

# CPSC 411A Mobile Device Application Programming

## ShoeApp



### Team Members:

Dhyey Desai 885451609

Saakshi Parikh 885147082

Mrunal Patil 884452392

Rishitha Bathini 885176255

### Instructor

Professor Farhang Zarrinkelk

Department of Computer Science  
California State University, Fullerton

12/08/2023

# REQUIREMENTS

## 1. Callbacks

- We implement 'onClickCategory', 'CategoryOnClickInterface'. To update the productList and retrieve data from the Firebase Realtime Database method.
- Another callback method used is 'onClickLike' Method. When a product's "like" button is clicked,
- We also implement 'setOnClickListner' method for some button.

```
holder.binding.btnLike.setOnClickListener { it: View!
    if(holder.binding.btnLike.isChecked){
        holder.binding.btnLike.backgroundTintList = ColorStateList.valueOf(Color.RED)

        likeClickInterface.onClickLike(currentItem)
    }
    else{
        holder.binding.btnLike.backgroundTintList = ColorStateList.valueOf(Color.WHITE)
        likeClickInterface.onClickLike(currentItem)
    }
}
```

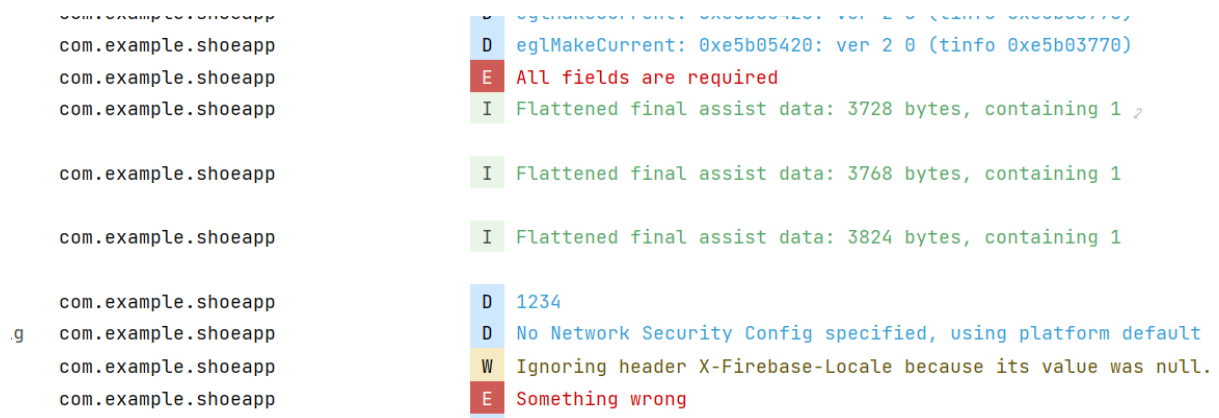
```
override fun onClickCategory(button: Button) {
    binding.tvMainCategories.text = button.text

    val valueEvent = object : ValueEventListener {...}

    databaseReference.addValueEventListener(valueEvent)
}
```

## 2. Logging

- For this requirement, we use LOG class methods.
- We use LOG.d() for the success message as well as LOG.i() for information basis and LOG.e() for error message on logcat



```
com.example.shoeapp
com.example.shoeapp
com.example.shoeapp
com.example.shoeapp

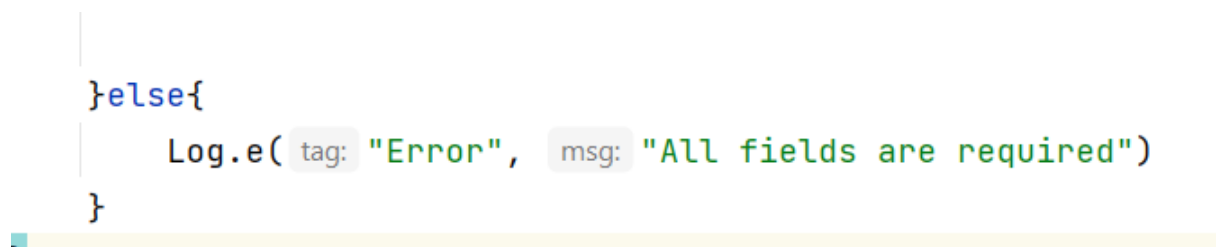
com.example.shoeapp

com.example.shoeapp

com.example.shoeapp
com.example.shoeapp
com.example.shoeapp
com.example.shoeapp

D eglMakeCurrent: 0xe5b05420: ver 2 0 (tinfo 0xe5b03770)
E All fields are required
I Flattened final assist data: 3728 bytes, containing 1
I Flattened final assist data: 3768 bytes, containing 1
I Flattened final assist data: 3824 bytes, containing 1
D 1234
D No Network Security Config specified, using platform default
W Ignoring header X-Firebase-Locale because its value was null.
E Something wrong
```

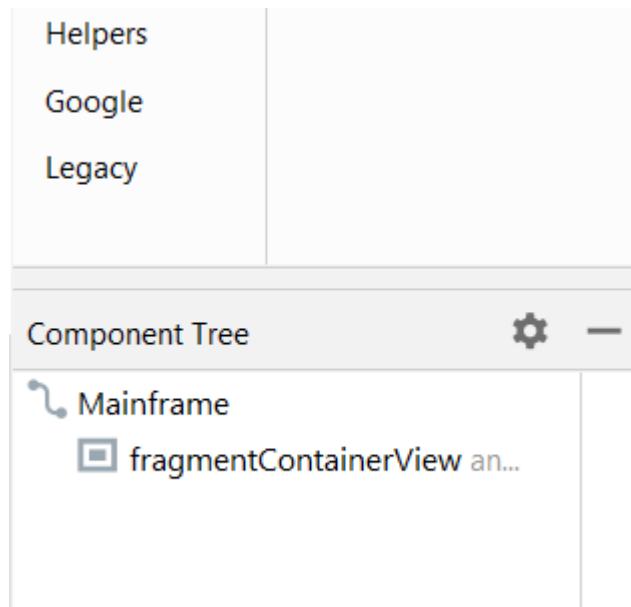
As you can see there are 2 errors and also it print my password “1234”. Below is screenshot of my code



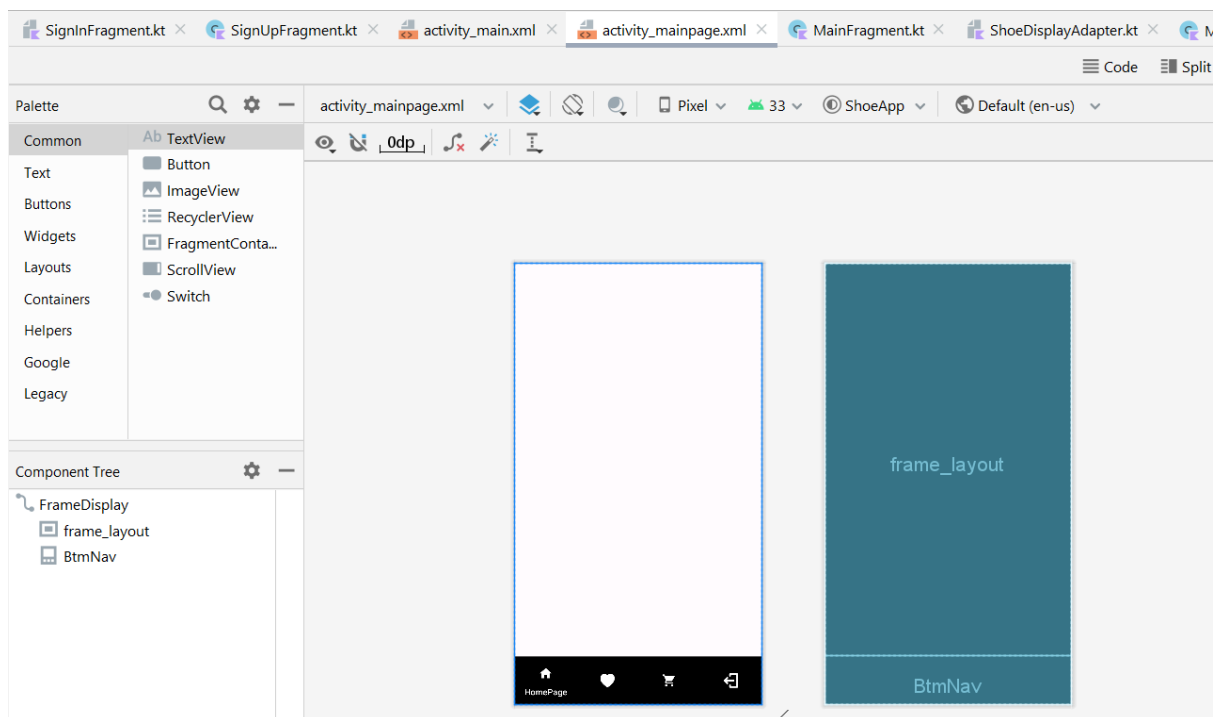
```
}else{
    Log.e( tag: "Error", msg: "All fields are required")
}
```

## 3. Layouts

- The ‘ConstraintLayout’ is used in “main\_activity.xml” define the constraints between the child view (FragmentManagerView) and the parent layout
- Also, we use the bottom navigation layout and frame layout to interact with different fragments in mainpage\_activity.xml.



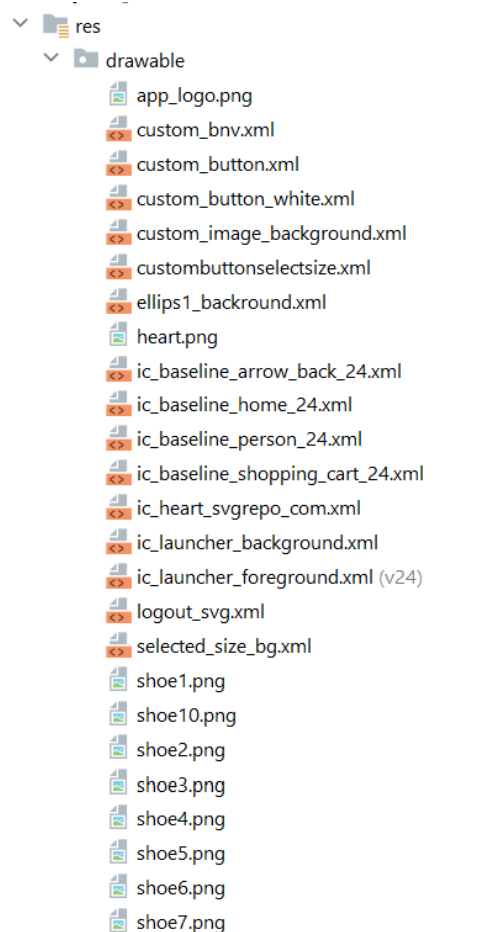
The above image is an activity\_main.XML file.



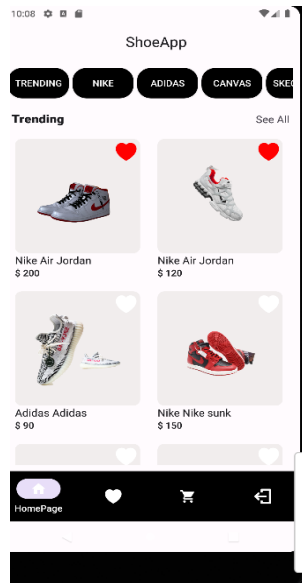
Here, on mainpage\_activity we can interact with different fragments within one activity which is the base requirement. The first layout “FrameDisplay” is constraintLayout.

#### 4. Resources

- We use different images for our products.
- First, we develop a database using these images which are in drawable package.
- Then, according to the product number or product id image will show to the user once he/she clicks on product.



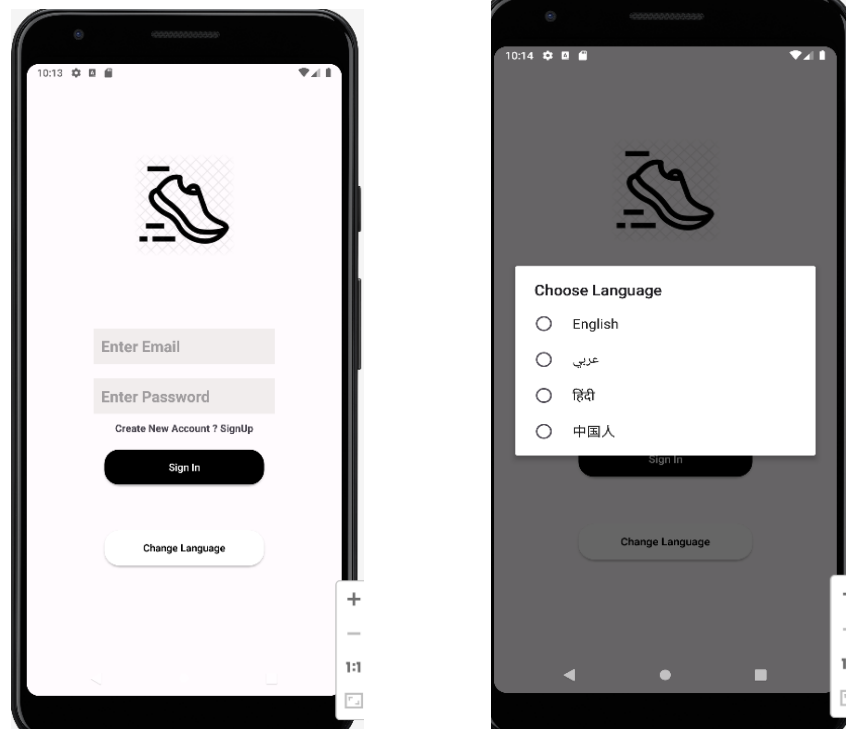
These are all images which we use to display our products.

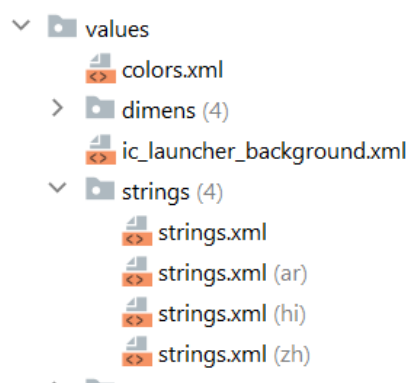


Above image is our HomePage screen.

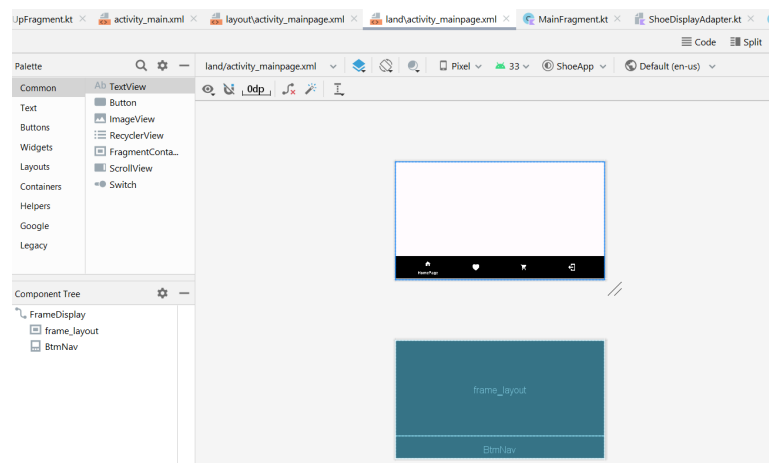
## 5. Resource Qualifiers

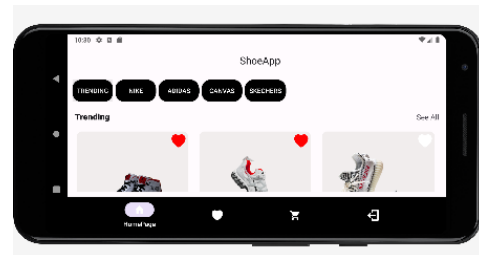
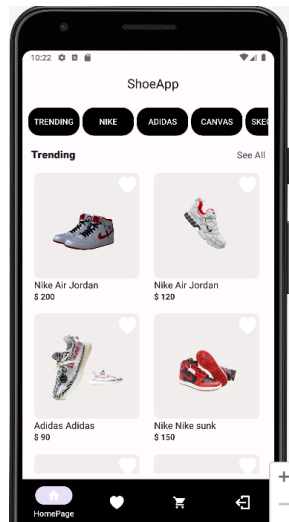
- First, We are adding 4 different languages (Hindi, Arabic, Chinese and English).
- We develop strings.xml for each language and also provide option on Singin Page to change the language by user side.
- Here, are some images of code as well as application



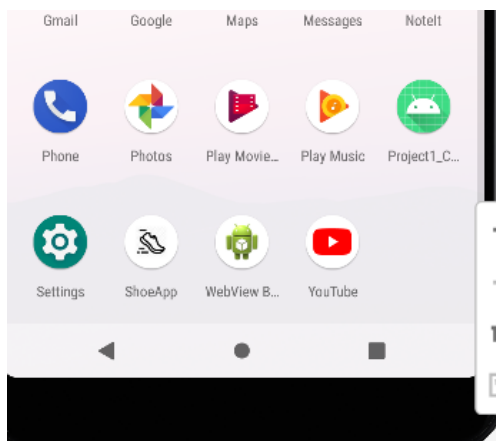


- Second, We add many different screen layout for landscap mode.
- We added package called “activity\_mainpage\_land”, “fragment\_likepage\_land”, etc.





- Third, We use the application icon for different sizes.
- Implement mipmap images for different configurations. For example, for tablets it will be square, for android it will be circle icons etc.



```

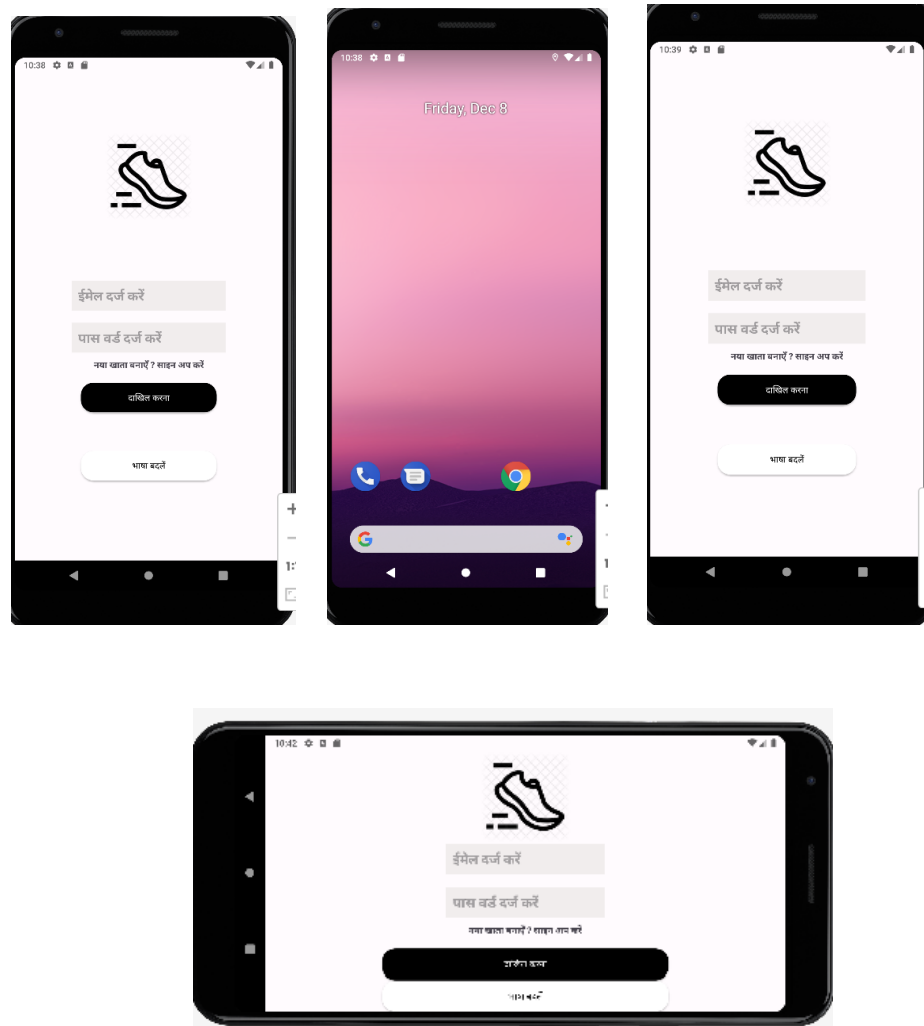
> menu
✓ mipmap
  > ic_launcher (6)
  > ic_launcher_foreground (5)
  > ic_launcher_round (6)

```

## 6. Persistence

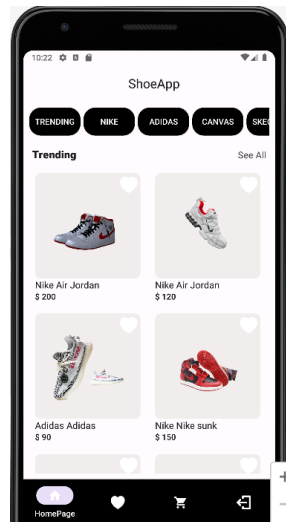
- We use firebase realtime database as well as firebase firestore to store user's data and retrieve the data even when the application is reopened.
- It will save the current state of user's, so whenever the user comes back, he/she can start from there.





## 7. User Interface

- We use different fragments as well as there two main activities.
- We develop user interface for all fragments such as MainActivity, DetailsFragment, CartFragment, SignInFragment, SignUpFragment, LikeFragment.
- We also implement a recycle view in fragment\_mainpage.xml for homepage to display all products.



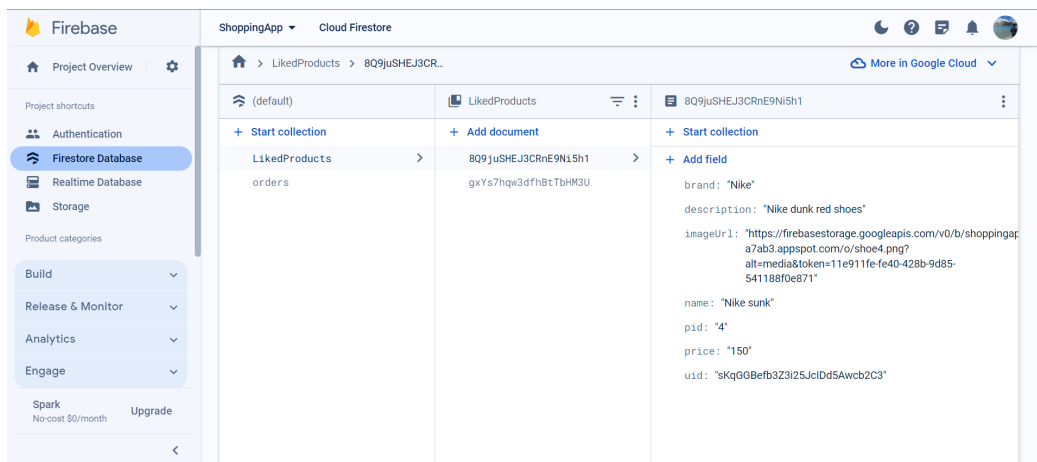
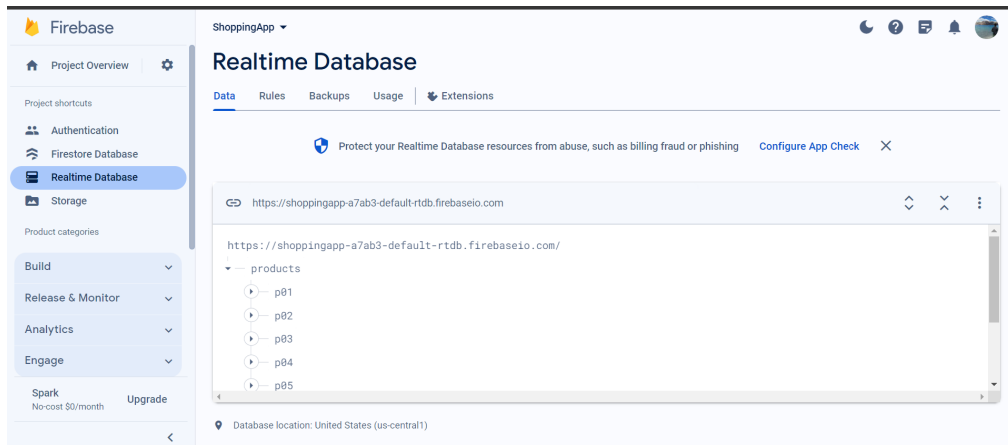
Here, there are three different fragments inside one fragment, which fragment container view. We can interact via bottom navigation and call each fragments from their only.

## 8. RESTful Activities

- As we use Firebase Database and firebase Firststore, first we make our database in firebase Firestore.
- Then, every user can signup in our app so it will create new data entry in our database.
- Also, once user signin, he/she can retrieve all his like products and cart products which he/she added during their last session.

```
auth.createUserWithEmailAndPassword(email, password)
    .addOnCompleteListener {task ->
        if(task.isSuccessful){
            requireActivity().toast( msg: "New User created")
            Log.d( tag: "Success", msg: "User Created")

            requireActivity().supportFragmentManager.popBackStack()
        }
        else{
            Log.e( tag: "Error", msg: "Something wrong")
            requireActivity().toast(task.exception!!.localizedMessage)
        }
    }
}
```

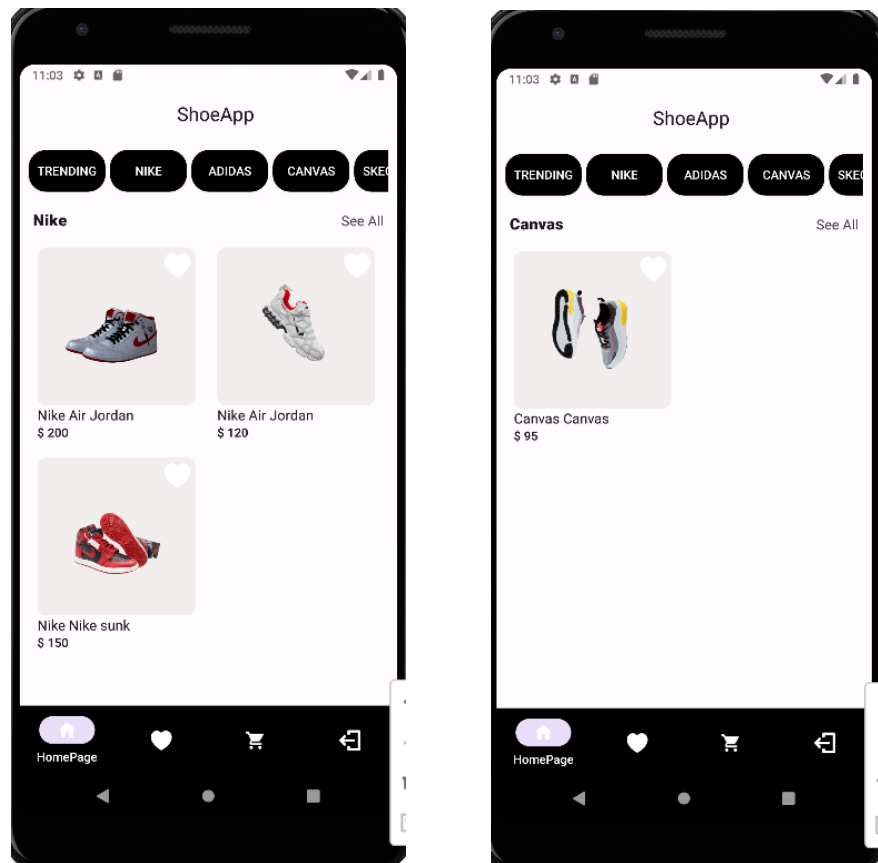


## Additional Requirements

As we are all four graduate students so, we implement 4 additional requirements

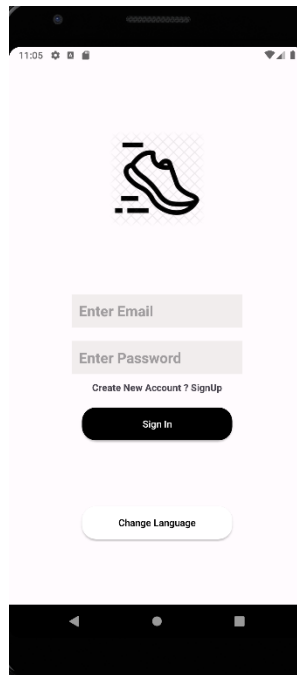
### 1. **CategoryList** (by Dhyey Desai )

- I implemented categories which is basically filter of our products.
- There are multiple categories in our app. Once user select any category he/she can show all related products only.



### 2. **Sign out** ( By Saakshi Parikh )

- I implemented sign out functionality at bottom navigation bar.
- Once user click on sign out he/she will navigate to sign in screen again where he/she can signin and come back to our app.



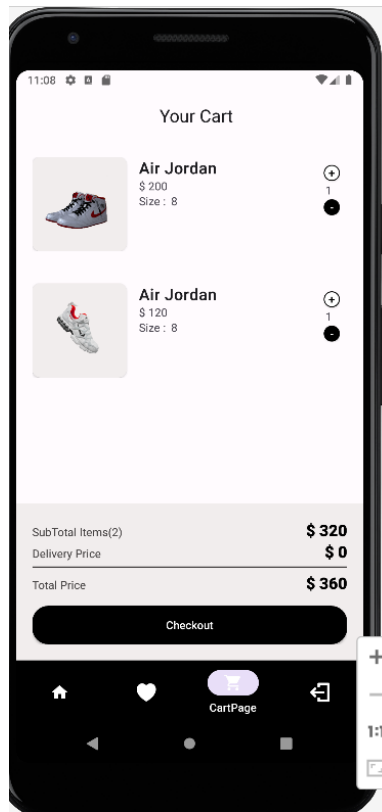
```
binding.BtmNav.setOnItemSelectedListener { it: MenuItem
    when (it.itemId) {
        R.id.mainFragment -> replaceFragment(MainFragment())
        R.id.likeFragment -> replaceFragment(LikeFragment())
        R.id.cartFragment -> replaceFragment(CartFragment())
        R.id.profileFragment -> {
            auth.signOut()
            finish()
        }

        else -> {

        }
    }
}
true ^setOnItemSelectedListener
}
```

### 3. Product Cart ( By Mrunal Patil)

- User can add products to the product cart and later on check it out as well.
- Also, user can adjust quantity over there.



#### 4. Product Remove on LongPress ( By Rishitha Bathini )

- I implemented long press functionality which will help to remove products from cart.
- Also, it will update database of particular user.

```

override fun onLongRemove(item: CartModel , position:Int) {

    orderDatabaseReference
        .whereEqualTo( field: "uid",item.uid) Query
        .whereEqualTo( field: "pid",item.pid)
        .whereEqualTo( field: "size",item.size)
        .get() Task<QuerySnapshot!>
        .addOnSuccessListener { querySnapshot ->

            for (item in querySnapshot){...}

        }
        .addOnFailureListener { it:Exception
            requireActivity().toast( msg: "Failed to remove")
        }
    }
}

```

## ScreenShot

