Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

PROTOTYP AV MOBILAPPLIKATION FÖR TIDREDOVISNING

- Med Flutter

Joel Blomberg

2022:47

Datum för godkännande: 20.12.2022
Handledare: Björn-Erik Zetterman

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Joel Blomberg
Arbetets namn:	Prototyp av mobilapplikation för tidredovisning: med Flutter
Handledare:	Björn-Erik Zetterman

Abstrakt

I mitt examensarbete beskriver jag mitt programmeringsprojekt med Flutter. Syftet med arbetet är att lära mig ett nytt programmeringsspråk och ramverk samt att skapa en mobil applikation för tidsredovisning. Tidsredovisnings applikationen skall kunna skicka information till en databas som senare kan användas för tidsredovisning. I arbetet använde jag ramverket Flutter, programmeringsspråket Dart och använde mig av Firebase för databasen. Mitt resultat är en tidsredovisnings applikation som uppfyller till största delen de funktionskrav jag ställt upp. Jag har även lärt mig använda Dart och Flutter och har fått en bra förståelse för vad jag kan åstadkomma med dem.

Nyckelord (sökord)

utveckling av mobila applikationer, Dart, Flutter, tidsredovisning

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2022:47	1458-1531	Svenska	45 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
5.12.2022	16.12.2022	20.12.2022

DEGREE THESIS
Åland University of Applied Sciences

Degree Programme:	Bachelor of Information Technology/Bachelor of Engineering
Author:	Joel Blomberg
Title:	Prototype of Mobile Application for Time Reporting: with Flutter
Academic Supervisor:	Björn-Erik Zetterman

Abstract

In my degree thesis I describe my programming project using Flutter. The purpose of this work is for me to learn a new programming language and framework along with creating an application for time reporting. The time reporting application should be able to send information to a database that can later be used for time reporting. In the project I use the framework Flutter, the programming language Dart and Firebase for the application's databases. My result is a time reporting application that fulfills most of the requirements I have set. I have also learned how to use Dart and Flutter and have gotten an understanding of what I can achieve with them.

Keywords

development of mobile applications, Dart, Flutter, time reporting

Serial number:	ISSN:	Language:	Number of pages:
2022:47	1458-1531	Swedish	45 pages

Handed in:	Date of presentation:	Approved:
5.12.2022	16.12.2022	20.12.2022

INNEHÅLLSFÖRTECKNING

1. INLEDNING

I den här texten skriver jag om min demo-applikation som jag skapade med hjälp av Flutter, Dart och Firebase. Jag skriver också kring hur man lägger upp ens arbetsmiljö så att man kan jobba med Flutter på samma sätt som jag har. Jag går också igenom vad Flutter, Dart och Firebase är och hur de kan användas. Slutligen så går jag igenom steg för steg utvecklingen av min demo-applikation som är en tidsredovisnings-applikation som skickar upp uppdateringar till en molndatabas.

1.1 Syfte

Syftet för detta projekt är att lära mig ett nytt språk och ramverk samt att skapa en grund för en tidredovisnings-applikation som jobbar med en molnbaserad databas. Det ingår även korta tips från egna erfarenheter över hur man använder Flutter, Dart och Firebase. Samt hur man går tillväga för att återskapa slut produkten.

1.2 Avgränsningar

Jag bygger applikationen för Android enheter även om Flutter är ett cross-platform ramverk. Anledningen för detta är att Firebase som vi använder för vår databas behöver skillt vara upplagt för de olika enheterna som applikationen ska köra på. Vi antar också att det redan finns en annan applikation som läser och noggrannare sorterar användarens registrerade tidsredovisningar.

1.3 Metod

Jag la upp en flexibel arbetsplan för mig själv under 10 veckor och noterade ner de datum som jag behövde hålla koll på. Sen fortsatte jag med att ge mig själv ett generellt uppdrag för varje vecka med utrymme att ändra på något när det behövdes.
Under mitt projekts gång arbetar jag främst med vad som kan beskrivas som “Evolutionary prototyping”. Eftersom före jag började programmera skapade en kravlista för funktionalitet för min produkt och sakta betade igenom dem utifrån det som var av högst prioritering. Efter att jag har skapat en funktion har jag testat att det fungerar som förväntat och om det uppstår ett problem har jag åtgärdat det före jag gått vidare till nästa del av projektet

(Bernardez Fernand, 2020).

Eftersom många funktioner arbetar med varandra inom detta projekt gick jag ofta tillbaka för att finslipa delar av koden såsom det behövdes och för att se till att UI såg bra ut och passade med resten av applikationen. Men eftersom jag arbetade på detta sätt var det en del av de första kravspecifikationens lösningar som inte exakt passade bas beskrivningen och jag diskuterar senare i min slutsats på vad jag i efterhand skulle ha ändrat på samt mina lärdomar (Georgia Tech, 2015).

2. TEORI

I detta kapitel beskriver jag de komponenter, som bakgrund och teori som används för implementation (kap 4) av den kravspecifikation som finns beskriven i kapitel 3.

2.1 Dart

Dart är ett programmeringsspråk som skapades av Google 2011. Tanken med språket var att det skulle vara ett alternativ istället för JavaScript med den största fördelen i jämförelse med JavaScript är att det skickar en virtuell maskin till en browser så att man kan köra programmen skrivna med Dart i browsern samt att det kompilerar till “Native” maskinkod. När Dart först introducerades så var det inte använt av många men efter att Flutter Ramverket började använda det så har det nu blivit mycket populärare som ett alternativ (Thomas Limbüchler, 2022). Dart har en “just in time”-kompilator som med hjälp av Flutter låter en använda “hot reload” så att man kan se hur ändringarna i koden påverkar applikationen direkt (Dart,2022).

Dart är ett flexibelt typsäkert programmeringsspråk som resulterar i mindre möjliga buggar men också möjligheten till att använda dynamiska data former där det behövs. Dart har också Null Säkerhet så att objekt inte kan vara Null om man inte specifikt har sagt att det kan det. Dart har många aspekter som liknar både Java och JavaScript och är objektorienterat. Dart har även en pakethanterare som kallas för “pub”, som kan hjälpa utvecklare hitta hundratals paket med öppen källkod som man kan använda sig av (Fireship,2021).

2.2 Flutter

Flutter skapades 2015 av Google och släppte en alpha version av det till allmänheten 2017. Flutter är ett UI ramverk för att bygga applikationer på Android, IOS, Webben och på desktop. Flutter kombinerar fördelarna av Dart med en kraftfull Grafik motor och Flutters UI är uppbyggd som ett träd av “widgets”. Flutter har också ett stort bibliotek till färdiga widgets som kan hantera layout, animationer och mycket mer. Man kan som utvecklare skapa sina egna widgets i fall man har behov av sådana som saknas. Man gör detta genom att ärva sin

widget från “StatlessWidget” eller “StatefulWidget” klasserna och “override” deras bygg metoder (Flutter,2022).

Allting inom Flutter är en widget och alla metoder inom ens kod returnerar widgets och om något ändras inom en av dem så kommer Flutter bygga om ens widget genom att kalla på bygg metoden på grund av Flutters “Reactive UI”. “StatelessWidget” är oföränderlig och har ingen intern data så om man försöker skapa en Widget där data kan påverkas av användaren så bör man använda dig av “StatefulWidget” (Fireship,2020).

2.3 Firebase

Firebase är en samling av verktyg som arbetar främst kring moln-databaser. Firebase skapades 2011 och köptes av Google 2014. Firebase har många olika former av “real time” databaser och är idag en service för att skapa ‘Backend’ lösningar för många olika saker som ens applikation kan behöva. Firebase förvarar data på en moln databas det kan nås snabbt och smidigt från ens applikation och kan ta hand om allt från inloggning, dataanalys, moln funktioner till att hålla upp webbplatser (Fireship,2022).

Firebase har också Cloud Firestore som Jag har använt i min applikation som är en molnbaserad databas som är avsedd för en bredare mängd applikation lösningar och är väldigt smidig att sätta upp. Firebase har verktyg till nästan alla plattformar och smidiga sätt att förstora mängden utrymme som din applikation använder utifrån behov (Firebase, 2022).

2.4 Android Studio IDE

Android Studio är det officiella kodnings-verktyget för Android applikation skapande. Android Studio har support för en stor mängd programmeringsspråk och har en smidig lösning för nedladdning av eventuella plugin som kan behövas inom applikations skapandet. Android Studio har även möjligheten att skapa smidiga emulatorer där man kan se resultatet av ens applikationskod som uppdateras i realtid utan att starta om applikationen och har inbyggd support för Google Cloud (Brain of Plastic, 2022).

2.5 Applikationer för mobilt användande

Applikationen bör vara lättanvänd och fylla den nytta som vi har behov för den men vi behöver också tänka på om en mobilapplikation är rätt val för tjänsten. I många situationer så kan det vara bättre att skapa en hemsida eller applikation för en dator istället för att skapa en telefon applikation men om det finns någon direkt nytta som man kan få av det så kan det vara en bra ide för ett större system. I mitt fall jobbar jag utifrån antagandet att min tidsredovisnings applikation har en stationär variant där man sedan kan läsa rapporterna i mer detalj. Så vi vill att vår applikation ska vara ett snabb använt och okomplicerat alternativ för att rapportera till vår databas när våra användare jobbar. Vi behöver därför tänka på att appen framför allt ska ha lättanvända knappar och text lådor samt en passlig mängd av funktionalitet för användaren (The London Academy of IT Limited, 2022).

3. KRAVSPECIFIKATION

"När en användare loggar tid på en kund dokumenteras samtidigt allt som behöver noteras för fakturering, dokumentation och arbetstidsredovisning"

För mitt demoprojekt för att visa funktionaliteten av Flutter och vad man kan skapa med det har jag skapa ett program för att underlätta fakturering, arbetstidsredovisning och dokumentation genom att ge användaren ett smidigare sätt att spara den information som behövs.

3.1 Skallkrav

Dessa krav är av typen att de skall finnas, de kan ses som minimikrav.

1. autentisera användare för att användare med e-postadress och lösenord för att använda applikationen
2. välj kund från ett befintligt register, kundens namn visas
3. användaren kan rapportera tidpunkt (datum tid), kort beskrivning samt intern dokumentations-text

3.2 Börkrav

Börkrav är krav på produkten som inte måste vara med men gärna får vara med.

4. skapa nya användare med e-post och nytt lösenord
5. skapa nya kunder/klienter
6. komplettera rapportering med bilagor/bild
7. införa resekostnader och övriga utlägg

4. IMPLEMENTATION

Implementation av produkten beskrivs i detta kapitel. Det innehåller en del erfarenheter, en del logiska steg i processen som beskrivs och en del figurer som visar hur det ser ut som utvecklingsmiljö samt resultat. Jag arbetar utifrån min egna telefon som har Android version 11 eftersom det är en OnePlus telefon. Därför har jag valt att arbeta med Flutter på Android Studio och har följt Flutters installations rekommendationer för att arbeta i denna miljö (Flutter,2022).

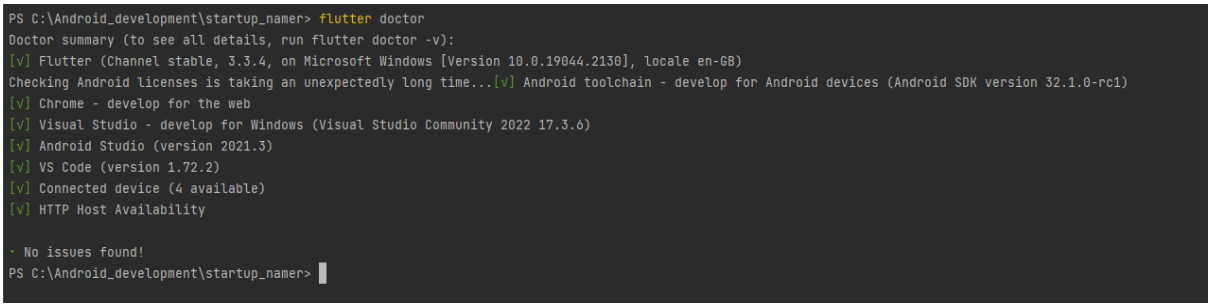
4.1 Upplägg

I början av mitt projekt planerade jag upp ungefär hur jag skulle arbeta över projektets gång samt vad som behövdes bli gjort. Jag började med att sätta upp min arbetsmiljö och bekanta mig själv med Flutter och Dart samt testade min miljö så att jag kunde vara säker att jag hade det jag skulle behöva senare. Så klart fick jag lära mig lite utöver det jag från första början hade tänkt mig med introduktionen an Firebase som en lösning för min applikationsdatabas men jag hade tillräckligt med tid för det också.

4.1.1 Installation

Mitt första steg var att se till att min arbetsmiljö var fullt funktionell samt att jag hade det jag behövde för att börja arbeta. Jag valde att arbeta i Android Studio eftersom jag har jobbat med det tidigare. Flutter ändå behöver det för att fungera (Flutter låter användaren arbeta i många olika kod editorer men behöver ändå vissa program för att fungera rätt) och jag har redan några telefon emulatorer uppsatta samt möjligheten att köra och testa applikationen på min egna smartphone via en kabel. Jag följde instruktionerna som man kan hitta på Flutters hemsida (Flutter,2022) för att installera Flutters SDK på Windows. Instruktionerna börjar med att gå igenom allt ens enhet behöver ha för att det skall fungera såsom Windows Version (10 eller senare), hur mycket utrymme det behöver (ca 2 GB inkluderar inte utrymme för IDE/verktyg) och verktyg utanför arbetsmiljön som man behöver (Windows PowerShell 5.0 och Git for Windows 2.x) De har länkar via hemsidan till det mesta man behöver vilket underlättade mycket gällande installationen.

Flutters hemsida har också instruktioner för hur man ska gå tillväga om man behöver uppdatera systemets sökvägar. Efter att man har laddat, Extraherat och Installerat Flutter SDK till en lämplig plats rekommenderar jag och Flutter att man kör “Flutter Doctor” i en kommandoprompt där man har Flutter installerat. Detta är ett kommando som kommer berätta vad man möjligen saknar för att få Flutter att fungera (Figur 1).



Figur 1. resultat av Flutter doctor

Om alla checkar är gröna som man kan se i figuren ovan (Figur 1) innebär det att man redan har allt som Flutter behöver för att fungera och att Flutter kan hitta det utan problem. Men annars kommer Flutter Doctor ge en tips på vad man saknar samt hur man kan gå tillväga för att få det att fungera. Problemen kan uppstå genom att de programmen som Flutter behöver saknas eller om dom är installerade på en plats av datorn som Flutter inte kan hitta. Ett exempel som jag nämnde tidigare är Android Studio men det behöver även Visual Studio för att fungera (Tips: Visual Studio Code fungerar inte det behöver vara Visual Studio 2022 eller Visual Studio Build Tools 2022).

Man behöver också se till att man har gått med på Androids Licenser men detta kan göras genom att köra “Flutter Doctor --android-licenses” i en kommandoprompt så att man kan läsa igenom termerna för varje licens och gå med på dem.

För att underlätta testandet av Android applikationer har Flutter instruktioner för hur man kan sätta upp en emulator eller en Androidtelefon så att man kan testa ens applikation på dem. (Notera att det kan vara användbart att läsa igenom dom även om man redan har emulatorer från tidigare i fall det uppstår något problem.)

4.1.2 Arbetsmiljö

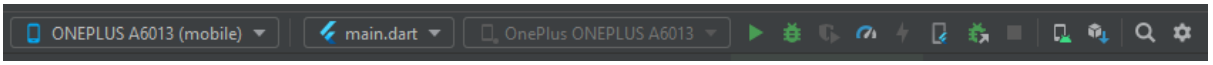
Flutter ger en ett par olika alternativ för kod editorer som man kan använda: Visual Studio Code, Android Studio, IntelliJ och Emacs är de som nämns på hemsidan. Som sagts tidigare har jag valt att jobba inom Android Studio så jag kommer snabbt gå över det man behöver ha för att börja. Samma process gäller om man vill jobba inom IntelliJ men om man vill jobba inom en annan kodeditor finns det instruktioner på Flutters hemsida. Om man använder sig av Windows eller Linux är det ganska enkelt. När man öppnar Android Studio behöver man gå till “File > Settings > Plugins”. Sedan går man till Marketplace så att man kan söka upp “Flutter” och “Dart” plugin och installera dom. Om man använder en Mac behöver man på ett liknande sätt öppna Android Studio och gå till “Preferences > Plugins”. Där man kan välja “Flutter” och “Dart” plugin och installera dem. Men efter att man har följt dessa steg borde man ha allt i ordning för att jobba med Flutter på Android Studio.

4.1.3 Starta Projekt

När man börjar jobba med ett Flutter Projekt rekommenderar jag att man först skapar en början för ens projekt via en kommandoprompt före man öppnar det i Android Studio.
Det finns andra sätt att börja men jag ser detta som en bra punkt att börja från eftersom man får tydliga exempel på hur Flutter samt Dart kan få en applikation att se ut samt en par exempel på några “widgets” som kan vara bra att jobba med. Man gör detta genom att ta sig till där man vill skapa projektet i kommandoprompten och skriv “Flutter create project_name” och byt ut “project_name” till vad man vill att det ska heta.
Som ett exempel så här gjorde jag när jag skapade början för min applikation (Figur 2).

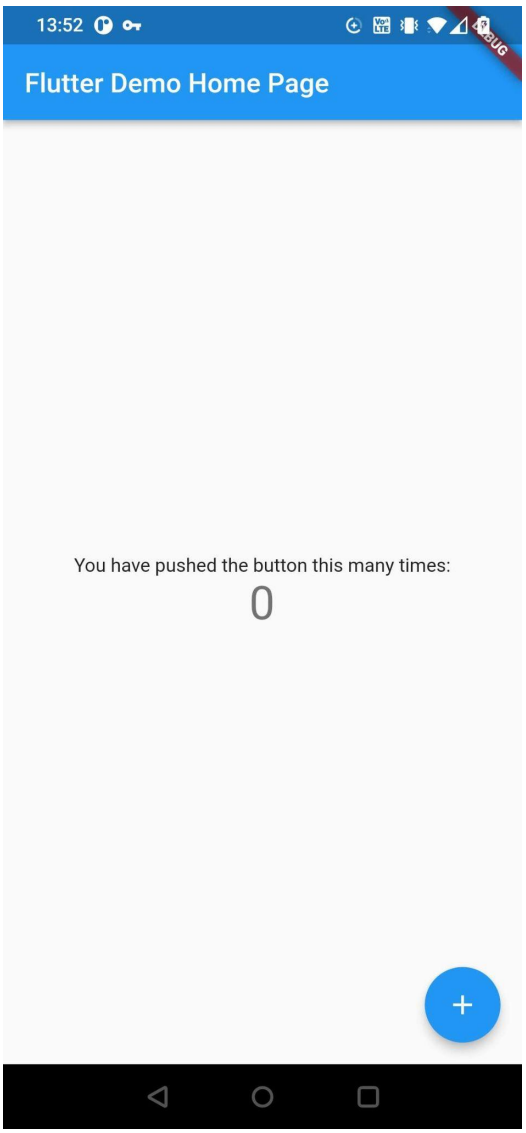
```
C:\Android_development> flutter create tid_demo
Creating project tid_demo...
Running "flutter pub get" in tid_demo...                        1,438ms
Wrote 127 files.

All done!
In order to run your application, type:

  $ cd tid_demo
  $ flutter run

Your application code is in tid_demo\lib\main.dart.


C:\Android_development>
```

Figur 2. terminal med Flutter create kommandot



Figur 3. Verktyg för att köra applikation

För att köra ett Flutter Projekt via Android Studio går man upp kring det högra hörnet där man kan se på vilken enhet man försöker köra applikationen samt vilken av projektets Dart-filer som den använder som "main". Här har man verktyg för olika sätt att köra ens applikation samt sätt att hålla koll på de enheter man har möjlighet att köra sin kod på. När jag sen öppnade mitt projekt och testade köra koden så fick jag fram detta utan större problem (Figur 4).



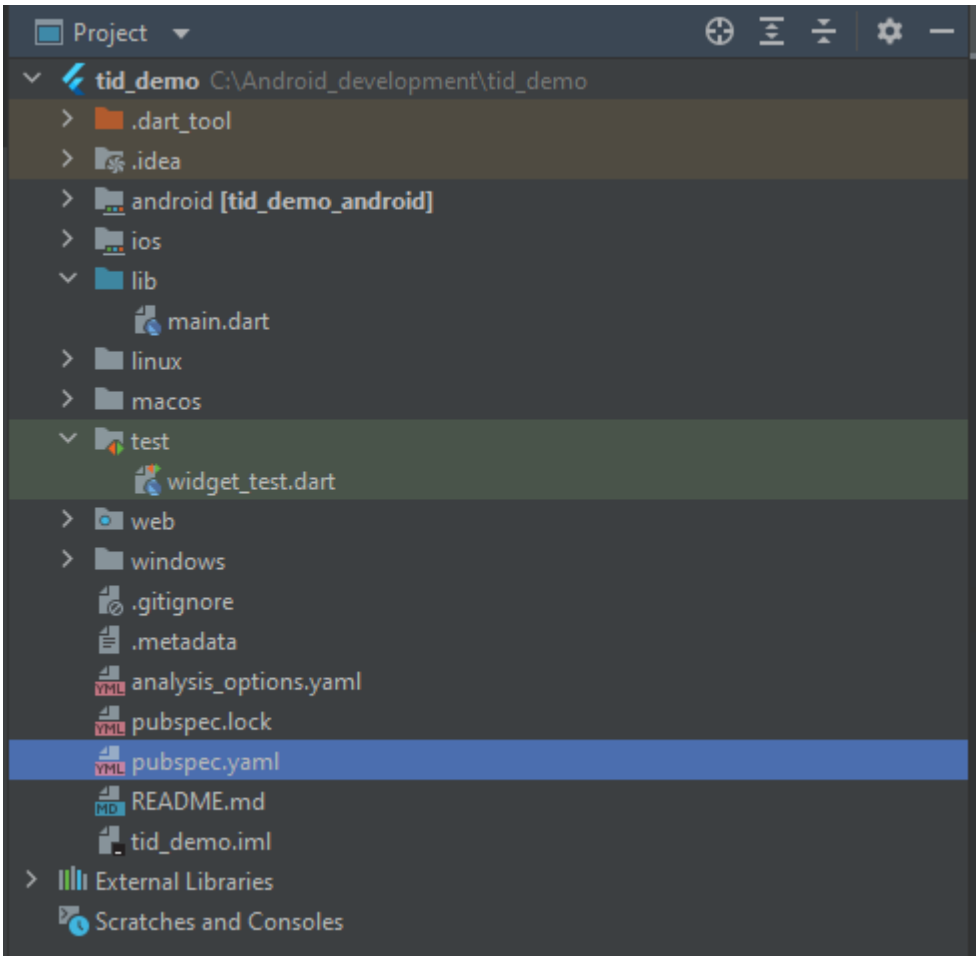
Figur 4. resultat av Flutter create

Som man kan se i figuren ovan (Figur 4) är demo koden som det ger ingenting helt fantastiskt men det ger en bra början. Men om man inte vill skapa början på sitt projekt på det här sättet så finns det andra alternativ och instruktioner på Flutters hemsida.

4.2 Tips

Något som är ganska litet men kan vara bra att tänka på är att ändra ens projekttyp från "Android" till "Projekt" när man öppnar sitt Flutter Projekt i Android Studio så att man kan se alla filer som man behöver jobba med som ni kan se nedan (Figur 5 och 6).

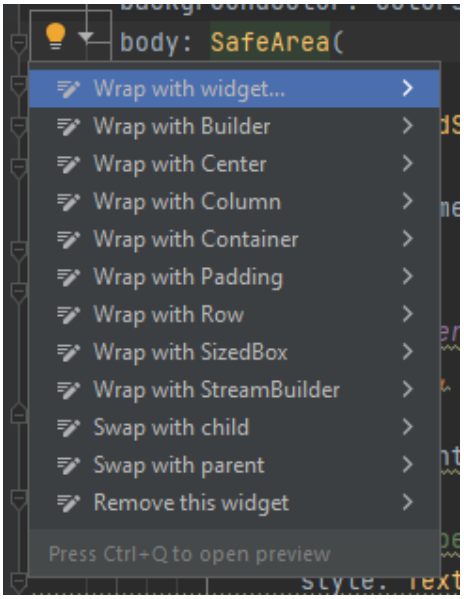
Figur 5. Android studio default projekt inställning



Figur 6. Projekt-inställning som bör användas

Anledningen för detta är att man behöver lättare åtkomst till "pubspec.yaml" eftersom det är där man kan lägga till alternativa beroenden som man behöver när man jobbar på sitt projekt.

Ett annat användbart tips när man jobbar med Flutter är att använda sig av wrap funktionalitet som ni kan se nere i (Figur 7). Wrap Funktionaliteten kommer sätta en widget runtom den widget man har markerat. Den har en bas widget utan mer funktionalitet men det kommer även med andra färdiga widgets för att göra arbetet snabbare.



Figur 7. Android studios wrap förslag för widgets

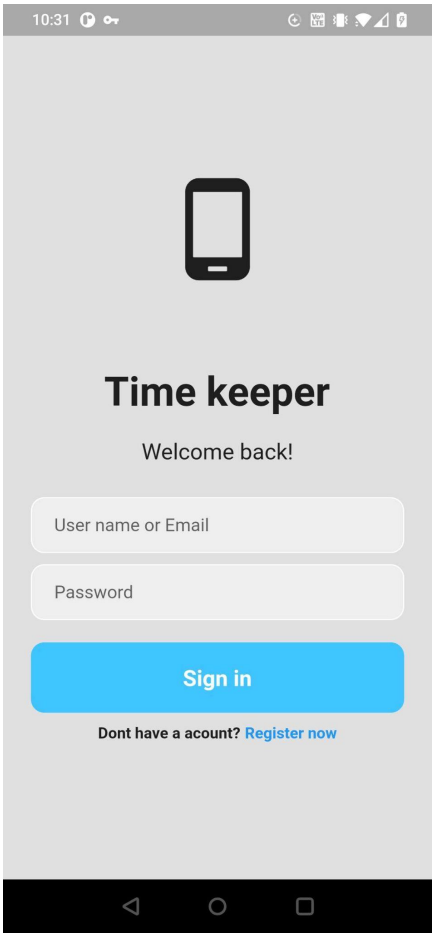
Detta kan snabbt hjälpa om man behöver lägga till en ny "widget" eller något liknande i koden. När man behöver ett tom utrymme för Användarens UI skapa en liten "SizedBox()" där man vill ha ett mellanrum. Man kan specificera lådans höjd och bredd med "height:" och "width:" inuti dem.

4.4 Resultat Step by Step

Före jag började jobba läste jag så klart mer om hur man ska gå tillväga för att använda sig av Flutter, Dart och Firebase.

4.4.1 Början av Projekt

När jag hade skapat basen för min applikation började jag först med att bygga upp Användarens UI (användarens interface) för inloggningssidan och så att jag kunde få en bättre idé hur Jag slutligen ville att min applikation skulle se ut. Jag gjorde några mindre ändringar senare men efter att ha letat upp några exempel och testat runt kom jag fram till detta (Figur 8).



Figur 8. Användarens interface första version

Jag valde att senare byta ut ikonen och vid det här läget hade jag ännu inte flyttat ut denna sida från “main.dart”. Inom Flutter så är nästan allting en widget detta innebär att nästan allt som man kan se och inte se i sin layout kommer bestå av flera lager widgets inuti varandra. Det finns widgets som hanterar text, knappar, figurer och ikoner m.m. samt widgets som fungerar som containrar för dessa widget som fokuserar mera på var på skärmen är alla de widgets som finns inuti dem. Jag byggde upp mina textfält så för att få fram stilen som jag kände mig nöjd med som kan ses i Figur 9.

```
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 25.0),
  child: Container(
    decoration: BoxDecoration(
      color: Colors.grey[200],
      border: Border.all(color: Colors.white),
      borderRadius: BorderRadius.circular(12),
    ),  // BoxDecoration
    child: Padding(
      padding: const EdgeInsets.only(left: 20.0),
      child: TextField(
        controller: _usernameEmailController,
        decoration: InputDecoration(
          border: InputBorder.none,
          hintText:'User name or Email',
        ),  // InputDecoration
      ),  // TextField
    ),  // Padding
  ),  // Container
),  // Padding
```

Figur 9. Upplägg för textfält

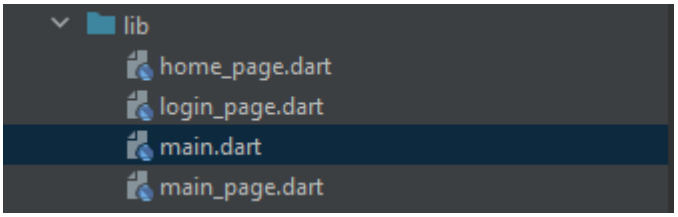
Som ni kan se har jag använt “Padding()” för att lägga till mellanrum runt om textfälten där det passar. Sedan har jag lagt en “Container()” inuti den som får dekoration från “BoxDecoration()” för att ge den en passande färg och form. Om jag inte gjorde detta skulle containern automatiskt ha en helt vit bakgrund och troligen smälta in med resten av bakgrunden. Jag skapade sedan knapparna på ett liknande sätt (Figur 10).

```
Padding(
  padding: const EdgeInsets.symmetric(horizontal: 25.0),
  child: GestureDetector(
    onTap: signIn,
    child: Container(
      padding: EdgeInsets.all(20),
      decoration: BoxDecoration(
          color: Colors.lightBlueAccent,
          borderRadius: BorderRadius.circular(12),
      ),  // BoxDecoration
      child: Center(
        child: Text(
            "Sign in",
            style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
                fontSize: 20
            )  // TextStyle
        ),  // Text
      ),  // Center
    ),  // Container
  ),  // GestureDetector
),  // Padding
```

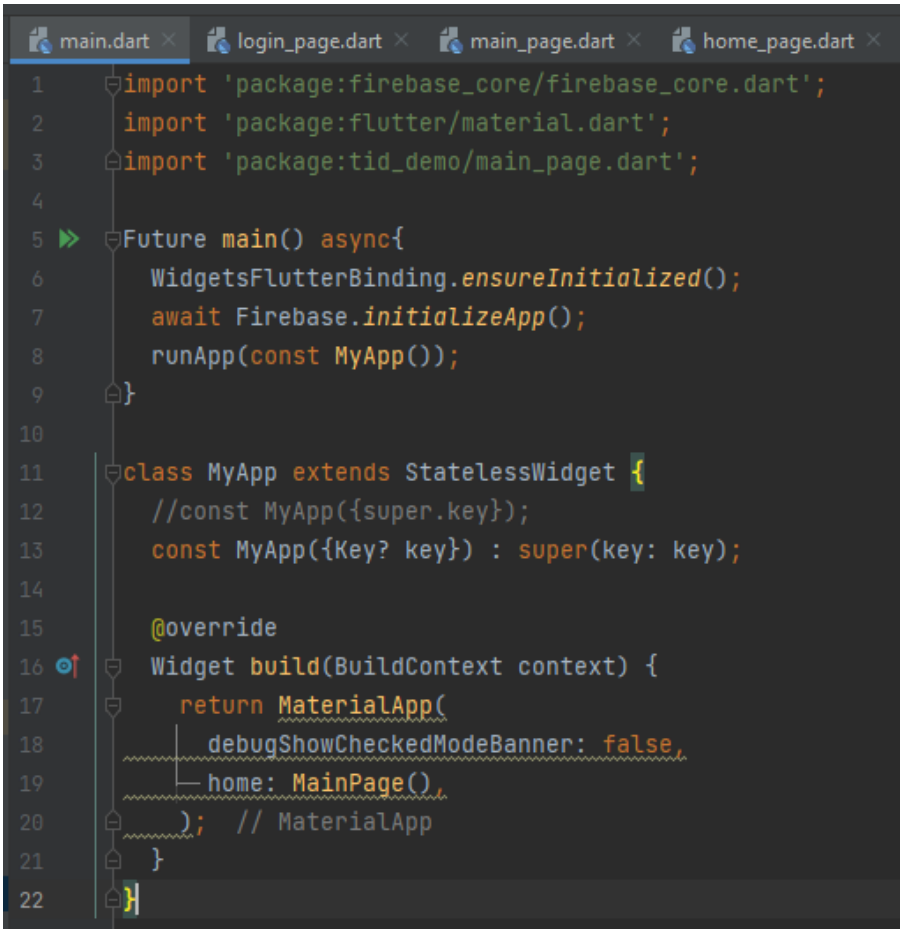
Figur 10. Upplägg för knappar

4.4.2 Utveckling

Efter att jag hade jobbat på det gick jag vidare till att dela upp applikationens sidor och flyttade koden som var i “main.dart” (Figur 12) till “login_page.dart”. Jag skapade “main_page.dart” och “home_page.dart” för att enkelt byta vad applikationen visar upp (Figur 11).

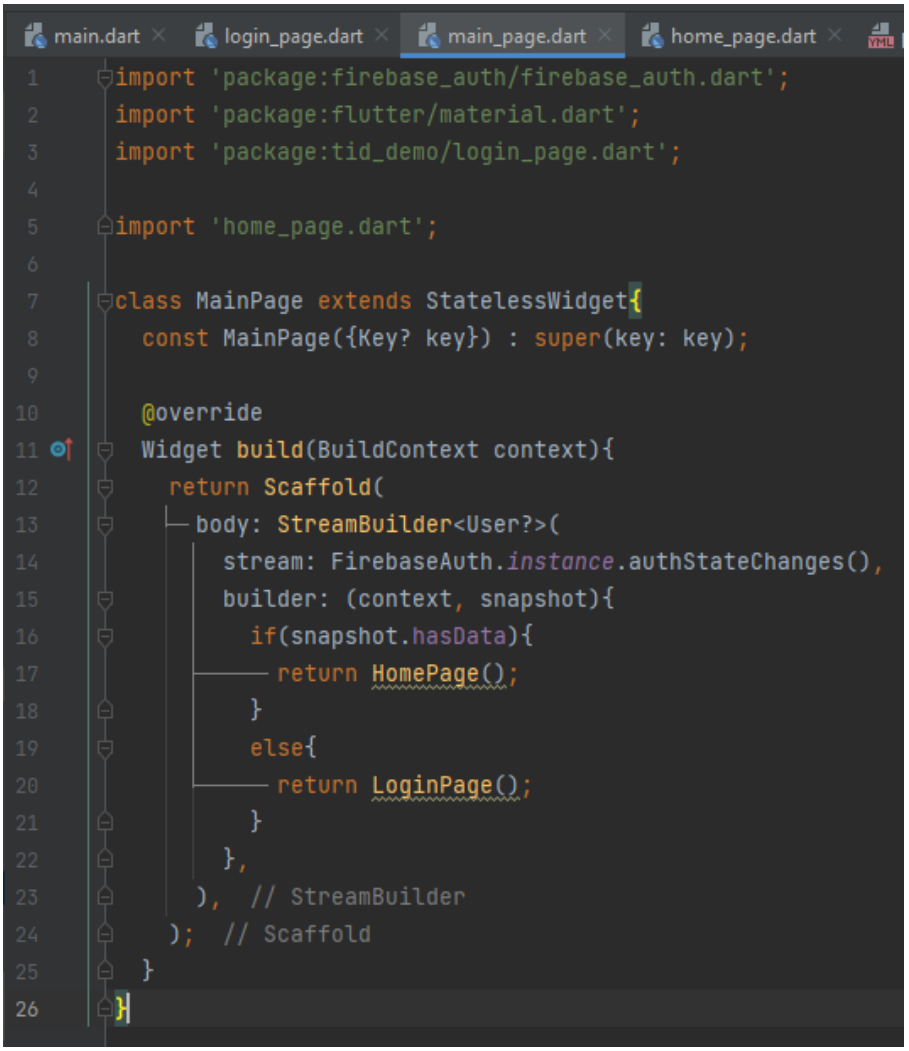


Figur 11. Alla Dart-filer för projektet



Figur 12. main.dart.

"main.dart" kollar att Flutter och Firebase har initialiseras. Vi startar sedan applikationer med "runApp(const MyApp);" i "MyApp" returnerar vi vår "MaterialApp" och säger att "home:" ska bestämmas utav "MainPage()" som är en funktion som vi hittar i "main_page.dart" (Figur 13).



Figur 13. main_page.dart

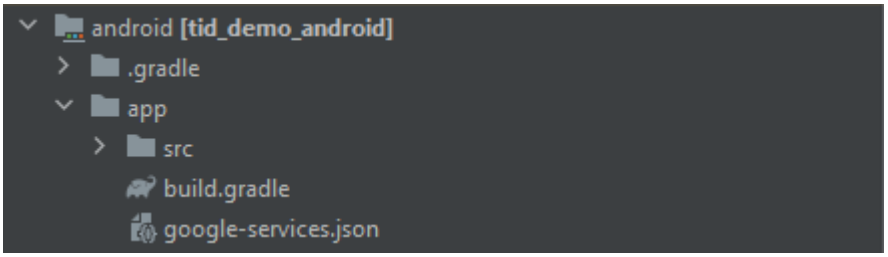
"main_page.dart" returnerar en "Scaffold()" som inuti sin "body:" har en ström som med hjälp av "FirebaseAuth" kollar om någon är inloggad på applikationen. Utifrån det bestämmer en "if" om användaren åker till inloggningssidan "LoginPage()" eller applikationens öppnade sida "HomePage()" genom att returnera dem. Men för att det här ska fungera så behöver man se till att Firebase är uppsatt ordentligt. För att göra detta så måste man ta sig till "pubspec.yaml" som kommer synas längre ner till vänster som jag nämnde tidigare (Figur 6). Notera även att det ska vara "pubspec.yaml" som man ska öppna och inte "pubspec.lock". När man har öppnat "pubspec.yaml" ska man scrolla ner till sina beroenden där man kommer lägga till några fler beroenden (Figur 14).

Figur 14. pubspec.yaml dependencies

Om man har skapat sitt projekt på samma sätt som jag gjorde så borde man redan hitta Flutters SDK där. I bilden ovanför (Figur 14) kan man se alla de beroenden som jag har lagt till just för detta projekt men för att logga in med hjälp av Firebase behöver man främst en version av "firebase_auth:". Men man behöver också sätta upp Firebase för de apparater som ens applikation kommer behöva köra på. Man kan sätta upp det så att det fungerar med samma databas för fler plattformar men eftersom jag som en grund jobbar utifrån en Androidtelefon kommer jag gå kort gå över vad man kommer behöva göra för att få det att fungera med en Androidapp.

Först bör man gå till Firebases hemsida och skapa ett Firebase Projekt. När man skapar projektet på steg 1 av 3 döper man ens projekt till något passande och Google kommer ge en unik id för det projektet. Efter det går man vidare till steg 2 av 3 där jag rekommenderar att den första gång man skapar ett Firebase Projekt att man stänger av Google analytics för projektet men om man tror att man kommer behöva det senare och redan är bekväm med Firebase kan man lämna det på. Sen går man vidare till steg 3 av 3 där Google kommer ta en stund för att lägga upp grunden för ens projekt. Sedan när man trycker på "continue" kommer ens Firebase Projekt vara uppsatt men det har ännu inte kopplat till en app. För att lägga till det till ens Android applikation behöver man först klicka på Android Logon som borde finnas runt mitten av skärmen men om man redan har en app kopplad till ens projekts databaser så finns alternativen för att lägga till en ny app lite högre upp på skärmen. När man har klickat på detta så kommer man behöva ge Firebase lite information angående ens app. Det Firebase kommer behöva är applikationens Android Paketnamn.

För att hitta detta så behöver man öppna projektet för sin applikation i “Android Studio” och öppna mapparna “android > app” och hitta sin “build.gradle” fil. Notera att detta inte är den enda “build.gradle”filen som kommer finnas i projektet efter man har skapat det (Figur 15).



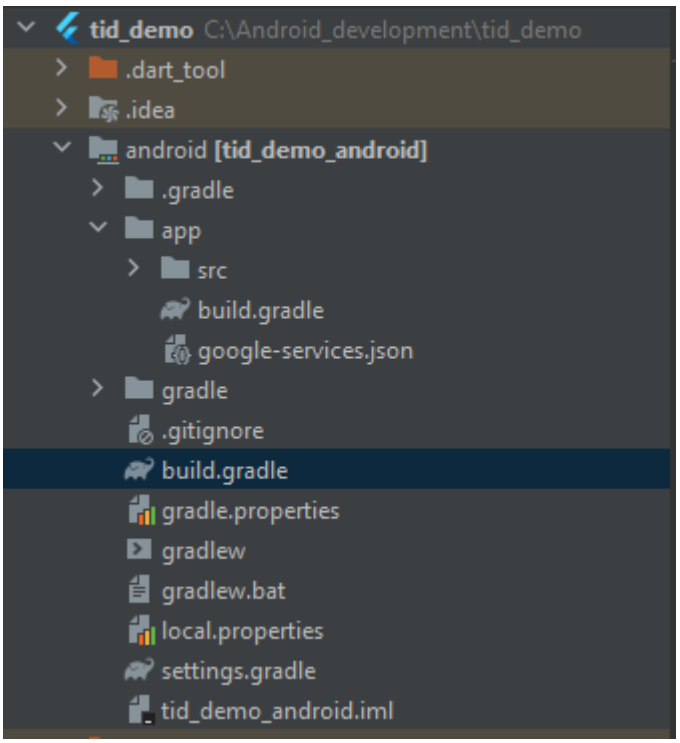
Figur 15. Väg till första build.gradle

När man har öppnat “build.gradle” filen bör man scrolla ner tills man hittar “applicationId: “com.example.something” ”. Denna string är ens Android Paketnamn så kopiera det och sätt in det på Firebases hemsida. Notera också att man behöver ändra “minSdkVersion” till 21 för att Firebase ska fungera (Figur 16).

```
46    defaultConfig {
47        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).
48        applicationId "com.example.tid_demo"
49        // You can update the following values to match your application needs.
50        // For more information, see: https://docs.flutter.dev/deployment/android#reviewing-the-build-configuration.
51        minSdkVersion 21 //flutter.minSdkVersion
52        targetSdkVersion flutter.targetSdkVersion
53        versionCode flutterVersionCode.toInteger()
54        versionName flutterVersionName
55    }
56
```

Figur 16. Applikations Id och min Sdk Version inuti första build.gradle

Efter det kan man välja vad ens applikation på Firebase ska heta och ge en debug logging certifiering men det är inte nödvändigt att göra. När man har klickat på “Register app” kommer Google att generera en “google-services.json”fil som man behöver ladda ner och sätta in den i samma mapp som en “build.gradle” som man kan se i en tidigare figur (Figur 15). När man har gjort detta och går vidare till nästa steg är det fortfarande ett par saker som man behöver fixa i sin projektnivå “build.gradle” som man hittar längre ner i ens “Android”mapp. Det är alltså inte samma fil som jag nämnde tidigare (Figur 17).



Figur 17. Andra build.gradle

Inuti denna “build gradle” så behöver man infoga den nya “classpath”linjen av kod som Google ger en i dess beroenden (Figur 18).

```
7
8       dependencies {
9           classpath 'com.android.tools.build:gradle:7.1.2'
10          classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
11          classpath 'com.google.gms:google-services:4.3.13'
12      }
13  }
```

Figur 18. Andra build.gradle dependencies-ändringar

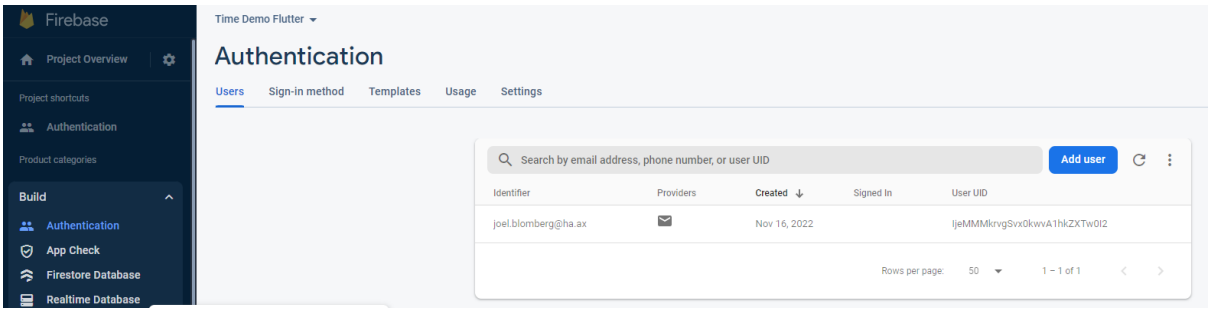
I det sista steget behöver nu gå tillbaka till den tidigare “build.gradle” som man hittar i “android > app”mappen. Där behöver man infoga en “apply plugin:”rad som Google ger (Figur 19).

```
23
24  apply plugin: 'com.android.application'
25  apply plugin: 'kotlin-android'
26  apply from: "$flutterRoot/packages/flutter_tools/gradle/flutter.gradle"
27  apply plugin: 'com.google.gms.google-services'
28
```

Figur 19. Andra build.gradle apply plugin-ändringar

Om man nu kan köra ens applikation utan fel borde Firebase vara färdigt upplagd för Android.

Efter att jag hade lagt upp Firebase skapade jag en autentisering på mitt Firebase Projekt och skapade en användare så att jag kunde börja implementera inloggning i koden (Figur 20).



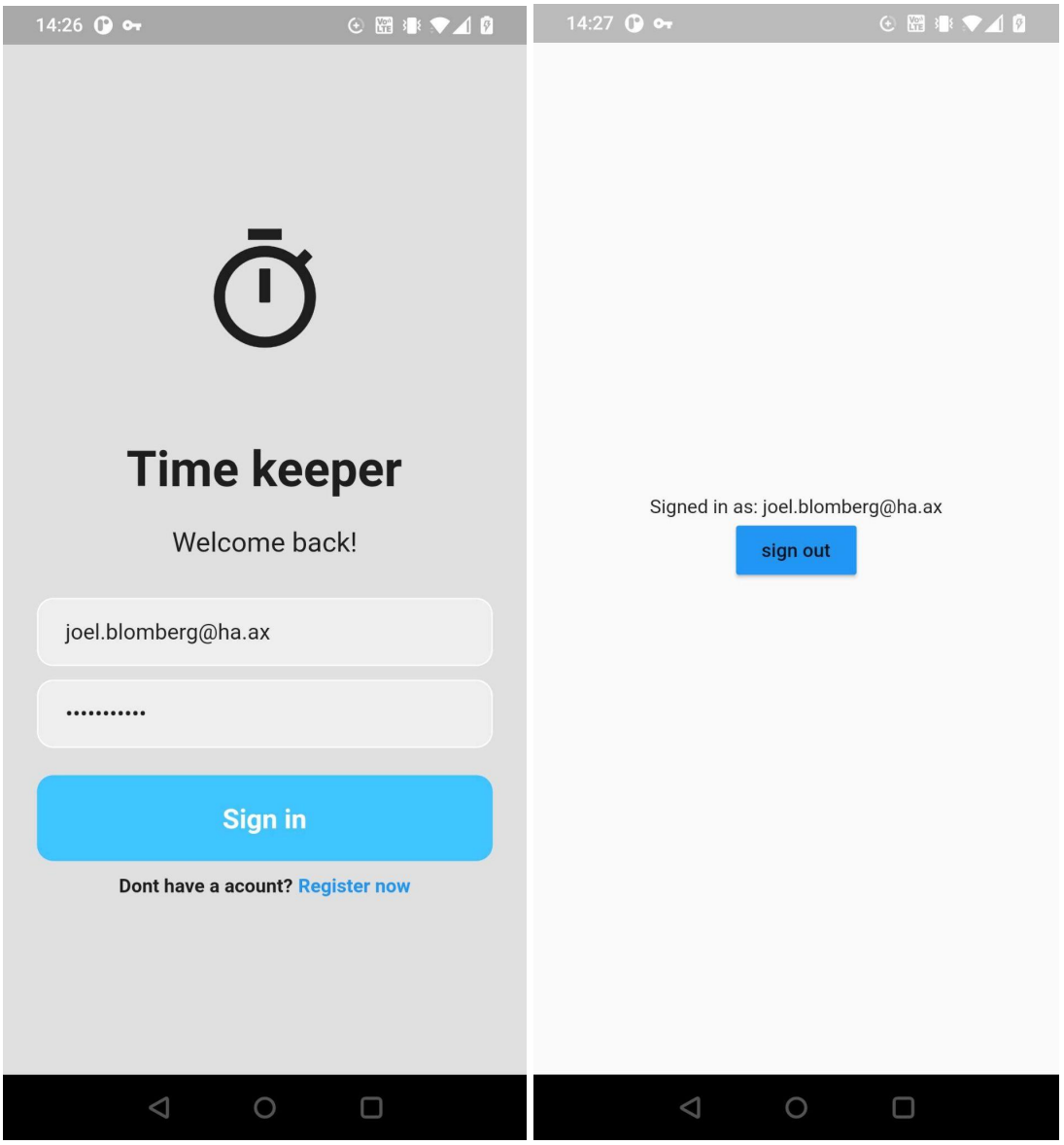
Figur 20. Firebase hemsida efter att man har skapat autentisering samt en användare

Efter att jag jobbat vidare på "login_page.dart" skapade jag en "signIn()"-funktion som kallades på när man klickade på login knappen samt två text kontroller som hanterade infon från de två textfälten för email och lösenord (Figur 21).

```
10  class _LoginPageState extends State<LoginPage> {
11
12    final _usernameEmailController = TextEditingController();
13    final _passwordController = TextEditingController();
14
15    Future signIn() async{
16      await FirebaseAuth.instance.signInWithEmailAndPassword(
17          email: _usernameEmailController.text.trim(),
18          password: _passwordController.text.trim(),
19      );
20    }
```

Figur 21. login_page.dart signIn()-funktion

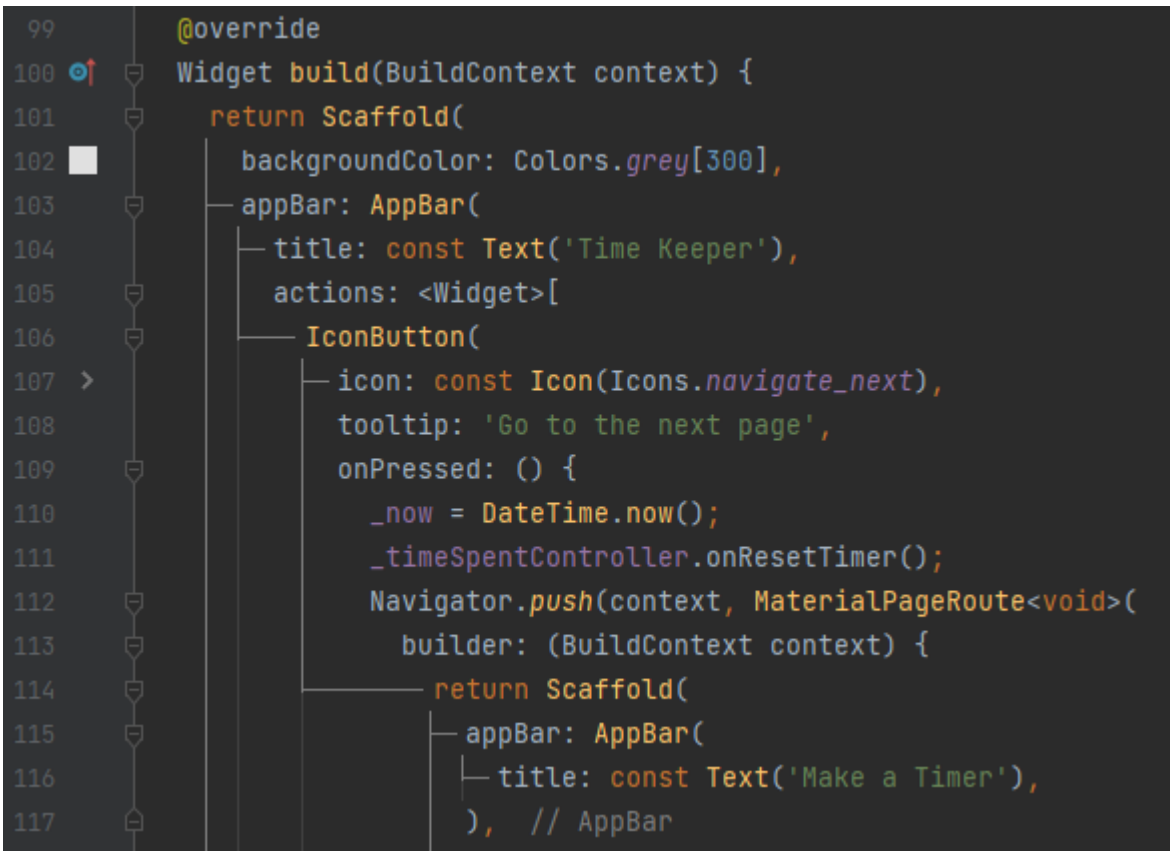
När jag även la till en kort bas för "home_page.dart" såg applikationen mer ut så här (Figur 22 och Figur 23).



Figur 22. Logga in med användaren på första sidan av applikationen

Figur 23. Applikationens utseende efter man har loggat in

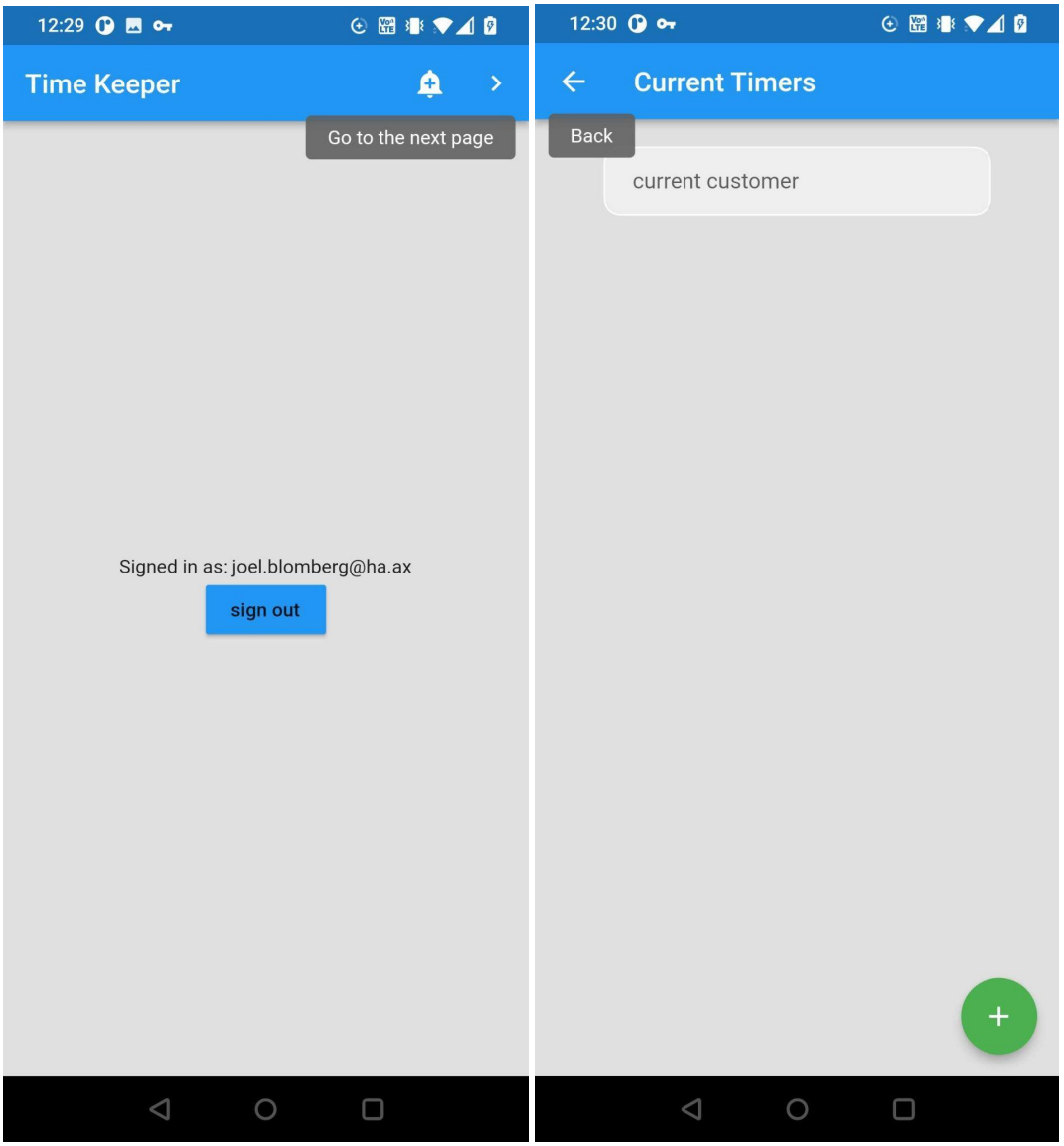
Nästa steg var att skapa själva databasen för applikationer och användarens UI för att kunna kicka upp sina tidsredovisningar. Jag bestämde att jag ville att användarens UI skulle ha en specifik sida för att skapa användarens tidsredovisningar så inuti “home_page.dart” bygg funktions “return Scaffold()” “AppBar()” sate jag in en “IconButton()” som returnerade en ny “return Scaffold()”. “Scaffold” är en användargränssnitt komponent som implementerar en visuell layout struktur. En “Scaffold” innehåller nästan allt man behöver för att skapa en funktionell samt responsiv applikation (Javatpoint,2022) (Figur 24).



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.grey[300],
    appBar: AppBar(
      title: const Text('Time Keeper'),
      actions: <Widget>[
        IconButton(
          icon: const Icon(Icons.navigate_next),
          tooltip: 'Go to the next page',
          onPressed: () {
            _now = DateTime.now();
            _timeSpentController.onResetTimer();
            Navigator.push(context, MaterialPageRoute<void>(
              builder: (BuildContext context) {
                return Scaffold(
                  appBar: AppBar(
                    title: const Text('Make a Timer'),
                  ),  // AppBar
```

Figur 24. Home_page.dart build Scaffold.

Jag skulle kunnat ha lagt in detta i dess egna sida men jag ville testa hur man kunde få det att fungera. Det ledde så klart till att “home_page.dart” är mycket större än de andra filerna och om jag skulle göra detta igen skulle jag separerat dem. Dock så fungerade den här lösningen väldigt bra och är ett snabbt sätt att få fram nya sidor på en applikation om man inte behöver ha särskilt komplicerade bygg-funktioner. Detta ledde till att min applikation nu såg ut så här efter att man har loggat in (Figur 25 och Figur 26).



Figur 25. Tillagd appBar för applikationen

Figur 26. Sidan som appBar knappen leder till

"IconButton()" låter en ge ett kort meddelande när man håller ner knappen så att man kan ge ett kort exempel på vad knappen gör. Jag infogade ett textfält att börja med samt en "FloatingActionButton()" nere i det högra hörnet som jag senare använde som post knappen för det användaren har skrivit. Men man vill så klart spara sin data i en databas på Firebase och jag valde att använda mig utav "Firestore Database". Firebase har många olika alternativ för databaser och jag är säker på att någon annan utav deras alternativ skulle ha kunnat utföra samma syfte. Men jag valde "Firestore Database" för att jag lätt kunde förstå hur det skulle Implementeras.

När man går in i sitt Firebase Projekt på Firebases hemsida såsom man gjorde tidigare med sin Autentisering kan man under “build” hitta “Firestore Database”. När man har hittat det och tryckt på “Create database” kommer man få upp en guidad prompt som tidigare. Jag rekommenderar att man börjar skapa databasen i “test mode” till att börja med. Efter det väljs en plats för ens Cloud Firestore som är nära ens hemland. Och eftersom man har satt allting i ordning tidigare så borde man nu ha möjlighet att skapa en “collection” för sin data genom att trycka på “Start collection”. När man har gjort detta så behöver man ge sin samling en “Collection ID” som kort sagt kommer vara namnet man använder när man pratar med det. Efter det när man går vidare så kan man enkelt auto generera en “Document ID” för sitt första dokument i databasens samling och fylla på den information man vill att den första posten ska innehålla. Man kan bestämma namnet på fältet samt vad för dataform det är sparat som och slutligen vad det innehåller. Notera att denna post bör innehålla någonting bara så att man kan se att det fungerar men innehållet vid det här läget borde inte påverka innehållen för framtida posts till databasen. För att kunna skicka upp något till den här databasen så behöver man först se till att man har en version av “firebase_core:” i sin “pubspec.yaml” fils beroenden som man såg tidigare i texten (Figur 14) .

Vid det här läget så fortsatte jag med att lägga till flera textfält för “home_page.dart” och några text controllers som skulle hantera dem samt en knapp som kallade på funktionen som slutligen skickade iväg informationen till databasen (Figur 27 och Figur 28).

```
13  class _HomePageState extends State<HomePage> {
14
15    final user = FirebaseAuth.instance.currentUser!;
16    final _customerController = TextEditingController();
17    final _taskController = TextEditingController();
18    final _descriptionController = TextEditingController();
```

Figur 27. Home_page.dart controller

```dart
Future sendInfo() async{
  if(isNotEmptyCheck()) {
    await FirebaseFirestore.instance.collection('time posts').add({
      'Employee': user.email,
      'Customer': _customerController.text.trim(),
      'Task': _taskController.text.trim(),
      'Description': _descriptionController.text.trim(),
      'Time Start': _now.day.toString() + '/' + _now.month.toString() + '/' +
          _now.year.toString() + ' - ' + _now.hour.toString() + ':' +
          _now.minute.toString(),
      'Time Spent': StopWatchTimer.getDisplayTime(
          _timeSpentController.rawTime.value, hours: _isHours),
    });
  }
  _now = DateTime.now();
  _customerController.clear();
  _taskController.clear();
  _descriptionController.clear();

  _timeSpentController.onResetTimer();
}
```

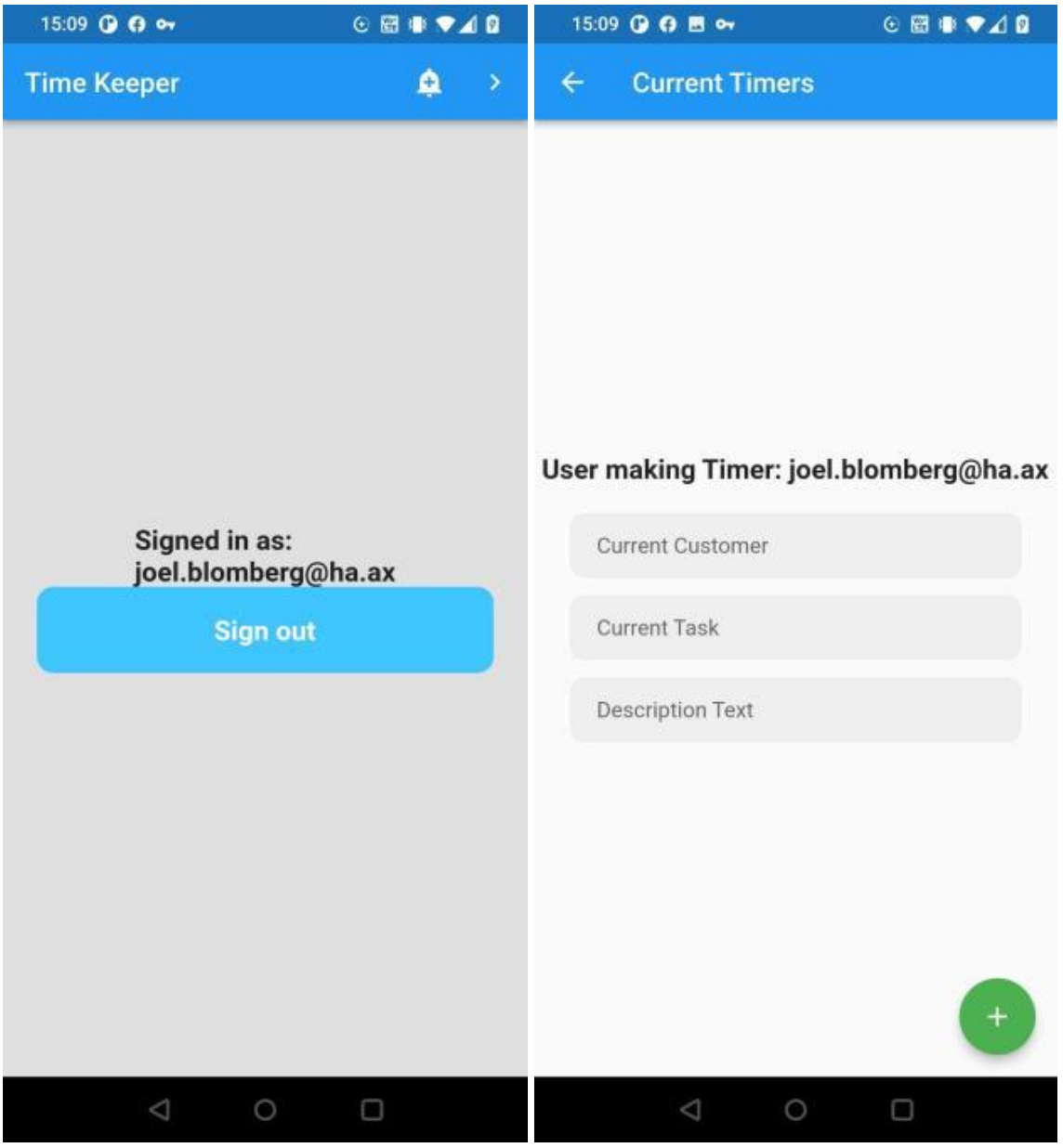
Figur 28. Home_page.dart sendInfo()-funktion

När jag först skapade denna funktion och började testa att den fungerade hade jag bara platshållartext för “Time Start” och “Time spent” eftersom jag ännu inte hade en lösning för den datan ännu. Jag skapade också en funktion som returnerar ett booleskt värde för att se till att vi inte skickade tomma dokument till databasen (Figur 29).

```dart
bool isNotEmptyCheck(){
  if(_customerController.text.isEmpty)
  {
      return false;
  }
  else if(_taskController.text.isEmpty)
  {
      return false;
  }
  else if(_descriptionController.text.isEmpty)
  {
      return false;
  }
  else{
    return true;
  }
}
```

Figur 29. Home_page.dart isNotEmptyCheck() funktion

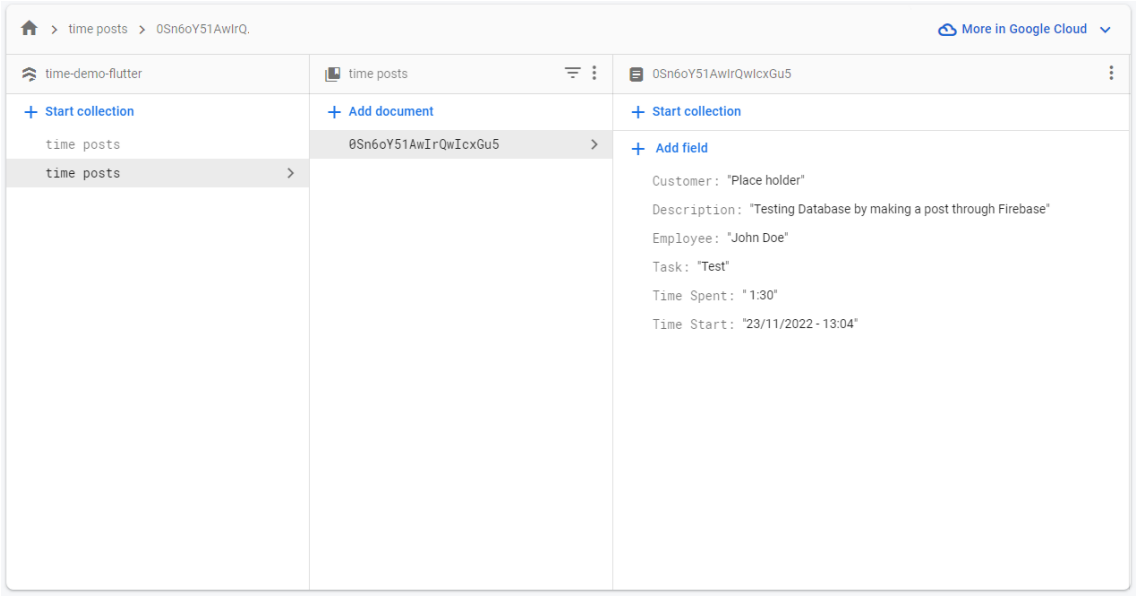
Men vid det här läget så såg “home_page.dart” på min applikation ut så här (Figur 30 och Figur 31).



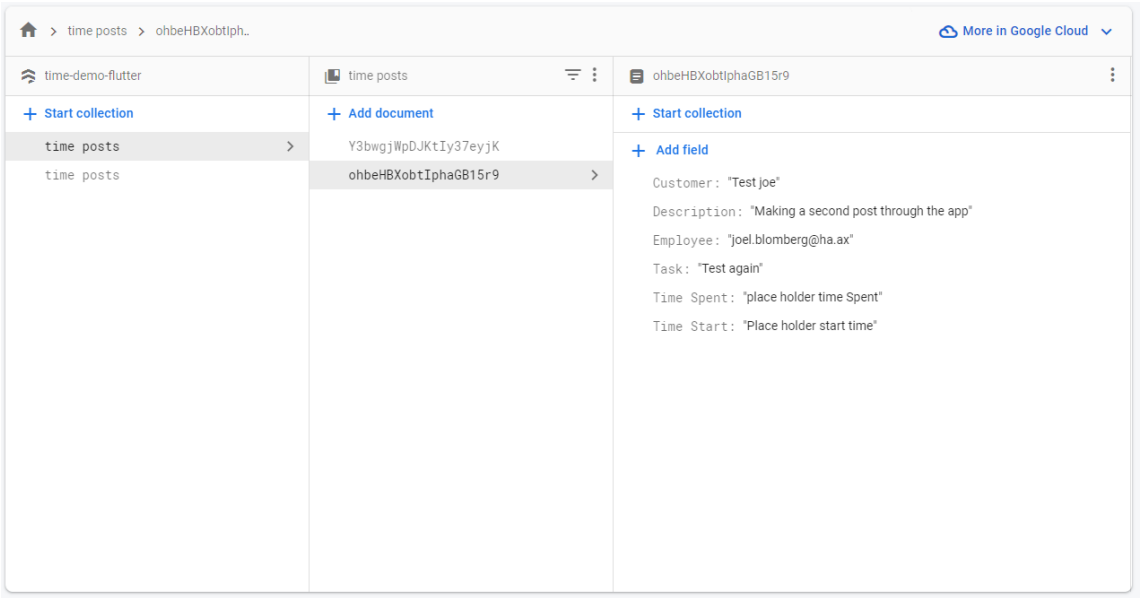
Figur 30. Home_page.dart sida 1 efter ändringar

Figur 31. Home_page.dart sida 2 efter ändringar

När jag sedan testade att min “sendInfo()”funktion fungerade kom jag fram till något som jag inte var beredd på och som jag inte riktigt kan förklara (Figur 32 och Figur 33).



Figur 32. Databaspost gjord från Firebase



Figur 33. Databaspost gjord från Applikation

Firebase satte mina posts som jag gjorde via min applikation i en separat samling från den som jag gjorde via hemsidan. Först var jag orolig att alla mina post skulle bli sparade på detta sätt men det verkade enbart som att Firebase ville hålla dom som jag hade skapat med min Applikation i sin egna samling istället för den jag skapade tidigare. Detta ledde inte till nåt större problem men det var bara en intressant observation eftersom jag hade förväntat mig att dom skulle samlas på samma plats först.

Det enda som var kvar nu var att se till att tidtagningen fungerade och för detta så vände jag mig ännu en gång till min “pubspec.yaml” där jag la till “stop_watch_timer:” för att underlätta tidtagaruret för “Time Spent:”(Figur 34).

```dart
final StopWatchTimer _timeSpentController = StopWatchTimer(
  mode: StopWatchMode.countUp,
  onChange: (value) => print('onChange $value'),
  onChangeRawSecond: (value) => print('onChangeRawSecond $value'),
  onChangeRawMinute: (value) => print('onChangeRawMinute $value'),
  onStopped: () {
    print('onStop');
  },
  onEnded: () {
    print('onEnded');
  },
);
```

Figur 34. StopWatchTimer Skapande

Jag hittade exempel på hur man smidigt kunde sätta upp en timer med hjälp av denna beroende och använde endas de delar som skulle behövas i mitt behov (Dart Packages,2022) (Figur 35).

```dart
@override
void initState() {
  super.initState();
  _timeSpentController.rawTime.listen((value) =>
      print('rawTime $value ${StopWatchTimer.getDisplayTime(value)}'));
  _timeSpentController.minuteTime.listen((value) => print('minuteTime $value'));
  _timeSpentController.secondTime.listen((value) => print('secondTime $value'));
  _timeSpentController.records.listen((value) => print('records $value'));
  _timeSpentController.fetchStopped
      .listen((value) => print('stopped from stream'));
  _timeSpentController.fetchEnded.listen((value) => print('ended from stream'));
}

```

Figur 35. StopWatchTimer status initialisering

Jag la till en ström där man kunde direkt se timerns tid gå upp samt la till en par knappar för att starta, pausa och nollställa timern (Figur 36).

```dart
StreamBuilder<int>(
  stream: _timeSpentController.rawTime,
  initialData: _timeSpentController.rawTime.value,
  builder: (context, snap) {
    final value = snap.data!;
    final displayTime =
    StopWatchTimer.getDisplayTime(value, hours: _isHours);
    return Column(
      children: <Widget>[
        Padding(
          padding: const EdgeInsets.all(8),
          child: Text(
            displayTime,
            style: const TextStyle(
                fontSize: 40,
                fontFamily: 'Helvetica',
                fontWeight: FontWeight.bold),  // TextStyle
          ),  // Text
        ),  // Padding
      ],  // <Widget>[]
    );  // Column
  },
),  // StreamBuilder
```

Figur 36. Timer stream upplägg

För "Time Start" så använde jag mig av en variabel som uppdaterades med "DateTime.now()" varje gång man öppnade andra sidan av "home_page.dart". Jag försökte få det att uppdatera när man resettade timern och skickade iväg sin post men jag lyckades inte få det att fungera av någon anledning. Och efter lite finslipning på Användarens UI samt lite små fixar så var min applikation i stort sett färdig.

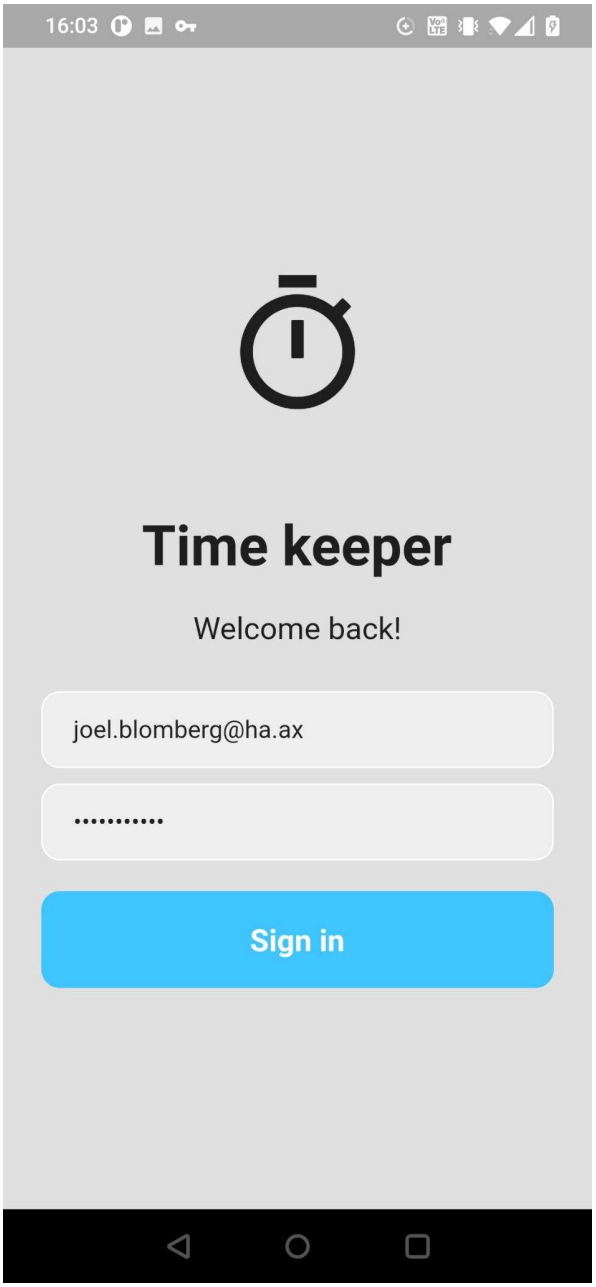
4.4.3 Problem och lösningar

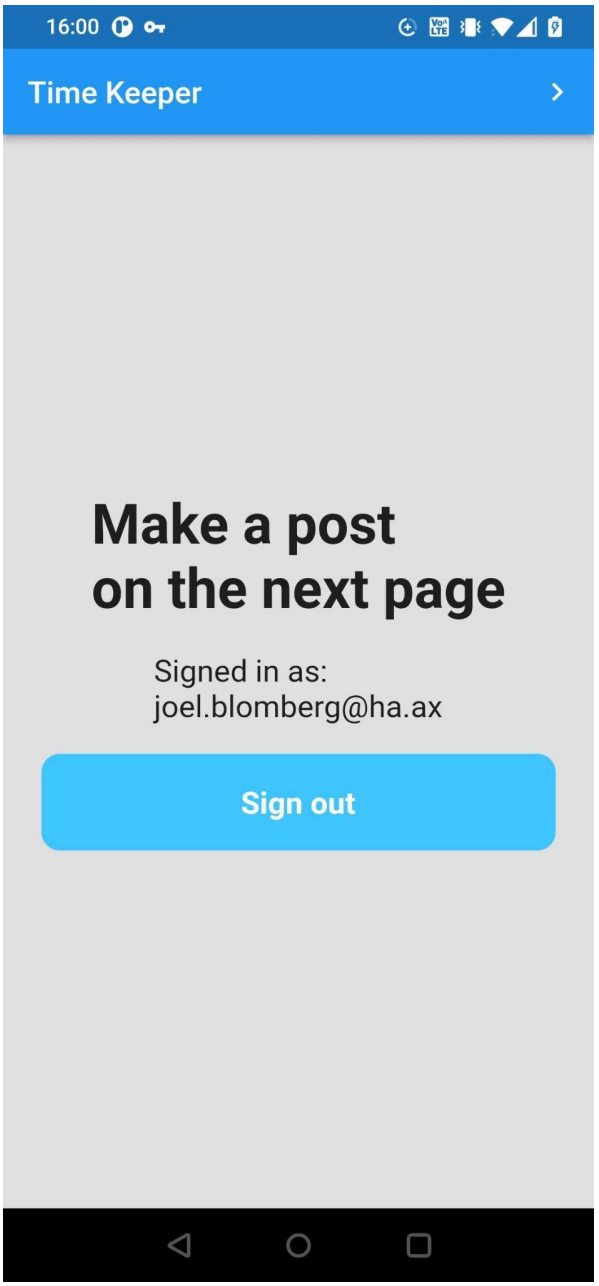
Jag hade lite problem med Installationen av allt när jag först började men det var mest på grund av missförstånd så mitt bästa tips för det är att läsa igenom instruktionerna noggrant och följa det så ordentligt man bara kan. Utöver detta så har Flutter och Dart varit väldigt användarvänligt och det mesta man möjligen fastna på kan lösas genom att hitta rätt widget eller beroende för jobbet. "Time Start" så försökte jag först se till att det förnyades varje gång man trycker på återställ knappen för timern samt varje gång man skickade iväg en post men det ville inte fungera och jag hade tyvärr inte tillräckligt med tid att hitta en vacker lösning på det. Jag testade mig fram till en simpel lösning för när man postar sin tidsredovisning men det involverade att man blev tvingad tillbaka till "home_pages.dart" första sida så att man måste

öppna upp sidan igen varje gång man skulle göra en post och jag kände att det bara kändes krångligt att använda så jag lät “Time Start” bestämmas bara när sidan öppnas istället för att hålla det simpelt.

4.4.4 Slutligt resultat

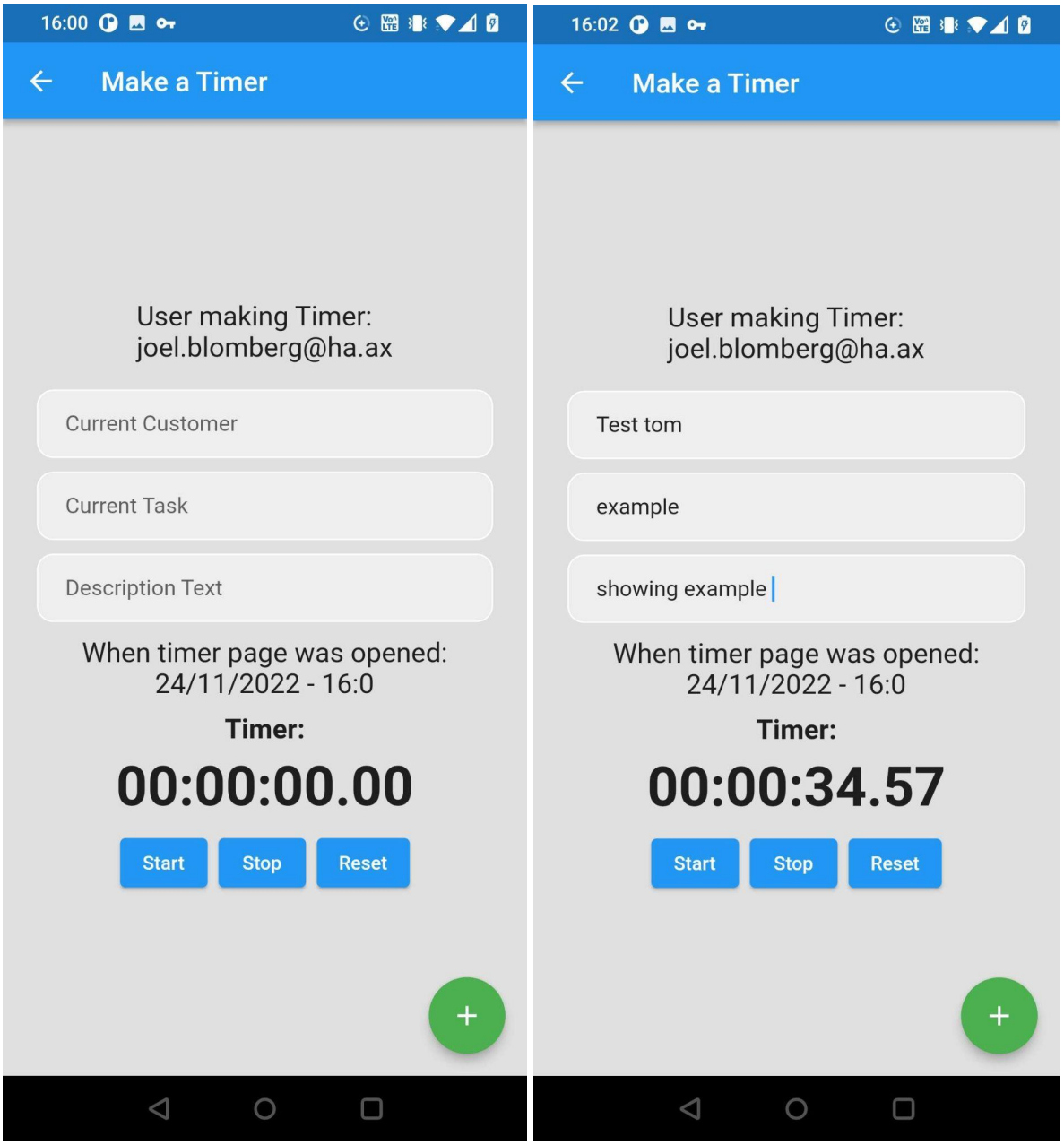
Som man kan se i figurerna nedan så har jag nu ett grundläggande tidtagarur samt några textfält där man fyller i kundens namn samt information kring arbetet som man utför. Som man redan kan se tidigare är det här inte den lösning jag hade velat ha men vid det här läget så var jag redan mot slutet av projektet och skulle inte haft ordentligt med tid att lösa det före jag behövde börja skriva om mitt arbete. Lösning för detta skulle vara en ny sida på applikationer där man kan registrera och spara kund information som sparas i en separat samling där man sedan kan hämta informationen man behöver för sin post. Men sen när man trycker på den gröna knappen om alla fält är ifyllda skickas det iväg informationen till databasen i samband med tiden och datumet som man öppnade “Make a timer”sidan, gällande vad timern var på när man skickar iväg det samt vem som är inloggad genom användarens email address. Efter att man har skickat iväg det töms alla textfält som man kan editera och återställer timern. Och all information sparas just nu för att hålla det enkelt som strings men det kan lätt justeras för den information som har behov av det (Figur 36, 37, 38, 39 & 40).





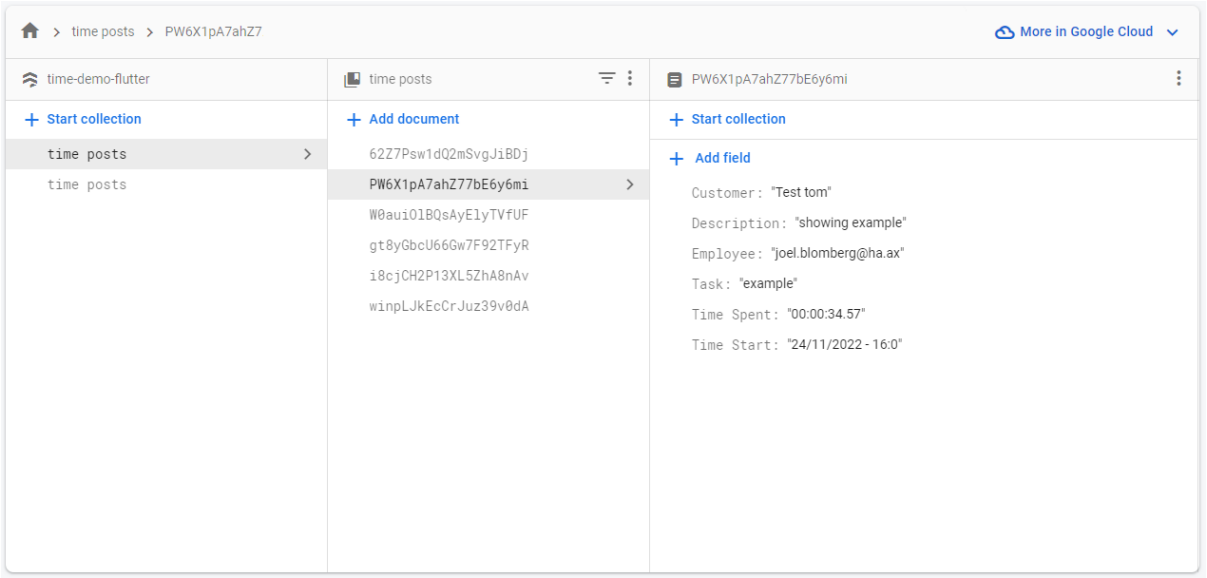
Figur 36. Slutliga Applikationens inloggningssida

Figur 37. Slutliga Applikationens grundsida



Figur 38. Slutliga Applikationens timersida

Figur 39. Slutliga Applikationens timersida med timer-exempel



Figur 40. Resultat på Firebase hemsida efter sista exempel post

5. SLUTSATSER

Min slutsats från detta arbete är att Flutter är ett väldigt användbart ramverk för att skapa applikationer även om man är ny till att använda det. Före jag började på det här projektet hade jag aldrig skrivit något med Dart men eftersom det involverar många likheter från JavaScript och Java är det väldigt lätt att plocka upp och använda så länge man är någorlunda bekväm med dem från tidigare. I kombination med detta ger Firebase ett väldigt smidigt alternativ för Backend funktioner och databaser. Om jag skulle ha mera tid att arbeta på min applikation eller göra om den så finns det så klart mycket som jag skulle ändra på såsom att spara kundinformation i en enskild databas istället för att behöva skriva in deras information varje gång man gör en post samt möjligheten att skapa nya användare och kunder från applikationen. Som sagt tidigare så började jag detta projekt med perspektivet att det borde finnas en motsvarande webbapplikation som fyller dessa roller i systemet samt läser upp och noggrannare sorterar de posts som görs men med den tiden jag har haft och med tanke på att jag har behövt lära mig mycket genom detta arbetet så hade jag aldrig tillräckligt med tid eller kunskap för att skapa en ytterligare applikation. Jag har fått många lärdomar från att arbeta med Flutter och Dart och jag själv kommer garanterat återvända till att jobba med dem i framtiden för att se vad jag ytterligare kan åstadkomma. Jag skulle rekommendera för de flesta som har tidigare kunskap att programmera att i alla fall testa på att skapa något med Flutter för att själv se hur de tycker det känns att arbeta inom det och själva kunna utforska den otroliga mängden widgets och paket som Dart och Flutter har att erbjuda. Men jag skulle även rekommendera att ta en titt på Firebase för deras smidiga lösningar för moln databaser. Mot slutet av detta projekt känner jag mig mer säker och kapabel som en programmerare och trots att det alltid finns något som jag kan förbättra med min applikation så känner jag mig nöjd med det jag har åstadkommit och erfarenheterna som jag har plockat upp under projektet. Jag hoppas att ni som läsare har lärt er något och fått ett intresse i att programmera med Flutter.

KÄLLFÖRTECKNING

Blair, Ian. 2017. “How to Start an App – 8 Steps You Must Take before Starting Mobile App Development.” BuildFire (blog). July 31, 2017. https://buildfire.com/steps-before-starting-mobile-app-development/.

Bradford, Samuel Clement. 1948. Documentation. Crosby Lockwood& Son Ltd, London.

Brain of Plastic. 2022. “Android Studio in 100 Seconds.” Youtube. August 10, 2022. https://www.youtube.com/watch?v=0uxfXwXAx10.

“Build Apps for Any Screen.” n.d. Accessed December 1, 2022. https://flutter.dev/.

“Cookbook.” n.d. Accessed December 1, 2022. https://docs.flutter.dev/cookbook.

“Dart Language Evolution.” n.d. Accessed December 1, 2022. https://dart.dev/guides/language/evolution.

“Dart Packages.” n.d. Dart Packages. Accessed December 1, 2022. https://pub.dev/.

“Dart Programming Language.” n.d. Accessed December 1, 2022. https://dart.dev/.

“Dart vs JavaScript: Detailed Comparison.” 2021. Codemagic Blog. January 5, 2021. https://blog.codemagic.io/dart-vs-javascript/.

“DateTime Class - Dart:core Library - Dart API.” n.d. Accessed December 1, 2022. https://api.flutter.dev/flutter/dart-core/DateTime-class.html.

“Firebase Console.” n.d. Accessed December 1, 2022. https://console.firebase.google.com/u/1/project/time-demo-flutter/firestore/data/~2Ftime%20posts~2F62Z7Psw1dQ2mSvgJiBDj.

“Firebase_core.” n.d. Dart Packages. Accessed December 1, 2022. https://pub.dev/packages/firebase_core.

Fireship. 2020. “Flutter in 100 Seconds.” Youtube. April 14, 2020. https://www.youtube.com/watch?v=lHhRhPV--G0.

———. 2021. “Dart in 100 Seconds.” Youtube. October 13, 2021. https://www.youtube.com/watch?v=NrO0CJCbYLA.

———. 2022. “Firebase in 100 Seconds.” Youtube. February 17, 2022. https://www.youtube.com/watch?v=vAoB4VbhRzM.

“Flutter Documentation.” n.d. Accessed December 1, 2022. https://docs.flutter.dev/.

“Flutter Samples.” n.d. Accessed December 1, 2022. https://flutter.github.io/samples/.

“Flutter Scaffold.” n.d. Www.javatpoint.com. Accessed December 13, 2022. https://www.javatpoint.com/flutter-scaffold.

Hindi, Daniel. 2022. “Getting Started: Coding an App.” BuildFire (blog). February 1, 2022. https://buildfire.com/learn-to-code-mobile-app-fast/.

Kudlacek, Peter. 2018. “5 Things to Keep in Mind When Developing a Mobile App.” APRO Software. March 14, 2018. https://apro-software.com/5-things-that-you-should-keep-on-mind-when-you-develop-mobile-app/.

“Language Samples.” n.d. Accessed December 1, 2022. https://dart.dev/samples.

Limbüchler, Bythomas. n.d. “Top in-Demand Programming Languages to Learn in 2021.” Accessed December 7, 2022. https://www.wearedevelopers.com/magazine/top-programming-languages-to-learn.

Patel, Parth. n.d. “Introduction To Google Firebase.” Accessed December 1, 2022. https://www.c-sharpcorner.com/article/introduction-to-google-firebase/.

Sherrell, Linda. 2013. “Evolutionary Prototyping.” In Encyclopedia of Sciences and Religions, edited by Anne L. C. Runehov and Lluis Oviedo, 803–803. Dordrecht: Springer Netherlands.

“Stop_watch_timer.” n.d. Dart Packages. Accessed December 1, 2022. https://pub.dev/packages/stop_watch_timer.

“Stopwatch Class - Dart:core Library - Dart API.” n.d. Accessed December 1, 2022. https://api.flutter.dev/flutter/dart-core/Stopwatch-class.html.

The. 2021. “A Brief History of Flutter.” The Pragmatic Programmers. January 29, 2021. https://medium.com/pragmatic-programmers/a-brief-history-of-flutter-939645f93255.

The London Academy of IT Limited. n.d. “7 Fundamental Principles of Mobile App Development That Need to Follow.” The London Academy of IT Learning Blog. Accessed December 8, 2022. https://www.londonacademyofit.co.uk/blog/7-fundamental-principles-of-mobile-app-development-that-need-to-follow.

“Widget Catalog.” n.d. Accessed December 1, 2022. https://docs.flutter.dev/development/ui/widgets.

“Windows Install.” n.d. Accessed December 1, 2022. https://docs.flutter.dev/get-started/install/windows.

ÖVRIGA RESURSER

Bernardez, Fernand. 2020. “3.4. Evolutionary Process Model (Prototyping).” Youtube. September 7, 2020. https://www.youtube.com/watch?v=fVu8LbjS0-c.

“Clear TextField in Flutter.” 2020. GeeksforGeeks. September 20, 2020. https://www.geeksforgeeks.org/clear-textfield-in-flutter/.

Dane, Mike. 2021. “Dart Programming in 4 Hours | Full Beginners Tutorial.” Youtube. February 1, 2021. https://www.youtube.com/watch?v=5xlVP04905w.

Developers, Google. 2012. “Introducing Dart.” Youtube. October 16, 2012. https://www.youtube.com/watch?v=5KlnlCq2M5Q.

———. 2018. “Introducing Flutter.” Youtube. February 23, 2018. https://www.youtube.com/watch?v=fq4N0hgOWzU.

eTechViral. 2022. “Flutter Firebase Setup | iOS, Android & Web.” Youtube. April 15, 2022. https://www.youtube.com/watch?v=PbUTQTMBFVI.

Firebase. 2016. “Introducing Firebase.” Youtube. May 18, 2016. https://www.youtube.com/watch?v=O17OWyx08Cg.

———. 2021. “Codelab: Get to Know Firebase for Flutter.” Youtube. May 12, 2021. https://www.youtube.com/watch?v=wUSkeTaBonA.

Flutter. 2020. “Welcome to Flutter.” Youtube. July 23, 2020. https://www.youtube.com/watch?v=4AoFA19gbLo.

———. 2021a. “How Do I Make My First Flutter App.” Youtube. March 24, 2021. https://www.youtube.com/watch?v=xWV71C2kp38.

———. 2021b. “What Is State?” Youtube. April 14, 2021. https://www.youtube.com/watch?v=QlwiL_yLh6E.

———. n.d. “Begin Learning Flutter.” Youtube. Accessed December 1, 2022a. http://www.youtube.com/playlist?list=PLjxrf2q8roU3wk7CDw4RfV3mEwOJbjx1k.

———. n.d. “Dart.” Youtube. Accessed December 1, 2022b. http://www.youtube.com/playlist?list=PLjxrf2q8roU0Net_g1NT5_vOO3s_FR02J.

———. n.d. Youtube. Accessed December 1, 2022c. https://www.youtube.com/channel/UCwXdFgeE9KYzlDdR7TG9cMw.

———. n.d. “Flutter Package of the Week.” Youtube. Accessed December 1, 2022d. http://www.youtube.com/playlist?list=PLjxrf2q8roU1quF6ny8oFHJ2gBdrYN_AK.

———. n.d. “Flutter Widget of the Week.” Youtube. Accessed December 1, 2022e. http://www.youtube.com/playlist?list=PLjxrf2q8roU23XGwz3Km7sQZFTdB996iG.

———. n.d. “Further Your Flutter Skills!” Youtube. Accessed December 1, 2022f. http://www.youtube.com/playlist?list=PLjxrf2q8roU1Vm2a9EySYgeMo2DIhXUeK.

———. n.d. “Learning to Fly.” Youtube. Accessed December 1, 2022g. http://www.youtube.com/playlist?list=PLjxrf2q8roU3X18pAQWLyCJaa79RpqWnn.

———. n.d. “Making Animations in Flutter.” Youtube. Accessed December 1, 2022h. http://www.youtube.com/playlist?list=PLjxrf2q8roU2v6UqYlt_KPaXlnjbYySua.

freeCodeCamp.org. 2019. “Dart Programming Tutorial - Full Course.” Youtube. July 24, 2019. https://www.youtube.com/watch?v=Ej_Pcr4uC2Q.

“Google Codelabs.” n.d. Google Codelabs. Accessed December 1, 2022. https://codelabs.developers.google.com/?client=avast-a-1.

Koko, Mitch. 2022a. “ 📱 Minimal Login UI • Flutter Tutorial ♡.” Youtube. April 24, 2022. https://www.youtube.com/watch?v=aJdIkRipgSk.

———. 2022b. “ 📱 Login & Logout • Firebase X Flutter Tutorial ♡.” Youtube. April 27, 2022. https://www.youtube.com/watch?v=TkuO8OLgvkk.

———. 2022c. “ 📱 Sign Up Users • Firebase X Flutter Tutorial ♡.” Youtube. April 29, 2022. https://www.youtube.com/watch?v=Mfa3u3naQew.

———. 2022d. “ 📱 Reset Password • Firebase X Flutter Tutorial ♡.” Youtube. May 1, 2022. https://www.youtube.com/watch?v=Sp4_2zi0kZg.

———. 2022e. “(CRUD) 📱 Create & Store User Data • Firebase X Flutter Tutorial ♡.” Youtube. May 2, 2022. https://www.youtube.com/watch?v=idJDAdn_jKk.

———. 2022f. “Flutter Snake Game 📱 HIGHSCORES + HOSTING Using Firebase ♡.” Youtube. June 7, 2022. https://www.youtube.com/watch?v=9jvJyLhJP00.

Milke, Johannes. 2020. “Flutter Tutorial - Log Everything - Smart Logging.” Youtube. September 20, 2020. https://www.youtube.com/watch?v=GUi0n9c33os.

Udacity. 2015. “Evolutionary Prototyping Process - Georgia Tech - Software Development Process.” Youtube. February 23, 2015. https://www.youtube.com/watch?v=bAEnaGG8Otc.

“Write Your First Flutter App, Part 1.” n.d. Google Codelabs. Accessed December 1, 2022. https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1?hl=en.

“Write Your First Flutter App, Part 2.” n.d. Google Codelabs. Accessed December 1, 2022. https://codelabs.developers.google.com/codelabs/first-flutter-app-pt2?hl=en.