Version – 1.1.0+1

Majornumber.minornumber .buildnumber

Dependencies may make your life easier in the start but after a short while it may become bad .

If the dependencies are written by a single person beware , better if that is organization

On pub.dev you can search for dependencies(packages)

Dev\_dependencies : Only dependencies you will use at the time of making software and wont be binded with your software

We would need some dependencies in future like firebase\_core , firebase\_auth , cloud\_firestore and firebase\_analytics : server to store your data on the cloud on firebase these dependencies are needed , firebase is from google

flutter pub add cloud\_firestore // adding dependencies cloud\_firestore in flutter project through terminal

command to add the dependencies in pubspec yaml – you know what it is

Hot Reload vs Hot Restart – hot restart again builds your complete program

We killed myapp and transferred the data of material app into main

**Stateless vs Stateful –**

State – data or information

Stateful widget – keeps the information – something on the screen generally we can see

Stateless – can not contain mutable values

And then we kill the homepage

Now we plan to create a stateless widget just type stl

Scaffold – the basic building structure of the application

Go to container and press scaffold

In flutter everything is a widget either stateless or statefull

 return Scaffold(

      appBar: AppBar(

        title: const Text('Register'),

      ),

      body: Center(

          child: TextButton(onPressed: () {}, child: const Text("Register"))),

    );

Here scaffold is a code inside which is our widget appbar then we create an instance of AppBar class using its constructor taking parameter -

Then we get the title widget and title it register

Then we use the body widget to create the body that body has an instance of centre with the child being textbutton – inside onpresssed our definition of what button will do when pressed resides

We can use firebase to enable various login methods in our application

Try the firebase auth documentation

Text editing controller .. – you can grab the information from your textfield and access into the text button ; kinda like a proxy object for the textbutton to read

  // late - promises the program to get a value in the later stages

A created text editing controller needs to be disposed as well or else functions would find it hard to be disposed

Hint – piece of information that tells user what that text column expects and automatically removed when the user enters something in this textfield

     FirebaseAuth.instance.createUserWithEmailAndPassword(

                    email: email, password: password);

this thing returns a future and when something returns a future you have to put an await with it

this function return Future<usercredential> and future could not be red normally

we need to initialize the firebase through our register button and then we have to ensure that firebase is up and running before anything starts – widgetbinding

Instead of awaiting firebase everytime the user does something what we could rather do is perform the future and give a callback and on the basis of what the callback returns with the further tasks will be done – FutureBuilder

What we will do using FutureBuilder is that we will make it so that the column will be created only after the future which we await in firebase.init is returned

Snapshot – passed as a parameter in builder in future builder – it contains the state of the firebase app a snapshot – it is the call back that we get through the returned future

We can use this to create a loading screen

Stf – stateful widget

e.runtimetype – tells the type of exceptiojn

} catch (e) {

                          print("Oh Oh Exception:$e");

                        } finally {

                          print('Something did happen');

                        }

Catch anything type catch block – catch all block

Hot reload does not have any effect on what you do inside main

Instead of writin all that code of Firebase including Futurebuilder and whatnot in every view we can simply write that in homepage only

Email Verification –

 if (user!.emailVerified)

                print('You are a verified user\n');

null check condition

            // // final emailVerified = user?.emailVerified ?? false; // dont know why does not work well  print(user);

The points at which you saved your work - commit in git

BuildContext – a bit of information about the context where your widget is at and thus could be passed from one widget to another

Navigator.of(context).push(MaterialPageRoute(

                  builder: (context) => const VerifyEmailView(),

                ));

Simply the syntax to push a widget on the screen

Routes – a journey from source to dest one view to another

Anonymous route – like the one above we didn’t tell flutter about it beforehand while opposite for named route

 routes: {

      '/login/' : (context)=> const LoginView(),

      '/register/' :(context) => const RegisterView(),

    }

Route naming syntax map : string to Function(buildcontext)

 Navigator.of(context)

                  .pushNamedAndRemoveUntil('/register/', (route) => false);

            },

You can most probably use a named route inside the future builder but not unnamed

We can have quite a few widgets inside the appbar

Print – poorman debugging – print gets to log and it is accessible in android

Instead of print we can use log

Pop up menu button vs pop up menu item –

Every item in the pop up menu is of type popup menu item

The pop up menu button is the actual button that is used to display the popupmenu – has itembuilder where every item is popup menu item

Child is what the user sees and value is what you as a developer see

import 'dart:developer' show log;

import 'dart:developer' as devtools show log;

alias for dart developer

if you do not wanna pass a route pass \_ instead

push – having a screen and pushing a screen upon it

pushnamedandremoveuntil – remove until tells that when you don’t need a screen then remove it

avoid hardcoding everywhere – rather use instances of similar things rather than hardcoding it everywhere

    Navigator.of(context).pop();

Popping a pushed navigator

   showErrorDialogue(context, 'Error: ${e.code}');

e.code wont work simply with dollar use curly braces instead