Mobilna aplikacija za računanje režijskih troškova

Vukosavić, Danijel

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Danijel Vukosavić

Mobilna aplikacija za računanje režijskih troškova

Pula, rujan, 2022

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Danijel Vukosavić

Mobilna aplikacija za računanje režijskih troškova

Diplomski rad

JMBAG: 0303069617, redoviti student
Studijski smjer: Informatika

Predmet: Mobilne aplikacije
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv.prof. dr. sc. Siniša Sovilj

Pula, rujan, 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Danijel Vukosavić**, kandidat za **magistra informatike** ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan 2022. godine

IZJAVA
o korištenju autorskog djela

Ja, Danijel Vukosavić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „ Mobilna aplikacija za računanje režijskih troškova“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Student

_____

U Puli, rujan, 2022.godine

SVEUČILIŠTE JURJA DOBRILE U PULI
FAKULTET INFORMATIKE

Pula, 28. ožujka 2021.

DIPLOMSKI ZADATAK

Pristupnik: **Danijel Vukosavić (2711997391927)**
Studij: Sveučilišni diplomski studij Informatike

Naslov (hrv.): **Mobilna aplikacija za računanje režijskih troškova.**
Naslov (eng.): Mobile application for computing utilities expenses.

Opis zadatka: Zadatak je izraditi mobilnu aplikaciju za računanje režija. Korisnik će imati opciju registracije i prijave. Korisnik će imati mogućnost obrisati svoj račun ili promijeniti lozinku i email ili korisničko ime. Korisnik će imati mogućnost računanja troškove vode, struje i plina s obzirom na mjesto stanovanja. Iste te račune će moći spremiti te kasnije imati uvid te će moći provjeriti prosječnu potrošnju režija. Aplikacija će sadržavati opciju spremanja i drugih računa kao što su: komunalne naknade, pričuva, smeće, internet, mobilne pretplate itd. Korisnik će u svakom trenutku imati mogućnost i izbrisati neželjene račune.
Aplikacija se treba temeljiti na tehnologijama Kotlin ili Java s Firebase bazom.
Opis sustav: korisničke scenarije, funkcionalnosti, izraditi potrebne UML dijagrame - klasne, use case, use sequence, opisati implementaciju te na kraju izraditi kratke korisničke upute.

Zadatak uručen pristupniku:
28. ožujka 2021.
Rok za predaju rada:
28. veljače 2022.

Mentor:
Siniša Sovilj
doc.dr.sc. Siniša Sovilj

Sadržaj

1.UVOD

90 % svjetskog stanovništva danas plaća određene režije kao što su voda, plin, struja ali i ostale režije kao što su : Komunalne usluge, Internet , mobilne pretplate.

Mobilna android aplikacija **My Utilities** je aplikacija koja drži sve navedene režije na jednom mjestu. Korisnik može unijeti stanja unutar aplikacije te tako dobiti uvid koliko će morati platiti za određene režije. Prednost ove aplikacije u odnosu na druge aplikacije je ta što korisnik ima sve režije na jednom mjestu, te ih tako može spremiti i na kraju imati uvid u sve svoje spremljene režije. Korisnik ima uvid samo u svoje spremljene režije, te ih sprema unutar **Firebase Firestore** baze podataka.

Korisnik je obvezan kreirati korisnički račun unutar aplikacije da bih mogao koristiti aplikaciju. Kada korisnik kreira korisnički račun potrebno je se ulogirati. Kada se korisnik ulogira unutar izbornika aplikacije može birati koje režije želi izračunati ili spremiti. Osim samog spremanja režija korisnik ih također može i izbrisati iz baze podataka tako da unese broj računa unutar sučelja mobilne aplikacije.

Aplikacija također ima i restrikcije koje korisniku onemogućuju unos ne ispravnih email formata. Isto tako korisnik mora unijeti sva polja da bih mogao kreirati račun, izračunati i spremiti režije. Sve restrikcije se ispišu na ekran mobilnog uređaja kako bih mogle upozoriti korisnika na određene pogreške

Ostale režije kao što su: Komunalne usluge, Internet i mobilne pretplate je moguće jedino spremiti unutar baze podataka zato što su te režije gotovo uvijek fiksne, te ih nije moguće izračunati.

U nastavku rada detaljno će sve biti objašnjeno kako je sama aplikacija izrađena, te koje sve tehnologije su korištene u svrhu izrade aplikacije.

2.Programi i alati za izradu aplikacije

Prije same izrade mobilne aplikacije potrebno je proučiti koje područje bi najbolje odgovaralo samoj izradi mobilne aplikacije. Ovo je mobilna aplikacija, te tako imamo više alata za samu izradu. Najbolji alati za izradu aplikacije su:

- Android studio
- Visual studio
- Eclipse
- IntelliJ IDEA

2.1 Android Studio

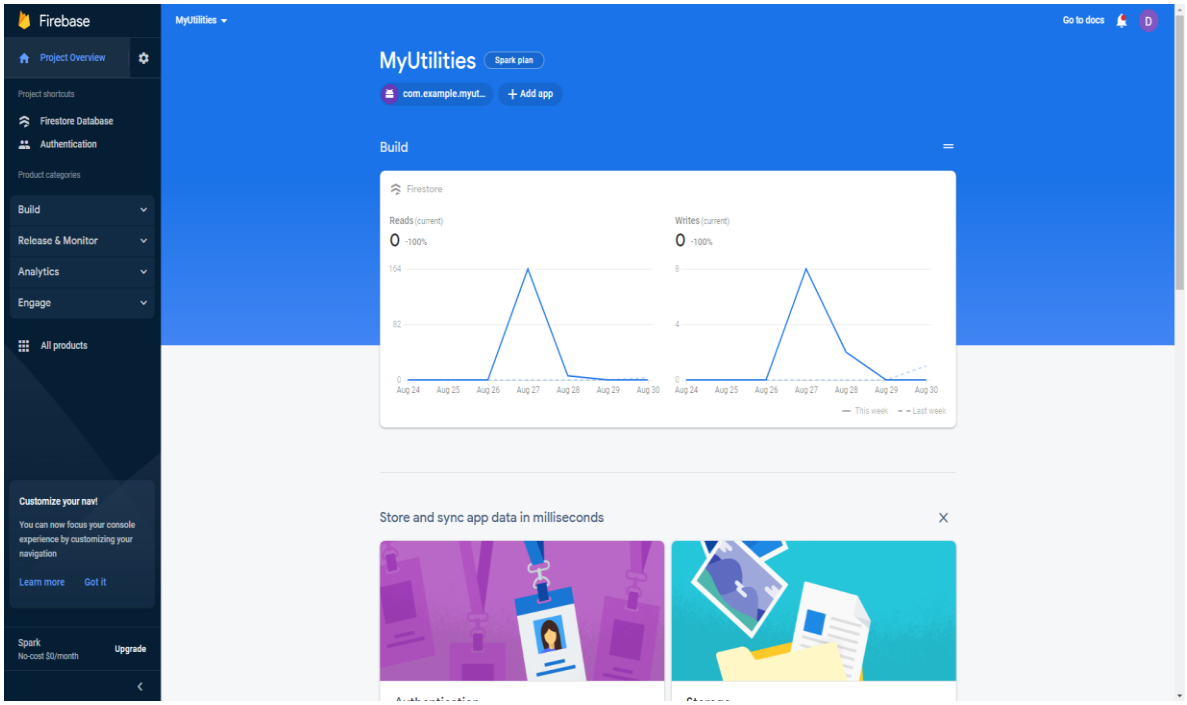
Android Studio službeno je integrirano razvojno okruženje za Googleov operativni sustav Android, izgrađeno na JetBrainsovom IntelliJ IDEA softveru i dizajnirano posebno za Android razvoj. Dostupan je za preuzimanje na Windows, macOS i Linux operativnim sustavima. Android studio nudi pregršt značajki za izradu same aplikacije.Android studio također može raditi i aplikacije za IOS uređaje. Neke od značajki android studia su:

- Brzo kodiranje i brza iteracija
- Brz emulator bogat značajkama.
- Firebase podrška i integrirani oblak.
- Lako korištenje GitHub sučelja unutar samog alata
- Savršen alat za timski rad.

2.2 Firebase

Google Firebase je softver za razvoj aplikacija uz podršku Googlea koji programerima omogućuje razvoj iOS, Android i web aplikacija. Firebase pruža alate za praćenje analitike, izvješćivanje i popravljanje grešaka koje dovode do prestanka rada aplikacije, stvaranje marketinških i eksperimentalnih proizvoda.

Firebase pomaže razvijanju visokokvalitetne aplikacije, povećanje baze korisnika i zarade novca. Svaka značajka radi neovisno, a još bolje rade zajedno.



1.Firebase korisničko sučelje

2.2.1 Firebase Firestore Database

Firestore je NoSQL baza podataka dokumenata izgrađena za automatsko skaliranje, visoku izvedbu i jednostavnost razvoja aplikacija. Iako Firestore sučelje ima mnoge iste značajke kao i tradicionalne baze podataka, kao NoSQL baza podataka razlikuje se od njih po načinu na koji opisuje odnose između podatkovnih objekata.

2.2.2 Firebase RealTime Database

Firebase Realtime Database je NoSQL baza podataka smještena u oblaku koja vam omogućuje pohranu i sinkronizaciju podataka između vaših korisnika u stvarnom vremenu. Cloud Firestore vam omogućuje pohranu, sinkronizaciju i upite podataka aplikacija na globalnoj razini.

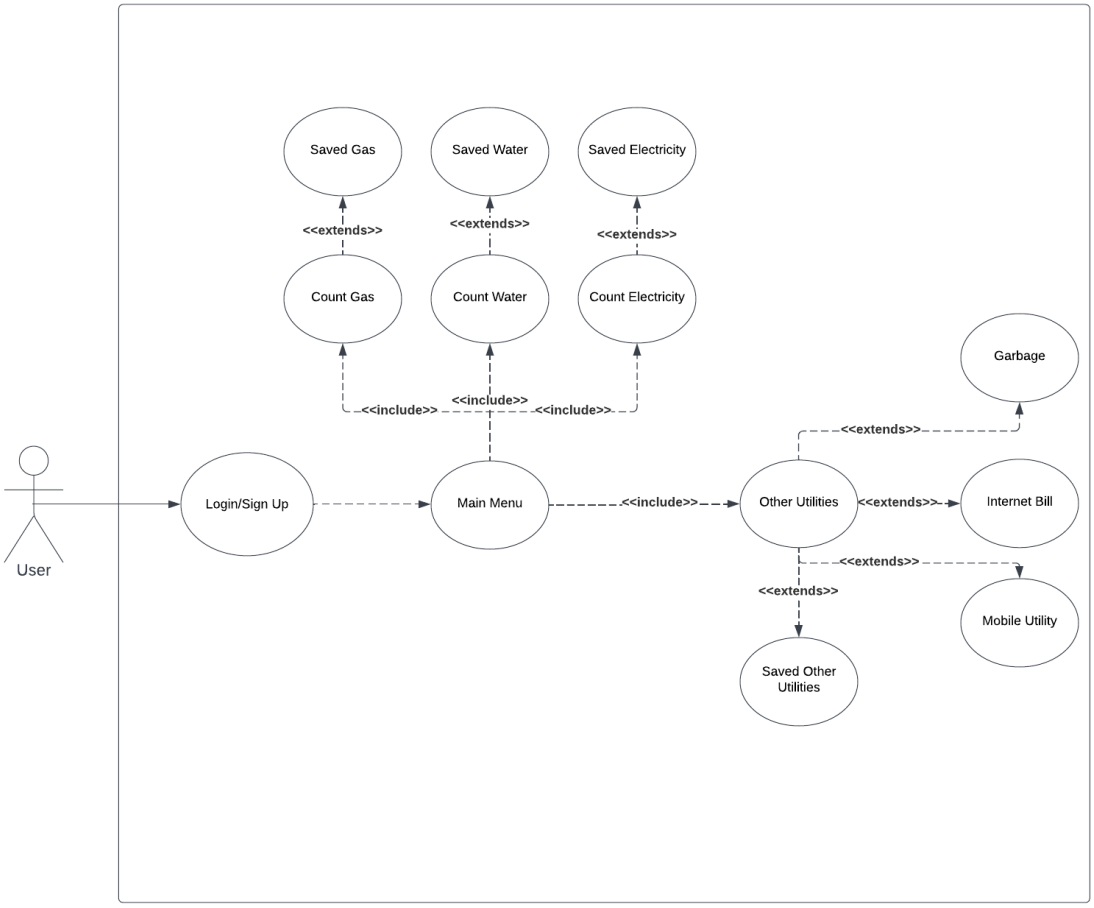
2.2.3 Firebase Authentication

Firebase Authentication pruža pozadinske usluge, SDK-ove jednostavne za korištenje i gotove UI biblioteke za autentifikaciju korisnika u vašoj aplikaciji. Podržava autentifikaciju korištenjem lozinki, telefonskih brojeva, popularnih pružatelja federalnih identiteta kao što su Google, Facebook i Twitter i više

3. Dijagrami

3.1 Dijagram slučaja (Use case diagram)

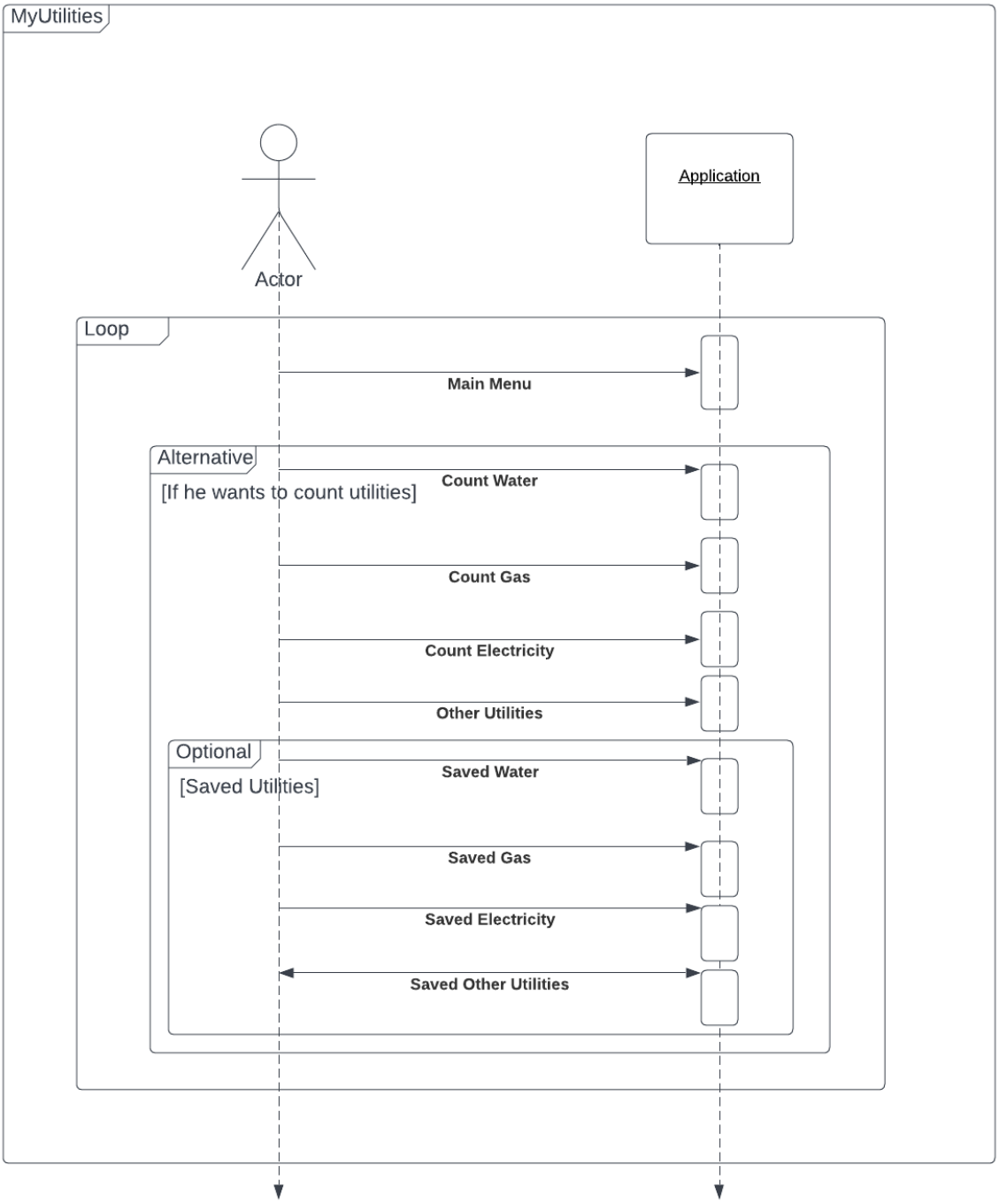
Dijagram slučaja sadrži jednog aktera , a to je korisnik. Korisnik ima mogućnosti računanja i spremanja režija , ali isto tako korisnik može pregledati svoje spremljene režije i ako želi iste obrisati iz baze podataka. Korisnik unutar izbornika ima mogućnost izabrati 4 opcije: Računanje vode, struje ,plina ili pohranu ostalih režija. Unutar svih tih opcija korisnik također može kliknuti na gumb koji ga vodi na ispis tih režija, te ako želi ima mogućnost i brisanja samih režija. Slika 2. Prikazuje interakciju korisnika s aplikacijom



Slika 2. Dijagram slučaja(Use case diagram)

3.2 Dijagram slijeda (Sequence diagram)

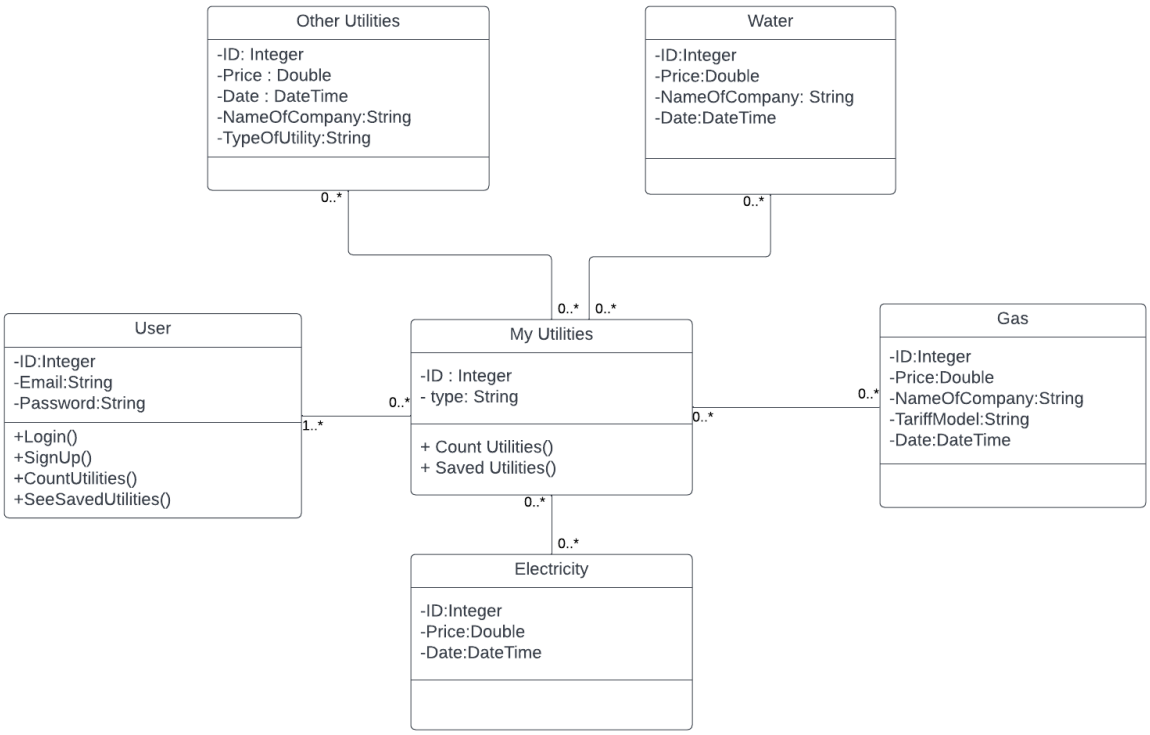
Dijagram slijeda je dijagram koji prikazuje samo korisnika. Korisnik kada se prijavi ulazi u izbornik. Unutar izbornika korisnik bira što želi sljedeće. Ako korisnik se odluči izabrati računanje vode, tu može izračunati potrošnju , te isto tako i spremiti izračun unutar baze podataka. Unutar svih tih opcija korisnik također ima opciju i provjeru spremljenih režija, te isto tako ih i obrisati. Slika 3. Prikazuje ilustraciju dijagrama slijeda.



Slika 3. Dijagram slijeda(sequence diagram)

3.3 Klasni dijagram(class diagram)

Klasni dijagram prikazuje cijelu strukturu sustava. Dijagram objašnjava sve klase unutar sustava, njihove atribute i funkcije, ali isto tako pokazuje i odnose između klasa. Npr. Ako izbrišemo MyUtility klasu brišu se i sve ostale klase osim klase User. Sve klase isto tako imaju i kardinalnosti. Korisnik može unijeti jednu ili više režija, dok režije mogu unijeti nula korisnika. Slika 4. Prikazuje klasni dijagram

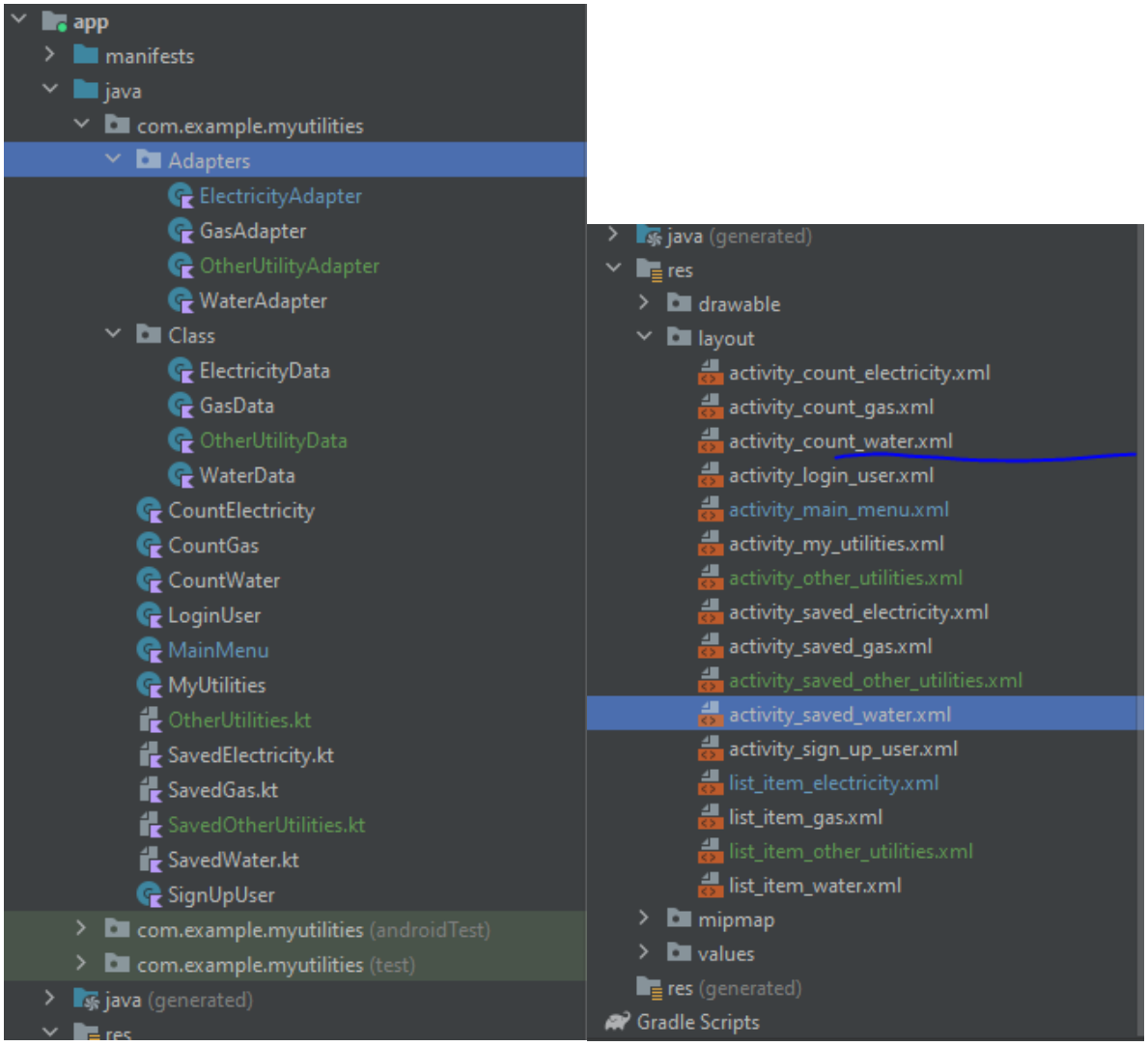


Slika 4. Klasni dijagram(class diagram)

4. Struktura projekta

Projekt u Android Studiju sadrži sve što definira vaš radni prostor za aplikaciju, od izvornog koda i sredstava, do testnog koda i konfiguracija izgradnje. Kada pokrenete novi projekt, Android Studio stvara potrebnu strukturu za sve vaše datoteke i čini ih vidljivima u prozoru projekta na lijevoj strani IDE-a.

Modul je zbirka izvornih datoteka i postavki izgradnje koje vam omogućuju da svoj projekt podijelite na diskretne jedinice funkcionalnosti. Projekt može imati jedan ili više modula, a jedan modul može koristiti drugi modul kao ovisnost. Svaki modul možete samostalno izgraditi, testirati i ispravljati pogreške.



Slika 5. Struktura projekta

Slika 5. prikazuje samu strukturu projekta. Datoteka manifest sadrži unutarnju datoteku AndroidManifest.xml. AndroidManifest.xml opisuje bitne informacije o vašoj aplikaciji alatima za izradu Androida, operativnom sustavu Android i Google Playu.

XML je temeljen na Standard Generalized Markup Language (SGML) koji se koristi za definiranje “markup” jezika. Primarna funkcija XML-a je stvaranje formata za podatke koji se koriste za kodiranje informacija za dokumentaciju, zapise baze podataka, transakcije i mnoge druge vrste podataka.

Java sadrži sve datoteke izvornog koda Kotlina.

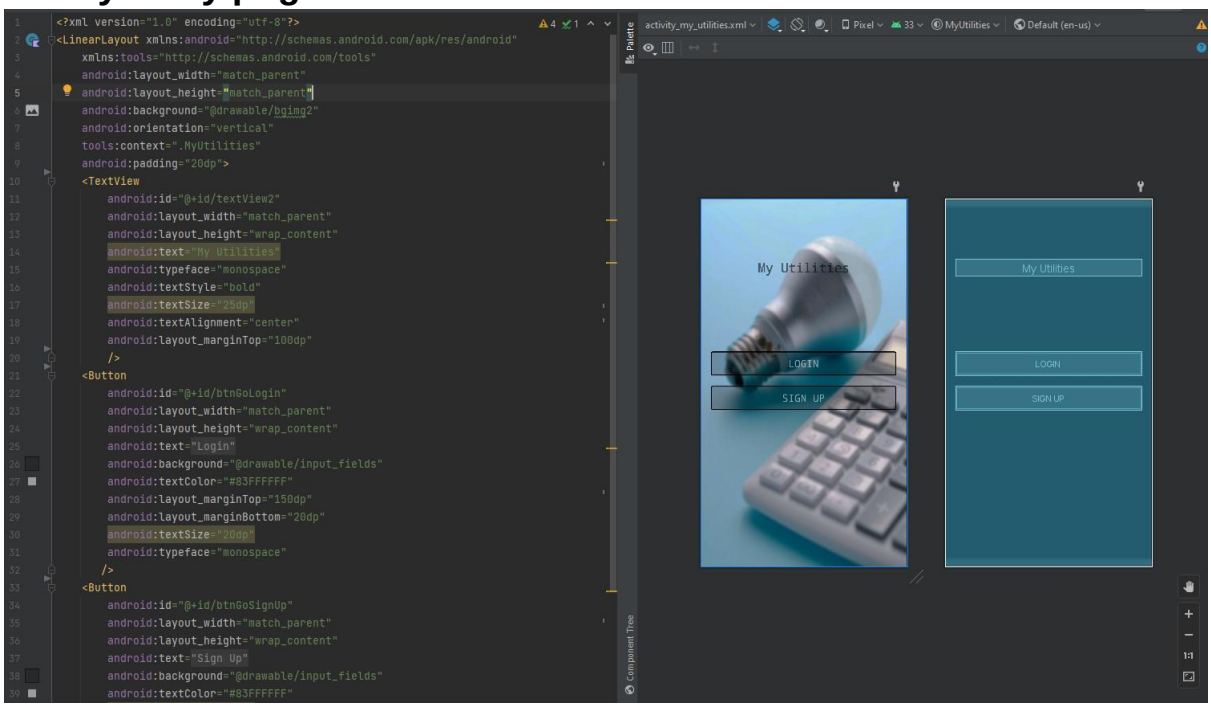
U Androidu, Adapter je poveznica između UI komponente i izvora podataka koji pomaže ispuniti podatke u UI komponenti. Sadrži podatke i šalje ih u prikaz adaptera, a zatim pogled može uzeti podatke iz prikaza adaptera i prikazati podatke na različitim prikazima kao što su ListView, GridView, Spinner itd.

Class folder sadrži sve klase projekta.

Layout izgled definira strukturu korisničkog sučelja u samoj aplikaciji, kao što je aktivnost. Svi elementi u layotu izgrađeni su pomoću hijerarhije objekata View i ViewGroup. Pogled obično crta nešto što korisnik može vidjeti i s čime može komunicirati. Layout je zapravo sam dizajn aplikacije.

5. Aktivnosti i pogledi za računanje i spremanje potrošnje režija

5.1 MyUtility pogled i aktivnost



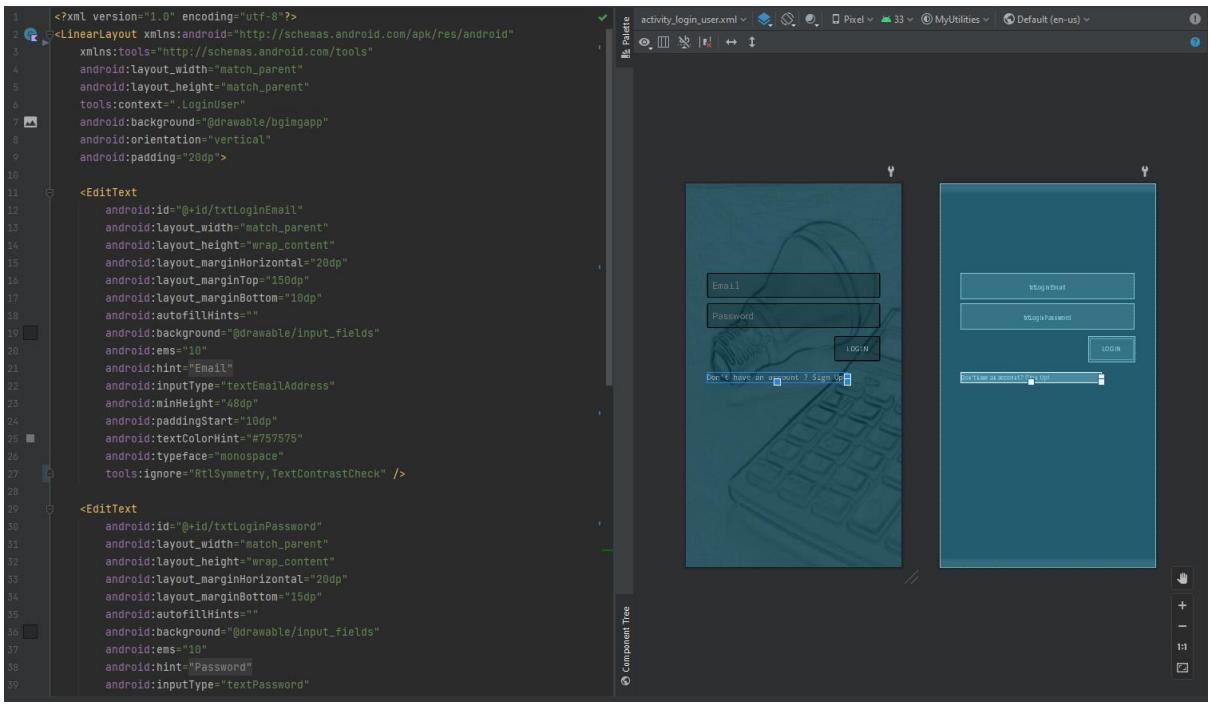
Slika 6. MyUtility pogled

Slika 6. prikazuje xml kod i dizajn my utility pogleda. Korisnik unutar ovog pogleda bira da li se želi registrirati na aplikaciju ili prijaviti. Slika 7. prikazuje funkcionalnost koda.

```
package com.example.myutilities


import ...


class MyUtilities : AppCompatActivity() {
    private lateinit var binding: ActivityMyUtilitiesBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMyUtilitiesBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.btnGoLogin.setOnClickListener {  it: View!
            val intent = Intent( packageContext: this,LoginUser::class.java)
            startActivity(intent)
        }
        binding.btnGoSignUp.setOnClickListener {  it: View!
            val intent = Intent( packageContext: this,SignUpUser::class.java)
            startActivity(intent)
        }


    }

}
```

Slika 7. Funkcionalnost MyUtilities aktivnosti.

5.2 Login i registracija



Slika 8. pogled prijave i xml kod

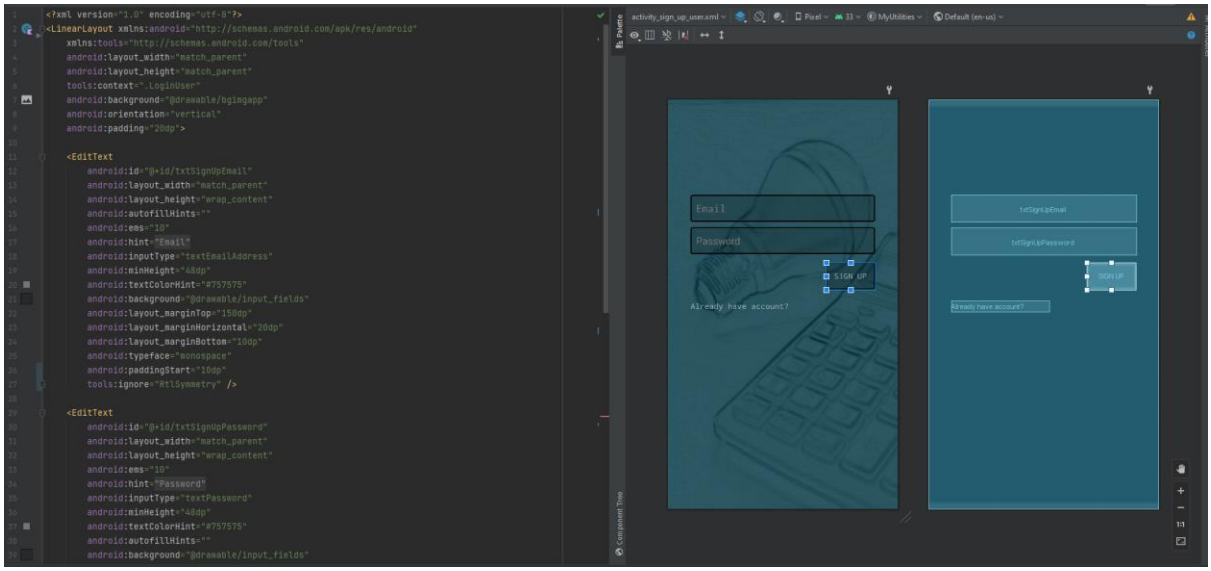
Kod prijave korisnik mora unijeti onaj i email i lozinku koji se podudaraju unutar **firebase** baze podataka. Ako Korisnik ne unese točne podatke ili ti podatci ne postoje u bazi podataka ispisat će se odgovarajuća poruka pogreške. Ako korisnik unese točne podatke, nakon klina na login tipku aplikacija će korisnika odvesti u sam izbornik. Korisnik također ima opciju otići na aktivnost registracije ako nema korisnički račun kao što je prikazano na slici 8.

Slika 9. Prikazuje funkcionalnost login aktivnosti. Unutar aktivnosti imamo funkciju loginUser koja provjerava da li su podatci uneseni, ali isto tako provjerava da li su ti podatci točni, ako nisu ispisat će se poruka pogreške.

```
class LoginUser : AppCompatActivity() {
    private lateinit var binding: ActivityLoginUserBinding
    private lateinit var mAuth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        mAuth = FirebaseAuth.getInstance()
        binding = ActivityLoginUserBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.btnLogin.setOnClickListener{ loginUser() }

        binding.txtDontHaveAcc.setOnClickListener{  it: View!
            val intent = Intent( packageContext: this, SignUpUser::class.java)
            startActivity(intent)
        }
    }
    private fun loginUser(){
        if(binding.txtLoginEmail.text.toString().isBlank() ||
                binding.txtLoginPassword.text.toString().isBlank())
        {
            Toast.makeText( context: this@LoginUser, text: "Please enter password or email!", Toast.LENGTH_LONG).show()
        }
        else
            mAuth.signInWithEmailAndPassword(binding.txtLoginEmail.text.toString(),
                binding.txtLoginPassword.text.toString()).addOnCompleteListener{task ->
                if(task.isSuccessful)
                {
                    Toast.makeText( context: this@LoginUser, text: "Log in successful!!", Toast.LENGTH_LONG).show()
                    val intent = Intent( packageContext: this, MainMenu::class.java)
                    startActivity(intent)
                }
                else Toast.makeText( context: this@LoginUser, task.exception!!.message.toString(), Toast.LENGTH_LONG).show()
                }
    }
}
```

Slika 9. Kod aktivnosti prijave

Pogled registracije je sličan pogledu prijave, razlika je ta u slučaju točnih formata lozinke i emaila oni se spremaju unutar **firebase** baze podataka. Slika 10 i 11 prikazuju pogled i aktivnost registracije.



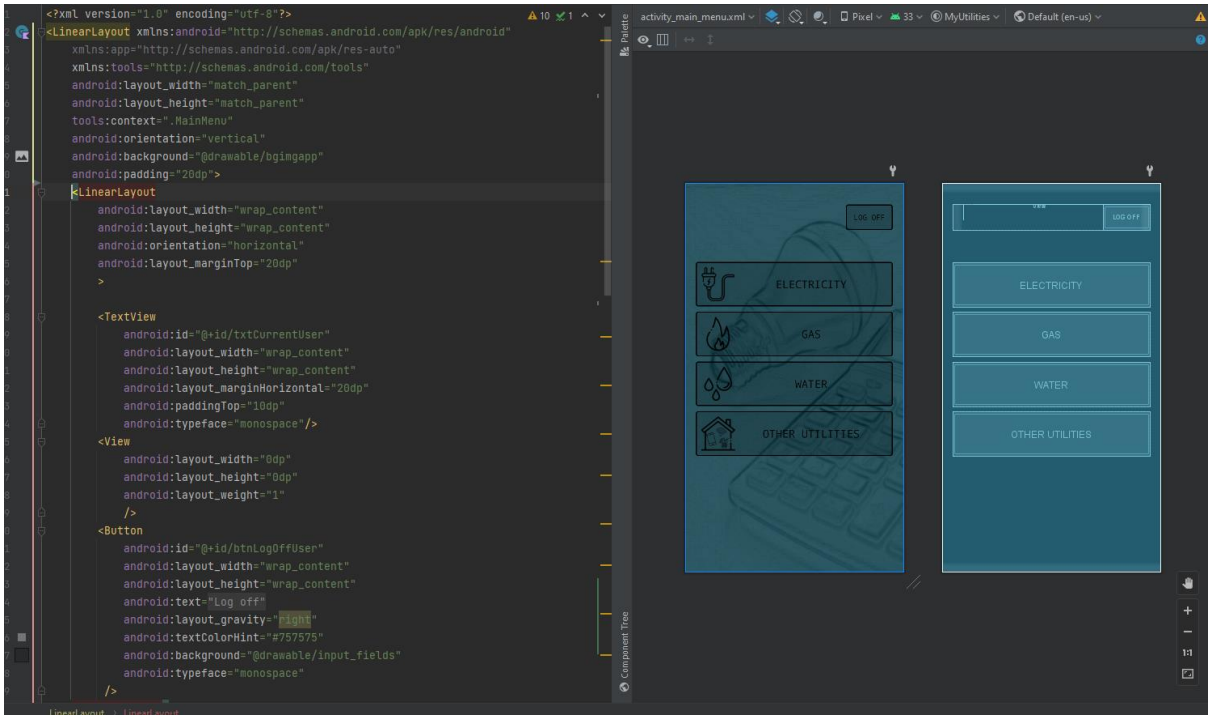
Slika 10. Pogled registracije

```
package com.example.myutilities

import ...
class SignUpUser : AppCompatActivity() {
    private lateinit var binding: ActivitySignUpUserBinding
    private lateinit var mAuth: FirebaseAuth
    private lateinit var refUsers: DatabaseReference
    private var firebaseUserID: String = ""

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        mAuth = FirebaseAuth.getInstance()
        binding = ActivitySignUpUserBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.btnSignUp.setOnClickListener { signupUser() }

        binding.txtHaveAcc.setOnClickListener{  it: View!
            val intent = Intent( packageContext: this, LoginUser::class.java)
            startActivity(intent)
        }
    }
    private fun signupUser(){
        if(binding.txtSignUpPassword.text.toString().isBlank()
            || binding.txtSignUpEmail.text.toString().isBlank()){
            Toast.makeText( context: this@SignUpUser, text: "Please enter password or email!",Toast.LENGTH_LONG).show()
        }
        else
            mAuth.createUserWithEmailAndPassword(
                binding.txtSignUpEmail.text.toString(),
                binding.txtSignUpPassword.text.toString()).addOnCompleteListener { task ->
                if(task.isSuccessful){
                    firebaseUserID = mAuth.currentUser!!.uid
                    refUsers = FirebaseDatabase.getInstance().reference.child( pathString: "Users")
                        .child(firebaseUserID)
                    Toast.makeText( context: this@SignUpUser,  text: "Sign Up Successful!", Toast.LENGTH_LONG).show()
                   val intent = Intent( packageContext: this, LoginUser::class.java)
                    startActivity(intent)
                }
```

Slika 11. Kod aktivnosti registracije

5.3 Izbornik

Nakon uspješne prijave korisnika će aplikacija odvesti unutar izbornika, Unutar izbornika se nalaze gumbi koji korisnika vode na ostakle aktivnosti kao što su računanje vode, plina ili struje, ali isto tako i spremanja ostalih režija kao što su : Mobilne pretplate, internet ili komunalne usluge. Unutar izbornika također imamo prikazano koji je korisnik trenutno prijavljen, te isto tako mogućnost i odjave iz same aplikacije, te vračanja na početak aplikacije. Slika 12 i 13 prikazuju kod i pogled izbornika.



Slika 12. Pogled izbornika

```
class MainMenu : AppCompatActivity() {
    private lateinit var binding: ActivityMainMenuBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainMenuBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        val user = Firebase.auth.currentUser
        if (user != null) binding.txtCurrentUser.text = user.email

        binding.btnLogOffUser.setOnClickListener { it: View!
            FirebaseAuth.getInstance().signOut()
            val intent = Intent( packageContext: this, MyUtilities::class.java)
            startActivity(intent)
        }
        binding.btnGoElectricity.setOnClickListener { it: View!
            val intent = Intent( packageContext: this, CountElectricity::class.java)
            startActivity(intent)
        }
        binding.btnGoGas.setOnClickListener { it: View!
            val intent = Intent( packageContext: this, CountGas::class.java)
            startActivity(intent)
        }
        binding.btnGoWater.setOnClickListener { it: View!
            val intent = Intent( packageContext: this, CountWater::class.java)
            startActivity(intent)

        }
        binding.btnGoOther.setOnClickListener { it: View!
            val intent = Intent( packageContext: this, OtherUtilities::class.java)
            startActivity(intent)
        }

    }
}
```

Slika 13. Kod aktivnosti izbornika

5.4 Računanje potrošnje vode

U ovoj aktivnosti korisnik treba unijeti staro i novo stanje potrošnje vode. Ako korisnik ne unese nešto od podataka pojavit će se odgovarajuća poruka. Ako korisnik unese da je novo stanje manje od starog stanja potrošnje vode, također će se ispisati poruka pogreške, zato što novo stanje ne smije biti manje od starog stanja. Korisnik je također obvezan prije spremanja izračuna unutar firebase baze podataka unijeti podatke kao što su : Broj računa i datum. Kada korisnik sve to unese ispisat će mu se cijena idućeg računa (Slika15). Slika 14. prikazuje dizajn ili pogled računanja potrošnje vode

Slika 14. Prikaz računanja potrošnje vode

```
private fun pickWaterCompany() {
    val pickCompany = resources.getStringArray(R.array.water_company)
    val spinnerCompany = binding.spinnerWaterCompany
    val adapterCompany = ArrayAdapter( context: this, android.R.layout.simple_spinner_item, pickCompany)
    spinnerCompany.adapter = adapterCompany
    spinnerCompany.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener {
        override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
            when (pickCompany[p2].toString()) {
                "Vodovod - Osijek d.o.o" -> {
                    waterCompanyPrice = 28.25
                    waterCompanyPDV = 12.68
                    waterCompanyName = "Vodovod - Osijek d.o.o"
                }
                "Vodovod i Kanalizacija d.o.o Rijeka" -> {
                    waterCompanyPrice = 19.20
                    waterCompanyPDV = 12.7157
                    waterCompanyName = "Vodovod i Kanalizacija d.o.o Rijeka"
                }
                "Vodoopskrba i odvodnja d.o.o Zagreb" -> {
                    waterCompanyPrice = 18.92
                    waterCompanyPDV = 15.2887
                    waterCompanyName = "Vodoopskrba i odvodnja d.o.o Zagreb"
                }
            }
        }

        override fun onNothingSelected(p0: AdapterView<*>?) {
            Toast.makeText(
                 context: this@CountWater,
                 text: "You have to select water company!",
                Toast.LENGTH_LONG
            ).show()
        }
    }
}
```

Slika 15. Kod za odabir tvrtke

```
private fun countWater() {
    var waterDifference = binding.txtLastReadingWater.text.toString().toInt()
        .toDouble() - binding.txtFirstReadingWater.text.toString().toInt().toDouble()
    waterDifference = waterDifference * waterCompanyPDV + waterCompanyPrice
    waterPrice = (waterDifference * 100.0).roundToInt() / 100.0
    binding.txtWaterBill.text = ("Your price is $waterPrice kn!")
}

private fun showDateView() {
    val dateView = Calendar.getInstance()
    val year = dateView.get(Calendar.YEAR)
    val month = dateView.get(Calendar.MONTH)
    val day = dateView.get(Calendar.DAY_OF_MONTH)

    val datePicker = DatePickerDialog( context: this, { _, dateYear, dateMonth, dayOfMonth ->
        saveDateYearWater = dateYear.toString()
        saveDateMonthWater = (dateMonth + 1).toString()
        saveDateDayWater = dayOfMonth.toString()
    }, year, month, day)
    datePicker.show()
    ifPressedDateWater = true
}

private fun saveWaterToFirebase() {
    val db = FirebaseFirestore.getInstance()
    val water: MutableMap<String, Any> = HashMap()
    val user = Firebase.auth.currentUser?.email.toString()
    water["Price"] = waterPrice
    water["User"] = user
    water["Date"] = "$saveDateMonthWater/$saveDateYearWater"
    water["Company"] = waterCompanyName
    water["Bill_ID"] = binding.txtBillNumberWater.text.toString()

    db.collection( collectionPath: "Water").add(water).addOnCompleteListener { it: Task<DocumentReference!>
        Toast.makeText( context: this@CountWater, text: "You saved your data successfully!", Toast.LENGTH_LONG).show()
        finish()
        overridePendingTransition( enterAnim: 0, exitAnim: 0)
        startActivity(intent)
        overridePendingTransition( enterAnim: 0, exitAnim: 0)
    }
}
```

Slika 16. Kod za računanje, spremanje vode i odabir datuma

Slika 16 prikazuje kako funkcionira kod za računanje potrošnje vode, ali isto tako kako i spremiti vodu unutar **firebase** baze podataka. Također tu je i funkcija koja otvara dijalog za odabir datuma računa.

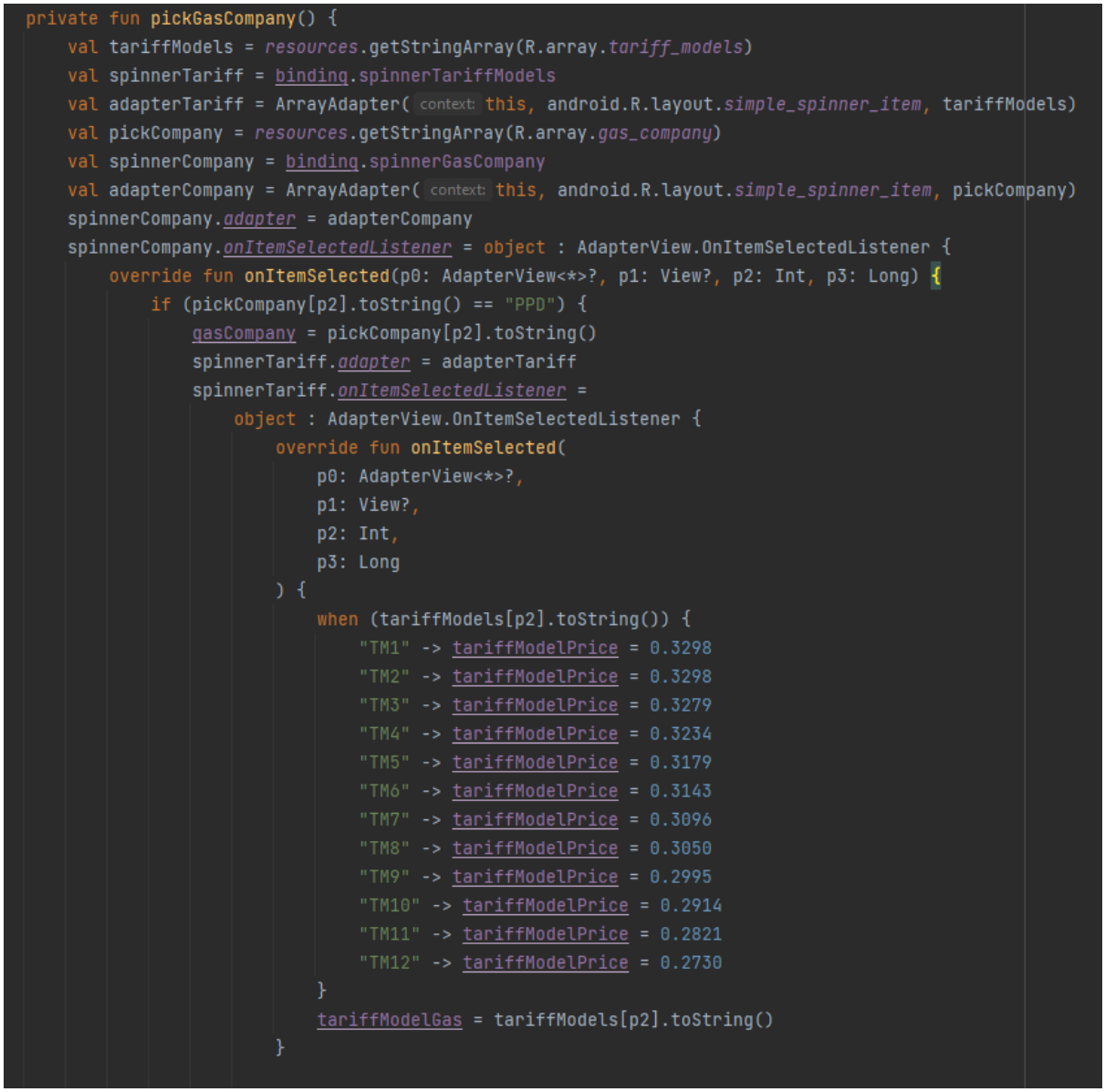
5.5 Računanje potrošnje plina

Računanje potrošnje plina radi na isti princip kao i računanje potrošnje vode, razlika je u vrijednostima, te kod plina imamo dvanaest tarifnih modela. Svaki tarifni model sadrži drugačiju vrijednost. Svaka tvrtka također ima različite tarifne modele.

```
private fun checkIfBlankGas(): Boolean {
    val firstReadingGas = binding.txtGasFirstReading.text
    val lastReadingGas = binding.txtGasLastReading.text
    val billNumber = binding.txtBillNumberGas.text
    if (firstReadingGas.isBlank() || lastReadingGas.isBlank() || billNumber.isBlank()) {
        return true
    }
    return false
}

private fun checkIfFirstReadingGasIsHigher(): Boolean {
    val firstReadingGas = binding.txtGasFirstReading.text.toString()
    val lastReadingGas = binding.txtGasLastReading.text.toString()

    if (firstReadingGas.toInt() >= lastReadingGas.toInt()) {
        return true
    }
    return false
}
```

Slika 17. Kod za provjeru ispravnosti podataka potrošnje plina

Slika 17. Prikazuje kod za provjeru da li su polja koja korisnik treba unijeti prazna, te isto tako prikazuje da li je staro stanje veće od novog. Ukoliko validnost podataka nije ispravna korisniku će se na ekran ispisat odgovarajuća poruka pogreške.

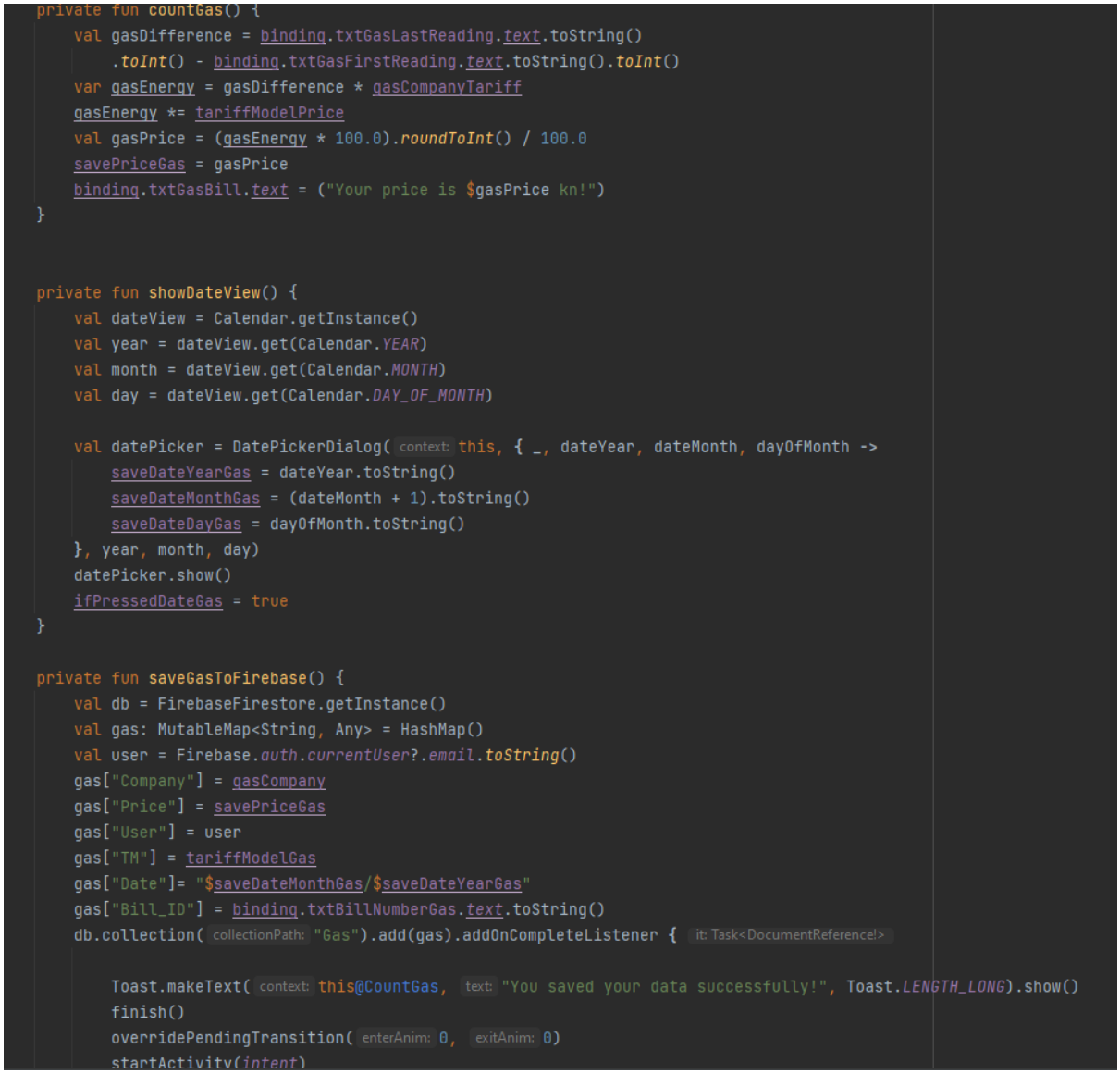


```
private fun pickGasCompany() {
    val tariffModels = resources.getStringArray(R.array.tariff_models)
    val spinnerTariff = binding.spinnerTariffModels
    val adapterTariff = ArrayAdapter( context: this, android.R.layout.simple_spinner_item, tariffModels)
    val pickCompany = resources.getStringArray(R.array.gas_company)
    val spinnerCompany = binding.spinnerGasCompany
    val adapterCompany = ArrayAdapter( context: this, android.R.layout.simple_spinner_item, pickCompany)
    spinnerCompany.adapter = adapterCompany
    spinnerCompany.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
        override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
            if (pickCompany[p2].toString() == "PPD") {
                gasCompany = pickCompany[p2].toString()
                spinnerTariff.adapter = adapterTariff
                spinnerTariff.onItemSelectedListener =
                    object : AdapterView.OnItemSelectedListener {
                        override fun onItemSelected(
                            p0: AdapterView<*>?,
                            p1: View?,
                            p2: Int,
                            p3: Long
                        ) {
                            when (tariffModels[p2].toString()) {
                                "TM1" -> tariffModelPrice = 0.3298
                                "TM2" -> tariffModelPrice = 0.3298
                                "TM3" -> tariffModelPrice = 0.3279
                                "TM4" -> tariffModelPrice = 0.3234
                                "TM5" -> tariffModelPrice = 0.3179
                                "TM6" -> tariffModelPrice = 0.3143
                                "TM7" -> tariffModelPrice = 0.3096
                                "TM8" -> tariffModelPrice = 0.3050
                                "TM9" -> tariffModelPrice = 0.2995
                                "TM10" -> tariffModelPrice = 0.2914
                                "TM11" -> tariffModelPrice = 0.2821
                                "TM12" -> tariffModelPrice = 0.2730
                            }
                            tariffModelGas = tariffModels[p2].toString()
                        }
```

Slika 18. Kod za odabir tarifnog modele i tvrtke

Kao što je prije navedeno svaki tarifni model ima drugačiju vrijednost, ali isto tako svaka tvrtka sadrži i drugačije tarifne modele, te isto tako svaka tvrtka ima drugačiju tarifu prema kojoj se izvršava potrošnja(Slika 18.)

Slika 19. Prikazuje samo izvršavanje koda za spremanje, računanje i odabir datuma za samu potrošnju. Ukoliko korisnik nije ispunio sve prijašnje uvjete, korisnik tada neće biti u mogućnosti spremiti izračun u bazu podataka.



```
private fun countGas() {
    val gasDifference = binding.txtGasLastReading.text.toString()
        .toInt() - binding.txtGasFirstReading.text.toString().toInt()
    var gasEnergy = gasDifference * gasCompanyTariff
    gasEnergy *= tariffModelPrice
    val gasPrice = (gasEnergy * 100.0).roundToInt() / 100.0
    savePriceGas = gasPrice
    binding.txtGasBill.text = ("Your price is $gasPrice kn!")
}


private fun showDateView() {
    val dateView = Calendar.getInstance()
    val year = dateView.get(Calendar.YEAR)
    val month = dateView.get(Calendar.MONTH)
    val day = dateView.get(Calendar.DAY_OF_MONTH)

    val datePicker = DatePickerDialog( context: this, { _, dateYear, dateMonth, dayOfMonth ->
        saveDateYearGas = dateYear.toString()
        saveDateMonthGas = (dateMonth + 1).toString()
        saveDateDayGas = dayOfMonth.toString()
    }, year, month, day)
    datePicker.show()
    ifPressedDateGas = true
}

private fun saveGasToFirebase() {
    val db = FirebaseFirestore.getInstance()
    val gas: MutableMap<String, Any> = HashMap()
    val user = Firebase.auth.currentUser?.email.toString()
    gas["Company"] = gasCompany
    gas["Price"] = savePriceGas
    gas["User"] = user
    gas["TM"] = tariffModelGas
    gas["Date"]= "$saveDateMonthGas/$saveDateYearGas"
    gas["Bill_ID"] = binding.txtBillNumberGas.text.toString()
    db.collection( collectionPath: "Gas").add(gas).addOnCompleteListener {  it: Task<DocumentReference!>

        Toast.makeText( context: this@CountGas,  text: "You saved your data successfully!", Toast.LENGTH_LONG).show()
        finish()
        overridePendingTransition( enterAnim: 0,  exitAnim: 0)
        startActivity(intent)
```

Slika 19. Kod za izračun, spremanje i odabir datuma potrošnje plina

Slika 20. Prikaz računanja potrošnje plina

5.6 Računanje potrošnje struje

Računanje struje se odvija isto kao i prijašnje režije uz razliku da struja ima četiri stanja: Staro i novo dnevno stanje , te staro i novo noćno stanje. Korisnik isto tako mora unijeti da je novo stanje veće od starog, polja ne smiju biti prazna, te isto tako mora unijeti i broj računa prilikom spremanja.

```
private fun checkIfBlank(): Boolean {

    val firstReadingDay = binding.txtFirstReadingDay.text.toString()
    val lastReadingDay = binding.txtLastReadingDay.text.toString()
    val firstReadingNight = binding.txtFirstReadingNight.text.toString()
    val lastReadingNight = binding.txtLastReadingNight.text.toString()
    val billNumber = binding.txtBillNumberElectricity.text.toString()

    if (firstReadingDay.isBlank() || lastReadingDay.isBlank() || firstReadingNight.isBlank() || last
            billNumber.isBlank()) {
        return true
    }
    return false
}

private fun checkIfFirstReadingIsHigher(): Boolean {
    val firstReadingDay = binding.txtFirstReadingDay.text.toString()
    val lastReadingDay = binding.txtLastReadingDay.text.toString()
    val firstReadingNight = binding.txtFirstReadingNight.text.toString()
    val lastReadingNight = binding.txtLastReadingNight.text.toString()

    if (firstReadingDay.toInt() >= lastReadingDay.toInt() || firstReadingNight.toInt() >= lastReadi
        return true
    }

    return false
}
@SuppressLint("SetTextI18n")
private fun countElectricity() {

    val differenceDay = binding.txtLastReadingDay.text.toString().toInt() - binding.txtFirstReading
    val differenceNight = binding.txtLastReadingNight.text.toString().toInt() - binding.txtFirstRea
    val firstFee = differenceDay.toDouble() * 0.11 + differenceNight.toDouble() * 0.05
    val secondFee = differenceDay.toDouble() * 0.24 + differenceNight.toDouble() * 0.12 + 10
    val thirdFee = (differenceDay.toDouble() + differenceNight.toDouble()) * 0.1050
    val x1 = differenceDay.toDouble() * 0.49
    val x2 = differenceNight.toDouble() * 0.25
    val pdv = ((x1 + x2 + 7.40 + firstFee + secondFee + thirdFee) * 0.13)
    val price = ((((x1 + x2 + 7.40 + firstFee + secondFee + thirdFee + pdv)) * 100.0).roundToInt()
    savePriceElectricity = price
    binding.txtElectricityBill.text = ("Your price is $savePriceElectricity kn!")
```

Slika 21. Kod za provjeru polja i izračun potrošnje

Slika 21. Prikazuje provjere da li su polja prazna. Ukoliko su polja prazna korisniku će se ispisati odgovarajuća poruka da tu grešku ispravi. Također u ovom kodu isto imamo provjeru da li je novo stanje veće od starog. Zadnja funkcija izvršava sam izračun koji se na kraju ispise na ekranu(Slika 22.)

Slika 22. Prikaz izračuna potrošnje struje.

5.7 Spremanje ostalih režija

Aktivnost spremanja ostalih režija nam nudi spremanje režija kao što su: Mobilne pretplate, Internet/fiksna mreža i komunalne usluge. Korisnik sam mora odabrati što želi spremiti, te isto tako navesti cijenu i ime pružatelja usluga. U ovoj aktivnosti korisnik samo sprema režije jer nema nikakvog izračuna zato što su cijene iz mjeseca u mjesec uglavnom fiksne. Ukoliko korisnik ne unese ime ili cijenu ispisat će se odgovarajuća poruka. Funkcionalnost koda je prikaza u slici 23.

```
private fun pickUtility() {
    val pickUtility = resources.getStringArray(R.array.utility)
    val spinnerUtility = binding.spnPickUtility
    val adapterUtility = ArrayAdapter( context: this, android.R.layout.simple_spinner_item, pickUtilit
    spinnerUtility.adapter = adapterUtility
    spinnerUtility.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener {
        override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
            when(pickUtility[p2].toString()){
                "Mobilna pretplata" -> pickedUtility = "Mobilna pretplata"
                "Komunalne usluge" -> pickedUtility = "Komunalne usluge"
                "Fiksna mreza/Internet" -> pickedUtility = "Fiksna mreza/Internet"
            }
        }
        override fun onNothingSelected(p0: AdapterView<*>?) {
            Toast.makeText( context: this@OtherUtilities, text: "You have to select utility", Toast.LENG
        }
    }

}

private fun checkIfBlankOther(): Boolean {
    val companyName = binding.txtCompanyName.text
    val otherPrice = binding.txtPrice.text
    val billNumberOther = binding.txtBillNumberOther.text
    if (companyName.isBlank() || otherPrice.isBlank() || billNumberOther.isBlank()) {
        return true
    }
    return false
}
```

Slika 23. Kod za biranje režije i provjere validnosti polja

Slika 24. Prikaz spremanja režija

6. Aktivnosti i pogledi za prikaz i brisanje spremljenih režija

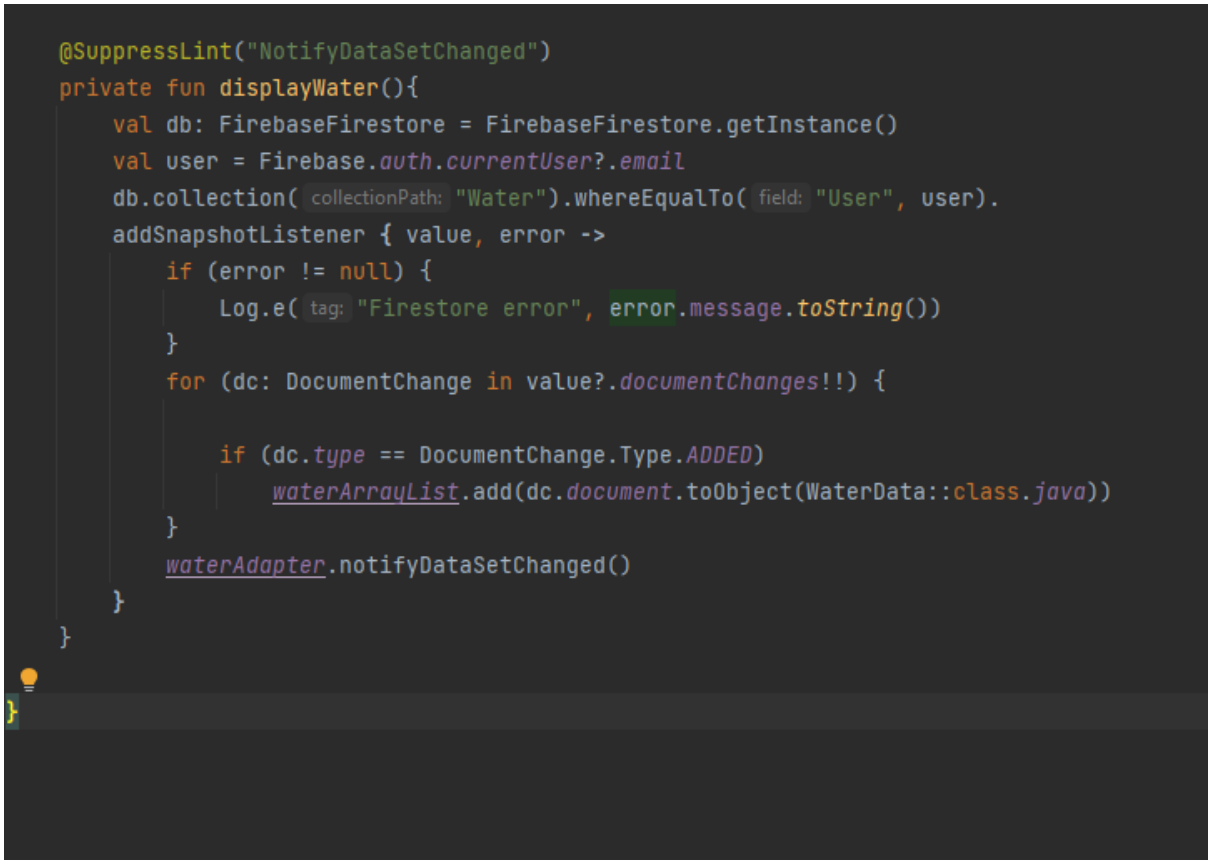
Iduće aktivnosti prikazat će spremanje i brisanje svih spremljenih režija. Korisnik ima uvid u samo svoje spremljene režije. Sve spremljene režije se ispisuju u RecyclerView. Korisnik briše režije pomoću broja računa

6.1 Prikaz spremljenih režija za potrošnju vode

Prikaz režija se odvija u dvije aktivnosti, tj. Aktivnosti i adapteru. Aktivnost uzima podatke iz baze podataka, te ih prosljeđuje adapter kao što je prikazano u slici 26., te se to onda ispisuje u RecyclerView. Unutar adaptera se također izvršava i brisanje režija(Slika 25.)

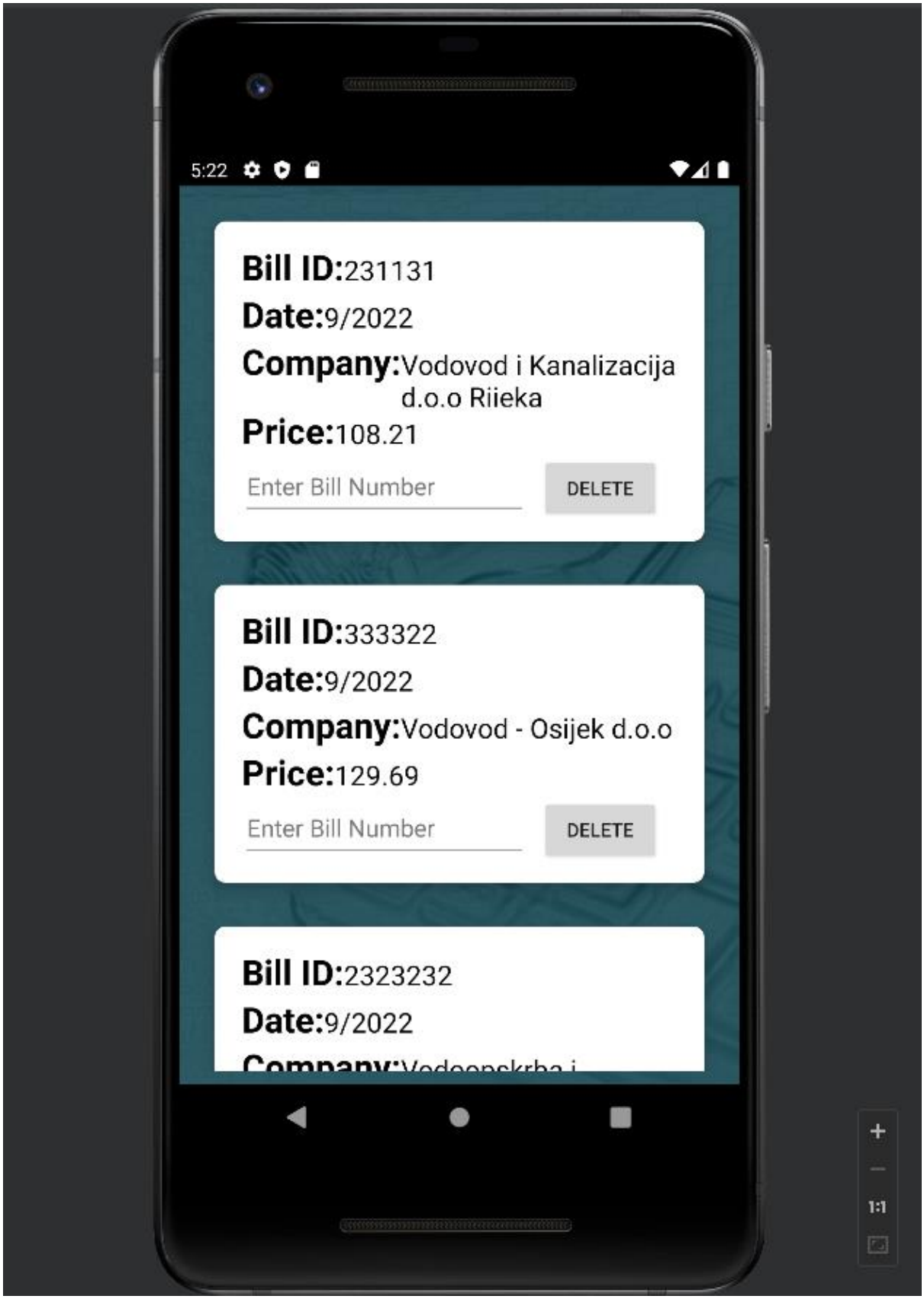
```
override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val water: WaterData = waterList[position]
    holder.billID.text = water.Bill_ID.toString()
    holder.date.text = water.Date
    holder.companyName.text = water.Company
    holder.price.text = water.Price.toString()
    holder.delete.setOnClickListener {  it: View!
        val db = FirebaseFirestore.getInstance()
        val query = db.collection( collectionPath: "Water").whereEqualTo( field: "Bill_ID",holder.deleteB
        query.addOnCompleteListener {  it: Task<QuerySnapshot!>
            for(document in it.result){
                db.collection( collectionPath: "Water").document(document.id).delete()
                waterList.removeAt(position)
                notifyDataSetChanged()
            }

        }

    }
}

override fun getItemCount(): Int {
    return waterList.size
}

class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
    val billID : TextView = itemView.findViewById(R.id.tvBillID)
    val date : TextView = itemView.findViewById(R.id.tvDate)
    val companyName : TextView = itemView.findViewById(R.id.tvCompany)
    val price : TextView = itemView.findViewById(R.id.tvPrice)
    var delete : Button = itemView.findViewById(R.id.btnDeleteWater)
    var deleteByID : EditText = itemView.findViewById(R.id.txtDelID)


}
```

Slika 25. Adapter za potrošnju vode



```
@SuppressLint("NotifyDataSetChanged")
private fun displayWater(){
    val db: FirebaseFirestore = FirebaseFirestore.getInstance()
    val user = Firebase.auth.currentUser?.email
    db.collection( collectionPath: "Water").whereEqualTo( field: "User", user).
    addSnapshotListener { value, error ->
        if (error != null) {
            Log.e( tag: "Firestore error", error.message.toString())
        }
        for (dc: DocumentChange in value?.documentChanges!!) {

            if (dc.type == DocumentChange.Type.ADDED)
                waterArrayList.add(dc.document.toObject(WaterData::class.java))
        }
        waterAdapter.notifyDataSetChanged()
    }
}
}
```

Slika 26. Prikaz koda koji dohvaća podatke iz baze podataka za potrošnju vode



Slika 27. Prikaz ispisanih spremljenih izračuna za potrošnju vode

6.2 Prikaz spremljenih režija za potrošnju plina

```
private fun displayGas(){
    db = FirebaseFirestore.getInstance()
    val user = Firebase.auth.currentUser?.email
    db.collection( collectionPath: "Gas").whereEqualTo( field: "User", user).
            addSnapshotListener { value, error ->
                if (error != null) {
                    Log.e( tag: "Firestore error", error.message.toString())
                }

                for (dc: DocumentChange in value?.documentChanges!!) {

                    if (dc.type == DocumentChange.Type.ADDED)
                        gasArrayList.add(dc.document.toObject(GasData::class.java))
                }
                gasAdapter.notifyDataSetChanged()
            }
}
```

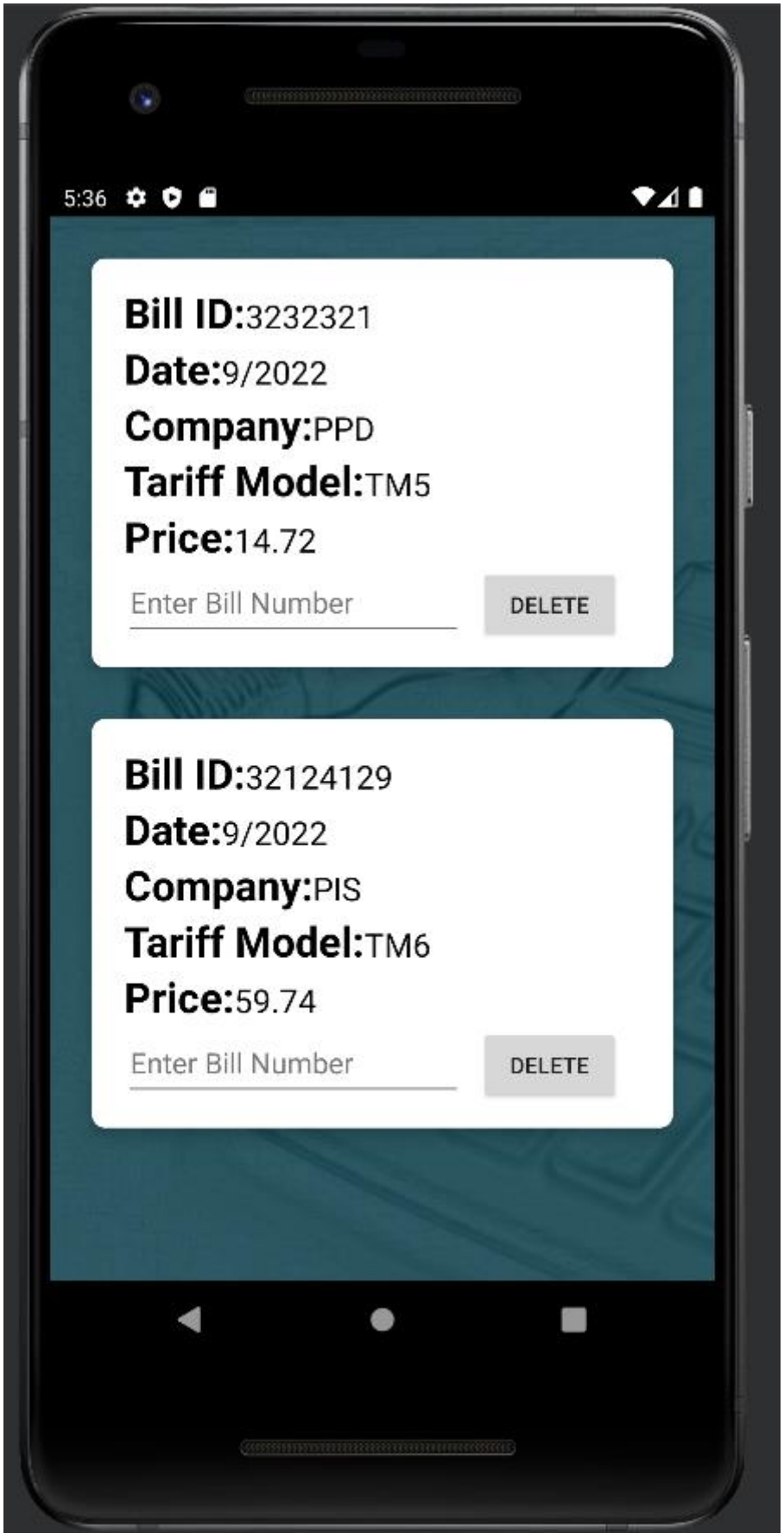
Slika 28. Prikaz koda koji dohvaća podatke iz baze podataka za potrošnju plina

Slika 28. prikazuje funkciju koja dohvaća podatke iz baze podataka. Sve režije koje se ispišu su vidljive samo od strane korisnika koji ih je spremio, te isto tako samo ih taj isti korisnik može i obrisati.

```
@SuppressLint("NotifyDataSetChanged")
override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val gas: GasData = gasList[position]
    holder.billID.text = gas.Bill_ID.toString()
    holder.date.text = gas.Date
    holder.companyName.text = gas.Company
    holder.tariffModel.text = gas.TM
    holder.price.text = gas.Price.toString()
    holder.delete.setOnClickListener {  it: View!
        val db = FirebaseFirestore.getInstance()
        val query = db.collection( collectionPath: "Gas").whereEqualTo( field: "Bill_ID",holder.deleteByID.t
        query.addOnCompleteListener {  it: Task<QuerySnapshot!>
            for(document in it.result){
                db.collection( collectionPath: "Gas").document(document.id).delete()
                gasList.removeAt(position)
                notifyDataSetChanged()
            }

    }

    }
}

override fun getItemCount(): Int {
     return gasList.size
}

class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
    val billID : TextView = itemView.findViewById(R.id.tvBillID)
    val date : TextView = itemView.findViewById(R.id.tvDate)
    val companyName : TextView = itemView.findViewById(R.id.tvCompany)
    val tariffModel : TextView = itemView.findViewById(R.id.tvTM)
    val price : TextView = itemView.findViewById(R.id.tvPrice)
    var delete : Button = itemView.findViewById(R.id.btnDeleteGas)
    var deleteByID : EditText = itemView.findViewById(R.id.txtDelID)


}
```

Slika 29. Adapter za potrošnju plina

Unutar adapter se odvija brisanje režije za koju se korisnik odlučio. Aplikacija prvo pomoću koda iz slike 28. dohvaća sve podatke , te ih onda pohranjuje unutar adaptera koji ih tada pomoću liste i polja ispisuje u RecyclerView.



Slika 30. Prikaz ispisanih spremljenih izračuna za potrošnju plina

6.3 Prikaz ispisanih spremljenih izračuna za potrošnju struje

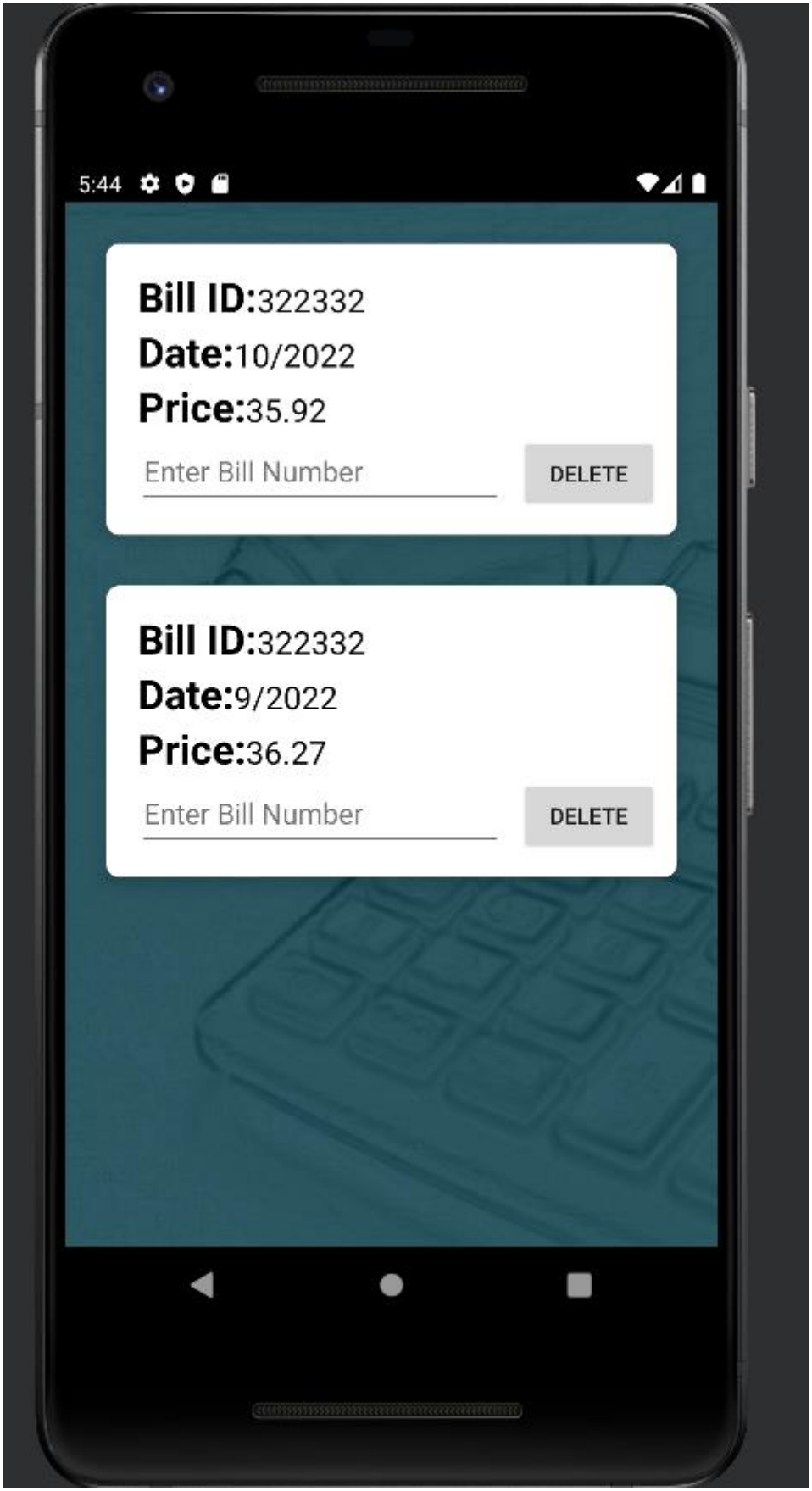
```
    @SuppressLint("NotifyDataSetChanged")
    private fun displayElectricity(){
        val db: FirebaseFirestore = FirebaseFirestore.getInstance()
        val user = Firebase.auth.currentUser?.email
        db.collection( collectionPath: "Electricity").whereEqualTo( field: "User", user).
        addSnapshotListener { value, error ->
            if (error != null) {
                Log.e( tag: "Firestore error", error.message.toString())
            }
            for (dc: DocumentChange in value?.documentChanges!!) {

                if (dc.type == DocumentChange.Type.ADDED)
                    electricityArrayList.add(dc.document.toObject(ElectricityData::class.java))
            }
           electricityAdapter.notifyDataSetChanged()
        }
    }

}
```

Slika 31. Prikaz koda koji dohvaća podatke iz baze podataka za potrošnju struje

Ispis spremljenih računa za potrošnju struje radi na isti način kao i ostale režije. Sve se prvo dohvaća iz baze , te se onda prosljeđuje u adapter za struju. Funkcionalnost koda se nalazi na Slici 31.

```
}
@SuppressLint("NotifyDataSetChanged")
override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val electricity: ElectricityData = electricityList[position]
    holder.billID.text = electricity.Bill_ID.toString()
    holder.date.text = electricity.Date
    holder.price.text = electricity.Price.toString()
    holder.delete.setOnClickListener {  it: View!
        val db = FirebaseFirestore.getInstance()
        val query = db.collection( collectionPath: "Electricity").whereEqualTo( field: "Bill_ID",holder.del
        query.addOnCompleteListener {  it: Task<QuerySnapshot!>
            for(document in it.result){
                db.collection( collectionPath: "Electricity").document(document.id).delete()
                electricityList.removeAt(position)
                notifyDataSetChanged()
            }

        }

    }
}
override fun getItemCount(): Int {
    return electricityList.size
}
class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
    val billID : TextView = itemView.findViewById(R.id.tvBillID)
    val date : TextView = itemView.findViewById(R.id.tvDate)
    val price : TextView = itemView.findViewById(R.id.tvPrice)
    var delete : Button = itemView.findViewById(R.id.btnDeleteOther)
    var deleteByID : EditText = itemView.findViewById(R.id.txtDelID)

}
```

Slika 32. Adapter za potrošnju struje



Slika 33. Prikaz ispisanih spremljenih izračuna za potrošnju struje

6.4 Prikaz ispisanih spremljenih ostalih režija

Ostale režije prikazuju: Mobilne pretplate, Internet/fiksna mreža, komunalne usluge, radi na isti način kao i ostale aktivnosti koje su navedene prije. Razlika je ta što su u ostalim režijama cijene fiksne.

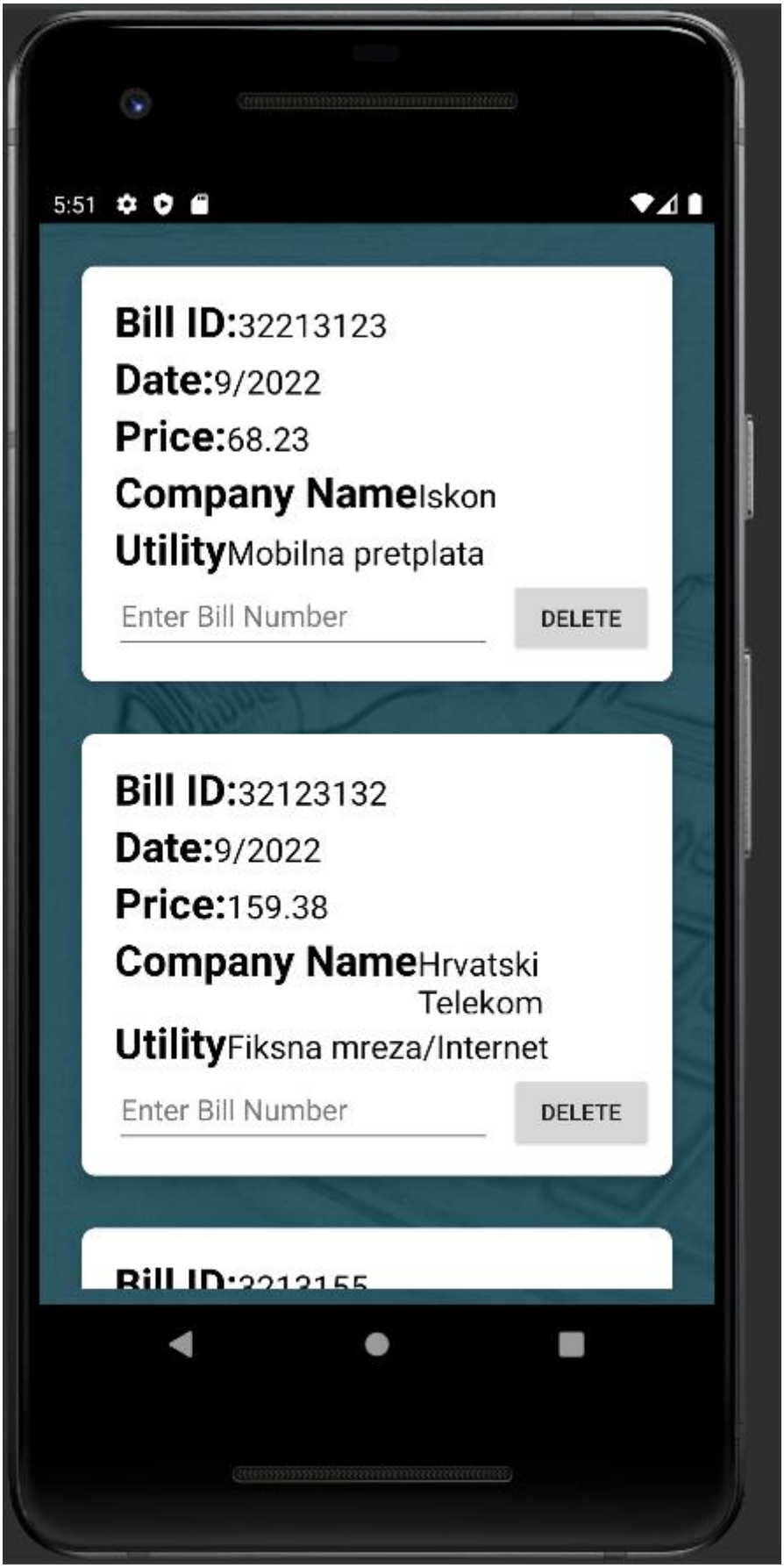
```
@SuppressLint("NotifyDataSetChanged")
private fun displayOtherUtilities(){
    var db: FirebaseFirestore = FirebaseFirestore.getInstance()
    val user = Firebase.auth.currentUser?.email
    db.collection( collectionPath: "Other").whereEqualTo( field: "User", user).
    addSnapshotListener { value, error ->
        if (error != null) {
            Log.e( tag: "Firestore error", error.message.toString())
        }

        for (dc: DocumentChange in value?.documentChanges!!) {

            if (dc.type == DocumentChange.Type.ADDED)
                otherArrayUtilityList.add(dc.document.toObject(OtherUtilityData::class.java))
        }
       otherUtilityAdapter.notifyDataSetChanged()
    }
}

}
```

Slika 34. Prikaz koda koji dohvaća podatke iz baze podataka za ostale režije

Slika 34. prikazuje funkciju koja dohvaća podatke iz firebase baze podataka, te ih prosljeđuje dalje u adapter za ostale režije.

```
override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val otherUtility: OtherUtilityData = otherUtilityList[position]
    holder.billID.text = otherUtility.Bill_ID.toString()
    holder.date.text = otherUtility.Date
    holder.companyName.text = otherUtility.Company
    holder.utility.text = otherUtility.Utility
    holder.price.text = otherUtility.Price.toString()
    holder.delete.setOnClickListener {  it: View!
        val db = FirebaseFirestore.getInstance()
        val query = db.collection( collectionPath: "Other").whereEqualTo( field: "Bill_ID",holder.deleteByID.
        query.addOnCompleteListener {  it: Task<QuerySnapshot!>
            for(document in it.result){
                db.collection( collectionPath: "Other").document(document.id).delete()
                otherUtilityList.removeAt(position)
                notifyDataSetChanged()
            }

        }

    }
}

override fun getItemCount(): Int {
    return otherUtilityList.size
}

class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){
    val billID : TextView = itemView.findViewById(R.id.tvBillID)
    val date : TextView = itemView.findViewById(R.id.tvDate)
    val price : TextView = itemView.findViewById(R.id.tvPrice)
    val companyName :TextView = itemView.findViewById(R.id.tvCompany)
    val utility: TextView = itemView.findViewById(R.id.tvUtility)
    var delete : Button = itemView.findViewById(R.id.btnDeleteOther)
    var deleteByID : EditText = itemView.findViewById(R.id.txtDelID)


}
```

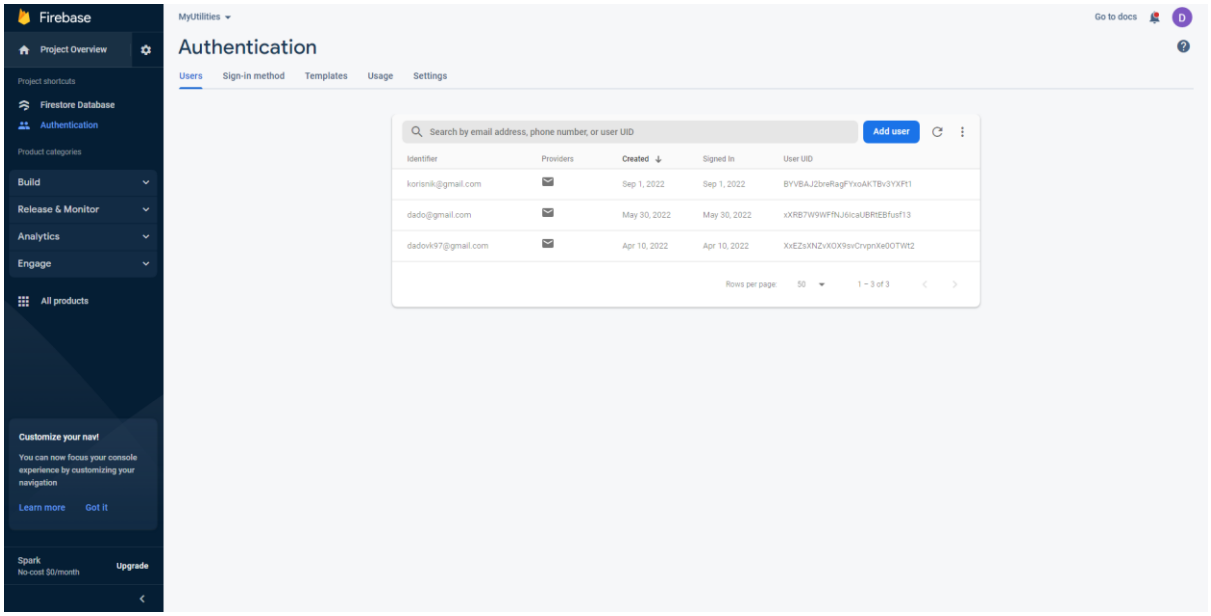
Slika 35. Adapter za ostale režije



Slika 36. Prikaz ispisanih spremljenih ostalih režija

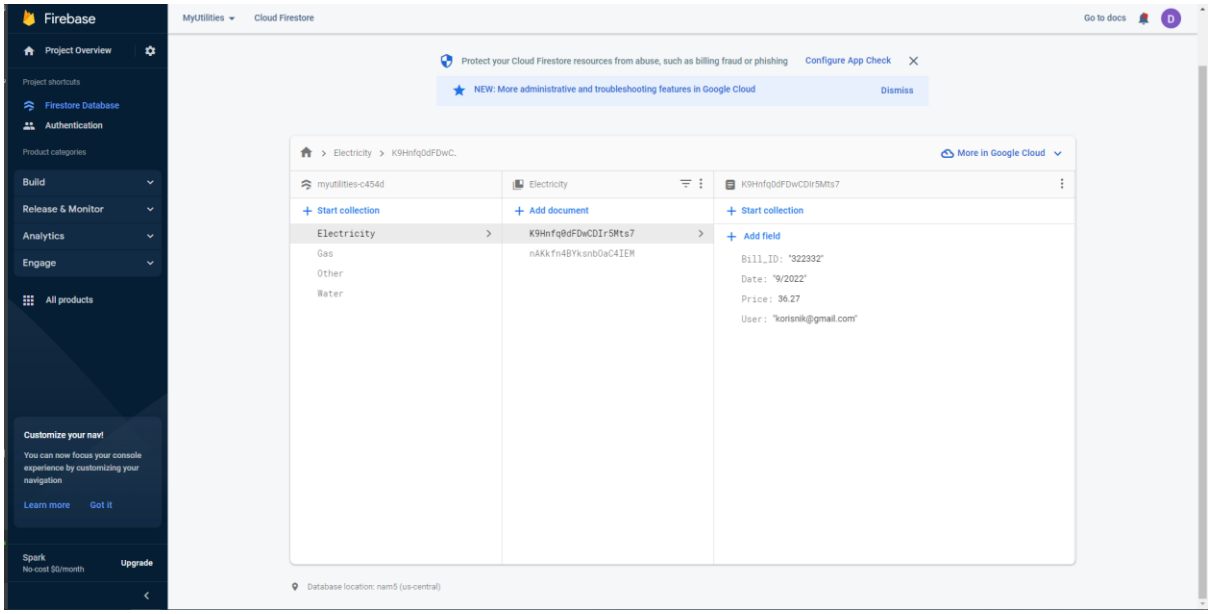
7. Prikaz svih podataka unutar firebase-a

Svi ovi podatci koji su navedeni prije su spremljeni u json tip podatka koji se nalazi na firebaseu. Firebase sadrži svoj top autentikacije korisnika, te se tako isto može implementirati da se korisnik prijavi putem Gmaila, Facebooka ili twittera. Postoje dvije vrste baze podataka na firebaseu, a to su realtime database i firestore database. Aplikacija koristi firebase firestore bazu podataka iz razloga što odgovara potrebama same aplikacije.



Slika 37. Prikaz spremljenih korisnika unutar baze podataka

Slika 37 prikazuje spremljene korisnike u bazi podataka, unutar baze je upisano kada je korisnik kreiran, te njegov ID. Svaki korisnik mora imati drugačiju email adresu, te nije moguće unijeti istu.



Slika 38. Prikaz spremljenih režija unutar baze podataka

Slika 38 prikazuje spremljene režije unutar baze podataka, sve režije su strukturirane unutar dokumenata, te svaki dokument ima svoj ID. Firestore je jako dobar način strukturiranja podataka, te snalaženje u istim.

8.Zaključak

MyUtility Aplikacija je vrlo jednostavna i pristupačna aplikacija za korištenje. Aplikacija korisniku donosi određenu organiziranost u potrošnji režija, Korisnik tako zna koliko novaca treba odvojiti za plaćanje režija.

Prednost ove aplikacije je ta što je vrlo jednostavna za korištenje, te je mogu koristiti svi uzrasti od mladih do starih, također korisnici ne moraju posjetiti stranice ovlaštenih poduzeća u svrhu računanja režija. Aplikacija također ukazuje i na pogreške u unosu vrijednosti. Korisnik vrlo lako može vidjeti sve svoje spremljene režije te obrisati iste.

Firebase se pokazao kao vrlo jednostavan za koristiti u skladu s aplikacijom, način spremanja i strukturiranja podataka je jako jednostavan.

Android studio se isto tako pokazao kao vrlo jednostavnim i pogodnim za koristit. Emulator je radio jako brzo, te isto tako povezivanje i s vlastitim mobilnim uređajem.

Tijekom korištenja android studia svi alati su bili lagani i dostupni za koristiti.

9. Literatura

[1]. https://www.hep.hr(01.09.2022)

[2]. www.pis.com.hr(01.09.2022)

[3]. www.ppd.hr(01.09.2022)

[4]. http://www.kdvik-rijeka.hr/(01.09.2022)

[5]. https://vodovod.com/(01.09.2022)

[6]. https://www.vio.hr(01.09.2022)

[7]. https://firebase.google.com/docs/auth/android/start(03.09.2022)

[8]. https://www.youtube.com/watch?v=8l5gCLaS25w&ab_channel=tutorialsEU(10.08.2022)

[9]. https://medium.com/fnplus/cloud-firestore-kotlin-33892886ce64(11.08.2022)

[10]. https://stackoverflow.com/questions/56237941/how-to-display-firestore-timestampdate-and-time-in-a-recyclerview(11.08.2022)

[11]. https://www.youtube.com/watch?v=Ly0xwWlUpVM&ab_channel=Foxandroid(15.08.2022)

10. Popis slika

11. Sažetak

Aplikacija MyUtilities je mobilna android aplikacija koja služi za računanje režija. Voda, struja i plin se mogu izračunati za određene gradove. Korisnik unosi vrijednosti potrošnje za novi i stari mjesec, te tako dobiva iznos koji bih trebao platiti. Sve izračune moguće je spremiti unutar baze podataka. Za izradu same aplikacije korištena je platforma Android studio i jezik kotlin, te firebase za bazu podataka.

Ključne riječi :Diplomski rad, Diplomski zadatak, MyUtilities, Utilities. Computing Utilities, Kotlin, Android Studio, Firebase, Firestore, emulator, app, gradle, manifest,Java, Mobilna aplikacija, Izračun režija, potrošnja režija, Layout

ABSTRACT

Myutilties is mobile android application which is used for computing overheads. Water, electricity and gas can be computed for different cities. User is manually entering the data inside the application for new and old month to get the amount he is supposed to play at the end of the month. All of the amounts if possible to save inside the database. Android studio and kotlin was used to build the appliaction and firebase was used as database.

Keywords : Diplomski rad, Diplomski zadatak, MyUtilities, Utilities. Computing Utilities, Kotlin, Android Studio, Firebase, Firestore, emulator, app, gradle, manifest,Java, Mobilna aplikacija, Izračun režija, potrošnja režija, Layout