**News API Json to Dart(newsApi.dart)**

import 'dart:convert';

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

/// Simple NewsAPI client. Pass your API key in the constructor

/// (avoid hardcoding in production).

class NewsApi {

  final String apiKey;

  final String baseUrl;

  NewsApi({

    required this.apiKey,

    this.baseUrl = 'https://newsapi.org/v2/top-headlines',

  });

  /// Get top headlines. You can pass country (default 'us'), category, pageSize.

  Future<List<Article>> getTopHeadlines({

    String country = 'us',

    String? category,

    int pageSize = 20,

  }) async {

    final Map<String, String> query = {

      'apiKey': apiKey,

      'country': country,

      'pageSize': pageSize.toString(),

    };

    if (category != null && category.isNotEmpty) {

      query['category'] = category;

    }

    final uri = Uri.parse(baseUrl).replace(queryParameters: query);

    final response = await http

        .get(uri)

        .timeout(

          const Duration(seconds: 10),

          onTimeout: () {

            throw Exception(

              'Request timed out — please check your connection.',

            );

          },

        );

    if (response.statusCode != 200) {

      // NewsAPI returns helpful body messages; include them for debugging

      final body = response.body.isNotEmpty ? response.body : 'no body';

      throw Exception('NewsAPI request failed (${response.statusCode}): $body');

    }

    final Map<String, dynamic> data = json.decode(response.body);

    if (data['status'] != 'ok') {

      throw Exception('NewsAPI error: ${data['message'] ?? data['status']}');

    }

    final List articlesJson = data['articles'] ?? <dynamic>[];

    final List<Article> articles = articlesJson

        .map((a) => Article.fromJson(a as Map<String, dynamic>))

        // filter out entries without a valid URL

        .where((a) => a.url.isNotEmpty)

        .toList();

    return articles;

  }

}

class Article {

  final String title;

  final String description;

  final String url;

  final String urlToImage;

  final String sourceName;

  final String author;

  final DateTime? publishedAt;

  Article({

    required this.title,

    required this.description,

    required this.url,

    required this.urlToImage,

    required this.sourceName,

    required this.author,

    required this.publishedAt,

  });

factory Article.fromJson(Map<String, dynamic> json) {

    final source = json['source'] ?? {};

    final published = json['publishedAt'];

    DateTime? when;

    if (published != null) {

      when = DateTime.tryParse(published.toString());

    }

    return Article(

      title: (json['title'] ?? '').toString(),

      description: (json['description'] ?? '').toString(),

      url: (json['url'] ?? '').toString(),

      urlToImage: (json['urlToImage'] ?? '').toString(),

      sourceName: (source['name'] ?? '').toString(),

      author: (json['author'] ?? '').toString(),

      publishedAt: when,

    );

  }

}

**CODE BREAKDOWN**

This file is a **Dart service** that connects your Flutter app to the **News API** and fetches the top news headlines 📰

We’ll break it into **4 main parts**:  
1️⃣ Imports  
2️⃣ NewsApi class (the brain)  
3️⃣ getTopHeadlines() method (the worker)  
4️⃣ Article class (the data model)

**🧩 1️⃣ Imports**

import 'dart:convert';

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

**💡 Explanation:**

* dart:convert → used to **decode JSON** data from the API response (convert text → Dart objects).
* package:http/http.dart → allows us to make **HTTP requests** (GET, POST, etc.) from Flutter to the internet.
* as http → gives a short name “http” to the package, so we can call things like http.get() easily.

🧠 **In short:**  
This part prepares the tools we’ll use to **talk to the web API** and **convert responses** into usable Dart objects.

**🌍 2️⃣ The NewsApi Class**

class NewsApi {

final String apiKey;

final String baseUrl;

NewsApi({

required this.apiKey,

this.baseUrl = 'https://newsapi.org/v2/top-headlines',

});

**💡 Explanation:**

* This class represents your **News API client** — like a helper that knows how to fetch data from NewsAPI.org.
* apiKey → your unique NewsAPI access key (required to use the API).
* baseUrl → the default endpoint where we fetch the “top headlines.”

**🧠 Constructor:**

NewsApi({

required this.apiKey,

this.baseUrl = 'https://newsapi.org/v2/top-headlines',

});

Means when you create an object like:

final api = NewsApi(apiKey: '39fc851e....');

It will use your API key, and if you don’t give a baseUrl, it will use the default one automatically.

✅ **In short:** This sets up the connection information.

**⚙️ 3️⃣ The getTopHeadlines() Method**

This is the **main function** that actually fetches news articles from the web.

**📦 Step 1: Method Definition**

Future<List<Article>> getTopHeadlines({

String country = 'us',

String? category,

int pageSize = 20,

}) async {

* Future<List<Article>> → means this function runs **asynchronously** and will return a **list of Article objects** later.
* It can take:
  + country → default is 'us'
  + category → like “sports”, “business”, etc. (optional)
  + pageSize → how many articles to fetch (default 20)

✅ So, it gives flexibility to fetch headlines based on **country or category**.

**🌐 Step 2: Build Query Parameters**

final Map<String, String> query = {

'apiKey': apiKey,

'country': country,

'pageSize': pageSize.toString(),

};

if (category != null && category.isNotEmpty) {

query['category'] = category;

}

💡 **Explanation:**

* Creates a **map** (key-value pairs) that holds all parameters we’ll send to the API.
* Converts pageSize (an integer) to string because URL parameters must be text.
* If category is provided, it’s added too.

🧠 Example:  
If country = 'in' and category = 'technology',  
your query becomes 👇

{

'apiKey': 'your\_api\_key\_here',

'country': 'in',

'pageSize': '20',

'category': 'technology'

}

**🔗 Step 3: Make the API URL**

final uri = Uri.parse(baseUrl).replace(queryParameters: query);

💡 Converts the baseUrl into a full **URL** with parameters attached.  
For example:

https://newsapi.org/v2/top-headlines?apiKey=xxxx&country=in&pageSize=20&category=technology

**🌍 Step 4: Make the HTTP GET Request**

final response = await http

.get(uri)

.timeout(

const Duration(seconds: 10),

onTimeout: () {

throw Exception(

'Request timed out — please check your connection.',

);

},

);

💡 Explanation:

* Sends a **GET request** to the News API server.
* Waits for the result using await.
* .timeout(...) ensures if no response comes within 10 seconds, we show an error message like:

“Request timed out — please check your connection.”

✅ This keeps your app from hanging forever if the network is slow.

**⚠️ Step 5: Handle HTTP Errors**

if (response.statusCode != 200) {

final body = response.body.isNotEmpty ? response.body : 'no body';

throw Exception('NewsAPI request failed (${response.statusCode}): $body');

}

💡 Explanation:

* API responses come with a **status code** (like 200 for OK, 404 for Not Found, 500 for Server Error).
* If it’s **not 200**, something went wrong.
* The code throws an exception with the error details.

✅ This helps you debug when API returns an unexpected response.

**🧠 Step 6: Decode the JSON Data**

final Map<String, dynamic> data = json.decode(response.body);

if (data['status'] != 'ok') {

throw Exception('NewsAPI error: ${data['message'] ?? data['status']}');

}

💡 Explanation:

* json.decode() → converts JSON text into a Dart map.
* Then it checks if "status": "ok" — if not, it throws an error message.

Example API response:

{

"status": "ok",

"totalResults": 38,

"articles": [ ... ]

}

If status ≠ ok → something’s wrong, and it shows the message.

**📰 Step 7: Convert JSON to Dart Objects**

final List articlesJson = data['articles'] ?? <dynamic>[];

final List<Article> articles = articlesJson

.map((a) => Article.fromJson(a as Map<String, dynamic>))

.where((a) => a.url.isNotEmpty)

.toList();

💡 Explanation:

* data['articles'] → contains a list of articles in JSON format.
* Each one is mapped (converted) into an Article Dart object using Article.fromJson.
* .where((a) => a.url.isNotEmpty) → removes any articles missing a valid URL.

✅ Finally returns a clean list of Article objects that your app can display.

**✅ Step 8: Return the List**

return articles;

Returns the final list of articles to the calling function.

**📰 4️⃣ The Article Class (Data Model)**

class Article {

final String title;

final String description;

final String url;

final String urlToImage;

final String sourceName;

final String author;

final DateTime? publishedAt;

Article({

required this.title,

required this.description,

required this.url,

required this.urlToImage,

required this.sourceName,

required this.author,

required this.publishedAt,

});

💡 Explanation:  
This defines the **structure of a news article** in your app.  
Each object holds:

* 📰 title → Headline text
* ✍️ description → Short summary
* 🔗 url → Link to the full article
* 🖼️ urlToImage → Image link
* 🏢 sourceName → News source (like BBC, CNN)
* 👤 author → Author’s name
* ⏰ publishedAt → Date/time of publishing

**Original code**

factory Article.fromJson(Map<String, dynamic> json) {

final source = json['source'] ?? {};

final published = json['publishedAt'];

DateTime? when;

if (published != null) {

when = DateTime.tryParse(published.toString());

}

return Article(

title: (json['title'] ?? '').toString(),

description: (json['description'] ?? '').toString(),

url: (json['url'] ?? '').toString(),

urlToImage: (json['urlToImage'] ?? '').toString(),

sourceName: (source['name'] ?? '').toString(),

author: (json['author'] ?? '').toString(),

publishedAt: when,

);

}

**Line-by-line explanation**

**factory Article.fromJson(Map<String, dynamic> json)**

* A **factory constructor**: it creates an Article instance from a JSON Map.
* Map<String, dynamic> is the common shape for decoded JSON in Dart.

**final source = json['source'] ?? {};**

* Tries to get the source field from JSON (NewsAPI returns something like {"name": "BBC"}).
* If json['source'] is null, it uses an empty map {} as fallback.
* **Caveat:** json['source'] might not be a Map (could be null or unexpected type). Using it later as source['name'] works only if source is a map-like object.

**final published = json['publishedAt'];**

* Reads the publishedAt value, usually an ISO8601 string like "2024-10-05T12:34:56Z".
* This could also be null or in another format (rare for NewsAPI but possible with other sources).

**DateTime? when; if (published != null) { when = DateTime.tryParse(published.toString()); }**

* when is nullable DateTime.
* DateTime.tryParse(...) attempts to parse an ISO-8601 string and returns null on failure (safe).
* .toString() ensures non-string values won't crash, but will produce possibly unexpected strings if published is not a standard date representation.
* **Note:** tryParse handles ISO strings and common formats; it will not parse epoch integers unless you convert them explicitly.

**title/description/url/... fields using (json['field'] ?? '').toString()**

* Each field gets the JSON value or fallback '' (empty string), then .toString() to ensure the field in the Article object is a non-null String.
* This prevents runtime null errors in UI code.
* **Tradeoff:** You lose the ability to detect a missing required field (e.g., url being empty might be a sign of bad data).

**sourceName: (source['name'] ?? '').toString(),**

* Accesses the name inside source. If source was {} this yields ''. If source wasn’t a map, this might throw or return unexpected results depending on the type.

**Potential problems & edge cases**

1. **source may not be a Map** — accessing source['name'] assumes a map. If source is a string or other type you'll get a runtime error.
2. **publishedAt format** — if not an ISO string (or is numeric epoch), DateTime.tryParse will return null. You might want to handle epoch integers explicitly.
3. **Empty urlToImage or url** — empty strings may cause Image.network('') to fail or show broken image; better to use a placeholder or conditionally show an image widget.
4. **Silent fallback to empty strings** — useful for preventing crashes, but hides missing data bugs. Sometimes you want to log or handle missing *required* fields (like url).
5. **Whitespace / extra characters** — calling .trim() on strings is often useful before displaying them.  
     
     
   **🧠 Summary in Hinglish**

| **Step** | **What Happens** |
| --- | --- |
| 1️⃣ | API key and base URL are set up |
| 2️⃣ | User calls getTopHeadlines() with country/category |
| 3️⃣ | Function builds the correct URL with query parameters |
| 4️⃣ | Sends HTTP GET request to NewsAPI.org |
| 5️⃣ | Waits for the response (with timeout) |
| 6️⃣ | If response is OK, decodes JSON data |
| 7️⃣ | Converts articles into Dart Article objects |
| 8️⃣ | Returns the list of readable articles to show in the app |