Programiranje grafičkog sučelja u QML jeziku

Škrabo, Petra

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Petra Škrabo

PROGRAMIRANJE GRAFIČKOG SUČELJA U QML JEZIKU

Diplomski rad

Voditelj rada:
prof. dr. sc. Mladen Jurak

Zagreb, veljača 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj obitelji i prijateljima

Sadržaj

Uvod

Danas se sa sigurnošću može reći kako se velik dio ljudske populacije oslanja na računala, mobitele i druge pametne uređaje. Upravo iz tog razloga, kao jedan od većih izazova nameće se distribucija aplikacija za velik broj platformi. Primarni cilj je omogućiti izvođenje aplikacije na što više različitih platformi, dok u pozadini stoji jedan programski jezik ili skup alata koje je potrebno održavati. Samim time ostavljena je mogućnost brzog i efikasnog rješavanja problema, koje je odmah dostupno na svim raspoloživim platformama.

Postoje brojna razvojna okruženja i alati namijenjeni višeplatformskom razvoju aplikacija. U ovom radu bit će istaknuta biblioteka Qt, koja se koristi upravo u tu svrhu. Ono što Qt zaista čini vrlo moćnim alatom za razvoj na više platformi jest QML, deklarativni programski jezik posebice pogodan za izradu aplikacija s grafičkim korisničkim sučeljem. Izgradnja sučelja koristeći QML dostupna je kroz Qt Quick modul koji omogućuje već gotove QML tipove kao i njihove funkcionalnosti.

Ovaj rad će kroz svoja tri poglavlja dati pregled Qt biblioteke i QML jezika, te naposljetku svo teorijsko znanje primijeniti u praksi.

U prvom poglavlju opisuje se razvojni paket kompanije The Qt Company, u najnovijoj postojećoj verziji Qt6. Daje se pregled modula na koje je biblioteka Qt podijeljena te se opisuje Qt-ova integrirana razvojna okolina – Qt Creator, centralni mehanizam biblioteke – sustav signala i utora, te osnovni sustavi za izgradnju. Na kraju se predstavljaju temeljni koncepti za razvoj Qt aplikacija i kreiranje grafičkih korisničkih sučelja.

Drugo poglavlje opisuje programski jezik QML, podržan Qt QML i Qt Quick modulima. Stoga se posebno prolazi kroz oba navedena modula i njihove funkcionalnosti. Moduli nude mogućnost korištenja popriličnog broja QML tipova unutar *.qml* datoteke, čime se raspolaže s dovoljno znanja za izradu vlastitog Qt Quick projekta. Početak tog procesa opisan je na samom kraju drugog poglavlja.

Konačno, kako bi se cjelokupna analiza Qt biblioteke zaokružila, treće poglavlje opisuje izgled, funkcionalnost i proces implementacije grafičkog korisničkog sučelja na primjeru QTher aplikacije prikaz podataka vremenske prognoze. Time se ujedno razrađuje i dovršava izgradnja Qt Quick projekta započeta u drugom poglavlju.

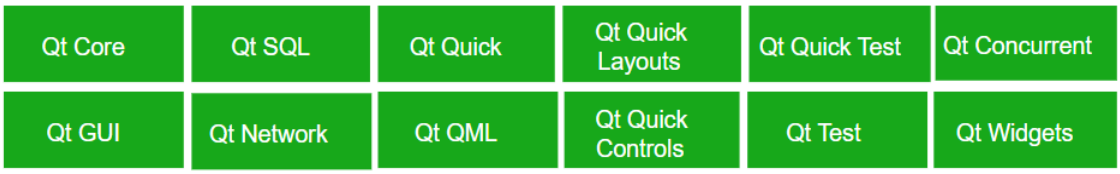
Poglavlje 1

Qt biblioteka

Qt je skup biblioteka i razvojnih alata namijenjenih jednostavnom razvoju korisničkih sučelja i softverskih aplikacija. Podržava razvoj za velik broj platformi, od Windows, Linux i macOS operativnih sustava, preko Android, iOS ili Windows mobilnih platformi, do ugrađenih (*embadded*) računalnih sustava.

Specifičnost Qt-a je da on sam po sebi nije programski jezik. Qt API[1] implementiran je u C++ jeziku koji je proširen određenim značajkama. Sam C++ je primarno namijenjen i za razvoj koda. Međutim, kod je moguće razvijati i u nekim drugim jezicima kao što su Python, PHP, Java ili QML. U ovom radu poseban naglasak bit će stavljen na deklarativni QML jezik.

Qt biblioteka sačinjena je od velikog broja modula, među kojima centralni dio otpada na module za razvoj grafičkog korisničkog sučelja. Dakako, biblioteka nudi i mnoštvo drugih funkcionalnosti poput podrške za mrežno programiranje, baze podataka, multimediju ili testiranje. Na taj način biblioteka je podijeljena u niz manjih modula prema funkcionalnostima. Slika 1.1 prikazuje glavne i najčešće korištene module prema službenoj Qt dokumentaciji.



Slika 1.1: Glavni moduli Qt biblioteke

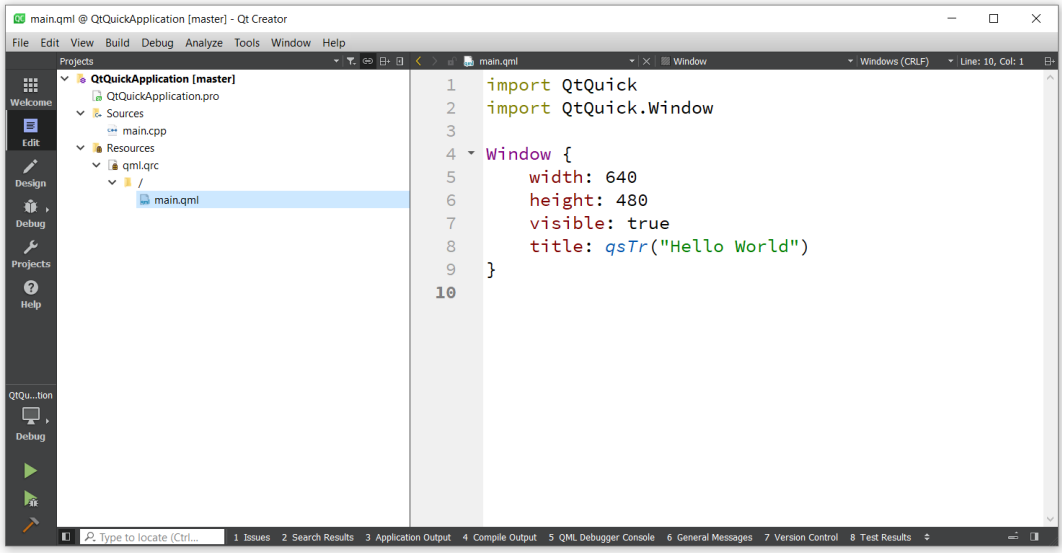
[1] Aplikacijsko programsko sučelje je skup definiranih pravila koja specificiraju komunikaciju s operativnim sustavom, hardverom ili drugim programima.

Dosad je predstavljeno šest glavnih verzija Qt-a. Trenutno je aktualna Qt6 inačica koja donosi niz noviteta, a fokus je stavljen na najčešće korištene module. Ključne promjene odnose se na podržanu C++17 verziju jezika, novu grafičku arhitekturu, uvođenje pomoćnog alata za izgradnju `CMake`, te novu generaciju QML-a.

O nadogradnji Qt QML modula i samom QML programskom jeziku bit će više riječi u nadolazećem poglavlju. Ususret tome, sada slijedi opis razvojnih alata, sustava za izgradnju te osnovnih koncepata za razvoj aplikacija i kreiranje korisničkih sučelja pomoću Qt-a.

1.1 Qt Creator

Qt dolazi zajedno s Qt Creatorom, vlastitom integriranom razvojnom okolinom (IDE) za C++, JavaScript i QML jezike. Jedan je od najbržih dostupnih IDE-a, a dostupan je na Linux, Windows i macOS platformama.



Slika 1.2: Integrirana razvojna okolina Qt-a

Qt Creator nudi niz značajki koji olakšavaju sve faze životnog ciklusa razvoja aplikacije, od kreiranja projekta do implementacije aplikacije za ciljne platforme. Kao jedna od glavnih prednosti ističe se lako upravljanje projektima. Qt omogućuje zajedničko korištenje projekata na različitim razvojnim platformama sa zajedničkim alatom za dizajn, razvoj i otklanjanje pogrešaka. Usto, u sebi ima integriran uređivač koda koji razumije C++ i QML programske jezike. Uređivač nudi semantičko isticanje koda, automatski dovršava kod

za vrijeme pisanja, zna ga refaktorirati, te vrši provjeru i analizu koda. Alati za otkrivanje pogrešaka, automatsko prevođenje jediničnih testova i analizu performansi programa velika su pomoć kod samog testiranja aplikacija. Također, pri stvaranju novog projekta preporučeno je korištenje jednog od podržanih sustava za verzioniranje koda.

Već je otprije poznato da je jedna od glavnih namjena Qt biblioteke razvoj grafičkih korisničkih sučelja. U skladu s tim, biblioteka uključuje i dizajner grafičkog korisničkog sučelja. Budući da Qt nudi dva načina programiranja sučelja (Qt Widgets i Qt Quick aplikacije), on dolazi s dvije razlite vrste uređivača – Qt Designer i Qt Quick Designer. Qt Designer namijenjen je za tradicionalna i jasno strukturirana korisnička sučelja, dok je Qt Quick Designer pogodniji za moderna i fluidna korisnička sučelja.

Od preostalih značajki bitno je još istaknuti one izravno vezane za QML jezik. Prva značajka odnosi se na konzolu za QML skriptu. Ona se koristi za izvršavanje JavaScript izraza u trenutnom kontekstu, odnosno dobivanje informacija o trenutnom stanju aplikacije i vrijednostima svojstava koje želimo ispitati. Osim toga, dostupan je i alat Inspector za istraživanje problema s Qt Quick korisničkim sučeljem te proučavanje strukture kreiranih objekata.

Sve spomenuto pokazuje koliko mogućnosti Qt Creator nudi i svjedoči o tome koliko ova okolina pojednostavljuje i ubrzava proces razvoja aplikacija. U nastavku slijedi detaljniji opis još jedne od mogućnosti koje ovo razvojno okruženje nudi, a to je sustav za automatsku izgradnju.

1.2 Sustav za izgradnju

Okolina Qt Creator zna kreirati projektne predloške za različite sustave izgradnje – `qmake`, `CMake`, `Meson` ili `Qbs`. Inicijalno, Qt dolazi s vlastitim sustavom za izgradnju `qmake`, dok je ostale podržane sustave za izgradnju potrebno postaviti (vidi [4]). Pri stvaranju novog projekta odabir sustava izgradnje je na programeru.

Primarni, `qmake` alat olakšava proces izrade razvojnih projekata na različitim platformama. To je ostvareno automatskim generiranjem datoteke *makefile* na temelju informacija iz projektne datoteke. Datoteka prepoznatljivog nastavka *.pro* u sebi može sadržavati listu korištenih Qt modula, opće konfiguracijske postavke projekta, datoteke izvornog koda, popis ostalih datoteka koje su dio projekta i druge varijable definirane u [14].

Primjer jedne takve projektne datoteke dan je isječkom koda 1.1. Prikazana je projektna datoteka Qt Quick projekta što se može iščitati iz prve linije koda gdje se varijabli `QT` pridružuje vrijednost `quick`. Time se u projekt uključuje Qt Quick modul. Nadalje, vrijednost `CONFIG` varijable označava da u projektu treba biti podržano korištenje `c++11`

standarda. Projekt još sadrži *main.cpp* datoteku izvornog koda, te *qml.qrc* datoteku, koje su pridružene varijablama `SOURCES` i `RESOURCES` redom.

```
1 QT += quick
2
3 CONFIG += c++11
4
5 SOURCES += \
6         main.cpp
7
8 RESOURCES += qml.qrc
```

Isječak koda 1.1: Primjer projektne datoteke

1.3 Signali i utori

Qt-ov mehanizam signala i utora vjerojatno je najveća posebnost u odnosu na značajke koje pružaju drugi okviri. Kod programiranja grafičkog korisničkog sučelja bitno je omogućiti međusobnu komunikaciju objekata bilo koje vrste. Primjerice, klik gumba za zatvaranje prozora trebao bi pozvati određenu metodu koja će podržati tu funkcionalnost. Većina drugih alata ovu vrstu komunikacije postiže pomoću povratnih (*callback*) poziva, no Qt za to koristi upravo mehanizam signala i utora.

Signali su funkcije povratnog tipa `void`, koje generira `moc` (*Meta-Object Compiler*). One se ne smiju implementirati u *.cpp* datoteci već ih programer samo deklarira. Signale emitira objekt kad se promijeni njegovo unutarnje stanje na neki način koji je od interesa za druge objekte ili korisnika. Objekti zainteresirani za odaslani signal definiraju utore. To su uobičajne C++ funkcije koje implementira programer i slijede standardna pravila jezika. Međutim, njihova posebnost je u tome da se mogu spojiti na signal i kao takve ih može pozvati bilo koja komponenta bez obzira na razinu pristupa. Veza signala i utora ostvaruje se funkcijom `connect()`.

```
connect(sender, &Sender::signalName, receiver, &Receiver::slotName);
```

Klasa koja emitira signal ne zna niti brine koji utori primaju signal. Isto tako, utori ne znaju ima li na njih spojenih signala. O svemu tome brine Qt-ov mehanizam koji osigurava da spajanjem signala na utor on bude pravovremeno pozvan s parametrima signala.

Na jedan utor može se spojiti po volji mnogo signala. U tom slučaju utori će se izvršavati jedan za drugim, redoslijedom kojim su spojeni, kada se signal emitira. Također, signal se

može spojiti na onoliko utora koliko je potrebno. Dopušta se i spajanje signala izravno na drugi signal pri čemu će se drugi signal emitirati kad god se emitira i prvi.

Sve to signale i utore čini vrlo moćnim mehanizmom. Koristiti ga mogu sve potklase ili klase izvedene iz `QObject` klase.

Isječkom koda 1.2 sad je dano rješenje problema zatvaranja prozora s početka.

```
1  #include <QApplication>
2  #include <QPushButton>
3
4  int main(int argc, char* argv[])
5  {
6      QApplication app(argc, argv);
7      QPushButton* quitButton = new QPushButton("Quit");
8      QObject::connect(quitButton, &QPushButton::clicked,
9                       &app, &QApplication::quit);
10
11     quitButton->show();
12     return app.exec();
13 }
```

Isječak koda 1.2: Spajanje signala i utora

Metoda `connect()` kao prva dva argumenta uzima pokazivač na element `QPushButton` koji emitira signal i pokazivač na signal `QPushButton::clicked`. Kao druga dva argumenta uzima pokazivač na objekt `app` koji je zainteresiran za taj signal i pokazivač na njegov utor `QApplication::quit`. Metoda naprosto povezuje signal na gumbu s utorom na objektu `app` što će rezultirati zatvaranjem prozora aplikacije pritiskom na gumb.

1.4 *Meta-Object* sustav

Mehanizam signala i utora omogućen je takozvanim *meta-object* sustavom Qt biblioteke. Sustav se, osim za komunikaciju među objektima, koristi i za identifikaciju tipova podataka za vrijeme izvršavanja programa odnosno korištenje dinamičkog sustava svojstava objekata. Temelji se na `QObject` klasi, `Q_OBJECT` makrou te *meta-object* prevodiocu (`moc`).

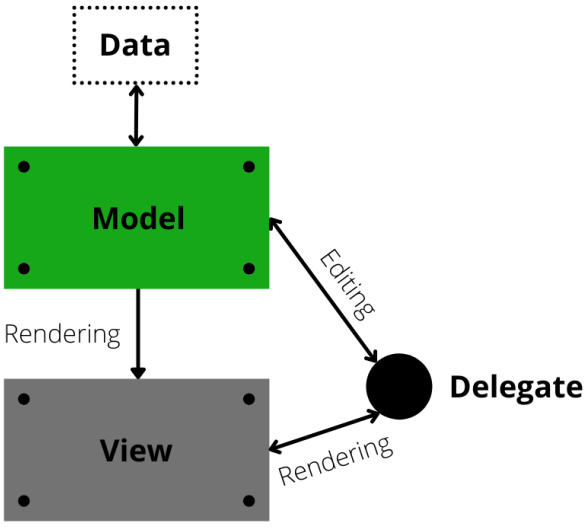
Klasa `QObject` je osnovna klasa koju objekt mora naslijediti kako bi mogao koristiti pogodnosti *meta-object* sustava. Osim toga, objekti koji je nasljeđuju imaju mogućnost korištenja i sustava roditeljstva. To znači da mogu imati definiranu djecu i svog roditelja, kao i metode za njihov pronalazak.

Usto, nužno je uvrstiti `Q_OBJECT` makro u privatni dio definicije klase koja koristi mogućnosti *meta-object* sustava Qt biblioteke. Ako pronađe barem jednu klasu koja u zaglavlju ima definiran `Q_OBJECT` makro, *meta-object* prevodilac će prilikom čitanja izvorne C++ datoteke stvoriti dodatnu C++ datoteku koja sadrži *meta-object* kod za sve klase u kojima je definiran makro. Dakle, osnovna zadaća `moc`-a je osigurati svakoj potklasi `QObject` klase kod potreban za implementaciju *meta-object* značajki.

1.5 Arhitekturalni obrazac *model-view*

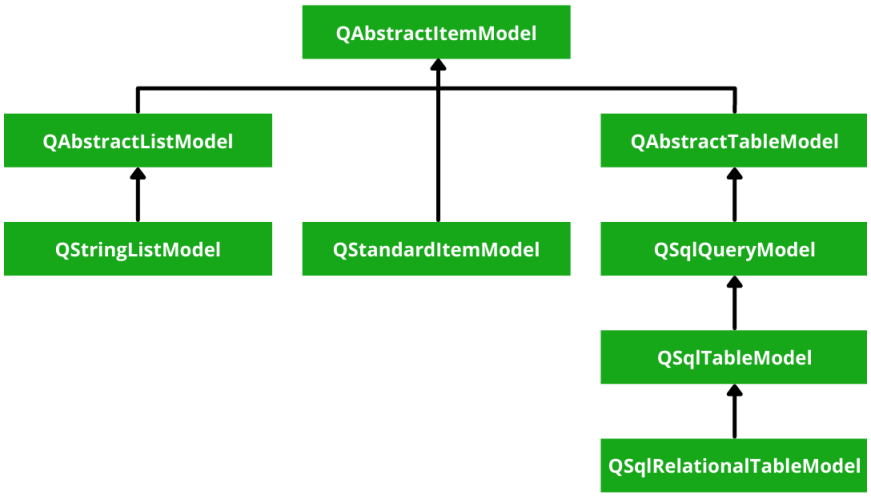
U većini slučajeva aplikacija mora prikazivati neke podatke korisniku. Ti podaci mogu dolaziti iz raznih izvora kao što su lokalne datoteke, baze podataka ili mrežni izvori. O njihovom prikazu Qt brine preko svojih modela, pogleda i delegata čime ujedno osigurava i podršku za *model-view* arhitekturalni obrazac.

Obrazac *model-view* često je korišten način izrade grafičkih korisničkih sučelja. Služi za odvajanje podataka od pogleda. Pri svakoj promjeni podataka, model obavještava poglede koji o njemu ovise, nakon čega se svaki pogled ažurira na odgovarajući način. Ovakav pristup omogućuje prikaz istih podataka u nekoliko različitih pogleda i implementaciju novih vrsta pogleda, bez mijenjanja temeljnih struktura podataka. U arhitekturu je uveden i koncept delegata kako bi rukovanje podacima bilo što fleksibilnije. Delegati omogućuju prilagođavanje načina na koji se ti podaci prikazuju i uređuju. Slijedi detaljan opis *model-view* arhitekture prikazane slikom 1.3.



Slika 1.3: Arhitekturalni obrazac *model-view*

Model čini vezu između podataka i korisničkog sučelja. On komunicira s izvorom podataka i pruža standardno sučelje za ostale komponente u arhitekturi. U Qt-u su svi modeli izvedeni iz klase `QAbstractItemModel` koja pruža dovoljno fleksibilno sučelje za različite poglede. Na slici 1.4 prikazani su gotovi modeli koje biblioteka nudi.



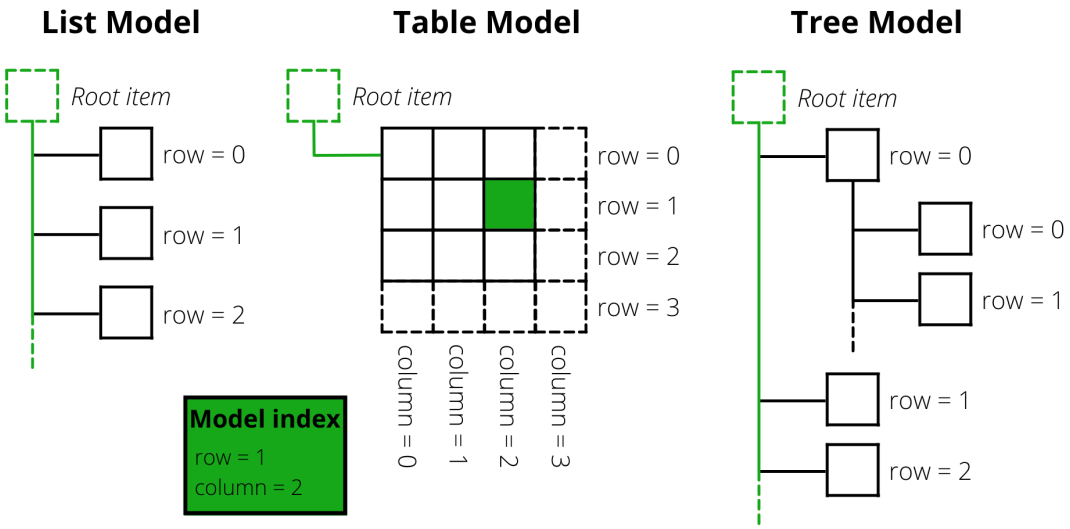
Slika 1.4: Hijerarhija klasa modela

Ukoliko standardni modeli ne zadovoljavaju zahtjeve, moguće je stvoriti i vlastiti. On će tada biti potklasa jedne od triju apstraktnih klasa modela.

Neovisno o načinu pohrane podataka, potklase reprezentiraju podatke kao hijerarhiju tablica s elementima (slika 1.5). Svaki element tablice predstavljen je svojim indeksom. Indeksi pružaju privremene reference na dijelove informacija te se mogu koristiti za dohvaćanje ili modificiranje podataka putem modela. Za dobivanje indeksa koji odgovara određenom podatku, modelu se moraju specificirati tri svojstva: broj retka, broj stupca te indeks modela nadređenog elementa.

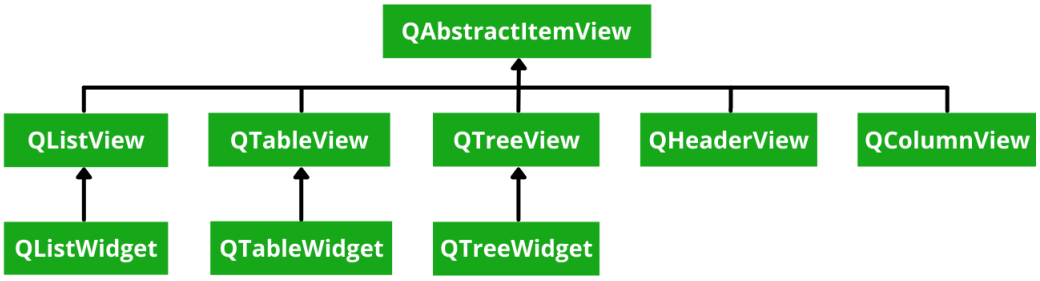
Pogledi sada preko indeksa dobivaju podatke iz modela te ih prikazuju korisnicima. Taj prikaz može biti potpuno drugačiji od temeljne strukture koja se koristi za pohranu podataka i ne mora nalikovati prikazu koji daje model. Dakle, pogledi upravljaju cjelokupnim izgledom podataka dobivenih iz modela. Usto, oni su još odgovorni i za obradu korisničkog unosa. U pozadini zapravo stoje delegati koji izvode interakciju i omogućuju fleksibilnost unosa.

Qt nudi gotove implementacije za različite vrste pogleda. Kao što je vidljivo sa slike 1.6, svaka od klasa pogleda temelji se na osnovnoj apstraktnoj klasi `QAbstractItemView`.



Slika 1.5: Modeli za reprezentaciju podataka

Također, iako su prikazane klase spremne za korištenje, moguće je prilagoditi i klasu za vlastiti pogled.



Slika 1.6: Hijerarhija klasa pogleda

Na kraju, cjelokupna komunikacija modela, pogleda i delegata odvija se pomoću signala i utora na sljedeći način:

- Signali iz modela obavještavaju pogled o promjenama podataka.
- Signali iz pogleda pružaju informacije o interakciji korisnika s elementima koji se prikazuju.
- Signali delegata koriste se tijekom uređivanja elemenata kako bi model i pogled obavijestili o trenutnom stanju.

Poglavlje 2

QML aplikacije

QML (*Qt Modeling Language*) je deklarativni programski jezik koji omogućava kreiranje aplikacija fokusiranih na korisničko sučelje u Qt biblioteci. Osmišljen je kako bi olakšao dinamičko povezivanje komponenti te osigurao jednostavnu ponovnu upotrebu i prilagodbu vizualnih komponenti. QML se ističe i kao specifikacija korisničkog sučelja budući da zapravo opisuje izgled komponenti, njihovu interakciju i međusobni odnos. Ovaj jezik ima vrlo čitljivu sintaksu i nalik je JSON formatu. Uključuje podršku za imperativne JavaScript izraze koji se najčešće koriste za dinamičko vezanje svojstava. Osim toga, QML u značajnoj mjeri koristi Qt koji omogućuje pristup gotovim tipovima i drugim Qt značajkama izravno iz QML aplikacija.

2.1 Qt QML modul

Sam QML programski jezik i osnovna infrastruktura za njega osigurani su Qt QML modulom. Osim okvira za razvoj aplikacija uključuje i razne biblioteke pogodne za ovaj jezik. Također, pruža API-je koji omogućuju programerima da prošire QML jezik prilagođenim tipovima i integriraju QML kod s JavaScript ili C++ kodom. To znači da ovaj modul obuhvaća i QML API i C++ API.

Korištenje C++ aplikacijskog programskog sučelja zahtijeva povezivanje s bibliotekom modula koji ga uključuje. Postoji nekoliko alata koji imaju namjensku podršku za takvo povezivanje. Primjerice, uz korištenje `CMake` sustava za izgradnju povezivanje se vrši korištenjem naredbe `find_package()` koja locira potrebne komponente modula u Qt6 paketu.

```
find_package(Qt6 COMPONENTS Qml REQUIRED)
target_link_libraries(mytarget PRIVATE Qt6::Qml)
```

U slučaju korištenja `qmake` sustava dovoljno je dodati modul kao vrijednost varijable `QT` u *.pro* datoteci.

```
QT += qml
```

Drugi, QML API kojeg Qt QML modul sadržava, definira i implementira različite pogodne tipove koji se koriste u tom jeziku. Osim toga, uključuje elementarne QML tipove koji daju osnovu za daljnja proširenja jezika. Temeljni nevizualni elementi pogodni za proširenja su `QtObject` i `Component`. Oni su, kao i ostali QML tipovi, dostupni tek nakon uvoza modula u *.qml* dokument na sljedeći način.

```
import QtQml
```

Na temelju svega navedenog vidljivo je da Qt QML modul pruža jezik i infrastrukturu za QML aplikacije. Međutim, većina bitnih elemenata koji su potrebni za izradu korisničkih sučelja s QML-om nalaze se u Qt Quick biblioteci. Ona nudi pregršt opcija za lakšu i efikasniju implementaciju grafičkih korisničkih sučelja. Detaljniji prikaz te biblioteke dan je u nastavku.

2.2 Qt Quick modul

Qt Quick je standardna biblioteka za pisanje QML aplikacija. Modul omogućava sve potrebne QML tipove i funkcionalnosti za izradu grafičkog korisničkog sučelja pomoću QML jezika. Pruža vizualni okvir s vlastitim koordinatnim sustavom, uključuje tipove za stvaranje i animiranje vizualnih komponenti, prima unos korisnika, kreira modele i poglede. Iz navedenog je jasno vidljivo kako Qt Quick osigurava sve što je potrebno za stvaranje aplikacije s bogatim fluidnim i dinamičkim korisničkim sučeljem. Svim Qt Quick funkcionalnostima biblioteke programer može pristupiti sljedećom naredbom za uvoz istoimenog modula.

```
import QtQuick
```

Pri izgradnji sučelja poseban naglasak stavljen je na ponašanje komponenti i način njihovog međusobnog povezivanja. Animacije i efekti prijelaza su osnovni koncepti ove biblioteke. Slijedi detaljan opis svih bitnih principa Qt Quicka.

2.2.1 Vizualni elementi

Biblioteka Qt Quick nudi širok spektar vizualizacija u QML-u. U njoj svi vizualni elementi nasljeđuju objekt `Item`. Taj objekt definira sve atribute koji su zajednički za vizualne komponente unatoč tome što nema vizualni izgled. Primjerice, pripadne koordinate `x` i `y`, širinu, visinu te smještaj elementa.

Za pozicioniranje vizualnih elemenata ovaj modul koristi sustav vizualnih koordinata. Na taj se način elementi smještaju unutar dvodimenzionalnog vizulanog okvira u odnosu na gornji lijevi kut koji predstavlja piksel $(0, 0)$. U ovom zamišljenom koordinatnom sustavu os x raste udesno, a os y prema dolje. Sve navedeno prikazano je slikom 2.1. Usto, moguće je definirati i poredak elemenata na z osi.

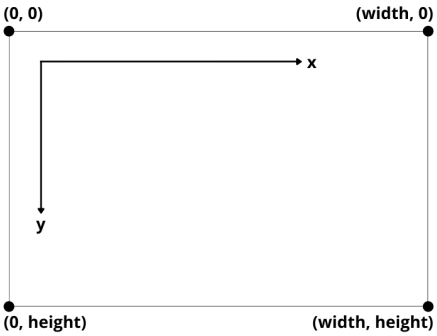
Ukoliko pri pozicioniranju elemenata dođe do kolizije, Qt Quick koristi rekurzivni algoritam za određivanje objekta koji će se nalaziti na vrhu. Općenito, elementi se crtaju redoslijedom kojim su stvoreni odnosno specificirani u QML datoteci. To potvrđuje isječak koda 2.1 koji će pri izvršavanju iscrtati plavi pravokutnik iznad zelenog (slika 2.2).

```
1  Rectangle {
2      color: "white"
3      width: 800; height: 500
4
5      Rectangle {
6          color: "green"
7          width: 300; height: 300
8      }
9
10     Rectangle {
11         color: "blue"
12         x: 100; y: 150;
13         width: 700; height: 300
14     }
15 }
```

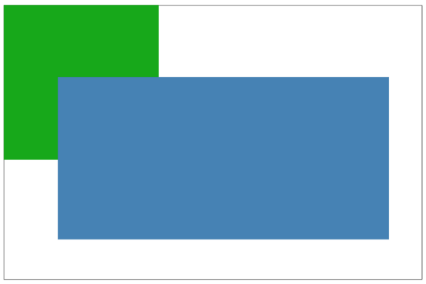
Isječak koda 2.1: Određivanje redoslijeda prikaza elemenata

U QML aplikaciji koja koristi Qt Quick postoje dvije vrste roditeljstva. Prva vrsta je vlasničko roditeljstvo u kojem je roditelj odgovoran za životni vijek svoje djece. Druga vrsta odnosi se na vizualno roditeljstvo. Takav odnos podrazumijeva da je položaj svakog pojedinačnog elementa određen u odnosu na koordinatni sustav njegovog roditelja. To znači da se za usporedbu `x` i `y` svojstava nesrodnih elemenata uglavnom treba odraditi pretvorba u isti koordinatni sustav. U tom se slučaju koriste koordinatne scene, odnosno prethodno opisani vizualni okvir, kao zajednički središnji koordinatni sustav.

U gotovo svim slučajevima obje vrste roditeljstva se poklapaju. Ipak, postoji i par iznimki o kojima se može detaljno pročitati u službenoj dokumentaciji Qt-a (vidi [5]).



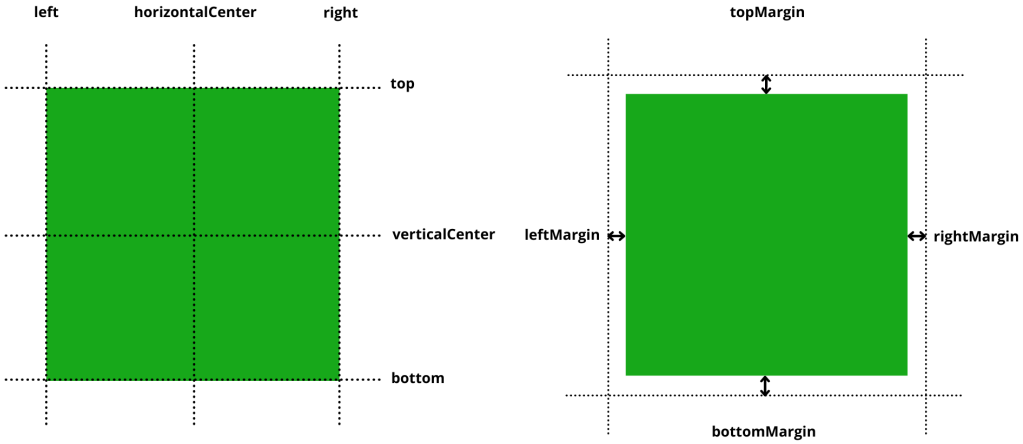
Slika 2.1: Vizualni okvir



Slika 2.2: Redoslijed prikaza elemenata

2.2.2 Pozicioniranje elemenata

Vizualni element u QML-u može se pozicionirati na razne načine. Najbitniji koncept odnosi se na sidrenje (*anchoring*), a podrazumijeva relativni razmještaj elemenata gdje se oni pričvršćuju jedan za drugog na točno definiranoj udaljenosti. Ovakvo pozicioniranje neovisno je o promjeni veličine ili lokacije usidrenih elemenata što uvelike pridonosi izradi dinamičkih korisničkih sučelja.



Slika 2.3: Pozicioniranje elemenata sidrenjem

Ostali koncepti pozicioniranja odnose se na apsolutno pozicioniranje, pozicioniranje definiranjem koordinata objekta i pozicioniranje dodjeljivanjem vezanih izraza svojstvima povezanim s njihovom lokacijom na ekranu. Na programeru je koji način će odabrati, no treba napomenuti kako fleksibilniji i intuitivniji načini mogu ići nauštrb određenih performansi.

2.2.3 Stanja, tranzicije i animacije

U svakom modernijem korisničkom sučelju prijelaz između stanja i animacija sučelja jedan je od glavnih koncepata. On se u Qt Quicku ističe kao najbitniji.

Stanje pojedine vizualne komponente smatra se skupom informacija koje opisuju kako i gdje se prikazuju pojedini sastavni dijelovi komponente, te svi podaci povezani s tim stanjem. Ime trenutnog stanja komponente sadržano je u svojstvu `state` kojeg imaju svi elementi koji nasljeđuju objekt `Item`. Osim inicijalno postavljenog stanja, mogu se definirati i dodatna, kreiranjem novih objekata tipa `State`.

```
1  Rectangle {
2      id: button
3      width: 75; height: 75
4      state: "released"
5
6      MouseArea {
7          anchors.fill: parent
8          onPressed: button.state = "pressed"
9          onReleased: button.state = "released"
10     }
11
12     states: [
13         State { name: "pressed"
14                 PropertyChanges { target: button; color: "green"} },
15         State { name: "released"
16                 PropertyChanges { target: button; color: "blue"} }
17     ]
18
19     transitions: [
20         Transition { from: "pressed"; to: "released"
21                      ColorAnimation { target: button; duration: 100} },
22         Transition { from: "released"; to: "pressed"
23                      ColorAnimation { target: button; duration: 100} }
24     ]
25 }
```

Isječak koda 2.2: Animirani prijelaz objekta između stanja

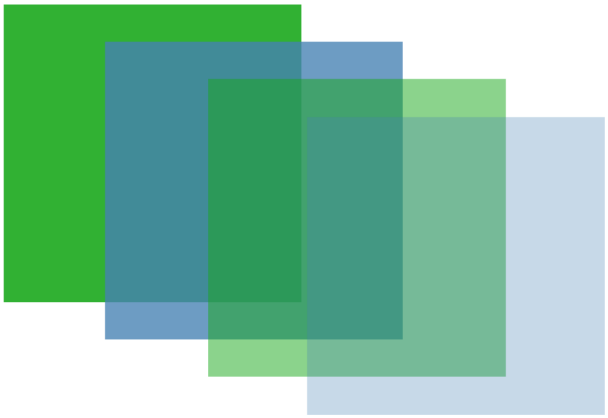
Promjene stanja izazivaju nagle promjene vrijednosti. Da bi one bile glađe i vizualno atraktivnije brinu se tranzicije. Odvijaju se na način da QML objekt `Transition` sadrži različite tipove animacija. Te animacije interpoliraju promjene svojstava koje su uzrokovane promjenama stanja. Tako naposljetku dolazimo do glatke animirane tranzicije između dvaju stanja pojedine vizualne komponente.

Jednostavni primjer koji reprezentira animirani prijelaz između definiranih stanja prikazan je isječkom koda 2.2. Naime, unutar QML tipa `Rectangle` definirana su dva stanja – `pressed` i `released`. Za svako stanje posebno je specificirano svojstvo `color` koje pravokutnik ima u tom stanju. Usto, kako bi se zagladio prijelaz između tih stanja, odnosno boja pravokutnika postepeno mijenjala, uvedene su tranzicije bazirane na `ColorAnimation` tipu animacije.

2.2.4 Grafički efekti

Pri izradi grafičkih korisničkih sučelja važno je obratiti pozornost na njihovu atraktivnost. Naravno, pretjerana upotreba vizualnih efekata može umanjiti korisničko iskustvo. Zato je važno osigurati jednostavnost i pružiti korisniku suptilnu komunikaciju. Slijedi kratki pregled najbitnijih efekata za postizanje željenog učinka.

Među najjednostavnije grafičke efekte ubrajamo prilagodbu prozirnosti elementa. Reguliranjem vrijednosti svojstva `opacity` privlačimo odnosno skrećemo pažnju s nekog elementa.



Slika 2.4: Svojstvo prozirnosti

Svi grafički elementi se mogu transformirati. Osnovne transformacije odvijaju se izmjenom ili postavljanjem vrijednosti pogodnim svojstvima – `x`, `y`, `scale`, `rotation`, `transformOrigin`. Također, moguće je definirati i detaljnije transformacije pridružujući liste objekata `Scale`, `Rotation` i `Translate` svojstvu `transform` vizualnog elementa.

Kod dodavanja efekata zanimljivo je imati u vidu modul Qt Quick Particles. On omogućuje prikaz 2D simulacija ili složenijih efekata poput eksplozija, vatrometa, gravitacije ili turbulencija.

Elementima je na vrlo jednostavan način moguće dodati i sjenu, zamutiti ih ili obojiti uz pomoć QML tipa `ShaderEffect`. Iako prekomjerno korištenje ovih efekata može rezultirati povećanom potrošnjom energije i usporenim performansama, uz pažljivo korištenje *shader* može uvelike pridonijeti vizualnom izgledu elementa.

Svi opisani grafički efekti privlače korisnikovu pažnju s ciljem pružanja intuitivnih naznaka o tome koji se događaji odvijaju u aplikaciji. Zato je pri izgradnji aplikacije važno da se oni ne zanemare.

2.2.5 Korisnički unos

Bitan dio pri dizajniranju grafičkog korisničkog sučelja odnosi mogućnost odgovora na unos korisnika. Unos se može odvijati preko pokazivačkih uređaja ili tipkovnice.

Prvi pristup podrazumijeva otkrivanje i reagiranje na klik miša prema položaju pokazivača u desktop aplikacijama, odnosno interakciju fizičkim dodirom zaslona na za to pogodnim uređajima. Tako koncipirana sučelja, vođena dodirom i mišem, u Qt Quicku podržana su različitim vizualnim tipovima i tipovima za unos. Među njima se ističu `MouseArea` i `Flickable`.

`MouseArea` je nevidljiva površina, ali sa svojstvom vidljivosti koje određuje je li područje transparentno za događaje miša. Informacije o položaju i kliku miša osigurane su signalima. To pokazuje isječak koda 2.3 u kojem objekt `button` u svom upravljaču signala reagira na klik miša, a potvrdu o reakciji ispisuje u konzolu.

`Flickable` objekt označava površinu koja se može pomicati i povlačiti čime uzrokuje promjenu pogleda na sebi podređene elemente. Takvo ponašanje čini osnovu elemenata koji su dizajnirani za prikaz velikog broja podređenih elemenata poput `ListView` i `GridView` pogleda.

```
1 Rectangle {
2     id: button
3     width: 100; height: 100
4
5     MouseArea {
6         anchors.fill: parent
7         onClicked: console.log("button clicked")
8     }
9 }
```

Isječak koda 2.3: Područje osjetljivo na klik miša

Nadalje, Qt Quick pruža bilo kojem vizualnom elementu mogućnost primitka unosa s tipkovnice. Jedan takav primjer dan je isječkom koda 2.4 u kojem se objekt tipa `Rectangle` postavlja osjetljivim na pritisak tipke „A”. Posebno, postoje i tekstualni elementi koji su automatski osjetljivi na događaje tipkovnice što rezultira simultanim prikazom odgovarajućeg teksta na ekranu. Tu se ubraja tip `TextInput` za prikaz jednog tekstualnog retka koji se može uređivati, odnosno tip `TextEdit` koji na zaslon postavlja više redaka s istim mogućnostima.

```
1  Rectangle {
2      id: button
3      width: 100; height: 100
4      focus: true
5
6      Keys.onPressed: {
7          if (event.key == Qt.Key_A) {
8              console.log('Key A was pressed');
9              event.accepted = true;
10         }
11     }
12 }
```

Isječak koda 2.4: Područje osjetljivo na unos s tipkovnice

Najbolji način za primitak unosa ovisi o samoj aplikaciji i uređaju na kojem se ona nalazi. Ono što je sigurno jest da ova biblioteka može programeru dati na izbor koji način odabrati.

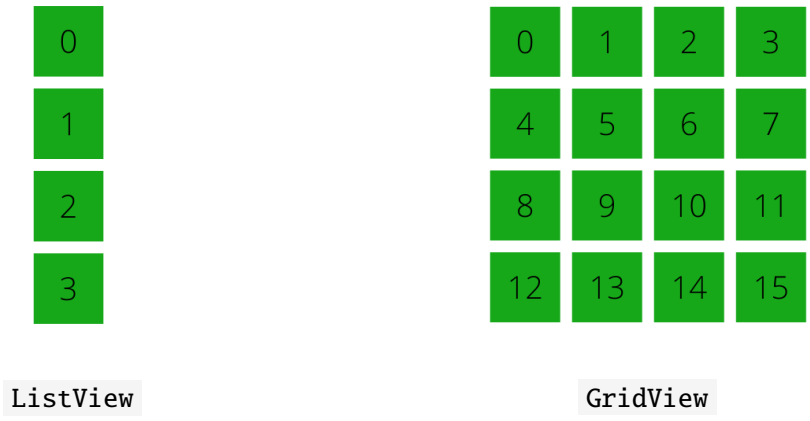
2.2.6 Modeli podataka i pogledi

Detaljan opis modela, pogleda i delegata u Qt-u prikazan je u poglavlju 1.5. Ovdje će fokus biti na njihovom korištenju preko Qt Quick biblioteke.

Kao što je već rečeno, podaci se korisnicima prikazuju preko pogleda. U Qt Quicku su standardni pogledi `ListView`, `GridView` i `PathView` dio osnovnih grafičkih elemenata. Svaki od pogleda ima definirana vlastita svojstva i ponašanja preko kojih je moguća njihova vizualna prilagodba. Za uređivanje izgleda sadržanih elemenata pogledi koriste delegate. Oni definiraju predložak za prikaz svakog pojedinog podatka.

Podaci koji se prikazuju i njihova struktura sadržani su u modelima. U Qt QML modulu postoji nekoliko tipova za kreiranje modela. Ovisno o prirodi problema, moguće je definirati jednostavnu hijerarhiju elemenata `ListModel`, konstruirati `XmlListModel` model iz podataka u XML formatu, napraviti `ObjectModel` model koji sadrži vizualne objekte ili specificirati model jednim objektom. Cijeli broj se također može koristiti kao model koji

će u tom slučaju sadržavati onoliko objekata koliko je definirano modelom. Uz već gotove modele, Qt nudi i opciju definiranja prilagođenog modela u C++ jeziku, kojeg je kasnije moguće učiniti dostupnim QML-u.



Slika 2.5: Inicijalni raspored elemenata pogleda

Na kraju, za povezivanje svih slojeva *model-view* arhitekture i prikaz dostupnih podataka potrebno je povezati svojstvo `model` korištenog pogleda s odgovarajućim modelom i svojstvo `delegate` tog pogleda s objektom `Component` ili nekim drugim kompatibilnim tipom. Konkretan primjer ovakvog povezivanja bit će prikazan i detaljno objašnjen u poglavlju 3.

2.2.7 Pogodni tipovi

U dinamičnom korisničkom sučelju treba često reagirati na velik broj događaja i to uglavnom s različitom logikom odgovora. Za ove koncepte QML u sebi ima ugrađenu podršku za dinamičko stvaranje i upravljanje objektima.

Među osnovne značajke jezika ubraja se vezivanje svojstava (*property bindings*). Jedan jednostavan primjer takvog vezivanja dan je isječkom koda 2.5. Naime, visina pravokutnika `blueRectangle` vezana je za visinu njegovog roditelja. Time se, svakom promjenom visine roditeljskog pravokutnika, visina plavog pravokutnika automatski ažurira kako bi bila jednaka roditeljskoj visini. QML tip `Binding` također dopušta dinamičko vezivanje svojstava te time definira međusobnu ovisnost svojstava različitih elemenata. Svakom promjenom vrijednosti vezanog svojstva, ono se automatski ažurira prema zadanom odnosu.

```
1  Rectangle {
2      width: 200; height: 200
3
4      Rectangle {
5          id: blueRectangle
6          width: 100
7          height: parent.height
8          color: "blue"
9      }
10 }
```

Isječak koda 2.5: Vezivanje svojstava visine

Pored toga, dinamičko upravljanje objektima podržano je u Qt Quick `Connections` tipu. Kreiranjem instance tog tipa moguće je dinamički potaknuti da signali koje emitira jedan objekt pokreću metode nekog drugog objekta. Uobičajan način za povezivanje na signale u QML jeziku je uvođenje upravljača signala (*signal handlers*) koji reagiraju na primitak signala.

Osim dinamičkog upravljanja, podržano je i dinamičko kreiranje objekata. Jedan od načina koje QML nudi za to je implementacija objekta unutar imperativnog JavaScript koda. Navedeni pristup je koristan za odgodu instanciranja objekta dok to ne bude potrebno. Usto, omogućava da vizualni elementi budu reakcija na događaje koji se odvijaju unutar aplikacije, čime se ujedno smanjuje vrijeme pokretanja aplikacije.
Postoje dva načina izrade objekata pomoću JavaScript koda. `Qt.createComponent()` poziv dinamički učitava komponentu definiranu u QML datoteci preko njenog URL-a, dok funkcija `Qt.createQmlObject()` kreira QML objekt iz zadanog stringa. Oba navedena načina prikazana su isječkom koda 2.7. Prvi način, dan linijom 1, kreira komponentu `MyComponent` iz QML datoteke koja se nalazi unutar istog direktorija. Drugim načinom (linije 3-6) kreira se tip `Rectangle`, sa svim zadanim svojstvima, unutar QML elementa čije svojstvo `id` je postavljeno na `parentItem`.

```
1  var component = Qt.createComponent("MyComponent.qml");
2
3  var newObject = Qt.createQmlObject(
4                      'import QtQuick 2.0;
5                      Rectangle {color: "red"; width: 20; height: 20}',
6                      parentItem);
```

Isječak koda 2.6: Dinamičko kreiranje QML objekta

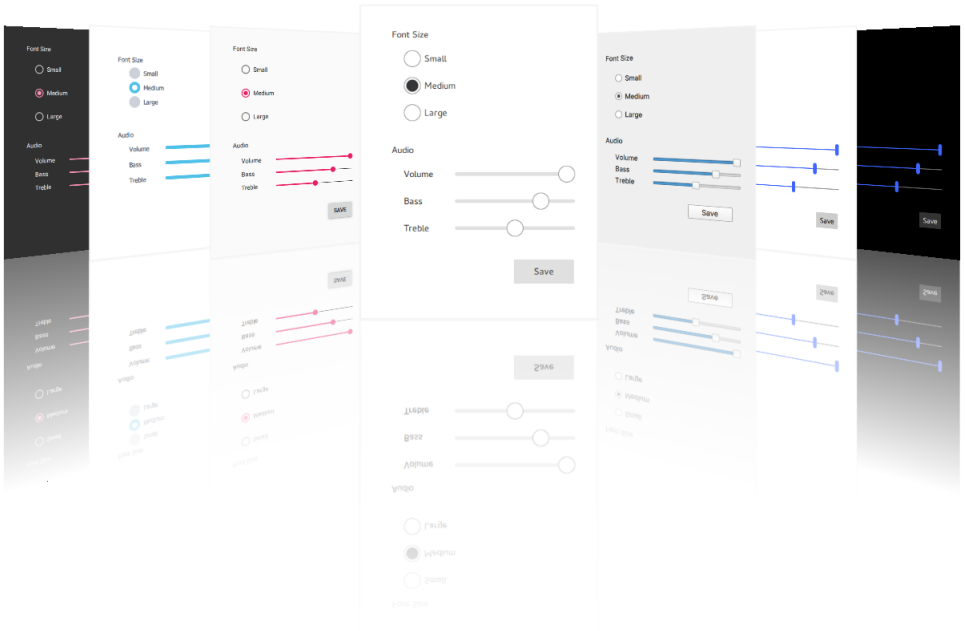
Pored dosad spomenutih tipova, uz uključenu Qt Quick biblioteku, moguće je koristiti i tipove `Loader`, `Repeater`, `ListView`, `GridView` ili `PathView`, ovisno o potrebi. Oni također podržavaju upravljanje dinamičkim objektima i pružaju deklarativni API.

Dakle, Qt Quick nadograđuje i dodatno proširuje osnovnu podršku pogodnih tipova koju nudi QML jezik.

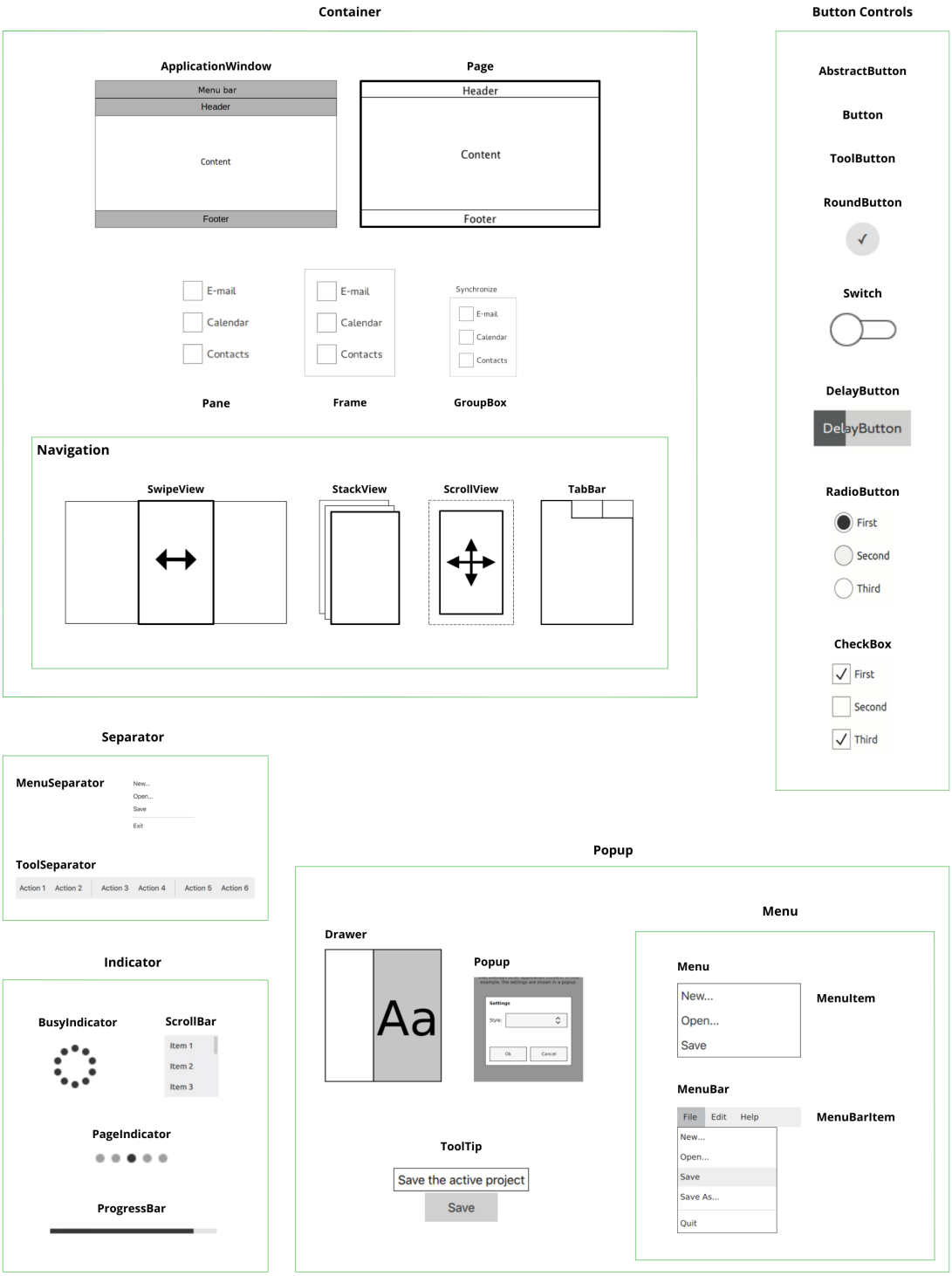
2.2.8 Kontrole korisničkog sučelja

Qt Quick pruža skup već gotovih kontrola koje se mogu koristiti za izgradnju kompletnih korisničkih sučelja. Za njihovu primjenu dovoljno je u projekt uključiti modul Qt Quick Controls. Među kontrolama koje nudi između ostalog mogu se naći i gumbi, izbornici ili pogledi. Na slici 2.7 može se vidjeti koliki je opseg ovog modula.

Također, modul Qt Quick Controls dolazi s nekoliko ugrađenih stilova – `Basic`, `Fusion`, `Imagine`, `macOS`, `Material`, `Universal`, `Windows`. Stilovi se mogu prilagođavati, a po potrebi je omogućena izrada vlastitih stilova.



Slika 2.6: Stilovi Qt Quick Controls modula



Slika 2.7: Korisničke kontrole Qt Quick biblioteke

2.3 Osnove QML jezika

Kao jezik s više paradigmi QML omogućuje definiranje objekata u smislu njihovih atributa i načina reakcije na promjene drugih objekata. Za razliku od čisto imperativnog koda, deklarativna sintaksa ovog jezika integrira promjene atributa i ponašanja izravno u definicije pojedinačnih objekata. Te definicije se kasnije, u slučaju da je potrebno složeno prilagođeno ponašanje aplikacije, mogu uključiti u imperativni kod.

Svaki QML dokument može se podijeliti na dva dijela. Prvi podrazumijeva dio koji sadrži jednu ili više naredbi za uvoz, dok se u drugom dijelu deklariraju objekti.

Uvoz može uključivati verzionirani prostor imena (*namespace*), JavaScript datoteku ili relativni put do direktorija koji sadrži prethodno definirane QML objekte. Takvi QML objektni tipovi definirani su samostalnim *.qml* dokumentom koji sadrži izvorni QML kod, nakon čega su spremni za daljnje korištenje unutar cijele aplikacije.

Sintaktički gledano, u drugom dijelu datoteke blok QML koda definira stablo QML objekata koje je potrebno kreirati. Objekti se definiraju korištenjem deklaracija koje opisuju tip objekta kao i atribute koji se tom objektu daju. Deklaracija se sastoji od naziva tipa objekta nakon čega slijedi skup vitičastih zagrada. Unutar ovih zagrada deklariraju se svi atributi i podređeni objekti.

2.3.1 QML tipovi

Tipovi koji se mogu koristiti u definiciji hijerarhije objekata iz drugog dijela QML dokumenta mogu biti podržani izravno QML jezikom, registrirani u C++ dijelu koda ili definirani u zasebnoj QML datoteci.

Osnovni QML tipovi

QML podržava niz osnovnih tipova odnosno tipova koji se referiraju na jednostavne vrijednosti. Takvi tipovi služe za označavanje jedne vrijednosti ili vrijednosti koja sadrži jednostavan skup parova svojstvo-vrijednost. Primjer prve uporabe je tip `int` čija vrijednost se odnosi na jedan broj, dok se tip `size` odnosi na vrijednosti atributa `width` i `height` čime označava jedan par. Prilikom pridruživanja varijable ili svojstva osnovnog tipa drugoj varijabli ili svojstvu, vrši se kopiranje po njihovoj vrijednosti.

Najosnovniji tipovi podržani su QML jezikom inicijalno i za njih nije potrebno uvoziti module. Međutim, moguće je proširiti paletu osnovnih tipova uvozom dodatnih QML modula. Kao primjer dana je tablica 2.1 osnovnih tipova koji se mogu koristiti ovisno o uvozu.

Osnovni QML tipovi	bool	double	int	real	list	enumeration	string	url	var		
Osnovni Qt Quick tipovi	color	date	font	matrix4x4	point	quaternion	rect	size	vector2d	vector3d	vector4d

Tablica 2.1: Osnovni tipovi

JavaScript tipovi

QML podržava JavaScript objekte i nizove. Prema tome, bilo koji standardni JavaScript tip može se stvoriti i pohraniti pomoću QML-ovog osnovnog generičkog tipa `var`.

QML objektni tipovi

Objektni QML tip je tip iz kojeg se može instancirati QML objekt. U sintaktičkom smislu, to je tip koji se može koristiti za deklariranje objekata, navođenjem naziva tipa nakon kojeg slijede atributi objekta unutar vitičastih zagrada. To je u potpunoj suprotnosti s osnovnim tipovima koji se ni u kojem slučaju ne mogu koristiti za deklariranje objekata.

Svi objektni tipovi u QML-u izvedeni su iz tipa `QtObject`. Uglavnom ih osiguravaju QML moduli i za njihovo korištenje je dovoljno uvesti modul u aplikaciju. Ipak, mogu se definirati i vlastiti objektni tipovi stvaranjem *.qml* datoteke koja definira tip ili deklariranjem objekta u C++ dijelu koda. Bitno je spomenuti da u svakom slučaju naziv tipa mora početi velikim slovom kako bi se uspješno deklarirao.

2.3.2 Atributi QML objekata

Svaki tip QML objekta ima definiran skup atributa. Instance objekta tog tipa kreiraju se zajedno sa skupom atributa koji su definirani upravo za tu vrstu objekta. Za pojedini element postoji više različitih vrsta atributa koje moguće je specificirati. U nastavku slijedi njihov pregled.

Atribut `id`

Svaki QML objekt ima točno jedan `id` atribut kojeg osigurava sam jezik. Ovaj atribut se ne može redefinirati niti nadjačati bilo kojim tipom QML objekta. Služi za identifikaciju objekta odnosno osigurava mogućnost da se drugi objekti unutar komponente na njega mogu referirati. Zbog toga je vrijednost ovog atributa unutar opsega svoje komponente uvijek jedinstvena.

Svojstva

Svakom svojstvu se može dodijeliti statička vrijednost ili ga vezati za dinamički izraz. Ono se može definirati u C++ klasi korištenjem `Q_PROPERTY` makroa ili u deklaraciji objekta u QML dokumentu na dolje prikazan način.

```
[default] [required] [readonly] property <propertyType><propertyName>
```

Ključne riječi unutar uglatih zagrada u navedenoj definiciji svojstva su opcionalne i mijenjaju semantiku svojstva koje se deklarira. Definicija objekta može imati samo jedno `default` svojstvo, kojem se dodjeljuje vrijednost ako je objekt deklariran unutar definicije drugog objekta, bez da se ono deklarira kao vrijednost određenog svojstva. Pored toga, korištenje ključne riječi `required` označava da su tako definirana svojstva obavezna za postavljanje pri kreiranju instance objekta. Posljednja, `readonly` ključna riječ, govori da se radi o svojstvu koje se nakon što mu je pri inicijalizaciji dodijeljena vrijednost može samo čitati. Nakon opcionalnih ključnih riječi slijedi obavezna `property` oznaka da se radi o definiciji svojstva.
Kao prilagođeni tip svojstva (u gornjem izrazu `propertyType`) može se koristiti bilo koji od osnovnih QML tipova opisanih u odjeljku 2.3.1 izuzev enumeracija. Kasnije se tom svojstvu može dodijeliti samo vrijednost koja odgovara definiranom tipu. Iza tipa, deklaracija svojstva treba sadržavati ime svojstva `propertyName`.

Nakon ispravne definicije, svojstvu se može dodijeliti vrijednost. Postoje dva načina za to, dodjeljivanje vrijednosti pri inicijalizaciji i imperativno pridruživanje vrijednosti svojstvu iz JavaScript koda. Definicija navedenih načina dana je sljedećim dvoma izrazima redom.

```
<propertyName> : <value>
[<objectId>] : <propertyName> = value
```

Vrijednosti koje se dodjeljuju svojstvima mogu biti ili statičke vrijednosti koje ne ovise o drugim svojstvima ili vezani izrazi. To su JavaScript izrazi koji opisuju odnos svojstva s drugim svojstvima. Pri promjeni vrijednosti bilo koje ovisnosti svojstava QML mehanizam automatski ponovno procjenjuje izraz i svojstvu dodjeljuje novu vrijednost.

QML uz osnovne vrste svojstava nudi i neke specijalne, kao što su svojstva koji sadrže listu QML objekata, grupirana svojstva ili *alias* svojstva. Detaljno objašnjene navedenih vrsta uz primjere može se pronaći u [10].

Naposljetku, bitno je još spomenuti da se deklaracijom prilagođenog svojstva implicitno stvara signal koji se emitira pri promjeni vrijednosti tog svojstva. Primjerice, definiranom svojstvu pod nazivom `propertyName`, pridružen je signal promjene vrijednosti s

upravljačem pod nazivom `onPropertyNameChanged`. No, više o signalima kao atributima objekta govori sljedeći odjeljak.

Signali i upravljači signala

Signal je obavijest od objekta da se dogodio neki događaj. Primjerice, može označavati da se promijenilo neko svojstvo ili da je neka animacija započela. Na taj događaj odgovara se tako da se pri emitiranju signala poziva odgovarajući upravljač signala. On bi trebao biti deklariran unutar definicije objekta koji emitira signal i sadržavati blok JavaScript koda koji će se izvršiti pri pozivu.

Signal za određeni tip objekta može se definirati i u C++ dijelu koda korištenjem makroa `Q_SIGNAL`. Drugi način definiranja jest u deklaraciji objekta unutar QML dokumenta na način prikazan donjim izrazom.

```
signal <signalName>[([<type><parameter name>[, ...]])]
```

Ukoliko signal nema parametre, oble zagrade nakon imena signala nisu obavezne. S druge strane, ako se koriste parametri, tipovi parametara moraju biti deklarirani. Mogući tipovi su, jednako kao i za tipove svojstava, svi osnovni QML tipovi osim enumeracija.

Dakle, poziv signala kao metode emitira signal. U tom slučaju bit će pozvani svi relevantni upravljači signala. Pokazat će se da oni su zapravo posebna vrsta atributa metode.

Metode

Metoda nekog QML objekta je funkcija koja se poziva kad god je potrebno izvršiti neku obradu ili pokrenuti daljnje događaje. Može se povezati sa signalom tako da se automatski poziva pri svakom emitiranju signala. Definicija metode može se vršiti u C++ kodu pomoću `Q_INVOKABLE` odnosno `Q_SLOT` makroa ili alternativno u QML dokumentu korištenjem sljedeće sintakse.

```
function <functionName>([<parameterName>[, ...]]) { <body> }
```

Za razliku od signala, tipovi parametara metode ne moraju se deklarirati budući da su zadani prema tipu `var`. Metode se mogu dodati QML tipu u svrhu definiranja samostalnih blokova JavaScript koda koji se kasnije mogu ponovno koristiti. Smiju se pozvati interno ili od strane vanjskih objekata.

Enumeracije

Enumeracija osigurava fiksni skup za izbor imena. U QML datoteci ovaj atribut se deklarira korištenjem ključne riječi `enum`. Detaljnije o enumeraciji kao atributu može se vidjeti u [6].

Na kraju odjeljka o atributima QML objekata, navodi se primjer jednog objekta sa svim bitnim tipovima atributa, kako bi se sve opisano pokazalo u praksi.

```
Rectangle {
    // atribut id
    id: rectangle
    // pridruživanje vrijednosti svojstvu pri inicijalizaciji
    property color itemColor: setColor()
    // definicija signala kao atributa
    signal blueAssigned

    // upravljač definiranog signala
    onBlueAssigned: console.log("blue assigned")
    // upravljač implicitno stvorenog signala promjene vrijednosti svojstva
    onItemColorChanged:  console.log("color changed")

    // metoda kao atribut
    function setColor() {
        blueAssigned() // emitiranje signala
        return "blue"
    }
}
```

Isječak koda 2.7: Osnovni tipovi atributa QML objekta

2.3.3 QML i JavaScript integracija

JavaScript okruženje koje pruža QML može pokretati različite valjane JavaScript izraze. Ujedno uključuje brojne pomoćne metode koje pojednostavljuju izgradnju korisničkog sučelja i interakciju s QML okruženjem. Također, podržan je i uvoz *.js* datoteke s JavaScript kodom te korištenje svih uvozom dobivenih funkcionalnosti.

JavaScript kod mogu sadržavati različiti dijelovi QML dokumenta. Preciznije, taj kod se može nalaziti u tijelu vezanog svojstva ili upravljača signala te u definiciji vlastite metode.
Prva primjena, već spomenuta u sklopu odjeljka u potpoglavlju 2.2.7, koristi JavaScript izraz za opis veze između svojstava QML objekta. Za takvo definiranje pogodan je svaki

JavaScript izraz, sve dok je rezultat izraza vrijednost čiji tip se može dodijeliti svojstvu. Primjer je jedne takve uporabe dan je isječkom koda 2.8. Svojstvo `color` objekta `Rectangle` ovisi o svojstvu `pressed` objekta `TapHandler`.

```
1  import QtQuick
2
3  Rectangle {
4      id: colorbutton
5      width: 200; height: 80;
6
7      color: inputHandler.pressed ? "steelblue" : "lightsteelblue"
8
9      TapHandler {
10         id: inputHandler
11     }
12 }
```

Isječak koda 2.8: Vezivanje svojstava JavaScript izrazom

JavaScript izraz može biti reakcija na neki događaj. U tom slučaju izraz se koristi u tijelu upravljača signala i evaluira se automatski kadgod QML objekt emitira pridruženi signal. Upravljač signala obično je vezan za takav izraz kako bi pokrenuo druge događaje ili dodijelio vrijednost svojstvima. U slučaju da se upravljaču signala pridružuje JavaScript metoda, moguće ju je definirati i samostalno unutar QML objekta, kao metodu tog objekta. Tako definirane metode još se mogu koristiti u slučaju vezivanja svojstava jednako kao i u funkcijama drugih QML objekata.

2.3.4 QML i C++ integracija

QML je dizajniran tako da se na lak način može proširiti C++ kodom. Klase u Qt QML modulu omogućuju učitavanje i upravljanje QML objektima iz C++ dijela koda, a priroda integracije dopušta da se C++ funkcionalnost poziva izravno iz QML-a. Također, moguće je odvojiti kod korisničkog sučelja od logičkog dijela aplikacije na način da se prvi dio implementira QML-om, a drugi C++ kodom. Sve to, kao i mogućnost kombiniranja s JavaScript kodom koja je opisana u prethodnom odjeljku, pogoduje razvoju hibridnih aplikacija koje u implementaciji koriste QML, JavaScript i C++ jezike.

Da bi QML mogao koristiti podatke i funkcionalnosti koje osigurava neka C++ klasa, ona mora biti izvedena iz `QObject` klase. Time QML-u postaju dostupni signali, metode i svojstva klase. Međutim, takva klasa se ne može koristiti kao QML tip sve dok se ne uvede u sustav tipova nakon čega se može deklarirati i instancirati kao bilo koja obična vrsta QML objekta. Da bi se klasa uspješno registrirala kao instancibilni QML tip, potrebno je

dodati makroe `QML_ELEMENT` ili `QML_NAMED_ELEMENT(<name>)` deklaraciji klase odnosno izraze `CONFIG+=qmltypes`, `QML_IMPORT_NAME` i `QML_IMPORT_MAJOR_VERSION` projektnoj datoteci.

Klasa izvedena iz `QObject` klase ne mora se uvijek moći instancirati. Za taj slučaj QML pruža nekoliko makroa za registriranje tipova koji nisu instancibilni. Ovdje se ubrajaju makroi `QML_ANONYMUS`, `QML_INTERFACE`, `QML_UNCREATABLE` te `QML_SINGLETON` o kojima detaljno piše [12].

Bitno je još izdvojiti često korištene makroe `Q_INVOKABLE` i `Q_PROPERTY`, već prije spomenute. Primjenom makro naredbe `Q_INVOKABLE` na deklaraciju funkcije članice klase ostvaruje se mogućnost pozivanja funkcije putem *meta-object* sustava. U deklaraciji funkcije naredba se piše ispred njenog povratnog tipa. Primjer deklaracije jedne takve funkcije prikazan je donjim izrazom.

```
Q_INVOKABLE void invokableMethod();
```

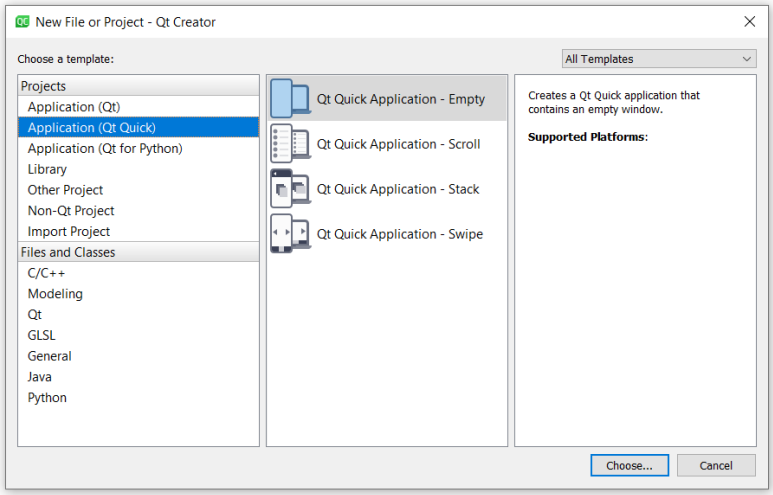
Drugi, `Q_PROPERTY` makro definiran u nastavku danom sintaksom, koristi se za deklariranje svojstava. Naziv i tip svojstva te funkcija pristupa `READ` obavezni su kod pisanja deklaracije. Tip svojstva može biti korisnički definiran ili neki od tipova podržanih klasom `QVariant`, koja je zapravo unija najčešće korištenih Qt tipova. Pri deklaraciji je još uobičajno koristiti funkciju `WRITE`, dok ostali atributi popisani u definiciji nisu obavezni. Svi atributi su inicijalno postavljeni na vrijednost `true` izuzev atributa `USER`.

```
Q_PROPERTY(type name
        (READ getFunction [WRITE setFunction] |
         MEMBER memberName [(READ getFunction | WRITE setFunction)
        [RESET resetFunction]
        [NOTIFY notifySignal]
        [REVISION int | REVISION(int[, int])]
        [DESIGNABLE bool]
        [SCRIPTABLE bool]
        [STORED bool]
        [USER bool]
        [BINDABLE bindableProperty]
        [CONSTANT]
        [FINAL]
        [REQUIRED])
```

U ovom dijelu opisani su osnovni načini integracije QML i C++ jezika. Primjena istih bit će detaljno opisana u odjeljku 3.3.1 idućeg poglavlja. No, postoje i drugi načini interakcije spomenutih jezika o kojima se više govori [7].

2.4 Izrada Qt Quick projekta

U odjeljku 1.1 opisano je kako integrirana razvojna okolina Qt Creator uvelike olakšava proces razvoja Qt aplikacija. U skladu s tim, ova okolina zna postaviti projektna okruženja i za sve tipove Qt Quick aplikacija. Qt Creator stvara projektni predložak sa svim karakterističnim datotekama ovisno o ciljnoj platformi.

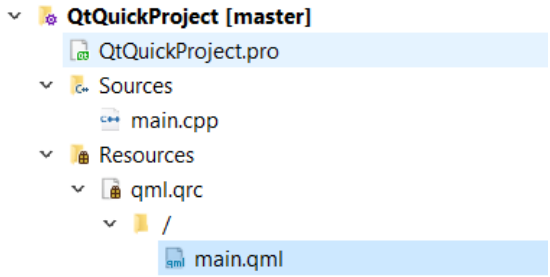


Slika 2.8: Kreiranje novog Qt Quick projekta

Kreiranje novog Qt Quick projekta vrši se kroz izbornik Qt Creatora biranjem sljedećih opcija File → New File or Project... → Application (Qt Quick). Taj postupak otvara prozor koji nudi mogućnost odabira različitih tipova projekta (slika 2.8). Budući da je fokus stavljen na Qt Quick aplikacije, izbor je sljedeći:

- **Empty** – Qt Quick aplikacija koja sadrži prazan prozor,
- **Scroll** – Qt Quick Controls aplikacija koja sadrži kontrolu `ScrollView`,
- **Stack** – Qt Quick Controls aplikacija s kontrolom `StackView` za prikaz sadržaja, te elementima `Drawer` i `ToolBar` za navigiranje aplikacijom,
- **Swipe** – Qt Quick Controls aplikacija s kontrolom `SwipeView` za navigaciju.

Nakon odabira vrste projekta, slijedi niz popratnih dijaloških okvira u kojima se postavljaju dodatne opcije projekta. Prvi u nizu je okvir Project Location u kojeg se unosi naziv i lokacija projekta. U ovom koraku posebno je bitno obratiti pozornost budući da naziv projekta kasnije nije jednostavno promijeniti. Potom je moguće odabrati sustav za izgradnju – `qmake`, `CMake` ili `Qbs`, stil Qt Quick kontrola, osnovni jezik aplikacije te minimalnu potrebnu verziju Qt-a koja određuje verziju Qt Quick modula za korištenje u QML datotekama. Također, budući da je Qt namijenjen razvoju aplikacija za velik broj platformi, Qt Creator nudi opciju odabira skupa alata potrebnih za platforme na kojima će se graditi aplikacija. Na kraju, projekt je moguće dodati u Git sustav za verzioniranje koda.



Slika 2.9: Struktura Qt Quick projekta

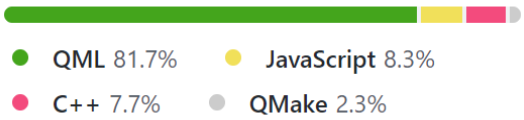
Opisanim postupkom Qt Creator je izgenerirao potrebne datoteke koje su naposljetku dostupne za izmjenu i uređivanje kroz Edit način rada. Sa slike 2.9 može se vidjeti da takav projekt sadrži projektnu datoteku s prepoznatljivim *.pro* nastavkom, datoteku izvornog koda *main.cpp*, XML datoteku *qml.qrc* s popisom resursa aplikacije, te QML datoteku *main.qml*.

S ovakvom minimalnom aplikacijom odabranog tipa napravljen je temelj za daljnji razvoj Qt Quick aplikacije.

Poglavlje 3

QTher aplikacija za vremensku prognozu

Kako bi se sve dosad opisano primijenilo u praksi, u sklopu ovog rada implementirana je aplikacija za prikaz vremenske prognoze. Aplikacija pod nazivom QTher korisniku će za odabrani grad, ovisno o izboru, prikazati dnevnu ili tjednu prognozu. Izvorni kod može se pronaći na [3]. U izradi je korištena najnovija Qt6 verzija biblioteke, te njena integrirana razvojna okolina Qt Creator u 5.0.1 inačici. Osnovni programski jezik aplikacije je QML, no u skladu sa znanjem o tom jeziku dobivenim iz prethodnih poglavlja, razumno je za očekivati integraciju QML-a s C++ i JavaScript jezicima. Da to zaista i jest tako pokazuje slika 3.1. Osim toga, sa slike se još može vidjeti da je za izgradnju korišten sustav `qmake`.



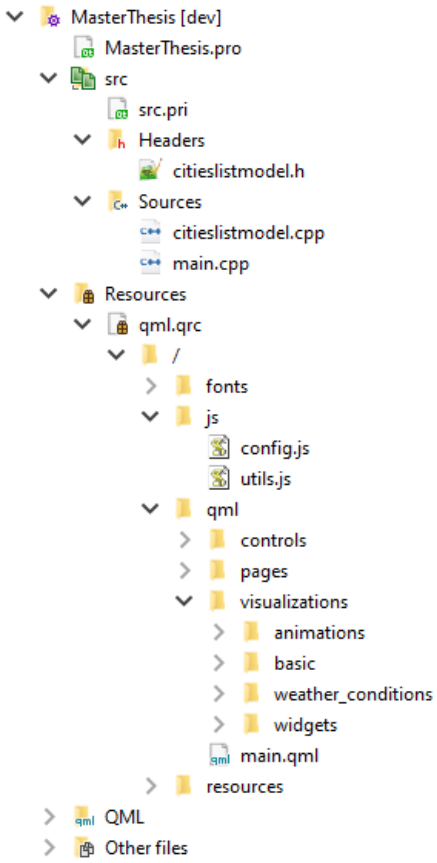
Slika 3.1: Programski jezici aplikacije QTher

U aplikaciji je fokus stavljen na grafičko korisničko sučelje. To podrazumijeva uvođenje Qt Quick biblioteke koja pruža većinu osnovnih elemenata i funkcionalnosti sučelja. Korištena verzija modula Qt Quick 6.0 usklađena je s verzijom Qt biblioteke. Pored toga, u projektnu datoteku je uključen Qt Charts modul koji daje mogućnost korištenja grafova kao QML tipova i time proširuje skup prethodno dostupnih elemenata. Ovakve postavke projekta, ispravno korištenje tipova QML elemenata te razvoj aplikacije u skladu s glavnim konceptima modernih korisničkih sučelja može rezultirati vrlo bogatim i vizualno atraktivnim sučeljem. U nadi da će QTher biti reprezentativni primjerak takve aplikacije, slijedi detaljan opis njene strukture, funkcionalnosti i implementacije.

3.1 Struktura projekta

Projektno okruženje aplikacije postavljeno je uz pomoć Qt Creatora na način opisan u odjeljku 2.4 drugog poglavlja. Posebnosti ovog Qt Quick projekta su – *Stack* vrsta aplikacije, *Git* sustav za verzioniranje koda, te već spomenuti `qmake` sustav za izgradnju.

Početna struktura je minimalno prilagođena dodavanjem `src` mape u kojoj će biti pohranjen C++ dio koda, dok se *qml.qrc* datoteka s resursima postepeno mijenjala u skladu s potrebama i logikom aplikacije. U njoj se nalazi sav QML kod kombiniran s JavaScript izrazima. Budući da je svaka ispravno imenovana *.qml* datoteka jedan QML tip, kreiran je velik broj takvih elemenata bitnih za samu aplikaciju. Osnovni JavaScript izrazi potrebni pojedinom tipu smješteni su unutar samog QML objekta, dok su veći dijelovi JavaScript koda, kao i oni osjetljiviji poput ključeva, izdvojeni u zasebne datoteke i pohranjeni u `js` mapu.



Slika 3.2: Struktura projekta `MasterThesis`

Ono što su oba direktorija zadržala jesu inicijalni *main.cpp* i *main.qml* dokumenti.

3.2 Dohvaćanje podataka

Naglasak pri izradi aplikacije stavljen je na prikaz podataka korisniku. Vremenski podaci koje aplikacija prikazuje dohvaćaju se putem API zahtjeva u JavaScript dijelu koda prikazanog isječkom 3.1.

Ajax (*Asynchronous JavaScript And XML*) upit uobičajan je način kreiranja zahtjeva putem HTTP protokola. Da bi se poziv izvršio potrebno je inicijalizirati novi objekt `XMLHttpRequest` i na njemu pozvati metodu `open()`, koja kao parametre redom uzima tip zahtjeva i lokaciju poslužitelja. Budući da se vremenski podaci žele primiti od poslužitelja, kao tip zahtjeva postavlja se GET.

```
1  var xhr = new XMLHttpRequest;
2  xhr.open("GET", "https://api.openweathermap.org/data/2.5/onecall?
3                      lat=" + latitude + "&lon=" + longitude +
4                      "&appid=" + Config.api_key + "&units=metric");
5  xhr.onreadystatechange = function() {
6      if (xhr.readyState === XMLHttpRequest.DONE) {
7          var obj = JSON.parse(xhr.responseText);
8          weatherData = obj;
9          setWeatherData(obj);
10     }
11 }
12 xhr.send();
```

Isječak koda 3.1: API zahtjev

API zahtjev šalje se *Open Weather Map* stranici koja nudi pristup vremenskim podacima za preko 200 000 gradova. Preciznije, poziv se upućuje *One Call* API-ju koji vraća podatke za zatraženi grad. Svaki grad reprezentiran je svojom geografskom širinom i dužinom koje se poslužitelju šalju preko parametara `lat` i `lon`. Osim ta dva parametra, nužno je poslati i `appid` kojeg je moguće dobiti registracijom na stranici [1]. Posljednji parametar `units` u prikazanom isječku koda 3.1 postavljen na `metric` nije obavezan, no ovdje označava zahtjev da su stupnjevi Celzija mjerna jedinica povratnih temperaturnih vrijednosti, što će kasnije biti važno za samu logiku aplikacije. Nakon specifikacije tipa zahtjeva, on se šalje poslužitelju pozivom metode `send()`.

Po primitku zahtjeva, poslužitelj vraća API korisniku odgovor. Odgovor se zapisuje u varijablu `obj` u pogodnom JSON formatu iz kojeg se kasnije čitaju podaci. Primjer odgovora u tom formatu na jedan API zahtjev dan je slikom 3.3.

```
{
    "lat": 42.6481,
    "lon": 18.0922,
    "timezone": "Europe/Zagreb",
    "timezone_offset": 3600,
  ▶ "current": { … }, // 14 items
  ▶ "minutely": [ … ], // 61 items
  ▶ "hourly": [ … ], // 48 items
  ▶ "daily": [ … ] // 8 items
}

"current": {
    "dt": 1643921782,
    "sunrise": 1643868004,
    "sunset": 1643904165,
    "temp": 6.8,
    "feels_like": 3.76,
    "pressure": 1022,
    "humidity": 39,
    "dew_point": -5.48,
    "uvi": 0,
    "clouds": 0,
    "visibility": 10000,
    "wind_speed": 4.63,
    "wind_deg": 30,
  ▼ "weather": [
      ▼ {
            "id": 800,
            "main": "Clear",
            "description": "clear sky",
            "icon": "01n"
        }
    ]
},

"hourly": [
  ▼ {
        "dt": 1643918400,
        "temp": 6.5,
        "feels_like": 3.61,
        "pressure": 1022,
        "humidity": 44,
        "dew_point": -4.31,
        "uvi": 0,
        "clouds": 11,
        "visibility": 10000,
        "wind_speed": 4.17,
        "wind_deg": 31,
        "wind_gust": 4.06,
      ▼ "weather": [
          ▼ {
                "id": 801,
                "main": "Clouds",
                "description": "few clouds",
                "icon": "02n"
            }
        ],
        "pop": 0
    },

"daily": [
  ▼ {
        "dt": 1643886000,
        "sunrise": 1643868004,
        "sunset": 1643904165,
        "moonrise": 1643873940,
        "moonset": 1643913840,
        "moon_phase": 0.08,
      ▼ "temp": {
            "day": 8.37,
            "min": 5.55,
            "max": 8.75,
            "night": 6.44,
            "eve": 6.39,
            "morn": 5.55
        },
      ▼ "feels_like": {
            "day": 4.66,
            "night": 4.3,
            "eve": 3.51,
            "morn": 1.46
        },
        "pressure": 1023,
        "humidity": 47,
        "dew_point": -3.26,
        "wind_speed": 8.45,
        "wind_deg": 350,
        "wind_gust": 13.76,
      ▼ "weather": [
          ▼ {
                "id": 800,
                "main": "Clear",
                "description": "clear sky",
                "icon": "01d"
            }
        ],
        "clouds": 1,
        "pop": 0.05,
        "uvi": 2.08
    },
```


Slika 3.3: Primjer rezultata API zahtjeva

Prvi isječak na slici minimizirani je prikaz ukupnih rezultata, dok je ostalim trima isječcima dan pregled dobivenih ključ-vrijednost parova za podatke koje je potrebno osigurati. Nazivi ključeva su vrlo intuitivni, međutim mjerne jedinice podataka nisu transparentne. Stoga tablica 3.1 prikazuje mjerne jedinice dobivenih vrijednosti koje će biti potrebne u daljnjem razvoju aplikacije.

Podatak	vrijeme	temperatura	tlak zraka	vlažnost	razina naoblake	vidljivost	brzina vjetra
Mjerna jedinica	unix	°C	hPa	%	%	m	m/s

Tablica 3.1: Mjerne jedinice rezultata API zahtjeva

Bitno je još reći nešto više o podatku `weather.id` koji se u primjeru 3.3 može naći među trenutnim, dnevnim ili podacima o vremenu po satu. Naime, vrijednost tog podatka predstavlja kod vremenskog uvjeta po kojem će se kasnije određivati animacija uvjeta. Svaki kod reprezentiran je troznamenkastim brojem. Prva znamenka broja predstavlja grupu vremenskog uvjeta. Postoji sedam takvih grupa: *Oluja (2xx)*, *Rominjanje (3xx)*, *Kiša (5xx)*, *Snijeg (6xx)*, *Atmosfera (7xx)*, *Vedro (800)* i *Oblačno (80x)*. Preostale dvije znamenke govore o jačini vremenskog uvjeta. Svi kodovi kao i detaljan opis njima pridruženih vremenskih uvjeta mogu se pronaći u [2]. Osim opisa, svakom kodu je pridružena i ikona, no ona je služila samo kao inspiracija za izradu animacija o kojima će više riječi biti kasnije.

3.3 Implementacija grafičkog sučelja

Sve Qt aplikacije su programi vođeni događajima. To zapravo znači da se cijeli program nalazi unutar jedne beskonačne petlje u kojoj se distribuiraju događaji generirani interakcijom korisnika sa sučeljem. Time se osigurava pravovremena reakcija na događaj tako da se događaji prosljeđuju elementima koji su za njega zainteresirani. Program završava zatvaranjem glavnog prozora.

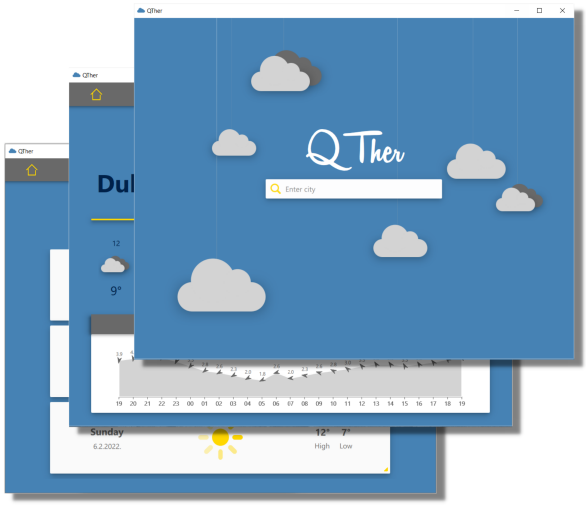
U skladu s tim, glavni dio same aplikacije nalazi se u *main.cpp* datoteci gdje se unutar funkcije `main()` poziva metoda `QApplication::exec()`. U njoj se nalazi cijela petlja događaja koja se izvodi sve do završetka programa odnosno zatvaranja prozora aplikacije. Primjer takve minimalne Qt aplikacije dan je isječkom koda 3.2. On je ujedno i bazični dio QTher aplikacije.

```
1 #include <QApplication>
2
3 int main(int argc, char *argv[])
4 {
5     QApplication app(argc, argv);
6     return app.exec();
7 }
```

Isječak koda 3.2: Izvorni kod minimalne Qt aplikacije

Osnovna struktura QTher aplikacije je stog. Naime, odabirom Stack Qt Quick predloška aplikacije pri izradi novog projekta zadana je osnovna struktura bazirana na QML tipu `StackView`. Taj tip je u ovom slučaju pogodan za korištenje jer osigurava razmjenu podataka između skupa međusobno povezanih stranica.

Dakle, u datoteku *main.qml* smješten je glavni prozor aplikacije `ApplicationWindow`, čija je osnovna komponenta `StackView`. Prvi element kojeg ovaj pogled prikazuje je onaj dodijeljen `initialItem` svojstvu objekta ili, ukoliko ono nije postavljeno, element s vrha stoga.



Slika 3.4: *Stack* struktura aplikacije

Navigiranje aplikacijom podržano je u tri primarne operacije, `push()`, `replace()` i `pop()`, koje odgovaraju klasičnim operacijama stoga. U kreiranoj aplikaciji, elementi se na stogu izmjenjuju odabirom željenog grada za prikaz prognoze odnosno odabirom jedne od ponuđenih mogućnosti za prikaz na alatnoj traci. To su početna stranica, stranica za prikaz dnevne vremenske prognoze i stranica za prikaz sedmodnevne vremenske prognoze. Naravno, sama logika izmjene stranica koje se prikazuju ostvarena je prethodno spomenutim metodama za navigaciju.

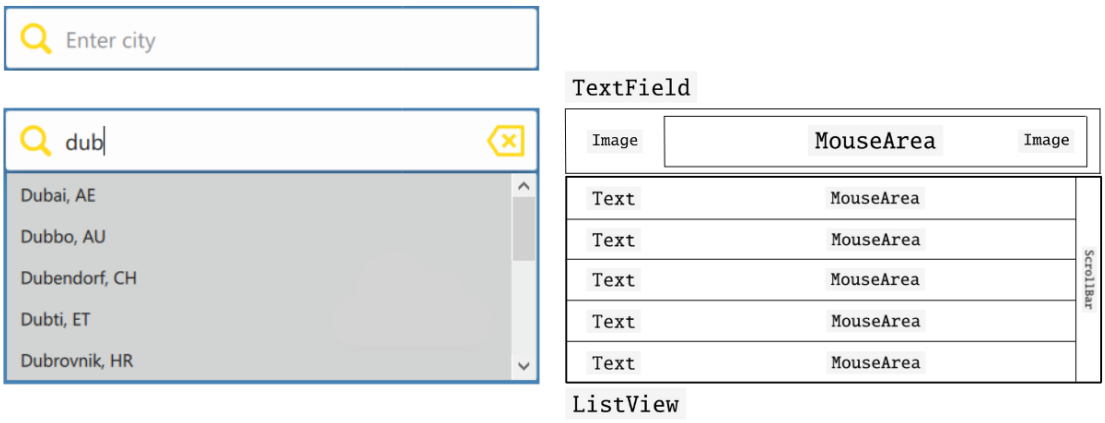
Izgledom, stog je obično usidren uz rubove prozora, osim eventualno na vrhu ili dnu gdje može biti usidren na statusnu traku odnosno neku drugu sličnu komponentu korisničkog sučelja. Tako je u slučaju QTher aplikacije stog na vrhu usidren na alatnu traku.

U nastavku ovog poglavlja bit će dan detaljan opis svakog od tri QML elementa koji se izmjenjuju na stogu: `HomePage`, `CurrentWeatherPage` i `SevenDaysWeatherPage`. Pobrojani elementi se unutar strukture projekta nalaze se u mapi `pages` (slika 3.4). Ipak, prije toga je dan opis bitnih komponenti odnosno objektnih QML tipova QTher aplikacije koji se koriste na gotovo svakoj stranici ili jako često puta. Za sve komponente, kao i za stranice, prvo se opisuje funkcionalnost elementa, nakon čega slijedi opis procesa implementacije.

3.3.1 Padajući izbornik

Prvi izdvojeni objektni QML tip unutar aplikacije je padajući izbornik za odabir grada. On se može okarakterizirati kao temeljni element cijele aplikacije. Naime, to je početni element koji se korisniku prikazuje da bi uopće krenuo na iole smisleniji prikaz vremenskih podataka i dobio uvid u aplikaciju.

Kao što je prikazano na slici 3.5 objekt `CitiesSuggestionBox` u sebi integrira više QML tipova. Neki se tiču dizajna, a neki funkcionalnosti samog elementa. Svrha ovog elementa je, na temelju korisnikovog unosa, unutar padajućeg izbornika ponuditi gradove dostupne za prikaz vremenske prognoze.



Slika 3.5: Padajući izbornik za odabir grada

U pozadini zapravo stoji implementacija klase u C++ dijelu koda koja služi kao model podataka za neki pogled. O integraciji QML i C++ jezika u teoriji više se spominjalo u odjeljku 2.3.4, dok je o *model-view* arhitekturalnom obrascu pisano u odjeljku 1.5. Ovdje će se, na primjeru C++ klase `CitiesListModel`, detaljno proći kroz sve potrebne korake implementacije da bi C++ klasa postala model dostupan QML-u.

Implementacija C++ modela

C++ klasa koja implementira model može nasljeđivati neke od već ponuđenih Qt klasa (vidi [13]). Prije početka implementacije bitno je ispravno odabrati osnovnu klasu koja će se proširivati. Budući da će padajući izbornik prikazivati popis gradova poredanih u listu, kao logičan odabir nameće se klasa `QAbstractListModel`. Ona pruža standardno sučelje za modele koji svoje podatke reprezentiraju jednostavnim nehijerarhijskim slijedom elemenata. Dio *citieslistmodel.h* datoteke zaglavlja u kojoj se deklarira klasa `CitiesListModel` prikazan je isječkom koda 3.3.

```
1  #ifndef CITIESLISTMODEL_H
2  #define CITIESLISTMODEL_H
3  #include <QAbstractListModel>
4  #include <QtQml>
5
6  typedef QPair<QString, QString> CityCountryPair;
7  typedef QPair<double, double> Coordinates;
8
9  class CitiesListModel : public QAbstractListModel
10 {
11      Q_OBJECT
12 public:
13     enum Roles {
14         CityRole = Qt::DisplayRole + 1,
15         CountryRole = Qt::DisplayRole + 2,
16         LongitudeRole = Qt::DisplayRole + 3,
17         LatitudeRole = Qt::DisplayRole + 4
18     };
19     CitiesListModel(QObject* parent = 0);
20     int rowCount(const QModelIndex& parent = QModelIndex()) const override;
21     QVariant data(const QModelIndex& index,
22                   int role = Qt::DisplayRole) const override;
23     QHash<int, QByteArray> roleNames() const override;
24
25 protected:
26     void readCities();
27
28 private:
29     bool isIndexValid(const QModelIndex& index) const;
30     QString mCitiesFileName;
31     QList<CityCountryPair> mCities;
32     QList<Coordinates> mCoordinates;
33 };
```

Isječak koda 3.3: Definicija klase modela `CitiesListModel`

Na početku je bitno istaknuti korištenje makroa `Q_OBJECT` u privatnom dijelu definicije klase. On osigurava mogućnost korištenja *meta-object* sustava Qt biblioteke, s posebnim naglaskom na sustav signala i utora.

Usto, privatni dio klase sadrži još deklaraciju triju varijabli članica (linije 30-32). Njihova inicijalizacija odvija se u konstruktoru. Naime, varijabli `mCitiesFileName` tipa `QString` pridružuje se vrijednost imena datoteke *citiesList.json* iz koje se čitaju potrebni podaci o svakom gradu dostupnom za prikaz prognoze. Ta datoteka, JSON tipa, dostupna je kao resurs aplikacije što znači da je navedena u *.qrc* datoteci projekta. Za čitanje podataka iz

datoteke klasa koristi funkciju članicu `readCities()` koja prolazeći kroz datoteku varijablama `mCities` i `mCoordinates` simultano dodaje elemente definiranog tipa.
Pročitani podatak može se sastojati od više informacija. QML pogled koji dohvaća podatke iz modela mora moći razlikovati svaku dobivenu informaciju odnosno svaki dio tog podatka. Iz tog razloga, Qt omogućuje različite vrste imenovanih uloga (vidi [8]). Budući da se podaci ovog modela koriste za prikaz, u fiksnom skupu izbora imena `enum Roles` korištena je uloga `DisplayRole`. Pomoću nje se definiraju četiri uloge – `CityRole`, `CountryRole`, `LongitudeRole` i `LatitudeRole`.
Uz varijable, klasa modela mora implementirati minimalan broj potrebnih funkcija naslijeđene klase. Konkretno, u slučaju nasljeđivanja klase `QAbstractListModel`, nužno je implementirati funkcije `rowCount()` i `data()`. Prva funkcija koristi se za dohvat broja elemenata u listi, dok je funkcija `data()` odgovorna za dohvat konkretnog podatka na traženom indeksu i s određenom ulogom. Također, budući da će model biti korišten unutar QML-a, zahtjeva se predefiniranje funkcije `roleNames()`. Ta funkcija, informacijama o ulogama pojedinih podataka, QML dijelu koda osigurava mogućnost pristupa podacima.
Ostale funkcije specifične su za klasu koja se definira. Funkcija `readCities()` već je prethodno opisana, dok funkcija `isValidIndex()` služi za provjeru postojanja elementa na indeksu s kojeg se on želi dohvatiti. Sve navedene funkcije koriste se za čitanje modela. Budući da je gradove potrebno samo prikazati i nije dozvoljeno njihovo uređivanje, nema potrebe za implementacijom funkcija koje bi omogućavale izmjenu modela.

Poslije prethodno opisanog, uspješno je razvijen model svih dostupnih gradova kojeg, nakon povezivanja, QML pogled može bez problema prikazati. Međutim, iz samog opisa funkcionalnosti komponente `CitiesSuggestionBox` moglo se vidjeti da taj element treba reagirati na unos korisnika. To znači da je, ovisno o unesenom tekstu, potrebno filtrirati trenutno kreirani model kako bi se prikazali samo oni podaci koji počinju zadanim unosom. U slučaju dobivenog modela, za filtriranje, ali i sortiranje elemenata koristi se klasa `CitiesProxyModel`. Njena definicija dana je isječkom koda 3.4.

```
1  class CitiesProxyModel : public QSortFilterProxyModel
2  {
3      Q_OBJECT
4  public:
5      CitiesProxyModel(QObject* parent = 0);
6      Q_INVOKABLE void setFilterString(QString input);
7      Q_INVOKABLE QString getCityName(const QModelIndex &index);
8      Q_INVOKABLE double getCityLongitude(const QModelIndex &index);
9      Q_INVOKABLE double getCityLatitude(const QModelIndex &index);
10 };
```

Isječak koda 3.4: Definicija klase za sortiranje i filtriranje modela

U definiciji klase makro `Q_OBJECT` nalazi se iz istog razloga kao i kod prethodno opisane klase `CitiesListModel`. Stvorena klasa je potklasa klase `QSortFilterProxyModel` koja omogućuje da se dani izvorni model restrukturira za pogled, bez potrebe za bilo kakvom preobrazbom temeljnih podataka ili dupliciranjem podataka u memoriji. Qt-ova klasa transformira strukturu izvornog modela preslikavanjem originalnog indeksa u novi indeks unutar filtriranih odnosno sortiranih rezultata.

Razvijeni model `CitiesProxyModel` u konstruktoru na modelu poziva funkciju `sort()`. Time se mijenja redoslijed podataka modela na način da se poštuje uzlazni slijed imena država. Nadalje, kao javne funkcije članice klasa definira funkcije opisane u nastavku.

- `setFilterString()`

  Funkcija kao ulaz prima string `input` i na temelju njega filtrira podatke po ulozi `CityRole`. Usto, filter se postavlja kao neosjetljiv na velika i mala slova.

- `getCityName()`

  Funkcija na temelju zadanog indeksa dohvaća naziv grada koji se u sortiranom i filtriranom modelu na njemu nalazi.

- `getCityLongitude()`

  Funkcija na temelju zadanog indeksa dohvaća geografsku dužinu grada koji se u sortiranom i filtriranom modelu na njemu nalazi.

- `getCityLatitude()`

  Funkcija na temelju zadanog indeksa dohvaća geografsku širinu grada koji se sortiranom i filtriranom modelu na njemu nalazi.

Bitno je primijetiti kako sve navedene funkcije u deklaraciji imaju makro `Q_INVOKABLE`. Naravno, takvom definicijom omogućen je poziv funkcija iz QML dijela što će biti od presudne važnosti za primjenu funkcionalnosti koje ova klasa nudi.

Na kraju je preostalo model postaviti vidljivim QML-u. Jedan mogući način je instancirati model u *main.cpp* datoteci te ga uz pomoć korijenskog objekta `QQmlContext` prikazati QML-u. Funkcija `setContextProperty()` omogućuje vezivanje svake C++ klase koja nasljeđuje `QObject` na QML svojstvo. Realizacija povezivanja `CitiesProxyModel` modela s QML-om dana je isječkom koda 3.5 koji se nalazi unutar funkcije `main()` u datoteci *main.cpp*.

```
QQmlApplicationEngine engine;
QQmlContext* context = engine.rootContext();

CitiesListModel citiesListModel;
CitiesProxyModel citiesFilterModel;

citiesFilterModel.setSourceModel(&citiesListModel);
citiesFilterModel.setFilterRole(CitiesListModel::Roles::CityRole);
citiesFilterModel.setSortRole(CitiesListModel::Roles::CountryRole);
context->setContextProperty("filterModel", &citiesFilterModel);
```

Isječak koda 3.5: Postavljanje vidljivosti modela unutar QML-a

Implementacija QML pogleda

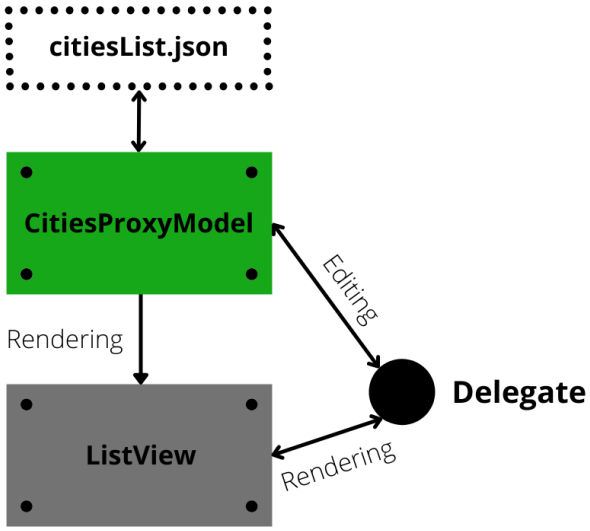
Prethodni odjeljak dao je detaljan opis svih koraka implementacije C++ modela kao i način postavljanja njegove vidljivosti unutar QML-a. Nakon svega, taj model napokon može biti model nekog QML pogleda. Slijedi opis tipa `CitiesSuggestionBox` iz QML perspektive te elemenata od kojih je on sastavljen. Poseban naglasak stavljen je na elemente koji služe za prikaz kreiranog C++ modela.

Logički gledano, padajući izbornik se može podijeliti na dva dijela: područje za unos teksta i područje za prikaz ponuđenih gradova. Prvo nabrojeno područje realizirano je u QML kontroli `TextField`. Kontrola se sastoji od tri dijela (slika 3.5), dekorativne ikone povećala, nevidljivog područja osjetljivog na klik miša `MouseArea`, te *backspace* ikone koja se pojavljuje jedino u slučaju da tekstualno polje nije prazno i služi za brisanje sadržaja polja.

Funkcionalnost svih elemenata ovog područja je vrlo jednostavna. Ono najbitnije jest promjena stanja elementa `CitiesSuggestionBox` ovisno o tekstu unesenom u polju za unos. Na svaku promjenu teksta, upravljač signala automatski poziva JavaScript funkciju za postavljanje stanja elementa. Postoje dva stanja koja se izmjenjuju, ono u kojem se prikazuje padajući element s ponuđenim gradovima i inicijalno u kojem je taj element skriven.
Ako je zadovoljen uvjet da u modelu postoji barem jedan grad čiji naziv počinje s dosad unesenim tekstom, tada je `CitiesSuggestionBox` u stanju koje prikazuje padajući element. Provjera postojanja takvog grada realizirana je preko, sad dostupnog, modela `filterModel` definiranog u C++ dijelu koda (vidi isječak koda 3.4). Za prikaz svih gradova koji zadovoljavaju dani uvjet koristi se QML tip `ListView`. Svojstvo `model` tog pogleda pruža skup podataka koji se u njemu prikazuju. Vrijednost pridružena tom svojstvu je upravo `filterModel` pomoću kojeg pogled zna koje podatke treba prikazati. Uređivanje

i prilagođavanje prikaza podataka odvija se unutar delegata koji je povezan s modelom i s pogledom. Svakoj ulozi u funkciji `roleNames()` (vidi isječak koda 3.3) pridružena je vrijednost koju delegat sad može koristiti u svrhu dolaska do pojedinačne informacije promatranog podatka.

Svaki grad u listi nalazi se unutar vlastitog pravokutnika kojeg ispunjava nevidljiva površina `MouseArea`. Klik na tu površinu uvijek vodi na stranicu za prikaz dnevne vremenske prognoze traženog grada. Bitnim svojstvima te stranice, potrebnim za dohvat podataka API zahtjevom, postavlja se vrijednost dobivena pomoću funkcija za dohvat iz modela `CitiesProxyModel`. Funkcijama se kao parametar šalje indeks kliknutog elementa u filtriranoj i sortiranoj listi.



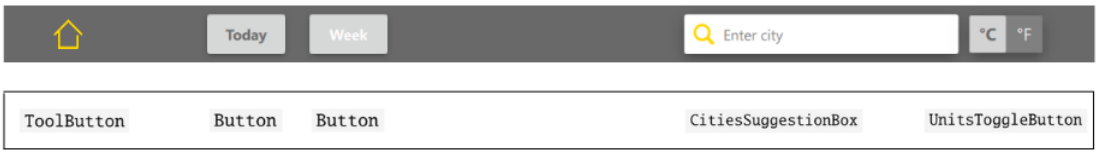
Slika 3.6: *Model-view* obrazac za `CitiesSuggestionBox`

Time je stvoren zasebni, potpuno funkcionalni element `CitiesSuggestionBox`. Osim opisa jedne od najosnovnijih komponenti aplikacije, ovaj odjeljak demonstrira implementaciju novog C++ modela podataka, način na koji se podaci tog modela mogu filtrirati te sam *model-view* obrazac za programiranje grafičkog sučelja u Qt-u. Sve navedeno ne bi bilo moguće ostvariti bez integracije QML, C++ i JavaScript jezika.

3.3.2 Alatna traka

Alatna traka glavni je navigacijski element QTher aplikacije. Smještena je u zaglavlju glavnog prozora aplikacije, a prikazuje se na svim stranicama osim početne. Njen prikaz

regulira se postavljanjem odgovarajuće vrijednosti svojstvu vidljivosti.



Slika 3.7: Objektni QML tip `PageToolBar`

Alatna traka je implementirana objektnim QML tipom `PageToolBar` koji sadrži razne gumbe kao i tip `CitiesSuggestionBox` za kretanje po aplikaciji. Neke od tih kontrola podržane su modulom Qt Quick Controls te su korištene uz manje prilagodbe. S druge strane, dio tipova implementiran je korištenjem osnovnih Qt Quick elemenata kombiniranih kako bi zadovoljili neku složeniju funkcionalnost. Kako alatna traka izgleda u aplikaciji i koji QML objektni tipovi stoje u pozadini prikazuje slika 3.7. Komponenta `CitiesSuggestionBox` neće se ponovno opisivati, no bitno je primijetiti kako uz pomoć alatne trake korisnik u svakom trenutku može odabrati novi grad za koji želi vidjeti podatke o vremenu.

Gumbi za navigaciju aplikacijom

Prva tri elementa izborne trake mogu se promatrati kao navigacijski gumbi. Jednako kao i kod običnih gumba, upravljač signala klik će osnovnim operacijama puniti odnosno prazniti stog.

```
1  import QtQuick.Controls 6.0
2
3  ToolButton {
4      id: homeButton
5
6      Image{
7          height: homeButton.height / 2
8          width: homeButton.width / 2
9          anchors.centerIn: homeButton
10         source: "qrc:/resources/icons/home.png"
11     }
12     onClicked: pageStack.pop(null)
13 }
```

Isječak koda 3.6: Implementacija gumba *Home*

Gumb s prigodnom *home* ikonom korisnika vodi na početnu stranicu aplikacije. Ta funkcionalnost implementirana je isječkom koda 3.6. Naime, dio JavaScript koda prikazan linijom 12 postavlja stog na inicijalnu vrijednost pomoću upravljača signala klik.

Sljedeća dva gumba, gotovo jednakog izgleda, označavaju stranicu koja se trenutno prikazuje te stranicu koju je moguće odabrati za prikaz. Prva je označena sivim, a druga bijelim slovima teksta kojeg gumb sadrži. Klik na gumb *Today* vodi na stranicu za pregled dnevne vremenske prognoze na način da QTher stavlja stranicu `CurrentWeatherPage` na vrh stoga ukoliko to već nije slučaj, dok gumb *Week* stogu dodaje `SevenDaysWeatherPage` stranicu. To je, jednako kako i kod gumba *Home*, u kodu ostvareno unutar upravljača signala i to konkretno metodom `push()`.

Gumb za promjenu temperaturne mjerne jedinice

Posljednje mjesto na alatnoj traci rezervirano je za gumb pomoću kojeg se mijenja trenutno aktivna temperaturna mjerna jedinica. Iako Qt Quick modul nudi kontrolu `ToggleButton` za ovakvu vrstu gumba, zbog potrebne velike stilske prilagodbe implementiran je novi objektni QML tip `UnitsToggleButton`.

Nova kontrola sastavljena je od dvaju pravokutnika. Donji, veći pravokutnik označava temperaturnu jedinicu koja nije odabrana, dok gornji, upola uži svjetliji pravokutnik, ističe odabranu mjernu jedinicu. Temperaturne mjerne jedinice koje je moguće odabrati su stupnjevi Celzija, odnosno stupnjevi Fahrenheita. Klikom na gumb mijenja se trenutno aktivna mjerna jedinica u onu suprotnu.

Opisana funkcionalnost u QML kodu je ostvarena definiranjem novog signala naziva `toggled()`. Signal kao parametar prima string koji označava novoizabranu mjernu jedinicu. Upravljač signala `onToggled`, na mjestu definicije gumba unutar `PageToolBar` objekta, sad reagira na promjenu na prigodan način. Budući da je potrebno svim podacima vezanim za temperaturu na trenutnoj stranici ažurirati vrijednosti, alatna traka deklarira svoj signal `unitsButtonToggled()` s istim string parametrom, kako bi u ovom slučaju stranica mogla registrirati promjenu jedinice. Na ovaj način je signal promjene došao od gumba `UnitsToggleButton` preko alatne trake `PageToolBar` sve do stranice koja sadrži podatke. Na kraju, upravljač registriranog signala na stranici ažurira sve temperaturne vrijednosti pozivajući JavaScript funkciju za preračunavanje jedinica. Sve opisano prikazano je isječkom koda 3.7. Iz izvornog koda izdvojeni su samo oni dijelovi odgovorni za funkcionalnost promjene temperaturne jedinice.

```
// controls\UnitsToggleButton.qml
Pane {
    id: unitsToggleButton
    property bool celsius: true
    signal toggled(string units)

    MouseArea {
        onClicked: {
            celsius = !celsius
            unitsToggleButton.toggled(celsius ? "celsius" : "fahrenheit")
        }
    }
}
// controls\PageToolBar.qml
ToolBar {
    id: toolBar
    signal unitsButtonToggled(string units)

    UnitsToggleButton {
        onToggled: function (units) {
            toolBar.unitsButtonToggled(units)
        }
    }
}
// main.qml
ApplicationWindow {
    readonly property alias pageStack: stackView

    header: PageToolBar {
        onUnitsButtonToggled: function(units) {
            pageStack.currentItem.unitsButtonToggled(units)
        }
    }
}
```

Isječak koda 3.7: Upravljanje signalom `toggled()` gumba `UnitsToggleButton`

3.3.3 Vizualizacije vremenskih uvjeta

Kao što je već opisano, podatke o vremenu aplikacija dohvaća putem API zahtjeva. Zahtjevi upućeni stranici s koje se podaci dohvaćaju odgovorit će, između ostalog, s podacima o vremenskim uvjetima trenutno, po satu unutar sljedećih 24 sata i po danu za idućih 7 dana. Ono što je za implementaciju vizualnih elemenata vremenskih prilika bitno jesu

vrijednosti `weather.id` i `weather.icon` ključeva u API odgovoru sa slike 3.3. Vrijednost prvog podatka izražena je brojem na način opisan u odjeljku 3.2, dok je druga vrijednost reprezentirana stringom od tri znaka. Ovdje će biti važan samo zadnji znak tog stringa, budući da on govori o dobu dana koje će vizualizacija prikazivati.

Dobivene podatke QTher interpretira te za svaki, ovisno o dobu dana, tipu i intenzitetu vremenskog uvjeta, definira vlastiti statični vizualni objekt ili animaciju. Ta interpretacija odvija se u JavaScript dijelu koda unutar funkcije `setWeatherAnimation()`.

```
1  function setWeatherAnimation(weatherAnimationLoader, weather_code,
2                               weather_icon, item_width, item_height){
3     var weatherCondition_0 = parseInt(weather_code.charAt(0))
4     var weatherCondition_1 = parseInt(weather_code.charAt(1))
5     var weatherCondition_2 = parseInt(weather_code.charAt(2))
6     var timeOfDay = weather_icon.charAt(2)
7
8     if (weatherCondition_0 === 2)       { ... }
9     else if (weatherCondition_0 === 3)  { ... }
10    else if (weatherCondition_0 === 5)  {
11      weatherAnimationLoader.setSource(
12      "qrc:/qml/visualizations/weather_conditions/Rain.qml" ,
13      {   timeOfDay: weatherCondition_1 === 0 ? timeOfDay : "",
14          isFreezingRain: weatherCondition_1 === 1,
15          rainIntensity: setRainIntensity(weatherCondition_1,
16                                          weatherCondition_2),
17          dropOfRainSize: weatherCondition_1 === 0 ? "small" : "big",
18          width: item_width,
19          height: item_height
20      })
21    }
22    else if (weatherCondition_0 === 6) { ... }
23    else if (weatherCondition_0 === 7) { ... }
24    else if (weatherCondition_0 === 8) { ... }
25 }
```

Isječak koda 3.8: Dio funkcije `setWeatherAnimation()`

Dio te funkcije prikazan je isječkom koda 3.8. Odatle se vidi da funkcija, ovisno o prvoj znamenci koda vremenskog uvjeta (parametar `weather_code`), postavlja objekt koji vizualizira grupu vremenskih uvjeta u objekt `weatherAnimationLoader`. Slijedi definicija metode koja je zadužena za ovu funkcionalnost.

object setSource(url *source*, object *properties*)

Naime, metoda `setSource()` kreira instancu objekta `source` koji će imati dana svojstva `properties`. Poziva na objektu `Loader` u kojeg se novokreirani objekt učitava. U primjeru grupe čiji kod počinje znamenkom 5 (*Kiša*), nakon predanog tipa koji vizualizira kišne vremenske uvjete (datoteka `Rain.qml`), funkciji `setSource()` šalju se svojstva karakteristična za taj objekt. Jednakom logikom kreiraju se instance objekata za bilo koju grupu vremenskih uvjeta.

Dakle, funkcija na temelju dobivenih `weather.id` i `weather.icon` podataka oblikuje animaciju. Animacija se sastoji od izrađenih QML tipova za osnovne vremenske prilike kojima se postavljaju prigodna svojstva kako bi se dobio željeni efekt. Slijedi pregled implementiranih vizualizacija po grupama vremenskih uvjeta, a kao uvod daje se opis kreiranih bazičnih QML tipova.

Objekt `Fog` predstavlja maglu i koristi se u grupama od po 3 ili 4 objekta. Može se prilagoditi visinom, širinom i trajanjem animacije iscrtavanja objekta na ekranu.

Objekt `Cloud` prikazuje statični oblak. Sastavni je element gotovo svake grupe vremenskih uvjeta. Može se prilagoditi bojom i veličinom.

Objekt `Moon` prikazuje mjesec. Objekt je statičan i može se prilagoditi veličinom.

Objekt `Sun` prikazuje animirano sunce u kojem okolne zrake trepere ukrug. Može se prilagoditi veličinom.

Objekt `Precipitation` je animacija padalina. Moguće je prilagoditi vrstu i intenzitet padalina, te veličinu čestica koje padaju.

Grupa *Vedro*

Neki od osnovnih elemenata samostalno već tvore gotov prikaz ili animaciju. Tu se ubrajaju elementi sunca i mjeseca koji zajedno čine grupu *Vedro*.

Slika 3.8: Elementi za prikaz vremenskih uvjeta grupe *Vedro*

Novostvoreni QML tip `Sun` koji predstavlja animirano sunce u svom stablu QML objekata definira:

- Element `Rectangle` dovoljno zaobljenih rubova tako da tvori krug. On predstavlja statični krug koji se nalazi u sredini samog elementa.
- QML tip `Repeater` koji se koristi za kreiranje velikog broja sličnih komponenti. Taj tip, slično kao i ostali vizualni elementi, ima svojstva `model` i `delegate` koja se očituju tako da za svaki podatak iz modela, delegat instancira kontekst u ga koji smješta. U ovom slučaju model je broj. On označava broj delegata koji će biti stvoreni. Preciznije, svojstvu `model` pridružen je broj 8 koji reprezentira 8 pravokutnika zaobljenih rubova poredanih ukrug oko središnjeg kruga.
- QML tip `Timer` koji se okida u zadanom intervalu i omogućuje treptanje svake zrake reguliranjem svojstva `opacity`.

Vizualizacija mjeseca realizirana je u tipu `Moon`. Objekt nije ništa drugo nego obojeni panel zaobljenih rubova. U ovom slučaju nije korišten obični pravokutnik kako bi se element mogao prilagoditi izgledom. Upravo iz tog razloga na njega se može primijeniti efekt uzvišenosti ostvaren u svojstvu `elevation` iz modula Qt Quick Controls Material.

Na ovaj način dobiveni su vizualni elementi koji predstavljaju sunce i mjesec. U grupi *Vedro* oni se koriste zasebno. Međutim, elementi se mogu kombinirati s drugim vizualnim tipovima zahvaljujući mogućnosti prilagodbe njihove veličine i pozicije. Tako oni mogu biti i dio nekog elementa koji ne pripada ovoj grupi. No, više riječi o tome bit će u nastavku.

Grupa *Oblačno*

Grupa *Oblačno* sastoji se od četiri različita vizualna elementa prikazana na slici 3.9. Oni se pridružuju vremenskom uvjetu ovisno o dobu dana i količini naoblake.

Slika 3.9: Elementi za prikaz vremenskih uvjeta grupe *Oblačno*

Implementacija vizualnih i animiranih dijelova realizirana je objektnim tipom `Clouds`. Ono što je zajedničko svim elementima ove grupe jest središnji svijetlo sivi oblak. Ostale komponente se postavljaju ovisno o potrebama uvjeta.

```
1  Item {
2      id: cloudsGroupItem
3      property bool brokenClouds
4      property string timeOfDay
5
6      Loader {
7          id: timeOfDayLoader
8      }
9
10     Component.onCompleted: {
11         if (timeOfDay == 'd')
12             timeOfDayLoader.setSource( "../basic/Sun.qml",
13                                        { radius: mainCloud.width * 0.2,
14                                          x: mainCloud.width * 0.7 })
15         else if (timeOfDay == 'n')
16             timeOfDayLoader.setSource( "../basic/Moon.qml",
17                                        { width: mainCloud.width * 0.6,
18                                          x: mainCloud.width * 0.55,
19                                          y: -mainCloud.width * 0.05 })
20     }
21 }
```

Isječak koda 3.9: Dio QML datoteke `Clouds.qml`

U te svrhe tip `Clouds` u sebi još definira:

- QML tip `Loader` za dinamičko učitavanje QML datoteke u kojoj se nalazi implementacija proizvoljnog objektnog tipa.
- Novi objekt `Cloud` za prikaz stražnjeg, tamnosivog oblaka.

Prvi nabrojeni element detaljnije se vidi u isječku koda 3.9 odnosno dijelu `Clouds.qml` datoteke koja implementira vizualne elemente grupe *Oblačno*. Preciznije, predočen je dio

koji podržava učitavanje datoteke u kojoj je definiran tip koji predstavlja sunce ili mjesec. Unutar upravitelja signala `Component.onCompleted`, ovisno o vrijednosti svojstva koje označava doba dana `timeOfDay`, poziva se već opisana metoda `setSource()` na objektu tipa `Loader`. Taj dio će se izvršiti odmah nakon instanciranja objekta `Clouds`, a kao rezultat QML tip identificiran kao `timeOfDayLoader` prikazat će zadani tip. Bitno je još napomenuti da isječak koda 3.9 neće funkcionirati bez uvezenog Qt Quick modula, te relativnog puta do direktorija koji sadrži prethodno definirane QML tipove koji se učitavaju.

Svim objektima, ovisno o dimenzijama roditeljskog tipa `Clouds`, prilagođava se veličina i pozicija. Za to su odgovorne JavaScript funkcije koje pozivaju elementi kojima je potrebna prilagodba. U slučaju elementa `Loader`, svojstva vezana za dimenziju i poziciju objekta koji se učitava, šalju se tom objektu prilikom stvaranja.

Grupa *Atmosfera*

Grupa vremenskih uvjeta, okarakterizirana nazivom *Atmosfera*, prikazuje atmosferske uvjete, od najblaže maglice pa sve do intenzitetom najjačeg tornada. Prikaz uvjeta je animiran.

Slika 3.10: Elementi za prikaz vremenskih uvjeta grupe *Atmosfera*

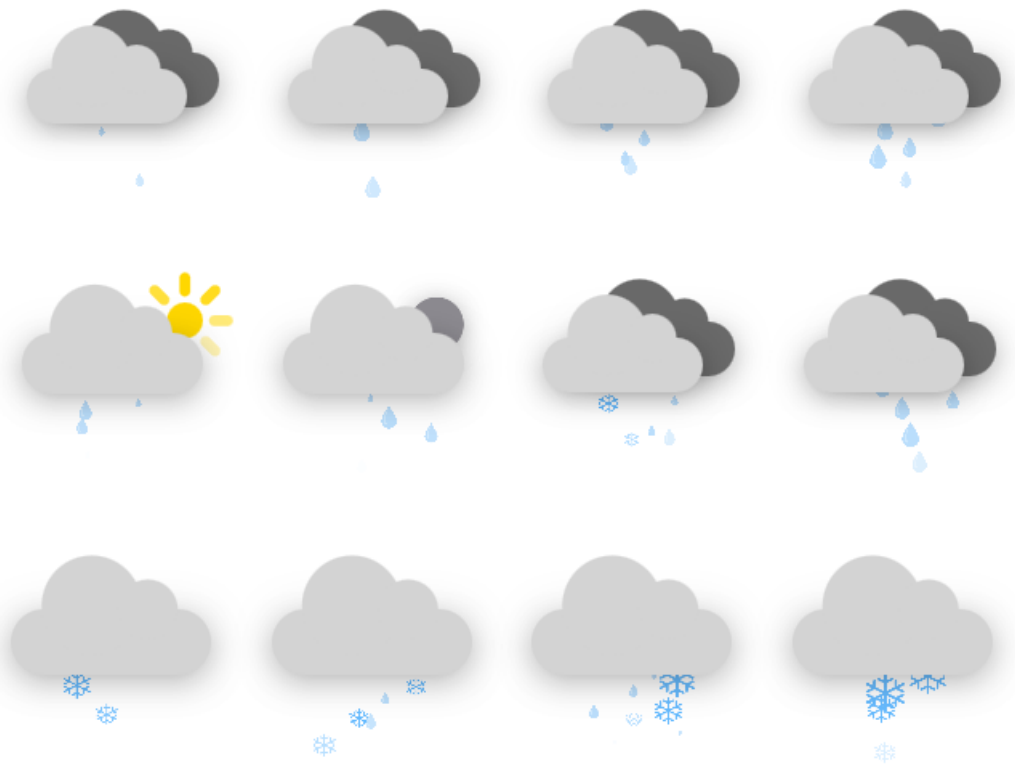
```
Item {
    id: fogItem
    property int lineLength
    property int lineHeight
    property int animationDuration

    Rectangle {
        height: lineHeight
        radius: 10
        color: "dimgray"
        NumberAnimation on width {
            from: 0; to: lineLength; duration: animationDuration;
        }
    }
}
```

Isječak koda 3.10: QML datoteka `Fog.qml`

QML objekt `Atmosphere` deklarira 3 ili 4 linije koje simboliziraju maglu. One su implementirane unutar datoteke `Fog.qml` koja predstavlja osnovni gradivni element atmosferskih uvjeta. Moguće je postaviti duljinu i visinu linije, te trajanje iscrtavanja linije na ekran zadavanjem svojstava `lineLength`, `lineHeight` odnosno `animationDuration` redom. Time vizualni prikaz elementa, definiran kao u isječku koda 3.10, postaje animiran. Usto, objekt još deklarira i QML tip koji prikazuje oblak, a vidljiv je ovisno o potrebama i jačini uvjeta.

Grupe padalina

U grupe padalina ubrajaju se *Rominjanje*, *Kiša* i *Snijeg*. Recima slike 3.11 dani su elementi pojedine grupe. Ono što je svim elementima ovih grupa zajedničko jesu komponente koje prikazuju oblak i padaline.



Slika 3.11: Vizualni elementi grupa padalina po recima

Zato su svi vizualni prikazi iz ovih grupa vremenskih uvjeta implementirani približno jednakom logikom. Naime, tipovi `Clouds` i `Preticipation` neizostavan su dio svakog QML tipa definiranog za prikaz ovih vizualnih elemenata. Pored toga, ukoliko je neki vremenski uvjet potrebno prikazati s dvije vrste padalina, u definiciju objekta uključen je tip `Loader` za njihovo učitavanje. Tako je primjerice, u slučaju susnježice, tip osnovne padaline snijeg definiran u glavnom objektu `Preticipation`, dok QML tip `Loader` učitava sporednu padalinu – kišu.

```
import QtQuick 6.0
import QtQuick.Particles 2.0

Item {
    id: precipitationItem
    property string type: "" // "rain" ili "snow"
    property int intensity: 2
    property int particlesSize: width * 0.08
    property int cloudBottom

    ParticleSystem {
        id: particles
    }
    ImageParticle {
        system: particles
        source: "qrc:/resources/icons/" + type + ".png"
    }
    Emitter {
        id: emmiter
        system: particles
        y: cloudBottom
        emitRate: intensity
        lifeSpan: 900
        velocity: PointDirection {
            y: precipitationItem.height
            yVariation: 10
        }
        size: particlesSize
        endSize: {
            if (type === "rain") return -1
            else if (type === "snow") return particlesSize * 0.5
        }
        sizeVariation: 5
    }
}
```

Isječak koda 3.11: Prerađeni QML tip `Preticipation`

Za svaku vrstu padalina podržana je promjena veličine čestica koje padaju, jednako kao i prilagodba brzine padanja čestica. Sam prikaz padalina omogućen je korištenjem modula Qt Quick Particles unutar definiranog tipa `Preticipation`. Za postizanje željenog efekta potrebno je deklarirati `ParticleSystem` koji će iscrtavati i emitirati čestice. Izgled čestica u ovom slučaju reprezentiran je slikom za što se koristi tip `ImageParticle` čije svojstvo `source` predstavlja željenu sliku pohranjenu kao resurs aplikacije. Naposljetku, QML tip `Emitter` emitira zadane čestice. Prethodno je dan minimalni isječak koda 3.11 QML datoteke `Preticipation.qml` koji je potreban da bi se podržale sve opisane funkcionalnosti.

Grupa *Oluja*

Olujne vremenske uvjete simbolizira animacija munje koja blješće ovisno o jačini oluje. Munja uvijek dolazi u kombinaciji s oblacima. Uz ta dva elementa, animacija koja pripada ovoj grupi, ovisno o jačini oluje može prikazivati i padaline.

Slika 3.12: Komponente animiranog prikaza olujnih vremenskih uvjeta

Munja se animira uz pomoć tipova `Timer` i `NumberAnimation` koje QML nudi. Ova vrsta animacije specijalni je slučaj tipa `PropertyAnimation`, a koristi se samo pri promjeni brojčane vrijednosti svojstva. Preciznije, u slučaju grupe *Oluja*, primjenjuje se na svojstvo `opacity` postavljajući početnu vrijednost tog svojstva za animaciju na nulu i krajnju vrijednost na jedinicu. Tako definiranu animaciju sad `Timer`, u svom upravljaču signala koji se emitira kad vremenski brojač istekne, pokreće odnosno zaustavlja. Definicije nabrojenih tipova dane su isječkom koda 3.12.

Zadnja komponenta koju ovaj tip može imati su padaline. One se u QML kodu dinamički učitavaju definiranim tipom `Loader` iz datoteke `Preticipation`, uz zadavanje svojstava koja označavaju tip i intenzitet padalina te širinu i visinu objekta. Učitavanje elemenata uz pomoć tipa `Loader` dosad je već detaljno opisano stoga se ovdje neće konkretizirati.

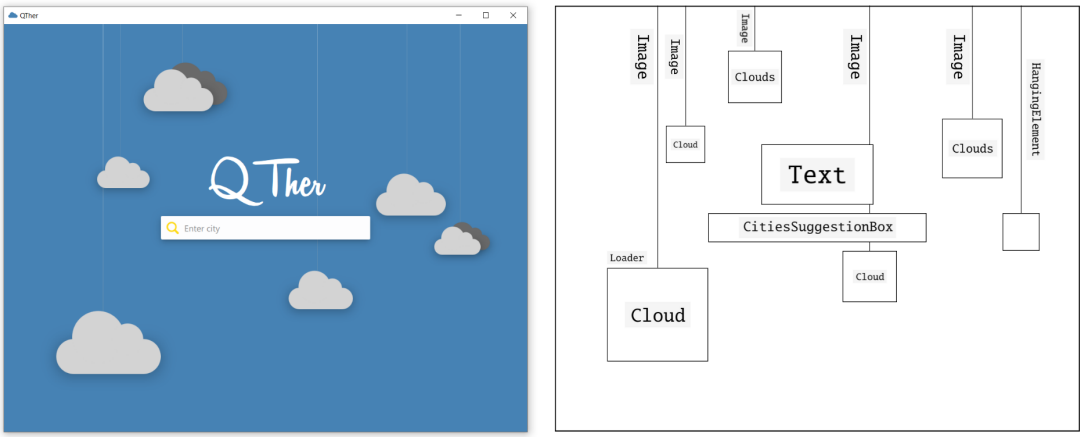
```
Timer {
    interval: 1300
    running: true
    repeat: true
    onTriggered: thunderboltAnimation.running ?
                 thunderboltAnimation.stop() : thunderboltAnimation.start()
}

NumberAnimation  {
    id: thunderboltAnimation
    target: thunderboltImage
    property: 'opacity'
    from: 0
    to: 1
    duration: 700
    loops: Animation.Infinite
}
```

Isječak koda 3.12: Dio QML datoteke `Thunderstorm.qml`

Nakon prolaska kroz sve grupe vremenskih prilika kao i elemente sadržane u njima, QTher je u stanju svaki dobiveni podatak o vremenu vizualno prikazati. Naravno, taj podatak mora doći od *Open Weather* API-ja. Budući da su vizualizacije od krucijalne važnosti pri izradi grafičkih korisničkih sučelja, isto vrijedi i za QTher aplikaciju. Dosad opisani tipovi za stvaranje vizualnih komponenti koristit će se na svakoj stranici aplikacije i to veći broj puta. Kako i na koji način bit će opisano u nastavku gdje napokon slijedi pregled svih stranica aplikacije.

3.3.4 Početna stranica

Odmah po pokretanju aplikacije prikazuje se inicijalni element stoga `HomePage`. Početna stranica sastoji se od tri logičke cjeline – naslova aplikacije, animiranih padajućih elemenata, te padajućeg izbornika za odabir grada. Padajući elementi se prikazuju isključivo radi vizualne atraktivnosti, dok je komponenta padajućeg izbornika svrsi shodna.



Slika 3.13: Struktura početne stranice

Svaki se viseći element sastoji dvije komponente. Prva predstavlja sliku niti na kojem će element visjeti realiziran u QML tipu `Image`. Druga komponenta, tip `Loader`, učitava proizvoljni zadani element koji će biti obješen o nit. Obje gradivne komponente u radu su već opisane stoga se ovdje neće posebno opisivati.

Animacija padajućih elemenata `HangingElementsAnimation` uključuje kolekciju visećih objekata koji padaju od vrha ekrana do postavljene visine odnosno svojstva `y`. To je realizirano kreiranjem dvaju stanja koja glavni animacijski objekt može imati, početnim `up`, te ciljnim `down` stanjem. Svako stanje za određeni padajući element definira njegovu visinu u tom stanju. Usto, pošto svaka promjena između stanja izaziva naglu promjenu pozicije elementa, za zaglađivanje prijelaza uvedene su tranzicije. Takav efekt postiže se definiranjem neke od mnogobrojnih QML tipova animacija (vidi [9]) unutar tipa `Transition`. Primjer jednog opisanog padajućeg elementa dan je isječkom koda 3.13. Kod prikazuje samo one dijelove koji su potrebni za postizanje efekta pada elementa. Dakle, izgled samog elementa se u potpunosti zanemaruje.

Što se tiče zadnje komponente, padajućeg izbornika, njegova funkcionalnost i implementacija prikazane su u odjeljku 3.3.1, stoga se neće ponovno opisivati. Ovdje je samo bitno naglasiti kako odabir bilo kojeg grada iz izbornika vodi direktno na stranicu za prikaz njegove dnevne vremenske prognoze.

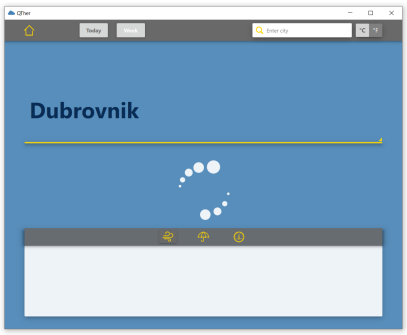
```
Item {
   id: hangingElementsAnimation
   state: "up"
   HangingElement {
      id: hangingElement
      Component.onCompleted: {
         hangingElement.setSource(
            "qrc:/qml/visualizations/basic/Cloud.qml")
      }
   }
   states: [
      State {
         name: "up"
         PropertyChanges { target: hangingElement
                           y: initialHeight }
      },
      State {
         name: "down"
         PropertyChanges { target: hangingElement
                           y: hangingElementHeight }
      }
   ]
   transitions: [
      Transition { to: "down"
                   NumberAnimation { properties: "y"
                                     target: hangingElement
                                     easing.type: Easing.OutBack
                                     duration: 700 }
                 },
      Transition { to: "up"
                   NumberAnimation { properties: "y"
                                     easing.type: Easing.Bezier
                                     duration: 600 }
                 }
   ]
   Timer {
      interval: 100
      running: true
      onTriggered: {
         hangingElementsAnimation.state =
            hangingElementsAnimation.state === "up" ? "down" : "up" }
    }
}
```

Isječak koda 3.13: Animacija padajućeg elementa

3.3.5 Stranica za prikaz dnevne prognoze

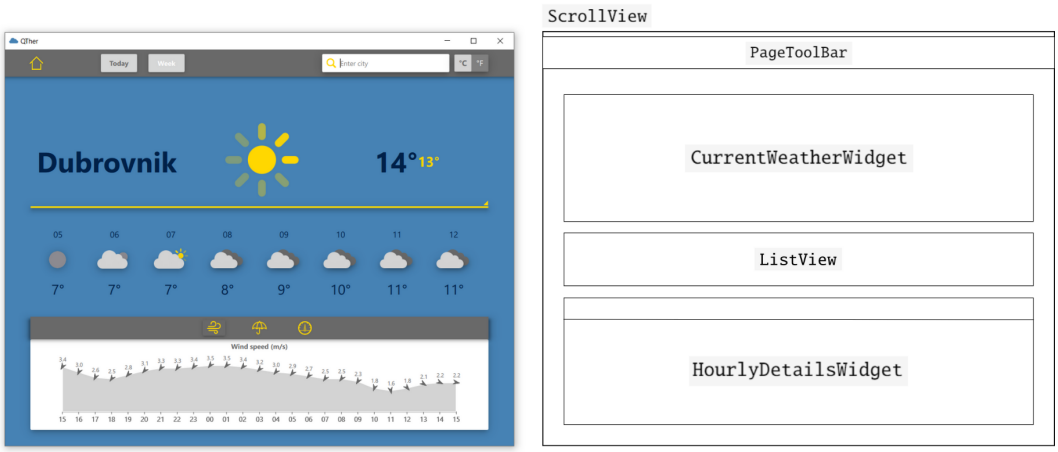
Svakim pristupom stranici za dnevnu vremensku prognozu, bilo s proizvoljne stranice preko padajućeg izbornika, bilo sa stranice za prikaz sedmodnevne prognoze preko gumba *Today*, trebaju se pojaviti vremenski podaci karakteristični za taj grad na aktualni dan.

Ukoliko se radi o prvom upitu na taj grad, podaci se moraju dohvatati API zahtjevom opisanim u potpoglavlju 3.2. Odgovor na taj zahtjev često nije momentalan. U tu svrhu QTher definira tip `Spinner` koji se prikazuje na ekranu korisnika sve dok potrebni vremenski podaci nisu dohvaćeni. Navedena situacija unutar same aplikacije predočena je slikom 3.14.



Slika 3.14: Aplikacija za vrijeme dohvaćanja podataka

U suprotnom, ako se ne radi novi upit na grad, već dohvaćeni podaci samo se prikazuju. Stranica za prikaz tih podataka može se podijeliti na četiri cjeline – alatnu traku, trenutne vremenske uvjete, te osnovne i dodatne vremenske uvjete za narednih 24 sata.



Slika 3.15: Struktura stranice za prikaz dnevne prognoze

Istu logiku prati i QML kod aplikacije gdje su nabrojene cjeline ostvarene u QML tipovima `PageToolBar`, `CurrentWeatherWidget`, `ListView` i `HourlyDetailsWidget` redom. Opis prve komponente dan je detaljno u odjeljku 3.3.2. Za ostala tri elementa isto slijedi u nastavku.

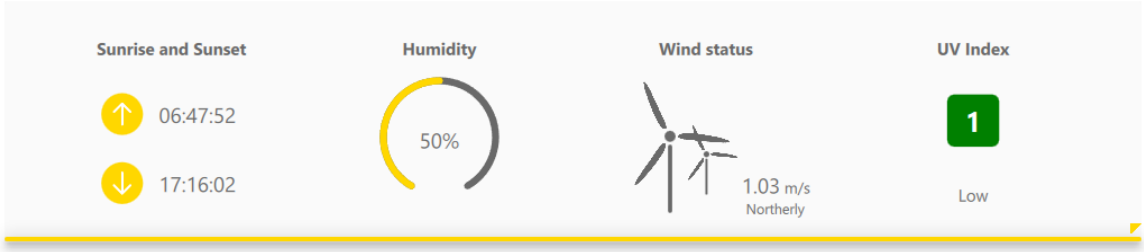
Trenutni vremenski podaci

Element za prikaz podataka o trenutnom vremenu sastoji se od gornjeg, centralnog dijela vidljivog sa slike 3.15, koji sadrži:

- ime grada,
- vizualizaciju trenutnih vremenskih prilika,
- podatak o trenutnoj temperaturi,
- podatak o osjetu trenutne temperature.

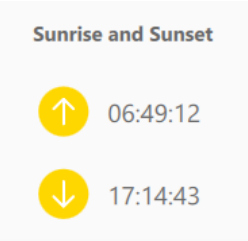
Svi potrebni podaci uredno se dohvaćaju putem API zahtjeva te se ili direktno prikazuju ili interpretiraju za prikaz. U svakom slučaju, podaci se na ekran raspoređuju uz pomoć QML tipa `RowLayout` u poretku slijeva nadesno.

Nadalje, strelica u donjem desnom kutu elementa `CurrentWeatherWidget` daje naslutiti da se element može proširiti. Naime, cijelo područje elementa ispunjava nevidljiva površina osjetljiva na klik miša čiji zadatak je prikaz odnosno skrivanje detalja o trenutnim vremenskim uvjetima. To je realizirano uvođenjem dodatnog stanja koje regulira visinu komponente s detaljima, a prijelaz između stanja je zaglađen tranzicijama. Dnevni detalji se u svakom trenutku mogu prikazati ili sakriti.



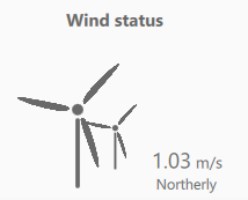
Slika 3.16: Detalji trenutne vremenske prognoze

Dio s detaljima, prikazan slikom 3.17, sastoji se od retka s četiri komponente. Ono što je svima njima zajedničko jest tip `Text` koji se koristi za prikaz naslova. Osim toga, posebnosti svake od komponenti dane su u nastavku.

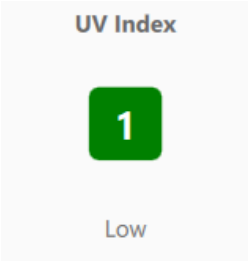


Element `SunsetSunriseWidget` koristi se za prikaz podataka o izlasku i zalasku sunca na aktualni dan. Dobivene API podatke `sunrise` i `sunset` u unix UTC formatu JavaScript metoda preračunava se u lokalno vrijeme. Podaci se zatim, zajedno s prikladnim slikama, raspoređuju unutar QML tipa `GridLayout` kako bi se postigao dani prikaz.

Objekt `HumidityWidget` služi za animirani prikaz postotka vlažnosti zraka. Sastoji se od statičnog kružnog luka sa središnjim kutom od 300°, te animiranog luka koji u njega učitava podatak o postotku. Uzastopno s promjenom vrijednosti kuta pri učitavanju, postotak se ispisuje na sredini komponente.



Za prikaz brzine i smjera vjetra QTher definira element `DailyWindWidget`. Brzina vjetra originalni je podatak iz API odgovora, dok se smjer određuje prema dobivenim stupnjevima. Osim toga, kao dodatni vizualni elementi definirane su dvije vjetrenjače, koje se vrte ovisno o brzini vjetra.

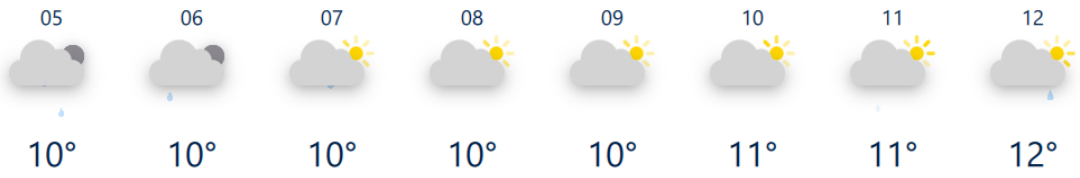


`UVIndexWidget` predstavlja komponentu koja prikazuje podatak o UV indeksu i njegovoj jačini. Uz naslov, sadrži pravokutnik za prikaz UV indeksa obojenog pripadajućom bojom te opisnu jačinu indeksa.

Na kraju, kako bi prikaz bio što pregledniji, element je od ostatka stranice odijeljen panelom s efektom uzvišenosti.

Osnovni dnevni vremenski podaci

Pod osnovne vremenske podatke QTher ubraja temperaturu i kod vremenskog uvjeta pomoću kojeg generira njegovu vizualizaciju. Aplikacija te podatke grupirane po satu prikazuje u listi od 24 elementa za naredna 24 sata.



Slika 3.17: Horizontalna klizna lista vremenskih uvjeta po satima

U kodu je to ostvareno pomoću *model-view* obrasca u kojem model `ListModel` sadrži podatak o satu, temperaturi i kodu vremenskog uvjeta. Interpretirane podatke delegat pogleda `ListView` prikazuje u horizontalnoj kliznoj listi inicijalno pozicioniranoj na sredini stranice. Lista se sastoji od niza `OneHourWidget` objekata, koji pojednostavljuju pozicioniranje dohvaćenih podataka, kao i učitavanje vizualizacija vremenskih uvjeta. Definicija opisanog pogleda dana je isječkom koda 3.14. Svi podaci koji se pridružuju svojstvima komponente `OneHourWidget` podaci su dohvaćeni putem API zahtjeva.

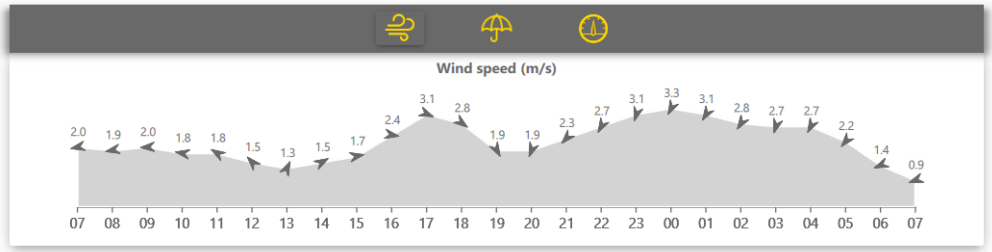
```
ListView {
    id: currentWeatherHourlyListView
    model: hourlyListModel
    anchors.fill: parent
    anchors.leftMargin: 15
    spacing: 10
    orientation: ListView.LeftToRight
    flickableDirection: Flickable.HorizontalFlick
    clip: true

    delegate: Rectangle {
        width: 90
        height: 140
        color: "transparent"

        OneHourWidget {
            id: oneHourWidget
            hour: formattedHour
            temperature: temp
            weatherCode: code
            weatherIcon: icon
        }
    }
}
```

Isječak koda 3.14: Pogled osnovnih dnevnih podataka

Dodatni dnevni vremenski podaci

Kao posljednja vizualna cjelina stranice *Today* ističu se podaci o vjetru, padalinama i tlaku zraka, smješteni na dnu stranice. Navedeni podaci reprezentiraju se kroz tri grafovska prikaza. Inicijalno se prikazuje onaj s podacima o vjetru, no klikom na jedan od tri gumba na vrhu komponente, moguće je po želji odabrati graf za prikaz.

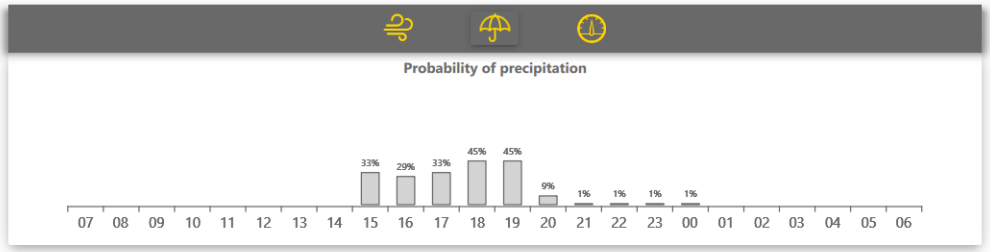
U pozadini zapravo QML tip `Loader`, definiran unutar `HourlyDetailsWidget` komponente, ovisno o odabiru učitava jedan od tri implementirana objektna tipa bazirana na pogledu `ChartView` koji se koristi za prikaz niza grafova i koordinatnih osi. Za korištenje tog tipa, kao i ostalih grafovskih tipova nužno je uvesti modul `QtCharts` u QML datoteku.



Slika 3.18: Grafički prikaz podataka o vjetru po satima

Slikom 3.18 dan je linijski graf koji prikazuje brzinu vjetra po satima. Na tom je grafu os x vremenska, odnosno prikazuje podatke o satima, dok se na osi y prikazuje brzina vjetra. Iz samog naslova grafa vidljivo je da je vrijednost brzine dana u m/s.

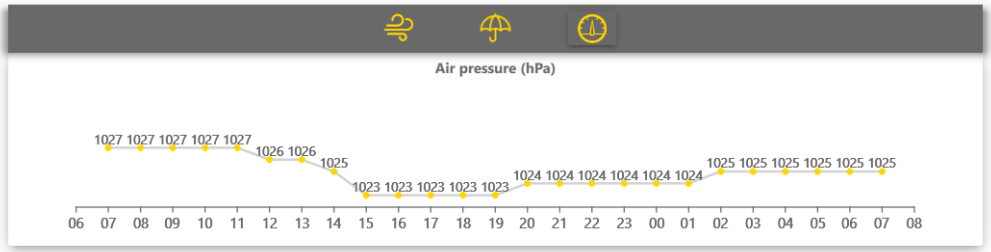
U definiciji grafa unutar komponente `HourlyWindWidget` korišten je `LineSeries` QML tip, koji podatke prikazuje kao niz točki povezanih ravnim linijama. Podaci o točkama na grafu čuvaju se u modelu `ListModel` kako bi se na graf dodatno postavile animirane strelice. Naime, svakim odabirom prikaza grafa, strelice postavljene u točkama vrijednosti brzine vjetra na određeni sat, zakreću se ovisno o smjeru vjetra.



Slika 3.19: Grafički prikaz mogućnosti padalina po satima

Grafički element na slici 3.19 prikazuje mogućnost padalina po satima na stupčastom grafu. Os *x* prikazuje podatke o satima, dok os *y* označava mogućnost padalina u postocima.

Za ovakvu vrstu grafa u implementaciji se koristi tip `BarSeries` u kojem se podaci vizualiziraju nizom okomitih traka, u ovom slučaju grupiranih po satima. Također, navedeni tip se može animirati, što će rezultirati postepenim podizanjem svakog od stupaca do postotka koji on prikazuje. Inicijalizacija koordinatnih osi i postotka kojeg stupci prikazuju odvija se integracijom s JavaScript kodom odmah nakon instanciranja grafičkog objekta `HourlyPrecipitationWidget`.



Slika 3.20: Grafički prikaz tlaka zraka po satima

Posljednji graf, predočen slikom 3.20, prikazuje tlak zraka po satima na linijskom grafu, gdje je os *x* vremenska, a os *y* prikazuje vrijednost tlaka zraka u hektopaskalima.

Podaci o tlaku zraka prikazuju se, jednako kao i kod grafovskog prikaza podataka o vjetru, linijskim grafom `LineSeries`. U ovom slučaju na graf se ne dodaju animirane strelice, već se za iscrtavanje točaka na grafu koristi QML tip `ScatterSeries` u koji se točke dodaju odmah nakon instanciranja grafičkog objekta `HourlyPressureWidget`.

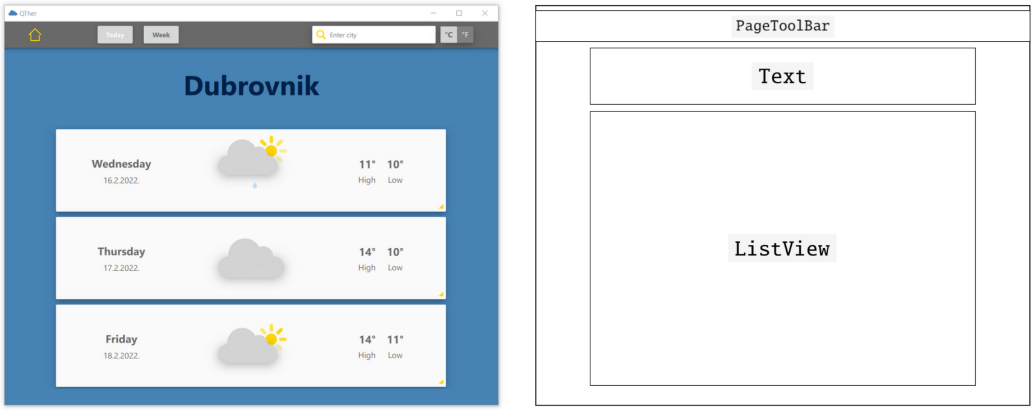
Primjeri uporabe svih navedenih grafovskih tipova koje QTher koristi u implementaciji, ali i još pregršt drugih, mogu se naći u službenoj dokumentaciji Qt-a [11].

3.3.6 Stranica za prikaz sedmodnevne prognoze

Treća, ujedno i posljednja stranica za koju QTher aplikacija zna, jest stranica za prikaz sedmodnevne vremenske prognoze. Korisnik na tu stranicu može doći isključivo odabirom opcije *Week* na alatnoj traci.

Time se na aplikacijski stog stavlja stranica `SevenDaysWeatherPage`, kojoj se predaju svi potrebni podaci. Na temelju njih se, na vrhu stranice, prikazuje naslov koji sadrži ime grada, te lista s tjednim vremenskim podacima ispod njega. Naravno, podacima za prikaz u listi upravlja delegat, koji zasebnoj komponenti `LongTermDayWidget` šalje sve bitne po-

datke iz modela. Delegat je prikazan isječkom koda 3.15, a sam izgled elemenata liste, kao i čitava struktura stranice za prikaz sedmodnevne vremenske prognoze slikom 3.21.

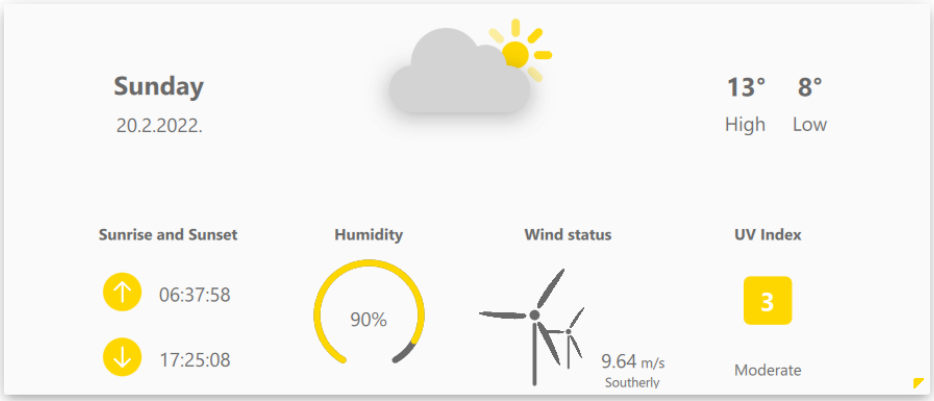


Slika 3.21: Stranica za prikaz sedmodnevne prognoze

```
delegate: Component {
    id: sevenDaysDelegate

    LongTermDayWidget {
        width: sevenDaysListView.width * 0.9
        // - mainItem -
        weekDay: weekDayModel
        date: dateModel
        weatherIcon: weatherIconModel
        weatherCode: weatherCodeModel
        temperatureMin: temperatureMinModel
        temperatureMax: temperatureMaxModel
        // - SunsetSunriseWidget -
        sunset: sunsetModel
        sunrise: sunriseModel
        // - DailyWindWidget -
        windSpeed: windSpeedModel
        windDirection: windDirectionModel
        // - HumidityWidget -
        humidity: humidityModel
        // - UVIndexWidget
        uvIndex: uviModel
    }
}
```

Isječak koda 3.15: Delegat pogleda tjednih vremenskih podataka

Iz navedenog koda može se iščitati kako uz osnovne, stranica za prikaz tjednih vremenskih prilika, šalje komponenti `LongTermDayWidget` i neke dodatne podatke. Razlog tome je što se svaka komponenta unutar liste može dodatno proširiti s detaljima. Jednako kao i kod trenutnih vremenskih podataka na dnevnoj stranici, detalji su prikazani kroz četiri komponente. One se odnose na prikaz podataka o izlasku i zalasku sunca, vlažnosti zraka, brzini i smjeru vjetra, te UV indeksu. Sve navedene komponente, kao i logika otvaranja prikaza detalja koja stoji u pozadini, detaljno su opisane u prethodnom poglavlju.



Slika 3.22: Element iz liste tjedne vremenske prognoze s detaljima

Zaključak

Qt se često naziva GUI bibliotekom. Istina jest da se velik dio modula Qt biblioteke koristi upravo za razvoj grafičkih korisničkih sučelja, no Qt pruža mnogo više od toga. Naime, biblioteka sadrži širok spektar modula koji nude različite funkcionalnosti čime razvoj i implementaciju aplikacija čine bržom i lakšom. Proces razvoja dodatno pojednostavljuje Qt-ova integrirana razvojna okolina Qt Creator. Također, posebnosti poput mehanizma signala i utora te *meta-object* sustava kojim je on omogućen, ono su što ovu biblioteku ističe pred drugima.

Između ostalog, Qt nudi i mogućnost međusobne integracije različitih programskih jezika. Osim što je pisana u C++ jeziku, u njemu se implementira i sva logika aplikacija razvijanih u Qt biblioteci. Budući da je u takvim aplikacijama naglasak stavljen na izgled grafičkog korisničkog sučelja, Qt je razvio vrlo moćan deklarativni programski jezik QML, koji se jednostavno proširuje C++ i JavaScript kodom. QML je podržan Qt QML i Qt Quick modulima koji nude osnovnu infrastrukturu, glavne funkcionalnosti, te posebne QML tipove koje jezik može koristiti.

Sve navedeno svjedoči o brojnim prednostima Qt biblioteke. Međutim, glavni cilj svake aplikacije je imati što veću mrežu korisnika. Zato se kao najveća posebnost ove biblioteke smatra mogućnost jednostavnog prijenosa aplikacija na velik broj platformi, na čemu se temelji i Qt-ov slogan *"Code once, deploy everywhere"*.

Bibliografija

[1] *Open Weather*, https://openweathermap.org/, 2021, (pristupljeno 2.2.2022.).

[2] *Weather Conditions*, https://openweathermap.org/weather-conditions#Weather-Condition-Codes-2, 2021, (pristupljeno 4.2.2022.).

[3] *GitHub*, https://github.com/skpetra/MasterThesis, 2022, (pristupljeno 15.2.2022.).

[4] The Qt Company Ltd, *Build Systems*, https://doc.qt.io/qtcreator/creator-project-other.html, 2022, (pristupljeno 28.1.2022.).

[5] _____, *Concepts - Visual Parent in Qt Quick*, https://doc.qt.io/qt-6/qtquick-visualcanvas-visualparent.html, 2022, (pristupljeno 22.1.2022.).

[6] _____, *Enumeration Attributes*, https://doc.qt.io/qt-6/qtqml-syntax-objectattributes.html#enumeration-attributes, 2022, (pristupljeno 29.1.2022.).

[7] _____, *Interacting with QML Objects from C++*, https://doc.qt.io/qt-6/qtqml-cppintegration-interactqmlfromcpp.html, 2022, (pristupljeno 30.1.2022.).

[8] _____, *QAbstractItemModel Class*, https://doc.qt.io/qt-5/qabstractitemmodel.html#roleNames, 2022, (pristupljeno 5.2.2022.).

[9] _____, *QML Animation and Transitions*, https://doc.qt.io/archives/qt-4.8/qdeclarativeanimation.html, 2022, (pristupljeno 10.2.2022.).

[10] _____, *QML Object Attributes*, https://doc.qt.io/qt-6/qtqml-syntax-objectattributes.html#property-attributes, 2022, (pristupljeno 26.1.2022.).

[11] _____, *Qt Charts Examples*, https://doc.qt.io/qt-5/qtcharts-examples.html, 2022, (pristupljeno 13.2.2022.).

[12] _____, *Registering Non-Instantiable Types*, https://doc.qt.io/qt-6/qtqml-cppintegration-definetypes.html#registering-non-instantiable-types, 2022, (pristupljeno 31.1.2022.).

[13] _____, *Using C++ Models with Qt Quick Views*, https://doc.qt.io/qt-5/qtquick-modelviewsdata-cppmodels.html, 2022, (pristupljeno 5.2.2022.).

[14] _____, *Variables*, https://doc.qt.io/qt-5/qmake-variable-reference.html, 2022, (pristupljeno 28.1.2022.).

Sažetak

U ovom diplomskom radu opisana je Qt6 biblioteka namijenjena razvoju grafičkih korisničkih sučelja. Prikazana je struktura navedene biblioteke, te osnovni mehanizmi i koncepti na kojima se temelji. Pored toga, dan je pregled deklarativnog programskog jezika QML, s posebnim naglaskom na module Qt QML i Qt Quick iz Qt biblioteke kojima je jezik podržan. Na kraju, kako bi se prethodno opisano teorijsko znanje primijenilo u praksi, prikazan je proces implementacije jednog prigodnog grafičkog korisničkog sučelja.

Summary

In this master thesis, the Qt6 library is analysed, it's purpose being to develop graphical user interface. The structure of the listed library is shown as are the main mechanisms and concepts on which it is based. Moreover, a review of the declarative programing language QML is given, with specific focus on Qt QML and Qt Quick from the Qt library with which the language is supported. The thesis concludes with a practical application of the previously described theoretical knowledge via the implementation of a convenient graphical user interface.

Životopis

Rođena sam 27. ožujka 1997. godine u Dubrovniku. Obrazovanje sam započela u Osnovnoj školi Mokošica, a zatim nastavila u Prirodoslovno-matematičkoj gimnaziji Dubrovnik. Godine 2015. završila sam srednjoškolsko obrazovanje i preselila se u Zagreb kako bih studirala na preddiplomskom sveučilišnom studiju Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Preddiplomski studij završila sam 2019. godine. Iste godine upisala sam diplomski sveučilišni studij Računarstvo i matematika.