**NEWS API ”Main Working Of Our Project” CODE:**

import 'package:flutter/material.dart';

import 'package:url\_launcher/url\_launcher.dart';

import 'news\_api.dart';

class HomeScreen extends StatefulWidget {

const HomeScreen({super.key});

@override

\_HomeScreenState createState() => \_HomeScreenState(); }

class \_HomeScreenState extends State<HomeScreen> {

// final NewsApi \_newsApi = NewsApi();

final NewsApi \_newsApi = NewsApi( apiKey:

'39fc8513af0648568fb3d8ca975195d4', // <-- replace or pass from main (avoid hardcoding) );

late Future<List<Article>> \_articlesFuture;

String \_selectedCategory = 'general';

final List<String> \_categories = [

'general',

'entertainment',

'business',

'health',

'science',

'sports',

'technology',

];

@override

void initState() {

super.initState();

// \_articlesFuture = \_newsApi.getTopHeadlines();

\_fetchArticles(); }

void \_fetchArticles() {

\_articlesFuture = \_newsApi.getTopHeadlines(category: \_selectedCategory); }

Future<void> \_refreshArticles() async {

setState(() {

// \_articlesFuture = \_newsApi.getTopHeadlines();

\_fetchArticles(); });

// wait for completion so RefreshIndicator finishes nicely

await \_articlesFuture; }

Future<void> \_openArticle(String? url) async {

if (url == null || url.trim().isEmpty) {

ScaffoldMessenger.of( context,

).showSnackBar(const SnackBar(content: Text('No URL available')));

return; }

// try parse; if no scheme present, prepend https://

Uri? uri = Uri.tryParse(url);

if (uri == null || !(uri.hasScheme)) { uri = Uri.tryParse('https://$url'); }

if (uri == null) {

ScaffoldMessenger.of( context,

).showSnackBar(const SnackBar(content: Text('Invalid URL')));

return; }

try {

final opened = await launchUrl(uri, mode: LaunchMode.externalApplication);

if (!opened) {

ScaffoldMessenger.of(context).showSnackBar(

const SnackBar(content: Text('Could not open the article')),

); }

} catch (e) {

ScaffoldMessenger.of( context,

).showSnackBar(SnackBar(content: Text('Error opening link: $e')));

} }

Widget \_buildImage(String? url) {

if (url == null || url.trim().isEmpty) { return Container(

height: 200,

color: Colors.grey[200],

child: const Center(child: Icon(Icons.image, size: 56)),

); }

// Show image with loading/error handlers

Uri? test = Uri.tryParse(url);

if (test == null || !test.hasScheme) {

// not a valid absolute url — show placeholder

return Container(

height: 200,

color: Colors.grey[200],

child: const Center(child: Icon(Icons.broken\_image, size: 56)),

); }

return ClipRRect(

borderRadius: const BorderRadius.vertical(top: Radius.circular(12)),

child: Image.network( url,

fit: BoxFit.cover, height: 200, width: double.infinity,

loadingBuilder: (context, child, progress) {

if (progress == null) return child;

return SizedBox(

height: 200,

child: Center(

child: CircularProgressIndicator(

value: progress.expectedTotalBytes != null

? progress.cumulativeBytesLoaded /

(progress.expectedTotalBytes ?? 1)

: null,

), ), ); },

errorBuilder: (context, error, stackTrace) => Container(

height: 200,

color: Colors.grey[200],

child: const Center(child: Icon(Icons.broken\_image, size: 56)),

), ), ); }

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text(

'All News Information',

style: TextStyle(fontWeight: FontWeight.bold),

),

actions: [

DropdownButton<String>(

value: \_selectedCategory,

items: \_categories.map((String category) {

return DropdownMenuItem<String>(

value: category, child: Text(category),

); }).toList(),

onChanged: (String? newValue) {

if (newValue != null) {

setState(() {

\_selectedCategory = newValue;

\_fetchArticles();

}); } }, ), ],

centerTitle: true, ),

body: FutureBuilder<List<Article>>(

future: \_articlesFuture,

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.waiting) {

return const Center(child: CircularProgressIndicator());

} else if (snapshot.hasError) {

return Center(child: Text('Error: ${snapshot.error}'));

} else if (!snapshot.hasData || snapshot.data!.isEmpty) {

return const Center(child: Text('No articles found.'));

} else {

final articles = snapshot.data!;

return RefreshIndicator(

onRefresh: \_refreshArticles,

child: ListView.builder(

physics: const AlwaysScrollableScrollPhysics(),

itemCount: articles.length,

itemBuilder: (context, index) {

final article = articles[index];

return Card(

margin: const EdgeInsets.symmetric(

horizontal: 10.0,

vertical: 8.0, ),

elevation: 3,

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(12), ),

child: InkWell(

borderRadius: BorderRadius.circular(12),

onTap: () => \_openArticle(article.url),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

\_buildImage(article.urlToImage),

Padding(

padding: const EdgeInsets.all(12.0), child: Column(

crossAxisAlignment: CrossAxisAlignment.start, children: [

Text(

article.title,

style: const TextStyle(

fontSize: 18,

fontWeight: FontWeight.bold, ), ),

const SizedBox(height: 8),

Text(

article.description.isNotEmpty

? article.description

: 'No description available.',

maxLines: 3,

overflow: TextOverflow.ellipsis,

style: TextStyle(

fontSize: 14,

color: Colors.grey[700],

), ),

const SizedBox(height: 8),

Row(

children: [

if (article.sourceName.isNotEmpty)

Text(

article.sourceName,

style: const TextStyle(

fontSize: 12,

fontWeight: FontWeight.w600,

), ),

const Spacer(),

if (article.publishedAt != null)

Text(

// simple formatting: date only

article.publishedAt!.toLocal().toIso8601String().split('T').first,

style: const TextStyle(fontSize: 12),

), ], ), ], ), ), ], ), ), ); }, ), ); } }, ), ); } }  
**CODE BREAKDOWN:**

Perfect approach bro ✅ — since you’re a beginner, if I try to explain this **entire file at once** it will be *too much*. Instead, let’s **break it into chunks** and go step by step like a teacher.

We’ll follow this plan:

1. **Class setup & State** (till initState)
2. **Data fetching methods** (\_fetchArticles, \_refreshArticles)
3. **Helper methods** (\_openArticle, \_buildImage)
4. **The UI (build method with Scaffold & FutureBuilder)**

👉 Today I’ll explain **Part 1: Class setup & State**. Then you can tell me to continue with Part 2 when ready.

**📍 Part 1 — Class Setup & State**

**Code piece:**

class HomeScreen extends StatefulWidget {

const HomeScreen({super.key});

@override

\_HomeScreenState createState() => \_HomeScreenState();

}

**Explanation:**

* HomeScreen is a **StatefulWidget**, because the data (articles, category) **changes during runtime**.
* createState() returns an instance of \_HomeScreenState.
* \_HomeScreenState is where the **logic and mutable data** live.

**Inside \_HomeScreenState**

final NewsApi \_newsApi = NewsApi(

apiKey: '39fc8513af0648568fb3d8ca975195d4',

);

late Future<List<Article>> \_articlesFuture;

String \_selectedCategory = 'general';

final List<String> \_categories = [

'general',

'entertainment',

'business',

'health',

'science',

'sports',

'technology',

];

**Explanation:**

1. final NewsApi \_newsApi
   * This is your helper class (from news\_api.dart) that **talks to the News API**.
   * The API key is hardcoded here (in real apps, better to hide it or pass it from main.dart).
2. late Future<List<Article>> \_articlesFuture;
   * Future = value that will be available **later** (async result).
   * Here it will hold the list of news articles when the API call completes.
   * late means: “I promise I’ll assign this before I use it.”
3. String \_selectedCategory = 'general';
   * Keeps track of which category the user selected (e.g., general, sports, etc.).
   * Initially set to **general**.
4. \_categories (a list of categories)
   * Used to build the dropdown menu in the AppBar.
   * User can switch between categories → updates \_selectedCategory.

**initState**

@override

void initState() {

super.initState();

\_fetchArticles();

}

**Explanation:**

* initState() runs **only once** when the screen is first created.
* Good place to:
  + Start fetching data
  + Initialize controllers
  + Set up listeners

Here, \_fetchArticles() is called to **load the initial set of news articles** when the app first opens.

✅ So, to summarize **Part 1**:

* HomeScreen is StatefulWidget because the news changes.
* \_newsApi handles fetching articles.
* \_articlesFuture stores the async data (articles).
* \_selectedCategory & \_categories are for filtering news.
* initState() loads the initial data.

**📍 Part 2 — Data Fetching Methods**

**Code Piece:**

void \_fetchArticles() {

\_articlesFuture = \_newsApi.getTopHeadlines(category: \_selectedCategory);

}

Future<void> \_refreshArticles() async {

setState(() {

\_fetchArticles();

});

await \_articlesFuture;

}

**🔹 \_fetchArticles()**

* Calls the getTopHeadlines method from NewsApi.
* Passes in the **currently selected category** (\_selectedCategory).
* Saves the result into \_articlesFuture.
  + This is important because \_articlesFuture is what the FutureBuilder (in your UI) listens to.

👉 So this function’s job: **“Go to the API kitchen and ask for the news in this category.”**

**🔹 \_refreshArticles()**

This is for **pull-to-refresh** (when the user drags down the screen to reload).

Steps:

1. setState(() { \_fetchArticles(); });
   * setState tells Flutter: “Hey, something changed, rebuild the UI.”
   * Inside it, we call \_fetchArticles() again → request fresh data.
2. await \_articlesFuture;
   * Waits for the new articles to finish loading.
   * This ensures the little **refresh spinner** disappears only when data is ready.

👉 Job: **“If the customer (user) says ‘give me the latest menu’, we throw away the old one and bring a fresh menu from the kitchen.”**

**🍔 Real-world Analogy (Restaurant Menu)**

Think of your app like a restaurant serving different categories of food (news).

* \_selectedCategory → the type of food the customer wants (e.g., Pizza, Burgers, Pasta).
* \_fetchArticles() → the waiter goes to the kitchen and asks the chef for the menu in that category.
* \_articlesFuture → the menu card the chef is preparing (might take time).
* \_refreshArticles() → the customer says, “Bring me today’s fresh menu” → waiter throws away the old menu and gets the updated one.

✅ So **Part 2 Summary**:

* \_fetchArticles() loads articles for the current category.
* \_refreshArticles() reloads everything when the user pulls down to refresh.
* Together, these make sure your app always shows **fresh, up-to-date news**.

**🔹 Part 3: Helper Methods in Your News App**

You had two helpers:

**1. \_openArticle(String url)**

* **Job** → Open a browser (or in-app web view) when the user taps an article.
* **Why needed** → Keeps onTap: short and readable.

Future<void> \_openArticle(String url) async {

final Uri uri = Uri.parse(url);

if (await canLaunchUrl(uri)) {

await launchUrl(uri, mode: LaunchMode.externalApplication);

} else {

throw 'Could not launch $url';

}

}

Usage inside your ListTile:

onTap: () => \_openArticle(article.url),

**2. \_buildImage(String? url)**

* **Job** → Show article image if available, otherwise show a fallback icon.
* **Why needed** → Prevents crashes when some articles don’t have images.

Widget \_buildImage(String? url) {

if (url == null || url.isEmpty) {

return const Icon(Icons.image\_not\_supported, size: 80);

} else {

return Image.network(

url,

fit: BoxFit.cover,

width: double.infinity,

height: 200,

errorBuilder: (context, error, stackTrace) {

return const Icon(Icons.broken\_image, size: 80);

},

);

}

}

Usage inside your card:

child: \_buildImage(article.urlToImage),

**🧾 Why these helpers matter?**

* **Cleaner UI code** → Instead of putting Image.network(...) everywhere with null checks, you call \_buildImage(url).
* **Separation of concerns** → \_openArticle handles navigation logic, not cluttering your UI.
* **Reusability** → If tomorrow you add a button or change image style, you only update the helper, not every place.

**🍔 Mini Analogy**

* \_openArticle = like **a shortcut key** to quickly open something.
* \_buildImage = like **a waiter deciding**: if food is ready, serve it; if not, bring a “no food available” sign.

**📍 Part 4: The UI (build method with Scaffold & FutureBuilder)**

This is where **everything comes together**:

* The app’s structure (Scaffold)
* The async data (FutureBuilder)
* The list of news (ListView)

**🔹 The build Method**

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text('All News Information'),

),

body: RefreshIndicator(

onRefresh: \_refreshArticles,

child: FutureBuilder<List<Article>>(

future: \_articlesFuture,

builder: (context, snapshot) {

...

},

),

),

);

}

**🏗 Step 1: Scaffold**

* Scaffold = the **basic layout structure** for a page.  
  Think of it as the **blueprint of a house** 🏠:
  + appBar = the roof/header of the house.
  + body = the main living area (your actual content).
  + (optional) drawer, floatingActionButton, bottomNavigationBar etc.

Here:

appBar: AppBar(title: const Text('All News Information')),

→ Gives a title bar at the top of the app.

**🏗 Step 2: RefreshIndicator**

body: RefreshIndicator(

onRefresh: \_refreshArticles,

child: ...

)

* Wraps your list in a **pull-to-refresh** widget.
* When user drags down → \_refreshArticles runs → reloads the data.

Analogy 🍔: Imagine pulling down a menu in a restaurant to see the **latest dishes available**.

**🏗 Step 3: FutureBuilder**

FutureBuilder<List<Article>>(

future: \_articlesFuture,

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.waiting) {

return const Center(child: CircularProgressIndicator());

} else if (snapshot.hasError) {

return Center(child: Text('Error: ${snapshot.error}'));

} else if (!snapshot.hasData || snapshot.data!.isEmpty) {

return const Center(child: Text('No articles found.'));

} else {

final articles = snapshot.data!;

return ListView.builder(

itemCount: articles.length,

itemBuilder: (context, index) {

final article = articles[index];

return Card(

child: ListTile(

title: Text(article.title),

onTap: () => \_openArticle(article.url),

subtitle: Text(article.description ?? ''),

leading: SizedBox(

width: 100,

child: \_buildImage(article.urlToImage),

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

**🔹 How FutureBuilder works**

* **future:**
  + Points to \_articlesFuture (the async result from \_fetchArticles).
  + Think of it as: “Here’s my waiter; check back later for food.”
* **builder:**
  + Runs every time the future updates.
  + snapshot = the waiter’s report about the food (data).

**🔹 The Four States**

1. **Loading** → ConnectionState.waiting
   * Show CircularProgressIndicator() (spinning loader).
   * Like waiting for food to arrive. 🍽
2. **Error** → snapshot.hasError
   * Show the error message.
   * Like the chef saying, “Sorry, we’re out of ingredients.” ❌
3. **Empty Data** → !snapshot.hasData || snapshot.data!.isEmpty
   * Show “No articles found.”
   * Like the waiter saying, “No dishes today.” 🚫
4. **Data Ready** → snapshot.data!
   * Build the ListView.builder with all articles.
   * Like the waiter finally serving a **list of dishes**. ✅

**🔹 ListView.builder**

* Efficiently builds only the visible list items (saves memory).
* Each Card contains a ListTile:
  + title → article title.
  + subtitle → article description.
  + leading → image on the left (\_buildImage).
  + onTap → open the article (\_openArticle).

So every news item = one **menu card** 📝 with details.

**📍 Detailed Breakdown of Each Component in the UI**

**🏗 1. Scaffold**

* The **backbone** of most Flutter screens.
* Provides a **default visual layout structure**: app bar, body, floating buttons, bottom nav, etc.
* Without Scaffold, you’d have to manually code these.

👉 Think of it like **building a house skeleton** 🏠:

* Roof = appBar
* Living room = body
* Side drawer = drawer
* Balcony = bottomNavigationBar
* Floating button = floatingActionButton

**🏗 2. AppBar**

appBar: AppBar(

title: const Text('All News Information'),

)

* A **material design header bar** at the top of the screen.
* Usually holds:
  + title (text in the middle or left)
  + actions (icons like search, settings)
  + leading (back button or drawer button).

👉 Like the **signboard** at the entrance of your restaurant 🍔 that tells visitors the name.

**🏗 3. RefreshIndicator**

RefreshIndicator(

onRefresh: \_refreshArticles,

child: ...

)

* A widget that adds **pull-to-refresh behavior** to scrollable content.
* Needs:
  + onRefresh → async function (what happens when pulled).
  + child → usually a scrollable widget (ListView, GridView).

👉 Analogy: pulling down the menu in a restaurant to ask the waiter for today’s specials.

**🏗 4. FutureBuilder**

FutureBuilder<List<Article>>(

future: \_articlesFuture,

builder: (context, snapshot) { ... }

)

* **Bridge between async data & UI**.
* Automatically rebuilds UI when the Future changes state.

States it handles:

* waiting → loading spinner.
* error → error message.
* empty → no data.
* done → show the list.

👉 It’s like your **waiter** 👨‍🍳:

* If kitchen is still cooking → shows "please wait."
* If kitchen fails → shows "sorry, no food."
* If food is ready → serves it on the table.

**🏗 5. ListView.builder**

ListView.builder(

itemCount: articles.length,

itemBuilder: (context, index) {

...

},

)

* A **scrollable list** that builds items **on-demand** (only when visible).
* More efficient than ListView(children: [...]).
* Parameters:
  + itemCount → total number of items.
  + itemBuilder → function that builds each item.

👉 Like a **menu book** 📖:

* Instead of writing every dish manually, you just say "there are 50 dishes" → waiter will **show only the ones needed now**.

**🏗 6. Card**

Card(

child: ListTile(...),

)

* A **material design card widget**.
* Used to make content look **elevated** with shadows and rounded corners.
* Great for grouping related content (like one article).

👉 Like each **dish card** in a restaurant menu 🍽 — neat and separate.

**🏗 7. ListTile**

ListTile(

title: Text(article.title),

subtitle: Text(article.description ?? ''),

leading: SizedBox(

width: 100,

child: \_buildImage(article.urlToImage),

),

onTap: () => \_openArticle(article.url),

)

* A **ready-made row layout** with:
  + leading → widget on the left (image/icon).
  + title → main text.
  + subtitle → smaller text under the title.
  + trailing → widget on the right (icon/button).
  + onTap → what happens when clicked.

👉 Like a **restaurant menu item row**:

* Left side → food photo.
* Middle → dish name + description.
* Right side → maybe a price or button.

**🏗 8. SizedBox**

SizedBox(

width: 100,

child: \_buildImage(article.urlToImage),

)

* A widget used to give **fixed width/height** to its child.
* Prevents widgets from expanding too much.

👉 Like **putting a frame** around a food picture so it always stays the same size.

**🏗 9. Text**

Text(article.title)

Text(article.description ?? '')

* Basic text display widget.
* Can be styled with TextStyle.

👉 Like **writing the dish’s name & description** on a menu card.

**🏗 10. Icon**

Icon(Icons.image\_not\_supported)

Icon(Icons.broken\_image)

* Shows **material design icons**.
* Used as placeholders or action symbols.

👉 Like drawing a **symbol** 🖼 if no photo of the dish is available.

**🏗 11. Image.network**

Image.network(

url,

fit: BoxFit.cover,

width: double.infinity,

height: 200,

errorBuilder: (context, error, stackTrace) {

return const Icon(Icons.broken\_image);

},

)

* Loads an image directly from the internet.
* Needs a URL string.
* errorBuilder → fallback if loading fails.

👉 Like showing a **photo of the dish** in the restaurant menu 📸.  
If photo missing → put a **"no photo available" icon**.

**✅ Final Teacher Recap**

* **Scaffold** = house structure 🏠
* **AppBar** = title board 🏷
* **RefreshIndicator** = pull-to-refresh 🍹
* **FutureBuilder** = async waiter 👨‍🍳
* **ListView.builder** = efficient menu book 📖
* **Card** = stylish plate 🍽
* **ListTile** = dish row (photo + name + desc) 📝
* **SizedBox** = frame for image 🎞
* **Text** = dish names 🖊
* **Icon** = placeholders / symbols 🔧
* **Image.network** = real dish photos 📸