World-Time API:

Now that we know how to fetch data from an external API using the http package, we’re now ready to fetch data from the actual site which has data useful to us. This is called the ‘World Time API’ at <http://worldtimeapi.org/>.

We follow all the previous steps, in the same getData function (which we’ll now call getTime() function) in the loading.dart file we create a Response object and store the output of the response of the request to the API. Now, the response of the API varies with the URL specified. So, if we wanted to query the time of ‘Kolkata’ suppose, we’ll need to query the url as: <http://worldtimeapi.org/api/timezone/Asia/Kolkata>

If we wanted Berlin, we need to query it as:

<http://worldtimeapi.org/api/timezone/Europe/Berlin>

Thus we see that whenever we change the last two sections of the url, we get different time-zones. The url basically has the format of:

<http://worldtimeapi.org/api/timezone/Continent/City>

We jsonDecode the body of this response object and store it in a map, as we did before.

Response response= **await** get(**'http://worldtimeapi.org/api/timezone/Asia/Kolkata'**);  
Map data=jsonDecode(response.**body**);  
print(data);

We get all the data dump as:

{"abbreviation":"IST","client\_ip":"182.70.91.217","datetime":"2020-05-10T21:25:14.489452+05:30","day\_of\_week":0,"day\_of\_year":131,"dst":false,"dst\_from":null,"dst\_offset":0,"dst\_until":null,"raw\_offset":19800,"timezone":"Asia/Kolkata","unixtime":1589126114,"utc\_datetime":"2020-05-10T15:55:14.489452+00:00","utc\_offset":"+05:30","week\_number":19}

Of these properties, we are interested in 2 specific ones:

1. DateTime
2. Utc\_offset

We’ll need to add the UTC offset to the dateTime in order to get the correct local time.

So, we extract those from our data and print them:

*//get properties from data*String dateTime=data[**'datetime'**];  
String offset=data[**'utc\_offset'**];

print(dateTime);  
print(offset);

We get the output as:

I/flutter(11285): 2020-05-10T21:25:14.489452+05:30

I/flutter(11285): +05:30

Currently however, these are a bunch of strings. It would be a million times better, if they were converted into the standard DART ‘DateTime’ object. Converting them into this format makes things much easier to extract data from. We can easily tap into the hours, minutes, seconds of the time as well as add or subtract hours, minutes and seconds, which is otherwise very difficult with strings.

This process is called ‘*parsing’.* Thus, we are parsing an ordinary string into a format compatible to be stored in the DateTime object in DART. This is done with the help of the method called ‘*parse’.*

Therefore, we write:

DateTime now=DateTime.*parse*(dateTime);

Now that we have the DateTime object, we can easily add the offset to it using the inbuilt method called add(). We are going to have to add the hours and minutes separately. Therefore, we need an instance of the Duration object to be specified in the add() method.

now=now.add(Duration(hours: ,minutes:);

We’ll need to extract the hours and minutes separately from the offset string, and we do that with the help of the substring method. We need to have 2 strings—one which stores the offset hours (with the sign), and the other which stores the offset minutes.

String offsetHours=data[**'utc\_offset'**].substring(0,3);// Sign is included  
String offsetMinutes=data[**'utc\_offset'**].substring(4,6);

Now, we’re ready to add the offset to the dateTime object. We’ll need to further parse these strings into integers with the help of the int.parse() method.

Our final time looks like:

now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));

The final function looks like:

Void getTime() **async**{  
 *//make the request* Response response= **await** get(**'http://worldtimeapi.org/api/timezone/Asia/Kolkata'**);  
 Map data=jsonDecode(response.**body**);  
 *//print(data);  
   
 //get properties from data* String dateTime=data[**'datetime'**];  
 String offsetHours=data[**'utc\_offset'**].substring(0,3);  
 String offsetMinutes=data[**'utc\_offset'**].substring(4,6);  
 *//print(dateTime);  
 //print(offsetHours);  
 //print(offsetMinutes);  
   
 //create dateTime object* DateTime now=DateTime.*parse*(dateTime);  
 now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));  
 print(now.**hour**);

}

The WorldTime Custom Class:

We have written quite a bit of logic in the getTime function which currently exists in loading.dart. In order to make our program more modular, we’ll have to place this logic into a separate place so that it can be reused by other classes.

We create a new package called ‘services’ into which we’ll add a new dart file called world\_time.dart.

We’ll need to import the http package as well as the dart:convert package so as to jsonconvert the data:

**import 'package:http/http.dart'**;  
**import 'dart:convert'**;

We’ll create a new class and call it WorldTime. It will have the following properties:

**class** WorldTime{  
  
 String **location**; *//location name for UI* String **time**;*//time in that location* String **flag**; *//url to a flag icon* String **url**;*//location url for API endpoint* WorldTime({**this**.**location**, **this**.**flag**, **this**.**url**});

}

Now, we bring up the getTime function from the loading.dart file, and place it here in this class. Also, instead of hard-coding the location, we’ll pass the url property to the http request:

**class** WorldTime{  
  
 String **location**; *//location name for UI* String **time**;*//time in that location* String **flag**; *//url to a flag icon* String **url**;*//location url for API endpoint* String **eventOfDay**;*//decides if morning,eveninng,night,etc*  
  
 **void** getTime() **async** {  
 *//make the request*  Response response= **await** get(**'http://worldtimeapi.org/api/timezone/**$**url'**);  
 Map data=jsonDecode(response.**body**);  
 *//print(data);  
   
 //get properties from data* String dateTime=data[**'datetime'**];  
 String offsetHours=data[**'utc\_offset'**].substring(0,3);  
 String offsetMinutes=data[**'utc\_offset'**].substring(4,6);  
 *//print(dateTime);  
 //print(offsetHours);  
 //print(offsetMinutes);  
   
 //create dateTime object* DateTime now=DateTime.*parse*(dateTime);  
 now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));  
 *//print(now.hour);*

}

Now, if we look at the class, we see that the time property is a string. But what we’ve got here at the end of the getTime() function is an instance of the DateTime class with the name ‘now’. We’ll need to convert it into a string. So, we simply add a now.toString() and set it equal to Time:

*//set the time property*

**time**=now.toString();

Now, we’ll have a basic idea of how our WorldTime class works. Whenever we create an instance of this class, we’ll need to pass in the ‘location’, the ‘url’ and the ‘flag’ property while instantiating the class. After receiving these properties, if we call the getTime function, the value of the ‘time’ property will be set.

So, we appropriately create a constructor for it:

WorldTime({**this**.**location**, **this**.**flag**, **this**.**url**});

Our class now looks like:

**import 'package:http/http.dart'**;  
**import 'dart:convert'**;  
**import 'package:intl/intl.dart'**;  
  
**class** WorldTime{  
  
 String **location**; *//location name for UI* String **time**;*//time in that location* String **flag**; *//url to a flag icon* String **url**;*//location url for API endpoint* String **eventOfDay**;*//decides if morning,eveninng,night,etc* WorldTime({**this**.**location**, **this**.**flag**, **this**.**url**});  
  
  
 **void** getTime() **async** {  
 *//make the request* Response response= **await** get(**'http://worldtimeapi.org/api/timezone/**$**url'**);  
 Map data=jsonDecode(response.**body**);  
 *//print(data);  
   
 //get properties from data* String dateTime=data[**'datetime'**];  
 String offsetHours=data[**'utc\_offset'**].substring(0,3);  
 String offsetMinutes=data[**'utc\_offset'**].substring(4,6);  
 *//print(dateTime);  
 //print(offsetHours);  
 //print(offsetMinutes);  
   
 //create dateTime object* DateTime now=DateTime.*parse*(dateTime);  
 now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));  
 *//print(now.hour);  
 //set the time property*   
 **time**=now.toString();

}  
  
 }

Now that we’re done with the class, we head on to the loading screen, and then import this class into the loading.dart file:

**import 'package:chronograph/services/world\_time.dart'**;

Inside the loading screen, we’ll create a new async function called ‘setupWorldTime’ which will initialise what time the app shows when the user first loads the app. This will of course be hard coded. In here, we’ll create a new instance of the WorldTime class, and specify the properties like ‘location’, ‘url’ and ‘flag’, then we’ll call the getTime function in an awaited manner in order to set the time property of the instance. The ‘await’ here is important as we don’t want anything to be executed before we fetch the time from the API.

**void** setupWorldTime() **async**{  
 WorldTime instance=WorldTime(location: **'Kolkata'**,flag: **'india.png'**,url: **'Asia/Kolkata'**);  
 **await** instance.getTime();  
   
}

Now that we’re awaiting the getTime() method, and if we want to use the await keyword, we need to remember that the request can also return an error. In that case, we’re not sure of what datatype our request returns. This datatype will therefore be resolved in the future—when the request completes. Since we’re unaware of the returned datatype here, we’ll wrap the void return type of the getTime function with the future keyword. It just signifies that—at the time of writing the code, the return type is unknown, *and will be resolved in the future.* So, the current return type is Future which is a placeholder value until the function is complete.

Future <**void**> getTime() **async**{

*//make the request* Response response= **await** get(**'http://worldtimeapi.org/api/timezone/**$**url'**);  
 Map data=jsonDecode(response.**body**);  
 *//print(data);  
   
 //get properties from data* String dateTime=data[**'datetime'**];  
 String offsetHours=data[**'utc\_offset'**].substring(0,3);  
 String offsetMinutes=data[**'utc\_offset'**].substring(4,6);  
 *//print(dateTime);  
 //print(offsetHours);  
 //print(offsetMinutes);  
   
 //create dateTime object* DateTime now=DateTime.*parse*(dateTime);  
 now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));  
 *//print(now.hour);  
 //set the time property*   
 **time**=now.toString();

}

We’ll call up this method inside the initState() method of the loading screen:

@override  
**void** initState() {  
 **super**.initState();  
 setupWorldTime();  
}

Now, as a test, we can actually print the obtained time on the UI in a text widget. In order to do that, we’ll create a dummy string whose initial value will be ‘loading’ and after the getTime function runs, we’ll fill it in with the time obtained.

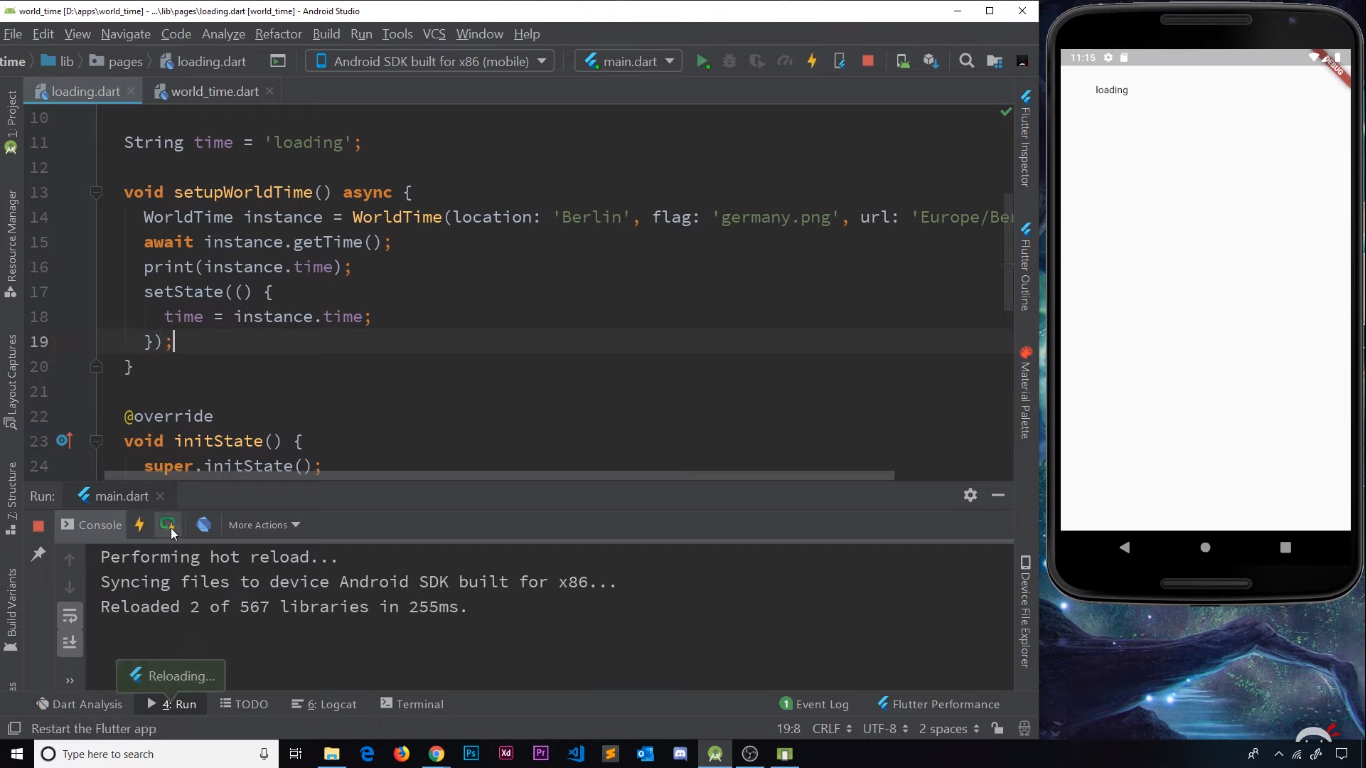
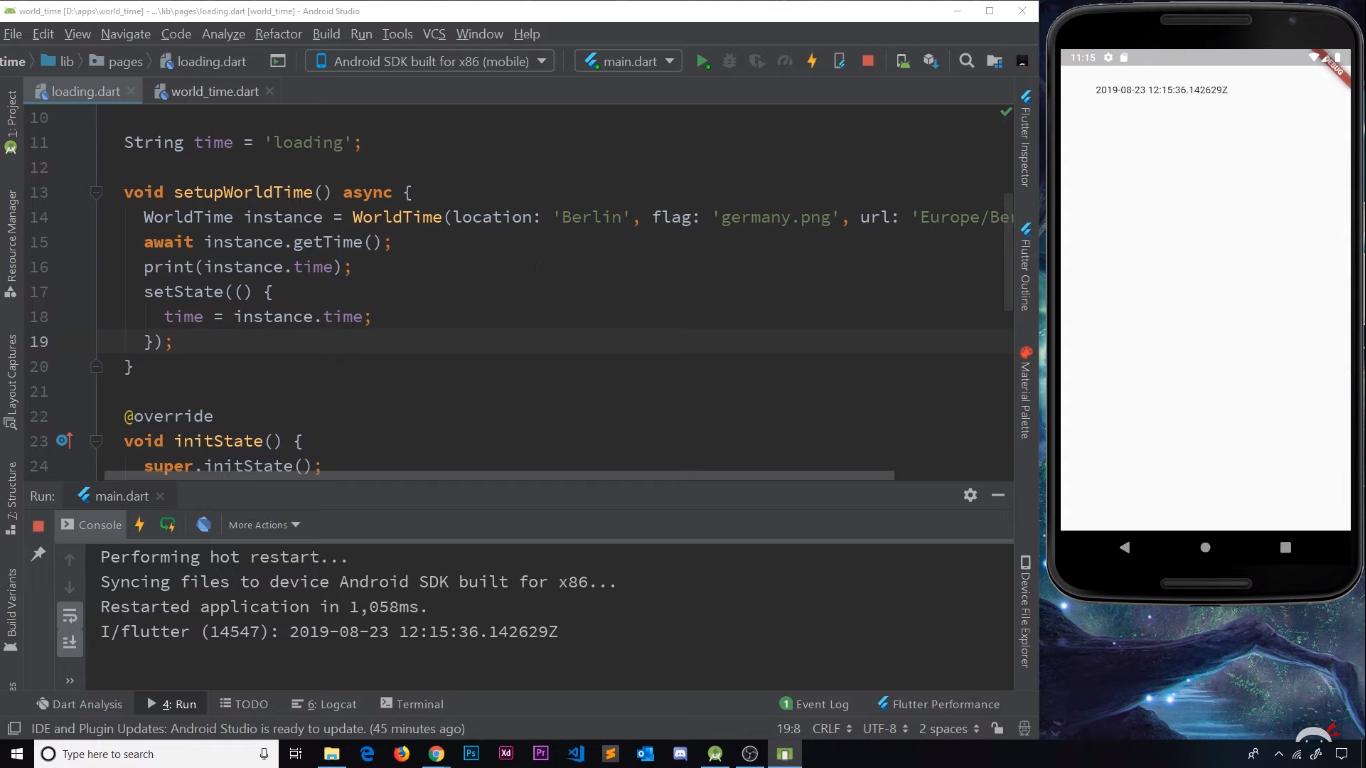
**String** time= ‘Loading’;

**void** setupWorldTime() **async**{  
 WorldTime instance=WorldTime(location: **'Kolkata'**,flag: **'india.png'**,url: **'Asia/Kolkata'**);  
 **await** instance.getTime();

setState((){

time=instance.getTime();

});  
 }

Error Handling:

There might be cases where an incorrect URL for the time zone will cause the entire app to crash. Because the appropriate JSON file is not found, we’ll not be able to map it to DART map, then the dateTime will not be created, and finally the time property cannot be set. In such cases we’ll need to manually handle the error. We do this with a simple try/catch block.

We’ll just update the ‘time’ text, to an error message it the catch block.

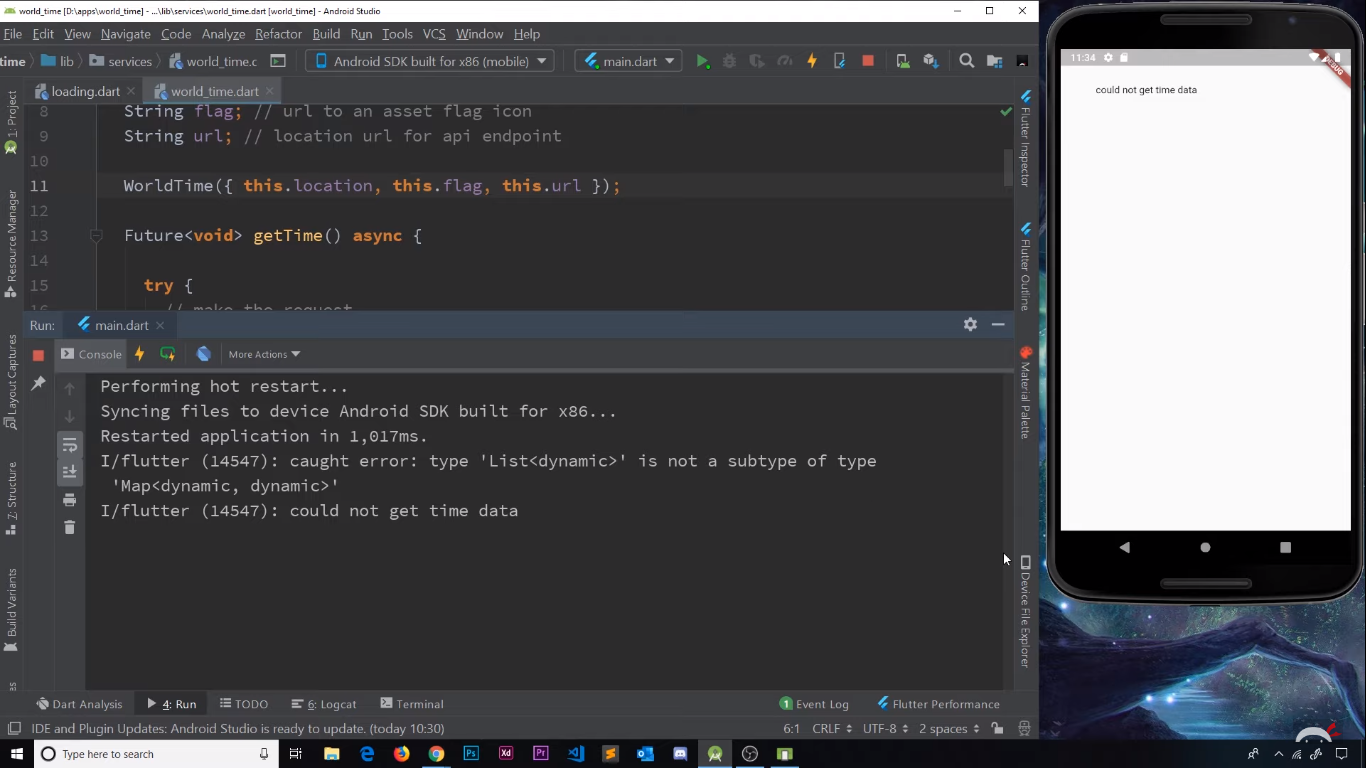
**try** {

//code of getTime function

}

**catch** (e) {  
 print(**'Caught Error** $e**'**);  
 **time**=**'Failed to get the time data'**;  
}

So, if we have some wrong url, we’ll see:



Passing Route Data:

Now that we have our loading screen set up, we need some mechanism to invoke the home screen and pass on the time data to it. If we remember from Android Java, we shall see that this is similar to the concept of passing data in bundles when we start an intent.

Once we have obtained the time data from the API, it would then be a good time to navigate to the home screen, thus disposing the loading screen. We have learnt earlier that in order to navigate to a different route, we’ll need to invoke the Navigator object, and use the method called ‘pushNamed’ wherein we pass the parameter of the context object, and then the route which we want to navigate to.

Navigator.pushNamed(context,’/home’);

But this method is just going to push the home route above the loading route, which means that the loading route will still be present underneath. Now, we don’t want that. So, instead we use the method called ‘pushReplacementNamed’ which disposes off the preceding route and pushed the newer one in place of it.

Now, in order to send data from one route to another, we pass in a third parameter along with the context object and the route name. This parameter is called ‘arguments’ which is actually a map. Thus, we pass in the arguments from one activity to another, in the form of ‘key-value’ pairs and we can name the keys as we want.

So, in this case, we’ll be sending the ‘location’, ‘time’, ‘flag’ as well as the ‘url’ from the loading screen to the home screen. The final method in loading.dart looks like:

**void** setupWorldTime() **async**{  
 WorldTime instance=WorldTime(location: **'Kolkata'**,flag: **'india.png'**,url: **'Asia/Kolkata'**);  
 **await** instance.getTime();  
 Navigator.*pushReplacementNamed*(**context**, **'/home'**,arguments: {  
 **'location'**:instance.**location**,  
 **'time'**:instance.**time**,  
 **'flag'**:instance.**flag**,  
 **'url'**:instance.**url** });  
  
}

The next step of the plan is to actually receive the data on the home screen. Now, in order to receive the data, we’ll first create an empty map, and call it data.

Map **data**={};

Now, in order to fill in this map with the data that we get from the loading route, we’ll use the method call of ModalRoute.settings. It is important that we receive this data inside the build method, as we shall need the context object for our reference:

ModalRoute.*of*(context).**settings**.**arguments**

The above function call returns a map, that we sent from the loading screen. We’ll just store that map inside the empty map we created.

**data**= ModalRoute.*of*(context).**settings**.**arguments**

Our code looks like:

**class** \_HomeState **extends** State<Home> {  
  
 Map **data**={};

@override  
Widget build(BuildContext context) {  
   
 **data**= ModalRoute.*of*(context).**settings**.**arguments**;

…

}

Formatting the time:

Currently, the time that is displayed on the screen, is slightly difficult to read. It is not properly formatted, and contains some stray characters that may look gibberish to the common user.

12:15:36 142629Z

In order to properly format the time, we shall use another flutter package called ‘intl’ which provides internationalisation of time and date formats, among many others. Read more [here](https://pub.dev/packages/intl).

In our WorldTime class, we shall update the time text in a better format using one of the methods in this package. We replace the ‘now.toString()’ with the following:

**time**=DateFormat.jm().format(now);

Our time string is now a normal string with the time displayed in AM/PM with the hours and minutes separated by a colon. Just like a normal clock, without any gibberish.

After certain UI changes, our code looks like:

**return** Scaffold(

body: SafeArea(

child: Padding(  
 padding: **const** EdgeInsets.fromLTRB(0, 120.0, 0, 0),  
 child: Column(  
 children: <Widget>[  
 FlatButton.icon(  
 onPressed: (){},

icon: Icon(  
 Icons.*edit\_location*,

),

label: Text(  
 **'Edit Location'**,

),

),

SizedBox(height: 20.0),  
Row(  
 mainAxisAlignment: MainAxisAlignment.**center**,  
 children: <Widget>[

Text(  
 **data**[**'location'**],  
 style: TextStyle(  
 fontSize: 28.0,  
 letterSpacing: 2.0,

),

],

),

SizedBox(height: 20.0),

Text(  
 **data**[**'time'**],  
 style: TextStyle(  
 fontSize: 66.0,

),

),

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

The output looks like:



Spinners:

At the moment our loading screen looks a little drab, with just a bit of text that disappears after some time. Let’s beautify it using spinners. A spinner is nothing but a loading icon that keeps animating until some action has been completed.

We’ll be using another flutter package called ‘flutter\_spinkit’. This package contains readymade spinner icons whose properties we can adjust as per our preference. Read more [here](https://pub.dev/packages/flutter_spinkit).

We first import the package:

**import 'package:flutter\_spinkit/flutter\_spinkit.dart'**;

We use the rotating circle spinner. We simply write the following code in order to display the spinner:

SpinKitRotatingCircle(

color: Colors.white,

size: 50.0,

)

Our loading screen code looks like:

**import 'package:flutter/material.dart'**;  
**import 'package:chronograph/services/world\_time.dart'**;  
**import 'package:flutter\_spinkit/flutter\_spinkit.dart'**;  
  
**class** Loading **extends** StatefulWidget {  
 @override  
 \_LoadingState createState() => \_LoadingState();  
}  
  
**class** \_LoadingState **extends** State<Loading> {  
  
 **void** setupWorldTime() **async** {  
 WorldTime instance=WorldTime(location: **'Kolkata'**,flag: **'india.png'**,url: **'Asia/Kolkata'**);  
 **await** instance.getTime();  
 Navigator.*pushReplacementNamed*(**context**, **'/home'**,arguments: {  
 **'location'**:instance.**location**,  
 **'time'**:instance.**time**,  
 **'flag'**:instance.**flag**,  
 **'url'**:instance.**url** });  
  
 }  
  
 @override  
 **void** initState() {  
 **super**.initState();  
 setupWorldTime();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 **return** Scaffold(  
 backgroundColor: Colors.*blue*[900],  
 body: Center(  
 child: SpinKitRotatingCircle(  
 color: Colors.*white*,  
 size: 50.0,  
 ),  
  
 )  
 );  
 }  
}

Styling the Home Screen:

It would be awesome to change the background of the home screen depending upon the time of the day, like early morning, morning, afternoon, evening, and night. In the WorldTime class, we shall create a new property called eventOfDay which is a string. It stores what time of the day it actually is, and then will be responsible for changing the wallpaper.

String **eventOfDay**;*//decides if morning,evening,night,etc*

Now, based on the time of the day, the value of this property will be set in the getTime() function along with the time variable. We shall decide the time of the day depending on the hour. So, we build a simple if-else-if ladder like so:

**if**(4<=now.**hour** && now.**hour**<=5)  
 {  
 **eventOfDay**=**'earlyMorning'**;  
 }  
**else if**(6<=now.**hour** && now.**hour**<=11)  
 {  
 **eventOfDay**=**'morning'**;  
 }  
**else if**(12<=now.**hour** && now.**hour**<=16)  
{  
 **eventOfDay**=**'afternoon'**;  
}  
**else if**(17<=now.**hour** && now.**hour**<=19)  
{  
 **eventOfDay**=**'evening'**;  
}  
**else if**(20<=now.**hour** && now.**hour**<=23)  
{  
 **eventOfDay**=**'night'**;  
}  
**else if**(0<=now.**hour** && now.**hour**<=3)  
{  
 **eventOfDay**=**'night'**;  
}

So, our WorldTime class looks like:

**import 'package:http/http.dart'**;  
**import 'dart:convert'**;  
**import 'package:intl/intl.dart'**;  
  
**class** WorldTime{  
  
 String **location**; *//location name for UI* String **time**;*//time in that location* String **flag**; *//url to a flag icon* String **url**;*//location url for API endpoint* String **eventOfDay**;*//decides if morning,evening,night,etc* WorldTime({**this**.**location**, **this**.**flag**, **this**.**url**});  
  
  
 Future <**void**> getTime() **async** {  
 *//make the request* **try** {  
 Response response= **await** get(**'http://worldtimeapi.org/api/timezone/**$**url'**);  
 Map data=jsonDecode(response.**body**);  
 *//print(data);  
   
 //get properties from data* String dateTime=data[**'datetime'**];  
 String offsetHours=data[**'utc\_offset'**].substring(0,3);  
 String offsetMinutes=data[**'utc\_offset'**].substring(4,6);  
 *//print(dateTime);  
 //print(offsetHours);  
 //print(offsetMinutes);  
   
 //create dateTime object* DateTime now=DateTime.*parse*(dateTime);  
 now=now.add(Duration(hours: int.*parse*(offsetHours),minutes: int.*parse*(offsetMinutes)));  
 *//print(now.hour);  
 //set the time property  
 //isDaytime=now.hour>6 && now.hour<20 ? true : false;* **if**(4<=now.**hour** && now.**hour**<=5)  
 {  
 **eventOfDay**=**'earlyMorning'**;  
 }  
 **else if**(6<=now.**hour** && now.**hour**<=11)  
 {  
 **eventOfDay**=**'morning'**;  
 }  
 **else if**(12<=now.**hour** && now.**hour**<=16)  
 {  
 **eventOfDay**=**'afternoon'**;  
 }  
 **else if**(17<=now.**hour** && now.**hour**<=19)  
 {  
 **eventOfDay**=**'evening'**;  
 }  
 **else if**(20<=now.**hour** && now.**hour**<=23)  
 {  
 **eventOfDay**=**'night'**;  
 }  
 **else if**(0<=now.**hour** && now.**hour**<=3)  
 {  
 **eventOfDay**=**'night'**;  
 }  
 **time**=DateFormat.jm().format(now);  
 }  
 **catch** (e) {  
 print(**'Caught Error** $e**'**);  
 **time**=**'Failed to get the time data'**;  
 }  
  
  
 }  
  
  
}

In the loading screen we should also pass in the eventOfDay property to the home screen along with the rest of the properties. Thus, our loading.dart file looks like:

**mport 'package:flutter/material.dart'**;  
**import 'package:chronograph/services/world\_time.dart'**;  
**import 'package:flutter\_spinkit/flutter\_spinkit.dart'**;  
  
**class** Loading **extends** StatefulWidget {  
 @override  
 \_LoadingState createState() => \_LoadingState();  
}  
  
**class** \_LoadingState **extends** State<Loading> {  
  
 **void** setupWorldTime() **async** {  
 WorldTime instance=WorldTime(location: **'Kolkata'**,flag: **'india.png'**,url: **'Asia/Kolkata'**);  
 **await** instance.getTime();  
 Navigator.*pushReplacementNamed*(**context**, **'/home'**,arguments: {  
 **'location'**:instance.**location**,  
 **'time'**:instance.**time**,  
 **'flag'**:instance.**flag**,  
 **'eventOfDay'**:instance.**eventOfDay**,  
 **'url'**:instance.**url** });  
  
 }  
  
 @override  
 **void** initState() {  
 **super**.initState();  
 setupWorldTime();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 **return** Scaffold(  
 backgroundColor: Colors.*blue*[900],  
 body: Center(  
 child: SpinKitRotatingCircle(  
 color: Colors.*white*,  
 size: 50.0,  
 ),  
  
 )  
 );  
 }  
}

Coming to the home screen, we need to set the background images. So, we create an assets directory as we have done in the previous lessons, and add our images to it. We also update our pubspec.yaml file to include the assets.

We create a string to store the time of the day received from the loading screen, and call it bgImageChooser. We also make another string called ‘bgImage’ which stores the name of the actual background image. We also make a Color variable which stores the value of the colour of the notif-bar when the background changes:

String bgImagechooser=**data**[**'eventOfDay'**];  
String bgImage;

Color bgColor;

We then again create a small if-else-if ladder to set the value of the bgImage:

**if**(bgImagechooser==**'earlyMorning'**)  
 {  
 bgImage=**'earlymorning.png'**;  
 bgColor=Colors.*purple*[400];  
 }  
**else if**(bgImagechooser==**'morning'**)  
 {  
 bgImage=**'day.png'**;  
 bgColor=Colors.*blue*;  
 }  
**else if**(bgImagechooser==**'afternoon'**)  
{  
 bgImage=**'afternoon.png'**;  
 bgColor=Colors.*orange*[700];  
  
}  
**else if**(bgImagechooser==**'evening'**)  
{  
 bgImage=**'evening.png'**;  
 bgColor=Colors.*red*[800];  
}  
**else if**(bgImagechooser==**'night'**)  
{  
 bgImage=**'night.jpg'**;  
 bgColor=Colors.*indigo*[800];  
}

In order to set the background, we shall be using a widget called ‘box- decoration’ which gives us a mechanism to have a background image which fits the screen.

First of all, we place the entire thing in a container. We then use a property of the Container called ‘decoration’ and set it equal to the ‘box-decoration’ widget. This boxDecoration property has an image property where we specify the background image in a DecorationImage widget.

This widget again has 2 properties: The actual image that we’re going to use (the bgImage which specifies the path), and the fit type. Here we’re using the BoxFit.cover fit. This means that it will cover the entire screen.

We also change the color of the top notification strip using the backgroundColor property of the Scaffold, and we set it equal to the bgColor variable. Making certain minor UI changes to the font-color, and letter spacing, we have the following code:

**import 'package:flutter/material.dart'**;  
**import 'dart:async'**;  
**import 'package:chronograph/services/world\_time.dart'**;  
**import 'package:flutter\_spinkit/flutter\_spinkit.dart'**;  
  
**class** Home **extends** StatefulWidget {  
 @override  
 \_HomeState createState() => \_HomeState();  
}  
  
**class** \_HomeState **extends** State<Home> {  
  
 Map **data**={};

@override  
 Widget build(BuildContext context) {  
   
 **data**= ModalRoute.*of*(context).**settings**.**arguments**;  
 print(**data**);  
  
 *//set background* String bgImagechooser=**data**[**'eventOfDay'**];  
 String bgImage;  
 Color bgColor;  
  
 **if**(bgImagechooser==**'earlyMorning'**)  
 {  
 bgImage=**'earlymorning.png'**;  
 bgColor=Colors.*purple*[400];  
 }  
 **else if**(bgImagechooser==**'morning'**)  
 {  
 bgImage=**'day.png'**;  
 bgColor=Colors.*blue*;  
 }  
 **else if**(bgImagechooser==**'afternoon'**)  
 {  
 bgImage=**'afternoon.png'**;  
 bgColor=Colors.*orange*[700];  
  
 }  
 **else if**(bgImagechooser==**'evening'**)  
 {  
 bgImage=**'evening.png'**;  
 bgColor=Colors.*red*[800];  
 }  
 **else if**(bgImagechooser==**'night'**)  
 {  
 bgImage=**'night.jpg'**;  
 bgColor=Colors.*indigo*[800];  
 }  
  
   
 **return** Scaffold(  
 backgroundColor: bgColor,  
 body: SafeArea(  
 child: Container(  
 decoration: BoxDecoration(  
 image: DecorationImage(  
 image: AssetImage(**'assets/**$bgImage**'**),  
 fit: BoxFit.**cover** )  
 ),  
 child: Padding(  
 padding: **const** EdgeInsets.fromLTRB(0, 120.0, 0, 0),  
 child: Column(  
 children: <Widget>[  
 FlatButton.icon(  
 onPressed: (){},  
 icon: Icon(  
 Icons.*edit\_location*,  
 color: Colors.*grey*[800],  
 ),  
 label: Text(  
 **'Edit Location'**,  
 style: TextStyle(  
 color: Colors.*grey*[800]  
 ),  
 )  
 ),  
 SizedBox(height: 20.0),  
 Row(  
 mainAxisAlignment: MainAxisAlignment.**center**,  
 children: <Widget>[  
 Stack(  
 children: <Widget>[  
 *// Stroked text as border.* Text(  
 **data**[**'location'**],  
 style: TextStyle(  
 fontSize: 28.0,  
 letterSpacing: 2.0,  
 foreground: Paint()  
 ..**style** = PaintingStyle.**stroke** ..**strokeWidth** = 0.7  
 ..**color** = Colors.*black*,  
 ),  
 ),  
 *// Solid text as fill.* Text(  
 **data**[**'location'**],  
 style: TextStyle(  
 fontSize: 28.0,  
 letterSpacing: 2.0,  
 color: Colors.*white*,  
 ),  
 ),  
 ],  
 )  
 ],  
 ),  
 SizedBox(height: 20.0),

Stack(  
 children: <Widget>[  
 *// Stroked text as border.* Text(  
 **data**[**'time'**],  
 style: TextStyle(  
 fontSize: 66.0,  
 letterSpacing: 2.0,  
 foreground: Paint()  
 ..**style** = PaintingStyle.**stroke** ..**strokeWidth** = 0.7  
 ..**color** = Colors.*black*,  
 ),  
 ),  
 *// Solid text as fill.* Text(  
 **data**[**'time'**],  
 style: TextStyle(  
 fontSize: 66.0,  
 letterSpacing: 2.0,  
 color: Colors.*white*,  
 ),  
 ),  
 ],  
 ),  
 SpinKitFoldingCube(  
 color: Colors.*white*,  
 size: 40.0,  
 ),  
 ],  
 ),  
 ),  
 )  
 ),  
 );  
 }  
}

The output for various backgrounds looks like: (backgrounds deliberately hardcoded for testing, the in the actual app will have backgrounds varying with time of day):

