



Software Requirements Specification

for

<Care Connect >

Version <1.0>

Team Members

<name>

<student ID>

Yahya Isaac (洛克)

20193290616

Ali Sharh (阿里)

20193290693

Mouai Shaban (米度)

20193290702

Salem Al-Mahdi (阿德)

20193290682

Mohammed Al-Kazrajy (莫汉)

20193290706

Teacher: Di Liu

Course: Innovative Design & Research

Date: 27/06/2022

1 Introduction

1.1 Document Purpose

In this document, we are going to show one of the great ideas we had in mind for a long time which is a mobile phone App. This App is going to focus on the medical field in which it allows the patients to connect with their doctors directly via chat, and with this App, patients don't have to worry about forgetting to take their medicines because in the app there will be a reminder feature where the patient can set their reminder to remind them of taking a certain medicine and they will be notified when it's the time to take the medicine. and we have named this App "Care Connect"

1.2 Product Scope

Helping patients save their money because they don't have to get a taxi to attend an appointment with their doctor to ask a simple question. They can have a quick chat with them.

Easy access to the doctor when it's an urgent case e.g. a person with a disability wants help they can reach the doctor very easily.

Constant communication between the patient and the doctor. Doctors can make sure that patients have taken their medicines on time and patients can talk to their doctors whenever they want.

By having the ability to chat and getting reminders both in one application, Elder people especially those who live alone and need to be taken care of and get reminded to take their medicines and also being supervised and check on them if they have taken their medicines or not yet, will get the help and service they needed from Care Connect.

1.3 Intended Audience and Document Overview

The targeted markets are mostly Yemen and North Africa. There is no existence of such applications or websites in those countries.

The targeted users:

- 1- Elder people
- 2- Disabled people
- 3- People with no helping hand near them
- 4- People who can't afford transportation in some areas or their areas have no transportation.
- 5- Sick people, those who have a highly infectious disease such as influenza or the new covid.

1.4 References and Acknowledgments

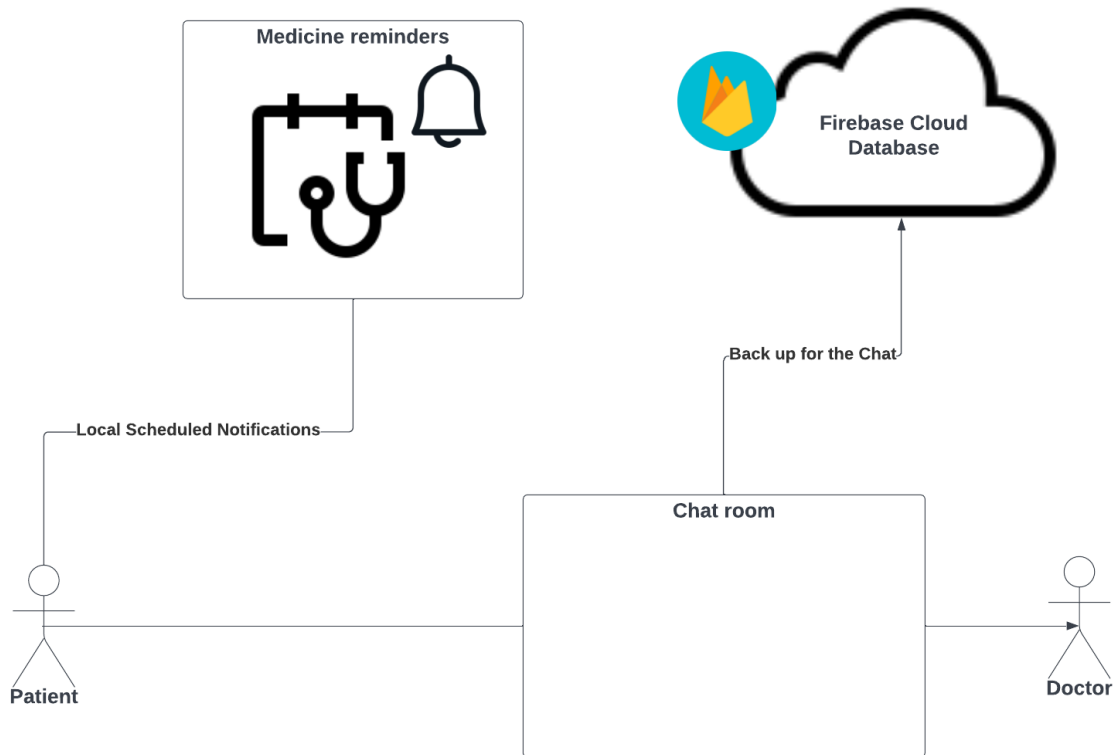
These are all of the technologies and websites that have made this document and Application possible to be made:

1. **Flutter & Dart:** Flutter is an open-source UI software development kit created by Google. It is used to develop cross-platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase. First described in 2015, Flutter was released in May 2017
2. **Figma:** Figma is a vector graphics editor and prototyping tool which is primarily web-based, with additional offline features enabled by desktop applications for macOS and Windows. The Figma mobile app for Android and iOS allows viewing and interaction with Figma prototypes on real-time mobile devices.
3. **Firebase:** Google Firebase is a Google-backed application development software, Firebase provides tools for tracking analytics, reporting and fixing app crashes, and creating marketing and product experiments. The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, real-time events continue to fire, giving the end-user a responsive experience.
4. **Git & GitHub:** Git is open-source version control software, used for managing and tracking file revisions. It can be used with any file type but is most often used for tracking code files. Git is the most widely used version control system in software development, and GitHub leverages this technology for its service, hence its name
5. **Lucidchart:** Lucidchart is a web-based diagramming application that allows users to visually collaborate on drawing, revising, and sharing charts and diagrams, and improve processes, systems, and organizational structures. It is produced by Lucid Software Inc., based in Utah, United States, and co-founded by Ben Dilts and Karl Sun.

2 Overall Description

2.1 Product Overview

This product is new self-contained software, It has its own database and is hosted in the cloud via google's service "Firebase Cloud Database". However, while making this project we used a lot of packages such as (Locat notification, Firebase core, and Provider) which are shared packages contributed by other developers to the Flutter and Dart ecosystems. This allowed us to quickly build an app without having to develop everything from scratch.



This diagram shows a general overview of how users will be using CareConnect. The Patient is going to have a chat with their doctors after the Patient signed up and then signed in to the app. After that, the Patient will search for the Doctor and add him/her to their list so they can chat with them, then, that chat will be backed up to the cloud, and if the Patient had a reminder to take a certain medicine they will get a scheduled local notification of taking that medicine.

2.2 Product Functionality

In this section I'm going to list all of the functionalities of Care Connect and give brief details of each one of them.

1. Sign Up: New users are going to be able to sign up for new accounts using their emails and those kind of users are going to be stored in the database as patients because doctors are going to be added to the system manually from the dashboard to avoid getting uncertified doctors in the app or normal people in the app pretending to be doctors.
2. Login: Users who already signed up in the app are going to be able to get into their accounts by login in directly after they put in their email and passwords they first will be authenticated and then get access to their accounts.

3. Authentication: The system will be authenticating users when they sign in or log in, Thanks to the Firebase package.
4. Search for a doctor: The user (Patient) should be able to search for the doctors by their emails.
5. Add the doctor: Users (Patients) after finding the doctor are going to be able to add the doctor to their doctor list.
6. Chatting: Doctors can chat with their patients and patients can chat with their doctors.
7. Reminder: Patients should be able to add, edit or delete their medicine reminders.
8. Notifications: Patients can assign their reminders at a specific time to give them a notification to take

2.3 Work-Flow and Techniques

In this project, we are planning to split the work among every member of the group. There are members who are responsible for the coding, designing, database, and testing -both the database and the actual application-.

Since the application has two major functionalities (Chatting and Reminders) we assigned each feature to one group member. One member is going to program the chatting functionality after he receives the design from the member who is responsible for that, and then test it separately by the group member who is assigned to do so.

The same thing goes with the reminder functionality. The member who is assigned to do that task will program the reminder functionality after receiving the design separately, and then test it with the group member who is responsible for that.

While working on the program if we needed to connect any function with the database and we need to set up the database in a certain way, the group member who is responsible for monitoring and managing the database will take care of that and he will be in contact with the programmer who wants that certain set up in the database.

After we finish programming, set up the database, and test each function separately we will connect these two functions together and then test them for the last time to check if they are working the way we want them to work.

Lastly, at the end we will work on the ability for the doctor to send a reminder that all the patient has to do is to add that reminder to their reminders list, and what is special about that reminder is that the patient does not have the ability to edit or delete the reminder from their side, and the doctor will get a notification if the patient took the medicine or not.

3 Specific Requirements

3.1 Design for User Interface

3.1.1 User Interfaces

The whole aesthetic and design of the app should be consistent, clean, minimal, and simple to understand and navigate throughout the app, the app will have two main colors: Blue and White, and the font is from the Roboto family.

These are the early design samples of the app that we did in Figma:



Login to your Account

Email

Password

Sign in

Don't have an account? [Sign](#)



Create a new Account

Full Name

Email

Password

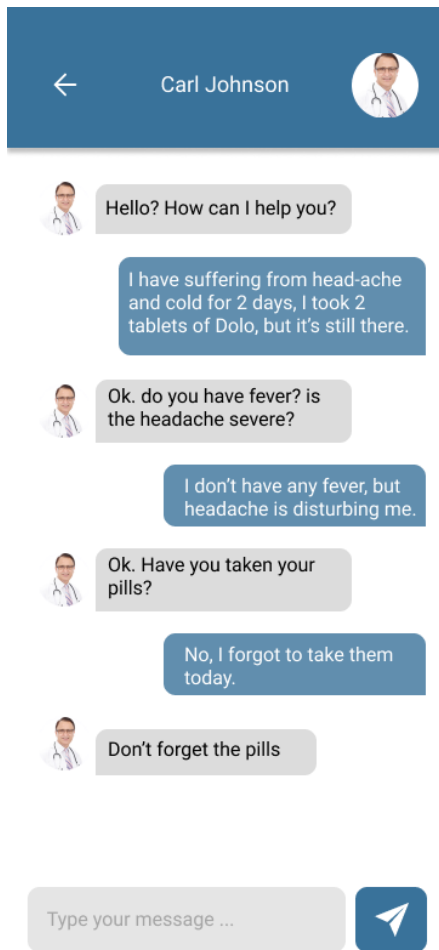
Date of birth:

Pick a date

Gender:



Sign up




3.2 Functional Requirements

in this section, I'm going to show screenshots of the app and a snippet of the code and explain briefly what this code does and what is used for

3.2.1 F1: Sign-up

2:13





Create a new Account :

Full Name

Email

Password

Date of birth: 2022 - 6 - 30

Gender:  

Sign Up

Already have an account? [Sign in](#)

Checking if email and password are not empty and then using the signup function

```
onPressed: () {  
    try {  
        final String email = Var.emailController.text.trim();  
        final String password = Var.passwordController.text.trim();  
        final String userName = Var.fullNameController.text.trim();  
        if (email.isEmpty()) {  
            print('Email is Empty');  
        } else {  
            if (password.isEmpty()) {  
                print('Password is Empty');  
            } else {  
                context  
                    .read<AuthenticationService>()  
                    .signup(  
                        email: email,  
                        password: password,  
                    )  
            }  
        }  
    }  
}
```


The sign-up function from firebase

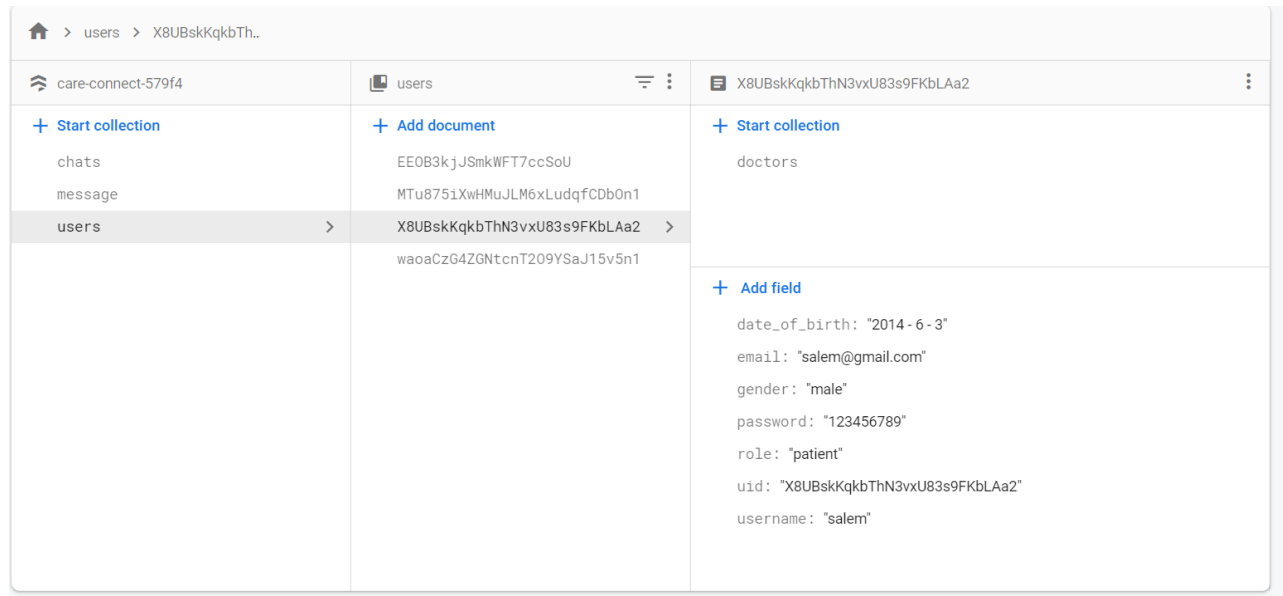
```
Future<String?> signup(
  {required String email, required String password}) async {
  try {
    await _firebaseAuth.createUserWithEmailAndPassword(
      email: email, password: password);
    return 'Signed Up';
  } on FirebaseAuthException catch (e) {
    return e.message;
  }
}
```

Sending the user's information to the database and save them in a collection called "users"

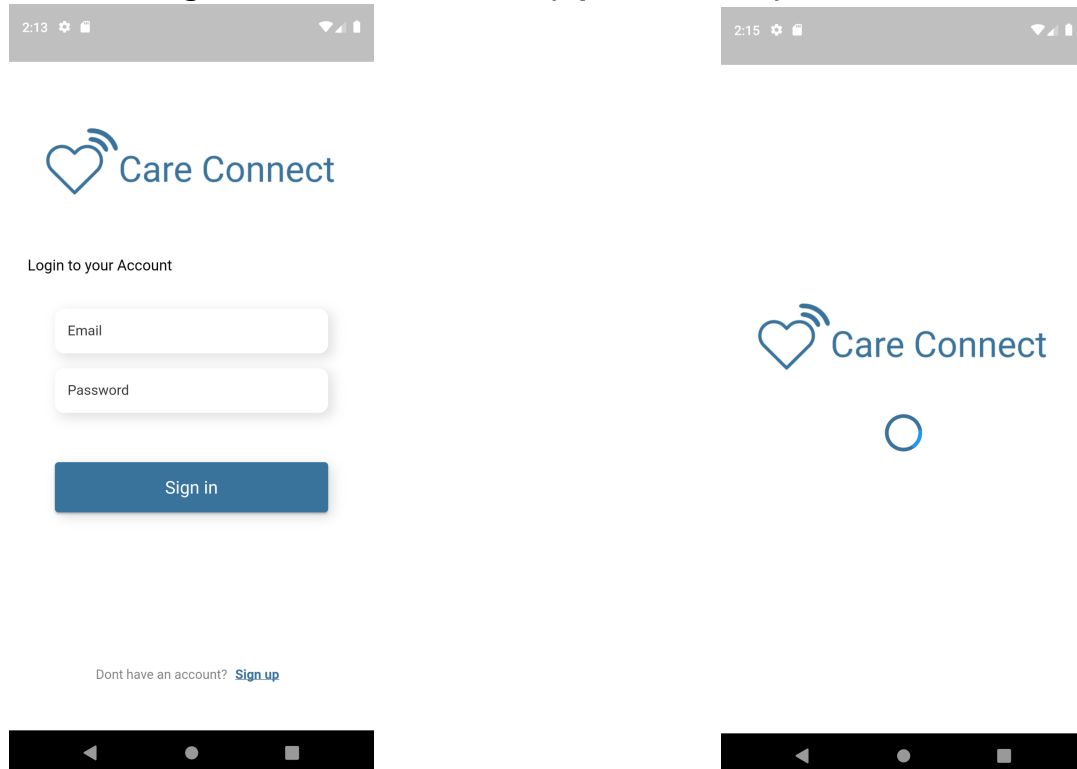
```
.then((value) async {
  User? user = FirebaseAuth.instance.currentUser;

  await FirebaseFirestore.instance
    .collection("users")
    .doc(user?.uid)
    .set({
      'uid': user?.uid,
      'email': email,
      'password': password,
      'username': userName,
      'gender': Var.gender,
      'date_of_birth':
        '${Var.dateofbirth!.year} - ${Var.dateofbirth!.month} - ${Var.dateofbirth!.day}',
      'role': Var.role,
    });
});
```

This is from the firebase Database



3.2.2 F2: Log-in and Authentication (Splash screen)



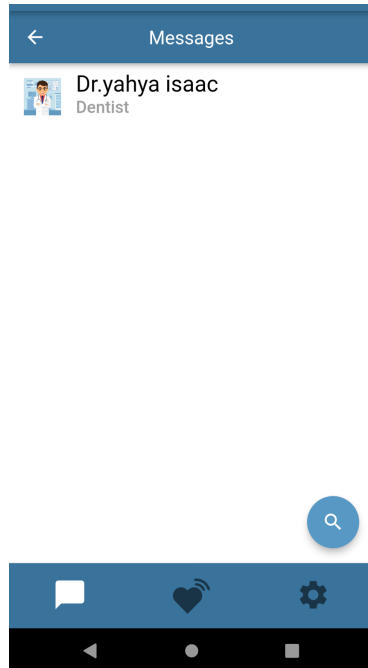
Checking if your email and password are not empty and then using the Login function and after that taking you to the splash screen

```
onPressed: () {}  
    final String email = Var.emailController.text.trim();  
    final String password = Var.passwordController.text.trim();  
    if (email.isEmpty) {  
        print('Email is Empty');  
    } else {  
        if (password.isEmpty) {  
            print('Password is Empty');  
        }  
    }  
    context.read<AuthentificationService>().login(  
        email: email,  
        password: password,  
    );  
    Navigator.pushNamed(context, SplashPage.pageRoute);  
},
```

The Login function

```
Future<String?> login(  
    {required String email, required String password}) async {  
    try {  
        await _firebaseAuth.signInWithEmailAndPassword(  
            email: email, password: password);  
        return 'Signed In';  
    } on FirebaseAuthException catch (e) {  
        return e.message;  
    }  
}
```

3.2.3 F3: Patient POV messages page



this stream is going to get all of the doctors that this particular user has added and it will read it from the database and display them on the main messages page of the patient

```
Stream<List<Doctor>> readDoctors() => FirebaseFirestore.instance
    .collection('users')
    .doc(_auth.currentUser!.uid)
    .collection('doctors')
    .snapshots()
    .map((snapshot) =>
        snapshot.docs.map((doc) => Doctor.fromJson(doc.data())).toList());
```

As you can see here this user has added this doctor to their list of doctors that they can chat with

The screenshot shows a web application interface with a breadcrumb trail: `users > X8UBskKqkbThN3vxU83s9FKbLAa2 > doctors > eqDWVKBKSVoGDdbIQYVW`. Below the trail, there are three panels. The first panel, titled 'X8UBskKqkbThN3vxU83s9FKbLAa2', contains a '+ Start collection' button and a list of fields for a user profile: `date_of_birth: "2014-6-3"`, `email: "salem@gmail.com"`, `gender: "male"`, `password: "123456789"`, `role: "patient"`, `uid: "X8UBskKqkbThN3vxU83s9FKbLAa2"`, and `username: "salem"`. The second panel, titled 'doctors', contains a '+ Add document' button and a list of fields for a doctor profile: `doctorname: "yahya isaac"`, `imageurl: "https://img.freepik.com/free-vector/doctor-character-background_1270-84.jpg?w=2000"`, and `specialty: "Dentist"`. The third panel, titled 'eqDWVKBKSVoGDdbIQYVW', contains a '+ Start collection' button and a '+ Add field' button.

3.2.4 F4: Searching page

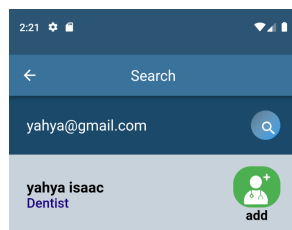
The screenshot shows a mobile application interface with a search bar. The search bar has a back arrow on the left, a 'Search' label in the center, and a magnifying glass icon on the right. Below the search bar, the text 'yahya@gmail.com' is visible.

The screenshot shows a mobile application interface with a keyboard. The keyboard is a standard QWERTY layout with a search bar at the top. The search bar has a back arrow on the left, a magnifying glass icon on the right, and a green checkmark button at the bottom right. The keyboard is open, showing the letters 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p' on the first row, 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l' on the second row, and 'z', 'x', 'c', 'v', 'b', 'n', 'm' on the third row. The bottom row contains a spacebar, a comma/semicolon key, a smiley face icon, and a green checkmark button.

onSearch() is the function that is responsible for comparing the text (email) that the user has texted to the emails of doctors in the database and display them if they exist

```
onSearch() async {  
  FirebaseFirestore _firestore = FirebaseFirestore.instance;  
  
  setState(() {  
    isLoading = true;  
  });  
  
  await _firestore  
    .collection('users')  
    .where('email', isEqualTo: _search.text)  
    .get()  
    .then((value) {  
      setState(() {  
        userMap = value.docs[0].data();  
        isLoading = false;  
      });  
    });  
});  
  
print(userMap);  
}
```

3.2.5 F5: Results of searching

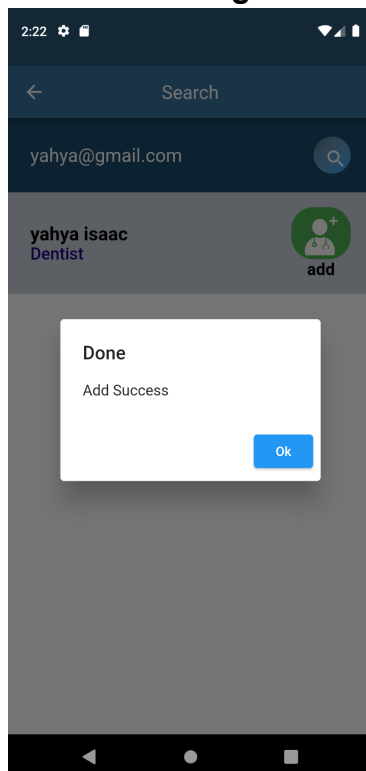


The userList() widget is going to display the doctor's name and specialty in the search results

```
Widget userList() {
  return userMap != null
    ? ListView.builder(
      shrinkWrap: true,
      itemCount: 1,
      itemBuilder: (context, index) {
        return userTile(userMap?['username'], userMap?['specialty']);
      })
    : Container();
}

Widget userTile(
  dynamic doctorname,
  dynamic specialty,
) {
```

3.2.6 F6: Adding a new doctor



When pressing on the add icon next to the doctor information, the doctor will be added to the user's doctors list (the "doctors" collection)

```
GestureDetector(  
  onTap: () async {  
    await FirebaseFirestore.instance  
      .collection('users')  
      .doc(_auth.currentUser!.uid)  
      .collection('doctors')  
      .add({  
        "doctorname": userMap!['username'],  
        "specialty": userMap!['specialty'],  
        "imageurl": userMap!['image'],  
      })  
  })  
)
```

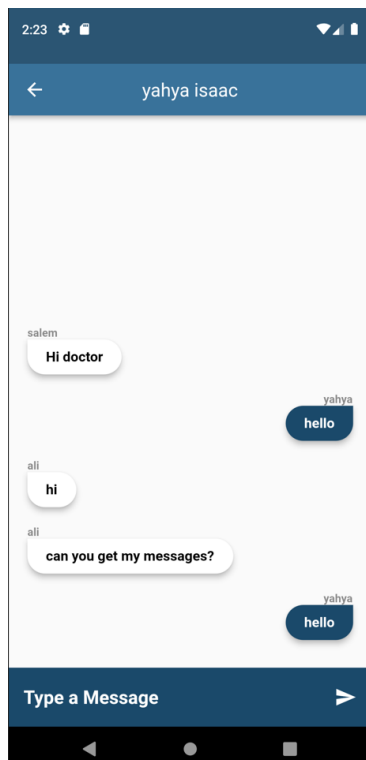
and when completing the process the app will show a message saying that the adding is done and you can chat with the doctor

```
.whenComplete(() => showDialog(  
  context: context,  
  builder: (BuildContext context) {  
    return AlertDialog(  
      title: Text('Done'),  
      content: Text('Add Success'),  
      actions: <Widget>[  
        ElevatedButton(  
          child: Text('Ok'),  
          onPressed: () {  
            Navigator.pushReplacementNamed(  
              context, PatientMainPage.pageRoute);  
            },  
        ), // ElevatedButton  
      ], // <Widget>[]  
    ); // AlertDialog  
  }  
)),
```


But right after you add the doctor, your chat will be available to the doctor in their main messages on page

```
.whenComplete(() async {  
  await FirebaseFirestore.instance  
    .collection('users')  
    .doc(userMap!['uid'])  
    .collection('patient')  
    .add({  
      'patientemail': _auth.currentUser!.email,  
      'patientId': _auth.currentUser!.uid  
    });  
});
```

3.2.7 F7: Patient POV chatting page



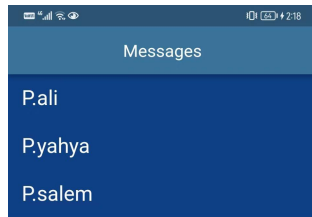
The chatting page isn't different between the Patient POV and the Doctor POV, and because the messages are backed up to the database therefor whenever any of the users send a message it will be collected and saved in the database

```
onTap: () {  
  _firestore.collection('message').add({  
    'text': messageText,  
    'sender':  
      signedInUser.email!.replaceAll('@gmail.com', ''),  
    'time': FieldValue.serverTimestamp(),  
  });  
  
  messageController.clear();  
},
```

the messages are going to be displayed on the screen using the StreamBuilder function which it will query the messages from the database and show them in the chat room ordered by the time this message has been sent on

```
@override  
Widget build(BuildContext context) {  
  return StreamBuilder<QuerySnapshot>(  
    stream: _firestore.collection('message').orderBy('time').snapshots(),  
    builder: ((context, snapshot) {  
      List<Widget> messageWidget = [];  
  
      if (!snapshot.hasData) {  
        return Center(  
          child: CircularProgressIndicator(),  
        ); // Center  
      }  
  
      final messages = snapshot.data!.docs.reversed;  
      for (var message in messages) {  
        final messageText = message.get('text');  
        final messageSender = message.get('sender');  
        final currentUser =  
          signedInUser.email!.replaceAll('@gmail.com', '');  
      }  
    })  
  );  
}
```

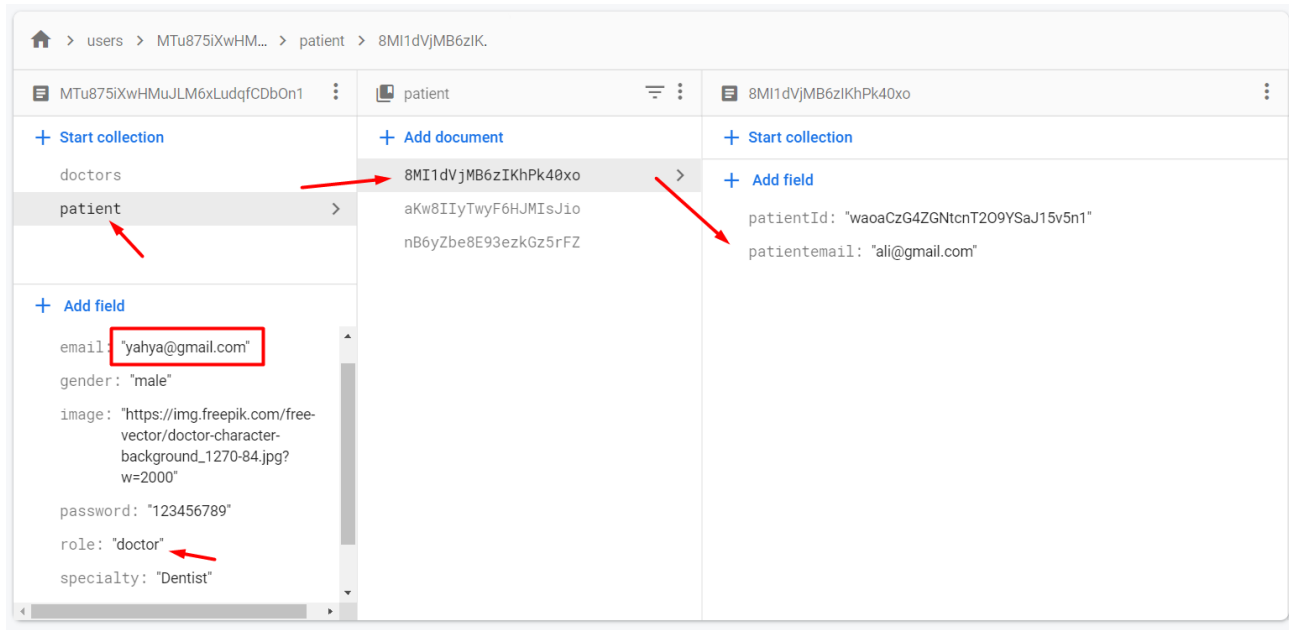
3.2.8 F8: Doctor POV messages page



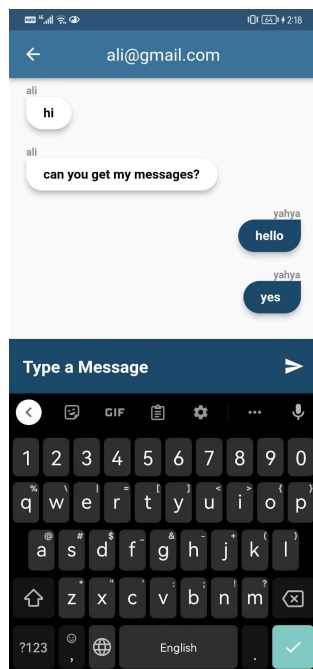
This stream is going to get access to the 'users' collection and check if the user's role is "Doctor" and then show all of the patients in the 'patient' collection who added this particular doctor

```
Stream<List<Patient>> readPatient() => FirebaseFirestore.instance
    .collection('users')
    .doc(_auth.currentUser!.uid)
    .collection('patient')
    .snapshots()
    .map((snapshot) =>
        snapshot.docs.map((doc) => Patient.fromJson(doc.data())).toList());
```

and here how it looks like in the database

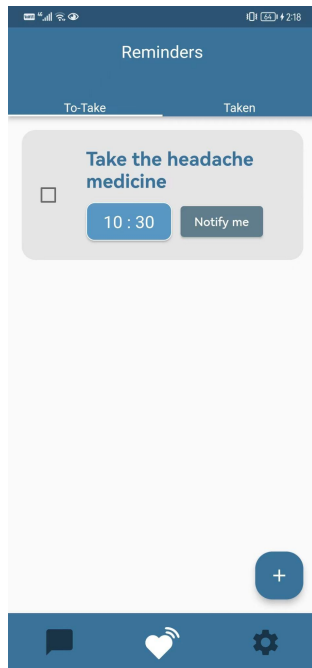


3.2.9 F9: Doctor POV chatting page



The chat room from the doctor's POV looks the same compared to the patient one but the only difference is that it shows the patient's email instead of his name

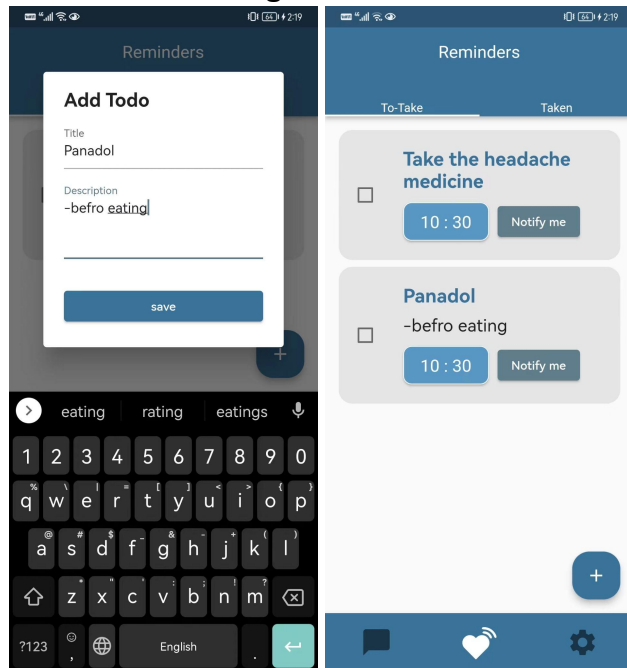
3.2.10 F10: Reminders main page



The reminder main page contain two tabs (to-take and taken) each one of them contains a Reminder card

```
bottom: TabBar(controller: controller, tabs: [
  Text(
    'To-Take',
    style: CustomTextStyle.style(
      fontSize: 14,
      fontWeight: FontWeight.normal,
      color: Colors.white,
    ),
  ), // Text
  Text(
    'Taken',
    style: CustomTextStyle.style(
      fontSize: 14,
      fontWeight: FontWeight.normal,
      color: Colors.white,
    ),
  ), // Text
])
```

3.2.11 F11: Adding a new reminder



the buildTodo() function is the one that creates the reminders and has all of the data you can interact with such as the title, description, and time

```
Widget buildTodo(BuildContext context) {  
  final hours = time.hour.toString().padLeft(2, '0');  
  final minutes = time.minute.toString().padLeft(2, '0');  
  return GestureDetector(  
    onTap: () => editTodo(context, widget.todo),  
    child: Container(  
      color: const Color(0xFFE5E5E5),  
      padding: const EdgeInsets.symmetric(horizontal: 7, vertical: 21),  
    ),  
  );  
}
```

the check box that indicates if the medicine reminder has been taken or not, and based of that it will either keep the reminder in the “To-Take” tab or in the “Taken” tab

```
Checkbox(
  activeColor: Theme.of(context).primaryColor,
  checkColor: Colors.white,
  value: widget.todo.isDone,
  onChanged: (_) {
    final provider =
      Provider.of<TodosProvider>(context, listen: false);
    final isDone = provider.toggleTodoStatus(widget.todo);

    Utils.showSnackBar(
      context,
      isDone ? 'Task completed' : 'Task marked incomplete',
    );
  },
), // Checkbox
```

and this is the dialog that pops up when pressing the + sign to add a new reminder the buildTitle() function takes the title from the user and it also checks if the title is empty or not

```
Widget buildTitle() {
  return TextFormField(
    maxLines: 1,
    initialValue: title,
    onChanged: onChangedTitle,
    validator: (title) {
      if (title!.isEmpty) {
        return 'The title cannot be empty';
      }
      return null;
    },
    decoration: const InputDecoration(
      border: UnderlineInputBorder(),
      labelText: 'Title',
    ), // InputDecoration
  ); // TextFormField
}
```

the same thing applies to the description field with the buildDescription() function

```
buildDescription() {  
  return TextFormField(  
    maxLines: 3,  
    initialValue: description,  
    onChanged: onChangedDescription,  
    decoration: const InputDecoration(  
      border: UnderlineInputBorder(),  
      labelText: 'Description',  
    ), // InputDecoration  
  ); // TextFormField  
}
```

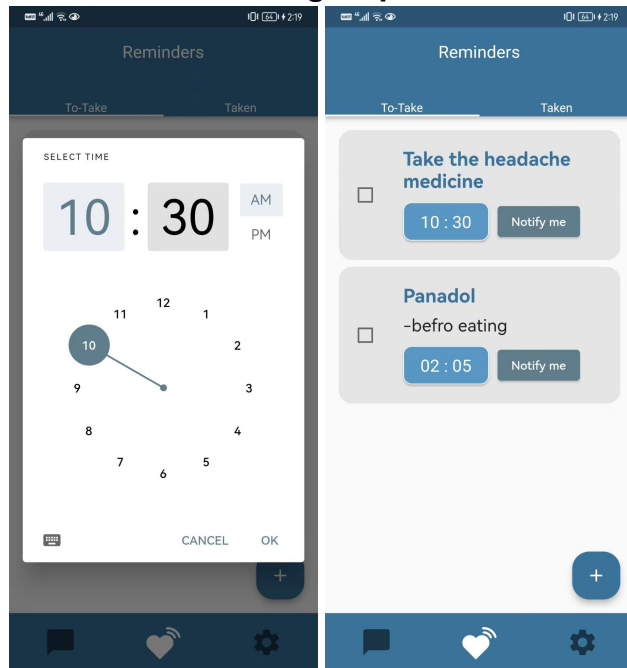
the buildButton() function is the one that takes all of this data and saves it so the reminder can be displayed later on the main reminder page

```
buildButton() {  
  return SizedBox(  
    width: double.infinity,  
    child: ElevatedButton(  
      onPressed: onSaveTodo,  
      child: const Text('save'),  
      style: ButtonStyle(  
        backgroundColor: MaterialStateProperty.all(const Color(0xFF39729A)),  
      ), // ButtonStyle  
    ), // ElevatedButton  
  ); // SizedBox  
}
```

NOTE: In order to be able to use all of these data across the app in different pages we used what it's called a Provider, it's job is to edit, delete add and toggle the reminders across the whole app, it's in the todos_provider.dart file

```
void addTodo(Todo todo) {  
  _todos.add(todo);  
  notifyListeners();  
}  
  
void removeTodo(Todo todo) {  
  _todos.remove(todo);  
  notifyListeners();  
}  
  
bool toggleTodoStatus(Todo todo) {  
  todo.isDone = !todo.isDone;  
  notifyListeners();  
}
```


3.2.12 F12: Choosing a specific time

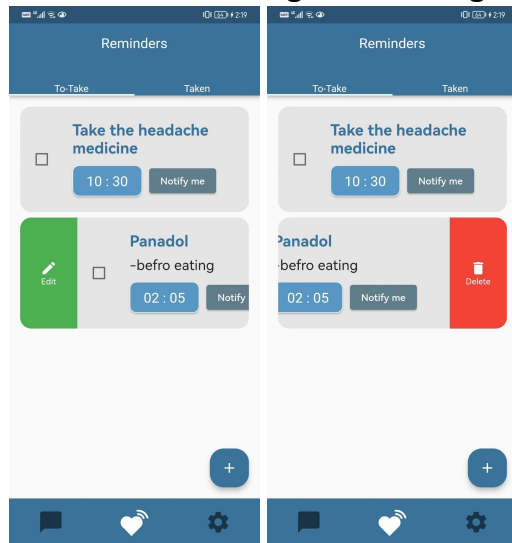


when pressing on the light blue button that has the time on it, it will show the time picker where you can choose the time you want to be reminded at, after choosing the time you want you can press on OK to set the time and also display it on the light blue button and if you pressed on Cancel it won't assign any time to the blue button

```
onPressed: () async {
  TimeOfDay? newTime = await showTimePicker(
    context: context,
    initialTime: time,
  );

  //?if 'CANCLE' => null
  if (newTime == null) return;
  //? if 'OK' => TimeOfDay
  setState(() => time = newTime);
},
```

3.2.13 F13: Deleting and Editing slidable options



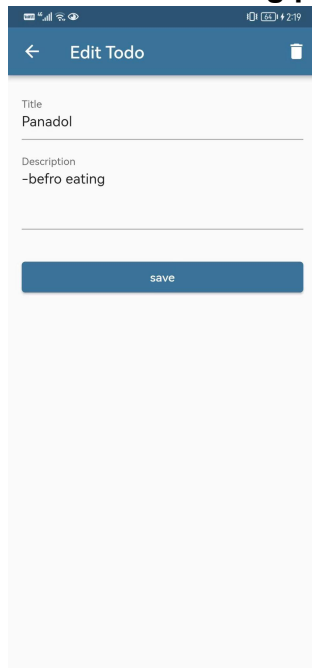
The Slidable widget is the one handling this slidable actions, thanks to the slidable package

```
child: Slidable(  
  actionPane: const SlidableDrawerActionPane(),  
  key: Key(widget.todo.id.toString()),  
  actions: [  
    IconSlideAction(  
      color: Colors.green,  
      onTap: () => editTodo(context, widget.todo),  
      caption: 'Edit',  
      icon: Icons.edit,  
    ) // IconSlideAction  
  ],  
  secondaryActions: [  
    IconSlideAction(  
      color: Colors.red,  
      caption: 'Delete',  
      onTap: () => deleteTodo(context, widget.todo),  
      icon: Icons.delete,  
    ) // IconSlideAction  
  ],  
  child: buildTodo(context),  
), // Slidable
```

The Slidable package

```
provider: ^6.0.1
firebase_core_web: ^1.6.2
google_fonts: ^2.3.2
flutter_slidable: ^0.5.7
flutter_local_notifications: ^6.1.0
```

3.2.14 F14: Editing page



← Edit Todo

Title
Panadol

Description
-befro eating

save

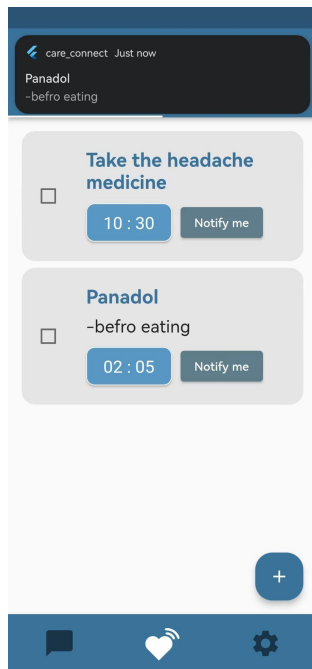
here you can put your new title and description

```
child: TodoFormWidget(  
  title: title,  
  description: description,  
  onChangedTitle: (title) => setState(() => this.title = title),  
  onChangedDescription: (description) =>  
    setState(() => this.description = description),  
  onSaveedTodo: saveTodo,  
), // TodoFormWidget
```

the `saveTodo()` function will save the new details of the reminder using the `updateTodo()` function that we have created in the provider

```
void saveTodo() {  
  final isValid = _formKey.currentState!.validate();  
  
  if (!isValid) {  
    return;  
  } else {  
    final provider = Provider.of<TodosProvider>(context, listen: false);  
  
    provider.updateTodo(widget.todo, title, description);  
  
    Navigator.of(context).pop();  
  }  
}
```

3.2.15 F15: Notifications



When you press on the “Notify me” it will trigger the `showNotification()` function that we have created in the `notification_api.dart` file

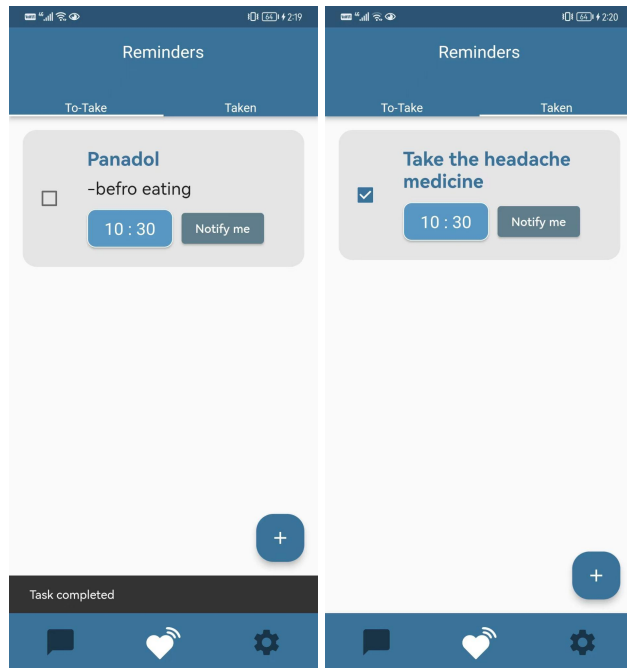
```
ElevatedButton(  
  onPressed: ()=> NotificationApi.showNotification(  
    title: widget.todo.title,  
    body: widget.todo.description,  
    payload: '',  
  ),  
  child: const Text("Notify me"),  
), // ElevatedButton
```

we are able to create such local notifications in a really easy way because we are using the notifications package

```
static Future showNotification({  
  int id = 0,  
  String? title,  
  String? body,  
  String? payload,  
  required DateTime scheduledDate,  
}) async =>  
  _notifications.zonedSchedule(  
    id,  
    title,  
    body,  
    tz.TZDateTime.from(scheduledDate, tz.local),  
    await _notificationDetails(),  
    payload: payload,  
    androidAllowWhileIdle: true,  
    uiLocalNotificationDateInterpretation:  
      UILocalNotificationDateInterpretation.absoluteTime,  
  );
```

```
firebase_core_web: ^1.6.2  
google_fonts: ^2.3.2  
flutter_slidable: ^0.5.7  
flutter_local_notifications: ^6.1.0
```

3.2.16 F16: Taken the medicine



when you press on the checkbox it will update the status of the reminder and will show a little snack bar message “Task Completed” and move the reminder to the “Taken” tab using the `toggleTodoStatus()` function that we have created in the provider

```
onChanged: (_) {  
  final provider =  
    Provider.of<TodosProvider>(context, listen: false);  
  final isDone = provider.toggleTodoStatus(widget.todo);  
  
  Utils.showSnackBar(  
    context,  
    isDone ? 'Task completed' : 'Task marked incomplete',  
  );  
},
```

4 Other Non-functional Requirements

4.1 Performance Requirements

The App should be Fast, Safe, Smooth, and Secure. The support team should take feedbacks every Friday and then work on these feedbacks and solve them before the next Friday

4.2 Safety and Security Requirements

Firebase provides the security and safety of the app since the app is hosted in the cloud and stores all of its data in the cloud-based database therefor the security and safety are going to be available from Firebase and Google themselves and the Firebase Security Rules can achieve that. Firebase Security Rules work by matching a pattern against database paths and then applying custom conditions to allow access to data at those paths. All Rules across Firebase products have a path-matching component and a conditional statement allowing read or write access.

The Full Project:

You can get the full project from the GitHub repo:

<https://github.com/Y-A-H-Y-A/Care-Connect.git>

Team member's contributions

Student ID	Name	Task	Details
20193290616	Yahya Isaac (洛克)	Coding, Designing & Project Management	<ul style="list-style-type: none">● Coding and designing the Reminder section of the project● Project management and planning of the tasks for each member of the group● Setting up the custom fonts, icons, and colors for the app● Designing the reminder section in Figma
20193290693	Ali Sharh (阿里)	Database Testing, Coding & Designing	<ul style="list-style-type: none">● Testing the functionality of the database and testing its distribution and the workflow of the data● Designing the chatting section of the app and the registration pages in Figma● Worked in the notification system and API
20193290682	Salem Al-Mahdi (阿德)	Coding & Database Management	<ul style="list-style-type: none">● Coding the Chatting section of the project● Creating the database and moderate its collections and documents, Also implementing the data encryption in the database● Connecting the database

			with the Sign in & Sign up pages and the authentication
20193290702	Mouaid Shaban (米度)	App Testing, Coding & Documentation	<ul style="list-style-type: none"> ● Testing the functionalities of the app ● Writing all of the documents that are related to the project including (reports, SRS files & PPTs) ● Worked in the Registration part of the app
20193290706	Mohammed Al-Kazrajy (莫汉)	App Testing, Coding & Documentation	<ul style="list-style-type: none"> ● Testing the functionalities of the app ● Writing all of the documents that are related to the project including (reports, SRS files & PPTs) ● Worked in the Authentication part of the app