Firebase Demo Project

When writing mobile apps, saving the data users input is critical. You can save the data locally, but then you can’t save it permanently or share it with others.

If you want to share your data, you need to save it in the cloud. One of the easiest solutions is Google’s **Firebase Firestore** database.

Firestore is a NoSQL style database. Unlike traditional table-based databases that require much work to insert data, you save JSON blobs to collections. Additionally, this method doesn’t impose a structure for your data and lets you change it at any time.

Of course, the downside is that you have to take that into account when reading the data. So, you need to plan how you want to store your data. Firestore offers fast, responsive data syncing without much code.

you’ll build PetMedical, an app to store your pets’ medical records. The app is a simple list of pets on the first screen with a floating action button at the bottom to add a new record.

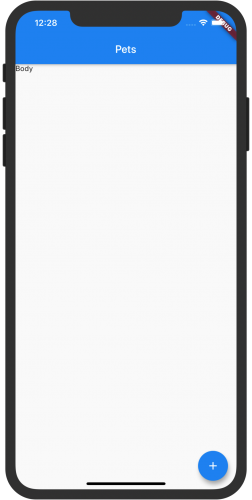
In this tutorial you will learn how to:

* Create a new Firebase project.
* Configure a Firestore database.
* Use and create collections.
* Add Firebase dependencies to your Flutter project.
* Use streams.
* Create models and repositories.
* And much more! overview of the basics of working with this SDK.

**Getting Started**

Download the begin project by using the **Download Materials** button at the top or bottom of the page. Open **pubspec.yaml** and click on **Packages get** to download the latest libraries.

Open either an iPhone simulator or an Android emulator and make sure the app runs. The app should like like this:

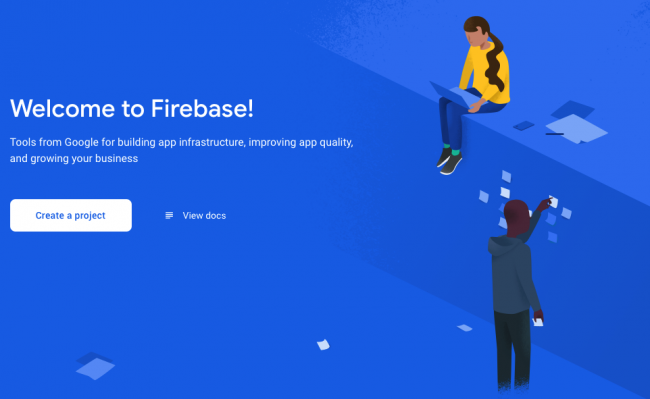
[](https://koenig-media.raywenderlich.com/uploads/2020/01/initial_startup.png)

The UI is in place, but you’ll need to setup Firestore and add the code to add and update records.

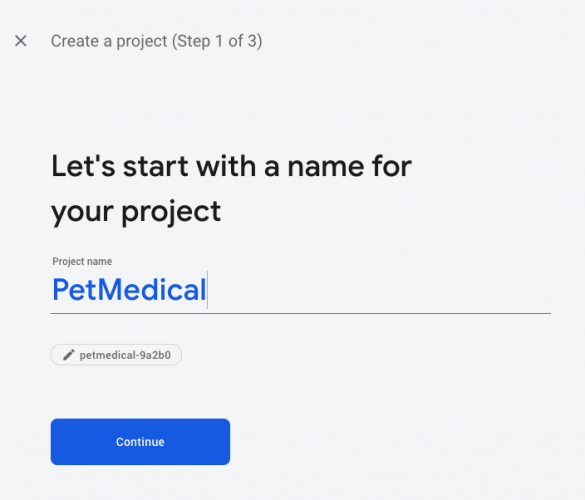
**Creating a Firebase Account**

In order to have a Firestore database, you need a Firebase account. Go to <https://firebase.google.com/> and sign up for an account.

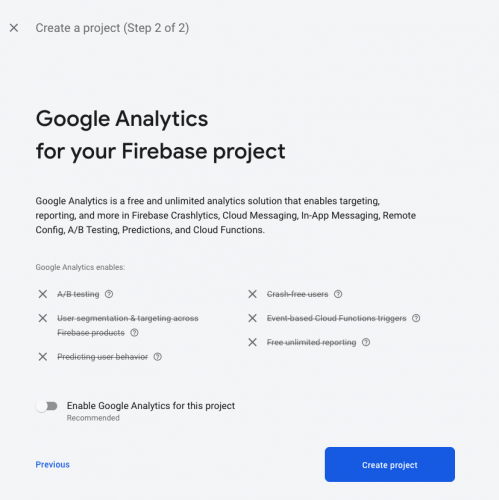
On the Welcome to Firebase page, click the **Create Project** button.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup1-1.png)

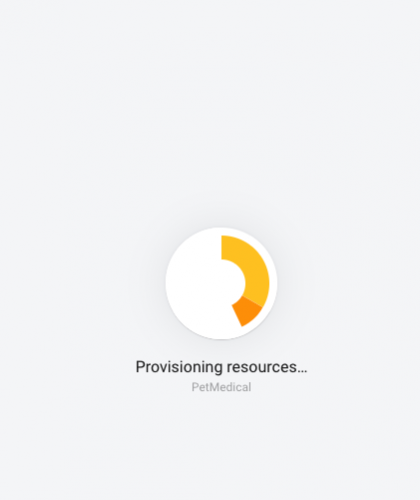
Now, enter the project name: PetMedical. Select the terms checkbox and press the **Continue** button.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup1.png)

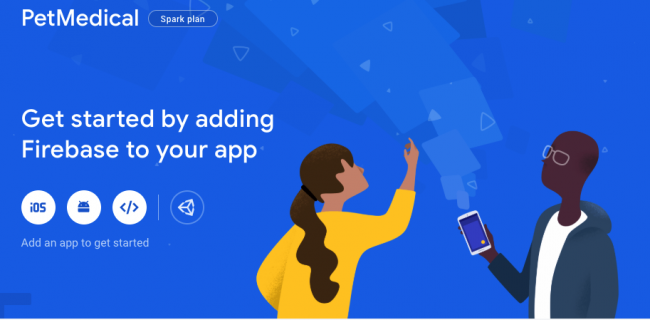
On the next page, click on the switch to **disable Analytics** as you won’t use it. Then, click **Create Project**.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup2.png)

You’ll see a few progress dialogs:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup3.png)

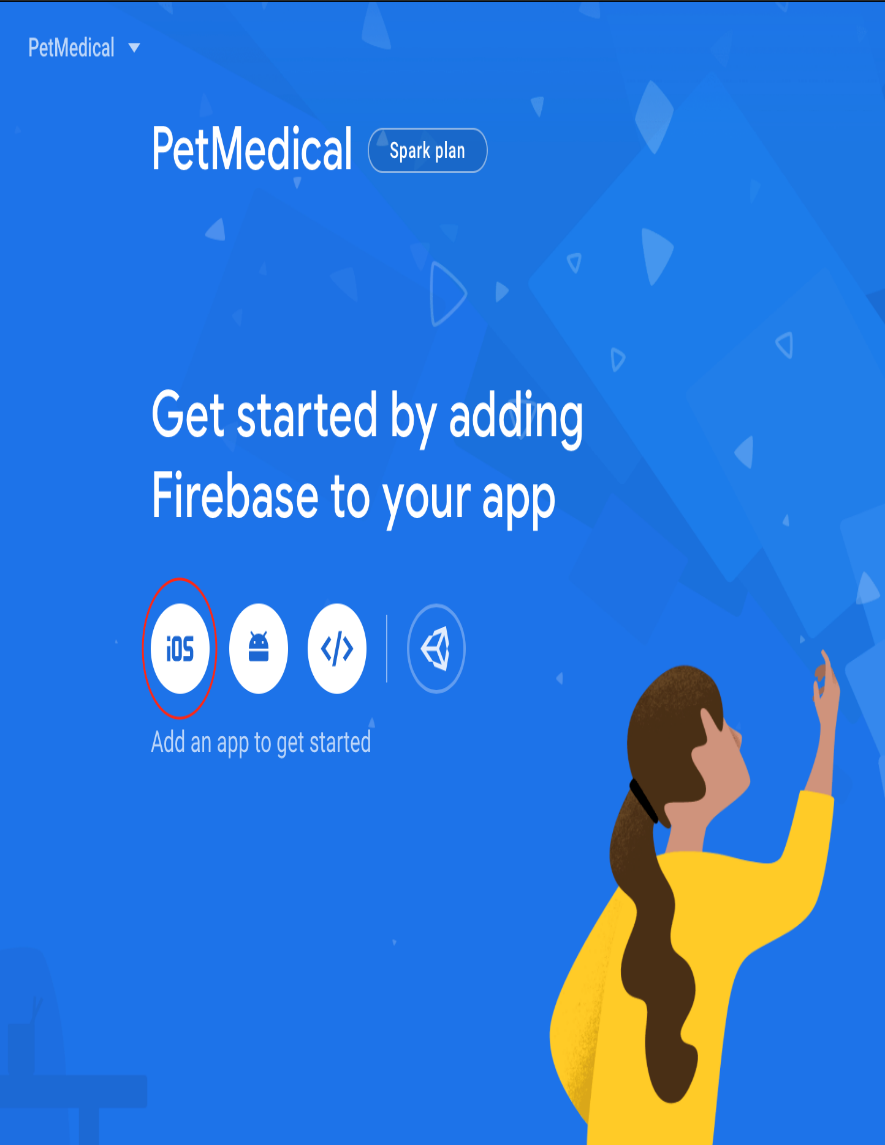
Once your new project is ready, press **Continue** to get to the Getting Started page:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup5.png)

Here you can add Firebase to both your iOS and Android apps. Start with the iOS app.

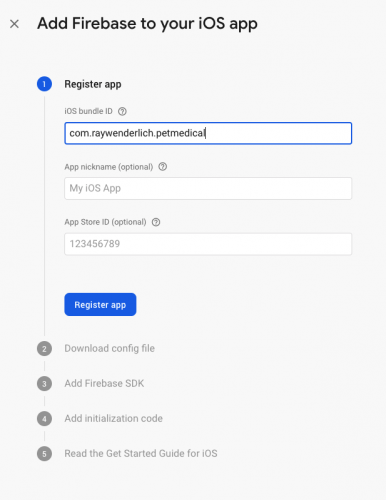
**Registering an iOS App**

To register the iOS app, click on the iOS circle:

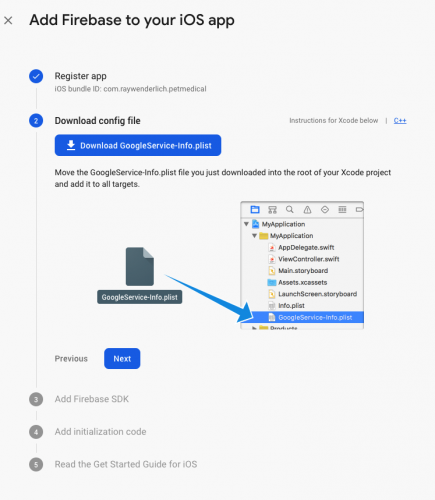
[](https://koenig-media.raywenderlich.com/uploads/2020/01/Captura-de-Tela-2020-01-14-%C3%A0s-21.19.35.png)

You’ll see a dialog to register your app. Enter **com.raywenderlich.petmedical** for the iOS bundle id and click on the **Register App** button.

**Note**: If you created the Flutter app from scratch, enter the bundle id you used to create the app.

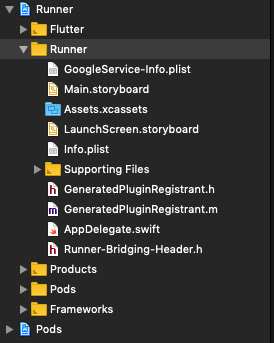
[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup6.png)

Next, click the **Download GoogleService-Info.plist** button.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup6-1.png)

Now, move this file into the **iOS ‣ Runner** folder. Then, from Android Studio in the **Tools ‣ Flutter** menu, choose **Open iOS module in Xcode**. In Xcode, right-click the **Runner** folder and choose **Add files to Runnner…**.

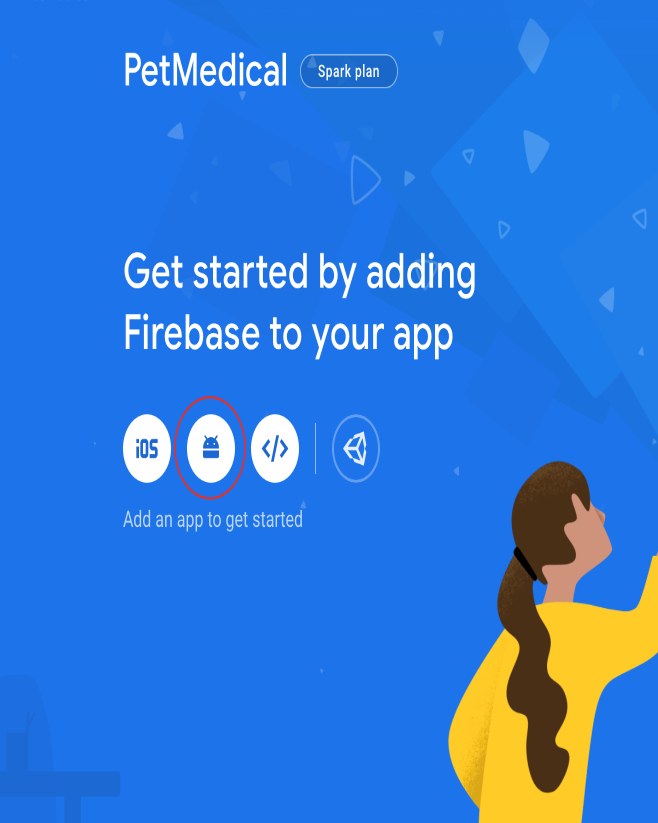
Next, add **GoogleService-Info.plist**:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/xcode1.png)

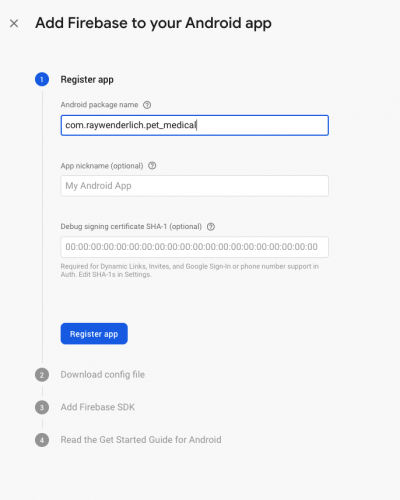
Nice job! Now it’s time to register the Android app. :]

**Registering an Android App**

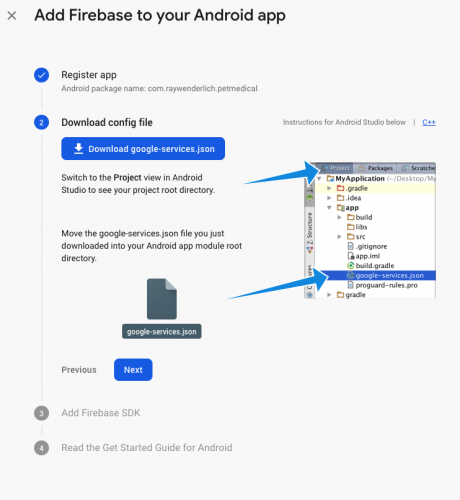
First, go back to the Firebase page. On the main page click the **Android** circle to start the process of adding Firebase to Android:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/Captura-de-Tela-2020-01-14-%C3%A0s-21.20.12.png)

You’ll see a dialog to register your app. Enter **com.raywenderlich.pet\_medical** in the Android package name. Next, click **Register App**:

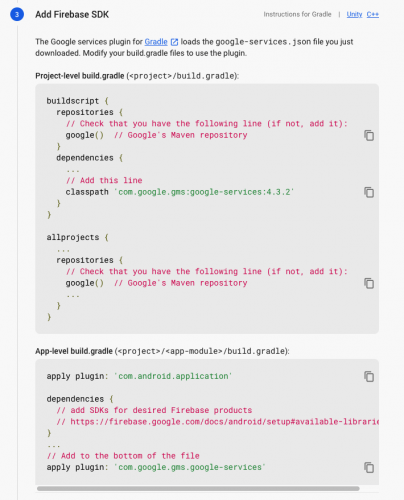
[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup7.png)

Then, click the **Download google-services.json** button. In the Finder, move this file into the **android ‣ app** folder.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup7-1.png)

Now, in Android Studio, open the Android folder and then open **build.gradle**. Then add classpath 'com.google.gms:google-services:4.3.3' after the last classpath entry.

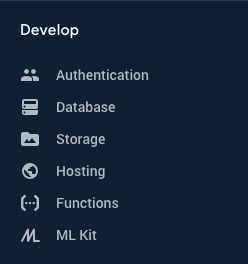
Now open the app **build.gradle** and add apply plugin: 'com.google.gms.google-services' to the bottom.

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_setup7-2.png)

Not too bad, right? :] Congrats! Now it’s time to create the Firebase database.

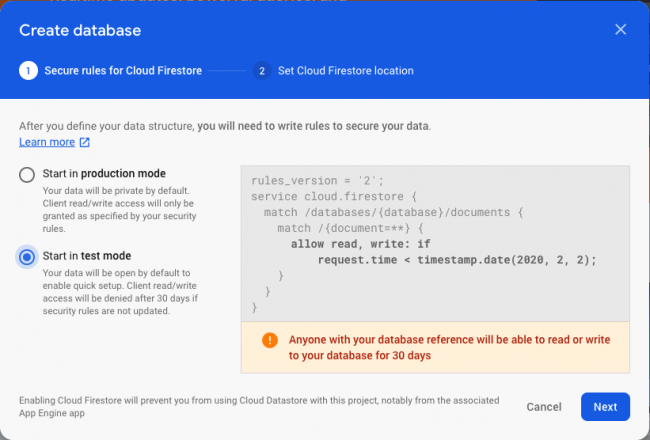
**Creating Firestore Database**

On the Firebase console choose the **Database** option under the Develop menu:

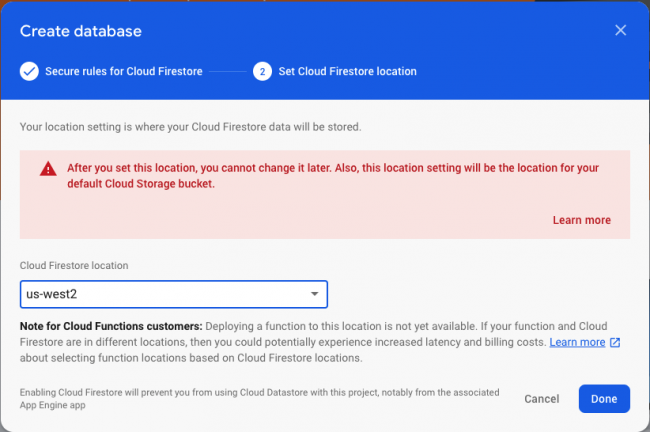
[](https://koenig-media.raywenderlich.com/uploads/2020/01/firebase_console1.png)

Now click the **Create Database** button and choose **Start in test mode**. This turns off any security so you can easily test your database:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/create_database.png)

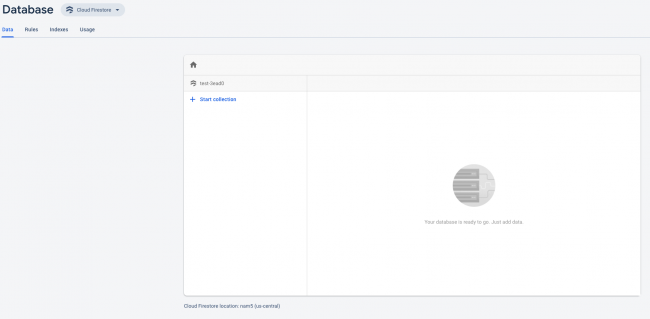
[](https://koenig-media.raywenderlich.com/uploads/2020/01/create_database2.png)

When you’re ready for production, change the setting back to production mode and add security rules. Now, click **Next**. Then choose a Firestore location and click **Done**:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/create_database3.png)

Nice! You created your first database.

Your screen won’t have any collections to start with:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/create_database4.png)

In Android Studio open **pubspec.yaml** and add cloud\_firestore: ^0.13.0+1 after flutter\_form\_builder: ^3.7.2. Then click **Packages get** to add the firestore library.

Before creating the model class, it’s time to talk about collections. :]

**Understanding Collections**

Firestore stores data in collections, which are similar to tables in a traditional database. They have a name and a list of **Documents**.

These **Documents** usually have a unique generated key in the database and they store data in key/value pairs.

These fields can have several different types:

* String.
* Number.
* Boolean.
* Map.
* Array.
* Null.
* Timestamp.
* Geopoint.
* Reference to another document.

You can use Firestore’s console to manually enter data and see the data appear almost immediately in your app. If you enter data in your app, you’ll see it appear on the web and other apps almost immediately.

Next, you’ll create the models for your app.

**Creating the Model Classes**

To retrieve your data from Firestore, you need to create two model classes where you’ll put the data: Vaccinations and pets.

In Android Studio, right-click the lib directory and select **New ‣ Directory**. Name the directory **models**. You’ll create the vaccinations model first.

**Creating the Vaccination Model**

First, right-click on the models folder and choose **New ‣ Dart File**. Then name the file **vaccination** and add the following:

class Vaccination {

// 1

String vaccination;

DateTime date;

bool done;

// 2

DocumentReference reference;

// 3

Vaccination(this.vaccination, {this.date, this.done, this.reference});

// 4

factory Vaccination.fromJson(Map<dynamic, dynamic> json) => \_VaccinationFromJson(json);

// 5

Map<String, dynamic> toJson() => \_VaccinationToJson(this);

@override

String toString() => "Vaccination<$vaccination>";

}

Here are descriptions of the comments above:

1. Define your fields. Name of the vaccination, date it was given and whether this vaccination is finished.
2. A reference to a Firestore document representing this vaccination.
3. Constructor where the vaccination is required and the others are optional.
4. A factory constructor to create a Vaccination from JSON.
5. Turn this vaccination into a map of key/value pairs.

Now add the helper functions outside the class:

//1

Vaccination \_VaccinationFromJson(Map<dynamic, dynamic> json) {

return Vaccination(

json['vaccination'] as String,

date: json['date'] == null ? null : (json['date'] as Timestamp).toDate(),

done: json['done'] as bool,

);

}

//2

Map<String, dynamic> \_VaccinationToJson(Vaccination instance) =>

<String, dynamic> {

'vaccination': instance.vaccination,

'date': instance.date,

'done': instance.done,

};

Here’s what you see:

1. **\_VaccinationFromJson** turns a map of values from Firestore into a Vaccination class.
2. **\_VaccinationToJson** converts the Vaccination class into a map of key/value pairs.

Next, you’ll create the pets model.

**Creating the Pet Model**

Now, right-click the models folder and choose **New ‣ Dart File**. Name the file **pets**. Add the following:

class Pet {

// 1

String name;

String notes;

String type;

// 2

List<Vaccination> vaccinations = List<Vaccination>();

// 3

DocumentReference reference;

// 4

Pet(this.name, {this.notes, this.type, this.reference, this.vaccinations});

// 5

factory Pet.fromSnapshot(DocumentSnapshot snapshot) {

Pet newPet = Pet.fromJson(snapshot.data);

newPet.reference = snapshot.reference;

return newPet;

}

// 6

factory Pet.fromJson(Map<String, dynamic> json) => \_PetFromJson(json);

// 7

Map<String, dynamic> toJson() => \_PetToJson(this);

@override

String toString() => "Pet<$name>";

}

Here you have:

1. Define your fields. Name of the pet, notes and the type of pet.
2. List of vaccinations for this pet.
3. A reference to a Firestore document representing this pet.
4. Constructor that pet name is required, the others are optional.
5. A factory constructor to create a Pet from a Firestore DocumentSnapshot. You want to save the reference for updating later.
6. A factory constructor to create a Pet from JSON.
7. Turn this pet into a map of key/value pairs.

Next, below the class add:

// 1

Pet \_PetFromJson(Map<String, dynamic> json) {

return Pet(

json['name'] as String,

notes: json['notes'] as String,

type: json['type'] as String,

vaccinations: \_convertVaccinations(json['vaccinations'] as List)

);

}

// 2

List<Vaccination> \_convertVaccinations(List vaccinationMap) {

if (vaccinationMap == null) {

return null;

}

List<Vaccination> vaccinations = List<Vaccination>();

vaccinationMap.forEach((value) {

vaccinations.add(Vaccination.fromJson(value));

});

return vaccinations;

}

// 3

Map<String, dynamic> \_PetToJson(Pet instance) => <String, dynamic> {

'name': instance.name,

'notes': instance.notes,

'type': instance.type,

'vaccinations': \_VaccinationList(instance.vaccinations),

};

// 4

List<Map<String, dynamic>> \_VaccinationList(List<Vaccination> vaccinations) {

if (vaccinations == null) {

return null;

}

List<Map<String, dynamic>> vaccinationMap =List<Map<String, dynamic>>();

vaccinations.forEach((vaccination) {

vaccinationMap.add(vaccination.toJson());

});

return vaccinationMap;

}

Here you:

1. Add a function to convert a map of key/value pairs into a Pet.
2. Add another function to convert a list of maps into a list of vaccinations.
3. Convert a Pet into a map of key/value pairs.
4. Convert a list of vaccinations into a list of mapped values.

Now that you’ve added the classes to hold your data, you need to add a way to retrieve and save it.

**Creating a DataRepository Class**

Next, you’ll create a DataRepository class, which retrieves and saves your data. You need to isolate your usage of Firebase as much as possible to follow Android best practices.

First, right-click the lib directory and select **New ‣ Directory**. Then name the directory **repository**.

Next, right-click the models folder and choose **New ‣ Dart File**. Name the file **dataRepository** and add the following:

class DataRepository {

// 1

final CollectionReference collection = Firestore.instance.collection('pets');

// 2

Stream<QuerySnapshot> getStream() {

return collection.snapshots();

}

// 3

Future<DocumentReference> addPet(Pet pet) {

return collection.add(pet.toJson());

}

// 4

updatePet(Pet pet) async {

await collection.document(pet.reference.documentID).updateData(pet.toJson());

}

}

Here’s what you added:

1. Your top level collection is called pets. Store a reference to this.
2. Use the snapshots method to get a stream of snapshots. This listens for updates automatically.
3. Add a new pet. This returns a Future if you want to wait for the result. Note that add will automatically create a new document id for Pet.
4. Update your pet class.

You’ve added classes to hold, retrieve and save your data. Now you need to add a way to update your lists when different uses add new data.

**Using Streams**

Streams are a sequence of asynchronous data that sends when ready. Firestore sends updates to your list of pets when someone else adds or modifies a pet.

You’ll use a stream in **main.dart** to listen for the list of pets. When a user adds a new pet or updates a pet, the stream redraws the list with the updated information.

**Add DataRepository to Main**

First, open **main.dart**. There are //TODO... items throughout the code where you need to add your pet and data repository code.

In \_HomeListState add:

final DataRepository repository = DataRepository();

This gives access to Firestore throughout this class. Then, in the \_buildHome, replace body with:

body: StreamBuilder<QuerySnapshot>(

stream: repository.getStream(),

builder: (context, snapshot) {

if (!snapshot.hasData) return LinearProgressIndicator();

return \_buildList(context, snapshot.data.documents);

}),

The **StreamBuilder** first checks to see if you have any data. If not, it’ll show a progress indicator. Otherwise, it’ll call \_buildList.

Now go to \_buildList and replace children: [] with:

children: snapshot.map((data) => \_buildListItem(context, data)).toList(),

This code maps the list from data, creates a new list item for each one and turns that into a list that the children parameter needs.

Next, go to the //TODO Add New Pet to repository above \_buildList in the onPressed() and add:

Pet newPet = Pet(dialogWidget.petName, type: dialogWidget.character);

repository.addPet(newPet);

This code creates a new Pet class and uses the repository to add the new Pet. When you build and run the app, notice that the list automatically updates without you having to write any code for it.

Now, go to \_buildListItem and change the method to:

Widget \_buildListItem(BuildContext context, DocumentSnapshot snapshot) {

This code adds the snapshot parameter.

Next, go to // TODO Get Pet from snapshot and add:

final pet = Pet.fromSnapshot(snapshot);

if (pet == null) {

return Container();

}

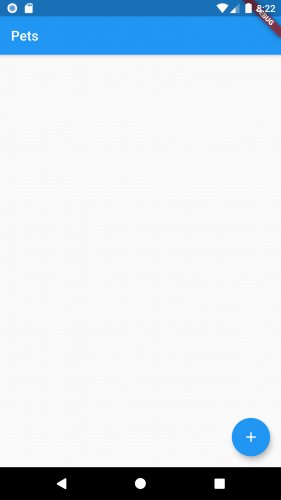
This creates a Pet class from the snapshot passed in. Do a null check to make sure it created a pet.

Now, go to // TODO add pet name and show the pet name replacing the expanded widget with:

Expanded(child: Text(pet.name == null ? "" : pet.name, style: BoldStyle)),

\_getPetIcon(pet.type)

Build and run the app to make sure it compiles.

[](https://koenig-media.raywenderlich.com/uploads/2020/02/Screenshot_1580610165.png)

Try clicking the floating action button to enter a pet name and type. Press add and make sure a new item appears.

If you see any errors, make sure you have the Firestore database setup and that you added all of your Google files.

Next go to // TODO add pet and pass your pet class to PetDetails:

builder: (context) => PetDetails(pet),

Nice job! Now it’s time for the Pet Detail screen.

**Building the Pet Detail Page**

First, open PetDetails.dart and add the pet field and constructor:

final Pet pet;

const PetDetails(this.pet);

Then import the Pet class and change the title to:

title: Text(pet.name== null ? "" : pet.name),

Next, add the pet field and constructor to **PetDetailForm** class:

final Pet pet;

const PetDetailForm(this.pet);

Then, in \_PetDetailFormState.initState, set type to the pet’s type:

type = widget.pet.type;

Next, in the build method, find the first initialValue and replace it with:

initialValue: widget.pet.name,

This build method uses the FormBuilder library to create a column of form fields that have validation built in. You can find more information at: [Flutter FormBuilder](https://pub.dev/packages/flutter_form_builder) .

Next, in the notes field, replace the initialvalue with:

initialValue: widget.pet.notes,

Now find the FormBuilderCustomField entry. Go to // TODO use vaccination count and // TODO Pass in vaccination and replace that code with:

itemCount: widget.pet.vaccinations == null ? 0 : widget.pet.vaccinations.length, itemBuilder: (BuildContext context, int index) {

return buildRow(widget.pet.vaccinations[index]);

},

This uses the vaccination list to get the count. It creates a new row for each vaccination.

Next, in the FloatingActionButton section, replace \_addVaccination with:

\_addVaccination(widget.pet, () {

You want to pass in the pet, so you’ll need to update this method later.

Now go to the next // TODO Update widget and add:

widget.pet.name = name;

widget.pet.type = type;

widget.pet.notes = notes;

repository.updatePet(widget.pet);

This code uses the variables set in the fields and updates the pet. It also updates the stream, so when you return to the list, it is already updated.

Next, replace buildRow() with:

Widget buildRow(Vaccination vaccination) {

Then, import Vaccination if it hasn’t already imported. Replace the rest of the method with:

return Row(

children: <Widget>[

Expanded(

flex: 1,

child: Text(vaccination.vaccination),

),

Text(vaccination.date == null ? "" : dateFormat.format(vaccination.date)),

Checkbox(

value: vaccination.done == null ? false : vaccination.done,

onChanged: (newValue) {

vaccination.done = newValue;

},

)

],

);

This creates a row with the vaccination name, date and checkbox.

Now update \_addVaccination to take a pet:

void \_addVaccination(Pet pet, DialogCallback callback) {

Finally, go to the very bottom of the file and add the following (replace TODO):

Vaccination newVaccination = Vaccination(vaccination, date: vaccinationDate, done: done);

if (pet.vaccinations == null) {

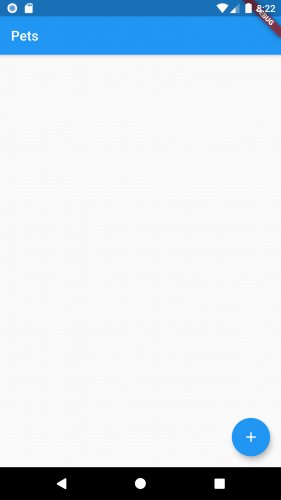
pet.vaccinations = List<Vaccination>();

}

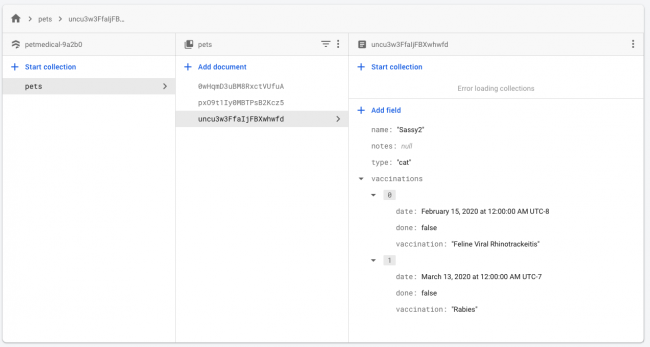
pet.vaccinations.add(newVaccination);

This creates a new Vaccination and adds it to your vaccination list. It uses the callback so the caller can update the state and have the UI update.

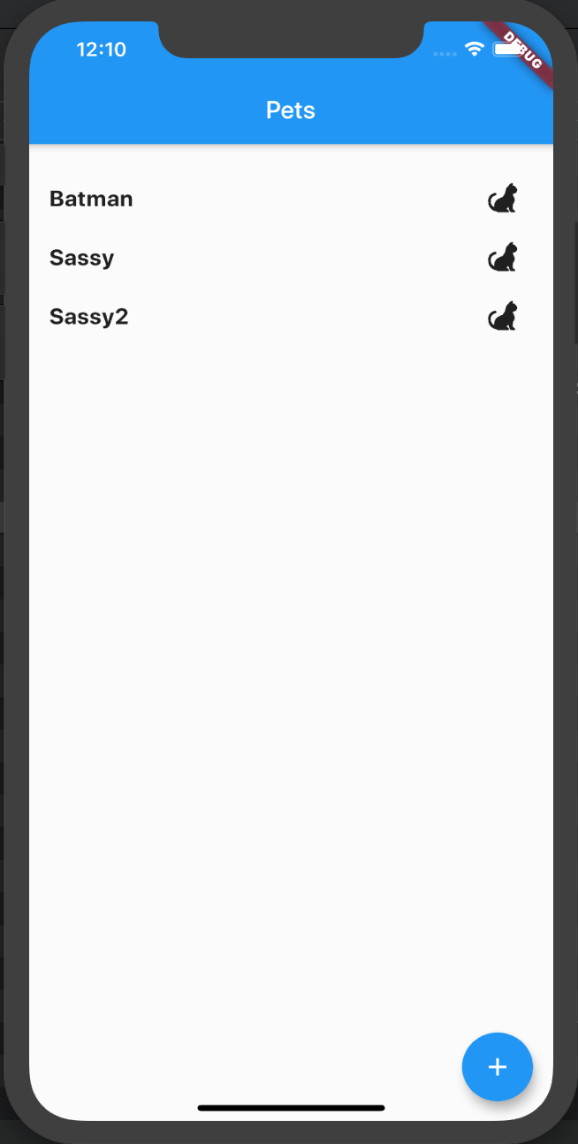
Build and run the app in either iOS or Android and make sure everything works. Don’t assume that both work as they have different Firestore setups.

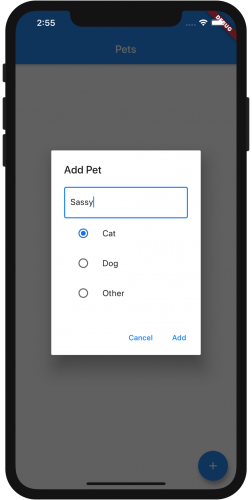
[](https://koenig-media.raywenderlich.com/uploads/2020/02/Screenshot_1580610165.png)

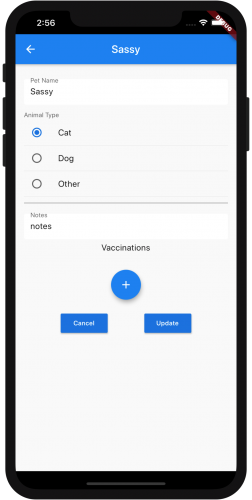
Try adding any pets you have as well as any vaccinations and check the Firestore console to see what the data looks like. This is an example of some data in Firestore:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/firestore1.png)

And some images from the app:

[](https://koenig-media.raywenderlich.com/uploads/2020/01/Captura-de-Tela-2020-01-16-%C3%A0s-00.10.17.png)

[](https://koenig-media.raywenderlich.com/uploads/2020/01/iOS2.png)

[](https://koenig-media.raywenderlich.com/uploads/2020/01/iOS4.png)

Congratulations! You created both an iOS and an Android app that uses the Firestore database!

This url is my own

https://console.firebase.google.com/u/0/project/petmedical-9deaf/firestore/data~2Fpets~2FAi1kdJ3a2wv0wknj4HfF