



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 from the top and bottom edges towards the center, creating a sense of movement and depth. The colors range from deep indigo and navy blue to lighter, almost white, lavender and pale blue tones. The central area of the slide is a clean, white space where the text is located.

# Dart Client Side

# Docente



Claudia Infante



[claudia.infante@bcsoft.net](mailto:claudia.infante@bcsoft.net)

# Sommario

- Uso di dart client side
- SPA
- Primo progetto
- Dart:html
- Elementi
- Stile

# Uso di dart Client side

La tecnologia client-side si riferisce a tutti i processi e le funzioni che vengono eseguiti sul lato del client di un'applicazione web. Questo significa che tutte le operazioni vengono eseguite sul computer o sul dispositivo dell'utente, utilizzando le risorse hardware e software disponibili sul suo dispositivo.

# Uso di dart Client side

Esempi di tecnologie client-side includono HTML, CSS e JavaScript, che vengono utilizzati per creare l'aspetto e il comportamento delle pagine web sul browser dell'utente.

# Uso di dart Client side

Altre tecnologie client-side includono anche l'utilizzo di cookie e il caching per memorizzare informazioni sul dispositivo dell'utente al fine di migliorare le prestazioni e la personalizzazione dell'esperienza utente.

# Uso di dart Client side

L'uso principale di Dart consiste nella creazione di web app complesse. Una web app solitamente è costituita da una single page application che, mostrata nella finestra del browser, contiene del codice interpretato in modo tale che l'utente possa interagire con i contenuti proposti.



# Uso di dart Client side

Quando si costruisce un'applicazione web in Dart, è buona norma costruirla lavorando offline.

# Uso di dart Client side

Lavorare offline ha una serie di vantaggi, sia per lo sviluppatore che per l'utente finale.

Quando si costruisce un'applicazione web offline per prima cosa, si forniscono alcuni dati finti che alla fine saranno sostituiti da una combinazione di archiviazione locale, lato client (supportata dalle API HTML5) e persistenza dei dati lato server.

# Uso di dart Client side

Questa configurazione consente di scrivere l'applicazione lato client in Dart senza bisogno di un componente server o di un databaseserver o di un database, riducendo così i tempi di debug e favorendo la possibilità di eseguire degli unit test dell'applicazione.

# SPA

Le principali differenze tra una single page application e un sito web tradizionale sono:

1. Caricamento dei contenuti: Nelle single page application, tutti i contenuti vengono caricati una sola volta all'avvio dell'applicazione e successivamente vengono aggiornati dinamicamente senza dover ricaricare l'intera pagina. Nei siti web tradizionali, invece, ogni volta che un utente naviga in una nuova pagina, viene caricata una pagina separata, rendendo il processo più lento.

# SPA

2. Esperienza utente: Le single page application offrono un'esperienza utente più fluida e interattiva, grazie al caricamento rapido dei contenuti e alla navigazione senza interruzioni. I siti web tradizionali possono essere più statici e meno coinvolgenti per gli utenti.

# SPA

3. SEO: I motori di ricerca possono avere difficoltà a indicizzare correttamente le single page application, poiché non riescono a scansionare il codice JavaScript per rilevare tutti i contenuti. Al contrario, i siti web tradizionali sono più facili da indicizzare e posizionare nei risultati di ricerca.

# SPA

4. Velocità di caricamento: Le single page application tendono ad essere più veloci nel caricare i contenuti, poiché non devono ricaricare l'intera pagina ogni volta che un utente fa clic su un link. I siti web tradizionali possono essere più lenti nel caricare le pagine, soprattutto se contengono molti contenuti multimediali o elementi grafici complessi.

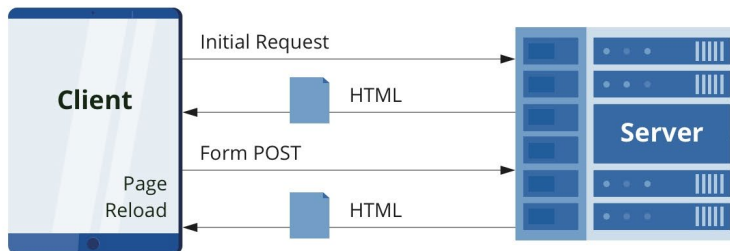
# SPA

5. Struttura e architettura: Le single page application sono generalmente basate su framework JavaScript come Angular, React o Vue, che consentono di organizzare il codice in moduli e componenti riutilizzabili. I siti web tradizionali possono essere costruiti utilizzando diversi linguaggi di programmazione e tecnologie, come HTML, CSS e JavaScript.



# SPA

## Traditional Page Lifecycle



## SPA Lifecycle



# Primo progetto

Creare un nuovo progetto con template web con il comando:  
`dart create -t web demo-web`

Posizionarsi nella cartella del progetto e lanciare il comando:

`dart pub global activate webdev`

far partire il progetto con il comando :

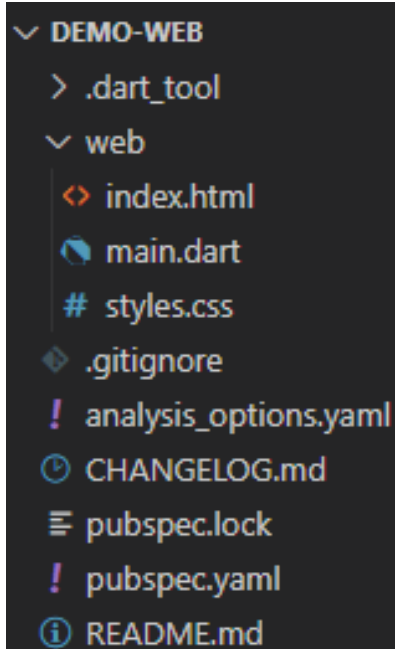
`webdev serve`

# Primo progetto

La porta in ascolto sarà la localhost:8080

# Primo progetto

Il progetto conterrà una serie di folder tra cui la cartella web contenente i file index.html e il file main.dart



```
▼ DEMO-WEB
  > .dart_tool
  ▼ web
    <> index.html
    🌐 main.dart
    # styles.css
    💎 .gitignore
    ! analysis_options.yaml
    🕒 CHANGELOG.md
    ≡ pubspec.lock
    ! pubspec.yaml
    ⓘ README.md
```

A screenshot of a file explorer interface showing the project structure of a Flutter web application. The root folder is 'DEMO-WEB', which is expanded. It contains a subfolder '.dart\_tool' and another subfolder 'web'. The 'web' folder is also expanded, showing files 'index.html' (with a code icon), 'main.dart' (with a Dart icon), and 'styles.css' (with a CSS icon). Below the 'web' folder, several other files are listed: '.gitignore' (with a diamond icon), 'analysis\_options.yaml' (with an exclamation mark icon), 'CHANGELOG.md' (with a clock icon), 'pubspec.lock' (with a list icon), 'pubspec.yaml' (with an exclamation mark icon), and 'README.md' (with an information icon).

# dart:html

I contenuti vengono strutturati utilizzando il linguaggio di markup HTML. Esso può essere utilizzato anche in Dart, grazie alla libreria dart:html.

# dart:html

Per programmare per il browser, lavorare con gli oggetti e gli elementi del DOM e accedere alle API HTML5, si può utilizzare la libreria `dart:html`, specifica per le web application e non può essere utilizzata in ambienti non web come applicazioni server side.

# dart:html

Dart:html è anche usata frequentemente per lo scambio di dati tramite WebSockets, il recupero di dati tramite richieste HTTP e la manipolazione dello stile (CSS).

# dart:html

Per utilizzare la libreria importare nella web app dart:html.

```
import 'dart:html';
```





# dart:html

Il file main.dart prevede questo import e la funzione main

A screenshot of a code editor window titled 'main.dart'. The editor shows Dart code for a web application. Line 1: 'import 'dart:html';'. Line 2: empty. Line 3: 'void main() {' with a 'Run | Debug' tooltip above it. Line 4: ' querySelector('#output')?.text = 'Your Dart app is running.';'. Line 5: '}'. Line 6: empty. The code is color-coded: 'import' is blue, 'void' is blue, 'main()' is blue, 'querySelector' is blue, and the string values are orange. The background of the editor is dark with light gray line numbers.

```
main.dart X
web > main.dart > ...
1  import 'dart:html';
2
   Run | Debug
3  void main() {
4    querySelector('#output')?.text = 'Your Dart app is running.';
5  }
6  |
```

# dart:html

Il file `dart` contiene la funzione `main()` dell'applicazione, che viene richiamata automaticamente quando il codice Dart è completamente caricato e pronto per essere avviato.

# dart:html

Tutte le applicazioni Dart basate su browser devono essere agganciate ad un file HTML. Questo è il file HTML di ingresso, che fa riferimento al file .dart dell' applicazione usando un tag HTML `<script>`.

# dart:html

Il file HTML minimo deve essere come il seguente :

```
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>demo_web</title>
10     <link rel="stylesheet" href="styles.css">
11     <script defer src="main.dart.js"></script>
12 </head>
13
14 <body>
15     <div id="output"></div>
16
17 </body>
18 </html>
19
20
```

# Elementi

Con l'oggetto Document, è possibile creare e manipolare oggetti Element all'interno del documento.  
Si noti che il documento stesso è un elemento e può essere manipolato.

# Elementi

In dart esistono due modi per creare elementi HTML.

Il primo consiste nell'inserimento manuale del tag all'interno della pagina html.

```
<body>  
  <h1> Pagina web con Dart</h1>  
  <div id="output"></div>  
</body>
```

# Elementi

Il secondo consiste nella creazione di una variabile nel file dart :

```
Run | Debug
void main() {
  ⚡querySelector('#output')?.text = 'Your Dart app is running.';
  var myElement = Element.html('<h2 id="title">Testo aggiunto con variabile</h2>');
  querySelector('#output')?.append(myElement);
}
```

# Elementi

Nel codice precedente è stata utilizzata la sintassi `querySelector()` per recuperare un elemento della pagina in base al suo attributo `id`.



# Elementi

Oltre al `querySelector` è presente anche il `querySelectorAll()` che seleziona una collezione di elementi richiesti.

```
// Find all elements by tag (<div>).  
List<Element> divElements = querySelectorAll('div');  
  
// Find all text inputs.  
List<Element> textInputElements = querySelectorAll(  
  'input[type="text"]',  
);
```

# Elementi

È possibile creare qualsiasi elemento e accedere alle sue proprietà tramite la notazione a punti

```
Run | Debug
void main() {
  querySelector('#output').text = 'Your Dart app is running.';
  var myElement = Element.html('<h2 id="title">Testo aggiunto con variabile</h2>');
  querySelector('#output').append(myElement);
  var itemp = Element.tag("p");
  itemp.id = "txt-item";
  itemp.style.color = "blue";
  itemp.text="paragrafo creato e stilizzato con dart";
  querySelector('#output').append(itemp);
}
```

# Elementi

Per aggiungere gli elementi creati al documento, si utilizza il metodo `append()`, invocato sul nodo di riferimento a cui si accede tramite l'elemento `document`.

# Elementi

Altro modo ancora per creare gli elementi oltre ad `Element.tag()` ed `Element.html()` consiste nel fare riferimento al tipo specifico dell'elemento di interesse.

# Elementi

Dart fornisce delle classi per creare degli elementi.

- `HeadingElement`: rappresenta gli elementi di intestazione di un documento html (h1....h6);
- `DivElement`: Rappresenta un element `<div>`, ovvero un container generico.

# Elementi

- **SpanElement:** Un elemento `<span>`, rappresenta un container inline solitamente utilizzato per raggruppare testi o elementi in linea.
- **AnchorElement:** Un tag `<a>`, utilizzato per gli iperlink.

# Elementi

- ImageElement: Rappresenta un tag `<img>`, utilizzato per mostrare delle immagini.
- ParagraphElement: Un elemento `<p>`, per definire i paragrafi di una struttura HTML.

# Elementi

- ListElement: Rappresenta un elemento lista (<ul>, <ol>) per creare liste disordinate e ordinate.
- TableElement: Crea un elemento <table>, utilizzato per rappresentare tabelle.



# Elementi

- **InputElement:** Rappresenta un elemento di input in un modulo HTML. Può essere usato per creare vari tipi di campi di input, come l'input di testo, l'input di password, la casella di controllo, il pulsante di opzione, ecc. Fornisce proprietà e metodi per lavorare con gli elementi di input.
- **ButtonElement:** Rappresenta un elemento pulsante in un documento HTML. Viene utilizzato per creare pulsanti cliccabili che possono attivare azioni o eventi quando vengono cliccati. Fornisce proprietà e metodi per lavorare con gli elementi pulsante.

# Elementi

```
1  import 'dart:html';
2
   Run | Debug
3  main() {
4      //Elemento intestazione
5      HeadingElement heading = HeadingElement.h1();
6      heading.text = 'Hello, World!';
7      document.body!.append(heading);
8
9      //Elemento input
10     InputElement input = InputElement();
11     input.type = 'text';
12     document.body!.append(input);
13
14     //Elemento bottone
15     ButtonElement button = ButtonElement();
16     button.text = 'Click Me';
17     document.body!.append(button);
18 }
19
```

Hello, World!

Click Me

# Interazione

Gli utenti possono interagire con gli elementi della pagina web interagendo con essi. L'interazione della UI è basata sugli eventi.

# Interazione

Gli EventListener in dart sono delle funzioni in dart:html in grado di ascoltare queste interazioni e reagire.

```
1  import 'dart:html';
2
3  Run | Debug
4  main() {
5    //Elemento intestazione
6    HeadingElement heading = HeadingElement.h1();
7    heading.text = 'Hello, World!';
8    document.body!.append(heading);
9
10   //Elemento input
11   InputElement input = InputElement();
12   input.type = 'text';
13   document.body!.append(input);
14
15   //Elemento bottone
16   ButtonElement button = ButtonElement();
17   button.text = 'Click Me';
18   button.onClick.listen((event) {
19     // gestione dell'evento
20     print('Button clicked!');
21   });
22   document.body!.append(button);
23 }
```

# Interazione

È anche possibile modificare gli elementi del dom in relazione agli eventi. Gli eventi del browser sono vari. I principali sono :

`onClick();`

`onDrag();`

`onMouseMove();`

`onHover();`

`onSubmit();`

`onFocus();`

# Interazione

```
main.dart x
web > main.dart > main

Run | Debug
3 void main() {
4   // Elemento intestazione
5   HeadingElement heading = HeadingElement.h1();
6   heading.text = 'Hello, World!';
7   document.body!.append(heading);
8
9   // Elemento input
10  InputElement input = InputElement();
11  input.type = 'text';
12  document.body!.append(input);
13
14  // Elemento bottone
15  ButtonElement button = ButtonElement();
16  button.text = 'Click Me';
17  document.body!.append(button);
18
19  // elemento di lista ordinata
20  OListElement orderedList = OListElement();
21  document.body!.append(orderedList);
22
23  button.onClick.listen((event) {
24    // Gestione dell'evento
25    var inputValue = input.value;
26    if (inputValue.isNotEmpty) {
27      // Creare un nuovo elemento di lista
28      LIElement listItem = LIElement();
29      listItem.text = inputValue;
30
31      // Aggiunge la voce alla lista ordinata
32      orderedList.append(listItem);
33    }
34  });
35}
```

# Interazione

Le funzioni possono essere scritte direttamente in linea oppure conservate in una variabile richiamata poi nella view. Inoltre possono essere rappresentate su più righe, con la dichiarazione del tipo oppure su linea singola, utilizzando la struttura ad arrow function.

# Interazione

```
int Foo(int a, int b) {  
  return a+b;  
}
```

```
Foo(int a, int b) {  
  return a+b;  
}
```

```
Foo(a, b) {  
  return a+b;  
}
```

```
var Foo = (a, b) {  
  return a+b;  
}
```

```
(a, b) {  
  return a+b;  
}
```

```
int Foo(int a, int b) => a+b;
```

```
Foo(int a, int b) => a+b;
```

```
Foo(a, b) => a+b;
```

```
var Foo = (a, b) => a+b;
```

```
(a, b) => a+b;
```



# Interazione

```
main.dart X
web > main.dart > main
1  import 'dart:html';
2
   Run | Debug
3  void main() {
4      // Funzione per stile line-through
5      toggleLineThrough(LIElement listItem) {
6          if (listItem.style.textDecoration == 'line-through') {
7              listItem.style.textDecoration = 'none';
8          } else {
9              listItem.style.textDecoration = 'line-through';
10         }
11     }
12 }
```

# Interazione

```
12
13 // Elemento intestazione
14 HeadingElement heading = HeadingElement.h1();
15 heading.text = 'Hello, World!';
16 document.body!.append(heading);
17
18 // Elemento input
19 InputElement input = InputElement();
20 input.type = 'text';
21 document.body!.append(input);
22
23 // Elemento bottone
24 ButtonElement button = ButtonElement();
25 button.text = 'Click Me';
26 document.body!.append(button);
27
28 // elemento di lista ordinata
29 OListElement orderedList = OListElement();
30 document.body!.append(orderedList);
31
```

# Interazione

```
31
32  button.onClick.listen((event) {
33      // Gestione dell'evento
34      var inputValue = input.value;
35      if (inputValue.isNotEmpty) {
36          // Creare un nuovo elemento di lista
37          LIElement listItem = LIElement();
38          listItem.text = inputValue;
39
40          // Aggiunge la voce alla lista ordinata
41          orderedList.append(listItem);
42          // Aggiungere un gestore per il clic per stile line-through
43          listItem.onClick.listen((event) {
44              toggleLineThrough(listItem);
45          });
46          //gestore per l'eliminazione al doppio click
47          listItem.onDoubleClick.listen((event) {
48              listItem.remove();
49          });
50      }
51  });
52  }
53
```

# Stile

Gli elementi possono ricevere dello stile personalizzato creando ad esempio delle classi e gestendo le proprietà nel filr html puntando ad un foglio di stile esterno o utilizzando il tag style.

# Stile

```
main.dart index.html
web > index.html > html > head > style
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>demo_web</title>
10  <link rel="stylesheet" href="styles.css">
11  <script defer src="main.dart.js"></script>
12  <style>
13    p {
14      margin: 2%;
15      color: darkblue;
16      font-family: Arial, Helvetica, sans-serif;
17    }
18
19  </style>
20 </head>
21 <div id="app"></div>
22 <body>
23
24 </body>
25 </html>
26
```

# Stile

```
main.dart x index.html
web > main.dart > ...
1 import 'dart:html';
2
3 Run | Debug
4 void main() {
5   // Create result elements
6   ParagraphElement paragrafo = ParagraphElement();
7   paragrafo.text = "Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style";
8   document.body!.append(paragrafo);
9 }
10
```



Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style

# Stile

Oppure è possibile accedere alle proprietà style dell'attributo appena creato.

# Stile

```
main.dart x index.html
web > main.dart > main
1 import 'dart:html';
2
   Run | Debug
3 void main() {
4   // Create result elements
5   ParagraphElement paragrafo = ParagraphElement();
6   paragrafo.text = "Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style";
7   ParagraphElement paragrafo2 = ParagraphElement()
8   ..text = "Paragrafo con attributi definiti nel file dart"
9   ..style.backgroundColor = "#99ccff"
10  ..style.padding = "12px"
11  ;
12  document.body!.append(paragrafo);
13  document.body!.append(paragrafo2);
14
15 }
16
```



Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style

Paragrafo con attributi definiti nel file dart



# Stile

Come mostrato nel codice precedente, le proprietà sono additive nel caso in cui siano dichiarate in più parti dell'applicazione e se fanno riferimento a proprietà diverse. In questo caso padding e colore di sfondo si aggiungono al colore del testo.

# Stile

Nel caso in cui invece la proprietà richiamata sia già stata indicata in precedenza verrà sovrascritta.

# Stile

```
main.dart x index.html
web > main.dart > ...
1 import 'dart:html';
2
Run | Debug
3 void main() {
4   // Create result elements
5   ParagraphElement paragrafo = ParagraphElement();
6   paragrafo.text = "Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style";
7   ParagraphElement paragrafo2 = ParagraphElement()
8   ..text = "Paragrafo con attributi definiti nel file dart"
9   ..style.backgroundColor = "#99ccff"
10  ..style.padding = "12px"
11  ..style.color = "purple"
12  ;
13  document.body!.append(paragrafo);
14  document.body!.append(paragrafo2);
15
16 }
17
```



Esempio di paragrafo stilizzato con il selettore di elemento nel con uso del tag style

Paragrafo con attributi definiti nel file dart

# Attività

Si crei una pagina calcolo del perimetro e dell'area del quadrato come la seguente utilizzando sia stile dichiarato nel file dart che nel file css/html. Al click su ciascun bottone apparirà un paragrafo valorizzato con il calcolo eseguito in relazione al valore inserito nel campo di input.

## Calcola perimetro e area del quadrato!

Calcola Area

Calcola Perimetro

Area: 4

Perimetro: 8

The background features a series of overlapping, wavy bands in various shades of blue and purple. The top section has darker, more saturated colors, while the middle section is a solid white space where the text is located. The bottom section returns to the blue and purple waves, with some lighter, more translucent bands. The overall effect is a sense of fluid motion and depth.

# Domande e approfondimenti