Fluttercon

JS

Enabling smooth communication between JS and Dart

Dart

Aseem Wangoo

@aseemwangoo
Flatteredwithflutter.com
Engineer @ShopBack

Willkommen

Ich bin Aseem Wangoo

Founder of FlatteredWithFlutter (Flutter blog)

Engineer at ShopBack

Blogger and TechTuber

Tech Writer **@FlutterCommunity**

Agenda

Introduction - Dart and Flutter

Dart Compiler - Overview

Dart Native & Web

JS interoperability in Dart

Types of interops - Static and Dynamic

JS<-> Dart Interop Usecases

Dart

client-optimized language for
fast apps on any platform

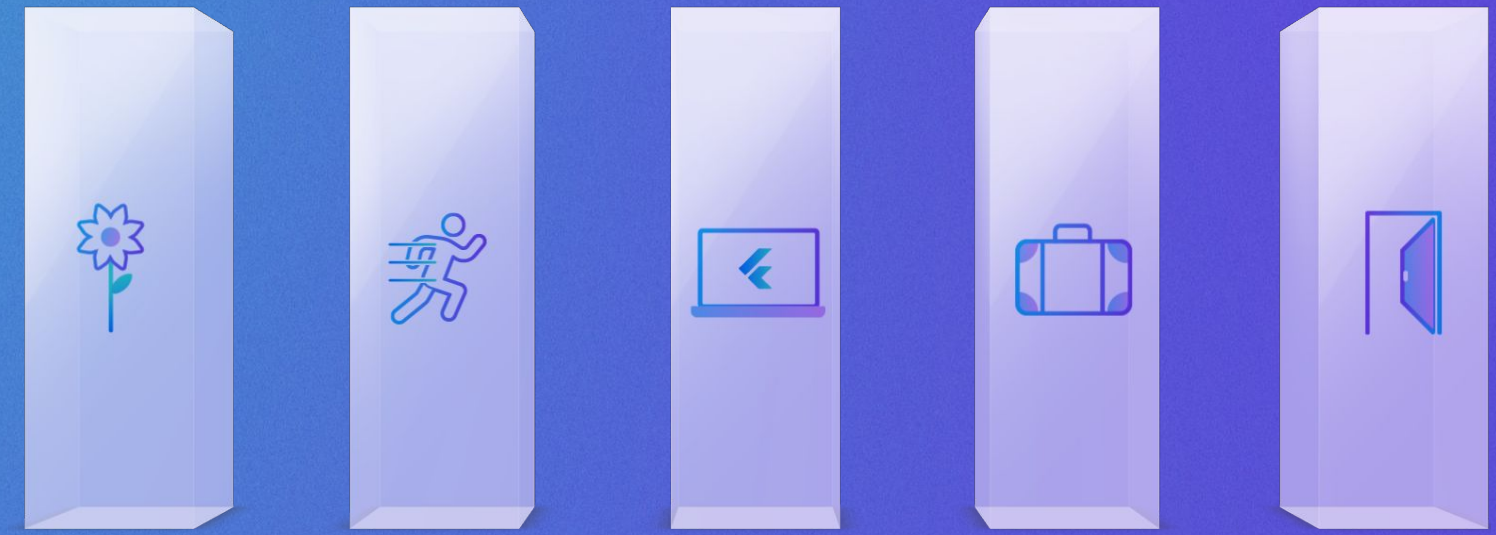
```
void main() => print('Hello, world!');
```

Flutter

beautiful, fast, productive and open UI toolkit for any screen

flutter.dev

Flutter aims for



Beautiful Fast Productive Portable Open

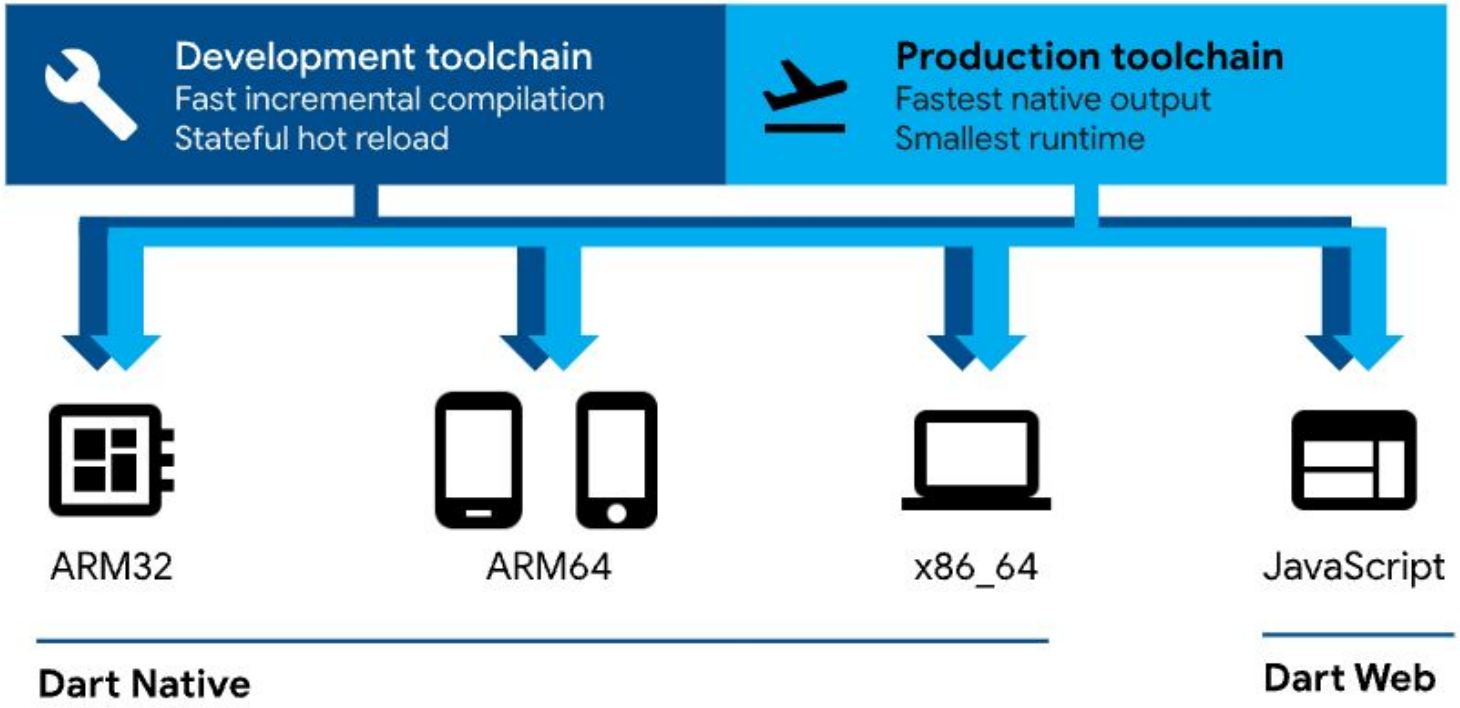
Setting Expectations... 🧐

Dart Compiler - Overview

Native Platform (Apps for mobile and desktop devices)

Web Platform (Apps targeting web)

Dart Compiler *(ctnd)*

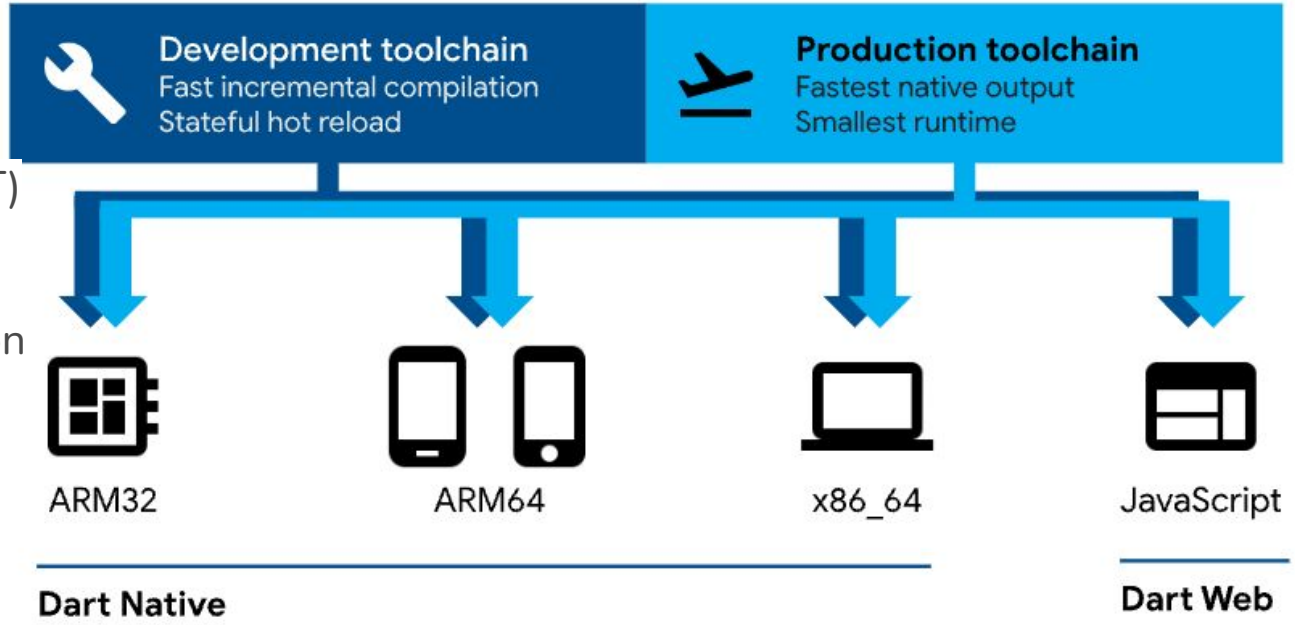


Dart Native - for development

Just-in-time compiler (JIT)

Incremental recompilation

Hot reload

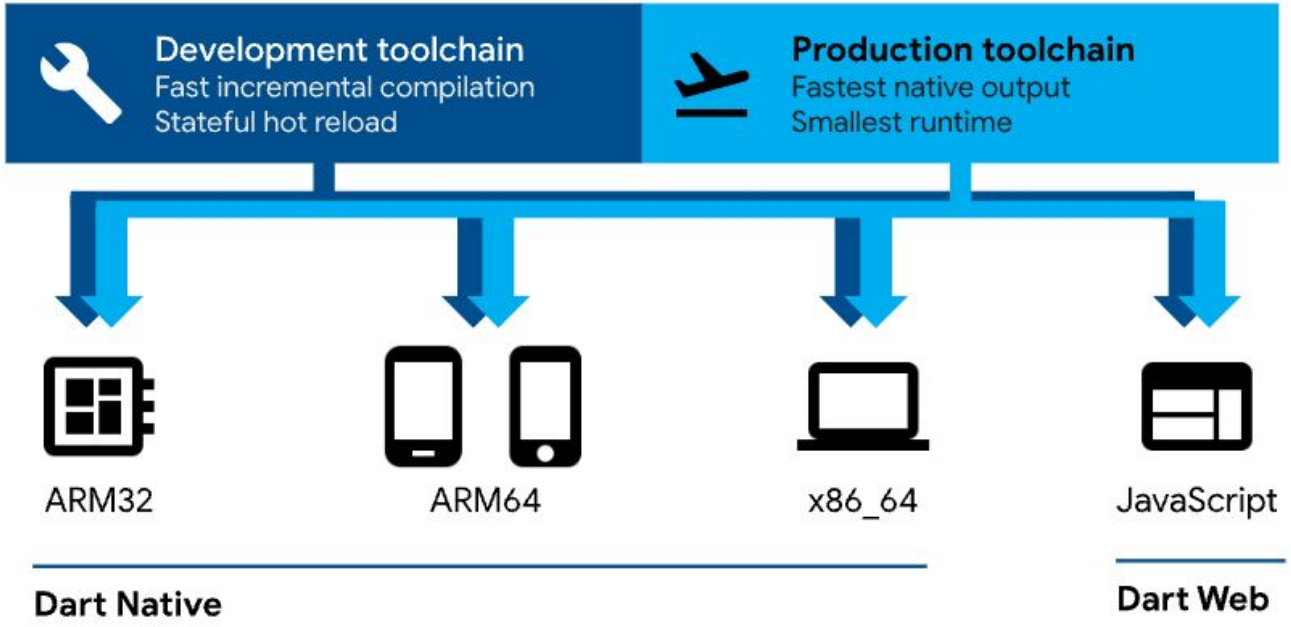


Dart Native - for production

- Ahead-of-time (AOT) compiler

- Consistent launch

- Short startup time

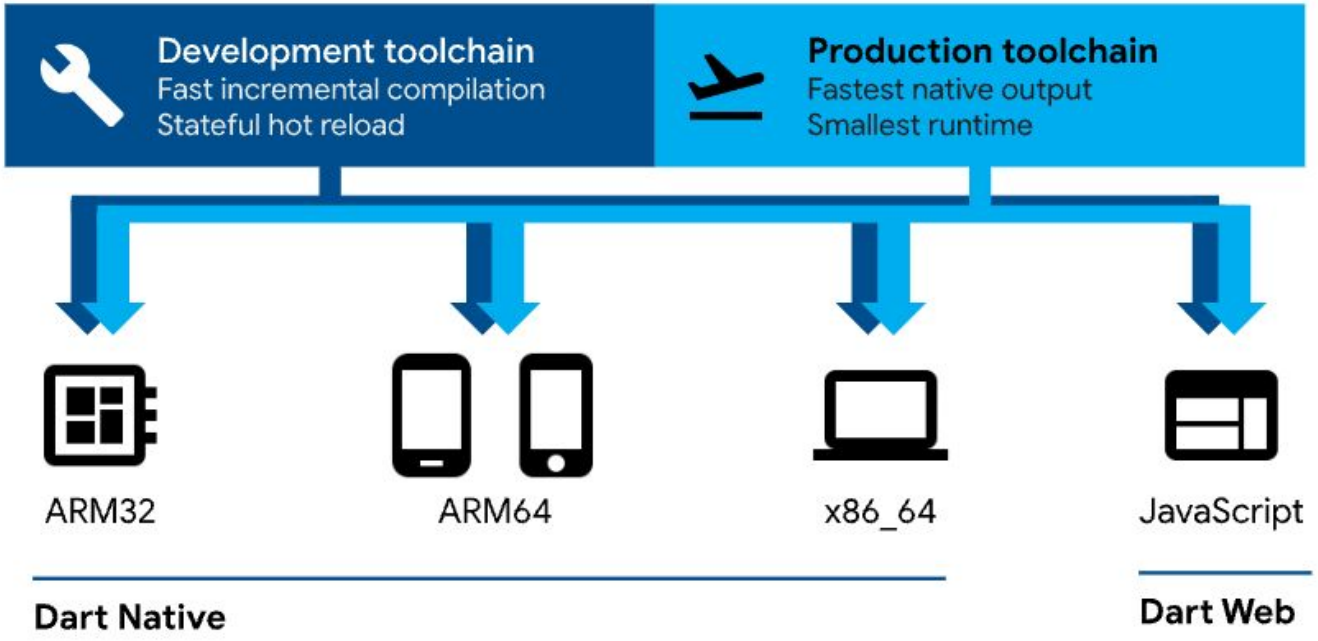


Let’s talk about web….

Dart Web - development and production

Incremental development compiler **(for development)**

Optimized production Compiler **(for production)**



Basics Covered
Let the talk begin

JavaScript interoperability in Dart

Dart Web enables running Dart code on web platforms powered by JS

Dart web supports calling JS using the **js package**

Packages using **js package**

sqlite3 2.0.0

Sass

About package:js

Allows to call JavaScript APIs from Dart

or vice versa.

js 0.6.7

Published 4 months ago • dart.dev Dart 3 compatible

SDK DART FLUTTER PLATFORM WEB

Provides annotations and functions specifying how Dart interoperates with JS

Support 2 interops (**Static & Dynamic interop)**

JS <-> Dart Interop constraints

Dart SDK constraint: >= 2.19

package:js constraint: >= 0.6.6

js - interops

Static Interop	Dynamic Interop
Next generation of JS interop	Dart team supported till now
Dart team recommends this	Dart team moving away from this
Performance benefits	Not much performance benefits

js - Static interop - Overview

Involves a predefined and static interface or contract between **JS and Dart**

Static interop specifies the types, methods, and properties for access

from **Dart and JS**

Interoperability established at compile-time, ensuring type safety

js - Dynamic interop - Overview

Involves a flexible and dynamic interaction between **JS and Dart**

Interoperability established at run-time. More flexibility in accessing objects from **Dart and JS**

Greater adaptability **but** tradeoff with type safety and potential runtime errors

Element embedding

Effects

Shadow

Mirror

Resize

Spin

Device

JS Interop

Screen

Counter

Value

0

Increment

Counter

You have pushed the button this many times:

0

+

Why static interop? 🧐

Idiomaticity

Follows conventions, patterns, and style of the language in a natural and concise manner

More readable and maintainable code

Performance

Efficient code in terms of speed, execution and resource usage

Why static interop? 🧐 *(ctnd)*

Type Soundness

Ensures type rules adheres throughout the execution of program

Type checking at compile-time

WASM compatibility

Dart team is going all-in on Web Assembly

Dive deeper into Static interop....

Static interop in package:js

`@staticInterop` annotation introduced in **0.6.6 onwards**

Classes with this annotation **don't support dynamic invocations**

Makes backends consistent -> development and production web compilers won't behave differently

⚠️ **Static interop experimental**

Calling Dart Function from JS

Import js package

Dart setter

Dart code associated with JS

Wrapping Dart function and sending to JS

```dart
@JS()
library callable_function;

import 'package:js/js.dart';

/// Allows assigning a function to be callable from `window.functionName()`
@JS('functionName')
external set _functionName(void Function() f);

/// Allows calling the assigned function from Dart as well.
@JS()
external void functionName();

void _someDartFunction() {
  print('Hello from Dart!');
}

void main() {
  _functionName = allowInterop(_someDartFunction);
  // JavaScript code may now call `functionName()` or `window.functionName()`.
}
```

Observations for static interop ✍️

Too **focused** on importing **JS code to Dart**, not the other way around

allowInterop becomes cumbersome for Dart Objects

Need to put **allowInterop** on all members manually

But, can we do better?

Improvements - createDartExport

Fluttercon

@JSExport makes Dart class exportable

JS acts as a proxy to exported Dart object.

createDartExport gets transformed to JS object

```
// The Dart class must have `@JSExport` on it or one of its instance members.
@JSExport()
class Counter {
  int value = 0;
  @JSExport('increment')
  void renamedIncrement() {
    value++;
  }
}

@JS()
@staticInterop
class JSCounter {}

extension on JSCounter {
  external int value;
  external void increment();
}

void main() {
  var dartCounter = Counter();
  var counter = createDartExport<Counter>(dartCounter) as JSCounter;
}
```

createDartExport - Summary

Annotation **@JSExport** makes Dart class exportable

createDartExport call is transformed into a JS object literal

Use the same names and syntax to access the members.

Usecases...

1️⃣ Web Animations ✅

2️⃣ Counter App + Dynamic Web Nav

3️⃣ ??? (Any Guesses)

Integrate ChatGPT in Flutter 😎

Integrate ChatGPT in Flutter

3 steps

1. Creating ChatGPT server locally

2. Creating ChatGPT UI

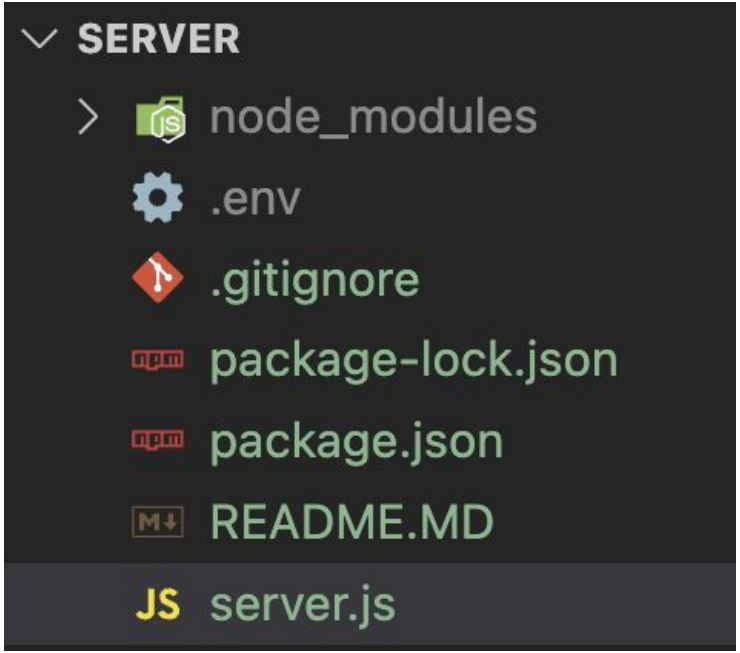
3. Send queries/prompts from **Dart to JS (ChatGPT)**

1.Create ChatGPT server

Install dependencies (**@openai/api express etc**)

Create **server.js** file and obtain API Key from **OpenAI**

Use **OpenAPI configuration object** for sending prompts



1.Create ChatGPT server *(cntd)*

Generate text completions

model: **text-davinci-003**

prompt: **text prompt** to generate com

max_tokens: **number of tokens**

model generates

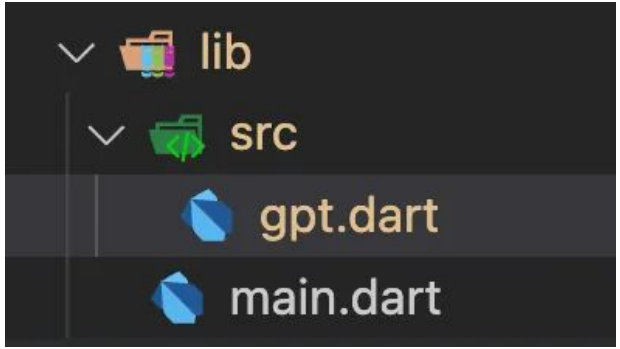
```
const response = await openai.createCompletion(
  {
    model: "text-davinci-003",
    prompt: `${prompt}`, // pass in the user query
    max_tokens: 3000,
  },
);

res.status(200).send({
    bot: response.data.choices[0].text,
});
```

Send JSON inside a property **“bot”**

2.Create ChatGPT UI

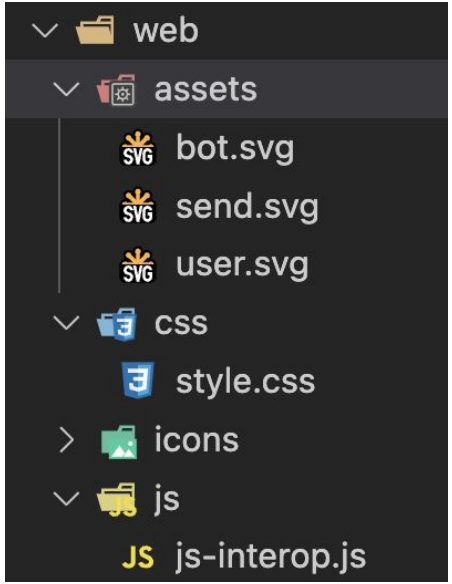
Create Flutter Web Project

2 dart files: **gpt.dart** and **main.dart**



Add **style** in css folder and **assets** inside assets folder

Create **js-interop.js** file



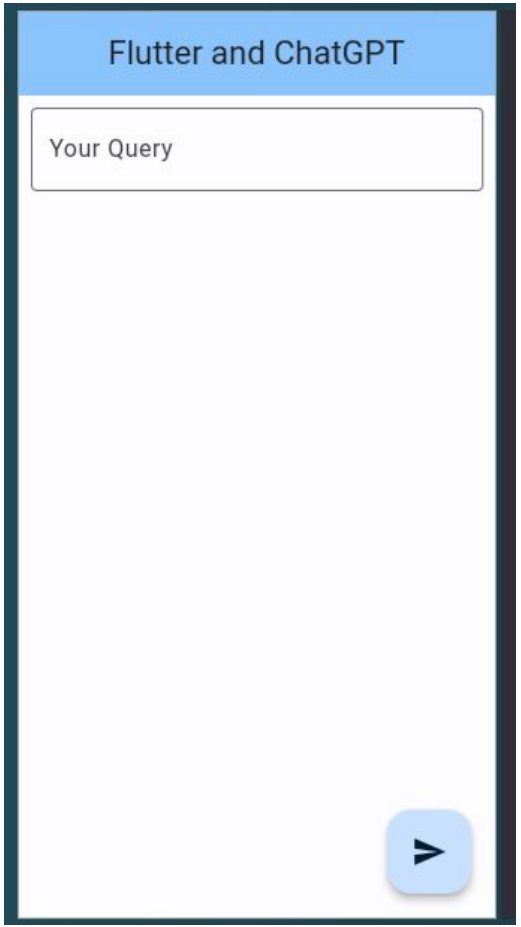
2.Create ChatGPT UI *(cntd)*

Include a **textfield** in the flutter side

Use a text controller for managing text

Customize web app initialization

Show the flutter app and ChatGPT UI side by side



Flutter App

Chat GPT UI

Customize Web App Initialization

Using the **_flutter.loader** JS API provided by **flutter.js**

Initialization process has 3 **steps**

- Fetching **main.dart.js** script and initialize the service worker
- Initialize Flutter web engine
- Run the Flutter app

Note: Flutter 2.10 or earlier, doesn't support customization

Default Web App *(cntd)*

Fluttercon

load **flutter.js**

loadEntrypoint calls **onEntrypointLoaded** callback

Engine initializer sets run-time configuration

Start Flutter Web engine

```
<html>
  <head>
    <!-- ... -->
    <script src="flutter.js" defer></script>
  </head>
  <body>
    <script>
      window.addEventListener('load', function (ev) {
        // Download main.dart.js
        _flutter.loader.loadEntrypoint({
          serviceWorker: {
            serviceWorkerVersion: serviceWorkerVersion,
          },
          onEntrypointLoaded: async function(engineInitializer) {
            // Initialize the Flutter engine
            let appRunner = await engineInitializer.initializeEngine();
            // Run the app
            await appRunner.runApp();
          }
        });
      });
    </script>
  </body>
</html>
```

Customized Web App

Flutter con

Access flutter html element

HTML Element into which Flutter renders app

Start Flutter Web engine

```
<script src="js/js-interop.js" defer></script>
<script>
    window.addEventListener('load', function(ev) {
      let target = document.querySelector("#flutter_target");

      // Download main.dart.js
      _flutter.loader.loadEntrypoint({
        onEntrypointLoaded: async function (engineInitializer) {
          let appRunner = await engineInitializer.initializeEngine({
            hostElement: target,
          });
          await appRunner.runApp();
        },
      });
    });
</script>
```

2.ChatGPT UI <-> Flutter side *(cntd)*

Annotate app state with **@js.JSExport()**

```
@js.JSExport()
You, 4 months ago | 1 author (You)
class _MyHomePageState extends State<MyHomePage> {…
```

Call **createDartExport** inside **initState**

```
@override
void initState() {
 super.initState();
 final export = js_util.createDartExport(this);

 // These two are used inside the [js/js-interop.js]
 js_util.setProperty(js_util.globalThis, '_appState', export);
 js_util.callMethod<void>(js_util.globalThis, '_stateSet', []);
}
```

Import **js** and **js_util** package

2.ChatGPT UI <-> Flutter side *(cntd)*

setProperty() method by js_util

_appState: export object created in the previous step.

```
@override
void initState() {
 super.initState();
 final export = js_util.createDartExport(this);

 // These two are used inside the [js/js-interop.js]
 js_util.setProperty(js_util.globalThis, '_appState', export);
 js_util.callMethod<void>(js_util.globalThis, '_stateSet', []);
}
```

stateSet: JS function defined in **js-interop.js** file

2.ChatGPT UI <-> HTML side *(cntd)*

```html
<section class="contents">
   <div id="parent">

   <!--This is the div which contains the flutter app-->

   <div id="flutter_target" class="center"></div>

   <!--This is the div which contains the ChatGPT UI-->

   <div id="app">
     <div id="chat_container"></div>

      <form id="form_container" style="display: none;">
        <textarea name="prompt" rows="1" cols="1" ></textarea>
        <button type="submit" id="submit"><img src="assets/send.svg"/>
      </form>
    </div>
  </div>
</section>
```

ChatGPT UI elements in html file

2.ChatGPT UI <-> JS side *(cntd)*

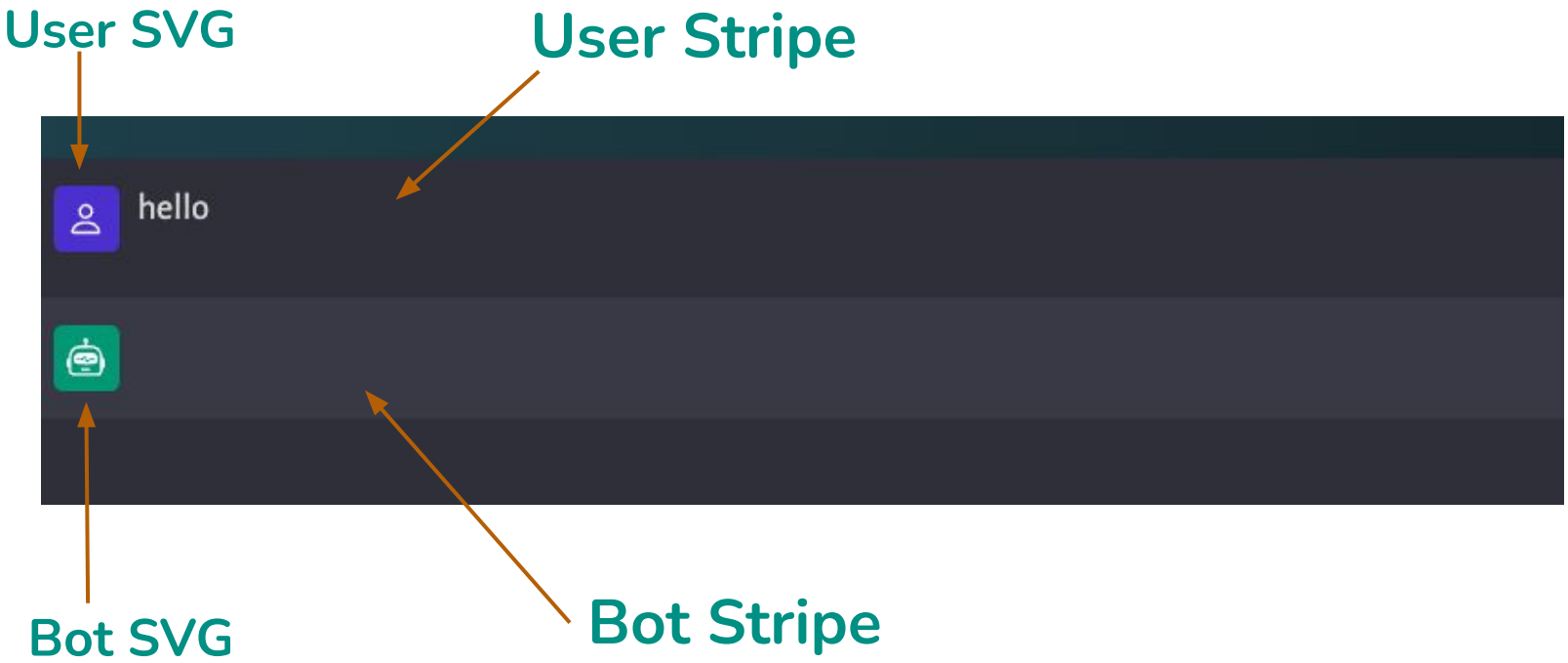
Access **HTML** elements from **JS** file

```
const form = document.querySelector('form')
const chatContainer = document.querySelector('#chat_container')
const formData = new FormData(form)
```

Create dynamic chat stripes

- For user prompt
- For bot answer

```
function chatStripe(isAi, value, uniqueId) {
  return (
    `
      <div class="wrapper ${isAi && 'ai'}">
        <div class="chat">
          <div class="profile">
            <img
              src=${isAi ? './assets/bot.svg' : './assets/user.svg'}
            />
          </div>
         <div class="message" id=${uniqueId}>${value}</div>
        </div>
      </div>
    `
  )
}
```


User SVG
User Stripe
hello
Bot SVG
Bot Stripe

Flutter and ChatGPT

Your Query

Final Step

3. Queries from Dart to JS

Annotate text controller functions

- For interoperability

```
@js.JSExport()
void textInputCallback(String value) {
  textFocusNode.requestFocus();
  setState(() {
    _textQuery = value;
    _streamController.add(null);
  );
}
```

Stream controller for receiving values

```
final _streamController = StreamController<void>.broadcast();

@js.JSExport()
void addHandler(void Function() handler) {
 // This registers the handler we wrote in [js/js-interop.js]
 _streamController.stream.listen((event) {
    handler();
 });
}
```

3. Queries from Dart to JS *(cntd)*

Flutter con

Use **IIFE(Immediately Invoked Function Expression)**

to create a function

Access **_appState** in JS

JS

```
(function () {
  "use strict";
  window._stateSet = function () {
    // IIFE
    window._stateSet = function () {

    };

    // State of flutter app in [src/gpt.dart]
    let appState = window._appState;
  };
}());
```

_appState value comes from Flutter

Dart

```
@override
void initState() {
 super.initState();
 final export = js_util.createDartExport(this);

 // These two are used inside the [js/js-interop.js]
 js_util.setProperty(js_util.globalThis, '_appState', export);
 js_util.callMethod<void>(js_util.globalThis, '_stateSet', []);
}
```

3. Flutter side *(cntd)*

Create **textQuery** function in flutter

Annotate with **@JSExport** property

```
@js.JSExport()
String get textQuery => _textQuery;
```

Used inside text controller

```
@js.JSExport()
void textInputCallback(String value) {
  textFocusNode.requestFocus();
  setState(() {
    _textQuery = value;
    _streamController.add(null);
  );
}
```

3. JS Side *(cntd)*

Send user query to **JS form**

```
let updateTextState = function () {
  formData.set('prompt', appState.textQuery);
  handleSubmit.call(form)
};

// Register a callback to update the text field
// from Flutter.
appState.addHandler(updateTextState);
```

Registered in **addHandler** callback

3. JS Side *(cntd)*

Attach event listeners

- Submit button

- Keyup button

(when user releases key)

```
// CHAT GPT FUNCTIONS
form.addEventListener("submit", (e) => {
  handleSubmit(e)
});

// CHAT GPT FUNCTIONS
form.addEventListener("keyup", (e) => {
  if (e.keyCode === 13) {
    handleSubmit(e)
  }
});
```

3. JS Side *(cntd)*

Include the localhost server

Send the prompt

Get the response

```
const response = await fetch('http://localhost:5001/', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      prompt: formData.get('prompt')
    })
})

const data = await response.json();
const parsedData = data.bot.trim()

// show parsedData
```

Flutter and ChatGPT

Your Query

Limitations

Need a class using **@JSExport**

Different members can't have same export name

```
@JSExport()
class DartClass {
  int member = 0;

  @JSExport('member')
  void method() {}
}

// Two incompatible members have the same export name.
```

Danke 🙇

And take swags!!