be developed.

Keywords:

Construction log, Android Studio

SADRŽAJ

1. UVOD ... **2**
2. ŠTO JE GRAĐEVINSKI DNEVNIK? ... **3**
2.1. Uvjeti i način vođenja građevinskog dnevnika ... 3
2.1.1. Podaci koji se unose u dnevnik ... 3
2.1.2. Pravila pisanja dnevnika u tekstualnom obliku ... 5
2.1.3. Građevinski dnevnik kao elektronički zapis ... 6
2.2. Potrebni pojmovi za razumijevanje pisanja dnevnika ... 7
3. OPERACIJSKI SUSTAV ANDROID ... **9**
3.1. Osnovne značajke Android OS-a ... 10
3.2. Struktura Android OS-a ... 10
3.3. Razvoj Android aplikacija ... 12
3.3.1. Android SDK ... 13
3.3.2. Komponente aplikacije ... 13
3.3.3. Životni ciklus aplikacije ... 15
3.3.4. Aktiviranje komponenata aplikacije ... 18
3.3.5. Android manifest dokument ... 19
3.3.6. Aplikacijski resursi ... 21
3.4. Java programski jezik ... 21
3.4.1. Struktura Java datoteke ... 22
4. IDEJNA RAZRADA VOĐENJA GRAĐEVINSKOG DNEVNIKA U ELEKTRONSKOM OBLIKU NA PLATFORMI ANDROID ... **23**
4.1. Alati korišteni pri izradi aplikacije ... 23
4.1.1. Razvojna okolina Android Studio ... 23
4.2. Opis grafičkog sučelja aplikacije ... 28
4.3. Programska izvedba aplikacije ... 43
4.3.1. Struktura podataka ... 43
4.3.2. Neke općenite metode korištenja Android Studia ... 47
4.3.3. Spremanje i čitanje podataka ... 49
4.3.4. Programska izvedba grafičkog prikaza podataka ... 55
5. ZAKLJUČAK ... **59**
LITERATURA ... **61**
SAŽETAK ... **62**

1. UVOD

Danas je život u potpunosti nezamisliv bez mobilnih uređaja, što je pogotovo došlo do izražaja nakon izuma pametnih telefona. Korisnici pametnih telefona i ostalih mobilnih uređaja dolaze iz gotovo svih dobnih skupina, što je još jedan čimbenik koji je utjecao na njihovu veliku rasprostranjenost. Karakterizira ga otvorena arhitektura sa specifičnim odnosom između aplikacija i prikaza na ekranu te mogućnost da se promijeni svaka uobičajena aplikacija koja dolazi s operacijskim sustavom, što govori dovoljno o fleksibilnosti koju pruža. Google Android je prvi operacijski sustav za mobilne uređaje otvorenog koda, što znači da je sav njegov kod u potpunosti dostupan programerima i smije se modificirati na bilo koji način bez potrebe za licenciranjem.

Tema ovog diplomskog rada je razrada ideje za vođenje građevinskog dnevnika na elektronskom uređaju. Daje se ideja za izradu aplikacije za pristup građevinskom dnevniku koji korisnik može imati na svom mobilnom uređaju ili tabletu i olakšati si svakodnevni upis podataka. Glavni cilj aplikacije je omogućiti korisniku vođenje dnevnika „na dohvat ruke“ kao i automatsko slanje podataka investitoru. Aplikacija ujedno pojednostavljuje unos podataka kao izbacivanje svakodnevnog upisa istih. Podaci koje aplikacija prikazuje su spremljeni u XML obliku. Za njihovo korištenje potrebno je organizirano spremanje.

U drugom poglavlju opisan je građevinski dnevnik kao takav. Iznose se uvjeti njegova vođenja, što se upisuje i na koji način, te se opisuju podaci koje je potrebno unijeti svaki dan tokom projekta. Iznesen je i link pravilnika po kojem se treba orijentirati u načinu pisanja.

Treće poglavlje opisuje karakteristike, osnovne značajke i strukturu Android operacijskog sustava. Na kraju poglavlja opisan je Java programski jezik.

Četvrto poglavlje opisuje razrađenu ideju aplikacije vođenja građevinskog dnevnika u elektronskom obliku. Programska okolina u kojoj je razvijena aplikacija je Android, a za pokretanje aplikacije koristio se mobilni uređaj sa pristupom opcijama za razvojne programere. Programska izvedba aplikacije podijeljena je u više dijelova. U prvom djelu opisana je struktura aplikacije od strane korisnika, a u drugom dijelu prikazana je organizacija aplikacije te klase koje su korištene za spremanje podataka. Na kraju je opisana izrada grafičkog dijela aplikacije. Uz opis aplikacije prikazani su i primjeri programskog koda.

2. ŠTO JE GRAĐEVINSKI DNEVNIK?

Građevinski dnevnik je dokument o tijeku gradnje kojim se dokazuje usklađenost uvjeta i načina gradnje odnosno izvođenja pojedinih radova s pretpostavkama i zahtjevima iz glavnog projekta, izvedbenog projekta, propisa i normi. O gradnji građevine, odnosno o izvođenju pojedinih radova za koje je potrebna građevna dozvola, izvođač je obvezan voditi građevinski dnevnik.

2.1. Uvjeti i način vođenja građevinskog dnevnika

Ako u gradnji sudjeluju dva ili više izvođača, građevinski dnevnik vodi izvođač odgovoran za međusobno usklađivanje radova. Građevinski dnevnik vodi se tijekom gradnje za cijelu građevinu od dana početka pripremnih radova do dana završetka gradnje. Građevinski dnevnik odnosno zasebni građevinski dnevnik (u daljnjem tekstu: dnevnik) vodi odgovorna osoba koja vodi gradnju odnosno pojedine radove (glavni inženjer gradilišta, inženjer gradilišta i voditelj gradilišta).

Ako u građenju sudjeluju dva ili više izvođača, a osim građevinskog dnevnika za cijelu građevinu vodi/e se i zasebni građevinski dnevnik/ci, onda građevinski dnevnik za cijelu građevinu vodi glavni inženjer gradilišta, a zasebni/e građevinski/e dnevnik/e vodi/e inženjer/i gradilišta odnosno voditelj/i gradilišta [5].

Prilikom građenja građevine na kojima se izvodi više vrsta radova ili radovi većeg opsega, a osim građevinskog dnevnika za cijelu građevinu vodi/e se i zasebni građevinski dnevnik/ci, u građevinski dnevnik za cijelu građevinu upisuje glavni nadzorni inženjer, a u zasebni/e građevinski/e dnevnik/e upisuje/u nadzorni inženjer/i za određenu vrstu radova [11].

2.1.1. Podaci koji se unose u dnevnik

U građevinski dnevnik se svakodnevno upisuju podaci o [5].:

- tijeku i načinu izvođenja radova kao i svi oni koji se u slijedećim fazama neće moći pregledati (temeljne jame i podloge prije nastavka radova, oplata i armatura prije betoniranja, posteljica prije izgradnje gornjeg sloja, zidani elementi prije žbukanja, instalacije prije zatvaranja izolacije, prije zatrpavanja i sl.)

- uzimanju uzorka materijala za ispitivanje

- ispitivanjima na gradilištu
- rezultatima ispitivanja i atestiranja
- eventualnim prirodnim događajima i udesima
- prispijeću, porijeklu i kvaliteti materijala i opreme koji se isporučuju na gradilište
- visinskim točkama, iskolčenju i sl.
- ispitivanju terena, pregledu gradilišta po inspekcijskim organima i njihovim nalazima
- kao i o drugim radovima i događajima od utjecaja na sigurnost i kvalitetu radova.

Nadalje, upisuju se svi oni podaci koji mogu služiti kao dokaz kod obračuna izvedenih radova, kao što su podaci o [5]:

- izmjenama i dopunama projekta
- zastojima i prekidima radova
- radovima koji se obračunavaju u režijskim satima
- nepredviđenim i naknadnim radovima
- podzemnim vodama
- izmjenama uvjeta rada
- broju uposlenih i njihovoj kvalifikacijskoj strukturi
- mehanizaciji na gradilištu
- kotama iskopa
- kategorizaciji zemljišta
- izmijenjenim uvjetima rada i dr.

O zasebnom građevinskom dnevniku obvezno se u građevinski dnevnik za cijelu građevinu upisuju sljedeći podaci:

- naziv odnosno opis tehničko-tehnološkog dijela građevine odnosno radova
- ime i adresa odnosno naziv i sjedište izvođača, ime odgovorne osobe koja vodi građenje, te broj suglasnosti (ako je za izvođenje predmetnih radova ista potrebna)
- ime i adresa odnosno naziv i sjedište osobe koja obavlja stručni nadzor te ime nadzornog inženjera odnosno imena glavnog i drugih nadzornih inženjera
- nadnevak početka i završetka vođenja zasebnog građevinskog dnevnika.

Glavni nadzorni odnosno nadzorni inženjer upisom u dnevnik:

- utvrđuje usklađenost iskolčenja s geodetskim projektom i odobrava početak izvođenja radova na građevini
- ocjenjuje usklađenost pregledanih izvedenih radova sa zahtjevima odnosno pretpostavkama iz glavnog projekta, izvedbenog projekta i tehničkih propisa
- odobrava odnosno zabranjuje nastavak radova te odobrava, odnosno određuje način otklanjanja utvrđenih nepravilnosti.

2.1.2. Pravila pisanja dnevnika u tekstualnom obliku

Odgovorna osoba koja vodi gradnju i glavni nadzorni inženjer odnosno nadzorni inženjer, potpisom na svakoj stranici ovjeravaju točnost upisa u dnevnik. Građevinski dnevnik vodi se u obliku uvezane knjige sa dvostruko obilježenim stranicama (original i kopija) čija se kopija može iz knjige trgati. Nadzori inženjer pohranjuje kod investitora kopiju potpisanih stranica, dok drugi primjerak ostaje kod izvođača. Podaci upisani u građevinski dnevnik ne smiju se ispravljati, mijenjati niti nadopunjavati. Eventualne ispravke, dopune ili promjene unose se u dnevnik novim upisom [11].

2.1.3. Građevinski dnevnik kao elektronički zapis

Građevinski dnevnik se može voditi i kao elektronički zapis. Građevinski dnevnik vođen kao elektronički zapis mora činiti jednoznačno povezan cjelovit skup podataka koji su elektronički oblikovani. Dnevnik se potpisuje naprednim elektroničkim potpisom.

Na nosaču podataka koji sadrži građevinski dnevnik izrađen kao elektronički zapis, osim dijelova građevinskog dnevnika, zasebno moraju biti pohranjeni:

1. sadržaj dnevnika
2. sadržaj dijelova građevinskih dnevnika koji čine cjeloviti građevinski dnevnik
3. podaci o gospodarskom subjektu koji provodi stručni nadzor.

Nosač podataka na kojem je pohranjen elektronički zapis građevinskog dnevnika mora na sebi imati podatke o:

1. nazivu građevine ili njezinog dijela
2. datumu izrade nosača podataka
3. rednom broju nosača podataka i ukupnom broju nosača podataka na kojima je pohranjen elektronički zapis građevinskog dnevnika (npr. 2/3)
4. sadržaju dijela građevinskog dnevnika pohranjenog na nosaču podataka.

Na nosaču podataka, osim sadržaja, ne smiju se nalaziti drugi podaci osim podataka o proizvođaču nosača podataka i tehničkih podataka o kapacitetu nosača podataka. Na nosaču podataka na kojem su pohranjeni elektronički zapisi građevinskih dnevnika koji se koriste za potrebe upravnih postupaka dodavanje novih sadržaja mora biti trajno onemogućeno. U slučaju izmjene upisa u građevinski dnevnik elektronički zapis građevinskog dnevnika pohranjuje se u izmijenjenom stanju na novi nosač podataka.

Pravila i način vođenja dnevnika izneseni su u prethodnom tekstu, a temelje se na Zakonu o gradnji koji se može naći na internetskoj stranici http://www.hkis.hr/Upload/Documents/Vijesti/2014-3-7_Pravilnik_nadzor_dnevnik.pdf u pdf obliku, a članci koji se odnose na izvođenje dnevnika, kako u tekstualnom, tako i u elektroničkom obliku, su članak 13.-članak 25. ovoga Zakona.

U tom pravilniku je prikazan i obrazac ispunjavanja građevinskog dnevnika u elektronskom obliku pri korištenju nosača podataka. Na taj zakon se treba obratiti pažnja pri konačnoj izradi aplikacije gdje bi korisnici mogli spremati podatke na nosaču podataka. Za sada je predviđeno korištenje građevinskog dnevnika kao aplikacije te moguće spremanje svih podataka dnevnika u nekom tekstualnom obliku [11].

2.2. Potrebni pojmovi za razumijevanje pisanja dnevnika

Sudionici u pisanju građevinskog dnevnika su izvođač, odgovorna osoba koja vodi građenje (glavni inženjer gradilišta, inženjer gradilišta, voditelj radova), nadzorni inženjer i/ili glavni nadzorni inženjer.

Izvođač je osoba koja gradi ili izvodi pojedine radove na građevini. Izvođač imenuje inženjera gradilišta, odnosno voditelja radova u svojstvu odgovorne osobe koja vodi građenje, odnosno koja vodi pojedine radove. Treba povjeriti izvođenje građevinskih radova i drugih poslova osobama koje ispunjavaju propisane uvjete za izvođenje tih radova, odnosno obavljanje poslova te sastaviti pisanu izjavu o izvedenim radovima i o uvjetima održavanja građevine.

Nadzorni inženjer provodi u ime investitora stručni nadzor građenja. U slučaju izvođenja više vrsta radova ili radova većeg opsega na određenoj građevini, nadzorni inženjer mora angažirati više nadzornih inženjera ili nadzorne inženjere odgovarajuće struke za te radove. Glavni nadzorni inženjer, kojeg imenuje investitor, odgovoran je za cjelovitost i međusobnu usklađenost stručnog nadzora građenja i dužan je o tome sastaviti završno izvješće. Nadzorni inženjer određuje način na koji se otklanjaju nedostaci odnosno nepravilnosti građenja građevine; način otklanjanja nedostataka, odnosno nepravilnosti upisuje se u građevinski dnevnik. Dužan je odrediti provedbu kontrolnih postupaka u pogledu ocjenjivanja sukladnosti, odnosno dokazivanja kvalitete određenih dijelova građevine putem ovlaštene osobe u slučajevima kada je zakonima, propisima ili projektom određena takva obveza. Nadzorni inženjer ne može biti zaposlenik osobe koja je izvođač na istoj građevini.

Investitor je pravna ili fizička osoba u čije ime se gradi objekt. On je i najodgovornija osoba u postupku građenja. Projektiranje, kontrolu i nostrifikaciju projekata, građenje i stručni nadzor građenja mora povjeriti osobama registriranima za obavljanje tih djelatnosti. Investitor osigurava stručni nadzor građenja građevine. Prije početka građenja treba ishoditi akt kojim se odobrava građenje i prijaviti početak građenja (elaborat iskolčenja + (ne)potvrđeni glavni projekt). Treba prijaviti početak gradnje tijelu graditeljstva i građevnoj inspekciji najkasnije u roku od 8 dana prije početka radova ili nastavka radova nakon prekida dužeg od tri mjeseca [10].

Građevinska dozvola je dokument (upravni akt - rješenje) na temelju kojega se može započeti gradnja građevine. Njime se utvrđuje da je glavni, odnosno idejni projekt izrađen u skladu s propisima i utvrđenim uvjetima koje mora ispunjavati građevina na određenoj lokaciji te da su ispunjeni svi potrebni preduvjeti za gradnju [7]. Građevinske dozvole izdaju veliki gradovi (iznad 35.000 stanovnika) i gradovi sjedišta županija za svoje područje ili županijski uredi za sve ostale općine i gradove na području županije. Popis nadležnih ureda pronaći ćete na stranicama Ministarstva graditeljstva i prostornoga uređenja. Zahtjev za izdavanje građevinske dozvole podnosi investitor, a potrebno ga je uputiti nadležnom uredu za graditeljstvo i prostorno uređenje u mjestu gdje se planira gradnja odnosno rekonstrukcija građevine. Sustav e-dozvole omogućava da se građevinska dozvola izda i elektroničkim putem. On se primjenjuje na području cijele Republike Hrvatske, a dinamika izdavanja dozvola ovisi o organiziranosti županijskih i gradskih nadležnih tijela [12]. Građevinska dozvola nosi klasifikacijsku oznaku i urudžbeni broj. Klasifikacijska oznaka označava predmet prema sadržaju, godini nastanka, obliku i rednom broju predmeta. Urudžbeni broj označava stvaratelja pismena, godinu nastanka i redni broj pismena unutar predmeta [8].

Akt je pismeno kojim tijelo odlučuje o predmetu postupka, odgovara na podnesak stranke, određuje, prekida ili završava neku službenu radnju te obavlja službeno dopisivanje s drugim tijelima odnosno pravnim osobama koje imaju javne ovlasti [8].

3. OPERACIJSKI SUSTAV ANDROID

Android OS je mobilni operativni sustav otvorenog koda (engl. *open-source*). *Opensource* je termin za softver koji se može besplatno i slobodno koristiti te čiji je programski kod javno dostupan pod licencom koja osigurava zaštitu autorskih prava. Android je nastao 2003. g. u kalifornijskom gradu Palo Alto od strane četvorice samostalnih programera: Andy Rubin, Rich Miner, Nick Sears i Chris White. Temeljna ideja je bila napraviti pametne mobilne uređaje s naglaskom na lokaciju korisnika. U kolovozu 2005. g. Android je postao podružnica tvrtke *Google Inc*, na čijem čelu su ostali njegovi utemeljitelji. S ovim potezom *Google* je uspješno proširio svoje djelovanje na tržište *smartphone*-a. Android-om upravlja *Open handset Alliance*, grupa osamdesetak tehnoloških kompanija među kojima su *Google, HTC, T-mobile* i drugi. Cilj ovog konzorcija je ubrzati inovacije na području mobilnih operativnih sustava s naglaskom na javnu dostupnost koda. Javnom dostupnošću koda želi potrošačima ponuditi bogatije, jeftinije i bolje iskustvo.

Android market je aplikacija koja je automatski uključena u Android uređajima, a omogućava korisnicima pregledavanje i preuzimanje aplikacija koje objavljuju neovisni programeri. Android je od svojeg izvornog izdanja doživio niz ažuriranja koja se odnose na ispravak bug-ova prethodnih verzija *OS*-a te dodavanje novih mogućnosti. Spremanje podataka se radi u S*QLite bazi*, koja je uključena u instalaciji Android *OS*-a. Takvo spremanje je korisno za aplikacije koje zauzimaju puno memorije i omogućava efikasniji rad s uređajem. Android podržava multitasking aplikacija. Multitasking znači da se više aplikacija može izvršavati u isto vrijeme ili jedna aplikacija može istodobno obavljati više funkcija.

Aplikacije za Android se uglavnom programiraju u *Javi*. Za izvršavanje koda aplikacije mora postojati *Javin* virtualni operativni sustav (*JVM*). *Virtual Machine* označava kompletno izolirani operativni sustav koji radi unutar normalnog operativnog sustava. Unutar Android platforme ne postoji *JVM*, već *Dalvik* virtualna mašina (*DVM*) koja je specijalizirana za Android, optimizirana za očuvanje baterije uređaja i korištenje ograničene radne memorije. Prije izvršavanja Java koda, aplikacije se konvertiraju u *Dalvik*-ov izvršni format s ekstenzijom .*dex*. Takav format je prilagođen ograničenim mogućnostima mobilnih uređaja.

3.1. Osnovne značajke Android OS-a

- otvorenost – programeru omogućava potpunu slobodu u razvoju novih i već postojećih aplikacija, a proizvođaču uređaja slobodno korištenje i prilagodbu platforme bez plaćanja autorskih prava;

- sve aplikacije su ravnopravne – što znači da ne postoji razlika između osnovnih jezgrenih aplikacija uređaja i dodatnih aplikacija.;

- automatsko upravljanje životnim ciklusom aplikacije – omogućava nadzor pokretanja i izvršavanja aplikacija na sistemskoj razini optimiziranim korištenjem memorije i procesorske snage. Krajnji korisnik više ne brine o gašenju određenih aplikacija prije pokretanja drugih;

- brisanje granica između "klasičnih" aplikacija – mogućnost razvoja novih i inovativnih aplikacija temeljenih na međusobnoj suradnji različitih tehnologija;

- brz i jednostavan razvoj aplikacija – omogućen je bogatom bazom korisnih programskih biblioteka (engl. *libraries*) i alata za izradu aplikacija;

- visokokvalitetni grafički prikaz i zvuk – podržana 2D vektorska i 3D OpenGL (engl. *Open Graphics Library*) grafika, te ugrađeni kodeci svih često korištenih audio i video formata;

3.2. Struktura Android OS-a

Arhitektura Android operacijskog sustava bazira se na Linux 2.6 jezgri (engl. *kernel*) koja se koristi kao sloj virtualizacije sklopovlja (engl. *Hardware Abstraction Layer* - HAL). Razlog za korištenje jezgre operacijskog sustava Linux je dokazana pogonska podrška (engl. *driver model*), mogućnost upravljanja memorijom i procesima, provjeren sigurnosni model, mrežna povezivost, kao i robusnost, konstantni razvoj i unapređivanje tog operacijskog sustava [2].

Izvorne programske biblioteke (eng. *native libraries*) pisane su u programskim jezicima C i C++ i čine idući sloj u arhitekturi sustava. Neke od značajnijih su:

- program za upravljanje grafičkim sučeljem (eng. *Surface Manager*) – biblioteka odgovorna za pravilno iscrtavanje različitih aplikacijskih komponenti u vremenu i prostoru;
- OpenGL ES (eng. *OpenGL for Embedded Systems*) – biblioteke koje se koriste za hardversku 3D akceleraciju (ukoliko je podržana) ili za 3D rasterizaciju;
- SGL (eng. *Scalable Graphics Library*) - predstavlja 2D biblioteke na kojima je temeljena većina aplikacija. Spomenimo još da se 2D i 3D elementi mogu kombinirano prikazivati u jednom korisničkom sučelju;
- *Media Framework* – skupina kodeka za snimanje i reprodukciju audio formata, video formata i nepomičnih slika. Omogućena je od strane *PacketVidea*;
- *FreeType* – biblioteka koja služi za vektorsku rasterizaciju oblika pisma (eng. *font*);
- SSL (eng. *Secure Socket Layer*) - omogućuje sigurnosnu komunikaciju preko Interneta;
- SQLite – programska biblioteka koja implementira bazu podataka (eng. *database engine*);
- *WebKit* – jezgra preglednika koji podržava *JavaScript* i ostale standarde na malom uređaju;
- S*ystem C library* – implementacija standardne C-ove sistemske biblioteke (*libc*) izvedene iz operacijskog sustava BSD.

Sljedeći sloj u arhitekturi Androida je radno okruženje (eng. *Android runtime*) kojeg čine virtualni stroj Dalvik (DVM, eng. *Dalvik Virtual Machine*) i jezgrene biblioteke (eng. *core library*).

DVM je registarski baziran virtualni stroj, dok je klasični Javin virtualni stroj (JVM, eng. *Java Virtual Machine*) baziran na stogu.

Jezgrene biblioteke pisane su u programskom jeziku Java i predstavljaju sve esencijalne klase kao što su klase za manipulaciju kolekcijama, klase za komunikaciju s okolinom i slično. Bitna novost je i to što se Androidove jezgrene biblioteke razlikuju od biblioteka u Java Standard Edition (J2SE) i Java 2 Micro Edition (J2ME).

Sloj aplikacijskih okvira (eng. *Application Framework*) napisan je u programskom jeziku Java i sadrži proširiv skup programskih komponenti kojeg koriste sve aplikacije uređaja. Neki od važnijih elemenata su [9]:

- upravljanje aktivnostima (eng. *Activity Manager*) – upravljanje životnim ciklusom aplikacije,
- upravljanje programskim paketima (eng. *Package Manager*) – sadrži informaciju o aplikacijama instaliranim na sustavu,
- upravljanje prozorima (eng. *Window Manager*) – upravljanje aplikacijskim prozorima,
- upravljanje pozivima (eng. *Telephony Manager*) – sadrži API-je koji se koriste pri izradi aplikacija za upravljanje pozivima,
- pružatelji sadržaja (eng. *Content Providers*) – omogućuju zajedničko korištenje podataka od strane više aplikacija,
- upravljanje resursima (eng. *Resource Manager*) – služi za pohranu dijelova aplikacije koji nisu kod (npr. slike),
- sustav grafičkog prikaza (eng. *View System*) – sadrži bazu gotovih grafičkih prikaza i alata (eng. *widget*),
- upravljanje lokacijski temeljenim uslugama (eng. *Location Manager*) i
- upravljanje obavijestima (eng. *Notification Manager*) – upravljanje obavijestima i događajima (npr. dospijeće poruke, nadolazeći sastanak).

Aplikacijski sloj je zadnji sloj u arhitekturi sustava Android i čine ga korisničke aplikacije uređaja. Predstavlja sloj vidljiv krajnjem korisniku. Uključuje neke od standardnih sistemskih aplikacija kao što su Web preglednik, lista kontakata, telefon, itd.

3.3. Razvoj Android aplikacija

Android aplikacije su pisane u *Java* programskom jeziku [1]. Android *SDK* kompajlira programski kod u izvršni kod. Izvršni kod je arhivska datoteka sa ekstenzijom *.apk*. Takva datoteka je aplikacija koja se može instalirati na Android-ovom mobilnom uređaju. Pravilo je da se pri instalaciji svakoj aplikaciji dodjeljuje jedinstveni Linux identifikator. Prema tom identifikatoru aplikaciji se daje pristup servisima, senzorima i drugim dijelovima uređaja i operativnog sustava. Android platforma primjenjuje princip najmanje privilegije, tj. aplikacija

ima samo pristup onim komponentama koje su joj potrebne za rad. Svaka se aplikacija pokreće u *DVM*-u i djeluje izolirano od drugih aplikacija.

3.3.1. Android SDK

Paket softverskih razvojnih alata Android SDK pruža podršku za razvoj, testiranje, pronalaženje i uklanjanje pogrešaka aplikacija [2]. Prvo izdanje SDK-a je izdano 2007. godine, a standardno uključuje sljedeće komponente [9]:

- android API-ja;
- razvojne alate – alati za prevođenje i otklanjanje pogrešaka. Najznačajniji je programski dodatak (engl. plugin) za Eclipse IDE naziva ADT (engl. Android Development Tools Plugin) koji omogućuje jednostavni pristup LogCatu, uređivaču datoteke AndroidManifest.xml, kontrolu dolaznih poziva i SMS-a i slično;
- emulator – služi za izvršavanje programa na računalu;
- DDMS (engl. Dalvik Debug Monitoring Service) – služi za kontrolu, nadzor pronalaženja i uklanjanja pogrešaka u aplikacijama;
- AAPT (engl. Android Asset Packaging Tool) – koristi se za stvaranje i distribuciju Androidovog programskog paketa u .apk formatu;
- ADB (engl. Android Debug Bridge) – klijentsko-poslužiteljska aplikacija za instaliranje i pokretanje datoteka .apk na emulatoru ili uređaju, te pristup naredbenom sučelju uređaja (CLI). Koristi se i za povezivanje standardnog programa za pronalaženje i uklanjanje pogrešaka (engl. debugger) s kôdom na emulatoru ili uređaju;
- detaljnu dokumentaciju;
- primjerke kôda – jednostavni primjerci koda za demonstraciju korištenja određenih API-ja, te mogućnosti koje pruža Android.

3.3.2. Komponente aplikacije

Aplikacijske komponente su temeljni sastavni dijelovi Android aplikacije. Svaka komponenta je samostalna cjelina, igra specifičnu ulogu u radu aplikacije i ima svoj način interakcije sistema sa aplikacijom. Postoje četiri različita tipa aplikacijskih komponenata, gdje svaka ima svoj ciklus trajanja koji određuje proces pokretanja i prekida aplikacije [9]:

- Aktivnosti (engl. *Activity*)
  Komponenta aktivnost predstavlja glavnu ulaznu točku korisnika u program. Aktivnost je najčešće korištena komponenta. Zadatak aktivnosti je prikaz korisničkog sučelja

programa i omogućavanje interakcije korisnika sa programom. Aplikacija se obično sastoji od više aktivnosti koje su međusobno povezane. Obično je jedna aktivnost označena kao početna aktivnost (engl. *Main Activity*) i ona se prezentira kod pokretanja aplikacije. Nakon aktiviranja početne aktivnosti pokreću se ostale funkcije programa po principu *„last in, first out"*. Po pokretanju nove aktivnosti, prekida se prethodna aktivnost i ponovno pokreće po prekidu trenutne aktivnosti. Ovim principom se štedi na radnoj memoriji mobilnog uređaja. Svaka aktivnost implementirana je kao klasa koja proširuje osnovnu klasu *Activity*. *Activity* klasa sadrži *onCreate(), onPause(), onRestart()* i druge metode koje određuju ciklus trajanja aktivnosti. Metoda *onCreate()* ključna je za izvršavanje klase. Unutar *onCreate()* metode se uređuje korisničko sučelje (*setContentView()*). Svaka aktivnost ima svoje korisničko sučelje koje se sastoji od objekata deriviranih iz *View* i *Viewgroup* klasa.

- Servisi (engl. *services*)

  Servisi predstavljaju aplikacijsku komponentu koja izvršava zadatke u pozadini programa tijekom duljeg vremenskog peroda. Servisi nemaju grafičko sučelje. Ostale aplikacijske komponente mogu sa servisima uzajamno djelovati i izvoditi *IPC* međuprocesnu komunikaciju. *IPC* komunikacija je potrebna u slučajevima kad postoji ovisnost procesa, kad neki proces želi predati neku informaciju drugim procesima, ili kad želimo provjeriti međusobno ometanje procesa. Na primjer, servisi mogu izvršavati mrežne transakcije ili slušanje glazbe iz pozadine dok korisnik radi sa drugom aplikacijom. Servisi mogu zauzeti dva oblika:

  - *Started*

    Servis počinje kad aplikacijska komponenta pozove servis metodom S*tartService()*. Tako pokrenut servis se izvršava i prekida svoj rad u pozadini neovisno o korisniku.

  - *Bound*

    Servis se veže za određenu komponentu metodom *bindService()***.** Takav servis ima ciklus trajanja jednak komponenti za koju je vezan.

Kako bi se kreirao servis potrebno je implementirati klasu *Service*, odnosno kreirati potklasu klase *Service* i deklarirati komponentu u *manifest* dokumentu.

- Dobavljači sadržaja (engl. *Content Providers*)

  Dobavljači sadržaja predstavljaju komponentu aplikacije kojoj je zadaća dobavljanje i spremanje sadržaja. Ova komponenta je ujedno i jedini način za izmjenjivanje podataka među aplikacijama. Kreiranje dobavljača sadržaja se radi tako da se odredi lokacija za

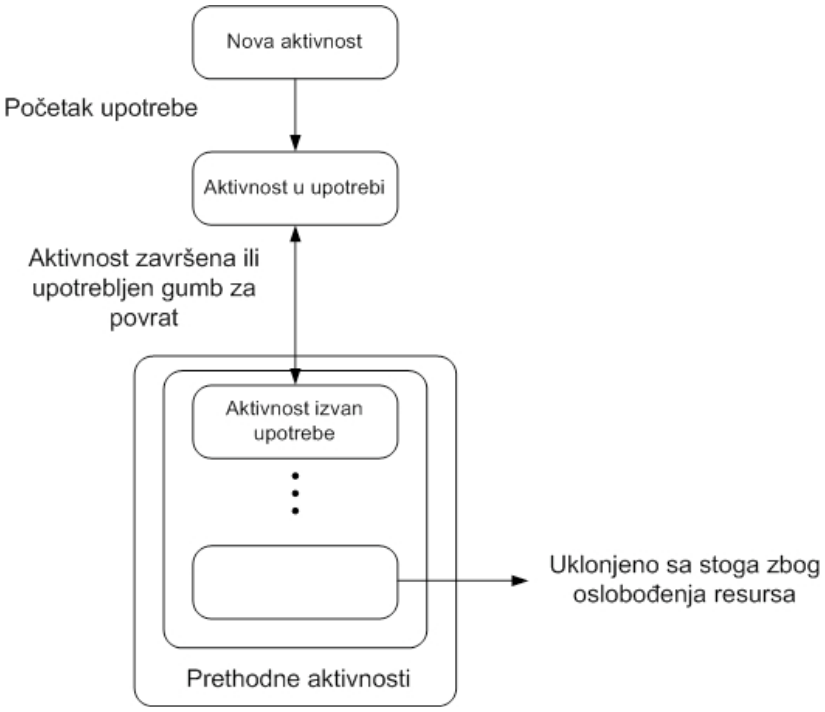
spremanje podataka. Dobavljač sadržaja daje sadržaj u formi tablice sa identifikatorom i ostalim atributima. Sadržaju se pristupa upitom pomoću razlučivača sadržaja (engl. *Content Resolver*). Upit sadrži adresu (*URI*) sadržaja, imena polja iz tablice i tip podataka – tekstualni tip (*String*) , cjelobrojni tip (*Integer*) ili decimalni tip (*Float*). Svi dobavljači sadržaja spremaju podatke u *SQLite* bazu podataka. Naravno, moguće je koristiti *PostgreSQL* bazu podataka ili neku drugi način za spremanja sadržaja. Svaki dobavljač sadržaja implementiran je kao klasa koja proširuje osnovnu klasu *ContentProvider*.

- Primatelji prijenosa (engl. *Broadcast Receiver*)
  Primatelji prijenosa imaju zadaću primanje i reagiranje na emitiranje obavijesti koje može dolaziti od strane sistema uređaja (npr. baterija je prazna, zaslon je uključen), ili od aplikacija (komunikacija s drugim aplikacijama). Ova komponenta nema korisničko sučelje, ali može pokrenuti aplikaciju kao rezultat primljene informacije. Također može koristiti klasu *NotificationManager* za upozorenje korisnika pomoću zvukova, vibracija ili svjetla.

3.3.3. Životni ciklus aplikacije

Kod klasične radne površine operacijskog sustava Windows ili Linux postoji jedan aktivni prozor i ravnopravni niz ostalih aplikacija, a kontrolu životnog ciklusa vrši korisnik [4]. Android sam brine o životnom ciklusu aplikacija, prikaz se vrši na principu LIFO (eg. *Last In First Out*) stoga na koji se spremaju aktivnosti pokrenutih aplikacija kao što je prikazano na slici 3.1.

Prilikom pokretanja aktivnosti, svaka aktivnost ima status *nove aktivnosti*. Prilikom upotrebe te aktivnosti, ona mijenja stanje u *aktivnost u upotrebi*. Kako je navedeno, u stog memoriju aktivnosti se upisuju po principu zadnji unutra, prvi vani. Aktivnost koje je zadnja došla u stog, prva se izvršava. Ukoliko je aktivnost završena ili je upotrjebljen gumb za povratak, aktivnost prelazi u stanje *aktivnost izvan upotrebe,* te se briše iz stog memorije. Ukoliko u stog memoriji nemamo dovoljno mjesta, aktivnost koja je prva upisana u stog, odnosno koja se zadnja izvršava briše se iz stog memorije zbog oslobađanja resursa.
Svaki zaslon korisničkog sučelja prikazan je pomoću klase *Activity*. Aplikaciju čine jedna ili više aktivnosti. Vrlo je važno napomenuti da aplikacija nije isto što i proces što bitno utječe na ponašanje i životni ciklus aplikacije.



Slika 3.1. Stog aktivnosti

Kako bi se očuvala efikasnost cijelog sustava, procesi i njihove aplikacije će, ukoliko je potrebno, biti ugašeni bez upozorenja s ciljem oslobađanja memorijskih resursa za aplikacije višeg prioriteta (uglavnom one koje korisnik koristi u tom trenu). Slika 3.2. prikazuje stablo prioriteta procesa.



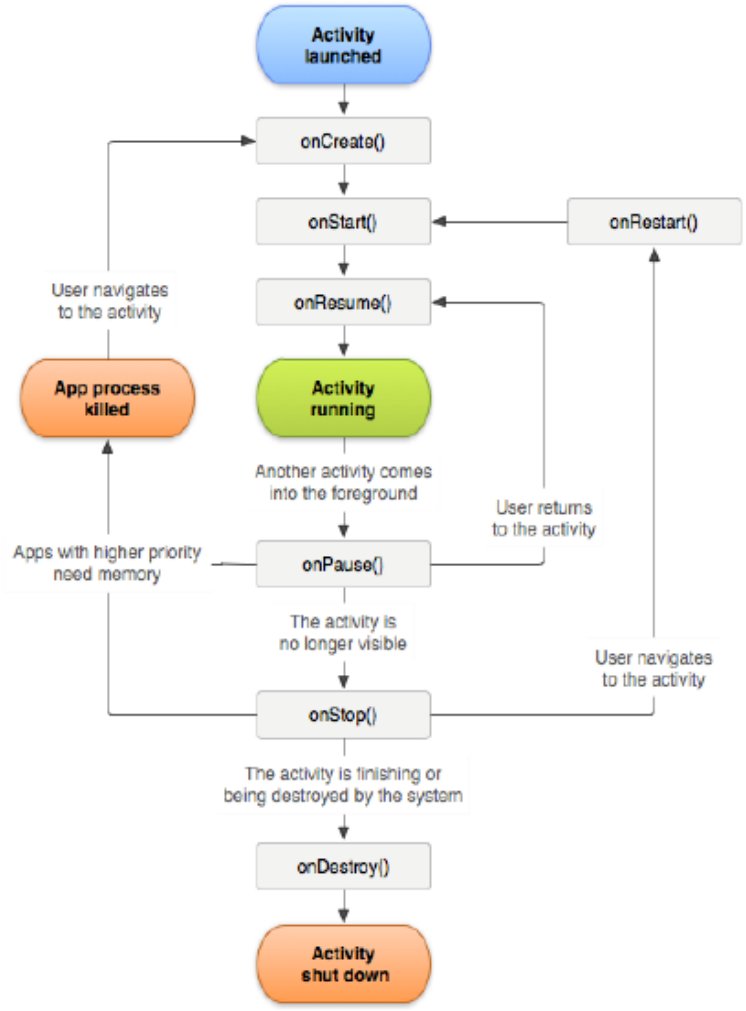
Slika 3.2. Stablo prioriteta procesa operacijskog sustava Android

Postoje slijedeće vrste procesa operacijskog sustava Android:

- aktivni procesi (engl. foreground process) – ovakvih procesa je uglavnom malo i imaju vrlo visok prioritet iz razloga što podržavaju aplikacije s kojima korisnik ostvaruje trenutačnu aktivnu interakciju. Proces se smatra aktivnim ako je riječ o aktivnosti koja se izvodi na zaslonu, ako se izvodi *BroadcastReciever* proces koji izvršavaju *onReceive()* metodu, te usluge koje izvršavaju kod nekog sistemskog poziva putem *onStart(), onCreate()* ili *onDestroy()* metode;
- vidljivi procesi (engl. visible process) – predstavljaju na zaslonu korisniku vidljive, ali neaktivne procese. Uglavnom ih je vrlo malo, smatraju se procesima visokog prioriteta te se gase samo u krajnjoj nuždi;
- pokrenuti uslužni procesi (engl. service processes) – procesi pokrenutih usluga s kojima korisnik nema aktivnu interakciju (npr. skidanje glazbe za reprodukciju na mp3 sviraču), te iz tog razloga imaju nešto niži prioritet. I dalje ih se smatra aktivnim procesima, te će biti ugašeni samo u nuždi;
- pozadinski procesi – predstavljaju procese nevidljivih aktivnosti bez pokrenutih usluga. Ovi procesi nemaju direktni utjecaj na doživljaj korisnika. U pravilu ih je veliki broj, te se gase po principu "zadnji viđen prvi ugašen";
- prazni procesi – predstavljaju zapis pokrenutih procesa u privremenoj memoriji s ciljem smanjenja vremena ponovnog pokretanja.

Na životni ciklus aplikacije utječe isključivo sustav, a ne sama aplikacija. Stanje u kojem se nalazi aplikacija određuje njezinu prioritetnu razinu. Aktivnost aplikacije može se nalaziti u nekom od sljedećih stanja prikazanih slikom 3.3.:

- aktivno – vidljiva aktivnost u fokusu;
- privremeno zaustavljeno – vidljiva aktivnost izvan fokusa;
- zaustavljeno – nevidljiva aktivnost izvan aktivne upotrebe. U memoriji se čuva njezino stanje i ostale informacije;
- neaktivno – stanje aktivnosti nakon gašenja, a prije pokretanja.



Slika 3.3. Stanja aktivnosti aplikacije

Prijelazi između pojedinih stanja nevidljivi su krajnjem korisniku, ali kontrolirani na razini sustava upotrebom odgovarajućih nadjačavajućih (engl. *override*) metoda.

3.3.4. Aktiviranje komponenata aplikacije

Komponente aktivnosti, servisi i primatelji prijenosa aktiviraju se preko asinkronih poruka koje se zovu namjere (engl. *intents*) [3]. Dobavljači sadržaja se aktiviraju zahtjevom ContentResolver-a. Namjere su objekti klase Intent, a sadrže informacije od interesa za komponentu koja prima asinkronu poruku. Za aktivaciju aktivnosti i servisa, namjera sadrži naziv komponente (npr. domena aplikacijskog paketa), ime tražene akcije (npr. editiranje podataka) i URI potrebnih podataka. Sadržaj namjere za primatelje prijenosa ime je tražene akcije. Prema sadržaju, namjere mogu biti eksplicitne ili implicitne.

Eksplicitne namjere sadrže ime komponente (npr. naziv *Java* klase koja se poziva). Implicitne namjere definiraju određenu radnju bez imenovanja klase koja je potrebna za tu radnju (npr. pregled web stranice).

3.3.5. Android manifest dokument

Prije pokretanja aplikacije, sustav mora znati od kojih se komponenata aplikacija sastoji. Registracija aplikacijskih komponenata se radi u *Androidovom manifest* dokumentu [6]. *Manifest* je strukturirana *XML* datoteka koja se zove *AndroidManifest.xml* za sve aplikacije. Funkcije ovog dokumenta su identificiranje aplikacijskog pristupa sistemu uređaja, definiranje minimalne verzije operativnog sustava na kojem će aplikacija raditi, deklariranje dijelova uređaja ili aplikacija koje program koristi, deklariranje eksternih biblioteka klasa s kojima je program povezan (Tablica 3.1.).

Tablica 3. 1. Općenita struktura Android manifesta

```
<? xml version = "1.0" encoding = "utf-8"? >
< manifest ... >
< uses-permission />
< uses-sdk />
< application android : icon = "@drawable/app_icon.png" ... >
< activity android:name = "com.example.project.ExampleActivity"
< intent-filter >
< action />
< category />
< data /> </ intent-filter >> </ activity >
< service >
< intent-filter > . . . </ intent-filter >
< meta-data /> </ service >
< receiver>
< intent-filter > . . . </ intent-filter >
< meta-data /> </ receiver >
< provider >
< grant-uri-permission />
< meta-data /> </provider >
< uses-library >… <uses-library /> </ application > </ manifest >
```

Neke konvencije i pravila vrijede općenito za sve elemente i atribute u Android manifest dokumentu. Jedino su elementi <*manifest*> i <*application*> obavezni dijelovi *manifesta* i mogu se samo jednom pojaviti u dokumentu.

- Elementi

  Opcionalni elementi <*activitiy*>, <*provider*>, <*service*> i <*receiver*> se moraju nalaziti unutar elementa <*application*> po proizvoljnom redoslijedu. Sve vrijednosti koje opisuje pojedini element unose se pomoću vrijednosti atributa.

- Atributi

  Atributi se imenuju na način *android : ime_atributa = vrijednost_atributa* (npr. *android:name = "com.example.project. ExampleActivity").*

- Nazivi klasa

Svaka klasa se mora deklarirati u *manifest* dokumentu. Definicija klase je moguća na dva načina. Kod naziva komponente se dodaje atribut *android:name*, a vrijednost atributa može sadržavati punu oznaku paketa ili samo ime klase. Na primjer, klasu možemo imenovati sa <*service android:name="com.example.project.SecretService"...*> ili sa <*service android:name=".SecretService" . . .* >.

Već je spomenuto da se aplikacijske komponente aktiviraju preko namjera (engl. *intents*). Kako bi informirali sustav koje namjere mogu izvršiti, komponente mogu imati jedan ili više filtera za namjere (engl. *intent filter*). Svaki filter opisuje skup namjera koje je komponenta voljna primiti. Eksplicitna namjera se uvijek dostavi na odredište, dok implicitne namjere stižu do odredišta samo ako prođu kroz jedan od filtera. Obavezni filteri kod deklaracije početne aktivnosti su *Main* i *Launcher*. *Main* definira koji će se ekran otvoriti po pokretanju aplikacije, a *Launcher* pokretanje aplikacije. Svaka *Android* aplikacija se izvodi u vlastitom procesu. Sigurnost između aplikacije i sustava je izvedena na procesnoj razini pomoću dodjeljivanja jedinstvenog *Linux* identifikatora. Dodatne sigurnosne mjere se provode pomoću mehanizama dopuštenja (engl. *permission*). Dopuštenje predstavlja ograničavajući pristup na dio koda ili podataka o uređaju. Osnovna *Android*ova aplikacija nema nikakvih dopuštenja povezanih sa sobom. Kako bi se mogle koristiti zaštićene mogućnosti mobitela, potrebno je u manifestu deklarirati dopuštenja preko oznaka <*uses-permission android: name=”...”/*>. Svaka aplikacija je povezana sa standardnom *Android*-ovom bibliotekom koja
se sastoji od osnovnih paketa za razvoj aplikacija (npr. *Activity*, *Service*, *View*, *Button*). Kad *Android*-ova biblioteka nije dostatna za potrebe programiranja, aplikacije koriste biblioteke s drugih izvora. Korištenje eksterne biblioteke deklarira se u *Android manifestu* putem oznake <*useslibrary*>.

3.3.6. Aplikacijski resursi

Za grafički prikaz aplikacije koristi se direktorij resursi u kojem mogu biti spremljene slike, ikone, audio datoteke, izbornici, boje, fontovi i ostalo što ima veze sa vizualnom prezentacijom. Korištenje resursa olakšava ažuriranje audio-grafike bez modificiranja koda. Za svaki resurs koji je uključen u Android projekt, *Android SDK* definira jedinstveni identifikator, koji se koristi kao poveznica aplikacijskog koda i drugih izvora definiranih u *XML*-u. Svi identifikatori se spremaju u autogeneriranoj datoteci *R.java. R.java* služi kao indeks svih resursa korištenih u projektu. Na primjer, aplikacija sadrži slikovnu datoteku pod nazivom *logo.png*. Slikovna datoteka *logo.png* je spremljena u *res / drawable* direktoriju. *Android SDK* generira identifikator resursa pod nazivom *R.drawable.logo* koji se koristi u programskom kodu za umetanje slike u korisničko sučelje. Jedan od važnijih aspekata izolacije resursa od programskog koda mogućnost je osiguranja alternativnih resursa za različite konfiguracije uređaja. Na primjer, moguće je kreirati aplikaciju koja će podržavati više jezika. Za svaki jezik se kreira zasebni direktorij u resursima sa sufiksom jezika, a u svakom direktoriju su datoteke na tom jeziku. Na temelju kvalifikatora jezika, Android sustav primjenjuje odgovarajući jezik na svom korisničkom sučelju.

3.4. Java programski jezik

Java je objektno-orijentirani programski jezik razvijen u timu predvođenim James Goslingom u kompaniji *Sun Microsystems* početkom 1990. g. [3]. *Java* pripada skupini viših programskih jezika i ima svojstvo prenosivosti. Prenosivost označava mogućnost izvršenja jednog te istog programa na različitim računalima. Program koji je pisan u višem programskom jeziku ne može biti izvršen izravno na računalu. Takav program se prevodi u strojni jezik. Tu pretvorbu obavlja zasebni program kompajler (engl. *compiler*). Nakon što je program jednom preveden, može se izvršavati neograničen broj puta, ali samo na jednom računalu. Da bi se mogao izvršavati na drukčijem računalu, potrebno ga je ponovno kompajlirati. Umjesto korištenja kompajlera, moguće je koristiti interpreter, koji prevodi naredbu po naredbu, prema potrebi. Interpreter omogućuje izvršavanje programa kompajliranog za jednu vrstu računala na drugom računalu. Kod programskog jezika *Java*, koristi se kombinacija kompajlera i interpretera. U *Java* programskom jeziku su objektno-orijentirani principi obavezni. Sve je u Javi objekt, a sav izvorni kod je pisan unutar klasa. Objekt u realnom svijetu može biti čovjek ili bilo što drugo.

Objekt je definiran svojim stanjem i ponašanjem. Na primjer, stanje objekta čovjek može biti da trči ili stoji, a ponašanje može biti brzina trčanja, smjer kretanja, itd. U programskom jeziku stanje se opisuje sa varijablama (promjenjivim vrijednostima), a ponašanje se definira sa metodama. Klasa predstavlja nacrt objekta. Na temelju klasa se proizvode objekti. Na primjer, klasa koja sadrži općeniti objekt lik može kreirati više likova sa različitim stanjem i ponašanjem. Klasa može označavati dio programskog koda ili cijeli programski kod. U pravilu, svaka klasa je deklarirana unutar datoteke sa istim imenom i ekstenzijom *.java*. Pravilo je da su imena klasa i datoteka u kojima su klase spremljene ista. Sve klase jedne aplikacije spremaju se u paket sa nazivom domene (npr. *example.com.data*). Paket predstavlja hijerarhijsko grupiranje razreda, a ime paketa odgovara strukturi direktorija na disku. Drugi pojam u Javi je metoda. Klase koje pokreću program moraju imati *Main()* metodu. Metode su zasebne cjeline unutar pojedine klase koje izvršavaju određene operacije. Metode su u pojmu objektno-orijentiranog programiranja objekti, tj. jedinice koje imaju svoje ponašanje, drže podatke i mogu međusobno djelovati. Programiranje se sastoji od oblikovanja skupa objekata koji na neki način opisuju problem koji treba riješiti.

3.4.1. Struktura Java datoteke

Programiranje u Javi se odvija u klasama. Ime klase je ujedno i ime programa. Svaka java datoteka se sastoji od tri dijela:

deklaracija paketa kojoj klasa pripada (engl. *package*), pri čemu ime paketa predstavlja hijerarhijski prikaz strukture direktorija u kojem se nalazi datoteka
opcionalni popis paketa koje treba uključiti (engl. *import*)
deklaracija klase; Java programski jezik podržava gniježđenje, što znači da je unutar jedne klase moguće deklarirati druge klase. Da bi se neki program mogao izvršiti, on mora sadržavati metodu Main(). Deklaracija metode sadrži 3 modifikatora, tj. *public*, *static* i *void*. *Public* označava javnu metodu, *void* metoda ne vraća ništa, a *static* metoda pripada samoj klasi te nije potrebno stvarati instancu kako bi se metoda mogla koristiti.

4. IDEJNA RAZRADA VOĐENJA GRAĐEVINSKOG DNEVNIKA U ELEKTRONSKOM OBLIKU NA PLATFORMI ANDROID

U okviru ovog diplomskog rada iznesena je ideja razrade aplikacije građevinskog dnevnika koja služi za njegovo vođenje na mobilnom uređaju ili tabletu, elektronski. Kao demonstracijski primjer uzet je projekt Zaobilaznica Split-Stobreč koju je investirao Grad Split, a za izvođača je izabran Konstruktor inženjering iz Splita. U ovom poglavlju opisana je predložena razrada same aplikacije koja u stvarnosti nije izvedena do kraja. Na temelju same ideje, izvedena su sučelja i njihova međudjelovanja te je iznesen plan i program daljnje razrade po završetku obrane diplomskog rada. Uz sve opisane funkcije aplikacije ispisane su ideje cijeloga rješenja aplikacije koja bi vrlo lako mogla biti razrađena u potpunosti kao prototip iste. Dati su primjeri i detaljna objašnjenja korištenja aplikacije kao i njena programska izvedba.

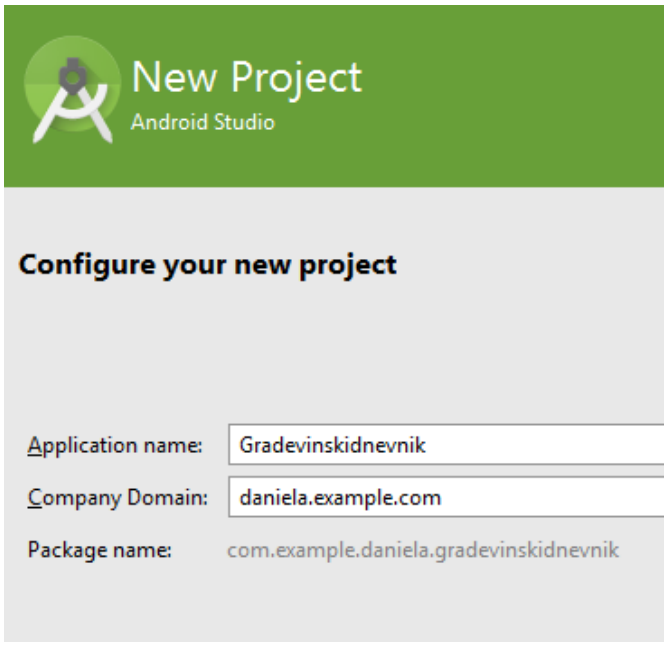
4.1. Alati korišteni pri izradi aplikacije

Pri izradi aplikacije koristila se razvojna okolina Android Studio. Naime Android Studio prepoznaje kod programa i korisniku pomaže dajući mu predloške. Jednostavan je za korištenje i omogućava brzu navigaciju. Programski kod je napisan u Java programskom jeziku, a za pokretanje aplikacije koristio se mobilni uređaj. Aplikacija ima jednaki rezultat i na tabletu koji bi bio povoljniji za unos izvođenja nekog građevinskog projekta na terenu, prvenstveno radi njegove veličine i širine ekrana, a uz to i bolje preglednosti same aplikacije.

4.1.1. Razvojna okolina Android Studio

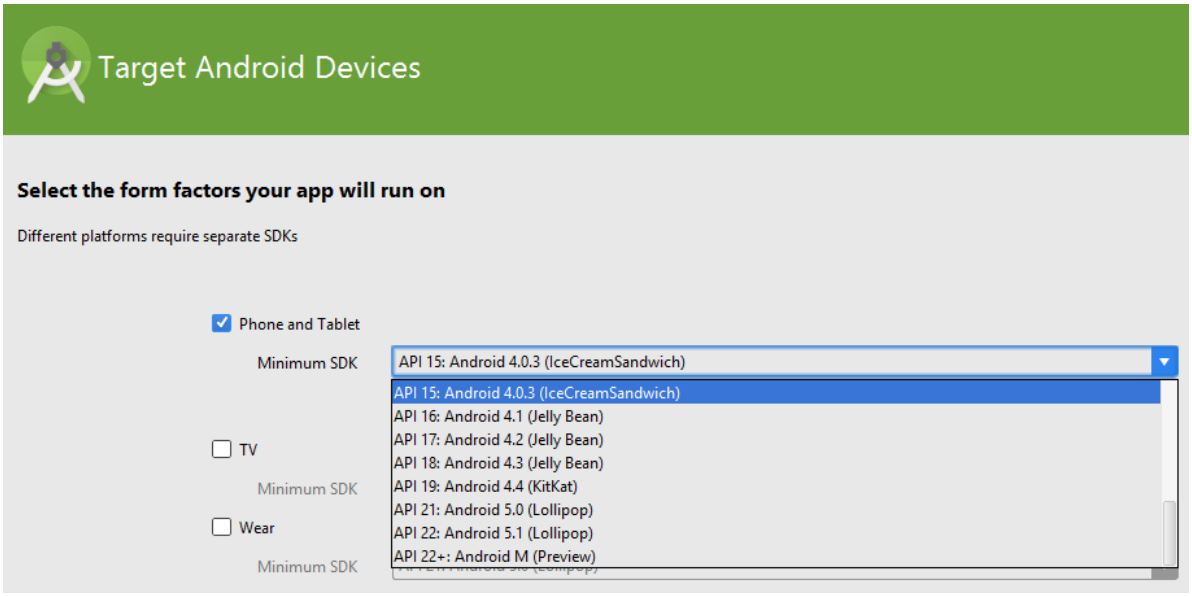
Android Studio je službeni IDE za Android programiranje. Može se preuzeti bez ikakvih troškova, a otvoren je za doprinose članova zajednice. Proizvod je izgrađen na IntelliJ platformi, koja je u potpunosti *open source*. Java IDE (Integrated Development Environment) je softverski program koji omogućuje korisnicima lakše pisanje i ispravljanje Java programa. Sa stranice http://www.oracle.com/technetwork/java/javase/downloads/index.html se može preuzeti Java JDK. Prije upotrebe treba preuzeti najnoviju verziju http://developer.android.com/sdk/index.html ADT paketa. Android Development Tools paket sadrži sve potrebno za programiranje za Android. Nakon instalacije potrebno je upaliti SDK manager (nalazi se u folderu android SDK tools gdje je instaliran Android SDK) i instalirati

predložene pakete. Bitno je instalirati najnoviju verziju SDK AP-ja. Nakon odabira novog projekta dato je ime projekta: Gradevinskidnevnik, što se vidi na slici 4.1.



Slika 4.1. Nazivanje projekta u Android Studiu

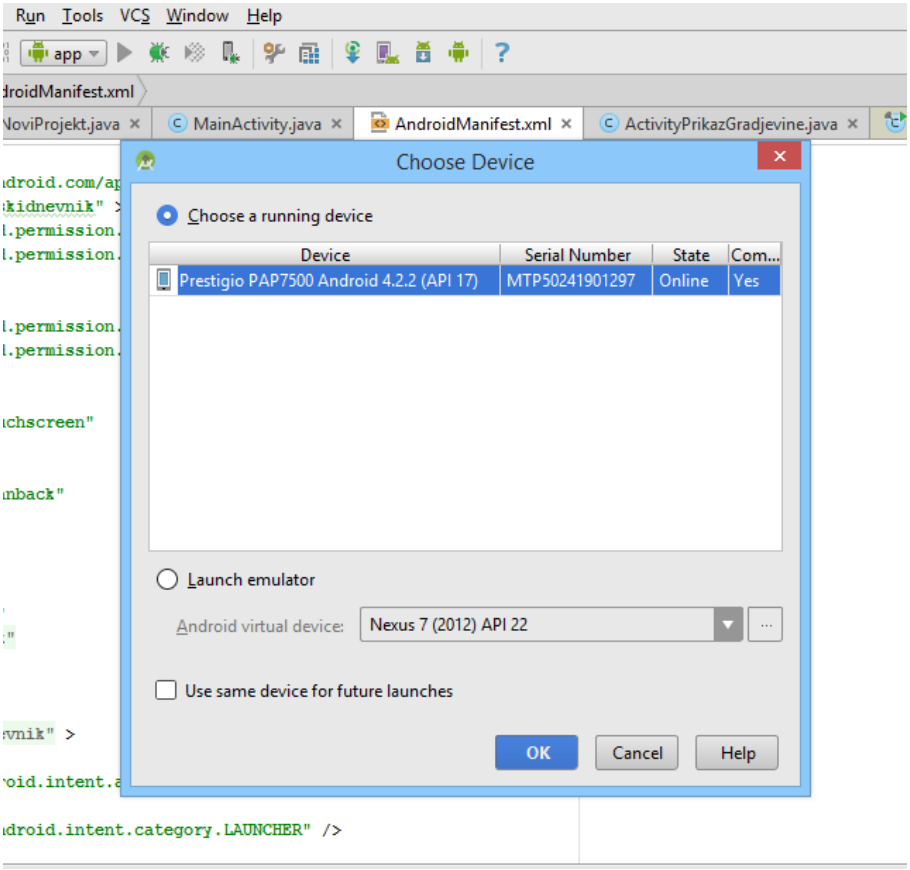
Biramo platformu API 15 : Android 4.0.3 (IceCreamSandwich) koja pokriva aproksimativno 90.4% uređaja aktivnih na Google Play Store-u. Mogućnosti pri kreiranju novog projekta vide se na slici 4.2.



Slika 4.2. Biranje platforme u Android Studiu

Mobilni uređaj je android uređaj uključen preko usb kabela na računalo i može ostati upaljen tijekom cijelog vremena dok programirate (ne pali se posebno svaki put kada se želi pokrenuti aplikacija).

Aplikacija se pokreće klikom na Run -> "Run" ili pritiskom Shift+F10 i otvara se na odabranom "virtualnom" ili stvarnom mobitelu. Na slici 4.3. prikazano je pokretanje novog projekta u Android Studiu.



Slika 4.3. Pokretanje novog projekta u Android Studiu

Layoute je moguće uređivati u oba načina rada (dizajn i text način rada). U ovom diplomskog radu je primijenjeno kombinirano postavljanje svih potrebnih *itemsa* za svaki *layout.*

U text mode pregledu dizajna, koji je prikazan na slici 4.4., umjesto umetanja određenih dodataka piše se kod. Pojedinom elementu se dodaju atributi, npr. *TextView* elementu se može dodati atribut id: *android:id="@+id/testTextView".*



Slika 4.4. Tekstualni način rada

Dizajn je dobar način za započeti razvoj, međutim pisanje XML-a je, jednom kada se uhvati praksa puno kvalitetnije. Jedan ili drugi se otvara: *res/layout/main.xml*, a zatim se bira Design ili Text mode pregleda aplikacije.

Sami dizajn aplikacije je izrađen preko internet stranice http://angrytools.com/ gdje na jednostavan način možemo kreirati pozadinu i/ili engl.*buttons* (u daljnjem tekstu botuni) aplikacije. Ovim diplomskim radom je iznesena jedna verzija dizajna aplikacije koja lako može biti promijenjena i prilagođena korisnicima ove aplikacije. Osnovni dizajn, pozadina aplikacije te izgled osnovnih botuna, je prikazan je na slici 4.5.



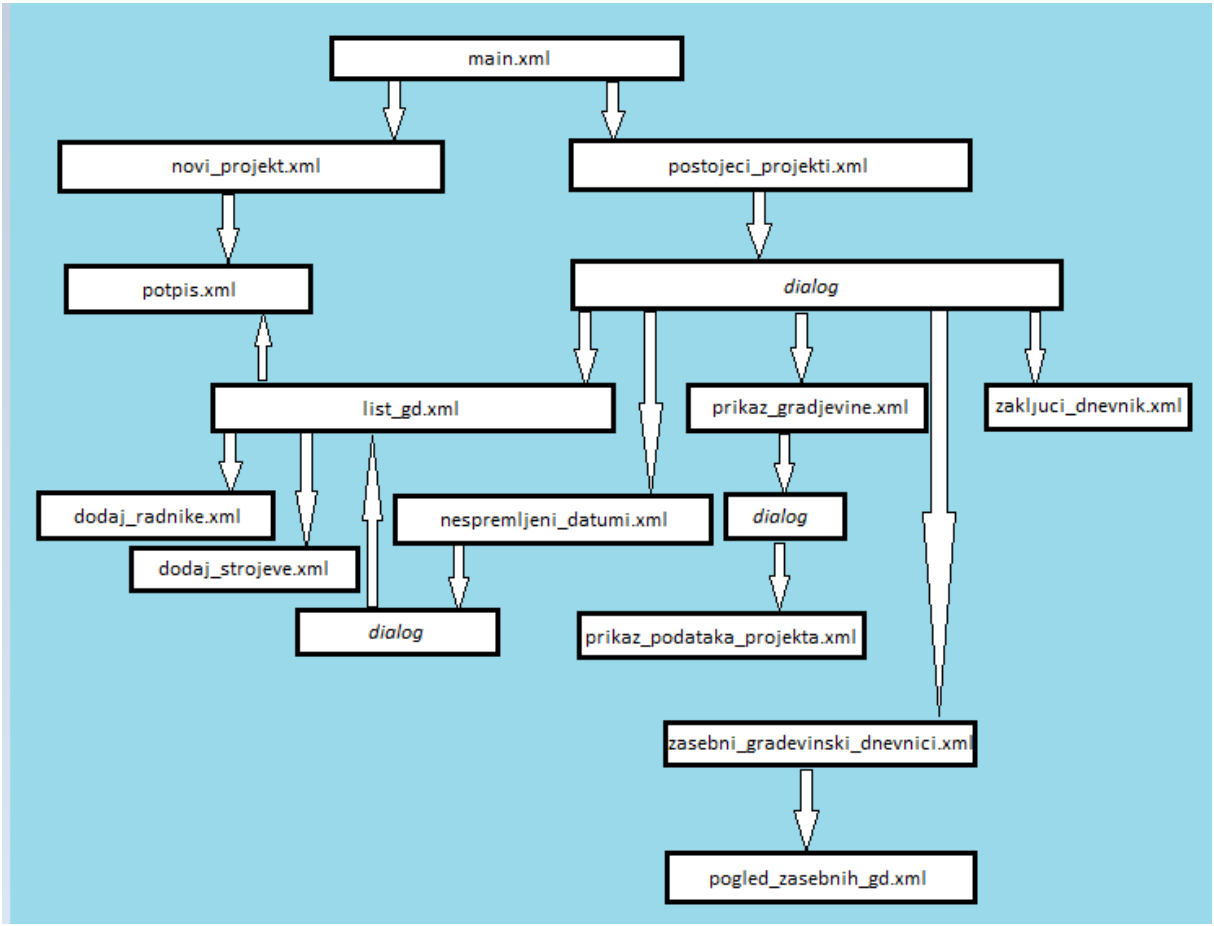
Slika 4.5. Dizajn način rada

Ovako izgleda prvo sučelje aplikacije odnosno ovakav *activity* (u daljnjem tekstu aktiviti) je pridodan početnom *activity_main.xml* aktivitiju i takav dizajn vrijedi za sve aktivitije unutar aplikacije. Ikona je izrađena u PhotoScape programu koji je omogućio spajanje dviju sličica skinutih za besplatno korištenje sa internet stranice http://www.iconarchive.com/ . Aplikacija se jednostavno „skine“ sa Android Market-a nakon što ju je razvojni inženjer (engl.*software developer*) tamo i postavio. Zbog takvog jednostavnog i nesmetanog „stavljanja-skidanja“ aplikacije je i izabran Android Studio.

Sama aplikacija ima mogućnost rotiranja ekrana i tako ostavlja na izbor korisniku način unosa. U daljnjim primjerima prilagođena je rotacija određenoj tematici, zavisno o načinu boljeg prikazivanja željenoga.

4.2. Opis grafičkog sučelja aplikacije

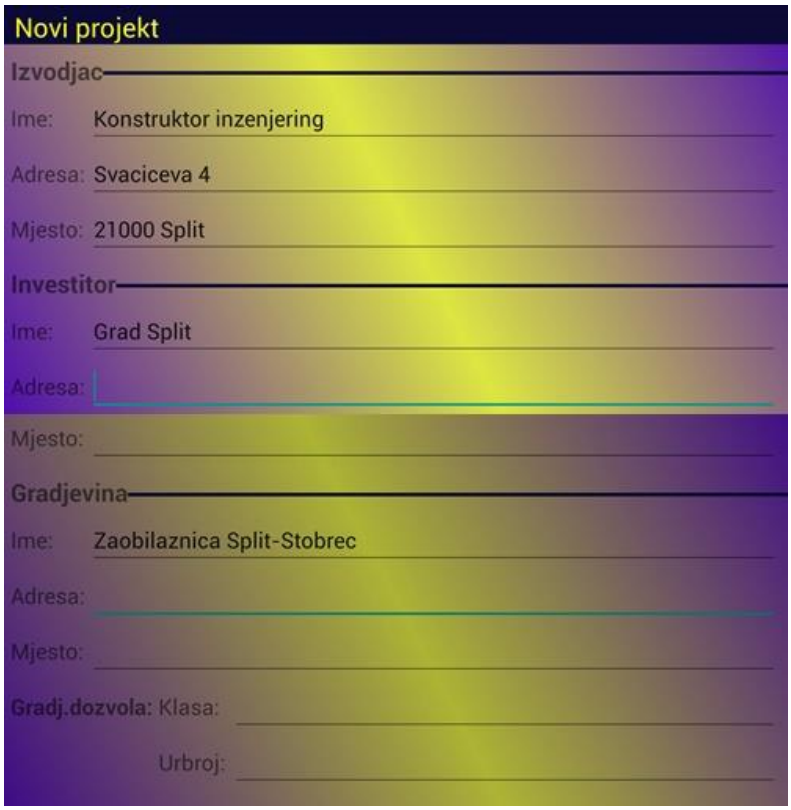
Cilj aplikacije je na jednostavan način unijeti podatke za pisanje građevinskog dnevnika. Na slici 4.6. vidi se grafički prikaz spajanja prozora razreda pri korištenju aplikacije. Prikazana je grana otvaranja novih prozora. Postoje još i botuni za povratak koji većinom vode u *activity_postojeci_projekti.xml* prozor.



Slika 4.6. Prikaz spajanja razreda pri korištenju aplikacije

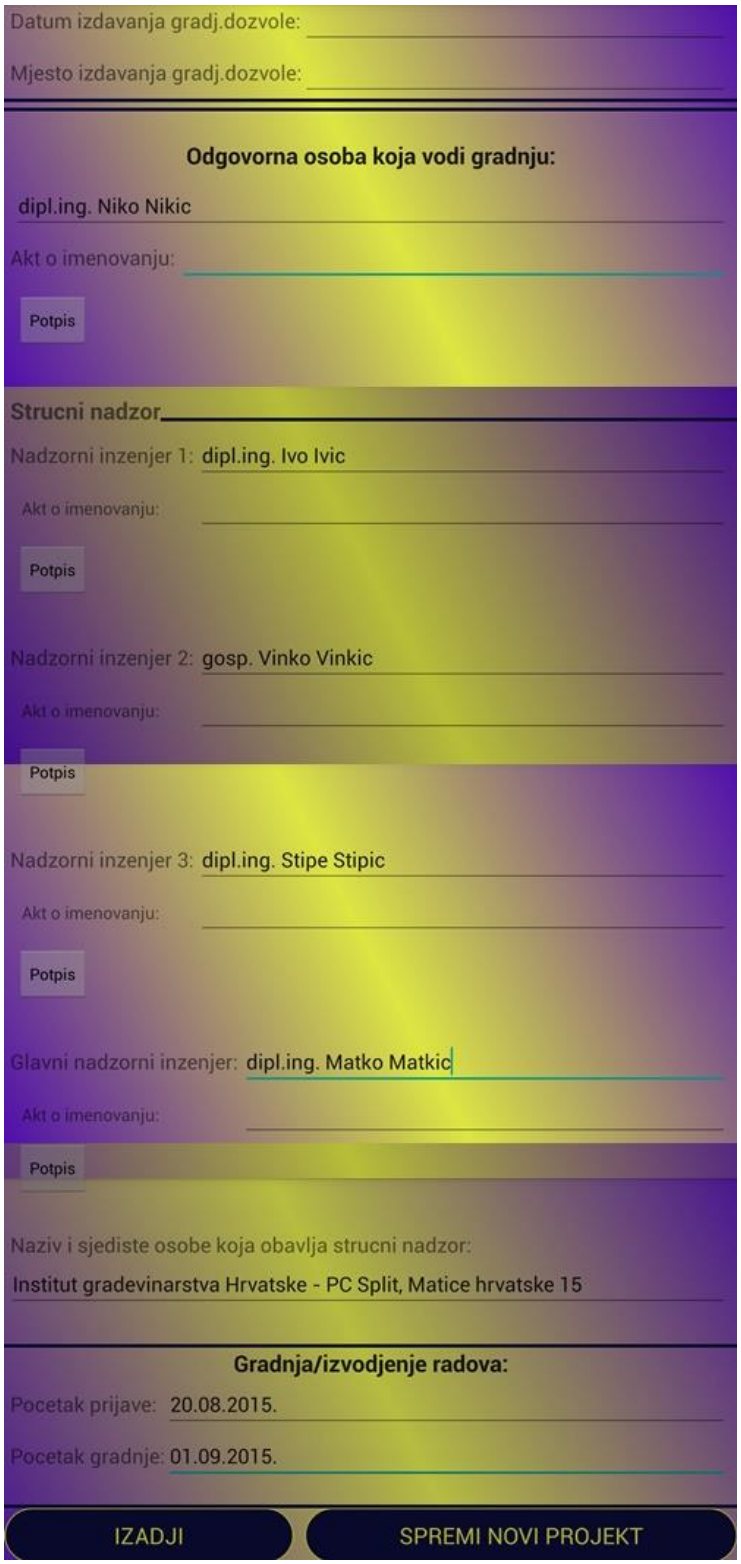
Prvi prozor koji se otvara prilikom pokretanja aplikacije je activity*_main.xml*, koji je prikazan na slici 4.5. Dakle, kad korisnik pokrene aplikaciju na ekranu se prikaže prva aktivnost sa imenom aplikacije *Gradevinski dnevnik* (nemogućnost pisanja cijelom latinicom pa su se određena slova morala zamijeniti na najbolji mogući način engleskim pismom). Tu su postavljena dva botuna, jedan za otvaranje prozora novog projekta *activity_novi_projekt.xml*, a drugi za otvaranje svih postojećih projekata *activity_postojeci_projekti.xml* na tom uređaju ove aplikacije. U samom dnu je postavljen link za skidanje pravilnika pisanja građevinskog

dnevnika na elektronski uređaj u pdf obliku. *Activity_novi_projekt.xml*, je prozor za unošenje novog projekta s njegovim podacima. Slika 4.7. i Slika 4.8. prikazuju taj prozor, koji omogućava korisniku upis podataka o izabranom izvođaču (ime, adresa i mjesto). Takvi podaci se upisuju i za investitora i za građevinu za koju radimo projekt. Samo ime građevine je ime projekta koje želimo spremiti u memoriju i pozvati ga svaki put kada želimo raditi daljnji dnevnik o toj građevini. Uz takve podatke o građevini upisuju se i podaci o građevinskoj dozvoli čije podatke možemo vidjeti i na ploči projekta na gradilištu. Znači, upisuje se klasa, urudžbeni broj, datum izdavanja građevne dozvole i mjesto tog izdavanja.



Slika 4.7. Prvi dio izgleda activity_novi_projekt.xml

Prikazana slika 4.7. je napravljena u Photo Scape-u na način da se *screenshotao* zaslon aplikacije na mobitelu i ukomponirao u jednu sliku rade bolje preglednosti što aplikacija i na koji način sadrži. Na takav način su napravljene i ostale slike koje možemo primijetiti. Sama aplikacija ima mogućnost *ScrollView*-a u bilo kojem smjeru da se uređaj okrene.



Slika 4.8. Krajnji dio izgleda activity_novi_projekt.xml

Ostatak ovog prozora otvara mogućnost upisa imena i prezimena odgovorne osobe koja vodi gradnju kao i podatke o stručnom nadzoru kojeg sačinjavaju nadzorni inženjeri te glavni

nadzorni inženjer. Za njih se upisuju i podaci o aktu o njihovom imenovanju. Svaki od njih se mora potpisati pa klikom na botun *Potpis* se otvara novi aktiviti *activity_potpis.xml* koji je prikazan na slici 4.9. Tu se korisnik potpisuje te pritiskom na botun ispod, sprema se njegov potpis preko kojeg se potvrđuje o kojem se korisniku radi.

Za ovakvu ideju prepoznavanja se predlaže baza podataka sa svim zaposlenicima pojedine firme koja bi imala mogućnost prepoznavanja potpisa na temelju unesenog i spremljenog u memoriju. Za ovaj primjer je dat potpis koji se po spremanju vraća u *activity_novi_projekt.xml*.



Slika 4.9. Primjer potpisa

Primjer kako bi trebao izgledati ekran nakon unesenog potpisa je dat na slici 4.10.



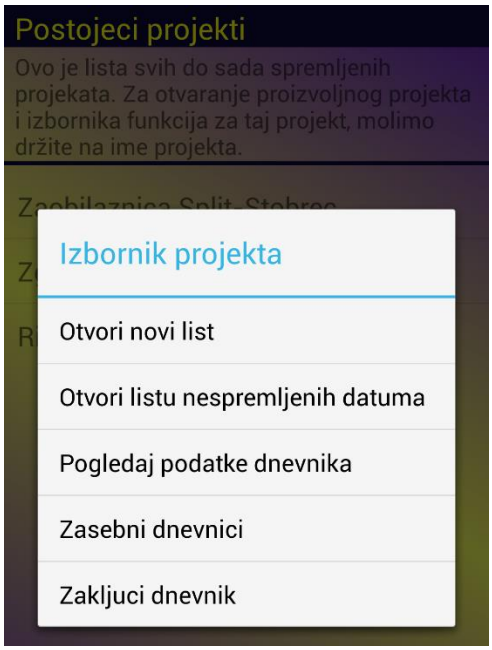
Slika 4.10. Izgled prozora novog projekta nakon digitalnog potpisivanja

U tom aktivitiju se upisuju podaci o početku prijave i početku građenja. Oba botuna koja su na dnu, *Izadji* i *Spremi novi projekt*, nas vraćaju na prvi aktiviti odakle ponovno biramo gdje ćemo. *Spremi novi projekt* nam sprema projekt te ga kreira u listi koja je ispisana u novom aktivitiju *activity_postojeci_projekti.xml*. Pritiskom na botun *Postojeci projekti* otvara se taj prozor. Dodavanjem novog projekta, lista se regenerira, a zadržavaju se svi spremljeni projekti u memoriji. Takav izgled pokazuje slika 4.11.



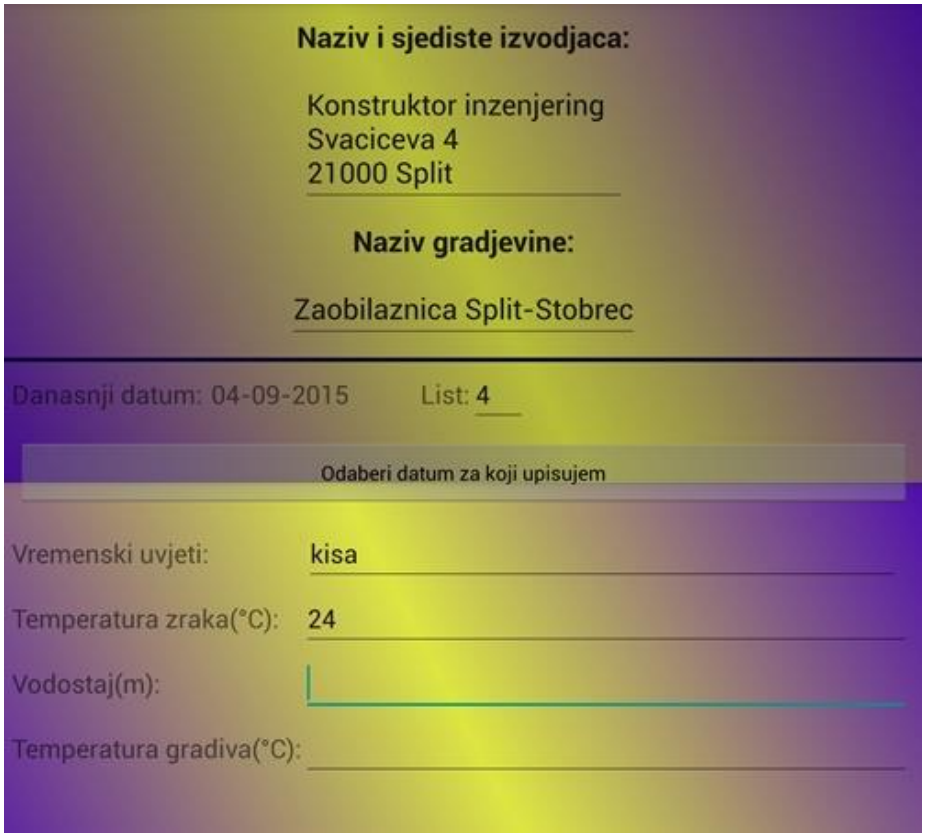
Slika 4.11. activity_postojeci_projekti.xml

Kao što sama napomena prikazuje, držeći na ime projekta otvara se izbornik kao na slici 4.12. Ako samo kliknemo na projekt, izađe *Toast* sa cijelim imenom projekta.



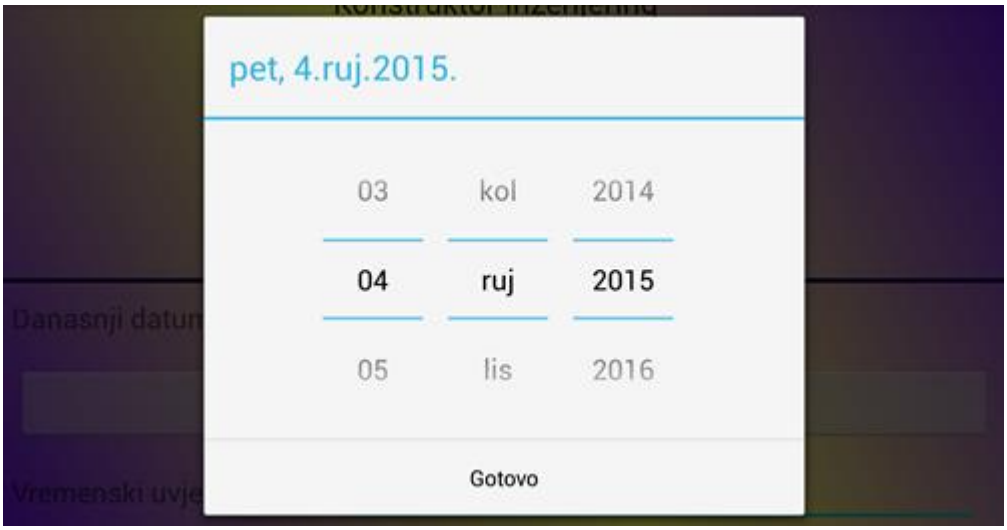
Slika 4.12. Izbornik projekta

Što se tiče samog izbornika za projekt, on sadržava prikazane naslove za daljnje korištenje dnevnika. *Otvori novi list* otvara novi prozor *activity_list_gd.xml* čiji sadržaj možemo vidjeti na slikama 4.13. i 4.15.



Slika 4.13. Početak prozora lista građevinskog dnevnika

Sa svakim novim unosom dnevnika po datumima se ulazi na poruku izbornika *Otvori novi list*. Kao što je na prvoj slici prikazano, automatski izbaci ime projekta te naziv i sjedište izvođača. Automatski broji stranice po otvaranju novog lista koji se baziraju na već spremljenim stranicama. Vidimo da automatski izbaci i današnji datum koji nema mogućnost mijenjanja. Ispod toga je botun sa imenom *Odaberi datum za koji upisujem* i on nam služi za korisnikovo biranje željenog datuma, odnosno biranje datuma za koji unosi i sprema ovaj list.



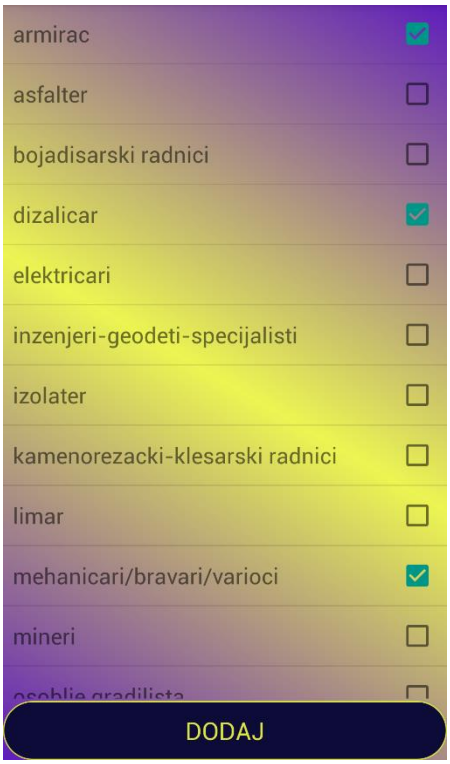
Slika 4.14. Dialog sa izborom datuma

Na slici 4.14. prikazan je *dialog* sa jednostavnim biranjem datuma, a prvi datum koji se pokaže je datum klikanja botuna. To uvelike pomaže korisniku da ne mora unositi podatke koji se svakodnevno upisuju, već je automatski urađeno te korisnik ne mora doći do zasićenja kada ispisuje dnevnik što već znamo da je jedan od problema koje imaju odgovorne osobe koje upisuju građevinski dnevnik. Drugi problem je što se zna dogoditi da npr. izvođač ne ispisuje dan za dan dnevnik ili da nadzornog inženjera nema pa je ovdje ideja da se tim način unosa datuma i automatskim datumom koji je odmah prikazan kontrolira upisivanje dnevnika.

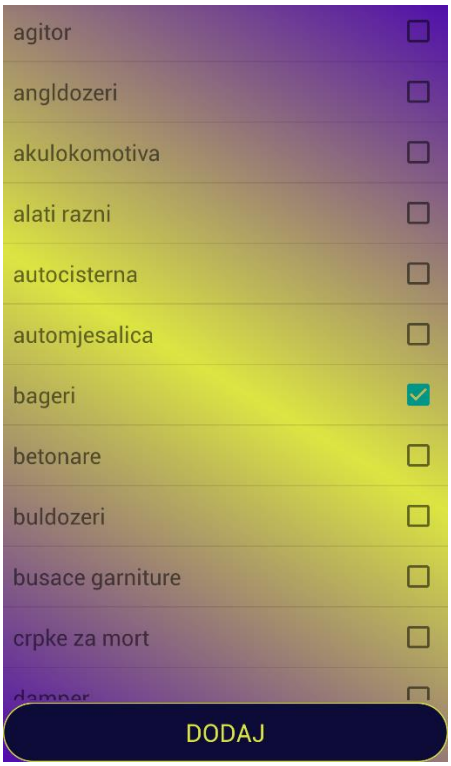


Slika 4.15. Krajnji dio prozora upisa novog lista dnevnika

Ako promatramo odjeljak Strukture zaposlenika i strojeva, vidimo da je tu prikazana samo napomena sa mogućnosti klikanja na botun ispod toga. Ako to uradimo, otvara se cijeli izbornik sa ispisanim zaposlenicima, odnosno strojevima, zavisno o tome koji botun kliknemo. Kod takvog ulaženja u novi aktiviti otvara se *activity_dodaj_radnike.xml* ili *activity_dodaj_strojeve.xml*. Njihov primjer je dat na slikama 4.16 i 4.17.

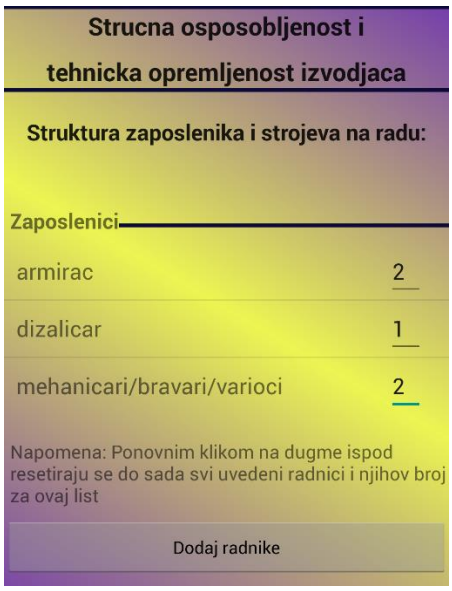


Slika 4.16. Zaposlenici

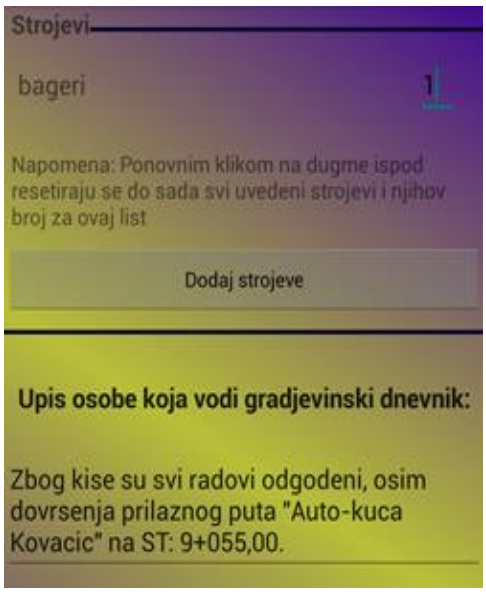


Slika 4.17. Strojevi

Jednostavnim klikom na prave zaposlenike odnosno strojeve i biranjem *Dodaj*, vraćaju se podaci u prozor *activity_listGD.xml*. Sada tu imamo ispisane označene *itemse* i možemo im pridodati broj, količinu. Takav primjer je prikazan na slikama 4.18 i 4.19.



Slika 4.18. Spremljeni zaposlenici



Slika 4.19. Spremljeni strojevi

Sa slike 4.15. možemo uočiti da postoji odjeljak o upisu osobe koja vodi građevinski dnevnik. Na isti način se potpisuje i po završetku upisa klikom na botun *Spremi danasnji list*, do tada uneseni podaci o toj građevini toga dana se spremaju i kreira se lista po datumima izabranim *dialogom* (slika 4.14.). Odjeljak koji je prikazan upisom nadzornog inženjera i drugih osoba je značajan po svom automatskom datumu koji se regenerira kad nadzorni inženjer želi upisati svoj dio. Ponovno, zbog kontrole upisa njihovih tekstova za pojedini dan. Kada on klikne botun ispod *Spremi*, tada je napokon zaključen taj dan upisa dnevnika. Nije moguće točno taj botun *Spremi* kliknuti, odnosno nije u funkciji, dok nije spremljen tekst odgovorne osobe i dok ostali podaci nisu uneseni.

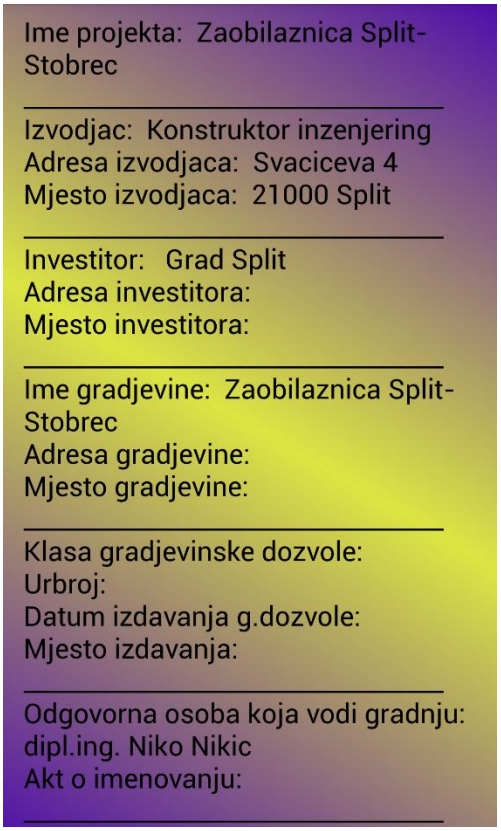
Znači, kada odgovorna osoba za izvođenje radova spremi svoj dio, kreira se lista u novom prozoru što se vidi na slici 4.20. U taj prozor se može ući odabirom *Otvori listu nespremljenih datuma* sa već definiranog *dialoga.*

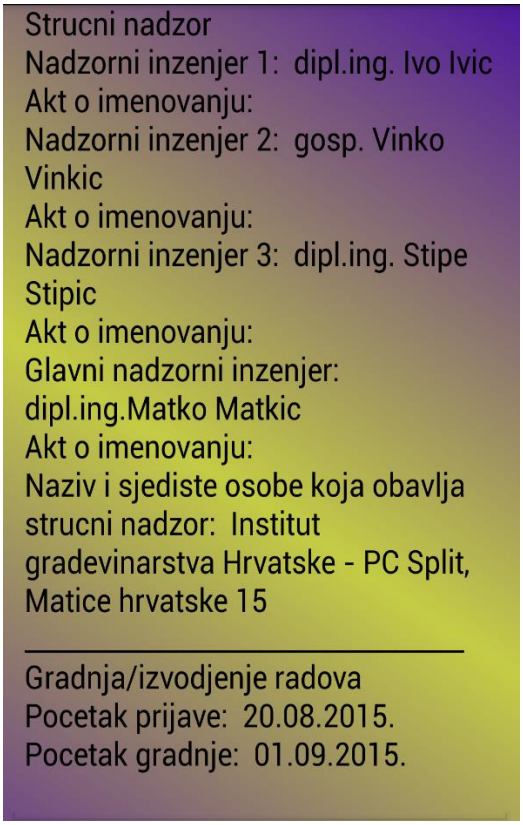


Slika 4.20. activity_nespremljeni_datumi .xml

Kao što piše u napomeni za zaključivanje odabranog datuma treba držati na taj isti datum gdje se otvori mali *dialog* s porukom *Zavrsi ovaj dnevnik.* Odabirom takve poruke nadzorni inženjer sada ponovno ulazi u aktiviti sa tom listom dnevnika gdje su tada svi podaci izvođača upisani te taj dio je zamrznut za mijenjanje. Nadzorni inženjer sada može dopisati svojim tekstom dnevnika, potpisati i spremiti. Kao što je rečeno, tada se taj datum briše iz privremene memorije te ostaje lista samo onih datuma tog projekta koje nadzorni inženjer nije spremio.

Takav spremljeni datum, zajedno sa ostalim spremljenim datumima, nalazi se u posebnoj listi u aktivitiju *activity_prikaz_gradjevine.xml*. Taj aktiviti se otvara s *dialoga* postojećih projekata pod porukom *Pogledaj podatke dnevnika* (slika 4.21. i slika 4.22.).





Slika 4.21.Početak activity_prikaz_gradjevine.xml *Slika 4.22.Kraj activity_prikaz_gradjevine.xml*

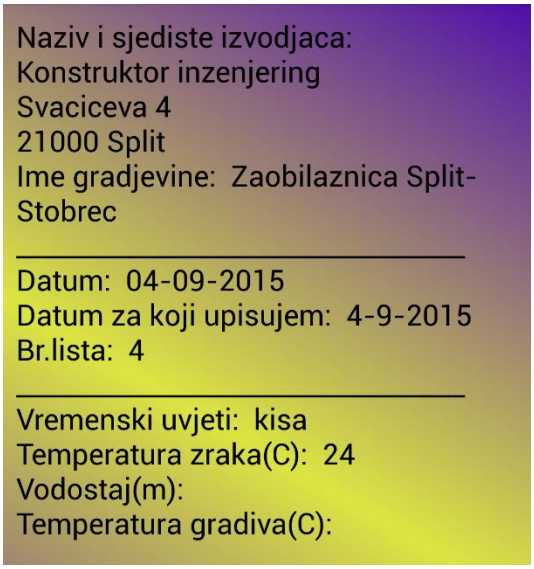
Ovakav prozor aplikacije nam omogućava uvid u sami projekt, zapravo, sve što je uneseno prilikom kreiranja takvog projekta se u tom prozoru ispisuje. Ispod tih podataka je lista svih do tada spremljenih datuma za taj projekt (slika 4.23).



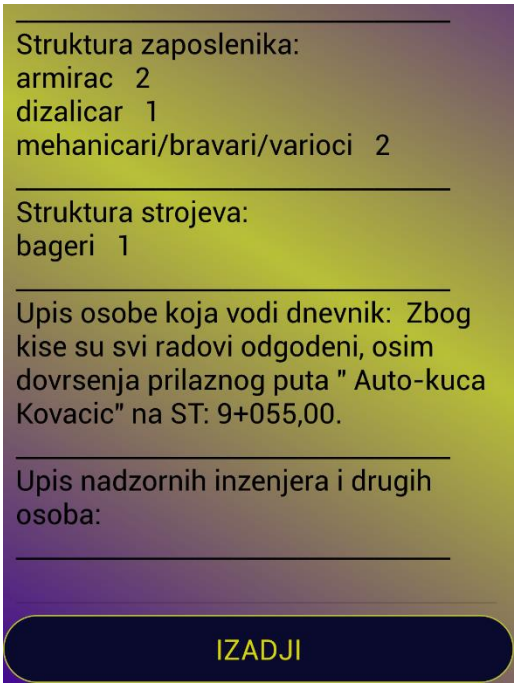
Slika 4.23. Lista spremljenih datuma dnevnika

Na njihov odabir, odnosno na odabir datuma se ulazi preko novog malog *dialoga* sa porukom *Pogledaj ovaj datum.* Tu se otvara novi prozor, *activity_prikaz_podataka_projekta.xml*.

U njemu je samo predviđen ispis podataka po datumima. Iz tog aktivitija se ne može ići nigdje naprijed, samo se botunom *Izadji* vraća na *activity_postojeci_projekti.xml*. Sami primjer je na slikama 4.24. i 4.25.

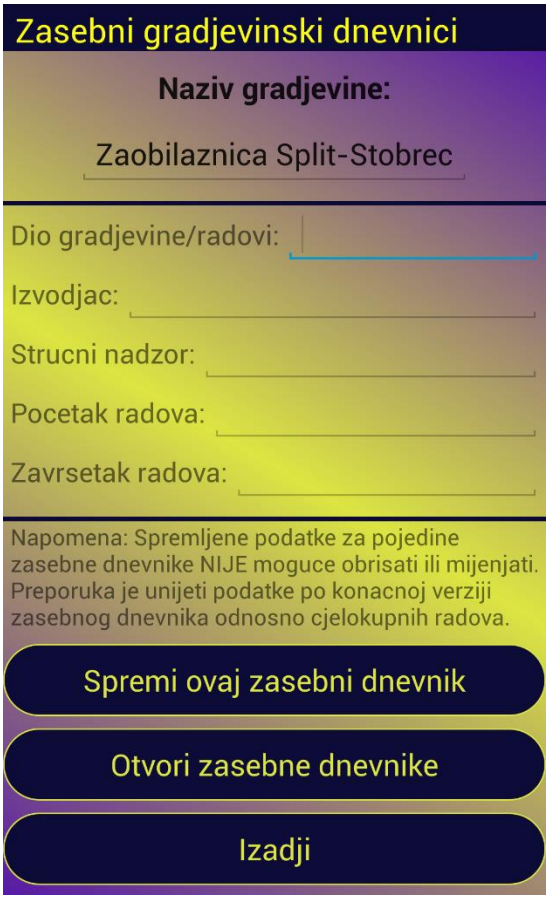


Slika 4.24.Ispis podataka po datumima, prvi dio



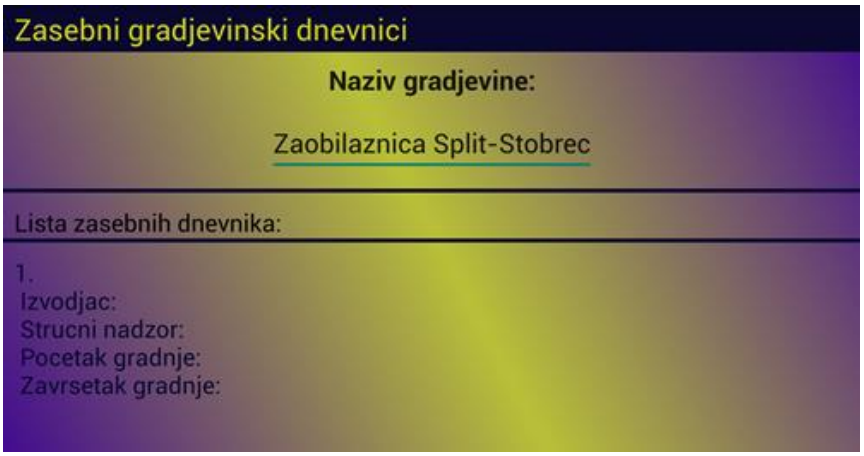
Slika 4.25.Ispis podataka po datumima, drugi dio

Iz prozora u kojem su ispisani svi projekti i kao što je rečeno, na njihov odabir se otvara prozorčić sa *dialogom* i ako sada izaberemo poruku sa imenom *Zasebni dnevnici* otvara nam se novi aktiviti *activity_zasebni_gradevinski_dnevnici.xml*. Sami izgled je dat na slici 4.26.



Slika 4.26. activity_zasebni_gradevinski_dnevnici.xml

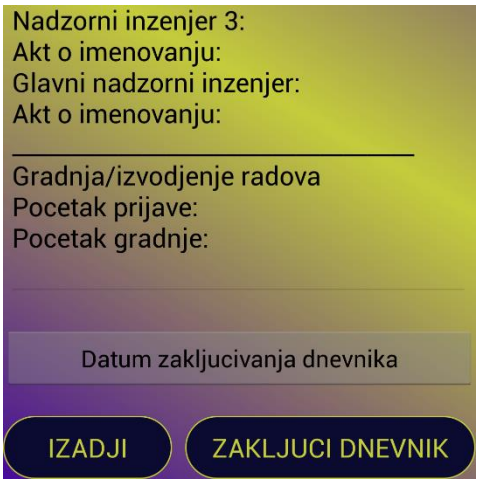
Prikazan je naziv projekta za kojeg se ispisuju zasebni građevinski dnevnici. To je ujedno i naziv građevine. Ispod korisnik ima mogućnost upisa podataka kao što su dio građevine/radovi, ime izvođača, tko je stručni nadzor te početak i završetak točno tih radova. Napomena je postavljena da se zna da, kada korisnik upiše podatke i stisne *Spremi ovaj zasebni dnevnik*, nije moguće mijenjati podatke. To naravno nije ništa teže, nego što je bilo kada se ispisivalo na papiru jer cijeli dnevnik sam po sebi ne smije biti brisan ili mijenjan. Može samo biti nadopunjen. Tako da korisnik ima mogućnost da pregleda upisane podatke prije nego se odluči na spremanje samih tih podataka. Klikom na taj botun, otvara se novi aktiviti. U taj aktiviti se može ući i samo ako se želi pogledati što sačinjavaju zasebni građevinski dnevnici. Da se to omogući korisniku, dizajniran je botun *Otvori zasebne dnevnike*. Aktiviti o kojem se priča je prikazan na slici 4.27.



Slika 4.27. activity_pogled_zasebnih_gd.xml

Ovaj aktiviti se zove *activity_pogled_zasebnih_gd.xml*. Opet se ispisuje naziv građevine, odnosno projekta, a ispod toga je sama lista zasebnih dnevnika. Kako nismo ništa unijeli po spremanju zasebnog dnevnika, ispisan je tekst koji stoji i čeka da mu se dodijele vrijednosti o dnevniku. Na jednostavan način se ispisuju ti podaci, a lista se stvara jedna ispod druge. Sa ovog prozora se ne može naprijed, već klikom za izaći se opet odlazi u aktiviti postojećih projekata, kao i sa prethodnog aktivitija.

Zadnja opcija koja je dana za izbor korisniku u vezi određenog projekta je da se zaključi dnevnik. U tom aktivitiju se opet otvaraju podaci o tom projektu. Na slici ispod (slika 4.28.) je prikazan primjer bez unesenih podataka pa je naveden samo tekst koji čeka na ispis podataka koji moraju biti prethodno uneseni.



Slika 4.28. activity_zakljuci_dnevnik.xml

Aktiviti o kojem se priča se zove *activity_zakljuci_dnevnik.xml*. Tu imamo mogućnost zaključivanja dnevnika po čemu se dalje ne mogu unositi podaci o tom dnevniku. Botunom *Datum zakljucivanja dnevnika* se otvara mogućnost lakšeg izbora upisa datuma, kao što smo to koristili u aktivitiju *activity_list_gd.xml* (slika 4.14.).

Pri zaključivanju dnevnika predloženo je da se dnevnik briše iz aplikacije, ali da se šalje preko interneta na još nerazrađeni način i da se lako pregleda u nekom tekstualnom obliku. Za daljnji razvoj aplikacije lako se može, uz stručno znanje programera, isprogramirati detaljno spremanje svih podataka projekata te automatski slati nadzornom inženjeru podatke dnevnika da bi on mogao na vrijeme zaključiti dnevnik, kao i slati gotove podatke investitoru da može popratiti izvođenje radova. Detaljno spremanje i način na koji bi se mogli automatski slati podaci potrebnim osobama ili tvrtki, potrebno je dodatno razmotriti u suradnji sa programerima koji bi lakše napravili konačnu prvu verziju ove aplikacije. Bitno je znati način povezivanja svih unesenih podataka na uređaju sa automatskim prijenosom ka uređaju sa kojeg se može očitati ili čak i ispisati napisano. Za sada je dana sama ideja rada aplikacije kao i sve ono što sadašnja aplikacija ima. Po završetku rada aplikacije, potrebno ju je staviti na Android Market odakle bi se mogla lako skinuti i koristiti.

4.3. Programska izvedba aplikacije

4.3.1. Struktura podataka

Android Studio ima za sebe već razrađene klase sa već potrebnim napisanim kodom te koristeći točno određene metode lako ih se pozove pa razvojni inženjer može lakše pisati kod. Tako Android Studio posjeduje već gotov XML file *AndroidManifest.xml*. XML je skraćenica od Exstensible Markup Language odnosno to je jezik za označavanje podataka, a razumljiv je ljudima i računalima. Princip realizacije je vrlo jednostavan: odgovarajući sadržaj treba se uokviriti odgovarajućim oznakama koje ga opisuju i imaju poznato ili lako shvatljivo značenje. Dio koda je dat u primjeru ispod (tablica 4.1.).

Tablica 4.1. Dio koda u AndroidManifest.xml-u

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.daniela.gradevinskidnevnik" >

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

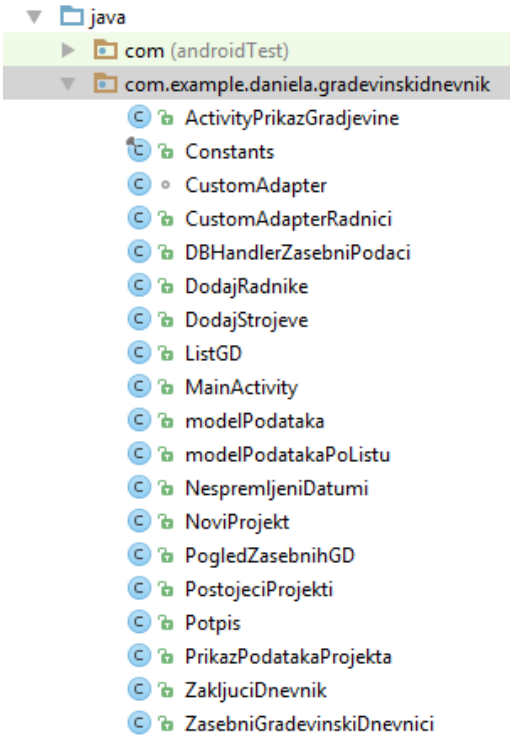
    <uses-feature
        android:name="android.hardware.touchscreen"
        android:required="false" />
    <uses-feature
        android:name="android.software.leanback"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

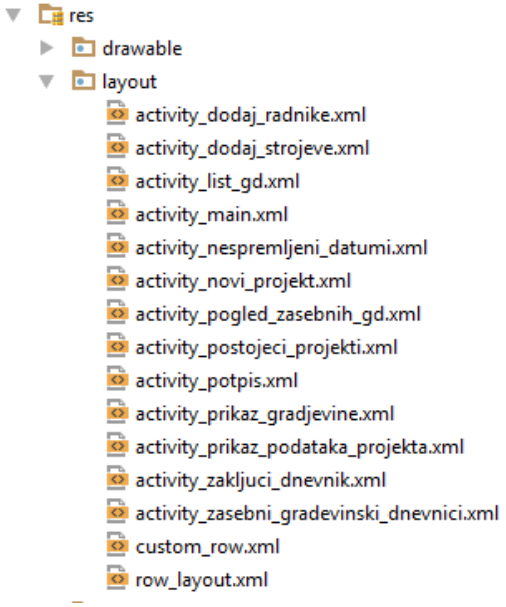
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
   ...
```

U Android Manifestu je potrebno pozvati neke stvari, kao na primjer korištenje interneta. Tu su definirane i sve klase koje su potrebne za stvaranje aplikacije. Glavna ikona koja se koristi za određenu aplikaciju se također poziva u Android Manifestu. Tu je definiran i stil, odnosno tema aplikacije.

Sve korištene klase i aktivitiji su prikazani na slikama 4.29. i 4.30.



Slika 4.29. .java klase



Slika 4.30. xml layouti

Već je spomenuto da se za građevinski dnevnik koristi struktura zaposlenika i strojeva. Cijela lista do sada unesenih strojeva i radnika je ispisana ispod (tablica 4.2. i 4.3), a zapravo je spremljena u Androidu u *strings.xml* file-u gdje se i upisuju, pa se lako može nadodati nepotrebni *item* ili izbrisati, promijeniti.

Tablica 4.2. Lista zaposlenika

```
<string-array name="dodaj_radnike">
    <item>armirac</item>
    <item>asfalter</item>
    <item>bojadisarski radnici</item>
    <item>dizalicar</item>
    <item>elektricari</item>
    <item>inzenjeri-geodeti-specijalisti</item>
    <item>izolater</item>
    <item>kamenorezacki-klesarski radnici</item>
    <item>limar</item>
    <item>mehanicari/bravari/varioci</item>
    <item>mineri</item>
    <item>osoblje gradilista</item>
    <item>parketar</item>
    <item>pocetnik NKV</item>
    <item>pomocnik NKV</item>
    <item>pomocnik PKV</item>
    <item>poslovodje</item>
    <item>staklorezac</item>
    <item>stolar</item>
    <item>strojari</item>
    <item>tesar</item>
    <item>voditelji gradj. pogona</item>
    <item>vozaci</item>
    <item>zidarsko-fasaderski radnici</item>
</string-array>
```

Tablica 4.3. Lista strojeva

```
<string-array name="dodaj_strojeve">
    <item>agitor</item>
    <item>angldozeri</item>
    <item>akulokomotiva</item>
    <item>alati razni</item>
    <item>autocisterna</item>
    <item>automjesalica</item>
    <item>bageri</item>
    <item>betonare</item>
    <item>buldozeri</item>
    <item>busace garniture</item>
    <item>crpke za mort</item>
    <item>damper</item>
    <item>dizalica</item>
    <item>Double brum</item>
    <item>dozatori i separacije agregata</item>
    <item>drobilice</item>
    <item>dodavac vibracijski</item>
    <item>dozeri</item>
    <item>dozer-utovarivac</item>
    <item>eksploziv i upaljači</item>
    <item>finiser</item>
    <item>glodalica</item>
    <item>grejder</item>
    <item>hidraulicki cekic</item>
    <item>japaneri</item>
    <item>jezevi i vibrojezevi</item>
    <item>jumbo</item>
    <item>kamion</item>
    <item>kompaktori</item>
    <item>kompresor</item>
    <item>krtica</item>
    <item>labudica</item>
    <item>mijesalice</item>
    <item>mlin</item>
    <item>motorna pila</item>
    <item>nabijaci</item>
    <item>plovna jaruzala</item>
    <item>pumpa za beton</item>
    <item>pumpa za inektiranje</item>
    <item>skela pomicna</item>
    <item>skrejper</item>
    <item>stroj za nabacivanje morta</item>
    <item>stroj za pranje kamene sitnezi</item>
    <item>strojevi za armaturu</item>
    <item>tildozeri</item>
    <item>transportno sredstvo</item>
    <item>uredaji za zavarivanje</item>
    <item>utovarivac</item>
    <item>valjci i vibrovaljci</item>
    <item>vedricar</item>
    <item>vibratori</item>
    <item>vibroploce</item>
    <item>vozila razna</item>
</string-array>
```

4.3.2. Neke općenite metode korištenja Android Studia

Za obavljanje akcija u Android Studiu korišten je jezik Java. U Javi radimo klase koje bi mogle obaviti tu akciju. Klase imaju ekstenziju .java. Prva klasa koja se otvori je *MainActivity.java* i pri otvaranju se pozivaju sve *override* metode koje su značajne kod proširenja klase sa *ActionBarActivity*. To je glavna klasa koja omogućava samo korištenje aplikacije. Tako pri kreiranju samog aktivitija bitno je imati metodu koja poziva sve ostale metode i reference za funkciju aplikacije. Takva *overrire* metoda se naziva *onCreate( )*. Tu je bitno postaviti naslov i povezati se sa XML file-om koji čita željeno sučelje. Osnovni prikaz jedne takve klase je dat u tablici 4.4., u ovom slučaju dio koda *MainActivity.java* klase.

Tablica 4.4. MainActivity.java

```
package com.example.daniela.gradevinskidnevnik;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;


public class MainActivity extends ActionBarActivity {

    TextView linkid;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle(R.string.app_name);
        setContentView(R.layout.activity_main);

        linkid = (TextView) findViewById(R.id.linkid);
    }

    public void onNoviProjekt(View view) {
        Intent i = new Intent(this, NoviProjekt.class);
        startActivity(i);
    }

    public void onPostojeciProjekti(View view) {
        Intent i = new Intent(this, PostojeciProjekti.class);
        startActivity(i);
    }
```

Po pravilnom pozivanju potrebnih stvari za izradu programa uvoze (*engl.import*) se određene klase kao što se vidi na samom primjeru u tablici 4.4. Vidimo da smo pozvali *TextView* preko njegovog id-a iz XML file-a. Taj *TextView* nam služi za skidanje pravilnika o pisanju građevinskog dnevnika u pdf obliku. Na jednaki način se pozovu i svi ostali *widgeti* koji su nam potrebni u određenim klasama za određen XML. Još je iznesen primjer prebacivanja iz jednog aktivitija u drugi. Za to nam je potrebno kreirati metodu koja će u sebi nositi taj postupak stvaranja namjere (*engl.intent*) i prebacivanja u drugi aktiviti. Kako se metoda naziva, biti će objašnjeno u drugom poglavlju.

Već smo spomenuli da pri klikom na botun za odabir datuma se otvara *dialog*. Jednostavan kod koji čita automatski datum i u kojem programer izabire način ispisivanja samog datuma je dan u tablici 4.5.

Tablica 4.5. Metoda izbora datuma

```
public void onDatum(View view){

    final Calendar c = Calendar.getInstance();
    int mYear = c.get(Calendar.YEAR);
    int mMonth = c.get(Calendar.MONTH);
    int mDay = c.get(Calendar.DAY_OF_MONTH);

    DatePickerDialog ddp = new DatePickerDialog(this, new
DatePickerDialog.OnDateSetListener() {

        @Override
        public void onDateSet(DatePicker view, int year, int monthOfYear,
                              int dayOfMonth) {

            Button NadnevakUpisa;
            NadnevakUpisa=(Button)findViewById(R.id.NadnevakUpisa);


            int mYear = year;
            int mMonth = monthOfYear;
            int mDay = dayOfMonth;
            // TODO Auto-generated method stub
            NadnevakUpisa.setText(new StringBuilder()
                    .append(mDay).append("-").append(mMonth +
1).append("-")
                    .append(mYear).append(" "));
        }
    }, mYear, mMonth, mDay);
    ddp.show();
}
```

4.3.3. Spremanje i čitanje podataka

U ovom diplomskom radu biti će i predočeno kako su spremljeni podaci. Za prvu verziju, potrebno je stručno razraditi cijelo spremanje podataka. Za spremanje podataka iz klase *NoviProjekt.java* potrebno je napraviti novu klasu koju smo nazvali *modelPodataka.java* (tablica 4.6.) Naravno, nazivanje klasa ostaje na samom programeru. Ovdje je iznesen samo dio koda za podatke izvođača, a jednako se unosi kod i za sve ostale podatke.

Tablica 4.6. Dio koda spremanja podataka iz NoviProjekt.java

```
private String ImeIzvodjac = null;
private String AdresaIzvodjac = null;
private String MjestoIzvodjac = null;
      …
public String getImeIzvodjac() {
    return ImeIzvodjac;
}

public void setImeIzvodjac(String imeIzvodjac) {
    ImeIzvodjac = imeIzvodjac;
}

public String getAdresaIzvodjac() {
    return AdresaIzvodjac;
}

public void setAdresaIzvodjac(String adresaIzvodjac) {
    AdresaIzvodjac = adresaIzvodjac;
}

public String getMjestoIzvodjac() {
    return MjestoIzvodjac;
}

public void setMjestoIzvodjac(String mjestoIzvodjac) {
    MjestoIzvodjac = mjestoIzvodjac;
}
```

Da bismo potrebne stvari imali ispisane, koristili smo metodu *toString( )* koju smo *overajdali* sa pozivanjem *get* metoda koje smo prethodno kreirali u *modelPodataka.java* razredu (tablica 4.7.).

Tablica 4.7. toString( ) metoda

```
@Override
public String toString() {
    return  "Ime projekta:  " + getImeGradjevine() + "\n"
            +"_____"+"\n"
            + "Izvodjac:  " + getImeIzvodjac() + "\n"
            +"Adresa izvodjaca:  "+getAdresaIzvodjac()+"\n"
            +"Mjesto izvodjaca:  "+getMjestoIzvodjac()+"\n"
            +"_____"+"\n"

            + "Investitor:   " + getImeInvestitor() + "\n"
            +"Adresa investitora:  "+getAdresaInvestitor()+"\n"
            +"Mjesto investitora:  "+getMjestoInvestitor()+"\n"
            +"_____"+"\n"

            + "Ime gradjevine:  " + getImeGradjevine() + "\n"
            + "Adresa gradjevine:  " + getAdresaGradjevine() + "\n"
            +"Mjesto gradjevine:  "+getMjestoGradjevine()+"\n"
            +"_____"+"\n"
            + "Klasa gradjevinske dozvole:  "+getKlasa()+"\n"
            +"Urbroj:  "+getUbroj()+"\n"
            +"Datum izdavanja g.dozvole:
"+getGradjevinskaDozvolaDatum()+"\n"
            +"Mjesto izdavanja:  "+getGradjevinskaDozvolaMjesto()+"\n"
            +"_____"+"\n"
            +"Odgovorna osoba koja vodi gradnju:  "+
getOdgovornaOsoba()+"\n"
            +"Akt o imenovanju:  "+getOdgovornaOsobaAkt()+"\n"
            +"_____"+"\n"
            +"Strucni nadzor"+"\n"
            + "Nadzorni inzenjer 1:  "+getNadzorniInzenjer1()+"\n"
            +"Akt o imenovanju:  "+getNadzorniInzenjer1Akt()+"\n"
            +"Nadzorni inzenjer 2:  "+getNadzorniInzenjer2()+"\n"
            +"Akt o imenovanju:  "+getNadzorniInzenjer2Akt()+"\n"
            +"Nadzorni inzenjer 3:  "+getNadzorniInzenjer3()+"\n"
            +"Akt o imenovanju:  "+getNadzorniInzenjer3Akt()+"\n"
            +"Glavni nadzorni inzenjer:
"+getGlavniNadzorniInzenjer()+"\n"
            +"Akt o imenovanju:  "+getGlavniNadzorniInzenjerAkt()+"\n"
            +"Naziv i sjediste osobe koja obavlja strucni nadzor:  "
+getUpisiNazivISjediste()+"\n"
            +"_____"+"\n"
            +"Gradnja/izvodjenje radova"+"\n"
            +"Pocetak prijave:  "+getPocetakPrijave()+"\n"
            +"Pocetak gradnje:  "+getPocetakGradnje()+"\n"
  ; }
```

Tako u klasi *NoviProjekt.java* imamo kod koji sprema podatke upisane u njegovom XML file-u. Potrebne stvari za spremanje su ispisane u tablici 4.8. Pronašli smo potrebne reference i sada koristili *set* metode koje smo pozvali preko konstruktora *modelPodataka.java* razreda.

Tablica 4.8. Dio koda NoviProjekt.java klase

```
final modelPodataka MP = new modelPodataka();

final EditText mUpisiImeIzvodjacaEdit =
(EditText)findViewById(R.id.UpisiImeIzvodjaca);
final EditText mUpisiAdresaIzvodjacaEdit =
(EditText)findViewById(R.id.UpisiAdresaIzvodjaca);
final EditText mUpisiMjestoIzvodjacaEdit =
(EditText)findViewById(R.id.UpisiMjestoIzvodjaca);
   ...
if (mUpisiImeIzvodjacaEdit!=null){
    MP.setImeIzvodjac(mUpisiImeIzvodjacaEdit.getText().toString());
}
if (mUpisiAdresaIzvodjacaEdit!=null){
    MP.setAdresaIzvodjac(mUpisiAdresaIzvodjacaEdit.getText().toString());
}
if (mUpisiMjestoIzvodjacaEdit!=null){
    MP.setMjestoIzvodjac(mUpisiMjestoIzvodjacaEdit.getText().toString());
}
   ...

String textToSaveString = MP.toString();

try {
    String funkcija = MP.getImeGradjevine().toString().trim();

    FileOutputStream fileout=openFileOutput(funkcija, MODE_PRIVATE);
    OutputStreamWriter outputWriter=new OutputStreamWriter(fileout);
    outputWriter.write(textToSaveString.toString());
    outputWriter.close();
}catch (Exception e) {
          e.printStackTrace();
      }
```

Na isti način se spremaju podaci i iz svakodnevnog upisivanja lista građevinskog dnevnika. Potrebno je razraditi da pri spremanju i pozivanju metode, koja iz *Stringa* glavnog spremanja podataka ispiše listu, po konačnici bude spremljeno sve za određeni projekt, a u nastavku svih do tada ispisanih dnevnika.

Da bi se kreirala lista preko koje se dalje radi izbornik o daljnjem radu na određenom projektu dnevnika, programira se kod (tablica 4.9.) koji pronalazi sve spremljene file-ove. Korištene varijable su prethodno definirane u toj klasi.

Tablica 4.9. Metoda pronalaska file-ova

```
private void findAllFiles() {
    final String strAppDir = getApplicationInfo().dataDir + "/files/";
    Log.d(Constants.LOG_TAG, "Current directory: " + strAppDir);

    final File f = new File(strAppDir);
    mLstFiles = f.listFiles();

    StringBuffer temp = new StringBuffer();

    Log.d(Constants.LOG_TAG, "Size: " + mLstFiles.length);
    for (int i=0; i < mLstFiles.length; i++)
    {
        temp.append(mLstFiles[i].getName() + ' ' );
        mLstFileNames.add(mLstFiles[i].getName());
        Log.d(Constants.LOG_TAG, "FileName:" + mLstFiles[i].getName());
    }


    Toast.makeText(this, temp.toString(), Toast.LENGTH_LONG);
}
```

Područje koda gdje su definirane opcije izbornika kod otvaranja *menu*-a pri izboru projekta je prikazano tablicom 4.10. Kod je inače detaljniji, ali ovdje nije iznesen radi složenosti samog čitanja koda. Poziva se klasa *Constants.java* gdje se definira što će točno pisati na *menu*-u.

Tablica 4.10. Menu za izbornik projekta

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    //za sad je samo ovo moguce, pa cemo ovdje dodati
    menu.add(Constants.CONTEXT_NOVI_LIST);
    menu.add(Constants.CONTEXT_POSTOJECI_LIST);
    menu.add(Constants.CONTEXT_POGLEDAJ_DNEVNIK);
    menu.add(Constants.CONTEXT_ZASEBNI_DNEVNICI);
    menu.add(Constants.CONTEXT_ZAKLJUCI_DNEVNIK);

    menu.setHeaderTitle("Izbornik projekta");

}
```

U klasi *ActivityPrikazGradjevine.java* očitavamo podatke koje korisnik unese. Za takvo ispisivanje zaslužna je kreirana metoda (tablica 4.11.) sa svojim varijablama.

Tablica 4.11. Čitanje spremljenih podataka

```
private EditText mEditText = null;
...
mEditText = (EditText) findViewById(R.id.txtPodaci);
final String value = getIntent().getExtras().getString(Constants.DATA);
popuniText(value);
...
private void popuniText(final String fileName) {

    FileInputStream fileIn = null;
    StringBuffer buffer = new StringBuffer();
    try {
        fileIn = openFileInput( fileName);
        InputStreamReader InputRead = new InputStreamReader(fileIn);
        char[] inputBuffer = new char[100];

        int charRead;

        while ((charRead = InputRead.read(inputBuffer)) > 0) {
            // char to string conversion
            String readstring = String.copyValueOf(inputBuffer, 0,
charRead);
            buffer.append(readstring);
        }
        InputRead.close();

        mEditText.setText("");
        mEditText.setText(buffer.toString());
        mEditText.setEnabled(false);
    } catch (Exception e) {
        e.printStackTrace();
    }}
```

Zasebni dnevnici su spremljeni na drukčiji način tokom izvedbe ove aplikacije iz jednostavnog razloga što se nadopunjuju podaci prilikom unesenog novog zasebnog dnevnika, što nije bio slučaj u spremanju podataka novog projekta gdje se podaci ne nadopunjuju. Za spremanje u klasi *ZasebniGradevinskiDnevnici.java* korištena je metoda *SQL Database*. Tu je potrebno prebaciti unesene podatke u *Stringove*. Nadalje, treba kreirati posebnu klasu koju smo nazvali *DBHandlerZasebniPodaci.java* gdje se kreira tablica u metodi i pri stvaranju novih podataka i spremanja, pridodan je novi redak točno tim podacima u toj tablici. Dio koda koji je bitan za razumijevanje kao i za način ispisivanja tih podataka je iznesen iz klase *DBHandlerZasebniPodaci.java* (tablica 4.12.).

Tablica 4.12. DBHandlerZasebniPodaci.java

```
public class DBHandlerZasebniPodaci {

    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "zasebni.db";
    private static final String TABLE_PRODUCTS = "ZASEBNI";
    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_UPISIDIOGRADJEVINE =
"UpisiDioGradjevine";
    public static final String COLUMN_UPISIIZVODJAC = "UpisiIzvodjac";
    public static final String COLUMN_UPISISTRUCNINADZOR =
"UpisiStrucniNadzor";
    public static final String COLUMN_UPISIDATUMPOCETKARADOVA=
"UpisiNadnevakPocetakRadova";
    public static final String COLUMN_UPISIDATUMZAVRSETKARADOVA =
"UpisiNadnevakZavrsetakRadova";
     …
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL( "CREATE TABLE " + TABLE_PRODUCTS + " (" + COLUMN_ID + "
INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    COLUMN_UPISIDIOGRADJEVINE + " TEXT NOT NULL, " +
                    COLUMN_UPISIIZVODJAC + " TEXT NOT NULL, " +
                    COLUMN_UPISISTRUCNINADZOR + " TEXT NOT NULL, " +
                    COLUMN_UPISIDATUMPOCETKARADOVA + " TEXT NOT NULL, " +
                    COLUMN_UPISIDATUMZAVRSETKARADOVA + " TEXT NOT NULL
);"
    );
}
     …
public String getData() {
    String[] red = new
String[]{COLUMN_ID,COLUMN_UPISIDIOGRADJEVINE,COLUMN_UPISIIZVODJAC,

COLUMN_UPISISTRUCNINADZOR,COLUMN_UPISIDATUMPOCETKARADOVA,COLUMN_UPISIDATU
MZAVRSETKARADOVA};
    Cursor c
=database.query(TABLE_PRODUCTS,red,null,null,null,null,null);
    String rezultat = "";
    int iColumn=c.getColumnIndex(COLUMN_ID);
    int iDioGradj=c.getColumnIndex(COLUMN_UPISIDIOGRADJEVINE);
    int iIzv=c.getColumnIndex(COLUMN_UPISIIZVODJAC);
    int iSN=c.getColumnIndex(COLUMN_UPISISTRUCNINADZOR);
    int iPR=c.getColumnIndex(COLUMN_UPISIDATUMPOCETKARADOVA);
    int iZR=c.getColumnIndex(COLUMN_UPISIDATUMZAVRSETKARADOVA);


    for(c.moveToFirst(); !c.isAfterLast();c.moveToNext()){
        rezultat=rezultat + c.getString(iColumn)+ ". " +
c.getString(iDioGradj)  + "  \n"
                +" Izvodjac:  "+c.getString(iIzv) + "  \n" +
                " Strucni nadzor:  "+  c.getString(iSN)+"  \n"
                +" Pocetak gradnje:  " + c.getString(iPR)+ "  \n"+
                 " Zavrsetak gradnje:  "  +c.getString(iZR)+
                "  \n" + "\n";
    }

    return rezultat;
}
}
```

Nova klasa nam omogućava čitanje tih podataka i točno je pridodana svom aktivitiju. Klasa u kojem se čitaju uneseni podaci je *PogledZasebnihGD.java*. Taj kod je prikazan tablicom 4.13.

Tablica 4.13. Metoda pronalaska upisanih podataka zasebnih dnevnika

```
public class PogledZasebnihGD extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pogled_zasebnih_gd);

        TextView SQLispisi = (TextView) findViewById(R.id.SQLispisi);
        DBHandlerZasebniPodaci info = new DBHandlerZasebniPodaci(this);
        info.open();
        String data = info.getData();
        info.close();
        SQLispisi.setText(data);
    }
     …
}
```

4.3.4. Programska izvedba grafičkog prikaza podataka

Svaki prozor koji korisnik vidi na aplikaciji ima svoj XML u kojem je definiran njegov izgled. Ako se na njemu događa neka akcija povezan je sa određenom klasom. Cijela aplikacija je rađena na jednake načine, a ovdje će biti prikazan jedan XML i to aktiviti *activity_novi_projekt.xml*, ne cijeli (tablica 4.14.).

Tablica 4.14. Primjer korištenja XML-a

```
<RelativeLayout
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:background="@drawable/shape_background"
    tools:context="com.example.daniela.gradevinskidnevnik.NoviProjekt">

    <ScrollView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:fillViewport="true">

        <RelativeLayout
            android:id="@+id/layoutnoviprojekt"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:orientation="vertical">

            <View
                android:id="@+id/viewblue"
                android:layout_width="fill_parent"
```

```
                android:layout_height="32dp"
                android:background="#ff0c0a37" />

            <TextView
                android:id="@+id/tV"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentLeft="true"
                android:layout_alignParentStart="true"
                android:layout_alignParentTop="true"
                android:layout_marginLeft="8dp"
                android:layout_marginTop="3dp"
                android:text="Novi projekt"

android:textAppearance="?android:attr/textAppearanceLarge"
                android:textColor="#fffcff13" />

    …

            <TableLayout
                android:id="@+id/IzvodjacId"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentLeft="true"
                android:layout_alignParentStart="true"
                android:layout_below="@+id/IzvodacCrta"
                android:stretchColumns="1">
                <TableRow
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">

                    <TextView
                        android:id="@+id/ImeIzvodjaca"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_column="0"
                        android:layout_marginLeft="5dp"
                        android:text="Ime:"

android:textAppearance="?android:attr/textAppearanceMedium" />

                    <EditText
                        android:id="@+id/UpisiImeIzvodjaca"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:layout_column="1"
                        android:layout_marginLeft="3dp"
                        android:layout_marginRight="8dp"
                        android:inputType="text"
                        />

                </TableRow>

                <TableRow
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">

                    <TextView
                        android:id="@+id/AdresaIzvodjaca"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
```

```
                        android:layout_column="0"
                        android:layout_marginLeft="5dp"
                        android:text="Adresa:"

android:textAppearance="?android:attr/textAppearanceMedium" />

                    <EditText
                        android:id="@+id/UpisiAdresaIzvodjaca"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:layout_column="1"
                        android:layout_marginLeft="3dp"
                        android:layout_marginRight="8dp"
                        android:inputType="text"/>

                </TableRow>

                <TableRow
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">

                    <TextView
                        android:id="@+id/MjestoIzvodjaca"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_column="0"
                        android:layout_marginLeft="5dp"
                        android:text="Mjesto:"

android:textAppearance="?android:attr/textAppearanceMedium" />

                    <EditText
                        android:id="@+id/UpisiMjestoIzvodjaca"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:layout_column="1"
                        android:layout_marginLeft="3dp"
                        android:layout_marginRight="8dp"
                        android:inputType="text"/>

                </TableRow>

            </TableLayout>
    …

            <Button
                android:id="@+id/BotunZaSpremiNoviProjekt"
                android:onClick="onSpremiNoviProjekt"
                android:layout_column="1"
                android:layout_marginLeft="5dp"
                android:background="@drawable/button_shape"
                android:text="Spremi novi projekt"
                android:textColor="#dde643"
                android:textSize="20sp"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"/>
        </RelativeLayout>
    </ScrollView>
</RelativeLayout>
```

Prvo je potrebno definirati *layout*-e, vrstu rasporeda svih *itemsa*. Da bi korisnik mogao listati cijeli prozor na ekranu uređaja dodaje se *ScrollView*. Kreiran je i *TableLayout* koji omogućava unos bilo kojih *widgeta* na pravilan način. Za lakše postavljanje željenih *itemsa* po *layoutu* lako se pozove neka određena linija koja sačinjava cijeli kod tog *itemsa*, kao na primjer koliko će biti odmaknuto lijevo, desno, ispod kojeg drugog *itemsa* će stajati novi i na kojoj udaljenosti, i dr. Svakom tom *itemsu* je pridodano njegovo identifikacijsko ime, id, preko kojeg se pozove u .java klasi. *Widgetima* kao što je botun, pridodaje se i određena funkcija, odnosno ime metode kada se klikne taj botun. To ime se koristi u .java klasi i točno tako se mora nazvati metoda da bi se pridodala željena metoda tom botunu.

5. ZAKLJUČAK

Android je operacijski sustav koji je u potpunosti promijenio industriju mobilnih uređaja i doživio ogroman uspjeh. Činjenica koja omogućava Androidu tako brz napredak je dobra organizacija, koja iskorištava sustav otvorenog koda. Ovakav pristup ostavlja puno prostora za inovativnost i kreativnost, što je u današnjem svijetu nužno za uspjeh.

Za izradu kvalitetne aplikacije nije dovoljno samo poznavati Java programski jezik i android sučelje, treba voditi računa o tome da su mobilni uređaji ograničeni u svojoj memoriji i brzini izvedbe aplikacije, pa programer mora paziti da aplikacija zauzima što manje memorije i da ima samo one funkcionalnosti koje su joj zbilja potrebne. Također, treba u obzir uzeti činjenicu da se radi o uređajima koji se razlikuju po veličini ekrana, fizičkim tipkama i slično. Dizajn aplikacije mora biti što jednostavniji, a atraktivan i zanimljiv korisniku. Dakle, prije kreiranja aplikacije treba dobro proučiti zahtjeve koje aplikacija mora ispuniti.

Građevinski dnevnik je nužan za vođenje bilo kojeg projekta. Sve mora biti čitljivo i jasno napisano te razumljivo investitoru. Upisuju se osnovni podaci, kao i podaci koji se svaki dan ponavljaju, kao datum, ime građevine, struktura zaposlenih i struktura strojeva itd. Bitno je što se on treba voditi svaki dan i što najmanje dvije osobe moraju svakodnevno ispisivati i kontrolirati upisano.

Tema diplomskog rada je razviti ideju po kojoj građevinski dnevnik bi bio u funkciji za unošenje podataka i njegovo cijelo vođenje u elektronskom obliku. Iznijeto je sve što je do sada napravljeno, kodovi i izgled sučelja. Pri suradnji sa stručnjakom u programiranju, aplikacija se brzo može učiniti kao prototipom i dati korisniku na njegovo korištenje. Za cijeli kraj programskog koda potrebno je detaljno razriješiti ostatak programiranja ove moguće aplikacije i pri tom pripaziti na trenutno postavljeni zakon budućih korištenja ovakve aplikacije na elektronskom uređaju.

Tehnologije koje su korištene pri izradi aplikacije su razvojna okolina Android Studio, programski jezik Java te jezik XML. Aplikacija je testirana pomoću virtualnog stroja na mobilnom uređaju.

Najzahtjevniji dio bio je samo spremanje podataka pri čemu se dalje treba surađivati sa programerskim inženjerom. Zahvaljujući otvorenosti programskog koda programerima je omogućen uvid u već postojeće aplikacije i primjerke koda. Na internetu je moguće pronaći primjerke različitih aplikacija i dijelova koda koji su upotpunjeni sa komentarima drugih programera.

Osnovna prednost u odnosu na dosadašnje vođenje građevinskog dnevnika je lakše upisivanje potrebnih podataka koji se ne ponavljaju svaki dan što je izvođaču predstavljalo nepotreban posao. Aplikacija treba jednostavno automatski unijeti neke potrebne podatke i odmah ažurirati ostalim potrebnim zaposlenicima. Također treba kontrolirati redovito vođenje građevinskog dnevnika kao i njegovu kontrolu. A uz to, dnevnik se može voditi i na licu mjesta jer je elektronski uređaj lako prenosiv.

LITERATURA

[1] Head First Java (http://www.headfirstlabs.com/books/hfjava/)

[2] Head First Android Development (http://www.it-ebooks.info/book/644/)

[3] Java for programmers
(http://www.deitel.com/Books/Java/JavaforProgrammers/tabid/3416/Default.aspx)

[4] Programiranje u Javi (Marko Čupić)

[5] http://www.wienerberger.hr/gra%C4%91evinski-dnevnik.html?lpi=1366066313798

[6] http://www.w3schools.com/xml/default.asp

[7] http://www.geopolis.hr/gradjevinska-dozvola.htm

[8] http://narodne-novine.nn.hr/clanci/sluzbeni/2009_01_7_171.html

[9] Operacijski sustav android (dr.sc. Josip Lorincz)

[10] predavanja Izv.prof..dr.sc. Nives Ostojić Škomrlj, dipl. ing. građ

[11] http://www.hkis.hr/Upload/Documents/Vijesti/2014-3-7_Pravilnik_nadzor_dnevnik.pdf

[12] https://gov.hr/moja-uprava/stanovanje/izgradnja-i-obnova-kuce/gradjevinska-dozvola/338

SAŽETAK

Tema ovog rada je izrada ideje za buduću aplikaciju vođenja građevinskog dnevnika na elektronskom obliku. Prvo je izvedena sama teorija, kako teorija građevinskog dnevnika, tako i teorija izabrane programerske okoline Android Studio. Sama ideja je posebno razrađena, ispisana i predočena u ovom radu. Prikazano je sučelje idejne aplikacije, bitne značajke kao i neki važni kodovi koji su činili da se može ideja razvijati u pravom smislu. Daljnja suradnja sa programerskim inženjerom može dati dobar rezultat ove idejne aplikacije.