functionalities include-

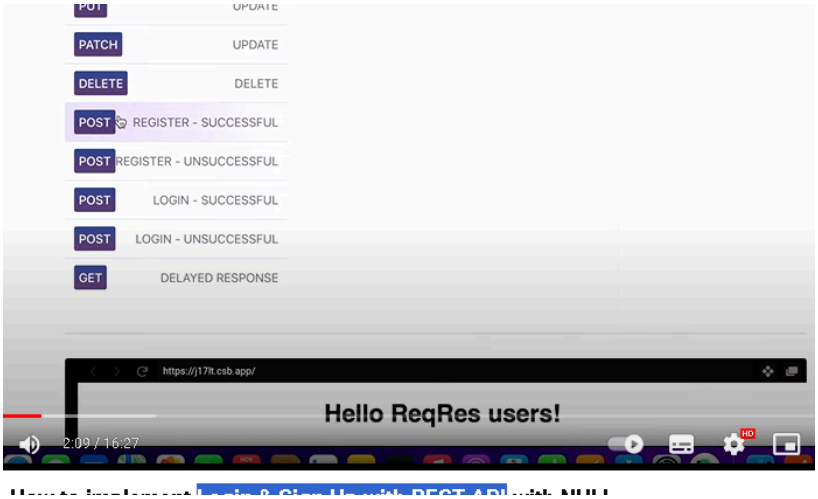
How to create account/sign up
How to sign in
How to upload images on server
How to send arrays on server

First we imported the following packages

- **cupertino_icons**: A package that provides a set of icons used in Cupertino (iOS) style apps.
- **modal_progress_hud_nsn**: A package that displays a modal progress indicator (HUD) with a spinning wheel and optional text, commonly used to show loading or progress states.
- **http**: A package that provides a set of high-level functions for making HTTP requests in Dart, allowing for easy interaction with web servers and APIs.

then created a new dart file SignUpScreen which we linked to our main.dart file

for sign up we will use the following api link
🌐 Reqres - A hosted REST-API ready to respond to your AJAX requests



Copy the link:
Copy the API link provided: https://reqres.in/api/register.

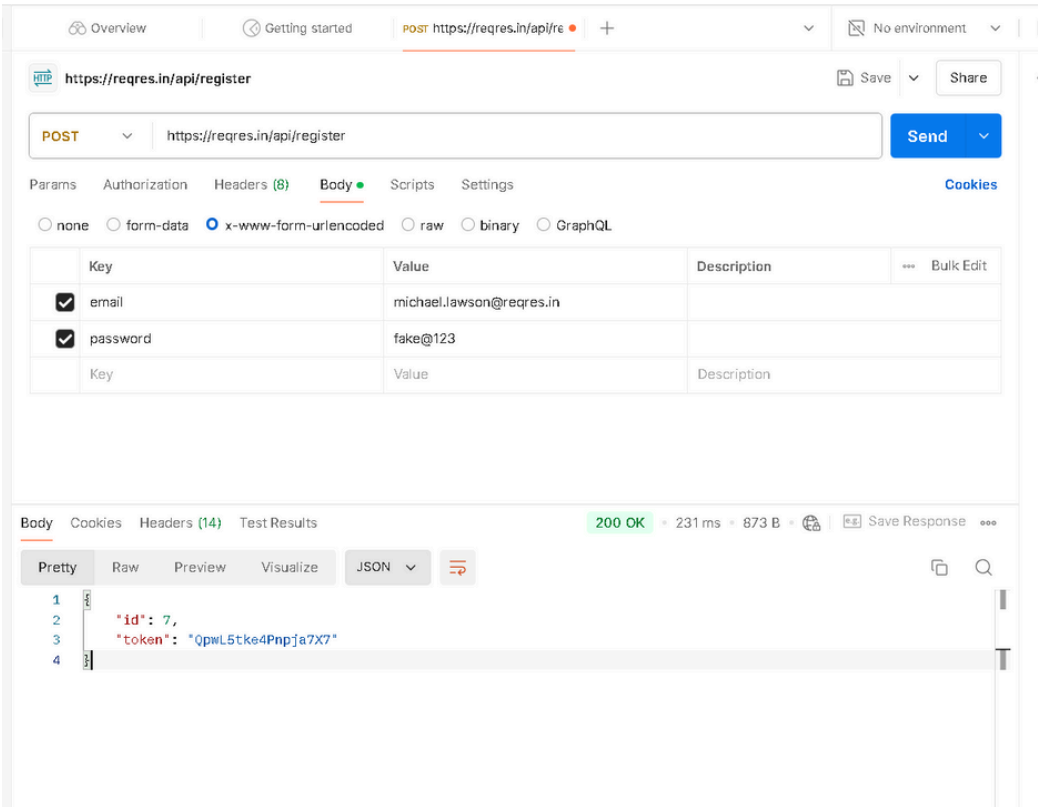
Open Postman:
Open Postman and paste the copied link into the request URL field.

Set the method to POST:
Select POST as the method since we are sending data to the server.

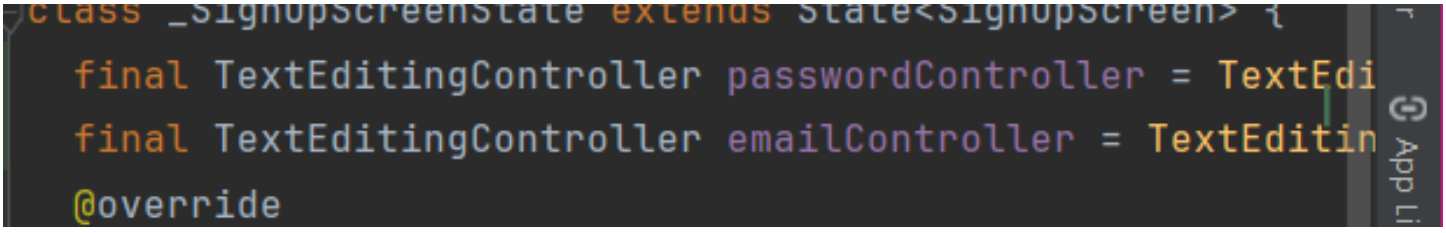
Input email and password:
Go to the **Body** tab, choose x-www-form-urlencoded as the format, and then manually enter the email and password fields with their corresponding values.

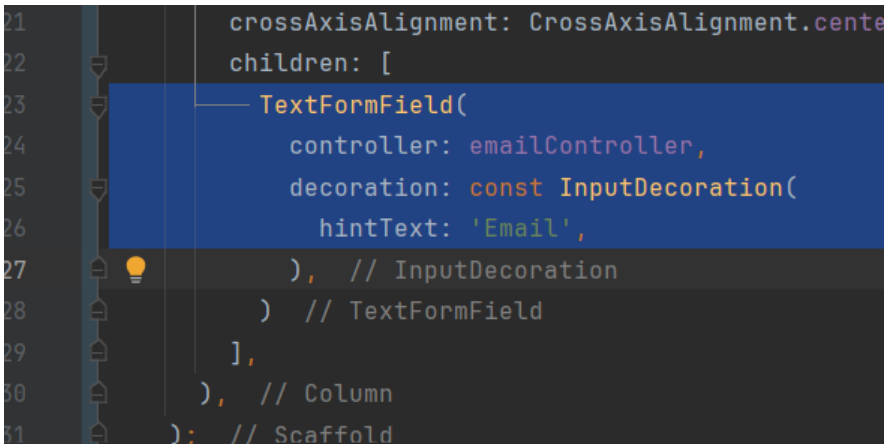
Send the request:
Click the Send button to hit the API.

View the response:
Once the request is sent, the response will be displayed in Postman, and it will be stored on the server as specified by the API.



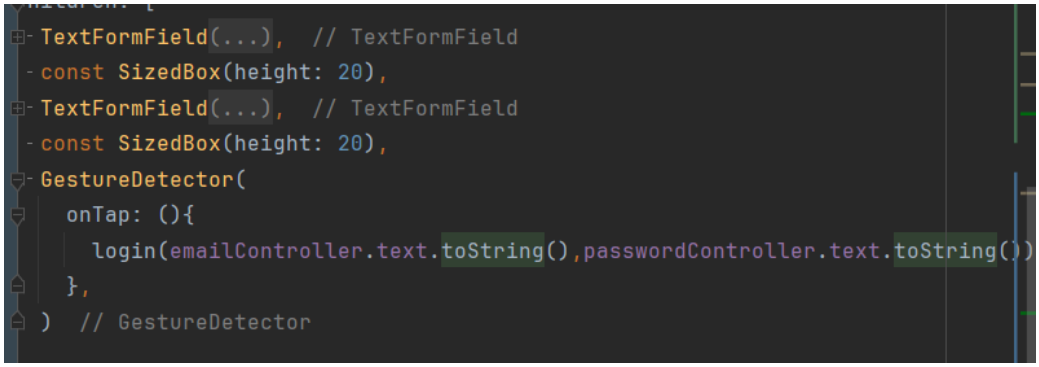
Now coming back to our app we will create an input text field to get the response. emailController is an instance of TextEditingController that is used to control the TextFormField where the user inputs their email address. The controller allows you to programmatically retrieve or modify the text in the field, as well as listen to changes in the text.





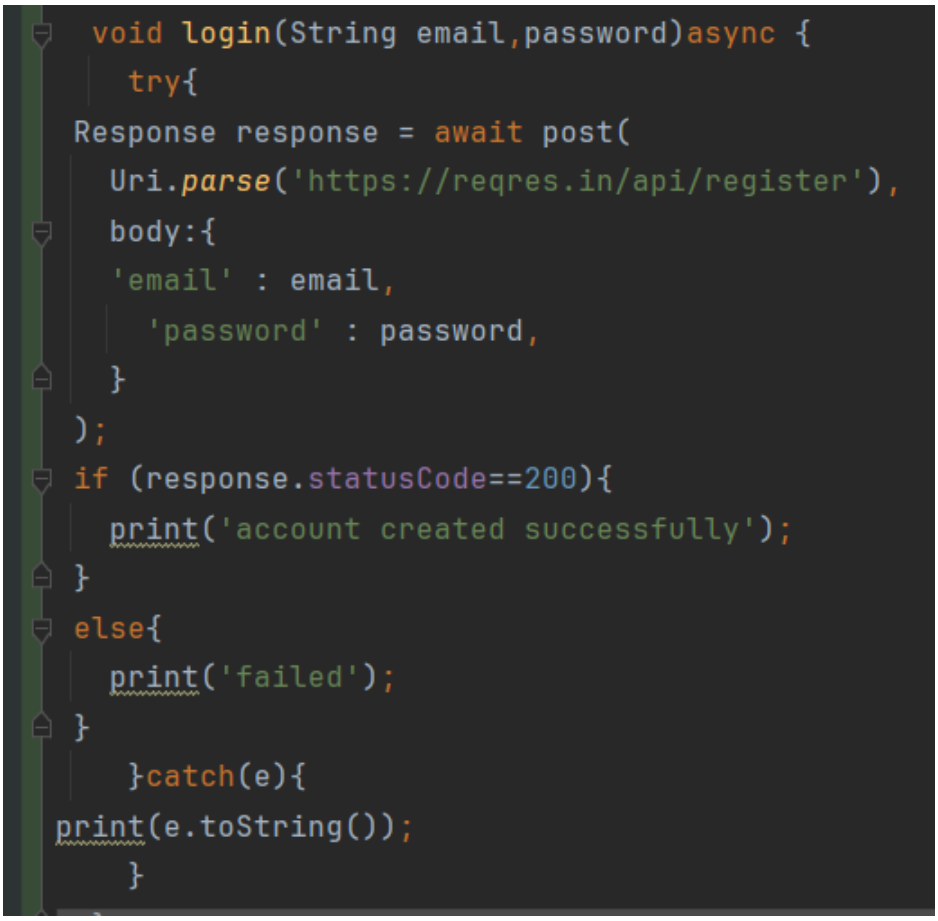
GestureDetector and login function

We created a GestureDetector that calls the login function when a gesture is detected. The login function takes two parameters: email and password, which are obtained from the TextFormField controllers emailController and passwordController respectively.



login function

The login function is an asynchronous function that sends a POST request to the API endpoint https://reqres.in/api/register with the email and password in the request body.



```
 void login(String email,password)async {
   try{
Response response = await post(
  Uri.parse('https://reqres.in/api/register'),
  body:{
  'email' : email,
    'password' : password,
  }
);
if (response.statusCode==200){
  print('account created successfully');
}
else{
  print('failed');
}
   }catch(e){
print(e.toString());
   }
```

- The function is marked as async to indicate that it returns a Future.
- The try block sends a POST request to the API endpoint using the post function from the http package.
- The request body is a JSON object with two fields: email and password, which are set to the values pass-ed as parameters to the login function.
- The await keyword is used to wait for the response from the API.
- If the response status code is 200, it prints "account created successfully" to the console.
- If the response status code is not 200, it prints "failed" to the console.
- The catch block catches any exceptions that occur during the execution of the try block and prints the error message to the console.

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class SignUpScreen extends StatefulWidget {
 const SignUpScreen({super.key});

 @override
 State<SignUpScreen> createState() => _SignUpScreenState();
```

```
}

class _SignUpScreenState extends State<SignUpScreen> {
 final TextEditingController passwordController = TextEditingController();
 final TextEditingController emailController = TextEditingController();

 void login(String email, String password) async {
  try {
   http.Response response = await http.post(
    //Uri.parse('https://reqres.in/api/register'), for sign up

    Uri.parse('https://reqres.in/api/login'), //for login
    body: {
     'email': email,
     'password': password,
    },
   );
   if (response.statusCode == 200) {
    var data = jsonDecode(response.body.toString());
    print(data['token']);
    print('Account created successfully');
   } else {
    print('Failed to create account');
   }
  } catch (e) {
   print(e.toString());
  }
 }


 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Sign Up API'),
   ),
   body: SingleChildScrollView(  // Wrap in SingleChildScrollView
    padding: const EdgeInsets.all(16.0),
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
```

```
crossAxisAlignment: CrossAxisAlignment.stretch,
children: [
 TextFormField(
  controller: emailController,
  decoration: const InputDecoration(
   hintText: 'Email',
  ),
  keyboardType: TextInputType.emailAddress,
 ),
 const SizedBox(height: 20),
 TextFormField(
  controller: passwordController,
  decoration: const InputDecoration(
   hintText: 'Password',
  ),
  obscureText: true,
 ),
 const SizedBox(height: 20),
 GestureDetector(
  onTap: () {
   login(
    emailController.text.toString(),
    passwordController.text.toString(),
   );
  },
  child: Container(
   padding: const EdgeInsets.all(16.0),
   decoration: BoxDecoration(
    color: Colors.blue,
    borderRadius: BorderRadius.circular(8.0),
   ),
   child: const Center(
    child: Text(
     'Sign Up',
     style: TextStyle(color: Colors.white),
    ),
   ),
  ),
 ),
],
```

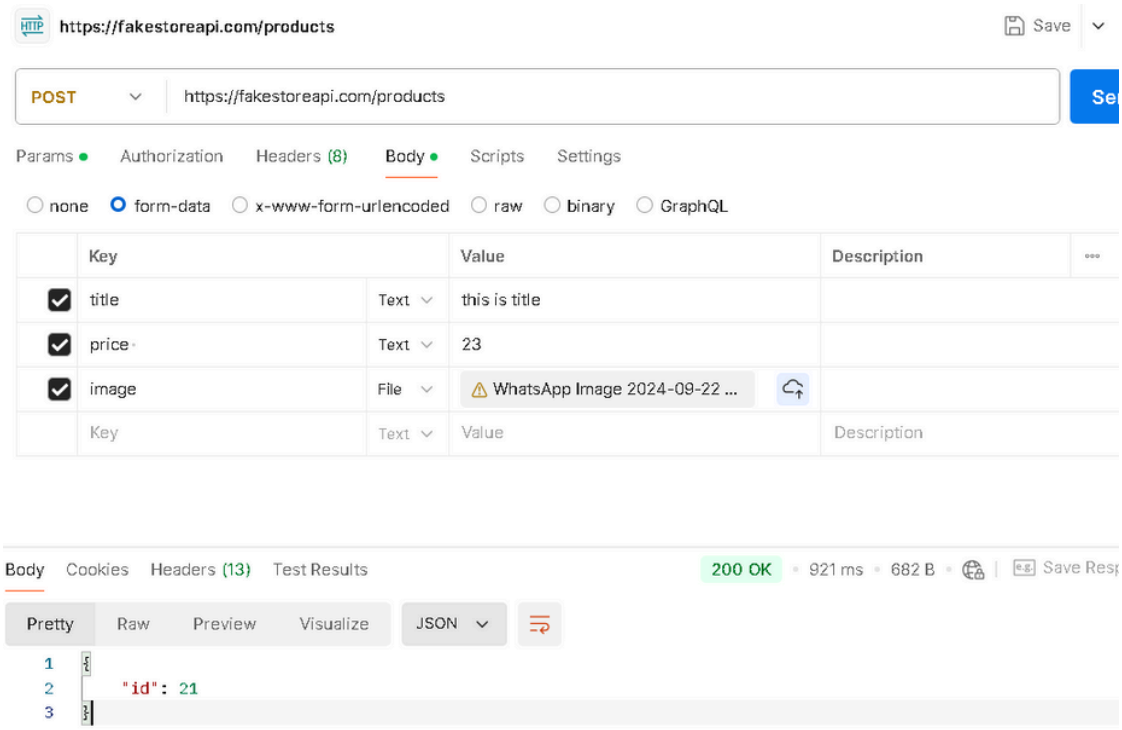
```
      ),
    ),
  );
 }
}
```

Flutter Upload File/Image To Rest API/Server Using Multiport Http Request with Null Safety

use this link to get access to fake api

🌐 Fake Store API

perform the following steps in postman



add the following dependencies

```
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.6
modal_progress_hud_nsn: ^0.5.1
http: ^1.2.2  #Handles making HTTP requests (GET, POST etc)
image_picker: ^1.1.2  #Enables users to pick images from their device's galle
```

```
File? image;
final _picker = ImagePicker();
bool showSpinner = false;
```

File? image;:

- This declares a nullable File variable named image. It will store the image file that a user selects or captures. Since it's nullable (File?), it can either hold a File object or be null if no image is picked.

final _picker = ImagePicker();:

- This initializes an instance of the ImagePicker class, which is part of the image_picker package. The _picker object will be used to access methods for picking images from the gallery or capturing them via the camera.

bool showSpinner = false;:

- This declares a boolean variable showSpinner that is initially set to false. It's likely used to control whether a loading spinner (progress indicator) should be shown on the UI while an image is being picked or uploaded. When the task is in progress, you can set it to true to show the spinner, and then back to false once the task is done

```
children: [
  Container(
    child: image == null? Center(child: Text('Pick Image'),),
  ),  // Container
  Container(
    child: Center(
      child: Image.file(File(
        image!.path).absolute,  // File
          height: 100,
        width: 200,
        fit: BoxFit.cover,
      ),  // Image.file
    ),  // Center
  )  // Container
],
)  // Column
```

Image Display

This code is used to display an image in a Flutter application. It consists of two Container widgets that are used to conditionally display either a text message or an image, depending on whether an image has been selected or not.

- The first Container widget displays a text message "Pick Image" if the image variable is null, indicating that no image has been selected.
- The second Container widget displays the selected image if the image variable is not null.

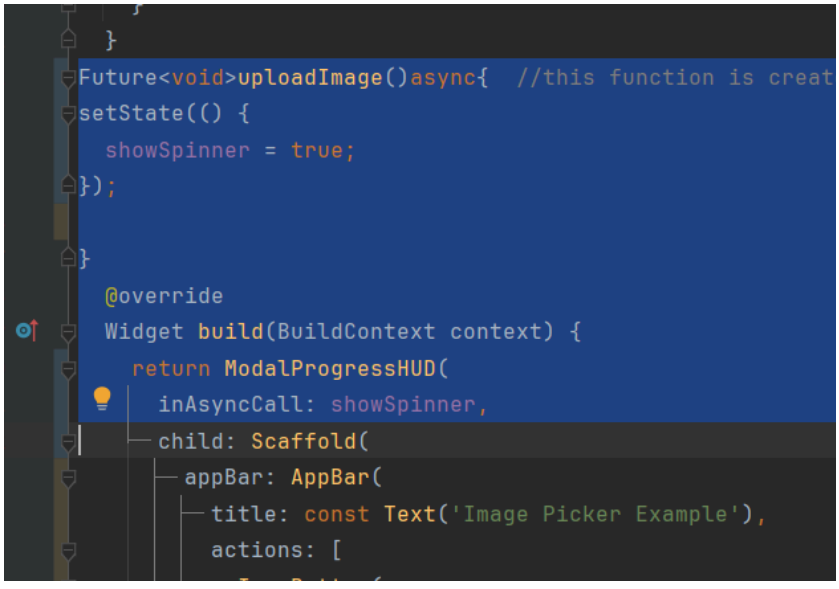
- The Image.file constructor is used to create an Image widget from a file.
- The File object is created from the image variable's path using the absolute property.
- The height, width, and fit properties are used to customize the display of the image.

```
Future<void> getImage() async {
 final pickedFile = await _picker.pickImage(source: ImageSource.gallery, imageQuality: 80);

 if (pickedFile != null) {
  setState(() {
   image = File(pickedFile.path);
  });
 } else {
  print("No image selected");
 }
}
```

we've got a function here called getImage() that's allowing users to pick an image from their device's gallery. Here's how it works:

- The function is marked as async, which means it can run in the background without blocking the rest of the app.
- Inside the function, _picker.pickImage() is called, which opens the device's gallery and lets the user select an image. The imageQuality parameter is set to 80, which compresses the image to reduce its file size.
- Once an image is selected, the pickedFile variable is checked to make sure it's not null. If it's not null, the setState() function is called to update the image variable with the selected image.
- If no image is selected, a message is printed to the console saying "No image selected".



The first code block is a function called uploadImage(). This function is marked as async, which means it can run in the background without blocking the rest of the app. When this

function is called, it sets the showSpinner variable to true using setState(). This is likely triggering a loading progress bar to appear on the screen.

The second code block is the build() method of a widget, which is responsible for building the user interface of the app. Here, we're using a ModalProgressHUD widget, which is a widget that can display a modal progress HUD (Heads-Up Display) to the user.

The inAsyncCall property of the ModalProgressHUD widget is set to showSpinner, which means that when showSpinner is true, the progress HUD will be displayed, and when it's false, the HUD will be hidden. This is how the temporarily loading progress bar is shown when the file is being uploaded.