1 Uvod

Razvoj mobilnih aplikacija jedna je od radionica u sklopu projekta Studentsko poduzetništvo - 404 koji se izvodi 2023. u suradnji Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek (FERIT), Fakulteta primijenjene matematike i informatike u Osijeku (MATHOS) te Ekonomskog fakulteta u Osijeku (EFOS). Cilj radionice jest pokazati učenicima osnovne koncepte razvoja mobilnih aplikacija s fokusom na platformu Android te ih potaknuti na samostalno učenje i istraživanje.

Ovaj minijaturni priručnik prati aktivnosti radionice gdje će se kroz kreiranje nove Android aplikacije učenici upoznati s osnovnim pojmovima kao što su izgradnja korisničkog sučelja, povezivanje programskog koda i elemenata korisničkog sučelja, ali i sa složenijim elementima aplikacija poput uporabe senzora ili prikaza podataka dostupnih na Internetu. Aplikacija koja će biti izrađena sastoji se od tri zasebna ekrana, ekrana za inspiraciju koji će poslužiti za prolazak kroz osnove kreiranja korisničkog sučelja, ekrana za igru koji će poslužiti za upoznavanje sa senzorima te ekrana za šalu koji će poslužiti za dohvaćanje podataka s interneta i njihov prikaz. Svijet razvoja mobilnih aplikacija puno je širi od sadržaja koji pokriva radionica pa će u posljednjem poglavlju biti prikazane poveznice na vanjske resurse koje je moguće koristiti za samostalan rad i učenje. Za sve naknadne informacije i pitanja, slobodno se javite putem e-maila na bruno.zoric@ferit.hr.

Informacija

Detaljan članak o povijesti Android platforme moguće je pročitati na `https://arstechnica.com/gadgets/2016/10/building-android-a-40000-word-history-of-googles-mobile-os/`

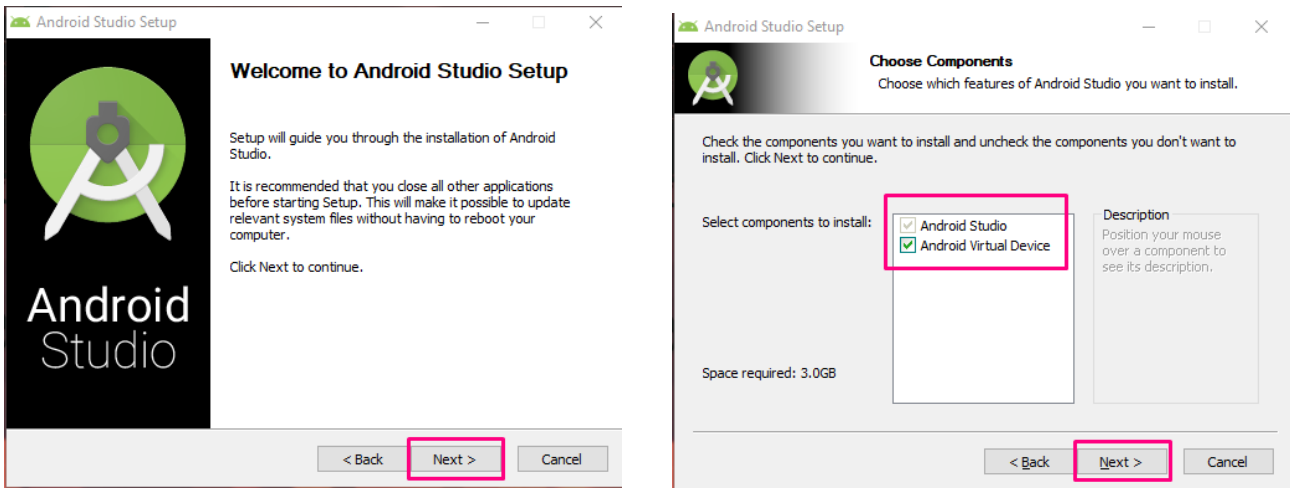
2 Instalacija alata

Kod izrade aplikacija za Android platformu u pravilu se rabi alat Android Studio koji održava tvrtka Google, a koji je zasnovan na alatu IntelliJ Idea tvrtke JetBrains. Isti je moguće preuzeti na `https://developer.android.com/studio/`, a u trenutku pisanja ovog priručnika aktualna je inačica *Electric Eel* koja će i biti korištena unutar priručnika. Kada se instalacijski paket preuzme na računalo i pokrene, prikazuje se ekran na slici 1. Odabirom opcije `Next` započinje instalacija alata. Kada se instalira Android studio, koji će se koristiti za pisanje programskog koda i sve ostale poslove vezane uz razvoj aplikacije, moguće je uključiti još i instalaciju Android virtualnog uređaja. Ovo je svakako preporučeno jer će omogućiti lako pokretanje aplikacije na emulatoru pa je tako moguće kreirati aplikacije čak i ako ne posjedujete vlastiti Android telefon. Označite stoga ovu mogućnost prema slici 1, a sam će uređaj biti konfiguriran kasnije.

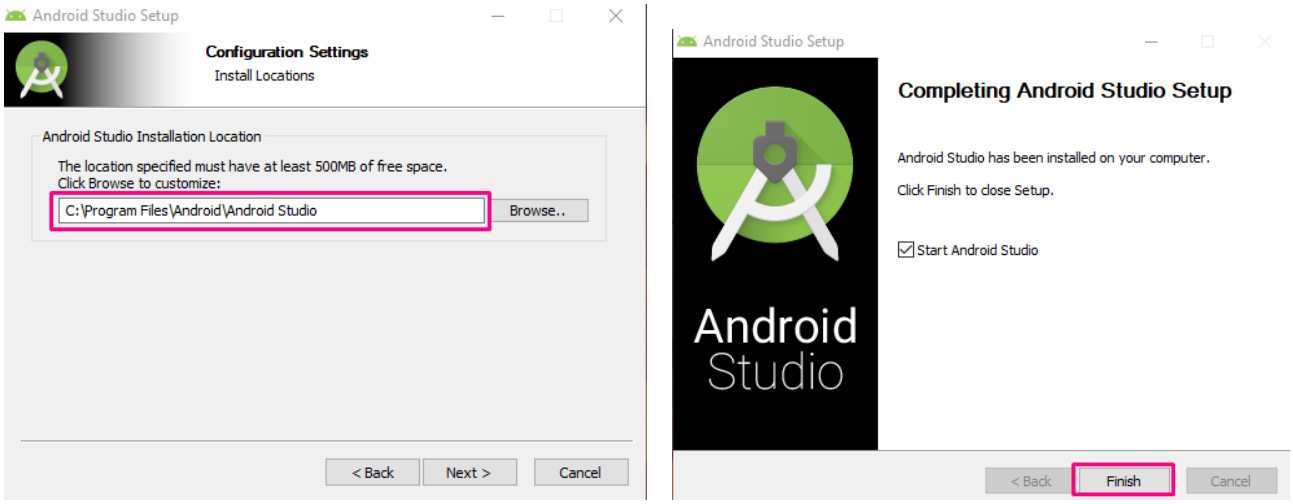
Zatim je potrebno odabrati lokaciju na koju će se instalirati Android studio. Osigurajte da imate dovoljno prostora na lokaciji na koju ga instalirate i ukoliko ste u mogućnosti instalirajte ga na SSD disk. Odabir lokacije prikazan je slikom 2. Što se tiče instalacije samog alata, ona je gotova. Sada je alat potrebno pokrenuti i instalirati dodatne stvari koje su neophodne prilikom razvoja aplikacija.

Prilikom pokretanja pojavljuje se čarobnjak koji nudi uvoz postavki iz neke od starijih inačica Android studija. S obzirom da je ovo prvi put kako ga instalirate, ignorirajte ovu opciju.

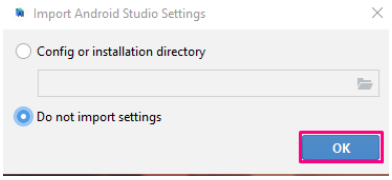
S obzirom da postavke nismo uvezli, potrebno je konfigurirati Android studio. Za početak, odaberite želite li slati Googleu anonimnu statistiku o korištenju alata. Zatim, odaberite da želite postaviti vlastite postavke, a ne koristiti standardne mogućnosti prilikom instalacije.



Slika 1: Instalacija Android Studija - 1. i 2. korak



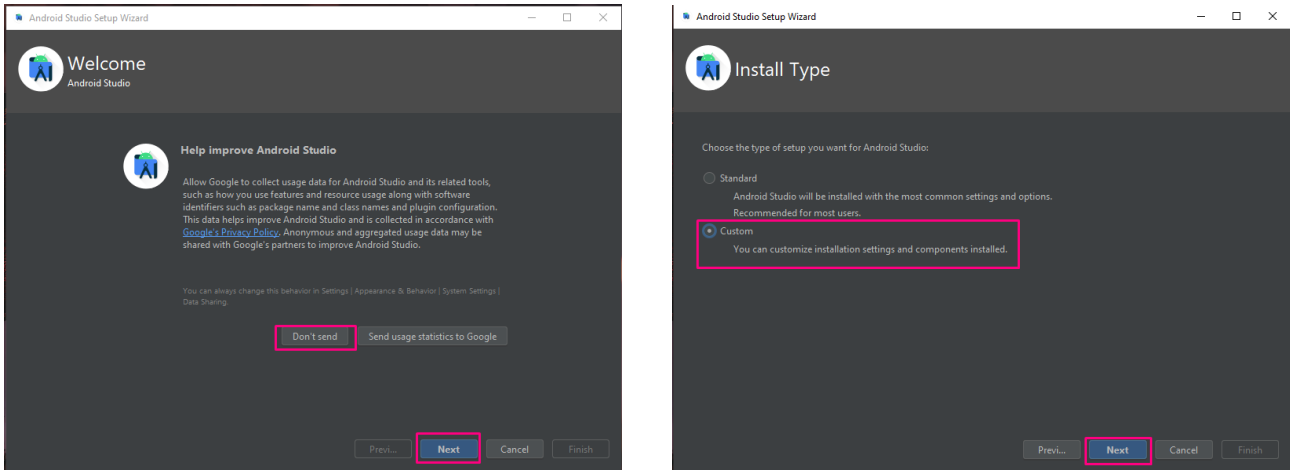
Slika 2: Instalacija Android Studija - 3. i 4. korak



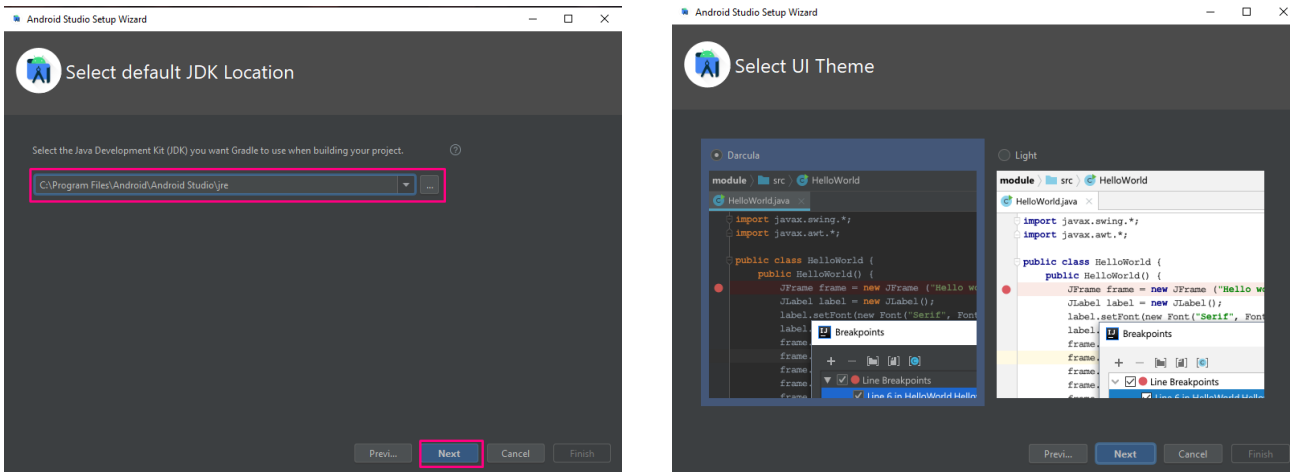
Slika 3: Instalacija Android Studija - 5. korak

Najprije je potrebno postaviti lokaciju gdje će biti instalirana razvojni paket za programski jezik Javu. Ovo možete ostaviti kako je podrazumijevano, no možete lokaciju i promijeniti, s obzirom da nije nužno da se ona nalazi u instalacijskom direktoriju Android studija, a moguće isti razvojni paket moguće je koristiti i za druge oblike programske podrške. Osim toga, postavlja se tema koja definira izgled razvojnog okruženja. Tamna tema u pravilu je popularnija i ugodnija očima, tako da ju i mi preporučujemo. Ipak, u slučaju da kod pokazujete na projektoru većoj skupini ljudi, svjetla tema se pokazala puno zahvalnijom i praktičnijom radi boljeg kontrasta.

Sljedeće je na redu instalacija Android razvojnog paketa koji sadrži sve klase i alate nužne za nesmetan razvoj aplikacija, Android platforme koju kanite ciljati prilikom razvoja (uobičajeno posljednja koja je dostupna), te Android virtualnog uređaja. Ovo potonje je u stvari emulator, odnosno programsko rješenje koje glumi stvarni uređaj u svakom smislu i trudi se ponašati jednako kako bi se ponašao

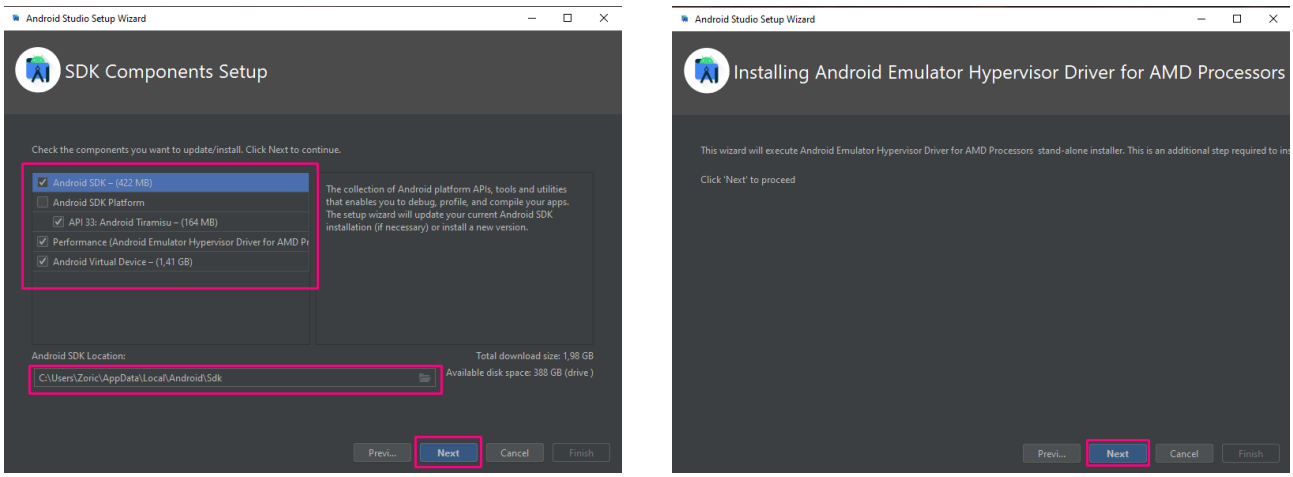


Slika 4: Instalacija Android Studija - 6. i 7. korak



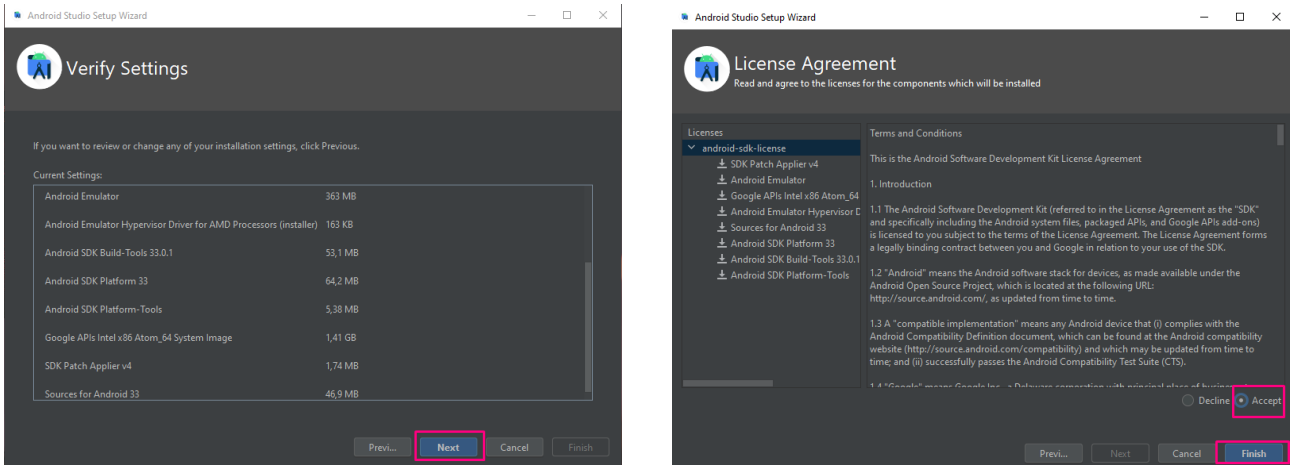
Slika 5: Instalacija Android Studija - 8. i 9. korak

i stvarni uređaj. Primijetite kako je količina podataka koju je nužno preuzeti doista velika. Opet, nije nužno da se razvojni kit i ostali alati nalaze unutar instalacije Android studija ili da se nalaze u podrazumijevanoj mapi. Slobodno izmijenite lokaciju prilikom instalacije. Na ovom ekranu bit će ponuđene još neke dodatne opcije, primjerice za ubrzavanje emulacije korištenjem odgovarajućih alata namjenjenih procesoru stroja na koji instalirate razvojno okruženje.



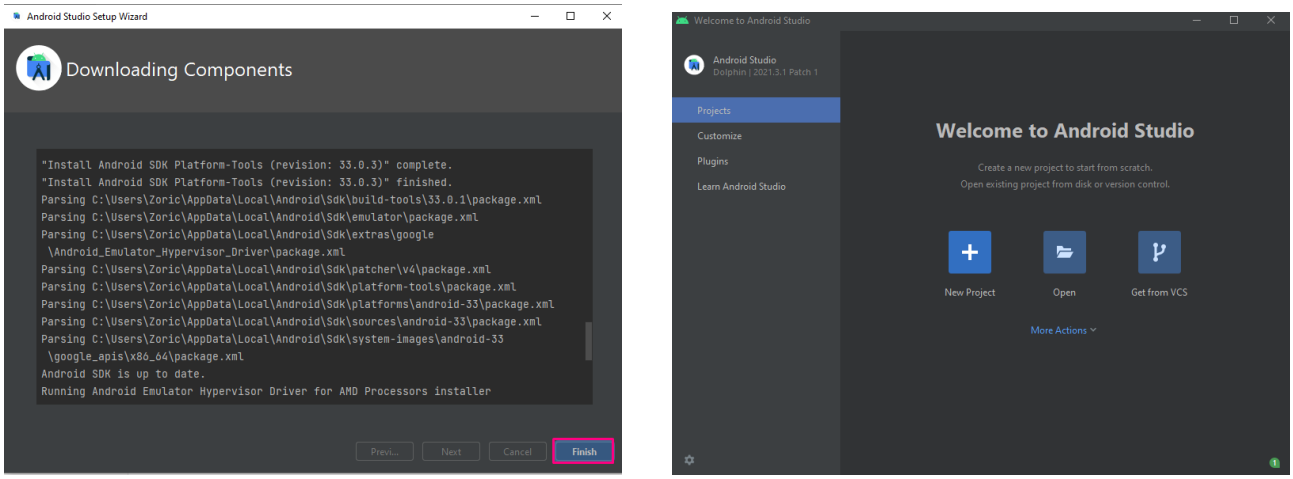
Slika 6: Instalacija Android Studija - 10. i 11. korak

Nakon svega, preostalo je samo provjeriti sve odabrane postavke i potvrditi da prihvaćate uvjete korištenja kako bi se započelo preuzimanje i instalacija svih željenih alata.



Slika 7: Instalacija Android Studija - 12. i 13. korak

U konačnici, nakon što ste popili kavu (ili 3) dok se sve obavilo, dočekat će vas ekran prikazan slikom 8. Klikom na Finish bit će prikazan početni ekran Android studija na kojem možete kreirati novi projekt ili otvoriti neki od postojećih na kojima ste već ranije radili.

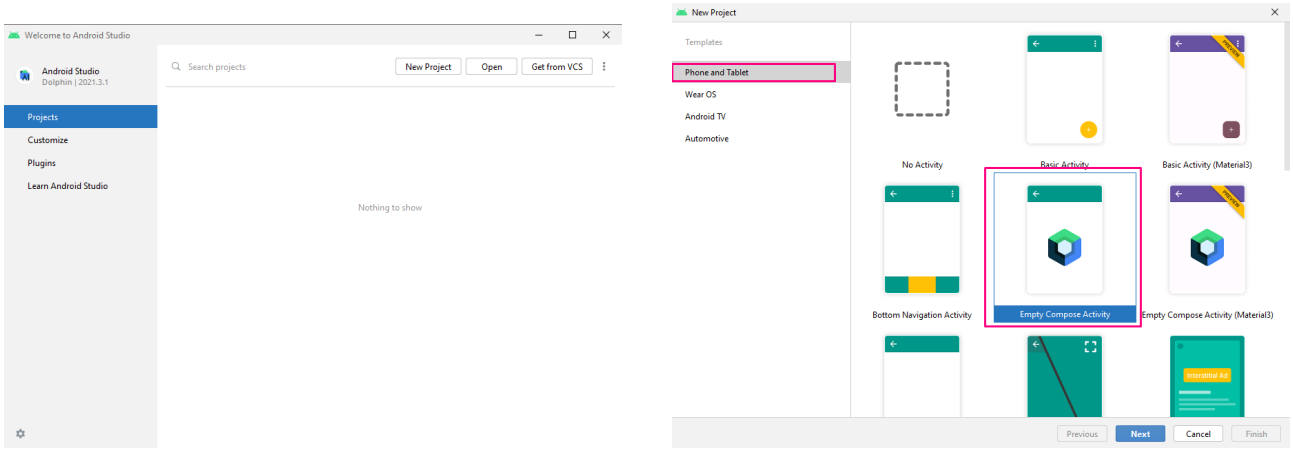


Slika 8: Instalacija Android Studija - 14. korak

3 Kreiranje projekta

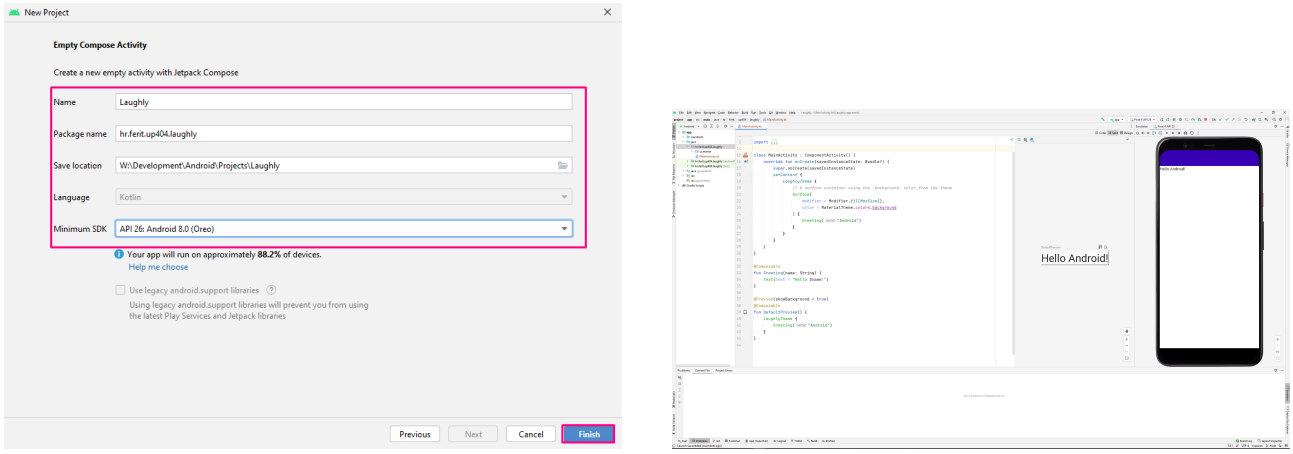
Kada je razvojno okruženje instalirano, može se pristupiti kreiranju novog projekta. Klikom na gumb `New Project` kreira se prazan projekt i otvara novi izbornik koji nudi velik broj gotovih predložaka. Svaki od predložaka namijenjen je nekom obliku aplikacije, a vjerojatno ste brojne od njih već vidjeli u aplikacijama koje svakodnevno koristite. Za početak ćemo odabrati platformu za koju želimo raditi aplikaciju iz izbornika. Nude se razne platforme, od telefona do automobila, no mi ćemo odabrati telefone i tablete. Što se tiče predloška, krenut ćemo od najjednostavnije moguće aplikacije s jednim ekranom, a zatim ćemo postupno uvoditi nove ekrane i sadržaj pa stoga odabiremo `Empty Compose Activity`.

Kada je odabran željeni predložak, potrebno je unijeti osnovne informacije o aplikaciji. Najprije



Slika 9: Kreiranje novog projekta - 1. i 2. korak

se unosi ime aplikacije. Riječ je samo o nazivu koji ne mora biti jedinstven, a bilo bi dobro da bude pamtljiv kako bi korisnici Vašu aplikaciju lakše pronašli i dijelili. Zatim se definira ime paketa. Naime, sav kod u programskim jezicima Java i Kotlin koji se mogu rabiti za kreiranje aplikacija za Android platformu razdvaja se po paketima, a potrebno je definirati vršni paket u kojem će biti sav kod. Ovo ime paketa ima još jednu važnu ulogu, ono predstavlja jedinstveni identifikator svake aplikacije u svijetu i mora biti jedinstveno. Preporučuje se stoga uzeti obrnuto ime domene i na kraj dodati ime projekta. Uz to se definira još lokacija gdje će biti pohranjena projkekt na računalu te najstarija inačica Androida na kojoj će biti moguće pokrenuti aplikaciju. Ovisno o najstarijoj inačici koja se odabere, našu aplikaciju moći će skinuti i koristiti više ili manje korisnika, a pružena je korisna poveznica koja u postotcima ukupnih korisnika Androida govori koliko će točno korisnika ona biti dostupna. Novije inačice sustava Android donose nove značajke i poboljšane funkcionalnosti koje ponekad zna biti teško podržati na starijim inačicama. Ovdje svaki programer mora odvagnuti koliko mu je važno dobiti još nekoliko postotaka korisnika u odnosu na cijenu u vremenu i novcu koji za to mora uložiti. Kada je sve spremno, aplikacija se prikazuje u razvojnom okruženju i moguće je započeti s radom.



Slika 10: Kreiranje novog projekta - 3. i 4. korak

4 Uvođenje ekrana i navigacija

Android aplikacije u pravilu imaju ekrane koji služe za prikaz sadržaja korisniku. Za potrebe prikaza pojedinog ekrana moguće je koristiti rarzličite gotove komponente koje se zovu `Activity` ili `Fragment`. Iako je ranije primaran način za oblikovanje i izgradnju korisničkih sučelja bio napisati `XML` kod koji ih opisuje i koji se vezuje uz pojedini `Activity` ili `Fragment`, danas to više nije tako. U novije vrijeme koristi se pristup gdje je svaka komponenta sučelja, uključujući i cijele ekrane, predstavljena funkcijom. Ovo je omogućeno korištenjem biblioteke imena `Jetpack Compose`, a taj pristup i navedena biblioteka bit će korišteni i u sklopu ove radionice. U ovom slučaju, mora postojati samo jedan `Activity` koji će služiti kao polazna točka i kontejner za sve ekrane aplikacije. Sama aplikacija imat će pet ekrana. Glavni ekran (`MainScreen`) sadržavat će komponente za navigaciju i određivati koji je početni ekran aplikacije. Početni ekran (`HomeScreen`) sadržavat će gumbe za navigaciju prema pojedinim aktivnostima aplikacije. Tri ekrana koja predstavljaju aktivnosti u aplikaciji bit će redom inspirirajući ekran (`InspireScreen`), zaigrani ekran (`PlayScreen`) te nasmijani ekran (`LaughScreen`).

Prvi korak koji ćemo napraviti je priprema projekta. Prilikom izrade aplikacije za bilo koju platformu programer se oslanja na kod i alate koje su razvili drugi programeri prije njega, a koji dolaze u obliku pripremljenih biblioteka. S obzirom da ih je razvio netko "izvana", često se nazivaju vanjskim ovisnostima (engl. *external dependency*). U suprotnom bi izrada bila osjetno dugotrajnija i zamornija. Kod Android aplikacija, ove se vanjske ovisnosti uključuju u projekt dodavanjem nekoliko linija koda u datoteku `build.gradle (Module: app)` gdje se opisuje što točno želimo uključiti i koristiti u našem projektu. Sve potrebne vanjske ovisnosti za cjelokupnu radionicu dane su izlistanjem 1.

Listing 1: Ovisnosnosti o vanjskim bibliotekama u datoteci build.gradle (Module:app)

```
/*****************************************************************/
/* Specific dependencies for the workshop:*/
/*****************************************************************/
// Navigation with jetpack compose:
def nav_version = "2.5.3"
implementation "androidx.navigation:navigation-compose:$nav_version"

// Ktor for networking:
def ktor_version = '2.2.2'
implementation "io.ktor:ktor-client-core:$ktor_version"
implementation("io.ktor:ktor-client-android:$ktor_version")
implementation "io.ktor:ktor-client-serialization:$ktor_version"
implementation "io.ktor:ktor-serialization-gson:$ktor_version"
implementation("io.ktor:ktor-client-content-negotiation:$ktor_version")
implementation "io.ktor:ktor-client-logging-jvm:$ktor_version"
```

Kroz sve ekrane aplikacije koristit ćemo resurse poput boja i dimenzija. Oni se mogu definirati u mapi `res/values` u odgovarajućim datotekama, a izgled tih datoteka dan je izlistanjem 2.

Listing 2: Dimenzije i boje - dimens.xml i colors.xml

```
<!-- dimens.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="padding_medium">16dp</dimen>
</resources>

<!-- Insert into colors.xml-->
<!-- Custom colors -->
<color name="hotpink">#ef7c8e</color>
<color name="cream">#fae8e0</color>
<color name="spearmint">#b6e2d3</color>
<color name="rosewater">#d8a7b1</color>
```

Kako bismo mogli pristupiti organizaciji ekrana naše aplikacije, najprije ćemo definirati sve ekrane i njihove jedinstvene oznake. Ove jedinstvene oznake nazivaju se još i rutama. Ruta predstavlja jedinstvenu oznaku svakog ekrana i omogućuje komponenti za navigaciju da točno zna kamo treba

odvesti korisnika. Ove rute definirat ćemo u posebnoj datoteci koju ćemo nazvati `Navigation.kt`, a koja je prikazana izlistanjem 3.

Listing 3: Navigacija - Navigation.kt

```
sealed class Screen(val route: String) {
  object Home : Screen("home")
  object Inspire : Screen("inspire")
  object Laugh : Screen("laugh")
  object Play : Screen("throw")
}
```

Svaka funkcija koja predstavlja dio korisničkog sučelja označena je oznakom `@Composable`. S obzirom da ćemo za početak samo kreirati prazne ekrane za pojedine aktivnosti koje ćemo nadograđivati i popunjavati sadržajem kako radionica bude odmicala, kreirajmo redom sva tri ekrana za aktivnosti, prema izlistanjima 4, 15, 21. Svaki od ovih ekrana kreira se u zasebnoj datoteci čije ime odgovara imenu ekrana, dakle `InspireScreen, PlayScreen, LaughScreen`.

Listing 4: Inspirirajući ekran - InspireScreen.kt

```
@Composable
fun InspireScreen() {
    Text(text = "Inspire screen")
}
```

Listing 5: Zaigrani ekran - PlayScreen.kt

```
  @Composable
  fun PlayScreen() {
    Text(text = "Play screen")
  }
```

Listing 6: Nasmijani ekran - LaughScreen.kt

```
  @Composable
  fun LaughScreen() {
    Text(text = "Laugh screen")
  }
```

Idući korak je kreiranje početnog ekrana. Riječ je o ekranu koji će sadržavati gumbe koji će korisniku omogućiti odlazak na neki od ranije pripremljenih ekrana u aplikaciji. Izgled početnog ekrana dan je izlistanjem **??**. Ovaj je ekran nešto kompleksniji od dosadašnjih. Vidimo da, kako bi taj ekran radio, traži od nas da mu damo `NavController`. Riječ je o komponenti za navigaciju koju ćemo pripremiti kasnije i predati ju početnom ekranu svaki put kada ga zatrebamo. Osim toga, sučelje ovog ekrana sadrži više sadržaja no prethodni ekrani. Za kreiranje sučelja koriste se druge `Composable` funkcije koje pozivamo unutar naše funkcije `HomeScreen()`. Najprije, želimo kreirati kontejner za sav sadržaj i postaviti mu pozadinsku sliku. Ovo postižemo funkcijom `Box()` i postavljanjem slike kao prvog elementa u nju (poziv funkciji `Image()`). Uz sliku, funkcija `Box()` sadržavat će i stupac s gumbima, za što se koristi funkcija `Column()`. Unutar potonje iskoristit će se tri poziva funkciji `NavigateButton()` koju smo sami definirali u datoteci `NavigateButton.kt`. Naime, čest je slučaj da se pojedine komponente koriste na više mjesta. Kako bi se izbjeglo ponavljanje koda na svim tim mjestima (npr. postavljanje boje, veličine teksta i slično), moguće je definirati komponentu zasebno u obliku funkcije sa željenim vrijednostima ovih postavku i onda ju pozvati gdje god je ona potrebna. Navigacijski gumb definiran je u datoteci `NavigateButton.kt`, a izgled je dan izlistanjem koda 8.

Listing 7: Početni ekran - HomeScreen.kt

```
@Composable
fun HomeScreen(navController: NavController) {
```

```
  Box(modifier = Modifier.fillMaxSize()) {
    Image(
    painter = painterResource(id = R.drawable.background),
    contentDescription = "Background gradient image",
    modifier = Modifier.matchParentSize(),
    contentScale = ContentScale.Crop
    )
    Column(
    modifier = Modifier.fillMaxSize(),
    verticalArrangement = Arrangement.Center,
    horizontalAlignment = Alignment.CenterHorizontally
    ) {
      NavigateButton(
      text = stringResource(id = R.string.label_inspire),
      onClick = { navController.navigate(Screen.Inspire.route) }
      )
      NavigateButton(
      text = stringResource(id = R.string.label_laugh),
      onClick = { navController.navigate(Screen.Laugh.route) }
      )
      NavigateButton(
      text = stringResource(id = R.string.label_play),
      onClick = { navController.navigate(Screen.Play.route) }
      )
    }
  }
}
```

Listing 8: Gumb za navigaciju - NavigateButton.kt

```
@Composable
fun NavigateButton(text: String, onClick: () -> Unit) {
  Button(
  onClick = onClick,
  Modifier.padding(dimensionResource(id = R.dimen.padding_medium)),
  colors = ButtonDefaults.buttonColors(
  backgroundColor = colorResource(id = R.color.rosewater),
  contentColor = colorResource(id = R.color.cream)
  )
  ) {
    Text(text = text, fontSize = 20.sp)
  }
}
```

Kada je sve ostalo pripremljeno, potrebno je sve i povezati u cjelinu i to kreiranjem glavnog ekrana. Za kreiranje glavnog ekrana rabi se stoga funkcija dana izlistanjem 9. U toj funkciji kreirana je komponenta za navigaciju kroz aplikaciju te je navedeno koji ekran je početni, a koji su ekrani na koje je moguće navigirati. Možete primijetiti da se ovdje kreira i postavlja navigacijska komponenta koja je potrebna početnom ekranu aplikacije.

Listing 9: Glavni ekran - MainScreen.kt

```
@Composable
fun MainScreen() {
  val navController = rememberNavController()
  NavHost(navController = navController, startDestination = Screen.Home.route) {
    composable(Screen.Home.route) { HomeScreen(navController) }
    composable(Screen.Laugh.route) { LaughScreen() }
    composable(Screen.Inspire.route) { InspireScreen() }
    composable(Screen.Play.route) { PlayScreen() }
  }
}
```

Preostalo je još samo reći Android sustavu da, kada se pokrene aplikacija, prikaže najprije glavni ekran (ovaj nema sadržaja) koji će onda odmah navigirati ka početnom ekranu. Ovo se postiže izmjenom linije koda u datoteci `MainActivity.kt` prema izlistanju **??** tako da se poziva ranije definirana funkcija `MainScreen()`.

Listing 10: Glavna aktivnost - MainActivity.kt

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

```
4
5       setContent {
6         LaughlyTheme {
7           // A surface container using the 'background' color from the theme
8           Surface(
9           modifier = Modifier.fillMaxSize(),
10          color = MaterialTheme.colors.background
11          ) {
12            MainScreen()
13          }
14        }
15      }
16    }
17  }
```

5 Značajka - inspiracija

Kao prvu aktivnost koju nudimo unutar naše aplikacije, omogućit ćemo korisnicima da se inspiriraju poznatom osobom koja je za nas inspirativna. Na radionici možete samostalno odabrati osobu koju želite, prikupiti potrebne materijale i zatim iskoristiti upute i kod dan na ovim stranicama kako biste aplikaciju učinili osobnijom.

Krenut ćemo s informacijom da je fiksni sadržaj u Android aplikacijama izdvojen u datoteke za specifične oblike sadržaja. Primjerice, tekst koji prikazujete na gumbima, dimenzije margina ili slike koje predstavljaju fiksne grafičke elemente u aplikacijama izdvajaju se izvan koda aplikacije. Razlog je taj što iz brojnih razloga u nekim situacijama trebate drugačiji izgled ili prikaz. Primjerice, korisnici u Njemačkoj očekuju sadržaj na njemačkom jeziku, dok bi stanovnicima Hrvatske to predstavljalo problem i oni očekuju hrvatski jezik. Svaka vrijednost koju kanite rabiti predstavljena je resursom koji ima jedinstveni identifikator. Ovo omogućuje programeru da koristi resurs preko identifikatora, a prepusti sustavu da, ovisno o postavkama sustava, brine o tome koji će se konkretan resurs učitati. Tekst se tako izdvaja u datoteku `strings.xml` koja je smještena u direktorij `res/values`. Cjelokupni izgled ove datoteke dan je izlistanjem **??** i uključuje tekst za cijelu aplikaciju.

Listing 11: Resursi - values/strings.xml

```
1  <resources>
2  <string name="app_name">Laughly</string>
3  <!-- Custom strings: -->
4  <string name="label_inspire">Get inspired!</string>
5  <string name="label_laugh">Laugh!</string>
6  <string name="label_play">Play!</string>
7  <string name="label_quote">Quote</string>
8  <string name="name_turing">Alan Turing</string>
9  <string name="info_turing">"He was the most important mathematician of the second world war. He helped to break the German's Enigma code at
       ↪ Blatchley Park, by designing a computer to decipher the German messages called the Bombe. The most important effects of this
       ↪ revolutionary invention were:\n - Shortening the length of the war: by decoding the German's messages Blatchley Park's team was able
       ↪ to identify the position of all Nazi's boats so the Royal Navy was able to protect English naval fleet from u-boats's attacks.\n -
       ↪ Saving millions lives: the avoided attacks permit a lot of people to stay in life. Turing's team, on the other hand, had to avoid that
       ↪  German discover that Enigma had been decrypted. An electromechanical device that helped the code-breakers to calculate the key of the
       ↪  day the German were using on their Enigma machine.Using a menu provided by the codebreaking team from a crib (plaintext that
       ↪ corresponded to ciphertext), the Bombe operators could quickly set up the machine and let it calculate possible Enigma settings.\n
       ↪ After the war Alan Turing came up with Turing Test, a method to test artificial intelligence.Persecuted for homosexual acts, he
       ↪ committed suicide in 1954.\n"</string>
10 <string name="quote_turing_1">Sometimes it is the people no one imagines anything of who do the things that no one can imagine.</string
       ↪ >
11 <string name="quote_turing_2">A computer would deserve to be called intelligent if it could deceive a human into believing that it was
       ↪ human.</string>
12 <string name="quote_turing_3">Unless in communicating with it one says exactly what one means, trouble is bound to result.</string>
13 <string name="label_reroll">Re-roll</string>
14 <string name="description_joke_icon">New joke icon</string>
15 <string name="label_refresh_joke">Get new joke!</string>
16 </resources>
```

Sadržaj koji će biti prikazan u Android aplikacijama uobičajeno se ne definira unutar ekrana koji definira kako će sadržaj izgledati (razdvajanje prezentacije od implementacije). Za potrebe razdvajanja ovog sadržaja, ali i ostvarivanje brojnih drugih prednosti, koriste se komponente koje se nazivaju

`ViewModel`. Tijekom radionice kreirat ćemo odgovarajuće `ViewModel` komponente za svaki ekran aplikacije. Za `InspireScreen` taj je `ViewModel` dan izlistanjem `??`. Može se uočiti kako taj `ViewModel` referencira resurse koje ćemo unutar ekrana iskoristiti za prikaz sadržaja te omogućuje da se dohvati resurs nasumičnog citata u bilo kojem trenutku kroz funkciju `getQuote()`.

Listing 12: Inspirirajući `view model`- InspireViewModel.kt

```
class InspireViewModel : ViewModel() {

  val name = R.string.name_turing
  val info = R.string.info_turing
  val image = R.drawable.turing
  private val quotes = listOf<Int>(
    R.string.quote_turing_1,
    R.string.quote_turing_2,
    R.string.quote_turing_3
  )

  fun getQuote() = quotes.random()
}
```

Jednom kada je `ViewModel` definiran, pristupamo definiranju ekrana koji ga rabi. U ovom slučaju je riječ o inspirirajućem ekranu `InspireScreen`. Inspire screen sastoji se od nekolicine komponenti koje omogućuju prikaz svog sadržaja smještenog unutar ranije danog `ViewModel`-a. Sam `ViewModel` ubacuje se u ekran svaki put kada ga je potrebno prikazati, a za te potrebe rabi se funkcija `viewModel()`.

Listing 13: Inspirirajući ekran - InspireScreen.kt

```
@Composable
fun InspireScreen(viewModel: InspireViewModel = viewModel()) {
  Column(
    modifier = Modifier.fillMaxSize()
    .background(colorResource(id = R.color.cream))
    .padding(dimensionResource(id = R.dimen.padding_medium))
  ) {
    Row(
      verticalAlignment = Alignment.CenterVertically,
      horizontalArrangement = Arrangement.Center,
      modifier = Modifier.weight(1f)
    ) {
      Image(
        painter = painterResource(id = viewModel.image),
        contentDescription = stringResource(id = viewModel.name),
        contentScale = ContentScale.Crop,
        modifier = Modifier.size(100.dp, 100.dp).clip(CircleShape)
      )
      Text(
        text = stringResource(viewModel.name),
        fontSize = 48.sp,
        textAlign = TextAlign.Center,
        color = colorResource(id = R.color.rosewater),
        modifier = Modifier.fillMaxWidth()
      )
    }
    Text(
      text = stringResource(id = viewModel.info),
      fontSize = 14.sp,
      modifier = Modifier
      .weight(3f)
      .fillMaxWidth()
      .padding(dimensionResource(id = R.dimen.padding_medium))
    )
    Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      verticalArrangement = Arrangement.SpaceEvenly,
      modifier = Modifier.weight(1f)
    ) {
      var quoteResourceId by remember { mutableStateOf(viewModel.getQuote()) }
      Text(
        text = stringResource(id = quoteResourceId),
        fontSize = 16.sp,
        textAlign = TextAlign.Center,
        color = colorResource(id = R.color.hotpink),
        modifier = Modifier.fillMaxWidth()
      )
      Button(onClick = { quoteResourceId = viewModel.getQuote() }) {
        Text(text = stringResource(R.string.label_quote))
      }
```

```
    }
  }
}
```

6 Značajka - igra

Kao drugu aktivnost koju nudimo unutar aplikacije, omogućit ćemo korisnicima da bacaju kockice za Jamb. Kako bismo ostvarili ovu funkcionalnost, dizajnirat ćemo jedan ekran aplikacije tako da prikazuje šest kockica za Jamb te mu dodati gumb ispod kockica klikom na koji će se izgenerirati slučajni brojevi koji će biti prikazani u obliku sličica kockica. Kasnije ćemo dodati funkcionalnost da se promjenom vrijednosti na senzoru akceleracije, primjerice trešnjom uređaja, okine bacanje kockica. Kao prvi korak, preuzet ćemo sličice kockica s Interneta i postaviti ih u mapu `res/drawable` i to tako da ime svake kockice bude slovo `d` popraćeno brojem koji kockica prikazuje. Kao idući korak, kreirat ćemo jednu komponentu sučelja koja će prikazivati šest kockica, prema izlistanju 14. Ovu komponentu smjestit ćemo u paket `ui/components` gdje držimo sve zasebne komponente koje smo samostalno kreirali. Uočite u izlistanju koda da smo definirali posebnu funkciju koja zna prevesti iz broja koji se treba prikazati na kockici u identifikator sličice koja predstavlja taj broj. Ovo smo napravili kako tu logiku ne bismo morali ponavljati svuda gdje se koristi kockica.

Listing 14: Komponenta kockice - Dice.kt

```
@Composable
fun Dice(values: List<Int>) {
  Column {
    Row(
    horizontalArrangement = Arrangement.SpaceEvenly,
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier.fillMaxWidth()
    ) {
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[0])),
      contentDescription = "Icon 1"
      )
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[1])),
      contentDescription = "Icon 1"
      )
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[2])),
      contentDescription = "Icon 1"
      )
    }
    Spacer(modifier = Modifier.size(16.dp))
    Row(
    horizontalArrangement = Arrangement.SpaceEvenly,
    verticalAlignment = Alignment.CenterVertically,
    modifier = Modifier.fillMaxWidth()
    ) {
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[3])),
      contentDescription = "Icon 1"
      )
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[4])),
      contentDescription = "Icon 1"
      )
      Image(
      painter = painterResource(id = GetDiceDrawableId(values[5])),
      contentDescription = "Icon 1"
      )
    }
  }
}
fun GetDiceDrawableId(value: Int): Int {
  return when (value) {
    1 -> R.drawable.d1
    2 -> R.drawable.d2
    3 -> R.drawable.d3
    4 -> R.drawable.d4
    5 -> R.drawable.d5
```

```
50      6 -> R.drawable.d6
51      else -> R.drawable.d1
52    }
53  }
```

Kada smo pripremili prikaz za kockice, dodat ćemo tu komponentu na ekran, a dodat ćemo i ranije spomenuti gumb.

Listing 15: Ekran za igru - PlayScreen.kt

```
1  @Composable
2  fun PlayScreen(viewModel: PlayViewModel = viewModel()) {
3    val values by viewModel.numbers.collectAsState()
4
5    Column(
6    verticalArrangement = Arrangement.SpaceEvenly,
7    horizontalAlignment = Alignment.CenterHorizontally,
8    modifier = Modifier
9    .fillMaxSize()
10   .background(colorResource(id = R.color.cream))
11   .padding(dimensionResource(id = R.dimen.padding_medium))
12   ) {
13     Dice(values)
14     Button(onClick = { viewModel.randomize() }) {
15       Icon(
16       imageVector = Icons.Default.Refresh,
17       contentDescription = stringResource(R.string.label_reroll)
18       )
19       Text(text = stringResource(id = R.string.label_reroll))
20     }
21   }
22 }
```

Kao i u ranijem slučaju ekrana za inspiraciju, definirat ćemo posebnu klasu koja će služiti za pohranu podataka i pisanje logike iza prikaza, a koja se naziva modelom pogleda (engl. *view model*). Već je sada uočljivo kako imamo listu brojeva koju prikazujemo, kao i da postoji funkcionalnost za ponovno bacanje kockica, odnosno generiranje nasumičnih vrijednosti. Ova funkcionalnost vezana je uz klik gumba. Dodatna funkcionalnost vezana uz senzore na Android platformi ostvaruje se tako da se koristi upravitelj senzorima i preko njega dođe do željenog objekta klase `Sensor`. Jednom kada je senzor dohvaćen (ako postoji). moguće je prijaviti alat za osluškivanje promjena vrijednosti koje taj senzor mjeri.

Listing 16: Model pogleda za igru - PlayViewModel.kt

```
1  class PlayViewModel : ViewModel() {
2    val numbers = MutableStateFlow(listOf(1, 1, 1, 1, 1, 1))
3    private val context = Laughly.application
4    private val sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
5    private val accelerationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
6    private val accelerationThreshold = 12
7    private val sensorEventListener = object : SensorEventListener {
8      override fun onSensorChanged(event: SensorEvent?) {
9        event?.let {
10         val total = event.values.map { it * it }.sum()
11         if (sqrt(total) > accelerationThreshold) {
12           randomize()
13         }
14       }
15     }
16     override fun onAccuracyChanged(p0: Sensor?, p1: Int) {}
17   }
18   init {
19     if (context.packageManager.hasSystemFeature(PackageManager.FEATURE_SENSOR_ACCELEROMETER)) {
20       accelerationSensor?.let {
21         sensorManager.registerListener(
22         sensorEventListener,
23         accelerationSensor,
24         SensorManager.SENSOR_DELAY_UI
25         )
26       }
27     }
28   }
29   fun randomize() {
30     val rolled = mutableListOf<Int>()
```

```
31      repeat(6) {
32        rolled.add(Random.nextInt(1, 7))
33      }
34      numbers.value = rolled
35    }
36  }
```

Još je jedan detalj preostao kako bi omogućili rad sa senzorima, a to je definirati klasu koja predstavlja aplikaciju. Ovu klasu nazvat ćemo `Laughly` i u nju smjestiti kod prikazan izlistanjem 17. Ovo je važno jer kad god se pristupa sustavskim uslugama na Android platforme, to se ostvaruje preko nečeg što se naziva kontekts. Riječ je o objektu koji zna sve o trenutnoj okolini aplikacije koja se izvodi i pruža pristupne točke prema sustavskim uslugama. Osim kreiranja klase `Laughly`, u datoteku `AndroidManifest.xml` potrebno je dodati liniju koda kako bismo tu novu klasu uistinu registrirali kao aplikacijsku klasu. To se postiže dodavanjem atributa `name` unutar oznake `application` u manifest datoteci.

Listing 17: Aplikacijska klasa - Laughly.kt

```
1  class Laughly : Application() {
2    companion object {
3      lateinit var application: Application
4    }
5    override fun onCreate() {
6      super.onCreate()
7      application = this
8    }
9  }
```

Listing 18: Manifest datoteka - AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    xmlns:tools="http://schemas.android.com/tools">
4    <uses-permission android:name="android.permission.INTERNET" />
5    <application
6      android:name=".Laughly"
7      <!-- ... -->
```

7 Značajka - šala

Konačna funkcionalnost aplikacije bit će nasmijavanje korisnika. Ovo ćemo postići tako da korisnik ode na ekran za smijanje, klikne gumb i time dobije prikaz nasumične šale. Bit će riječ o jednostavnim šalama, na engleskom jeziku, koje se sastoje od dva dijela - postavke (engl. *setup*) i vica (engl. *punchline*). Kako ne bismo sami sastavljali brojne viceve, poslužit ćemo se servisom koji je dostupan na Internetu i od kojeg ćemo zatražiti da nam da novi vic, svaki put kada mi pošaljemo upit prema njemu. Upit ćemo izvesti putem protokola HTTP, a odgovor ćemo dobiti zapakiran u oblik JSON. Ove Vas kratice ne trebaju pretjerano brinuti, riječ je o standardiziranim načinma i formatima komunikacije između aplikacija. Sam servis smješten je na adresi `https://official-joke-api.appspot.com/random_joke`. Možete pokušati na navedenu adresu otići uporabom internetskog preglednika i primijetit ćete da svaki put kada osvježite stranicu, dobijete novu šalu. S obzirom da uvodimo pojmove poput šale i servisa za šale, krenut ćemo implementirati s te strane. Naučit ćemo naš programski jezik te pojmove tako što ćemo mu ih opisati korištenjem klasa. U paket `laughing` dodat ćemo dvije nove Kotlin klase, `Joke` i `JokeApiService`.

Listing 19: Klasa Joke - Joke.kt

```
1  data class Joke(
2    val setup: String,
```

```
  val punchline: String
)
```

Listing 20: Klasa JokeApiService - JokeApiService.kt

```
class JokesApiService {
  private val client = HttpClient(Android) {
    install(DefaultRequest) {
      headers.append("Content-Type", "application/json")
    }
    install(ContentNegotiation) {
      gson {
        setPrettyPrinting()
      }
    }
    engine {
      connectTimeout = 60_000
      socketTimeout = 60_000
    }
  }
  suspend fun getJoke(): Joke {
    return client.get("https://official-joke-api.appspot.com/random_joke").body()
  }
}
```

Prikaz šale na ekranu nije kompliciran, zapravo se sastoji od dva tekstualna elementa i jednog gumba koji osvježava prikaz postavke i vica.

Listing 21: Ekran šale - LaughScreen.kt

```
@Composable
fun LaughScreen(viewModel: LaughViewModel = viewModel()) {
  Column(
  verticalArrangement = Arrangement.SpaceEvenly,
  horizontalAlignment = Alignment.CenterHorizontally,
  modifier = Modifier
  .fillMaxSize()
  .background(colorResource(id = R.color.cream))
  .padding(dimensionResource(id = R.dimen.padding_medium))
  ) {
    val joke by viewModel.joke.collectAsState()
    Text(
    text = joke.setup,
    fontSize = 32.sp,
    textAlign = TextAlign.Center
    )
    Text(
    text = joke.punchline,
    fontSize = 32.sp,
    textAlign = TextAlign.Center
    )
    Button(onClick = { viewModel.updateJoke() }) {
      Image(
      painterResource(id = android.R.drawable.ic_menu_more),
      contentDescription = stringResource(R.string.description_joke_icon),
      modifier = Modifier.size(20.dp)
      )
      Text(
      text = stringResource(R.string.label_refresh_joke),
      Modifier.padding(start = 10.dp)
      )
    }
  }
}
```

Preostalo je još samo definirati model prikaza za dani ekran unutar kojeg će biti sačuvana trenutna šala i omogućeno dohvaćanje nove šale.

Listing 22: Model prikaza šale - LaughViewModel.kt

```
class LaughViewModel : ViewModel() {
  private val apiService = JokesApiService()
  val joke = MutableStateFlow(Joke("", ""))
  init {
    updateJoke()
  }
  fun updateJoke() {
    viewModelScope.launch {
```

```
9           joke.value = apiService.getJoke()
10        }
11    }
12 }
```

8 Što dalje?

- Android app fundamentals: `https://developer.android.com/courses/fundamentals-training/toc-v2`
- Android kolegij `https://www.udacity.com/course/android-kotlin-developer-nanodegree--nd940yout`
- Besplatan kolegij `https://www.youtube.com/watch?v=fis26HvvDII`