Mobilne Aplikacije

Nositelj: doc. dr. sc. Nikola Tanković
Izvođač: dr. sc. Robert Šajina
Asistent: mag. inf. Alesandro Žužić

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli

Fakultet informatike u Puli

[7] - Fragments, Navigation & ViewModel

Posljednje ažurirano: 14. siječnja 2026.

Sadržaj

Fragmenti

`Fragment` predstavlja modularni dio korisničkog sučelja koji se može koristiti u jednoj ili više aktivnosti. Za razliku od aktivnosti, fragmenti nemaju vlastit prozor, već se uvijek prikazuju unutar aktivnosti. Fragmenti omogućuju bolju kontrolu nad korisničkim sučeljem i fleksibilnost pri dizajniranju aplikacije, posebno na tablet uređajima gdje možemo prikazati više fragmenata istovremeno.

Omogućuju:

- **Modularnost** – odvajanje logike korisničkog sučelja u manje, ponovno iskoristive dijelove
- **Fleksibilnost** – mogućnost zamjene ili dodavanja fragmenata tijekom izvođenja aplikacije
- **Prilagodba** – jednostavnije prilagođavanje sučelja različitim veličinama ekrana
- **Ponovna upotreba** – isti fragment može se koristiti u različitim aktivnostima

> Mogu se dodavati, uklanjati ili zamjenjivati tijekom izvođenja aplikacije.

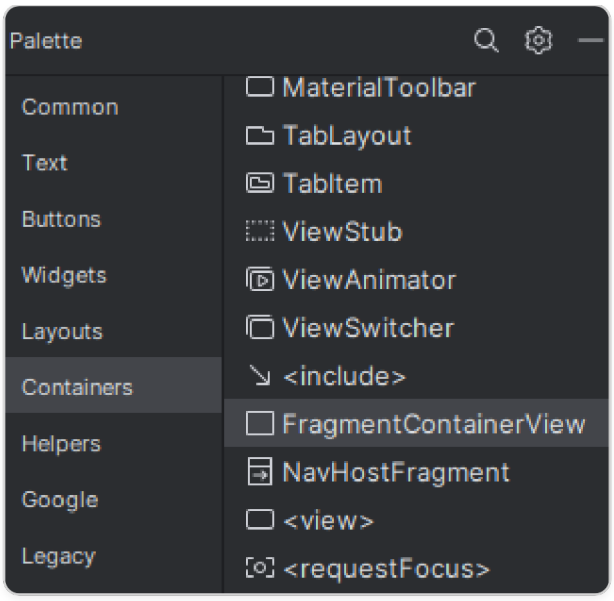
Da bi se koristili fragmenti, potrebno je definirati njihov izgled u XML datoteci i implementirati njihovu logiku u odgovarajućoj klasi.

Napravit ćemo prazni projekt s jednom aktivnošću `MainActivity`.

U `MainActivity.xml` datoteci definiramo layout `FrameLayout` za fragmente:

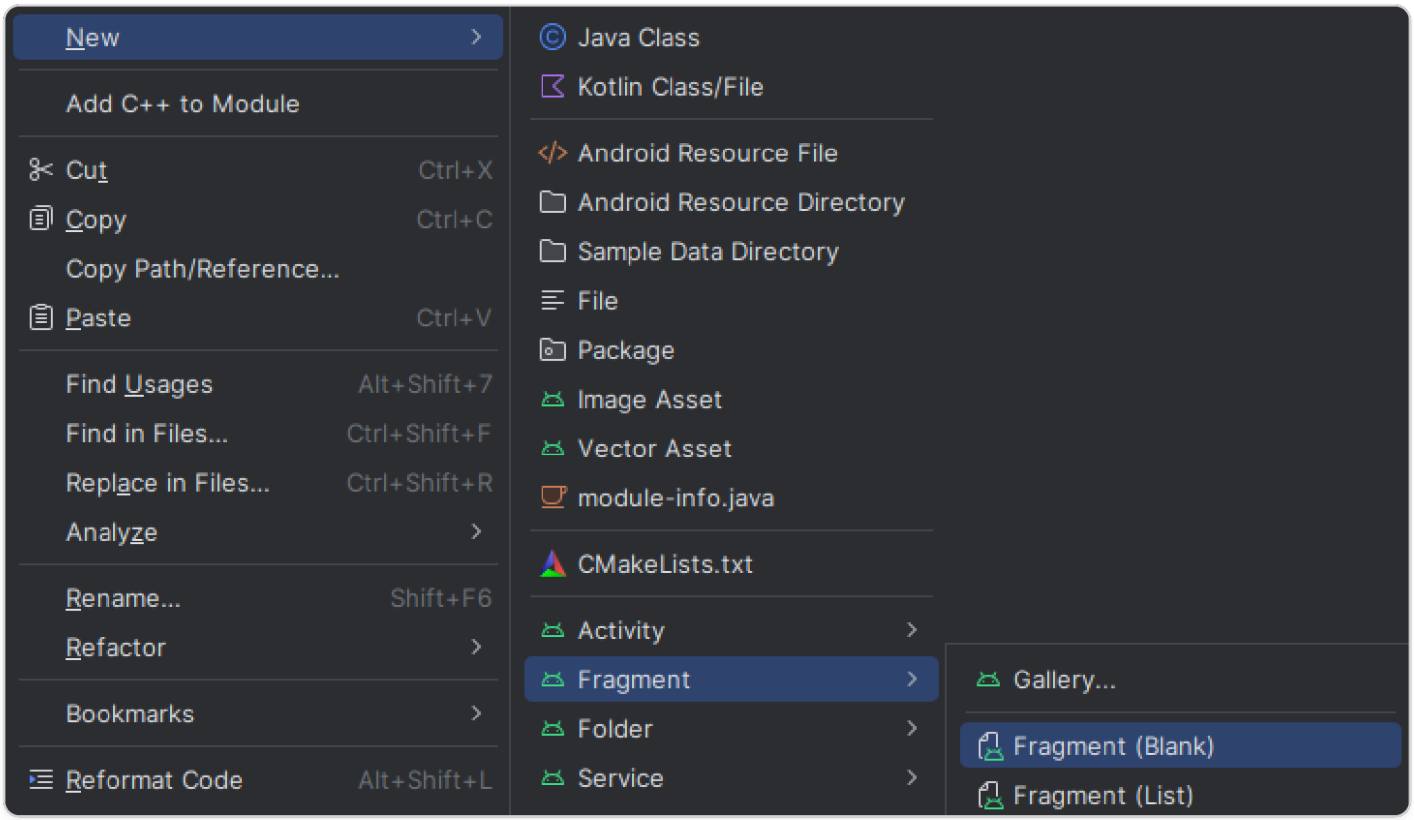
```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/bottom_nav"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

- Možemo ga dodati i nakon izrade fragmenta koristeći paletu **Design**



Dodavanje FrameLayout spremnika za fragmente

- `FrameLayout` s `id="@+id/fragment_container"` služi kao **spremnik** u koji će se učitavati fragmenti tijekom izvođenja te je odabran jer omogućuje prikaz samo jednog fragmenta istovremeno



Izrada novog praznog fragmenta

Zatim kreiramo prvi fragment `HomeFragment` s pripadajućim layoutom `fragment_home.xml`:

Da bi kreirali fragment u Android Studiju, desni klik na `java` direktorij unutar `app/src/main`, zatim **New > Fragment > Fragment (Blank)**.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".HomeFragment">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="home" />

</FrameLayout>
```

`HomeFragment.java` ima puno predefiniranog koda, no najvažniji dio je `onCreateView()` metoda i `HomeFragment()` konstruktor:

```
    public class HomeFragment extends Fragment {

        public HomeFragment() {
            // Required empty public constructor
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
            View view = inflater.inflate(R.layout.fragment_home, container,
false);

            return view;
        }
    }
```

- Svaki fragment mora imati **javni konstruktor bez parametara**
- `onCreateView()` metoda vraća View koji predstavlja korisničko sučelje fragmenta
- `LayoutInflater.inflate()` učitava XML layout datoteku i pretvara je u View objekt

U `MainActivity.java` učitavamo `HomeFragment` unutar `onCreate()` metode:

```
getSupportFragmentManager().beginTransaction()
    .replace(R.id.fragment_container, new HomeFragment())
    .commit();
```

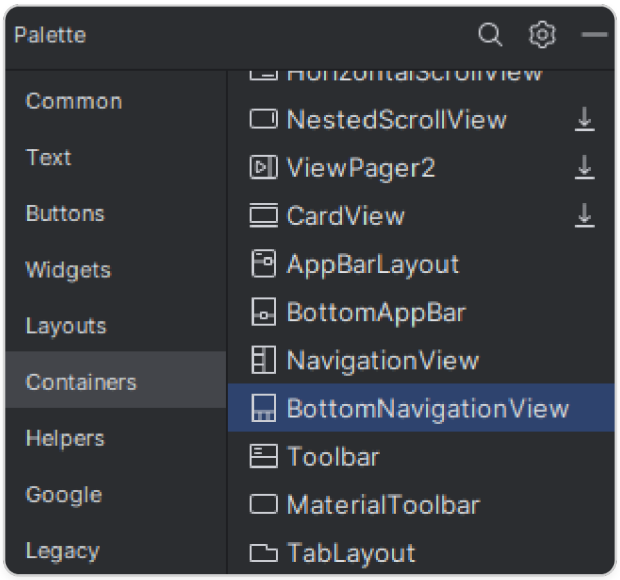
- `getSupportFragmentManager()` vraća `FragmentManager` koji upravlja fragmentima unutar aktivnosti
- `.beginTransaction()` počinje transakciju za dodavanje, uklanjanje ili zamjenu fragmenata
- `.replace(R.id.fragment_container, new HomeFragment())` zamjenjuje fragment koji se nalazi u spremniku s novim fragmentom
- `.commit()` potvrđuje transakciju i primjenjuje je

Navigacija između fragmenata

Navigacija između fragmenata može se postići na nekoliko načina:

1. **BottomNavigationView** – metoda s ikonama na dnu zaslona
2. **Side Navigation Drawer** – metoda s izbornikom koji se pojavljuje s lijeve strane

Koristit ćemo `BottomNavigationView` tako ćemo stvoriti dodatne fragmente `BooksFragment` i `SettingsFragment`.



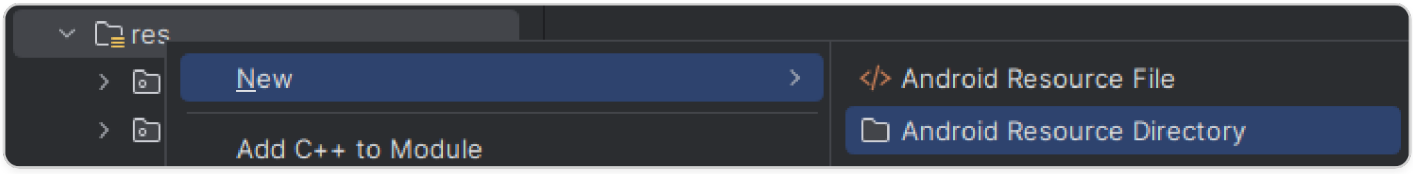
Dodavanje BottomNavigationView za navigaciju

U `activity_main.xml` dodajemo `BottomNavigationView` za navigaciju između fragmenata:

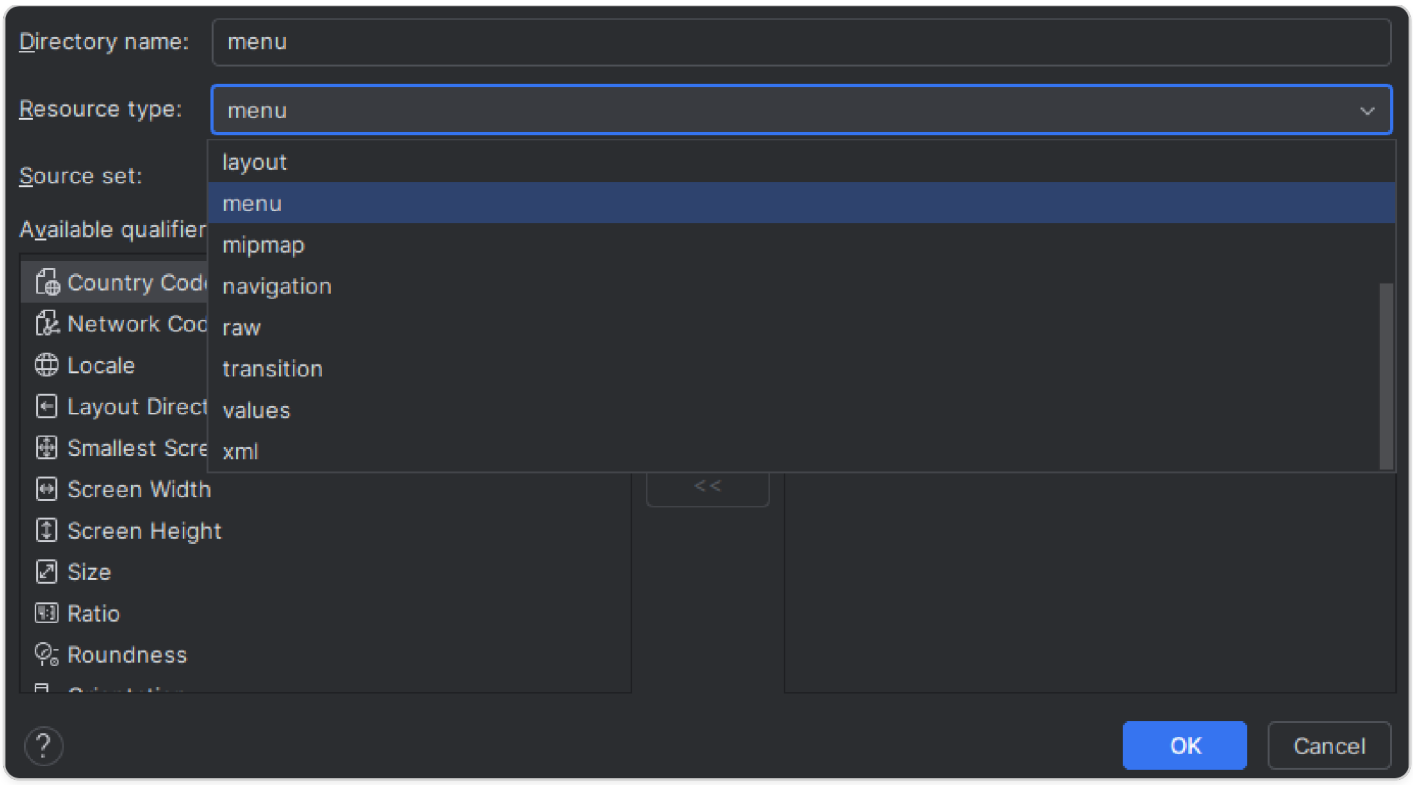
```
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottom_nav"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:menu="@menu/bottom_menu"/>
```

- `app:menu="@menu/bottom_menu"` povezuje izbornik definiran u XML datoteci s `BottomNavigationView`
- trenutačno ne postoji `bottom_menu.xml`, pa ga moramo kreirati

Stvorit ćemo direktorij `menu` unutar `res` direktorija ako već ne postoji.

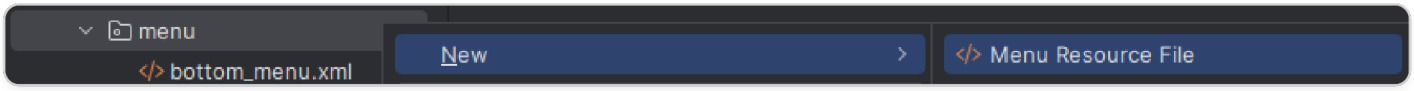


Izrada menu direktorija



Resource Type: menu

Zatim kreiramo novu `Menu Resource File` pod nazivom `bottom_menu.xml` unutar `res/menu` direktorija.

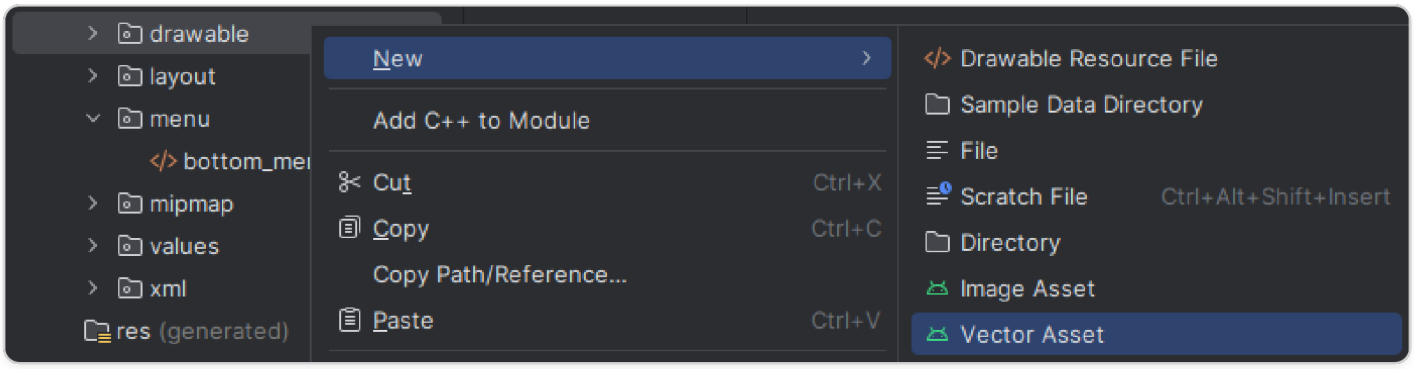


Izrada bottom_menu.xml datoteke

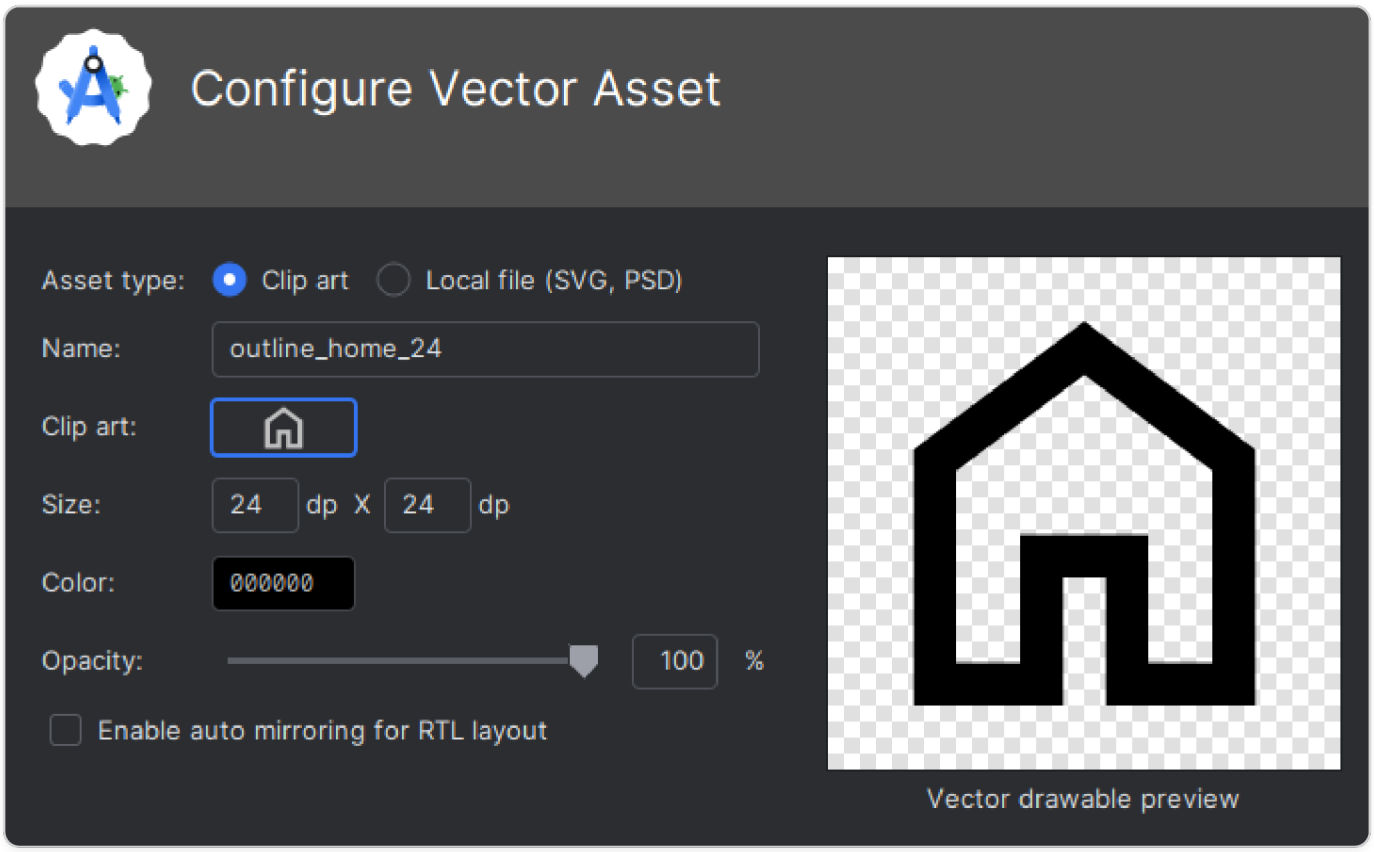
U `bottom_menu.xml` definiramo stavke izbornika tako da svaka stavka predstavlja jedan fragment. Dodajemo tri stavke: Books, Home i Settings koristeći `<item>` elemente ili putem **Design** prikaza `Menu Item`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/nav_books"
        android:icon="@drawable/outline_book_2_24"
        android:title="Books" />
    <item
        android:id="@+id/nav_home"
        android:icon="@drawable/outline_home_24"
        android:title="Home" />
    <item
        android:id="@+id/nav_settings"
        android:icon="@drawable/outline_apps_24"
        android:title="Settings" />
</menu>
```

Da bismo dodali ikone za stavke izbornika, koristimo **Vector Asset** alat u Android Studiju. Ikone stvaramo tako da desni klik na `drawable` direktorij unutar `res`, zatim **New > Vector Asset** i odabiremo željene ikone iz Material Design biblioteke.

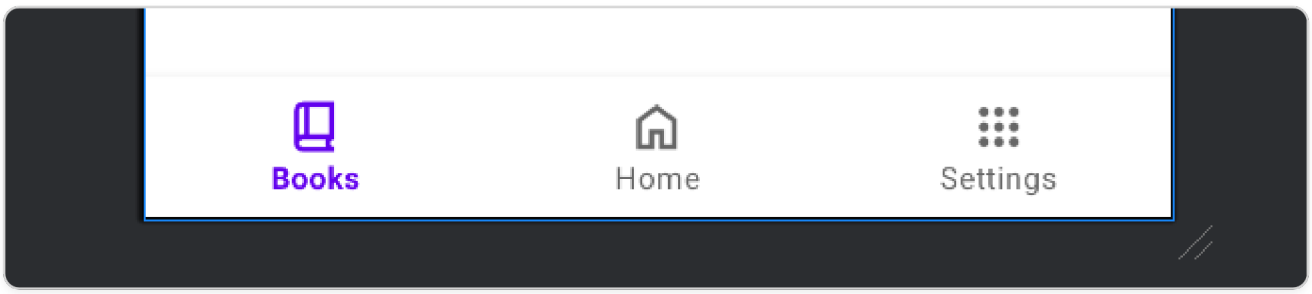


Stvaranje Vector Asset ikona



Odabir ikona iz Material Design biblioteke

Sada `BottomNavigationView` ima tri stavke izbornika s pripadajućim ikonama koje možemo vidjeti u **Design** prikazu `activity_main.xml` datoteke:



Prikaz BottomNavigationView u Design prikazu

Sada ćemo u MainActivity postaviti listener za odabir stavki izbornika:

```
BottomNavigationView bottomNav = findViewById(R.id.bottom_nav);

bottomNav.setOnItemSelectedListener(item -> {
    Fragment fragment = new BooksFragment();

    if (item.getItemId() == R.id.nav_home)
        fragment = new HomeFragment();
    else if (item.getItemId() == R.id.nav_settings)
        fragment = new SettingsFragment();

    getSupportFragmentManager()
        .beginTransaction()
        .replace(R.id.fragment_container, fragment)
        .addToBackStack(null)
        .commit();

    return true;
});
```

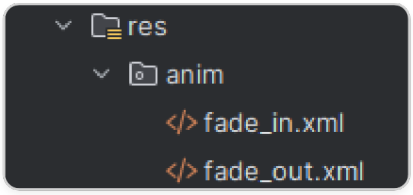
- `setOnItemSelectedListener()` postavlja **listener** koji se poziva kada korisnik odabere stavku u izborniku
- `item.getItemId()` vraća ID odabrane stavke koju možemo usporediti s ID-jevima iz `bottom_menu.xml`
- Ovisno o odabranoj stavki, kreiramo odgovarajući fragment
- `addToBackStack(null)` dodaje transakciju u back stack, što omogućuje vraćanje na prethodni fragment pritiskom na `back` gumb
- Vraćamo `true` kako bi označili da je stavka obrađena

Za dodavanje animacija prilikom navigacije između fragmenata, možemo koristiti `setCustomAnimations` metodu:

```
.setCustomAnimations(
    R.anim.fade_in,  // enter
    R.anim.fade_out, // exit
    R.anim.fade_in,  // popEnter
    R.anim.fade_out  // popExit
)
```

> Potrebno je napomenuti da `setCustomAnimations` treba pozvati prije `.replace()` metode u transakciji.

Za nju je potrebno je definirati animacije u `res/anim` direktoriju, ako ga nema potrebno ga je kreirati.



Animacije za tranziciju fragmenata

Fade In:

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromAlpha="0.0"
    android:toAlpha="1.0"
    android:duration="300"/>
```

Fade Out:

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromAlpha="1.0"
    android:toAlpha="0.0"
    android:duration="300"/>
```

ViewModel

`ViewModel` je komponenta čija je primarna svrha **čuvanje i upravljanje podacima vezanima uz korisničko sučelje** na način koji je otporan na promjene konfiguracije uređaja, poput rotacije zaslona ili promjene jezika.

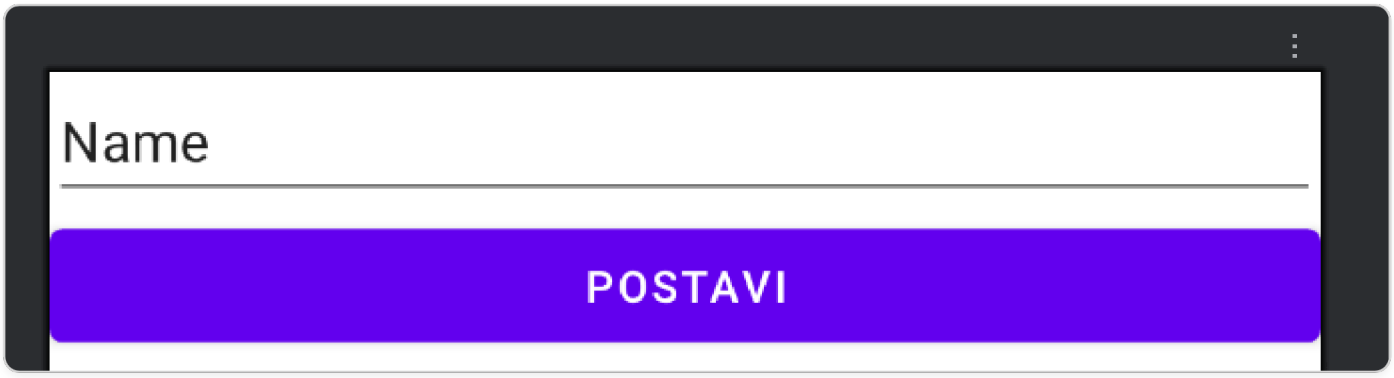
- **Perzistentan tijekom rotacije zaslona** – obično se aktivnost ponovno kreira pri rotaciji, ali ViewModel ostaje isti
- **Dijeljenje podataka** – više fragmenata unutar iste aktivnosti može dijeliti isti ViewModel
- **Odvajanje logike** – odvojena je logika obrade podataka od korisničkog sučelja

U primjeru kreiramo `SharedViewModel` koji će biti dostupan svim fragmentima:

```
public class SharedViewModel extends ViewModel {

    private final MutableLiveData<String> selectedItem = new MutableLiveData<>
();

    public void selectItem(String item) {
        selectedItem.setValue(item);
    }

    public LiveData<String> getSelectedItem() {
        return selectedItem;
    }
}
```

- `MutableLiveData<String>` je svojstvo koje se može mijenjati i promatrati
- `LiveData` automatski obavještava promatrače kada se podaci promijene
- `selectItem()` postavlja novu vrijednost koju aktivni fragment može koristiti i promatrati
- `getSelectedItem()` vraća `LiveData` koji fragmenti mogu promatrati

Proširit ćemo `BooksFragment` kako bi omogućio korisniku unos podataka koji će biti pohranjeni u `SharedViewModel` i dugme za postavljanje, te `HomeFragment` koji će prikazivati novopostavljeni unos.



Prošireni fragment_books.xml

Korištenje `SharedViewModel` u `OnCreateView` metodi u `BooksFragment.java`:

```
SharedViewModel viewModel = new
ViewModelProvider(requireActivity()).get(SharedViewModel.class);

Button postaviBtn = view.findViewById(R.id.postaviBtn);
postaviBtn.setOnClickListener(v -> {
    EditText editText = view.findViewById(R.id.inputKnjiga);
    String input = editText.getText().toString();
    viewModel.selectItem(input);
});
```

- `new ViewModelProvider(requireActivity())` kreira instancu ViewModelProvider-a vezanu uz aktivnost
- `.get(SharedViewModel.class)` dohvaća `SharedViewModel` instancu
- `requireActivity()` vraća aktivnost koja sadrži ovaj fragment jer oba fragmenta trebaju istu ViewModel instancu
- `viewModel.selectItem(input)` postavlja novu vrijednost kroz ViewModel koja će biti dostupna ostalim fragmentima

Korištenje `SharedViewModel` u `OnCreateView` metodi u `HomeFragment.java`:

```
SharedViewModel viewModel = new
ViewModelProvider(requireActivity()).get(SharedViewModel.class);

viewModel.getSelectedItem().observe(getViewLifecycleOwner(), item -> {
    TextView textView = view.findViewById(R.id.textView);
    textView.setText(item);
});
```

- `viewModel.getSelectedItem()` vraća `LiveData` objekt koji možemo promatrati
- `.observe(getViewLifecycleOwner(), item -> {...})` registrira **observer** koji će biti pozvan kada se podaci u `LiveData` promijene

Samostalni zadatak za vježbu

Napraviti aplikaciju koja omogućuje pregled/dodavanje knjiga i označavanje kao omiljena. Aplikacija treba koristiti više fragmenata za različite dijelove funkcionalnosti, a podaci trebaju biti dijeljeni između fragmenata putem `ViewModel`-a.

Klasa knjige:

Treba napraviti klasu `Book` koja predstavlja jednu knjigu. Klasa treba sadržavati:

- Svojstva: `title` i `author`
- Konstruktor koji prima naslov i autora
- Getter metode

Adapter za RecyclerView:

Treba napraviti `BookAdapter` koji će:

- Sadržavati listu knjiga
- koristiti `book_item.xml` layout za prikaz naslova i autora

ViewModel (`BookViewModel`):

Treba napraviti `BookViewModel` koji će:

- Sadržavati `MutableLiveData<List<Book>>` listu svih knjiga
- Sadržavati `MutableLiveData<List<Book>>` listu omiljenih knjiga
- Implementirati metode: `addBook(Book book)`, `addFavorite(Book book)`
- Getter metode

Fragmenti:

1. **HomeFragment**
    - Prikazuje **nasumičnu knjigu dana** (naslov + autor)
    - Dugme "Sljedeća" prikazuje novu nasumičnu knjigu
    - Dugme "Dodaj u omiljene" dodaje knjigu u omiljene
2. **BooksFragment**
    - Prikazuje **listu knjiga**
    - Mogućnost dodavanja nove knjige
3. **FavoritesFragment**
    - Prikazuje sve knjige koje je korisnik označio kao **omiljene**.
    - Dugme “Dodaj u omiljene” dostupno je u HomeFragment.

BottomNavigationView:

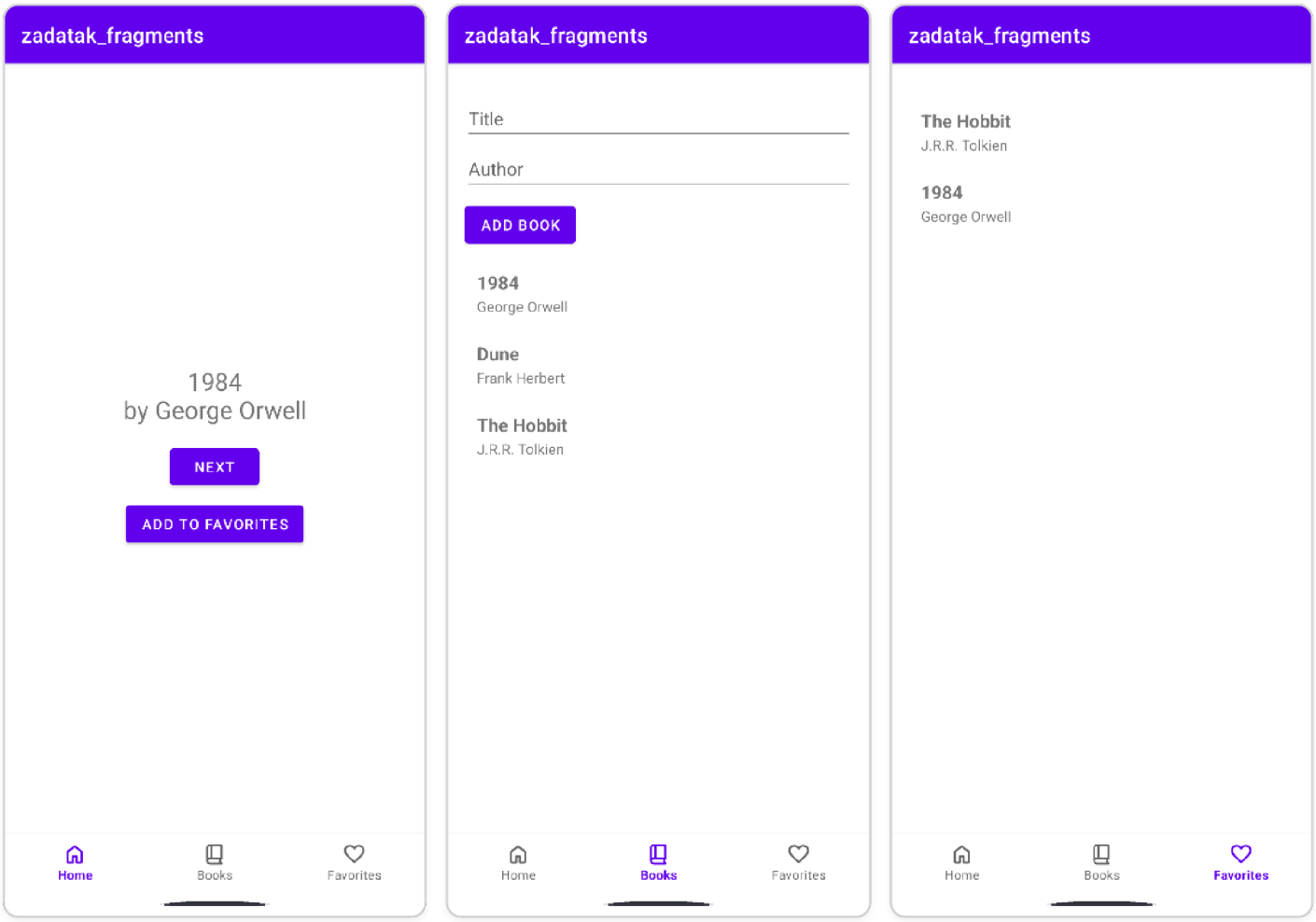
- Home / Books / Favorites
- Home je default fragment

Dodatno:

- Svi fragmenti trebaju **koristiti ViewModel** za dijeljenje podataka između njih
- Knjige se mogu spremati u **lokalni popis u ViewModel-u**, nije potreban database

Inicijalni podaci:

```
new Book("1984", "George Orwell");
new Book("Dune", "Frank Herbert");
new Book("The Hobbit", "J.R.R. Tolkien");
```



Primjer rješenja zadatka