For connecting the server

To connect the file with the server of the fast api, use the

ipv4 address. other wise you cant access it from your phone.

For that run the main\_3.py using this

uvicorn main\_3:app --host 192.168.110.109 --port 8000

uvicorn main\_3:app --host 192.168.0.5 --port 8000

it will run your server in the correct host.

Replace it in your phone. Also To check it, use postman.

Repository.

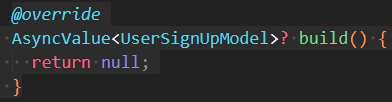
Repository holds the ai calls. It will call the api. auth\_remote\_api will call the api perform the authentication.

View model

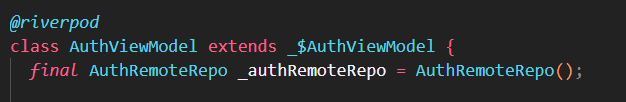
View model task is to tell views what to show. And for that state management river pod will be used. View model will also communicate with the repository to communicate with to get the different state.

Riverpod

What is the build function do?



The build function will initialize the notifier.  The build function will also return an initial value. Another good thing is that it will provide access to the REF which will be used a lot.



While using the annotation of the provider, another annotation needs to be mentioned like part 'auth\_viewmode.g.dart'. This class file holds the data for **auto disposable**.

The build function in the auth\_viewmodel.dart or in river pod helps to keep track of the dependencies  and also provide the ref function which is important to capture the changes.

Using the AsyncValue the state of the authViewModel is updated. For example :

1. When the function is starting out     state = const AsyncValue.loading();
2. When it fails to receive the data, then state = AsyncValue.error(l.message, StackTrace.current)
3. when it success the state is updated to state = AsyncValue.data(r)

normal way to listen the provider.

    ref.listen(authViewModelProvider, (previous, next) { });

the previous = previous holds the value previous state and next holds the updated state.

To run the code generation:

dart run build\_runner watch -d

The Provider that is used is auto disposed. The provider is built with the annotation.

Auth view model

The point of the auth view model is this. It’s going to determine what's state is going to show in the UI.

it’s also going to communicate with repository. So that ui and repo dont communicate directly.

Uncontrolled Provider Scope



Riverpod Uncontrolled-Provider-Scope is special widget and that helps you prevent rebuild of your widget.

Prevent unnecessary rebuild of widget may help performance optimize. In general, Flutter causes a lot of rebuild of widget even if it's not meant to rebuild or you just don't need to rebuild.

The only condition for using Uncontrolled-Provider-Scope() you have to provide it Provider-Container() instance.



**Manually Managed**:

* Requires you to create and pass a Provider-Container manually.
* You are responsible for disposing of the container to avoid memory leaks.

 **Primary Use Case**:

* For advanced scenarios where you need more control over the provider container, such as testing, debugging, or embedding independent provider states.

 **Lifecycle Management**:

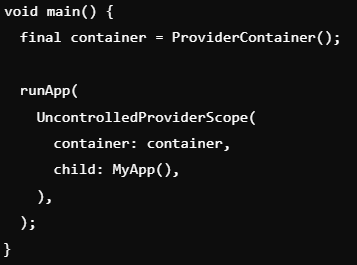
* You must explicitly dispose of the Provider-Container when it is no longer needed.
* Offers flexibility to share the same container across multiple parts of the app.

 **Overrides**:

* Supports provider overrides by configuring the Provider-Container manually before passing it to Uncontrolled-Provider-Scope.

Provider-Container

The container is an instance of Provider-Container. It is used to manage the state of all the providers defined in your Riverpod-based application.



The Provider-Container is created manually and passed to Uncontrolled-Provider-Scope. This allows you to have full control over the providers' lifecycle and their associated states within your application.

If you create the Provider-Container manually, you must dispose of it when no longer needed to release resources.

Provider Scope

**Provider-Scope**

* **Automatically Managed**:
  + Provider-Scope automatically creates and manages a Provider-Container for you.
  + You don't need to manually create or dispose of the container.
* **Primary Use Case**:
  + Used in most Flutter apps as the root widget of the app to provide access to Riverpod providers.
* **Lifecycle Management**:
  + When the Provider-Scope widget is removed from the widget tree, the associated Provider-Container is automatically disposed of, cleaning up all provider states.
* **Overrides**:
  + Allows for provider overrides via the overrides parameter, making it useful for injecting test or mock providers for specific parts of the app.

12.1.2025 – 5.00H

To listen to a particular change

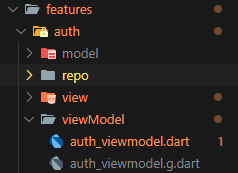


So this line will listen to any changes that has happened in the authviewmodel provider. So any changes that happens in the authViewmodel provider, it will rebuild the widget.

To listen to any particular changes then :

 Now this function will listen to only the isLoading changes.

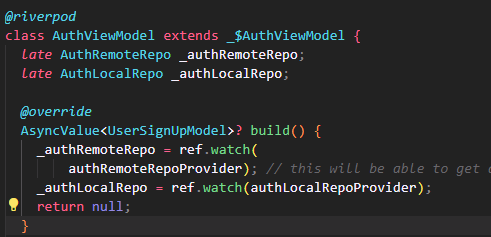
MVVM structure

 The auth view model holds the state of the UI. This is mainly a ui updater states.   
On the other hand the repo class is the handler of local and remote api calls. View is for UI. And model is for the model classes.

Build Method in Riverpod? Why authViewModel has build method but auth\_remote\_repo don’t have any build method?

* The build method is part of the lifecycle of a Notifier or AsyncNotifier class in Riverpod.
* It is called when the provider is first initialized or when it needs to rebuild its state.
* This method defines the **initial state** of the provider.

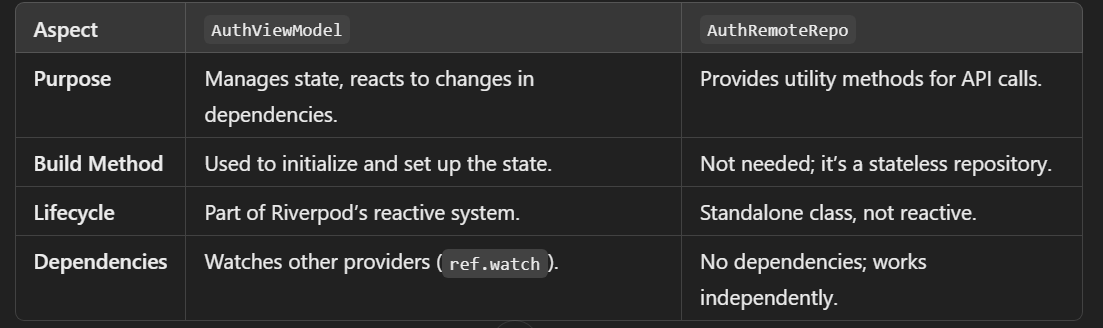
In your AuthViewModel, you are initializing the dependencies (\_authRemoteRepo and \_authLocalRepo) using the ref.watch() method.



These dependencies are watched so that if their state changes, the AuthViewModel can react accordingly.

**Why AuthRemoteRepo Doesn't Have a build Method**

* The AuthRemoteRepo is a simple **data repository**:
  + It doesn't need to manage any reactive state or depend on other providers.
  + It provides methods (executeSignUp) that perform network calls but doesn’t track or react to state changes.

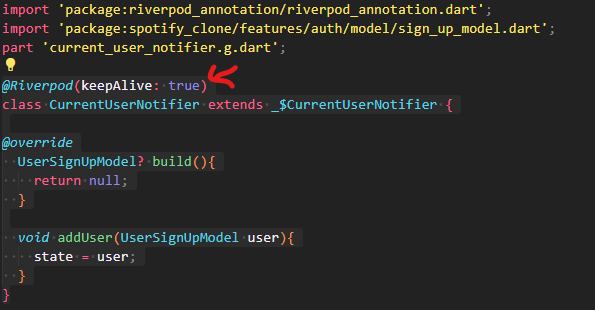


If not tracking any changes in the authRemoteRepo, then why uses ref.watch(authRemoteRepo) in AuthViewModel.

You're using ref.watch(authRemoteRepoProvider) in the AuthViewModel to **retrieve an instance** of AuthRemoteRepo and inject it as a dependency into your AuthViewModel. However, **this does not mean the AuthRemoteRepo provider tracks or reacts to state changes** because:

1. **AuthRemoteRepo is a Stateless Class**:
   * The AuthRemoteRepo class does not hold or manage state. It primarily handles API calls and returns results.
   * There's no state to track or observe within AuthRemoteRepo. It is used as a service layer.
2. **ref.watch(authRemoteRepoProvider) in AuthViewModel**:
   * ref.watch is used here to access the AuthRemoteRepo instance provided by the authRemoteRepoProvider.
   * This approach allows you to inject the dependency dynamically. If you later replace authRemoteRepoProvider with another implementation (e.g., a mock for testing), your AuthViewModel will automatically adapt.
   * In this case, ref.watch doesn't trigger rebuilds because the AuthRemoteRepo provider doesn't have any state changes.
3. **Why Use ref.watch Instead of ref.read?**
   * Technically, you could use ref.read(authRemoteRepoProvider) here because AuthRemoteRepo is stateless, and its state won't change during the app's lifecycle.
   * However, using ref.watch is future-proof. If AuthRemoteRepo were to ever depend on another provider that *does* track state, ref.watch ensures AuthViewModel would react to those changes. It's a safer pattern when building scalable apps.

What is the purpose of keep alive = true



* When you annotate a provider with @Riverpod(keepAlive: true), it ensures that the state inside the provider is **retained across app rebuilds**. This means that even if the widget tree is rebuilt (e.g., because of a state change in other parts of the app), the state of this provider will not be reset.
* Without keepAlive, the provider would be disposed of and recreated whenever the widget that depends on it is rebuilt.

3.2.2025. Starting From 6:16h

What is the purpose of repository?  
The purpose of the repository is to make api call. In the home Repository you can see the http call is made there.