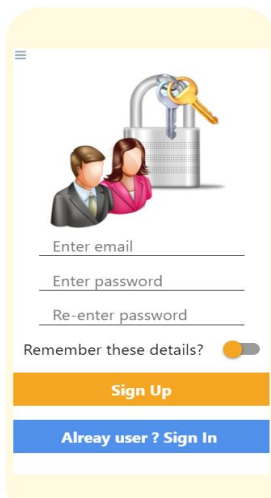


CREATE YOUR OWN PROFILE

The Sign In template demonstrates the 4 functions of the [Sign In](#) component: Sign Up, Sign In, Sign Out, and Reset Password.

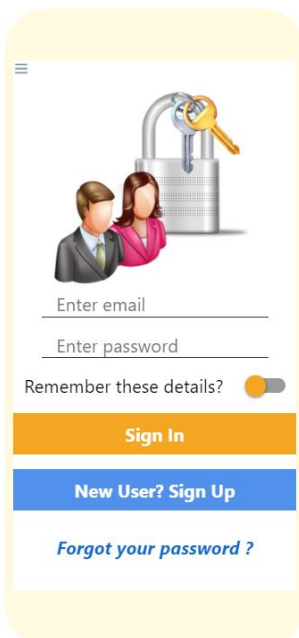
The Sign Up/Sign In functions will be demonstrated on the same screen. This screen looks so beautiful because it is based on a remix of the existing [Sign In screen](#)

Sign Up Screen:



A mobile app mockup for a sign-up screen. At the top, there is a header with a hamburger menu icon on the left and an illustration of a man and a woman standing next to a large padlock with a key inserted. Below the illustration are three input fields: 'Enter email', 'Enter password', and 'Re-enter password'. A toggle switch labeled 'Remember these details?' is positioned below the input fields. At the bottom, there are two buttons: an orange 'Sign Up' button and a blue 'Already user ? Sign In' button.

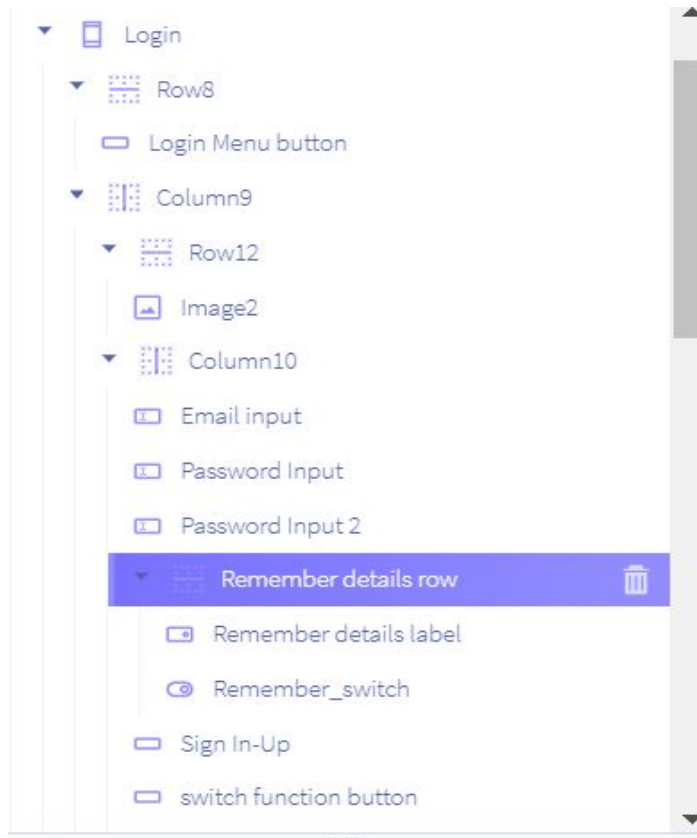
Sign In Screen:



A mobile app mockup for a sign-in screen. At the top, there is a header with a hamburger menu icon on the left and an illustration of a man and a woman standing next to a large padlock with a key inserted. Below the illustration are two input fields: 'Enter email' and 'Enter password'. A toggle switch labeled 'Remember these details?' is positioned below the input fields. At the bottom, there are three elements: an orange 'Sign In' button, a blue 'New User? Sign Up' button, and a blue link 'Forgot your password ?'.

Components

Because this is more complex than other Screens I have demonstrated in tutorials, I will show the component tree and explain which components are visible when a user is signing in or signing up.



Visible in both: Email Input, Password Input, RememberDetailsRow (containing RememberDetailsLabel and Remember_Switch), SignIn-Up, SwitchFunctionButton, and ResultLabel (when there is a result to be shown).

Visible only when signing up: Password Input 2

Visible only when signing in: ForgotPassword

Invisible Components: Alert1, Sign_In1

We'll be setting the Title and Message of the Alert in the blocks, so you just need to set the Confirm Button text and the Cancel Button text for it in the Design tab.

Blocks

Begin by initialising some variables.

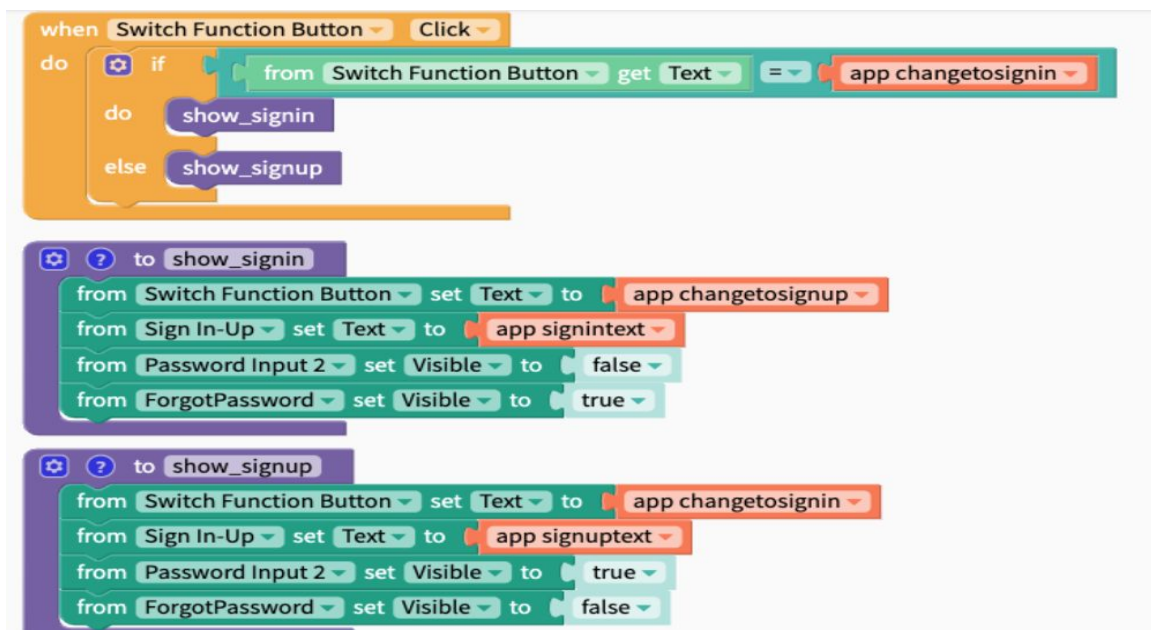


We will initialize some stored variables that can store the user's email and password, and some app variables that will be displayed as the text of different Buttons depending on whether the user is signing in or signing up.

The user will see a Button saying 'Sign In' when they sign in and a Button saying 'Sign Up' when they sign up.

The user will see an option to switch to signing up on the Sign In screen, and to switch to signing in on the Sign Up screen.

We can turn these settings into functions.

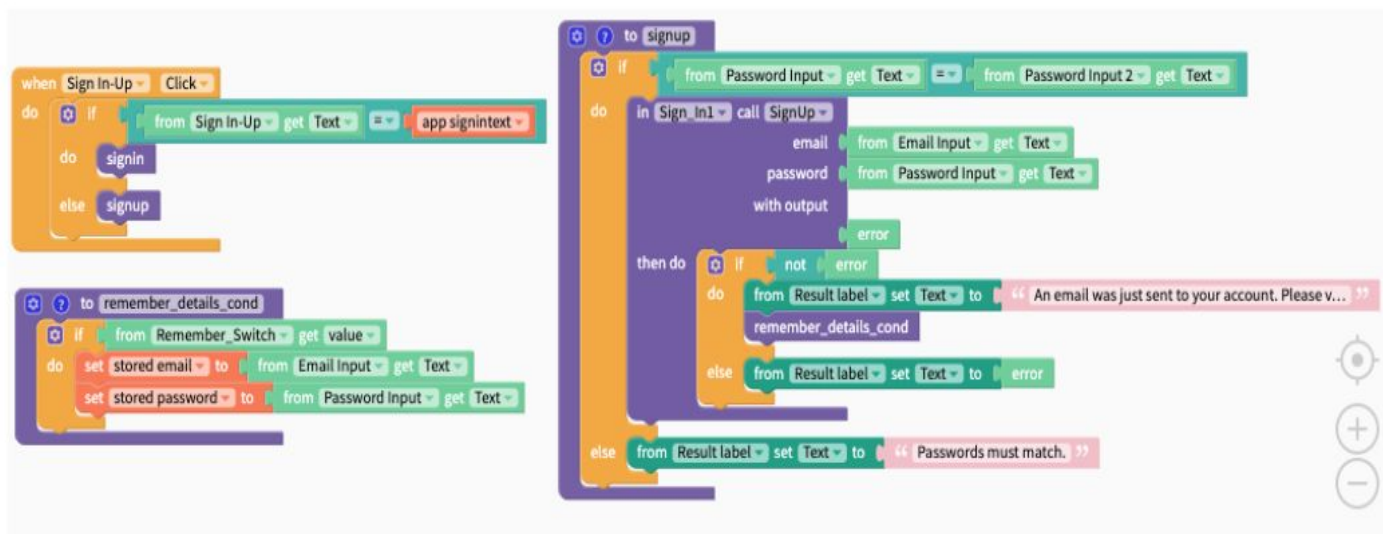


When the user switches to 'sign in', we'll show them the text described above, we'll show them a password recovery option, and we'll hide the second password input box.

When the user switches to 'sign up', we'll show them the text described above, we'll show the second password input box, and we'll hide the 'forgot password' Button.

Sign Up Blocks:

When the user clicks the 'Sign Up' button, which is the Button 'signup' when the text says 'sign up', they will call the 'sign up' function seen below.



First, the blocks check if the passwords in both Password inputs match.

If they do, the 'sign up' function of the Sign In component will be called. The user will either be shown an error or will be asked to check their email to verify their email address.

Once they have verified their email address, they will be able to sign in.

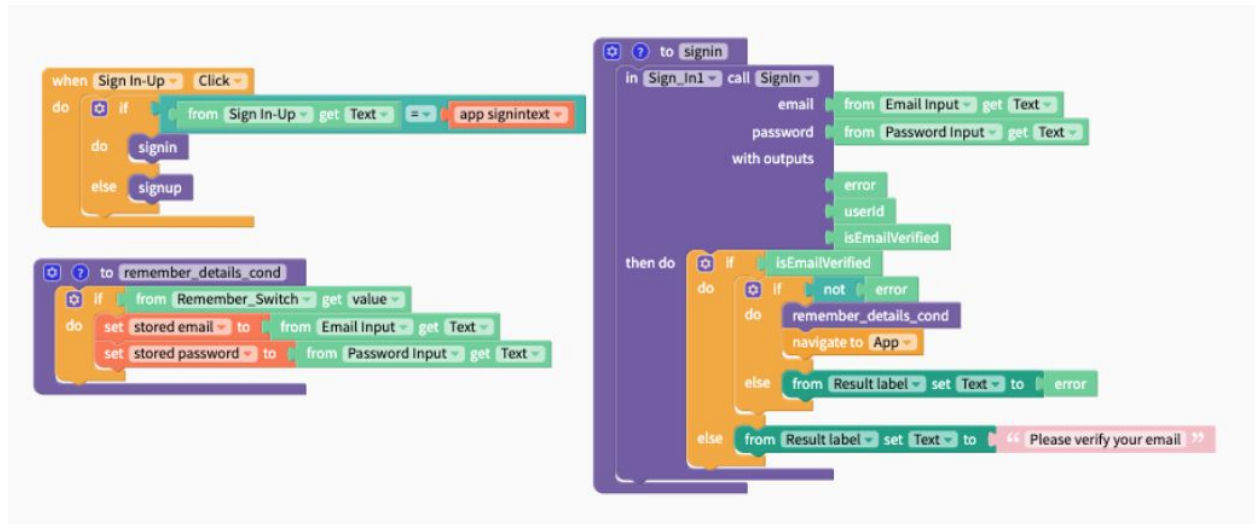
The extra function saying 'remember_details_cond' is very simple: if the user has set the 'remember my details' switch to be true, the app will save their email and password as stored variables.

You can learn how to sign someone in with stored email and password details in the next post in this series.

Sign In Block

The previous post explained how to let a user sign up for your app, as well as explaining how to make one Screen function as a Sign In screen and a Sign Up screen with a few minor tweaks.

This post will explain how to sign a user in. This post will also show you how to log a user in with saved details.



When the user clicks the Sign In button, which is the Button 'signinup' showing the text 'Sign In', we will call the Sign In component's 'sign in' function.

If the user isn't verified, we'll ask them to verify their email.

If there is an error signing the user in, the error message will display.

If the user is verified and there is no error signing them in, we'll call the remember_details_cond function and then show whatever the first screen of the app itself is.

As we saw in the last post, the function remember_details_cond will save a user's email and password as stored variables if the user has set the 'remember my details' Switch to TRUE.

But how do we sign a user in with these details?

We'll actually do this before the user has the chance to press any buttons.



When the Sign In Screen opens, we'll check if the user has saved values for their email and password as stored variables in their device. This is simply done by seeing if the stored variables 'email' and 'password' are more than 0 characters long.

(NB: this function uses the pink 'length of' block from the *Text* drawer of blocks, not the blue 'length of' block from the *List* drawer of blocks!)

If they have saved values for 'email' and 'password', we'll sign them in with these details.

If they haven't, we'll show them the Sign Up screen.

But what if the user didn't save their details, and now they've forgotten their password?

The next post will show you how to use the 'Recover Password' function of the Sign In component.

Reset Password :

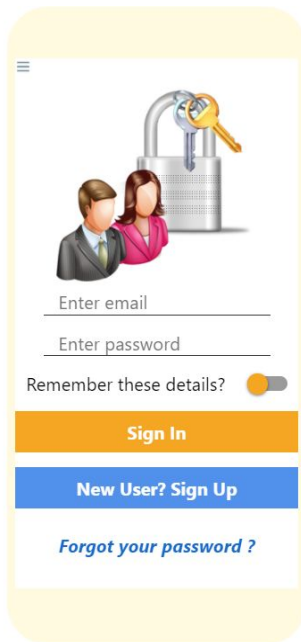
This post will explain how to add a Reset Password function to your Sign In screen.

The Sign In template lets users reset their password from the Sign In Screen, but you can also put this function inside your app.

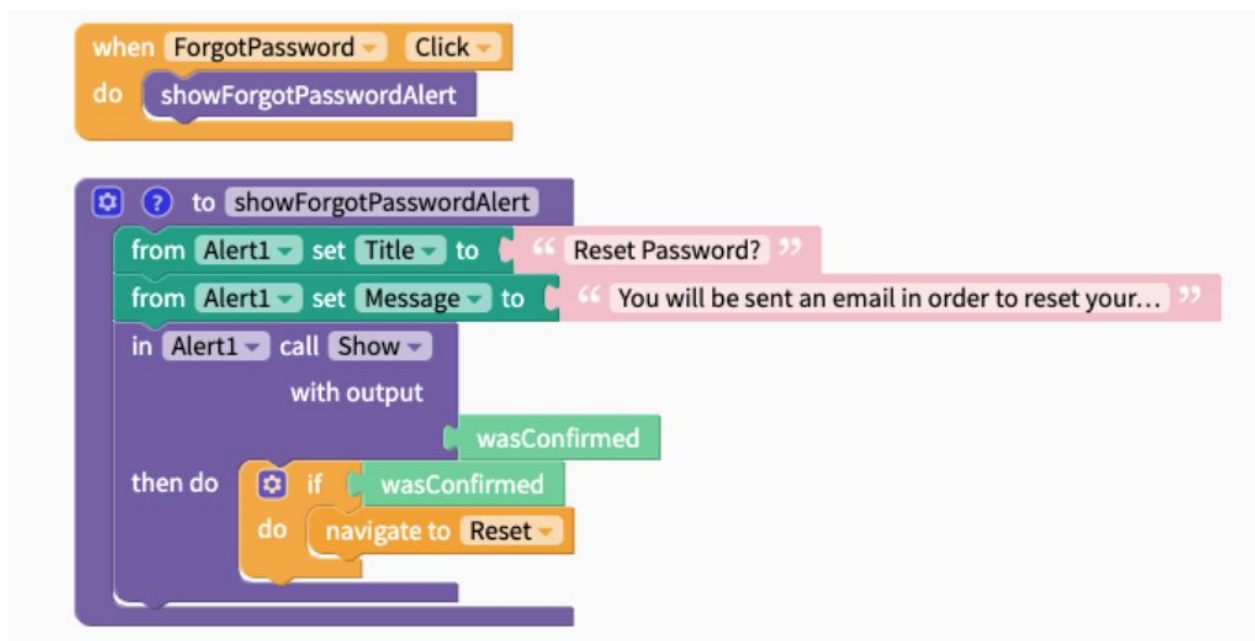
This method will bring the user to another Screen in order to reset their password.

Accessing Reset Password Screen

On our Sign In screen, you can see that there is a Button that says 'Forgot Your Password?'

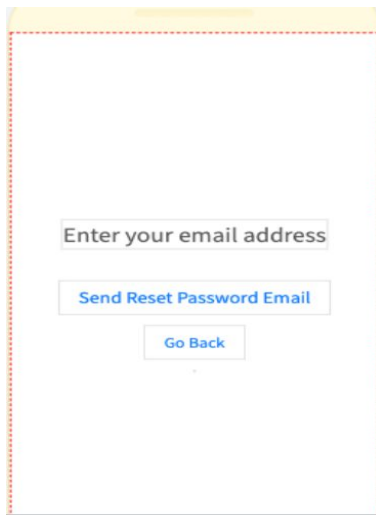


When the user clicks this, an alert will show with the title 'Forgot Password?' and the text 'You will be sent an email in order to reset your password.'



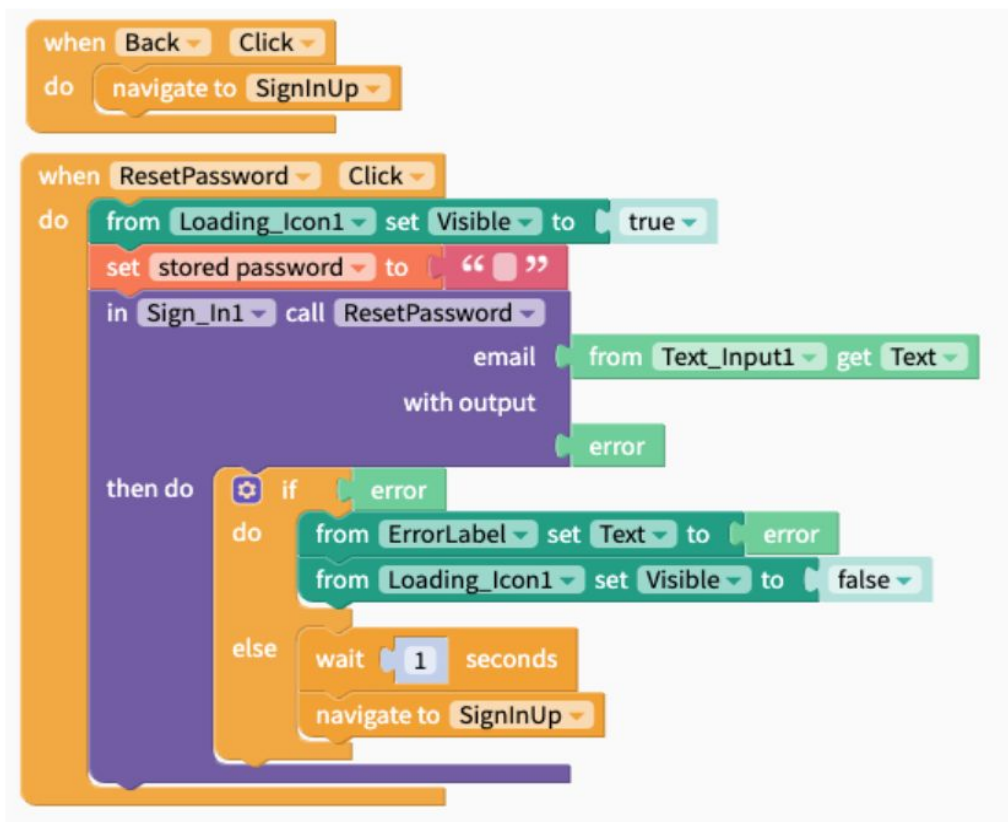
If the user confirms this Alert, they will be taken to the Reset Screen.

Reset Screen Design



A mockup of a mobile app screen for password reset. It features a light yellow header bar. Below it, a white rectangular area contains a text input field with the placeholder text "Enter your email address". Below the input field is a blue button with the text "Send Reset Password Email". At the bottom of the white area is a smaller blue button with the text "Go Back".

This screen is very simple, with a Text Input, a 'Reset Password' button, and a 'Back' button. There is also an Error Label (which currently has no text) and a Loading Icon (currently set to invisible).



On this Screen, if the user clicks 'Go Back', they will be brought to the Sign In screen.

If they click 'Send Reset Password Email', the user's stored password will be set to blank and we'll show the Loading Icon.

A reset password email will be sent to the email address entered into the Text Input.

If there's an error, the Error Label will display it.

If there's no error, the user will be brought back to the Sign In screen.

Once they've completed the password reset process, they can sign in as usual.

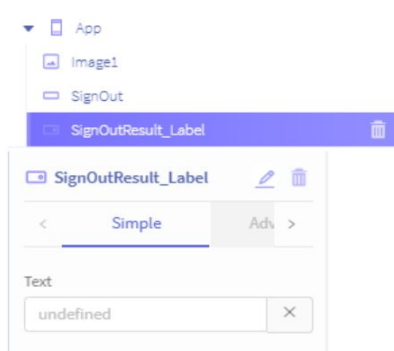
What about if a user has successfully logged in in the past but now wants to sign out? The next post will demonstrate how to use the Sign Out function.

Sign Out:

So your user logged in successfully, has been enjoying your app, and now wants to sign out. Of course, you have the best app ever, so it's weird that someone would want to sign out. Maybe they want to make their friends create an account right now! Maybe they have a bunch of accounts because your app is that good!

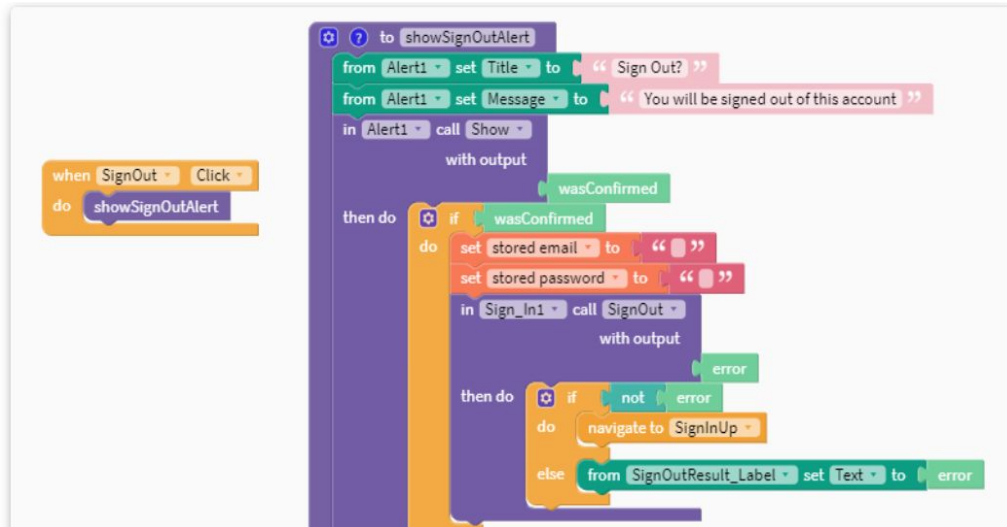
Either way, it is frowned upon to trap users inside your app, so you have to give them a Sign Out button.

This app also has an empty Label which will display any error the user has with signing out.



This sign-out method also uses the Alert component.

If you followed the previous post's instructions about adding a Password Reset to your app, you already have an Alert in your app.



When the user clicks the Sign Out Button, these blocks set the Alert's Title to 'Sign Out?' and the Message to 'You will be signed out of this account'.

If the user confirms this decision, we'll clear the stored email and password and call the Sign Out function of our Sign In component.

If there's an error, the Label I mentioned above will show the error.

If there's no error, the user will be brought back to the Sign In Screen.

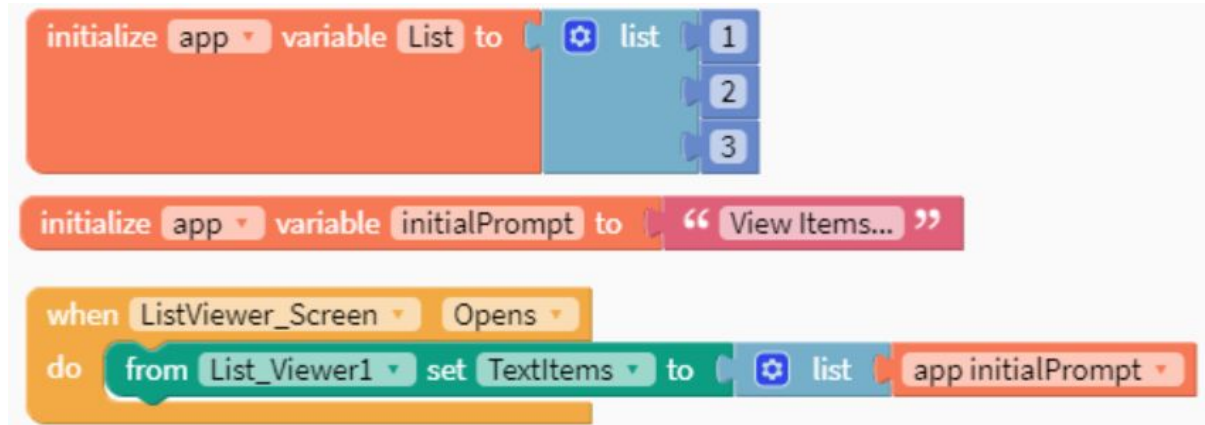
And that's the Sign In component masterpost! Let me know if you have any questions or feedback in the comments.

Drop Down Menu:

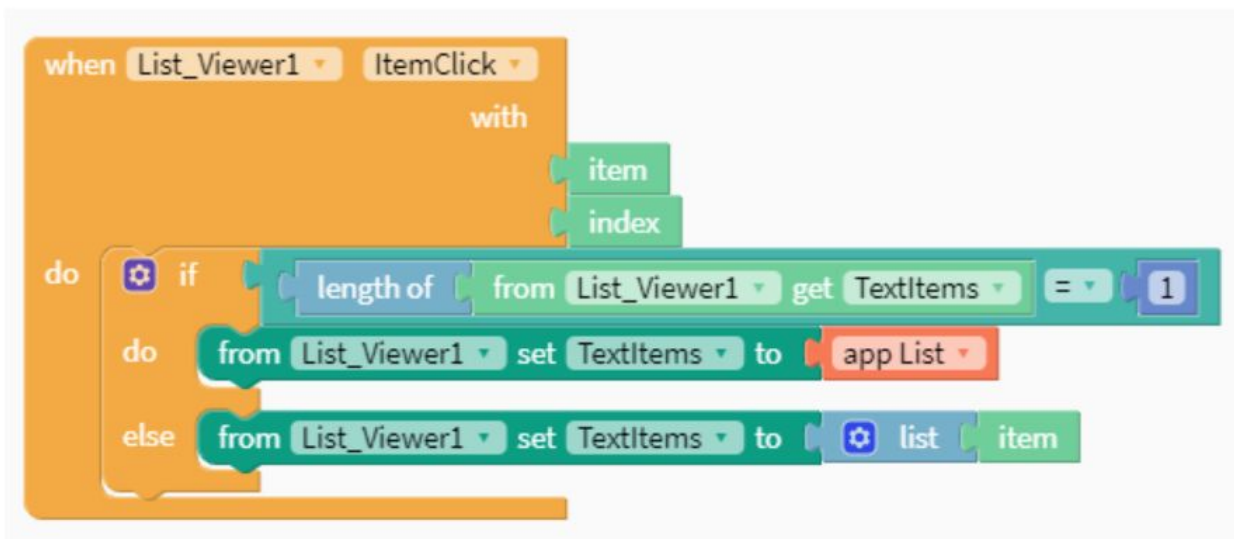
This method uses a single List Viewer to make a drop-down menu.



The Layout needed for this feature is simply an empty List Viewer.



Start by initialising the list of whatever items you want to display in your drop-down menu. Then initialise the message you want the menu to display before the user selects anything. Set the List Viewer to show this message as a 1-item list when the Screen opens.



For this List Viewer, when the user clicks an item, one of two things is happening:

1. They want to see the menu of items
2. They are looking at the menu of items and want to select an item

We use an if-else block to handle this choice. If the current List Viewer is displaying a 1-item list, it is either displaying the initial message or it is displaying something the user has selected. So if the current List Viewer is displaying a 1-item list, we make it display the full menu of items.

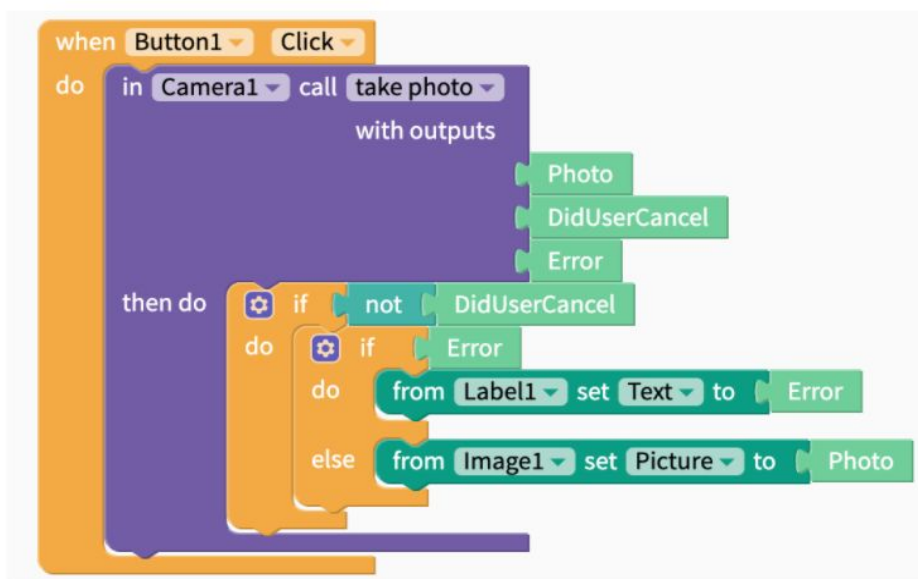
If the List Viewer is displaying a list that is longer than one item, it can only be displaying the full menu of items. So we set the List Viewer to display the selected item as a 1-item list.

Camera:

The camera block has three outputs:

- i) **Photo:** This saves the photo taken by the user.
- ii) **DidUserCancel:** This is either true or false depending on whether the user cancelled taking the picture
- iii) **Error:** Indicates if there was an error when the photo was taken

These blocks say that when Button1 is clicked, open the camera. If the user does not cancel the action, one of two things will happen: if there is an error, display the error message on a Label. If there is not an error, set the photo taken to be displayed in an Image component.



Select Photo



The blocks above say that when Button1 is clicked, open the photo library and select a picture from it.

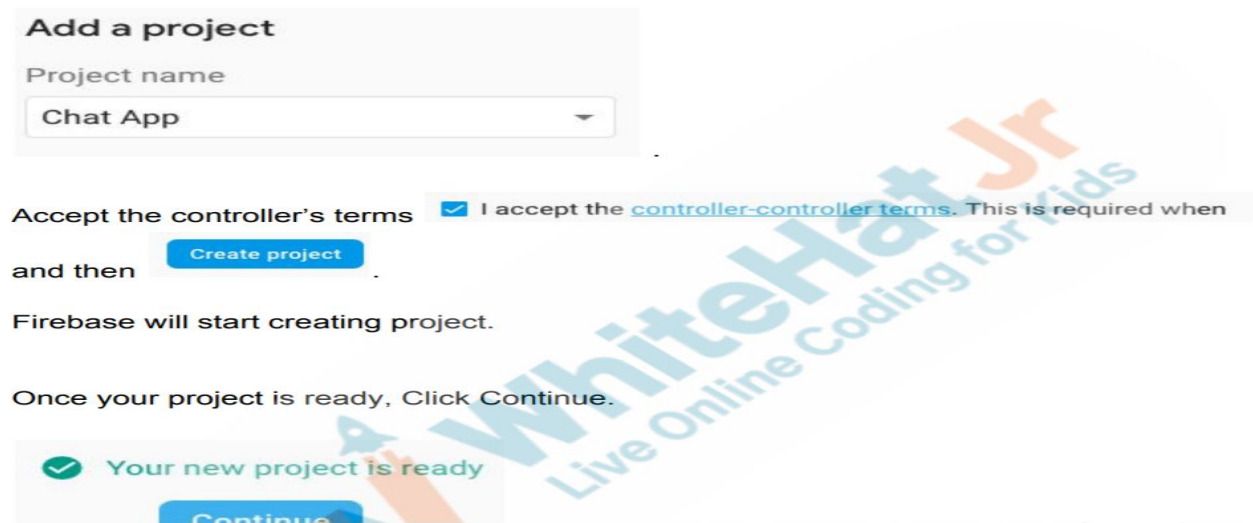
Select Photo: Opens the camera and saves the Photo after the user takes a picture.

- DidUserCancel is either 'true' or 'false' depending on whether or not the user cancelled selecting a photo.
- Error is the error message if there was an error in selecting the photo

Firestore:

1. So let's begin by logging in to Google FireBase. Click on top right.
2. Once you are logged-in click on top right.
3. Create a project by clicking "Add Project (+)" and give your project a name. Since we are going to create a chat app let's call this project chat app.

3. Create a project by clicking [Add project](#) and give your project a name. Since we are going to create a chat app let's call this project chat app.



Add a project

Project name

Chat App

Accept the controller's terms ☒ I accept the [controller-controller terms](#). This is required when

and then [Create project](#)

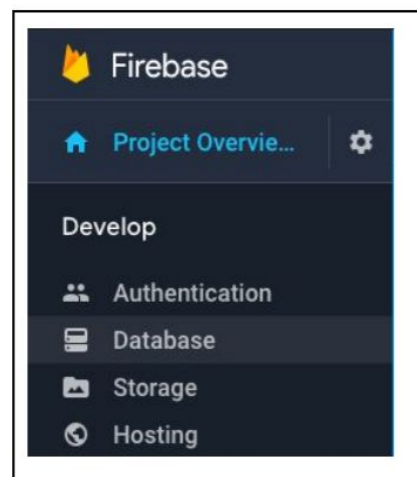
Firebase will start creating project.

Once your project is ready, Click Continue.

✓ Your new project is ready

[Continue](#)

4. On the left you click Database under the Develop category



Let's now build this database.

5. Scroll down to choose RealTime Database and click

Create database

Or choose Realtime Database



Realtime Database

Firebase's original database. Like Cloud Firestore, it supports realtime data synchronization.

[View the docs](#) [Learn more](#)

Create database

☒ **Start in test mode**
Get set up quickly by allowing all reads and writes to your database

6. Click

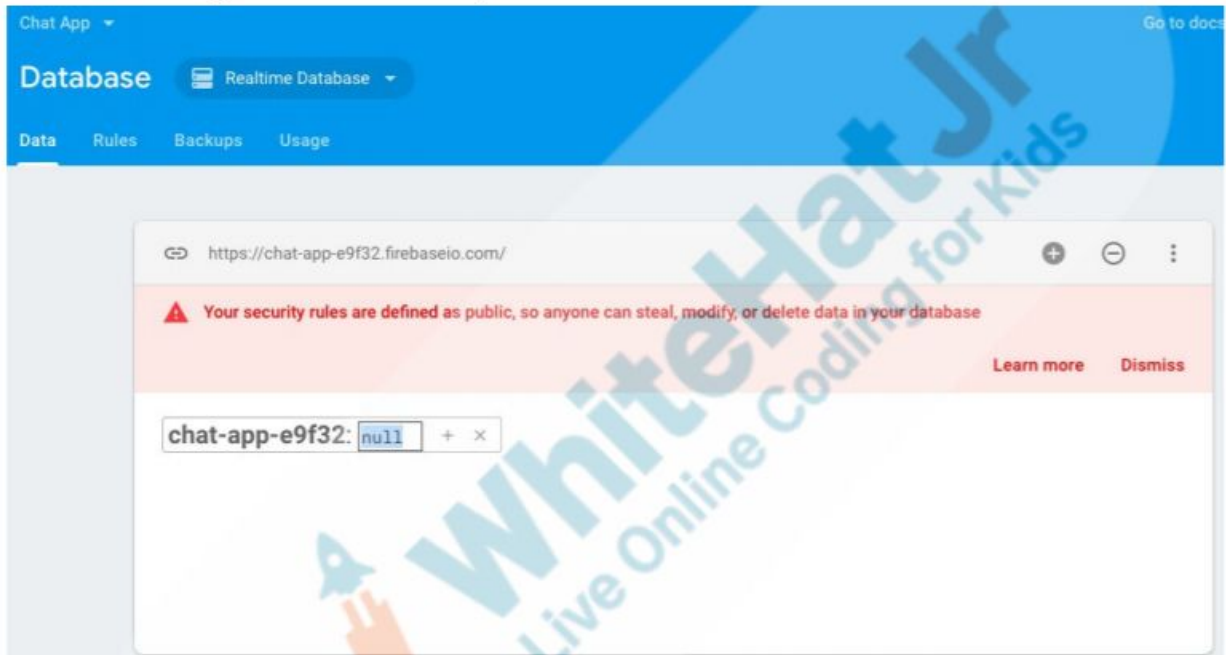
What this means is anyone using the Chat will be able to send text and receive texts from the database.

Anyone with your database reference will be able to read or write to your database

Enable

Click

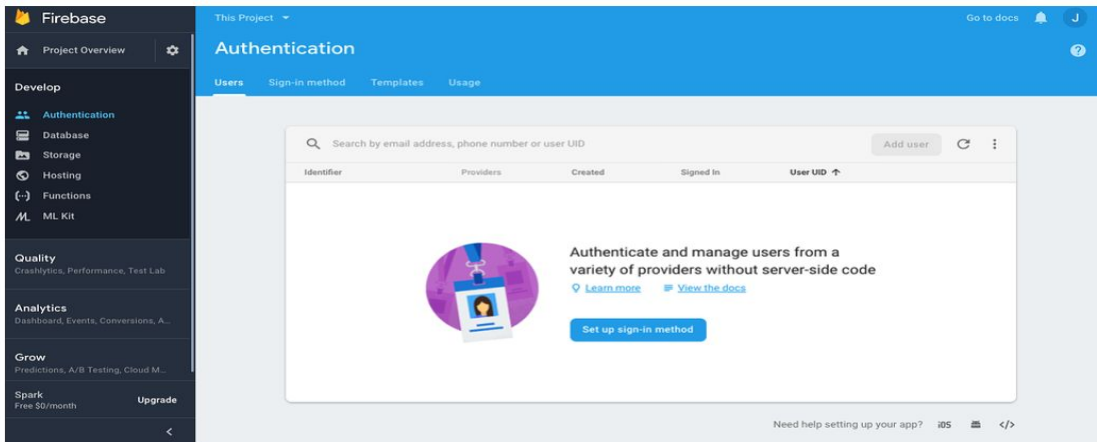
This will create your database and you can see it.



Right now your database shows **null**, which means the database has nothing in it.

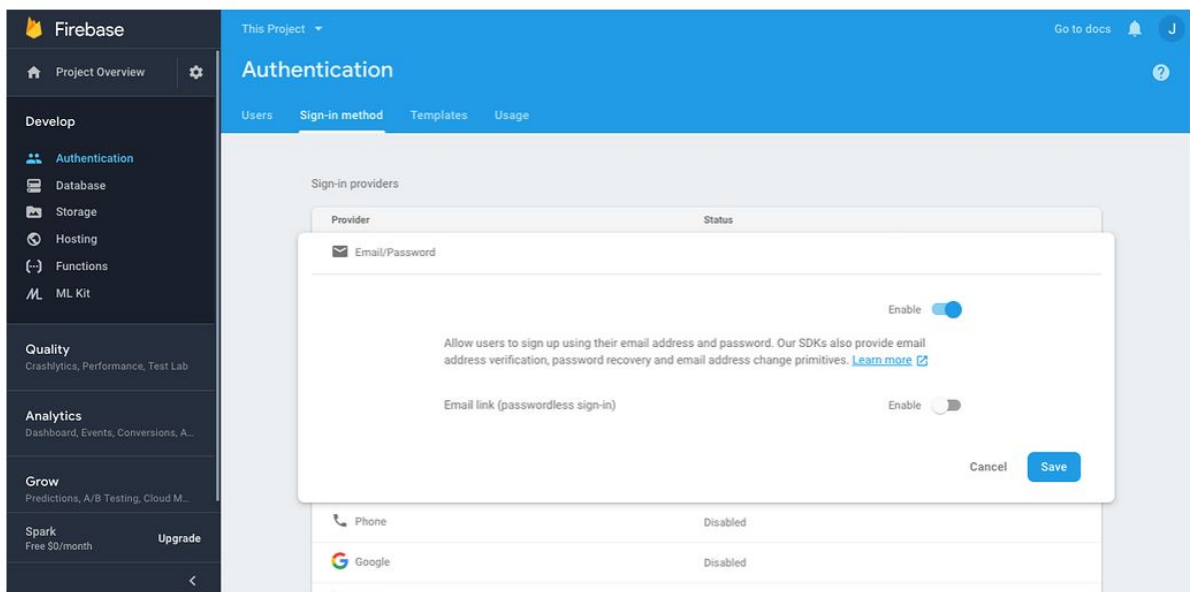
Add Email Authentication

Click on the 'Authentication' option in the Develop sidebar.



Click the button that says 'Set up sign-in method'.

If you hover your mouse over the 'Email/Password' option, you'll see a pencil. Click this pencil to enable/disable Email + Password Authentication in your Firebase project. Click 'Save' to save these settings.



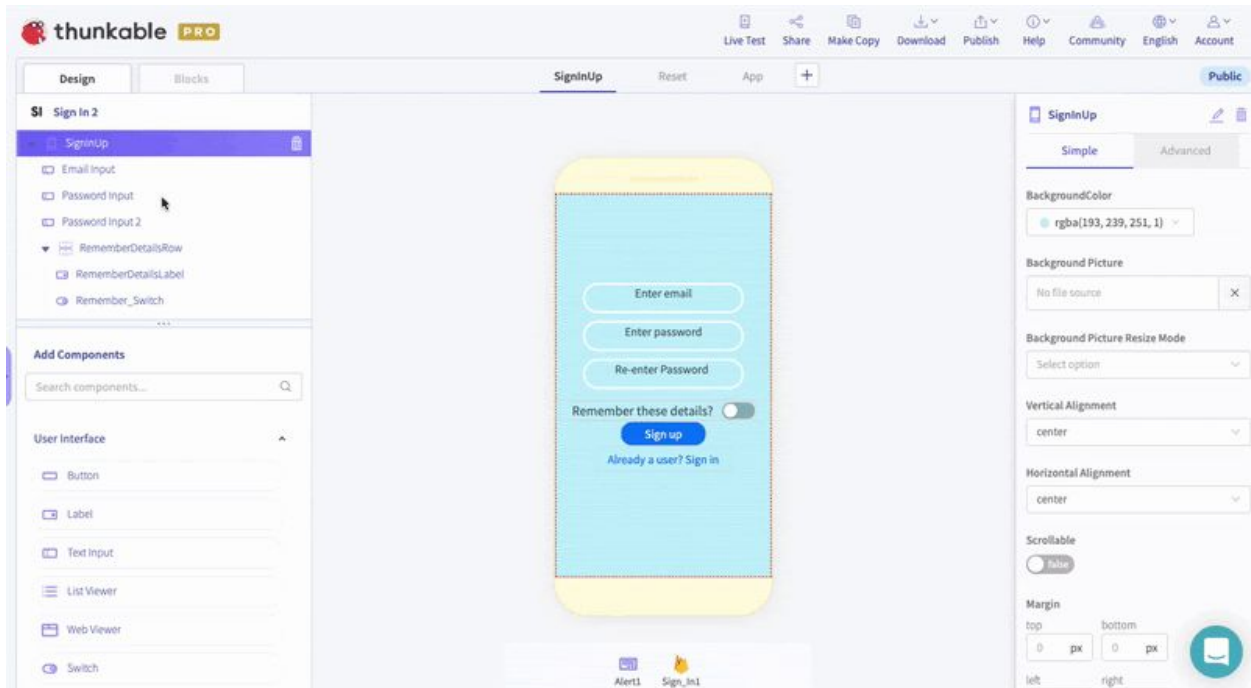
Add your Firebase DB to your App Project

Click the gear icon next to the 'Project Overview' text, underneath the Firebase logo, to access the Settings page. You can find the Web API key in the General tab of the Settings page.

You can find the databaseURL in the 'Service accounts' tab of the Settings page.

To add these details to your Thunkable X App Project, click on your project title/app icon. This will show your App Settings in the Properties tab on the right-hand side of the screen.

Scroll down to the 'Firebase Settings' section and paste in your API Key and your database URL.



Your project is now connected to your Firebase DB!