



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, purple, and pink. These bands flow horizontally across the frame, creating a sense of movement and depth. The colors transition from deep blues and purples on the left to lighter, more ethereal tones on the right.

Dart Server Side

Docente



Claudia Infante



claudia.infante@bcsoft.net

Sommario

- Uso di Dart server side
- Dart HTTP Server

Uso di dart Server side

La virtual machine di dart è ospitata in due ambienti : client (che utilizza dart:html per accedere agli elementi del dom) e dart:io utilizzato per accedere al sistema operativo come file e network sockets.

Uso di dart Server side

L'applicazione lato server Dart File Browser viene eseguita come uno script Dart, sia dalla linea di comando che dall'editor Dart, e può emettere del testo nella console utilizzando la funzione `print("")`.

Uso di dart Server side

Quando si usa l'editor Dart per creare un nuovo progetto, si può scegliere se creare un'applicazione web o un'applicazione a riga di comando.

Uso di dart Server side

La grande differenza tra script lato client e script lato server è l'ambiente di hosting. Gli script lato client vengono eseguiti in una macchina virtuale ospitata da un browser web e hanno accesso al DOM del browser.

Uso di dart Server side

Questo permette loro di importare la libreria `dart:html`. Gli script lato server non esistono in un browser web e si otterrà un errore se si tenta di utilizzare la libreria `dart:html` lato server.

Uso di dart Server side

Gli script lato server hanno accesso alla libreria `dart:io`, che fornisce classi e funzioni per l'accesso al filesystem, il servizio di HTTP e web socket e comunicare attraverso la rete con i socket di rete.

Uso di dart Server side

Per creare un progetto dart server side lanciare dal cmd il comando `dart create -t console-full <project_name>`.

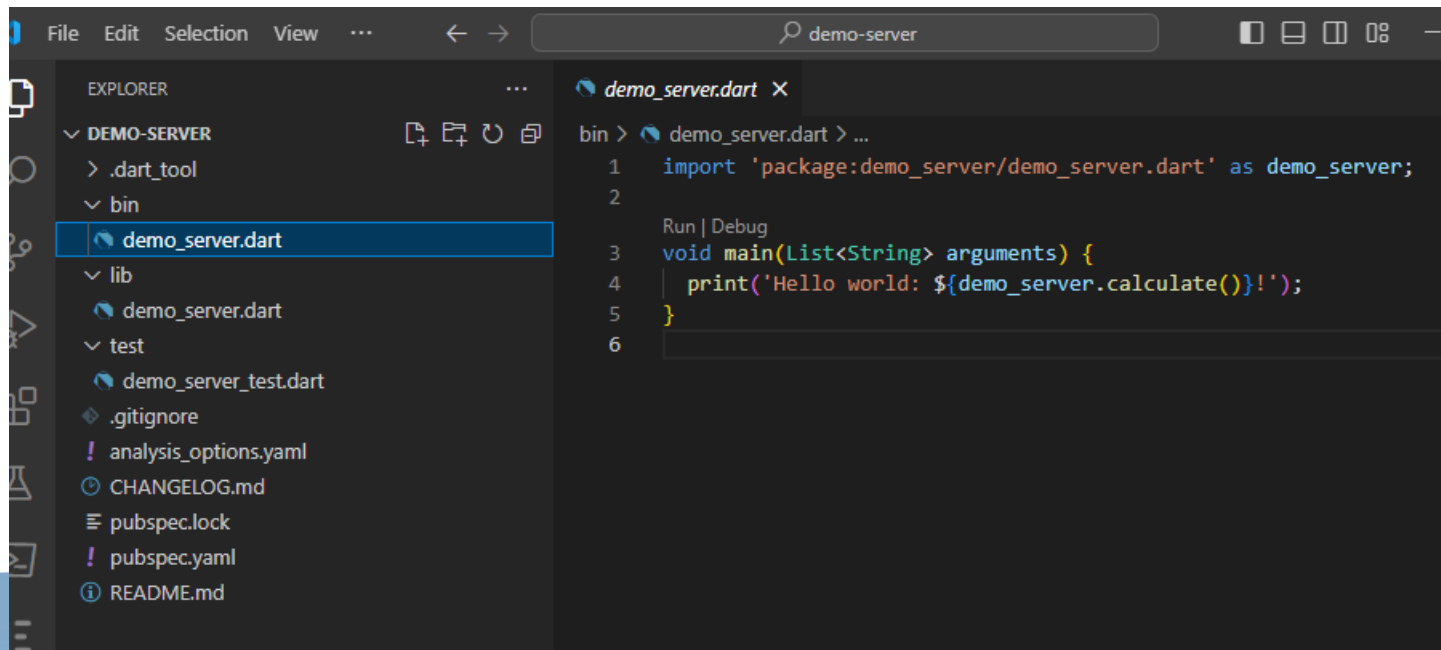
Una volta creato il progetto, posizionarsi nella folder ed eseguire il comando `dart run`.

Uso di dart Server side

Il progetto non prevederà più la cartella web e avrà solo file con estensione.dart.

Uso di dart Server side

Il file presente nella cartella bin si occuperà di lanciare la funzione principale che potrà invocare altre funzioni dell'applicazione:



The screenshot shows an IDE window with a dark theme. The Explorer panel on the left displays the project structure for 'DEMO-SERVER'. The 'bin' directory is expanded, and 'demo_server.dart' is selected. The main editor area shows the code for 'demo_server.dart'.

```
bin > demo_server.dart > ...
1  import 'package:demo_server/demo_server.dart' as demo_server;
2
3  Run | Debug
4  void main(List<String> arguments) {
5    | print('Hello world: ${demo_server.calculate()}!');
6    | }
7
```

Uso di dart Server side

È possibile eseguire una serie di script, ad esempio per la lettura di file presenti in locale, grazie alla libreria dart:io

```
demo_server.dart X
bin > demo_server.dart > ...
1  import 'dart:io';
2
3
4  Run | Debug
5  void main() {
6      File file = File('C:/Users/Claudia/Desktop/Flutter/Dart/Note.txt');
7
8      file.readAsString().then((String contents) {
9          print(contents);
10         }).catchError((e) {
11             print('Error reading file: $e');
12         });
13 }
```

Dart HTTP Server

Grazie al pacchetto shelf è possibile creare e comporre server web in dart.

Dart HTTP Server

Per creare un server in dart utilizzando il pacchetto shelf, andrà installato come dipendenza nel file pubspec.yaml.

```
10 dependencies:  
11 |   shelf: ^1.0.0  
12
```


Dart HTTP Server

Importare anche shelf_router che consentirà di definire la rotta di accesso per la gestione della request

```
dependencies:  
  shelf: ^1.0.0  
  shelf_router: ^1.0.0
```

Dart HTTP Server

Successivamente creare il file Dart per la gestione delle richieste HTTP.

Dart HTTP Server

Nel file principale si importano le dipendenze necessarie

```
demo_server.dart X
bin > demo_server.dart > _router
1  import 'package:shelf/shelf.dart';
2  import 'package:shelf/shelf_io.dart' as shelf_io;
3  import 'package:shelf_router/shelf_router.dart';
4
5  Response _helloHandler(Request request) {
6    |  return Response.ok('Hello, World!');
7  }
8
```

Dart HTTP Server

Si definisce il metodo per la gestione della rotta. Si vuole mostrare il messaggio nel path principale

```
8
9  Router _router() {
10    final router = Router();
11
12    router.get('/', (Request request) {
13      return _helloHandler(request);
14    });
15
16    return router;
17  }
18
```

Dart HTTP Server

Con una gestione asincrona si definisce una serie di middleware e un gestore per processare le richieste http.

```
19 void main() async {  
20   final handler = const Pipeline()  
21     .addMiddleware(logRequests())  
22     .addHandler(_router());  
23  
24   final server = await shelf_io.serve(handler, 'localhost', 8080);  
25   print('Server running on localhost:${server.port}');  
26 }  
27
```

Dart HTTP Server

`addMiddleware(logRequests())`: Aggiunge middleware alla pipeline. In questo caso, `logRequests()` è una funzione integrata che registra le informazioni sulle richieste in arrivo.

```
19 void main() async {  
20   final handler = const Pipeline()  
21     .addMiddleware(logRequests())  
22     .addHandler(_router());  
23  
24   final server = await shelf_io.serve(handler, 'localhost', 8080);  
25   print('Server running on localhost:${server.port}');  
26 }  
27
```

Dart HTTP Server

`addHandler(_router().handler)`: Aggiunge il gestore del router alla pipeline. La proprietà `handler` dell'istanza del router è usata per fornire la logica principale di gestione delle richieste per il server.

```
19 void main() async {  
20   final handler = const Pipeline()  
21     .addMiddleware(logRequests())  
22     .addHandler(_router());  
23  
24   final server = await shelf_io.serve(handler, 'localhost', 8080);  
25   print('Server running on localhost:${server.port}');  
26 }  
27
```

Dart HTTP Server

`shelf_io.serve(handler, 'localhost', 8080)`: Avvia un server HTTP su localhost alla porta 8080 utilizzando il gestore fornito (che include il gestore del middleware e del router).

```
19  void main() async {  
20      final handler = const Pipeline()  
21          .addMiddleware(logRequests())  
22          .addHandler(_router());  
23  
24      final server = await shelf_io.serve(handler, 'localhost', 8080);  
25      print('Server running on localhost:${server.port}');  
26  }  
27
```


Dart HTTP Server

`print('Server running on localhost:${server.port}')` : Invia alla console un messaggio che indica che il server è in esecuzione e specifica la porta su cui è in ascolto.

```
19 void main() async {  
20   final handler = const Pipeline()  
21     .addMiddleware(logRequests())  
22     .addHandler(_router());  
23  
24   final server = await shelf_io.serve(handler, 'localhost', 8080);  
25   print('Server running on localhost:${server.port}');  
26 }  
27
```

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 light, airy blues to deep, rich purples, with some areas showing a gradient effect. The overall composition is modern and artistic, typical of a professional presentation design.

Domande e approfondimenti