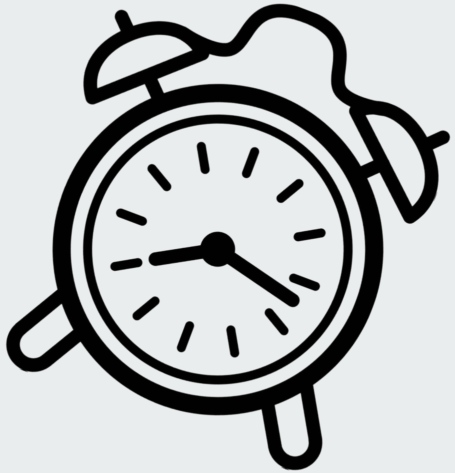
Clocks App
Group 3



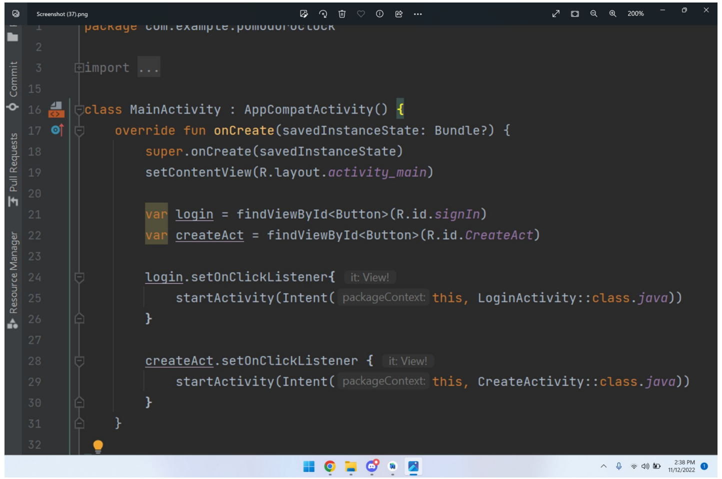
Prepared By:
Brian Ramirez, Rachel Arellano, Eric Lozano, Eduardo Terrazas

Features Overview

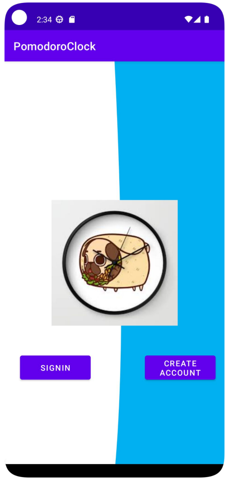
- Navigation
- Fragments
- Firebase
- Room(local)
- Unit Testing
- ViewModel and LiveData
- Recycler View
- Permissions
- Material Design
- Notifications

Main Activity

- MainActivity is a simple two button layout. If the user clicks on the login button they must have an existing account in our app. So they will be sent to the login page.
- If the user clicks on create account they will be sent onto the sign up page to create an account.

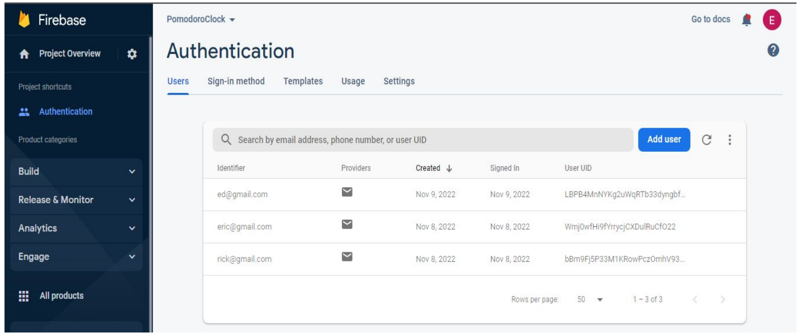


```kotlin
package com.example.pomodoroclock

import ...

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

        var login = findViewById<Button>(R.id.signIn)
        var createAct = findViewById<Button>(R.id.CreateAct)

        login.setOnClickListener{ it: View!
            startActivity(Intent( packageContext: this, LoginActivity::class.java))
        }

        createAct.setOnClickListener { it: View!
            startActivity(Intent( packageContext: this, CreateActivity::class.java))
        }
    }
```



Firebase Authentication(Login)

- These are emails we have generated so far and can be used to login into our clocks app. Through the use of firebase authentication via email.
- How we achieved this was by creating a view binding first. View binding allows you to simplify your code by not having to use findviewbyID and set root to layout you desire for example LoginActivity in this case.
- Login user function takes the inputted email and password from the user stores it and checks if its empty if not user.signinwithemail checks for emails in authentication and confirms and starts and activity if successful. If not display toast message.



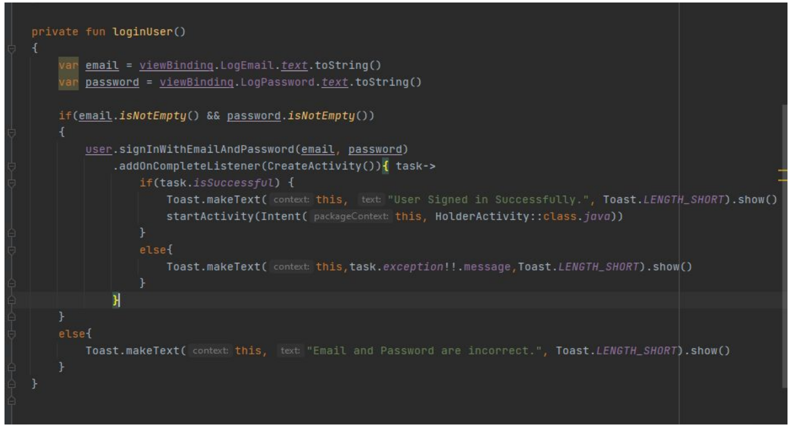
Authentication

Users | Sign-in method | Templates | Usage | Settings

Search by email address, phone number, or user UID | Add user

Identifier	Providers	Created ↓	Signed In	User UID
ed@gmail.com	✉	Nov 9, 2022	Nov 9, 2022	LBPB4MnNYKg2uWqRTb33dyngbf…
eric@gmail.com	✉	Nov 8, 2022	Nov 8, 2022	Wmj0wfHi9fYrrycjCXDulRuCfO22
rick@gmail.com	✉	Nov 8, 2022	Nov 8, 2022	bBm9Fj5P33M1KRowPczOmhV93…

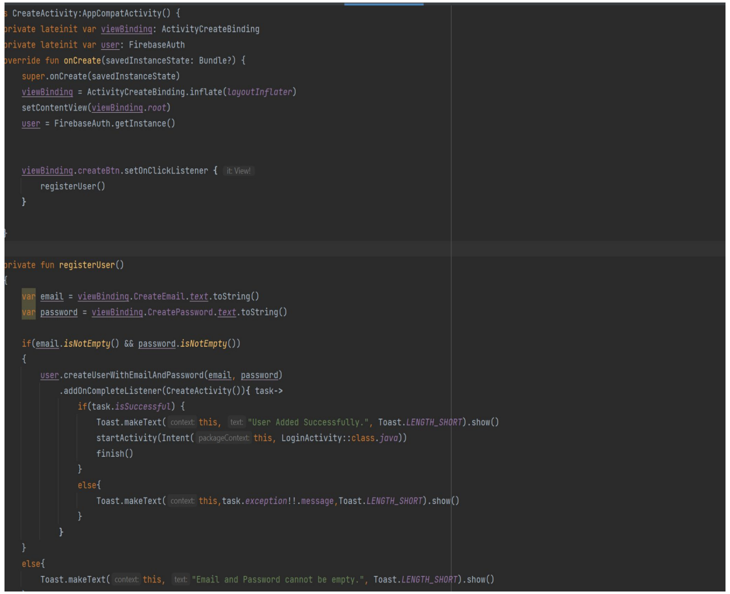
Rows per page: 50 1 – 3 of 3



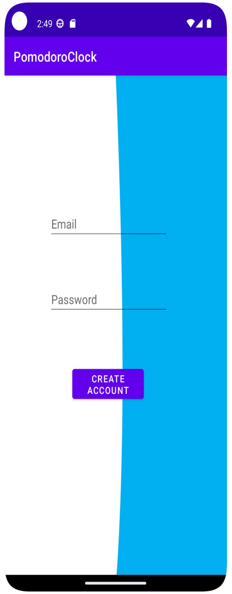
```
private fun loginUser()
{
    var email = viewBinding.LogEmail.text.toString()
    var password = viewBinding.LogPassword.text.toString()

    if(email.isNotEmpty() && password.isNotEmpty())
    {
        user.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(CreateActivity()){ task->
                if(task.isSuccessful) {
                    Toast.makeText( context: this,  text: "User Signed in Successfully.", Toast.LENGTH_SHORT).show()
                    startActivity(Intent( packageContext: this, HolderActivity::class.java))
                }
                else{
                    Toast.makeText( context: this,task.exception!!.message,Toast.LENGTH_SHORT).show()
                }
            }
    }
    else{
        Toast.makeText( context: this,  text: "Email and Password are incorrect.", Toast.LENGTH_SHORT).show()
    }
}
```

Signup Activity

- SImilarly we used view binding to simplify the code.
- As well as, in emulator we grab the users inputted email and password(should be 6 characters) they desire for their account.
- If the input is not empty we enter if and create the email if the email already exist toast message pops up saying email exists try again.
- Once account is created the user gets sent to login activity.

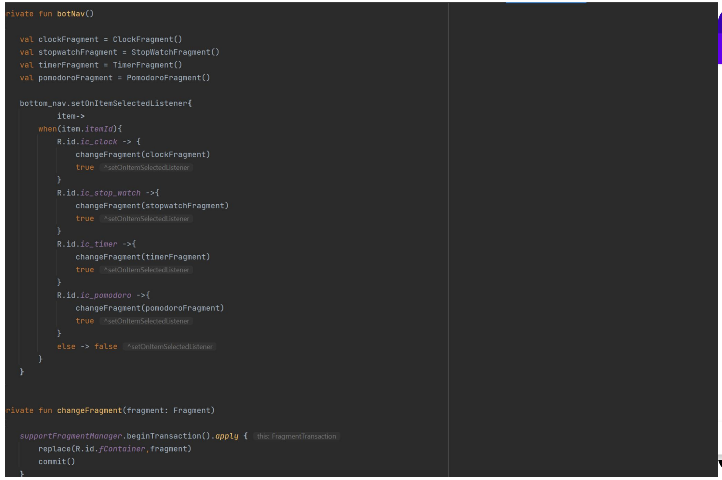


```
s CreateActivity:AppCompatActivity() {
rivate lateinit var viewBinding: ActivityCreateBinding
rivate lateinit var user: FirebaseAuth
verride fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   viewBinding = ActivityCreateBinding.inflate(layoutInflater)
   setContentView(viewBinding.root)
   user = FirebaseAuth.getInstance()


   viewBinding.createBtn.setOnClickListener { it: View!
       registerUser()
   }

}

rivate fun registerUser()
{
   var email = viewBinding.CreateEmail.text.toString()
   var password = viewBinding.CreatePassword.text.toString()

   if(email.isNotEmpty() && password.isNotEmpty())
   {
       user.createUserWithEmailAndPassword(email, password)
           .addOnCompleteListener(CreateActivity()){ task->
               if(task.isSuccessful) {
                   Toast.makeText( context: this,  text: "User Added Successfully.", Toast.LENGTH_SHORT).show()
                   startActivity(Intent( packageContext: this, LoginActivity::class.java))
                   finish()
               }
               else{
                   Toast.makeText( context: this,task.exception!!.message,Toast.LENGTH_SHORT).show()
               }
           }
   }
   else{
       Toast.makeText( context: this,  text: "Email and Password cannot be empty.", Toast.LENGTH_SHORT).show()
```

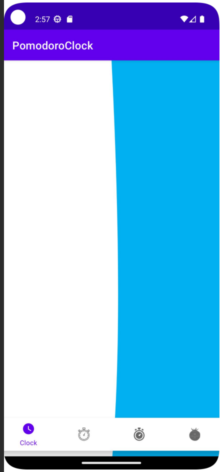


Bottom Navigation Activity

- We chose to use bottom navigation as our navigation format. Fragment container was used to swap between fragments,
- At the moment we have only created the bottom nav with 4 different fragments. Which will be filled in later down the line.
- Each fragment will contain a different variant for example, a clock, a stopwatch, a timer and a pomodoro clock.
- Each Icon has a little text field under it displaying what watch or clock type it is. Upon clicking the icon the fragments will be called and switched to.
- A floating action button with the ability to sign out of your account will be implemented later.



```
rivate fun botNav()

   val clockFragment = ClockFragment()
   val stopwatchFragment = StopWatchFragment()
   val timerFragment = TimerFragment()
   val pomodoroFragment = PomodoroFragment()

   bottom_nav.setOnItemSelectedListener{
           item->
       when(item.itemId){
           R.id.ic_clock -> {
               changeFragment(clockFragment)
               true  ^setOnItemSelectedListener
           }
           R.id.ic_stop_watch ->{
               changeFragment(stopwatchFragment)
               true  ^setOnItemSelectedListener
           }
           R.id.ic_timer ->{
               changeFragment(timerFragment)
               true  ^setOnItemSelectedListener
           }
           R.id.ic_pomodoro ->{
               changeFragment(pomodoroFragment)
               true  ^setOnItemSelectedListener
           }
           else -> false  ^setOnItemSelectedListener
       }
   }


rivate fun changeFragment(fragment: Fragment)

   supportFragmentManager.beginTransaction().apply {  this: FragmentTransaction
       replace(R.id.fContainer,fragment)
       commit()
   }
```



Future Implementations

- Room database is in the process of being developed. It will store specific times like alarm times of the user. It will hold the users based on the fire authentication code given by firebase.
- Viewmodel Live data will be implemented by adding a counter to see how many current users are creating accounts for the app. As well as view model live data will be used on the pomodoro if the user presses the start pomodoro button it will count up just so we know how many users actually using that specific clock feature.
- Recyclerview will read from room database and display the alarm times users have saved in there accounts. The pomodoro break times/active times and cool down times.
- Unit testing will be used with room database much like the in class example. To see if we are adding or removing correctly into the database.
- Notifications will be used when the pomodoro clock is about to finish its countdown as to push the users towards those final seconds.
- Permissions will be asking the user if we can get there in general region location so the clock can be set to there time zone automatically.
- Material design as you can see with the background in our mock ups we have already started to implement backgrounds and buttons. Towards the design aspect of our layouts. Will try to add clock design in the fragments and count down text boxes to make it seem more sleek and professional.