УНИВЕРЗИТЕТ „УНИОН“
РАЧУНАРСКИ ФАКУЛТЕТ
Кнез Михаилова 6/VI
11000 БЕОГРАД

Број:
Датум:

UNIVERZITET UNION
RAČUNARSKI FAKULTET
BEOGRAD
Računarske nauke

DIPLOMSKI RAD

Kandidat: Miloš BIljanović
Broj indeksa: RN-21/11

Tema rada: Realizacija nadzorno/upravljačkog softverskog sistema pod windows operativnim sistemom, za primenu kod MobiSun uređaja (mobilni robotizovani solarni elektrogenerator)

Mentor rada: dr Miloš Jovanović

Beograd, 04.11.2015.

Apstrakt

Cilj ovog rada je razvoj upravljačko/nadzornog softvera pod android platformom za MobiSun uređaj (mobilni robotizovani solarni elektrogenerator). MobiSun je automatizovani robotski mehanizam za proizvodnju električne energije, koristeći sunčevu svetlosnu energiju. Zadatak ovog softvera jeste da omogući korisnicima jednostavno i efikasno upravljanje uređajem, kao i prikazivanje trenutnih vrednosti senzora i ostalih relevantnih podataka preko smartfona ili tableta. Glavna prednost i potreba za ovim softverom jeste što omogućuje daljinsko upravljanje uređajem, čime se otklanja potreba da korisnik bude fizički prisutan pored uređaja da bi ga stavio u pogon ili kontrolisao njegov rad.

DEO I – UVOD

1. UVOD

Tema ovog rada je izrada nadzorno/upravljačkog softvera za MobiSun uređaj na Android platformi. Rad se sastoji iz dva dela: klijenta i servera.

Klijent i server su pisani u Java [1] programskom jeziku. Server treba da predstavlja simulaciju pravog servera koji će se u budućnosti nalaziti na MobiSun uređaju. U trenutku izrade ovog rada nije tačno definisan konačan server kao ni sam protokol komunikacije između klijenta i servera. Scenario korišćenja je sledeći: klijent koji komunicira sa serverom preko tcp/ip protokola, a server zatim komunicira sa MobiSun uređajem. S obzirom na to, urađeno je da se komunikacija odvija samo između klijenta i servera, čime se ostavlja sloboda da se server kasnije jasno definiše, nezavisno od platforme na kojoj će se nalaziti, kao i od samog jezika.

Na ovaj način je omogućen razvoj i testiranje funkcionalnosti klijenta, koji će kasnije u nekom trenutku, uz minimalne promene nakon što server bude gotov, moći da se isporuči svojim prvim kupcima uređaja. Što se tiče klijenta, iako postoje alternative Javi na Androidu [2] , smatra se da je to ipak najbolja opcija iz mnogobrojnih razloga.

Glavni ciljevi, osim funkcionalnosti klijenta, jesu jasan, “lak” i konzistentan vizuelni prikaz klijenta kao i laka upotreba softvera, tako da tehnički ne previše stručnim licima omogući korišćenje klijenta bez obuke. Time se omogućava lako upravljanje uređajem, bez gubljenja vremena na obuku.

Pod funkcionalnošću klijenta se podrazumeva prikaz svih relevantnih informacija sa uređaja kao što su vrednosti sa senzora, informacije o samoj lokaciji uređaja, koje su potrebne korisniku, kao i upravljanje uređajem (npr. njegovo paljenje i gašenje).

Ovaj rad bi trebao da posluži kao osnova za dalji razvoj klijenta, tj za dalje nadogradnje, i zbog toga je veoma važno da klijent bude lako proširiv i održiv.

Uz ovaj rad neće biti priložen instalacioni fajl samog klijenta zbog određenih okolnosti o kojima će biti reč kasnije, ali će biti detaljno opisan način dobijanja instalacionog fajla, kao i njegovo prebacivanje i instaliranje na android uređaju.

Dalje u radu će biti dat detaljan opis samog klijenta. Sam server je polužio za testiranje i simulaciju, te nije potrebno više pisati o njemu.

Uz ovaj rad dolaze i fajlovi koji su nastali tokom razvoja sistema. To su:

1) MobiSun – Android Studio projekat (klijentski program)

2) Server – Android Studio projekat (server za simulaciju)

Ovaj rad je proizašao iz investicionog projekta pod nazivom **„Razvoj daljinski upravljanog mobilnog robotizovanog solarnog elektrogeneratora za unapređenje poljoprivredne proizvodnje“**, broj projekta 451–03–2082 IP, koji je finansiran od strane Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije. Realizator projekta je Institut Mihajlo Pupin, Beograd.

2. O UREĐAJU

2.1. Šta je mobiSun?

Mobilni robotizovani solarni elektrogenerator MobiSun je savremeni, automatizovani uređaj za proizvodnju električne energije (struje) korišćenjem nepresušne sunčeve svetlosne energije. U osnovi, to je dvoosni robotski mehanizam koji pokreće tri svetlosno-osetljiva fotonaponska panela, koji je postavljen na standardnu putničku auto prikolicu (Slika 2.1), a koja mu omogućava laku pokretljivost i brzu promenu lokacije. Odgovarajući sistem za praćenje dnevnog sunčevog kretanja na horizontu (poput pokretanja suncokreta u prirodi) omogućava mobilnom solarnom generatoru maksimalno iskorišćenje raspoložive svetlosne energije koja se pretvara u električnu energiju uz pomoć pokretnih fotonaponskih panela i odgovarajuće elektronike. Dobijena električna energija (monofazna struja 220V, 50Hz) se može koristiti direktno iz invertora ili se čuvati u baterijskom skaldištu (paketu kvalitetnih baterija) visokog kapaciteta i dubine pražnjenja. Na taj način, uređaju je omogućeno

produženo radno dejstvo, odnosno proizvodnju struje tokom dana i noći. Uređaj ima dopunsku mogućnost daljinskog komandovanja uključivanja i isključivanja, praćenja napona na baterijama, ostvarene snage i proizvedene energije, ukupnog vremena rada itd., posredstvom mobilnog telefona (npr. smart uređaja). Na taj način, korisnik nije u obavezi da bude fizički prisutan pored uređaja da bi ga stavio u pogon ili kontrolisao rad. Uređaj ima mogućnost i daljinske elektronske zaštite od krađe, odvlačenja ili oštećenja. Mobilni solarni generator može funkcionisati i bez prikolice kao noseće platforme, jednostavnim postavljanjem na pomoćne stabilizatore (noge, Slika 2.3). U transportnom položaju, solarni generator je uredno sklopljen na prikolici i pokriven ceradom radi zaštite od prašine iz atmosfere (Slika 2.1). U periodima dana ili godine kada nema dovoljno sunčeve svetlosti uređaj se može dopunjavati i iz kućne elektromreže ili iz motornih benzinskih i dizel agregata.

Slika 2.1 MobiSun u položaju spremnom za transport

Slika 2.2 MobiSun u sklopljenom stanju

Slika 2.3 mobiSun u radnom položaju

Sa porastom cene goriva na tržištu, ali i odsustvom odgovarajuće elektro građevinske infrastrukture, efikasnost i pristupačnost vanmrežnih (off-grid) fotonaponskih sistema neprekidno raste čineći da solarna energija postane konkurentna i pametna alternativa za budućnost. Kako dobri elektroagregati na fosilna goriva (benzinski i dizel) mogu služiti nekoliko godina, tako radni vek solarnih generatora može pouzdano proizvoditi električnu energiju 30 i više godina bez posebnog održavanja. To čini MobiSun uređaj ne samo ekološki čistim uređajem koji proizvodi struju, već takođe vrednu i mudru investiciju za budućnost. Mobilni robotizovani solarni generatori postavljeni na autoprikolici koriste solarne panele, baterije i elektroniku najvišeg kvaliteta. Zato naš solarni strujni agregat obezbeđuje pouzdano snabdevanje energijom decenijama.

2.2. Karakteristike

Čist – Nema izduvnih gasova niti emisije ugljen-dioksida u atmosferu. Nema zamene motornog ulja, curenja istog, problema sa hladnim paljenjem i sl.

Bešuman – U okolini radnog mesta ne stvara buku niti neprijatne zvuke čime je veoma pogodan za primenu u naseljima, ali i u prorodi, pošto ne uznemirava živi svet. Na mestima gde su pouzdane komunikacije stvar bezbednosti, ovaj bešumni pokretni solarni generator ispoljava svoju punu vrednost.

Pouzdan – Mobilni solarni generator se ne oslanja isključivo na energiju dobijenu od sunca. Sistem se može dopunjavati i iz elektromreže ili iz tradicionalnih dizel i benzinskih agregata u danima kada je oblačno i kada intenzitet sunčeve svetlosti nije dovoljan da napuni baterije u željenom roku.

Pristupačan – Iako je cena solarnog generatora viša od konkurentnih motornih agregata na fosilna goriva, ovaj uređaj štedi uložena sredstva posmatrano na duži rok. Ovi uređaji traju duže, ne zahtevaju servisno održavanje i ne zavise od porasta cene goriva.

Izdržljiv – Sa foto-panelima i mehaničkom konstrukcijom projektovanim da izdrže grad i ostale atmosferske uticaje kao i olujni vetar do 100 km/h, industrijski kvalitet baterija za solarne aplikacije, strujnih invertora otpornih na neravnom terenu, ovaj uređaj se može prenositi na udaljena mesta i služiti decenijama.

Bezbedan – Ovaj solarni generator nema negativnih uticaja na ostale uređaje (kućne aparate, mašine, računare i sl.), daje stabilan napon i ne uzrokuje elektromagnetne smetnje. Bezbedan je za korisnike jer je projektovan po međunarodno važećim standardima zaštite od strujnog udara.

Mobilan – Uređaj je pokretan i prenosiv tako što je postavljen na standardnu autoprikolicu, ali može raditi i bez nje postavljen na četiri noge (stabilizatore). Uređaj se

iz transportnog režima stavlja u radni režim za svega 15 minuta, ne zahtevajući pri tom posebnu obučenost korisnika.

Daljinsko komandovanje – Mobilni robotizovani solarni generator može biti uključen i isključen korišćenjem prednosti daljinskog komandovanja sa mobilnog telefona ili tableta. Takođe, korisniku je omogućeno praćenje napona na baterijama, kapaciteta napunjenosti, vremena rada, maksimalno ostvarene i trenutne snage, kao i količine dnevno proizvedene energije. Takođe, moguće je daljinski pratiti i parametre vremenskih prilika (temperature, vlažnosti vazduha, barometarskog pritiska) na lokaciji postavljanja.

Efikasan – Mobilni solarni elektrogenerator je energetski efikasan uređaj. Opremljen je kontrolerom koji obezbeđuje u automatskom režimu rada maksimalno iskorišćenje sunčeve svetlosne energije na bazi ugrađenog pokretnog sistema za praćenje položaja sunca na horizontu. Ovakav sistem obezbeđuje na našem geografskom prostoru i do 40% više dobijene energije u poređenju sa stacionarnim (nepokretnim) solarnim generatorima. Sistem ima kontroler praćenja tačke maksimalne električne snage. Takođe, kapacitet baterijskog spremišta je dimenzionisan tako da omogućava svakodnevnu eksploataciju uređaja uz dopunjavanje energijom dobijenom sa postojećih fotonaponskih panela.

2.3. Gde se primenjuje?

Mobilni robotizovani solarni elektrogenerator MobiSun predstavlja osnovni ili dopunski (alternativni) “vanmrežni” off-grid izvor snabdevanja električnom energijom individualnih korisnika, malih i srednjih potrošača električne energije. Univerzalne je namene, može se koristiti u domaćinstvu (spolja), kućama za odmor, poljoprivredi (za navodnjavanje bašti, njiva, staklenika, za klimatizaciju plastenika, higijensko-temperaturno održavanje poljoprivrednih proizvoda, održavanje ribnjaka, u planinskom stočarstvu), turizmu (elektrifikacija lovišta, lovačkih domova, etno-naselja i restorana, parkovi prorode), saobraćaju (terenska signalizacija, održavanje puteva), za potrebe opremanja vojske i policije, na kampovanju, itd.

Pokretni solarni generator je projektovan da bude autonoman uređaj koji se ne priključuje na elektromrežu i ne zahteva nikakvu dodatnu građevinsku ili elektro infrastrukturu. Kao takav, ovaj uređaj je veoma pogodan za primenu na različitim terenima, kako ruralnim tako i urbanim, gde god postoje dobri uslovi za korišcenje obnovljive svetlosne energije sunca i gde ne postoje uslovi za korišćenje energetske infrastrkture (strujne mreže). Uređaj zadovoljava standarde kvaliteta i bezbednosti za ovu klasu proizvoda i otporan je na raličite vremenske uslove – kišu, sneg, grad, vetar, visoku i nisku temperaturu, itd. U organskoj proizvodnji hrane ovaj ekološki-ispravan

uređaj praktično da nema alternativu. Idealan je za navodnjavanje useva u poljoprivredi, za porodična poljoprivredna gazdinstva do 10ha korisne površine. Korišćenje uređaja ne zahteva posebnu obuku operatera niti tehničko obrazovanje. Uređaj je operativan u roku od 15 minuta iz transportnog režima.

3. ZAHTEVI

Potreba za ovim softverom ogleda se u tome da se ne tehničkim korisnicima (običnim korisnicima smartfona i tableta), omogući lako upravljanje, i nadzor MobiSun uređaja. Samim tim, to je od početka značilo da treba da se napravi što jednostavniji i intuitivniji program, koji je lak za korišćenje i ne zahteva mnogo (ako je moguće ni malo) vremena za učenje. Tako da je jedan od glavnih zahteva bio jednostavnost. Drugi bitan zahtev odnosi se na podatke koje treba prikazati korisniku, kao i upravljačke akcije koje korisnik može izvršiti. One su podeljene na 3 celine, i kao takve su i predstavljene korisniku, kroz 3 različita tab-a:

- Upravljački i bezbednosni podaci

1. Uređaj - Uključi/Isključi
2. Lampa - Uključi/Isključi
3. Kamera - Uključi/Isključi
4. Pozicioniranje panela - azimut (stepeni), elevacija (stepeni)
5. GPS pozicija
6. Ziroskopski položaj
7. Orijentacija panela
8. Kinezo senzor
9. Greška fid sklopa
10. Preopterećenje izlaza

- Tehnički podaci o uređaju

1. Napon baterija (V)
2. Kapacitet baterija (Ah)
3. Temperatura u kućištu (C)
4. Trenutna proizvdnja/snaga (KW)
5. Trenutna potrošnja (W)
6. Dnevna proizvodnja (KWh)
7. Proizvodnja u definisanom periodu (Kwh)
8. Vreme rada od uključenja (min)
9. Raspoloživo vreme rada (min)
10. SOC % Status napunjenosti
11. Napon panela
12. Temperatura panela
13. Status sistema

- Meteo podaci na lokaciji

1. Temperatura vazduha (C)
2. Barometarski pritisak (bar)
3. Vlažnost vazduha (%)
4. Brzina vetra (m/s)
5. Sunčevo zračenje (W/m2)
6. Senzor 1 (npr. vlažnost zemljišta)
7. Senzor 2 (npr. temperatura zemljišta)
8. Senzor 3 (npr. zađubrenost)
9. Senzor 4
10. Senzor 5

Ostali zahtevi podrazumevaju poseban tab na kome su prikazane greške koje su se dogodile tokom rada, kao i opis svih grešaka, postavljanje logo-a na prikladna mesta, jasna informacija korisniku o povezanosti klijenta i servera, kao i dodatna uputstva o daljem razvoju samog klijenta i njegovoj instalaciji. Takođe treba naglasiti da je jedan od zahteva bio i dobro iskomentarisan kod koji bi neko kasnije lako mogao i brzo da shvati.

Svi gore navedeni zahtevi su ispunjeni i biće predstavljeni dalje u ovom radu.

4. OPIS KORIŠĆENIH TEHNOLOGIJA

S obzirom da se radi o softveru namjenom android platformi, kao najefikasnije i najlogičnije rešenje se izdvaja programski jezik Java. Kao što je pomenuto postoje i alternativna rešenja koja ne koriste Javu, ali je smatrano da to nije pravi put ni u kom slučaju.

Druga stvar je upotreba TCP/IP protokola kao način komunikacije. Ova grupa protokola naziv je dobila po dva najvažnija protokola TCP (Transmission Control Protocol) i IP (Internet Protocol). TCP/IP omogućuje komunikaciju preko različitih, međusobno povezanih mreža, i danas je najrasprostranjeniji protokol na lokalnim mrežama. Na njemu se zasniva i globalna mreža, internet.

<table>
<tr><th>OSI model</th><th>TCP/IP model</th></tr>
<tr><td>7. Aplikacioni sloj</td><td rowspan="3">4. Aplikacioni sloj</td></tr>
<tr><td>6. Prezentacioni sloj</td></tr>
<tr><td>5. Sloj sesije</td></tr>
<tr><td>4. Transportni sloj</td><td>3. Transportni sloj</td></tr>
<tr><td>3. Mrežni sloj</td><td>2. Mrežni sloj</td></tr>
<tr><td>2. Sloj veze</td><td rowspan="2">1. Sloj veze</td></tr>
<tr><td>1. Fizički sloj</td></tr>
</table>

5. OPIS KORISĆENIH ALATA (IDE)

Tokom razvoja klijentskog i serverskog dela, korišćeno je okruženje Android Studio.

5.1. Android Studio

Android Studio je oficijalni IDE za razvoj Android aplikacija, koji se zasniva na poznatom okruženju IntelliJ IDEA [3].

Pošto je tokom razvoja sistema izlazila nova verzija Android Studio, ovde je napisana poslednja korišćena verzija Android Studio-a.

Android Studio 1.4.1
Build #AI-141.2343393, datum Oktobar 15, 2015

Link za preuzimanje: http://developer.android.com/sdk/index.html

5.2. Gimp 2

Možda treba napumenuti da je za obradu tj. Modifikaciju ikona koje se nalaze na tabovima korišćen Gimp 2 [4] program. Gimp je besplatan program za rad sa slikama.

Link za preuzimanje: http://www.gimp.org/downloads/ , korišćena verzija 2.8

5.3. Argouml

Argouml [5] je open-source za izradu UML dijagrama. Podržava standard UML 1.4. Može se pokrenuti na bilo kojoj platformi koja podržava javu.

Korišćena verzija je:
ArgoUML 0.32.2

Link za preuzimanje: http://argouml.tigris.org/

DEO II – TEHNIČKA DOKUMENTACIJA

6. GUI (GRAPHICAL USER INTERFACE)

U ovom poglavlju, biće prikazan finalni izgled programa, koji se sastoji iz dve aktivnosti, glavna aktivnost i aktivnost pomoć. Poseban odeljak će uzeti i orijentacija Android uređaja (Landscape i Portrait), opseg veličina Android uređaja koje softver pokriva, kao i lista GUI elemenata koja je posebno pravljena.

6.1. Glavna Aktivnost

Glavna aktivnost (slika 6.1) se sastoji iz više delova. To su :

1) **ToolBar** ima ulogu da prikaže MobiSun ime.

2) Deo za tab ikone, koje korisniku omogućavaju lak prelaz sa jednog taba na drugi.

3) Na samom dnu se nalazi status deo. Na njemu je uvek prikazana informacija o tome da li je klijent povezan sa serverom ili pak nema interneta. Kada nema interneta prikazana je crvena lampica i odgovarajuća poruka. Kada ima interneta, ali uređaj nema povezanosti klijenta i servera, prikazana je narandžasta lampica, dok u krajnjem slučaju, kad je klijent povezan sa serverom, prikazana je zelena lampica. U svim slučajevima postoji i propratna poruka koja još detaljnije objašnjava trenutni status. Sva tri slučaja se mogu videti na slikama 6.1, 6.2 i 6.3.

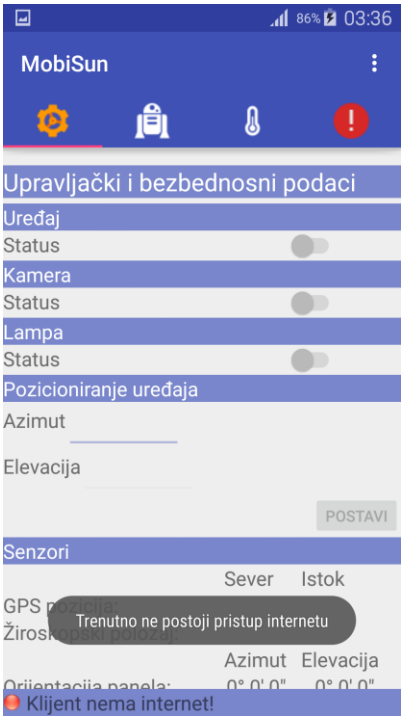
4) U sredini se nalazi jedan od četiri postojaća taba. To su:

    1. Upravljački i bezbednosni podaci
    2. Tehnički podaci
    3. Meteo podaci na lokaciji
    4. Istorija grešaka

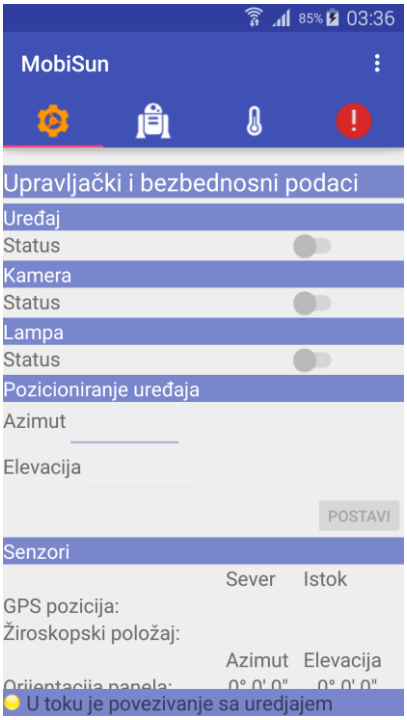
    O svakom od ovih tabova biće više reči u nastavku.

Treba još naglasiti da grafički izgled zavisi u odnosu na to da li je klijent povezan sa serverom ili ne. U odnosu na to, dugmići su "omogućeni-onemogućeni", kao i sam prikaz povratnih informacija koje se stalno "osvežavaju-neosvežavaju".

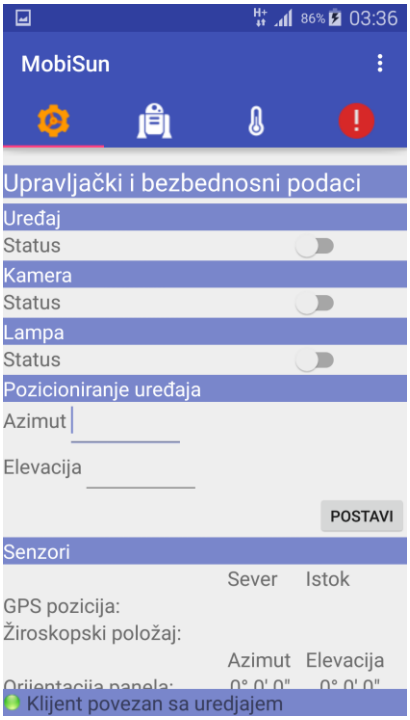
Što se tiče samog prikaza svih elemenata grafičkog interfejsa i koje sve uređaje u zavisnosti od veličine podržava biće reči na samom kraju ovog dela.



Slika 6.1



Slika 6.2



Slika 6.3

Slika 6.1 – Prikaz početnog fragmenta i statusne linije koja pokazuje da nema Interneta
Slika 6.2 – Prikaz početnog fragmenta i statusne linije da je povezivanje u toku
Slika 6.3 – Prikaz početnog fragmenta i statusne linije klijent povezan

6.1.1. Upravljački i bezbednosni podaci

Ovaj tab ima dve uloge. Jedna uloga je da omogući korisniku da obavi određene akcije, i zapravo sve akcije koje korisnik može da aktivira na uređaju nalaze se ovde, otuda i naziv taba. Druga uloga samog taba je da prikaže informacije o uređaju, npr. GPS pozicija ili da prikaže neku povratnu informaciju, npr. nakon što korisnik postavi panele u položaj, taj položaj će biti prikazan.

Ista podela se može i primeniti na izgled tabova. Prva polovina je upravljanje samim uređajem, dok je druga prikaz nekih informacija. Dakle u prvom delu imamo:

1) Naslov Uređaj, status (**Switch** komponenta) o tome da li je uređaj upaljen ili isključen za koju je takođe vezana akcija za paljenje i gašenje uređaja.

2) Naslov Kamera, status (**Switch** komponenta) o tome da li je kamera upaljena ili ugašena za koju je takođe vezana akcija za paljenje i gašenje kamere.

Slika 6.1, 6.2, 6.3 je prikaz ovog taba.

3) Naslov Lampa, status (**Switch** komponenta) o tome da li je lampa upaljena ili ugašena za koju je takođe vezana akcija za paljenje i gašenje lampe.

4) Naslov Pozicioniranje panela, dva polja za unos pozicije na osnovu azimuta i elevacije kao i dugme postavi koje je vezano za određenu akciju.

Druga polovina počinje od naslova Senzori i sadrži informacije koje dobija od servera. Spisak senzora je naveden u delu **Zahtevi**.

Treba napomenuti da se sve nalazi u komponenti **ScrollView** što omogućava vertikalan skrol ukoliko sav sadržaj ne može da stane.

6.1.2. Tehnički podaci

Tab Tehnički podaci sadrži jedan naslov Senzori i služi za prikaz vrednosti različitih senzora koje se nalaze na uređaju, a mogu se svrstati pod tehničke podatke. Spisak senzora je naveden u delu **Zahtevi.** Slika 6.4 je prikaz ovog taba.

Treba napomenuti da se sve nalazi u komponenti **ScrollView** što omogućava vertikalan skrol ukoliko sav sadržaj ne može da stane.

Slika 6.4. Fragment tehnički podaci

6.1.3. Meteo podaci na lokaciji

Tab Meteo podaci na lokaciji sadži jedan naslov Senzori i služi za prikaz vrednosti različitih senzora koje se nalaze na uređaju, a mogu se svrstati pod meteo podatke. Spisak senzora je naveden u delu **Zahtevi.** Slika 6.5 je prikaz ovog taba.

Treba napomenuti da se sve nalazi u komponenti **ScrollView** što omogućava vertikalan skrol ukoliko sav sadržaj ne može da stane.

Slika 6.5 Fragment Meteo podaci

6.1.4. Istorija grešaka

Tab istorija se sastoji od jedne tabele i dva dugmeta. Tabela se sastoji iz dve kolone datum i kod greške. Sama tabela je uokvirena kako bi prikaz nje bio jedna celina, a za to i sama polja tabele su korišćeni oblici koji su navedeni u odeljku FALI. Unutar okvira se nalazi komponenta **ScrollView** kako bi sve greške mogle da se pregledaju vertikalnim skrolom.

Dugme Obriši istoriju koje nakon klika prikazuje još jedno prozorče i pita korisnika da li je zaista siguran da želi da obriše sve greške iz istorije.

Dugme Pomoć je nova aktivnost pomoć o kojoj će biti reči kasnije (AktivnostPomoć odeljak 8.2.1.10) .

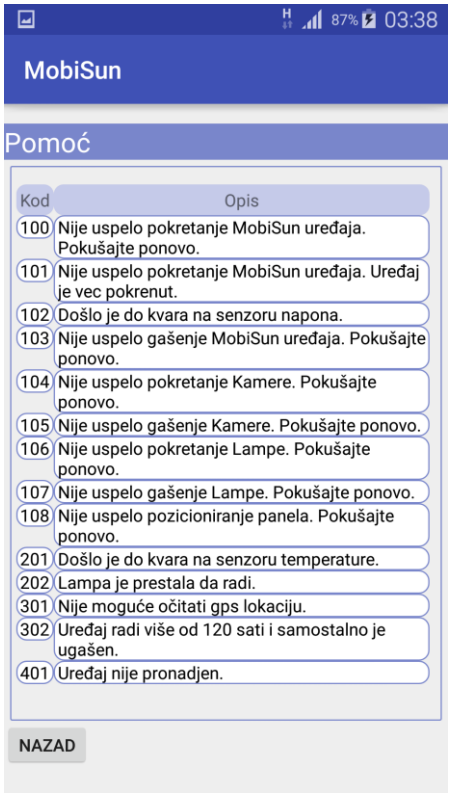
Slika 6.6 je prikaz ovog taba.

Slika 6.6 Fragment Istorija grešaka

6.2. Aktivnost Pomoć

Pomoć forma je forma koja služi za detaljan prikaz o greškama. Greška se sastoji iz koda i opisa. Sama aktivnost se sastoji iz jedne tabele i jednog dugmeta. Tabela po izgledu je ista kao tabela iz odeljka **Istorija grešaka.** Jedina razlika je u tome što kolone predstavljaju kod greške i opis greške.

Dugme nazad nije potrebno detaljnije objašnjavati i vraća korisnika na tab Istorija grešaka. Slika 6.7 je prikaz ove aktivnosti.

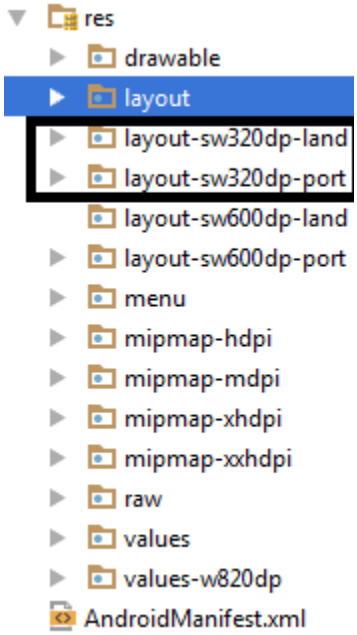


Slika 6.7 Aktivnost Pomoć

6.3. Orijentacija i veličina Android uređaja

Ove dve stvari su jako važne za GUI na Androidu. Dobra stvar je što se na lak način mogu regulisati obe stvari jednostavnim izmenama. Aplikacija trenutno grafički pokriva obe (**Landscape i Portrait**) orijentacije za veličine uređaja čija je najmanja dužina koju mogu da obezbede aplikaciji 320dp i 600dp. Jedna je za smartfon, a druga za tablete.

Npr.za 320dp postoje dva foldera (Slika 6.8). Ideja je da svaki uređaj koji može da obezbedi najmanje 320dp dužine ekrana za aplikaciju, android će koristiti ta dva foldera. Ukoliko može više npr. 600dp onda će koristiti odgovarajuća dva foldera. U svakom od tih foldera se nalaze sve već pomenute komponente, koje se doteruju (pomere, smanje, povećaju) tako da to izgleda dobro. Na taj način je vizuelni prikaz potpuno odvojen od akcija i događaja koje mogu da se dese na njemu. Takođe se na taj način pokriva i orijentacija i veličina ekrana dostupna aplikaciji. Za još preciznije veličine postoje finije granice. [6]



Slika 6.8

6.4. Posebni elementi

U ovom delu se nalaze elementi koji su korišćeni za tabele iz odeljaka (6.1.4. i 6.2). Na slici 6.7 nalaze se u folderu drawable. Fajlovi zakolonu.xml i celijazared.xml kao i okvir okvir.xml, se koriste za specifičan prikaz tabela.

7. FUNKCIONALNOST KLIJENTA

U ovom poglavlju, navedene su sve funkcionalnosti koje klijent pruža i koje su trenutno podržane:

1) Paljenje i gašenje uređaja

Na tabu *upravljački i bezbednosni podaci* nalazi se komponenta **Switch**, koja omogućava paljenje i gašenje uređaja. Takođe se prikazuje i trenutni status uređaja na toj komponenti.

2) Paljenje i gašenje lampe

Na tabu *upravljački i bezbednosni podaci* nalazi se komponenta **Switch**, koja omogućava paljenje i gašenje lampe. Takođe se prikazuje i trenutni status lampe na toj komponenti.

3) Paljenje i gašenje kamere

Na tabu *upravljački i bezbednosni podaci* nalazi se komponenta **Switch**, koja omogućava paljenje i gašenje kamere. Takođe se prikazuje i trenutni status kamere na toj komponenti.

4) Postavljanje panela u željeni položaj

Na tabu *upravljački i bezbednosni podaci* ispod već navedenih komandi nalaze se dva polja za unos vrednosti i dugme Postavi. Novi položaj, ako je uspešno promenjen, se pojavljuje pored tabele *Orijentacija panela*.

5) Prikazivanje poruka

Pod prikazivanjem poruka se smatraju pozitivni odgovori i uspešne akcije kao i greške koje je poslao server ili koje su nastale na klijentu.

1.Prikazivanje grešaka

Ako dođe do greške prilikom rada aplikacije ona se odmah prikazuje na ekranu nezavisno od toga na kom tabu se trenutno aplikacija nalazi. Osim toga, ako je grešku poslao server, onda se ta greška pamti u istoriju grešaka i pojavljuje se kao dodatna greška na tabu Istorija grešaka.

2.Prikazivanje poruka o uspešnim akcijama

Ako akcija uspe i server vrati potvrdan odgovor, prikazaće se odgovarajuća poruka o tome.

6) Prikazivanje svih grešaka koje mogu da se dogode

Pritiskom na dugme pomoć na tabu Istorija grešaka i otvaranjem aktivnosti Pomoć, prikazuju se greške koje su definisane i koje može server da pošalje.

7) Prikaz statusa programa (bez interneta, nije povezan, povezan)

Progam u zavisnosti od ova tri iznad pomenuta slučaja prikazuje određenu poruku i lampicu.

8) Senzori

Na tabovima *Tehnički podaci, Meteo podaci na lokaciji,* i u delu kartice *upravljački i bezbednosni podaci,* se prikazuju razni senzori koje uređaj ima.

9) Istorija grešaka

U kartici *Istorija grešaka* nalaze se hronološki poređane greške koje su se dogodile na uređaju. Moguće je obrisati istoriju, klikom na dugme Obriši, i prikazati Pomoc, klikom na dugme Pomoć.

8. OPIS SISTEMA

Ceo sistem se sastoji iz dva dela, servera i klijenta. Dalje u tekstu server će biti ukratko opisan, dok će uglavnom biti sve vezano za klijenta.

8.1. Server

Test server predstavlja deo sistema, koji prima poruke od klijenta i odgovara na njih. Za server važe sledeće pretpostavke:

1. Server sadrži informacije o svim senzorima na uređaju (u ovoj simulaciji ih sam generiše)
2. Pri pokretanju, ima javnu IP adresu i ona je unapred poznata klijentima
3. Server odgovara na svaku komandu koju prepoznaje, bez provere da li potiče od validnog korisnika

Kada se pokrene, server čeka nove zahteve.

Server za svaku konekciju kreira novu nit koja je zadužena za obradu konekcije.

Posao niti se sastoji iz dva zadatka:

Prvi je da parsira primljenu poruku, po protokolu koji je definisan.

Drugi je da na osnovu komande koja je poslata, pozove određenu klasu koja obrađuje tu komandu. Svaka komanda ima posebnu klasu, koja u zavisnosti od komande priprema odgovor i šalje ga klijentu. Dodavanje nove komande je lako, samo treba napraviti novu klasu koja će reagovati na zadatu komandu i registrovati je u glavnoj aktivnosti u onCreate metodi.

8.1.1. Komande

Komande su tako implementirane da se lako mogu dodavati. Svaka komanda implementira IListener. Za svaku novu metodu potrebno je definisati kako se komanda incijalizuje, njenu obradu zahteva kad stigne neki zahtev i ime komande koje mora da bude jedinstveno.

Server podržava sledeće komande:

1) Eho – Prima poruku od klijenta i vraća istu poruku nazad. Svrha upotrebe je čisto radi testiranja.

2) Ping – Ova komanda odgovara klijentu sa OK ako je server spreman da prima komande, ili NOK ako server radi ali trenutno iz nekog razloga nije spreman da prima zahteve.

3) Postavi panele – Komanda prima dve celobrojne vrednosti i šalje OK i iste te dve vrednosti. Na ovaj način se simulira uspešno postavljanje panela u traženi položaj. Takođe, komanda može da pošalje i NOK čime server javlja klijentu da nije u mogućnosti da postavi panele u određen položaj. Nakon toga se šalje i poruka tj. kod greške.

4) Meteo podaci – Komanda šalje OK i vrednosti senzora čije vrednosti mogu da se svrstaju u grupu meteo podataka.

5) Tehnički podaci – Komanda šalje OK i vrednosti senzora čije vrednosti mogu da se svrstaju u grupu tehničkih podataka.

6) Uključi uređaj – Komanda šalje OK, što znači da je uspešno upaljen uređaj. Takođe može da pošalje i NOK zbog neuspešnog paljenja uređaja i određen kod greške se šalje.

7) Isključi uređaj – Komanda šalje OK, što znači da je uspešno ugašen uređaj. Takođe može da pošalje i NOK zbog neuspešnog gašenja uređaja i određen kod greške se šalje.

8) Uključi kameru – Komanda šalje OK, što znači da je uspešno upaljena kamera. Takođe može da pošalje i NOK zbog neuspešnog paljenja kamere i određen kod greške se šalje.

9) Isključi kameru – Komanda šalje OK, što znači da je uspešno ugašena kamera. Takođe može da pošalje i NOK zbog neuspešnog gašenja kamere i određen kod greške se šalje.

10) Uključi lampu – Komanda šalje OK, što znači da je uspešno upaljena lampa. Takođe može da pošalje i NOK zbog neuspešnog paljenja lampe i određen kod greške se šalje.

11) Isključi lampu – Komanda šalje OK, što znači da je uspešno ugašena lampa. Takođe može da pošalje i NOK zbog neuspešnog gašenja lampe i određen kod greške se šalje.

8.2. Klijent

Klijent predstavlja komplikovaniji i zanimljiviji deo. Njegova uloga je da korisniku prikaže sve informacije i omogući upravljanje uređajem. Za klijenta važi da:

1. Unapred pretpostavlja da se server nalazi na istom uređaju na kome se i klijent pokreće. Trenutno se u klijentu zove metoda klase Konektivnost uzmiIpAdresu() koja vraća aktivnu ip adresu i na njoj pokušava da dobije server.

8.2.1. Klase

Pre nego što počnemo sa opisom svih klasa, potrebno je možda naglasiti da su imena promenljivih takva da budu što intuitivnija.

Napomena 1: Zbog problema promene orijentacije uređaja (smartfona i tableta), svaka klasa koja ima posledice toga će imati dodatno poglavlje o tome kako se rešava taj slučaj.

Napomena 2. Za bilo kakve izmene na grafičkom interfejsu potrebno je uraditi ih iz GUIThread-a, što se postiže na sledeći način:

```
runOnUiThread(new Runnable() {

@Override
public void run() {
       izemene koje želimo da izvršimo

}
});
```

Napomena 3. Za neke komponente nije potrebno održavati stanje pri promeni orijentacije panela programerski, već dodatnom linijom u GUI prikazu može da se obezbedi isto ponašanje. Npr komponenta **TextView** može da sadrži element

```
android:freezesText="true"
```

i taj problem se neće desiti.

Napomena 4. Uz napomenu 2 bi uvek trebalo da ceo *kod* bude uokviren ako se ta metoda izvršava unutar fragmenta sa

```
if(isAdded()){
}
```

Zato što postoje situacije kada fragment iako postoji nije povezan na aktivnost.

8.2.1.1. Glavna Aktivnost

Glavna aktivnost (kao i četiri fragmenta koji će kasnije biti opisani) su kontroleri kojim se omogućavaju sve akcije, izmene vezane za vizuelni prikaz. Da se primetiti da npr. za klasu GlavnaAktivnost, postoji i odgovarajući vizuelni prikaz glavna_aktivnost.xml.

Pri kreiranju ove aktivnosti poziva se metoda OnCreate(...), što je na neki način konstruktor. Tu se vrše razne incijalizacije i provere. To su:

1) Postavljanje pri prikazu glavne aktivnosti da nijedan EditText nema fokus, jer nema smisla da ga ima ukoliko korisnik nije pritisnuo na njega.

```
this.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_AL
WAYS_HIDDEN);
```

2) Provera da li fajl istorija.xml postoji. U fajlu se nalazi istorija svih grešaka. Pri prvoj instalaciji aplikacije i pokretanju fajl nece postojati, te će se tada kreirati. Važno je naglasiti da se čuva u internoj memoriji uređaja (smartfona i tableta), što po Android dokumentaciji znači da samo aplikacija koja kreira fajl može i da mu pristupa, što na neki način omogućava dobru zaštitu da korisnik slučajno ne može da ga obriše.Deo za kreiranje fajla ukoliko ne postoji:

```
FileOutputStream fileos;
try {
	fileos = this.getApplicationContext().openFileOutput("istorija.xml",
	Context.MODE_PRIVATE);
	PrintWriter br =  new PrintWriter(fileos);
	br.println(XmlParser.glavniTagZaIstoriju());
	br.close();
	fileos.close();
}catch (IOException e) {
	e.printStackTrace();
}
```

3) Incijalizacija toolBar-a, na kome je prikazano ime aplikacije

4) Kreiranje dugmića za navigaciju po tabovima kao i postavljanje njihovih sličica. TabLayout se kasnije prosleđuje klasi **ViewPager**, a ona se poziva pri pritisku na

nekom od tabova (sličica) kako bi se promenio određeni tab.

```
TabLayout tabLayout = (TabLayout) this.findViewById(R.id.tab_layout);
TabLayout.Tab dugmeUpravljanjeIBezbednosniPodaci = tabLayout.newTab();
dugmeUpravljanjeIBezbednosniPodaci.setIcon(R.drawable.kotur);
```

5) Kreiranje tabova(fragmenata). Dodaju se na nivou aplikacije ali se prosleđuju i klasi **ViewPager** kako bi funkcionisale već pomenute tranzicije. Takođe i GlavnaAktivnost čuva reference ka fragmentima, kako bi npr. komande mogle da pristupe metodama za izmenu vizuelnog prikaza na određenom fragmentu.

6) Pokretanje niti koja hvata greške koje nisu uhvaćene i koje uzrokuju kraj aplikacije (*8.2.1.1.2*)

7) Pokretanje niti GlavnaNit (odsek 8.2.1.8)

8) Pokretanje niti AžuriranjePodataka (odsek 8.2.1.9)

9) Takođe treba napomenuti da se uzimaju reference svih vizuelnih objekata koji su prikazani i vezani za glavnu aktivnost. Npr. slika na statusnoj liniji kao i njen tekst koji su potrebni da se izmene kada dođe do promene stanja.

Pored *onCreate(...)* metode važna metoda je i *onDestroy()*. Ta metoda se poziva npr. pri promeni orijentacije ekrana. Takođe se u ovoj klasi nalazi i obrada greške, gde sve komande nakon dobijanja greške od servera prosleđuju *kod* greške *metodi* `obradaGreske(String kod)`. Ta metoda nalazi opis greške, dodaje je u istoriju grešaka i indirektno prikazuje, sve se to postiže preko klase **MenadzerGresaka** (odeljak 8.2.1.7).

Od izmene GUI-a glavne aktivnosti postoji samo metoda *podesiFooter (EStanjePovezanosti stanje)*, koja u skladu sa stanjem koje se prosledi menja izgled statusne linije.

8.2.1.1.1. Promena orijentacije uređaja

Zbog promene orijentacije uređaja na kome se aplikacija izvršava, urađene su neke stvari. To su:

1) Fragmenti kako je već pomenuto se kreiraju na nivou aplikacije tj dodaju se aplikaciji, tako da klasa **FragmentMenager** vodi računa o njima. Pri prvom pokretanju Glavne Aktivnosti i poziva onCreate(`Bundle savedInstanceState`) metode parametar `savedInstanceState` nije null ukoliko je pozvana metoda više od jedanput čime se dolazi do zaključka da je možda došlo i do promene orijentacije. Dakle ukoliko nije null ne prave se novi fragmenti već se od klase **FragmentMenager** uzimaju postojaći. Na taj način se očuvava stanje fragmenata.

2) U onDestroy() metodi se zaustavljaju pokrenute niti, kako bi u onCreate ponovo započeli svoj rad. Na taj način se izbegava memory leak i pristup pogrešnim ili nepostojaćim objektima. Npr. situacija gde pri promeni orijentacije Android uništava našu glavnu aktivnost, a nit GlavnaNit pokušava baš u tom trenutku da joj pristupi, program će se ugasiti.

8.2.1.1.2. Exception nit [7]

Ova nit je jako korisna tj. ideja je da svaki **exception** koji se desi, a aplikacija ga nije uhvatila, ta nit će ga pokupiti i ugasiti aplikaciju. Ključ je u tome što se tu može lako implementirati da se informacije o greški pošalju razvojnom timu na mail, kako bi se u sledećoj verziji to ispravilo.

8.2.1.2. FragmentUpravljanjeIBezbednosniPodaci

Kao i kod glavne aktivnosti najvažnija metoda je onCreate metoda. Ovde ćemo opisati jedan od tri slična funkcionisanja **Switch** komponente (paljenje uređaja, kamere, lampe) kao i način na koji je implementirana funkcionalnost postavi panele u određen položaj.

1) Kao primer Switch komponente uzećemo funkcionalnost paljenja uređaja sa stanovništva fragmenta. Važan deo se nalazi u onCreate metodi, ali ima i dosta sitnih stvari koje su podjednako važne. Krenućemo od promenljivih koje su važne za ispravno funkcionisanje i njihov detaljan opis.

    1. Sama instanca komponente **Switch**

```
private Switch stanjeUredjaja;
```

U onCreate metodi povezujem instancu sa prikazom na GUI-u

```
this.stanjeUredjaja = (Switch) prikazZaTab1.findViewById(R.id.stanjeUredjaja);
```

Nakon toga povezujemo Listener koji zapravo se poziva pri pritisku ali i pri promeni orijentacije i pravi problem koji će biti opisan. U samom Listeneru tj. metodi koja se poziva pri pritisku, prvo moramo da proverimo da nije u pitanju promena orijentacija panela, a to radimo sa ovom proverom:

```
if (prosloStanjeUredjaja == isChecked) {
      return;
}
```

O ovoj promenljivi će biti više reči kasnije, i još važnija stvar jeste da se zaključa pristup tj. time se obezbeđuje da se svaki pritisak pojedinačno izvrši i na taj način održi stanje

`prosloStanjeUredjaja`. To se obezbeđuje na ovaj način unutar listenera:

```
stanjeUredjajaKljuc.lock();
```

I o ovoj promenljivoj ćе biti više reči. Nakon toga potrebno je samo invertovati stanje i u odnosu na to da li je Switch komponenta čekirana ili nije izvršiti određenu komandu. Dalje se sistem sam brine o tome koja će metoda biti pozvana.
Jako važna stvar koja omoguće očuvanje reference na ključeve je ova linija koda koja iako dođe promene orijentacije uređaja čuva objekte:

```
setRetainInstance(true);
```

2. Prethodno stanje komponente **Switch**

```
private boolean prosloStanjeUredjaja;
```

Postoji samo iz razloga promene orijentacije uređaja jer se dešava da pri promeni dolazi do ponovnog zvanja istog stanja kao da je korisnik ponovo pritisnuo Switch komponentu. Na ovaj način se taj problem rešava, rešenje je pomenuto pri opisu 1. . Takođe stanje je potrebno ažurirati na određenim mestima ako ne uspe pokretanje-gašenje uređaja ali je dosta intuitivno pa nije potrebno detaljnije objašnjavati.

3. Sinhronizacija poziva tj. pritisak na komponentu **Switch**

```
private ReentrantLock stanjeUredjajaKljuc;
```

Pošto samo prethodno opisano stanje nije sinhronizaciono obezbeđeno dolazi do problema ako korisnik uzasatopno, par puta, brzo pritisne Switch komponentu, tada dolazi do gubitka pravog stanja i na dalje se pogrešne poruke prikazuju. Ključ se uzima pri pozivu metode Listenera (opisano iznad), a ključ se vraća pri kraju bilo koje metode koja može da bude pozvana kao posledica pritiska na Switch komponentu. Te metode za uključivanje uređaja su:

```
public void prikazUkljucenUredjajUspesno();
public void ukljucenUredjajNeuspesno();
public void prikazIskljucenUredjajUspesno();
public void iskljucenUredjajNeuspesno();
```

Ove metode će biti opisane nešto kasnije. Sve što je ovde navedeno je isto tako urađeno i za rad sa Lampom i Kamerom.

2) Postavi panele u određen položaj ima nešto lakšu implementaciju. Pri pritisku na dugme Postavi, uzimaju se dva unete vrednost (Azimut i Elevacija) i rade određene provere. Provere su da li su zapravo brojevi uneti i da li su u dobrom intervalu. Ako je sve uredu šalje se određena komanda, a ako nije ispisuje se određena poruka. Nakon pristiglog odgovora od servera biće pozvana ova

metoda:

```
public void pozicioniranjePanelaUspesno(final Polozaj noviPolozajAzimut, final
Polozaj noviPolozajElevacija);
```

Nakon što server odgovori, biće pozvana jedna od gore spomenutih 5 metoda. Njihova funkcionalnost jeste da prikažu odgovarajuću poruku o tome da li je uspela akcija, kao i da postave stanje, ili da postave vrednosti u slučaju pozicioniranje panela. Npr. u slučaju da uređaj nije uspešno upaljen, tj. takav odgovor nam dođe od servera, poziva se metoda `ukljucenUredjajNeuspesno()` i ona treba da vrati prošlo stanje na još jednu unazad i da se sama switch komponenta postavi u ne čekirano stanje i da se na kraju oslobodi ključ. Ovako izgleda impelemntacija te metode:

```
public void ukljucenUredjajNeuspesno(){
	getActivity().runOnUiThread(new Runnable() {
		@Override
		public void run() {
		prosloStanjeUredjaja = false;
		// podesiti switch na staru vrednost
		stanjeUredjaja.setChecked(false);
		stanjeUredjajaKljuc.unlock();
		}
	});

}
```

Ono što je važno, a vodi se kao Napomena na početku ovog poglavlja, jeste da je potrebno sve izvršavati nad GUIThread-om što se postiže na ovaj način (`getActivity().runOnUiThread`).

8.2.1.2.1. Promena orijentacije uređaja

Zbog promene orijentacije uređaja uvedene su dodatne promenljive koje su opisane i njihovo stanje je takođe potrebno održati tokom promene. Za fragmente pred promenu orijentacije i ponovno pozivanje onCreate metode, poziva se metoda

```
public void onSaveInstanceState(Bundle outState);
```

Tu npr. za čuvanje stanja Switch komponente imamo:

```
outState.putBoolean("stanjeUredjaja", this.stanjeUredjaja.isEnabled());
outState.putBoolean("prosloStanjeUredjaja", this.prosloStanjeUredjaja);
```

gde nam npr. prvo služi da ako je čekirano dugme tako i vizualno ostane pri promeni orijentacije.

8.2.1.3. FragmentMeteoPodaci

Ovaj tab je dosta implementaciono jednostavan. U onCreate metodi se povezuju sve komponente.

```
this.vrednostTemperaturaVazduha = (TextView)
prikazZaTab3.findViewById(R.id.vrednostTemperaturaVazduha);
```

Ovu promenljivu kasnije kada stignu novi podaci koristimo kako bi izmenli GUI. To se radi u metodi

```
public void postaviNovePodatke(final double[] noviPodaci);
```

ovom linijom koda

```
vrednostTemperaturaVazduha.setText(String.valueOf(noviPodaci[0]));
```

Ovo je primer za jednu vrednost, ali je isto i za sve ostale.

8.2.1.3.1. Promena orijentacije uređaja

Za sada nema problema za održavanje stanja prilikom promene orijentacije uređaja.

8.2.1.4. FragmentTehničkiPodaci

Ovaj tab je dosta implementaciono jednostavan. U onCreate metodi se povezuju sve komponente. Npr.:

```
this.naponBaterije = (TextView)
prikazZaTab2.findViewById(R.id.vrednostNaponBaterija);
```

Ovu promenljivu, kasnije kada stignu novi podaci, koristimo kako bi izmenli GUI. To se radi u metodi

```
public void postaviNovePodatke(final double[] noviPodaci);
```

ovom linijom koda

```
naponBaterije.setText(String.valueOf(noviPodaci[0]));
```

Ovo je primer za jednu vrednost ali je isto i za sve ostale.

8.2.1.4.1. Promena orijentacije uređaja

Za sada nema problema za održavanje stanja prilikom promene orijentacije uređaja.

8.2.1.5. FragmentIstorijaGrešaka

Za ovaj fragment imamo dve metode:

1) `public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState);`
2) `public void osveziIstorijuGresaka();`

U prvoj metodi radimo incijalizaciju tabele koja je prikaza na GUI-u i njeno učitavanje iz fajla. To radimo na kraju pozivom druge metode `osveziIstorijuGresaka()`. Pri incijalizaciji definišemo ponašanje dva dugmeta:

1. Dugme za brisanje istorije grešaka gde imamo malo dužu liniju za prikaz novog prozora koje će da nas upozori da li stvarno želimo da obrišemo. Ovo je kod tog dela:

```
new AlertDialog.Builder(getActivity(), android.R.style.Theme_Dialog)
 .setIcon(android.R.drawable.ic_dialog_alert)
 .setTitle(R.string.upozorenje)
 .setMessage(R.string.zaistaBrisanje)
 .setPositiveButton(R.string.potvrdanOdgovor, new DialogInterface.OnClickListener() {
 @Override
 public void onClick(DialogInterface dialog, int which) {
                          MenadzerGresaka menadzerGresaka = new
                          MenadzerGresaka(getActivity());
                          menadzerGresaka.obrisiIstoriju();
                          osveziIstorijuGresaka();
                    }
                    })
                    .setNegativeButton(R.string.negativanOdgovor, null)
                    .show();
```

2. Dugme za prikaz pomoći. Tu se startuje nova aktivnost AktivnostPomoć

```
Intent intent = new Intent(getActivity(), AktivnostPomoc.class);
startActivity(intent);
```

Metoda `osveziIstorijuGresaka()` popunjava tabelu redovima grešaka koje čita iz fajla istorija.xml uz pomoć klase MenadzerGresaka. Sve se to radi u jedno *for* petlji.

8.2.1.6. KomandMenadzer

Ova klasa se bavi komunikacijom sa serverom. U konstruktoru može da dobije informaciju o tome ko izvršava komandu (npr. koji fragment) i sam zahtev. Svi zahtevi koje ova klasa može da primi se nalaze u klasi Zahtevi. Takođe, klasa u konstruktoru podešava ip adresu na koju će probati da se konektuje; pretpostavka je da zna na kom portu je server (trenutno na 8080 podešeno). U sklopu kontruktora ulazi i metoda incijalizacija svih komandi (potpis metode `private void inicijalizacijaKomandi();`) u kojoj se u objekat komande, koja je tipa **HashMap,** ubacuju sve komande koje klijent podržava npr.

```
this.komande.put(UkljuciUredjajKomanda.komanda, new
UkljuciUredjajKomanda(this));
```

Važno je napomenuti da je klasa nit, tako da impelementira i metodu run, što ukazaje da će se svaka komanda nezavisno odvijati u odnosu na sistem. Ovo je implementacija run metode:

```
@Override
public void run() {
     // ukoliko klijent nije povezan na bilo kakvu mrezu
     if(this.ipAdresa.equals("")){
          return;
     }
     String odgovorOdServera = this.slanjeZahteva(this.zahtev);
     // ako je doslo do problema prilikom uspostavljanja veze sa serverom
     if(odgovorOdServera.equals("GRESKA_USPOSTAVLJANJE_VEZE")){
          // potrebna provera u slucaju da nema interneta a postavimo na nije povezan
          if(GlobalnoStanje.getSingleton().uzmiStanje() == EStanjePovezanosti.POVEZAN)
GlobalnoStanje.getSingleton().podesiStanjePovezanSaUredjajem(EStanjePovezanosti.NIJE_P
OVEZAN);
          return;
     }else if(odgovorOdServera.equals("GRESKA_KOMUNIKACIJA")){
     }else {
          String komanda = this.zahtev.split(System.getProperty("line.separator"))[0];
          if (this.komande.containsKey(komanda)) {
               this.komande.get(komanda).izvrsi(odgovorOdServera);
          }else{
               // greska : komanda ne postoji
          }
     }
}
```

Dakle, prvo se proverava da li postoji Internet, a zatim se poziva metoda za slanje zahteva.

```
public String slanjeZahteva(String zahtev)
```

Ona obavlja komunikaciju sa serverom ukoliko je moguća. Kao povratnu vrednost vraća ili odgovor servera ili grešku. Ukoliko je sve u redu, tj odgovor nije greška za klijenta, onda se traži odgovarajuća komanda iz već pomenutog objekta komande i poziva se izvršavanje te komande. Ukoliko je došlo do greške npr. `"GRESKA_USPOSTAVLJANJE_VEZE"`, znači da server više nije dostupan i potrebno je postaviti globalno stanje na nije povezan.

8.2.1.7. GlobalnoStanje i EStanjePovezanosti

U klasi GlobalnoStanje se obezbeđuje sigurno očitavanje i upisivanje globalnog stanja koje može da ima tri različite vrednosti i to je definisano u klasi EstanjePovezanosti. Te tri vrednosti su: nema interneta, nije povezan i povezan sa serverom. Jako je bilo bitno da bude sinhronizaciono obezbeđeno globalno stanje jer sa više mesta može da se pristupi tom stanju. U principu, klasa GlobalnoStanje je

singleton jer nema mnogo smila da postoji više od jedne instance bilo kad u životnom ciklusu aplikacije.

8.2.1.8. MenadzerGresaka

Ova klasa sadrži informacije o predefinisanim greškama i greškama koje su se tokom rada aplikacije dogodile. Takođe se ovde definišu greške koje su nastale na klijentu. Npr:

```
public static final String internetNePostoji = "Trenutno ne postoji pristup internetu";
public static final String losFormat = "Uneti parametri nisu validni";
```

U konstruktoru klase se prosleđuje kontekst, koji je važan jer je potreban kasnije XMLParseru da radi operacije čitanja i pisanja u fajl istorija.xml i takođe je vazan za ove dve metode koje prikazuju upozorenja i greške:

```
public void prikaziGresku(String poruka)
public void prikaziUpozorenje(String poruka)
```

Važno je takođe da ove metode budu pozvane unutar GUIThread-a što je naglašeno u API dokumentaciji. Metode koje ova klasa trenutno podržava su čitanje istorije grešaka, dodavanje greške u istoriju grešaka, brisanje svih grešaka iz istorije, čitanje i vraćanje predefinisanih grešaka. Predefinisane greške se nalaze unutar android studio projekta na ovoj putanji app/res/raw u fajlu greske.xml. Na taj način su lako dostupne programu, i generalno bi u tom folderu trebalo da stoje samo statičke stvari. Predefinisane greške i greške iz istorije su u fajlovima definisane kao XML, koji je jasan i čitljiv.

8.2.1.9. GlavnaNit

Uloga ove klase je da:

1) Periodično pinguje sever.

2) Proverava da li ima uopšte interneta, a ukoliko nema onemogućuje dugmiće i unos vrednosti i poziva ispis upozorenja o tome.

3) Ukoliko je došlo do promene stanja, npr. nije bila aplikacija povezana a sada jeste, onda se omogućuju dugmići.

4) U zavisnosti od povezanosti podešava se i statusna linija glavne aktivnosti.

Klasa drži unutar sebe promenljivu u kojoj čuva interno stanje (`private EStanjePovezanosti internoStanjePovezanostiSaUredjajem;`) kako bi na osnovu njega utvrđivalo da li je došlo do promena, direktno vezano za 3) .

8.2.1.10. AzuriranjePodatakaNit

Ova nit ne radi direktno ažuriranje podataka, ali ukoliko je aplikacija povezana sa serverom, ona šalje komande za traženjem meteo i tehničkih podataka, koje kad stignu od servera, unutar komanda će biti pozvane metode za prikaz novih podataka (MeteoPodaci 8.2.1.3. i Tehnički podaci 8.2.1.4.).

8.2.1.11. Komande

Sve komande su klase koje nasleđuju interfejs Komanda i implementiraju metodu `izvrsi()`. Takođe je važno da svaka komanda ima unikatno ime pod kojim će KomandMenadzer (8.2.1.5.) moći da je sačuva.

Primer klase UkljuciUredjajKomanda:

```
public class UkljuciUredjajKomanda implements Komanda{
    // cuvamo instancu komandMenadzera, jer on sadrzi
    // objekat koji je poslao ovaj zahtev
    private KomandMenadzer komandMenadzer;
    // tacan nayiv komande, moraju da se imena komandi podudaraju
    // na klijentu i serveru
    public static final String komanda = "UKLJUCI_UREDJAJ";
    public UkljuciUredjajKomanda(KomandMenadzer komandMenadzer){
        this.komandMenadzer = komandMenadzer;
    }
    @Override
    public void izvrsi(String odgovorServera) {
        String statusOdgovora = Usluge.statusOdgovora(odgovorServera);
        // ovde mozemo da izvrsimo cast jer znamo ko ustvari salje ovu komandu
        FragmentUpravljanjeIBezbednosniPodaci fragmentUpravljanjeIBezbednosniPodaci =
(FragmentUpravljanjeIBezbednosniPodaci) komandMenadzer.getPozivaoca();
        if(statusOdgovora.equals("OK")){
            // ukoliko je sve OK samo pozovemo metodu iz fragmenta
            fragmentUpravljanjeIBezbednosniPodaci.prikazUkljucenUredjajUspesno();
        }else if(statusOdgovora.equals("NOK")){
            // ako je doslo do greske na serveru
            // onda uzimamo kod greske, dobijamo
            // referencu ka glavnoj aktivnosti preko
            // getActivity(), obradjujemo gresku
            // i obavestavamo fragment o neuspehu
            String kodGreske = Usluge.kodGreske(odgovorServera);
            GlavnaAktivnost glavnaAktivnost = (GlavnaAktivnost)
fragmentUpravljanjeIBezbednosniPodaci.getActivity();
            glavnaAktivnost.obradaGreske(kodGreske);
            fragmentUpravljanjeIBezbednosniPodaci.ukljucenUredjajNeuspesno();
        }
    }
}
```

8.2.1.12. AktivnostPomoć

Ova aktivnost ima samo jednu metodu onCreate, u kojoj se učitavaju sve predefinisane greške preko menadžera grešaka. Nakon toga se popunjava tabela učitanim greškama. Postoji dugme nazad nakon koga se završava aktivnost i korisnik se vraća na glavni deo.

8.2.1.13. Ostale klase

Ostale klase su jako intuitivne, tako da neće biti detaljnih opisa. Na osnovu njihovih naziva i naziva metoda može se naslutiti čemu služe.

1) `enum EtredStatus` – Stanja u kojima može tred da se nađe

2) `class GreskaIzIstorije` – Tip greške, sadrži polja: datum i kod

3) `class GreskaPredefinisana` – Tip greške, sadrži polja: opis i kod

4) `class Konektivnost` – Klasa koja pruža informacije o trenutno aktivnoj konekciji ka internetu

5) `class FragmentAdapter` – Koristi se u sklopu dela za prikaz određenog fragmenta

6) `class Polozaj` – Iako se klasa ne koristi u potpunosti, ideja je da se kasnije za prikaz bilo kog položaja omogući laka upotreba stepeni, minuta i sekundi. Trenutno se koriste samo stepeni

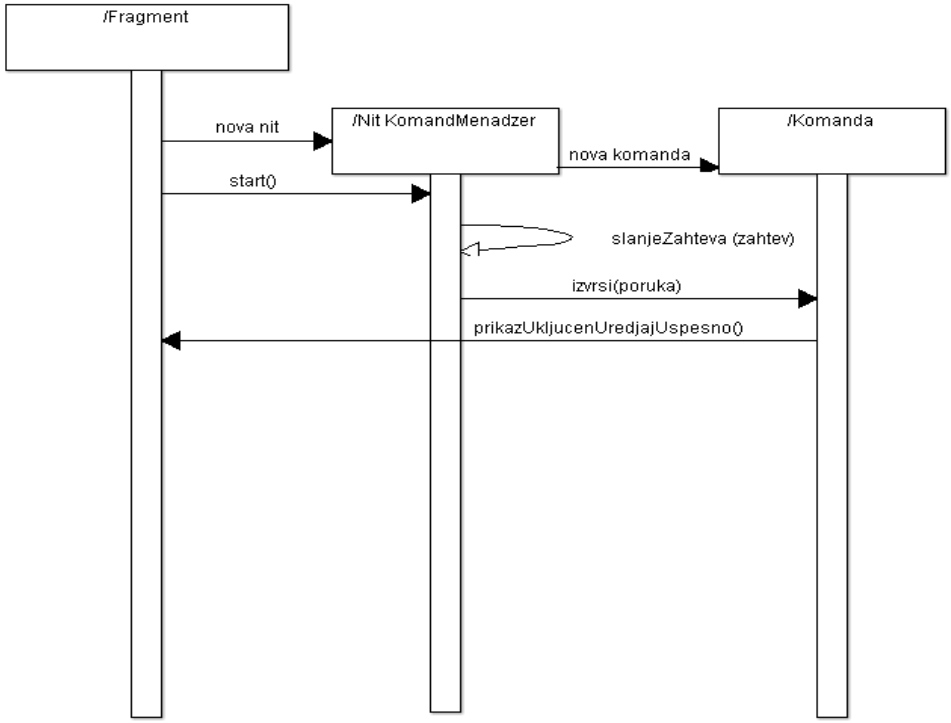
7) `class Usluge` – U ovoj klasi se nalaze metode koje nisu našle pravo mesto

8) `class XmlParser` – Ova klasa je zadužena za xml parsiranje fajlova [8]

9) `class Zahtevi` – U ovoj klasi se nalazi lista svih zahteva koje klijent aplikacija može da uputi serveru

9. UML DIJAGRAMI

Ovo poglavlje trenutno sadrži samo dijagram sekvence. Na njemu je prikazana sekvenca akcija koja se desi kada korisnik pritisne ili pomeri Swipe komponentu na Fragmentu Upravljački i bezbednosni podaci koja je zadužena za paljenje uređaja. Pretpostavka je da je server odgvorio sa potvrdnom porukom.



Slika 9.3 Dijagram sekvence za pomeraj-pritisak na komponentu Switch, odgvornu za paljenje Uređaja

Sve ostale akcije koje klijent podržava se odvijaju na isti način, samo se nazivi metoda i komandi razlikuju.

DEO III – UPUTSTVA

10. UPUTSTVO ZA POKRETANJE

Da bi se pokrenuo klijent, potrebno je prvo aplikaciju instalirati na uređaju (smartfonu, tabletu). I server i klijent prate isti proces instalacije. Pokreću se kao i bilo koja druga aplikacija na androidu (isto je na virtuelnoj mašini i na fizičkom uređaju). Da bi se videla puna funkcionalnost klijenta potrebno je na serveru kliknuti dugme Start kako bi počeo sa radom. Takođe, projekat iz Android Studio-a se može pokrenuti tako što se ode na karticu Run, opcija Run (Shift+F10) i izabere aktivan android uređaj (to može da bude virtualna mašina ili fizički uređaj). Na taj način se kombinuje automatska instalacija i pokretanje.

11. POTPISIVANJE APLIKACIJE

Bilo da je testiranje ili distribucija softvera, svaka aplikacija mora da bude potpisana. Kada se iz Android Studio-a pokreće proces instalacija+pokretanje aplikacije, na Android uređaju automatski se dodeljuje ključ tj. potpis. Za potipsivanje prilikom distribucije softvera potrebno je uraditi sam proces generisanja ključa i sačuvati ga negde. U ovom radu se nije išlo dalje od testiranja i automatskog dobijanja ključa. Više o tome ima na android developer sajtu:
http://developer.android.com/tools/publishing/app-signing.html

12. UPUTSTVO ZA KOMPAJLIRANJE

Što se tiče projekata klijent i server za kompajliranje nije ništa potrebno osim u klijentu što je već napisano u build.gradle (Module: app) bude naznačeno:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.1'
    compile 'com.android.support:design:22.2.0'
}
```

To su biblioteke koje su korišćene u projektu klijenta. Na serveru nema takvih zahteva. Za fajlove na klijentu nema nikakvih dodatnih podešavanja. Postoje samo dva gde je jedan već unutar projekta, a drugi se naknadno pravi pri samoj instalaciji.

Kompajliranje se može podeliti u dve grupe:

1) Kompajiralnje u cilju testiranja

Kompajliranje(Build) projekta može da se uradi u Android Studio-u, a na jako jedostavan način. Otvori se npr. projekat za klijent, kartica Build, opcija Make Project (Ctrl+F9). Pri isprobavanju ove opcije i pregledu log-a iz Android Studio-a došlo se do zaključka da ova komanda zapravo ne pravi .apk fajlove. Umesto toga, praćenjem instrukcija za pokretanje aplikacije preko Run-a pravi se odgovarajući .apk fajl.
Fajl se nalazi u imeProjekta/app/build/outputs folderu pod nazivom app-debug.apk. Iz naziva se vidi da je svrha testiranje. Ovo isto može da se postigne i preko konzole tj. command prompt-a. Potrebno je da JAVA_HOME promenljiva bude podešena kao promenljiva okruženja. Iz konzole to može da se uradi ovom komandom:

```
set JAVA_HOME=c:\Progra~1\Java\<jdkdir>
```

, gde je jdkdir mesto gde je instaliran Java jdk. Nakon toga treba otići u root folder projekta i ukucati sledeću komandu (na Windows):

```
gradlew.bat assembleDebug
```

Nakon što se izvrši kompajliranje biće isti .apk fajl pod već pomenutom putanjom.

2) Kompajliranje u cilju distribucije softvera

Ova vrsta kompajliranja se može postići iz Android Studio-a tako što se ode na karticu Build, opcija Generate Signed Apk. Nakon toga se otvara prozor gde treba popuniti određene informacije. Može da se izabere fajl gde se nalazi ključ i da se ukuca šifra. To isto može i preko konzole da se postigne. [9]

Kompajliranju i pokretanju iz Android Studio-a [10]

13. INSTALACIJA

Nakon što se dobije .apk fajl, postoje dva načina za distribuciju.
1) Google Play Store

Pretpostavka da je mnogo jednostavnije updejtovanje softvera, pošto sve ide preko njihovog sistema. Nema problema nesigurnog izvora.
2) Sopstveni server

Na ovaj način bi .apk fajl stajao na serveru, a preuzimao se direktno preko Android uređaja. Mana ovog pristupa je što mora da se podesi na telefou preuzimanje

android instalacionih fajlova koji nisu sa sigurnih izvora. Druga mana je što ceo sistem updejtovanja softvera mora da se implementira.

14. UPUTSTVO ZA NASTAVAK RAZVOJA

14.1. Izmene na GUI-ju

S obzirom na način implementacije vizuelnog prikaza svih fragmenata i aktivnosti (trenutno samo AktivnostPomoc), dodavanje na GUI-u je krajnje jednostavno. Komponente se samo dodaju jedna ispod druge, a ScrollView omogućuje da one sve budu vidljive skrolom. Najfleksibilinije je definisanje novih komponenti direktno u xml Text delu.

14.2. Dodavanje greški

Dodavanje novih greški, koje se šalju sa servera i koje se prikazuju na formi pomoć se lako rešava. Naime, sve greške se nalaze u fajlu *greške.xml*, koji se nalazi u folderu *raw (Odeljak* 8.2.1.7). Te greške se u programu parsiraju i prikazuju korisniku. Ne bi trebalo menjati format u kome se greške prikazuju, koji izgleda ovako:

```
<?xml version="1.0" encoding="utf-8"?>
<greske>
    <greska>
        <kod>100</kod>
        <opis>Nije uspelo pokretanje MobiSun uređaja. Pokušajte ponovo.</opis>
    </greska>
 ....
</greske>
```

Za novu grešku, dodati tag oblika:

```
<greska>
      <kod>noviKod</kod>
      <opis>noviOpis</opis>
</greska>
```

14.3. Izmena teksta greški koje se prikazuju korisniku

Pošto bi trebalo da se sve greške prikazuju kroz klasu *MenadzerGresaka*, ovo nije težak zadatak. U ovoj klasi, postoje polja, za svaku grešku, koje čuvaju tekst greške. Potrebno je samo na ovom mestu izmeniti tekst i on će biti promenjen. Takođe, za dodavanje nove greške ili teksta je sličan proces, potrebno je dodati novo polje sa tekstom greške, i grešku prikazati pozivom metoda iz ove klase. Primer:

```
public static final String internetNePostoji = "Trenutno ne postoji pristup
internetu";
```

Za prikaz greške koristiti metodu, gde je String poruka neka od gore navedenih statičkih polja:

```
/**
 * Potrebno je obezbediti da se metoda izvrsava nad UI tredom
 * Ukoliko se ne obezbedi nista nece biti ispisano u obliku toast poruke
 * @param poruka
 */
public void prikaziUpozorenje(String poruka){
	// provera da li se ne nalazimo u UI tredu
	if(Looper.myLooper() == Looper.getMainLooper() == false)
		return;
	Toast toast = Toast.makeText(this.kontekst , poruka, Toast.LENGTH_LONG);
	toast.show();
}
```

14.4. Izmena intervala tajmera

Svaki tred ima u sebi definisanu promenljivu koja određuje koliko će spavati, što lako može da se promeni ako je potrebno.

```
class AzuriranjePodatakaNit
private int vremeSpavanja = 500
```

Vrednost je u milisekundama.

14.5. Dodavanje komandi

Ukoliko dođe do potrebe za novim komandama, one se mogu jednostavno dodati. Prvo je potrebno napraviti novu klasu, koja implementira interfejs *Ikomanda().*
Primer ovakve jednostavne klase je *PingKomanda* klasa*:*

```
ublic class PingKomanda implements Komanda{
 // tacan nayiv komande, moraju da se imena komandi podudaraju
 // na klijentu i serveru
 public static final String komanda = "PING";
 public PingKomanda(){}

 @Override
 public void izvrsi(String odgovorServera) {
 if(Usluge.statusOdgovora(odgovorServera).equals("OK")){
GlobalnoStanje.getSingleton().podesiStanje(EStanjePovezanosti.POVEZAN);
 }else if(Usluge.statusOdgovora(odgovorServera).equals("NOK")){
 // u slucaju da nesto nije u redu sa severom i ne moze da prima dalje konekcije
 // ima smisla da ovde stoji podesavanje na nije_povezan ili da se uvede
 // novo stanje cekaj
 }
 }
}
```

String komanda čuva tačan naziv komande. Kada server pošalje isti takav string, to znači da ova klasa treba da se pozove. Metoda *izvrši()*, treba da izvrši tu komandu. Drugi korak koji je potrebno uraditi je da se doda nova komanda u listu svih komandi. Ovo se radi u konstruktoru klase *KomandMenadzer*. Primer:

```
this.komande.put(PingKomanda.komanda, new PingKomanda());
```

Na ovaj način smo obezbedili da kada server pošalje komandu, koja je u promenljivoj *String komanda*, ta klasa, tj njena metoda *izvrši* će biti pozvana.

14.6. Dodavanje metoda koje se pozivaju van GuiThread niti

Ovo je već donekle objašnjeno tokom rada (Odeljak 8.2.1. napomena 2.). U principu svaka metoda koja treba da radi neke izmene na vizuelnom prikazu potrebno je da se to izvršava u GuiThread-u. To se obezbeđuje preko *runOnUiThread* metode i unutar tog novog treda smo sigurni da smemo da vršimo izmene. Postoje i drugi načini koji obezbeđuju istu stvar.

15. TESTIRANJE APLIKACIJE

Aplikaciju bi trebalo da podržavaju svi uređaji koji na sebi imaju minimalno Android SDK 15. Po proceni Google App Store-a, broj uređaja koji podržava to je 94.5%, što bi trebao da bude prihvatljiv broj.

Testiranje možemo podeliti na dve grupe. Vizuelno testiranje i funkcionalno testiranje. Trenutno je na fizičkom android uređaju Samsung Galaxy S5 testirano i jedno i drugo, dok je na virtuelnoj mašini Nexus 7ice testirano samo vizualni prikaz.

16. OPIS PORUKA PRI KOMUNIKACIJI KLIJENTA I SERVERA

Sve poruke koje klijent šalje su ovakvog oblika:

ImeKomande
(novi red)
Vrednosti (0 ili više) (za svaku vrednost ide novi red)
(novi red) (KomandMenadzer dodaje da bi definisao kraj poruke)

Sve poruke koje server šalje su oblika:

OK
(novi red)
Vrednosti (0 ili više) (za svaku vrednost ide novi red)
(novi red) (definisanje kraja poruke)

ili

NOK
(novi red)
KOD_GRESKE
(novi_red)
(novi_red) (definisanje kraja poruke)

17. ZAKLJUČAK

U ovom radu, predstavljena je potpuno funkcionalna GUI aplikacija za upravljanje i nadzor MobiSun uređaja. U izradi aplikacije korišćen je programski jezik Java. Tokom izrade same aplikacije, jedan od većih izazova bio je da se kod lepo organizuje, tako da se omogući jednostavan nastavak razvoja aplikacije, kao i jednostavne izmene postojećih funkcionalnosti. Drugi veliki izazov bio je da se aplikacija prilagodi tehnički ne obučenim licima, i da se omogući njeno jednostavno korišćenje, bez prethodnog učenja. Ovo je urađeno kroz jednostavan, konzistentan i intuitivan GUI. Aplikacija, iako potpuno funkcionalna, predstavlja dobru bazu za dalji razvoj i dodavanje novih funkcionalnosti, čime će se korisnicima omogućiti što jednostavnije i prijatnije upravljanje MobiSun uređajem.

Literatura

1) Java - http://www.oracle.com/technetwork/java/index-138747.html
2) Android API dokumentacija - https://developer.android.com/reference/packages.html
3) IntelliJ IDEA - https://www.jetbrains.com/idea/features/
4) Gimp2 - http://docs.gimp.org/2.8/en/
5) ArgoUML - http://ftp.stu.edu.tw/BSD/FreeBSD/ports/distfiles/argouml/manual-0.34.pdf
6) Android podrška uređaja raznih veličina - http://developer.android.com/guide/practices/screens_support.html
7) Obrada i prikupljanje o ne uhvaćenom exceptionu - http://developer.android.com/reference/java/lang/Thread.UncaughtExceptionHandler.html
8) XML parsiranje - http://www.ibm.com/developerworks/library/x-android/
9) Android kompajliranje - http://developer.android.com/tools/building/building-cmdline.html
10) Android Studio kompajliranje i pokretanje - http://developer.android.com/tools/building/building-studio.html