BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION Computer Science in English

DIPLOMA THESIS

Cross-Platform Social Media Mobile Application For Traveling

Supervisor
Lect. Ph.D Radu D. Găceanu

Author
Luca Denisa

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA Informatică Engleză

LUCRARE DE LICENŢĂ

Aplicație Mobilă Multi-Platformă De Socializare Axată Pe Traveling

Conducător științific
Lect. Ph.D Radu D. Găceanu

Absolvent
Luca Denisa

2021

ABSTRACT

This thesis introduces a cross-platform social-media application focused on traveling developed in Flutter. While there are many apps out there dedicated exclusively to traveling, the main idea is to put an emphasis on the social and cultural aspects of it. This application is meant to offer users the possibility to explore new places through the eyes of a local.

Flutter allows developers to construct apps that work on Android, iOS, and, more recently, web apps. As a result, compared to a native app, cross-platform software will reach a substantially larger audience. The application does not require any additional servers to be run, since everything happens on the Firebase servers. Firebase offers real-time data synchronization, easy authentication, push notifications, crashlytics, and many more. All the information is securely stored, backed up, and can be easily be retrieved.

The first part of the thesis focuses on describing the technologies used for developing the application, together with some reasons why those technologies were chosen. The second chapter presents a brief history of cross-platform development, starting with the web app and switching to cross-platform development in recent times. Then Flutter Framework is introduced in the topic, together with some pros and cons. The third chapter gives an overview of Google's Firebase, describing some of its core functionalities. Details about how data is handled are given.

The last two chapters focus on the practical part and future prospects. Details about the implementation, functionalities, and the used packages are provided. Finally, we contemplate on the achievements and propose future goals.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Luca Denisa

[illegible]

Contents

List of Figures

List of Tables

Introduction

Humans are social creatures. Community is what makes people so special. We feel the need to socialize and meet new persons every once in a while. It is crucial for us to form real connections with the people around us.

Now that this global pandemic seems to finally retreat, people want to recover the lost time and take advantage of this newly-found sense of freedom. One of the best ways to get to know people and cultures is through traveling. The main purpose of this thesis is to offer a platform for travelers to connect with people around the globe in order to not only have an outstanding travelling experience, but also a social and cultural one.

The app is designed for people that feel the need for a little spontaneity in their lives. While there are many apps out there dedicated exclusively to traveling, my idea is to put an emphasis on the social aspect of it, while still keeping everything low-cost.

Users will be able to chat with other persons(guides) from different locations and eventually meet. In this way, travelers will be able to see the city through the eyes of a local.

The thesis is structured in 4 main chapters: Cross-platform Development, Firebase, Application Structure, and Future Work and Possible Improvements.

Chapter 1, Cross-platform Development, presents a short history of the cross-platform applications and the current context. In the second part, the main focus is the Flutter Framework and explaining why I chose to develop my application using it.

Chapter 2, Firebase, starts by giving a short introduction of the subject and explain how it is used. A short description of the functionalities used is given. After that, the focus switches on how data is stored and managed, providing a comparison between the Realtime Database and the Cloud Firestore. Some difficulties encountered while choosing a database are included.

Chapter 3: Application Structure focuses on what the application offers in terms of functionalities and user experience. The database and the relations between entities are briefly presented. Towards the end of the chapter, details about state management and data-processing over the network are given.

The last chapter, Future Work and Possible Improvements, reiterates the main purpose of the app while providing insights about the achieved goals and proposing ideas for further development.

Chapter 1

Cross-platform development

The main focus of this chapter is to define cross-platform application development, why such frameworks are used, and explore some alternatives.

Cross-platform software, also referred to as multi-platform or platform-independent software, is a type of software that can run on multiple platforms, thus offering the possibility of being exposed to a substantially larger set of users compared to applications that only run natively [4].

1.1 Brief History Of Cross-Platform Frameworks For Mobile Development

1.1.1 Web App

Unsurprisingly, one of the first cross-platform software to be used is the good old web application. A web application is a software that runs on a server and utilizes web browsers and web technology to perform tasks over the Internet [8].

The first WorldWideWeb [3] browser was developed in 1990 by Sir Timothy John Berners-Lee. Back then, static HTML pages ruled the Internet. With the advancements in the field of computer science, it soon became possible to add images, videos, and sounds to web pages.

One huge step for the Internet and web development was taken in 1995, with the release of client-side scripting language Javascript. The language allows interactivity/communication between the client and the webserver, thus making web pages faster and more intuitive. JavaScript is one of the three most notable technologies (with HTML and CSS) of content production for WWW [5]. Towards the end of the same decade, in 1999 to be more precise, the concept of the web application was introduced in Java.

With the introduction of Ajax in 2005, the whole world dynamic web pages. Ajax is a set of web development techniques that enable programmers to deliver asynchronous web applications.

In 2008, HTML5 saw the light for the first time and it was finalized in 2014. The main objectives of HTML5 were to eliminate client-side plugins such as Flash, reducing the need for extra Javascript code by introducing new elements, and providing consistency across browsers [17].

Progressive Web Apps were presented as the new standard of developing web applications during the annual Google IO conference in 2016.

1.1.2 Hybrid App

Hybrid applications work just like any other mobile application: they are installed on a mobile device and offer the user certain functionalities and solutions. What makes a hybrid app different is the fact that it is built using both native elements and web elements. They are basically web applications wrapped into a native container. Hybrid applications allow easier and smoother access to device system, such as camera or contacts, compared to their web-based counterparts. This is achieved by using third-party plugins.

It is forecast that by 2024, there will be 7.41 billion mobile phone users [15]. In this context, mobile app development is crucial for the success of a company. Whoever manages to deliver the best user experience and touches the majority of the users through a mobile application has a definite advantage.

Some of the most popular hybrid applications in 2021 are Facebook, Gmail, Instagram, Twitter. Table 1.1

App	Users(billions)
Facebook	2.80
Gmail	1.80
Instagram	1.07
Twitter	0.35

Table 1.1: Hybrid app users

1.1.3 Cross-platform Development Today

Cross-platform development frameworks allow the programmers to deliver applications that are compatible with two or more operating systems. Cross-platform software offers a good alternative to native or hybrid applications. This type of software manages to maintain a native experience all while making developers' lives easier.

In the previous subsection it was mentioned that developers are able to access the device's system only through third-party plugins. While cross-platform development also has its own limitations, using the device's features is generally easier while working cross-platform than within a hybrid app.

Some of the most popular frameworks for cross-platform development are:

- Flutter
- .NET Framework
- Ionic
- Xamarin

1.2 Why Flutter?

With all the new technology advancements in the mobile industry, it does seem like the future is a "portable" one. Smartphones now offer an all-in-one solution to all the needs in modern human's everyday life. In a matter of milliseconds, your tiny mobile phone can switch from being a communication device to a camera, a wallet, and many more. We rely on our phones now more than ever. In this context, it is crucial to develop fast, intuitive, cross-functional applications.

Flutter is a cross-platform framework released by Google that aims at developing high-performance mobile applications [20].

1.2.1 Cross-Functionality

One of the main reasons why I chose to develop my application in Flutter is cross-functionality. Flutter enables developers to create apps compatible with Android, iOS, and, more recently, web applications. As a result, your application will have a significantly wider audience reach in comparison to a native app.

1.2.2 Performance

The advantages of building apps in a native environment are clear. They are fast, stable, and don't depend on external libraries. However, Flutter brings all these advantages and even more. Performance-wise, a flutter application is almost as fast as a native app and is a definite winner in comparison to other non-native frameworks. Tables 1.3 and 1.2

[1]

Platform	Borwein(ms)	Gauss-Legendre(ms)
Native	144	223
Flutter	285	273
React Native	822	3289

Table 1.2: Speed of computations performed in Android in milliseconds

Platform	Borwein(ms)	Gauss-Legendre(ms)
Native	26	173
Flutter	180	189
React Native	582	2992

Table 1.3: Speed of computations performed in iOS in milliseconds

Chapter 2

Firebase

2.1 About Firebase

Firebase is Google's mobile application development platform that helps you build, improve, and grow your app.

Firebase is best suited for applications that use real-time chats, push notifications, and fast querying all while eliminating the need to store the data on your own servers. Thus the entities stored on the Firebase Cloud are secured, backed up, and safe. The users won't have to worry about sensitive data being forwarded without their consent. Figure 2.1 [1]

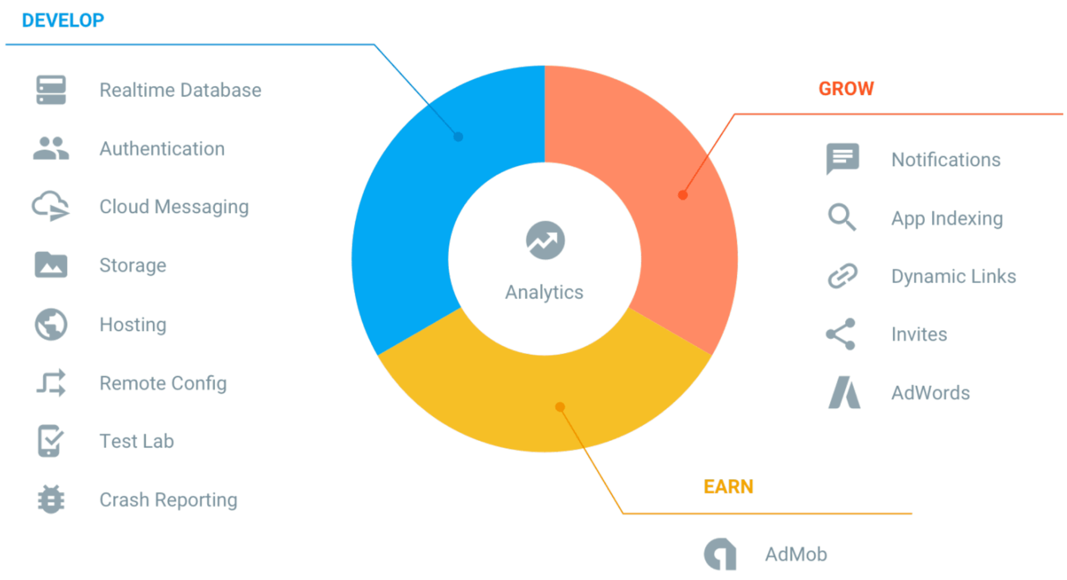


Figure 2.1: Firebase Features

[1] `https://superdevresources.com/why-use-firebase/`

2.1.1 Authentication

Firebase Authentication makes user management easier. It offers authentication services using multiple platforms, such as Google, Facebook, using a phone number, or the basic registration with email and password. Firebase Authentication is fast, reliable, and secure. Users' data backed-up and passwords are hashed using a secret key. [13]

```
  Future<UserCredential> signIn({String email, String password}) async{
    UserCredential user = await _firebaseAuth.signInWithEmailAndPassword(email:
        email, password: password);
    return user;
}

Future<UserCredential> signUp({String email, String password}) async{
    UserCredential user = await
        _firebaseAuth.createUserWithEmailAndPassword(email: email, password:
        password);
    return user;
}

Future<UserCredential> signInWithGoogle() async {
  final googleAccount = await googleSignIn.signIn();

  if (googleAccount == null) {
    return null;
  }

  else {
    final googleAuth = await googleAccount.authentication;
    final credentials = GoogleAuthProvider.credential(
      accessToken: googleAuth.accessToken,
      idToken: googleAuth.idToken
    );
    UserCredential user = await
        FirebaseAuth.instance.signInWithCredential(credentials);

    return user;
  }
 }
```

2.1.2 Cloud Storage

Cloud Storage enables quick and safe file transmission for Firebase apps, and it is independent of network quality. It is supported by Google Cloud Storage, a low-cost object storage service. It can be used to store photographs, audio, video, and other user-generated content by the developer [11].

```
Reference firebaseStorageRef =
    FirebaseStorage.instance.ref().child('/images').child(imageFile.path.split('/').last);
TaskSnapshot snapshot = await firebaseStorageRef.putFile(imageFile);
```

2.1.3 Analytics

Firebase Analytics provides insight into app usage. It is a paid app measurement solution that also provides user engagement. This unique feature enables the application developer to understand how users are using the application. The SDK has the feature of capturing events and properties on its own and also allows getting custom data [6].

Analytics helps you understand how your users behave, so you can make informed decisions about how to market your app. See the performance of your campaigns across organic and paid channels to understand which methods are most effective at driving high-value users. If you need to perform custom analysis or join your data with other sources you can link your Analytics data to BigQuery, which allows for more complex analysis like querying large data sets and joining multiple data sources [12].

2.1.4 Cloud Firestore

Cloud Firestore is a No-SQL database and it facilitates easy and secure file transfer [6]. In the context of a social media application, Cloud Firestore can be used for storing data related to specific users, such as user details or profile pictures and media files(pictures, videos, documents, etc.) linked to posts.

2.1.5 Realtime Database

The Firebase Realtime Database is a NoSQL cloud-based database [19] that syncs data across all clients in real-time, and provides offline functionality. Data is stored in the Realtime Database as JSON, and all connected clients share one instance, automatically receiving updates with the newest data [6].

2.2 Why Firebase?

When developing an app, one of the first steps and most important decisions is choosing which server to use, local or cloud, and where to store your data. The main advantage of local servers is that they are basically free of charge and could be well-suited for small applications, that don't deal with critical data. However, in the long run, a cloud server, in this case, Firebase, is a definite winner.

The main problem that Firebase solves is data loss due to system failures or catastrophic events. Data is backed up and stored safely in multiple regions, such that, if one of the regions fails, all the data can be recovered without affecting any clients. This cannot be said about a local server. The risk of data loss is much higher here.

But Firebase is more than a simple cloud server. It offers real-time data synchronization, easy authentication, push notifications, crashlytics, and many more. It eliminates the need for a back-end server, saving the developers a lot of time and most importantly, effort. There is no need to set up servers or write back-end code. Everything can be done with Firebase but in a faster and cheaper way. Figure 2.2 [2]

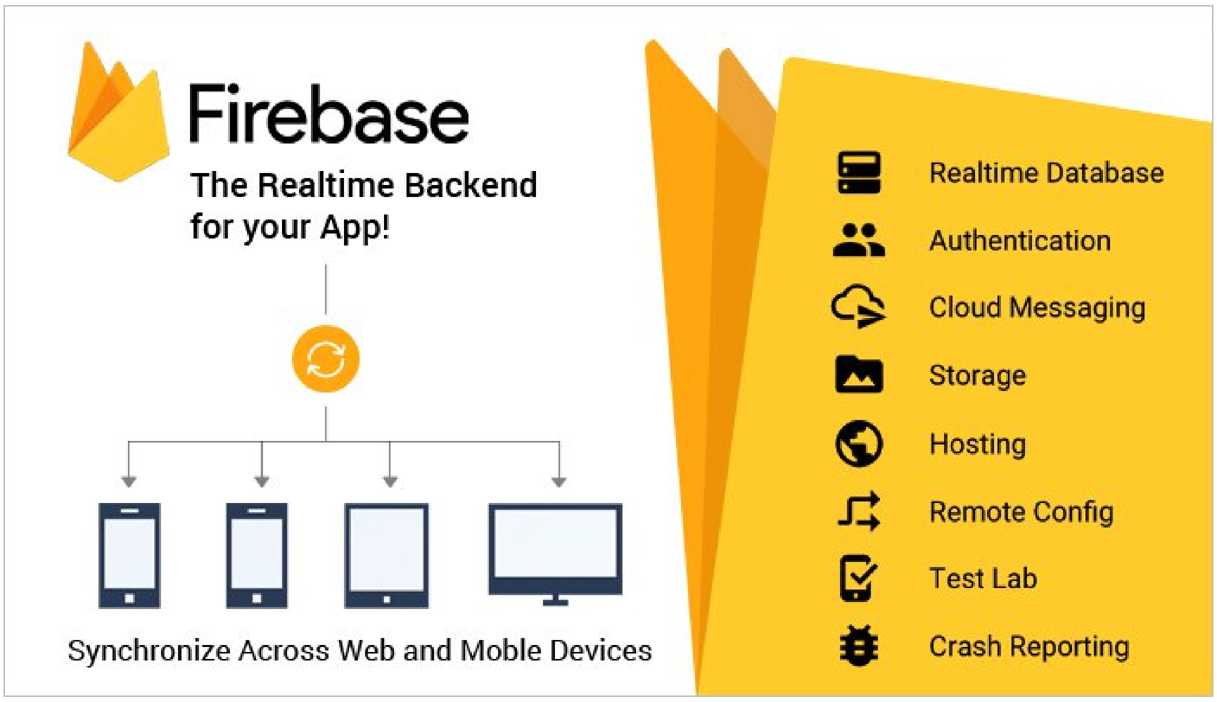


Figure 2.2: Firebase as a Backend

[2]https://www.tristatetechnology.com/blog/firebase-backend-mobile-app/

2.3 Realtime Database Versus Cloud Firestore

Both the Realtime Database and the Cloud Firestore available in Firebase are cloud-based and offer real-time access and updates. They have at their core a NoSql Database, thus providing a flexible and efficient way to store the data [14].

2.3.1 Structure

The data added to the Realtime Database is organized in a JSON [16] tree, while the data in the Cloud Firestore is structured in documents. Documents are very similar to JSON format, but the data is better organized, which makes querying easier and more efficient than compared to the Realtime Database.

2.3.2 Datasets

Cloud Firestore deals exceptionally well with complex, large datasets. It also allows the combination of filtering and sorting functionalities at the same time. Unfortunately, the same thing cannot be said for the Realtime Database. Since data is organized as a tree, it is better to keep things as ”flat” as possible. As a result, Realtime Database is better suited for a simple scheme, that does not require a lot of querying or a complex structure.

2.3.3 Security

From a security point of view, Cloud Firestore seems to be again the winner. The only security provided by the Realtime Database is the Firebase Database Rules since the data validation is done separately, by the user. On the contrary, the Cloud Firestore deals with validation automatically, using Cloud Firestore Security Rules and Identity and Access Management.

2.3.4 Choosing A Database

Choosing the right type of storage seemed to be a rather difficult task, I must admit. I underestimated the complexity of my own data collection and started developing the application using the Realtime Database. It made sense at the time. The Realtime Database is after all a reliable, thoroughly-tested product. It offers real-time response and efficiency when dealing with simple data sets.

However, as the application evolved, the data became more complex, and querying seemed to be too slow and too limited for the new requirements. Eventually, I decided to switch to the Cloud Firestore.

The differences are noticeable. Fetching the data takes considerably less time, and queries became easier to write and process. The data was also easier to organize since the Cloud Firestore admits multiple data types, such as arrays, and subcollections within a document.

Chapter 3

Application structure

In this chapter, I will talk more about the application I have been working on. Some requirements and specifications will be presented, it will also be described how user data is stored, the relations between entities and how they interact with each other, and also some security features.

The application was developed using Flutter framework, version 2.1.0. In addition to the framework, some other plugins were installed. For the back-end part, Google's Firebase was used.

```
dependencies:
  flutter :
    sdk: flutter
  cupertino_icons: ^1.0.0
  firebase_core : ^1.0.0
  firebase_database: ^6.0.0
  firebase_analytics : ^5.0.2
  firebase_auth: ^1.0.0
  google_sign_in:
  json_annotation: 3.1.1
  provider: ^4.3.3
  firebase_dynamic_links: ^0.5.1
  http: ^0.13.0
  cloud_firestore : ^1.0.4
  firebase_storage : ^8.0.3
  logger: ^0.9.4
  font_awesome_flutter: ^9.0.0
  flutter_datetime_picker : ^1.5.0
  image_picker: ^0.7.4
  circular_profile_avatar : ^2.0.0
  image_cropper: ^1.4.0
```

```
firebase_messaging: ^10.0.0
async: any
url_launcher: ^6.0.4
overlay_support: any
  flutter_local_notifications : any
geocoding: ^2.0.0
geolocator: ^7.0.3
```

To get the app working on your mobile device, you should first install it locally. The application is compatible with Android devices having at least 18 SDK version and the user should be connected to the Internet.

3.1 Database

For better security over sensitive data, but also for a better, more efficient user experience, I chose to use the cloud database offered by Google's Firebase.

The app is working with six types of entities: users, connections, chats, messages, posts, comments.

Each user is uniquely identified by an id and has a password, a date of birth, a location, and a photo URL that represents the photo location on Firebase Storage. A connection or friendship is formed between two users and it has a date. Chats work in a similar way, but they have a collection of messages in addition.

Users, posts, and comments are tightly connected. A post is linked to a user and it can have several comments and upvotes. Comments have a text field, a date, and are identified by a post and a user (which can be different from the user that made the post).

3.2 Functionalities

3.2.1 Authentication And User Profile

Before being able to use the app, a user has to sign up first. Figure 3.2

App users have the option to sign in using a Google account. In this case, the data will be retrieved from Google and the account is ready to go. Figure 3.3

It is also possible to sign up using an email and password. In order to successfully create an account, the user must provide a valid, existing email address. After signing up, the user will be redirected to the edit profile page, where they are requested to provide information such as first name, last name, date of birth, and set a profile picture. Figure 3.4. Users have the possibility to update their profile picture

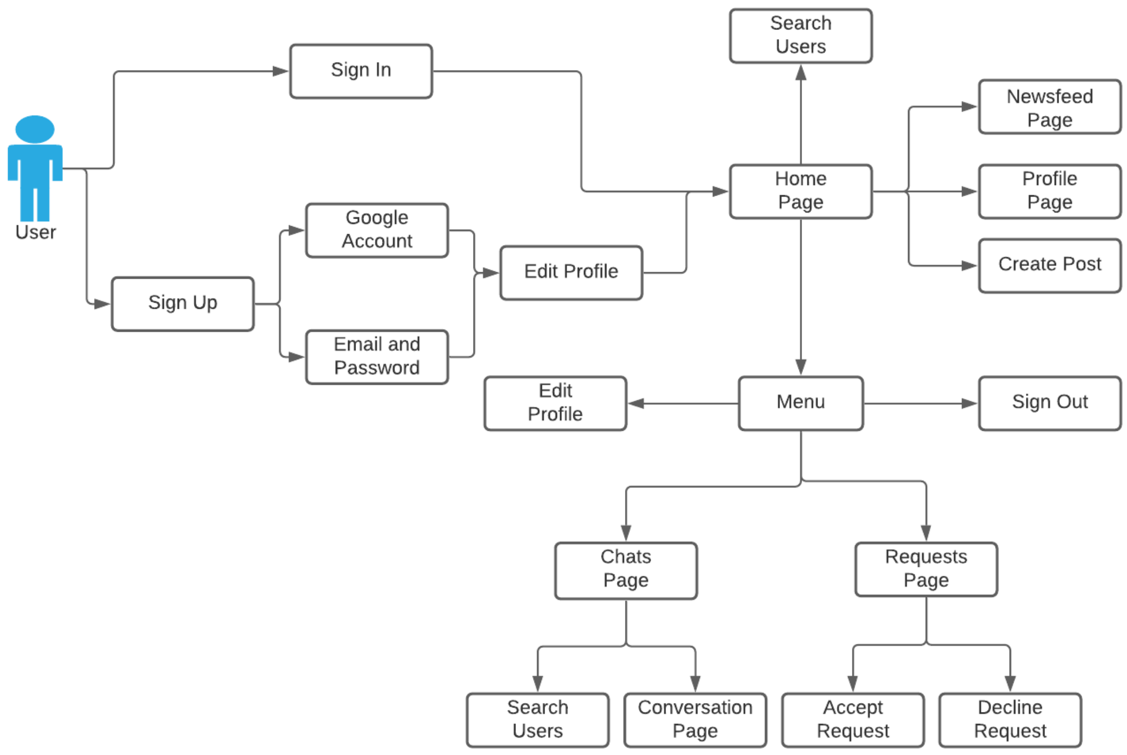


Figure 3.1: Application Flow

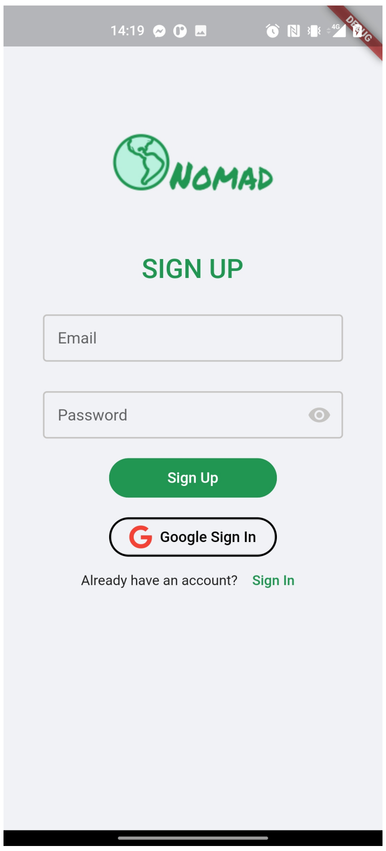
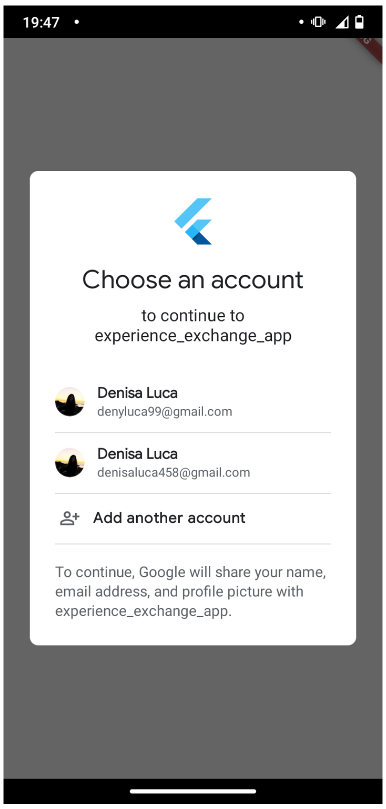


Figure 3.2: Sign Up Page


19:47

Choose an account

to continue to experience_exchange_app

Denisa Luca
denyluca99@gmail.com

Denisa Luca
denisaluca458@gmail.com

Add another account

To continue, Google will share your name, email address, and profile picture with experience_exchange_app.

Figure 3.3: Google Sign In

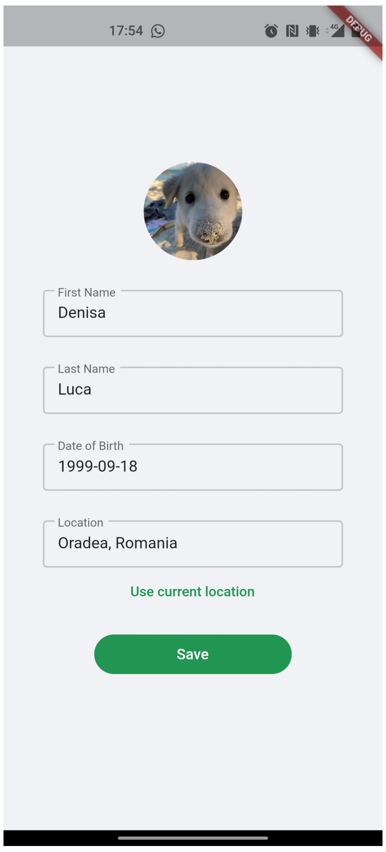
by either picking an image from their local storage or taking a new picture from the camera. When setting a location, users can opt for manually introducing an address, or using their current location. After all the fields have been filled, user information is updated and the image file is uploaded to the Cloud Firestore.

After all of these steps have been completed, the user can sign in to the application using the previously created credentials. The email must be valid and existent and the password must be the same as the one mentioned on the register page. Otherwise, an error message will be displayed.

3.2.2 User Posts

User posts are stored in the Cloud Firestore. Each user is able to create a post containing a text or a media file. Each of those fields is optional, however, an empty post cannot be created. If the application user chooses to add a photo to their post, it will be uploaded and stored safely on the Cloud Storage.

```
class Helper {
  static Future<String> uploadImage(File imageFile) async {
    Reference firebaseStorageRef =
        FirebaseStorage.instance.ref().child('/images').child(imageFile.path.split('/').last);
```



First Name
Denisa

Last Name
Luca

Date of Birth
1999-09-18

Location
Oradea, Romania

Figure 3.4: Edit Profile Page

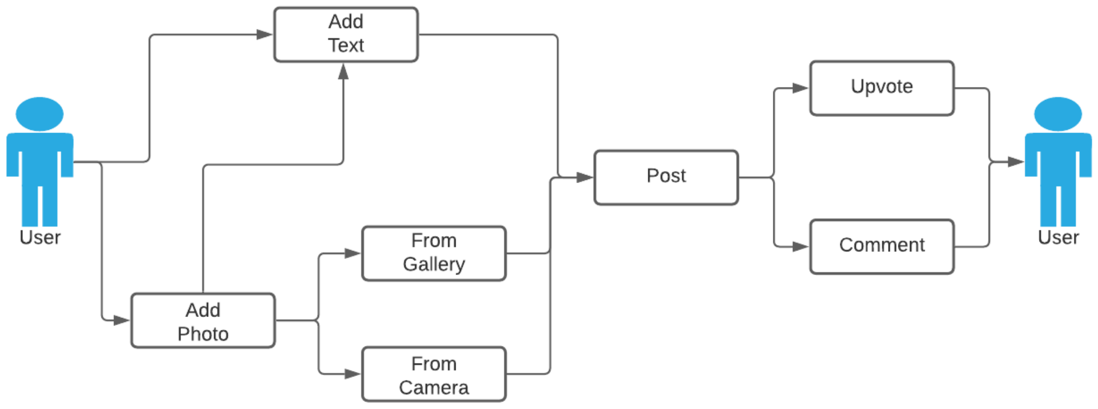


Figure 3.5: Creating a new post

```
      TaskSnapshot snapshot = await firebaseStorageRef.putFile(imageFile);
      return await snapshot.ref.getDownloadURL();
    }

    static Future<File> selectImageFromGallery() async {
      final pickedFile = await ImagePicker().getImage(source: ImageSource.gallery);
      return File(pickedFile.path);
    }

    static Future<File> selectImageFromCamera() async {
      final pickedFile = await ImagePicker().getImage(source: ImageSource.camera);
      return File(pickedFile.path);
    }

    static Future<File> cropImage(File selectedImage) async {
      return await ImageCropper.cropImage(
        sourcePath: selectedImage.path,
        maxWidth: 1080,
        maxHeight: 1080,
      );
    }
  }
```

When adding a photo to a post, the current user has the option to either select an existing photo from the gallery or to take a new photo using the built-in camera. In each case, permissions must be granted. Figures 3.6 and 3.7

A user can upvote or comment on posts. A post can be upvoted a single time, however, multiple comments can be added by the same user. A user can comment on a post only if they are connected to the post's creator. Figure 3.8

All the posts belonging to a specific user are displayed on their profile page. Figure 3.9

3.2.3 Connections

Connections are made between two users. When a connection between two users exists, each of them will receive updates about the other user's activity in their newsfeed.

A connection can be Pending, Accepted or Declined. When a user wants to connect to someone, a request is sent to the other user. The "receiver" can either accept the request, in which case the connection's status will be changed to Accepted, or

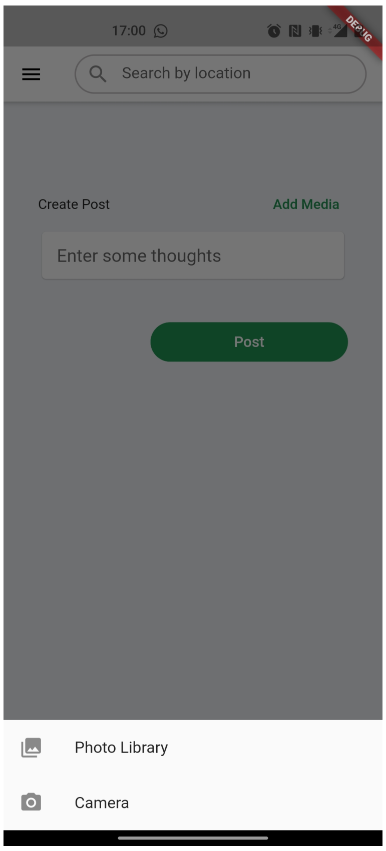
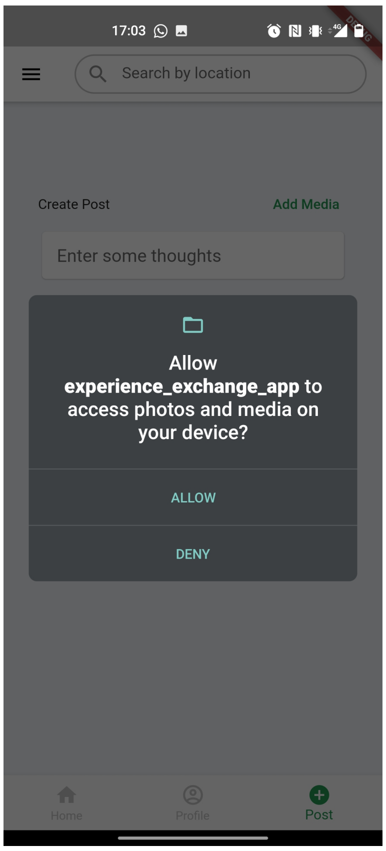
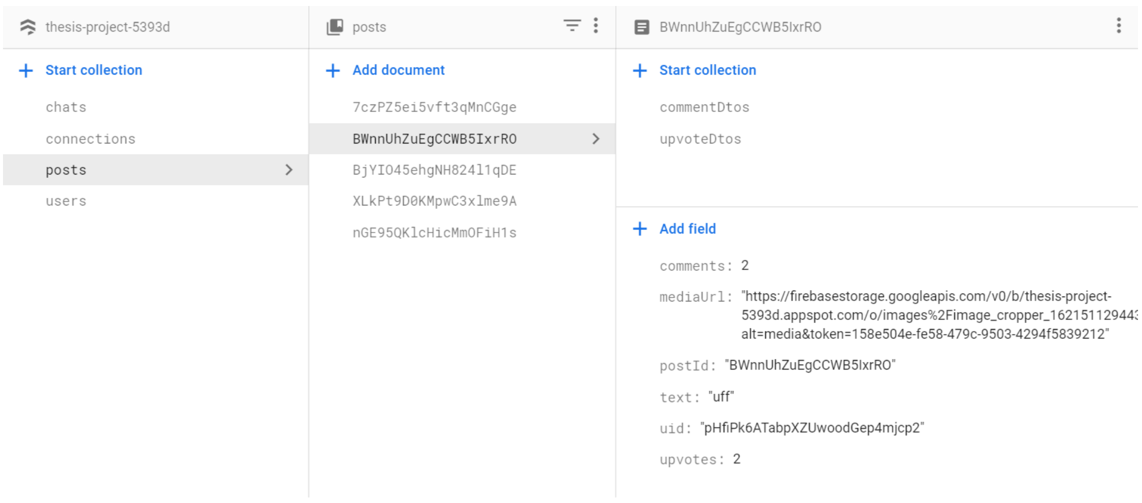


Figure 3.6: Photo Options



Allow **experience_exchange_app** to access photos and media on your device?

Figure 3.7: Ask for permission



```
chats
connections
posts
users

7czPZ5ei5vft3qMnCGge
BWnnUhZuEgCCWB5IxrRO
BjYI045ehgNH82411qDE
XLkPt9D0KMpwC3xlme9A
nGE95QKlcHicMmOFiH1s

commentDtos
upvoteDtos

comments: 2
mediaUrl: "https://firebasestorage.googleapis.com/v0/b/thesis-project-
          5393d.appspot.com/o/images%2Fimage_cropper_16215112944
          alt=media&token=158e504e-fe58-479c-9503-4294f5839212"
postId: "BWnnUhZuEgCCWB5IxrRO"
text: "uff"
uid: "pHfiPk6ATabpXZUwoodGep4mjcp2"
upvotes: 2
```

Figure 3.8: Posts Collection

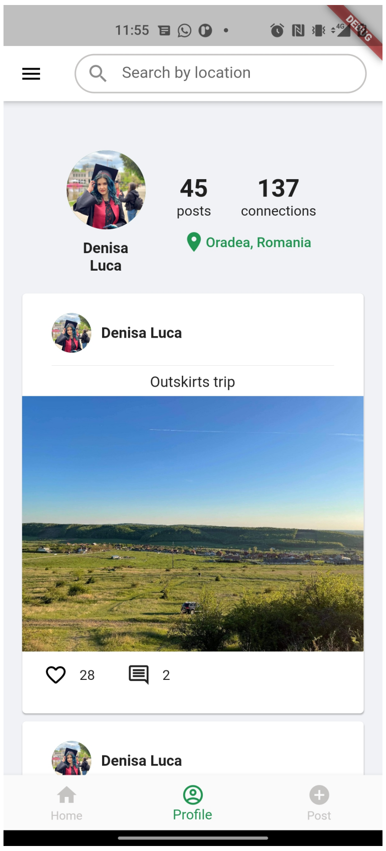
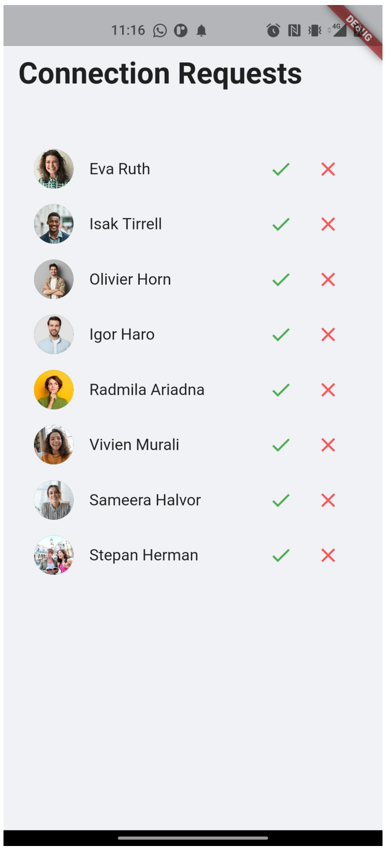


Figure 3.9: Profile Page

decline it. Figure 3.10



Connection Requests

Eva Ruth	✓	✕
Isak Tirrell	✓	✕
Olivier Horn	✓	✕
Igor Haro	✓	✕
Radmila Ariadna	✓	✕
Vivien Murali	✓	✕
Sameera Halvor	✓	✕
Stepan Herman	✓	✕

Figure 3.10: Requests

Until the receiver chooses an action, the connection will remain in pending, and the request can be canceled by the sender at any time. Figure 3.11

3.2.4 Chats

In the context of a social media application, offering the users a platform for chatting, where they can get to know each other is crucial. A user has the possibility to see their list of chats at any time, as well as search for a user by name and start a new chat. After a chat is selected from the list, the individual chat page opens and the conversation can begin. Fig 3.12 and 3.13

A chat consists of two users and a collection of messages. Fig 3.14

In turn, messages described by a sender(uid1), a receiver(uid2), a date, the actual content, and a boolean that is set to true when a message is read by the receiver. Fig 3.15

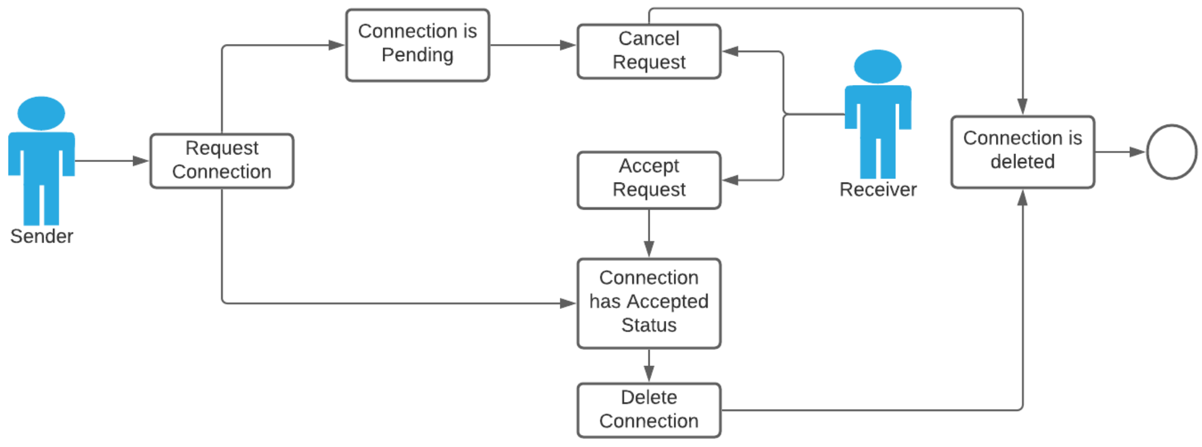
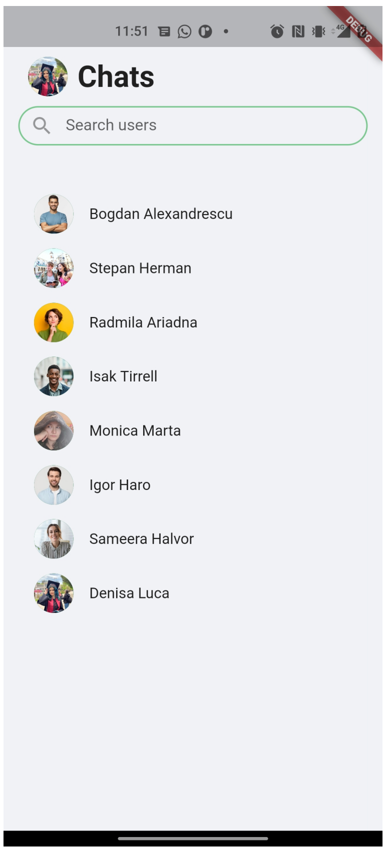
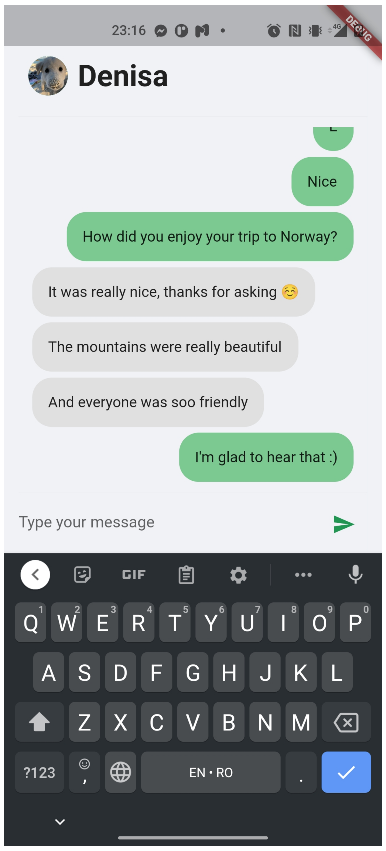


Figure 3.11: Connection Flow



Bogdan Alexandrescu
Stepan Herman
Radmila Ariadna
Isak Tirrell
Monica Marta
Igor Haro
Sameera Halvor
Denisa Luca

Figure 3.12: Chats Page



Nice

How did you enjoy your trip to Norway?

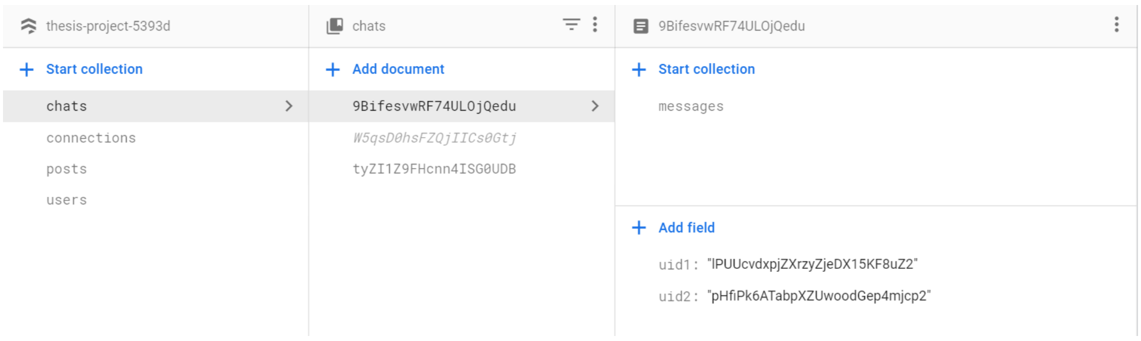
It was really nice, thanks for asking ☺️

The mountains were really beautiful

And everyone was soo friendly

I'm glad to hear that :)

Figure 3.13: Conversation View



```
chats
connections
posts
users
```

```
9BifesvwRF74ULOjQedu
W5qsD0hsFZQjIICs0Gtj
tyZI1Z9FHcnn4ISG0UDB
```

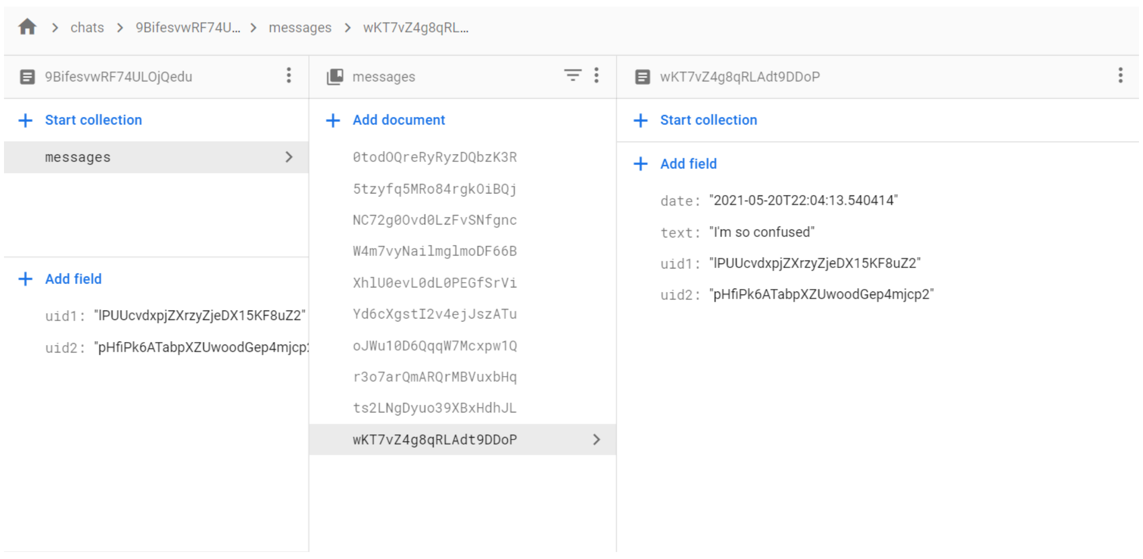
9BifesvwRF74ULOjQedu

messages

uid1: "lPUUcvdxpjZXrzyZjeDX15KF8uZ2"

uid2: "pHfiPk6ATabpXZUwoodGep4mjcp2"

Figure 3.14: Chat Collection



```
messages

uid1: "lPUUcvdxpjZXrzyZjeDX15KF8uZ2"
uid2: "pHfiPk6ATabpXZUwoodGep4mjcp2"
```

```
0tod0QreRyRyzDQbzK3R
5tzyfq5MRo84rgk0iBQj
NC72g0Ovd0LzFvSNfgnc
W4m7vyNailmglmoDF66B
XhlU0evL0dL0PEGfSrVi
Yd6cXgstI2v4ejJszATu
oJWu10D6QqqW7Mcxpw1Q
r3o7arQmARQrMBVuxbHq
ts2LNgDyuo39XBxHdhJL
wKT7vZ4g8qRLAdt9DDoP
```

```
date: "2021-05-20T22:04:13.540414"
text: "I'm so confused"
uid1: "lPUUcvdxpjZXrzyZjeDX15KF8uZ2"
uid2: "pHfiPk6ATabpXZUwoodGep4mjcp2"
```

Figure 3.15: Message Collection

3.2.5 Search

The search functionality is available both on the chat page, as well as the main page. On the chat page, a search is performed by first name and last name. When the user taps on a search result, they are redirected to the conversation page.

On the main page, the search displays all the users in a given location. Figure 3.16 Then, the current user is redirected to the profile page of the user in the result set. On that profile, they can tap on the location button and open a link redirecting them to google maps with the set location. If the app is not installed on the device, a browser will be opened.

3.3 Geolocator And Geocoding Packages

The Geolocator is used to get the current/last known physical location of the device in terms of latitude and longitude. Once that data is retrieved, the Geocoding converts those parameters to human-readable strings, such as city, country, postal code.

In order to use these packages, a bit of setup is needed.

For iOS, in the info.plist file, the following key-value pairs should be added:

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>This app needs access to location when open.</string>

<key>NSLocationAlwaysUsageDescription</key>
<string>This app needs access to location when in the background.</string>
```

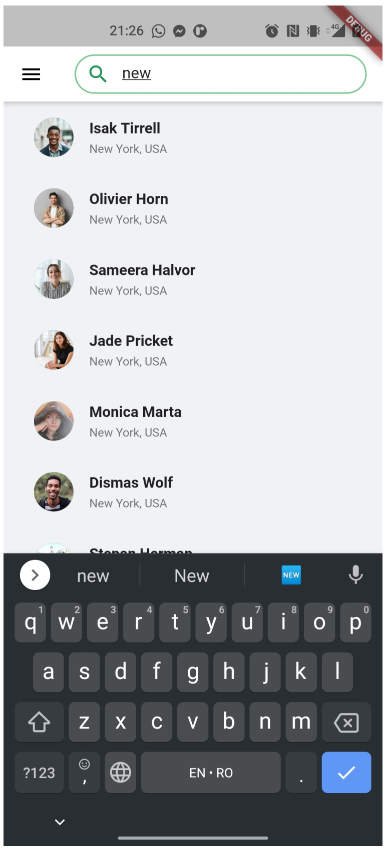


Figure 3.16: Search Users

```
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>
<string>This app needs access to location when open and in the
    background.</string>
```

For Android, one can either opt for coarse or fine location. Fine location is the most precise, whereas coarse location gives results equal to about a city block.[9] For the purposes of this application, coarse location sufficed.

In AndroidManifest.xml file, as a direct child of the ¡manifest¿ tag, add either

```
<uses–permission
    android:name=”android.permission.ACCESS_COARSE_LOCATION”/>
```

or

```
<uses–permission
    android:name=”android.permission.ACCESS_FINE_LOCATION”/>
```

3.4 Provider

The provider package in Flutter offers a good solution to persisting and passing data throughout all the states of the app.[2] I chose to use a provider(instead of redux [7], bloc, etc.) because it is easier to understand and doesn't use much code, yet it's still effective.

Provider is mainly a dependency injection mechanism [18]. It can be used in order to track the current user within an application, as well as listen to authentication changes. Moreover, one can use the Provider to inject certain services and make them available throughout the app. A Multi provider can be used to inject multiple values/providers at once.

```
return MultiProvider(providers: [
      ChangeNotifierProvider(create: (context) => AuthenticationService()),
      ChangeNotifierProvider(create: (context) => UserService()),
      ChangeNotifierProvider(create: (context) => PostService()),
      ChangeNotifierProvider(create: (context) => ChatService()),
      ChangeNotifierProvider(create: (context) => ConnectionService()),
      StreamProvider(create: (context) =>
          context.read<AuthenticationService>().authStateChanges),
    ],
```

3.5 Sending data over the network

Data is sent to the Firebase server in a JSON format. For data conversion, I added build_runner and json_serializable as dev_dependencies, together with json_annotation as a dependency.

```
dev_dependencies:
   flutter_test :
     sdk:  flutter
  build_runner: any
   json_serializable :  3.5.1
```

Let's take the UserDto class as an example. When running "flutter pub run build_runner build" in the terminal, the file 'userdto.g.dart' is generated. The @JsonSerializable annotation lets the compiler know that the userdto.g.dart should contain methods for encoding and decoding the data in a JSON format. The methods _$UserDtoFromJson(json) and _$UserDtoToJson(this) are automatically implemented by the build_runner [10].

```
part 'userdto.g.dart';

@JsonSerializable()
class UserDto{
  String firstName;
  String lastName;
  String location;
  DateTime dateOfBirth;
  String photoUrl;

  UserDto(this.firstName, this.lastName, this.location, this.dateOfBirth,
      this.photoUrl);

  factory UserDto.fromJson(Map<String, dynamic> json) =>
      _$UserDtoFromJson(json);

  Map<String, dynamic> toJson() => _$UserDtoToJson(this);
}
```

Chapter 4

Future Work And Possible Improvements

This chapter is meant to summarize the main goals and achievements of this thesis, as well as provide ways of further developing and improving the application.

The aim of this thesis was to promote a socialization platform designed for travelers, where people can get in touch with others based on location, find a travel companion, and take those "virtual" friendships to the next level.

As for now, users are able to share their experiences through posts, as well as upvote or comment on them. Users' location can be set to the current physical location, using the Geolocator and Geocoding packages from Flutter. It is possible to find and connect to any users from the desired location. Most importantly, the application offers chat functionalities where people can get in touch and start making the plans for the next adventure.

For now, the application is still in the development phase. Multiple functionalities can be added in order to improve user experience and safety.

First of all, a section for personal description and interests should be added to the profile page, to get a grasp of one's personality.

A connection recommendation system can be implemented based on common interests. In this way, it will be easier for travelers to find a companion with a similar personality. In the same manner, connections may also be recommended based on location, age, or shared connections. Calendar support can also be included. This will enable users to make plans together and agree on specific dates with some visual help as well.

In regards to user safety, one of the top priorities is to introduce verified profiles. A verified profile implies that there is a certainty that the owner of the specific profile is whom they claim to be. A way to verify someone's identity is through ID card validation. Users will have to upload a photo of their ID card, together with a photo of themselves taken on the spot, and send it for further verification. After these steps will have been completed, the profile will be verified.

Lastly, the UI has room for improvement. When developing this application, the main focus was the functionalities and getting them to work properly. The interface wasn't a top priority and, unfortunately, it shows. The design should be more intuitive and easy to use. For a visually appealing effect, animations can be added when upvoting or commenting on a post, or when sending a connection request.

In conclusion, the main focus of this application was to provide a platform where travelers can connect and make plans while focusing on the social and cultural aspects of the journey.

Bibliography

[1] React native vs. flutter: What is better for app development in 2021? https://nix-united.com/blog/flutter-vs-react-native/performance. Online; accessed 14 May 2021.

[2] Simple app state management. https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple. Online; accessed 29 January 2021.

[3] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, 1994.

[4] J. Bishop and N. Horspool. Cross-platform development: Software that lasts. *Computer*, 39(10):26–35, 2006.

[5] D. Blog. From history of web application development. https://www.devsaran.com/. Online; accessed 25 April 2021.

[6] P. S. Chunnu Khawas. Application of firebase in android app development-a study. https://www.researchgate.net/publication/325791990_Application_of_Firebase_in_Android_App_Development-A_Study. Online; accessed 26 April 2021.

[7] N. Delic. Redux and epics for better-organized code in flutter apps. https://medium.com/upday-devs/reduce-duplication-achieve-flexibility-means-success-for-the-flutter-app-e5e432839e61. Online; accessed 14 June 2021.

[8] R. Gibb. What is a web application? https://blog.stackpath.com/web-application/. Online; accessed 25 April 2021.

[9] P. Halliday. How to get a user's location with the geolocator plugin in flutter. https://www.digitalocean.com/community/tutorials/flutter-geolocator-plugin. Online, accessed 4 June 2021.

[10] D. Mackier. Automatic json serializing in flutter using json annotation. https://medium.com/flutter-community/generate-the-code-to-parse-your-json-in-flutter-c68aa89a81d9. Online; accessed 14 June 2021.

[11] L. Moroney. Cloud storage for firebase. In *The Definitive Guide to Firebase*, pages 73–92. Springer, 2017.

[12] L. Moroney. Google analytics for firebase. In *The Definitive Guide to Firebase*, pages 251–270. Springer, 2017.

[13] L. Moroney. Using authentication in firebase. In *The Definitive Guide to Firebase*, pages 25–50. Springer, 2017.

[14] L. Moroney, Moroney, and Anglin. *Definitive Guide to Firebase*. Springer, 2017.

[15] S. O'Dea. Forecast number of mobile users worldwide from 2020 to 2024. https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/. Online; accessed 27 April 2021.

[16] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, pages 263–273, 2016.

[17] J. Reifman. What is html5? https://code.tutsplus.com/tutorials/what-is-html5–cms-25803. Online; accessed 25 April 2021.

[18] D. Slepnev. State management approaches in flutter. 2020.

[19] C. Strauch, U.-L. S. Sites, and W. Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 20:24, 2011.

[20] W. Wu. React native vs flutter, cross-platforms mobile application frameworks. 2018.