Widgets i Llista de personatges d'una API

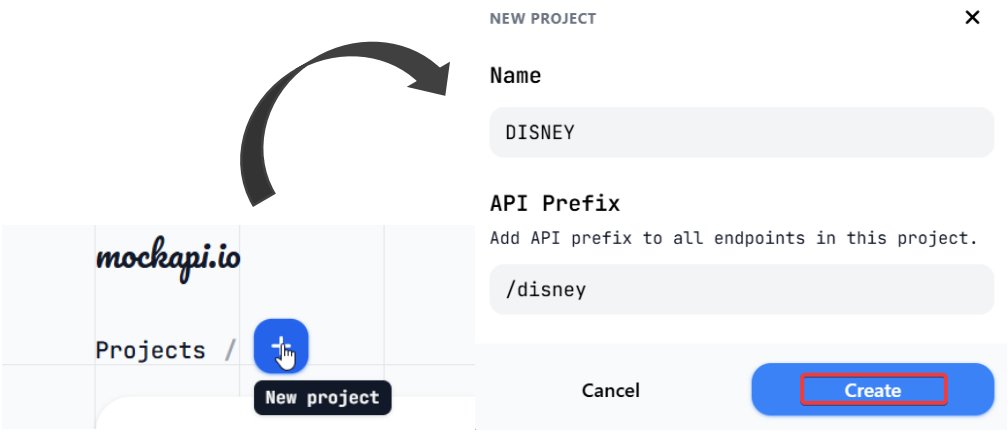
Ainhoa Sánchez Salvago

SALESIANS SARRIÀ | S2AM | MÒBILS

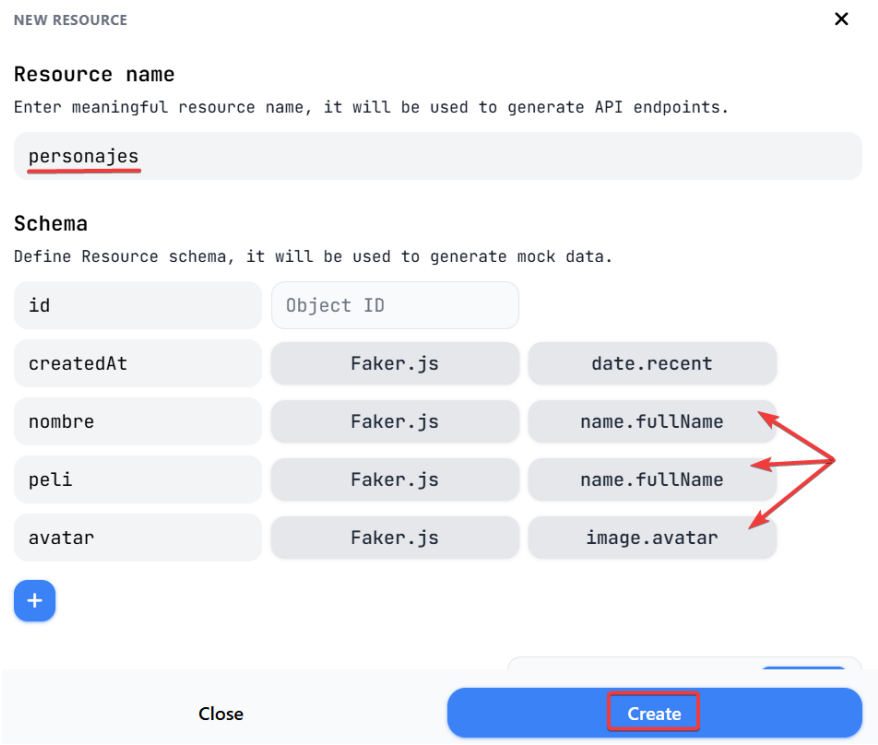
ÍNDEX

1. CREACIÓ DE LA LLISTA DE PERSONATGES

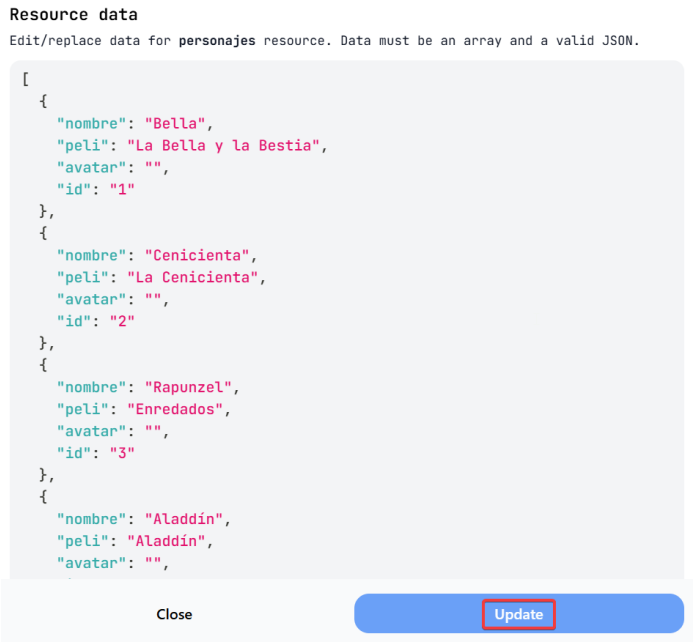
Per començar amb aquesta nova pràctica crearem la nostra pròpia API sobre personatges de "Disney", he intentat trobar-ne alguna ja feta però com cap s'adaptava ben bé a l'estil de l'aplicació doncs he optat per utilitzar l'aplicació de "mockapi" i fer-la des de zero. En primer lloc, per crear el nostre primer projecte ens demanarà registrar-nos, cliquem sobre l'opció de "nou projecte" i seguidament, ens demanarà posar-li un nom al nou projecte i el nom que volem que vagi davant dels noms de tots els "Endpoints" a la ruta del nostre arxiu.



Un cop creat, si cliquem dins del projecte allà on posa crear un nou recurs, clicarem i podrem crear els diferents camps que volem que tingui el nostre arxiu Json.

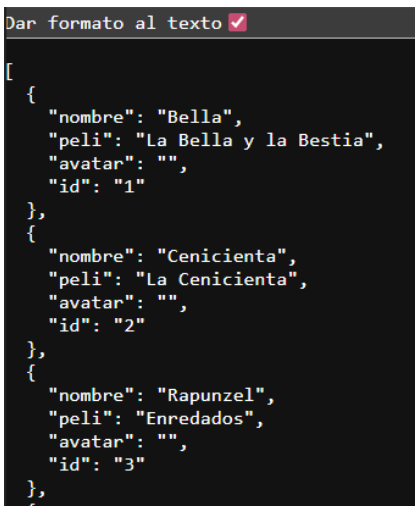


Un cop creat, podrem establir manualment el nombre de registres que volem que tingui l'arxiu. D'altra banda, si cliquem sobre "Data", se'ns obrirà una pestanya on podrem editar totes les dades que vulguem i per guardar els canvis només haurem de clicar al botó d'actualitzar.



Ara, ja tenim la nostra api creada i si enganxem el següent enllaç al navegador podrem veure totes les dades que conté en format Json.

https://692475d63ad095fb84744a6e.mockapi.io/disney/personajes



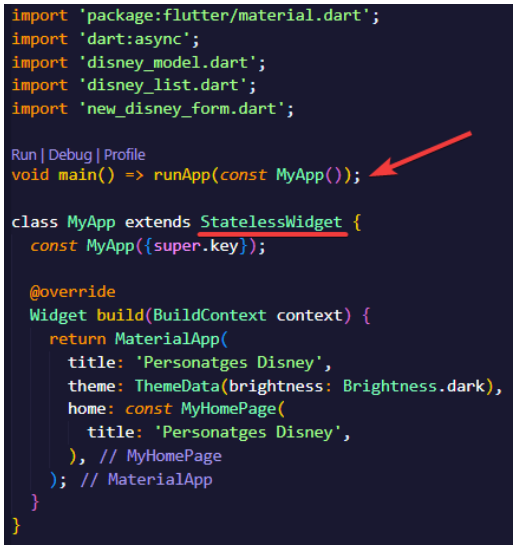
2. MODIFICACIÓ DEL CODI BASE

2.1. ADAPTACIÓ DEL CODI A LA NOVA TEMÀTICA

Com que partim d'un codi ja estructurat i funcional, el primer i més important és adaptar-lo a la nova API que hem creat sobre personatges de Disney. Cal tenir en compte que, en començar a modificar un sol arxiu, és probable que la resta comencin a fallar, això és totalment normal, ja que tots els fitxers estan connectats entre si. En primer lloc, comencem canviant els noms dels arxius, així treballem d'una manera més neta i organitzada. En acabat, passem a modificar els arxius un per un per aconseguir les funcionalitats i estils desitjats.

2.1.1. MAIN.DART

Comencem el codi important les diferents biblioteques o classes necessàries per realitzar el treball. Seguidament, trobem el "main" que és el punt de partida de qualsevol programa Dart i amb el runApp iniciem tot l'arbre de "widgets". Aquest widget, al no tenir un estat intern que canviï durant l'execució de l'aplicació, utilitzem el "StatelessWidget", amb això només es configura l'aplicació i no necessita redibuixar-se. Un cop fet això, amb el "MaterialApp" definim el títol, el tema global i la pàgina inicial amb el títol que mostrarem a l'appBar.



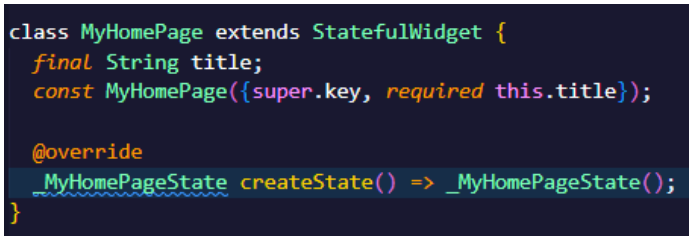
```dart
import 'package:flutter/material.dart';
import 'dart:async';
import 'disney_model.dart';
import 'disney_list.dart';
import 'new_disney_form.dart';

Run | Debug | Profile
void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Personatges Disney',
      theme: ThemeData(brightness: Brightness.dark),
      home: const MyHomePage(
        title: 'Personatges Disney',
      ), // MyHomePage
    ); // MaterialApp
  }
}
```

Ara, si que utilitzem un "StatefulWidget" perquè en aquesta pantalla carreguem les dades de la nostra api, actualitzem la interfície i tenim l'opció d'afegir elements a la llista. Totes aquestes accions requereixen un estat dinàmic.



```dart
class MyHomePage extends StatefulWidget {
  final String title;
  const MyHomePage({super.key, required this.title});

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

Seguidament, declarem una classe privada dins la qual creem la llista on es guardaran els personatges i una variable la qual ens ajudarà a controlar quan volem mostrar el símbol de "Carregant". Un cop declarades aquestes variables, s'executa el initState(), aquest s'executa una vegada s'ha creat el widget , és el millor lloc per carregar les dades ja que permet que la llista es torni a carregar cada cop que aquestes canviïn i ho fem amb la funció de "_loadDisneyCharacters".

```
class _MyHomePageState extends State<MyHomePage> {
  List<Disney> personatges = [];
  bool isLoading = true;

  @override
  void initState() {
    super.initState();
    _loadDisneyCharacters();
  }
```

El següent mètode ens retorna una List<Disney>, és a dir, un cop rep les dades de l'API actualitza la llista i desactiva el "isLoading". És una de les parts que hem hagut de canviar en comparació al codi dels "Digimons" ja que, en aquest es carregaven les dades en local.

```
  Future<void> _loadDisneyCharacters() async {
    // ignore: avoid_print
    print('Iniciando carga de personajes...');
    final characters = await Disney.loadDisneyCharacters();
    // ignore: avoid_print
    print('Personajes cargados: ${characters.length}');

    setState(() {
      personatges = characters;
      isLoading = false;
    });
  }
```

A continuació, en el codi, trobem un mètode amb el qual obrim una nova pantalla amb el "AddDisneyForm", aquesta pantalla és la que obrim quan volem afegir un nou personatge a la nostra llista. Aquest, espera si el nouPersonatge és nul o no, si no és nul afegeix el personatge i torna a dibuixar al llista.

```
Future _showNewDisneyForm() async {
  Disney? newPersonatge = await Navigator.of(context).push(MaterialPageRoute(builder: (BuildContext context) {
    return const AddDisneyFormPage();
  })); // MaterialPageRoute

  if (newPersonatge != null) {
    setState(() {
      personatges.add(newPersonatge);
    });
  }
}
```

Finalment, dins de l'arxiu "main.dart" trobem el "Widget Build" i l'estructura visual de la pantalla. En primer lloc, tenim el "Scaffold" que com ja sabem és l'estructura principal i després, trobem el "AppBar" la qual conté el títol i el botó d'afegir. Per acabar, trobem el "body" el qual mostra el "CircularProgressIndicator" sempre que el "isLoading" sigui "true" i quan les dades estan carregades mostra la llista.

```dart
@override
Widget build(BuildContext context) {
  var key = GlobalKey<ScaffoldState>();
  return Scaffold(
    key: key,
    appBar: AppBar(
      title: Text(widget.title, style: const TextStyle(color: Colors.white)),
      centerTitle: true,
      backgroundColor: const Color.fromARGB(255, 0, 0, 0),
      actions: <Widget>[
        IconButton(
          icon: const Icon(Icons.add),
          onPressed: _showNewDisneyForm,
        ), // IconButton
      ], // <Widget>[]
    ), // AppBar
    body: Container(
      color: const Color.fromARGB(255, 0, 0, 0),
      child: isLoading
          ? const Center(child: CircularProgressIndicator())
          : Center(
              child: DisneyList(personatges),
            )), // Center // Container
  ); // Scaffold
}
}
```

2.1.2. DISNEY_MODEL.DART

En el següent arxiu, comencem important les diferents llibreries que necessitarem al llarg del codi. Seguidament, creem el que entenem per objecte Disney on l'única informació obligatòria és el nom i la variable "rating" la inicialitzem en 10. Utilitzem el símbol d'interrogació després de definir la String ja que les dades poden no existir al crear el personatge però completar-se després des de l'API. En les variables on posem el '?' significa que aquestes poden ser nul·les.

```dart
import 'dart:convert';
import 'dart:io';
import 'dart:async';

class Disney {
  String? id;
  final String nombre;
  String? peli;
  String? avatar;
  String? apiname;
  int rating = 10;

  Disney(this.nombre);
```

A continuació, creem un objecte Disney a partir d'un Map (Diccionari clau-valor). On el nom i el rating veiem '??' això significa que si algun d'aquests valors és nul utilitzarà l'altre definició que li donem.

```dart
  factory Disney.fromJson(Map<String, dynamic> json) {
    final disney = Disney(json['nombre'] as String? ?? '');
    disney.id = json['id'] as String?;
    disney.peli = json['peli'] as String?;
    disney.avatar = json['avatar'] as String?;
    disney.rating = json['rating'] as int? ?? 10;
    return disney;
  }
```

El següent mètode fa tot el contrari, converteix l'objecte Disney en un mapa per poder transformar-lo després a json i així enviar-ho a l'API, com per exemple quan creem un nou personatge i volem que se'ns afegeixi a la llista.

```
Map<String, dynamic> toJson() {
  return {
    'nombre': nombre,
    'peli': peli,
    'avatar': avatar,
    'rating': rating,
  };
}
```

Tot seguit, trobem el mètode de "getImageURL" aquest mira si la imatge és nul·la i si ho és no fa res, així evitem errors. Després, utilitzem el httpCleint per definir la url de l'API i ho fem utlitzant la variable "apiname". En primer lloc, fem un get a l'api per tal d'obtenir les dades, les transformem a a text (utf8) i després a json.

```
Future getImageUrl() async {
  if (avatar != null) {
    return;
  }

  HttpClient http = HttpClient();
  try {
    apiname = nombre.toLowerCase().trim();
    final uri = Uri.https('mockapi.io', '/disney/personajes/$apiname');
    final request = await http.getUrl(uri);
    final response = await request.close();
    final responseBody = await response.transform(utf8.decoder).join();

    List data = json.decode(responseBody);
    avatar = data[0]["img"];
    peli = data[0]["peli"];
  } catch (exception) {
    //print(exception);
  }
}
```

Ara, trobem el mètode que carrega la llista, definim una variable de llista i la inicialitzem buida. Si la connexió triga més de 15 segons es cancel·la.

```
static Future<List<Disney>> loadDisneyCharacters() async {
  final List<Disney> personatgesDisney = [];
  HttpClient http = HttpClient();
  http.connectionTimeout = const Duration(seconds: 15);

  const ruta = '/api/disney/personajes';
```

Seguidament, si la resposta es un "status 200" transforma les dades a text i entrem en una comparativa que fa que el codi sigui més resistent a canvis dins de l'API i es que per tenir un codi més eficient hem de tenir en compte que l'API ens pot retornar o bé, una llista o bé, un objecte que dins contindrà una llista. A l'hora de crear la llista, recorre les dades una per una i ho converteix en un objecte Disney, si hi ha algun que no pot convertir, ho ignora i segueix amb la

resta evitant que es trenqui tota la llista. Utilitzem el control d'excepcions i mostrem missatges per pantalla per saber que està passant en el moment en el que ens falla el codi.

```
try {
  var uri = Uri.https('692475d63ad095fb84744a6e.mockapi.io', ruta);

  var request = await http.getUrl(uri);
  var response = await request.close();
  var responseBody = await response.transform(utf8.decoder).join();


  if (response.statusCode == 200) {

    dynamic data = json.decode(responseBody);

    // Si la resposta és una llista directa
    if (data is List) {

      for (int i = 0; i < data.length; i++) {
        try {
          final disney = Disney.fromJson(data[i] as Map<String, dynamic>);
          if (disney.nombre.isNotEmpty) {
            personatgesDisney.add(disney);
          }
        } catch (e) {
          // ignore: avoid_print
          print('Error parsing item $i: $e');
        }
      }
    }
    // Si és un objecte amb la llista dins
    else if (data is Map) {
      // Intentar trobar la llista dins de l'objecte
      List<dynamic>? records;

      if (data['records'] != null && data['records'] is List) {
        records = data['records'] as List<dynamic>;
      } else if (data['data'] != null && data['data'] is List) {
        records = data['data'] as List<dynamic>;
      } else if (data['items'] != null && data['items'] is List) {
        records = data['items'] as List<dynamic>;
      } else if (data['personajes'] != null && data['personajes'] is List) {
        records = data['personajes'] as List<dynamic>;
      } else {
        // Buscar qualsevol key que contingui una llista
        for (var key in data.keys) {
          if (data[key] is List) {
            records = data[key] as List<dynamic>;
            break;
          }
        }
      }
```

```
        if (records != null) {
          for (int i = 0; i < records.length; i++) {
            try {
              final disney = Disney.fromJson(records[i] as Map<String, dynamic>);
              if (disney.nombre.isNotEmpty) {
                personatgesDisney.add(disney);
              }
            } catch (e) {
              // ignore: avoid_print
              print('Error parsing item $i: $e');
            }
          }
        }
      }

      return personatgesDisney;
    }
  } catch (e)
  {
    // ignore: avoid_print
    print('Error fetching from $ruta: $e');
  }

  // ignore: avoid_print
  print('No s\'ha pogut carregar des de cap URL');
  return personatgesDisney;
}
```

En el següent mètode, ens encarreguem d'enviar les dades a un servidor per guardar-los. En el moment en que rebem una resposta del servidor valorem aquesta i si es exitosa, ens retorna l'objecte creat i si no ens retorna un objecte nul amb un missatge d'error. Apart del control d'excepcions que hem estat executant durant tot el codi, tenim el finally que s'executa sempre, hi hagi error o no, en aquest cas s'utilitza per tancar el "http".

```dart
static Future<Disney?> createDisneyCharacter(Disney disney) async {
  HttpClient http = HttpClient();
  http.connectionTimeout = const Duration(seconds: 10);

  try {
    //var uri = Uri.https('692475d63ad095fb84744a6e.mockapi.io', '/disney/personajes');
    var uri = Uri.https('692475d63ad095fb84744a6e.mockapi.io', '/disney/personajes');
    var request = await http.postUrl(uri);
    request.headers.set('Content-Type', 'application/json');

    // Ens assegurem que les dades no siguin null
    final data = {
      'nombre': disney.nombre,
      'peli': disney.peli ?? '',
      'avatar': disney.avatar ?? '',
      'rating': disney.rating,
    };

    request.write(json.encode(data));

    var response = await request.close();
    var responseBody = await response.transform(utf8.decoder).join();

    if (response.statusCode == 201 || response.statusCode == 200) {
      Map<String, dynamic> responseData = json.decode(responseBody);
      return Disney.fromJson(responseData);
    } else {
      // ignore: avoid_print
      print('Error: ${response.statusCode} - $responseBody');
      return null;
    }
  } catch (e) {
    return null;
  } finally {
    http.close();
  }
}
}
```

2.1.3. DISNEY DETAIL PAGE

En el moment en el que cliquem sobre qualsevol dels personatges de la nostra llista, se'ns obra aquesta pantalla on ens mostra amb més detall les dades rebudes dels personatges i ens permet interactuar amb elles.

En primer lloc, aquest codi rep obligatòriament un objecte Disney (personatge que l'usuari selecciona) i com la pantalla anirà canviant és un "StatefulWidget". Seguidament, com volem utilitzar animacions per les imatges dels personatges sense que ens doni error el "animationController", utilitzem el "TickerProviderStateMixin". Dins d'aquesta classe, definim les variables d'estat i aquestes guarden per exemple, els valors de la valoració que l'usuari li posa al personatge. Tot seguit, tenim el initState() que només s'executa quan es monta la

pantalla. D'altra banda, en aquesta classe també configurem les animacions. Finalment, netegem aquesta pantalla.

```dart
import 'package:flutter/material.dart';
import 'disney_model.dart';


class DisneyDetailPage extends StatefulWidget {
  final Disney disney;
  const DisneyDetailPage(this.disney, {super.key});

  @override
  // ignore: library_private_types_in_public_api
  _DisneyDetailPageState createState() => _DisneyDetailPageState();
}

class _DisneyDetailPageState extends State<DisneyDetailPage> with TickerProviderStateMixin {
  final double disneyAvarterSize = 150.0;
  late double _sliderValue;
  late AnimationController _animationController;
  late Animation<double> _rotationAnimation;

  @override
  void initState() {
    super.initState();
    _sliderValue = widget.disney.rating.toDouble();

    _animationController = AnimationController(
      duration: const Duration(milliseconds: 800),
      vsync: this,
    ); // AnimationController

    _rotationAnimation = Tween<double>(begin: 0.0, end: 1.0).animate(_animationController);
    _animationController.forward();
  }

  @override
  void dispose() {
    _animationController.dispose();
    super.dispose();
  }
```

En segon lloc, trobem el widget de "addYourRating" en aquest controlem els canvis que es produeixen quan l'usuari mou la barra de valoració visualment gràcies al "OnChanged()". Cada cop que es mou una mica la barra es crida a aquesta funció.

```
Widget get addYourRating {
  return Column(
    children: <Widget>[
      Container(
        padding: const EdgeInsets.symmetric(vertical: 16.0, horizontal: 16.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: <Widget>[
            Flexible(
              flex: 1,
              child: Slider(
                activeColor: const Color.fromARGB(255, 132, 0, 255),
                min: 0.0,
                max: 10.0,
                value: _sliderValue,
                onChanged: (newRating) {
                  setState(() {
                    _sliderValue = newRating;
                  });
                },
              ), // Slider
            ), // Flexible
            Container(
                width: 50.0,
                alignment: Alignment.center,
                child: Text(
                  '${_sliderValue.toInt()}',
                  style: const TextStyle(color: Color.fromARGB(255, 255, 255, 255), fontSize: 25.0),
                )), // Text // Container
          ], // <Widget>[]
        ), // Row
      ), // Container
      submitRatingButton,
    ], // <Widget>[]
  ); // Column
}
```

El següent que fem és prendre el valor de la barra de valoració el converteix en enter i sobreescriu el valor del objecte original. Després, un cop cliquem sobre el botó de “Guardar” tanca la pantalla actual i torna a l’anterior i en aquesta se’ns mostrarà també el nou valor del “rating”.

```
void updateRating() {
  setState(() {
    widget.disney.rating = _sliderValue.toInt();
  });
  Navigator.of(context).pop();
}
```

```
Widget get submitRatingButton {
  return ElevatedButton(
    onPressed: () => updateRating(),
    child: const Text('Guardar'),
  ); // ElevatedButton
}
```

Dins del següent bloc, construïm la imatge circular del personatge. Utilitzem el “Hero” que fa que si la imatge té el mateix “tag” passarà de la llista fins la posició corresponent a l’obrir la pantalla de detalls, és a dir, és únicament un efecte visual. Tot seguit, definim l’estil que volem que tingui la imatge i la transició que volem que facin.

```
Widget get disneyImage {
  return Hero(
    tag: widget.disney,
    child: RotationTransition(
      turns: _rotationAnimation,
      child: Container(
        height: disneyAvarterSize,
        width: disneyAvarterSize,
        constraints: const BoxConstraints(),
        decoration: BoxDecoration(
            shape: BoxShape.circle,
            boxShadow: const [
              BoxShadow(offset: Offset(1.0, 2.0), blurRadius: 2.0, spreadRadius: -1.0, color: Color(0x33000000)),
              BoxShadow(offset: Offset(2.0, 1.0), blurRadius: 3.0, spreadRadius: 0.0, color: Color(0x24000000)),
              BoxShadow(offset: Offset(3.0, 1.0), blurRadius: 4.0, spreadRadius: 2.0, color: Color(0x1f000000))
            ],
            image: DecorationImage(fit: BoxFit.cover, image: NetworkImage(widget.disney.avatar ?? ""))), // BoxDecoration
      ), // Container
    ), // RotationTransition
  ); // Hero
}
```

En el widget de “getRating” construïm la fila on mostrem la nota de valoració del personatge amb una estrella.

```
Widget get rating {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      const Icon(
        Icons.star,
        size: 40.0,
        color: Color.fromARGB(255, 255, 255, 255),
      ), // Icon
      Text('${widget.disney.rating}/10', style: const TextStyle(color: Color.fromARGB(255, 255, 255, 255), fontSize: 30.0))
    ], // <Widget>[]
  ); // Row
}
```

Ara, definim l’estil de la part superior de la pantalla, on trobem el nom, la pel·lícula, la imatge i la puntuació.

```
Widget get disneyCharProfile {
  return Container(
    padding: const EdgeInsets.symmetric(vertical: 32.0),
    decoration: const BoxDecoration(
      color: Color.fromARGB(255, 0, 0, 0),
    ), // BoxDecoration
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
        disneyImage,
        Text(widget.disney.nombre, style: const TextStyle(color: Color.fromARGB(255, 255, 255, 255), fontSize: 32.0)),
        Text('${widget.disney.peli}', style: const TextStyle(color: Color.fromARGB(255, 255, 255, 255), fontSize: 20.0)),
        Padding(
          padding: const EdgeInsets.only(top: 20.0),
          child: rating,
        ), // Padding
      ], // <Widget>[]
    ), // Column
  ); // Container
}
```

Finalment, dins del codi d’aquest arxiu trobem l’estructura principal i en relació amb l’estil utilitzem una “ListView” per poder fer “scroll” ja que, sinó podria ser que no hi càpigues el contingut depenent del mòbil.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color.fromARGB(255, 0, 0, 0),
    appBar: AppBar(
      backgroundColor: const Color.fromARGB(255, 0, 0, 0),
      title: Text('Detalls sobre ${widget.disney.nombre}'),
    ), // AppBar
    body: ListView(
      children: <Widget>[disneyCharProfile, addYourRating],
    ), // ListView
  ); // Scaffold
 }
}
```

2.1.4. DISNEY_CARD.DART

Amb aquest arxiu, definim el component visual individual de cada element de la llista. És a dir, la "targeta" que veiem per cada personatge. En primer lloc, rep un objecte Disney i el gestiona. Amb la variable "renderUrl" quan la app es descarrega la URL de la imatge i aquesta variable al omple i provoca que la pantalla s'actualitzi. Amb el initState() s'executa un sol cop i cridem al mètode "renderDisneyPic" per buscar la foto a internet.

```dart
import 'package:digimon_2526/disney_model.dart';
import 'disney_detail_page.dart';
import 'package:flutter/material.dart';


class DisneyCard extends StatefulWidget {
  final Disney disney;

  const DisneyCard(this.disney, {super.key});

  @override
  _DisneyCardState createState() => _DisneyCardState(disney);
}

class _DisneyCardState extends State<DisneyCard> {
  Disney disney;
  String? renderUrl;

  _DisneyCardState(this.disney);

  @override
  void initState() {
    super.initState();
    renderDisneyPic();
  }
```

```dart
void renderDisneyPic() async {
  await disney.getImageUrl();
  if (mounted) {
    setState(() {
      renderUrl = disney.avatar;
    });
  }
}
```

En el "widget disneyImage" gestiona la transició que hi ha entre el "carregant" i l'aparició de la foto. Gestionem el que l'usuari veu mentre que la foto carrega i la transició en si.

```
Widget get disneyImage {
  var disneyAvatar = Hero(
    tag: disney,
    child: Container(
      width: 100.0,
      height: 100.0,
      decoration:
          BoxDecoration(shape: BoxShape.circle, image: DecorationImage(fit: BoxFit.cover, image: NetworkImage(renderUrl ?? ''))),
    ), // Container
  ); // Hero

  var placeholder = Container(
    width: 100.0,
    height: 100.0,
    decoration: const BoxDecoration(
        shape: BoxShape.circle,
        gradient:
            LinearGradient(begin: Alignment.topLeft, end: Alignment.bottomRight, colors: [Colors.black54, Colors.black, Color.fromARGB(255, 84, 110, 122)])), // BoxDecoration
    alignment: Alignment.center,
    child: const Text(
      'DISNEY',
      textAlign: TextAlign.center,
    ), // Text
  ); // Container

  var crossFade = AnimatedCrossFade(
    firstChild: placeholder,
    secondChild: disneyAvatar,
    // ignore: unnecessary_null_comparison
    crossFadeState: renderUrl == null ? CrossFadeState.showFirst : CrossFadeState.showSecond,
    duration: const Duration(milliseconds: 1000),
  ); // AnimatedCrossFade

  return crossFade;
}
```

Finalment, trobem el “widget disneyCard” que dibuixa l’estil del rectangle, en aquest cas, lila amb les puntes arrodonides i amb espai a la esquerre perquè la imatge no tapi el text.

```
Widget get disneyCard {
  return Positioned(
    right: 0.0,
    child: SizedBox(
      width: 290,
      height: 115,
      child: Card(
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(15.0)),
        color: const Color.fromARGB(255, 165, 102, 219),
        child: Padding(
          padding: const EdgeInsets.only(top: 8, bottom: 8, left: 64),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: <Widget>[
              Text(
                widget.disney.nombre,
                style: const TextStyle(color: Color.fromARGB(255, 0, 0, 0), fontSize: 27.0),
              ), // Text
              Row(
                children: <Widget>[
                  const Icon(Icons.star, color: Color(0xFF000000)),
                  Text(': ${widget.disney.rating}/10', style: const TextStyle(color: Color(0xFF000000), fontSize: 14.0))
                ], // <Widget>[]
              ) // Row
            ], // <Widget>[]
          ), // Column
        ), // Padding
      ), // Card
    ), // SizedBox
  ); // Positioned
}
```

Seguidament, naveguem per la pantalla de detall que hem comentat anteriorment i ajuntem les diferents “peces” del codi dins del build.

```dart
showDigimonDetailPage() async {
  await Navigator.of(context).push(MaterialPageRoute(builder: (context) {
    return DisneyDetailPage(disney);
  })); // MaterialPageRoute
  // Cuando volvemos, refrescamos la card
  if (mounted) {
    setState(() {});
  }
}

@override
Widget build(BuildContext context) {
  return InkWell(
    onTap: () => showDigimonDetailPage(),
    child: Padding(
      padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),
      child: SizedBox(
        height: 115.0,
        child: Stack(
          children: <Widget>[
            disneyCard,
            Positioned(top: 7.5, child: disneyImage),
          ], // <Widget>[]
        ), // Stack
      ), // SizedBox
    ), // Padding
  ); // InkWell
}
```

2.1.5. DISNEY_LIST

Aquest arxiu és el contenidor visual principal encarregat de mostrar la nostra col·lecció de personatges. Rep la llista de dades i la converteix en una llista “scrollable”. Per cada fila detectada, agafa una dada de la llista i crea una instancia de DisneyCard per visualitzar-la.

```dart
import 'package:digimon_2526/disney_card.dart';
import 'package:flutter/material.dart';
import 'disney_model.dart';

class DisneyList extends StatelessWidget {
  final List<Disney> personatgesList;
  const DisneyList(this.personatgesList, {super.key});

  @override
  Widget build(BuildContext context) {
    return _buildList(context);
  }

  ListView _buildList(context) {
    return ListView.builder(
      itemCount: personatgesList.length,
      // ignore: avoid_types_as_parameter_names
      itemBuilder: (context, int) {
        return DisneyCard(personatgesList[int]);
      },
    ); // ListView.builder
  }
}
```

2.1.6. NEW_DISNEY_FORM.DART

Arribem al nostre últim arxiu de codi, en aquest, definim la pantalla de formulari on l’usuari pot introduir les dades d’un nou personatge i enviar-los a l’API.

```dart
import 'package:digimon_2526/disney_model.dart';
import 'package:flutter/material.dart';


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

  @override
  _AddDisneyFormPageState createState() => _AddDisneyFormPageState();
}
```

Seguidament, definim les variables estat, això ens permet llegir el que l'usuari ha escrit o modificar-ho des del codi i es crea un per cada camp.

```dart
class _AddDisneyFormPageState extends State<AddDisneyFormPage> {
  TextEditingController ctrNombre = TextEditingController();
  TextEditingController ctrPeli = TextEditingController();
  TextEditingController ctrAvatar = TextEditingController();
  bool isLoading = false;
  String? errorMessage;
```

Ara, veiem el mètode que s'executa al clicar sobre el botó "Afegir". Dins d'aquest, validem el que l'usuari ha introduït els camps obligatoris, després activem el mode de carregar i creem l'objecte Disney amb les dades del formulari. En tercer lloc, enviem les dades a l'API si hi ha la resposta és exitosa, es tanca la pantalla i afegeix el nou personatge a la llista, si hi ha algun error, l'app simula que s'ha creat correctament però salta un avís de que en veritat s'ha creat només localment.

```dart
void submitPup(BuildContext context) async {
  if (ctrNombre.text.isEmpty) {
    setState(() {
      errorMessage = 'You forgot to insert the character name';
    });
    // ignore: use_build_context_synchronously
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
      backgroundColor: Colors.redAccent,
      content: Text(errorMessage!),
    )); // SnackBar
  } else {
    setState(() {
      isLoading = true;
      errorMessage = null;
    });
```

```
    var newDisney = Disney(ctrNombre.text);
    newDisney.peli = ctrPeli.text.isNotEmpty ? ctrPeli.text : null;
    newDisney.avatar = ctrAvatar.text.isNotEmpty ? ctrAvatar.text : null;

    final createdDisney = await Disney.createDisneyCharacter(newDisney);

    if (mounted) {
      setState(() {
        isLoading = false;
      });

      if (createdDisney != null) {
        // Éxito - personaje creado en la API
        // ignore: use_build_context_synchronously
        Navigator.of(context).pop(createdDisney);
      } else {
        // Si falla la API, agregamos localmente de todas formas
        setState(() {
          errorMessage = 'Personaje agregado localmente (conexión a API limitada)';
        });

        // Asignar un ID local si es necesario
        newDisney.id = 'local_${DateTime.now().millisecondsSinceEpoch}';
        newDisney.rating = 10;

        // ignore: use_build_context_synchronously
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
          backgroundColor: Colors.orange,
          content: const Text('Personaje guardado localmente'),
          duration: const Duration(seconds: 2),
        )); // SnackBar

        // Cerrar y retornar el personaje de todas formas
        // ignore: use_build_context_synchronously
        Navigator.of(context).pop(newDisney);
      }
    }
  }
}
```

Finalment, definim com volem que es vegi el nostre formulari. Utilitzem un estil que concordi amb la resta de l'app i també definim com volem que es vegin els tres camps "inputs".

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Afegeix un nou personatge'),
      backgroundColor: const Color.fromARGB(255, 0, 0, 0),
    ), // AppBar
    body: Container(
      color: const Color.fromARGB(255, 0, 0, 0),
      child: Padding(
        padding: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 32.0),
        child: Column(children: [
          Padding(
            padding: const EdgeInsets.only(bottom: 8.0, top: 16.0),
            child: Column(children: [
              TextField(
                controller: ctrNombre,
                style: const TextStyle(decoration: TextDecoration.none),
                onChanged: (v) => ctrNombre.text = v,
                decoration: const InputDecoration(
                  labelText: 'Nom del personatge',
                  labelStyle: TextStyle(color: Color.fromARGB(255, 255, 255, 255)),
                  hintText: 'e.g., Mickey Mouse',
                ), // InputDecoration
              ), // TextField
              const SizedBox(height: 16.0),
              TextField(
                controller: ctrPeli,
                style: const TextStyle(decoration: TextDecoration.none),
                onChanged: (v) => ctrPeli.text = v,
                decoration: const InputDecoration(
                  labelText: 'Pel·lícula',
                  labelStyle: TextStyle(color: Color.fromARGB(255, 255, 255, 255)),
                  hintText: 'e.g., Steamboat Willie',
                ), // InputDecoration
              ), // TextField
              const SizedBox(height: 16.0),
              TextField(
                controller: ctrAvatar,
                style: const TextStyle(decoration: TextDecoration.none),
                onChanged: (v) => ctrAvatar.text = v,
                decoration: const InputDecoration(
                  labelText: 'Imatge',
                  labelStyle: TextStyle(color: Color.fromARGB(255, 255, 255, 255)),
                  hintText: 'https://...',
                ), // InputDecoration
              ), // TextField
            ]), // Column
          ), // Padding
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Builder(
              builder: (context) {
                return ElevatedButton(
                  onPressed: isLoading ? null : () => submitPup(context),
                  child: isLoading
                      ? const SizedBox(
                          height: 20,
                          width: 20,
                          child: CircularProgressIndicator(strokeWidth: 2),
                        ) // SizedBox
                      : const Text('Afegir personatge a la llista'),
                ); // ElevatedButton
              },
            ), // Builder
          ) // Padding
        ]), // Column
      ), // Padding
    ), // Container
  ); // Scaffold
}
```

3. VÍDEO DEMOSTRACIÓ

https://drive.google.com/file/d/125FaJYSDDfy0ee310jdom_FKdktZupZl/view?usp=sharing