Univerzitet u Beogradu

Matematički fakultet

Master rad

Modelovanje Android aplikacije korišćenjem projektnih i arhitekturalnih uzoraka na Android platformi

Vladimir Milošević
broj indeksa: 1013/2010

1. Uvod

Aplikacija o kojoj se govori u ovom radu je *White label* [1] proizvod koji omogućava umetnicima, prvenstveno muzičarima, da na jednostavan način predstave i približe svoj rad njihovim slušaocima preko Android i iOS aplikacija. Umetnici unose sadržaj koji žele preko CMS-a koji će prilikom narednog učitavanja aplikacije biti vidljiv krajnjem korisniku.

Klijent definiše kategorije i sadržaj koji hoće da predstavi korisniku pri čemu taj sadržaj može biti tekst, slika i audio zapis. Takođe, klijent može da pošalje notifikaciju kranjem korisniku koja će se pojaviti na uređaju bez obzira na to da li je aplikacija pokrenuta (eng. *Push notification*).

Tehnologije koje su korišćene za izradu ovog projekta su Yoghurt CMS, PHP & MySql, Android SDK, iOS SDK.

1.1. Ideja i cilj projekta

Ideja ovog projekta je umetnicima ponuditi komercijalnu aplikaciju, odnosno portal, preko kojeg će na relativno jeftin, brz i jednostavan način doći do svojih slušaoca koji sve više koriste prenosive uređaje i socijalne mreže.

Klijentu je potrebno omogućiti jednostavnu izmenu sadržaja i to što pre dostaviti krajnjem korisniku. Dok je krajnjem korisniku neophodno predstaviti intuitivnu i dobro dizajniranu aplikaciju koju će moći da koriste na svojim prenosivim uređajima.

Zbog sve veće dominacije Android i iOS operativnih sistema na tržištu prenosivih uređaja neophodno je napraviti dve aplikacije za ove platforme koje će izgledati i omogućiti kranjem korisniku relativno slično iskustvo.

[1] White label proizvod je proizvod koji jedna kompanija nudi svojim klijentima tako da ti klijenti mogu prilogoditi i preimenovati (eng. rebrand) taj proizvod kako bi odgovarao njihovim potrebama. [1]

2. Zahtevi

2.1. Poslovni zahtev

2.1.1. Koncept

- Umetnici kreiraju sadržaj.
- Korisnici aplikacija treba da dožive umetnika na novi način, i moraju biti nagrađeni ekskluzivnim sadržajem pošto mobilne aplikacije nude nivo intime koje ostale digitalne forme ne mogu da dostignu.
- Sadržaj mora biti fluidan i čitljiv

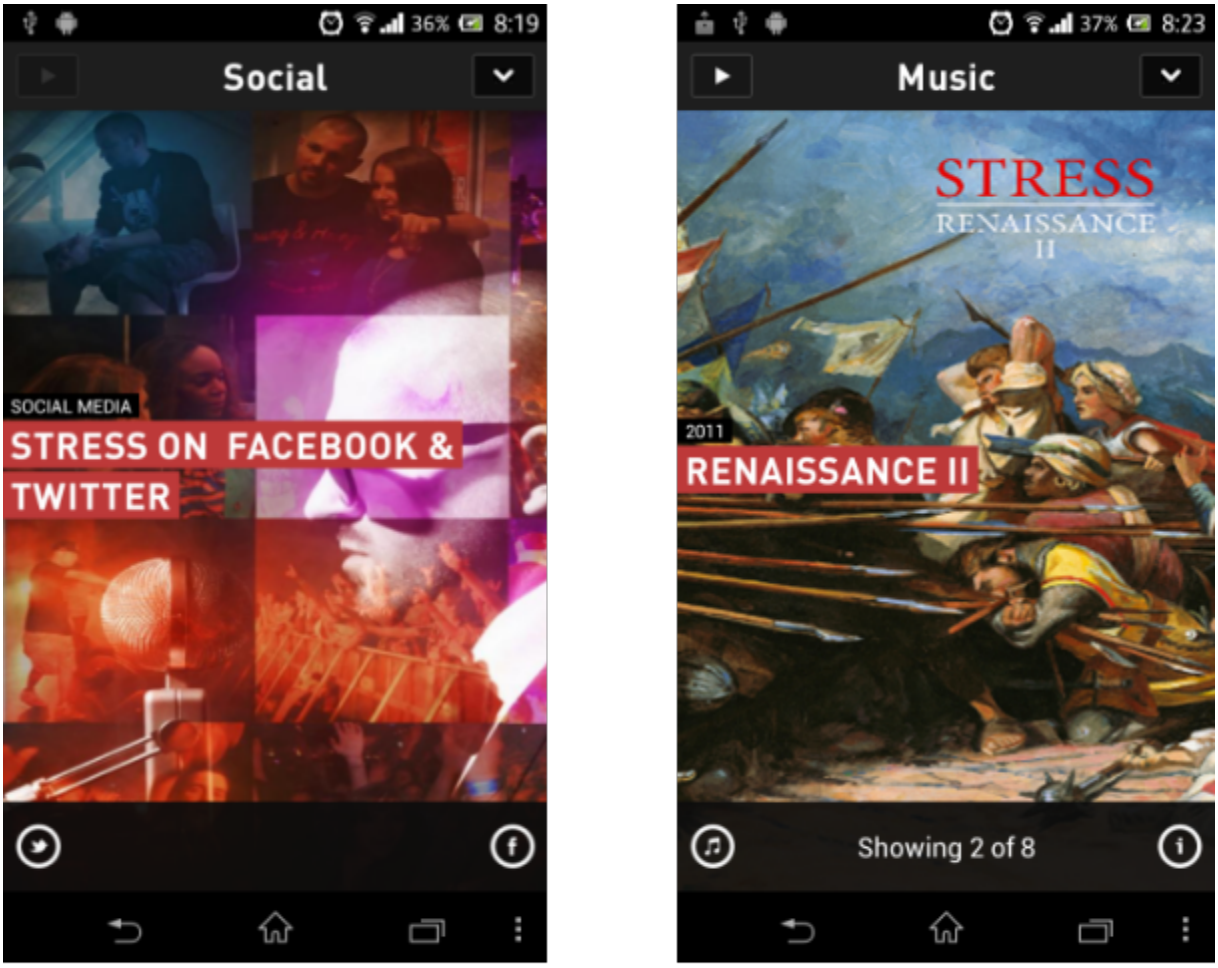
2.2. Zahtevi za dizajn

Sa manjom veličinom ekrana, skrolovanje je najefikasniji način navigacije na prenosivim uređajima. Treba napomenuti da prenosivi uređaj nije, niti treba da bude smanjeni desktop računar tako da je neophodno primeniti druge tehnike dizajniranja aplikacija kako bi se krajnjem korisniku omogućilo da na jednostavan način konzumira željeni sadržaj.

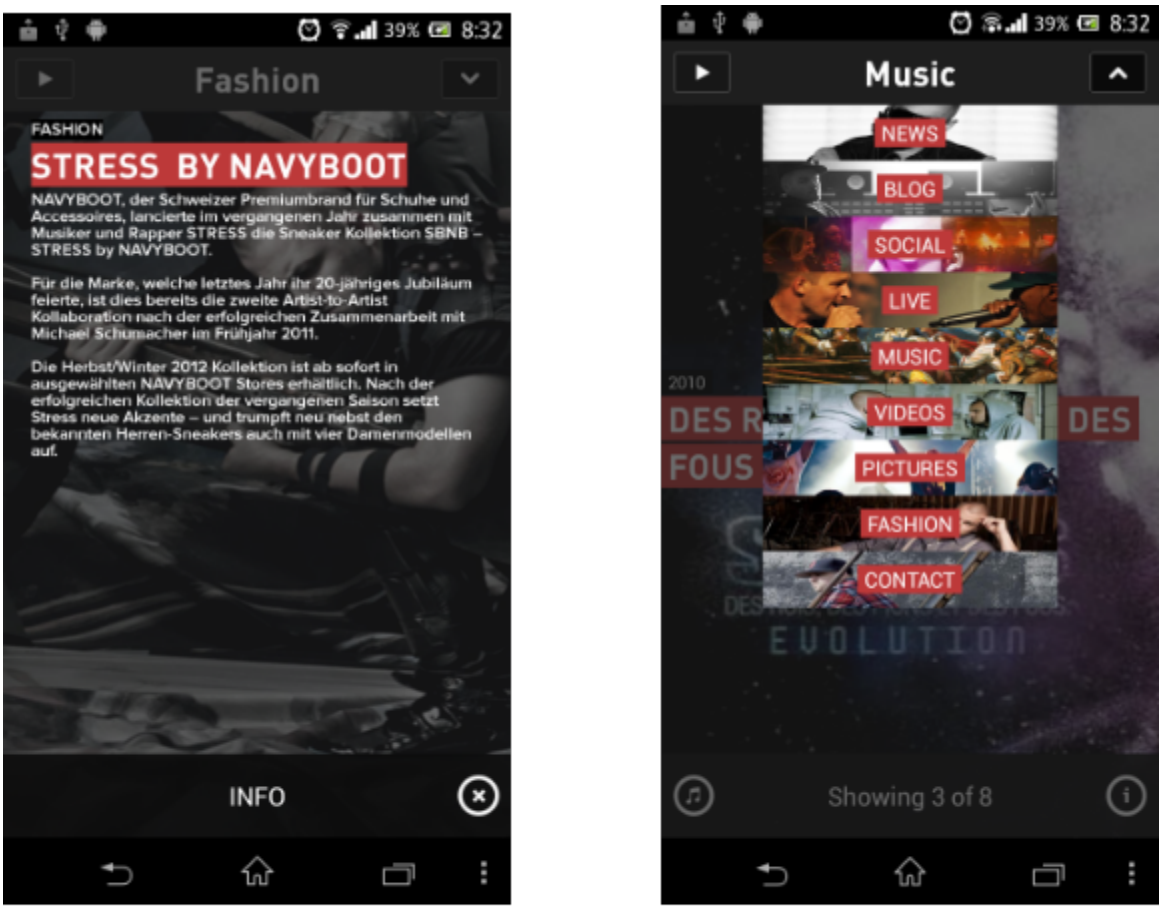
Grupisanjem sadržaja prvo u vertikalne, potom i horizontalne strane, korisnici na jednostavan način mogu da dožive sadržaj. Vertikalne strane predstavljaju različite kategorije. Predložene kategorije su Koncerti, Albumi, Biografija, Socijalne mreže... Korisnici vertikalnim skrolovanjem menjaju kategorije, dok se kroz same stranice unutar kategorije kreće horizontalnim skrolovanjem.

Kako bi se poboljšalo korisničko iskustvo za konzumiranje vizuelnog sadržaja, koriste se slike koje zauzimaju celu površinu ekrana. Za lakši pristup svakoj od kategorija dodaje se dugme na koje će korisnik jednim klikom otvoriti transparentni navigacijski panel koji će u sebi sadržati slike sa nazivima kategorija. Klikom na svaku od kategorija aplikacija će odvesti korisnika do tražene strane. U svakom trenutku korisnik može da kontroliše muziku preko “Plej” dugmeta koji se nalazi u navigacijskoj traci. Muzika se čuje sve dokle god je korisnik unutar aplikacije ili dok ne klikne na dugme za pauzu.

Kako bi se korisnik informisao na kojoj je trenutno strani, u navigacijskoj traci je neophodno ispisati naziv kategorije. Tekst treba da se nalazi na sredini trake.



Slika 2.1: Izgled nekih od glavnih ekrana aplikacije



Slika 2.2: Primer info ekrana i navigacijske trake

2.3. Funkcionalni zahtevi

Za korisnika:

- NAVIGACIJASKA TRAKA: Odražava trenutnu lokaciju, kontroliše muziku i omogućava direktnu navigaciju
- INFO: Prikazuje transparentni panel sa tekstom i linkovima.
- PLEJ DUGME: Otvara dodeljen audio/video snimak

Za umetnika:

- YOGHURT CMS: Omogućava umetniku da definiše kategorije, kreira strane, dodeljuje slike, audio i video zapise i određuje redosled kategorija i strana.

2.4. Tehnički zahtevi

Usled sve većeg udela u tržištu prenosivih uređaja Android i iOS operativnih sistema neophodno je napraviti aplikacije za iOS i Android platforme koristeći odgovarajući SDK. Aplikacije moraju da budu dostupne za što veći broj uređaja tako da je neophodno podržati Android API verzija 8 i veća, dok je minimalna verzija iOS operativnog sistema koja mora da bude podržana verzija 4. Razlozi za podržavanje Android API verzije 8 i iOS 4 operativnih sistema je to što aplikacija mora da podržava i notifikacije, a one su podržane tek od gore navedenih verzija ovih operativnih sistema. [2][3]

Zbog postojeće infrastrukture i prethodno napravljenog Yoghurt CMS-a (PHP, Symphony, MySql) i Push Notification Servera (ubuduće PNS) koristiće se Apache i MySql u *nix okruženju za serverski deo aplikacije.

3. Arhitektura aplikacije

U ovom delu rada će biti reči o rešenjima koja su korišćena za izradu PNS-a i Android aplikacije. Za izradu PNS-a je korišćen PHP i MySql, dok je za izradu Android aplikacije korišćen Android SDK[2].

3.1. Android platforma

Android je operativni sistem otvorenog koda, prvenstveno namenjen za prenosive uređaje osetljive na dodir, kao što su pametni telefoni i tableti. Baziran je na Linuks jezgru i trenutno postoje implementacije za ARM i Intel x86 arhitekture procesora. [10]

Pisanje aplikacija za Android OS je moguće u programskom jeziku Java (koristeći Android SDK) i jezicima C/C++ (Android NDK[3]). Kod ispisan u programskom jeziku Java se izvršava u virtuelnoj mašini pod nazivom Dalvik. Dalvik VM je implementacija Java virtualne mašine sa dodatnim paketima specijalizovanim za prenosive uređaje. Dalvik koristi JIT kompajler i prevodi fajlove sa ekstenzijom .class u fajlove sa ekstenzijom .dex. Kada se Android aplikacija iskompajlira dobija se fajl sa .apk ekstenzijom kojeg je moguće instalirati na bilo koji Android uređaj. [11]

Svaka Android aplikacija se izvršava u izolovanom (eng. sandbox) okruženju. To znači da je svaka aplikacija izolovana od svih drugih aplikacija na uređaju i da ne može da pristupa resursima uređaja bez eksplicitne dozvole korisnika. To je obezbeđeno tako što svaka aplikacija dobija zaseban korisnički ID (eng. User ID, preuzet iz Linuksa). Svaka aplikacija dobija zasebnu virtualnu mašinu što obezbeđuje još jedan nivo izolovanosti. Takođe, svaka aplikacija se izvršava u odvojenom Linuks procesu. Taj proces se pokreće svaki put kada bilo koji deo aplikacije mora da se izvrši, nakon čega se taj proces gasi kada više nije neophodan, ili kada sistem zahteva memoriju za drugu aplikaciju. [6]

Svaka Android aplikacija može sadržati četiri osnovne komponente koje pruža operativni sistem. [12] Te komponente su:

---

[2] SDK - (eng. Software Development Kit) deo Android API-ja koji sadrži neophodne biblioteke i alate za razvoj Android aplikacija. Pisan je za programski jezik Java. [10]

[3] NDK - (eng. Native Development Kit) deo Android API-ja koji se koristi za razvoj procesorsko i memorijski zahtevnijih aplikacija. Piše se u programskom jeziku C++ i najčešće se koristi za grafički zahtevne aplikacije. [11]

1. Aktivnost (eng. Activity) - klasa koja predstavlja jedan ekran aplikacije. Nju je moguće pokrenuti na više načina pri čemu je jedan od njih i samo pokretanje aplikacije iz glavnog menija. Svaki ekran koji postoji u aplikaciji mora naslediti klasu Aktivnosti.

2. Servis (eng. Service) - je komponenta koja se izvršava u pozadini. Ona nema korisnički interfejs i najčešće se koristi za neke procese koji se dugo izvršavaju. Ti procesi mogu biti na primer slušanje muzike dok je neka druga aplikacija u prvom planu, ili sinhronizacija aplikacije sa serverom dok aplikacija nije pokrenuta. Servise je moguće pokrenuti na nekoliko načina, a jedan je iz klase Aktivnosti sa kojom može i da komunicira.

3. Dobavljači sadržaja (eng. Content providers) - su klase koje služe da drugim aplikacijama izlože podatke aplikacije u kojoj se ti dobavljači sadržaja koriste.

4. Prijemnici poruka (eng. Broadcast receivers) - služe da prihvate poruke koje mogu slati druge aplikacije ili čak sam sistem. Te poruke mogu da idu od sistemskih događaja kao što je uključivanje telefona ili prijem SMS poruke do kraja puštanja nekog audio fajla.

Struktura Android aplikacija

Struktura većine Android aplikacija se bazira na MVC arhitekturalnom obrascu.

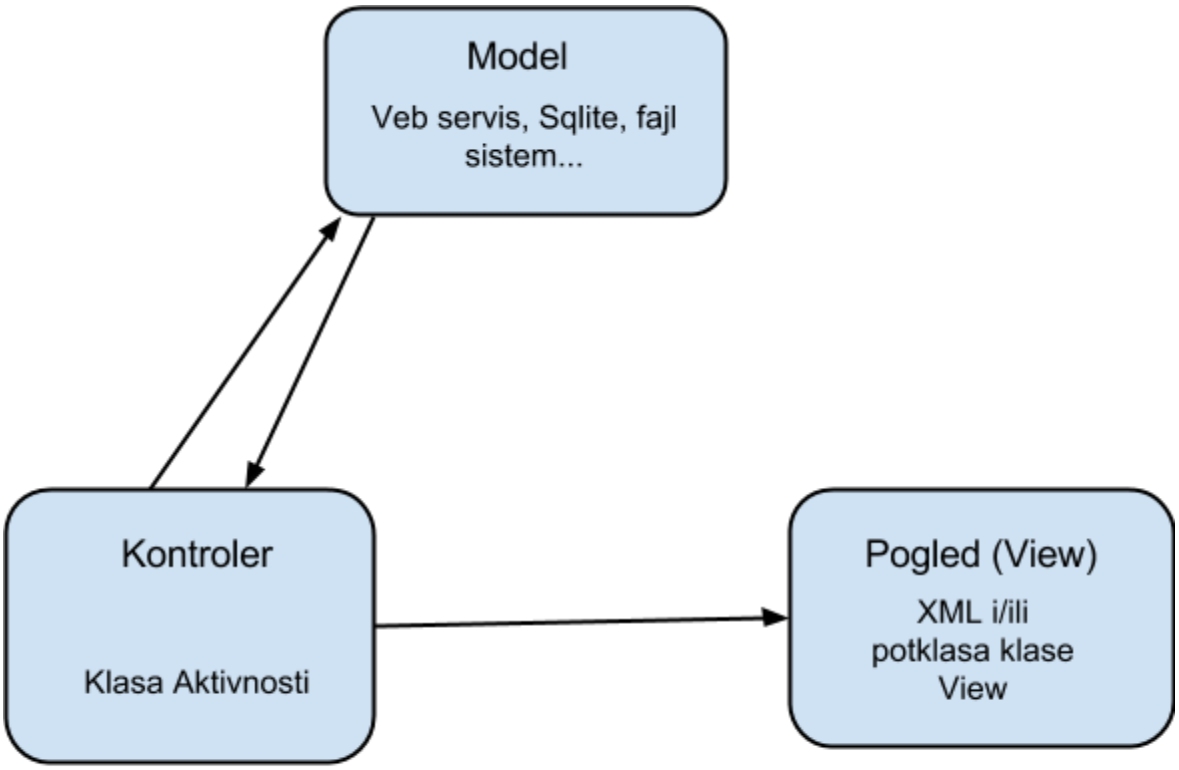
U sloju Modela se najčešće nalaze klase ili paketi klasa koje služe za dohvatanje podataka sa interneta. Ti podaci se najčešće dohvataju sa CMS servera koji su posebno pravljeni za tu Android aplikaciju, ali je sve češći slučaj da, pored rastućeg uticaja socijalnih mreža i blogova, ti podaci se mogu dohvatati koristeći SDK ili API-je koje većina najpopularnijih socijalnih mreža i pruža. Kako bi se optimizovao prenos podataka najčešće se koriste RESTful Veb servisi.[5] Zbog potencijalno visoke cene prenosa podataka preko mreže, RESTful servisi su zgodni jer minimizuju količinu podataka koje je neophodno preneti sa servera na korisnikov uređaj. Podaci se najčešće prenose u XML ili JSON (Java Script Object Notation) formatu i kasnije parsiraju i smeštaju na korisnikov uređaj. Tako prenete podatke moguće je sačuvati na nekoliko načina. Ukoliko je neophodna relaciona baza podataka, u tu svrhu se koristi Sqlite koji dolazi sa Android operativnim sistemom. Takođe, podatke je moguće čuvati na fajl sistemu u internoj memoriji uređaja, kao i na eksternoj kartici. Važno je napomenuti da su podaci (uključujući i

SQLite bazu podataka) koji se čuvaju u internoj memoriji samo vidljivi aplikaciji koja ih je kreirala i njima je moguće pristupiti iz drugih aplikacija samo korišćenjem klasa koje nasleđuju klasu dobavljača sadržaja (eng. Content provider) .

Sloj Pogled (eng. View) se najčešće definiše XML-om. Android SDK pruža programerima mogućnost definisanja komponenti (dugmad, liste, tabele...) u XML formatu. XML-om je takođe moguće opisati pored osnovnih atributa komponenti kao što su visina, širina, ravnanje i druge, složenije atribute kao što su boje, selektori, bitmape, teme... XML fajlovi se u toku kompajliranja aplikacije prevode u Java klase tako da je njima moguće pristupati i iz Java koda.

Pored XML fajlova, sam izgled je moguće i definisati iz Java koda. Osnovna klasa svih komponenti koje je moguće videti na ekranu je klasa View. Ukoliko je potrebno da se napravi sopstvena UI komponenta, nju je neophodno izvesti iz ove klase. Važno je napomenuti da je poželjno koristiti XML pri definisanju ekrana jer se na taj način lakše održava kod i pojednostavljuje se prilagođavanje za više različitih rezolucija i formata ekrana. Takođe, XML-om definisane komponente se brže iscrtavaju od komponenti definisanih u Java kodu.

Ulogu kontrolera u Android aplikacijama najčešće preuzimaju klase nasleđene iz klase Aktivnosti. Svaka Aktivnost ima svoj životni ciklus. Tokom životnog ciklusa potrebno je dohvatiti neophodne podatke i instancirati View koji će se koristiti u toj aktivnosti.



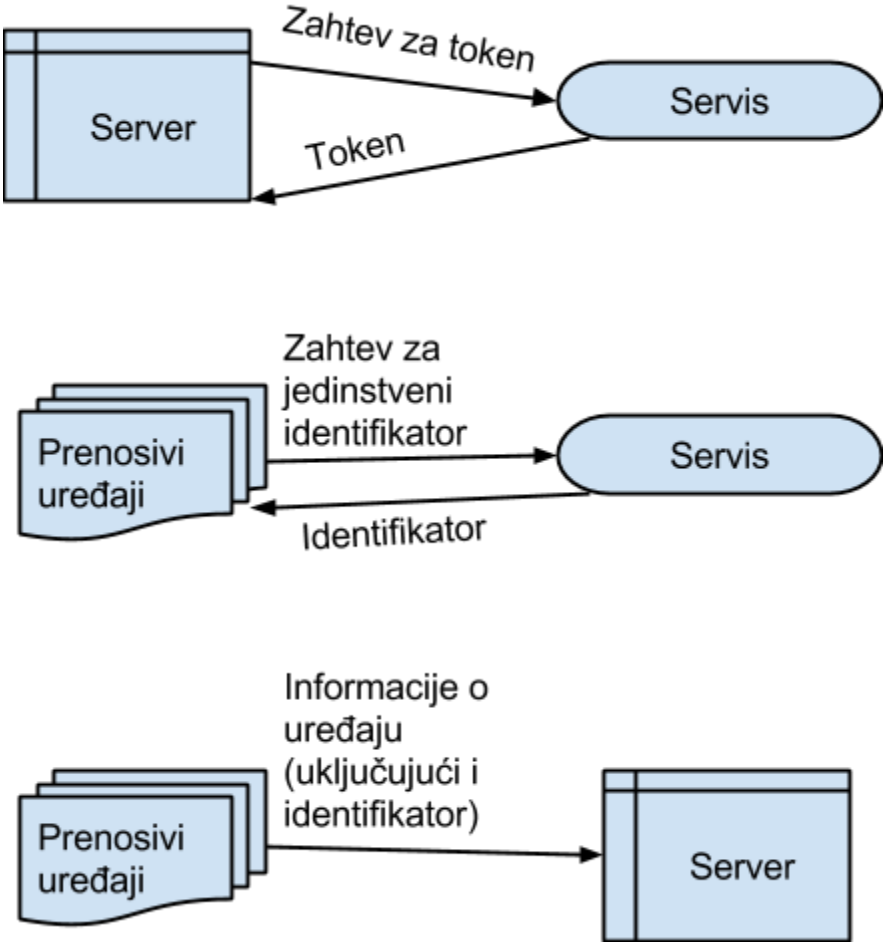
Slika 3.1: MVC prikaz Android aplikacije

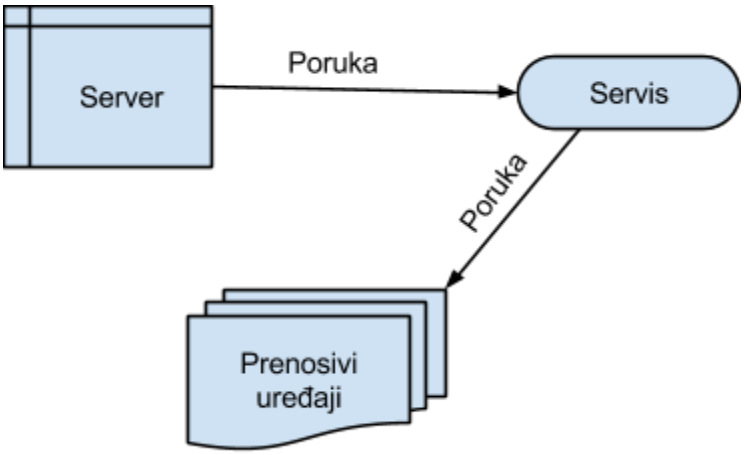
3.2 Arhitektura Push Notification Server-a

Notifikacije koje se “guraju” na uređaje (eng. Push notifications) su kratke poruke koje se šalju na prenosive uređaje bez potrebe da korisnik ili aplikacija proveravaju stanje na serveru sa koga žele da dobiju neku novost. Ovakav način slanja poruka na uređaje smanjuje potrošnju već ograničenih resursa na prenosivim uređajima kao što su, između ostalih, baterija i troškovi prenosa podataka (2G, 3G i 4G mreža). Ovakav način slanja poruka podržava većina trenutno aktuelnih operativnih sistema prenosivih uređaja (iOS, Android, Symbian, Windows Phone 7-8...).

Uz različite implementacije i načine definisanje formata poruka koje se šalju na uređaj, svi proizvođači operativnih sistema koji podržavaju ovaj tip notifikacija imaju zajednički način komunikacije između servera sa kog se gura i uređaja na koji se gura poruka. Server ne može direktno poslati poruku na uređaj, već se mora obratiti servisu koji pruža proizvođač operativnog sistema. Taj servis će obraditi poruku, proveriti njenu validnost i proslediti tu poruku krajnjem korisniku (uređaju). Kako bi uređaj primio i na kraju prikazao poruku, na njemu mora biti instalirana aplikacija koja će procesuirati poruku, u suprotnom poruka neće biti prikazana, a u nekim implementacijama neće ni biti poslata na uređaj. Kako bi se napravio server koji će gurati poruke neophodno je ispratiti 4 koraka (Slika 2):

1. Server mora dobiti autentifikacioni token od servisa za guranje poruka na uređaj
2. Uređaj na koji se šalje poruka mora se prijaviti servisu za guranje poruka od koga će dobiti token za primanje poruka, odnosno jedinstveni identifikator
3. Uređaj mora poslati serveru jedinstveni identifikator koji je dobio od servisa
4. Server koristeći jedinstveni identifikator se obraća servisu koji obrađuje poruku i prosleđuje je uređaju





Slika 3.2: Opis generičkog rešenja za PNS

Vremenom su se operativni sistemi, kao i servisi unapređivali tako da je pored teksta moguće poslati i sliku, link i ton. Uprkos tome postoje i ograničenja. Neka od njih su veličina poruke, frekvencija slanja, broj uređaja na koji je moguće poslati poruku i sl. i variraju od proizvođača.

Takođe je važno naglasiti da PNS služi da informiše aplikaciju da postoji promena na serveru na kom se nalaze podaci i da bi aplikacija trebala da zahteva te podatke direktno od servera, a ne iz same notifikacije. Glavni razlog za ovo je što dostavljanje notifikacije nije zagarantovano. Razlozi za nedostavljenje mogu biti da korisnik u nekom trenutku nema konekciju sa interenetom, ili je ugasio poruku, ili je isključio mogućnost notifikacije na samom uređaju...

4. Implementacija Android aplikacije

Poštujući smernice za razvoj Android aplikacija i u ovoj aplikaciji postoje tri jasno odvojena sloja iz MVC arhitekture, stoga će i opis implementacije biti podeljen u ova tri sloja.

4.1. Model

U sloj modela aplikacije se može svrstati nekoliko različitih Java paketa koji pored apstrakcije podataka imaju za svrhu da dohvate i obrade te podatke. Pošto je smisao aplikacije da u svakom trenutku ima ažurne informacije i multimediju, neophodno je bilo osmisliti optimalan način dohvatanja podataka i čuvanja istih zbog ograničenja koja se nalaze na mobilnim platformama (veličina dostupne memorije, postojanost internet konekcije, velika cena transporta podataka...). Pored toga, mora se uzeti u obzir da aplikacija mora da bude vrlo prilagodljiva jer ne treba da zavisi ni od jezika, države, vrste umetnosti kojom se umetnik (klijent) bavi...

4.1.1 Paket za dohvatanje podataka (eng. Downloader)

4.1.1.1 Manifest fajl

Prilikom pokretanja aplikacije, proverava se stanje podataka na CMS serveru (ukoliko postoji internet konekcija, u suprotnom se prikazuju podaci koji su bili prethodno dohvaćeni). Provera stanja obuhvata proveru verzije stanja (jedinstven broj verzije) i heš vrednost manifest fajla. Manifest fajl je tekstualni fajl u kome se nalaze putanje do svih JSON i multimedijalnih fajlova koji se koriste u trenutnoj verziji aplikacije kao i njihove CRC[4] vrednosti. Proverava se da li je svaki fajl iz liste već prethodno bio preuzet kao i da li nije promenjen u međuvremenu tako što se proverava njegova CRC vrednost. Na ovaj način se obezbeđuje da je svaki fajl ažuran, a da se pritom minimizuje mrežni protok. Provera se vrši upoređivanjem starog manifest fajla (koji je već bio preuzet u prethodnom ažuriranju aplikacije) i novog manifest fajla. Svi fajlovi koji postoje u

[4] eng. Cyclic Redundancy Check - Algoritam koji služi da detektuje male promene u blokovima podataka tako što računa ostatak sadržaja pri deljenju polinomom. [7]

starom manifest fajlu, a ne postoje u novom se brišu sa fajl sistema kako bi se korišćenje interne memorije, koja je vrlo ograničen resurs na mobilnim uređajima, svelo na minimum. Prilikom parsiranja novog manifest fajla generiše se lista fajlova koji trebaju da se preuzmu sa CMS servera.

```
0x95f89bab structure.json
0x9aa6f378 27.json
0x349f15ac 28.json
0x24f12cd0 Au_Poste_Snippet.mp3
0x24f12cd0 Elle_Snippet.mp3
0x24f12cd0 Animal_Life_Snipppet.mp3
```

Slika 4.1: Primer dela manifest fajla

4.1.1.2 Vešenitni menadžer za preuzimanje podataka

Generisana lista novih fajlova služi kao osnova za pokretanje višenitnog menadžera za preuzimanje podataka (eng. Multithreaded Download Manager[5]). U višenitnom menadžeru za preuzimanje podataka se definiše broj niti koje mogu da se pokrenu odjednom, dok se ostale niti stavljaju u red i čim neka od već pokrenutih niti prijavi višenitnom menadžeru da je završila preuzimanje započinje se novo preuzimanje. Ovaj broj niti zavisi od broja jezgara procesora koji se nalazi na uređaju, a skalu treba empirijskim metodama generisati u zavisnosti od potreba aplikacije.

Razlog za korišćenje ovakvog načina preuzimanja podataka u odnosu na serijski možda nije očigledan na prvi pogled. Naime, mobilne mreže, odnosno sami uređaji, imaju ograničen protok i ovo rešenje može da izgleda kao nepotrebno komplikovanje samog koda. Ali ovakav način preuzimanja može i nekoliko puta ubrzati preuzimanje ukoliko je u pitanju preuzimanje puno fajlova manje veličine. [13]

Najsporiji resurs na računarskim uređajima jeste mreža. Isto važi i za Android uređaje. Ujedno, samo ostvarivanje konekcije sa serverom i njeno zatvaranje, na Android uređajima, je deo preuzimanja nekog fajla na kome se gubi

[5] Ovaj deo aplikacije je kasnije prerađen u zasebnu biblioteku koju je moguće koristiti i za druge projekte. Ovde se nalazi modifikovana verzija koda te biblioteke koja zadovoljava potrebe ovog projekta

najviše vremena, tako da je logično da se i ostali resursi iskoriste dok traje ovo ostvarivanje konekcije korišćenjem više različitih niti.

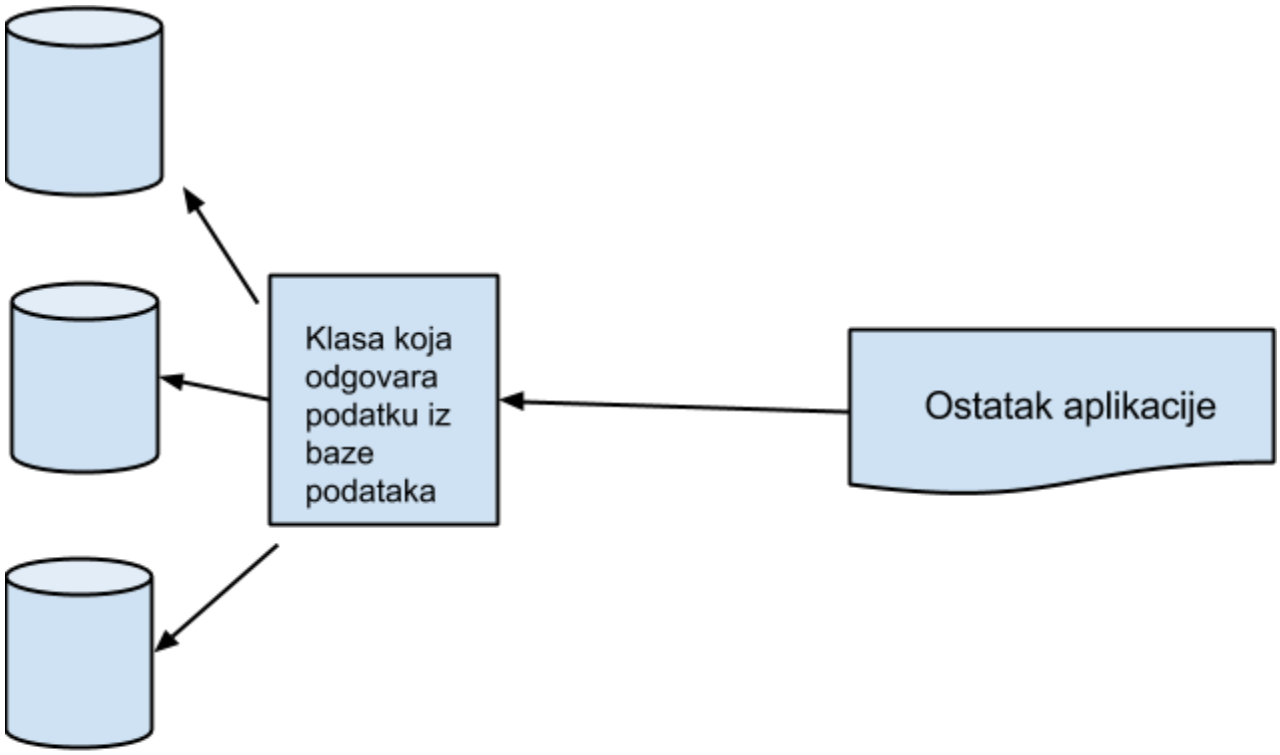
4.1.2 Apstrakcija modela podataka

Kada se završi dohvatanje svih neophodnih fajlova što ujedno znači i kada se završe i zaustave sve niti za dohvatanje fajlova započinje se nova, glavna aktivnost same aplikacije. Glavna aktivnost aplikacije (koja je definisana u fajlu HomeScreen.java), kao i ostale aktivnosti prilikom kreiranja parsiraju odgovarajuće JSON fajlove i njih koriste kao model po kome će prikazivati sadržaj korisnicima.

Pored JSON-a kao baze podataka koriste se, kao što je i ranije napomenuto, i veb servisi socijalnih mreža i blogova. Kako bi se olakšao pristup API-ju koji se dohvata sa socijalnih mreža ili parsiranje RSS-XML fajla sa bloga klijenta implementiran je i DAO obrazac. [8]

DAO obrazac se koristi da apstrahuje bazu podataka. Ideja je da se omogući aplikaciji da bude nesvesna (do neke mere) od baze podataka koja pohranjuje podatke. To se rešava tako što se definišu klase (ili interfejsi) za pristup podacima, i implementira se deo aplikacije koji će pristupati svakoj bazi podataka zasebno (baza može biti relaciona, objektna, hijerarhijska...). Svaka implementacija pristupa bazi podataka je odgovorna da kreira objekte koji odgovaraju već dogovorenim klasama ili interfejsa podataka. Često se korsti u ORM[6] framework-cima.

[6] ORM (eng. Object Relational Mapping) - je tehnika prevođenja relacionog modela podataka u objektni i obrnuto. Neki od poznatijih ORM framework-a su Hibernate i JPA u Javi, Entity Framework u .NET-u, Symphony u PHP-u...



Slika 4.2 - DAO patern

Na slici 4.2 se vidi osnovna ideja DAO paterna pri čemu valjkovi predstavljaju različite izvore podataka koji mogu da se mapiraju u klasu podatka koja će biti iskorišćena u ostatku aplikacije.

4.1.2.1 Konkretna implementacija DAO obrasca

U implementaciji ove aplikacije bazom podataka iz DAO obrasca možemo shvatati nekoliko različitih stvari. To mogu biti JSON fajlovi, veb servisi... Jedna od implementacija DAO obrasca se nalazi u delu aplikacije odgovornim za prikaz blog postova.

Većina trenutno aktuelnih blog CMS rešenja sadrži i RSS feed[7] tako da se taj RSS feed koristi kao izvor podataka za prikaz svih blog postova umetnika. Kao i sva ostala mrežna komunikacija, dohvatanje odgovarajućeg RSS feed-a se vrši u pozadinskoj niti. Prilikom okončanja rada niti za dohvatanje blog postova, kreira se lista jednostavnih Java objekata (POJO[8]) koji se kasnije učitavaju u sam grafički

[7] RSS feed (eng. Rich Site Summary) - predstavlja sumarni prikaz nekog veb sajta, najčešće bloga. Koristi standardizovan XML format.

[8] POJO (eng. Plain Old Java Object) - U bukvalnom prevodu Jednostavan stari Java objekat. Kolokvijalni izraz koji se često koristi među softver inženjerima koji se bave razvojem aplikacija baziranim na Java tehnologiji, a označava

prikaz liste (eng. ListView). Na ovaj način se može omogućiti jednostavnija implementacija nekog drugog RSS feed-a čiji se format može razlikovati od inicijalnog.

Blog klasa kao deo DAO obrasca (ujedno i POJO objekat):

objekat koji nema nikakvih drugih ograničenja do onih nametnutih samom sintaksom Java jezika. Pri tome se misli da klase koje kreiraju takve objekte ne sadrže neke posebne anotacije, niti nasleđuju ili implementiraju neke ne-POJO klase. Najjednostavniji primer je klasa koja je direktan naslednik Object klase.

Klasa Blog

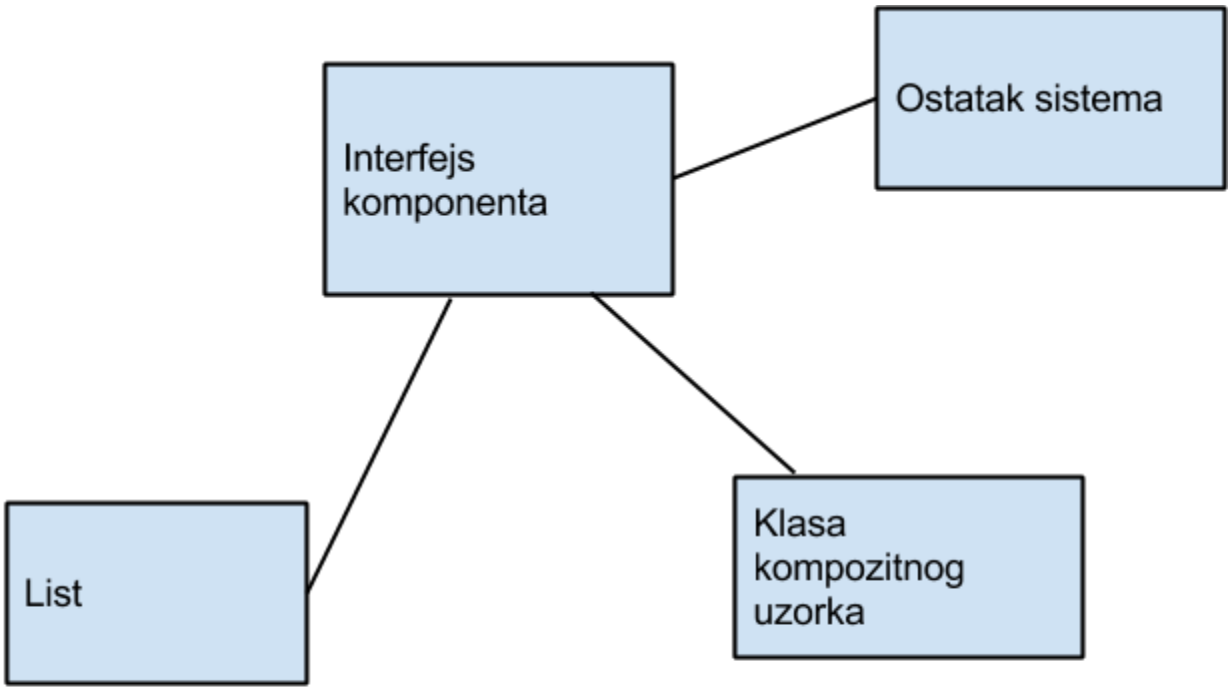
```
public class Blog {
	private String title;
	private String link;
	private String publishedDate;
	private String desc;

	public Blog(String title, String link, String publishedDate, String desc) {
		super();
		this.title = title;
		this.link = link;
		this.publishedDate = publishedDate;
		this.desc=desc;
	}
	public String getDesc() {
		return desc;
	}
	public void setDesc(String desc) {
		this.desc = desc;
	}
	public String getTitle() {
		return title;
	}
	public void setTitle(String title) {
		this.title = title;
	}
	public String getLink() {
		return link;
	}
	public void setLink(String link) {
		this.link = link;
	}
	public String getPublishedDate() {
		return publishedDate;
```

```
        }
        public void setPublishedDate(String publishedDate) {
                this.publishedDate = publishedDate;
        }
}
```

4.2 Sloj pogleda (eng. View layer)

Sloj pogleda u mobilnim aplikacijama je možda i najvažniji deo aplikacije. Zbog ograničenog prostora na ekranu prenosivog uređaja veliki je izazov osmisliti aplikaciju koja će biti pregledna, jednostavna za korišćenje, dovoljno intuitivna i povrh svega dovoljno privlačna za krajnjeg korisnika. Pored poteškoća koje se javljaju prilikom dizajniranja grafičkog interfejsa (eng. GUI - Graphical User Interface) mora se i uzeti u obzir, pošto je u pitanju Android aplikacija, i velika fragmentacija uređaja kao i samih verzija operativnih sistema, što dalje vodi do velikog broja različitih veličina ekrana (podjednako i dužina dijagonale, kao i rezolucija samog displeja), snage procesora, veličine memorije, implementacije framework-a... Kada se uzmu u obzir sva ograničenja dolazi se do zaključka da je neophodno dobro osmisliti i optimizovati same grafičke komponente, pošto grafička izračunavanja najviše oduzimaju procesorsko vreme, dok slike mogu poprilično opteretiti radnu memoriju uređaja.

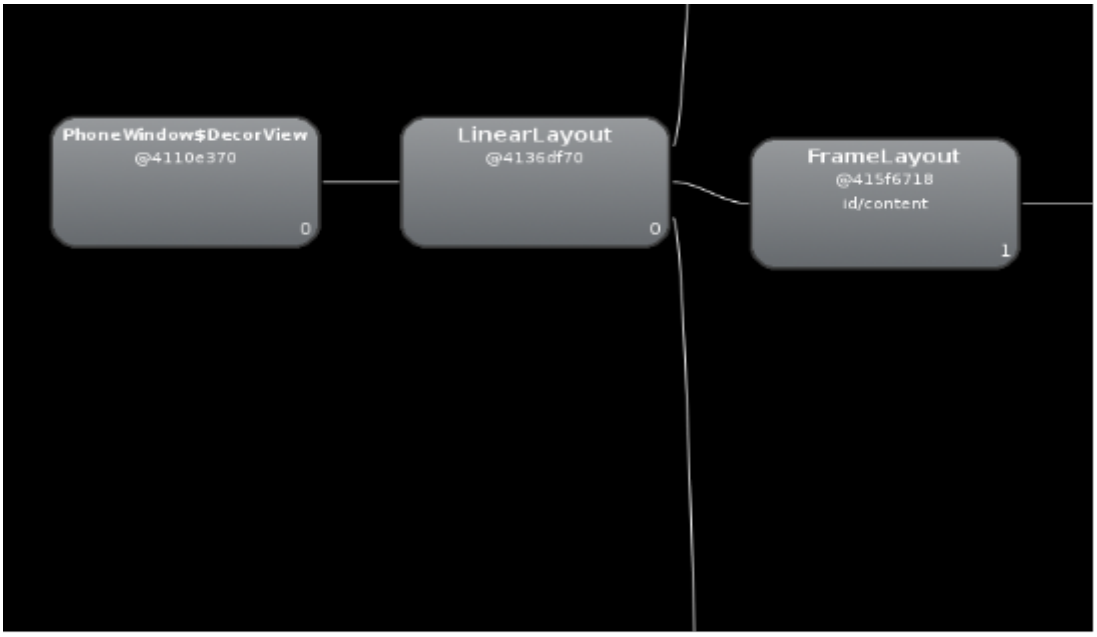
U ovom sloju, jedan od projektnih uzoraka koji se često koriste u pravljenju sloja pogleda Android aplikacije je Kompozitni uzorak (eng. Composite pattern) . Ideja ovog uzorka jeste da se što više iskoriste već postojeći objekti tako što generišu stablo objekata iste klase. [14]



Slika 4.3: Unutar komponente interfejsa su definisane metode koje treba da implementira svaka klasa kompozitnog uzorka. Metode koje se najčešće nalaze tu su uglavnom dodavanje,brisanje i dohvatanje objekta, kao i neka specifična operacija.

Tipičan primer ovog uzorka je sama Hijerarhija pogleda (eng. View Hiearchy). Hijerarhija pogleda predstavlja stablo svih elemenata koji se nalaze u jednoj aktivnsti, odnosno ekranu. Osnovna klasa kompozitnog uzorka je klasa Pogled (eng. View). Prilikom kreiranja jedne aktivnosti, prvo se kreira instanca samog ekrana koja je korena komponenta svake aktivnosti koja se iscrtava na ekranu[9]. Ta korena komponenta sadrži samo jednog potomka koji je klase Pogleda. Klasa pogleda je definisana od strane aplikacije koja se trenutno vidi na ekranu.

[9] Klasa aktivnosti može da postoji i ako nema ekran, odnosno pogled koji se vezuje za nju. U tom slučaju, aktivnost neće biti prikazana korisniku, ali će proći kroz svoj životni tok (eng. lifecycle).



Slika 4.4: Primer korena stabla pogleda u jednoj aktivnosti prikazana u Hijerarhijskom pregledaču [10]

Primer klase nasleđene iz klase pogleda (eng. View) sa implementacijom Kompozitnog projektnog uzorka

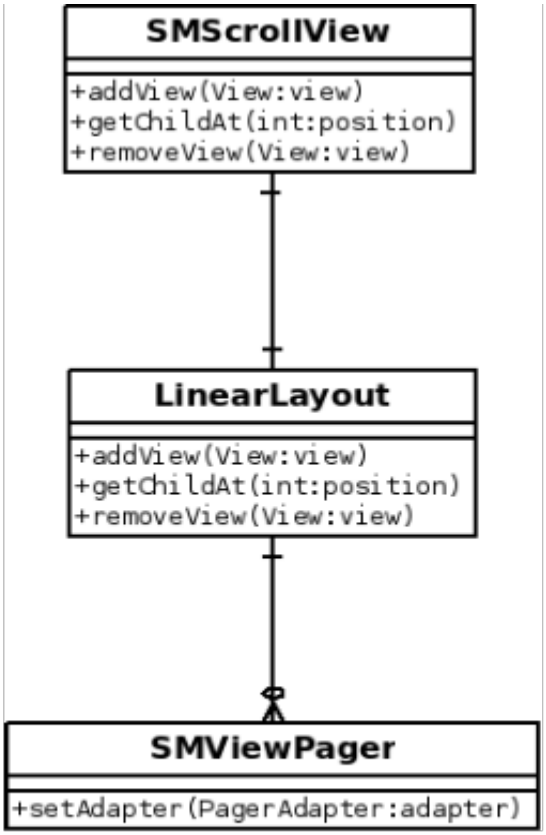
Kao što je već ranije napomenuto, svaka kontrola koju želimo da prikažemo na ekranu Android uređaja mora kao svog pretka imati klasu pogleda. Jedna od specifičnijih kontrola koje se koriste u ovoj aplikaciji je klasa pod nazivom *SMScrollView*. Ova klasa realizuje kontrolu koja omogućava korisniku da vertikalnim pokretima prsta po ekranu menja vertikalne stranice, odnosno kategorije[11].

Sama klasa SMScrollView je naslednik klase ScrollView koja se nalazi u Android framework-u, a koja omogućava drugim naslednicima View klase da skrolovanjem prikažu ceo svoj sadržaj ukoliko on prelazi postavljene okvire same kontrole. Kontrola, koja je proširena dodavanjem u objekat klase SMScrollView, je

[10] Hijerarhijski pregledač (eng. Hierarchy viewer) je aplikacija koja dolazi sa Android SDK-om i koristi se da prikaže celo stablo pogleda sa samim prikazom tih komponenti na ekranu, kao i potencijalne problematične tačke koje se mogu javiti prilikom iscrtavanja. [9]

[11] Ovakav način navigacije je postao jedan od čestih načina kretanja kroz Android aplikacije od verzije 4.0 Android OS-a. Iz tih razloga se na tržištu pojavilo dosta besplatnih i komercijalnih biblioteka koje implementiraju ovakvu funkcionalnost.

instanca klase LinearLayout-a[12]. U instancu klase LinearLayout-a su, na osnovu podataka dobijenih iz CMS servera, redom dodati odgovarajući objekti klase SMViewPager. Ovde se može prepoznati projektni uzorak pod nazivom Dekorater. Dekorater projektni uzorak služi da proširi funkcionalnost nekog objekta, bilo statički ili dinamički, ali tako da ne promeni funkcionalnost same klase.

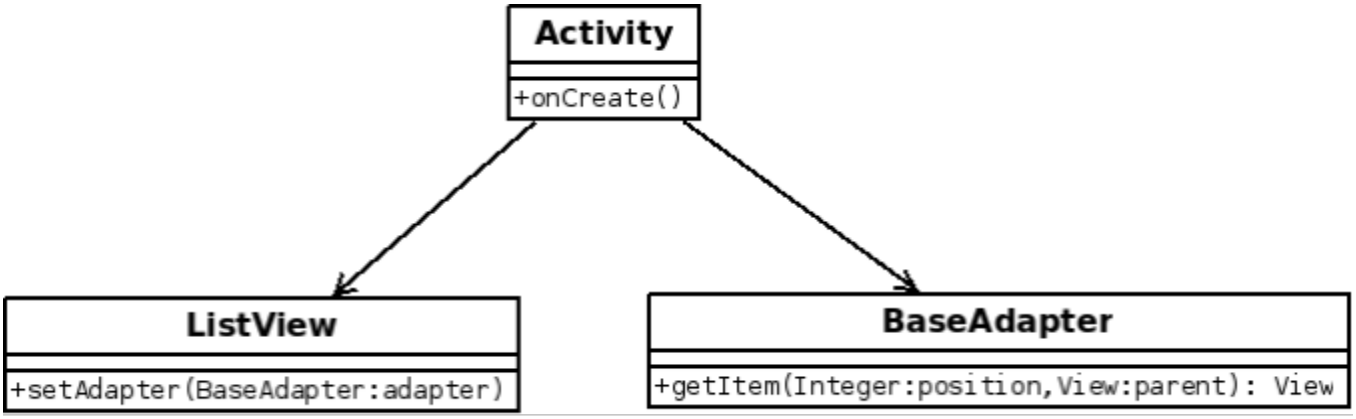


Slika 4.5: Arhitektura glavnog ekrana aplikacije

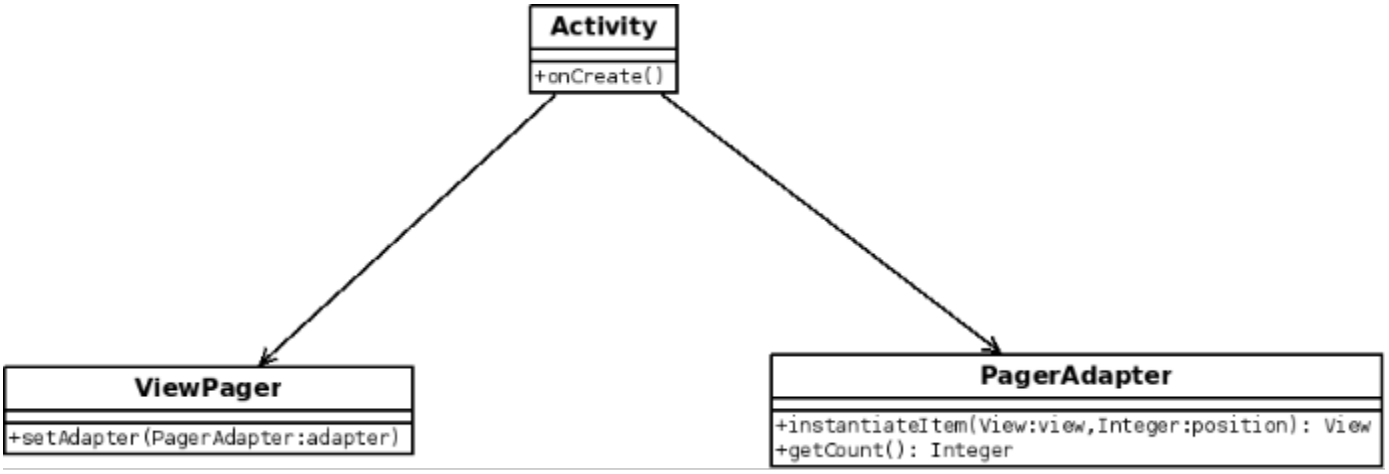
SMViewPager je klasa nasleđena iz klase ViewPager. Sama klasa ViewPager je kontrola koja omogućava korisniku da horizontalnim pokretima prsta po ekranu Android uređaja pomera sadržaj po stranicu levo ili desno. Sadržaj te kontrole se podešava tako što se postavi adapter klase PagerAdapter u kome se vrši instanciranje i definisanje drugih pogleda. Ovakav način definisanja kontrola je standardan na Android platformi i tu se može primetiti MVP (eng. Model View Presenter) arhitekturalni uzorak. MVA je baziran na MVC uzorku i služi da potpuno razdvoji komunikaciju između prikaza i samih podataka korišćenjem

[12] LinearLayout je klasa koja služi da prikazuje svoju decu redom u kom su dodate, koja su takođe potomci View klase bilo horizontalno bilo vertikalno. LinearLayout klasa je dobar primer korišećnja Kompozitnog projektnog uzorka.

dodatnog sloja koje je odgovoran za instanciranje i kontrolisanje kako podataka tako i sloja pogleda.



Slika 4.6: Tipičan primer upotrebe MVA u Android aplikacijama



Slika 4.7: Prikaz MVA projektnog uzorka implementiranog u aplikaciji

4.2.2. Prikazi nekih od interesantnijih klasa vezanih za sloj pogleda

Klasa SMScrollView

```
public class SMScrollView extends ScrollView {
	private boolean mIsFirstMove=true;
	private boolean mIsProccesed=false;
	private float mXMove;
	private float mYMove;
	private float mXMoveStart;
	private float mYMoveStart;
	private int mScrollPosition=0;
	private int mItemHeight;
	private int mScrollViewItemPosition=0;
	private int mNumberOfViewPagers=0;
	private SMScrollViewListener mVerticalPagedelegate;
	private float mPreviousMove;
	private boolean scrollDisabled=false;
	private boolean disableTouch=false;


	public boolean isScrollDisabled() {
		return scrollDisabled;
	}

	public void setScrollDisabled(boolean scrollDisabled) {
		this.scrollDisabled = scrollDisabled;
	}

	private boolean isVerticalScroll;
	private float mDistanceTraveled=0;
```

```
public SMScrollViewListener getVerticalPageDelegate() {
		return mVerticalPagedelegate;
}

public void setVerticalPageDelegate(SMScrollViewListener delegate) {
		this.mVerticalPagedelegate = delegate;
}


/**
 * gets the height of one ViewPager inside
 * this control
 * @return integer that represents item
 */
public int getItemHeight() {
		return mItemHeight;
}

/**
 * sets the variable
 * that represents item height
 * for one ViewPager
 * @param itemHeight of ViewPager
 */
public void setItemHeight(int mItemHeight) {
		this.mItemHeight = mItemHeight;
}

/**
 *
 * @return gets the number of view pagers
 * inside this control
 */
public int getNumberOfViewPagers() {
		return mNumberOfViewPagers;
}
```

```
/**
 * sets the number of
 * view pagers inside this control
 * @param mNumberOfViewPagers
 */
public void setNumberOfViewPagers(int mNumberOfViewPagers) {
		this.mNumberOfViewPagers = mNumberOfViewPagers;
}

public SMScrollView(Context context, AttributeSet attrs, int defStyle) {
		super(context, attrs, defStyle);
}

public SMScrollView(Context context, AttributeSet attrs) {
		super(context, attrs);
}

public SMScrollView(Context context) {
		super(context);

}

@Override
public boolean onTouchEvent(MotionEvent ev) {
		if (scrollDisabled) return true;


		switch (ev.getAction()){
		case MotionEvent.ACTION_DOWN:
			disableTouch=true;
			break;
```

```
case MotionEvent.ACTION_MOVE:
                this.scrollBy(0, Math.round(mPreviousMove -
               (ev.getY()<0?0:ev.getY())));
                mYMove=mPreviousMove;
                mPreviousMove = (ev.getY()<0?0:ev.getY());
                mDistanceTraveled+=
               (mPreviousMove>mYMove?
      mPreviousMove-mYMove:mYMove-mPreviousMove);

                break;
case MotionEvent.ACTION_UP:
                if (!mIsFirstMove) mIsFirstMove=true;
                mIsProccesed=false;

                if (isVerticalScroll){
                        if (Math.abs(mYMove-mPreviousMove)<30 &&
           mDistanceTraveled<getHeight()/2-30){
                                this.smoothScrollTo(0, mScrollPosition);
                                break;
                        }

                        //added for bottom strip animation
                        //remove it it slows down the app
                        SMViewPager vp = (SMViewPager)((LinearLayout)
             (this.findViewById(R.id.contentLayout))).
               getChildAt(mScrollViewItemPosition);
                        vp.stopBottomStripAnimation();

                        if (mYMove-mPreviousMove>=0 &&
           mDistanceTraveled<getHeight()-10) {
                                if (mScrollViewItemPosition<mNumberOfViewPagers-1){
                                        mScrollPosition+=getHeight();
                                        mScrollViewItemPosition++;
                                }

                        } else {
                                if (mScrollViewItemPosition>0){
```

```
                                        mScrollPosition-=getHeight();
                                        mScrollViewItemPosition--;
                                    }
                                }

                                mPreviousMove=0.0f;

                                this.smoothScrollTo(0, mScrollPosition);
                            }
                            break;
                    case MotionEvent.ACTION_CANCEL:
                            break;
                    default:
                            break;
                    }
                    return false;
            }

            @Override
            protected void onScrollChanged(int l, int t, int oldl, int oldt) {
                    if (t==mScrollPosition){
                            if (mVerticalPagedelegate!=null && mScrollViewItemPosition>=0
                        && mScrollViewItemPosition<mNumberOfViewPagers){
                                    mVerticalPagedelegate.onVerticalPageChanged(
                              mScrollViewItemPosition);
                                    disableTouch=false;
                            }
                    }
                    super.onScrollChanged(l, t, oldl, oldt);
            }




            @Override
            public boolean onInterceptTouchEvent(MotionEvent event) {
```

```
if (event.getAction() == MotionEvent.ACTION_DOWN){
        mPreviousMove = event.getY();

        mIsProccesed=false;
        mXMoveStart = event.getX();
        mYMoveStart = event.getY();
        mDistanceTraveled=0;

        if (disableTouch) return true;
        return false;
}

if (event.getAction() == MotionEvent.ACTION_MOVE){
        if (mIsProccesed){
                if (isVerticalScroll){
                        this.onTouchEvent(event);
                        return true;
                }else{
                        return false;
                }
        }

        mXMove=mXMoveStart-event.getX();
        mYMove=mYMoveStart-event.getY();

        if (Math.abs(mXMove)>40){
                mIsProccesed=true;
                isVerticalScroll=false;
                return false;
        }else if (Math.abs(mYMove)>=20){

                mIsProccesed=true;

                isVerticalScroll=true;

                return false;
        }
```

```
                }

                return false;
        }

        /**
         * scrolls to view pager on the set position
         * @param position of the view pager to scroll to
         */
        public void scrollToViewPager(int position){
                mScrollPosition=position*getHeight();
                mScrollViewItemPosition=position;
                this.scrollTo(0, mScrollPosition);
        }
}
```

Klasa SMViewPager

```
public class SMViewPager extends ViewPager {

        .
        .
        .

        @Override
        public boolean onTouchEvent(MotionEvent event) {
                if (!mIsVerticalScrollable) return true;

                if (this.enabled) {
                        return super.onTouchEvent(event);
                }

                if (event.getAction() == MotionEvent.ACTION_MOVE){
                        if (mIsProccesed){
                                return false;
                        }

                        mXMove=mXMoveStart-event.getX();
                        mYMove=mYMoveStart-event.getY();
                        if (Math.abs(mXMove)>10){
                                mIsProccesed=true;
                                return false;
                        }else if (Math.abs(mYMove)>20){
```

```
                                        mIsProccesed=true;

                                        return super.onInterceptTouchEvent(event);
                                }
                        }

                        return super.onTouchEvent(event);
                }

                @Override
                public boolean onInterceptTouchEvent(MotionEvent event) {
                        if (!mIsVerticalScrollable) return true;
                        if (this.enabled) {
                                return super.onInterceptTouchEvent(event);
                        }

                        if (event.getAction() == MotionEvent.ACTION_DOWN){
                                mIsProccesed=false;
                                mXMoveStart = event.getX();
                                mYMoveStart = event.getY();
                        }
                        return super.onInterceptTouchEvent(event);
                }
                .
                .
                .
}
```

4.3 Sloj kontrolera (eng. Controler) - klasa Atkivnosti

Sloj kontrolera u Android aplikacijama se može videti u klasama nasleđenim iz klase Aktivnosti. Potklasa klase Aktivnosti se instancira od strane sistema i ona je zadužena da pribavi podatke i na odgovarajući način ih prikaže korisniku. Prikazivanje pribavljenih podataka korisniku je moguće raditi na nekoliko načina. Najjednostavniji način prikazivanja (instanciranja) sloja pogleda je korišćenjem već definisanih XML Layout-a. Definisani XML Layout-i u suštini predstavljaju (još uvek neinstancirane) objekte neke od potklasa klase View koje u sebi već imaju definisane vrednosti svih neophodnih atributa potrebnih za odgovarajuće prikazivanje tih vizuelnih komponenti na ekranu. Ovo instanciranje izvršava sam operativni sistem. Takođe, pored korišćenja XML fajlova, moguće je i implementirati u Javi neku drugu vizuelnu komponentu i instancirati je iz objekta neke od potklasa klase Aktivnosti.

Osim kontrolisanja View-ova i pribavljanja podataka, u potklasama klase Aktivnosti se takođe mogu i izvršavati i drugi poslovi. Neki od njih mogu biti kontrolisanje multimedijalnog sadržaja, komunikacija sa operativnim sistemom, startovanje drugih aktivnosti i sl.

5. Zaključak i dalji rad

Android je za jako kratko vreme postao najpopularniji operativni sistem za veliki broj najrazličitijih uređaja. Tome su pored pristupačnosti samih uređaja, kao i jednostavnost korisničkog interfejsa doprineli i sami softverski inžejneri koji su na vrlo jednostavan i efikasan način pravili aplikacije koje su brzo došle do krajnjih korisnika. Dobra dokumentacija, jasna objašnjenja i načini korišćenja Android SDK-a su omogućili jednostavan razvoj i brzo savladavanje framework-a.

Pošto je ova aplikacija bila dizajnirana u trenutku kada je Android još bio u povoju (Android 2.3, API 10) nisu iskorišćeni svi potencijali koji bi trenutno mogli biti iskorišćeni. Od dolaska Androida 4.0 (API 14) se mnogo stvari promenilo, tako da bi mogla biti urađena bolja optimizacija za tablet uređaje, veća integracija sa socijalnim mrežama, bolja podrška i uvezenost sa Google servisima …

Korišćenjem samih projektnih uzoraka se daleko može olakšati dalji razvoj Android aplikacija kao i postići bolja optimizovanost samih aplikacija.

No, često se može doći do slučaja kada samo korišćenje nekog projektnog uzroka može dodatno opteretiti hardver na uređaju. U tim slučajevima je bolja ideja razmisliti ponovo o arhitekturi i možda izbeći korišćenje određenog projektnog uzorka, jer je i dalje na prvom mestu da krajnji korisnik aplikacije bude zadovoljan izgledom i funkcionalnošću.

6. Reference

[1] Wikipedia, White-label product,
http://en.wikipedia.org/wiki/White-label_product, datum pristupa 01.11.2012.

[2] Android developers, GCM
http://developer.android.com/about/versions/android-2.2.html, datum pristupa 2.11.2012

[3] Mac developer library, APN server
http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction/Introduction.html, datum pristupa 2.11.2012.

[4] Wikipedia, Android (operating system)
http://en.wikipedia.org/wiki/Android_(operating_system), datum pristupa 20.11.2012.

[5] IBM developer works
http://www.ibm.com/developerworks/webservices/library/ws-restful/, datum pristupa 12.02.2012.

[6] Android Developers veb sajt
http://developer.android.com/guide/components/fundamentals.html, datum pristupa 12.02.2013.

[7] Ritter, T. 1986. The Great CRC Mystery. *Dr. Dobb's Journal of Software Tools*. 11(2): 26-34, 76-83

[8] J2EE Patterns - Data Access Object
http://www.oracle.com/technetwork/java/dataaccessobject-138824.html, datum pristupa 02.07.2013.

[9] Hierarchy viewer - Android developers
http://developer.android.com/tools/help/hierarchy-viewer.html, datum pristupa 27.08.2013.

[10] James Steele, Nelson To, "The Android Developer's Cookbook: Building Applications with the Android SDK", Pearson Education, 2011

[11] Ronan Schwarz, Phil Dutson, James Steele, Nelson To, "The Android Developer's Cookbook: Building Applications with the Android SDK", Addison-Wesley,2013

[12] Shane Conder, Lauren Darcey,
"Android Wireless Application Development: Barnes & Noble Special Edition", Pearson Education, 2011

[13] Lauren Darcey, Shane Conder,
"Android Wireless Application Development Volume II: Advanced Topics", Addison-Wesley, 2012

[14] Erich Gamma et al.,
Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994