



Empowering Digital Skills For The Jobs Of The Future



by



The background of the slide features a series of overlapping, wavy bands in various shades of blue and purple. These bands flow horizontally across the frame, creating a sense of movement and depth. The colors range from deep, dark blues and purples to lighter, almost white tones, with the most intense colors concentrated at the top and bottom edges.

Dart Client Side

Docente



Claudia Infante



claudia.infante@bcsoft.net

Sommario

- Custom Library
- Future
- Async
- Http

Librerie

In Dart esistono diverse librerie oltre `dart:html`. È inoltre possibile anche creare delle librerie personalizzate.

Librerie

Le librerie custom si dividono in due tipi :

1. Librerie esterne : importate da pacchetti esterni
2. Librerie locali : definite all'interno del progetto

Librerie

Una libreria non è altro che un insieme di funzioni e classi utilizzate per incapsulare funzionalità specifiche. Sono utili perché favoriscono il riuso, l'astrazione, l'organizzazione del codice e ottimizzano le performance.

Librerie

- **Riuso** : si crea codice riutilizzabile in più porzioni dell'applicazione evitando codice duplicato.
- **Astrazione** : forniscono modelli di interfacce pulite da utilizzare anche in altri progetti.

Librerie

- **Organizzazione del codice** : le librerie custom organizzano il codice in unità logiche rendendo più semplice la manutenzione.
- **Ottimizzazioni delle Performance** : incapsulando calcoli in moduli separati si migliora l'efficienza del codice.

Librerie

Le librerie sono un concetto importante nella programmazione orientata agli oggetti, perché aiutano a minimizzare l'accoppiamento stretto e a rendere il codice mantenibile.

Librerie

Per creare una libreria si utilizza la parola chiave `library` seguita dal nome della libreria:

```
library my_library_name;
```

Si potrà poi importare la libreria nel file dart desiderato grazie alla sintassi di `import`:

```
import "path/to/my_library_name";
```

Time converter library

Si parta quindi dalla creazione della libreria, che potrà fare uso di una classe in modo tale da richiamare staticamente i metodi associati.

Time converter library

time_converter.dart X

web > time_converter.dart > ...

```
1  library time_converter;
2
3  String convertToHours(int value) {
4      if (value > 60) {
5          int hours = value ~/ 60;
6          int minutes = value % 60;
7          return '$hours ore e $minutes minuti';
8      } else {
9          return '$value minuti';
10     }
11 }
12
```

Time converter library

```
<> index.html X
web > <> index.html > ...
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5      <meta charset="utf-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <meta name="scaffolded-by" content="https://github.com/dart-lang/sdk">
9      <title>Time converter</title>
10     <link rel="stylesheet" href="styles.css">
11     <script defer src="main.dart.js"></script>
12 </head>
13
14 <body>
15
16     <input type="number" id="timeInput" placeholder="Minuti da convertire">
17     <button id="convertBtn">Converti</button>
18     <p id="result"></p>
19
20 </body>
21 </html>
22
23 <!DOCTYPE html>
24
25
```

Time converter library

```
main.dart X
web > main.dart > ...
1  import 'dart:html';
2  import 'time_converter.dart';
3
   Run | Debug
4  void main() {
5      var convertBtn = document.querySelector('#convertBtn');
6      var timeInput = document.querySelector('#timeInput');
7      var result = document.querySelector('#result');
8
9
10     convertBtn?.onClick.listen((event) {
11         if (timeInput is InputElement) {
12             String? inputValue = (timeInput).value;
13             if (inputValue != null) {
14                 int parsedValue = int.tryParse(inputValue) ??
15                 | 0; // Valore di default se il parsing fallisce
16                 String convertedTime = convertToHours(parsedValue);
17                 result!.text = '$inputValue minuti equivalgono a $convertedTime ';
18             } else {
19                 // valore di default se il valore è null
20                 result!.text = '0';
21             }
22         }
23     });
24
25
26 }
27
```

Flutter - C. Infante

Callback e programmazione asincrona

Grazie alla gestione asincrona, Dart può effettuare operazioni come chiamate API richiedendo dati da un server utilizzando una `HttpRequest`.

Callback e programmazione asincrona

L' applicazione chiederà dei dati dal server e ritornerà un controllo per la loro gestione.

Callback e programmazione asincrona

Tale gestione avverrà controllando il loop degli eventi il quale chiama la funzione anonima event-handler una volta che il server ha trasmesso i dati.

Callback e programmazione asincrona

Esempio di una funzione di callback per una HttpRequest

```
HttpRequest.get("http://example.com", (data) {  
  // handle data being returned  
});
```

Callback e programmazione asincrona

Le callback sono utili quando si desidera eseguire un'azione solo dopo il completamento di un'operazione asincrona o in risposta a un determinato evento.

Callback e programmazione asincrona

In un modello sincrono, tutto avviene in ordine, in attesa che la fase precedente sia completamente completata. Questo approccio è utile in alcuni casi ma, nelle web non è possibile bloccare l'esecuzione mentre si attende il completamento dell'attività precedente.

Callback e programmazione asincrona

In JavaScript, si usano funzioni di callback con il modello di programmazione async e si può fare lo stesso in Dart.

Future

Dart mette per la gestione asincrona mette a disposizione la classe Future.

Le sue istanze rappresentano il risultato di un'operazione asincrona che ha due stati : completa o incompleta.



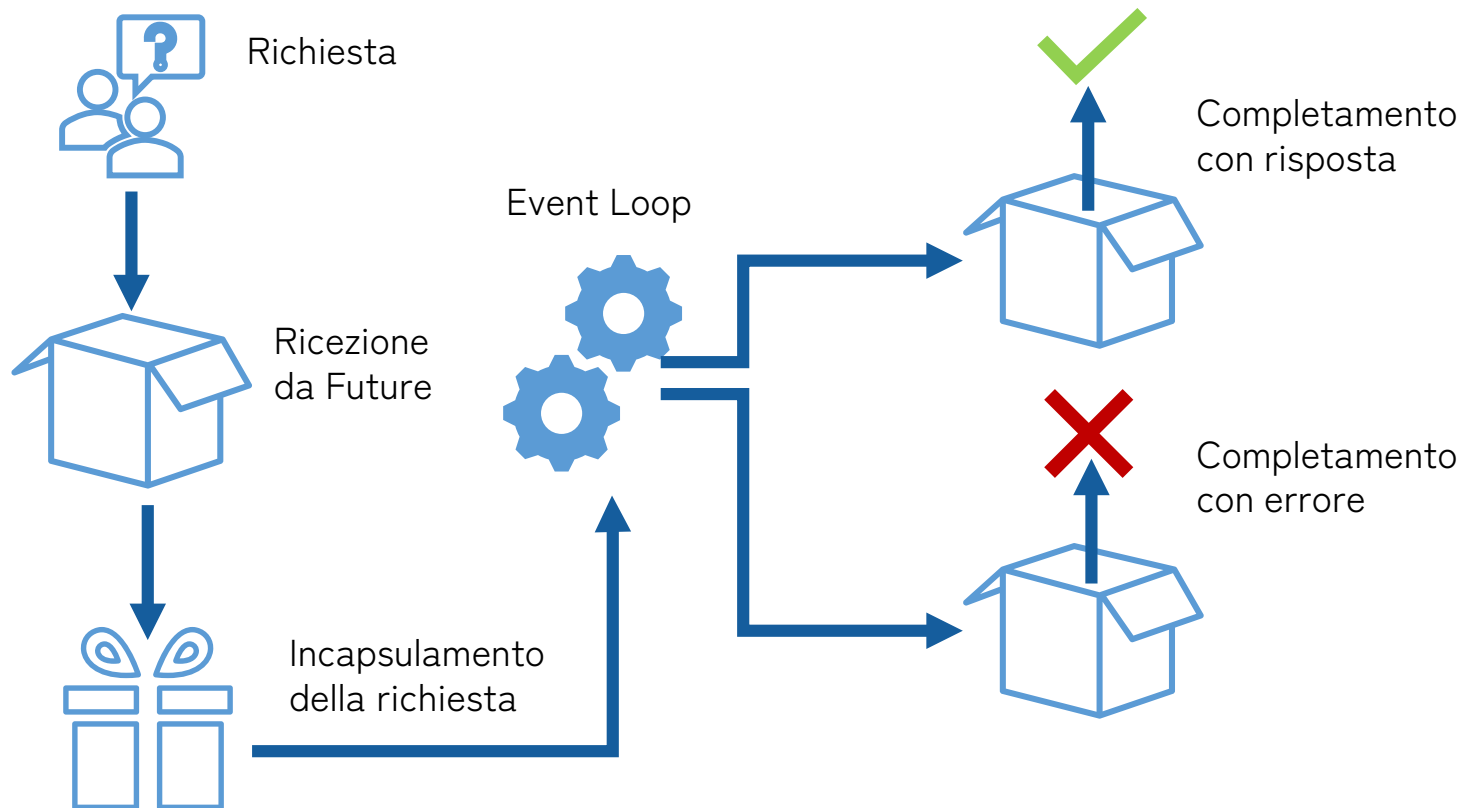
Future

Incompleta (Uncompleted) è il caso in cui l'istanza di Future sta aspettando che l'operazione asincrona finisca o lanci un errore.

Future

Completa : nel caso in cui l'operazione asincrona vada a buon fine e viene emesso un valore. Altrimenti si completa con un messaggio di errore.

Future



Future

Il valore emesso è di tipo T. Per esempio, un'istanza con tipo `Future<String>` produce un valore di tipo stringa. Se non sono prodotti valori utilizzabili sarà di tipo `Future<void>`.

Future

Quale delle due funzioni sarà eseguita per prima?

```
1 ▾ Future<void> fetchUserMessage() {  
2     return Future.delayed(  
3         const Duration(seconds: 2), () => print('Hello')  
4     );  
5 }  
6  
7 ▾ void main() {  
8     fetchUserMessage();  
9     print('Goodmorning!');  
10 }
```

Future

`fetchUserMessage()` viene eseguita per prima ma il messaggio viene stampato con un ritardo di due secondi, per questo verrà stampato prima il messaggio Goodmorning e poi Hello.



The screenshot shows an IDE window with a Dart file. The code defines a function `fetchUserMessage()` that returns a `Future` delayed by 2 seconds, and a `main()` function that calls `fetchUserMessage()` and prints 'Goodmorning!'. The console output shows 'Goodmorning!' followed by 'Hello' on the next line, demonstrating that the first print statement executes immediately while the second waits for the `Future` to complete.

```
Dart
Install SDK  Format  Reset  Run

1 Future<void> fetchUserMessage() {
2   return Future.delayed(
3     const Duration(seconds: 2), () => print('Hello')
4   );
5 }
6
7 void main() {
8   fetchUserMessage();
9   print('Goodmorning!');
10 }
```

Console

Goodmorning!
Hello

Future

Il Future sarà di tipo void in quanto ritorna un messaggio nella console.

Future

In caso di errori si lancerà un'eccezione

```
1▼ Future<void> fetchUserData() {  
2    return Future.delayed(  
3        const Duration(seconds: 2),  
4        () => throw Exception('Login failed: user ID is invalid'),  
5    );  
6 }  
7  
8▼ void main() {  
9    fetchUserData();  
10   print('Fetching user data...');  
11 }
```

Future: async e await

Async e await sono due parole chiave utilizzate per definire delle funzioni asincrone e utilizzare i loro risultati.

Future: async e await

- Per definire una funzione asincrona, aggiungere `async` prima del corpo della funzione.
- La parola chiave `await` funziona solo nelle funzioni asincrone.

Future: async e await

Seguendo queste indicazioni, si trasforma il codice precedente utilizzando async e await

```
1▼ Future<void> fetchUserMessage()async {  
2    await Future.delayed(  
3        const Duration(seconds: 2), () => print('Hello')  
4    );  
5 }  
6  
7▼ void main() {  
8    fetchUserMessage();  
9    print('Goodmorning!');  
10 }  
11
```

Future: async e await

- La parola chiave `async` appare prima del corpo della funzione;
- La parola chiave `await` appare prima di chiamare la funzione asincrona

Future: async e await

È possibile anche avere più funzioni asincrone e attendere il completamento dell'una per far scattare l'altra.

Future: async e await

```
1 Future<String> createUserMessage() async {  
2   var message = await fetchUserMessage();  
3   return 'User says: $message';  
4 }  
5  
6 Future<String> fetchUserMessage() =>  
7   Future.delayed(  
8     const Duration(seconds: 2),  
9     () => 'Goodmorning!',  
10  );  
11  
12 Future<void> main() async {  
13   print('Fetching user message...');  
14   print(await createUserMessage());  
15 }
```

Deve attendere l'esecuzione di
fetchUserMessage

Mostrerà il messaggio dopo 2
secondi

Parola chiave async perché una delle
funzioni richiamate è asincrona e sarà
preceduta dalla parola chiave await

Future: async e await

- Il tipo di ritorno di `createUserMessage` diventa `Future<String>`
- La parola chiave `async` appare prima del corpo delle funzioni `createUserMessage()` e `main()`;
- La parola chiave `await` appare prima di chiamare le funzioni asincrone: `fetchUserMessage()` e `createUserMessage()`.
- Il codice non preceduto da `await` viene eseguito immediatamente

Attività

Implementando codice asincrono scrivere una serie di funzioni:

- `createUserName()` : Mostra il nome di un utente;
- `fetchUserName()` : Definisce il nome di un utente;
- `presentUser()` : una volta mostrato il nome dell'utente, questa funzione riceve il nome di un altro utente creato in precedenza;
- `sayHello()` : riceve come parametro il nome dell'utente definito dalla funzione `fetchUserName()` e il nome del secondo utente, sempre creato in precedenza. Ad esempio, se `fetchUserName` ritorna 'Andrea', la funzione ritornerà 'Ciao Paolo, sono Andrea'.

Attività

Esempio del risultato

```
Fetching user Name...  
First user name is: Andrea  
I'm Paolo !  
I'm Andrea , hello Paolo!
```


Attività Soluzione

Si definisce la funzione asincrona per la creazione del primo utente che dovrà attendere il nome dell'utente affinché possa essere mostrato :

```
1▼ Future<String> createUserName() async {  
2    var user1 = await fetchUserName();  
3    return 'First user name is: $user1';  
4 }  
5  
6 Future<String> fetchUserName() =>  
7     Future.delayed(  
8         const Duration(seconds: 2),  
9         () => 'Andrea',  
10    );  
11
```

Attività Soluzione

Successivamente si crea la funzione per il secondo utente

```
11
12 Future<String> fetchUserName2() =>
13     Future.delayed(
14         const Duration(seconds: 4),
15         () => 'Paolo',
16     );
17
18
19 ▾ Future<String> presentUser() async {
20     var user2 = await fetchUserName2();
21     return 'I\'m $user2 !';
22 }
23
```

Attività Soluzione

La funzione che riceve entrambi gli utenti :

```
--  
24 ▾ Future<String> sayHello() async {  
25     var greeting = await fetchUserName();  
26     var user2 = await fetchUserName2();  
27     return 'I\'m $greeting , hello $user2!';  
28 }  
29
```

Programmazione asincrona

Si supponga di avere un'applicazione Lotteria che mostra in modo randomico i numeri vincitori. La logica dell'applicazione prevedrà una libreria che rappresenta la gestione asincrona dell'API e fornisce informazioni per la generazione dei numeri mostrati a video.

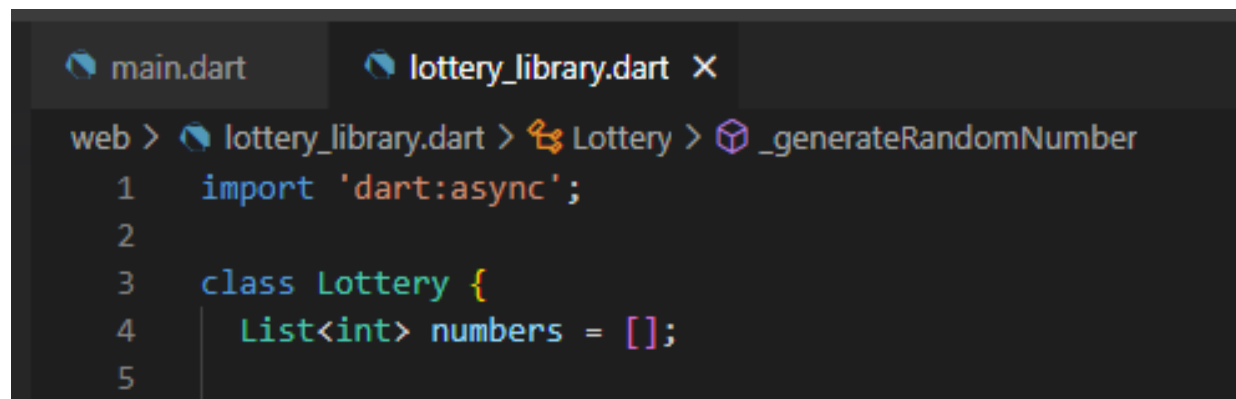
Programmazione asincrona

I numeri saranno mostrati grazie alla funzione `main()` e importerà la libreria.

Programmazione asincrona

All'interno del file creato per la libreria `lottery_library.dart` si importa `dart:async`.

Si definisce poi una variabile per conservare i numeri random.



The screenshot shows an IDE with two tabs: `main.dart` and `lottery_library.dart`. The `lottery_library.dart` tab is active, showing the following code:

```
web > lottery_library.dart > Lottery > _generateRandomNumber
1  import 'dart:async';
2
3  class Lottery {
4    List<int> numbers = [];
5
```

Programmazione asincrona

Si definisce poi un metodo per effettuare la creazione dei numeri. Ogni numero sarà creato dopo 1 secondo. Per fare ciò si utilizza Future che consente la gestione asincrona dei dati.

Programmazione asincrona

Il metodo `generateNumbers` è responsabile della generazione di 3 numeri casuali con un ritardo di 1 secondo tra una generazione e l'altra. All'interno del metodo, viene eseguito un ciclo per 3 volte per generare numeri casuali utilizzando il metodo `_generateRandomNumber` che si occuperà di aggiungere numeri all'elenco.

Programmazione asincrona

Ogni numero generato viene aggiunto all'elenco e viene stampato un messaggio per indicare che il numero è stato generato.

```
6      Future<void> generateNumbers() async {  
7          for (int i = 0; i < 3; i++) {  
8              await Future.delayed(Duration(seconds: 3));  
9              int number = _generateRandomNumber();  
10             numbers.add(number);  
11             print('Number ${numbers[i]} generato.');
```

Programmazione asincrona

Il metodo `_generateRandomNumber` genera un numero casuale compreso tra 1 e 100 basato sui millisecondi correnti dell'orologio di sistema.

```
int _generateRandomNumber() {  
  return (1 + (DateTime.now().millisecondsSinceEpoch % 100)).abs();  
}
```

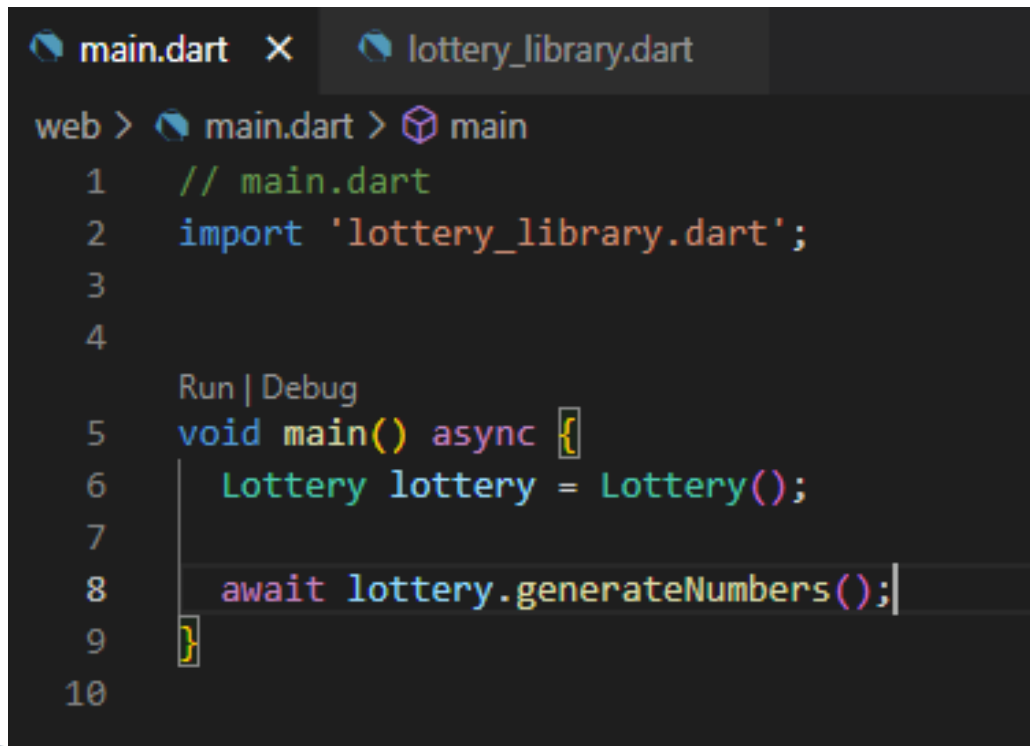
Programmazione asincrona

lottery_library.dart

```
main.dart  lottery_library.dart X
web > lottery_library.dart > Lottery
1  import 'dart:async';
2
3  class Lottery {
4    List<int> numbers = [];
5
6    Future<void> generateNumbers() async {
7      for (int i = 0; i < 3; i++) {
8        await Future.delayed(Duration(seconds: 3));
9        int number = _generateRandomNumber();
10       numbers.add(number);
11       print('Number ${numbers[i]} generato.');
```

Programmazione asincrona

main.dart



```
main.dart x lottery_library.dart
web > main.dart > main
1 // main.dart
2 import 'lottery_library.dart';
3
4
5 Run | Debug
6 void main() async {
7     Lottery lottery = Lottery();
8     await lottery.generateNumbers();
9 }
10
```

Programmazione asincrona

Per mostrarli nella UI, la libreria dovrà importare la libreria built in di dart:html così da selezionare l'elemento della view da valorizzare.

```
6
7   Future<void> generateNumbers() async {
8       for (int i = 0; i < 3; i++) {
9           await Future.delayed(Duration(seconds: 3));
10          int number = _generateRandomNumber();
11          numbers.add(number);
12          print('Numero ${numbers[i]} generato.');
13          querySelector('.number-$i')?.text = '$number';
14      }
15  }
16
```

Programmazione asincrona

Index.html

```
main.dart x index.html x lottery_library.dart # styles.css
web > <> index.html > ...
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5      <meta charset="utf-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <meta name="scaffolded-by" content="https://github.com/dart-lang/sdk">
9      <title>Time converter</title>
10     <link rel="stylesheet" href="styles.css">
11     <script defer src="main.dart.js"></script>
12 </head>
13 <div class="container">
14 <div class="randomNum number-0"> </div>
15 <div class="randomNum number-1"> </div>
16 <div class="randomNum number-2"> </div>
17 </div>
18 <body>
19
20 </body>
21 </html>
```

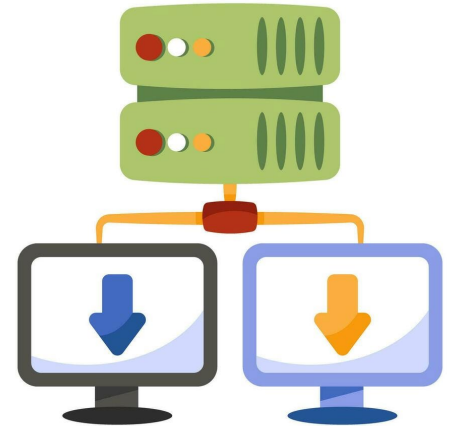
Programmazione asincrona

style.css

```
main.dart # styles.css X index.html lottery_library.dart
web > # styles.css > .randomNum
1  @import url(https://fonts.googleapis.com/css?family=Roboto);
2
3  html, body {
4    width: 100%;
5    height: 100%;
6    margin: 0;
7    padding: 0;
8    font-family: 'Roboto', sans-serif;
9  }
10
11 .container{
12   display: flex;
13   margin: 2%;
14   justify-content: space-between;
15 }
16 .randomNum{
17   padding: 8%;
18   background-color:  rgb(141, 230, 178);
19   border-radius: 50%;
20 }
```

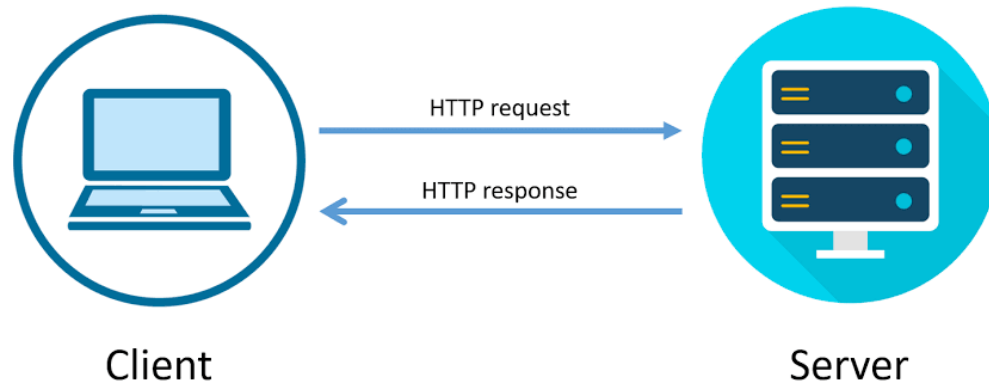
Client to server

Come accennato in precedenza, le funzioni asincrone sono molto utili per recuperare dei dati esterni all'applicazione.



Client to server

Molte applicazioni eseguono questa operazione tramite richieste HTTP, che vengono inviate da un client a un server per eseguire un'azione specifica per una risorsa identificata tramite un URI (Uniform Resource Identifier).



Client to server

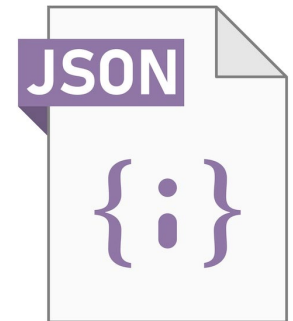
I dati comunicati via HTTP possono essere di qualsiasi tipo ma solitamente sono sottoforma di oggetti JSON (JavaScript Object Notation) grazie alla sua facilità di lettura e alla natura indipendente dal linguaggio.

Client to server

Anche l'SDK e l'ecosistema Dart offrono un ampio supporto per JSON, con diverse opzioni per soddisfare al meglio i requisiti della propria applicazione.

JSON

Il formato JSON (JavaScript Object Notation) è un formato basato su una sintassi di oggetti e array che rappresenta i dati in un formato leggibile sia per gli esseri umani che per le macchine.



JSON

In JSON, i dati sono rappresentati come coppie chiave-valore all'interno di oggetti, e gli oggetti possono essere annidati all'interno di altri oggetti o di array.

JSON

In questo esempio, si ha un oggetto JSON che rappresenta le informazioni di una persona con nome, cognome, età, indirizzo e interessi. Le chiavi sono stringhe e i valori possono essere stringhe, numeri, oggetti, array, booleani o null.

```
{
  "nome": "Mario",
  "cognome": "Rossi",
  "eta": 30,
  "indirizzo": {
    "via": "Via Roma",
    "citta": "Milano"
  },
  "interessi": ["calcio", "musica", "viaggi"]
}
```

HttpRequest

Lo scambio di informazioni tra lato client e server si basa sul protocollo http che definisce una serie di metodi di interazione come il get per il recupero dei dati, e il post per il loro invio.

HttpRequest

In Dart, queste richieste vengono gestite con il pacchetto della libreria `package:http`.

HttpRequest

Per effettuare una richiesta HTTP, è necessario fornire un URI (Uniform Resource Identifier) alla risorsa.

Un URI è una stringa di caratteri che identifica in modo univoco una risorsa.

Un URL (Uniform Resource Locator) è un tipo specifico di URI che fornisce anche la posizione della risorsa.

HttpRequest

Gli URL delle risorse sul Web contengono tre informazioni:

- Lo schema del protocollo usato (http o https)
- L'hostname del server
- Il percorso della risorsa

HttpRequest

Per esempio per il path di wikipedia

https://it.wikipedia.org/wiki/Pagina_principale si avrà :

- Lo schema del protocollo usato : https
- L'hostname del server : it.wikipedia.org
- Il percorso della risorsa : /wiki/Pagina_principale

HttpRequest

Ci sono poi anche dei parametri opzionali che possono essere passati.

HttpRequest

In dart ci sono vari modi per eseguire richieste http come l'uso del pacchetto http o del pacchetto html.

HttpRequest

Per aggiungere il pacchetto http, nella folder principale del progetto eseguire il comando `dart pub add http`.

Questo comando aggiornerà le dipendenze del file `pubspec.yaml`

```
9   # Add regular dependencies here.  
10  dependencies:  
11    http: ^1.2.0  
12    # path: ^1.8.0  
13
```

HttpRequest

Nel file in cui saranno eseguite le chiamate si utilizzerà la sintassi di import :

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

HttpRequest

Per interpretare i dati in modo che questi possano essere mostrati nella view si utilizzerà `dart:convert` che consentirà di decodificare i dati.

HttpRequest

Si definisce una funzione asincrona per il recupero dei dati:

```
void fetchUsers() async {  
  var response =  
    await HttpRequest.getString('https://jsonplaceholder.typicode.com/users');  
  List<dynamic> users = jsonDecode(response);  
  
  displayUsers(users);  
}
```

HttpRequest

Si definisce il punto della view in cui si vogliono mostrare :

```
15
16 void displayUsers(List<dynamic> users) {
17   var userList = querySelector('#user-list');
18
19   for (var user in users) {
20     var userElement = Element.html('<li>${user['name']}</li>');
21     userList?.append(userElement);
22   }
23 }
24
```

HttpRequest

Si definisce il punto di invocazione del metodo nella funzione `main()`:

```
4 void main() {  
5   | fetchUsers();  
6 }
```

HttpRequest

File dart :

```
main.dart x index.html
web > main.dart > fetchUsers
1  import 'dart:convert';
2  import 'dart:html';
3
   Run | Debug
4  void main() {
5    |   fetchUsers();
6  }
7
8  void fetchUsers() async {
9    |   var response =
10     |   |   await HttpRequest.getString('https://jsonplaceholder.typicode.com/users');
11     |   List<dynamic> users = jsonDecode(response);
12
13     |   displayUsers(users);
14   }
15
16  void displayUsers(List<dynamic> users) {
17    |   var userList = querySelector('#user-list');
18
19    |   for (var user in users) {
20     |   |   var userElement = Element.html('<li>${user['name']}</li>');
21     |   |   userList?.append(userElement);
22     |   }
23   }
24
```

HttpRequest

File html :

```
main.dart  index.html X
web > index.html > html > body > script
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5      <meta charset="utf-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <meta name="scaffolded-by" content="https://github.com/dart-lang/sdk">
9      <title>Dart Web App </title>
10 </head>
11 <body>
12     <h1>User List</h1>
13     <ul id="user-list"></ul>
14
15     <script defer src="main.dart.js"></script>
16 </body>
17 </html>
```

HttpRequest

Avviare il progetto con il comando *webdev serve* e modificare il riferimento allo script di esecuzione del file `main.dart` in `main.dart.js`, in modo che il codice inserito all'interno venga interpretato come codice javascript.

HttpRequest

Nel caso in cui si voglia utilizzare il pacchetto http, più specifico, è possibile definire l'url da leggere e mostra i dati quando la risposta va a buon fine.

HttpRequest

File dart :

```
main.dart X index.html
web > main.dart > fetchUsers
1  import 'dart:convert';
2  import 'package:http/http.dart' as http;
3  import 'dart:html';
4
Run | Debug
5  void main() {
6    |   fetchUsers();
7  }
8
9  void fetchUsers() async {
10   |   var response =
11   |   |   await http.get(Uri.parse('https://jsonplaceholder.typicode.com/users'));
12   |
13   |   if (response.statusCode == 200) {
14   |   |   List<dynamic> users = jsonDecode(response.body);
15   |   |   displayUsers(users);
16   |   } else {
17   |   |   print('Failed to fetch users. Error: ${response.reasonPhrase}');
18   |   }
19   |
20   |
21   void displayUsers(List<dynamic> users) {
22   |   |   var userList = querySelector('#user-list');
23   |   |
24   |   |   for (var user in users) {
25   |   |   |   var userElement = Element.html('<li>${user['name']}</li>');
26   |   |   |   userList?.append(userElement);
27   |   |   }
28   |   }
29 }
```

Flutter - C. Infante

dart:html vs dart:http

La libreria `dart:html` è specifica per le applicazioni web e fornisce API per interagire con il DOM (Document Object Model) e le funzionalità del browser.

dart:html vs dart:http

La classe `HttpRequest` di `dart:html` è utilizzata per effettuare richieste HTTP direttamente dall'ambiente del browser lato client. È adatta alle applicazioni web in cui è necessario interagire con il DOM e gestire gli eventi, oltre a effettuare richieste HTTP.

dart:html vs dart:http

Il pacchetto http è un pacchetto multipiattaforma che può essere utilizzato in applicazioni Dart sia web che lato server. Fornisce un'API più flessibile e potente per effettuare richieste HTTP rispetto a dart:html.

dart:html vs dart:http

Il pacchetto http è comunemente usato nelle applicazioni Dart lato server, nelle applicazioni Flutter e nelle applicazioni web in cui non è richiesta la manipolazione diretta del DOM.

The background features a series of overlapping, wavy bands in various shades of blue and purple. The top section has darker, more saturated blue and purple tones, while the bottom section transitions into lighter, more ethereal shades of lavender and pale blue. The waves flow horizontally across the frame, creating a sense of movement and depth.

Domande e approfondimenti

Attività

Dato un campo di input e un bottone si crei una web app in dart che, tramite una libreria custom mostra nella view il numero di secondi mancanti. L'utente inserisce nel campo di input il valore iniziale da cui partire e grazie alla libreria custom definita, si mostra nella view il conto alla rovescia.

Tra quanti secondi vuoi far partire il timer?

Countdown: 11 secondi rimanenti