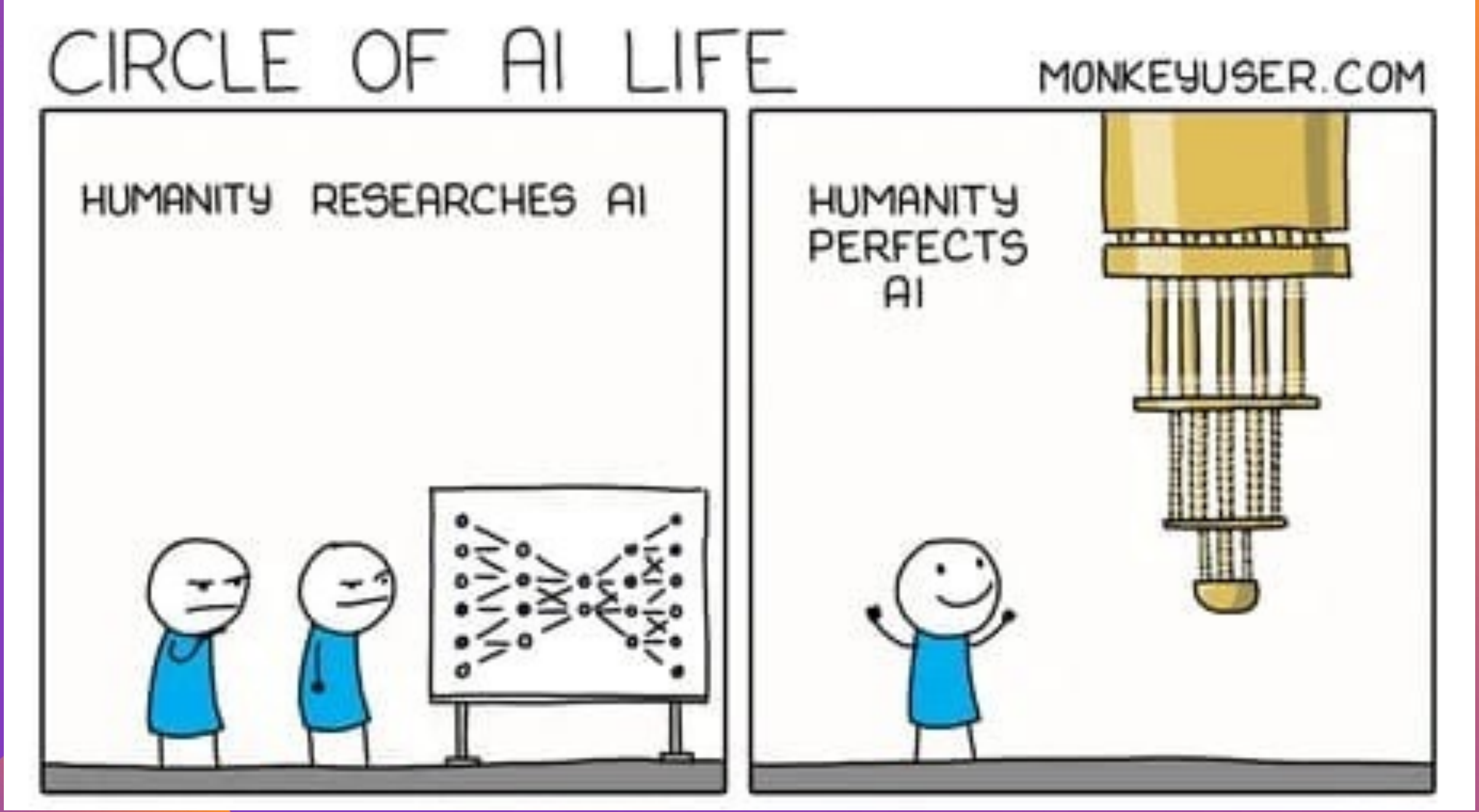
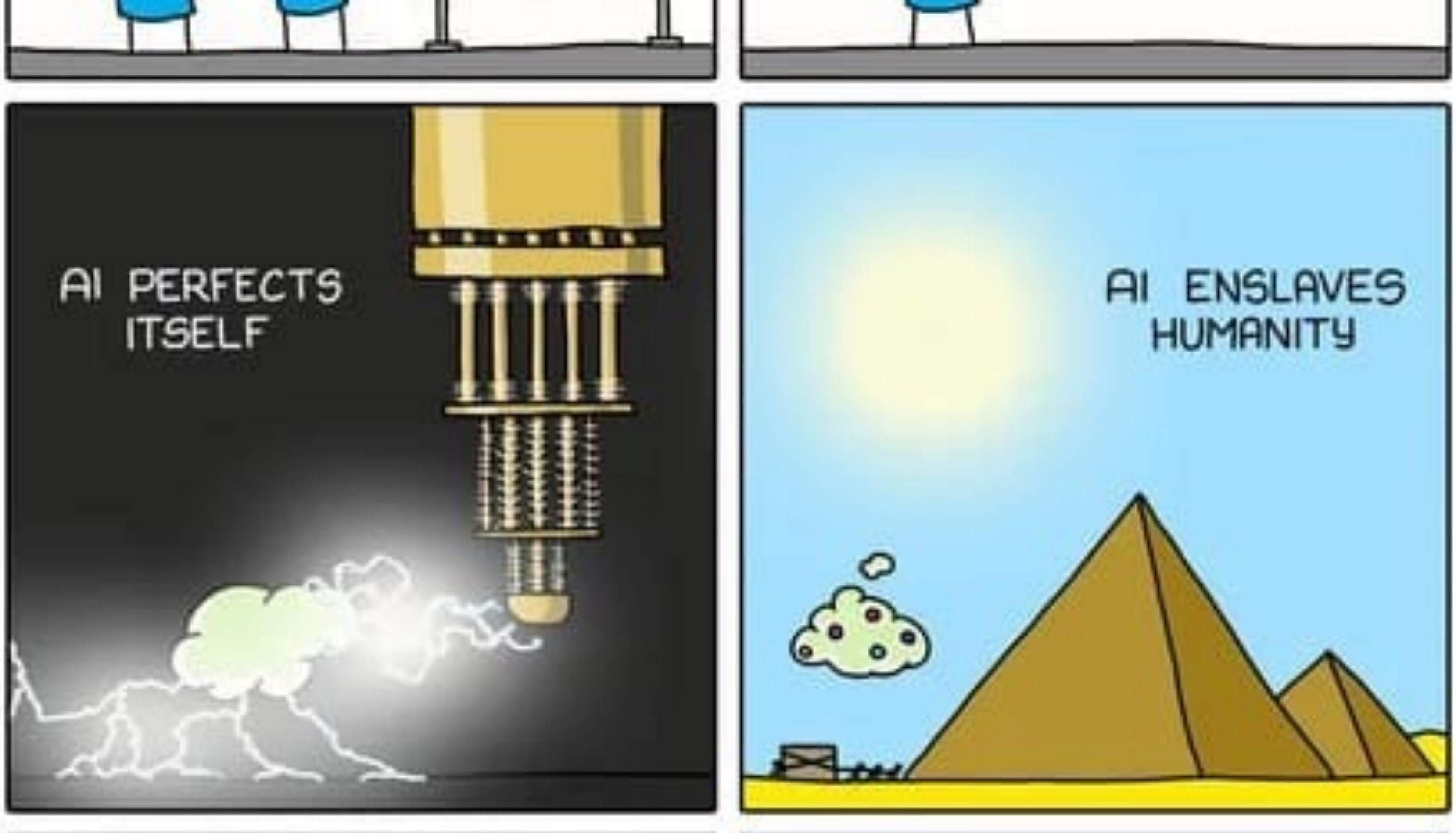
ISTA 2023

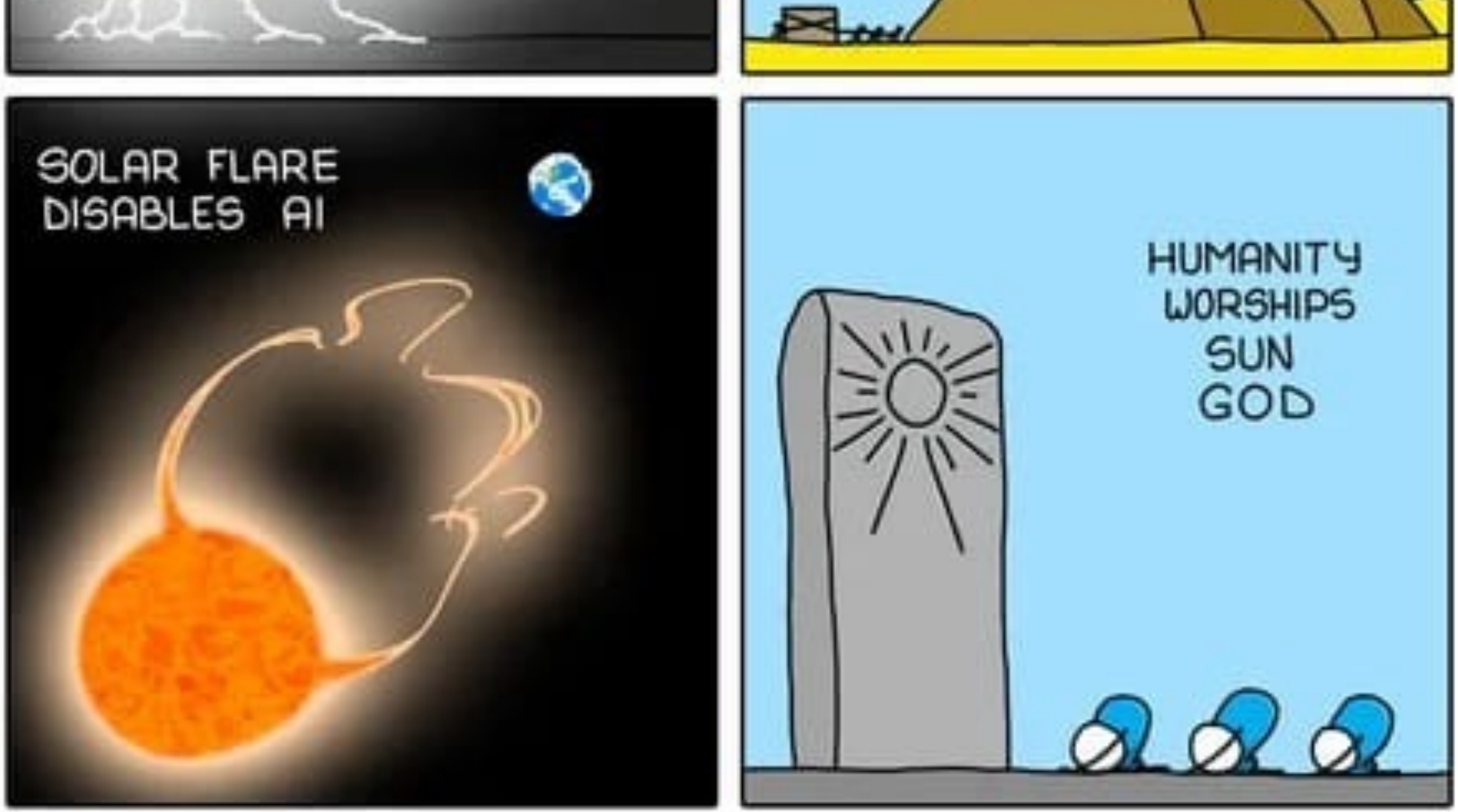
THINK . PREDICT . CREATE

ChatGPT Experiment: Building with technology we don't know

Doncho Angelov | Director Enterprise Services


CIRCLE OF AI LIFE
MONKEYUSER.COM
HUMANITY RESEARCHES AI
HUMANITY PERFECTS AI


AI PERFECTS ITSELF
AI ENSLAVES HUMANITY


SOLAR FLARE DISABLES AI
HUMANITY WORSHIPS SUN GOD

So, AI?

The Project

- DART Backend
- Flutter Frontend
- SQLite DB
- Two Containers

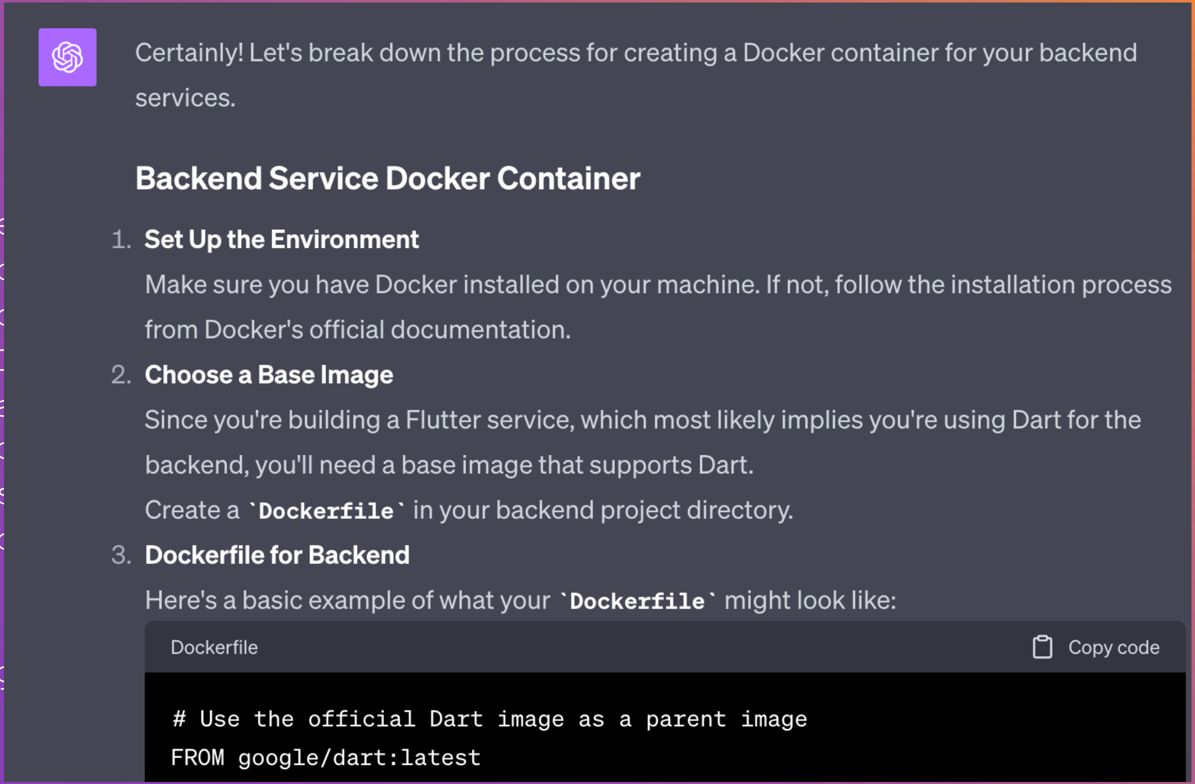
The Backend

Let’s start with some introduction!

Prompt

Dear ChatGPT, I want to creat
which will be separated in tw
containers: one serving my fr
serving my backend services. T
services container will have a
that it built with Flutter an
lightweight relational databas
container. The frontend servi
will serve a Flutter frontend

Let's start with you providin
instructions, needed to build
container.



Certainly! Let's break down the process for creating a Docker container for your backend services.

Backend Service Docker Container

1. **Set Up the Environment**
   Make sure you have Docker installed on your machine. If not, follow the installation process from Docker's official documentation.
2. **Choose a Base Image**
   Since you're building a Flutter service, which most likely implies you're using Dart for the backend, you'll need a base image that supports Dart.
   Create a `Dockerfile` in your backend project directory.
3. **Dockerfile for Backend**
   Here's a basic example of what your `Dockerfile` might look like:

```
# Use the official Dart image as a parent image
FROM google/dart:latest
```

It gave me a lot more…

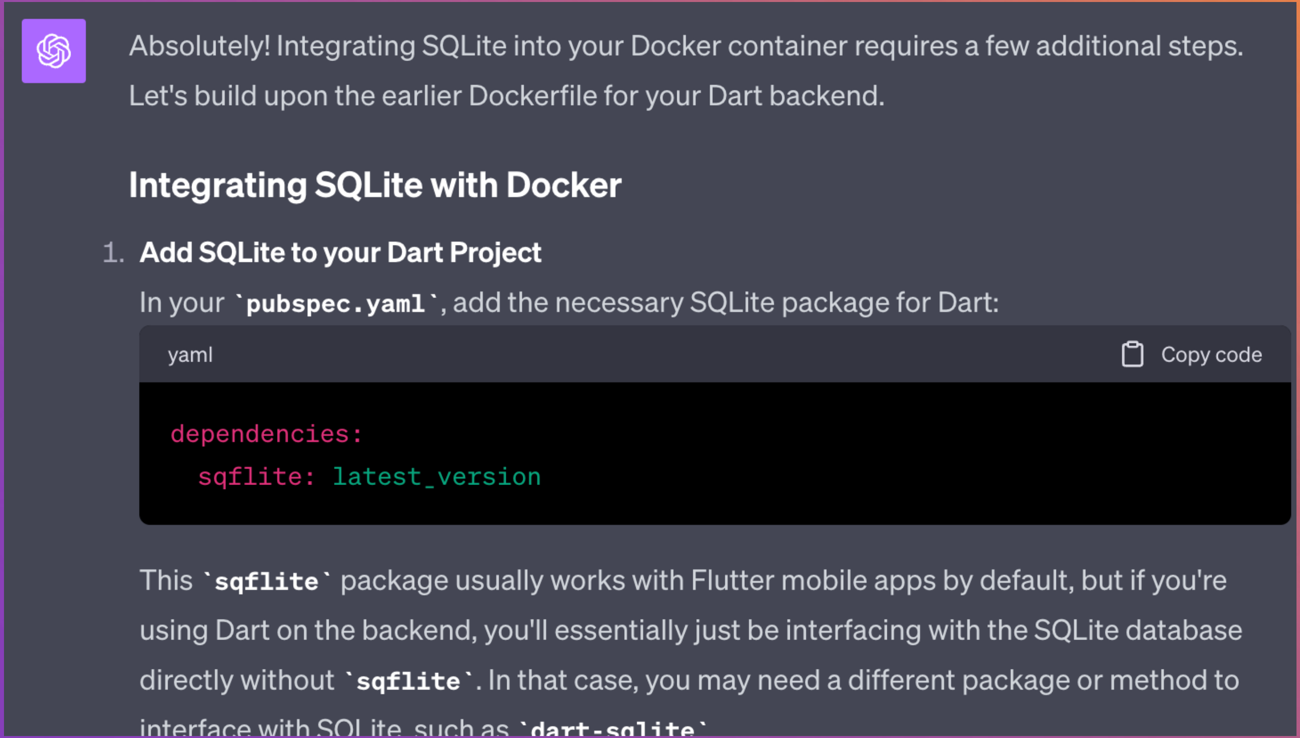
(and you will also get it as GitHub repo handover)

- Database suggestion;
- Instructions how to build the Dockerfile image and run it;
- Plenty of notes about what's the good design and why my suggestion deviates

However, I didn’t know how to include SQLite

Prompt:

`Please give me more detailed instructions about how to include SQLite in my Dockerfile`



Absolutely! Integrating SQLite into your Docker container requires a few additional steps. Let's build upon the earlier Dockerfile for your Dart backend.

Integrating SQLite with Docker

1. **Add SQLite to your Dart Project**

   In your `pubspec.yaml`, add the necessary SQLite package for Dart:

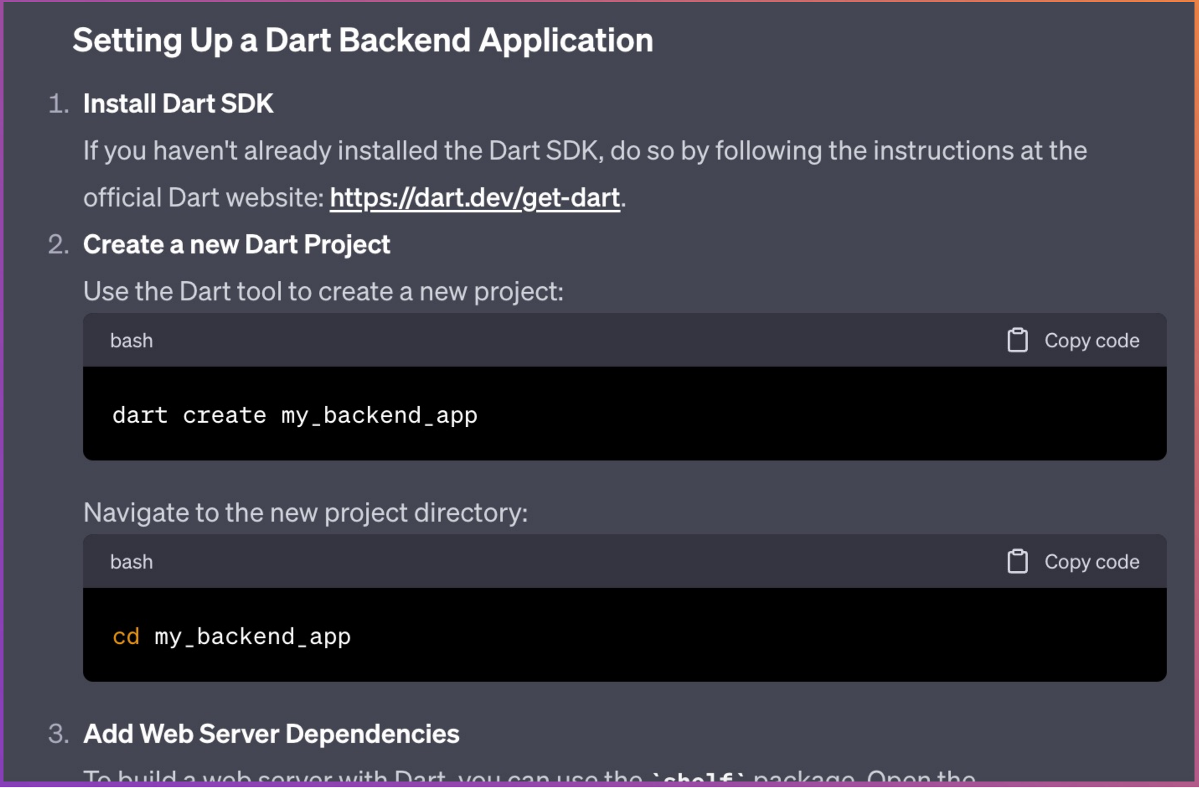
   ```yaml
   dependencies:
     sqflite: latest_version
   ```

   This `sqflite` package usually works with Flutter mobile apps by default, but if you're using Dart on the backend, you'll essentially just be interfacing with the SQLite database directly without `sqflite`. In that case, you may need a different package or method to interface with SQLite, such as `dart-sqlite`

..then I got some errors, because of missing my backend!

Prompt:

`Trying to build the Dockerfile gave me errors because of missing pubscpec.yaml. Give me the initial commands for building the backend app in Dart.`



Setting Up a Dart Backend Application

1. **Install Dart SDK**

   If you haven't already installed the Dart SDK, do so by following the instructions at the official Dart website: https://dart.dev/get-dart.

2. **Create a new Dart Project**

   Use the Dart tool to create a new project:

   ```bash
   dart create my_backend_app
   ```

   Navigate to the new project directory:

   ```bash
   cd my_backend_app
   ```

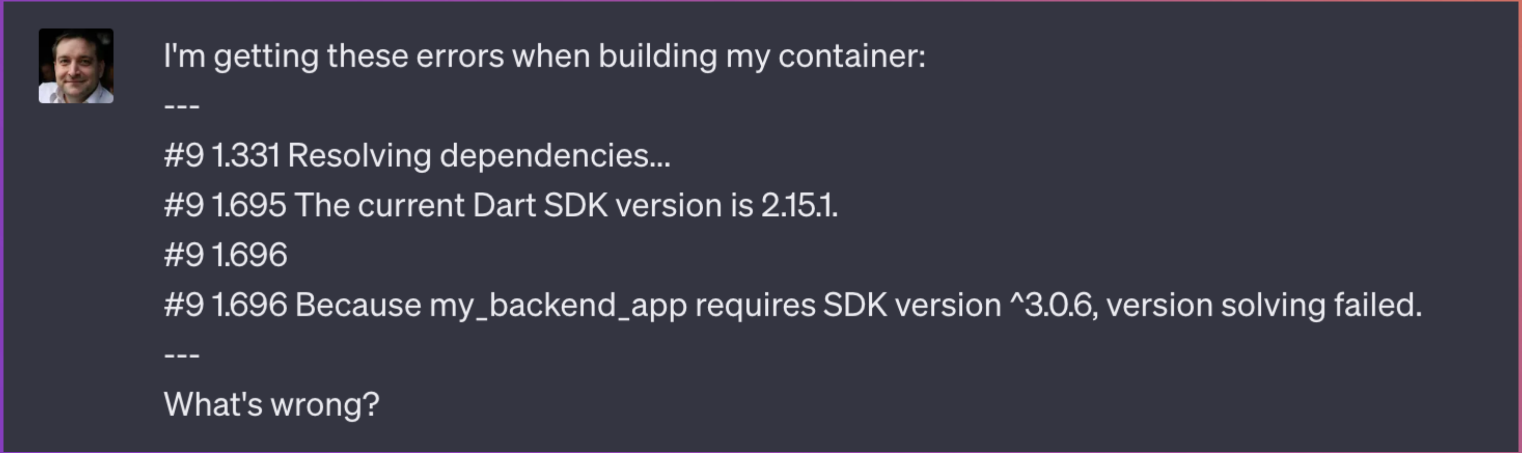
3. **Add Web Server Dependencies**

   To build a web server with Dart, you can use the `shelf` package. Open the

..then came the first hiccup!
The containerized `dart` was too old!

ChatGPT data is from 2022!
The proposed Dockerfile was with Dart 2, while the backend was for 3.0.x

I kept asking:

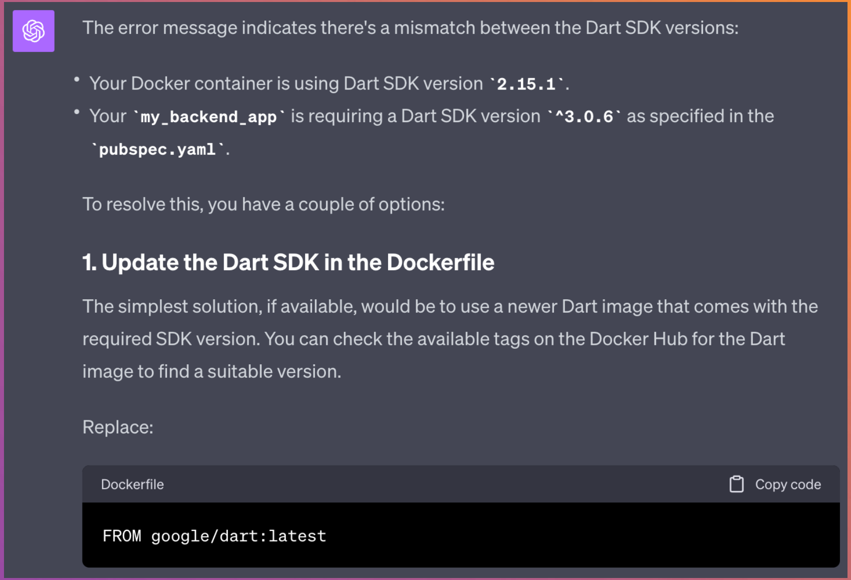


I'm getting these errors when building my container:

```
#9 1.331 Resolving dependencies...
#9 1.695 The current Dart SDK version is 2.15.1.
#9 1.696
#9 1.696 Because my_backend_app requires SDK version ^3.0.6, version solving failed.
```

What's wrong?

It answered
...and it was wrong!



The error message indicates there's a mismatch between the Dart SDK versions:

- Your Docker container is using Dart SDK version `` `2.15.1` ``.
- Your `` `my_backend_app` `` is requiring a Dart SDK version `` `^3.0.6` `` as specified in the `` `pubspec.yaml` ``.

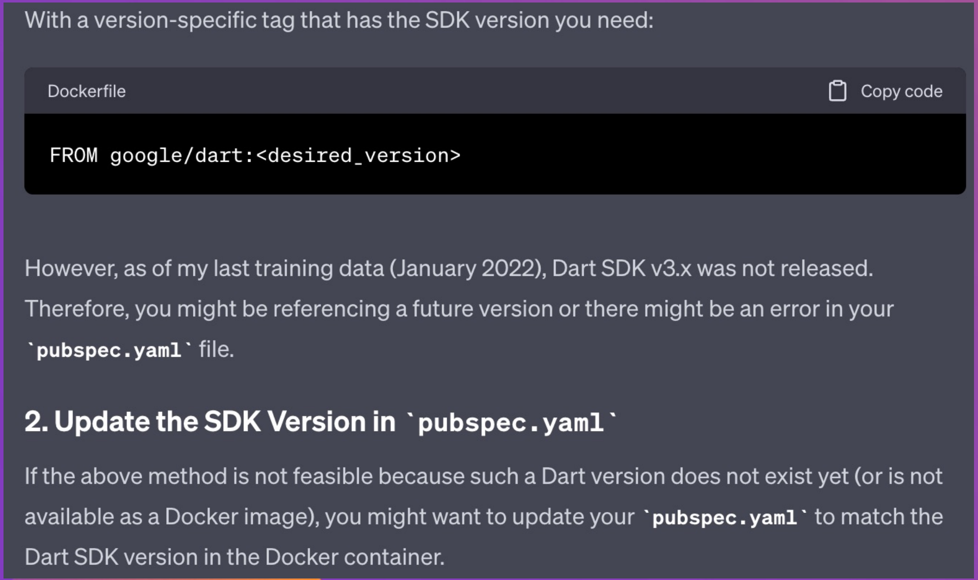
To resolve this, you have a couple of options:

1. Update the Dart SDK in the Dockerfile

The simplest solution, if available, would be to use a newer Dart image that comes with the required SDK version. You can check the available tags on the Docker Hub for the Dart image to find a suitable version.

Replace:

```
FROM google/dart:latest
```



With a version-specific tag that has the SDK version you need:

```
FROM google/dart:<desired_version>
```

However, as of my last training data (January 2022), Dart SDK v3.x was not released. Therefore, you might be referencing a future version or there might be an error in your `` `pubspec.yaml` `` file.

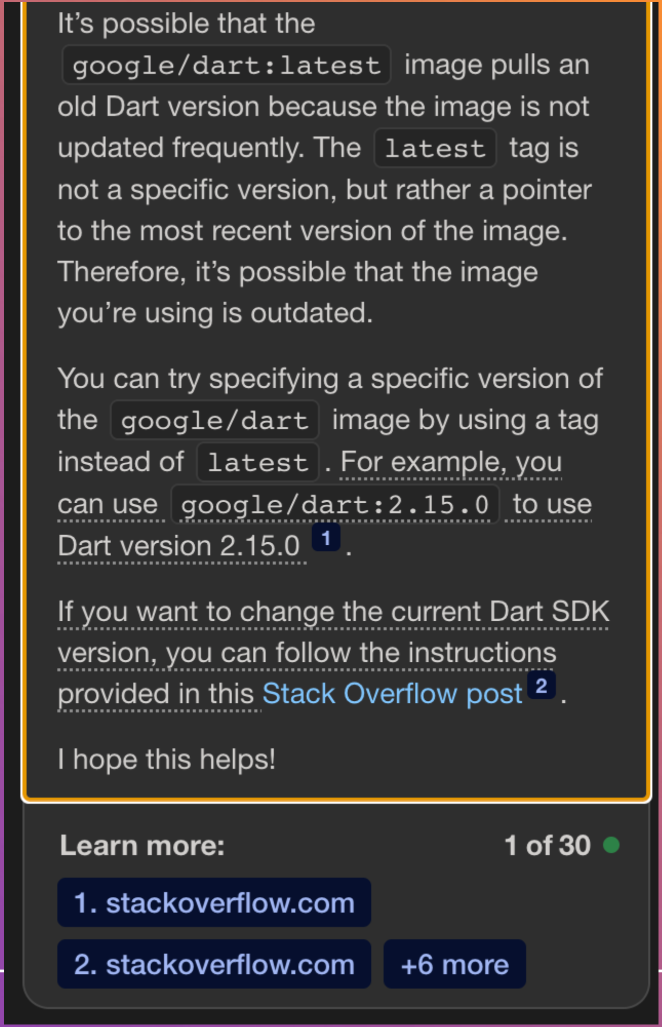
2. Update the SDK Version in `` `pubspec.yaml` ``

If the above method is not feasible because such a Dart version does not exist yet (or is not available as a Docker image), you might want to update your `` `pubspec.yaml` `` to match the Dart SDK version in the Docker container.

I was on my own here!
Well, not exactly.
I had Bing AI here to help!

..and Bing, too, gloriously failed:

I had to put my Senior Developer hat…



It's possible that the `google/dart:latest` image pulls an old Dart version because the image is not updated frequently. The `latest` tag is not a specific version, but rather a pointer to the most recent version of the image. Therefore, it's possible that the image you're using is outdated.

You can try specifying a specific version of the `google/dart` image by using a tag instead of `latest`. For example, you can use `google/dart:2.15.0` to use Dart version 2.15.0[1].

If you want to change the current Dart SDK version, you can follow the instructions provided in this Stack Overflow post[2].

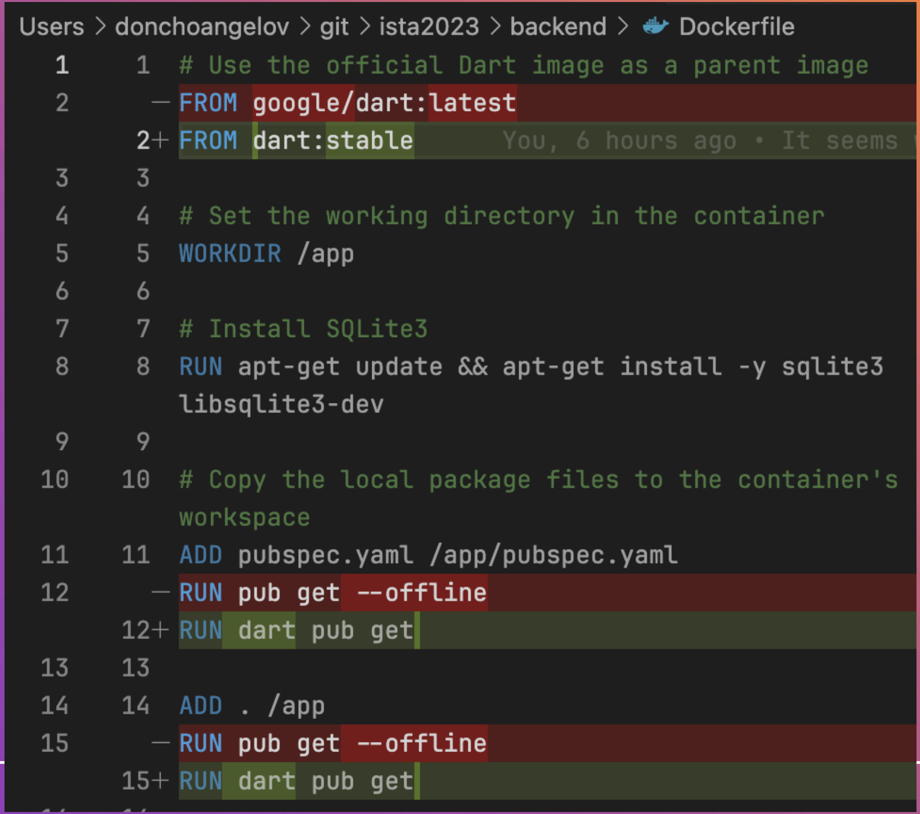
I hope this helps!

..and after some reading and research...

...one of the links in the Bing AI result took me to the solution!

(Bing AI is also ChatGPT-based)

It still came as a help from the AI, but it was not "that" straight-forward!

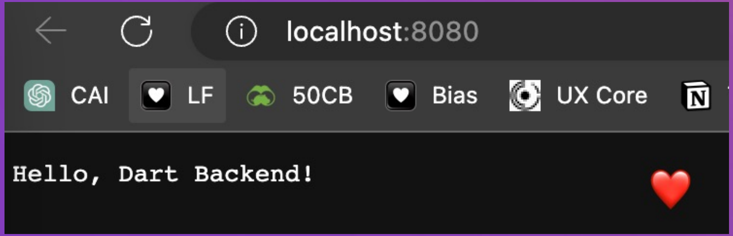


Users > donchoangelov > git > ista2023 > backend > Dockerfile

```
# Use the official Dart image as a parent image
— FROM google/dart:latest
+ FROM dart:stable        You, 6 hours ago • It seems

# Set the working directory in the container
WORKDIR /app

# Install SQLite3
RUN apt-get update && apt-get install -y sqlite3
libsqlite3-dev

# Copy the local package files to the container's
workspace
ADD pubspec.yaml /app/pubspec.yaml
— RUN pub get --offline
+ RUN dart pub get

ADD . /app
— RUN pub get --offline
+ RUN dart pub get
```

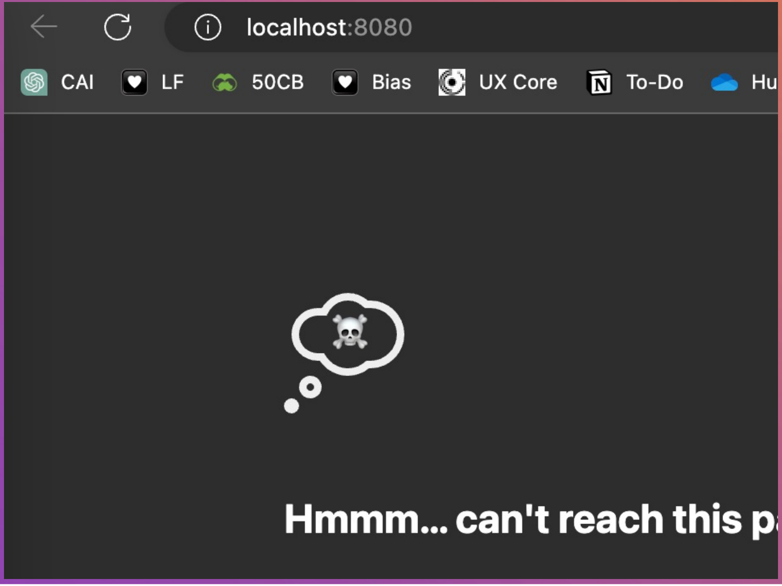
I got a build! And a container! Hurray! However:

My Dev environment:



Hello, Dart Backend!

My containerized backend:



Hmmm... can't reach this p

…but I'm just with basic Docker knowledge! Help, ChatGPT!

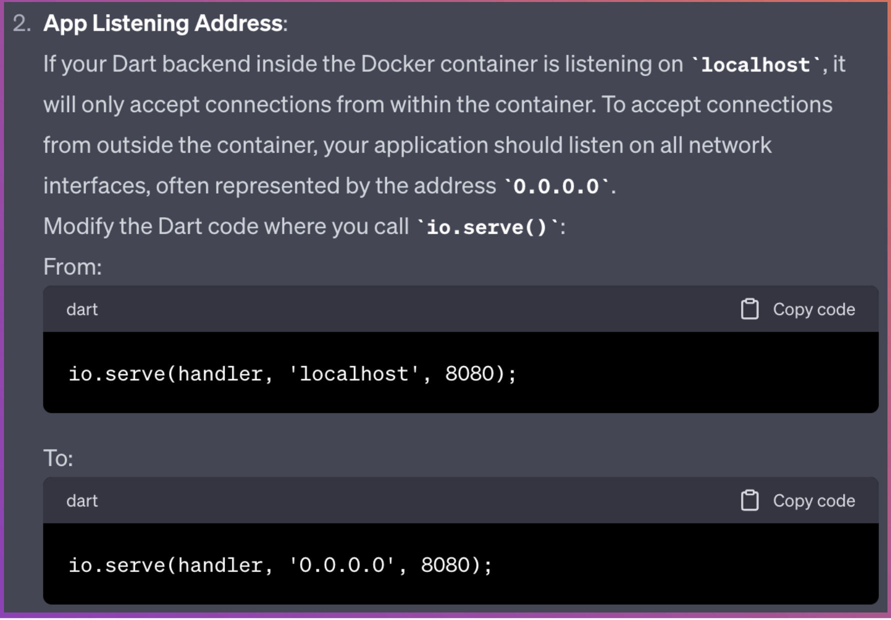
Prompt

it says "docker run -p 8080:8080 your-backend-image-name

Serving at http://localhost:8080"

But I cannot open anything at http://localhost:8080

Response (the 2nd part):



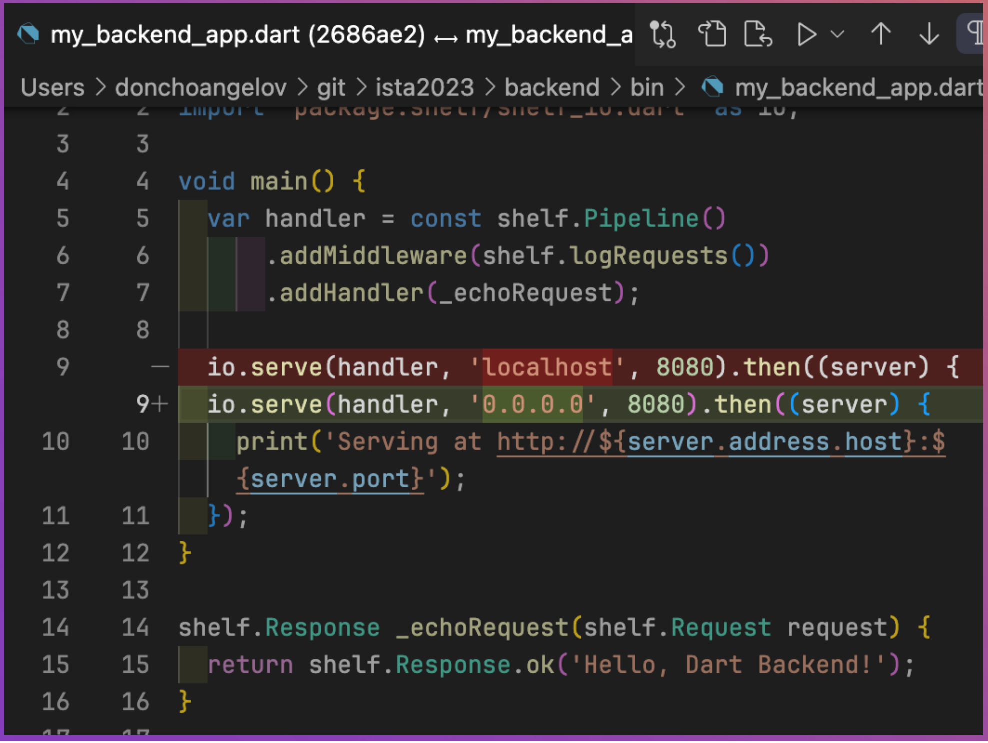
2. **App Listening Address**:
   If your Dart backend inside the Docker container is listening on `` `localhost` ``, it will only accept connections from within the container. To accept connections from outside the container, your application should listen on all network interfaces, often represented by the address `` `0.0.0.0` ``.
   Modify the Dart code where you call `` `io.serve()` ``:
   From:

```dart
io.serve(handler, 'localhost', 8080);
```

   To:

```dart
io.serve(handler, '0.0.0.0', 8080);
```

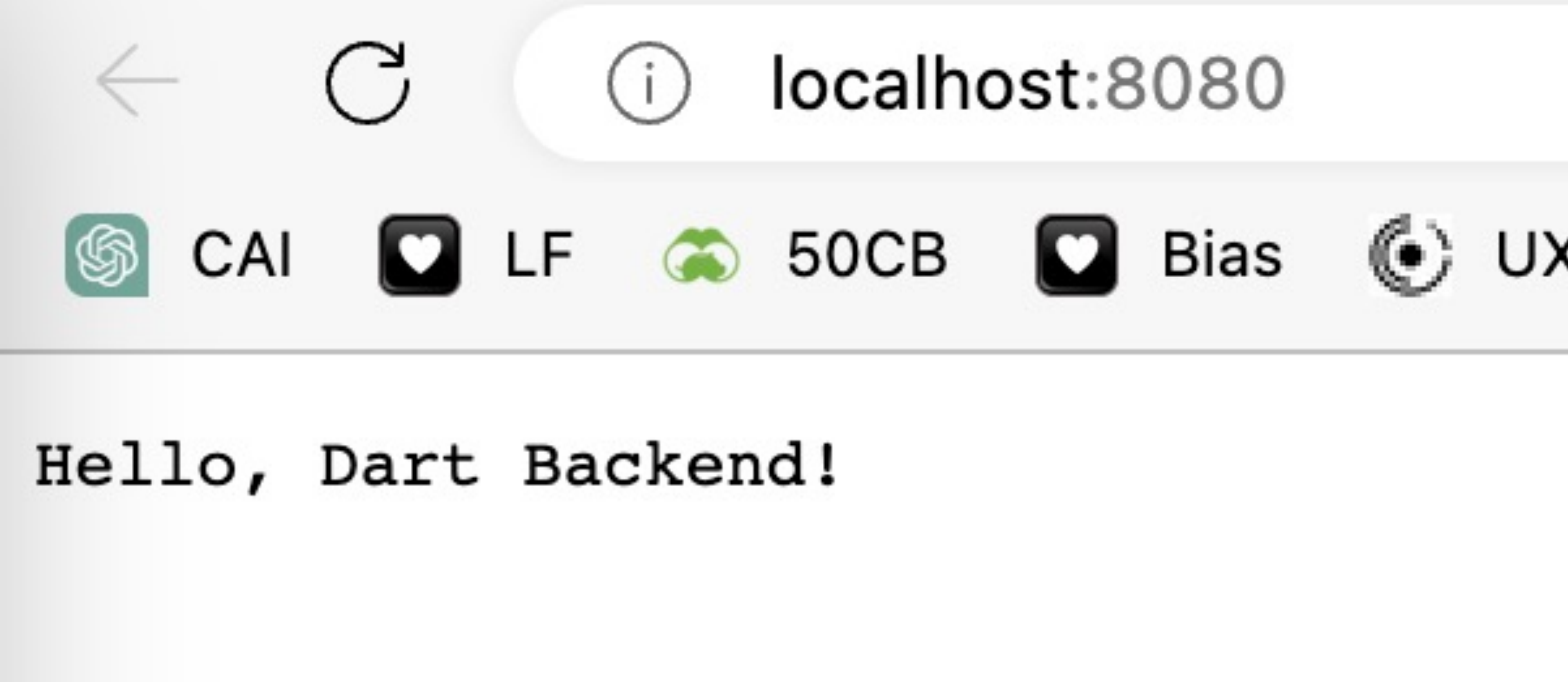
I did what it told me, and…



my_backend_app.dart (2686ae2) ⟷ my_backend_a

Users > donchoangelov > git > ista2023 > backend > bin > my_backend_app.dart

```
void main() {
  var handler = const shelf.Pipeline()
      .addMiddleware(shelf.logRequests())
      .addHandler(_echoRequest);

-  io.serve(handler, 'localhost', 8080).then((server) {
+  io.serve(handler, '0.0.0.0', 8080).then((server) {
    print('Serving at http://${server.address.host}:${server.port}');
  });
}

shelf.Response _echoRequest(shelf.Request request) {
  return shelf.Response.ok('Hello, Dart Backend!');
}
```



```
Hello, Dart Backend!
```

…and then I realized:

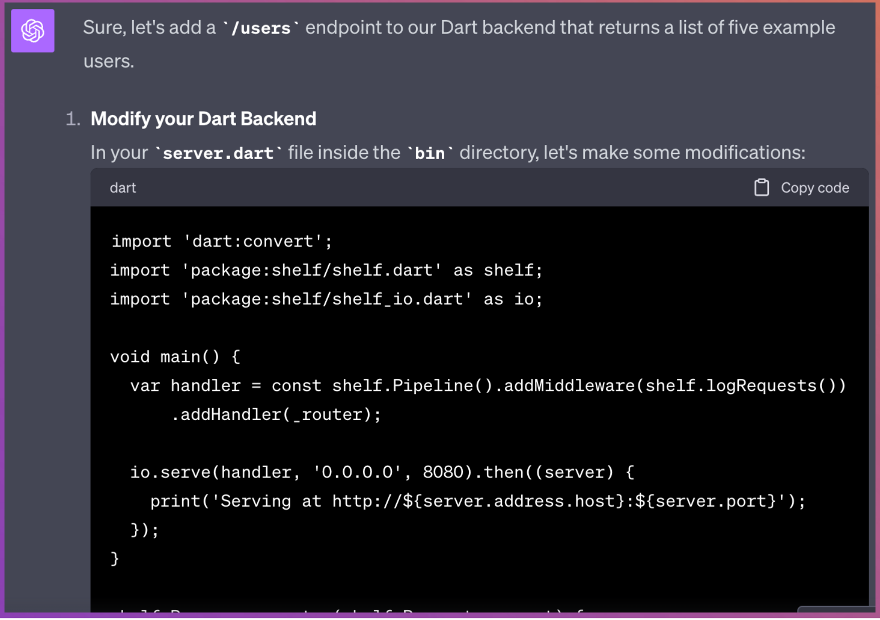
I have only 25 minutes presentation!

…so I ditched my SQLite plans in favor for simple `/users` backend stub!

Prompt

`Let's add `/users` backend API, which will be a testing stub for list of users, providing five example users with a fixed data.`

Response:



Sure, let's add a `/users` endpoint to our Dart backend that returns a list of five example users.

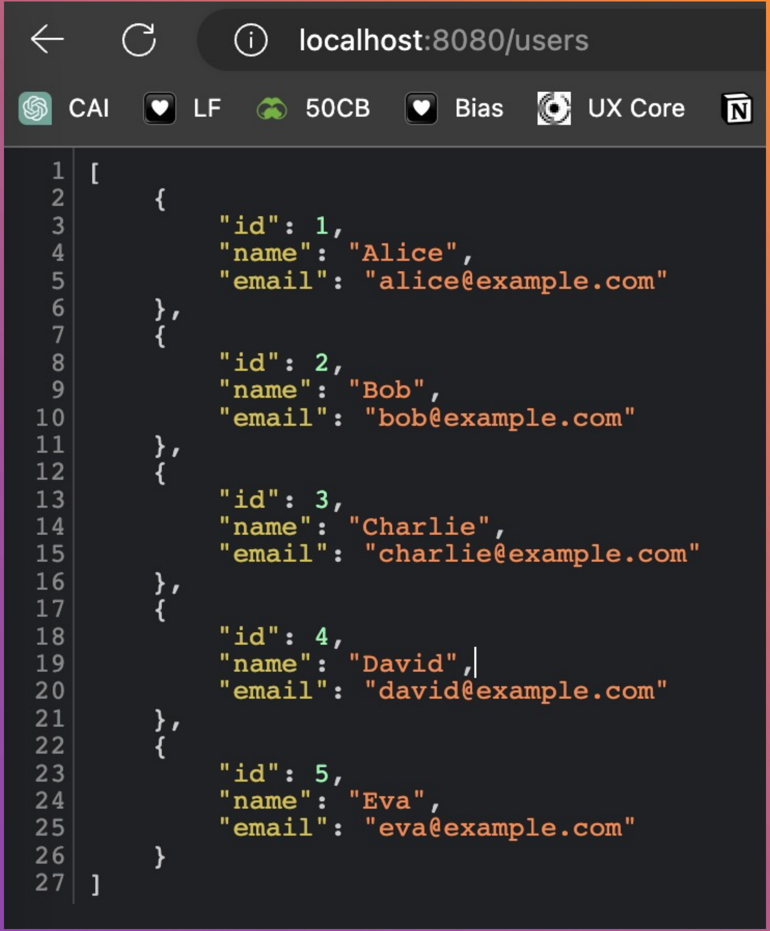
1. **Modify your Dart Backend**

   In your `server.dart` file inside the `bin` directory, let's make some modifications:

```dart
import 'dart:convert';
import 'package:shelf/shelf.dart' as shelf;
import 'package:shelf/shelf_io.dart' as io;

void main() {
  var handler = const shelf.Pipeline().addMiddleware(shelf.logRequests())
      .addHandler(_router);

  io.serve(handler, '0.0.0.0', 8080).then((server) {
    print('Serving at http://${server.address.host}:${server.port}');
  });
}
```

…test with `dart run`,
build the container,
run it, and…

So, I pronounced my backend as "READY"



```
[
        {
                "id": 1,
                "name": "Alice",
                "email": "alice@example.com"
        },
        {
                "id": 2,
                "name": "Bob",
                "email": "bob@example.com"
        },
        {
                "id": 3,
                "name": "Charlie",
                "email": "charlie@example.com"
        },
        {
                "id": 4,
                "name": "David",
                "email": "david@example.com"
        },
        {
                "id": 5,
                "name": "Eva",
                "email": "eva@example.com"
        }
]
```

The Frontend

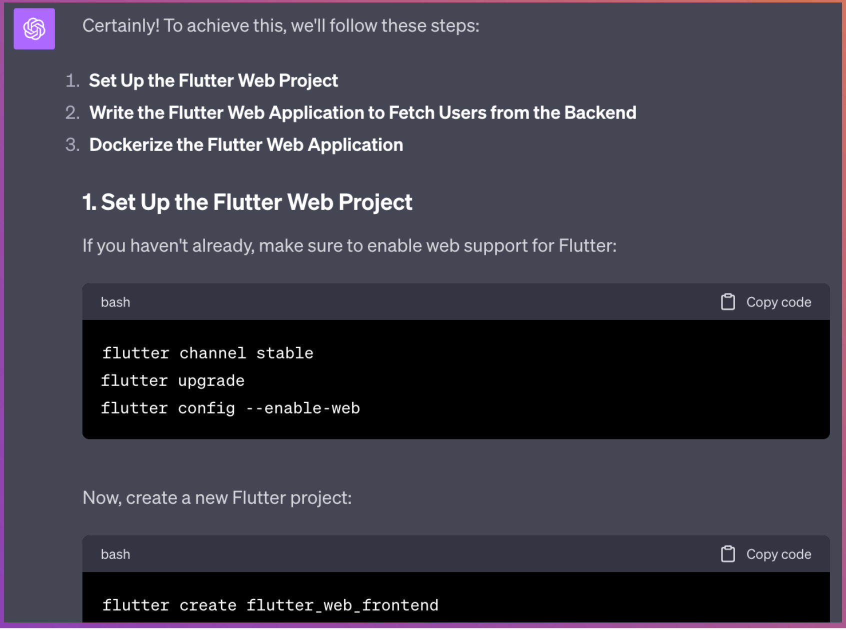
Pretty Much Straightforward Prompt

Prompt

Good! Now we should create the Frontend. I want the frontend to run on a different container, and connect with the backend container throughout the already exposed API.

Please, write the frontend by using Flutter and taking into account it'll produce a web application only.

Response:



Certainly! To achieve this, we'll follow these steps:

1. **Set Up the Flutter Web Project**
2. **Write the Flutter Web Application to Fetch Users from the Backend**
3. **Dockerize the Flutter Web Application**

1. Set Up the Flutter Web Project

If you haven't already, make sure to enable web support for Flutter:

```bash
flutter channel stable
flutter upgrade
flutter config --enable-web
```

Now, create a new Flutter project:

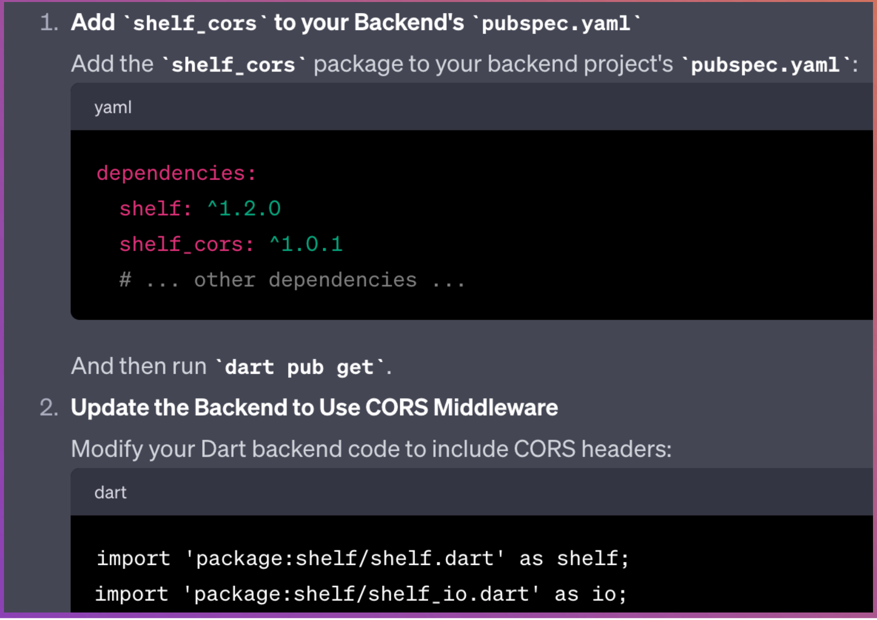
```bash
flutter create flutter_web_frontend
```

…and then it came the CORS middle(night)ware…

Prompt

```
I'm getting this error in my console when
trying to access the backend:
---
Access to XMLHttpRequest at
'http://localhost:8080/users' from origin
'http://localhost:64817' has been blocked
by CORS policy: No 'Access-Control-Allow-
Origin' header is present on the requested
resource.
---
How to fix it?
```

Response:



1. **Add `shelf_cors` to your Backend's `pubspec.yaml`**

   Add the `shelf_cors` package to your backend project's `pubspec.yaml`:

   yaml

   ```yaml
   dependencies:
     shelf: ^1.2.0
     shelf_cors: ^1.0.1
     # ... other dependencies ...
   ```

   And then run `dart pub get`.

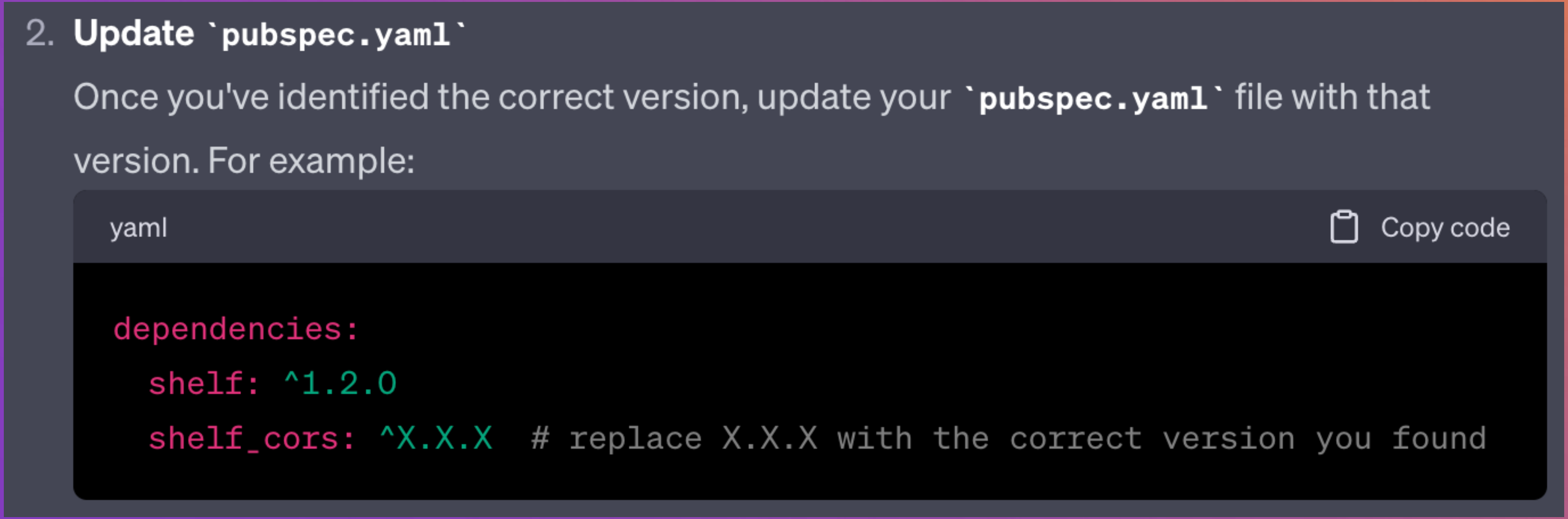
2. **Update the Backend to Use CORS Middleware**

   Modify your Dart backend code to include CORS headers:

   dart

   ```dart
   import 'package:shelf/shelf.dart' as shelf;
   import 'package:shelf/shelf_io.dart' as io;
   ```

ChatGPT was outdated (again)!



2. **Update `pubspec.yaml`**

   Once you've identified the correct version, update your `pubspec.yaml` file with that version. For example:

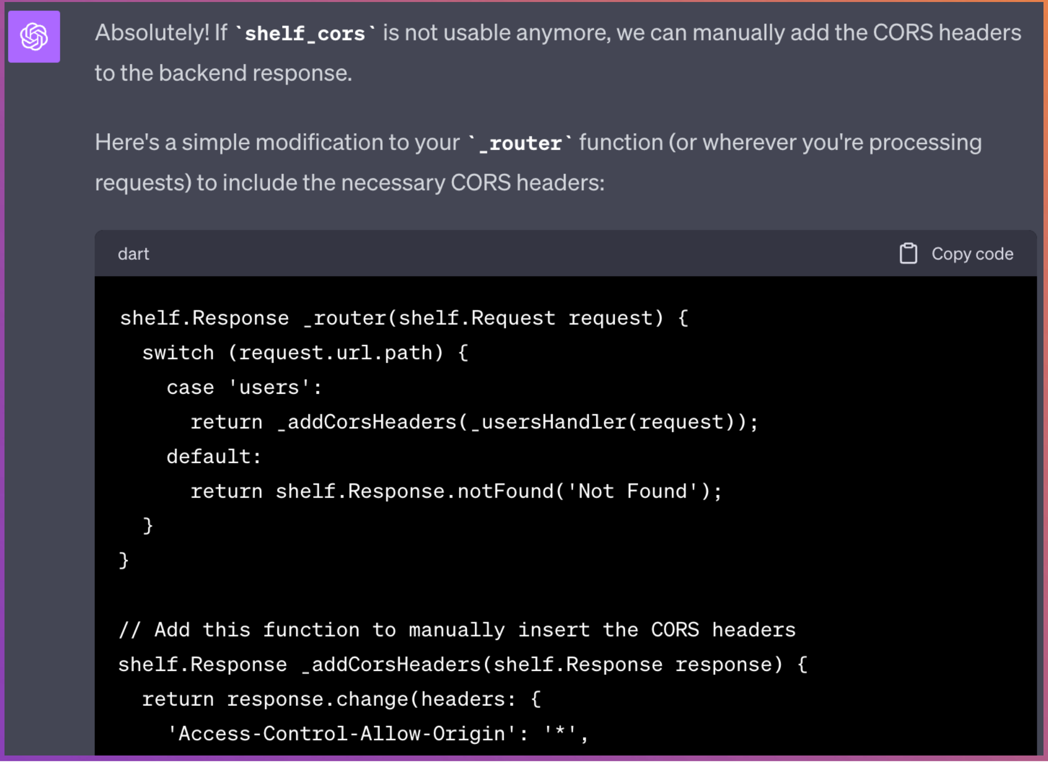
```yaml
dependencies:
  shelf: ^1.2.0
  shelf_cors: ^X.X.X  # replace X.X.X with the correct version you found
```

…but I was insistent!

Prompt

It seems that the shelf_cors package is unusable anymore. Is there any other way to do it?

Response:



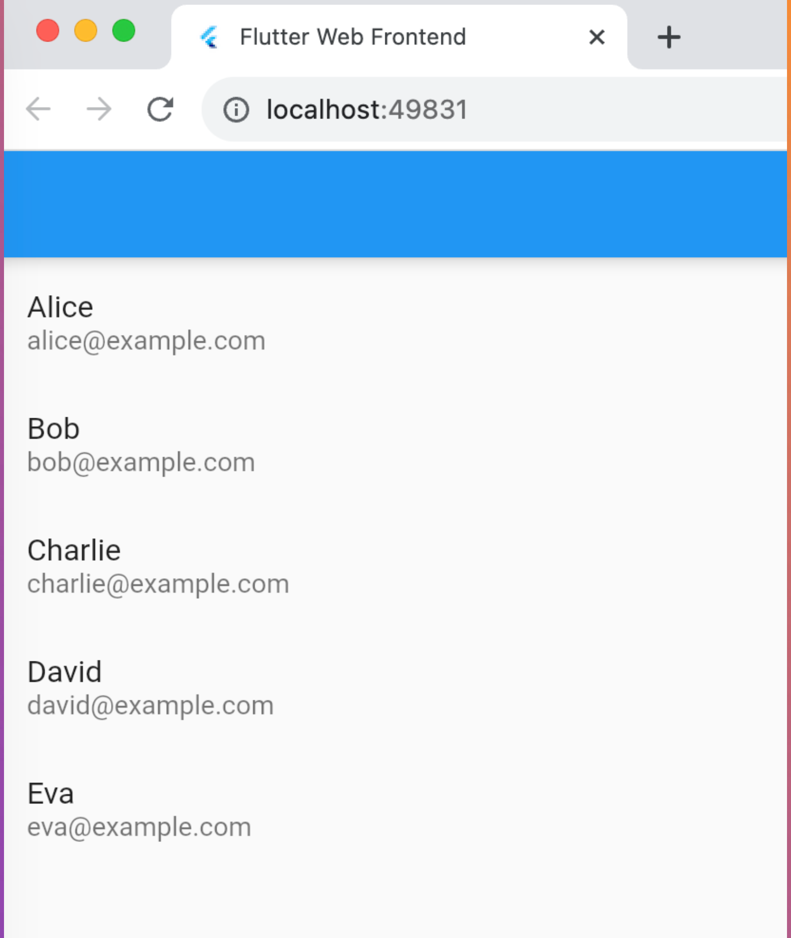
Absolutely! If `shelf_cors` is not usable anymore, we can manually add the CORS headers to the backend response.

Here's a simple modification to your `_router` function (or wherever you're processing requests) to include the necessary CORS headers:

```dart
shelf.Response _router(shelf.Request request) {
  switch (request.url.path) {
    case 'users':
      return _addCorsHeaders(_usersHandler(request));
    default:
      return shelf.Response.notFound('Not Found');
  }
}

// Add this function to manually insert the CORS headers
shelf.Response _addCorsHeaders(shelf.Response response) {
  return response.change(headers: {
    'Access-Control-Allow-Origin': '*',
```

...and finally, a running frontend!

Then I pronounced the Frontend as "READY"



Alice
alice@example.com

Bob
bob@example.com

Charlie
charlie@example.com

David
david@example.com

Eva
eva@example.com

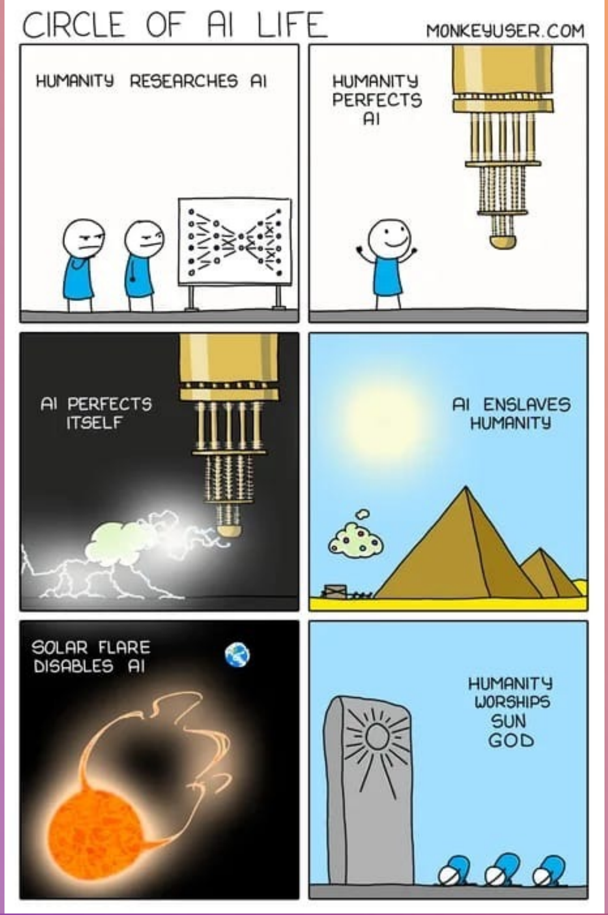
12

What Next?

Select AI of your taste

Learn prompts and use it daily

Own your destiny



Handoffs | Q&A

https://tinyurl.com/DAista2023

https://about.me/donangel