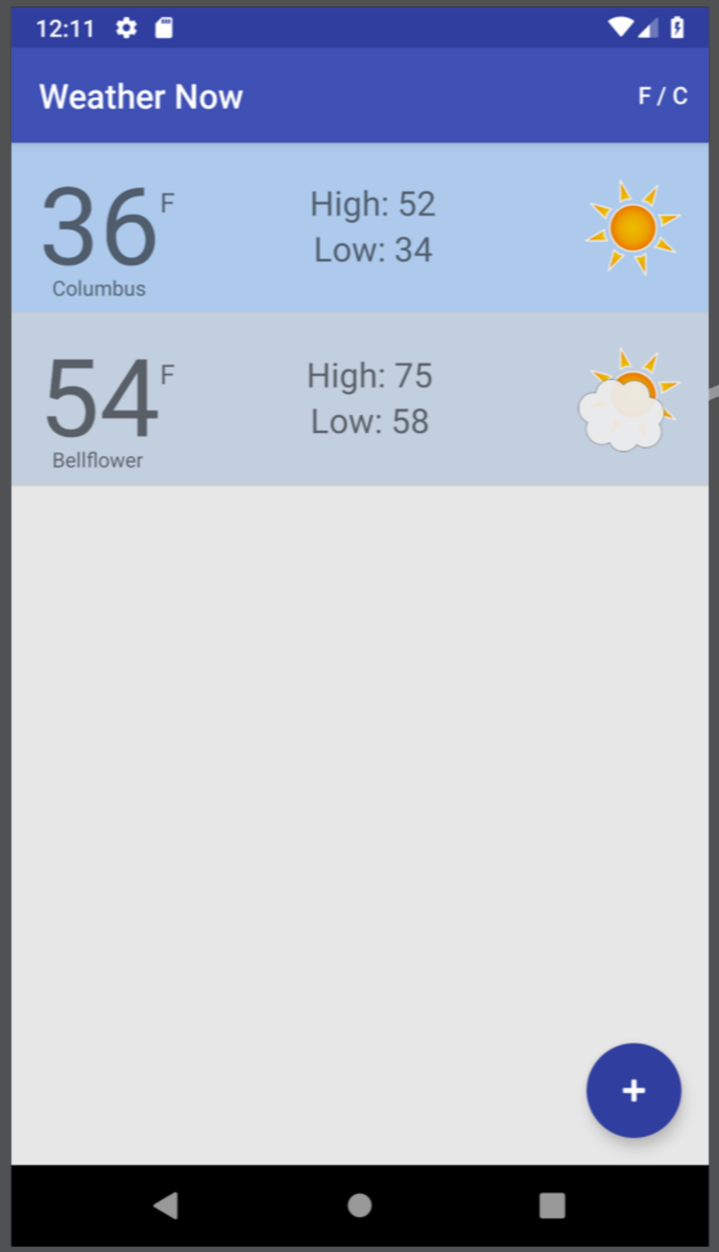
# Destination Navigation

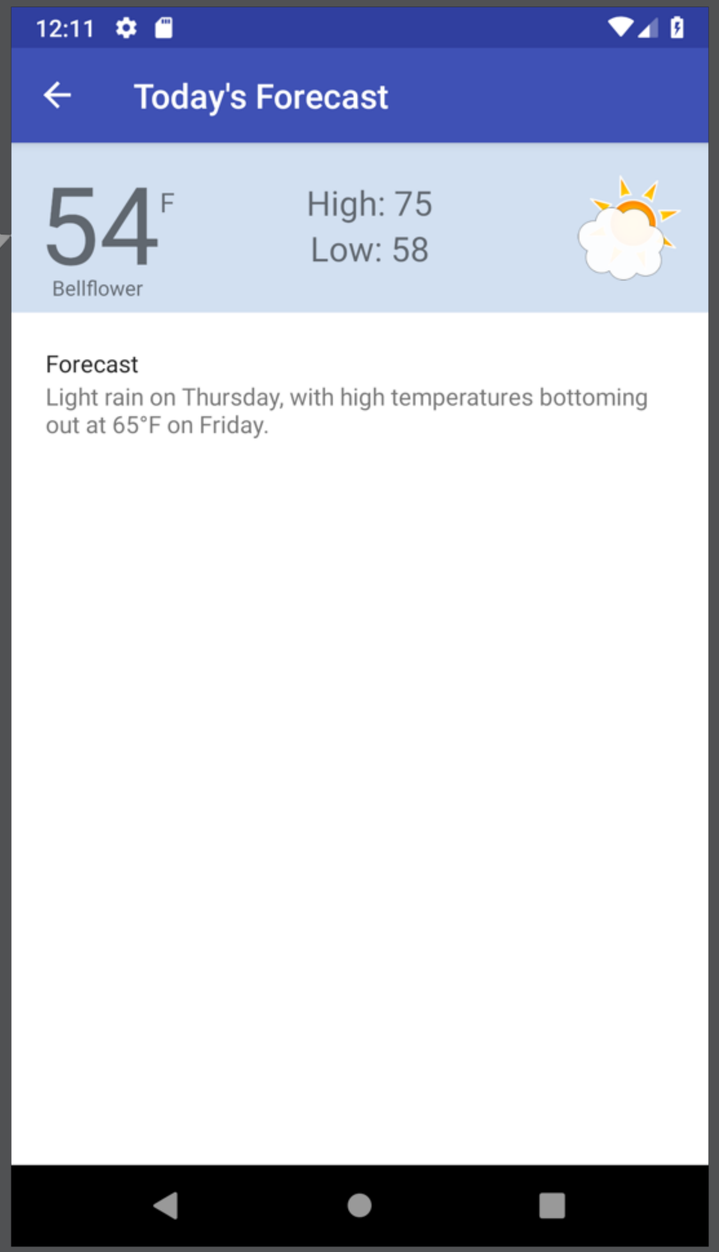

Eric Maxwell
@emmax

# Motivations

# Simplify + Standardize Navigation

- Easy to Follow Best Practices
- Argument Type Safety / Consistency
- Simplified Deep Linking
- Higher Level of Abstraction


12:11
Weather Now
F / C
36 F
Columbus
High: 52
Low: 34
54 F
Bellflower
High: 75
Low: 58
+


12:11
Today's Forecast
54 F
Bellflower
High: 75
Low: 58
Forecast
Light rain on Thursday, with high temperatures bottoming out at 65°F on Friday.

# Pre Navigation

## Navigate to Activity

```
val intent = Intent(this,
   DetailedWeatherForecast::class.java)

intent.putExtra("zip_code", city.zipCode)
intent.putExtra("use_fahrenheit", true)

startActivity(intent)
```

## Navigate to Fragment

```
val fragment = DetailedWeatherForecast()

fragment.arguments = Bundle().apply {
    putString("zip_code", city.zipCode)
    putBoolean("use_fahrenheit", true)
}

requireFragmentManager().beginTransaction()
        .addToBackStack(createTag(city.id))
        .replace(R.id.main_content, fragment)
        .commit()
```

```
// .. inside DetailedWeatherForecast


val zipCode = intent
       .getStringExtra("zip_code")

val useFahrenheit = intent
       .getBooleanExtra(“celsius", true)
```

```
// .. inside DetailedWeatherForecast


val zipCode = requireArguments()
       .getString(“zip_code”)

val useFahrenheit = requireArguments()
       .getBoolean("celsius", true)
```

# Pre Navigation

## Navigate to Activity

```
val intent = Intent(this,
   DetailedWeatherForecast::class.java)

intent.putExtra("zip_code", city.zipCode)
intent.putExtra("use_fahrenheit", true)

startActivity(intent)
```

## Navigate to Fragment

```
val fragment = DetailedWeatherForecast()

fragment.arguments = Bundle().apply {
    putString("zip_code", city.zipCode)
    putBoolean("use_fahrenheit", true)
}

requireFragmentManager().beginTransaction()
        .addToBackStack(createTag(city.id))
        .replace(R.id.main_content, fragment)
        .commit()
```

---

```
// .. inside DetailedWeatherForecast


val zipCode = intent
        .getStringExtra("zip_code")

val useFahrenheit = intent
        .getBooleanExtra(“celsius", true)
```

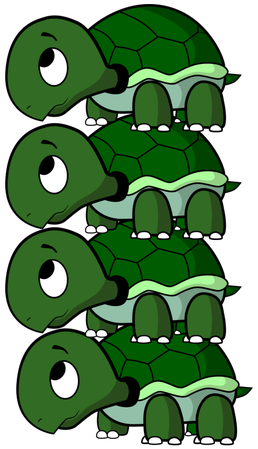
```
// .. inside DetailedWeatherForecast


val zipCode = requireArguments()
        .getString(“zip_code”)

val useFahrenheit = requireArguments()
        .getBoolean("celsius", true)
```

# Navigation

```
findNavController().navigate(
    toDetailForecast(city.zipCode)
            .setUseFahrenheit(true))
```

## Navigate to Activity

```
val intent = Intent(this,
   DetailedWeatherForecast::class.java)

intent.putExtra("zip_code", city.zipCode)
intent.putExtra(“use_fahrenheit", true)

startActivity(intent)


// .. inside DetailedWeatherForecast


val zipCode = intent
       .getStringExtra(“zip_code”)
val useFahrenheit = intent
       .getBooleanExtra(“use_fahrenheit”, true)

// .. do stuff.
```



## Navigate to Fragment

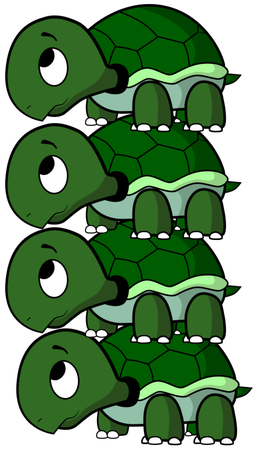
```
val fragment = DetailedWeatherForecast()

fragment.arguments = Bundle().apply {
    putString("zip_code", city.zipCode)
    putBoolean(“use_fahrenheit”, true)
}

requireFragmentManager().beginTransaction()
        .addToBackStack("WeatherDetails_${1}")
        .replace(R.id.main_content, fragment)
        .commit()

// .. inside DetailedWeatherForecast


val zipCode = requireArguments()
        .getString(“zip_code”)

val useCelsius = requireArguments()
        .getBoolean("celsius", true)

// .. do stuff
```

# Navigation

```
val args = WeatherDetailFragmentArgs by navArgs()

val zipCode = args.zipCode
val shouldShowFahrenheit = args.useFahrenheit
```

## Navigate to Activity

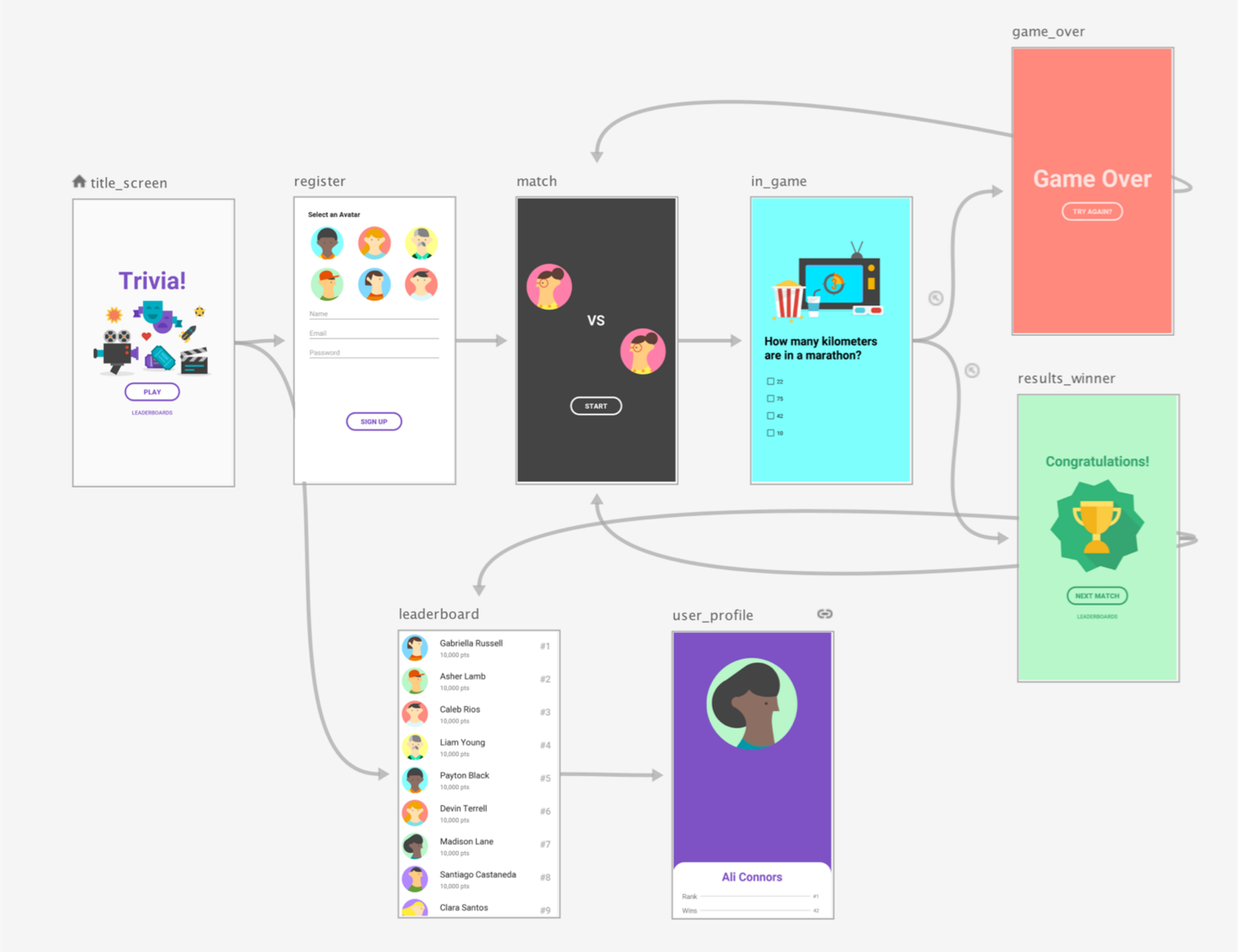
```
val intent = Intent(this,
   DetailedWeatherForecast::class.java)

intent.putExtra("zip_code", city.zipCode)
intent.putExtra(“use_fahrenheit", true)

startActivity(intent)


// .. inside DetailedWeatherForecast


val zipCode = intent
       .getStringExtra(“zip_code”)
val useFahrenheit = intent
       .getBooleanExtra(“use_fahrenheit”, true)

// .. do stuff.
```



## Navigate to Fragment

```
val fragment = DetailedWeatherForecast()

fragment.arguments = Bundle().apply {
    putString("zip_code", city.zipCode)
    putBoolean(“use_fahrenheit”, true)
}

requireFragmentManager().beginTransaction()
        .addToBackStack("WeatherDetails_${1}")
        .replace(R.id.main_content, fragment)
        .commit()

// .. inside DetailedWeatherForecast


val zipCode = requireArguments()
        .getString(“zip_code”)

val useCelsius = requireArguments()
        .getBoolean("celsius", true)

// .. do stuff
```

# How Does it Work?




https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample

# Minimal Dependencies

Navigation

```
dependencies {
    ...
    implementation "androidx.navigation:navigation-runtime-ktx:$version" /* Navigation Runtime */
}
```

# Minimal Dependencies

## Navigation

```
dependencies {
    ...
    implementation "androidx.navigation:navigation-runtime-ktx:$version" /* Navigation Runtime */
    implementation "androidx.navigation:navigation-fragment-ktx:$version" /* Fragment Destination */

}
```

# Minimal Dependencies

## Navigation

```
dependencies {
    ...
    implementation "androidx.navigation:navigation-runtime-ktx:$version" /* Navigation Runtime */
    implementation "androidx.navigation:navigation-fragment-ktx:$version" /* Fragment Destination */
    implementation “androidx.navigation:navigation-ui-ktx:$version" /* Top-Level Navigation */
}
```

# Minimal Dependencies

## Navigation

```
dependencies {
    ...
    implementation "androidx.navigation:navigation-runtime-ktx:$version" /* Navigation Runtime */
    implementation "androidx.navigation:navigation-fragment-ktx:$version" /* Fragment Destination */
    implementation “androidx.navigation:navigation-ui-ktx:$version" /* Top-Level Navigation */
}
```

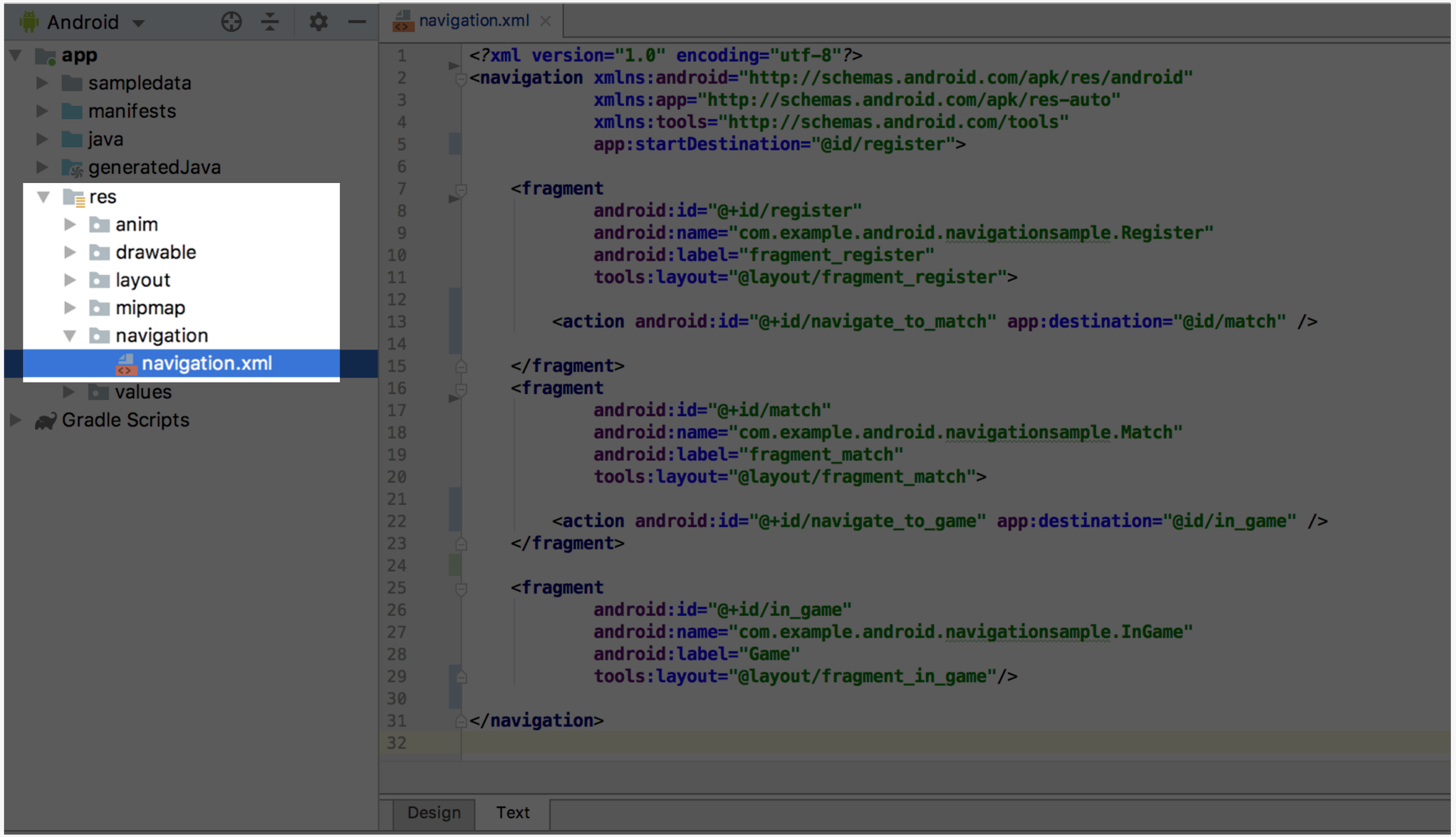
## Safe Args

Project build.gradle:

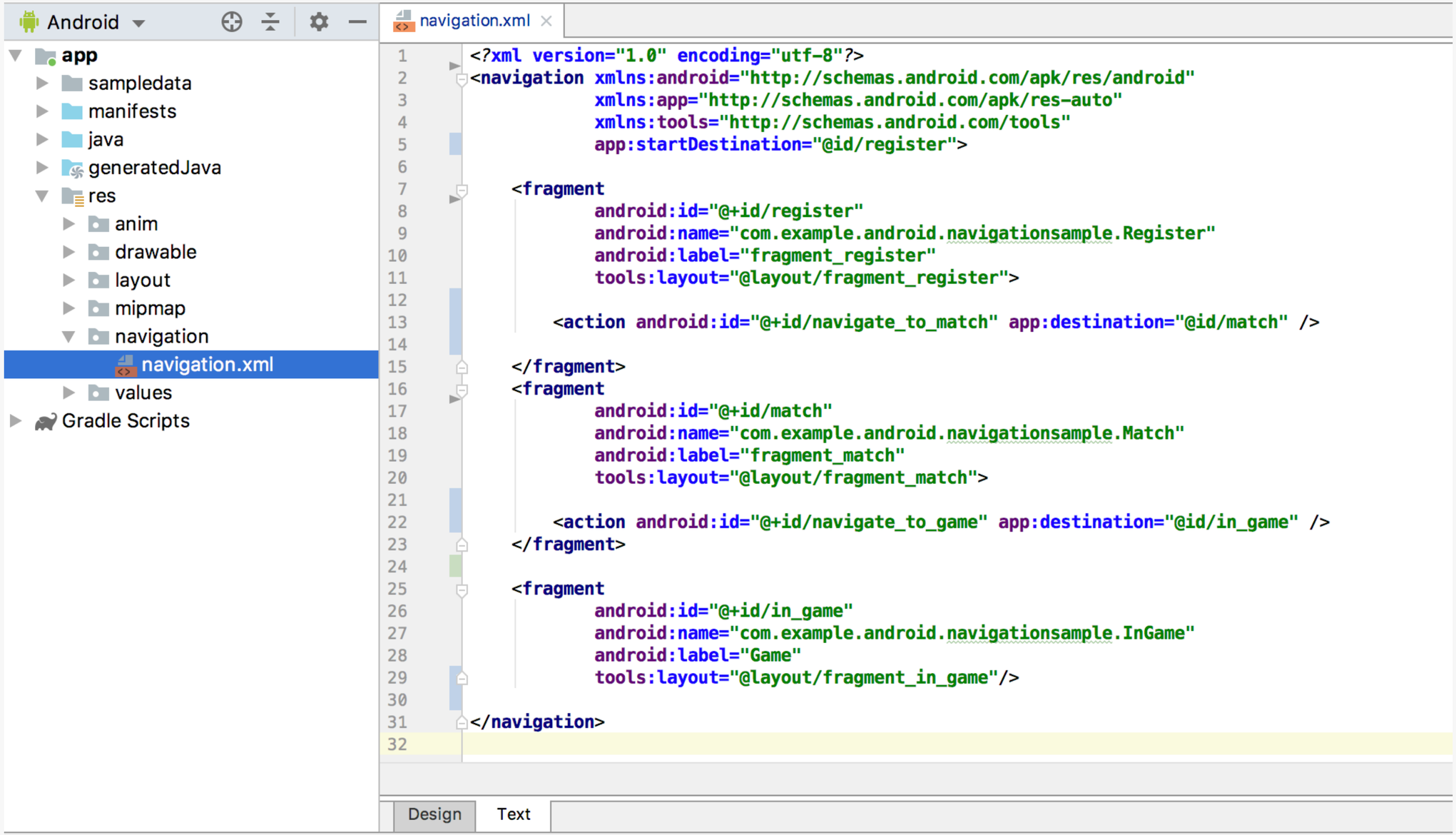
```
dependencies {
   ...
   classpath “android.arch.navigation:navigation-safe-args-gradle-plugin:$version"
}
```

Module build.gradle:

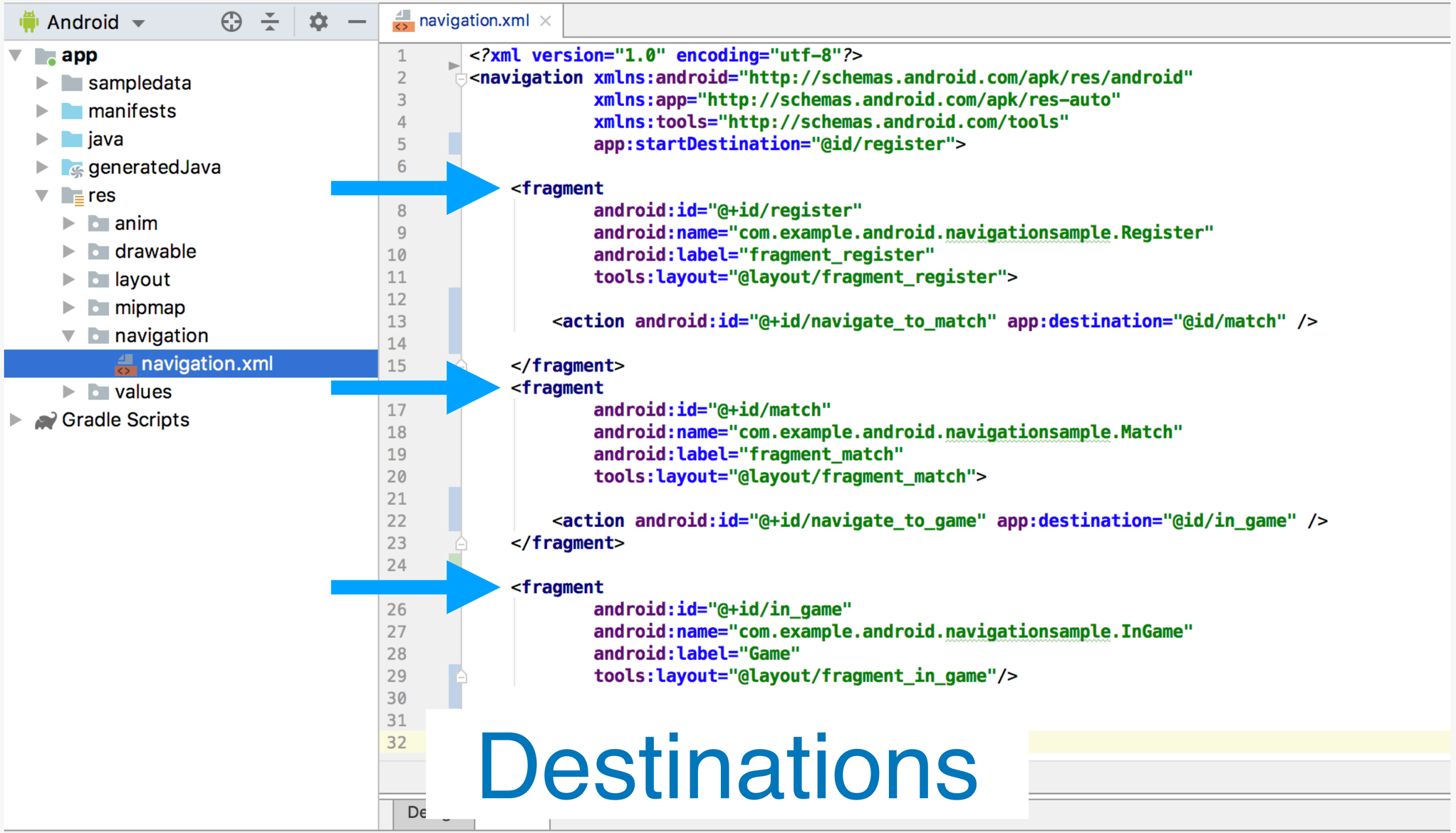
```
apply plugin: 'androidx.navigation.safeargs'
```




https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample

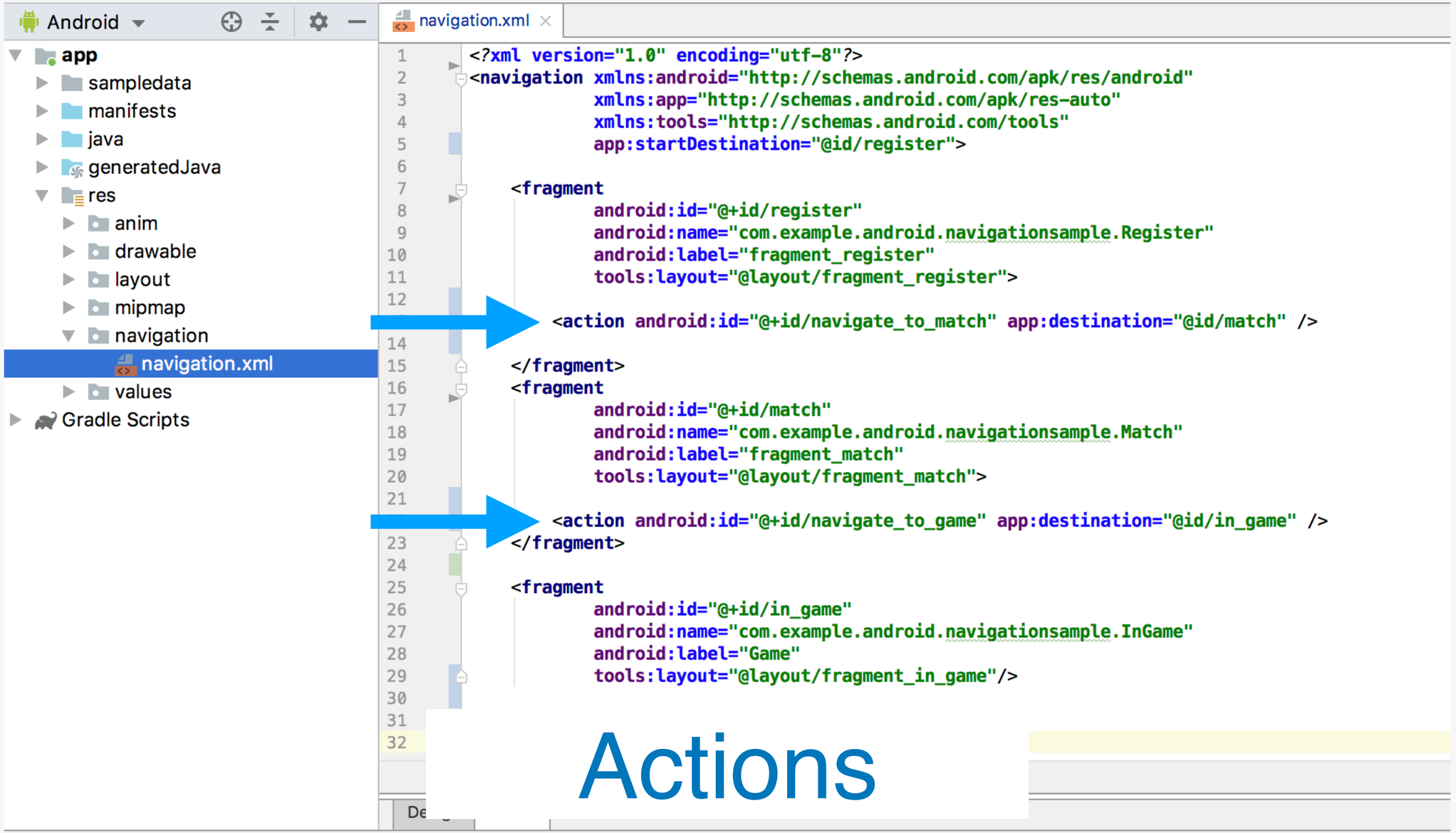



https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample

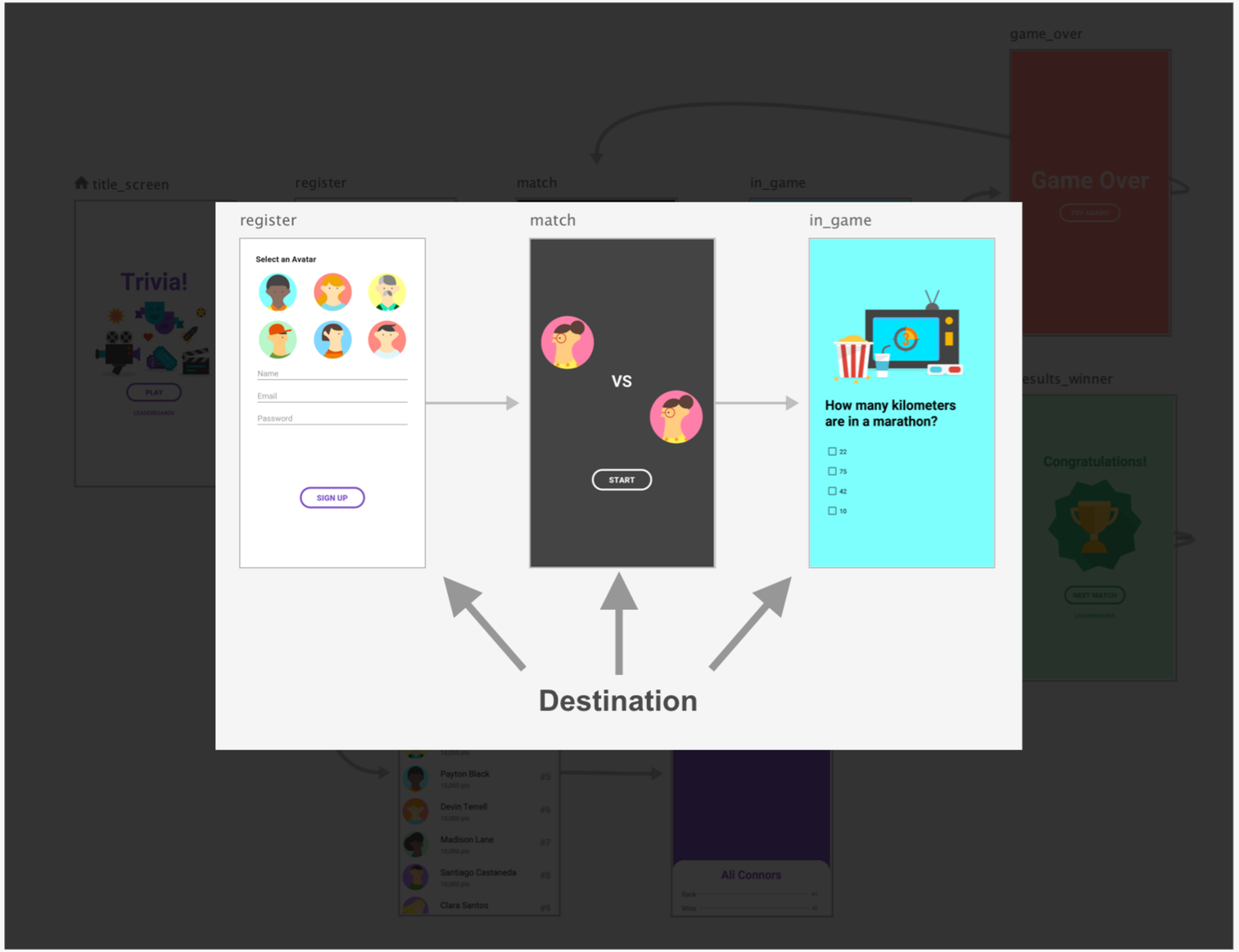



# Destinations

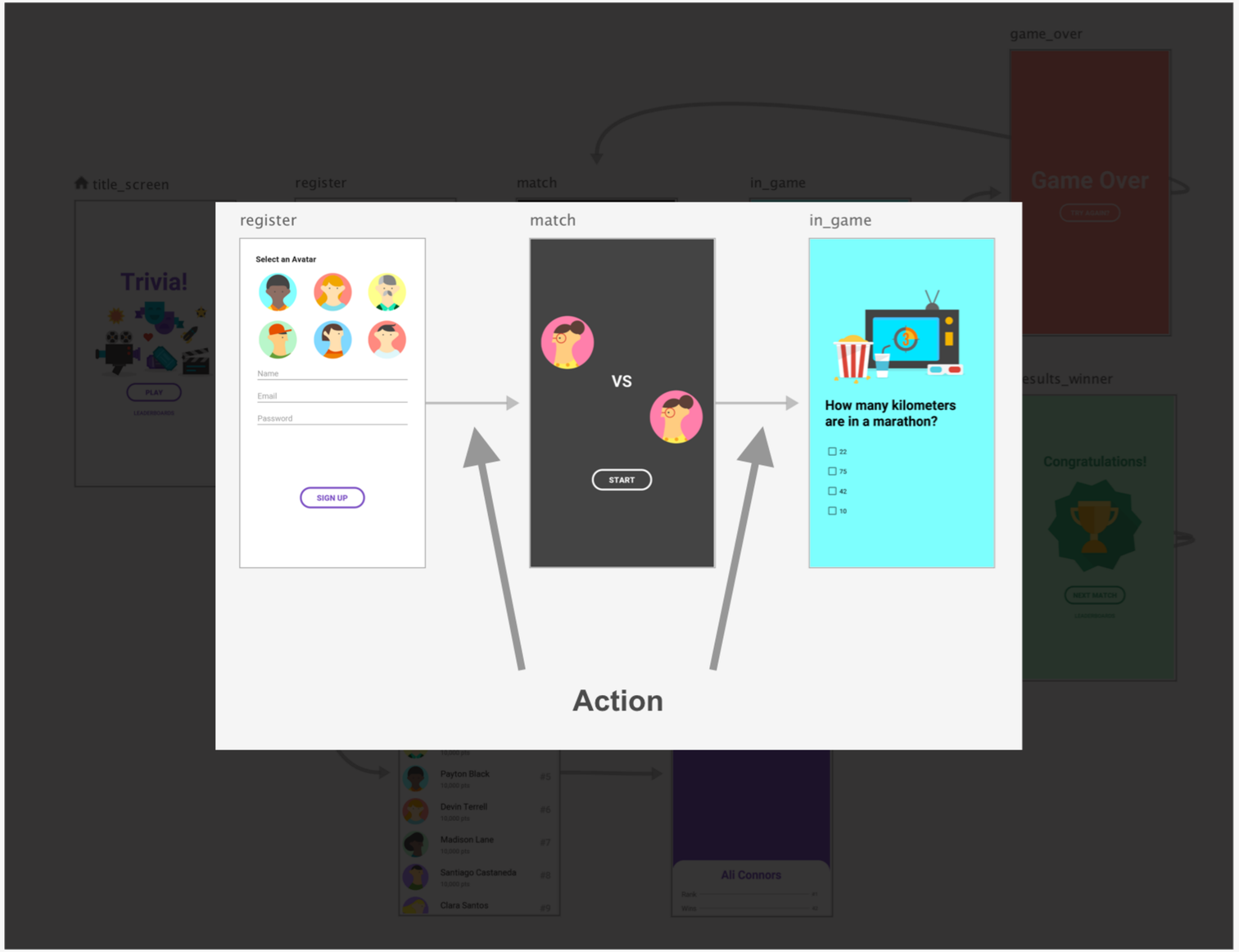
https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample




https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample



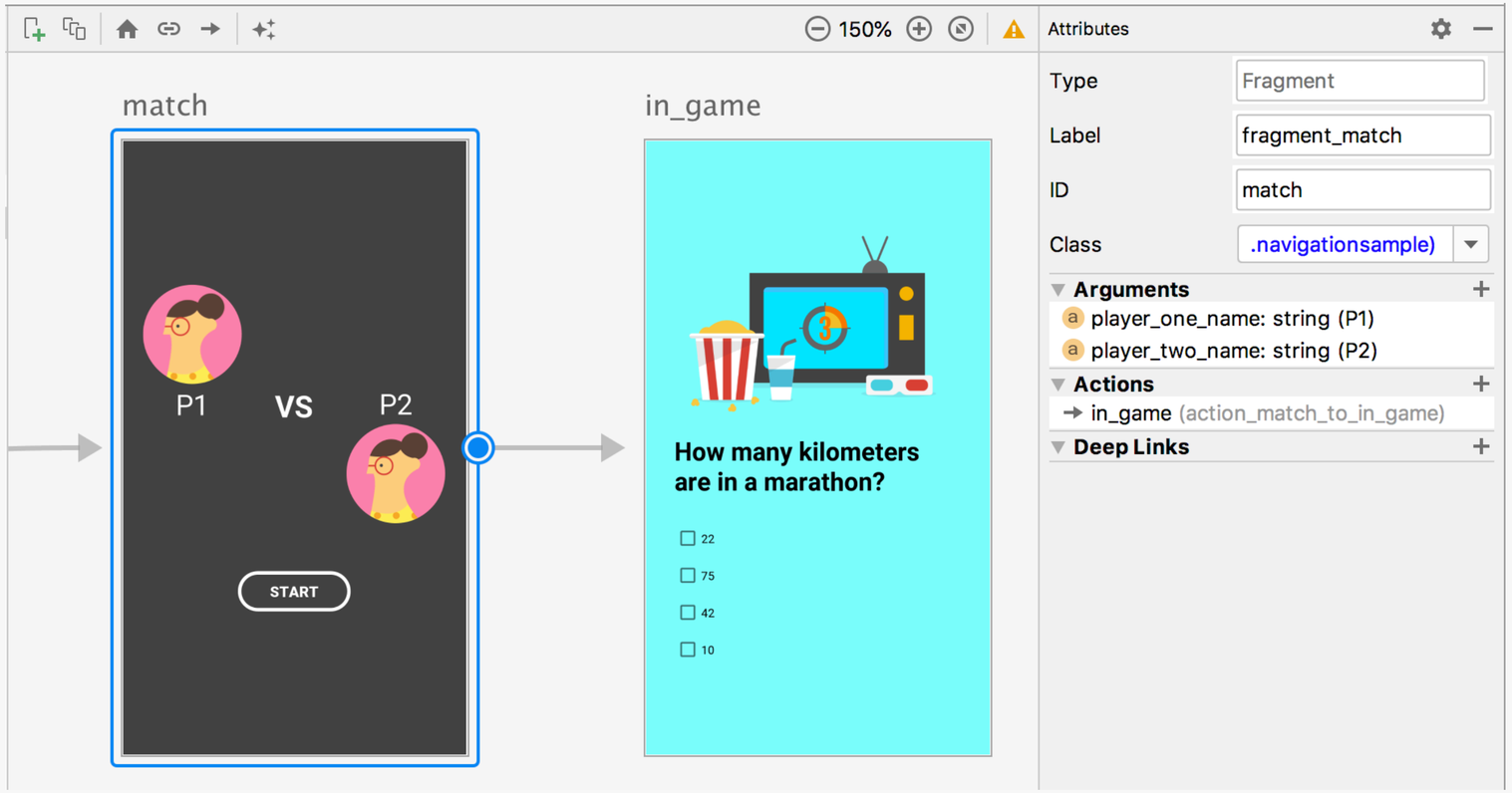

https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample




https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample

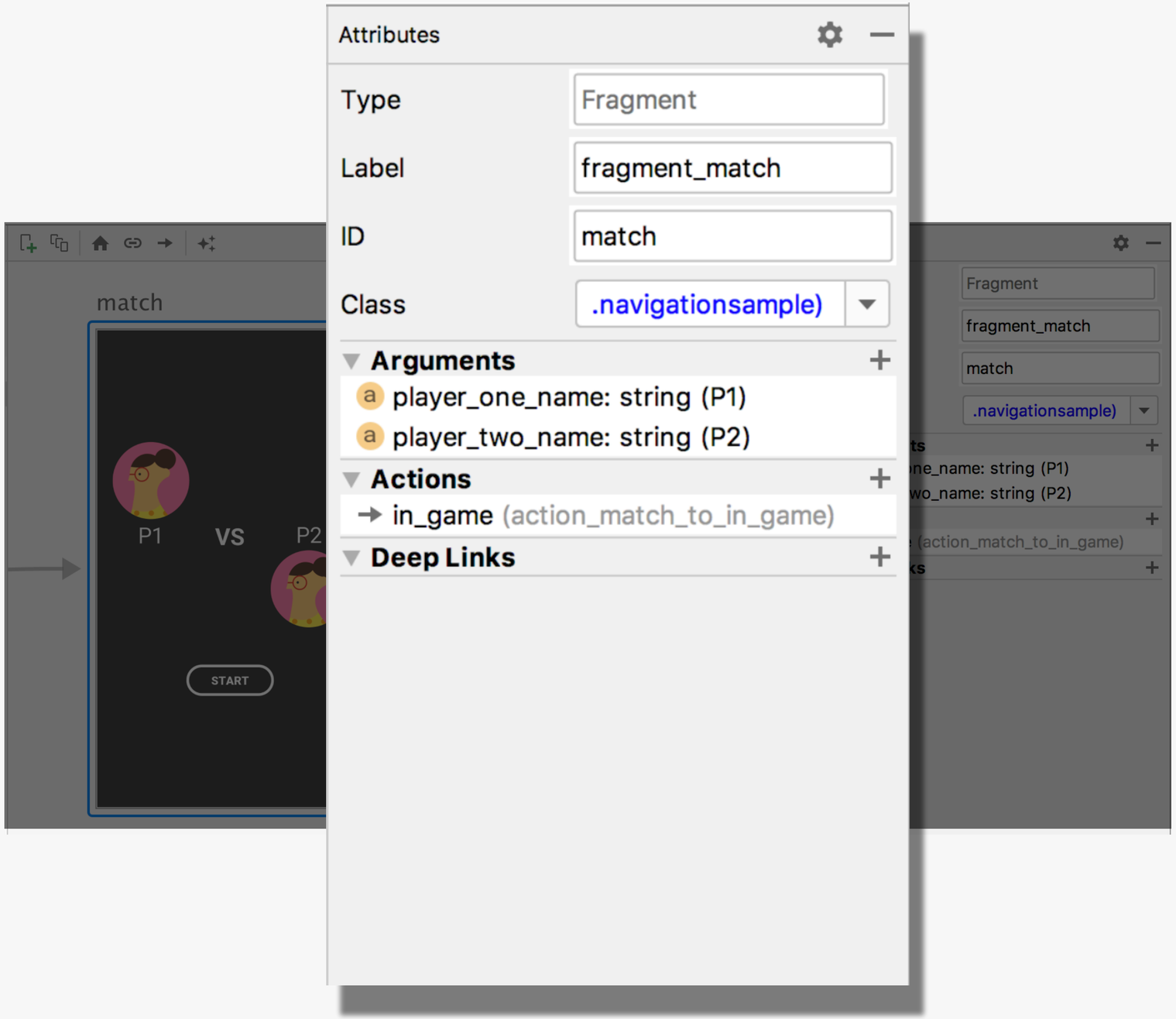
# Destinations

- Arguments
- Deep Links
- Actions



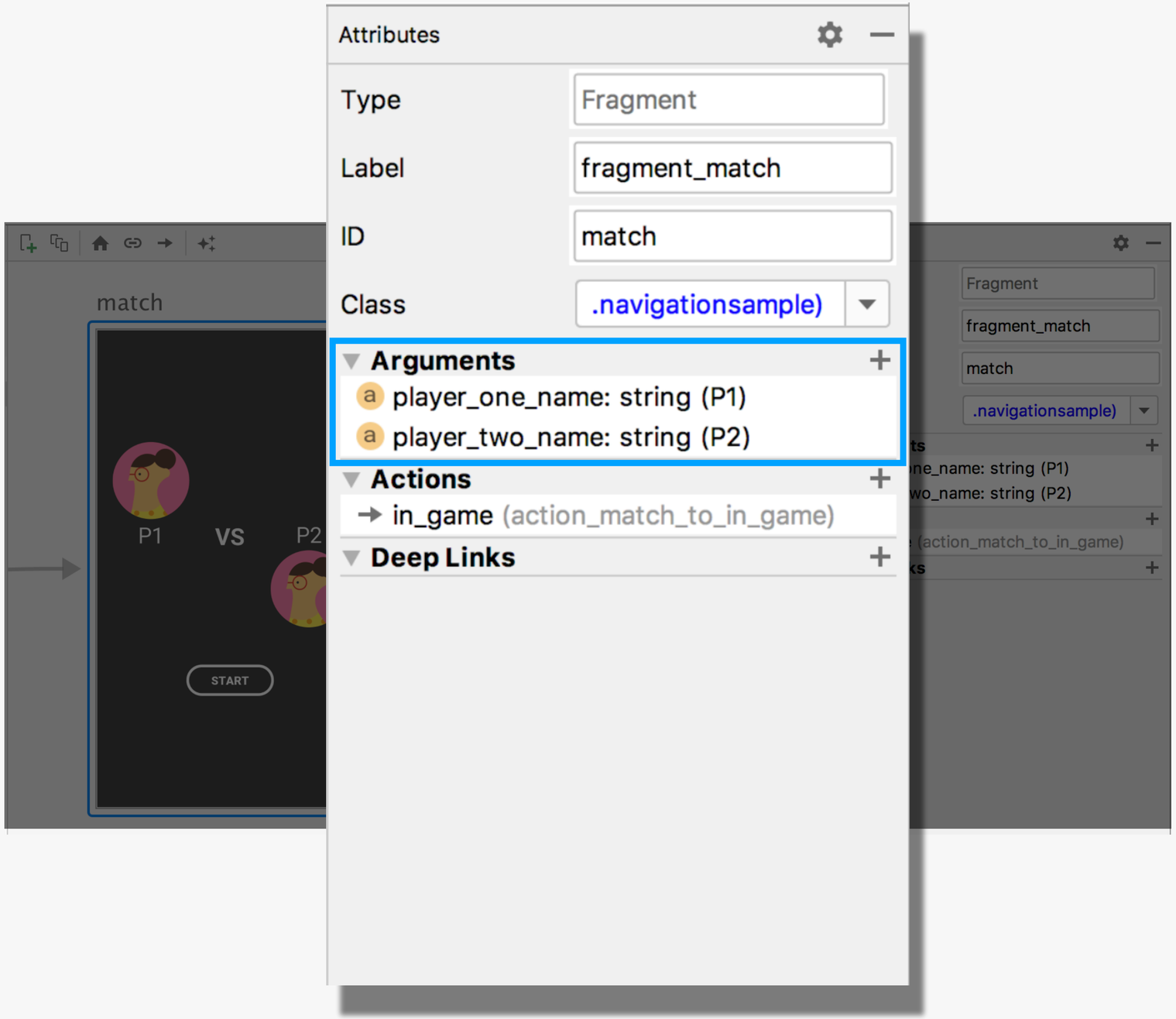
# Destinations

- Arguments
- Deep Links
- Actions



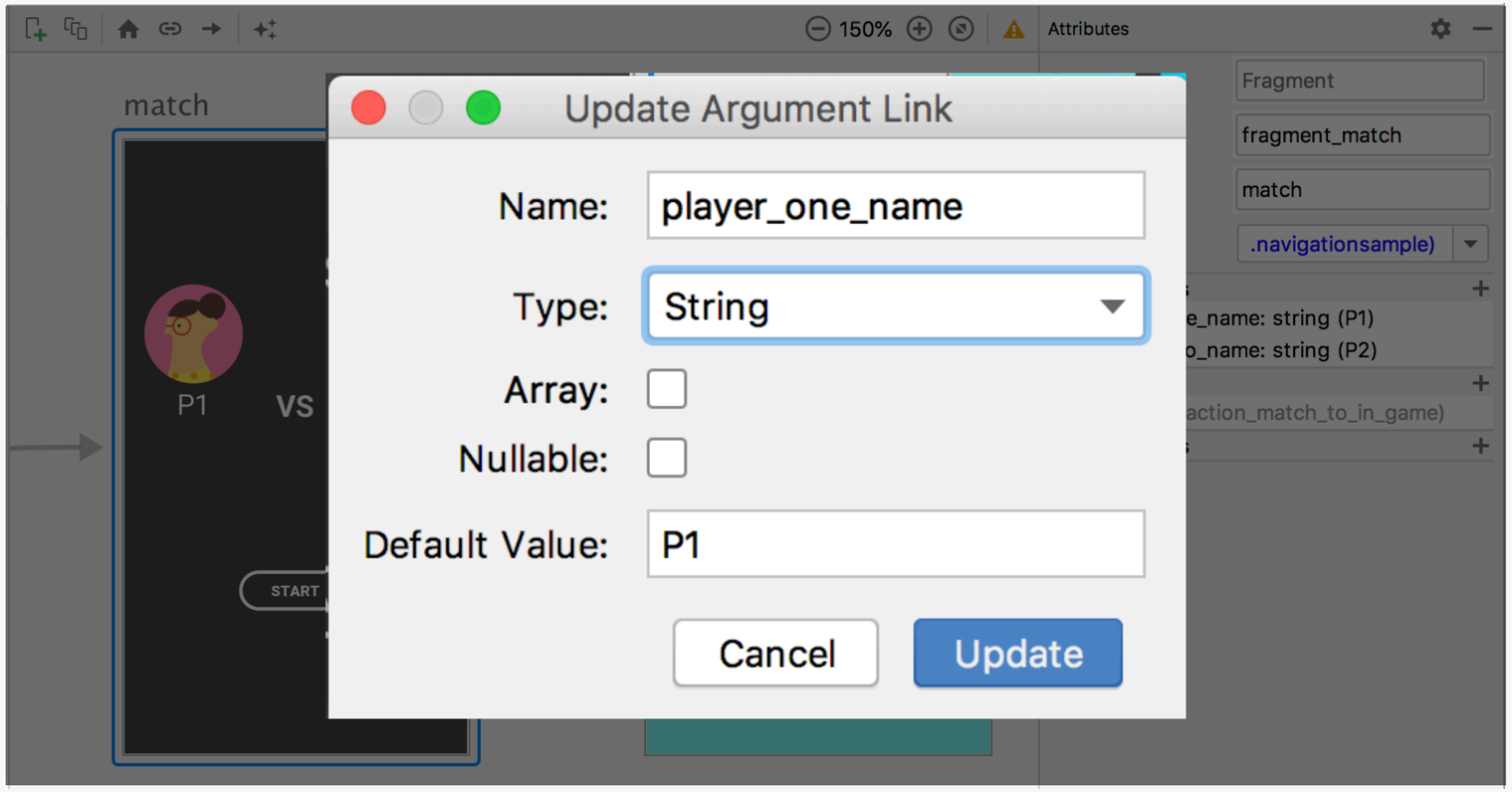
# Destinations

- **Arguments**
- Deep Links
- Actions



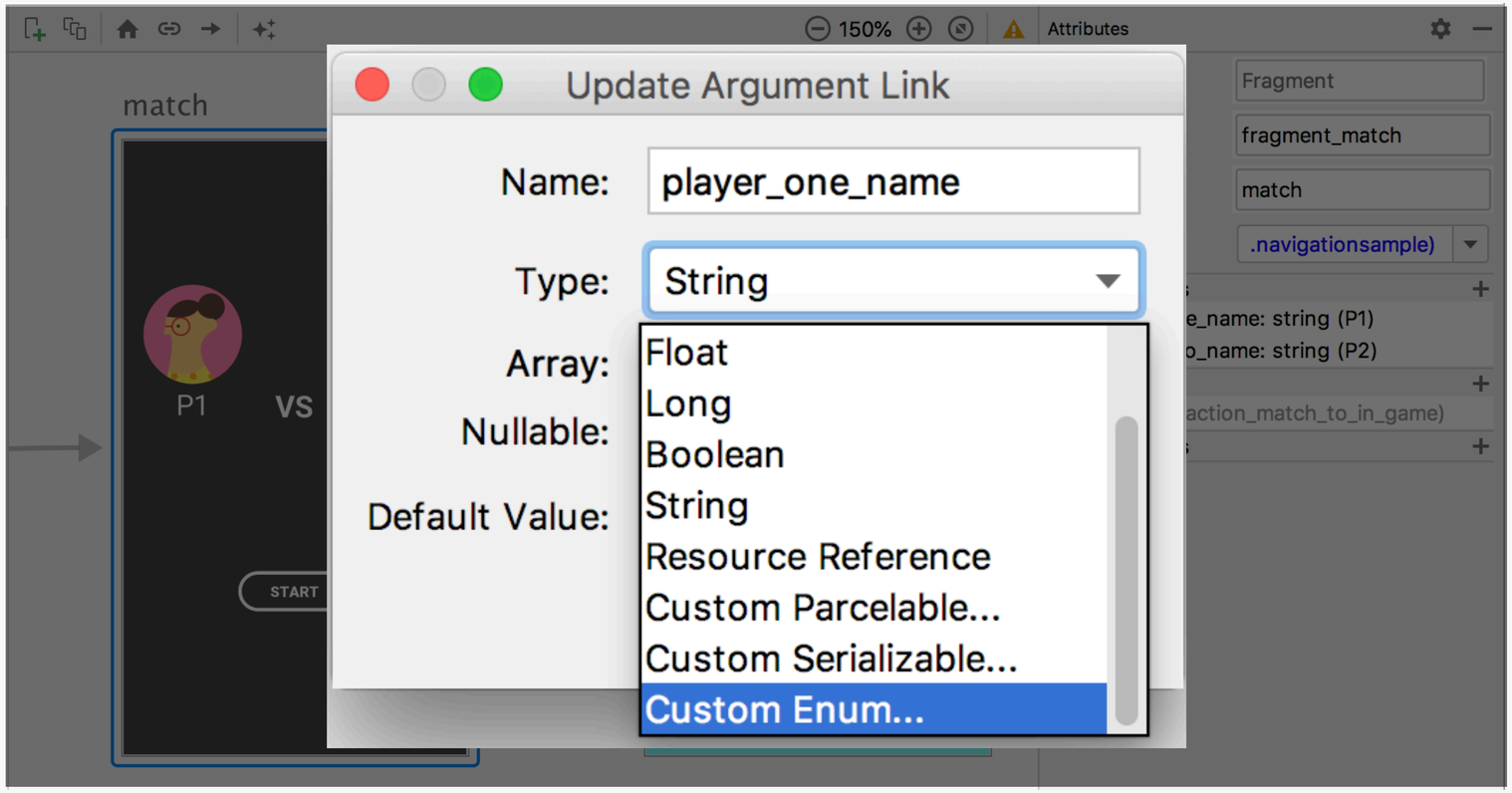
# Destinations

- **Arguments**
- Deep Links
- Actions



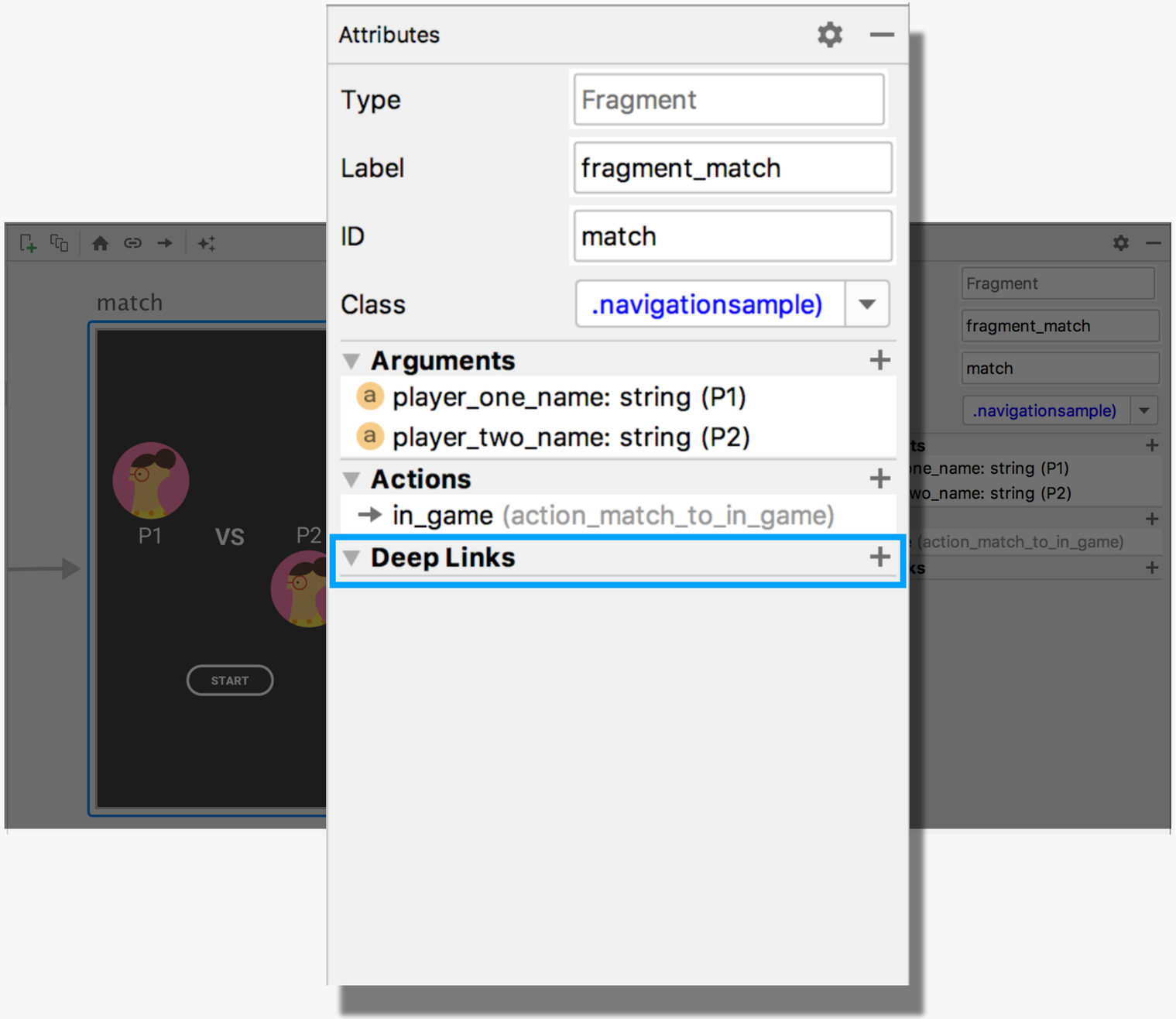
# Destinations

- **Arguments**
- Deep Links
- Actions



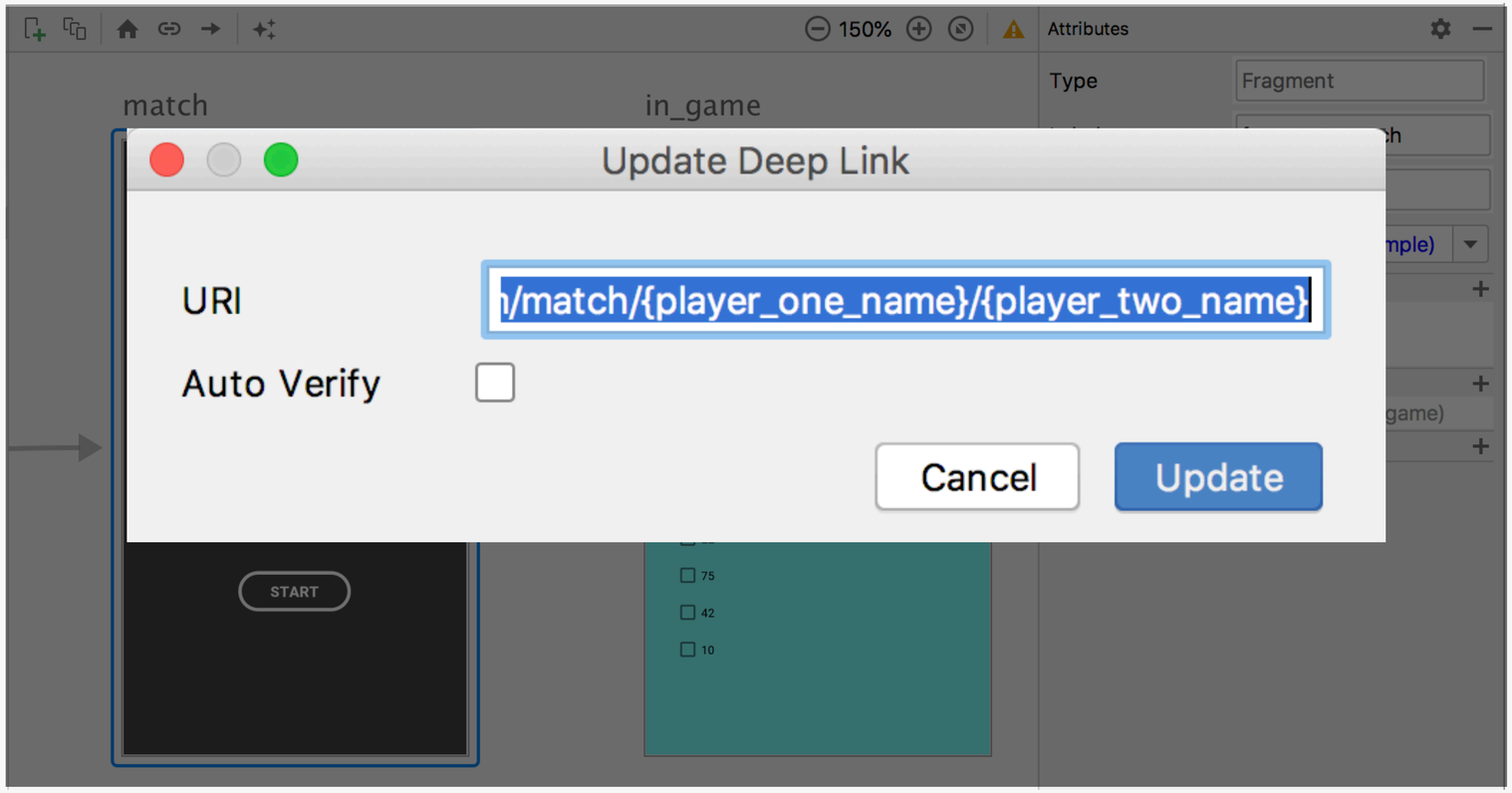
# Destinations

- **Arguments**
- Deep Links
- Actions



# Destinations

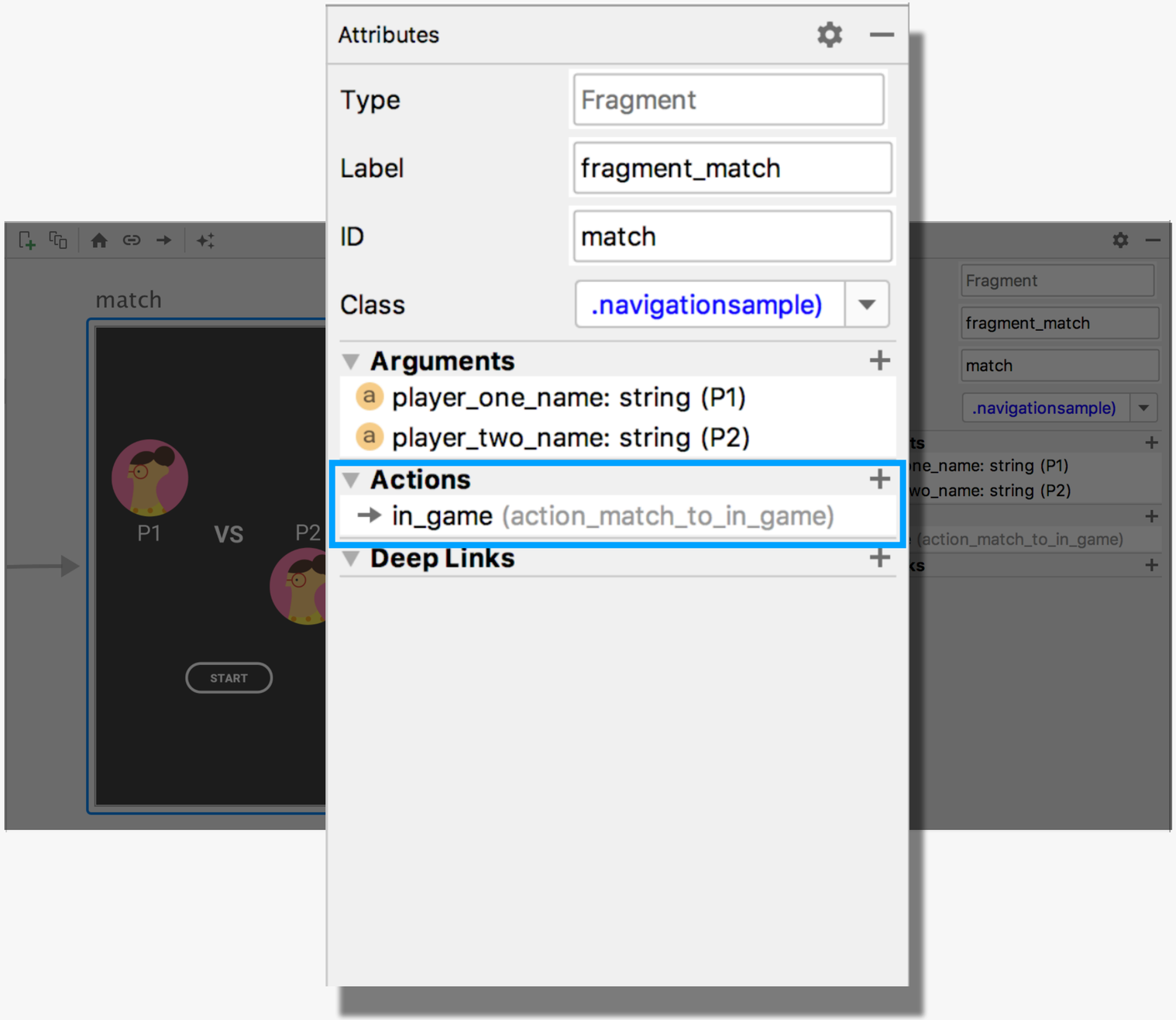
- Arguments
- **Deep Links**
- Actions




```
<deepLink app:uri="www.example.com/match/{player_one_name}/{player_two_name} " />
```

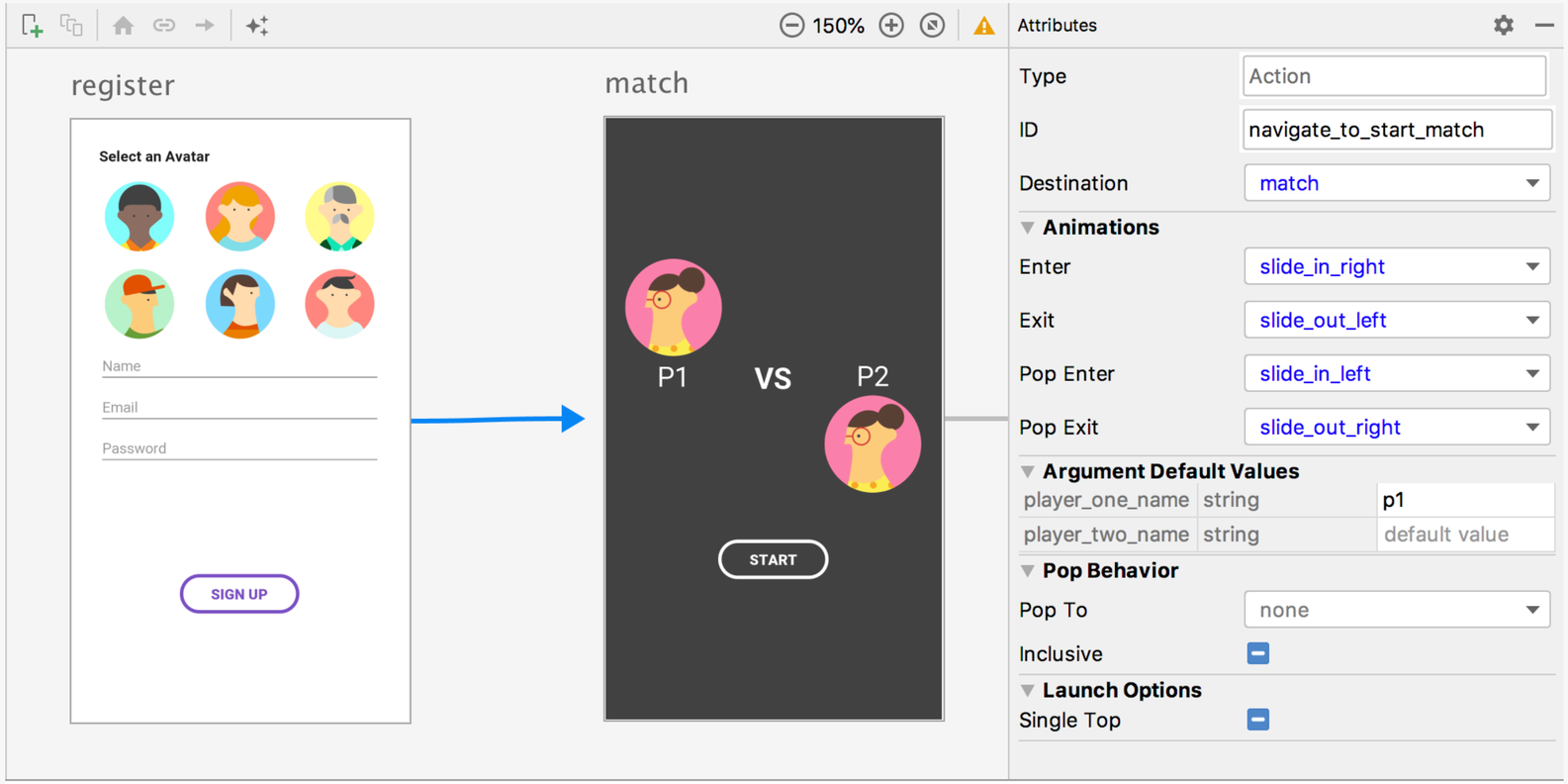
# Destinations

- Arguments
- Deep Links
- **Actions**



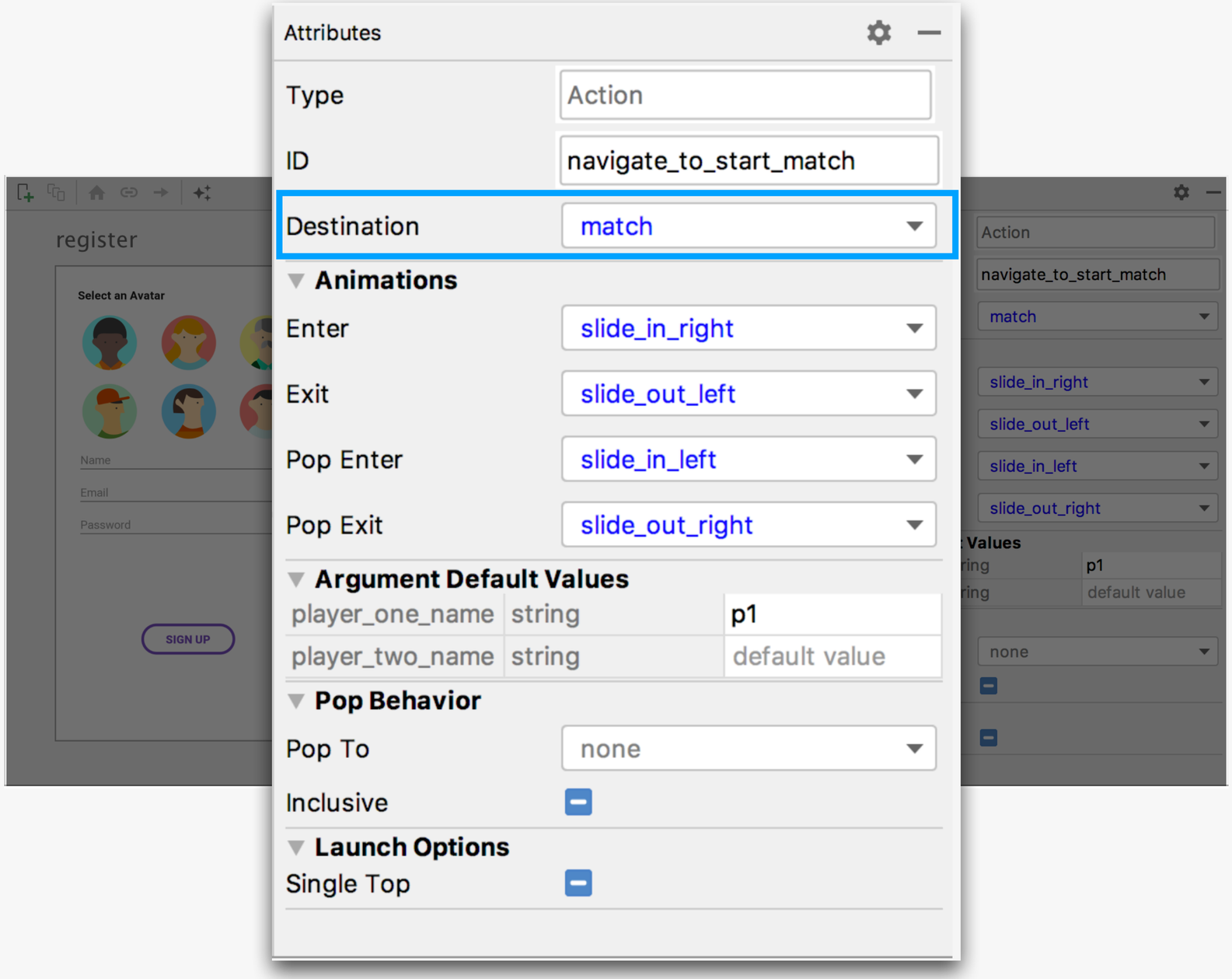
# Actions

- Destination
- Argument Default Values
- Pop Behavior
- Launch Options
- Animations



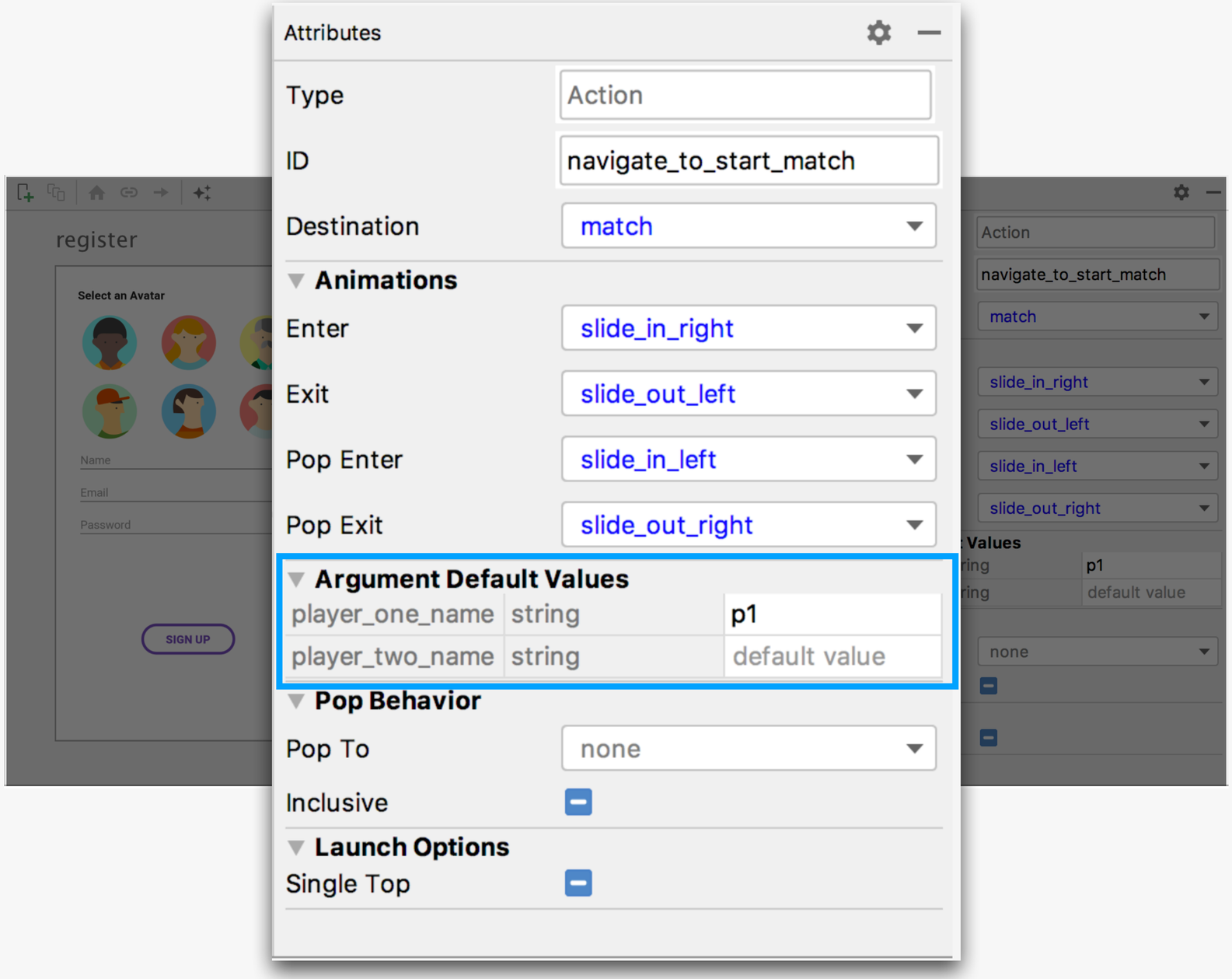
# Actions

- **Destination**
- Argument Default Values
- Pop Behavior
- Launch Options
- Animations



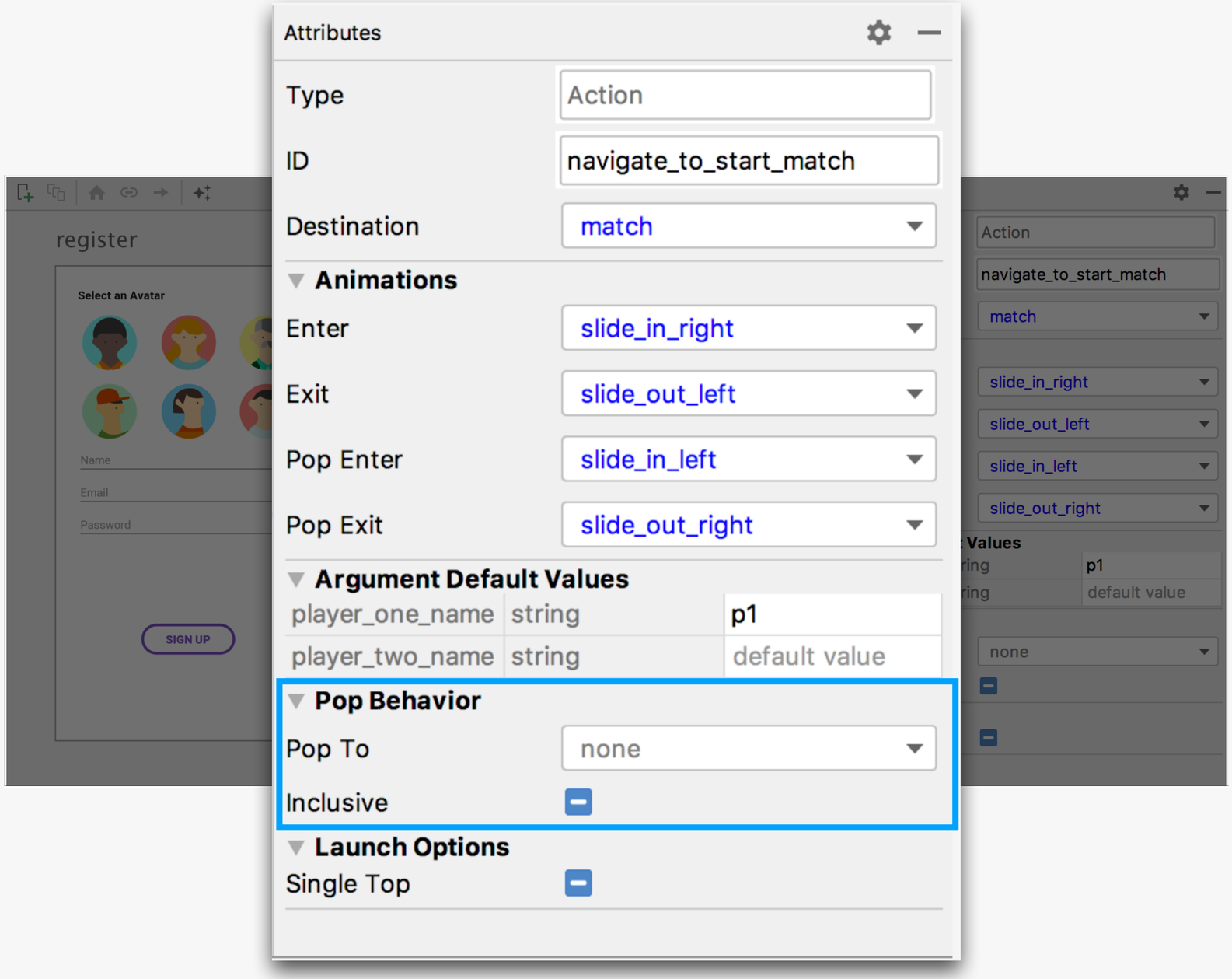
# Actions

- Destination
- **Argument Default Values**
- Pop Behavior
- Launch Options
- Animations

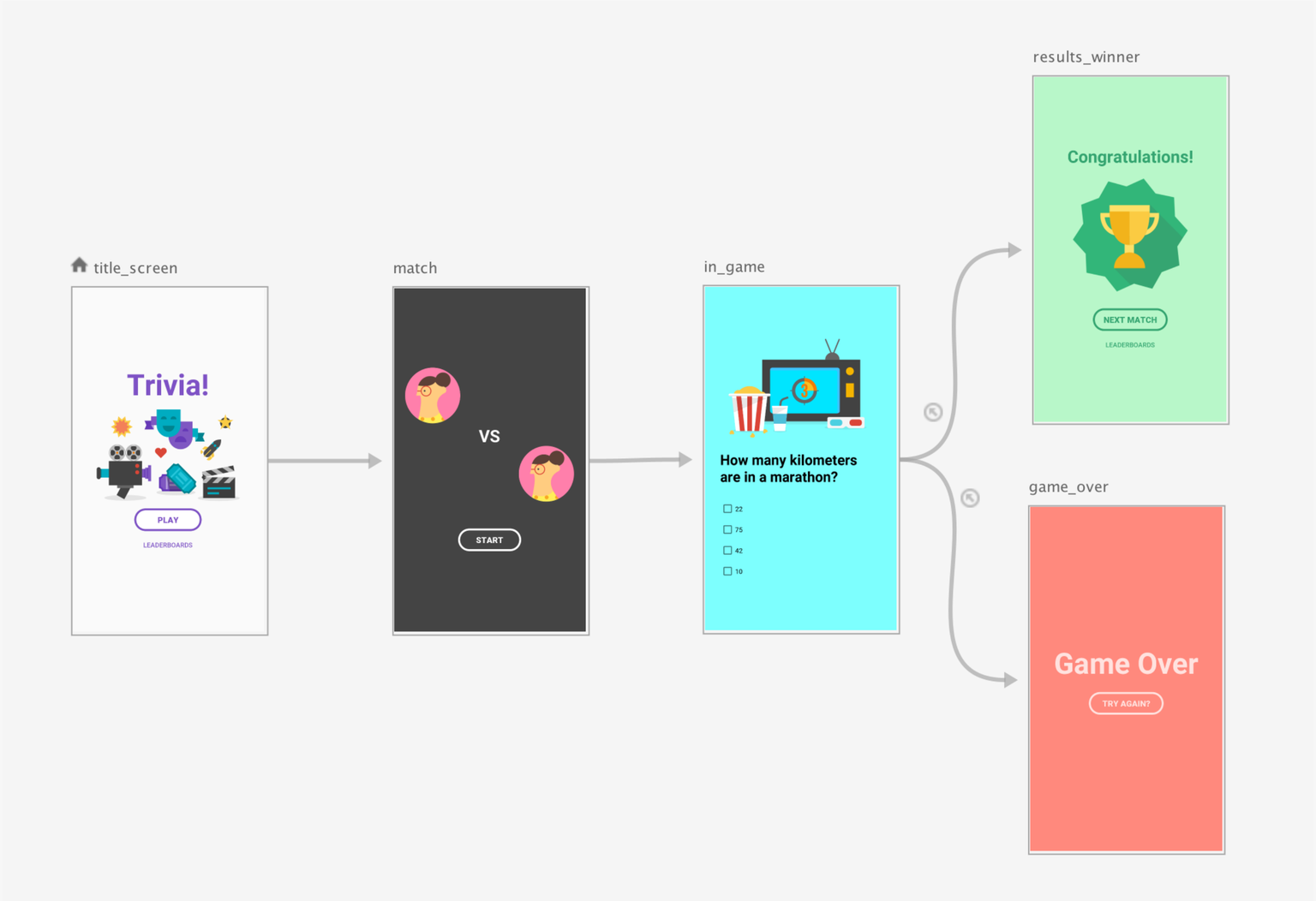


# Actions

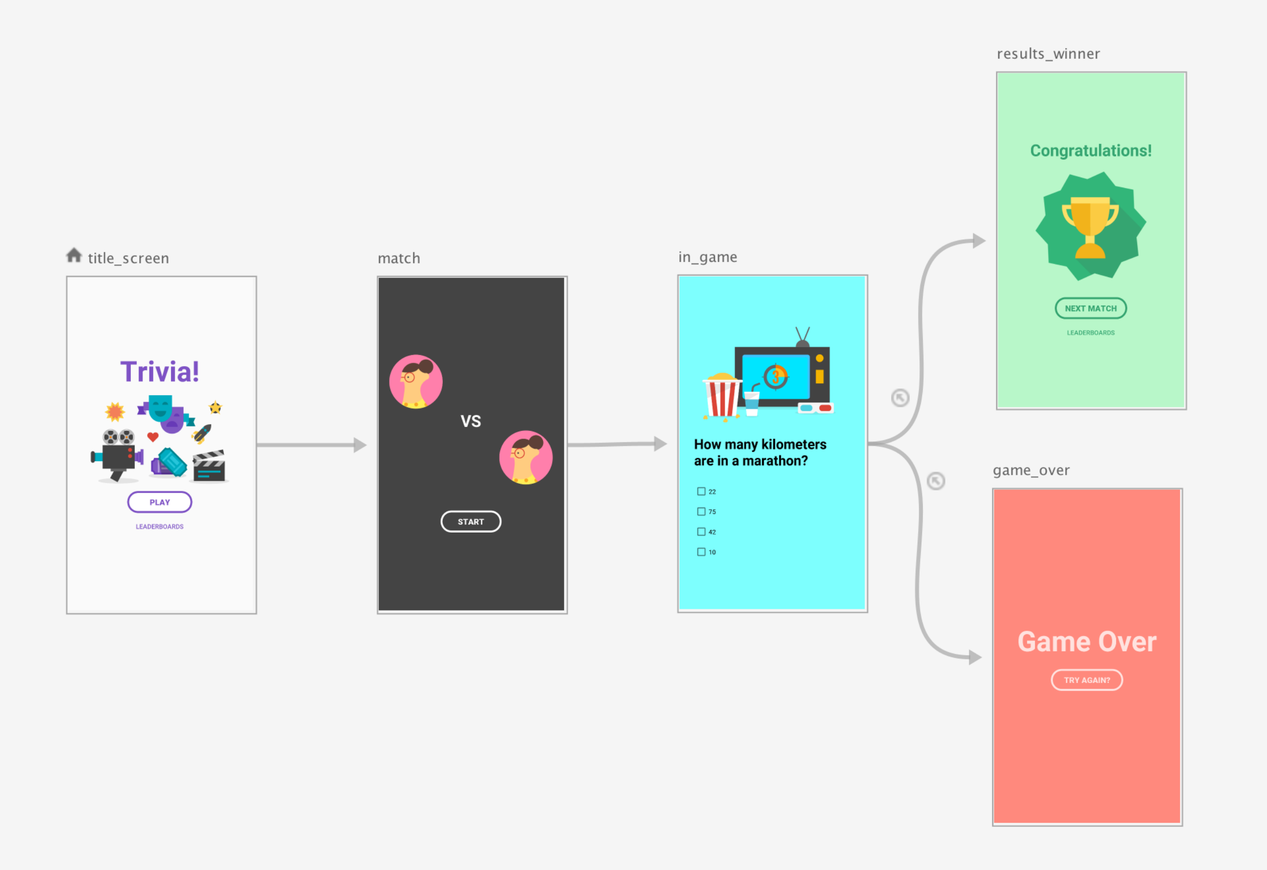
- Destination
- Argument Default Values
- **Pop Behavior**
- Launch Options
- Animations



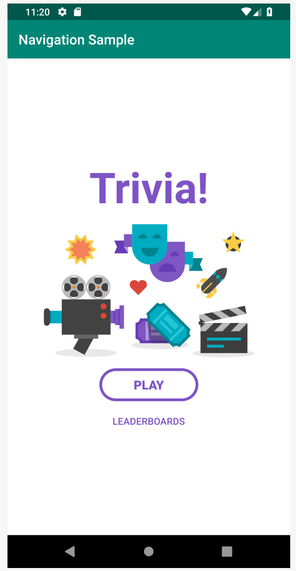
# Stack Based Navigation



# Stack Based Navigation

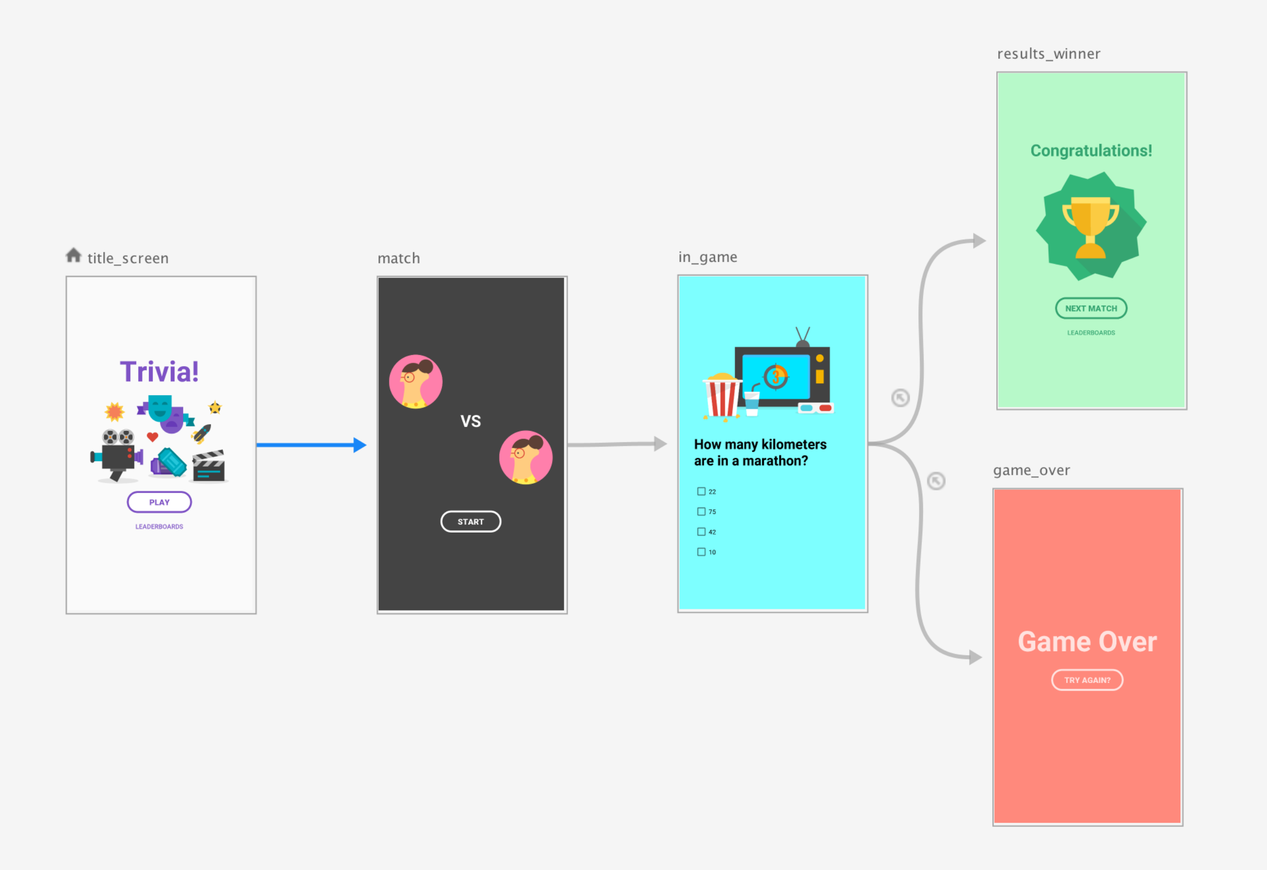



Navigation Graph

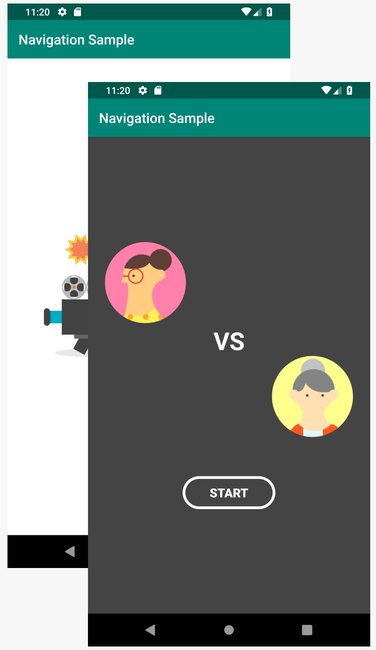



Stack

# Stack Based Navigation

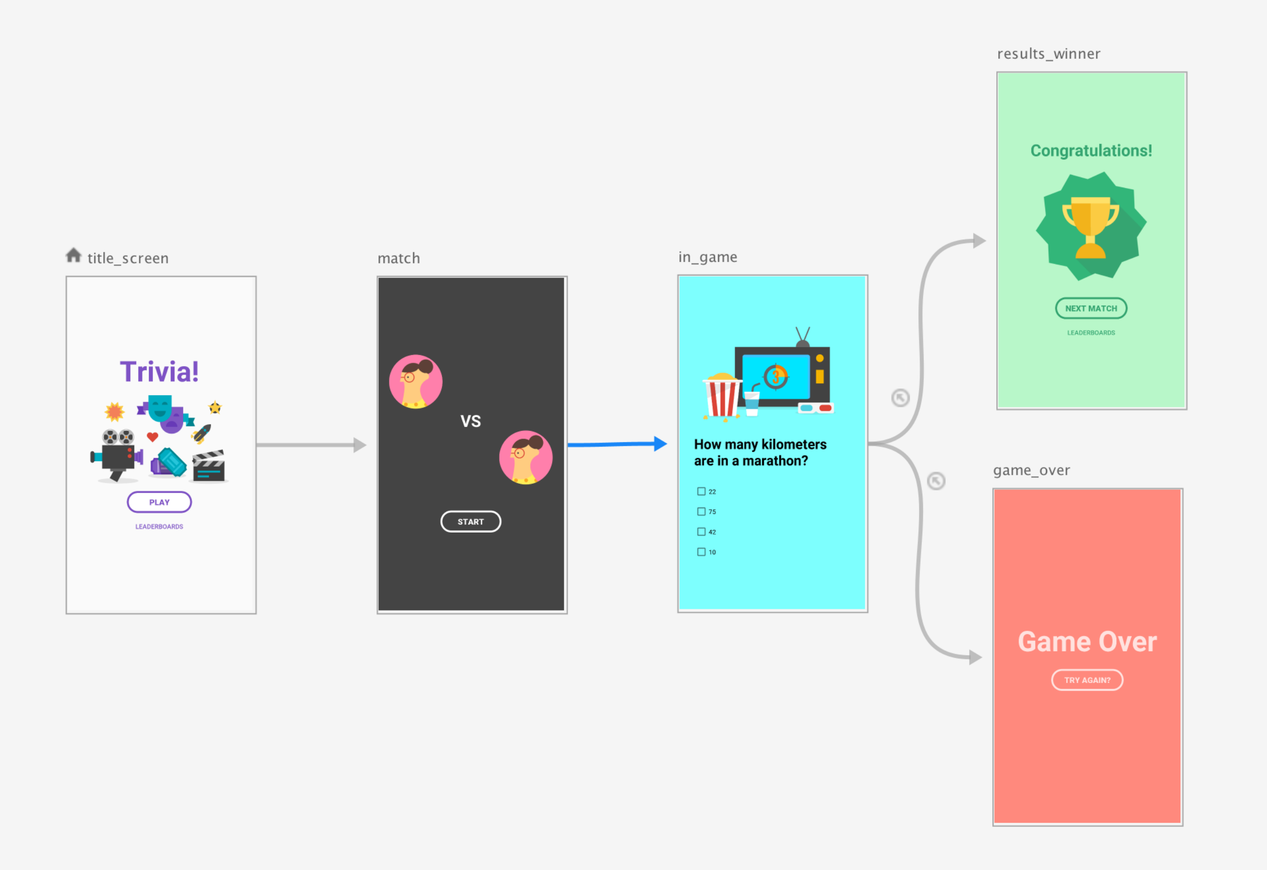



Navigation Graph

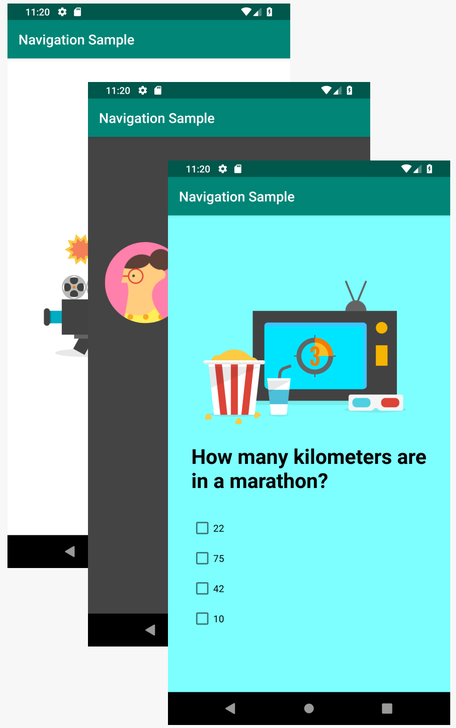



Stack

# Stack Based Navigation

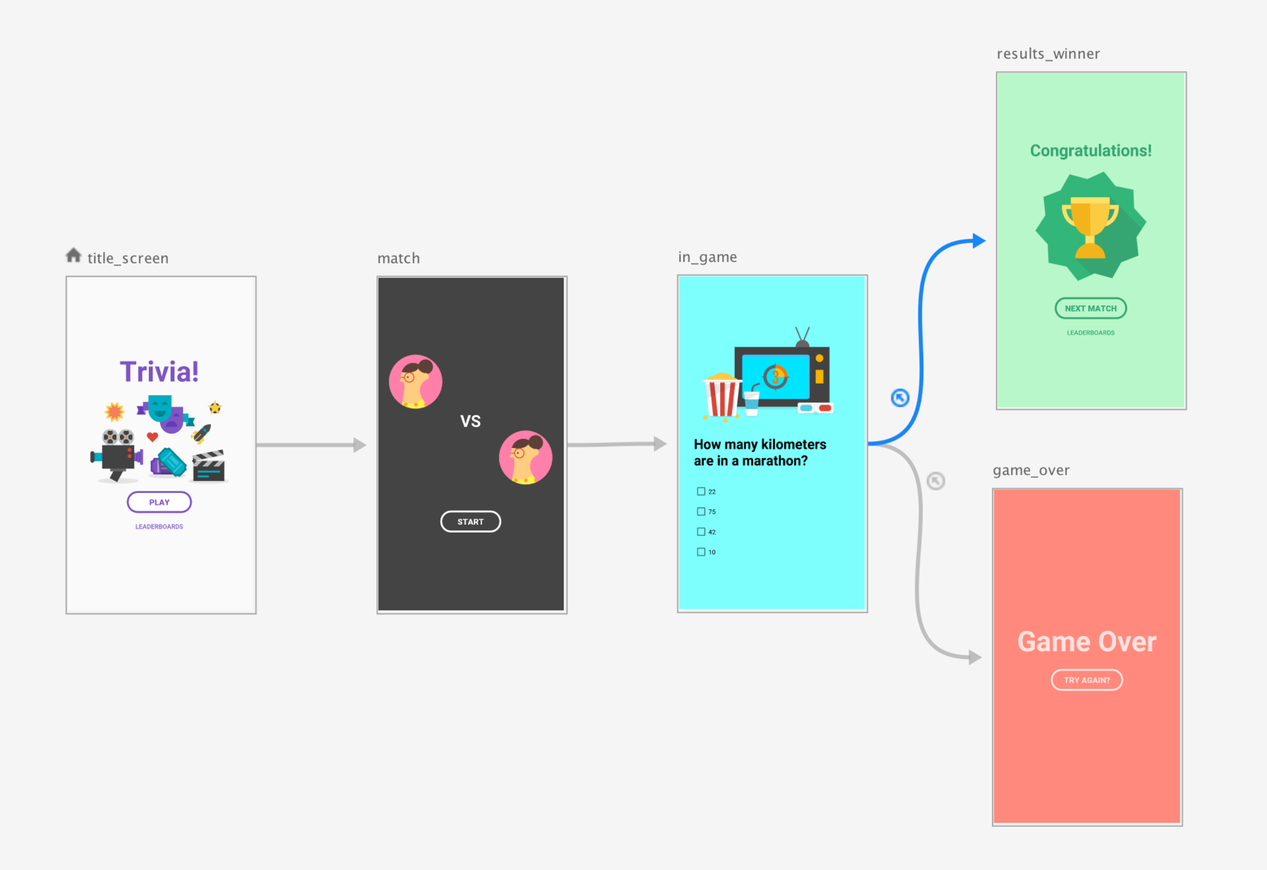



Navigation Graph

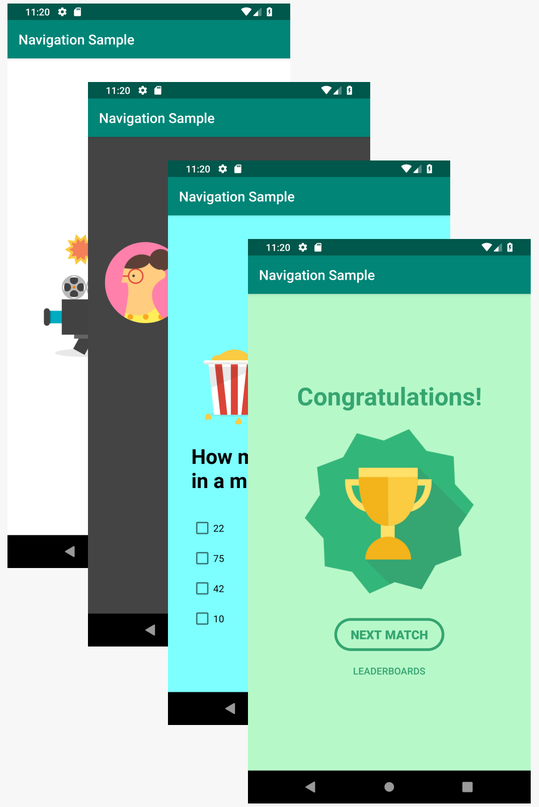



Stack

# Stack Based Navigation



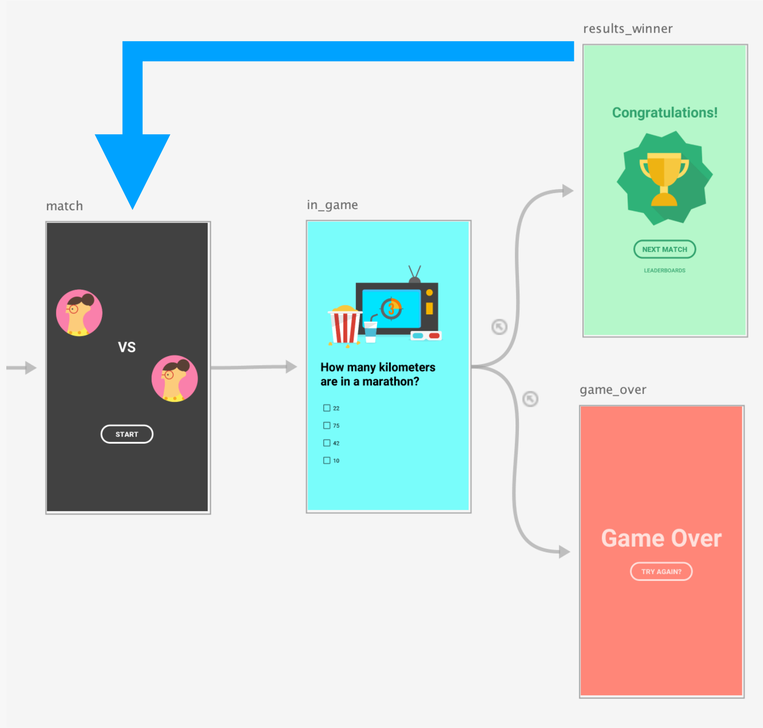

Navigation Graph




Stack

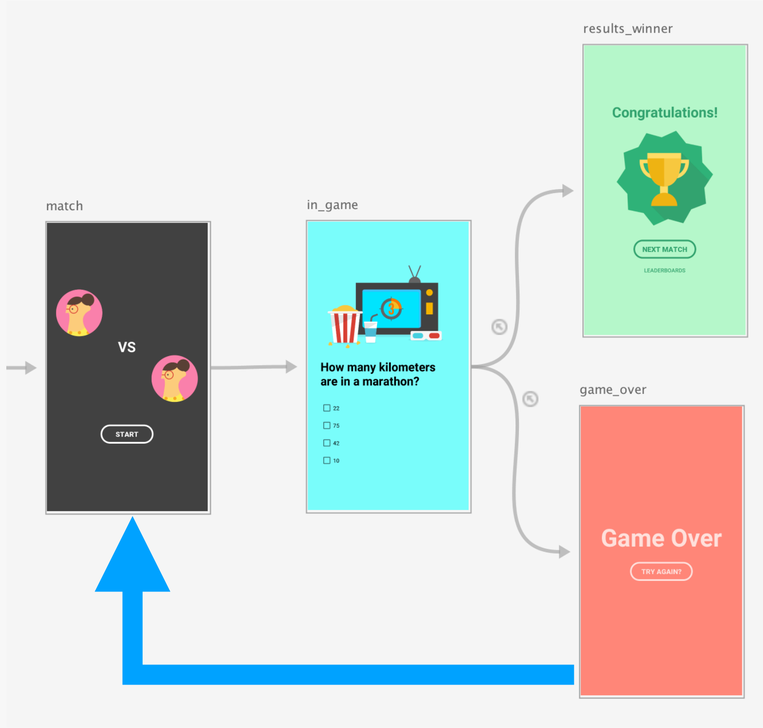
# Popping Back to Match

```
<fragment android:id="@+id/match" ...>

<fragment android:id="@+id/in_game" ...>

    <action
        android:id="@+id/action_in_game_to_results_winner"
        app:destination="@id/results_winner"
        app:popUpTo="@id/match" />
    <action
        android:id="@+id/action_in_game_to_game_over"
        app:destination="@id/game_over"
        app:popUpTo="@id/match" />
</fragment>

<fragment
        android:id="@+id/results_winner"
        android:name="com.example.android.navigationsample.ResultsWinner"/>

<fragment
        android:id="@+id/game_over"
        android:name="com.example.android.navigationsample.GameOver" />
```



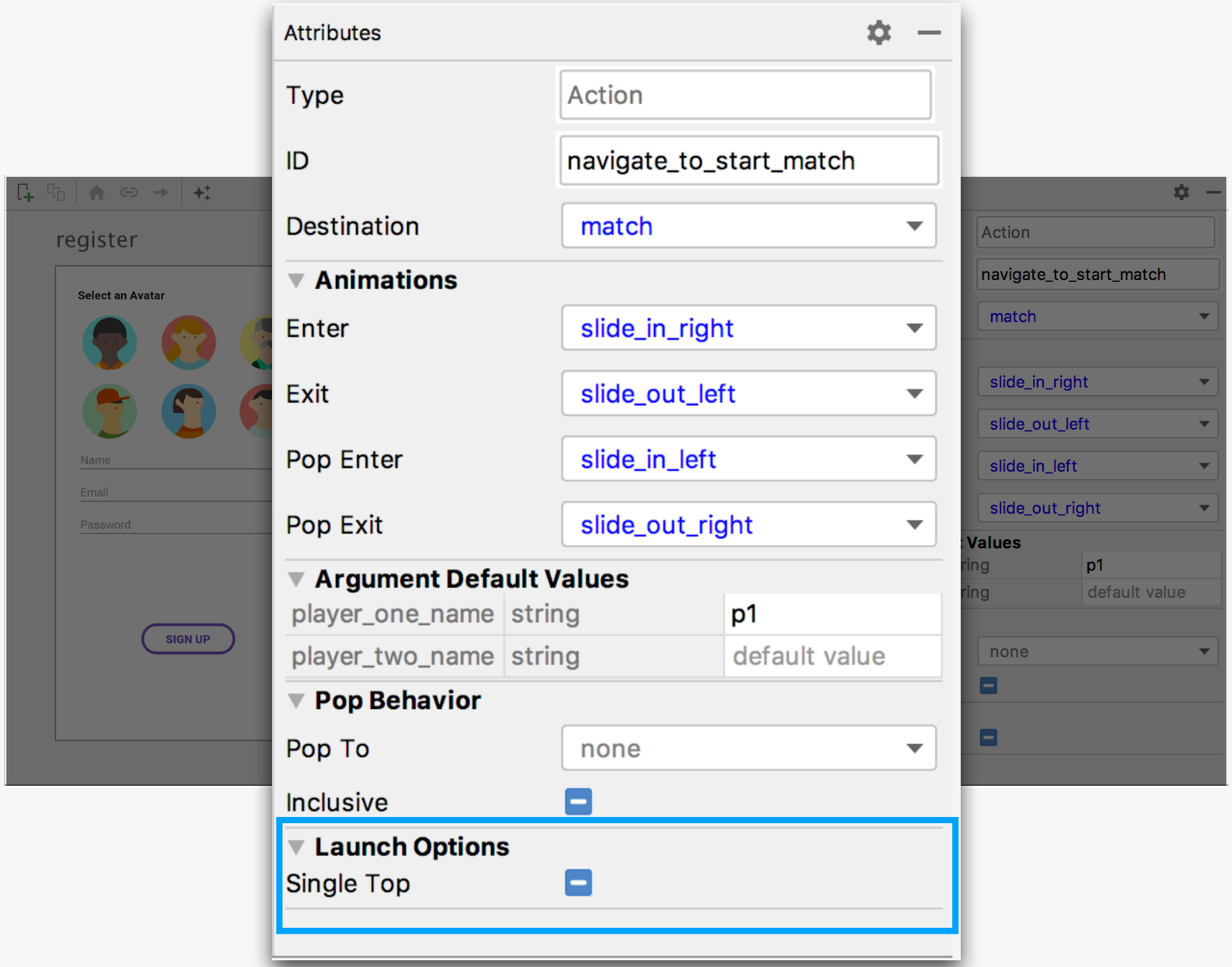
# With Inclusive

```
<fragment android:id="@+id/match" ...>
<fragment android:id="@+id/in_game" ...>

    <action
        android:id="@+id/action_in_game_to_results_winner"
        app:destination="@id/results_winner"
        app:popUpTo="@id/match" />
    <action
        android:id="@+id/action_in_game_to_game_over"
        app:destination="@id/game_over"
        app:popUpTo="@id/in_game"
        app:popUpToInclusive="true"
 />
</fragment>

<fragment
        android:id="@+id/results_winner"
        android:name="com.example.android.navigationsample.ResultsWinner"/>

<fragment
        android:id="@+id/game_over"
        android:name="com.example.android.navigationsample.GameOver" />
```



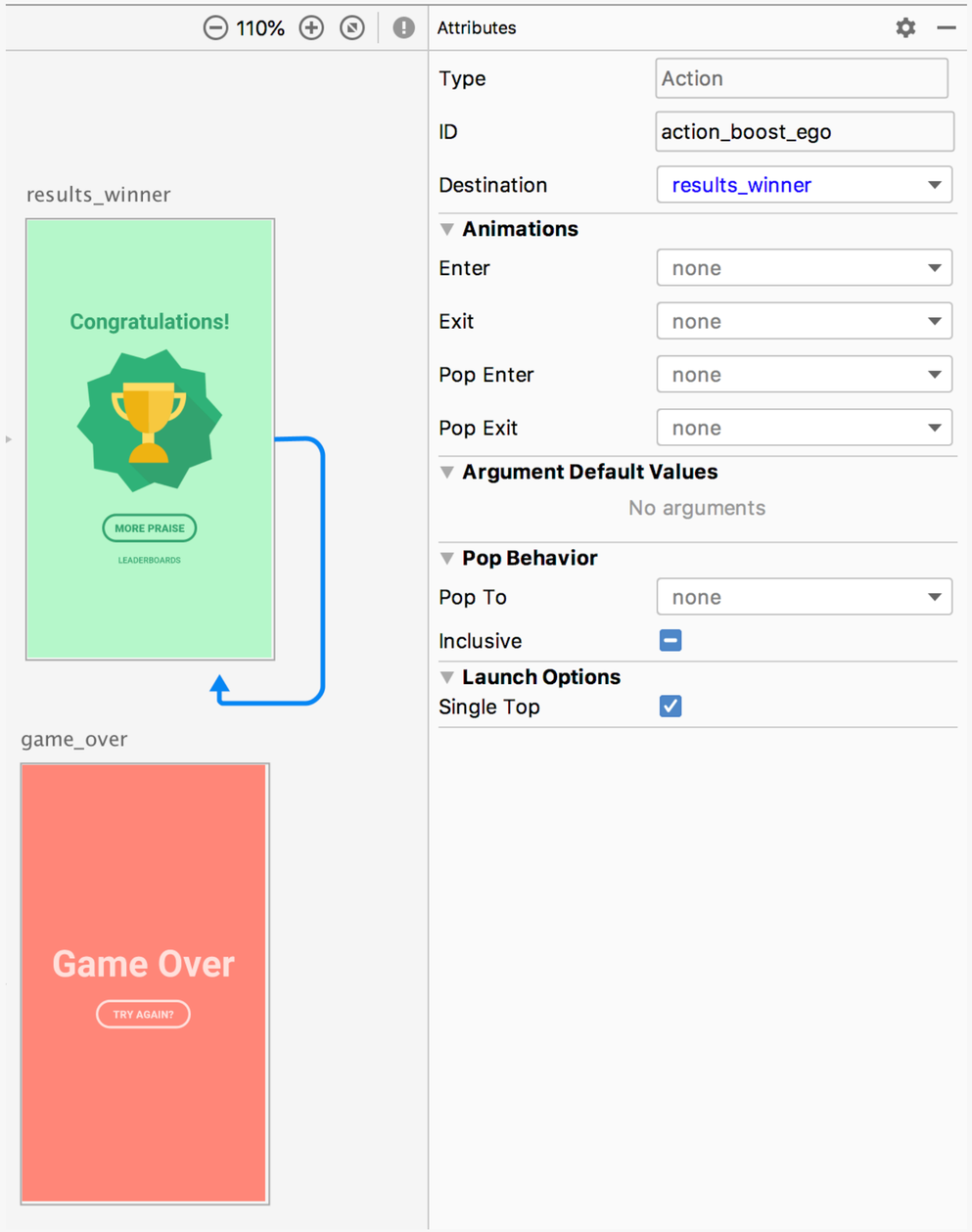
# Actions

- Destination
- Argument Default Values
- Pop Behavior
- **Launch Options**
- Animations



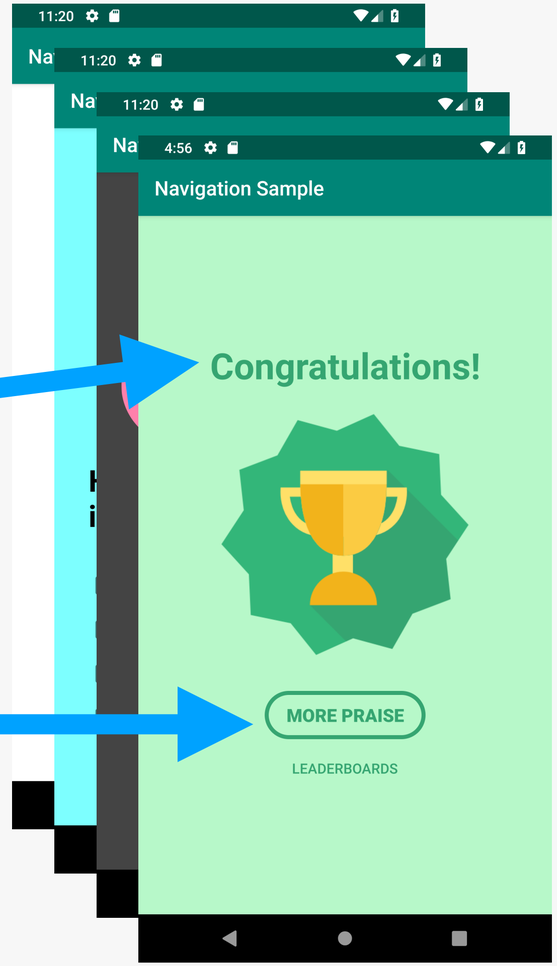
# Single Top

```
<fragment
        android:id="@+id/results_winner"
        android:name=".ResultsWinner"
        tools:layout="@layout/fragment_results_winner">

    <action
        android:id="@+id/action_boost_ego"
        app:launchSingleTop="true"
        app:destination="@id/results_winner" />

</fragment>
```



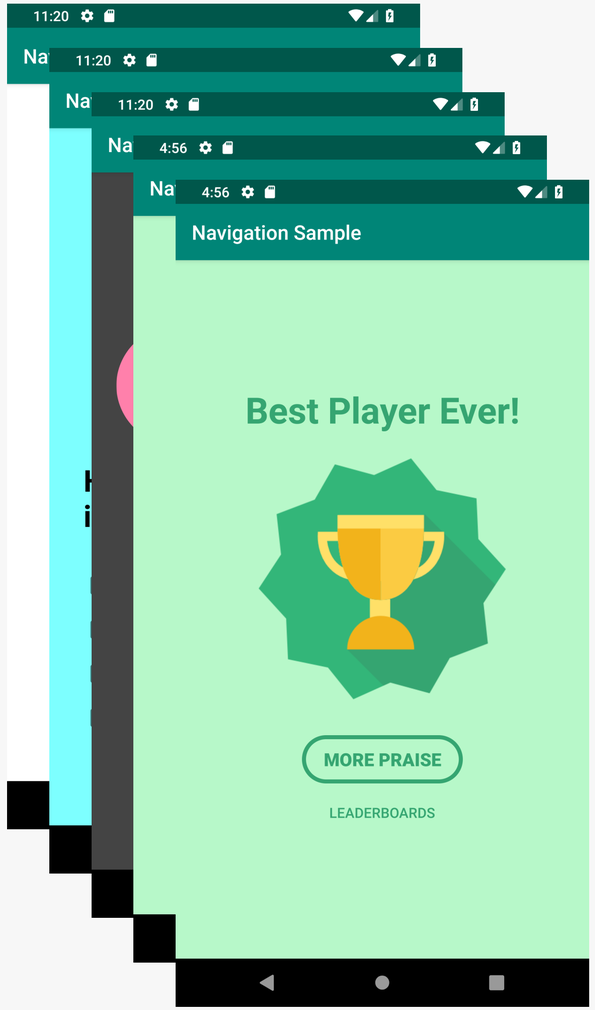
# Single Top

```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```



# Without Single Top

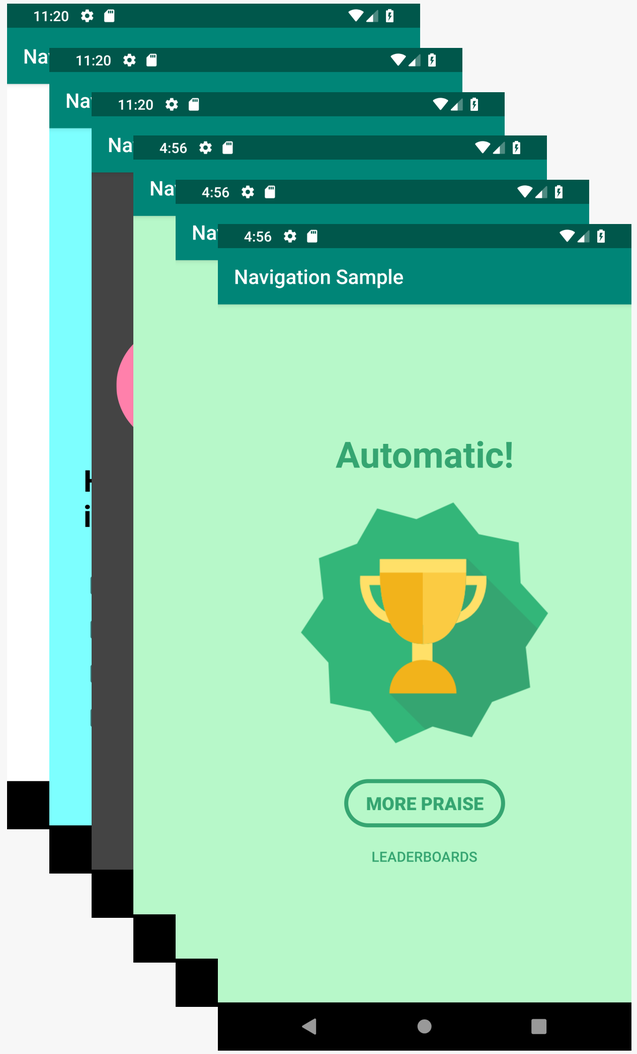
```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```




Click More Praise 1x

# Without Single Top

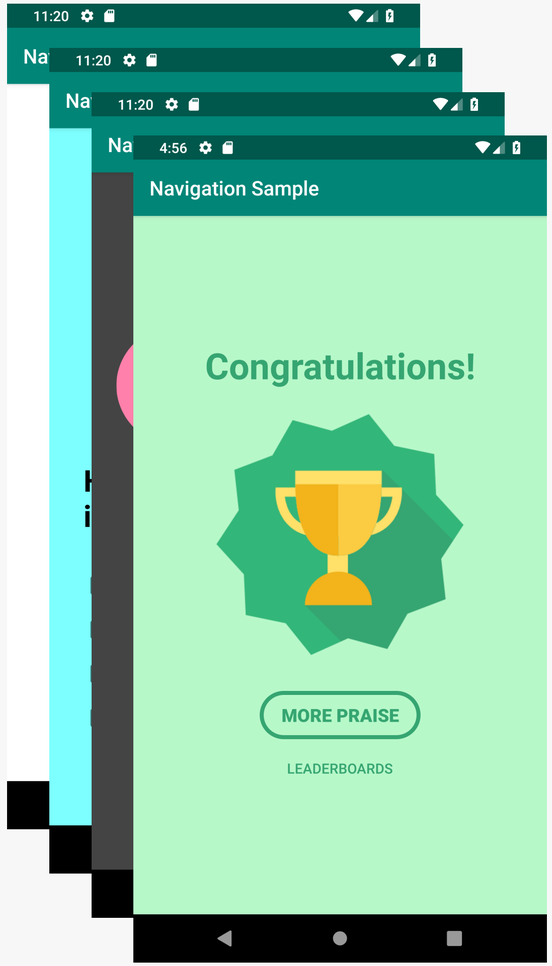
```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```




Click More Praise 2x

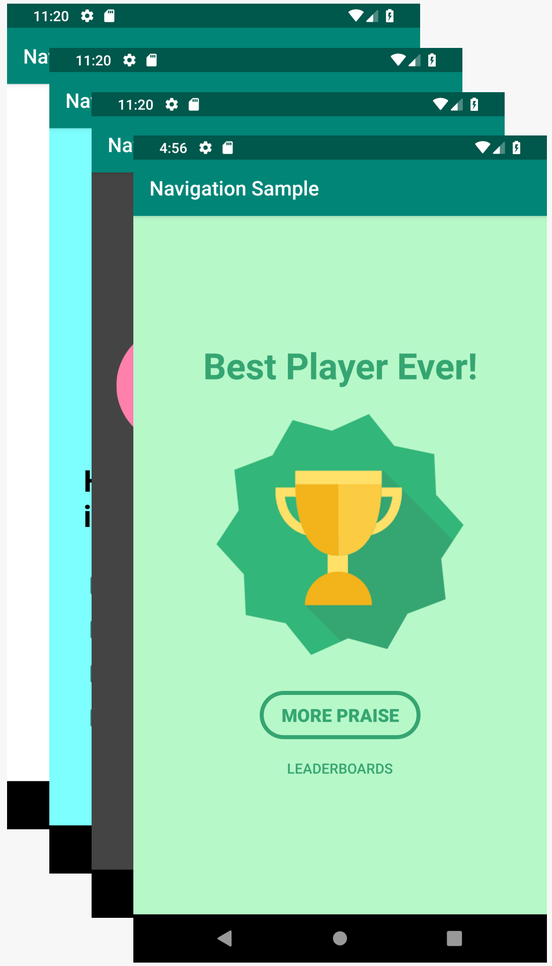
# With Single Top

```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```



# With Single Top

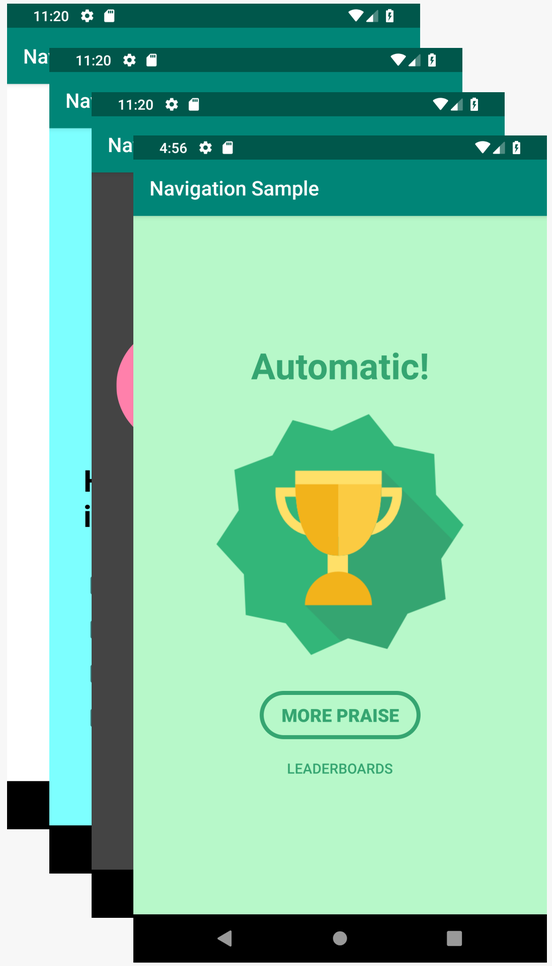
```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```




Click More Praise 1x

# With Single Top

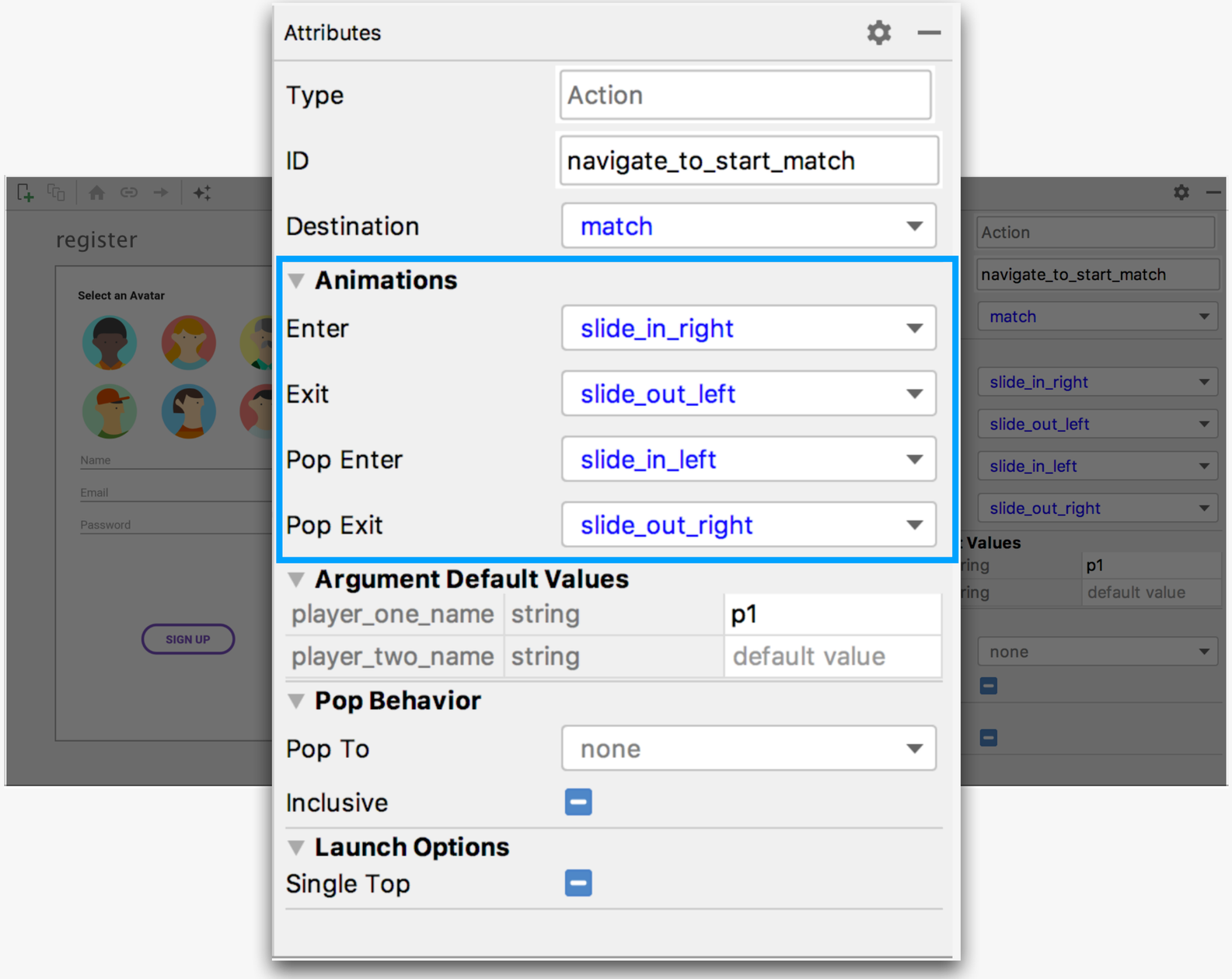
```
class ResultsWinner : Fragment() {

    ...

    override fun onViewCreated(...) {

        view.findViewById<TextView>(
             R.id.positive_reinforcement).text =
                  generateRandomAccolade()

        view.findViewById<Button>(
             R.id.praise_btn).setOnClickListener {
               findNavController().navigate(R.id.action_boost_ego)
        }

    }
```




Click More Praise 2x

# Actions

- Destination
- Argument Default Values
- Pop Behavior
- Launch Options
- **Animations**



# Inside Activity

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id=“@+id/nav_host_fragment”
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="androidx.navigation.fragment.NavHostFragment"
        app:defaultNavHost="true"
        app:navGraph=“@navigation/navigation_graph”/>

</FrameLayout>
```

# Include NavHostFragment

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <fragment
        android:id="@+id/nav_host_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="androidx.navigation.fragment.NavHostFragment"
        app:defaultNavHost="true"
        app:navGraph="@navigation/navigation_graph" />

</FrameLayout>
```

# HOSTING NAVIGATION

Navigation Graph

NavHostFragment

**NavHostFragment Initialization**

1. **Create NavController**
2. Registers Fragment as a Destination Type
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

# HOSTING NAVIGATION

Navigation Graph

NavHostFragment

**NavHostFragment Initialization**

1. Create NavController
2. **Registers Fragment as a Destination Type**
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

# HOSTING NAVIGATION

Navigation Graph

NavHostFragment

**NavHostFragment Initialization**

1. Create NavController
2. Registers Fragment as a Destination Type
3. ***Restore Navigation State***
4. Set Graph on NavController
5. Navigate to first Destination

# HOSTING NAVIGATION

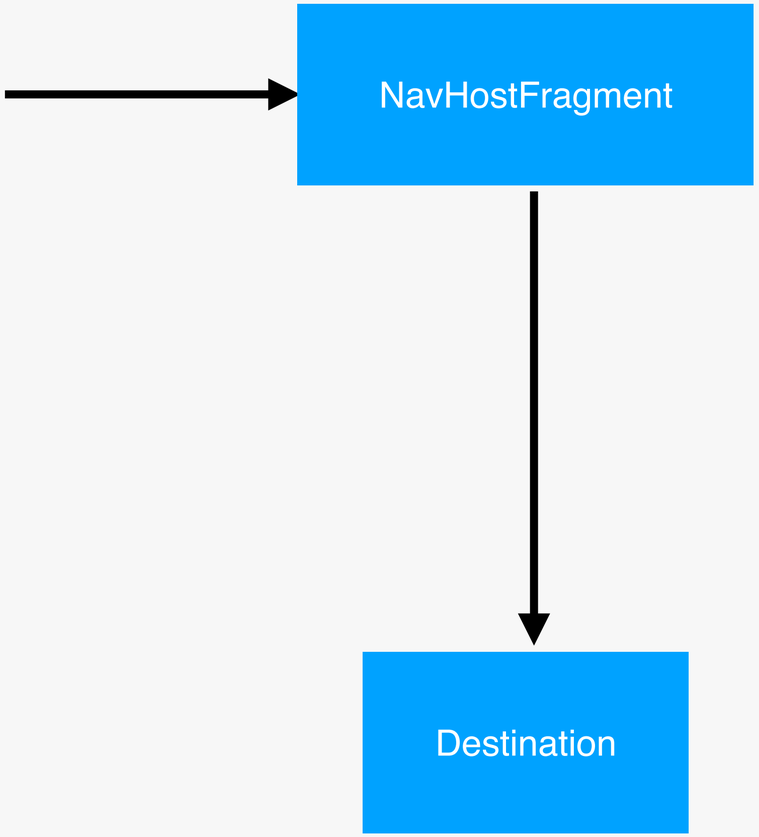
Navigation Graph

NavHostFragment

**NavHostFragment Initialization**

1. Create NavController
2. Registers Fragment as a Destination Type
3. *Restore Navigation State*
4. **Set Graph on NavController**
5. Navigate to first Destination

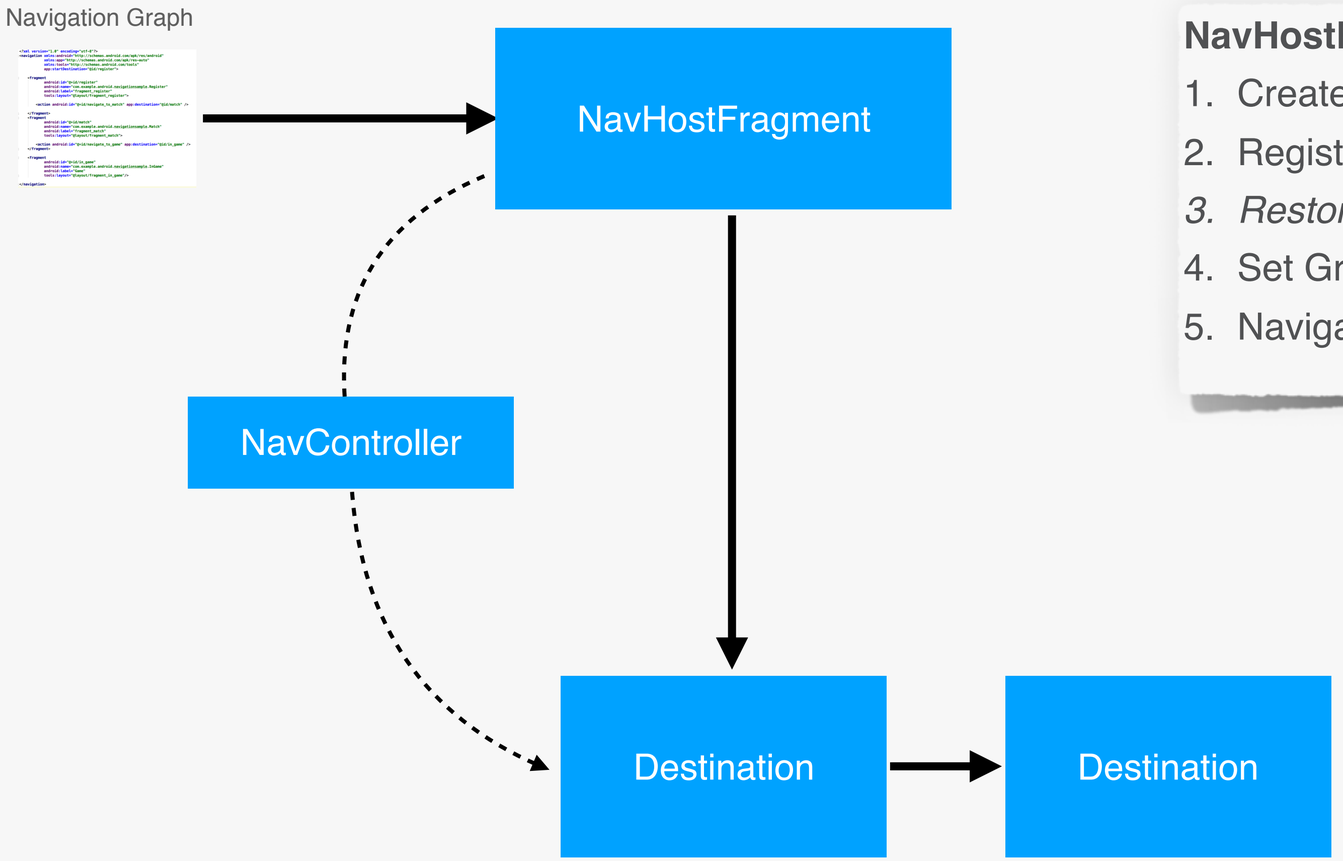
Navigation Graph




**NavHostFragment Initialization**

1. Create NavController
2. Registers Fragment as a Destination Type
3. *Restore Navigation State*
4. Set Graph on NavController
5. **Navigate to first Destination**

```
<navigation xmlns:android="..."
    xmlns:app="..."
    xmlns:tools=..."
    android:id="@+id/weather_graph"
    app:startDestination="@id/weather_list">

    <fragment
        android:id="@+id/weather_list"
        android:name=".WeatherListFragment"
        android:label="Weather Now"
        tools:layout="@layout/weather_list_fragment">
```

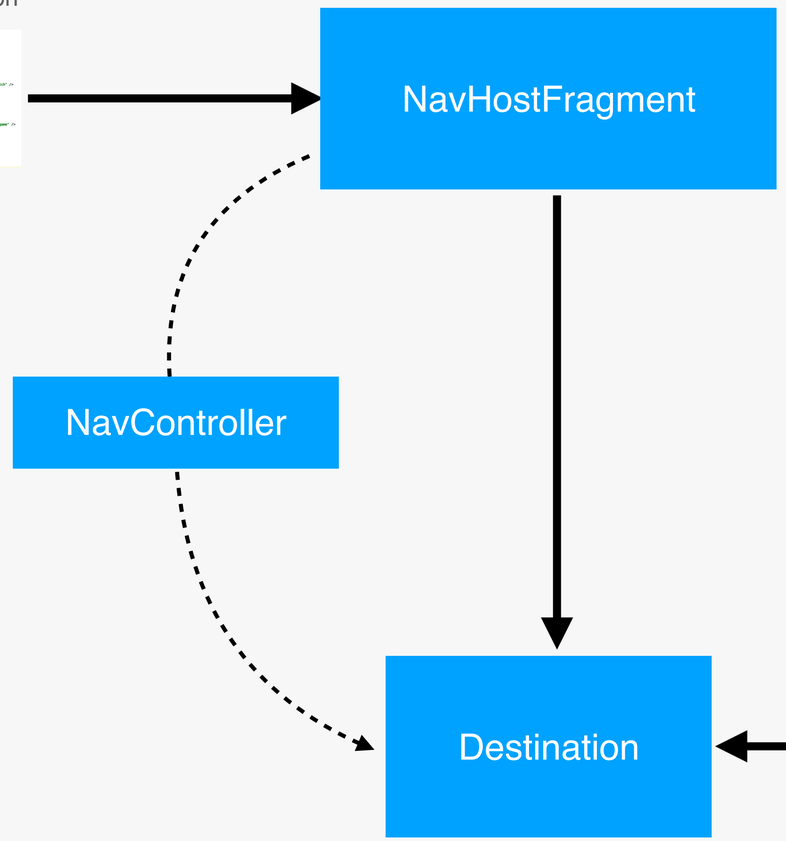



**NavHostFragment Initialization**

1. Create NavController
2. Registers Fragment as a Destination Type
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

```
navController.navigate(...)
```

Navigation Graph




**NavHostFragment Initialization**

1. Create NavController
2. Registers Fragment as a Destination Type
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

```
navController.popBackStack()
```

# Accessing NavController

Fragment

```
class MyDestinationFragment : Fragment()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val controller: NavController = findNavController()
    }
}
```

Activity

```
class MainActivity : AppCompatActivity() {

    private lateinit var navController: NavController

     override fun onCreate(savedInstanceState: Bundle?) {
         super.onCreate(savedInstanceState)
         setContentView(R.layout.activity_main)

         val controller: NavController = findNavController(R.id.nav_host_fragment)
}
```

# Navigate to Another Destination

```
class MyDestinationFragment : Fragment()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val controller: NavController = findNavController()

        view.findViewById<Button>(R.id.next)).setOnClickListener {

            controller.navigate(R.id.action_id)

        }

    }
}
```

# Pop Back to Previous Destination

```
class MyDestinationFragment : Fragment()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val controller: NavController = findNavController()

        view.findViewById<Button>(R.id.next)).setOnClickListener {

            controller.navigate(R.id.action_id)

        }

        view.findViewById<Button>(R.id.previous)).setOnClickListener {

            controller.popBackStack()


        }

    }
}
```

# Operations

### Navigate to Actions or Destinations by Resource ID

```
fun navigate(@IdRes resId: Int)
fun navigate(@IdRes resId: Int, args: Bundle?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?, navigatorExtras: Navigator.Extras?)
```

### Navigate using Safe Args generated classes

```
fun navigate(directions: NavDirections)
fun navigate(directions: NavDirections, navOptions: NavOptions?)
fun navigate(directions: NavDirections, navigatorExtras: Navigator.Extras)
```

### Back Navigation

```
fun popBackStack(): Boolean
fun popBackStack(@IdRes destinationId: Int, inclusive: Boolean): Boolean
```

# Operations

**Navigate to Actions or Destinations by Resource ID**


```
fun navigate(@IdRes resId: Int)
fun navigate(@IdRes resId: Int, args: Bundle?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?, navigatorExtras: Navigator.Extras?)
```

**Navigate using Safe Args generated classes**

```
fun navigate(directions: NavDirections)
fun navigate(directions: NavDirections, navOptions: NavOptions?)
fun navigate(directions: NavDirections, navigatorExtras: Navigator.Extras)
```

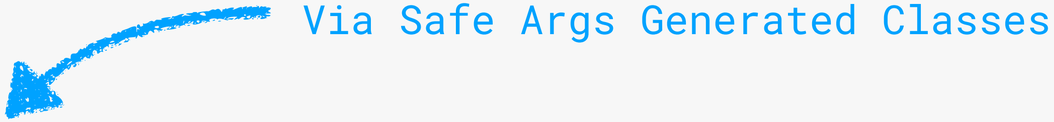
**Back Navigation**

```
fun popBackStack(): Boolean
fun popBackStack(@IdRes destinationId: Int, inclusive: Boolean): Boolean
```

# Operations

**Navigate to Actions or Destinations by Resource ID**

```
fun navigate(@IdRes resId: Int)
fun navigate(@IdRes resId: Int, args: Bundle?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?, navigatorExtras: Navigator.Extras?)
```




**Navigate using Safe Args generated classes**

```
fun navigate(directions: NavDirections)
fun navigate(directions: NavDirections, navOptions: NavOptions?)
fun navigate(directions: NavDirections, navigatorExtras: Navigator.Extras)
```

**Back Navigation**

```
fun popBackStack(): Boolean
fun popBackStack(@IdRes destinationId: Int, inclusive: Boolean): Boolean
```

# Operations

### Navigate to Actions or Destinations by Resource ID

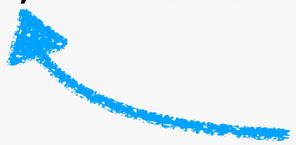
```
fun navigate(@IdRes resId: Int)
fun navigate(@IdRes resId: Int, args: Bundle?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?)
fun navigate(@IdRes resId: Int, args: Bundle?, navOptions: NavOptions?, navigatorExtras: Navigator.Extras?)
```

### Navigate using Safe Args generated classes

```
fun navigate(directions: NavDirections)
fun navigate(directions: NavDirections, navOptions: NavOptions?)
fun navigate(directions: NavDirections, navigatorExtras: Navigator.Extras)
```

### Back Navigation

```
fun popBackStack(): Boolean
fun popBackStack(@IdRes destinationId: Int, inclusive: Boolean): Boolean
```



Pop back to destination up the stack

# Safe Args

# Safe Args Dependencies

## Safe Args

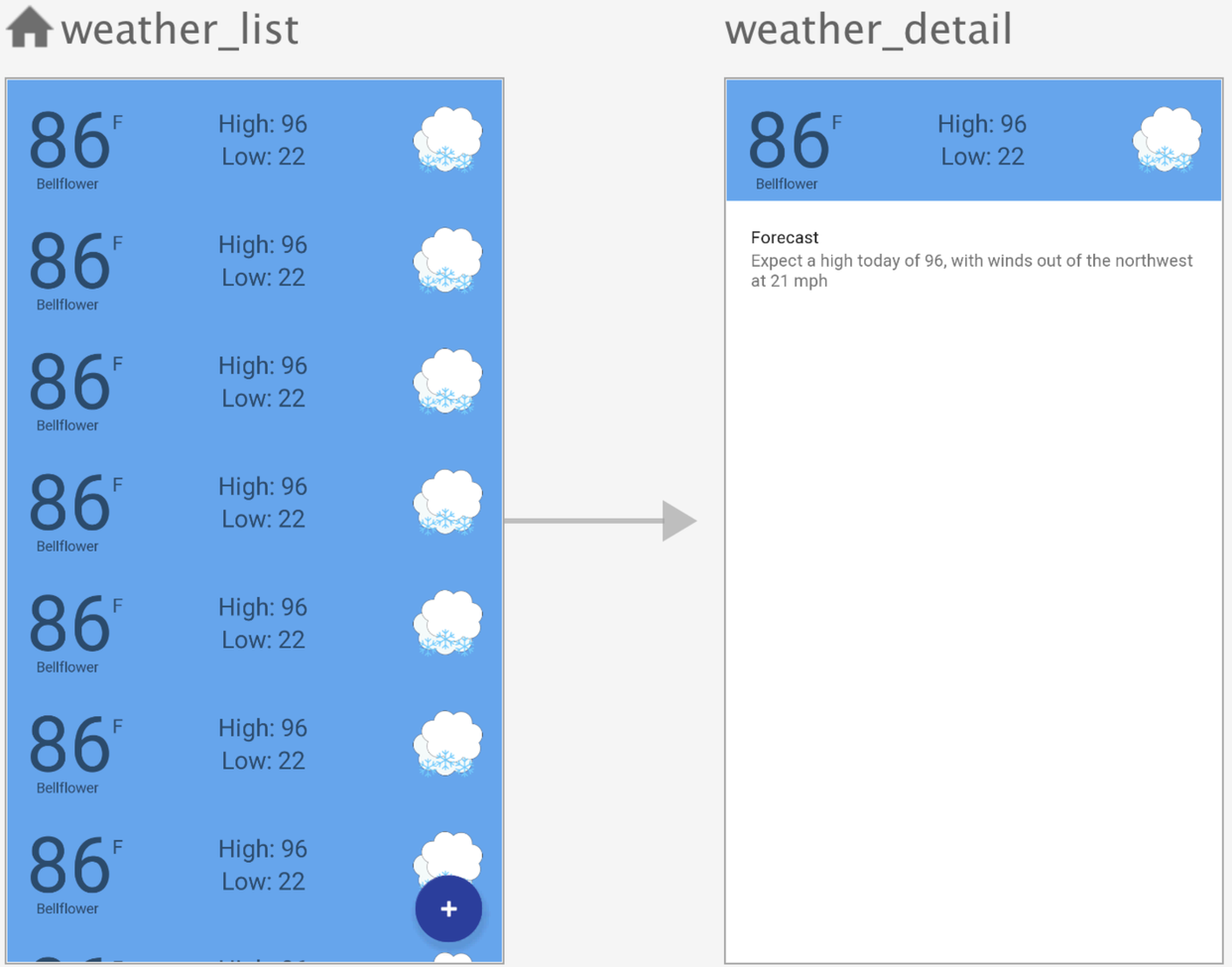
Project build.gradle:

```
dependencies {
    ...
    classpath “android.arch.navigation:navigation-safe-args-gradle-plugin:$version"
}
```

Module build.gradle:

```
apply plugin: 'androidx.navigation.safeargs'
```

# Generated Directions + Arguments




**Destination Requires**

- Zip Code
- Use Fahrenheit Flag

# Generated Arguments + Directions

```
<navigation
    android:id="@+id/weather_nav"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list" android:name=".WeatherListFragment">
        <action
            android:id="@+id/navigate_to_weather_details"
            app:destination="@id/weather_detail" >

            <argument android:defaultValue="true" android:name="use_fahrenheit" />

        </action>
    </fragment>
    <fragment android:id=“@+id/weather_detail"
              android:name="com.acme.weather.view.WeatherDetailFragment">

        <argument android:name=“zip_code” app:argType=“string" />

        <argument android:name="use_fahrenheit" app:argType="boolean" />

    </fragment>
</navigation>
```

# Generated Arguments

```
public class WeatherDetailFragmentArgs implements NavArgs {

    // Create Args from Bundle or output to Bundle
    public static WeatherDetailFragmentArgs fromBundle(@NonNull Bundle bundle) {}
    public Bundle toBundle() {}

    // Type Safe Property Access
    public String getZipCode()
    public boolean getUseFahrenheit()

    // Builder for type safe argument creation
    public static class Builder {}
}
```

Navigation Graph

```
<fragment android:id="@+id/weather_detail"
          android:name="com.acme.weather.view.WeatherDetailFragment">

    <argument android:name="zip_code" app:argType="string" />
    <argument android:name="use_fahrenheit" app:argType="boolean" />

</fragment>
```

# Generated Arguments

Construct from android.os.Bundle

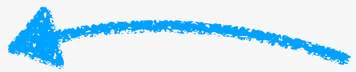
```
public class WeatherDetailFragmentArgs implements NavArgs {

    // Create Args from Bundle or output to Bundle
    public static WeatherDetailFragmentArgs fromBundle(@NonNull Bundle bundle) {}
    public Bundle toBundle() {}

    // Type Safe Property Access
    public String getZipCode()
    public boolean getUseFahrenheit()

    // Builder for type safe argument creation
    public static class Builder {}
}
```

Navigation Graph

```
<fragment android:id="@+id/weather_detail"
          android:name="com.acme.weather.view.WeatherDetailFragment">

    <argument android:name="zip_code" app:argType="string" />
    <argument android:name="use_fahrenheit" app:argType="boolean" />

</fragment>
```

# Generated Arguments

```
public class WeatherDetailFragmentArgs implements NavArgs {

    // Create Args from Bundle or output to Bundle
    public static WeatherDetailFragmentArgs fromBundle(@NonNull Bundle bundle) {}
    public Bundle toBundle() {}

    // Type Safe Property Access
    public String getZipCode()
    public boolean getUseFahrenheit()

    // Builder for type safe argument creation
    public static class Builder {}
}
```



Access args as properties by type

Navigation Graph

```
<fragment android:id="@+id/weather_detail"
          android:name="com.acme.weather.view.WeatherDetailFragment">

    <argument android:name="zip_code" app:argType="string" />
    <argument android:name="use_fahrenheit" app:argType="boolean" />

</fragment>
```

# Accessing Args in Destination

```
val args =
WeatherDetailFragmentArgs.fromBundle(arguments!!)

– or –

val args: WeatherDetailFragmentArgs by navArgs()
```

## Args

```
public class WeatherDetailFragmentArgs implements NavArgs {

   // Create Args from Bundle or output to Bundle
   public static WeatherDetailFragmentArgs fromBundle(@NonNull Bundle bundle) {}
   public Bundle toBundle() {}

   // Type Safe Property Access
   public String getZipCode()
   public boolean getUseFahrenheit()

    // Builder for type safe argument creation
   public static class Builder {}
}
```

## Navigation Graph

```
<fragment android:id=“@+id/weather_detail"
          android:name="com.acme.weather.view.WeatherDetailFragment">

    <argument android:name=“zip_code” app:argType=“string" />
    <argument android:name="use_fahrenheit" app:argType="boolean" />

</fragment>
```

# Generated Arguments + Directions

```
<navigation
    android:id="@+id/weather_nav"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list" android:name=".WeatherListFragment">
        <action
            android:id="@+id/navigate_to_weather_details"
            app:destination="@id/weather_detail" >

            <argument android:defaultValue="true" android:name="use_fahrenheit" />

        </action>
    </fragment>
    <fragment android:id=“@+id/weather_detail"
              android:name="com.acme.weather.view.WeatherDetailFragment">

        <argument android:name=“zip_code” app:argType=“string" />

        <argument android:name="use_fahrenheit" app:argType="boolean" />

    </fragment>
</navigation>
```

# Generated Arguments + Directions

```
<navigation
    android:id="@+id/weather_nav"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list" android:name=".WeatherListFragment">
        <action
            android:id="@+id/navigate_to_weather_details"
            app:destination="@id/weather_detail" >

            <argument android:defaultValue="true" android:name="use_fahrenheit" />

        </action>
    </fragment>
    <fragment android:id=“@+id/weather_detail"
              android:name="com.acme.weather.view.WeatherDetailFragment">

        <argument android:name=“zip_code” app:argType=“string" />

        <argument android:name="use_fahrenheit" app:argType="boolean" />

    </fragment>
</navigation>
```

# Generated Directions

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(@NonNull String zipCode) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Generated Directions

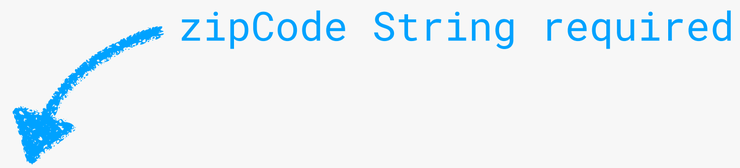
navigate_to_weather_details

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(@NonNull String zipCode) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Generated Directions




```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(@NonNull String zipCode) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Generated Directions

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(@NonNull String zipCode) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

useFahrenheit can be overridden

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Generated Directions

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(@NonNull String zipCode) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

All navigate(...) details

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
val directions =
        WeatherListFragmentDirections
           .navigateToWeatherDetails(zipCode)
                .setUseFahrenheit(true)

findNavController().navigate(directions)
```

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
findNavController().navigate(
        toWeatherDetails(zipCode)
                .setUseFahrenheit(true))
```

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
val directions =
        WeatherListFragmentDirections
           .navigateToWeatherDetails(zipCode)
                .setUseFahrenheit(true)

findNavController().navigate(directions)
```

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
val directions =
        WeatherListFragmentDirections
            .navigateToWeatherDetails(zipCode)
                .setUseFahrenheit(true)

findNavController().navigate(directions)
```

Static import of NavDirections class

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
val directions =
        WeatherListFragmentDirections
            .navigateToWeatherDetails(zipCode)
                .setUseFahrenheit(true)

findNavController().navigate(directions)
```

Static import of NavDirections class

Change name of action to "to_weather_details"

```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

```
<action
   android:id="@+id/navigate_to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Navigating with Directions

```
findNavController().navigate(
        toWeatherDetails(zipCode)
                .setUseFahrenheit(true))
```

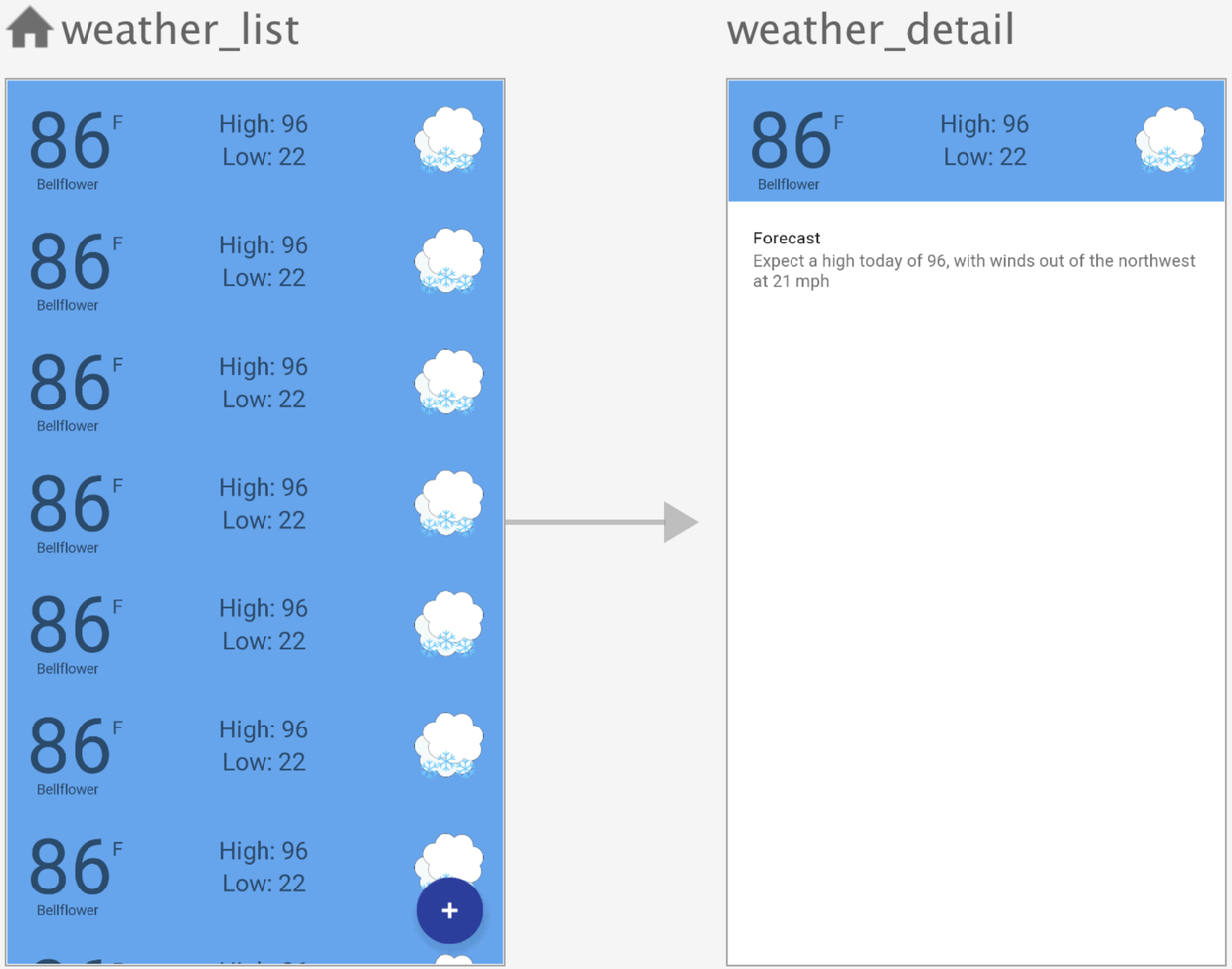
```
public class WeatherListFragmentDirections {

  // Static factory for each Action tied to this destination
  public static NavigateToWeatherDetails navigateToWeatherDetails(...) {}

  public static class NavigateToWeatherDetails implements NavDirections {

     // Setters to override default argument values
     public NavigateToWeatherDetails setUseFahrenheit(boolean useFahrenheit) {}

     // Accessors for Bundle arguments, Action (…and NavOptions)
     public Bundle getArguments() {}
     public int getActionId() {}
}
```

Navigation Graph

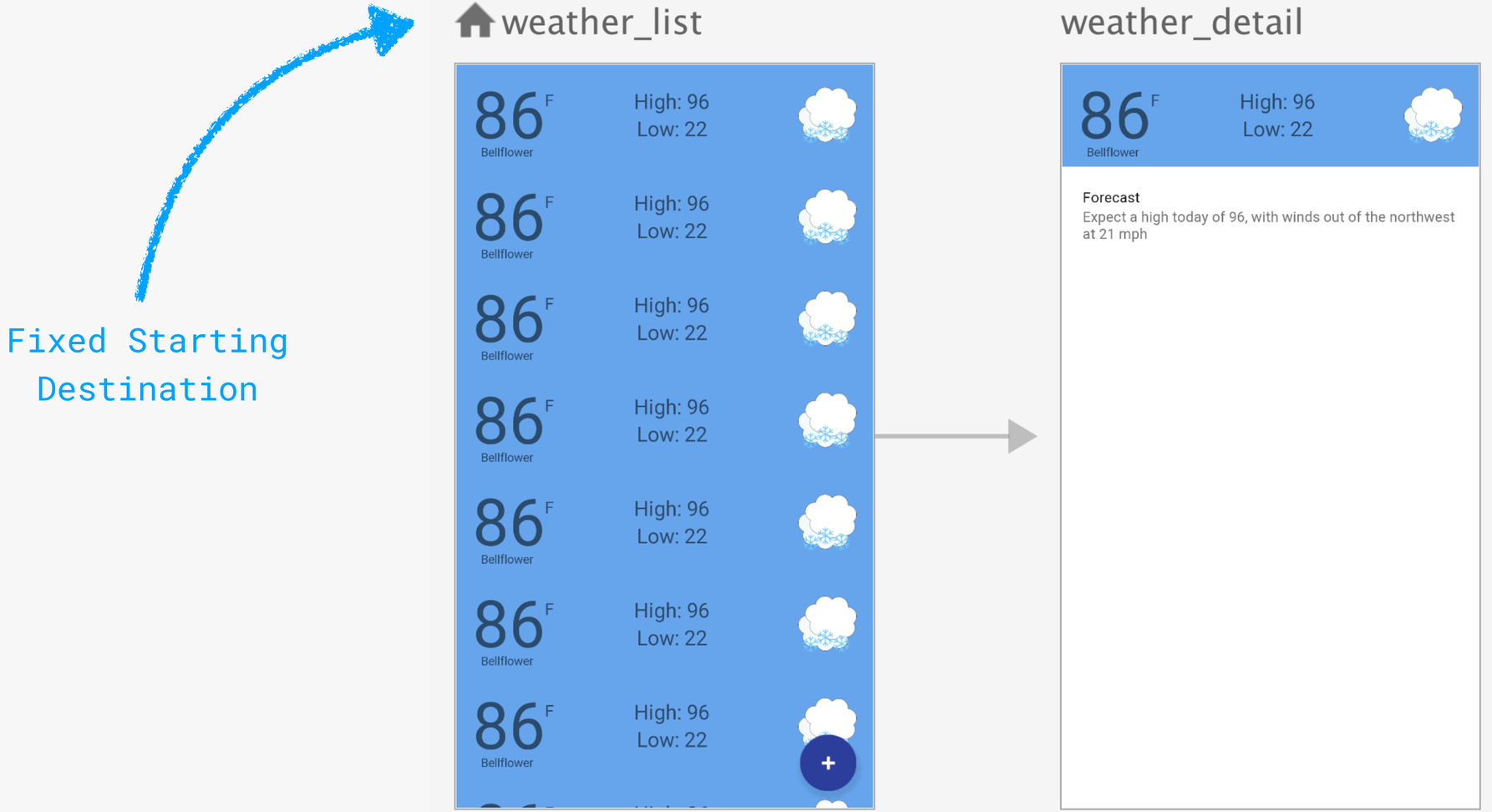
```
<action
   android:id="@+id/to_weather_details"
   app:destination="@id/weather_detail" >

    <argument android:defaultValue="true" android:name="use_fahrenheit" />

</action>
```

# Conditional Navigation

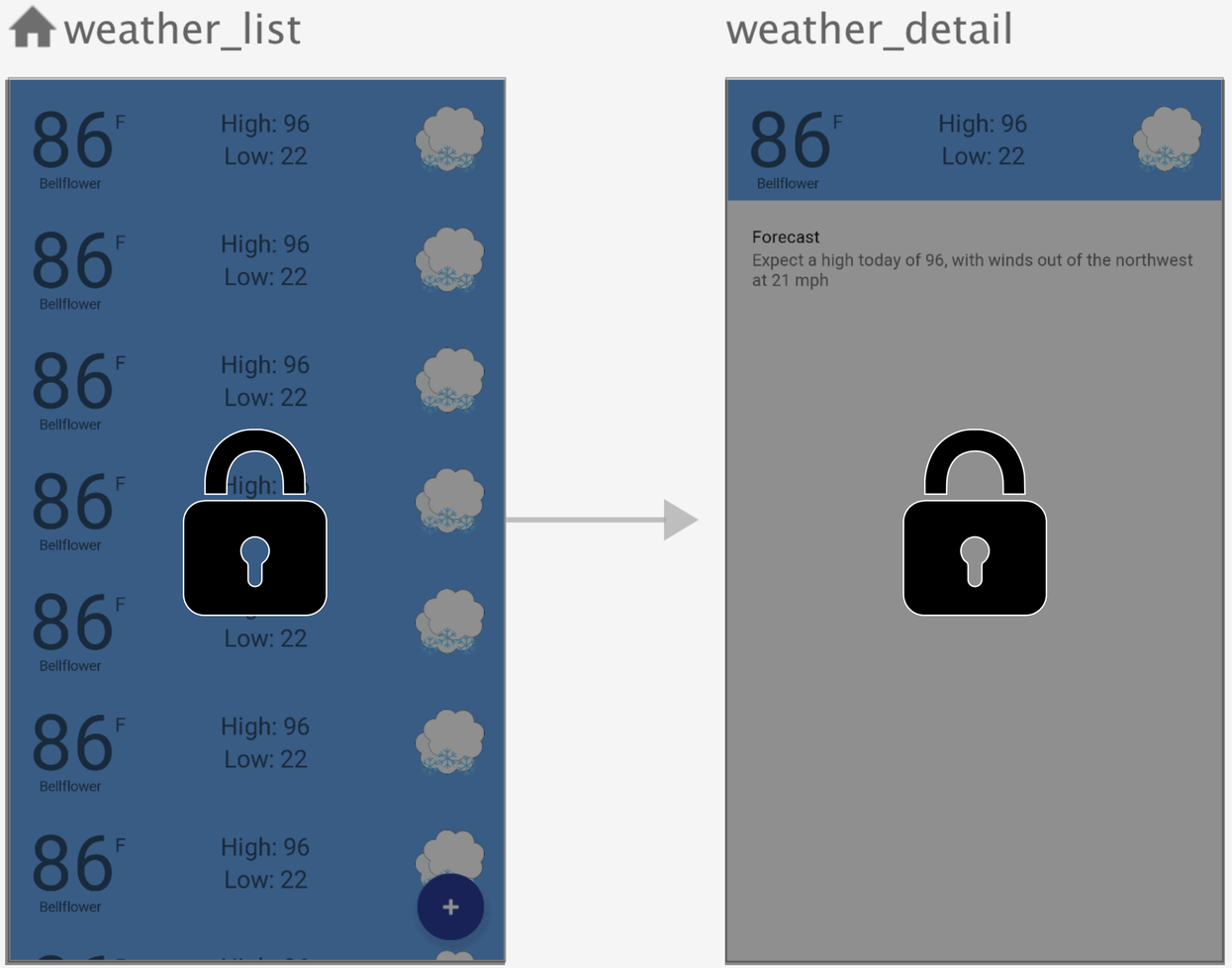
# CONDITIONAL NAVIGATION



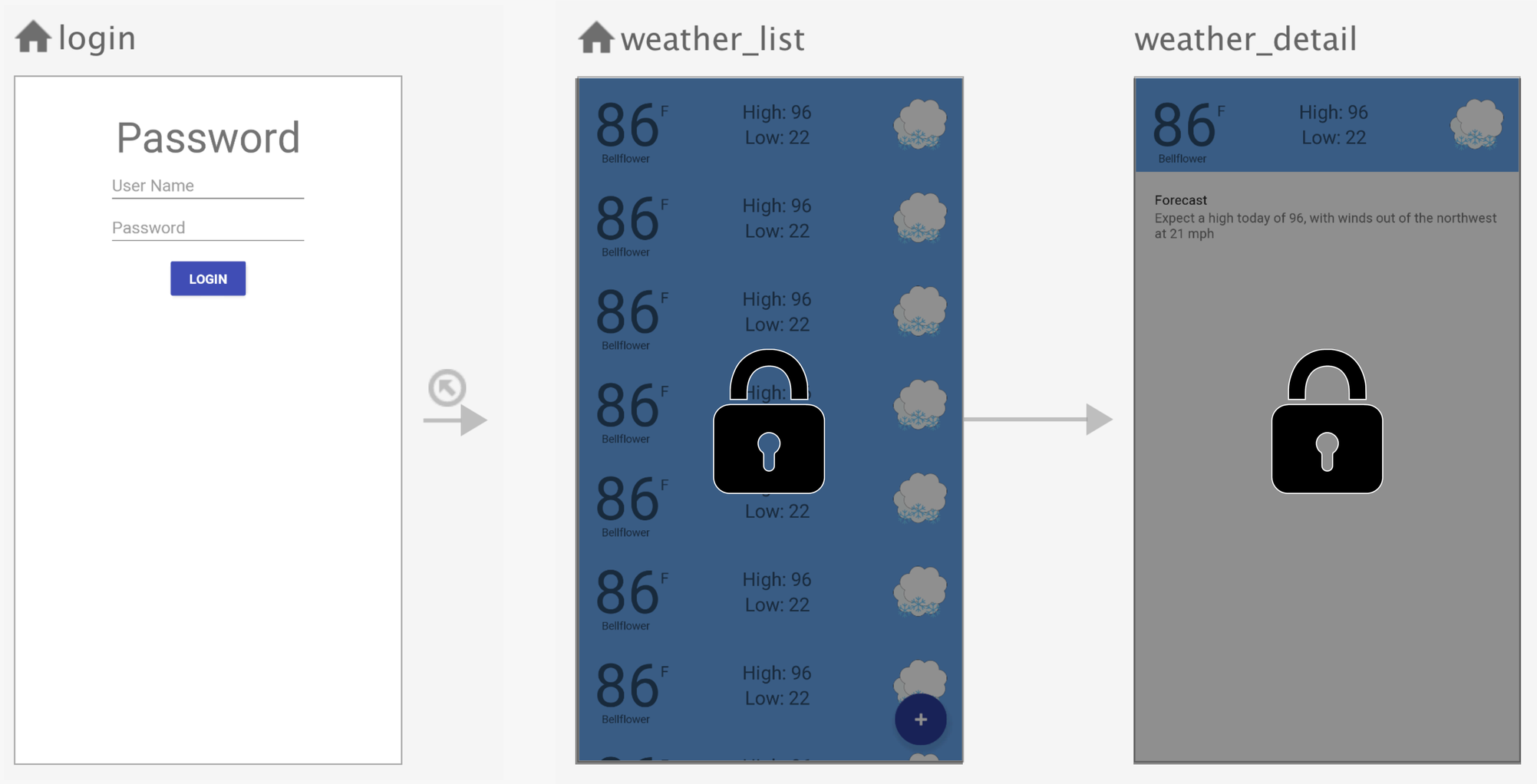
# CONDITIONAL NAVIGATION



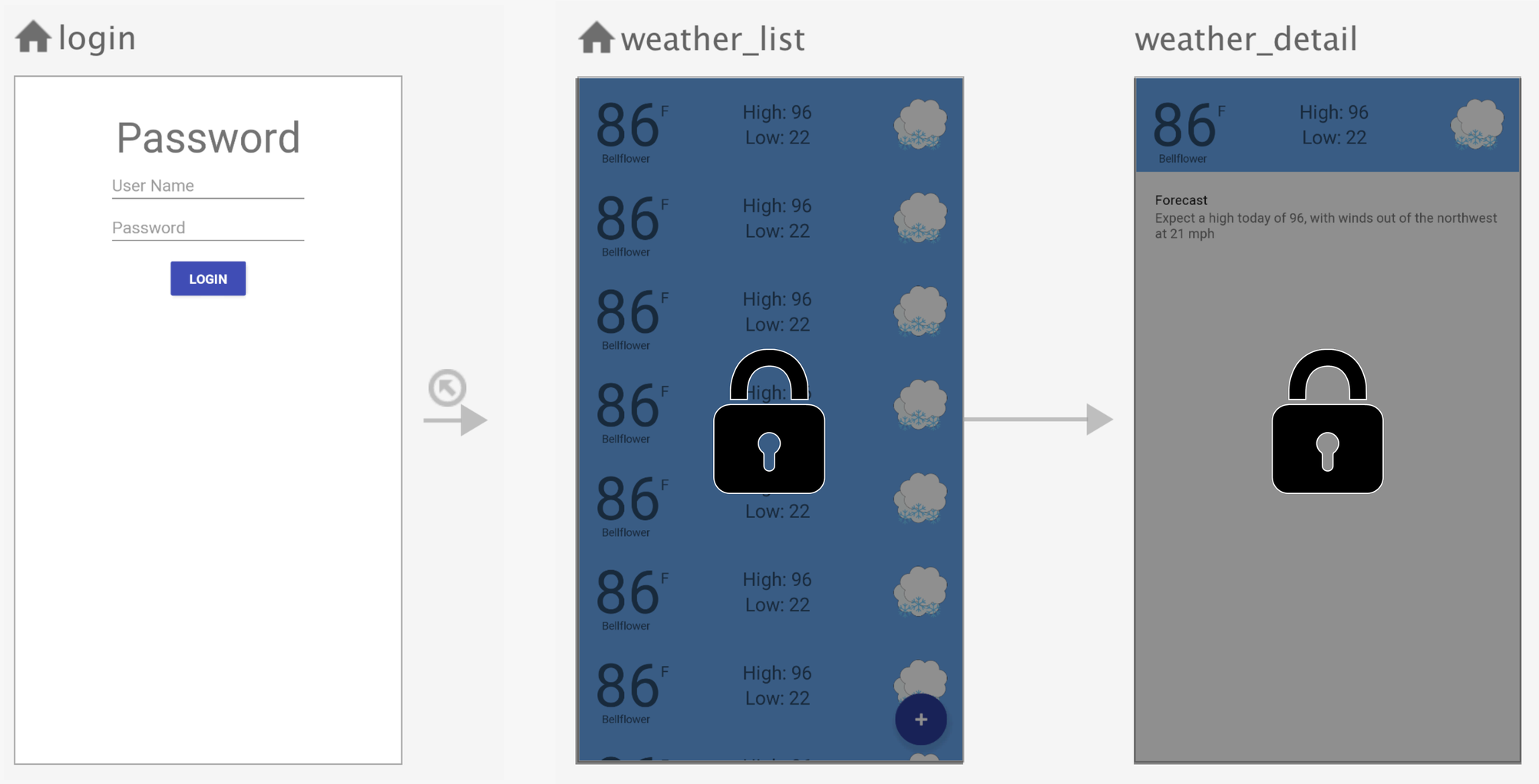
# CONDITIONAL NAVIGATION



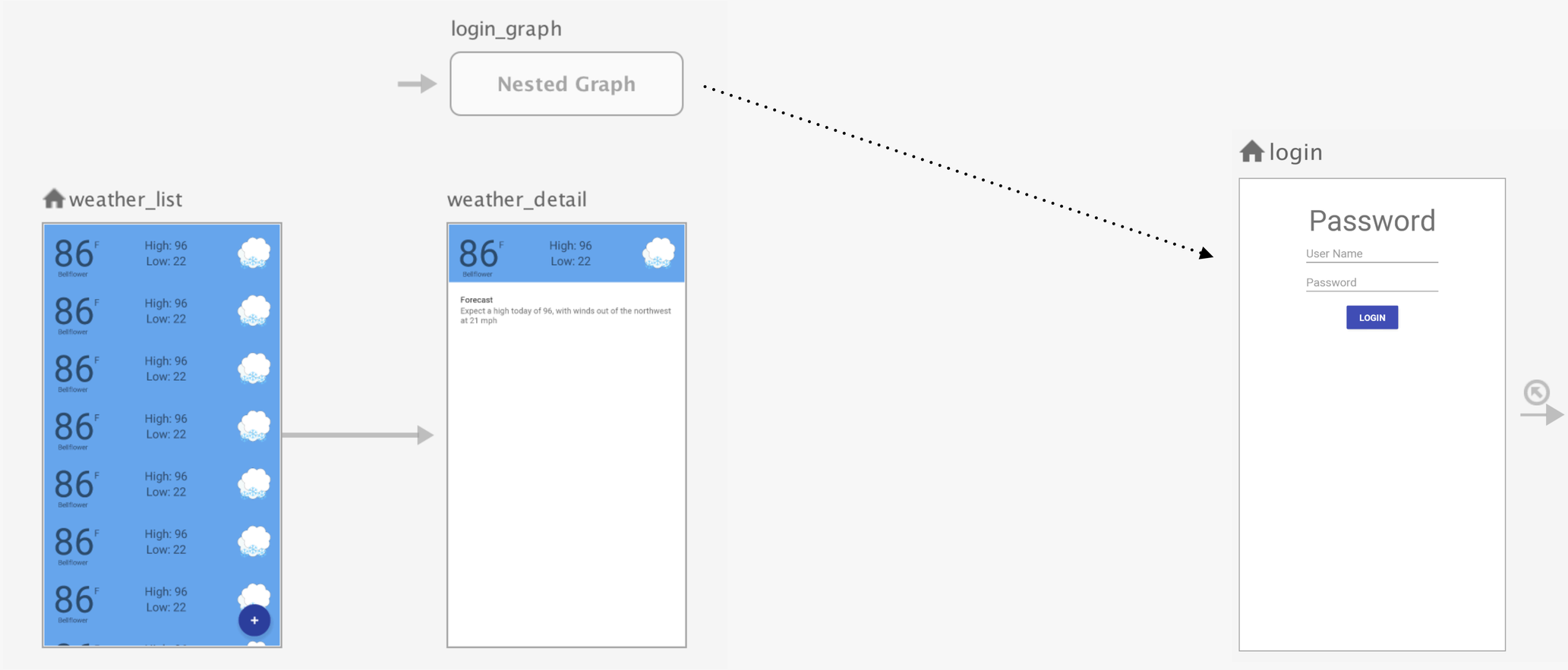
# Conditional Starting Destination



# First Time User Experience



# First Time User Experiences




Main Navigation Graph

Login Graph

# First Time User Experiences

```
<navigation ...
    android:id="@+id/weather_graph"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list">...</fragment>
    <fragment android:id="@+id/weather_detail">...</fragment>

    <navigation
        android:id="@+id/login_graph"
        android:label="Login"
        app:startDestination="@id/login">

        <fragment
            android:id="@+id/login"
            android:name="com.acme.weather.security.view.Login"
            android:label="Log In" />

    </navigation>

</navigation>
```

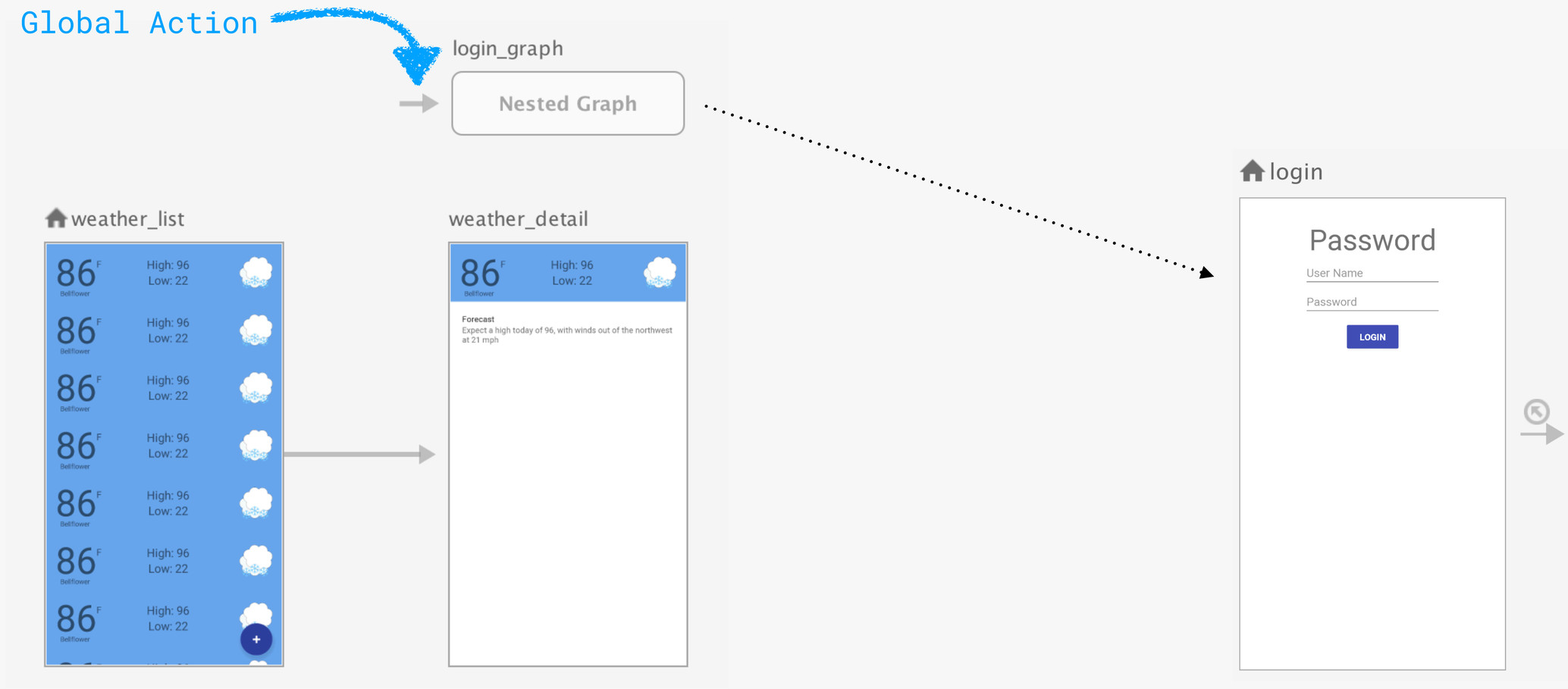
Nested Graph

# First Time User Experiences

```
<navigation ...
    android:id="@+id/weather_graph"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list">...</fragment>
    <fragment android:id="@+id/weather_detail">...</fragment>

    <include app:graph="@navigation/login_graph" />

</navigation>
```

Include as Separate Graph

# First Time User Experiences




Main Navigation Graph

Login Graph

# Global Action

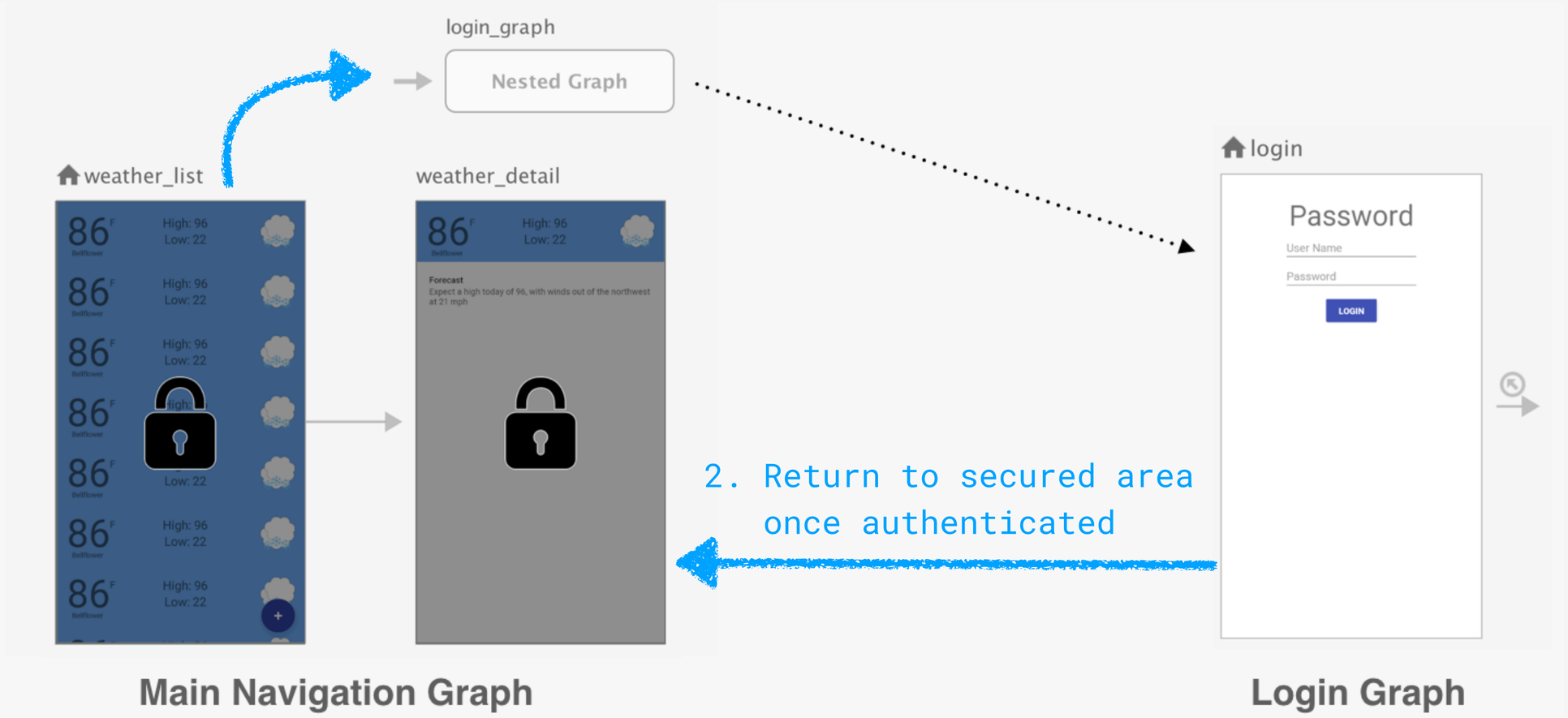
```
<navigation ...
    android:id="@+id/weather_graph"
    app:startDestination="@id/weather_list">

    <fragment android:id="@+id/weather_list">...</fragment>
    <fragment android:id="@+id/weather_detail">...</fragment>


    <include app:graph="@navigation/login_graph" />


    <action
        android:id="@+id/action_global_login"
        app:destination="@id/login_graph" />

</navigation>
```

Global Action is Global because it is not tied to a destination

It is a peer to destinations in this navigation graph **and** accessible by any destination inside this graph

# First Time User Experiences

1. If user is unauthenticated, redirect to login



# Global Navigation

```
class WeatherDetailFragment : SecureFragment() {

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

        super.onViewCreated(view, savedInstanceState)


    }
}
```

# Global Navigation

```
abstract class SecureFragment : Fragment() {

    val authenticationViewModel by activityViewModels<AuthenticationViewModel>()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) { ... }

    private fun popBackStackOrExit() { ... }
}
```

# Global Navigation

```
abstract class SecureFragment : Fragment() {

    val authenticationViewModel by activityViewModels<AuthenticationViewModel>()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        authenticationViewModel.authenticationStatus.observe(viewLifecycleOwner, Observer { authStatus ->
            when(authStatus) {
                UNAUTHENTICATED -> navController.navigate(actionGlobalLogin())
                USER_DECLINED -> popBackStackOrExit()
                else -> Unit
            }
        })
    }

    private fun popBackStackOrExit() { ... }
}
```

# Global Navigation

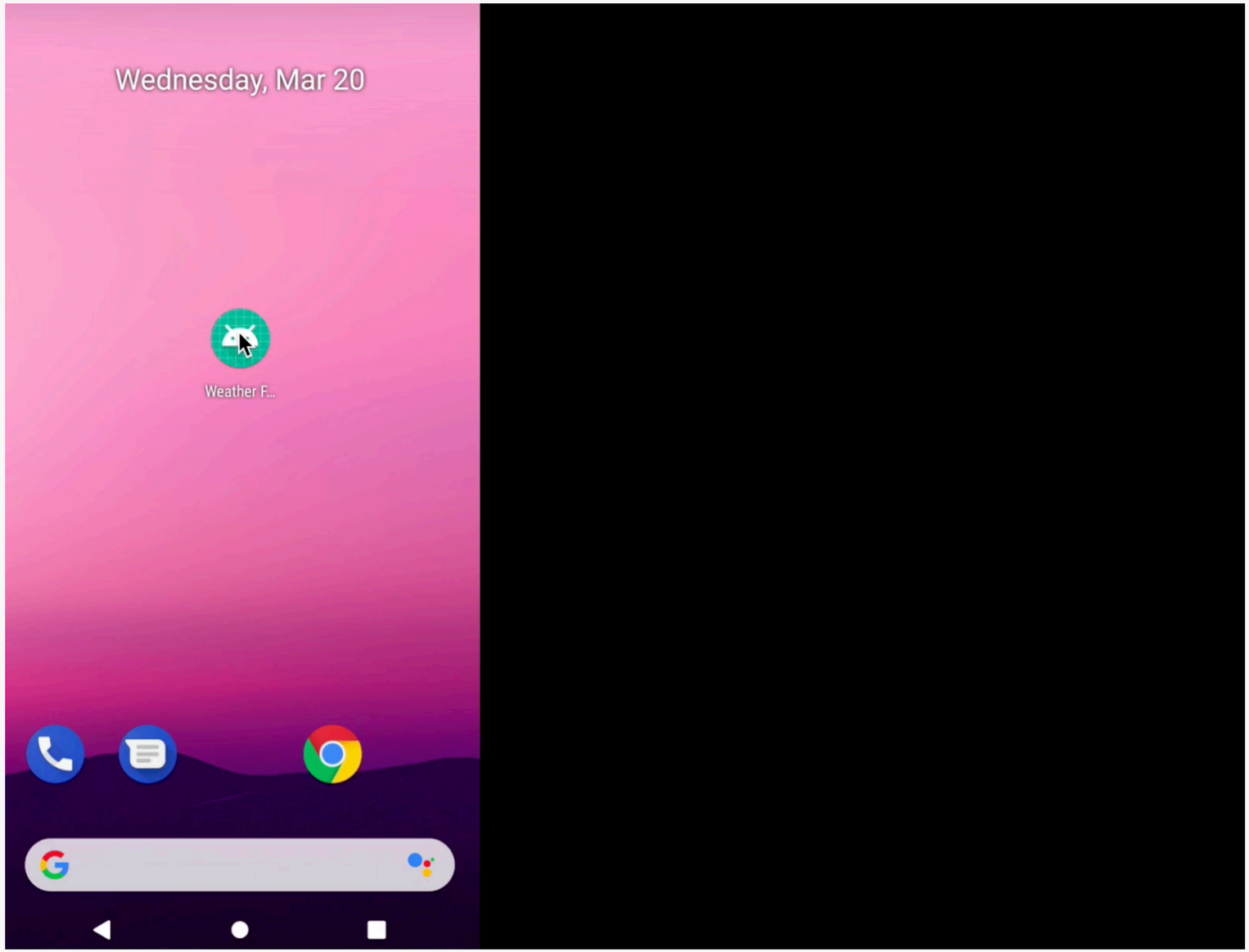
```
abstract class SecureFragment : Fragment() {

    val authenticationViewModel by activityViewModels<AuthenticationViewModel>()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        authenticationViewModel.authenticationStatus.observe(viewLifecycleOwner, Observer { authStatus ->
            when(authStatus) {
                UNAUTHENTICATED -> navController.navigate(actionGlobalLogin())
                USER_DECLINED -> popBackStackOrExit()
                else -> Unit
            }
        })
    }

    private fun popBackStackOrExi
}
```

```
enum class AuthenticationStatus {
    UNAUTHENTICATED, // user needs to authenticate state
    AUTHENTICATED, // user is in authenticated state
    USER_DECLINED, // user has declined to authenticate
}
```

# Global Navigation

```
abstract class SecureFragment : Fragment() {

    val authenticationViewModel by activityViewModels<AuthenticationViewModel>()


    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        authenticationViewModel.authenticationStatus.observe(viewLifecycleOwner, Observer { authStatus ->
            when(authStatus) {
                UNAUTHENTICATED -> navController.navigate(actionGlobalLogin())
                USER_DECLINED -> popBackStackOrExit()
                else -> Unit
            }
        })
    }

    private fun popBackStackOrExit() {
        if(!navController.popBackStack()) {
            requireActivity().setVisible(false)
            requireActivity().finish()
        }
    }
}
```

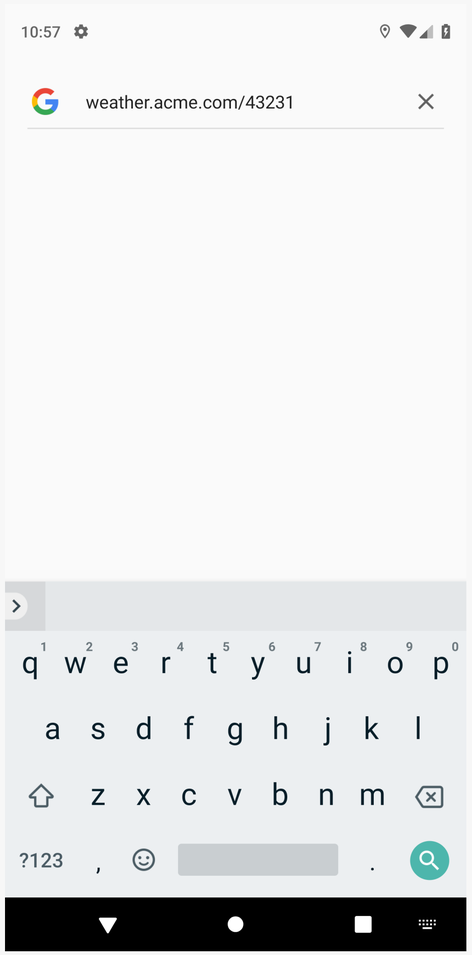
# Demo



# Deep Link

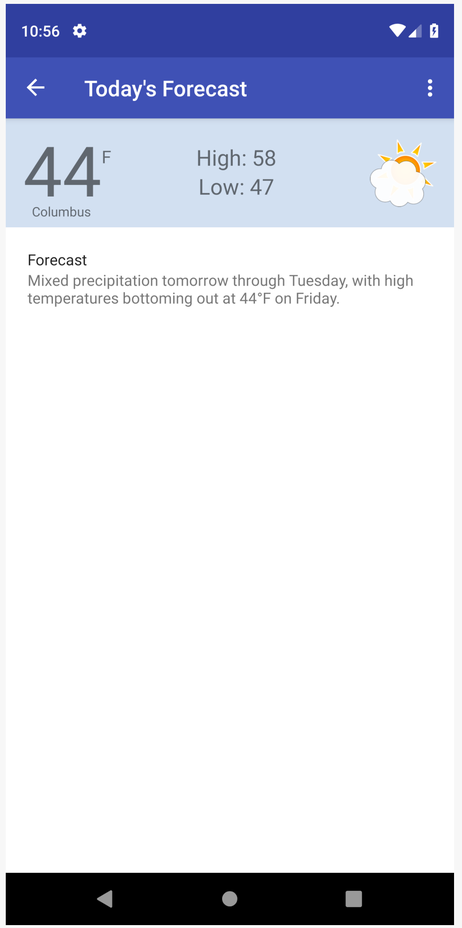
DEEP LINK

# Implicit




URI

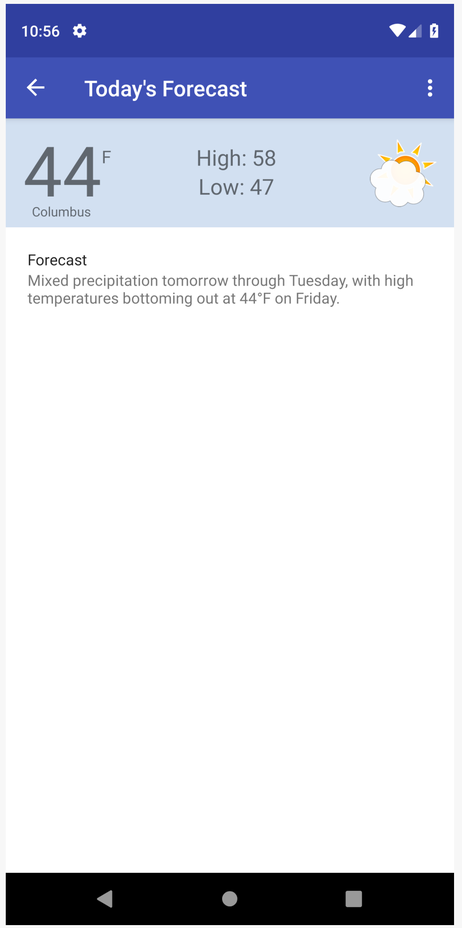
`weather.acme.com/{zip_code}`




https://github.com/ericmaxwell2003/Weather

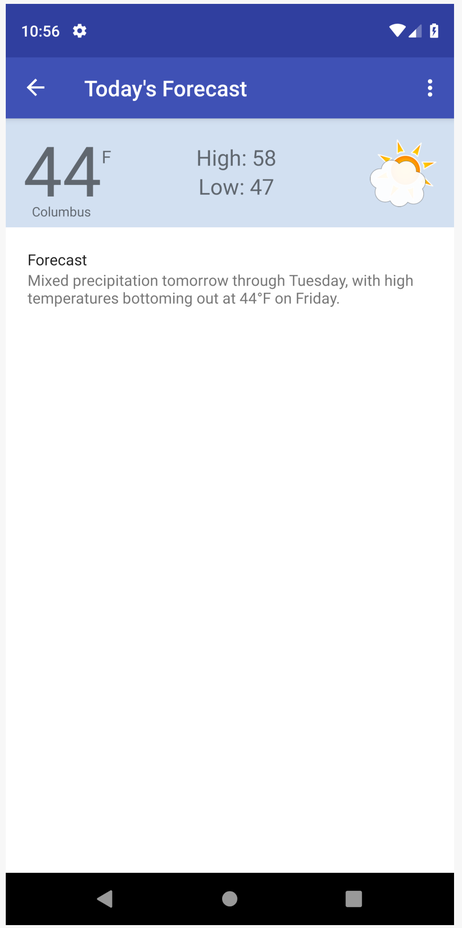
# Implicit

```
<fragment
    android:id="@+id/weather_detail"
    android:name="com.acme.weather.app.view.WeatherDetailFragment"
    android:label="Today's Forecast"
    tools:layout="@layout/weather_detail_fragment">

    <argument
        android:name="zip_code"
        app:argType="string" />

    <argument
        android:name="use_fahrenheit"
        app:argType="boolean"
        android:defaultValue="true" />

    <deepLink
        android:id="@+id/deepLink"
        app:uri="weather.acme.com/{zip_code}" />

</fragment>
```



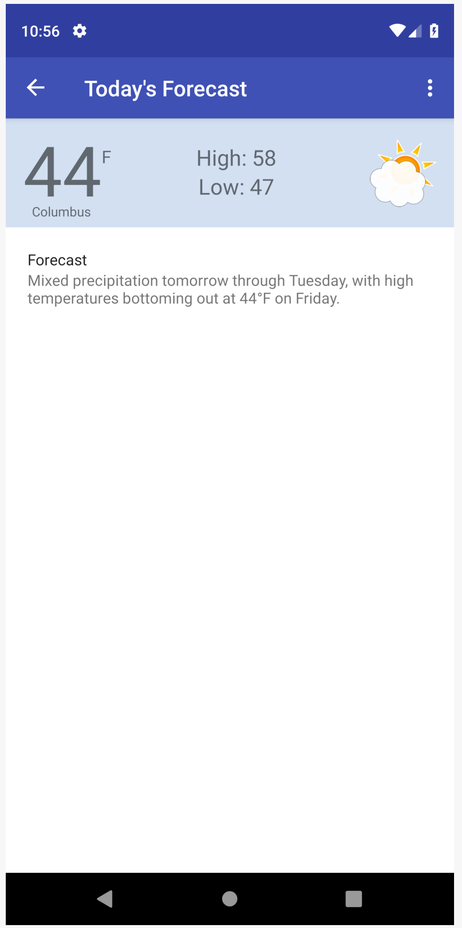
# Implicit

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.acme.weather">

   <application
        ...

        <activity android:name=".app.view.MainActivity">

            <nav-graph android:value="@navigation/weather_graph" />

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>

</manifest>
```



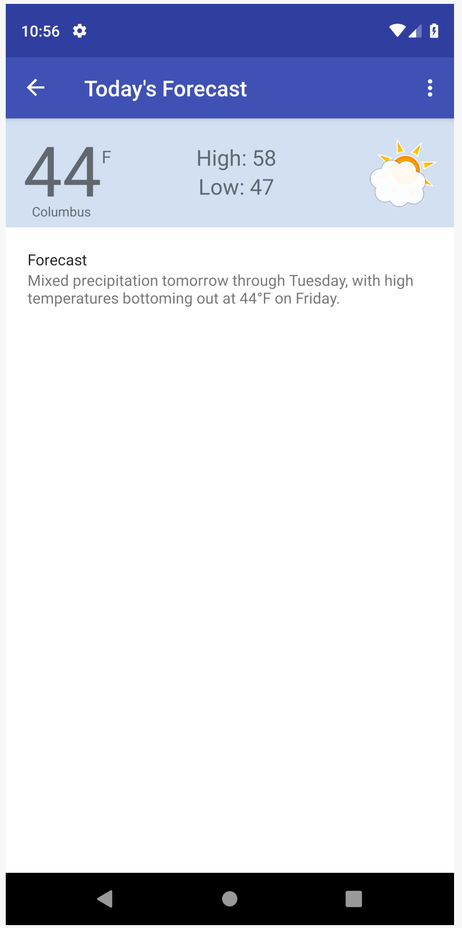
# Implicit

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.acme.weather">

   <application
        ...

        <activity android:name=".app.view.MainActivity">

            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />

                <data android:scheme="http" />
                <data android:scheme="https" />
                <data android:host="weather.acme.com" />
                <data android:pathPrefix="/" />
            </intent-filter>
        ...
        </activity>

    </application>

</manifest>
```



# On SetGraph

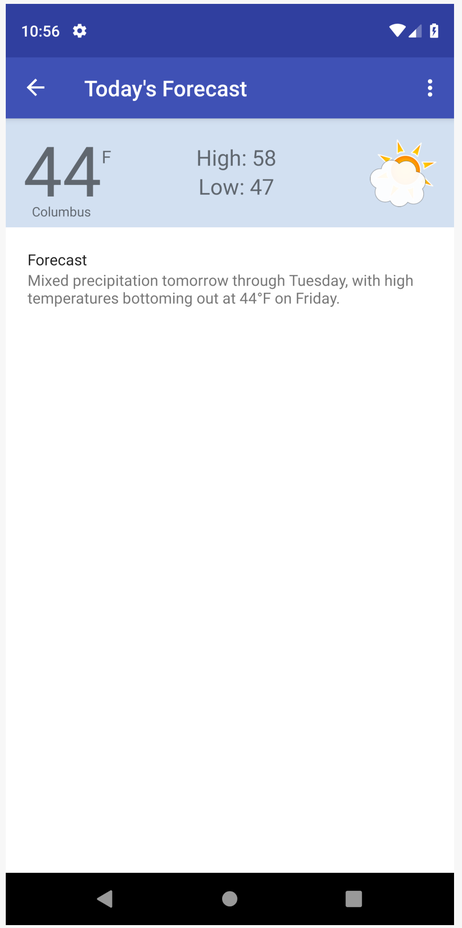
1. Search Graph for best matching deep link
   - URI exact matches preferred (Intent to Dest Deep Link)
   - By matching arguments



# On SetGraph

1. Search Graph for best matching deep link
   - URI exact matches preferred (Intent to Dest Deep Link)
   - By matching arguments

2. Build array of destination IDs leading to this Destination
   - R.id.weather_list
   - R.id.weather_detail (includes self)

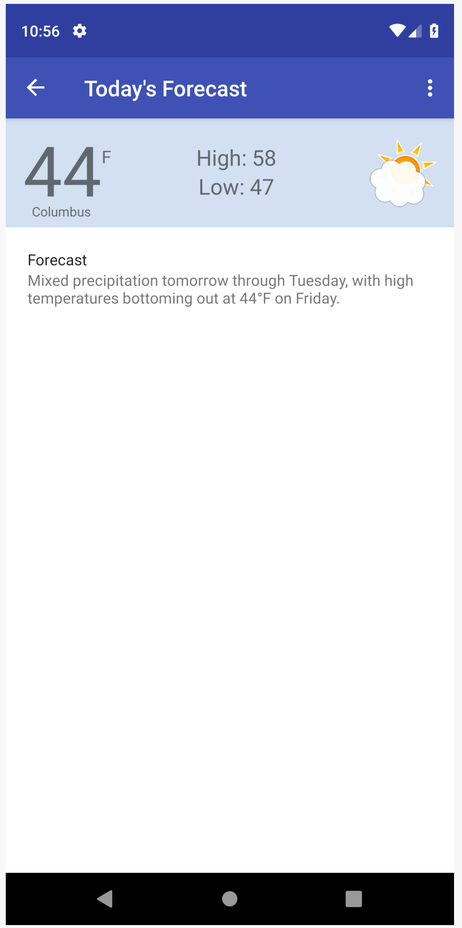


# On SetGraph

1. Search Graph for best matching deep link
   - URI exact matches preferred (Intent to Dest Deep Link)
   - By matching arguments

2. Build array of destination IDs leading to this Destination
   - R.id.weather_list
   - R.id.weather_detail (includes self)

3. Push each destination on the stack
   - (If new task)

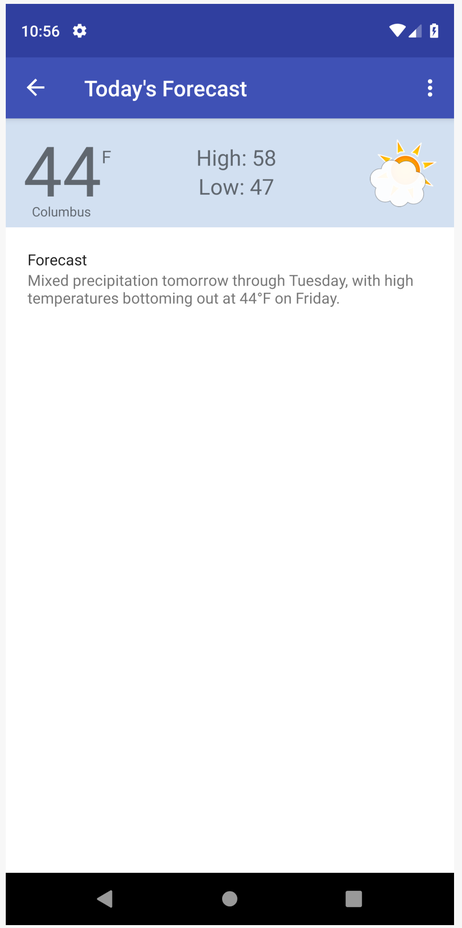


# On SetGraph

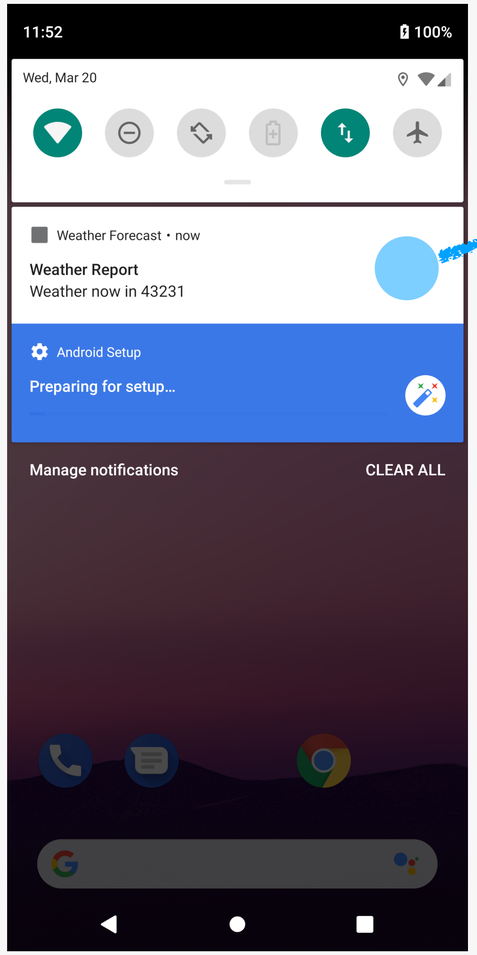
1. Search Graph for best matching deep link
   - URI exact matches preferred (Intent to Dest Deep Link)
   - By matching arguments

2. Build array of destination IDs leading to this Destination
   - R.id.weather_list
   - R.id.weather_detail (includes self)

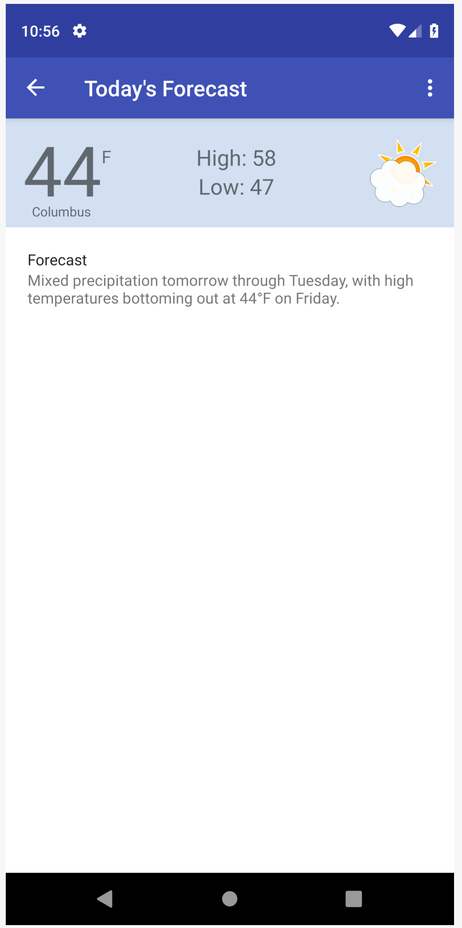
3. Push last destination on the stack
   - (If part of an existing task)



# Explicit



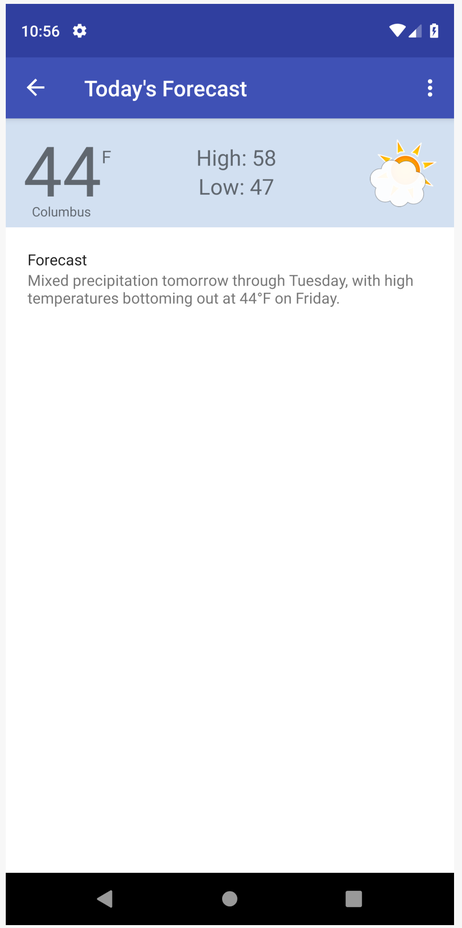

Launch via `PendingIntent`




https://github.com/ericmaxwell2003/Weather

# Create Pending Intent

```
fun scheduleNotificationForZip() {

    /* Create a deep link and pending intent */
    val pendingIntent = navController.createDeepLink()
            .setDestination(R.id.weather_detail)
            .setArguments(arguments)
            .createPendingIntent()


    /* Build the Notification */
    createNotificationChannel()

    val builder = NotificationCompat.Builder(requireContext(), CHANNEL_ID)
            .setSmallIcon(R.drawable.notification_icon)
            .setContentTitle("Weather Report")
            .setContentText("Weather now in ${args.zipCode}")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setContentIntent(pendingIntent)
            .setAutoCancel(true)

    notificationManager.notify(1, builder.build())
}
```

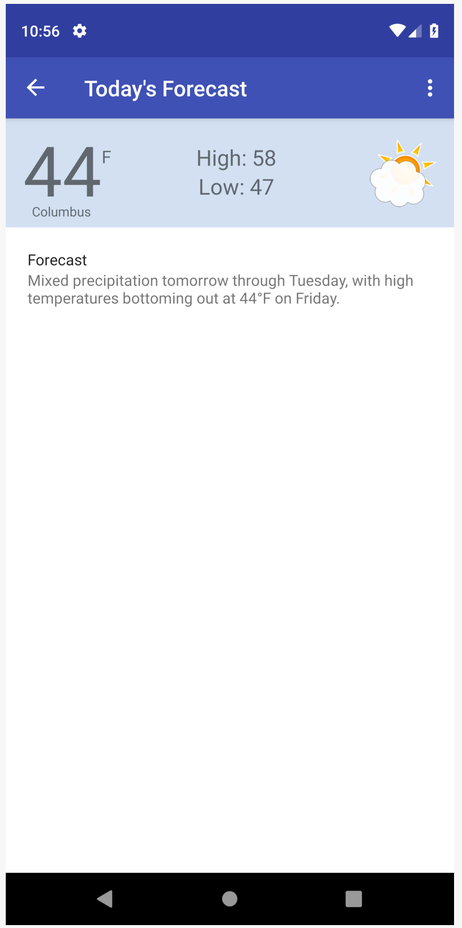


# Explicit Deep Link Intent

1. **android-support-nav:controller:deepLinkIds**
   - R.id.weather_list
   - R.id.weather_detail

2. **android-support-nav:controller:deepLinkExtras**
   - zip_code
   - use_fahrenheit

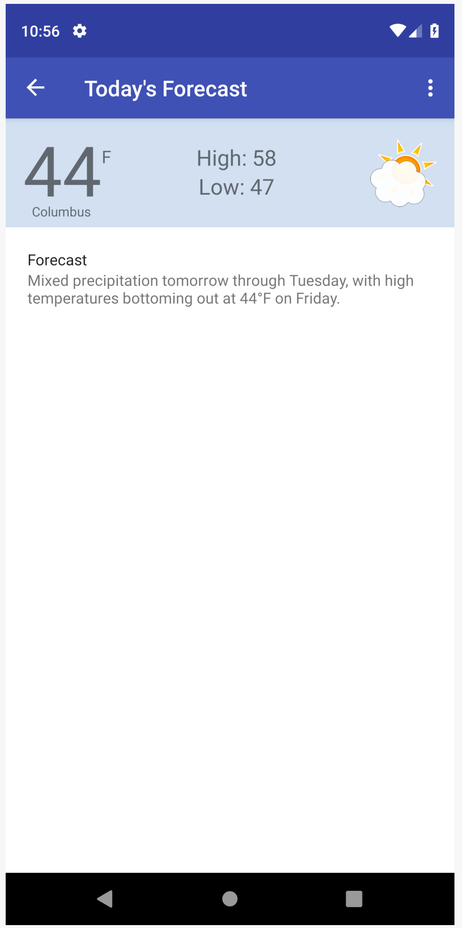
3. **android-support-nav:controller:deepLinkIntent**
   - Intent that triggered the deep link



# Testing Navigation

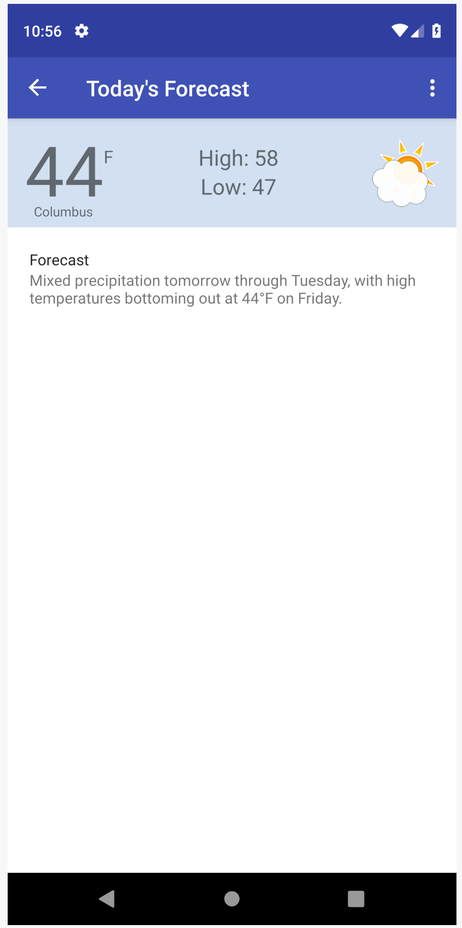
# Mock NavController

```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario =
        launchFragmentInContainer<WeatherDetailFragment>(bundle)

    fragmentScenario.onFragment { fragment ->

        // Access to set after onResume()
        Navigation.setViewNavController(
                  fragment.requireView(), mockNavController)
    }


    // Drive view with Espresso and verify navigation interactions
    onView(...)
    verify(mockNavController).navigate(...)
}
```



# FragmentScenario

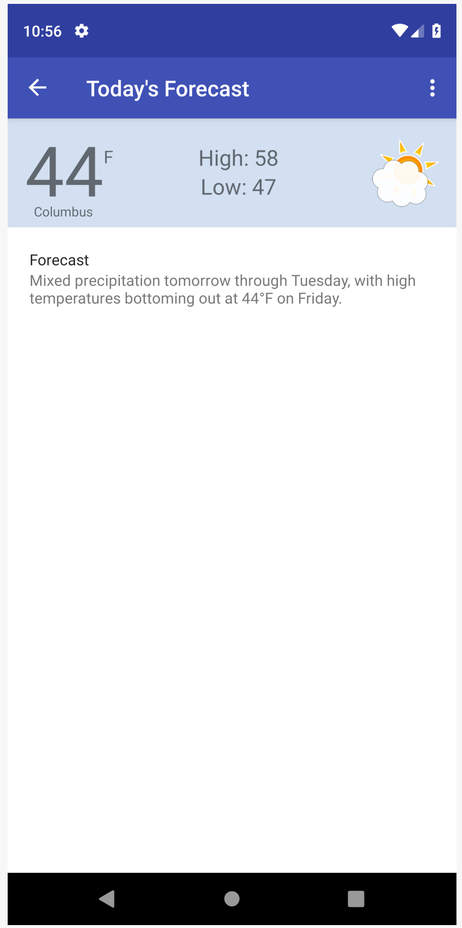
```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario =
        launchFragmentInContainer<WeatherDetailFragment>(bundle)

    fragmentScenario.onFragment { fragment ->

        // Access to set after onResume()
        Navigation.setViewNavController(
                  fragment.requireView(), mockNavController)
    }


    // Drive view with Espresso and verify navigation interactions
    onView(...)
    verify(mockNavController).navigate(...)
}
```




FragmentScenario Requires : 'androidx.fragment:fragment-testing:1.1.0**-alpha01**'

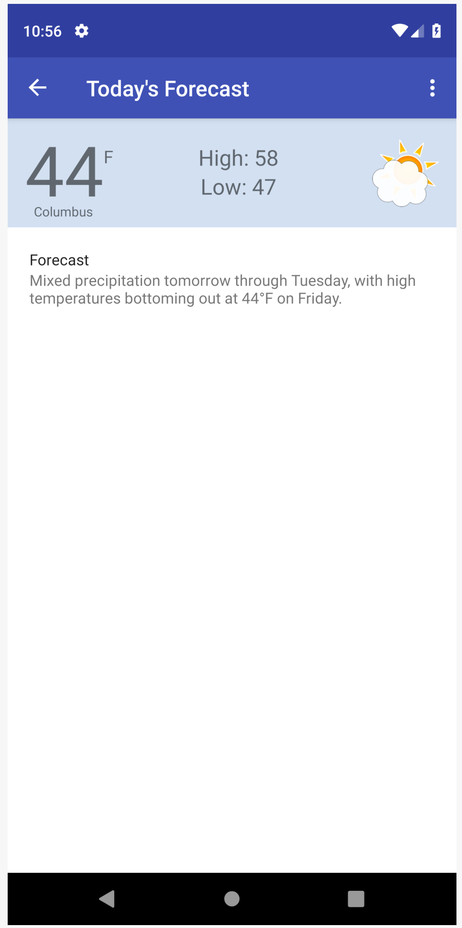
# FragmentScenario

```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario =
        launchFragmentInContainer<WeatherDetailFragment>(bundle)

    fragmentScenario.onFragment { fragment ->

        // Access to set after onResume()
        Navigation.setViewNavController(
                  fragment.requireView(), mockNavController)
    }

    // Drive view with Espresso and verify navigation interactions
    onView(...)
    verify(mockNavController).navigate(...)
}
```



# FragmentScenario

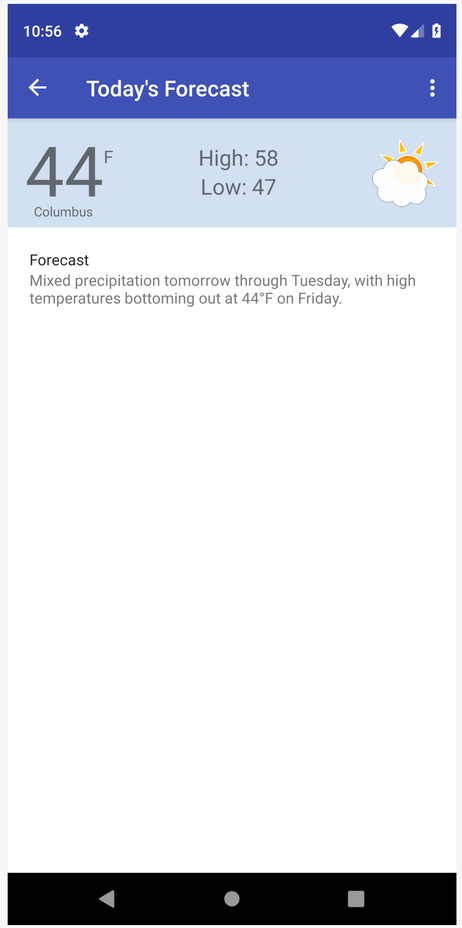
```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario =
        launchFragmentInContainer<WeatherDetailFragment>(bundle)

    fragmentScenario.onFragment { fragment ->

        // Access to set after onResume()
        Navigation.setViewNavController(
                  fragment.requireView(), mockNavController)
    }

    // Drive view with Espresso and verify navigation interactions
    onView(...)
    verify(mockNavController).navigate(...)
}
```



# FragmentScenario

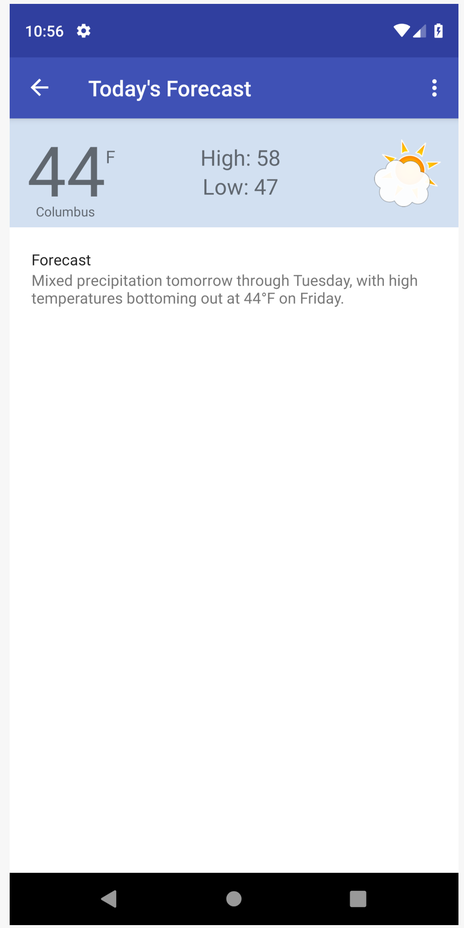
```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario =
        launchFragmentInContainer<WeatherDetailFragment>(bundle)

    fragmentScenario.onFragment { fragment ->

        // Access to set after onResume()
        Navigation.setViewNavController(
                  fragment.requireView(), mockNavController)
    }

    // Drive view with Espresso and verify navigation interactions
    onView(...)
    verify(mockNavController).navigate(...)
}
```

Called after Fragment.onResume()



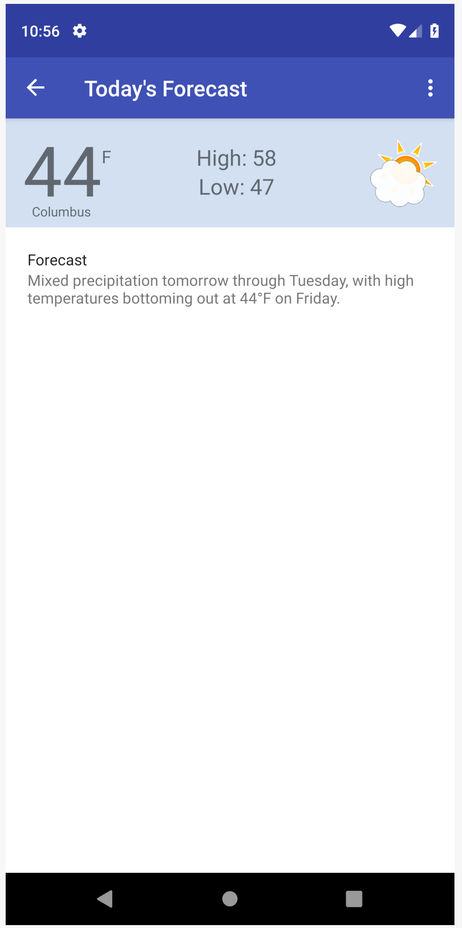
# FragmentScenario

```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario = launchFragmentInContainer(bundle) {

       WeatherDetailFragment().also { fragment ->
           fragment.viewLifecycleOwnerLiveData.observeForever { viewLifecycleOwner ->

            // Observe after onCreateView before onViewCreated
            if (viewLifecycleOwner != null) {
                Navigation.setViewNavController(
                   fragment.requireView(), mockNavController)
            }
        }
    }
    ...
}
```



# FragmentScenario

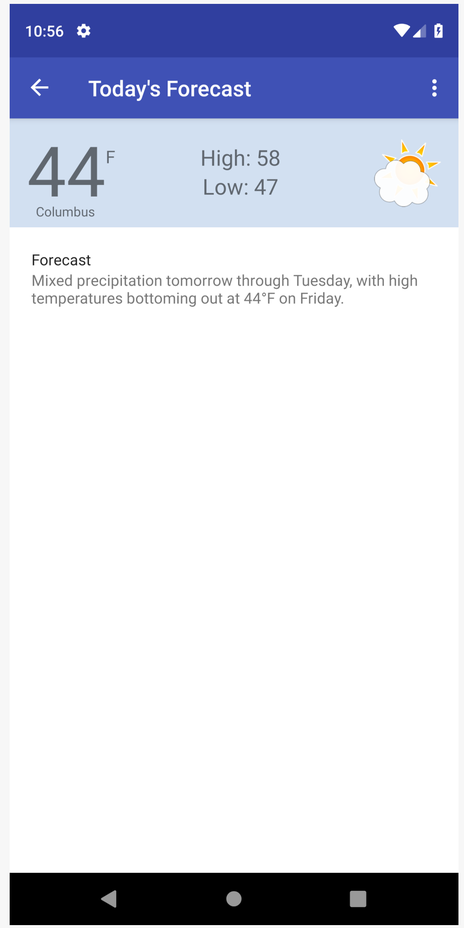
```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario = launchFragmentInContainer(bundle) {

        WeatherDetailFragment().also { fragment ->
            fragment.viewLifecycleOwnerLiveData.observeForever { viewLifecycleOwner ->

                // Observe after onCreateView before onViewCreated
                if (viewLifecycleOwner != null) {
                    Navigation.setViewNavController(
                        fragment.requireView(), mockNavController)
                }
            }
        }
    }
    ...
}
```




Construct the Fragment

# FragmentScenario

```
@Test
fun testNavigationToInGameScreen() {

    val mockNavController = mock(NavController::class.java)

    val bundle = WeatherDetailFragmentArgs.Builder("43231")
        .build()
        .toBundle()

    val fragmentScenario = launchFragmentInContainer(bundle) {

        WeatherDetailFragment().also { fragment ->
            fragment.viewLifecycleOwnerLiveData.observeForever { viewLifecycleOwner ->

                // Observe after onCreateView before onViewCreated
                if (viewLifecycleOwner != null) {
                    Navigation.setViewNavController(
                        fragment.requireView(), mockNavController)
                }
            }
    }
    ...
}
```

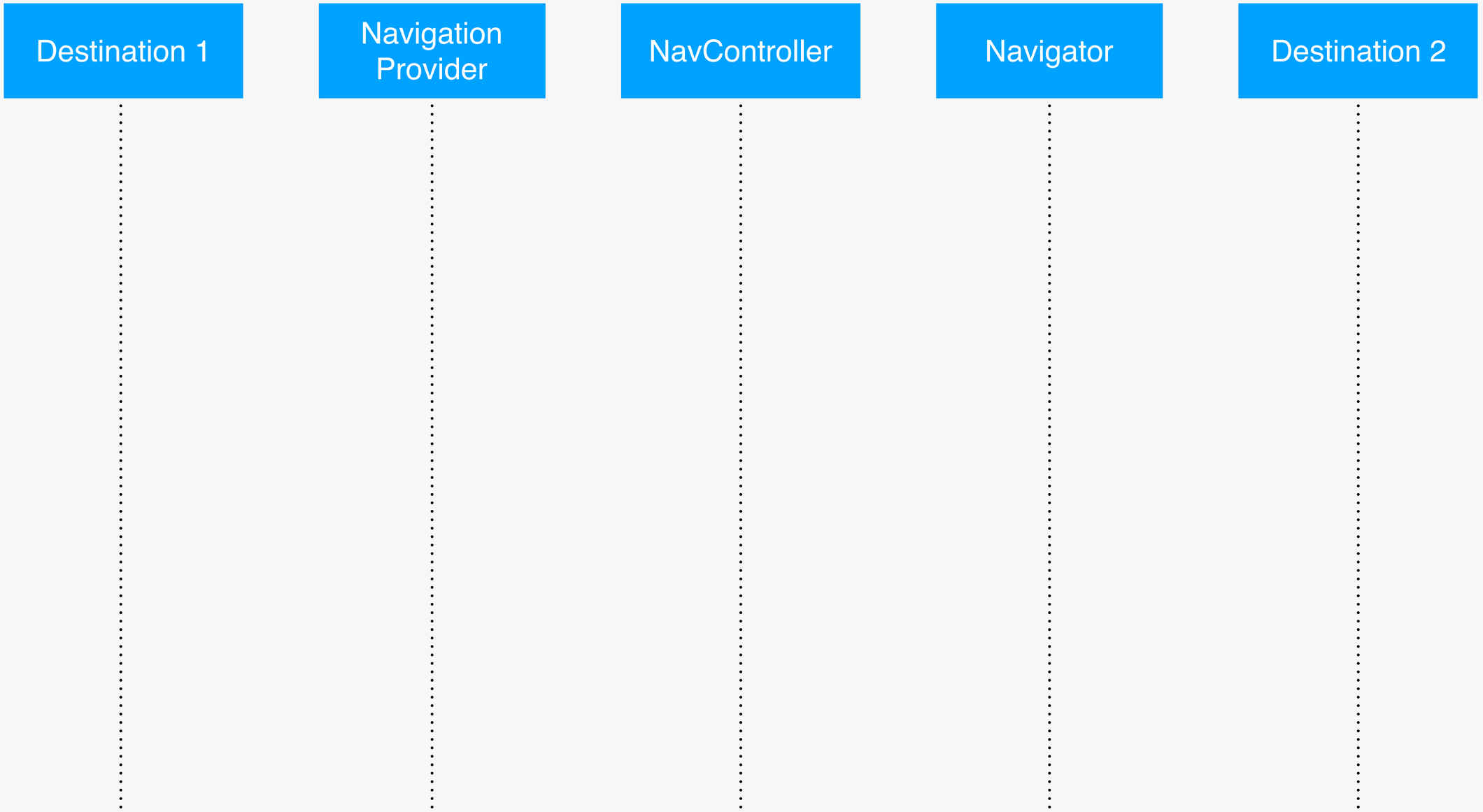


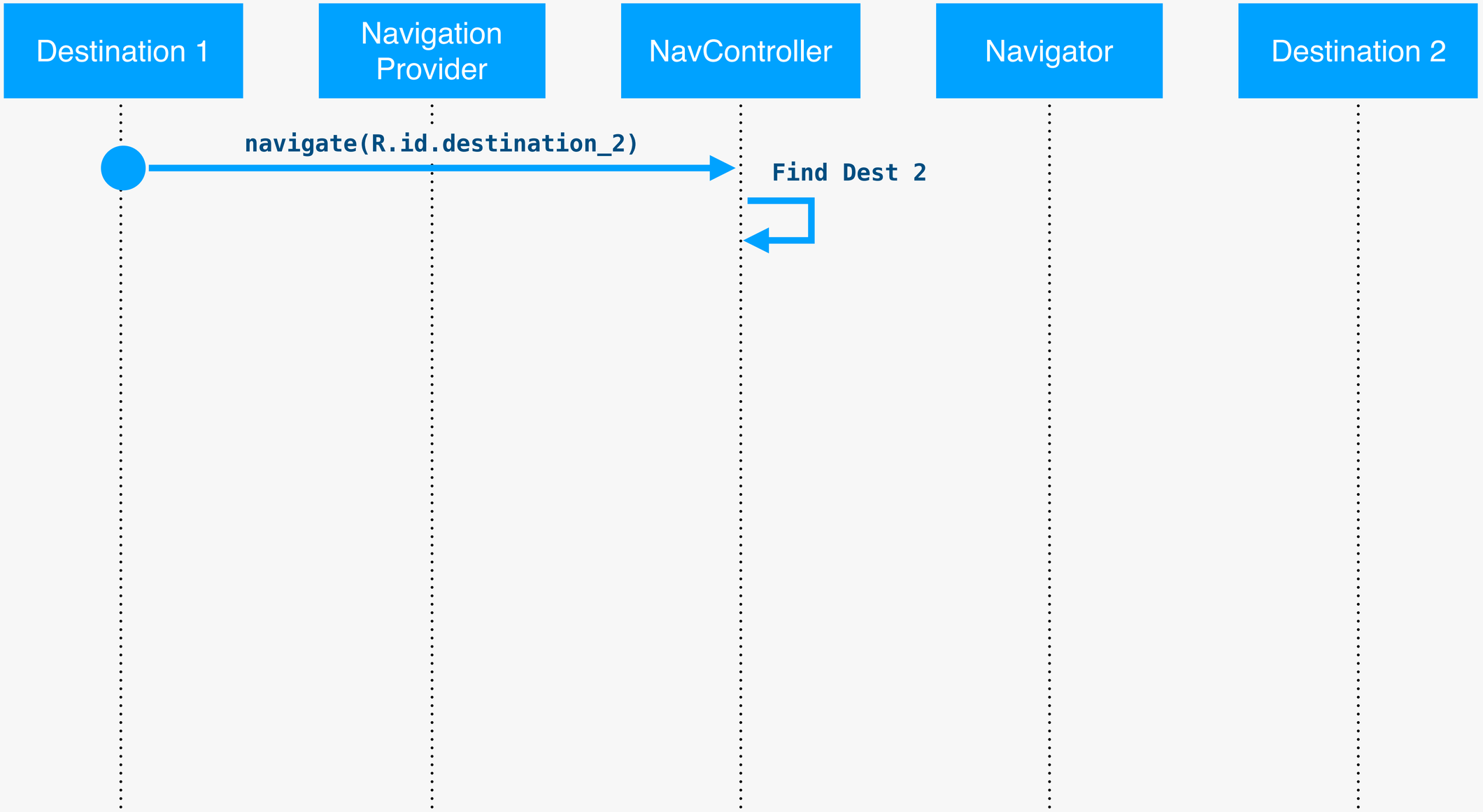

Observe its lifecycle
(Starting after **onCreateView(...)**)

# Custom Destination Types

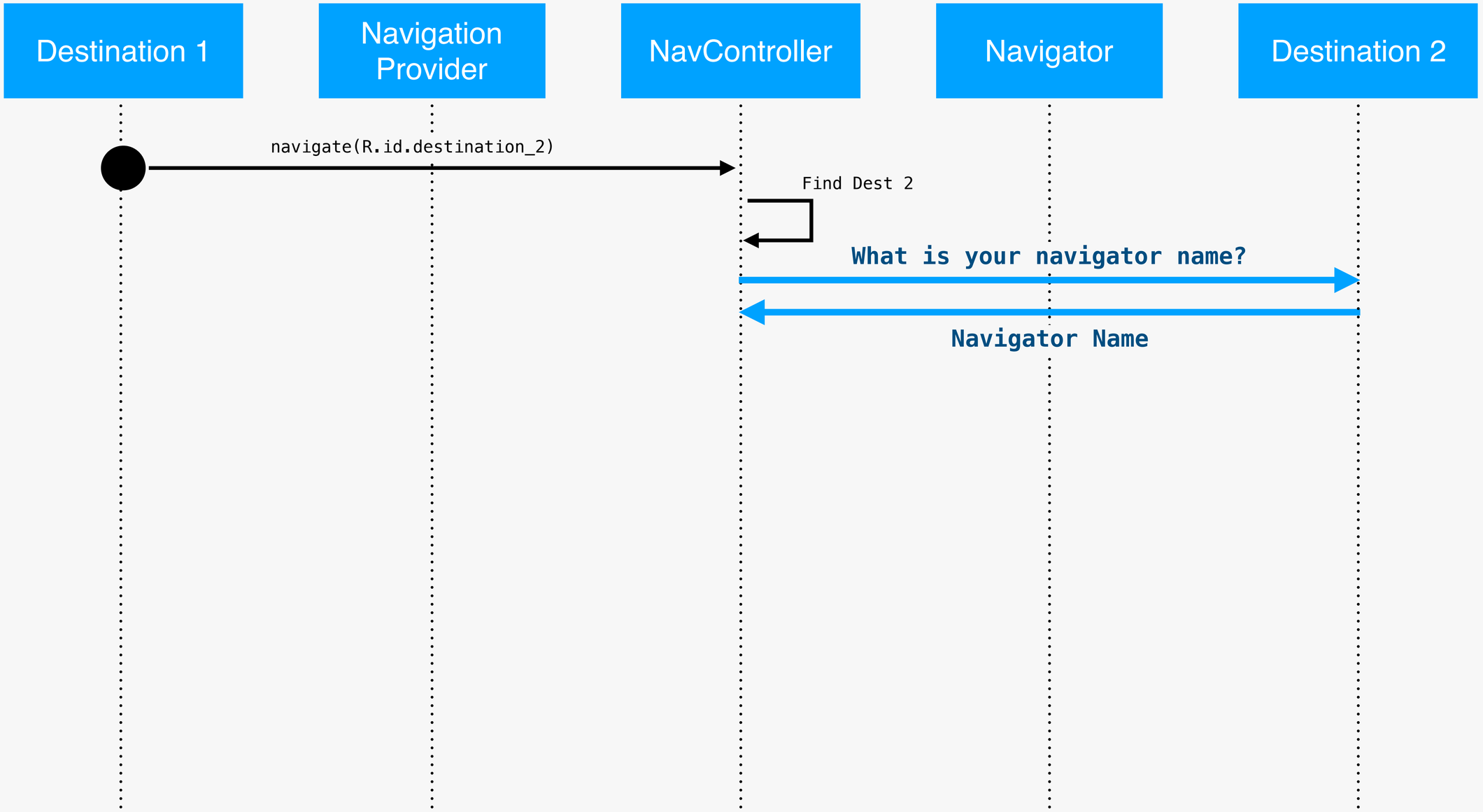

```
navigate(R.id.destination_2)
```

## NAVIGATION ARCHITECTURE

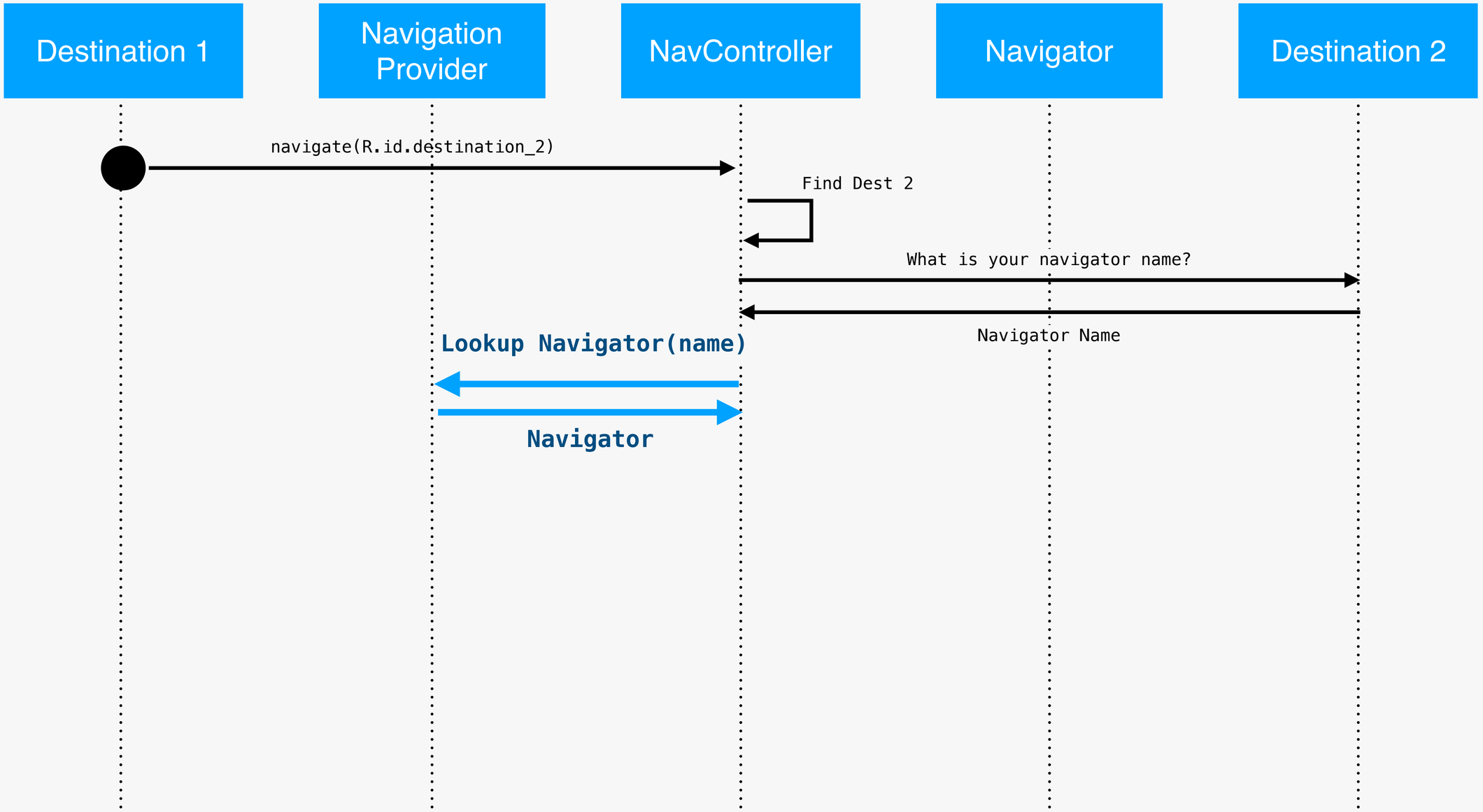



Destination 1
Navigation Provider
NavController
Navigator
Destination 2
navigate(R.id.destination_2)
Find Dest 2

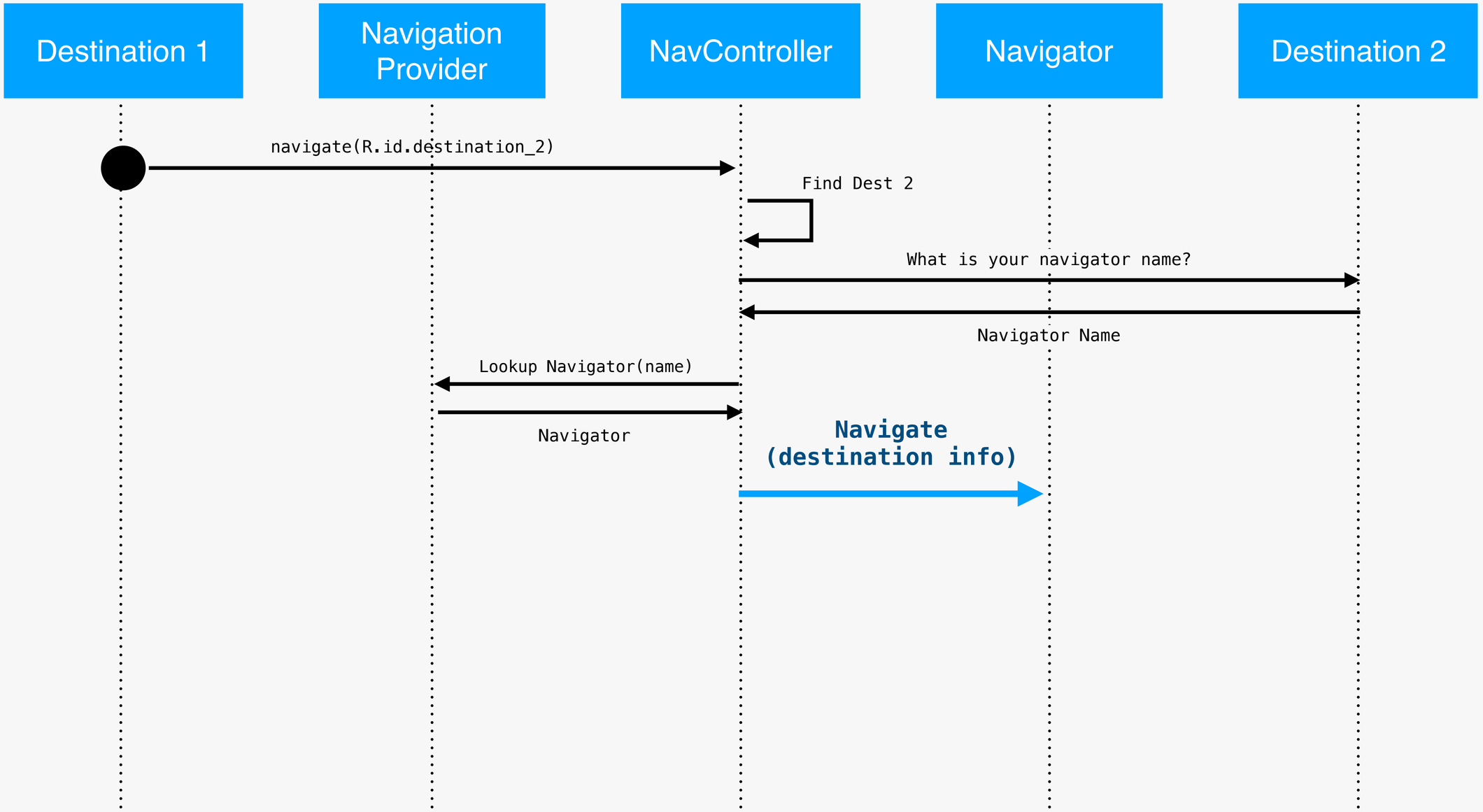
## NAVIGATION ARCHITECTURE



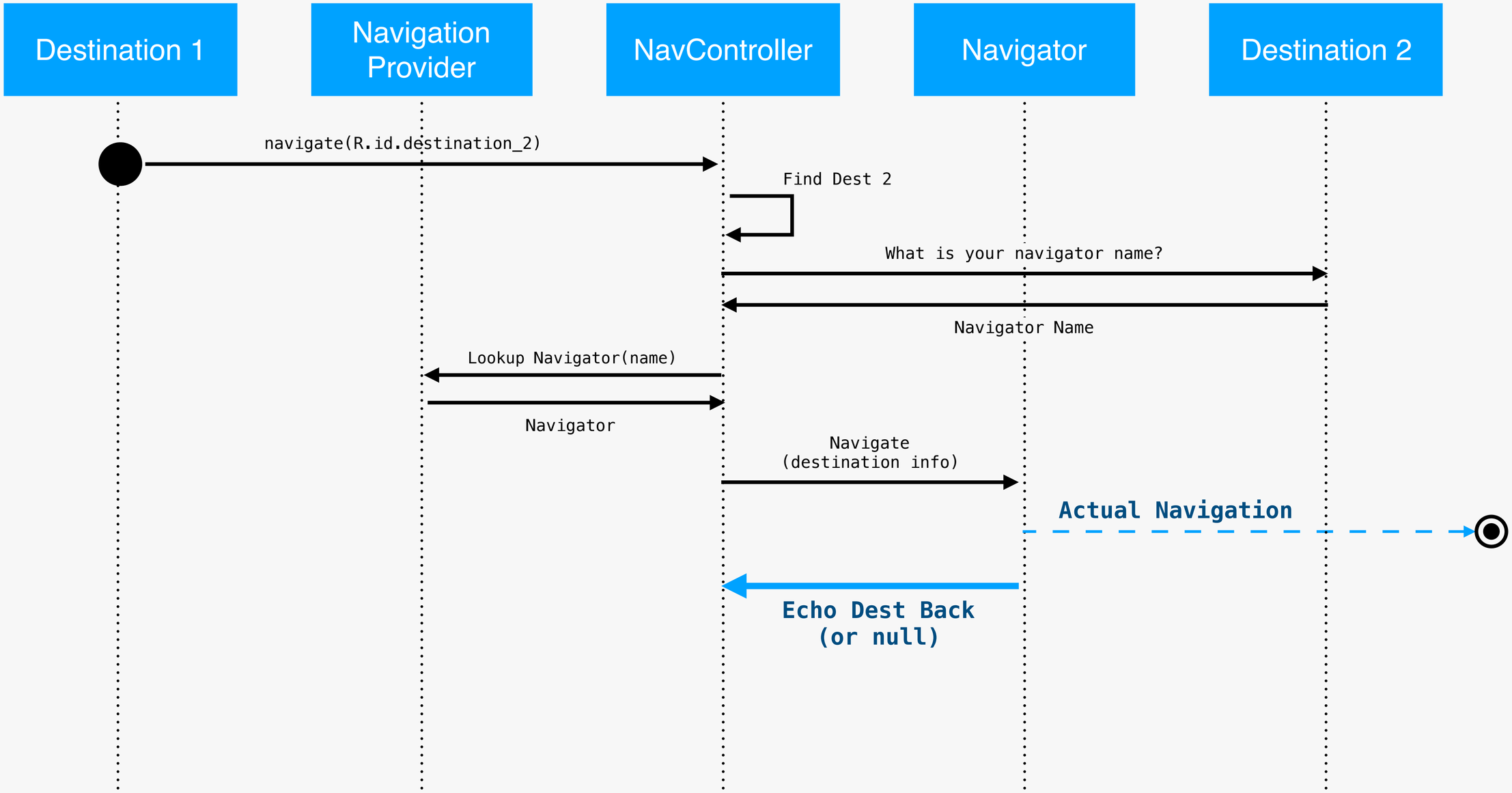
# NAVIGATION ARCHITECTURE

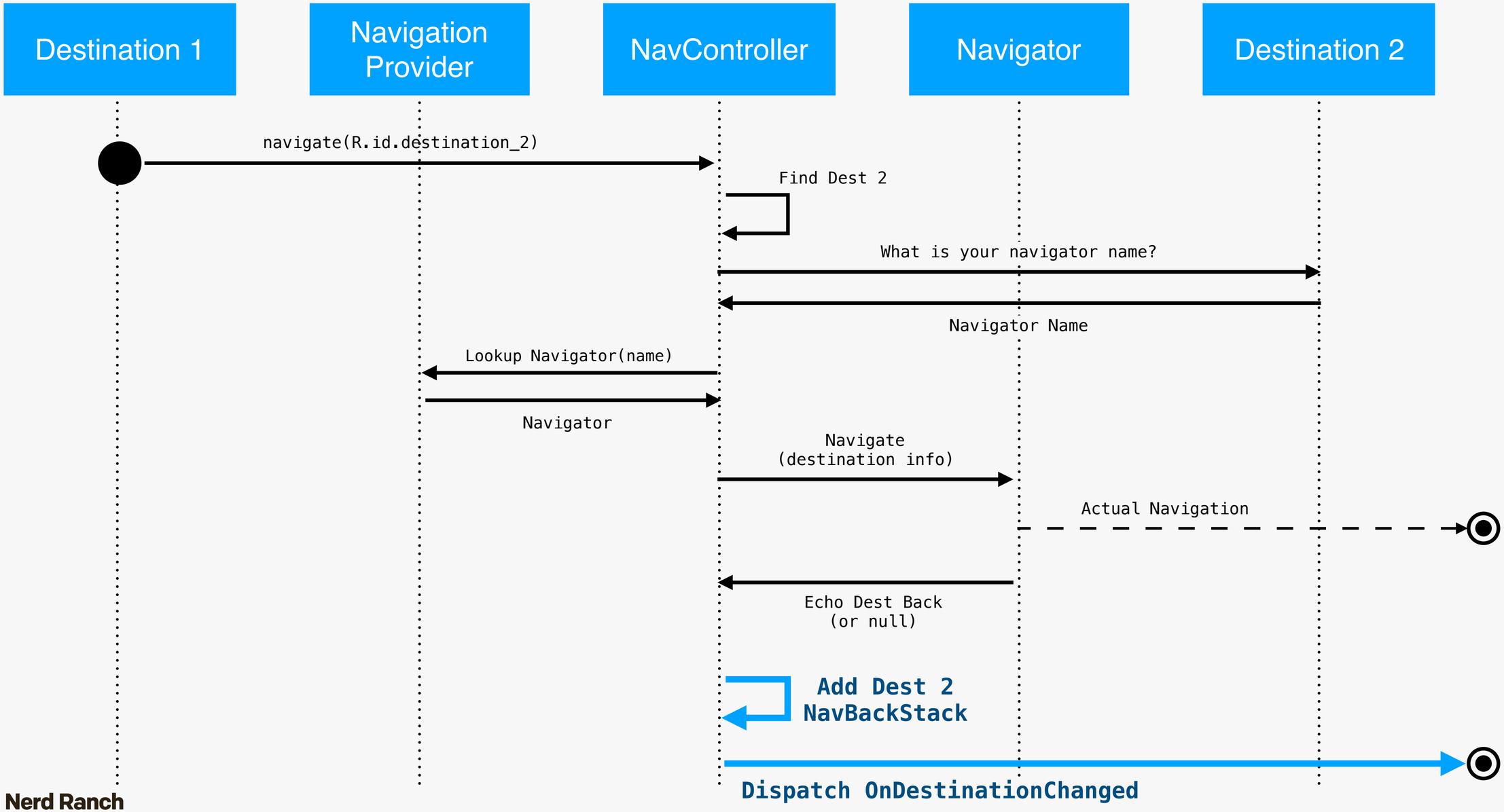


# NAVIGATION ARCHITECTURE

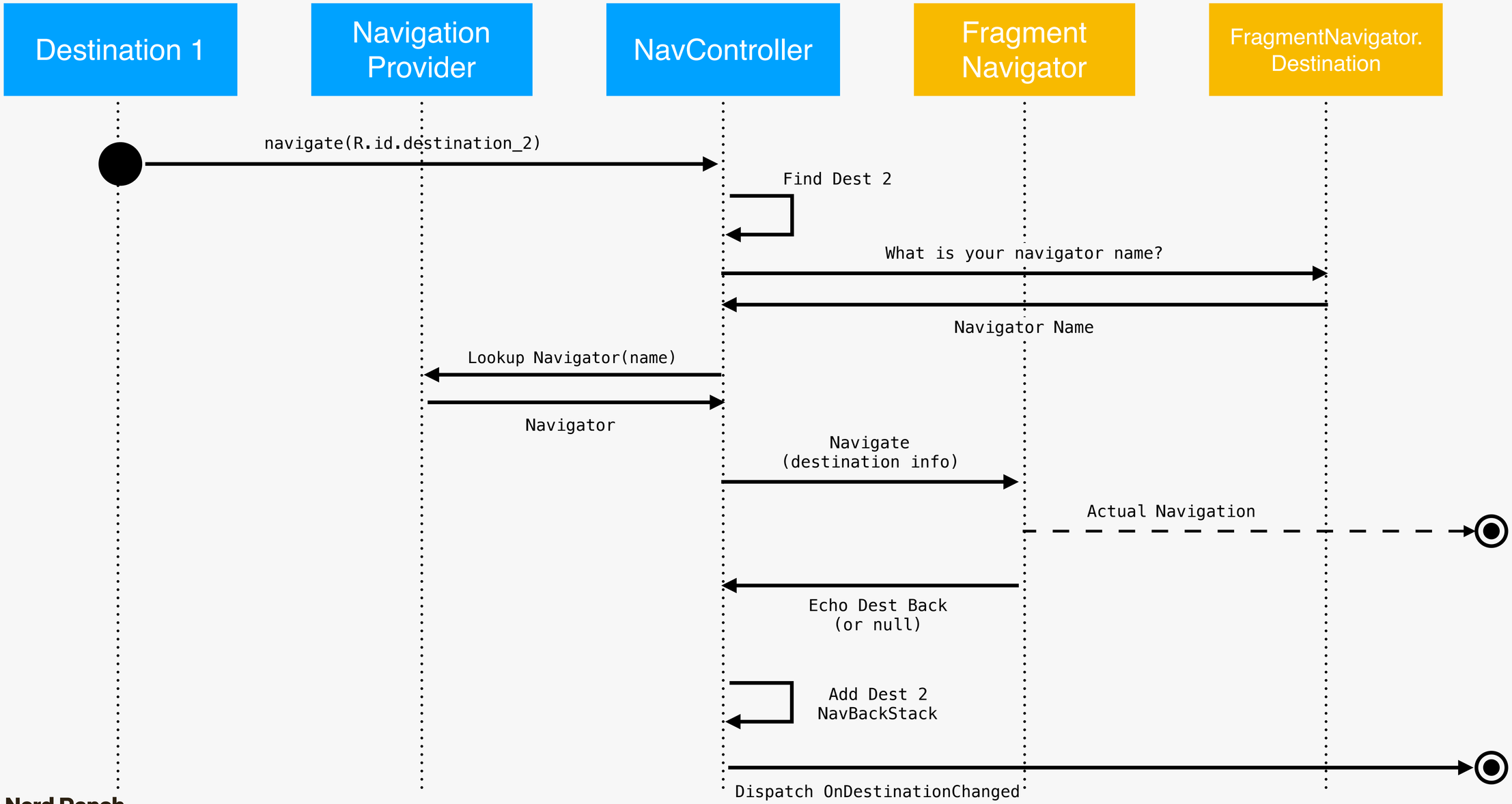


# NAVIGATION ARCHITECTURE




Destination 1
Navigation Provider
NavController
Navigator
Destination 2
navigate(R.id.destination_2)
Find Dest 2
What is your navigator name?
Navigator Name
Lookup Navigator(name)
Navigator
Navigate
(destination info)
Actual Navigation
Echo Dest Back
(or null)
Add Dest 2
NavBackStack
Dispatch OnDestinationChanged

# NAVIGATION ARCHITECTURE



## HOSTING NAVIGATION

Navigation Graph

NavHostCustom

**NavHostFragment Initialization**

1. Create NavController
2. **Registers ??? as a Destination Type**
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

```
class NavHostCustom : NavHost {

  init {
    ...
    navigationController.navigatorProvider += createControllerNavigator()
    ...
  }

  override fun getNavController(): NavController = navigationController

}
```

# HOSTING NAVIGATION

Navigation Graph

NavHostCustom

**NavHostFragment Initialization**

1. Create NavController
2. **Registers ??? as a Destination Type**
3. *Restore Navigation State*
4. Set Graph on NavController
5. Navigate to first Destination

```
class NavHostCustom : NavHostFragment() {

    override fun createFragmentNavigator(): Navigator<out FragmentNavigator.Destination> {
        navController.navigatorProvider += DialogNavigator(requireContext(), childFragmentManager)
        return super.createFragmentNavigator()
    }
}
```

| | Responsibilities | Reference Implementation(s) | Notes |
|---|---|---|---|
| NavHost | • Container for navigation<br>• Provide NavController<br>• Manage Navigation State<br>• Register | • **NavHostFragment** | • Usually registers additional Navigator types (e.g. FragmentNavigator) |
| NavController | • Navigate to action or destination<br>• Pop back to previous destination<br>• Generate bundle for saveState<br>• Restore itself from a previously saved bundle | -- | • Calls to navigate() and popBack(), delegated to the navigator of the next or current destination type. |
| Navigator <NavDestination> | • Delegate class used by the NavController to do the work of pushing/popping.<br>• Has intimate knowledge of the given destination type | • **NavGraphNavigator**<br>• **ActivityNavigator**<br>• **FragmentNavigator**<br>• **NoOpNavigator** | • Navigators can be registered<br><br>navController<br>.navigatorProvider +=<br>*someNavigator* |
| NavDestination | • Represents a node within the nav graph<br>• Usually a nested class of a corresponding Navigator of this type.<br>• Encapsulate the configured actions, arg declarations, default values, etc. | • **NavGraph**<br>• **ActivityNavigator.Destination**<br>• **FragmentNavigator.Destination** | • These define the elements you see in the nav graph too:<br><navigation><br><activity><br><fragment> |

# Custom Nav Samples

- Conductor POC - View Based Destinations
  - Navigating Conductor and the Navigation Architecture Component -- Tevin Jeffrey -- (https://bit.ly/2MD9LSn)

- DialogFragmentDestination
  - https://gist.github.com/fbarthelery/ad0062a88875b46e0065137ff03807a0

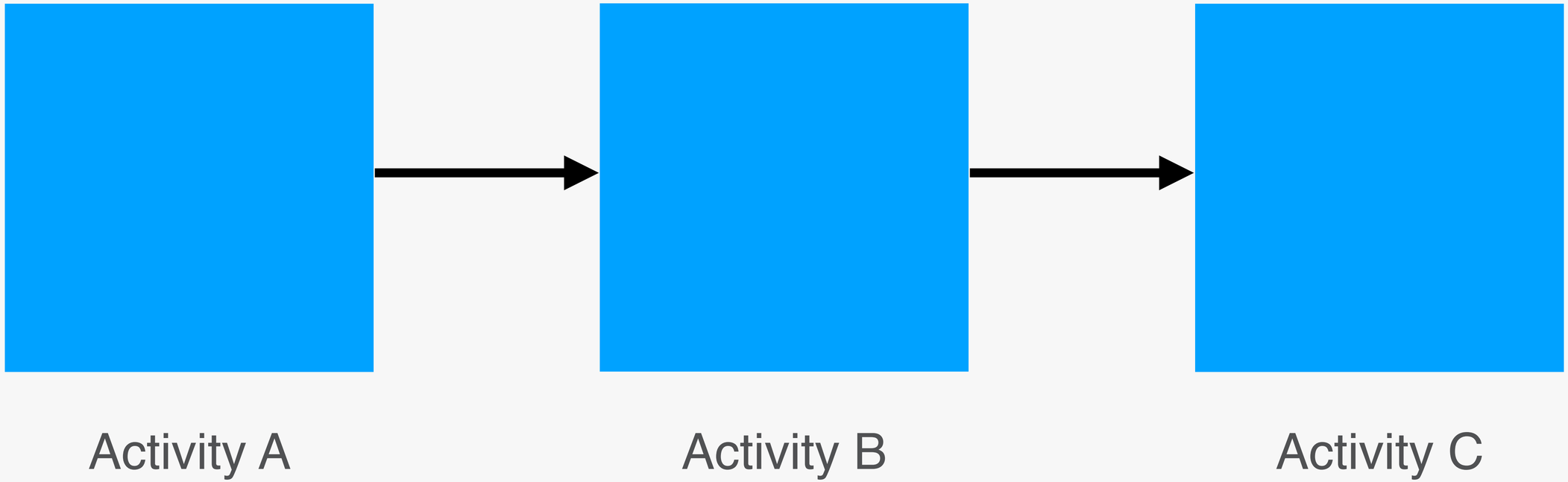
# Single Activity?

- Why Single Activity?
  - Navigation Graph per Activity
  - Lighter weight mechanism
  - Shared Lifecycle Aware Scope

- Why ~~Single~~ Fewer Activities?
  - Navigation Graph per Activity
  - Lighter weight mechanism
  - Shared Lifecycle Aware Scope
  - Need more convincing?

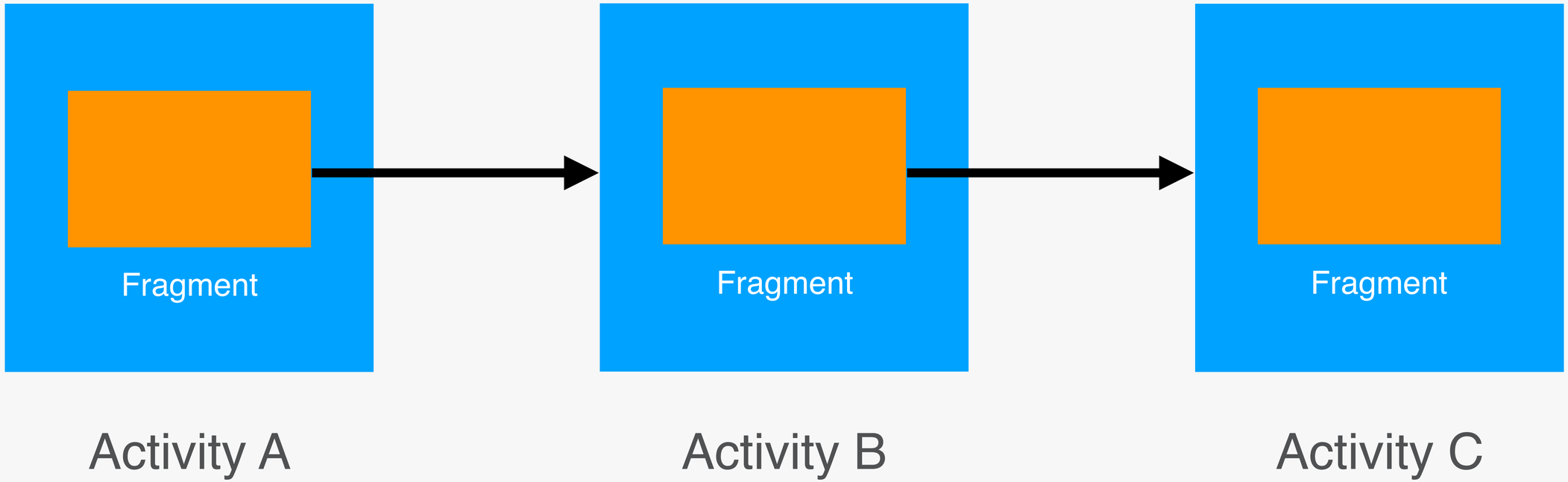
    Single Activity: Why, When, How -- DevSummit '18 -- Ian Lake -- (https://www.youtube.com/watch?v=2k8x8V77CrU)

# Getting to Single Activity



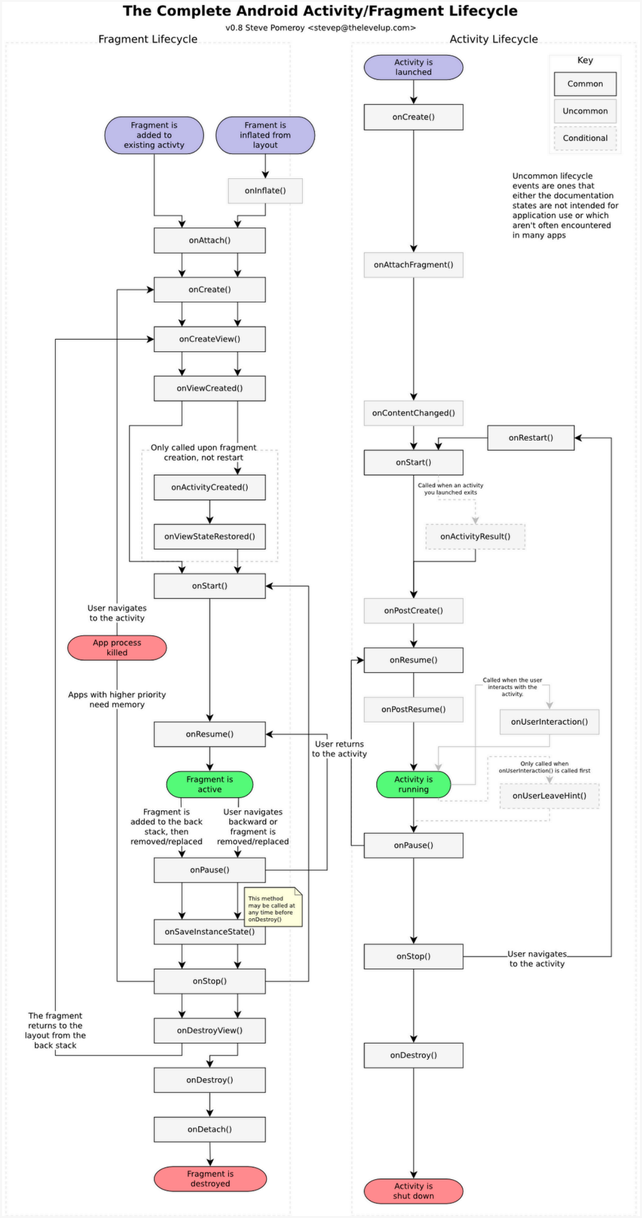

https://developer.android.com/guide/navigation/navigation-migrate

# Introduce Fragments

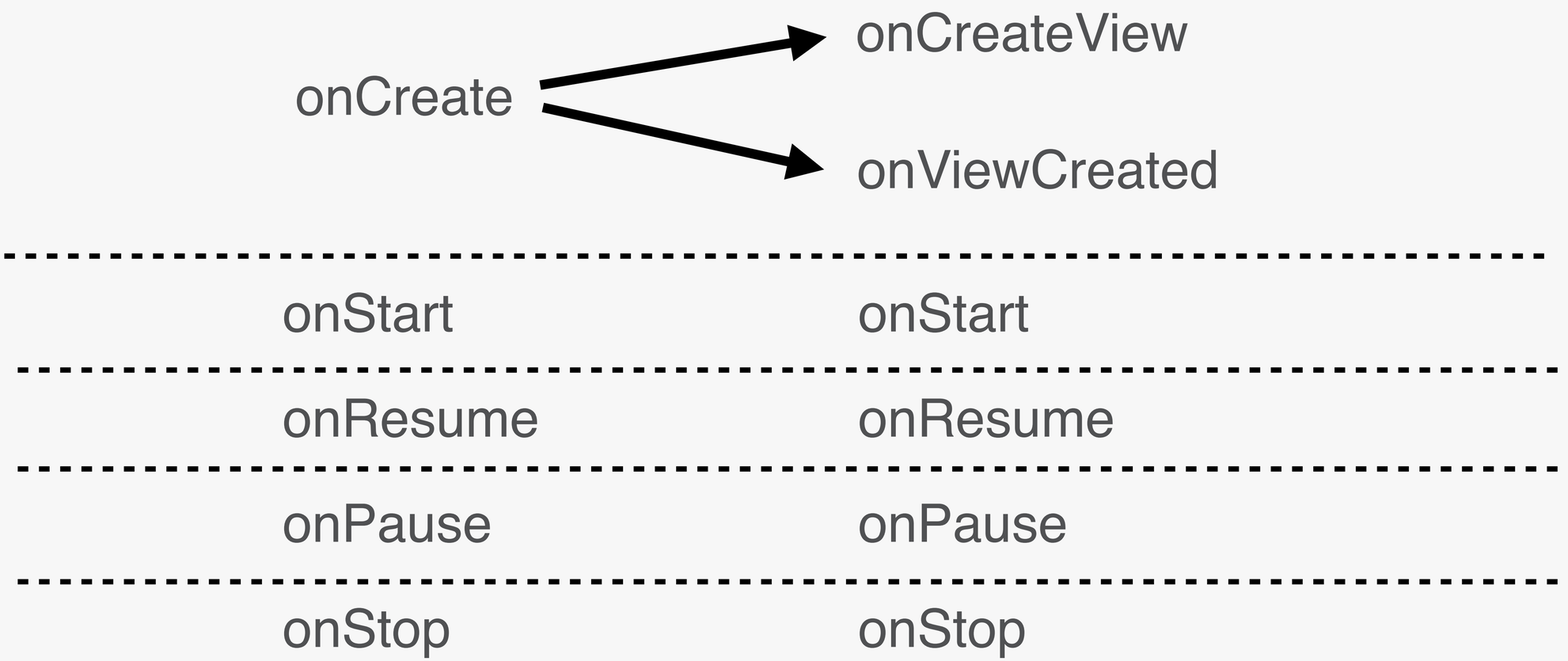



https://developer.android.com/guide/navigation/navigation-migrate

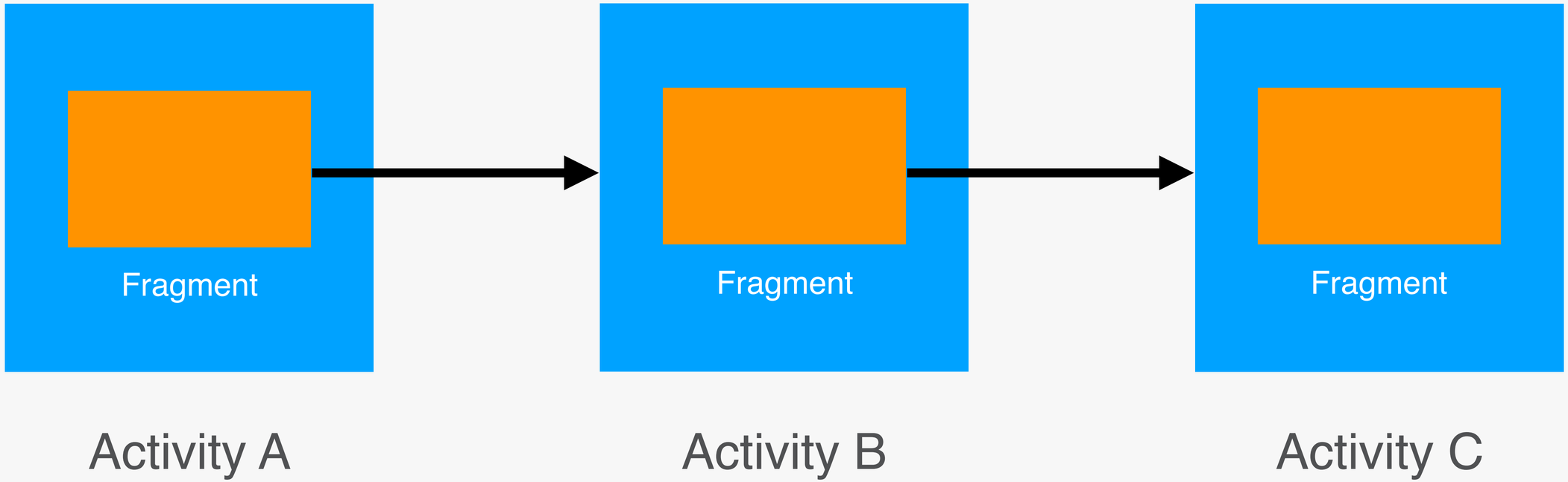
# Introduce Fragments



# Introduce Fragments

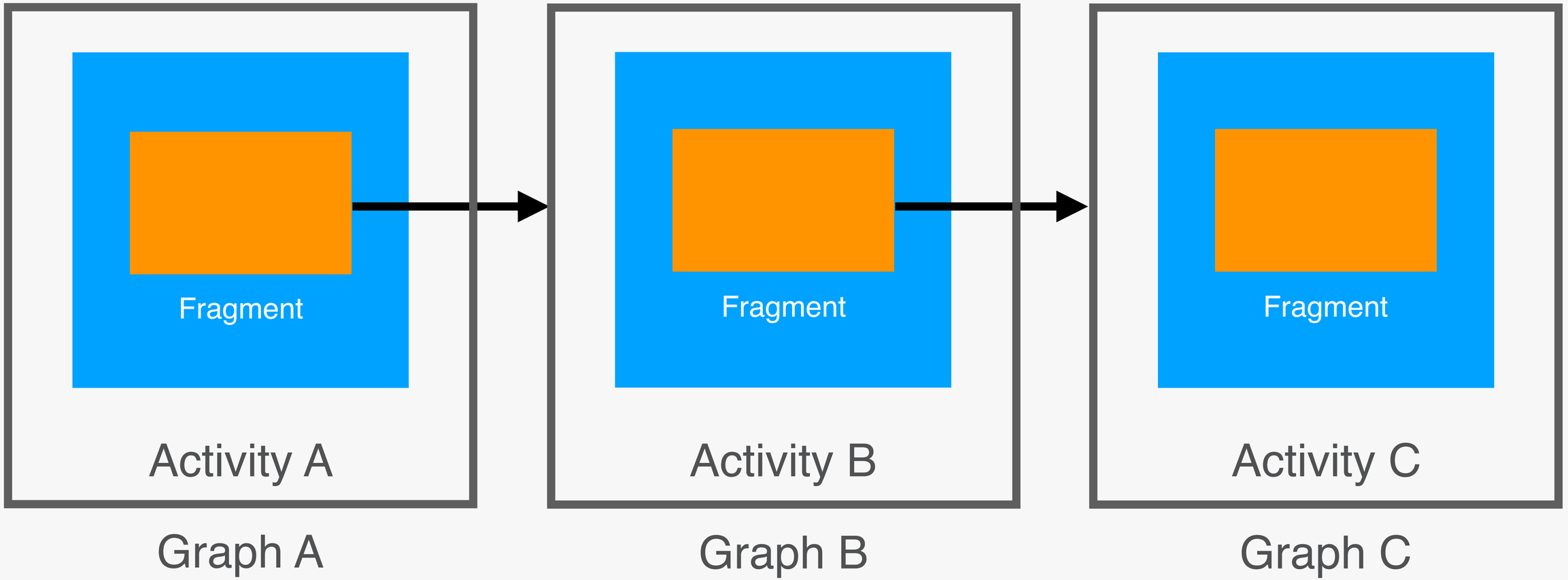


# Introduce Fragments



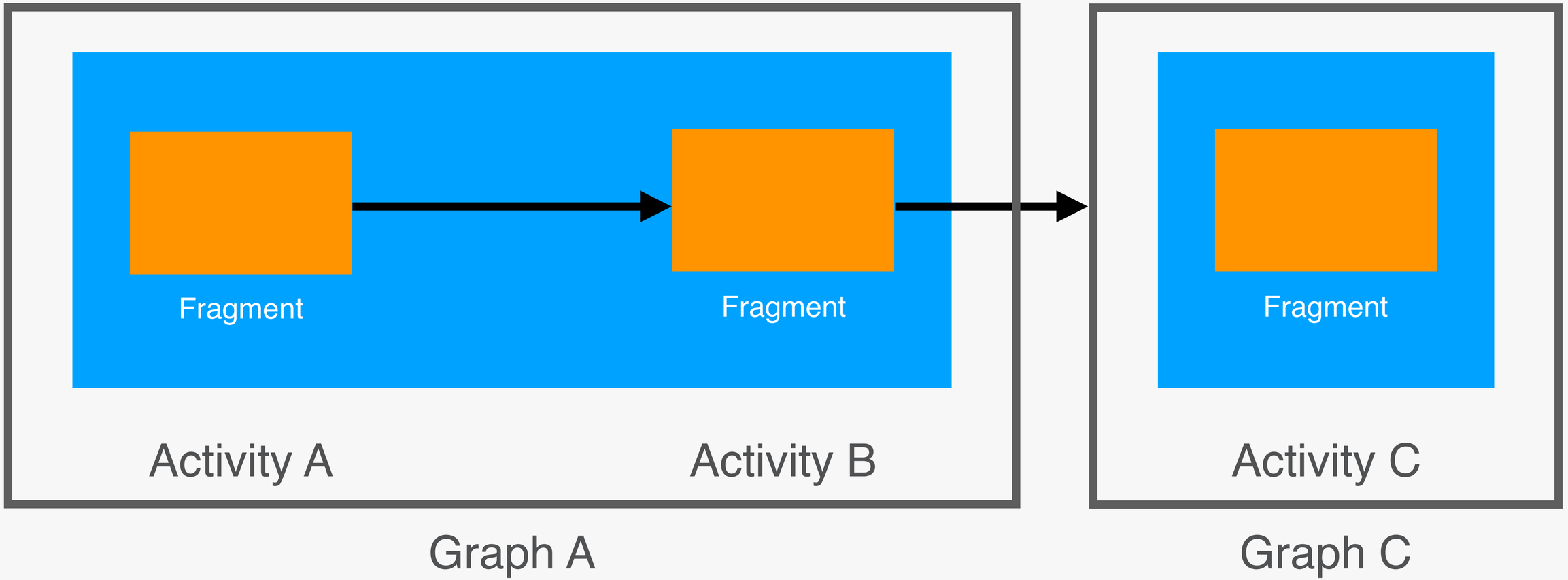

https://developer.android.com/guide/navigation/navigation-migrate

# Introduce Individual Graphs




https://developer.android.com/guide/navigation/navigation-migrate

# Combine Graphs




https://developer.android.com/guide/navigation/navigation-migrate

☆ ☆ ☆ ☆ ☆

# Migrate to the Navigation component

The Navigation component is a library that can manage complex navigation, transition animation, deep linking, and compile-time checked argument passing between the screens in your app.

This document serves as a general-purpose guide to migrate an existing app to use the Navigation component.

★ **Note:** This documentation uses fragments as examples, as they allow for integration with other Jetpack lifecycle-aware components. In addition to fragments, the Navigation component also supports custom destinations.

At a high level, migration involves four steps:

1. Move screen-specific UI logic out of activities - Move your app's UI logic out of activities, ensuring that each activity owns only the logic of global navigation UI components, such as a `Toolbar`, while delegating the implementation of each screen to a fragment or custom destination.
2. Integrate the Navigation component - For each activity, build a navigation graph which contains the one or more fragments managed by that activity. Replace fragment transactions with Navigation component operations.
3. Add activity destinations - Replace `startActivity()` calls with actions using activity destinations.
4. Combine activities - Combine navigation graphs in cases where multiple activities share a common layout.

★ **Important:** To ensure success, approach migration as an iterative process, thoroughly testing your app with each step. While a single-activity architecture allows you to take full advantage of the Navigation component, you do not need to fully migrate your app to benefit from Navigation.

Contents



**https://developer.android.com/guide/navigation/navigation-migrate**

@emmax

## ADDITIONAL RESOURCES

Official

- Navigation Documentation (https://developer.android.com/guide/navigation) 💪
- Source (AOSP) (https://android.googlesource.com/platform/frameworks/support/)
- Issue Tracker (https://issuetracker.google.com/issues?q=componentid:409828%20type:process%20status:open&p=1
- Code Lab https://codelabs.developers.google.com/codelabs/android-navigation/#0

Other Talks

- Adventures in Navigation -- DroidCon SF '18 -- Lyla Fujiwara -- (https://www.youtube.com/watch?v=ELGShpd17wc)
- Single Activity: Why, When, How -- DevSummit '18 -- Ian Lake -- (https://www.youtube.com/watch?v=2k8x8V77CrU)

Posts

- Using Navigation in a large banking app -- David Vávra -- (https://bit.ly/2JbT2Yy)
- Master-Detail views with Navigation Components --Lara Martín -- (https://bit.ly/2JqEKU7)
- Navigating Conductor and the Navigation Architecture Component -- Tevin Jeffrey -- (https://bit.ly/2MD9LSn)
- Modularization : Real Life Example -- Jeroen Mols -- (https://buff.ly/2I8mtZj)

Sample Projects

- Getting Started Sample - https://github.com/googlesamples/android-architecture-components/tree/master/NavigationBasicSample
- Advanced Navigation - https://github.com/googlesamples/android-architecture-components/tree/master/NavigationAdvancedSample
- DialogNavigator Gist - https://gist.github.com/fbarthelery/ad0062a88875b46e0065137ff03807a0

# Thanks!


Eric Maxwell

twitter @emmax

www.bignerdranch.com