

Projecte Final de Cicle

HoopsArchive



Alumne:

Enrique Gómez Gascón

DNI:

73607523W

Tutor Individual:

Joan Gerard Camarena Estruch

Tutor del grup:

Jose Alfredo Murcia Andres



IES Jaume II El Just
Tavernes de la Valldigna

Dades del Projecte

Dades de l'alumne

Nom i cognoms	Enrique Gómez Gascón
NIF/NIE	10577749
Curs i CF	2n DAM

Dades del projecte

Títol del projecte	HoopsArchive
Nom del tutor individual	Joan Gerard Camarena Estruch
Nom del tutor del grup	Jose Alfredo Murcia Andres
Resum	<i>Aquesta és una aplicació desenvolupada en Flutter que utilitza una base de dades en MongoDB per a consultar jugadors, equips, estadístiques...De diferents temporades de la NBA i tractar aquestes dades a través d'una API en NodeJS.</i>
Abstract	<i>This is an application developed in Flutter that uses MongoDB database to consult players, teams, statistics... From diferents NBA seasons and through a NodeJS API process this data.</i>
Mòduls implicats	<ul style="list-style-type: none">• Accés a Dades• Programació multimèdia i dispositius mòbils• Programació de Serveis i Processos
Data de presentació	13 de juny de 2023

Índex

1. Introducció del projecte.....	4
1.1 Descripció i tipus de projecte.....	4
1.2 Objectius.....	4
1.3 Orientacions per al desenvolupament i recursos.....	5
2. Disseny de la solució.....	7
2.1 Diagrama de comportament.....	7
2.2 Persistència: Diagrames Entitat-Relació. Pas a taules.....	9
2.3 Planificació temporal i pressupost.....	10
3. Desenvolupament de la solució.....	11
3.1 API.....	11
3.2 FLUTTER.....	15
Component general.....	19
Clasificación.....	20
Equipos.....	23
InfoEquipos.....	25
Lideres.....	28
4. Implantació de la solució.....	32
5. Conclusions i treballs futurs.....	33
Dificultats.....	33
Futurs treballs i/o millores.....	33
Bibliografia.....	33

1. Introducció del projecte

1.1 Descripció i tipus de projecte

Aquest projecte es tracta d'una aplicació tant mòbil, com d'escriptori en la que es pot consultar classificacions, estadístiques, plantilles... De diferents temporades de la història de la NBA. Aquesta aplicació utilitza una API Rest connectada a una base de dades per a accedir a la informació.

La idea sorgeix de la meua part, com a fanàtic del bàsquet i sobretot la NBA, per d'alguna forma, ja fora per curiositat o per la necessitat d'aquesta informació, tindre-la en la palma de la ma o almenys a pocs clicks de distància.

1.2 Objectius

L'objectiu del projecte és portar aquesta informació a la ma de qui la necessita i facilitar-ho amb una aplicació senzilla i pràctica que es puga utilitzar des de diferents dispositius i a més que siga per a qualsevol tipus d'usuari.

1.3 Orientacions per al desenvolupament i recursos

Ací passaré a enumerar les ferramentes i tecnologies que he utilitzat.



El principal recurs alhora de desenvolupar ha sigut Visual Studio Code, ja que estic familiaritzat i té moltes extensions per a utilitzar el llenguatge que necessite, a més de per a escriure el codi, ha sigut depurar, fer proves de l'estat del projecte i gestionar de forma directa els canvis al repositori. I amb açò passem al següent punt.



GitHub a més de fer-lo servir com a ferramenta per a guardar els canvis que podia fer en remot i controlar les versions que feia del projecte també la he pogut fer servir de ferramenta bibliogràfica per a buscar informació, idees i solucions.



El sistema de base de dades que he triat ha sigut MongoDB i esque al treballar en col·leccions en JSON alhora de tractar la informació ha sigut el més senzill, a més de que per a grans bases de dades és una gran elecció.

El framework per a desenvolupar la API ha sigut NodeJS, ja que JavaScript es un llenguatge que per a aquesta faena en concret es fàcil d'utilitzar i la seua implementació junt a MongoDB i Flutter es fa senzilla.

Llibreries utilitzades:

- **Express:** Llibreria per a treballar en serveis web i APIs.
- **Cors:** Ajuda a relaxar la seguretat web de cara a fer més accessible la nostra API.
- **Mongoose:** Llibreria per a connectar Node i Mongo, a més de per a crear esquemes de les col·leccions.



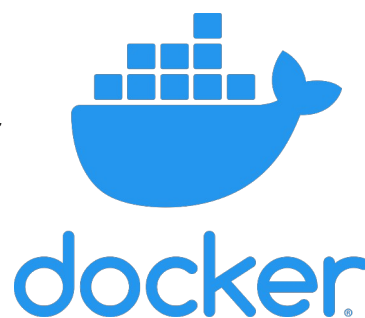
Flutter ha sigut el framework que he decidit utilitzar per a l'aplicació ja que te avantatges com:

És multiplataforma, pots decidir per a quins sistemes vols generar la teua aplicació.

Facilita el treball en documents JSON per a la recollida de dades.



Docker ha sigut l'he utilitzat per a tindre un contenidor amb el servidor de la base de dades .



2. Disseny de la solució

2.1 Diagrama de comportament

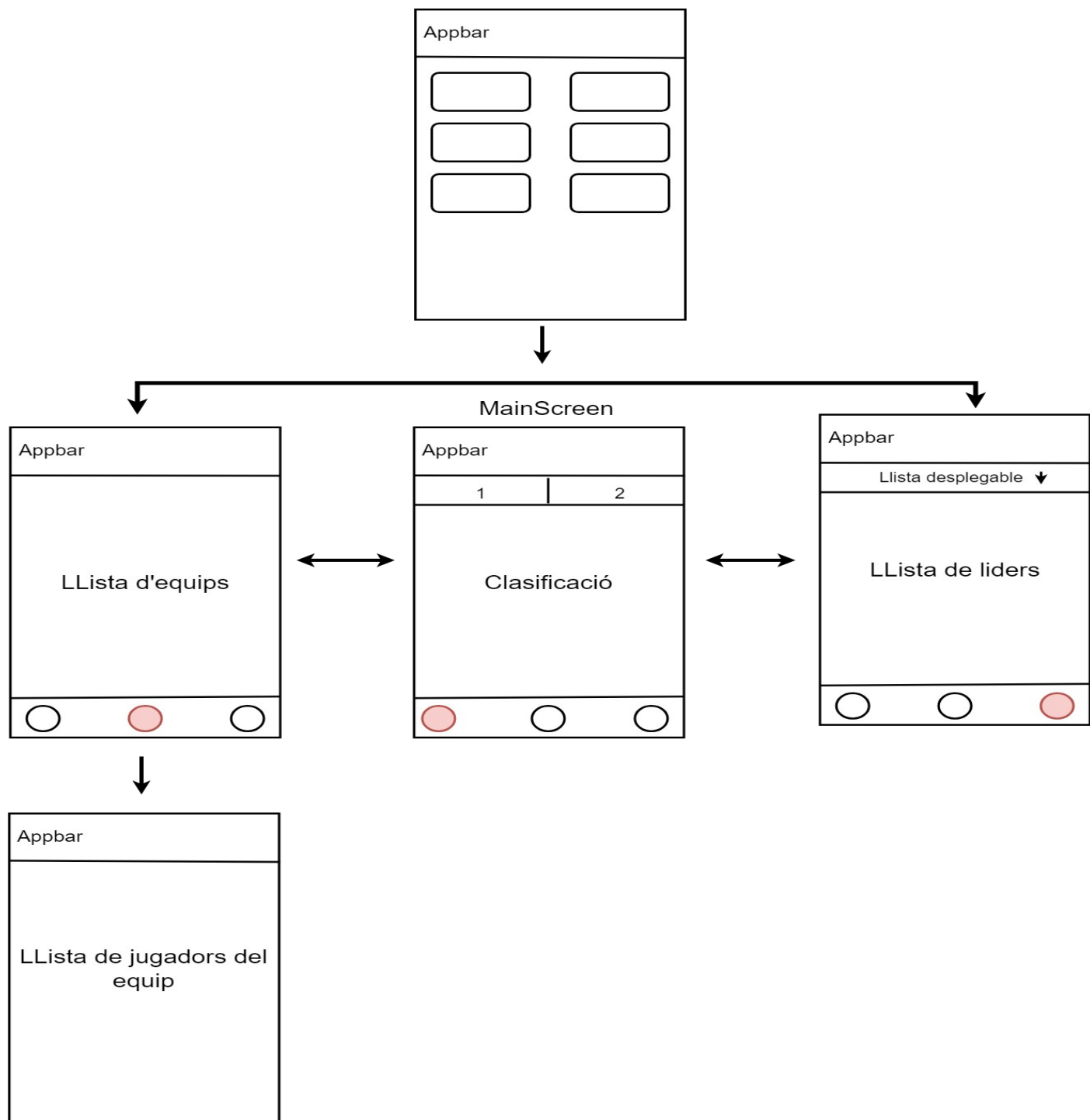
Ací tenim un esquema de com funciona la navegació dins de l'aplicació.

Bàsicament està dividida en 3 pantalles:

- Pantalla de selecció: En aquesta pantalla trobem un llista de botons per a seleccionar la temporada de la qual volem veure el contingut.
- Pantalla principal: En aquesta trobem una barra de navegació a la part inferior on poder desplaçar-nos pel contingut de la aplicació:
 - Classificació: On trobem la classificació de la temporada seleccionada dividida per conferències*.
 - Llistat d'equips: Ací podem veure la llista completa d'equips de la competició amb els seus respectius logos. A més d'açò si volem veure dels jugadors d'un equip només hi haurà que clicar en ell.
 - Ranking de líders: Ací es mostra als jugadors en els màxims promitjos en cadascuna de les estadístiques.
- Pantalla d'informació dels equips: A aquesta pantalla s'accedirà a través del llistat dels equips clickant en un d'ells i aleshores podrem veure una llista amb els jugadors que conformen aquest equip i la seua posició, a més de fent click en qualsevol d'ells podrem les seues estadístiques en la temporada que hem seleccionat al principi.

*conferències: És la forma en la que es divideixen les classificacions de la lliga depenent de la posició geogràfica on es troben les ciutats dels equips(Oest i Est).

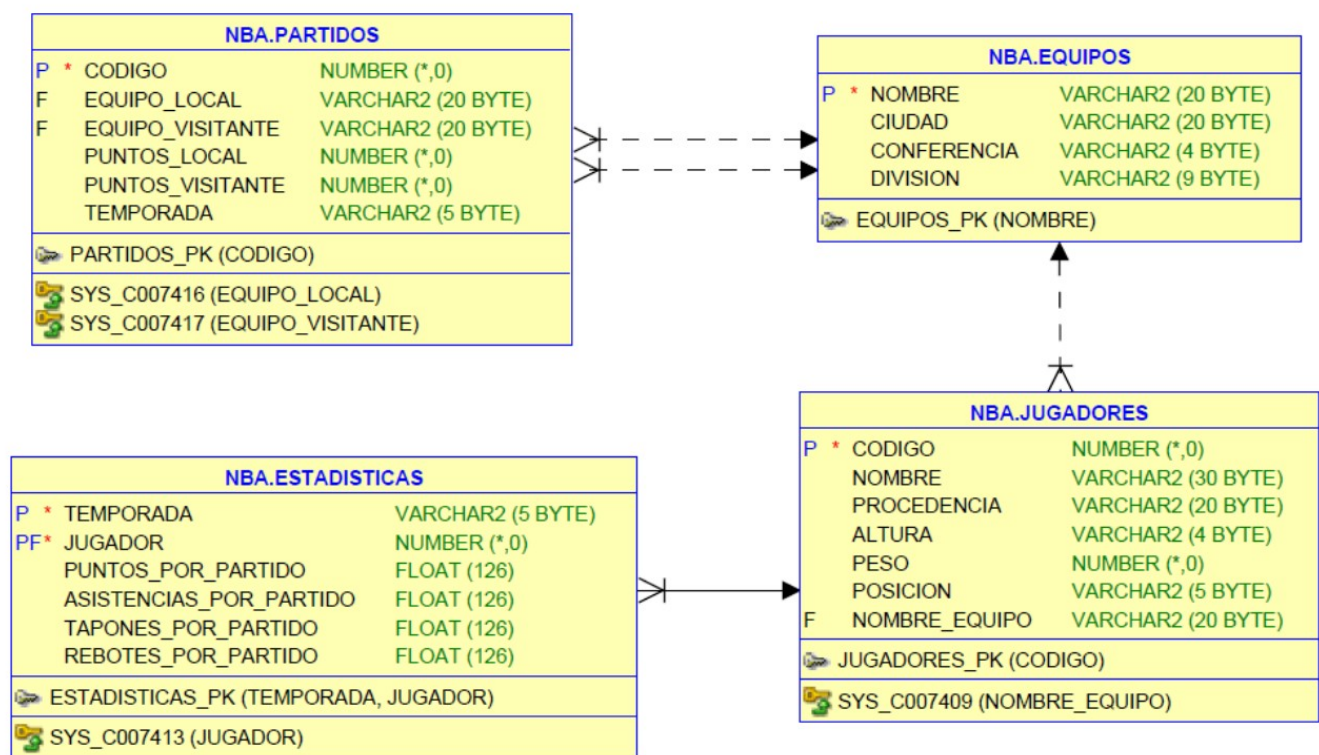




2.2 Persistència: Diagrames Entitat-Relació. Pas a taules.

Al utilitzar MongoDB com a sistema de base de dades no hi ha relacions entre taules, però originalment aquesta base de dades va ser feta en MySQL i el que vaig fer jo va ser migrar-la creient que era el més convenient. Al ser així per a extraure dades que es creuaven entre taules el que he fet ah sigui utilitzar consultes amb agregació.

Així seria com originalment estaria dissenyada la base de dades:



En aquest projecte no me proposat una gran quantitat de tasques ni objectius sols he buscat tindre les diferents parts com el Backend i el Frontend abans de certes dates per a controlar que el ritme al que treballava era correcte.

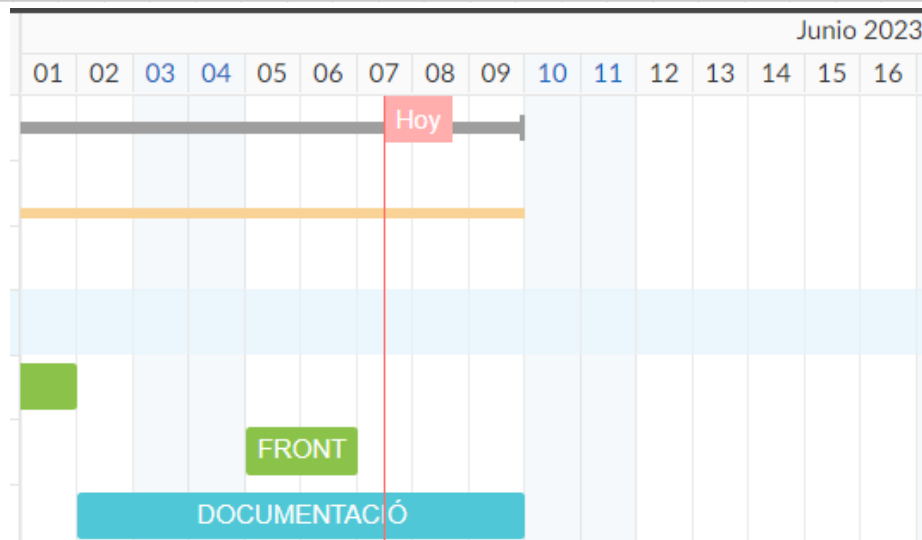
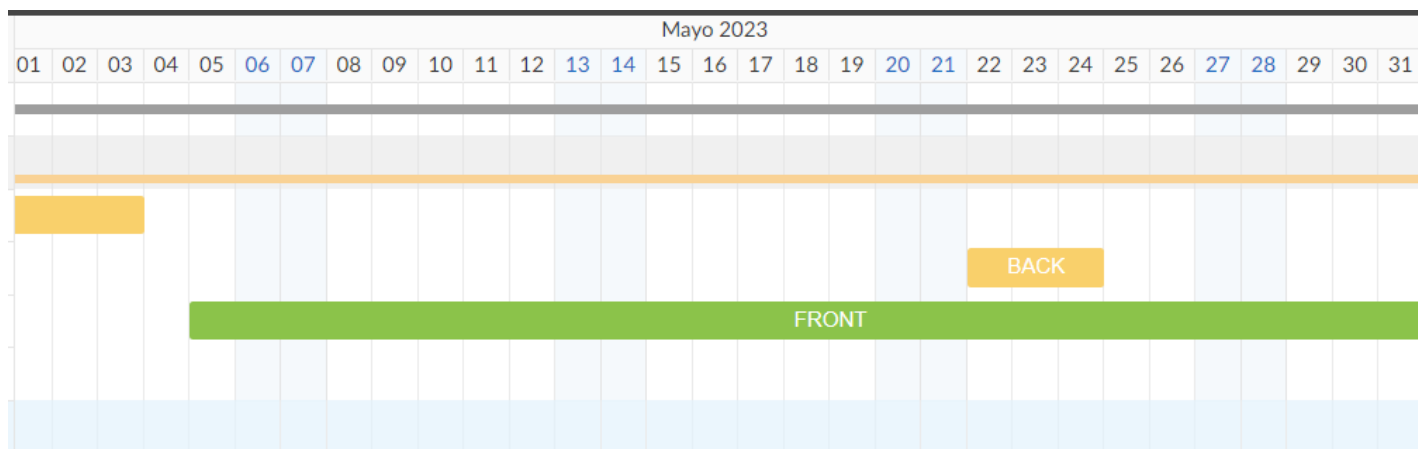
o 20

Abril 2023

31 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

TFG | 2023/03/31 - 2023/06/09

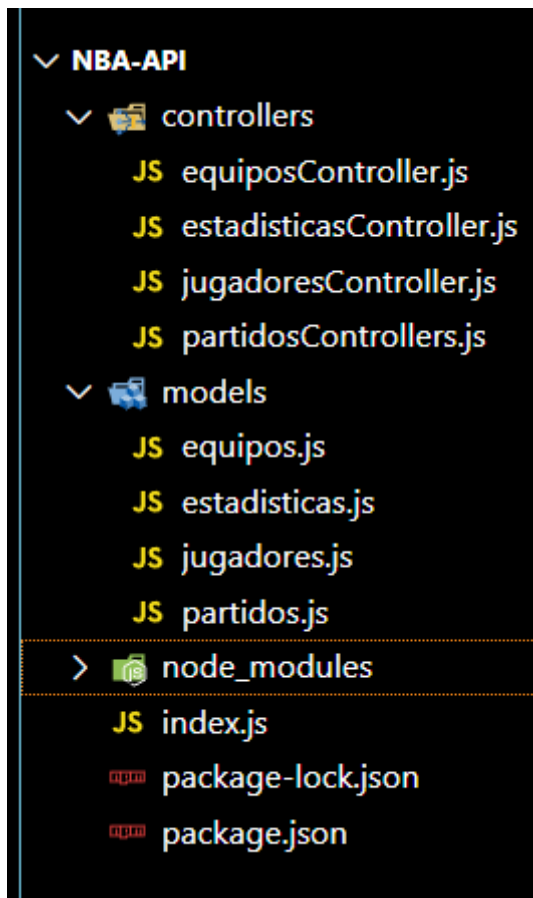
BACK



3. Desenvolupament de la solució

3.1 API

El primer serà iniciar el projecte la estructura d'aquest en el meu cas va quedar tal que així:



El següent pas es definir els esquemes de les col·leccions de mongo amb la llibreria de mongoose, utilitzaré la classe Equipos com a exemple:

```
mongoose.connect(`mongodb://${user}:${pass}@localhost:27017/NBA?authSource=admin`, { useNewUrlParser: true, useUnifiedTopology: true });
mongoose.connection.on("error", function(e) { console.error(e); });

let Equipos = new mongoose.Schema({
  Nombre : String,
  Ciudad : String,
  Conferencia : String,
  Division : String
});
let eqps = mongoose.model('equipos', Equipos);
```

Creem la connexió a la base de dades i definim l'esquema de la col·lecció amb els seus respectius camps.

Ara escriurem que consultes volem a base de dades, cadascuna d'aquestes en un mètode separat:

```
export async function getEquiposPorConferencia(conferencia){
  let res= await eqps.find({'Conferencia': conferencia},{Nombre:1,_id:0});
  if (res) {
    let llista = [];
    for (let equipo of res) {
      llista.push(equipo);
    }
    return llista;
  } else {
    console.log(err);
    return null;
  }
}
```

Escrivim la consulta, recorrem els resultats i els afegim a una llista, aquesta llista es la que retornem per al pas següent.

Ara passem al controlador del mètode anterior

```
static async ObtenerEquiposPorConferencia(req,res){
    let response;
    let type = "application/json";
    let status;

    if(typeof(req.params.conferencia) === typeof(undefined) &&
    typeof(req.query.conferencia) === typeof(undefined)) {

        let llista = await getEquipos();
        response = { llista };
        status = 200;
    }else{
        let conferencia;
        if(typeof(req.params.conferencia) !== typeof(undefined)) conferencia = req.params.conferencia;
        else conferencia = req.query.conferencia;

        response = await getEquiposPorConferencia(conferencia);
        if (!response) response = { "status": "error", "msg": "Not Found" };
    }

    res.type = type;
    res.status = status;
    res.send(response);
}
```

En aquest comencem comprovant que els paràmetres de la ruta no siguin undefined en eixe cas li passarem un mètode general simplement per evitar errors. Una volta sabem que els tipus son correctes fem la crida a al mètode que desitgem i el guardem com a «response» i l'enviem com a la resposta.

Els mètodes mostrats son sols una mostra dels diversos que ens trobem a l'API, ja que podem trobar consultes més complexes però utilitzem aquests perquè la metodologia per a controlar els errors a la resta es igual i/o similar.

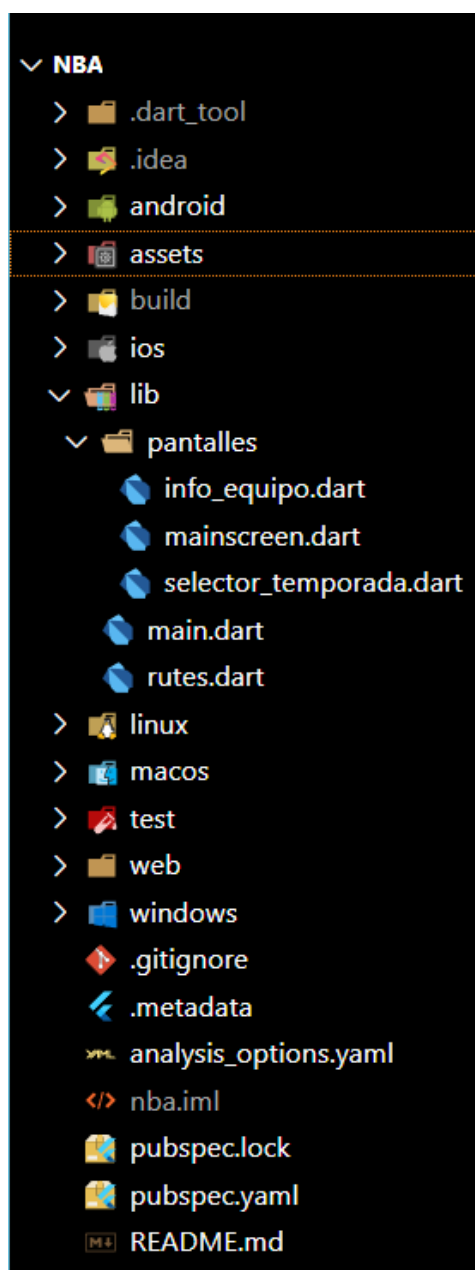
Ara tornem a l'índex del projecte per a trobar les rutes per les quals accedir als controladors creats i utilitzar-los des de qualsevol plataforma exteran.

```
router.get('/equipos',equiposController.NombresEquipos);  
router.get('/:equipo/jugadores', jugadoresController.ObtenerJugadoresPorEquipo);  
router.get('/:conferencia/equipos',equiposController.ObtenerEquiposPorConferencia);  
router.get('/anotadores',estadisticasController.ObtenerPPP);  
router.get('/asistentes',estadisticasController.ObtenerAPP);  
router.get('/reboteadores',estadisticasController.ObtenerRPP);  
router.get('/taponadores',estadisticasController.ObtenerTPP);  
router.get('/:conferencia/clasificacion',partidosController.EquiposClasificacion);  
router.get('/stats/:codigo', estadisticasController.StatsJugador);
```

Aquestes rutes com es mostren ací no estan «completes» ja que algunes utilitzen query strings per a enviar informació addicional.

3.2 FLUTTER

Ací tenim la estructura del projecte en flutter on dins de la carpeta de lib trobem el codi en Dart el qual mirarem amb més deteniment més avant, altres carpetes son android, linux, ios... Que seran les diferents plataformes per a les que podrem fer us de l'aplicació, a més de que aquestes carpetes contenen configuració específica de cadascun. En assets es pot trobar les imatges, logos i tipografies que s'han utilitzat.



Comencem pel principi, veient les crides a la API a través de les rutes abans nomenades. Com podem haver nomenat anteriorment podem veure que ací trobem una de les query strings que es fan servir per a enviar una de les dades.

```
Future <dynamic> ClasiConferencia(String temporada, String conferencia) async {  
  String url = 'http://localhost:8080/api/$conferencia/clasificacion?temporada=$temporada';  
  
  http.Response response = await http.get(Uri.parse(url));  
  
  if (response.statusCode == HttpStatus.ok) {  
    String body = utf8.decode(response.bodyBytes);  
    final result = jsonDecode(body);  
  
    debugPrint(result.runtimeType.toString());  
    return result;  
  } else {  
    throw Exception('No connecta');  
  }  
}
```

Ací vegem que a partir dels paràmetres que rep el mètode es construeix la URL de la petició, fem la crida a l'API i es guarda la resposta, una volta rebuda es comprova que siga correcte i en cas de ser-ho es codifica la resposta a utf-8 i es guarda com a objecte JSON.

D'ací passem a la primera pantalla, en la que ens trobem només entrem a l'aplicació.

```
children: List.generate(10, (index) {  
  int startYear = 98 + index;  
  int endYear = startYear + 1;  
  String buttonText="";  
  if(endYear>=100 || startYear>=100){  
    if (endYear>=100){  
      endYear=endYear-100;  
      buttonText = '$startYear/0$endYear';  
    }  
    if (startYear>=100){  
      startYear=startYear-100;  
      buttonText = '0$startYear/0$endYear';  
    }  
  }  
  else{  
    buttonText = '$startYear/$endYear';  
  }  
})
```

El primer que tenim es generar una llista amb el nombre de botons que volem a la pantalla i ficar les condicions per a que el text tinga la forma que vulguem.

```
}  
return ElevatedButton(  
  child: Text(buttonText,  
    style: TextStyle(  
      fontSize: 20,  
      fontFamily: "Vintage",  
    )), // TextStyle // Text  
  style: ButtonStyle(  
    foregroundColor: MaterialStateProperty.all(Colors.white),  
    backgroundColor: MaterialStateProperty.all(Colors.red),  
  ), // ButtonStyle  
  onPressed: () {  
    Navigator.push(  
      context,  
      MaterialPageRoute(  
        builder: (context) => MainScreen(temporada: buttonText),  
      ), // MaterialPageRoute  
    );  
  }  
);
```

Ací li donem color i format al text dels botons i comportament al botó en qüestió, el qual es tracta de navegar a la següent pantalla passant-li la temporada indicada al botó.

Resultat:

Selecciona una Temporada

98/99	99/00
00/01	01/02
02/03	03/04
04/05	05/06
06/07	07/08

El següent que veurem serà la pantalla principal de l'aplicació, aquesta dependrà sempre del que que hajam clickat a la pantalla anterior. Es a dir, visualment no canviarà però el seu contingut si ho farà.

Component general

La barra de navegació serà independent de quin punt de la pantalla principal ens trobem, ja que està ens la que ens permet moure'ns per ella.

```
body: Center(  
  | child: _widgetOptions(widget.temporada).elementAt(_selectedIndex),  
), // Center  
bottomNavigationBar: BottomNavigationBar(  
  items: const <BottomNavigationBarItem>[  
    BottomNavigationBarItem(  
      | icon: Icon(Icons.format_list_numbered),  
      | label: 'Clasificacion',  
    ), // BottomNavigationBarItem  
    BottomNavigationBarItem(  
      | icon: Icon(Icons.shield_outlined),  
      | label: 'Equipos',  
    ), // BottomNavigationBarItem  
    BottomNavigationBarItem(  
      | icon: Icon(Icons.equalizer),  
      | label: 'Lideres',  
    ), // BottomNavigationBarItem  
  ], // <BottomNavigationBarItem>[]  
  currentIndex: _selectedIndex,  
  selectedItemColor: Colors.amber[800],  
  onTap: _onItemTapped,  
), // BottomNavigationBar
```

Ací trobem definida la NavigationBar amb 3 elements que seran les pantalles que més tard crearem, a destacar quan ens trobem a una de les pagines aquesta apareixerà destacada.

Clasificación

Primera de les pantalles que mostrarà informació recuperada de l'API, aquesta sempre serà la pantalla a la que entrarà de primeres per defecte i ens mostra la classificació dividida per conferencies.

```
class _ClasificacionPageState extends State<ClasificacionPage> with SingleTickerProviderStateMixin {  
  late TabController _tabController;  
  late Future<dynamic> _eastClasificacion;  
  late Future<dynamic> _westClasificacion;  
  
  @override  
  void initState() {  
    super.initState();  
    _tabController = TabController(length: 2, vsync: this);  
    _eastClasificacion = ClasiConferencia(widget.temporada, "East");  
    _westClasificacion = ClasiConferencia(widget.temporada, "West");  
  }  
}
```

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    body: Column(  
      children: <Widget>[  
        TabBar(  
          indicatorColor: Colors.red,  
          controller: _tabController,  
          labelColor: Colors.red,  
          tabs: [  
            Tab(text: 'East'),  
            Tab(text: 'West'),  
          ],  
        ), // TabBar  
        Expanded(  
          child: TabBarView(  
            controller: _tabController,  
            children: [  

```

Ací definim al TabController quants tabs tindrà la pantalla i en el TabBar definim quins van a ser estos elements i l'aspecte de d'aquesta.

```
FutureBuilder<dynamic>(
  future: _westClasificacion,
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(
        child: CircularProgressIndicator(),
      ); // Center
    } else if (snapshot.hasError) {
      return Center(
        child: Text('Error: ${snapshot.error}'),
      ); // Center
    } else {
      final List clasificacion=snapshot.data;
      return ListView.builder(
        itemCount: clasificacion.length,
        itemBuilder: (context, index) {
          final equipo = clasificacion[index];
          var pos=index+1;
          return ListTile(
            leading: Text(pos.toString(),
              style: TextStyle(fontWeight: FontWeight.bold, fontSize: 16)), // Text
            title: Text(equipo['equipo'],
              style: TextStyle(fontSize: 16)), // Text
            trailing: Text("${equipo['victorias']} - ${equipo['derrotas']}",
              style: TextStyle(fontSize: 18)), // Text
          );
        },
      );
    }
  },
);
```

Està formada per un ListView amb una llista amb la informació que es mostra com el nom, nombre de victòries i nombre de derrotes del equip. Abans de carregar la informació es comprova que ja estiga carregada i mentre no ho estiga apareixerà un cercle de carrega i una vegada carregada es comprova hi haga errors.

Resultat:

← Temporada 02/03		
East		West
1	Cavaliers	36 - 21
2	Wizards	36 - 21
3	Pistons	33 - 25
4	Nets	32 - 25
5	Magic	30 - 26
6	Raptors	30 - 27
7	Knicks	30 - 28
8	76ers	29 - 28
9	Bucks	28 - 30
10	Pacers	27 - 31
11	Bobcats	26 - 32
12	Heat	25 - 31

 Clasificación
 Equipos
 Lideres


Equipos














Segona de les pantalles principals, aquesta es bàsicament un llistat complet dels equips acompanyat d'una imatge del seu logo, el punt d'aquesta pantalla és el fet de que al clicar en un equip obrira informació dels seus jugadors.


```
body: FutureBuilder<dynamic>(  
  future: _listaEquipos,  
  builder: (BuildContext context, AsyncSnapshot<dynamic> snapshot) {  
    if (snapshot.connectionState == ConnectionState.waiting) {  
      return Center(  
        | child: CircularProgressIndicator(),  
        | ); // Center  
      } else if (snapshot.hasError) {  
        return Center(  
          | child: Text('Error: ${snapshot.error}'),  
          | ); // Center  
        } else {  
          final List equipos = snapshot.data["llista"];  
          return ListView.builder(  
            itemCount: equipos.length,  
            itemBuilder: (BuildContext context, int index) {  
              final equipo = equipos[index];  
              var nombre=equipo['Nombre'];  
              var ciudad = equipo['Ciudad'];  
              return GestureDetector(  
                onTap: () {  
                  _toInfoEquipo(nombre, ciudad);  
                }  
              ),  
              child: ListTile(  
                leading : new Image(image: new AssetImage("../assets/${nombre}.png")),  
                title: Text(ciudad+" "+nombre,  
                  | style: TextStyle(fontSize: 16),), // Text  
              ), // ListTile  
            }  
          );  
        }  
      }  
    }  
  )
```


Una volta mentre no ha carregat el contingut mostrarà el cercle de carrega, mentre això, es comprova que les dades siguin correctes i quan ja ho són recollim les dades que ens interessin, en aquest cas el nom i la ciutat del equip i a més agafem la imatge des dels assets. Per últim al onTap de la llista afegirem el mètode que s'encarregarà de navegar a la pantalla del equip.


Resultat:

 Temporada 02/03

	Philadelphia 76ers
	Charlotte Bobcats
	Milwaukee Bucks
	Chicago Bulls
	Cleveland Cavaliers
	Boston Celtics
	Los Angeles Clippers
	Memphis Grizzlies
	Atlanta Hawks
	Miami Heat
	New Orleans Hornets
	Utah Jazz
	Sacramento Kings

 Clasificacion

 Equipos

 Lideres

InfoEquipos


Aquesta serà la pantalla a la que arribarem després de fer click a un equip i podrem veure els seus jugadors i més informació d'aquests.

```
appBar: AppBar(  
  backgroundColor: Colors.red,  
  centerTitle: true,  
  title: Text(widget.ciudad+ " "+widget.equipo),  
  actions: [new Image(image: AssetImage("../assets/${widget.equipo}.png"))],  
) , // AppBar  
body: FutureBuilder<dynamic>(  
  future: __listaJugadores,  
  builder: (BuildContext context, AsyncSnapshot<dynamic> snapshot) {  
    if (snapshot.connectionState == ConnectionState.waiting) {  
      return Center(  
        child: CircularProgressIndicator(),  
      ); // Center  
    } else if (snapshot.hasError) {  
      return Center(  
        child: Text('Error: ${snapshot.error}'),  
      ); // Center  
    } else {  
      final jugadores = snapshot.data;  
      return ListView.builder(  
        itemCount: jugadores.length,  
        itemBuilder: (BuildContext context, int index) {  
          final jugador = jugadores[index];
```

```
        ),  
        child: ListTile(  
          title: Text(jugador['Nombre']),  
          trailing: Text(  
            jugador['Posicion'],  
            style: TextStyle(  
              fontSize: 18,  
              fontWeight: FontWeight.bold,  
            ), // TextStyle  
          ), // Text  
        ), // ListTile
```

La informació que teníem abans a la llista ara la col·loquem a la part superior de la pantalla. Una volta més carreguem i comprovem la informació i a la llista mostrem el nom dels jugadors i la seua posició.

Resultat:

←	Boston Celtics	
Ray Allen		G
Tony Allen		G
P.J.Brown		F-C
Sam Cassell		G
Glen Davis		F
Kevin Garnett		F
Eddie House		G
Kendrick Perkins		C
Paul Pierce		F
Scot Pollard		C-F
James Posey		F
Leon Powe		F
Gabe Pruitt		G
Rajon Rondo		G

A més del que ja s'ha mostrat al fer click en un jugador es mostrara un diàleg amb les seues estadístiques en la temporada seleccionada inicialment.

```

return GestureDetector(
  onTap: () async {
    dynamic stats = await StatsporJugador(jugador['codigo'].toString(), widget.temporada);
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          content: Container(
            width: 300,
            padding: EdgeInsets.all(16),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.min,
              children: [
                Text(jugador['Nombre'], textAlign: TextAlign.center,
                  style: TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                  )), // TextStyle // Text
                SizedBox(height: 16),
                Text('Puntos por Partido: '+stats[0]['Puntos_por_partido'].toString()),
                Text('Rebotes por Partido: '+stats[0]['Rebotes_por_partido'].toString()),
                Text('Asistencias por Partido: '+stats[0]['Asistencias_por_partido'].toString()),
                Text('Tapones por Partido: '+stats[0]['Tapones_por_partido'].toString()),
              ],
            ),
          ),
        );
      },
    );
  },
);

```

Indiquem al onTap que mostre el diàleg i d'on trau les dades, el següent sols es donar-li forma amb el nom i cadascuna de les estadístiques a mostrar.

Resultat:



Lideres

Última de les pestanyes de la pantalla principal, en aquesta podem veure al 15 jugadors amb millor mitjana en cadascuna de les estadístiques durant la temporada que hem seleccionat.

```
DropDownButton<String>(  
  value: _dropdownValue,  
  onChanged: (String? newValue) {  
    setState(() {  
      _dropdownValue = newValue!  
      if(_dropdownValue=='Puntos por Partido'){  
        _listaLideres = MaxPPP(widget.temporada);  
      }  
      if(_dropdownValue=='Rebotes por Partido'){  
        _listaLideres = MaxRPP(widget.temporada);  
      }  
      if(_dropdownValue=='Asistencias por Partido'){  
        _listaLideres = MaxAPP(widget.temporada);  
      }  
      if(_dropdownValue=='Tapones por Partido'){  
        _listaLideres = MaxTPP(widget.temporada);  
      }  
    });  
  })
```




Ací li donem opcions a la llista desplegable en la que triarem la estadística de la que volem veure el ranking.

```
items: _lideresOptions.map<DropDownMenuItem<String>>((String valor) {  
  return DropDownMenuItem<String>(  
    value: valor,  
    child: Text(valor),  
  ); // DropDownMenuItem  
}).toList(),  
// DropDownButton  
f(_dropdownValue=='Puntos por Partido')  
Expanded(  
  child: Container(  
    child: FutureBuilder<List>(  
      future: _listaLideres,  
      builder: (context, snapshot) {  
        if (snapshot.hasData) {  
          final list = snapshot.data!;  
          return ListView.builder(  
            itemCount: list.length,  
            itemBuilder: (context, index) {  
              final item = list[index];  
              var pos = index+1;  
              return ListTile(  
                leading: Text(pos.toString(),  
                  style: TextStyle(fontWeight: FontWeight.bold)), // Text  
                title: Text(item['Nombre']),  
                trailing: Text(item['Puntos_por_partido'].toString(),  
                  style: TextStyle(fontSize: 18)), // Text  
              );  
            }  
          );  
        }  
      }  
    )  
  )  
)
```

Ara li donem forma a la llista amb el nom del jugador, la posició que ocupa entre aquests jugadors i la seua estadística.

Resultat:

← Temporada 02/03		
Puntos por Partido		
1	Tracy McGrady	32.1
2	Kobe Bryant	30
3	Shaquille O'Neal	27.5
4	Paul Pierce	25.9
5	Dirk Nowitzki	25.1
6	Ray Allen	24.5
7	Tim Duncan	23.3
8	Kevin Garnett	23
9	Baron Davis	22.9
10	Stephon Marbury	22.3
11	Antawn Jamison	22.2
12	Michael Finley	21.5

 Clasificación
  Equipos
  Lideres

← Temporada 02/03		
Asistencias por Partido ▼		
1	Jason Kidd	8.9
2	Jason Williams	8.3
3	Stephon Marbury	8.1
4	Jamaal Tinsley	7.5
5	Baron Davis	7.5
6	Jason Terry	7.4
7	Steve Nash - C	7.3
8	Andre Miller	6.7
9	Eric Snow	6.6
10	Gilbert Arenas	6.3
11	Steve Francis	6.2
12	Kevin Garnett	6


Clasificación


Equipos


Líderes

4. Implantació de la solució

De cara a la instal·lació en un dispositiu real no he fet molts avanços ja que he aconseguit la instal·lació com a tal però accedir a la informació de l'API ja que ha sigut un intent d'última hora, perquè no ho veia com una prioritat.



Opciones de desarrollador

Opciones de desarrollador



DEPURACIÓN

Depuración USB

Modo depuración cuando el USB esté conectado



Revocar autorizaciones de depuración USB >

Instalar vía USB

Permitir la instalación de aplicaciones vía USB



Una volta al nostre dispositiu i amb aquest connectat per USB a l'equip on es trobe el projecte haurem de tindre les opcions per a desenvol·ladors activades i dins d'aquestes la depuració i la instal·lació per USB habilitada. Amb açò simplement des del terminal de l'equip executem **flutter run** i començarà la instal·lació.

5. Conclusions i treballs futurs

Com a conclusió, ha sigut un treball on el que més me trobat ha sigut no tant dificultats sinó coses que he tingut que investigar perquè desconeixia o simplement no havia tingut la oportunitat de provar fins al moment i ara he acabat «controlant» o almenys coneixent per a futurs conflictes.

Dificultats

- La utilització de consultes de mongo que implicaren diferents col·leccions.
- El maneig dels tipus de dades des de l'api i alhora de rebre'ls al frontend.
- La implantació de la aplicació a dispositius mòvils.

Futurs treballs i/o millores

- Ampliar la base de dades ja siga en quantitat de temporades com en profunditat de les estadístiques.
- Investigar més a fons el traure l'aplicació a dispositius mòvils.

Bibliografia

Base de dades extreta de:

<https://www.discoduroderoer.es/ejercicios-propuestos-y-resueltos-consultas-sql-nba/>

<https://chat.openai.com/>

<https://www.mongodb.com/docs/>

<https://docs.flutter.dev/>

Canal que he freqüentat per a resolució de dubtes:

<https://www.youtube.com/@CodigoCorrecto>