# Phone authentication setup with Flutter

## Introduction

Phone authentication is one of the most reliable authentication methods for mobile applications. And Appwrite has now introduced phone authentication in our recent release. In this article, we will learn how to implement Appwrite’s phone authentication in our Flutter applications.

**New to Appwrite?**

[Appwrite](http://appwrite.io/) is an open-source back-end-as-a-service that abstracts all the complexity of building a modern application by providing a set of REST and Realtime APIs for your core back-end needs. Appwrite minimizes the heavy lifting for developers and handles user authentication and authorization, databases, file storage, cloud functions, webhooks, and much more!

## Prerequisites

* Access to an Appwrite project or permission to create one. If you don’t already have an Appwrite server running, follow the official [installation tutorial](https://appwrite.io/docs/installation) for setup instructions.
* A [Flutter](https://flutter.dev)-configured development machine.

## Twilio Account Setup

We first need to set up a phone service provider for authentication. Twilio is a popular choice – a Twilio account can provide API access keys, allowing Appwrite to send codes to mobile phones. If you don’t have an existing Twilio account, you can visit <https://www.twilio.com/try-twilio> and create a free account using your email and password. Via email, you will then receive a verification link, clicking on which will verify your Twilio account. It will also ask you to verify your phone number.

Enter your phone number and click on the **Verify** button. You will then receive a code via SMS. Enter the received code in Twilio and click **Submit** to confirm your number.

On the next screen, enter your choices. It doesn’t matter which coding language you choose, as we will integrate Twilio with Appwrite, so we don’t have to interface with Twilio directly. Next, click on **Get started with Twilio** button.



You should then be redirected to the Twilio console. Click the **Getting your Twilio number** button. The initial trial account can only send texts and calls to your verified numbers, and for testing, that’s perfect. However, for production use, you must upgrade your Twilio account.



You can see the Account SID, Auth Token, and Phone number in the console. You will need these in the next session to set up Twillio as an SMS provider Appwrite can use. Your Twilio account should now be ready to integrate with Appwrite.

## Appwrite Setup

Follow the [official guide](https://appwrite.io/docs/installation) to set up and run Appwrite. For Appwrite to know about Appwrite, we need to open the `.env` file from the appwrite directory where you installed Appwrite, and update the following environment variables.

\_APP\_PHONE\_PROVIDER=twilio

\_APP\_PHONE\_USER=<ACCOUNT\_SID\_FROM\_TWILIO\_CONSOLE>

\_APP\_PHONE\_SECRET=<AUTH\_TOKEN\_FROM\_TWILIO\_CONSOLE>

\_APP\_PHONE\_FROM=<TWILIO\_PHONE\_NUMBER>

Once the environment variables are updated, run the following command from the command line in the installation directory that contains the `.env` and `docker-compose.yml` files.

docker-compose up -d

This will restart the containers, providing new values for the environment variables, so Appwrite can connect to Twilio. We are now ready to implement phone authentication in our application.

**Install and Configure Appwrite’s Flutter SDK**

Cloning the project we created for this tutorial lets us skip the setup and UI part, and focus on the main topic. Clone <https://github.com/lohanidamodar/appwrite_phone_auth.git>. Once cloned, open it in your favorite IDE.

We will start by adding Appwrite’s SDK as a dependency. Open pubspec.yaml file and add Appwrite’s SDK under dependencies.

dependencies:

 appwrite: ^6.0.0

If the IDE doesn’t get dependencies automatically, run the following command from the terminal.

flutter pub get

Next, open lib/providers/app\_state.dart file. This is where we will configure our Appwrite SDK. First, import the appwrite package.

import ‘package:appwrite/appwrite.dart’;

import ‘package:appwrite/models.dart’; // Response type definitions

In the AuthState class, add new properties.

class AuthState extends ChangeNotifier {

 final \_client = Client();

 late final Account \_account;

}

Then in the constructor, let’s configure the client and instantiate the account.

 AuthState() {

 \_client.setEndpoint(‘YOUR\_ENDPOINT’)

.setProject(‘YOUR\_PROJECT\_ID’);

 \_account = Account(client);
 }

Once the SDK is installed and configured, the final step is adding the platform to our Appwrite project. Open the Appwrite console and choose the project you created for this project. You can find the **Add Platform** button on the project home page. Click on the **Add Platform** button and select **New Flutter App**. In the dialog box that appears, choose the appropriate Flutter platform, assign it a recognizable name, and add the application ID or package name based on the platform. You must complete this step for each Flutter platform you build in your application.



Flutter developers can find more detail instructions on using Appwrite in our official [Getting Started for Flutter](https://appwrite.io/docs/getting-started-for-flutter) guide.

Now that our SDK is installed and configured, we are ready to implement phone authentication in our application.

## Creating a Session with a Phone

Two steps are needed to create a valid session for users with phone numbers. First, to initiate the session creation process, we will use the setup provider for sending a verification code to the provided phone number. Next, the verification code will confirm and create a valid session for the user. So let’s implement those two methods in our AuthState class.

To initiate the session, we call createPhoneSession from the account service and pass the userId and phone numbers. Passing ‘unique()’ as userId allows Appwrite to generate a unique userId.

 Future<bool> createSession(String phone) async {

 try {

 final token =

 await \_account.createPhoneSession(userId: 'unique()', number: phone);

 userId = token.userId;

 return true;

 } on AppwriteException catch (e) {

 \_error = e.message ?? e.toString();

 isLoggedIn = false;

 notifyListeners();

 return false;

 }

 }

Calling createPhoneSession returns a token object with userId. We save the userId because we need it to confirm authentication and create a valid session. To do that, we need to call updatePhoneSession from the account service. Let’s follow the confirmSession method.

 Future<bool> confirmSession(String secret) async {

 if (userId.isEmpty) {

 \_error = 'userId not known, call `createSession` first to get userId';

 notifyListeners();

 return false;

 }

 try {

 await \_account.updatePhoneSession(userId: userId, secret: secret);

 isLoggedIn = true;

 notifyListeners();

 return true;

 } on AppwriteException catch (e) {

 \_error = e.message ?? e.toString();

 isLoggedIn = false;

 return false;

 }

 }

Now let’s call these methods in the appropriate UI. Open lib/login.dart and find the Login button. In the onPressed action of the button, add the following code after the check.

final success = await ref

 .read(AuthState.provider.notifier)

 .createSession(\_phoneController.text);

if (!mounted) return;

if (success) {

 Navigator.pushNamed(context, '/verify');

}

Open lib/verify\_phone.dart and find the Verify button. In the onPressed action of the button, add the following code after the check.

final loggedIn = await ref

 .read(AuthState.provider.notifier)

 .confirmSession(currentText);

if (!mounted) return;

if (loggedIn) {

 Navigator.pushReplacementNamed(context, '/');

} else {

 ScaffoldMessenger.of(context).showSnackBar(SnackBar(

 content: Text(ref.read(AuthState.provider).error),

 ));

}

Here we are calling Appwrite with the code received in the SMS. There’s one last thing we need to do -- we need to make a call to get an account. Open lib/providers/app\_state.dart and add the following function.

Future<bool> getAccount() async {

 try {

 user = await \_account.get();

 isLoggedIn = true;

 notifyListeners();

 return true;

 } on AppwriteException catch (e) {

 \_error = e.message ?? e.toString();

 isLoggedIn = false;

 user = null;

 notifyListeners();

 return false;

 }

 }

Call this function in the constructor of the AuthState, so when the application starts, it checks if we already have a valid session. You can now run the application and see the phone authentication in action.

// gif

## Conclusion

That’s how easy it is to set up phone authentication with Appwrite. You can choose between different providers --[Twilio](https://twilio.com/), [Telesign](https://www.telesign.com/), and [TextMagic](https://www.textmagic.com/) at the moment. You are also welcome to contribute by adding more providers. To learn more about Appwrite and to integrate it with Flutter, here are a few more resources that might help:

* [Getting Started for Flutter](https://appwrite.io/docs/getting-started-for-flutter)
* [Appwrite Docs](https://appwrite.io/docs)
* [Appwrite Discord](https://appwrite.io/discord)
* [Appwrite Github](https://github.com/appwrite/appwrite)