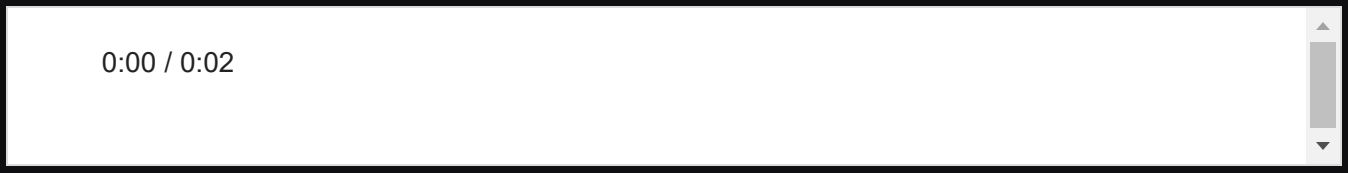
Audio

Overview

Audio is the equivalent of an `<audio>` HTML element. Audio displays the browser's native audio controls.

Examples




```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/audio",
)
def app():
  """
  In order to autoplay audio, set the `autoplay` attribute to `True`,
  Note that there are autoplay restrictions in modern browsers, including Chrome,
  are designed to prevent audio or video from playing automatically without user interaction.
  This is intended to improve user experience and reduce unwanted interruptions.
  You can check the [autoplay ability of your application](https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay_guide#autoplay_availability)
  """
  me.audio(
    src="https://interactive-examples.mdn.mozilla.net/media/cc0-audio/t-rex-roar.mp3",
    # autoplay=True
  )
```

API

`func` `audio`

Creates an audio component.

PARAMETER	DESCRIPTION
`src`	The URL of the audio to be played. **TYPE:** `str | None` **DEFAULT:** `None`
`autoplay`	boolean value indicating if the audio should be autoplayed or not. **Note**: There are autoplay restrictions in modern browsers, including Chrome, are designed to prevent audio or video from playing automatically without user interaction. This is intended to improve user experience and reduce unwanted interruptions **TYPE:** `bool` **DEFAULT:** `False`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Auth

To ensure that the users of your Mesop application are authenticated, this guide provides a detailed, step-by-step process on how to integrate Firebase Authentication with Mesop using a web component.

Mesop is designed to be auth provider agnostic, allowing you to integrate any auth library you prefer, whether it's on the client-side (JavaScript) or server-side (Python). You can support sign-ins, including social sign-ins like Google's or any others that you prefer. The general approach involves signing in on the client-side first, then transmitting an auth token to the server-side.

Firebase Authentication

This guide will walk you through the process of integrating Firebase Authentication with Mesop using a custom web component.

Pre-requisites: You will need to create a Firebase account and project. It's free to get started with Firebase and use Firebase auth for small projects, but refer to the pricing page for the most up-to-date information.

We will be using three libraries from Firebase to build an end-to-end auth flow:

- Firebase Web SDK: Allows you to call Firebase services from your client-side JavaScript code.
- FirebaseUI Web: Provides a simple, customizable auth UI integrated with the Firebase Web SDK.
- Firebase Admin SDK (Python): Provides server-side libraries to integrate Firebase services, including Authentication, into your Python applications.

Let's dive into how we will use each one in our Mesop app.

Web component

The Firebase Authentication web component is a custom component built for handling the user authentication process. It's implemented using Lit, a simple library for building lightweight web components.

JS code

firebase_auth_component.js

```
import {
  LitElement,
  html,
} from 'https://cdn.jsdelivr.net/gh/lit/dist@3/core/lit-core.min.js';

import 'https://www.gstatic.com/firebasejs/10.0.0/firebase-app-compat.js';
import 'https://www.gstatic.com/firebasejs/10.0.0/firebase-auth-compat.js';
import 'https://www.gstatic.com/firebasejs/ui/6.1.0/firebase-ui-auth.js';

// TODO: replace this with your web app's Firebase configuration
const firebaseConfig = {
  apiKey: 'AIzaSyAQR9T7sk1lElXTEUBYHx7jv7d_Bs2zt-s',
  authDomain: 'mesop-auth-test.firebaseapp.com',
  projectId: 'mesop-auth-test',
  storageBucket: 'mesop-auth-test.appspot.com',
  messagingSenderId: '565166920272',
  appId: '1:565166920272:web:4275481621d8e5ba91b755',
};

// Initialize Firebase
firebase.initializeApp(firebaseConfig);

var uiConfig = {
  // TODO: change this to your Mesop page path.
  signInSuccessUrl: '/web_component/firebase_auth/firebase_auth_app',
  signInOptions: [firebase.auth.GoogleAuthProvider.PROVIDER_ID],
  // tosUrl and privacyPolicyUrl accept either url string or a callback
  // function.
  // Terms of service url/callback.
  tosUrl: '<your-tos-url>',
  // Privacy policy url/callback.
  privacyPolicyUrl: function () {
    window.location.assign('<your-privacy-policy-url>');
  },
};

// Initialize the FirebaseUI Widget using Firebase.
var ui = new firebaseui.auth.AuthUI(firebase.auth());

class FirebaseAuthComponent extends LitElement {
  static properties = {
    isSignedIn: {type: Boolean},
    authChanged: {type: String},
  };

  constructor() {
    super();
    this.isSignedIn = false;
  }

  createRenderRoot() {
```

```
    // Render in light DOM so firebase-ui-auth works.
    return this;
  }

  firstUpdated() {
    firebase.auth().onAuthStateChanged(
      async (user) => {
        if (user) {
          this.isSignedIn = true;
          const token = await user.getIdToken();
          this.dispatchEvent(new MesopEvent(this.authChanged, token));
        } else {
          this.isSignedIn = false;
          this.dispatchEvent(new MesopEvent(this.authChanged, ''));
        }
      },
      function (error) {
        console.log(error);
      },
    );

    ui.start('#firebaseui-auth-container', uiConfig);
  }

  signOut() {
    firebase.auth().signOut();
  }

  render() {
    return html`
      <div
        id="firebaseui-auth-container"
        style="${this.isSignedIn ? 'display: none' : ''}"
      ></div>
      <div
        class="firebaseui-container firebaseui-page-provider-sign-in firebaseui-id-page-provider-sign-in firebaseui-use-spinner"
        style="${this.isSignedIn ? '' : 'display: none'}"
      >
        <button
          style="background-color:#ffffff"
          class="firebaseui-idp-button mdl-button mdl-js-button mdl-button--raised firebaseui-idp-google firebaseui-id-idp-button"
          @click="${this.signOut}"
        >
          <span class="firebaseui-idp-text firebaseui-idp-text-long"
            >Sign out</span
          >
        </button>
      </div>
    `;
  }
}

customElements.define('firebase-auth-component', FirebaseAuthComponent);
```

What you need to do:

- Replace `firebaseConfig` with your Firebase project's config. Read the Firebase docs to learn how to get yours.
- Replace the URLs `signInSuccessUrl` with your Mesop page path and `tosUrl` and `privacyPolicyUrl` to your terms and services and privacy policy page respectively.

How it works:

- This creates a simple and configurable auth UI using FirebaseUI Web.
- Once the user has signed in, then a sign out button is shown.
- Whenever the user signs in or out, the web component dispatches an event to the Mesop server with the auth token, or absence of it.
- See our web component docs for more details.

Python code

firebase_auth_component.py

```
from typing import Any, Callable

import mesop.labs as mel


@mel.web_component(path="./firebase_auth_component.js")
def firebase_auth_component(on_auth_changed: Callable[[mel.WebEvent], Any]):
  return mel.insert_web_component(
    name="firebase-auth-component",
    events={
      "authChanged": on_auth_changed,
    },
  )
```

How it works:

- Implements the Python side of the Mesop web component. See our web component docs for more details.

Integrating into the app

Let's put it all together:

firebase_auth_app.py

```
import firebase_admin
from firebase_admin import auth

import mesop as me
import mesop.labs as mel
from mesop.examples.web_component.firebase_auth.firebase_auth_component import (
  firebase_auth_component,
)

# Avoid re-initializing firebase app (useful for avoiding warning message because of hot reloads).
if firebase_admin._DEFAULT_APP_NAME not in firebase_admin._apps:
  default_app = firebase_admin.initialize_app()


@me.page(
  path="/web_component/firebase_auth/firebase_auth_app",
  stylesheets=[
    "https://www.gstatic.com/firebasejs/ui/6.1.0/firebase-ui-auth.css"
  ],
  # Loosen the security policy so the firebase JS libraries work.
  security_policy=me.SecurityPolicy(
    dangerously_disable_trusted_types=True,
    allowed_connect_srcs=["*.googleapis.com"],
    allowed_script_srcs=["*.google.com"],
  ),
)
def page():
  email = me.state(State).email
  if email:
    me.text("Signed in email: " + email)
  else:
    me.text("Not signed in")
  firebase_auth_component(on_auth_changed=on_auth_changed)


@me.stateclass
class State:
  email: str


def on_auth_changed(e: mel.WebEvent):
  print("AUTH", e.value)
  firebaseAuthToken = e.value
  if not firebaseAuthToken:
    me.state(State).email = ""
    return

  decoded_token = auth.verify_id_token(firebaseAuthToken)
  if decoded_token["email"] != "allowlisted.user@gmail.com":
    raise me.MesopUserException("Invalid user: " + decoded_token["email"])
  me.state(State).email = decoded_token["email"]
```

How it works:

- The `firebase_auth_app.py` module integrates the Firebase Auth web component into the Mesop app. It initializes the Firebase app, defines the page where the Firebase Auth web component will be used, and sets up the state to store the user's email.
- The `on_auth_changed` function is triggered whenever the user's authentication state changes. If the user is signed in, it verifies the user's ID token and stores the user's email in the state. If the user is not signed in, it clears the email from the state.

Next steps

Congrats! You've now built an authenticated app with Mesop from start to finish. Read the Firebase Auth docs to learn how to configure additional sign-in options and much more.

Badge

Overview

Badge decorates a UI component and is oftentimes used for unread message count and is based on the Angular Material badge component.

Examples

```
with
badge
```

```python
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/badge",
)
def app():
  with me.box(
    style=me.Style(
      display="block",
      padding=me.Padding(top=16, right=16, bottom=16, left=16),
      height=50,
      width=30,
    )
  ):
    with me.badge(content="1", size="medium"):
      me.text(text="text with badge")
```

API

`func` badge

Creates a Badge component. Badge is a composite component.

PARAMETER	DESCRIPTION
`color`	The color of the badge. Can be `primary`, `accent`, or `warn`. **TYPE:** `Literal['primary', 'accent', 'warn']` **DEFAULT:** `'primary'`
`overlap`	Whether the badge should overlap its contents or not **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Whether the badge is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`position`	Position the badge should reside. Accepts any combination of 'above'|'below' and 'before'|'after' **TYPE:** `Literal['above after', 'above before', 'below before', 'below after', 'before', 'after', 'above', 'below']` **DEFAULT:** `'above after'`
`content`	The content for the badge **TYPE:** `str` **DEFAULT:** `''`
`description`	Message used to describe the decorated element via aria-describedby **TYPE:** `str` **DEFAULT:** `''`
`size`	Size of the badge. Can be 'small', 'medium', or 'large'. **TYPE:** `Literal['small', 'medium', 'large']` **DEFAULT:** `'small'`
`hidden`	Whether the badge is hidden. **TYPE:** `bool` **DEFAULT:** `False`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Box

Overview

Box is a content component which acts as a container to group children components and styling them.

Examples


```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/box",
)
def app():
  with me.box(style=me.Style(background="red", padding=me.Padding.all(16))):
    with me.box(
      style=me.Style(
        background="green",
        height=50,
        margin=me.Margin.symmetric(vertical=24, horizontal=12),
        border=me.Border.symmetric(
          horizontal=me.BorderSide(width=2, color="pink", style="solid"),
          vertical=me.BorderSide(width=2, color="orange", style="solid"),
        ),
      )
    ):
      me.text(text="hi1")
      me.text(text="hi2")
```

API

`func` box

Creates a box component.

PARAMETER	DESCRIPTION
`style`	Style to apply to component. Follows HTML Element inline style API. **TYPE:** `Style | None` **DEFAULT:** `None`
`on_click`	The callback function that is called when the box is clicked. It receives a ClickEvent as its only argument. **TYPE:** `Callable[[ClickEvent], Any] | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

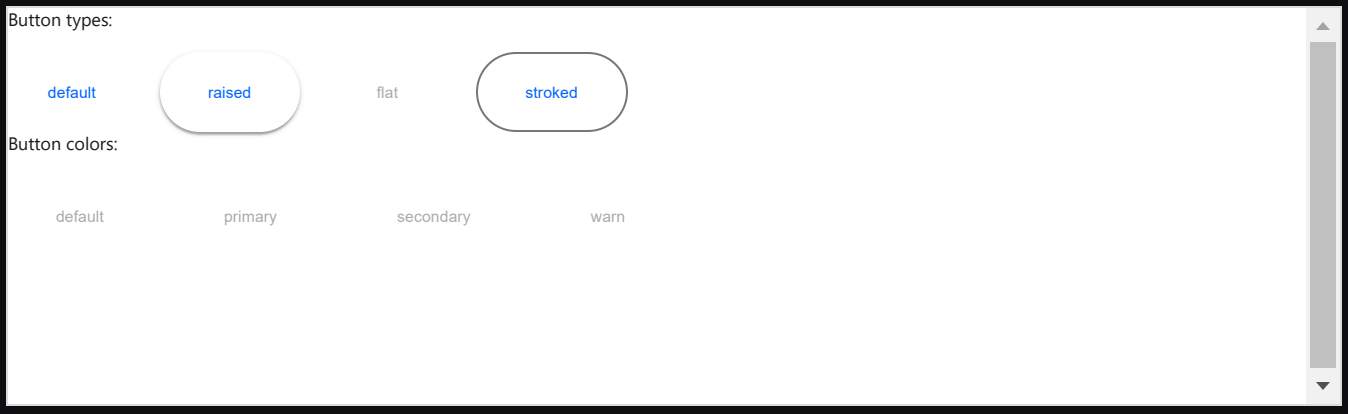
RETURNS	DESCRIPTION
`Any`	The created box component.

Button

Overview

Button is based on the Angular Material button component.

Examples




```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/button",
)
def main():
  me.text("Button types:", style=me.Style(margin=me.Margin(bottom=12)))
  with me.box(style=me.Style(display="flex", flex_direction="row", gap=12)):
    me.button("default")
    me.button("raised", type="raised")
    me.button("flat", type="flat")
    me.button("stroked", type="stroked")

  me.text("Button colors:", style=me.Style(margin=me.Margin(bottom=12)))
  with me.box(style=me.Style(display="flex", flex_direction="row", gap=12)):
    me.button("default", type="flat")
    me.button("primary", color="primary", type="flat")
    me.button("secondary", color="accent", type="flat")
    me.button("warn", color="warn", type="flat")
```

API

`func` button

Creates a simple text Button component.

PARAMETER	DESCRIPTION
`label`	Text label for button **TYPE:** `str | None` **DEFAULT:** `None`
`on_click`	click is a native browser event. **TYPE:** `Callable[[ClickEvent], Any] | None` **DEFAULT:** `None`
`type`	Type of button style to use **TYPE:** `Literal['raised', 'flat', 'stroked'] | None` **DEFAULT:** `None`
`color`	Theme color palette of the button **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`disable_ripple`	Whether the ripple effect is disabled or not. **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Whether the button is disabled. **TYPE:** `bool` **DEFAULT:** `False`

PARAMETER	DESCRIPTION
`style`	Style for the component. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`func` content_button

Creates a button component, which is a composite component. Typically, you would use a text or icon component as a child.

Intended for advanced use cases.

PARAMETER	DESCRIPTION
`on_click`	click is a native browser event. **TYPE:** `Callable[[ClickEvent], Any] | None` **DEFAULT:** `None`
`type`	Type of button style to use **TYPE:** `Literal['raised', 'flat', 'stroked', 'icon'] | None` **DEFAULT:** `None`
`color`	Theme color palette of the button **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`disable_ripple`	Whether the ripple effect is disabled or not. **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Whether the button is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`style`	Style for the component. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`class` ClickEvent `dataclass`

Bases: `MesopEvent`

Represents a user click event.

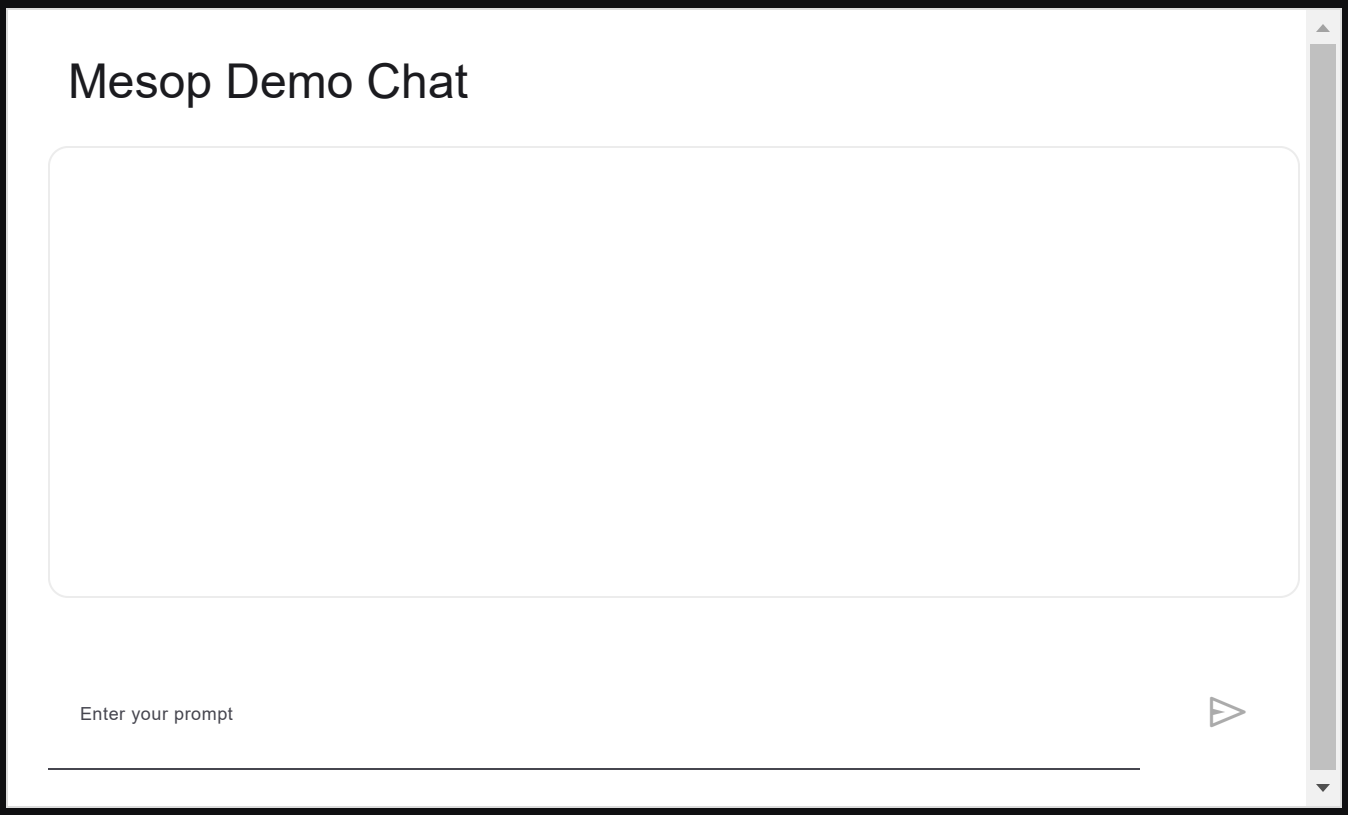
ATTRIBUTE	DESCRIPTION
`key`	key of the component that emitted this event. **TYPE:** `str`

Chat

Overview

Chat component is a quick way to create a simple chat interface. Chat is part of Mesop Labs.

Examples




```
import random
import time

import mesop as me
import mesop.labs as mel


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/chat",
  title="Mesop Demo Chat",
)
def page():
  mel.chat(transform, title="Mesop Demo Chat", bot_user="Mesop Bot")


def transform(input: str, history: list[mel.ChatMessage]):
  for line in random.sample(LINES, random.randint(3, len(LINES) - 1)):
    time.sleep(0.3)
    yield line + " "


LINES = [
  "Mesop is a Python-based UI framework designed to simplify web UI development for engineers without frontend experience.",
  "It leverages the power of the Angular web framework and Angular Material components, allowing rapid construction of web demos and internal tools.",
  "With Mesop, developers can enjoy a fast build-edit-refresh loop thanks to its hot reload feature, making UI tweaks and component integration
seamless.",
  "Deployment is straightforward, utilizing standard HTTP technologies.",
  "Mesop's component library aims for comprehensive Angular Material component coverage, enhancing UI flexibility and composability.",
  "It supports custom components for specific use cases, ensuring developers can extend its capabilities to fit their unique requirements.",
  "Mesop's roadmap includes expanding its component library and simplifying the onboarding processs.",
]
```

API

`func` `chat`

Creates a simple chat UI which takes in a prompt and chat history and returns a response to the prompt.

This function creates event handlers for text input and output operations using the provided function `transform` to process the input and generate the output.

PARAMETER	DESCRIPTION
`transform`	Function that takes in a prompt and chat history and returns a response to the prompt. **TYPE:** `Callable[[str, list[ChatMessage]], Generator[str, None, None] | str]`
`title`	Headline text to display at the top of the UI. **TYPE:** `str | None` **DEFAULT:** `None`
`bot_user`	Name of your bot / assistant. **TYPE:** `str` **DEFAULT:** `_BOT_USER_DEFAULT`

Checkbox

Overview

Checkbox is a multi-selection form control and is based on the Angular Material checkbox component.

Examples


```
import mesop as me


@me.stateclass
class State:
  checked: bool


def on_update(event: me.CheckboxChangeEvent):
  state = me.state(State)
  state.checked = event.checked


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/checkbox",
)
def app():
  state = me.state(State)
  me.checkbox(
    "Simple checkbox",
    on_change=on_update,
  )

  if state.checked:
    me.text(text="is checked")
  else:
    me.text(text="is not checked")
```

API

`func` checkbox

Creates a simple Checkbox component with a text label.

PARAMETER	DESCRIPTION
`label`	Text label for checkbox **TYPE:** `str | None` **DEFAULT:** `None`
`on_change`	Event emitted when the checkbox's `checked` value changes. **TYPE:** `Callable[[CheckboxChangeEvent], Any] | None` **DEFAULT:** `None`
`on_indeterminate_change`	Event emitted when the checkbox's `indeterminate` value changes. **TYPE:** `Callable[[CheckboxIndeterminateChangeEvent], Any] | None` **DEFAULT:** `None`
`label_position`	Whether the label should appear after or before the checkbox. Defaults to 'after' **TYPE:** `Literal['before', 'after']` **DEFAULT:** `'after'`
`disable_ripple`	Whether the checkbox has a ripple. **TYPE:** `bool` **DEFAULT:** `False`
`tab_index`	Tabindex for the checkbox. **TYPE:** `int` **DEFAULT:** `0`
`color`	Palette color of the checkbox.

PARAMETER	DESCRIPTION
	TYPE: `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`checked`	Whether the checkbox is checked. **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Whether the checkbox is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`indeterminate`	Whether the checkbox is indeterminate. This is also known as "mixed" mode and can be used to represent a checkbox with three states, e.g. a checkbox that represents a nested list of checkable items. Note that whenever checkbox is manually clicked, indeterminate is immediately set to false. **TYPE:** `bool` **DEFAULT:** `False`
`style`	Style for the component. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`func` content_checkbox

Creates a Checkbox component which is a composite component. Typically, you would use a text or icon component as a child.

Intended for advanced use cases.

PARAMETER	DESCRIPTION
`on_change`	Event emitted when the checkbox's `checked` value changes. **TYPE:** `Callable[[CheckboxChangeEvent], Any] | None` **DEFAULT:** `None`
`on_indeterminate_change`	Event emitted when the checkbox's `indeterminate` value changes. **TYPE:** `Callable[[CheckboxIndeterminateChangeEvent], Any] | None` **DEFAULT:** `None`
`label_position`	Whether the label should appear after or before the checkbox. Defaults to 'after' **TYPE:** `Literal['before', 'after']` **DEFAULT:** `'after'`
`disable_ripple`	Whether the checkbox has a ripple. **TYPE:** `bool` **DEFAULT:** `False`
`tab_index`	Tabindex for the checkbox. **TYPE:** `int` **DEFAULT:** `0`
`color`	Palette color of the checkbox. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`checked`	Whether the checkbox is checked. **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Whether the checkbox is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`indeterminate`	Whether the checkbox is indeterminate. This is also known as "mixed" mode and can be used to represent a checkbox with three states, e.g. a checkbox that represents a nested list of checkable items. Note that whenever checkbox is manually clicked, indeterminate is immediately set to false. **TYPE:** `bool` **DEFAULT:** `False`
`style`	Style for the component. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`class` CheckboxChangeEvent `dataclass`

Bases: `MesopEvent`

Represents a checkbox state change event.

ATTRIBUTE	DESCRIPTION
`checked`	The new checked state of the checkbox. **TYPE:** `bool`
`key`	key of the component that emitted this event. **TYPE:** `str`

`class` CheckboxIndeterminateChangeEvent `dataclass`

Bases: `MesopEvent`

Represents a checkbox indeterminate state change event.

ATTRIBUTE	DESCRIPTION
`checked`	The new indeterminate state of the checkbox.
`key`	key of the component that emitted this event. **TYPE:** `str`

Code

Overview

Code is used to render code with syntax highlighting. `code` is a simple wrapper around markdown.

Examples

API

`func` code

Creates a code component which displays code with syntax highlighting.

Debugging

VS Code is recomended for debugging your Mesop app, but you should be able to debug Mesop in other IDEs.

Debugging in VS Code

Pre-requisite: ensure VS Code is downloaded.

1. Install the Python Debugger VS Code extension.
2. Include the following in your `.vscode/launch.json`:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Remote Attach",
      "type": "python",
      "request": "attach",
      "connect": { "host": "localhost", "port": 5678 },
      "pathMappings": [
        { "localRoot": "${workspaceFolder}", "remoteRoot": "." }
      ],
      "justMyCode": true
    }
  ]
}
```

3. At the top of your Mesop app (e.g. main.py), including the following snippet to start the debug server:

```
import debugpy

debugpy.listen(5678)
```

4. Connect to your debug server by going to the Run & Debug tab in VS Code and selecting "Python: Remote Attach".

Congrats you are now debugging your Mesop app!

To learn more about Python debugging in VS code, read VS Code's Python debugging guide.

Deployment

To deploy your Mesop application, we recommend using Google Cloud Run because it's easy to get started and there's a free tier. However, it's possible to deploy your Mesop to other Cloud platforms.

Example application

Python

`main.py` which is your Mesop application code:

main.py

```
import mesop as me

@me.page(title="Home")
def home():
  me.text("Hello, world")
```

> Note: if you choose to use a different filename than main.py, you will need to modify the `Procfile` below.

Procfile

`Procfile` which configures `gunicorn` to run Mesop.

Procfile

```
web: gunicorn --bind :8080 main:me
```

The `--bind: 8080` will run Mesop on port 8080.

The `main:me` syntax is `$(MODULE_NAME):$(VARIABLE_NAME)` : (see Gunicorn docs for more details):

- Because the Mesop python file is `main.py` , the module name is `main` .
- By convention, we do `import mesop as me` so the `me` refers to the main Mesop library module which is also a callable (e.g. a function) that conforms to WSGI.

requirements.txt

`requirements.txt` specifies the Python dependencies needed. You may need to add additional dependencies depending on your use case.

requirements.txt

```
mesop
Flask==3.0.0
gunicorn==20.1.0
Werkzeug==3.0.1
```

Pre-requisites

You will need to create a Google Cloud account and install the `gcloud` CLI.

Deploy to Google Cloud Run

In your terminal, go to the application directory, which has the files listed above.

Run the following command:

```
$ gcloud run deploy
```

Follow the instructions and then you should be able to access your deployed app.

Divider

Overview

Divider is used to provide visual separation and is based on the Angular Material divider component.

Examples


```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/divider",
)
def app():
  me.text(text="before")
  me.divider()
  me.text(text="after")
```

API

`func` `divider`

Creates a Divider component.

PARAMETER	DESCRIPTION
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`
`inset`	Whether the divider is an inset divider. **TYPE:** `bool` **DEFAULT:** `False`

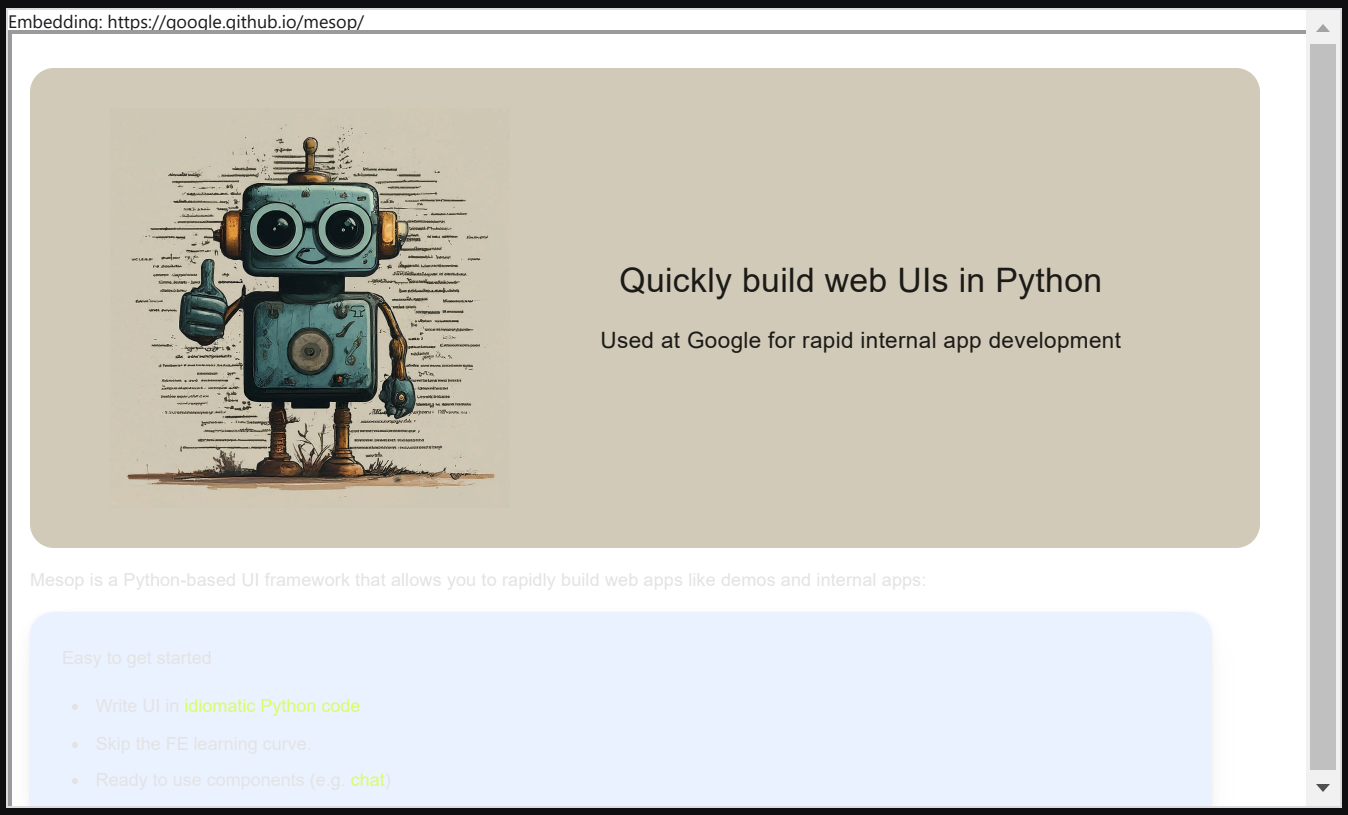
Embed

Overview

Embed allows you to embed/iframe another web site in your Mesop app.

Examples

Embedding: https://google.github.io/mesop/




```python
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/embed",
)
def app():
  src = "https://google.github.io/mesop/"
  me.text("Embedding: " + src)
  me.embed(
    src=src,
    style=me.Style(width="100%", height="100%"),
  )
```

API

`func` `embed`

This function creates an embed component.

PARAMETER	DESCRIPTION
`src`	The source URL for the embed content. **TYPE:** `str`
`style`	The style to apply to the embed, such as width and height. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

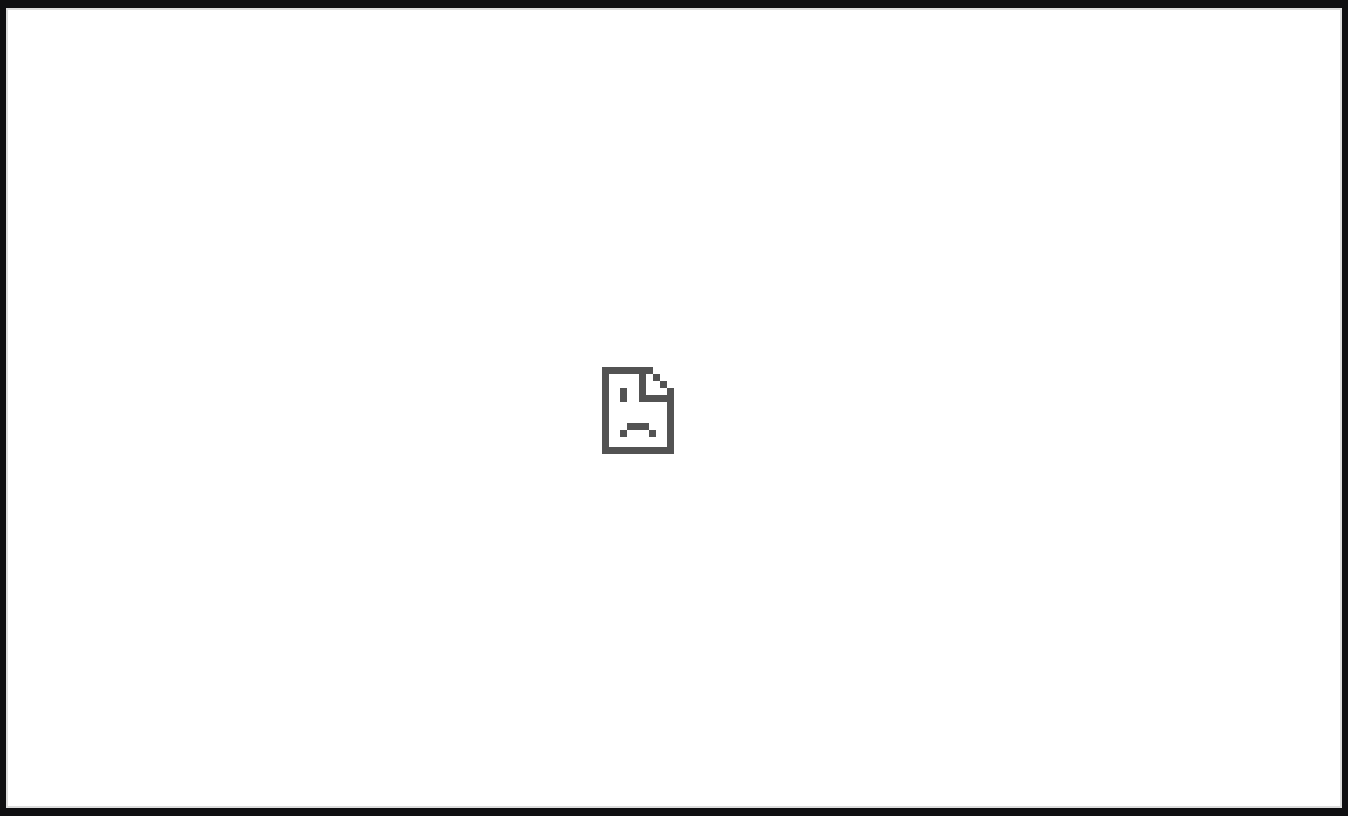
HTML

Overview

The HTML component allows you to add custom HTML to your Mesop app.

> Note: the HTML is sanitized by Angular for web security reasons so potentially unsafe code like JavaScript is removed.

Examples



```
import mesop as me


@me.page(path="/html_demo")
def app():
  me.html(
    """
Custom HTML
<a href="https://google.github.io/mesop/" target="_blank">mesop</a>
"""
  )
```

API

`func` `html`

This function renders custom HTML inside an iframe for web security isolation.

PARAMETER	DESCRIPTION
`html`	The HTML content to be rendered. **TYPE:** `str` **DEFAULT:** `''`
`style`	The style to apply to the embed, such as width and height. **TYPE:** `Style` | `None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str` | `None` **DEFAULT:** `None`

Icon

Overview

Icon displays a Material icon/symbol and is based on the Angular Material icon component.

Examples

home icon

```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/icon",
)
def app():
  me.text("home icon")
  me.icon(icon="home")
```

API

`func` `icon`

Creates a Icon component.

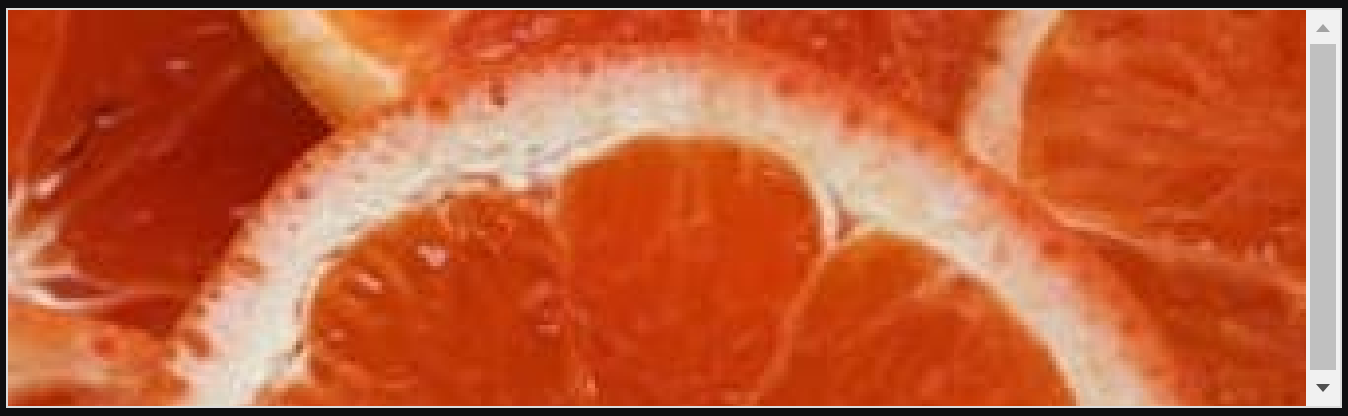
PARAMETER	DESCRIPTION
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`
`icon`	Name of the Material Symbols icon. **TYPE:** `str | None` **DEFAULT:** `None`
`style`	Inline styles **TYPE:** `Style | None` **DEFAULT:** `None`

Image

Overview

Image is the equivalent of an `<img>` HTML element.

Examples



```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/image",
)
def app():
  me.image(
    src="https://interactive-examples.mdn.mozilla.net/media/cc0-images/grapefruit-slice-332-332.jpg",
    alt="Grapefruit",
    style=me.Style(width="100%"),
  )
```

API

`func` `image`

This function creates an image component.

PARAMETER	DESCRIPTION
`src`	The source URL of the image. **TYPE:** `str` | `None` **DEFAULT:** `None`
`alt`	The alternative text for the image if it cannot be displayed. **TYPE:** `str` | `None` **DEFAULT:** `None`
`style`	The style to apply to the image, such as width and height. **TYPE:** `Style` | `None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str` | `None` **DEFAULT:** `None`

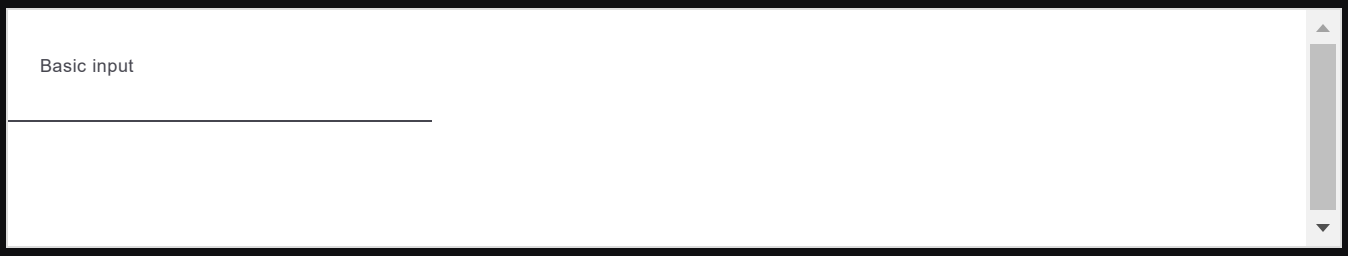
Input

Overview

Input allows the user to type in a value and is based on the Angular Material input component.

For longer text inputs, also see Textarea

Examples




```
import mesop as me


@me.stateclass
class State:
  input: str = ""


def on_input(e: me.InputEvent):
  state = me.state(State)
  state.input = e.value


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/input",
)
def app():
  s = me.state(State)
  me.input(label="Basic input", on_input=on_input)
  me.text(text=s.input)
```

API

`func` `input`

Creates a Input component.

PARAMETER	DESCRIPTION
`label`	Label for input. **TYPE:** `str` **DEFAULT:** `''`
`on_input`	input is a native browser event. **TYPE:** `Callable[[InputEvent], Any] | None` **DEFAULT:** `None`
`on_enter`	triggers when the browser detects an "Enter" key on a keyup native browser event. **TYPE:** `Callable[[EnterEvent], Any] | None` **DEFAULT:** `None`
`type`	Input type of the element. For textarea, use `me.Textarea(...)` **TYPE:** `Literal['color', 'date', 'datetime-local', 'email', 'month', 'number', 'password', 'search', 'tel', 'text', 'time', 'url', 'week'] | None` **DEFAULT:** `None`
`appearance`	The form field appearance style. **TYPE:** `Literal['fill', 'outline']` **DEFAULT:** `'fill'`
`style`	Style for input. **TYPE:** `Style | None` **DEFAULT:** `None`
`disabled`	Whether it's disabled. **TYPE:** `bool` **DEFAULT:** `False`

PARAMETER	DESCRIPTION
`placeholder`	Placeholder value **TYPE:** `str` **DEFAULT:** `''`
`required`	Whether it's required **TYPE:** `bool` **DEFAULT:** `False`
`value`	Initial value. **TYPE:** `str` **DEFAULT:** `''`
`readonly`	Whether the element is readonly. **TYPE:** `bool` **DEFAULT:** `False`
`hide_required_marker`	Whether the required marker should be hidden. **TYPE:** `bool` **DEFAULT:** `False`
`color`	The color palette for the form field. **TYPE:** `Literal['primary', 'accent', 'warn']` **DEFAULT:** `'primary'`
`float_label`	Whether the label should always float or float as the user types. **TYPE:** `Literal['always', 'auto']` **DEFAULT:** `'auto'`
`subscript_sizing`	Whether the form field should reserve space for one line of hint/error text (default) or to have the spacing grow from 0px as needed based on the size of the hint/error content. Note that when using dynamic sizing, layout shifts will occur when hint/error text changes. **TYPE:** `Literal['fixed', 'dynamic']` **DEFAULT:** `'fixed'`
`hint_label`	Text for the form field hint. **TYPE:** `str` **DEFAULT:** `''`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`func native_textarea`

Creates a browser native Textarea component. Intended for advanced use cases with maximum UI control.

PARAMETER	DESCRIPTION
`on_input`	input is a native browser event. **TYPE:** `Callable[[InputEvent], Any] | None` **DEFAULT:** `None`
`autosize`	If True, the textarea will automatically adjust its height to fit the content, up to the max_rows limit. **TYPE:** `bool` **DEFAULT:** `False`
`min_rows`	The minimum number of rows the textarea will display. **TYPE:** `int | None` **DEFAULT:** `None`
`max_rows`	The maximum number of rows the textarea will display. **TYPE:** `int | None` **DEFAULT:** `None`
`style`	Style for input. **TYPE:** `Style | None` **DEFAULT:** `None`
`disabled`	Whether it's disabled. **TYPE:** `bool` **DEFAULT:** `False`
`placeholder`	Placeholder value **TYPE:** `str` **DEFAULT:** `''`
`value`	Initial value. **TYPE:** `str` **DEFAULT:** `''`
`readonly`	Whether the element is readonly. **TYPE:** `bool` **DEFAULT:** `False`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Installing

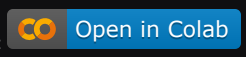
If you are familiar with setting up a Python environment, then run the following command in your terminal:

```
$ pip install mesop
```

If you're not familiar with setting up a Python environment, follow one of the options below.

A. Colab (Recommended for beginners)

Colab is a free hosted Jupyter notebook product provided by Google.

Try Mesop on Colab: 

B. Command-line

If you'd like to run Mesop locally on the command-line, follow these steps.

Pre-requisites: Make sure you have Python version 3.10 or later installed by running:

```
python --version
```

If you don't, please download Python.

Create a venv environment

1. **Open the terminal** and navigate to a directory: `cd foo`
2. **Create a virtual environment** by using venv, which will avoid Python environment issues. Run:

```
python -m venv .venv
```

3. **Activate your virtual environment:**
   - macOS and Linux:

     ```
     source .venv/bin/activate
     ```

   - Windows command prompt:

     ```
     .venv\Scripts\activate.bat
     ```

   - Windows PowerShell

     ```
     .venv\Scripts\Activate.ps1
     ```

Once you've activated the virtual environment, you will see ".venv" at the start of your terminal prompt.

4. **Install mesop:**

```
$ pip install mesop
```

Make sure your Python environment is setup correctly by running a hello world app.

Copy the following hello world code into a file `hello_world.py`:

hello_world.py

```
import mesop as me


@me.page()
def app():
  me.text("Hello World")
```

Then run the following command in your terminal:

```
$ mesop hello_world.py
```

Open the URL printed in the terminal (i.e. http://localhost:32123) in the browser to see your Mesop app loaded.

If you make changes to the code (e.g. change `"Hello World"` to `"Hi"`), the Mesop app should be automatically hot reloaded.

Interactivity

This guide continues from the Counter app example in Quickstart and explains advanced interactivity patterns for dealing with common use cases such as calling a slow blocking API call or a streaming API call.

Intermediate loading state

If you are calling a slow blocking API (e.g. several seconds) to provide a better user experience, you may want to introduce a custom loading indicator for a specific event.

> Note: Mesop has a built-in loading indicator at the top of the page for all events.

```
import time

import mesop as me


def slow_blocking_api_call():
  time.sleep(2)
  return "foo"


@me.stateclass
class State:
  data: str
  is_loading: bool


def button_click(event: me.ClickEvent):
  state = me.state(State)
  state.is_loading = True
  yield
  data = slow_blocking_api_call()
  state.data = data
  state.is_loading = False
  yield


@me.page(path="/loading")
def main():
  state = me.state(State)
  if state.is_loading:
    me.progress_spinner()
  me.text(state.data)
  me.button("Call API", on_click=button_click)
```

In this example, our event handler is a Python generator function. Each `yield` statement yields control back to the Mesop framework and executes a render loop which results in a UI update.

Before the first yield statement, we set `is_loading` to True on state so we can show a spinner while the user is waiting for the slow API call to complete.

Before the second (and final) yield statement, we set `is_loading` to False, so we can hide the spinner and then we add the result of the API call to state so we can display that to the user.

> Tip: you must have a yield statement as the last line of a generator event handler function. Otherwise, any code after the final yield will not be executed.

Streaming

This example builds off the previous Loading example and makes our event handler a generator function so we can incrementally update the UI.

```
from time import sleep

import mesop as me


def generate_str():
  yield "foo"
  sleep(1)
  yield "bar"


@me.stateclass
class State:
  string: str = ""


def button_click(action: me.ClickEvent):
  state = me.state(State)
  for val in generate_str():
    state.string += val
    yield


@me.page(path="/streaming")
```

```
def main():
  state = me.state(State)
  me.button("click", on_click=button_click)
  me.text(text=f"{state.string}")
```

Troubleshooting

User input race condition

If you notice a race condition with user input (e.g. input or textarea) where sometimes the last few characters typed by the user is lost, you are probably unnecessarily setting the value of the component.

See the following example using this **anti-pattern** ⚠️:

Bad example

```
@me.stateclass
class State:
  input_value: str

def app():
  state = me.state(State)
  me.input(value=state.input_value, on_input=on_input)

def on_input(event: me.InputEvent):
  state = me.state(State)
  state.input_value = event.value
```

The problem is that the input value now has a race condition because it's being set by two sources:

1. The server is setting the input value based on state.
2. The client is setting the input value based on what the user is typing.

The way to fix this is by *not* setting the input value from the server.

The above example **corrected** would look like this ✅:

Good example

```
@me.stateclass
class State:
  input_value: str

def app():
  state = me.state(State)
  me.input(on_input=on_input)

def on_input(event: me.InputEvent):
  state = me.state(State)
  state.input_value = event.value
```

Avoid using closure variables in event handler

One subtle mistake when building a reusable component is to have the event handler use a closure variable like the following example:

Bad example of using closure variable

```
@me.component
def link_component(url: str):
   def on_click(event: me.ClickEvent):
     me.navigate(url)
  return me.button(url, on_click=on_click)

def app():
    link_component("/1")
    link_component("/2")
```

The problem with this above example is that Mesop only stores the last event handler. This means that both instances of the link_component will refer to the last `on_click` instance which references the same `url` closure variable set to `"/2"`. This almost always produces the wrong behavior.

Instead, you will want to use the pattern of relying on the key in the event handler as demonstrated in the following example:

Good example of using key

```
@me.component
def link_component(url: str):
   def on_click(event: me.ClickEvent):
     me.navigate(event.key)
  return me.button(url, key=url, on_click=on_click)
```

For more info on using component keys, please refer to the Component Key docs.

Labs

Mesop Labs is built on top of the core Mesop framework and may evolve in the future.

Using Labs

You will need to add an import statement to use labs:

```
import mesop.labs as mel
```

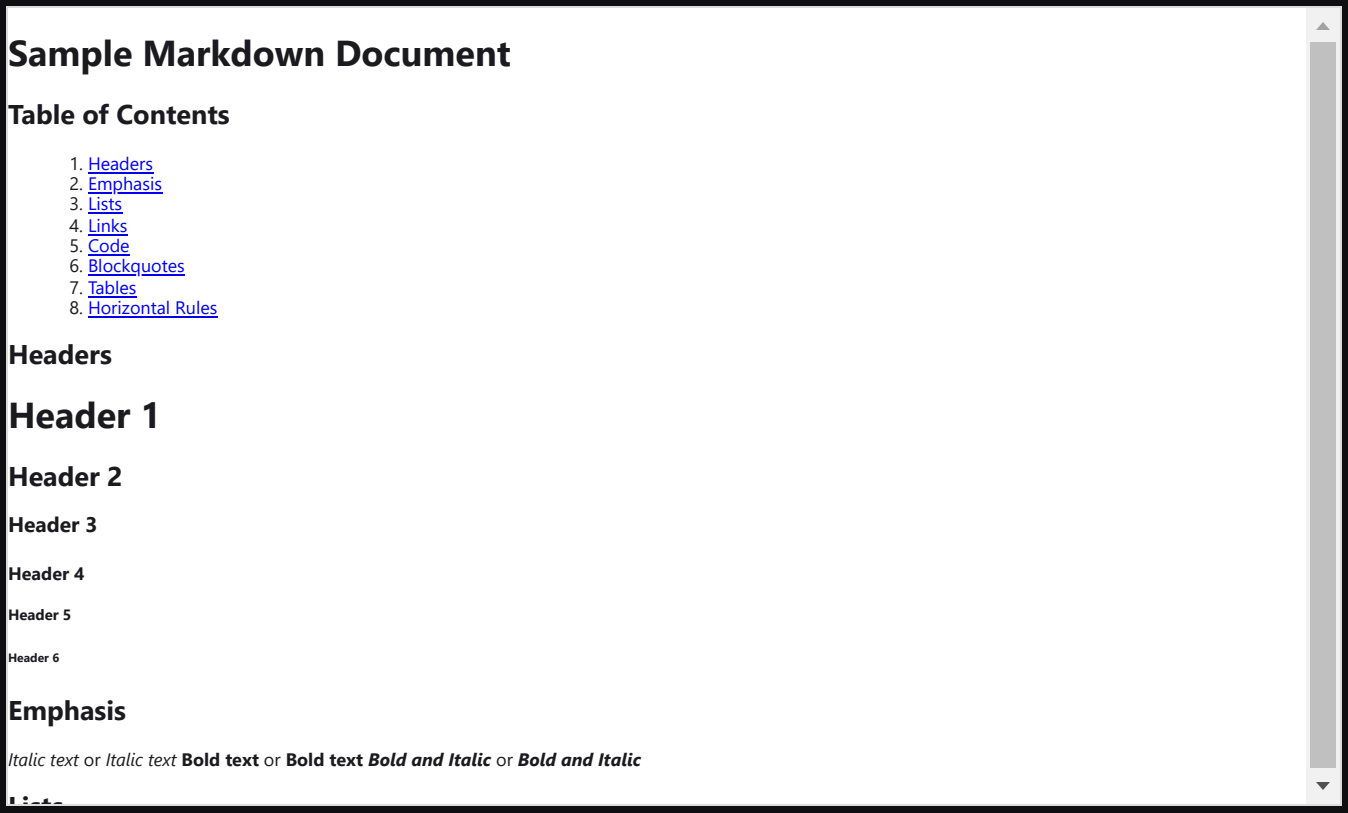
The code inside Mesop Labs is intended to be simple to understand so you can copy and customize it as needed.

Markdown

Overview

Markdown is used to render markdown text.

Examples



Sample Markdown Document

Table of Contents

1. Headers
2. Emphasis
3. Lists
4. Links
5. Code
6. Blockquotes
7. Tables
8. Horizontal Rules

Headers

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

Emphasis

Italic text or *Italic text* **Bold text** or **Bold text** ***Bold and Italic*** or ***Bold and Italic***

Lists

```
import mesop as me

SAMPLE_MARKDOWN = """
# Sample Markdown Document

## Table of Contents
1. [Headers](#headers)
2. [Emphasis](#emphasis)
3. [Lists](#lists)
4. [Links](#links)
5. [Code](#code)
6. [Blockquotes](#blockquotes)
7. [Tables](#tables)
8. [Horizontal Rules](#horizontal-rules)

## Headers
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6

## Emphasis
*Italic text* or _Italic text_
**Bold text** or __Bold text__
***Bold and Italic*** or ___Bold and Italic___

## Lists

### Unordered List
- Item 1
- Item 2
  - Subitem 2.1
  - Subitem 2.2

### Ordered List
1. First item
2. Second item
   1. Subitem 2.1
   2. Subitem 2.2

## Links
```

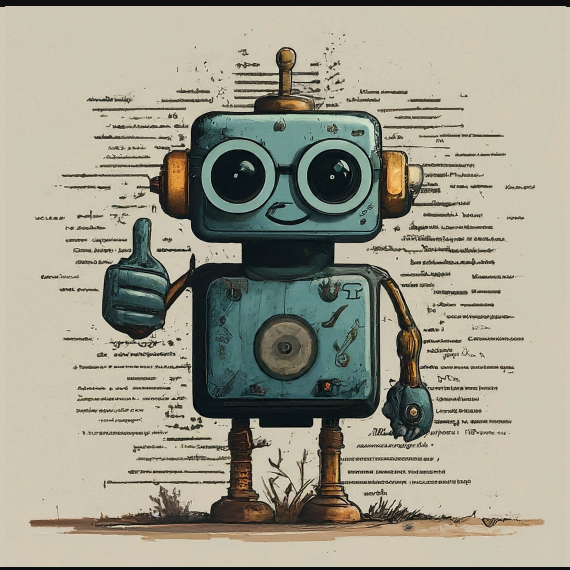
```
[Google](https://www.google.com/)

## Code
Inline `code`
"""


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/markdown_demo",
)
def app():
  me.markdown(SAMPLE_MARKDOWN)
```

API

`func` markdown

This function creates a markdown.

PARAMETER	DESCRIPTION
`text`	**Required.** Markdown text **TYPE:** `str` | `None` **DEFAULT:** `None`
`style`	Style to apply to component. Follows HTML Element inline style API. **TYPE:** `Style` | `None` **DEFAULT:** `None`



Quickly build web UIs in Python

Used at Google for rapid internal app development

Mesop is a Python-based UI framework that allows you to rapidly build web apps like demos and internal apps:

Easy to get started

- Write UI in idiomatic Python code
- Skip the FE learning curve.
- Ready to use components (e.g. chat)

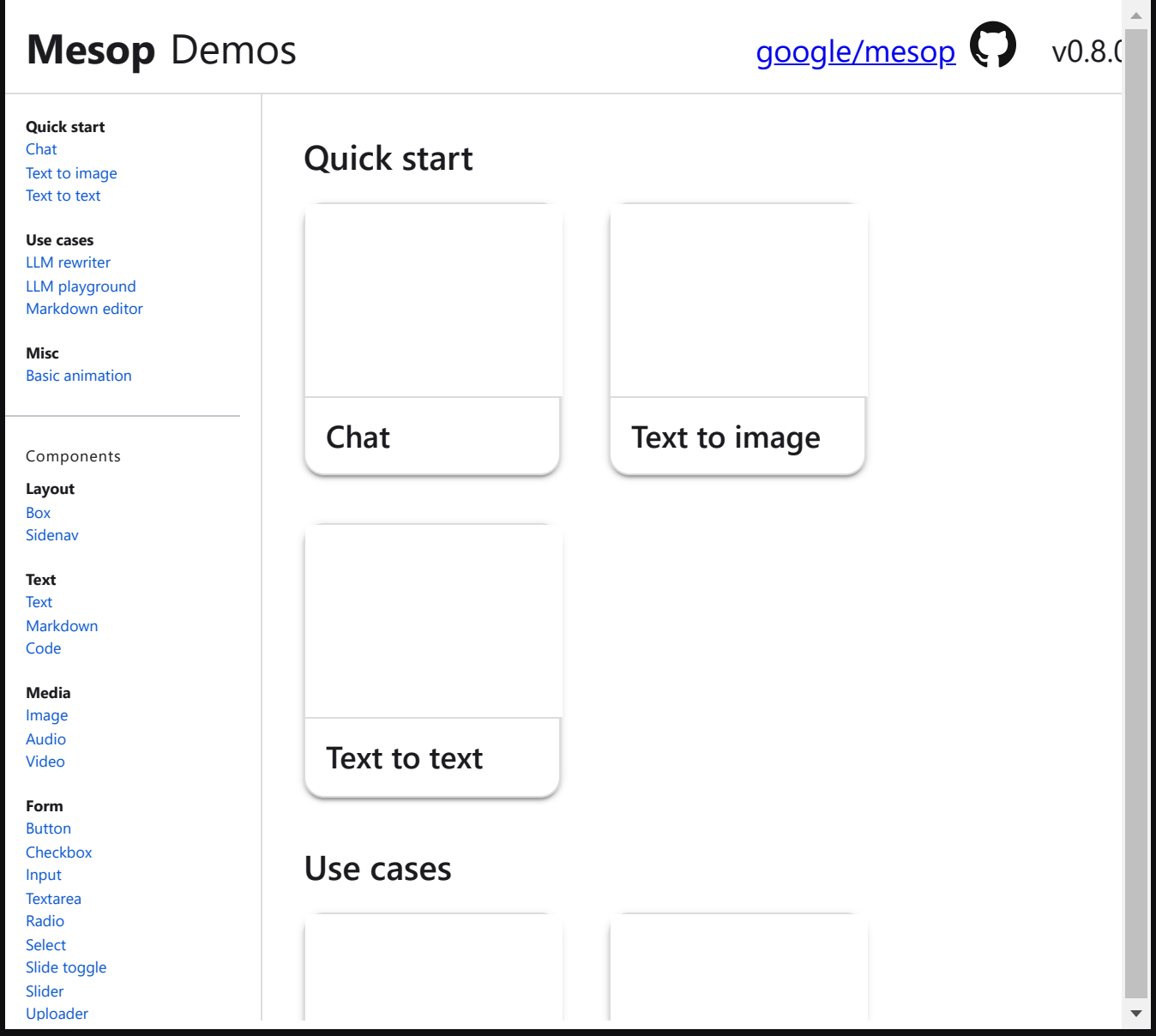
Fast iteration

- **Hot reload** so the browser automatically reloads and preserves state
- Rich IDE support with strong type safety

Flexible & composable

- Build custom UIs *without* writing Javascript/CSS/HTML
- Compose your UI into components, which are just Python functions

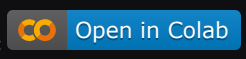
See what you can build in less than 10 lines of code...



Check out how the above demo gallery was built in pure Mesop!

Try it

Colab

Try Mesop on Colab: 

Locally

Step 1: Install it

```
$ pip install mesop
```

Step 2: Copy the example above into `main.py`

Step 3: Run the app

```
$ mesop main.py
```

Next Steps

Learn more in Getting Started.

Disclaimer

This is not an officially supported Google product.

Multi-Pages

You can define multi-page Mesop applications by using the page feature you learned from Quickstart.

Multi-page setup

```
import mesop as me

@me.page(path="/1")
def page1():
    me.text("page 1")

@me.page(path="/2")
def page2():
    me.text("page 2")
```

Learn more about page configuration in the page API doc.

Navigation

If you have multiple pages, you will typically want to navigate from one page to another when the user clicks a button. You can use `me.navigate("/to/path")` to navigate to another page.

Example:

```
import mesop as me


def on_click(e: me.ClickEvent):
  state = me.state(State)
  state.count += 1
  me.navigate("/multi_page_nav/page_2")


@me.page(path="/multi_page_nav")
def main_page():
  me.button("Navigate to Page 2", on_click=on_click)


@me.page(path="/multi_page_nav/page_2")
def page_2():
  state = me.state(State)
  me.text(f"Page 2 - count: {state.count}")


@me.stateclass
class State:
  count: int
```

> Note: you can re-use state across pages. See how the above example uses the `State#count` value across pages.

Navigate

To navigate to another page, you can use `me.navigate`. This is particularly useful for navigating across a multi-page app.

Example

```
import time

import mesop as me


def on_load(e: me.LoadEvent):
  state = me.state(State)
  state.default_values.append("a")
  yield
  time.sleep(1)
  state.default_values.append("b")
  yield


@me.page(path="/docs/on_load_generator", on_load=on_load)
def app():
  me.text("onload")
  me.text(str(me.state(State).default_values))


@me.stateclass
class State:
  default_values: list[str]
```

API

`func navigate`

Navigates to the given URL.

PARAMETER	DESCRIPTION
`url`	The URL to navigate to. **TYPE:** `str`

Components

Please read Quickstart before this as it explains the basics of components. This page provides an overview of the different types of components in Mesop.

Types of components

Native components

Native components are components implemented using Angular/Javascript. Many of these components wrap Angular Material components. Other components are simple wrappers around DOM elements.

If you have a use case that's not supported by the existing native components, please file an issue on GitHub to explain your use case. Given our limited bandwidth, we may not be able to build it soon, but in the future, we will enable Mesop developers to build their own custom native components.

User-defined components

User-defined components are essentially Python functions which call other components, which can be native components or other user-defined components. It's very easy to write your own components, and it's encouraged to split your app into modular components for better maintainability and reusability.

Web components

Web components in Mesop are custom HTML elements created using JavaScript and CSS. They enable custom JavaScript execution and bi-directional communication between the browser and server. They can wrap JavaScript libraries and provide stateful client-side interactions. Learn more about web components.

Content components

Content components allow you to compose components more flexibly than regular components by accepting child(ren) components. A commonly used content component is the button component, which accepts a child component which oftentimes the text component.

Example:

```
with me.button():
  me.text("Child")
```

You can also have multiple content components nested:

```
with me.box():
  with me.box():
    me.text("Grand-child")
```

Sometimes, you may want to define your own content component for better reusability. For example, let's say I want to define a scaffold component which includes a menu positioned on the left and a main content area, I could do the following:

```
@me.content_component
def scaffold(url: str):
  with me.box(style="background: white"):
    menu(url=url)
    with me.box(style=f"padding-left: {MENU_WIDTH}px"):
      me.slot()
```

Now other components can re-use this scaffold component:

```
def page1():
  with scaffold(url="/page1"):
    some_content(...)
```

This is similar to Angular's Content Projection.

Component Key

Every native component in Mesop accepts a `key` argument which is a component identifier. This is used by Mesop to tell Angular whether to reuse the DOM element.

Resetting a component

You can reset a component to the initial state (e.g. reset a select component to the unselected state) by giving it a new key value across renders.

For example, you can reset a component by "incrementing" the key:

```
class State:
  select_menu_key: int

def reset(event):
  state = me.state(State)
```

```
  state.select_menu_key += 1

def main():
  state = me.state(State)
  me.select(key=str(state.select_menu_key),
            options=[me.SelectOption(label="o1", value="o1")])
  me.button(label="Reset", on_click=reset)
```

Event handlers

Every Mesop event includes the key of the component which emitted the event. This makes it useful when you want to reuse an event handler for multiple instances of a component:

```
def buttons():
  for fruit in ["Apple", "Banana"]:
    me.button(fruit, key=fruit, on_click=on_click)

def on_click(event: me.ClickEvent):
  fruit = me.key
  print("fruit name", fruit)
```

Because a key is a `str` type, you may sometimes want to store more complex data like a dataclass or a proto object for retrieval in the event handler. To do this, you can serialize and deserialize:

```
import json
from dataclasses import dataclass

@dataclass
class Person:
  name: str

def buttons():
  for person in [Person(name="Alice"), Person(name="Bob")]:
    # serialize dataclass into str
    key = json.dumps(person.asdict())
    me.button(person.name, key=key, on_click=on_click)

def on_click(event: me.ClickEvent):
  person_dict = json.loads(me.key)
  # modify this for more complex deserialization
  person = Person(**person_dict)
```

Use component key for reusable event handler

This avoids a subtle issue with using closure variables in event handlers.

Page API

Overview

Pages allow you to build multi-page applications by decorating Python functions with `me.page`. To learn more, read the see multi-pages guide.

Examples

Simple, 1-page setup

To create a simple Mesop app, you can use `me.page()` like this:

```
import mesop as me

@me.page()
def foo():
    me.text("bar")
```

> NOTE: If you do not provide a `path` argument, then it defaults to the root path `"/"`.

Explicit 1-page setup

This is the same as the above example which explicitly sets the route to `"/"`.

```
import mesop as me

@me.page(path="/")
def foo():
    me.text("bar")
```

API

`func` page

Defines a page in a Mesop application.

This function is used as a decorator to register a function as a page in a Mesop app.

PARAMETER	DESCRIPTION
`path`	The URL path for the page. Defaults to "/". **TYPE:** `str` **DEFAULT:** `'/'`
`title`	The title of the page. If None, a default title is generated. **TYPE:** `str | None` **DEFAULT:** `None`
`stylesheets`	List of stylesheet URLs to load. **TYPE:** `list[str] | None` **DEFAULT:** `None`
`security_policy`	The security policy for the page. If None, a default strict security policy is used. **TYPE:** `SecurityPolicy | None` **DEFAULT:** `None`
`on_load`	An optional event handler to be called when the page is loaded. **TYPE:** `OnLoadHandler | None` **DEFAULT:** `None`

RETURNS	DESCRIPTION
`Callable[[Callable[[], None]], Callable[[], None]]`	A decorator that registers the decorated function as a page.

`class` LoadEvent `dataclass`

Represents a page load event.

ATTRIBUTE	DESCRIPTION
`path`	The path loaded **TYPE:** `str`

`on_load`

You may want to do some sort of data-processing when a page is first loaded in a session.

Simple handler

An `on_load` handler is similar to a regular event handler where you can mutate state.

```
import time

import mesop as me


def fake_api():
  yield 1
  time.sleep(1)
  yield 2
  time.sleep(2)
  yield 3


def on_load(e: me.LoadEvent):
  for val in fake_api():
    me.state(State).default_values.append(val)
    yield


@me.page(path="/docs/on_load", on_load=on_load)
def app():
  me.text("onload")
  me.text(str(me.state(State).default_values))


@me.stateclass
class State:
  default_values: list[int]
```

Generator handler

The `on_load` handler can also be a generator function. This is useful if you need to call a slow or streaming API and want to return intermediate results before all the data has been received.

```
import time

import mesop as me


def on_load(e: me.LoadEvent):
  state = me.state(State)
  state.default_values.append("a")
  yield
  time.sleep(1)
  state.default_values.append("b")
  yield


@me.page(path="/docs/on_load_generator", on_load=on_load)
def app():
  me.text("onload")
  me.text(str(me.state(State).default_values))


@me.stateclass
class State:
  default_values: list[str]
```

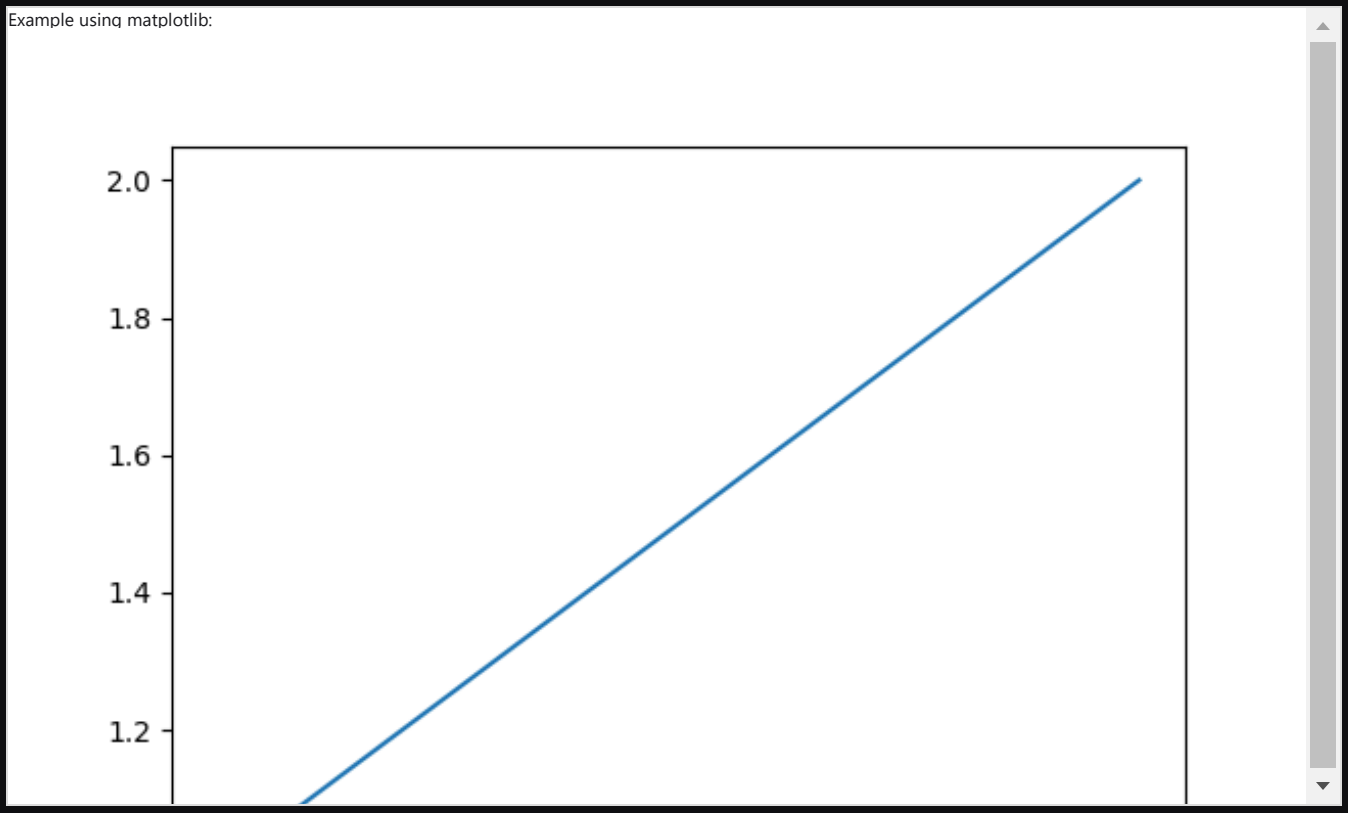
Plot

Overview

Plot provides a convenient way to render Matplotlib figures as an image.

Examples

Example using matplotlib:




```
from matplotlib.figure import Figure

import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/plot",
)
def app():
  # Create matplotlib figure without using pyplot:
  fig = Figure()
  ax = fig.subplots()  # type: ignore
  ax.plot([1, 2])  # type: ignore

  me.text("Example using matplotlib:")
  me.plot(fig, style=me.Style(width="100%"))
```

API

`func` `plot`

Creates a plot component from a Matplotlib figure.

PARAMETER	DESCRIPTION
`figure`	A Matplotlib figure which will be rendered. **TYPE:** `Figure`
`style`	An optional Style object that defines the visual styling for the plot component. If None, default styling (e.g. height, width) is used. **TYPE:** `Style` | `None` **DEFAULT:** `None`

Progress bar

Overview

Progress Bar is used to indicate something is in progress and is based on the Angular Material progress bar component.

Examples


```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/progress_bar",
)
def app():
  me.text("Default progress bar")
  me.progress_bar()
```

API

`func` progress_bar

Creates a Progress bar component.

PARAMETER	DESCRIPTION
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`
`color`	Theme palette color of the progress bar. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`value`	Value of the progress bar. Defaults to zero. Mirrored to aria-valuenow. **TYPE:** `float` **DEFAULT:** `0`
`buffer_value`	Buffer value of the progress bar. Defaults to zero. **TYPE:** `float` **DEFAULT:** `0`
`mode`	Mode of the progress bar. Input must be one of these values: determinate, indeterminate, buffer, query, defaults to 'determinate'. Mirrored to mode attribute. **TYPE:** `Literal['determinate', 'indeterminate', 'buffer', 'query']` **DEFAULT:** `'indeterminate'`
`on_animation_end`	Event emitted when animation of the primary progress bar completes. This event will not be emitted when animations are disabled, nor will it be emitted for modes with continuous animations (indeterminate and query). **TYPE:** `Callable[[ProgressBarAnimationEndEvent], Any] | None` **DEFAULT:** `None`

`class` ProgressBarAnimationEndEvent `dataclass`

Bases: `MesopEvent`

Event emitted when the animation of the progress bar ends.

ATTRIBUTE	DESCRIPTION
`value`	The value of the progress bar when the animation ends. **TYPE:** `float`
`key`	Key of the component that emitted this event. **TYPE:** `str`

Progress spinner

Overview

Progress Spinner is used to indicate something is in progress and is based on the Angular Material progress spinner component.

Examples

```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/progress_spinner",
)
def app():
  me.progress_spinner()
```

API

`func` `progress_spinner`

Creates a Progress spinner component.

PARAMETER	DESCRIPTION
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`
`color`	Theme palette color of the progress spinner. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`diameter`	The diameter of the progress spinner (will set width and height of svg). **TYPE:** `float` **DEFAULT:** `48`
`stroke_width`	Stroke width of the progress spinner. **TYPE:** `float` **DEFAULT:** `4`

Quickstart

Let's build a simple interactive Mesop app.

Before you start

Make sure you've installed Mesop, otherwise please follow the Installing Guide.

Text to text app

The simplest way to get started with Mesop is to use the `text_to_text` component

```
import mesop as me
import mesop.labs as mel


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/text_to_text",
  title="Text to Text Example",
)
def app():
  mel.text_to_text(
    upper_case_stream,
    title="Text to Text Example",
  )


def upper_case_stream(s: str):
  return "Echo: " + s
```

The rest of this guide will show you step-by-step how something like text_to_text is implemented.

Hello World app

Let's start by creating a simple Hello World app in Mesop:

```
import mesop as me


@me.page(path="/hello_world")
def app():
  me.text("Hello World")
```

This simple example demonstrates a few things:

- Every Mesop app starts with `import mesop as me`. This is the only recommended way to import mesop, otherwise your app may break in the future because you may be relying on internal implementation details.
- `@me.page` is a function decorator which makes a function a *root component* for a particular path. If you omit the `path` parameter, this is the equivalent of doing `@me.page(path="/")`.
- `app` is a Python function that we will call a **component** because it's creating Mesop components in the body.

Components

Components are the building blocks of a Mesop application. A Mesop application is essentially a tree of components.

Let's explain the different kinds of components in Mesop:

- Mesop comes built-in with **native** components. These are components implemented using Angular/Javascript. Many of these components wrap Angular Material components.
- You can also create your own components which are called **user-defined** components. These are essentially Python functions like `app` in the previous example.

Counter app

Let's build a more complex app to demonstrate Mesop's interactivity features.

```
import mesop as me


@me.stateclass
class State:
  clicks: int
```

```
def button_click(event: me.ClickEvent):
  state = me.state(State)
  state.clicks += 1


@me.page(path="/counter")
def main():
  state = me.state(State)
  me.text(f"Clicks: {state.clicks}")
  me.button("Increment", on_click=button_click)
```

This app allows the user to click on a button and increment a counter, which is shown to the user as "Clicks: #".

Let's walk through this step-by-step.

State

The `State` class represents the application state for a particular browser session. This means every user session has its own instance of `State`.

`@me.stateclass` is a class decorator which is similar to Python's dataclass but also sets default value based on type hints and allows Mesop to inject the class as shown next.

> Note: Everything in a state class must be serializable because it's sent between the server and browser.

Event handler

The `button_click` function is an event handler. An event handler has a single parameter, `event`, which can contain a value (this will be shown in the next example). An event handler is responsible for updating state based on the incoming event.

`me.state(State)` retrieves the instance of the state class for the current session.

Component

Like the previous example, `main` is a Mesop component function which is decorated with `page` to mark it as a root component for a path.

Similar to the event handler, we can retrieve the state in a component function by calling `me.state(State)`.

> Note: it's *not* safe to mutate state inside a component function. All mutations must be done in an event handler.

Rendering dynamic values in Mesop is simple because you can do standard Python string interpolation use f-strings:

```
me.text(f"Clicks: {state.clicks}")
```

The button component demonstrates connecting an event handler to a component. Whenever a click event is triggered by the component, the registered event handler function is called.:

```
me.button("Increment", on_click=button_click)
```

In summary, you've learned how to define a state class, an event handler and wire them together using interactive components.

What's next

At this point, you've learned all the basics of building a Mesop app and now you should be able to understand how Text to Text is implemented under the hood.

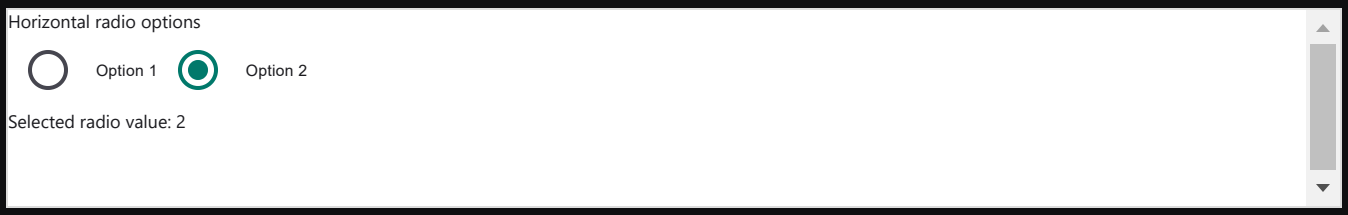
To learn more about Mesop, I recommend reading the Guides and then spend time looking at the examples on GitHub. As you build your own applications, you'll want to reference the Components docs.

Radio

Overview

Radio is a single selection form control based on the Angular Material radio component.

Examples




```
import mesop as me


@me.stateclass
class State:
  radio_value: str = "2"


def on_change(event: me.RadioChangeEvent):
  s = me.state(State)
  s.radio_value = event.value


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/radio",
)
def app():
  s = me.state(State)
  me.text("Horizontal radio options")
  me.radio(
    on_change=on_change,
    options=[
      me.RadioOption(label="Option 1", value="1"),
      me.RadioOption(label="Option 2", value="2"),
    ],
    value=s.radio_value,
  )
  me.text(text="Selected radio value: " + s.radio_value)
```

API

`func` `radio`

Creates a Radio component.

PARAMETER	DESCRIPTION
`options`	List of radio options **TYPE:** `Iterable[RadioOption]` **DEFAULT:** `()`
`on_change`	Event emitted when the group value changes. Change events are only emitted when the value changes due to user interaction with a radio button (the same behavior as `<input type-"radio">`). **TYPE:** `Callable[[RadioChangeEvent], Any] | None` **DEFAULT:** `None`
`color`	Theme color for all of the radio buttons in the group. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`label_position`	Whether the labels should appear after or before the radio-buttons. Defaults to 'after' **TYPE:** `Literal['before', 'after']` **DEFAULT:** `'after'`
`value`	Value for the radio-group. Should equal the value of the selected radio button if there is a corresponding radio button with a matching value. **TYPE:** `str` **DEFAULT:** `''`
`disabled`	Whether the radio group is disabled. **TYPE:** `bool` **DEFAULT:** `False`

PARAMETER	DESCRIPTION
`style`	Style for the component. **TYPE:** `Style` | `None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str` | `None` **DEFAULT:** `None`

class `RadioOption` dataclass

ATTRIBUTE	DESCRIPTION
`label`	Content to show for the radio option **TYPE:** `str` | `None`
`value`	The value of this radio button. **TYPE:** `str` | `None`

class `RadioChangeEvent` dataclass

Bases: `MesopEvent`

Event representing a change in the radio component's value.

ATTRIBUTE	DESCRIPTION
`value`	The new value of the radio component after the change. **TYPE:** `str`
`key`	key of the component that emitted this event. **TYPE:** `str`

Scroll into view

If you want to scroll a component into the viewport, you can use `me.scroll_into_view` which scrolls the component with the specified key into the viewport.

Example

```
import time

import mesop as me


@me.stateclass
class State:
  more_lines: int = 0


@me.page(path="/scroll_into_view")
def app():
  me.button("Scroll to middle line", on_click=scroll_to_middle)
  me.button("Scroll to bottom line", on_click=scroll_to_bottom)
  me.button(
    "Scroll to bottom line & generate lines",
    on_click=scroll_to_bottom_and_generate_lines,
  )
  for _ in range(100):
    me.text("Filler line")
  me.text("middle_line", key="middle_line")
  for _ in range(100):
    me.text("Filler line")
  me.text("bottom_line", key="bottom_line")
  for _ in range(me.state(State).more_lines):
    me.text("More lines")


def scroll_to_middle(e: me.ClickEvent):
  me.scroll_into_view(key="middle_line")


def scroll_to_bottom(e: me.ClickEvent):
  me.scroll_into_view(key="bottom_line")


def scroll_to_bottom_and_generate_lines(e: me.ClickEvent):
  state = me.state(State)
  me.scroll_into_view(key="bottom_line")
  yield
  state.more_lines += 5
  time.sleep(1)
  yield
  state.more_lines += 5
  time.sleep(1)
  yield
  state.more_lines += 5
  time.sleep(1)
  yield
  state.more_lines += 5
  time.sleep(1)
  yield
```

API

`func` `scroll_into_view`

Scrolls so the component specified by the key is in the viewport.

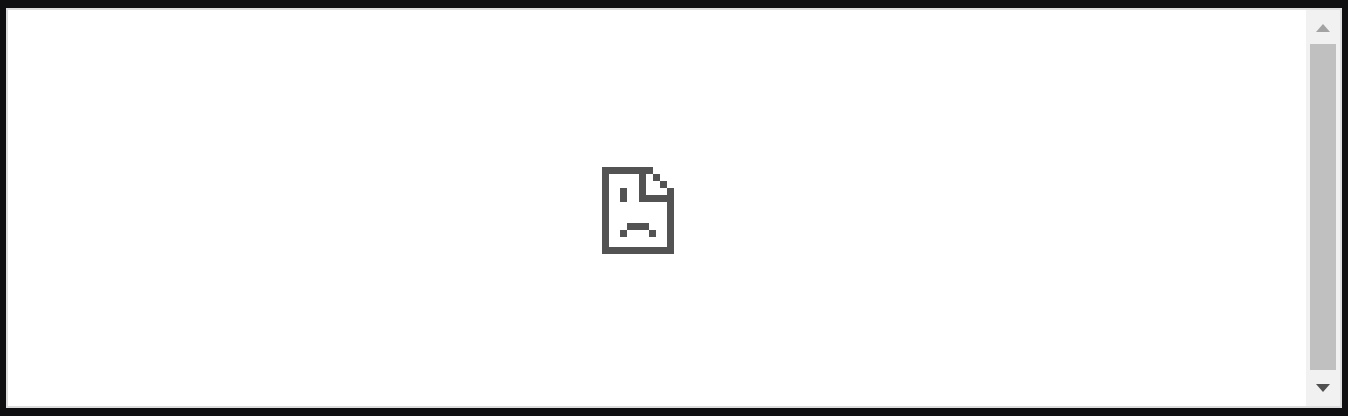
PARAMETER	DESCRIPTION
`key`	The unique identifier of the component to scroll to. This key should be globally unique to prevent unexpected behavior. If multiple components share the same key, the first component instance found in the component tree will be scrolled to. **TYPE:** `str`

Select

Overview

Select allows the user to choose from a list of values and is based on the Angular Material select component.

Examples



API

`func` `select`

Creates a Select component.

PARAMETER	DESCRIPTION
`options`	List of select options. **TYPE:** `Iterable[SelectOption]` **DEFAULT:** `()`
`on_selection_change`	Event emitted when the selected value has been changed by the user. **TYPE:** `Callable[[SelectSelectionChangeEvent], Any] | None` **DEFAULT:** `None`
`on_opened_change`	Event emitted when the select panel has been toggled. **TYPE:** `Callable[[SelectOpenedChangeEvent], Any] | None` **DEFAULT:** `None`
`disabled`	Whether the select is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`disable_ripple`	Whether ripples in the select are disabled. **TYPE:** `bool` **DEFAULT:** `False`
`multiple`	Whether multiple selections are allowed. **TYPE:** `bool` **DEFAULT:** `False`
`tab_index`	Tab index of the select. **TYPE:** `int` **DEFAULT:** `0`
`placeholder`	Placeholder to be shown if no value has been selected. **TYPE:** `str` **DEFAULT:** `''`
`value`	Value of the select control. **TYPE:** `str` **DEFAULT:** `''`
`style`	Style. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

class SelectOption dataclass

Represents an option within a select component.

ATTRIBUTE	DESCRIPTION
label	The content shown for the select option. **TYPE:** str | None
value	The value associated with the select option. **TYPE:** str | None

class SelectSelectionChangeEvent dataclass

Bases: MesopEvent

Event representing a change in the select component's value(s).

ATTRIBUTE	DESCRIPTION
values	New values of the select component after the change. **TYPE:** list[str]
key	Key of the component that emitted this event. **TYPE:** str

attr **value** property

Shortcut for returning a single value.

class SelectOpenedChangeEvent dataclass

Bases: MesopEvent

Event representing the opened state change of the select component.

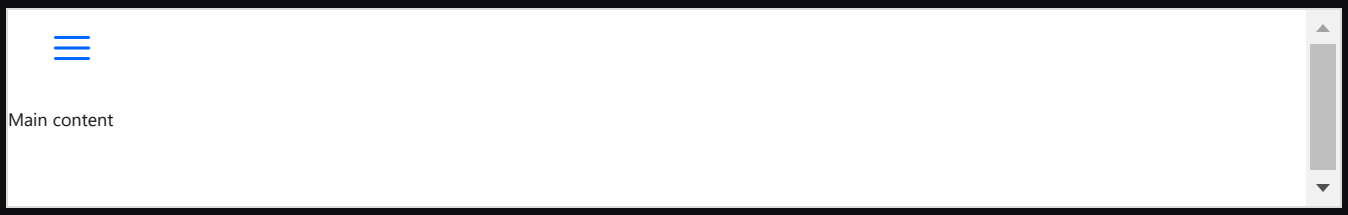
ATTRIBUTE	DESCRIPTION
opened	A boolean indicating whether the select component is opened (True) or closed (False). **TYPE:** bool
key	key of the component that emitted this event. **TYPE:** str

Sidenav

Overview

Sidenav is a sidebar typically used for navigation and is based on the Angular Material sidenav component.

Examples




```
import mesop as me


@me.stateclass
class State:
  sidenav_open: bool


def on_click(e: me.ClickEvent):
  s = me.state(State)
  s.sidenav_open = not s.sidenav_open


SIDENAV_WIDTH = 200


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/sidenav",
)
def app():
  state = me.state(State)
  with me.sidenav(
    opened=state.sidenav_open, style=me.Style(width=SIDENAV_WIDTH)
  ):
    me.text("Inside sidenav")

  with me.box(
    style=me.Style(
      margin=me.Margin(left=SIDENAV_WIDTH if state.sidenav_open else 0),
    ),
  ):
    with me.content_button(on_click=on_click):
      me.icon("menu")
    me.markdown("Main content")
```

API

`func` `sidenav`

This function creates a sidenav.

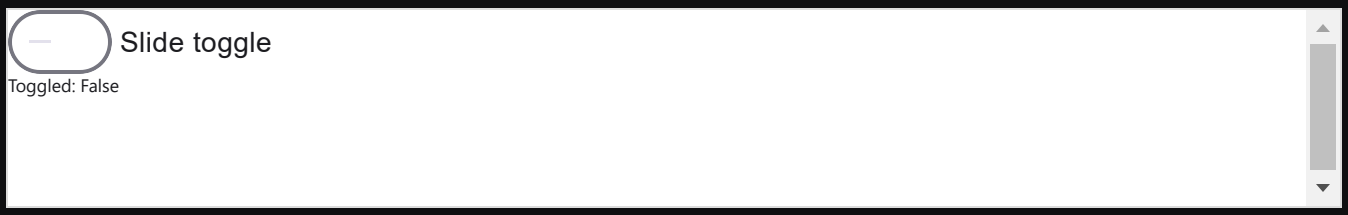
PARAMETER	DESCRIPTION
`opened`	A flag to determine if the sidenav is open or closed. Defaults to True. **TYPE:** `bool` **DEFAULT:** `True`
`style`	An optional Style object to apply custom styles. Defaults to None. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Slide toggle

Overview

Slide Toggle allows the user to toggle on and off and is based on the Angular Material slide toggle component.

Examples




```
import mesop as me


@me.stateclass
class State:
  toggled: bool = False


def on_change(event: me.SlideToggleChangeEvent):
  s = me.state(State)
  s.toggled = not s.toggled


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/slide_toggle",
)
def app():
  me.slide_toggle(label="Slide toggle", on_change=on_change)
  s = me.state(State)
  me.text(text=f"Toggled: {s.toggled}")
```

API

`func` `slide_toggle`

Creates a simple Slide toggle component with a text label.

PARAMETER	DESCRIPTION
`label`	Text label for slide toggle **TYPE:** `str | None` **DEFAULT:** `None`
`on_change`	An event will be dispatched each time the slide-toggle changes its value. **TYPE:** `Callable[[SlideToggleChangeEvent], Any] | None` **DEFAULT:** `None`
`label_position`	Whether the label should appear after or before the slide-toggle. Defaults to 'after'. **TYPE:** `Literal['before', 'after']` **DEFAULT:** `'after'`
`required`	Whether the slide-toggle is required. **TYPE:** `bool` **DEFAULT:** `False`
`color`	Palette color of slide toggle. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`disabled`	Whether the slide toggle is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`disable_ripple`	Whether the slide toggle has a ripple. **TYPE:** `bool` **DEFAULT:** `False`
`tab_index`	Tabindex of slide toggle. **TYPE:** `int` **DEFAULT:** `0`

PARAMETER	DESCRIPTION
`checked`	Whether the slide-toggle element is checked or not. **TYPE:** `bool` **DEFAULT:** `False`
`hide_icon`	Whether to hide the icon inside of the slide toggle. **TYPE:** `bool` **DEFAULT:** `False`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`func` `content_slide_toggle`

Creates a Slide toggle component which is a composite component. Typically, you would use a text or icon component as a child.

Intended for advanced use cases.

PARAMETER	DESCRIPTION
`on_change`	An event will be dispatched each time the slide-toggle changes its value. **TYPE:** `Callable[[SlideToggleChangeEvent], Any] | None` **DEFAULT:** `None`
`label_position`	Whether the label should appear after or before the slide-toggle. Defaults to 'after'. **TYPE:** `Literal['before', 'after']` **DEFAULT:** `'after'`
`required`	Whether the slide-toggle is required. **TYPE:** `bool` **DEFAULT:** `False`
`color`	Palette color of slide toggle. **TYPE:** `Literal['primary', 'accent', 'warn'] | None` **DEFAULT:** `None`
`disabled`	Whether the slide toggle is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`disable_ripple`	Whether the slide toggle has a ripple. **TYPE:** `bool` **DEFAULT:** `False`
`tab_index`	Tabindex of slide toggle. **TYPE:** `int` **DEFAULT:** `0`
`checked`	Whether the slide-toggle element is checked or not. **TYPE:** `bool` **DEFAULT:** `False`
`hide_icon`	Whether to hide the icon inside of the slide toggle. **TYPE:** `bool` **DEFAULT:** `False`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`class` `SlideToggleChangeEvent` `dataclass`

Bases: `MesopEvent`

Event triggered when the slide toggle state changes.

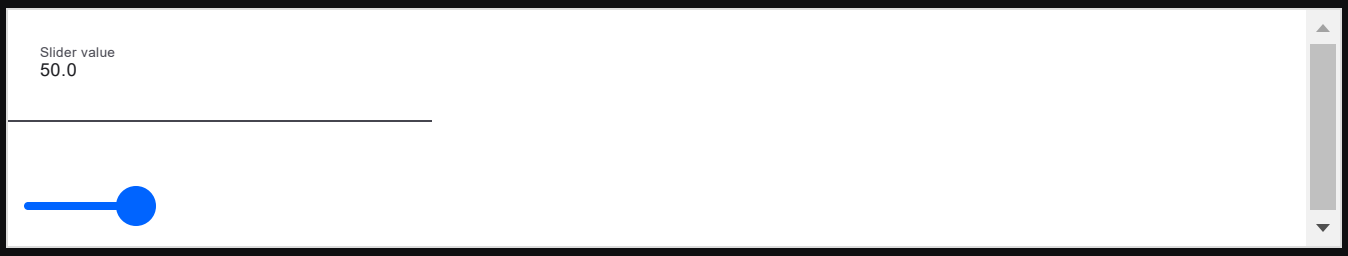
ATTRIBUTE	DESCRIPTION
`key`	Key of the component that emitted this event. **TYPE:** `str`

Slider

Overview

Slider allows the user to select from a range of values and is based on the Angular Material slider component.

Examples




```
import mesop as me


@me.stateclass
class State:
  initial_input_value: str = "50.0"
  initial_slider_value: float = 50.0
  slider_value: float = 50.0


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/slider",
)
def app():
  state = me.state(State)
  with me.box(style=me.Style(display="flex", flex_direction="column")):
    me.input(
      label="Slider value", value=state.initial_input_value, on_input=on_input
    )
    me.slider(on_value_change=on_value_change, value=state.initial_slider_value)
    me.text(text=f"Value: {me.state(State).slider_value}")


def on_value_change(event: me.SliderValueChangeEvent):
  state = me.state(State)
  state.slider_value = event.value
  state.initial_input_value = str(state.slider_value)


def on_input(event: me.InputEvent):
  state = me.state(State)
  state.initial_slider_value = float(event.value)
  state.slider_value = state.initial_slider_value
```

API

`func` `slider`

Creates a Slider component.

PARAMETER	DESCRIPTION
`on_value_change`	An event will be dispatched each time the slider changes its value. **TYPE:** `Callable[[SliderValueChangeEvent], Any] | None` **DEFAULT:** `None`
`value`	Initial value. If updated, the slider will be updated with a new initial value. **TYPE:** `float | None` **DEFAULT:** `None`
`min`	The minimum value that the slider can have. **TYPE:** `float` **DEFAULT:** `0`
`max`	The maximum value that the slider can have. **TYPE:** `float` **DEFAULT:** `100`
`step`	The values at which the thumb will snap.

PARAMETER	DESCRIPTION
	TYPE: `float` **DEFAULT:** `1`
`disabled`	Whether the slider is disabled. **TYPE:** `bool` **DEFAULT:** `False`
`discrete`	Whether the slider displays a numeric value label upon pressing the thumb. **TYPE:** `bool` **DEFAULT:** `False`
`show_tick_marks`	Whether the slider displays tick marks along the slider track. **TYPE:** `bool` **DEFAULT:** `False`
`color`	Palette color of the slider. **TYPE:** `Literal['primary', 'accent', 'warn']` **DEFAULT:** `'primary'`
`disable_ripple`	Whether ripples are disabled in the slider. **TYPE:** `bool` **DEFAULT:** `False`
`style`	Style for the component. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

`class` `SliderValueChangeEvent` `dataclass`

Bases: `MesopEvent`

Event triggered when the slider value changes.

ATTRIBUTE	DESCRIPTION
`value`	The new value of the slider after the change. **TYPE:** `float`
`key`	Key of the component that emitted this event. **TYPE:** `str`

State Management

State management is a critical element of building interactive apps because it allows you store information about what the user did in a structured way.

Basic usage

You can register a class using the class decorator `me.stateclass` which is like a dataclass with special powers:

```
@me.stateclass
class State:
  val: str
```

You can get an instance of the state class inside any of your Mesop component functions by using `me.state`:

```
@me.page()
def page():
    state = me.state(State)
    me.text(state.val)
```

How State Works

`me.stateclass` is a class decorator which tells Mesop that this class can be retrieved using the `me.state` method, which will return the state instance for the current user session.

> If you are familiar with the dependency injection pattern, Mesop's stateclass and state API is essentially a minimalist dependency injection system which scopes the state object to the lifetime of a user session.

Under the hood, Mesop is sending the state back and forth between the server and browser client so everything in a state class must be serializable.

Multiple state classes

You can use multiple classes to store state for the current user session.

Using different state classes for different pages or components can help make your app easier to maintain and more modular.

```
@me.stateclass
class PageAState:
    ...

@me.stateclass
class PageBState:
    ...

@me.page(path="/a")
def page_a():
    state = me.state(PageAState)
    ...

@me.page(path="/b")
def page_b():
    state = me.state(PageBState)
    ...
```

Under the hood, Mesop is managing state classes based on the identity (e.g. module name and class name) of the state class, which means that you could have two state classes named "State", but if they are in different modules, then they will be treated as separate state, which is what you would expect.

Nested State

You can also have classes inside of a state class as long as everything is serializable:

```
class NestedState:
  val: str

@me.stateclass
class State:
  nested: NestedState

def app():
  state = me.state(State)
```

> Note: you only need to decorate the top-level state class with `@me.stateclass`. All the nested state classes will automatically be wrapped.

Nested State and dataclass

Sometimes, you may want to explicitly decorate the nested state class with `dataclass` because in the previous example, you couldn't directly instantiate `NestedState`.

If you wanted to use NestedState as a general dataclass, you can do the following:

```
@dataclass
class NestedState:
  val: str = ""

@me.stateclass
class State:
  nested: NestedState

def app():
  state = me.state(State)
```

> Reminder: because dataclasses do not have default values, you will need to explicitly set default values, otherwise Mesop will not be able to instantiate an empty version of the class.

Now, if you have an event handler function, you can do the following:

```
def on_click(e):
    response = call_api()
    state = me.state(State)
    state.nested = NestedState(val=response.text)
```

If you didn't explicitly annotate NestedState as a dataclass, then you would get an error instantiating NestedState because there's no initializer defined.

Tips

Set mutable default values (e.g. list) correctly

Similar to regular dataclasses which disallow mutable default values, you need to avoid mutable default values such as list and dict for state classes. Allowing mutable default values could lead to erroneously sharing state across users which would be bad!

Bad: Setting a mutable field directly on a state class attribute.

```
@me.stateclass
class State:
  x: list[str] = ["a"]
```

Good: Use dataclasses `field` method to define a default factory so a new instance of the mutable value is created with each state class instance.

```
from dataclasses import field

@me.stateclass
class State:
  x: list[str] = field(default_factory=lambda: ["a"])
```

State performance issues

Because the state class is serialized and sent back and forth between the client and server, you should try to keep the state class reasonably sized. For example, if you store a very large string (e.g. base64-encoded image) in state, then it will degrade performance of your Mesop app. Instead, you should try to store large data outside of the state class (e.g. in-memory, filesystem, database, external service) and retrieve the data as needed for rendering.

Style

Overview

Mesop provides a Python API that wraps the browser's native CSS style API.

API

`class Style` `dataclass`

Represents the style configuration for a UI component.

ATTRIBUTE	DESCRIPTION
`align_content`	Aligns the flexible container's items on the cross-axis. See MDN doc. **TYPE:** `ContentAlignmentValues | None`
`align_items`	Specifies the default alignment for items inside a flexible container. See MDN doc. **TYPE:** `ItemAlignmentValues | None`
`align_self`	Overrides a grid or flex item's align-items value. In Grid, it aligns the item inside the grid area. In Flexbox, it aligns the item on the cross axis. See MDN doc. **TYPE:** `ItemAlignmentValues | None`
`aspect_ratio`	Specifies the desired width-to-height ratio of a component. See MDN doc. **TYPE:** `str | None`
`background`	Sets the background color or image of the component. See MDN doc. **TYPE:** `str | None`
`border`	Defines the border properties for each side of the component. See MDN doc. **TYPE:** `Border | None`
`border_radius`	Defines the border radius. See MDN doc. **TYPE:** `int | str | None`
`bottom`	Helps set vertical position of a positioned element. See MDN doc. **TYPE:** `int | str | None`
`box_shadow`	Defines the box shadow. See MDN doc. **TYPE:** `str | None`
`box_sizing`	Defines the box sizing. See MDN doc. **TYPE:** `str | None`
`color`	Sets the color of the text inside the component. See MDN doc. **TYPE:** `str | None`
`column_gap`	Sets the gap between columns. See MDN doc. **TYPE:** `int | str | None`
`columns`	Specifies the number of columns in a multi-column element. See MDN doc. **TYPE:** `int | str | None`
`cursor`	Sets the mouse cursor. See MDN doc. **TYPE:** `str | None`
`display`	Defines the display type of the component. See MDN doc. **TYPE:** `Literal['block', 'inline', 'inline-block', 'flex', 'inline-flex', 'grid', 'inline-grid', 'none', 'contents'] | None`
`flex_basis`	Specifies the initial length of a flexible item. See MDN doc. **TYPE:** `str | None`
`flex_direction`	Establishes the main-axis, thus defining the direction flex items are placed in the flex container. See MDN doc. **TYPE:** `Literal['row', 'row-reverse', 'column', 'column-reverse'] | None`
`flex_grow`	Defines the ability for a flex item to grow if necessary. See MDN doc. **TYPE:** `int | None`

ATTRIBUTE	DESCRIPTION
`flex_shrink`	Defines the ability for a flex item to shrink if necessary. See MDN doc. **TYPE:** `int | None`
`flex_wrap`	Allows flex items to wrap onto multiple lines. See MDN doc. **TYPE:** `Literal['nowrap', 'wrap', 'wrap-reverse'] | None`
`font_family`	Specifies the font family. See MDN doc. **TYPE:** `str | None`
`font_size`	Sets the size of the font. See MDN doc. **TYPE:** `int | str | None`
`font_style`	Specifies the font style for text. See MDN doc. **TYPE:** `Literal['italic', 'normal'] | None`
`font_weight`	Sets the weight (or boldness) of the font. See MDN doc. **TYPE:** `Literal['bold', 'normal', 100, 200, 300, 400, 500, 600, 700, 800, 900] | None`
`gap`	Sets the gap. See MDN doc. **TYPE:** `int | str | None`
`grid_area`	Sets the grid area. See MDN doc. **TYPE:** `str | None`
`grid_auto_columns`	CSS property specifies the size of an implicitly-created grid column track or pattern of tracks. See MDN doc. **TYPE:** `str | None`
`grid_auto_flow`	CSS property controls how the auto-placement algorithm works, specifying exactly how auto-placed items get flowed into the grid. See MDN doc. **TYPE:** `str | None`
`grid_auto_rows`	CSS property specifies the size of an implicitly-created grid row track or pattern of tracks. See MDN doc. **TYPE:** `str | None`
`grid_column`	CSS shorthand property specifies a grid item's size and location within a grid column. See MDN doc. **TYPE:** `str | None`
`grid_column_start`	Sets the grid column start. See MDN doc. **TYPE:** `int | str | None`
`grid_column_end`	Sets the grid column end. See MDN doc. **TYPE:** `int | str | None`
`grid_row`	CSS shorthand property specifies a grid item's size and location within a grid row. See MDN doc. **TYPE:** `str | None`
`grid_row_start`	Sets the grid row start. See MDN doc. **TYPE:** `int | str | None`
`grid_row_end`	Sets the grid row end. See MDN doc. **TYPE:** `int | str | None`
`grid_template_areas`	Sets the grid template areas; each element is a row. See MDN doc. **TYPE:** `list[str] | None`
`grid_template_columns`	Sets the grid template columns. See MDN doc. **TYPE:** `str | None`
`grid_template_rows`	Sets the grid template rows. See MDN doc. **TYPE:** `str | None`
`height`	Sets the height of the component. See MDN doc. **TYPE:** `int | str | None`
`justify_content`	Aligns the flexible container's items on the main-axis. See MDN doc. **TYPE:** `ContentAlignmentValues | None`
`justify_items`	Defines the default justify-self for all items of the box, giving them all a default way of justifying each box along the appropriate axis. See MDN doc.

ATTRIBUTE	DESCRIPTION
	TYPE: `ItemJustifyValues | None`
`justify_self`	Sets the way a box is justified inside its alignment container along the appropriate axis. See MDN doc. **TYPE:** `ItemJustifyValues | None`
`left`	Helps set horizontal position of a positioned element. See MDN doc. **TYPE:** `int | str | None`
`letter_spacing`	Increases or decreases the space between characters in text. See MDN doc. **TYPE:** `int | str | None`
`line`	Set the line height (relative to the font size). See MDN doc. **TYPE:** `height`
`margin`	Sets the margin space required on each side of an element. See MDN doc. **TYPE:** `Margin | None`
`opacity`	Sets the opacity property. See MDN doc. **TYPE:** `float | str | None`
`outline`	Sets the outline property. Note: `input` component has default browser stylings. See MDN doc. **TYPE:** `str | None`
`overflow_wrap`	Specifies how long text can be broken up by new lines to prevent overflowing. See MDN doc. **TYPE:** `OverflowWrapValues | None`
`overflow_x`	Specifies the handling of overflow in the horizontal direction. See MDN doc. **TYPE:** `OverflowValues | None`
`overflow_y`	Specifies the handling of overflow in the vertical direction. See MDN doc. **TYPE:** `OverflowValues | None`
`padding`	Sets the padding space required on each side of an element. See MDN doc. **TYPE:** `Padding | None`
`position`	Specifies the type of positioning method used for an element (static, relative, absolute, fixed, or sticky). See MDN doc. **TYPE:** `Literal['static', 'relative', 'absolute', 'fixed', 'sticky'] | None`
`right`	Helps set horizontal position of a positioned element. See MDN doc. **TYPE:** `int | str | None`
`rotate`	Allows you to specify rotation transforms individually and independently of the transform property. See MDN doc. **TYPE:** `str | None`
`row_gap`	Sets the gap between rows. See MDN doc. **TYPE:** `int | str | None`
`text_align`	Specifies the horizontal alignment of text in an element. See MDN doc. **TYPE:** `Literal['start', 'end', 'left', 'right', 'center'] | None`
`text_decoration`	Specifies the decoration added to text. See MDN doc. **TYPE:** `Literal['underline', 'none'] | None`
`text_overflow`	Specifies how overflowed content that is not displayed should be signaled to the user. See MDN doc. **TYPE:** `Literal['ellipsis', 'clip'] | None`
`top`	Helps set vertical position of a positioned element. See MDN doc. **TYPE:** `int | str | None`
`transform`	Lets you rotate, scale, skew, or translate an element. It modifies the coordinate space of the CSS visual formatting model. See MDN doc. **TYPE:** `str | None`
`visibility`	Sets the visibility property. See MDN doc. **TYPE:** `Literal['visible', 'hidden', 'collapse', 'inherit', 'initial', 'revert', 'revert-layer', 'unset'] | None`
`white_space`	Specifies how white space inside an element is handled. See MDN doc. **TYPE:** `Literal['normal', 'nowrap', 'pre', 'pre-wrap', 'pre-line', 'break-spaces'] | None`

ATTRIBUTE	DESCRIPTION
`width`	Sets the width of the component. See MDN doc. **TYPE:** `int` | `str` | `None`
`z-index`	Sets the z-index of the component. See MDN doc. **TYPE:** `int` | `str` | `None`

`class` Border `dataclass`

Defines the border styles for each side of a UI component.

ATTRIBUTE	DESCRIPTION
`top`	Style for the top border. **TYPE:** `BorderSide` | `None`
`right`	Style for the right border. **TYPE:** `BorderSide` | `None`
`bottom`	Style for the bottom border. **TYPE:** `BorderSide` | `None`
`left`	Style for the left border. **TYPE:** `BorderSide` | `None`

`meth` `all` `staticmethod`

Creates a Border instance with all sides having the same style.

PARAMETER	DESCRIPTION
`value`	The style to apply to all sides of the border. **TYPE:** `BorderSide`

RETURNS	DESCRIPTION
`Border`	A new Border instance with the specified style applied to all sides. **TYPE:** `Border`

`meth` `symmetric` `staticmethod`

Creates a Border instance with symmetric styles for vertical and horizontal sides.

PARAMETER	DESCRIPTION
`vertical`	The style to apply to the top and bottom sides of the border. **TYPE:** `BorderSide` | `None` **DEFAULT:** `None`
`horizontal`	The style to apply to the right and left sides of the border. **TYPE:** `BorderSide` | `None` **DEFAULT:** `None`

RETURNS	DESCRIPTION
`Border`	A new Border instance with the specified styles applied symmetrically. **TYPE:** `Border`

`class` BorderSide `dataclass`

Represents the style of a single side of a border in a UI component.

ATTRIBUTE	DESCRIPTION
`width`	The width of the border. Can be specified as an integer value representing pixels, a string with a unit (e.g., '2em'), or None for no width. **TYPE:** `int` | `str` | `None`
`color`	The color of the border, represented as a string. This can be any valid CSS color value, or None for no color.

ATTRIBUTE	DESCRIPTION
	TYPE: `str | None`
`style`	The style of the border, which can be 'none' for no border, 'solid' for a solid line. **TYPE:** `Literal['none', 'solid'] | None`

`class` Margin `dataclass`

Bases: `_EdgeInsets`

Defines the margin space around a UI component.

ATTRIBUTE	DESCRIPTION
`top`	Top margin (note: `2` is the same as `2px`) **TYPE:** `int | str | None`
`right`	Right margin **TYPE:** `int | str | None`
`bottom`	Bottom margin **TYPE:** `int | str | None`
`left`	Left margin **TYPE:** `int | str | None`

`meth` all `staticmethod`

Creates a Margin instance with the same value for all sides.

PARAMETER	DESCRIPTION
`value`	The value to apply to all sides of the margin. Can be an integer (pixel value) or a string. **TYPE:** `int | str`

RETURNS	DESCRIPTION
`Margin`	A new Margin instance with the specified value applied to all sides. **TYPE:** `Margin`

`meth` symmetric `staticmethod`

Creates a Margin instance with symmetric values for vertical and horizontal sides.

PARAMETER	DESCRIPTION
`vertical`	The value to apply to the top and bottom sides of the margin. Can be an integer (pixel value) or a string. **TYPE:** `int | str | None` **DEFAULT:** `None`
`horizontal`	The value to apply to the right and left sides of the margin. Can be an integer (pixel value) or a string. **TYPE:** `int | str | None` **DEFAULT:** `None`

RETURNS	DESCRIPTION
`Margin`	A new Margin instance with the specified values applied to the vertical and horizontal sides. **TYPE:** `Margin`

`class` Padding `dataclass`

Bases: `_EdgeInsets`

Defines the padding space around a UI component.

ATTRIBUTE	DESCRIPTION
`top`	Top padding (note: `2` is the same as `2px`) **TYPE:** `int | str | None`

ATTRIBUTE	DESCRIPTION
`right`	Right padding **TYPE:** `int` | `str` | `None`
`bottom`	Bottom padding **TYPE:** `int` | `str` | `None`
`left`	Left padding **TYPE:** `int` | `str` | `None`

`meth` **`all`** `staticmethod`

Creates a Padding instance with the same value for all sides.

PARAMETER	DESCRIPTION
`value`	The value to apply to all sides of the padding. Can be an integer (pixel value) or a string. **TYPE:** `int` | `str`

RETURNS	DESCRIPTION
`Padding`	A new Padding instance with the specified value applied to all sides. **TYPE:** `Padding`

`meth` **`symmetric`** `staticmethod`

Creates a Padding instance with symmetric values for vertical and horizontal sides.

PARAMETER	DESCRIPTION
`vertical`	The value to apply to the top and bottom sides of the padding. Can be an integer (pixel value) or a string. **TYPE:** `int` | `str` | `None` **DEFAULT:** `None`
`horizontal`	The value to apply to the right and left sides of the padding. Can be an integer (pixel value) or a string. **TYPE:** `int` | `str` | `None` **DEFAULT:** `None`

RETURNS	DESCRIPTION
`Padding`	A new Padding instance with the specified values applied to the vertical and horizontal sides. **TYPE:** `Padding`

Table

Overview

Table allows the user to render an Angular Material table component from a Pandas data frame.

Examples

NA	Index	Bools	Ints	Floats	Strings	Date Times
<NA>	3	True	101	2.3	Hello	2018-03-10 00:00:00
<NA>	2	False	90	4.5	World	2023-03-10 00:00:00
<NA>	1	True	-55	-3.000000003	!	2023-01-01 12:12:01

No cell selected.

```
from datetime import datetime

import numpy as np
import pandas as pd

import mesop as me


@me.stateclass
class State:
  selected_cell: str = "No cell selected."


df = pd.DataFrame(
  data={
    "NA": [pd.NA, pd.NA, pd.NA],
    "Index": [3, 2, 1],
    "Bools": [True, False, np.bool_(True)],
    "Ints": [101, 90, np.int64(-55)],
    "Floats": [2.3, 4.5, np.float64(-3.000000003)],
    "Strings": ["Hello", "World", "!"],
    "Date Times": [
      pd.Timestamp("20180310"),
      pd.Timestamp("20230310"),
      datetime(2023, 1, 1, 12, 12, 1),
    ],
  }
)


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/table",
)
def app():
  state = me.state(State)

  with me.box(style=me.Style(padding=me.Padding.all(10), width=500)):
    me.table(
      df,
      on_click=on_click,
```

```
      header=me.TableHeader(sticky=True),
      columns={
        "NA": me.TableColumn(sticky=True),
        "Index": me.TableColumn(sticky=True),
      },
    )

  with me.box(
    style=me.Style(
      background="#ececec",
      margin=me.Margin.all(10),
      padding=me.Padding.all(10),
    )
  ):
    me.text(state.selected_cell)


def on_click(e: me.TableClickEvent):
  state = me.state(State)
  state.selected_cell = (
    f"Selected cell at col {e.col_index} and row {e.row_index} "
    f"with value {str(df.iat[e.row_index, e.col_index])}"
  )
```

API

`func` `table`

This function creates a table from Pandas data frame

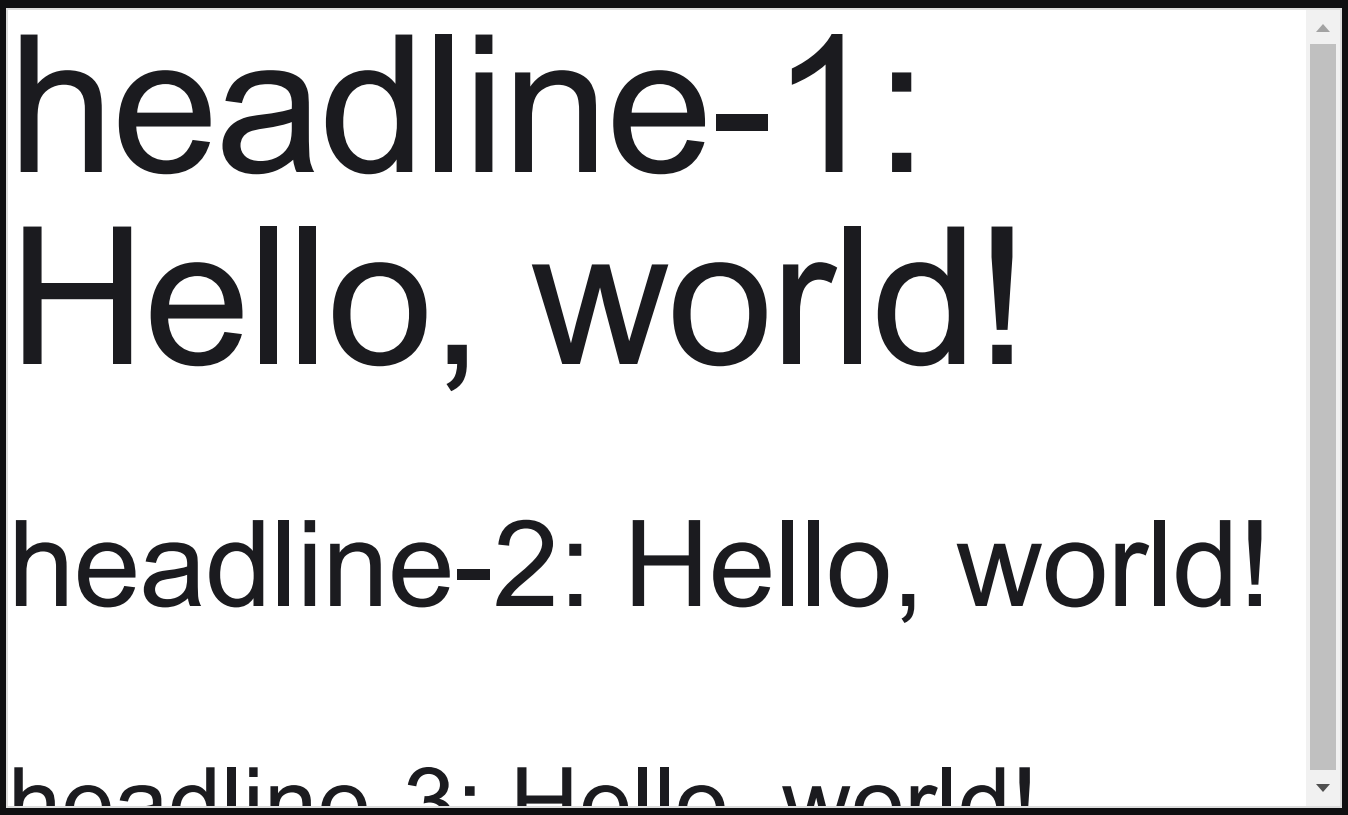
PARAMETER	DESCRIPTION
`data_frame`	Pandas data frame. **TYPE:** `Any`
`on_click`	Triggered when a table cell is clicked. The click event is a native browser event. **TYPE:** `Callable[[TableClickEvent], Any] | None` **DEFAULT:** `None`
`header`	Configures table header to be sticky or not. **TYPE:** `TableHeader | None` **DEFAULT:** `None`
`columns`	Configures table columns to be sticky or not. The key is the name of the column. **TYPE:** `dict[str, TableColumn] | None` **DEFAULT:** `None`

Text

Overview

Text displays text as-is. If you have markdown text, use the Markdown component.

Examples




```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/text",
)
def text():
  me.text(text="headline-1: Hello, world!", type="headline-1")
  me.text(text="headline-2: Hello, world!", type="headline-2")
  me.text(text="headline-3: Hello, world!", type="headline-3")
  me.text(text="headline-4: Hello, world!", type="headline-4")
  me.text(text="headline-5: Hello, world!", type="headline-5")
  me.text(text="headline-6: Hello, world!", type="headline-6")
  me.text(text="subtitle-1: Hello, world!", type="subtitle-1")
  me.text(text="subtitle-2: Hello, world!", type="subtitle-2")
  me.text(text="body-1: Hello, world!", type="body-1")
  me.text(text="body-2: Hello, world!", type="body-2")
  me.text(text="caption: Hello, world!", type="caption")
  me.text(text="button: Hello, world!", type="button")
```

API

`func` `text`

Create a text component.

PARAMETER	DESCRIPTION
`text`	The text to display. **TYPE:** `str` | `None` **DEFAULT:** `None`
`type`	The typography level for the text.

PARAMETER	DESCRIPTION
	TYPE: `Literal['headline-1', 'headline-2', 'headline-3', 'headline-4', 'headline-5', 'headline-6', 'subtitle-1', 'subtitle-2', 'body-1', 'body-2', 'caption', 'button'] | None` **DEFAULT:** `None`
`style`	Style to apply to component. Follows HTML Element inline style API. **TYPE:** `Style | None` **DEFAULT:** `None`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Text to Image

Overview

Text To Image component is a quick and simple way of getting started with Mesop. Text To Image is part of Mesop Labs.

Examples


```
import mesop as me
import mesop.labs as mel


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/text_to_image",
  title="Text to Image Example",
)
def app():
  mel.text_to_image(
    generate_image,
    title="Text to Image Example",
  )


def generate_image(prompt: str):
  return "https://www.google.com/logos/doodles/2024/earth-day-2024-6753651837110453-2xa.gif"
```

API

`func` `text_to_image`

Creates a simple UI which takes in a text input and returns an image output.

This function creates event handlers for text input and output operations using the provided function `transform` to process the input and generate the image output.

PARAMETER	DESCRIPTION
`transform`	Function that takes in a string input and returns a URL to an image or a base64 encoded image. **TYPE:** `Callable[[str], str]`
`title`	Headline text to display at the top of the UI. **TYPE:** `str | None` **DEFAULT:** `None`

Text to Text

Overview

Text to text component allows you to take in user inputted text and return a transformed text. This is part of Mesop Labs.

Examples


```
import mesop as me
import mesop.labs as mel


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/text_to_text",
  title="Text to Text Example",
)
def app():
  mel.text_to_text(
    upper_case_stream,
    title="Text to Text Example",
  )


def upper_case_stream(s: str):
  return "Echo: " + s
```

API

`func` `text_to_text`

Creates a simple UI which takes in a text input and returns a text output.

This function creates event handlers for text input and output operations using the provided transform function to process the input and generate the output.

PARAMETER	DESCRIPTION
`transform`	Function that takes in a string input and either returns or yields a string output. **TYPE:** `Callable[[str], Generator[str, None, None] | str]`
`title`	Headline text to display at the top of the UI **TYPE:** `str | None` **DEFAULT:** `None`

PARAMETER	DESCRIPTION
`transform_mode`	Specifies how the output should be updated when yielding an output using a generator. - "append": Concatenates each new piece of text to the existing output. - "replace": Replaces the existing output with each new piece of text. **TYPE:** `Literal['append', 'replace']` **DEFAULT:** `'append'`

`func` text_io

Deprecated: Use `text_to_text` instead which provides the same functionality with better default settings.

This function creates event handlers for text input and output operations using the provided transform function to process the input and generate the output.

PARAMETER	DESCRIPTION
`transform`	Function that takes in a string input and either returns or yields a string output. **TYPE:** `Callable[[str], Generator[str, None, None] | str]`
`title`	Headline text to display at the top of the UI **TYPE:** `str | None` **DEFAULT:** `None`
`transform_mode`	Specifies how the output should be updated when yielding an output using a generator. - "append": Concatenates each new piece of text to the existing output. - "replace": Replaces the existing output with each new piece of text. **TYPE:** `Literal['append', 'replace']` **DEFAULT:** `'replace'`

Textarea

Overview

Textarea allows the user to type in a value and is based on the Angular Material input component for `<textarea>`.

This is similar to Input, but Textarea is better suited for long text inputs.

Examples


```
import mesop as me


@me.stateclass
class State:
  input: str = ""


def on_input(e: me.InputEvent):
  state = me.state(State)
  state.input = e.value


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/textarea",
)
def app():
  s = me.state(State)
  me.textarea(label="Basic input", on_input=on_input)
  me.text(text=s.input)
```

API

`func` `textarea`

Creates a Textarea component.

PARAMETER	DESCRIPTION
`label`	Label for input. **TYPE:** `str` **DEFAULT:** `''`
`autosize`	If True, the textarea will automatically adjust its height to fit the content, up to the max_rows limit. **TYPE:** `bool` **DEFAULT:** `False`
`min_rows`	The minimum number of rows the textarea will display. **TYPE:** `int | None` **DEFAULT:** `None`
`max_rows`	The maximum number of rows the textarea will display. **TYPE:** `int | None` **DEFAULT:** `None`
`on_input`	input is a native browser event. **TYPE:** `Callable[[InputEvent], Any] | None` **DEFAULT:** `None`
`rows`	The number of lines to show in the text area.

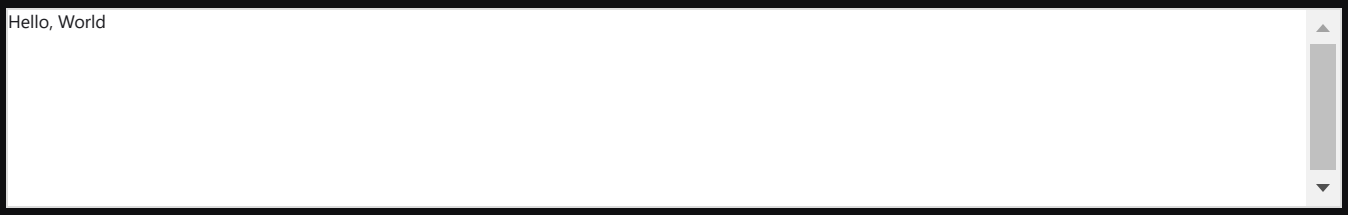
PARAMETER	DESCRIPTION
	TYPE: `int` **DEFAULT:** `5`
`appearance`	The form field appearance style. **TYPE:** `Literal['fill', 'outline']` **DEFAULT:** `'fill'`
`style`	Style for input. **TYPE:** `Style | None` **DEFAULT:** `None`
`disabled`	Whether it's disabled. **TYPE:** `bool` **DEFAULT:** `False`
`placeholder`	Placeholder value **TYPE:** `str` **DEFAULT:** `''`
`required`	Whether it's required **TYPE:** `bool` **DEFAULT:** `False`
`value`	Initial value. **TYPE:** `str` **DEFAULT:** `''`
`readonly`	Whether the element is readonly. **TYPE:** `bool` **DEFAULT:** `False`
`hide_required_marker`	Whether the required marker should be hidden. **TYPE:** `bool` **DEFAULT:** `False`
`color`	The color palette for the form field. **TYPE:** `Literal['primary', 'accent', 'warn']` **DEFAULT:** `'primary'`
`float_label`	Whether the label should always float or float as the user types. **TYPE:** `Literal['always', 'auto']` **DEFAULT:** `'auto'`
`subscript_sizing`	Whether the form field should reserve space for one line of hint/error text (default) or to have the spacing grow from 0px as needed based on the size of the hint/error content. Note that when using dynamic sizing, layout shifts will occur when hint/error text changes. **TYPE:** `Literal['fixed', 'dynamic']` **DEFAULT:** `'fixed'`
`hint_label`	Text for the form field hint. **TYPE:** `str` **DEFAULT:** `''`
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`

Tooltip

Overview

Tooltip is based on the Angular Material tooltip component.

Examples




```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/tooltip",
)
def app():
  with me.tooltip(message="Tooltip message"):
    me.text(text="Hello, World")
```

API

`func` tooltip

Creates a Tooltip component. Tooltip is a composite component.

PARAMETER	DESCRIPTION
`key`	The component key. **TYPE:** `str | None` **DEFAULT:** `None`
`position`	Allows the user to define the position of the tooltip relative to the parent element **TYPE:** `Literal['left', 'right', 'above', 'below', 'before', 'after']` **DEFAULT:** `'left'`
`position_at_origin`	Whether tooltip should be relative to the click or touch origin instead of outside the element bounding box. **TYPE:** `bool` **DEFAULT:** `False`
`disabled`	Disables the display of the tooltip. **TYPE:** `bool` **DEFAULT:** `False`
`show_delay_ms`	The default delay in ms before showing the tooltip after show is called **TYPE:** `int` **DEFAULT:** `0`
`hide_delay_ms`	The default delay in ms before hiding the tooltip after hide is called **TYPE:** `int` **DEFAULT:** `0`
`message`	The message to be displayed in the tooltip **TYPE:** `str` **DEFAULT:** `''`

Uploader

Overview

Uploader is the equivalent of an `<input type="file>` HTML element except it uses a custom UI that better matches the look of Angular Material Components.

Examples


```
import base64

import mesop as me


@me.stateclass
class State:
  name: str
  size: int
  mime_type: str
  contents: str


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/uploader",
)
def app():
  state = me.state(State)
  me.uploader(
    label="Upload Image",
    accepted_file_types=["image/jpeg", "image/png"],
    on_upload=handle_upload,
  )

  if state.contents:
    with me.box(style=me.Style(margin=me.Margin.all(10))):
      me.text(f"File name: {state.name}")
      me.text(f"File size: {state.size}")
      me.text(f"File type: {state.mime_type}")

    with me.box(style=me.Style(margin=me.Margin.all(10))):
      me.image(src=state.contents)


def handle_upload(event: me.UploadEvent):
  state = me.state(State)
  state.name = event.file.name
  state.size = event.file.size
  state.mime_type = event.file.mime_type
  state.contents = f"data:{event.file.mime_type};base64,{base64.b64encode(event.file.getvalue()).decode()}"
```

API

`func` `uploader`

This function creates an uploader.

PARAMETER	DESCRIPTION
`label`	Upload button label. **TYPE:** `str`

PARAMETER	DESCRIPTION
`accepted_file_types`	List of accepted file types. See the accept parameter. **TYPE:** `Sequence[str] | None` **DEFAULT:** `None`
`on_upload`	File upload event handler. **TYPE:** `Callable[[UploadEvent], Any] | None` **DEFAULT:** `None`

`class` `UploadEvent` `dataclass`

Bases: `MesopEvent`

Event for file uploads.

ATTRIBUTE	DESCRIPTION
`file`	Uploaded file. **TYPE:** `UploadedFile`

`class` `UploadedFile`

Bases: `BytesIO`

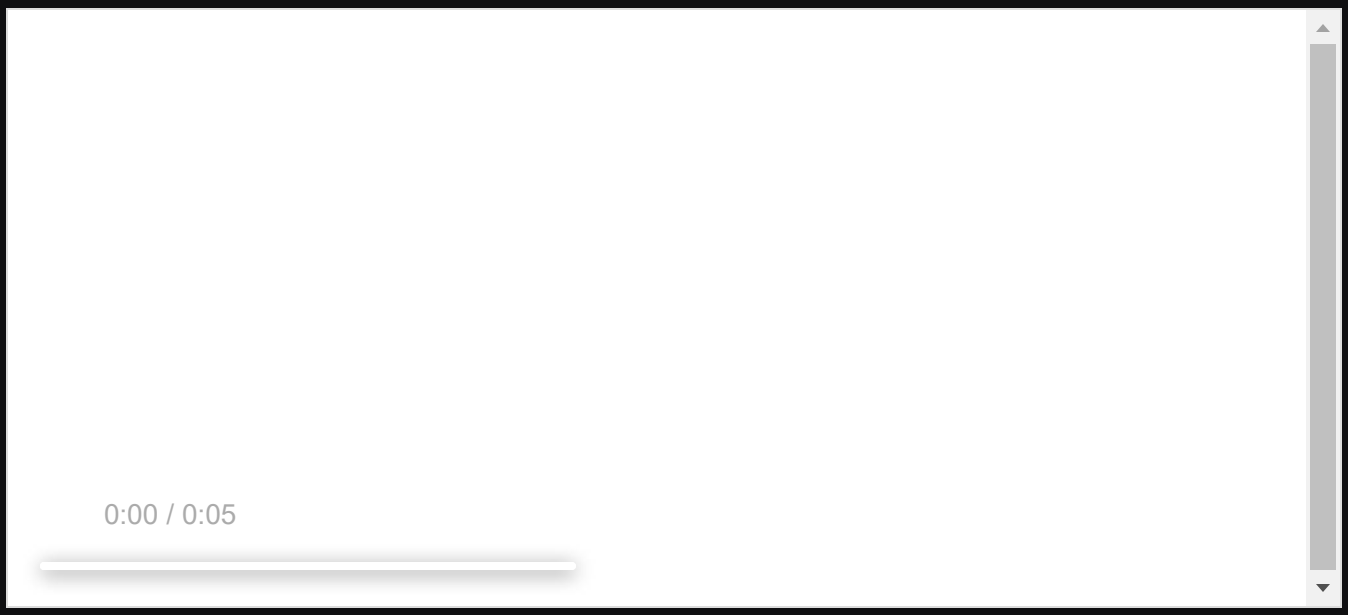
Uploaded file contents and metadata.

Video

Overview

Video is the equivalent of an `<video>` HTML element. Video displays the browser's native video controls.

Examples




```
import mesop as me


@me.page(
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.github.io"]
  ),
  path="/video",
)
def app():
  me.video(
    src="https://interactive-examples.mdn.mozilla.net/media/cc0-videos/flower.webm",
    style=me.Style(height=300, width=300),
  )
```

API

`func` `video`

Creates a video.

PARAMETER	DESCRIPTION
`src`	URL of the video source **TYPE:** `str`
`style`	The style to apply to the image, such as width and height. **TYPE:** `Style | None` **DEFAULT:** `None`

Viewport size

Overview

The viewport size API allows you to access the current viewport size. This can be useful for creating responsive and adaptive designs that are suitable for the user's screen size.

Examples

Responsive Design

Responsive design is having a single fluid layout that adapts to all screen sizes.

You can use the viewport size to dynamically set the property of a style. This can be useful if you want to fit two boxes in a row for larger screens (e.g. desktop) and a single box for smaller screens (e.g. mobile) as shown in the example below:

```
import mesop as me

@me.page()
def page():
    if me.viewport_size().width > 640:
        width = me.viewport_size().width / 2
    else:
        width = me.viewport_size().width
    for i in range(8):
      me.box(style=me.Style(width=width))
```

> Tip: Responsive design tends to take less work and is usually a good starting point.

Adaptive Design

Adaptive design is having multiple fixed layouts for specific device categories at specific breakpoints, typically viewport width.

For example, oftentimes you will hide the nav component on a mobile device and instead show a hamburger menu, while for a larger device you will always show the nav component on the left side.

```
import mesop as me

@me.page()
def page():
    if me.viewport_size().width > 480:
        nav_component()
        body()
    else:
        body(show_menu_button=True)
```

> Tip: Adaptive design tends to take more work and is best for optimizing complex mobile and desktop experiences.

API

`func` `viewport_size`

Returns the current viewport size.

RETURNS	DESCRIPTION
`Size`	The current viewport size. **TYPE:** `Size`

`class` `Size` `dataclass`

ATTRIBUTE	DESCRIPTION
`width`	The width of the viewport in pixels. **TYPE:** `int`
`height`	The height of the viewport in pixels. **TYPE:** `int`

Web Components API

> Note: Web components are a new experimental feature released under labs and may have breaking changes.

Example usage:

```
import mesop.labs as mel


@mel.web_component(...)
def a_web_component():
    mel.insert_web_component(...)
```

API

func web_component

A decorator for defining a web component.

This decorator is used to define a web component. It takes a path to the JavaScript file of the web component and an optional parameter to skip validation. It then registers the JavaScript file in the runtime.

PARAMETER	DESCRIPTION
path	The path to the JavaScript file of the web component. **TYPE:** str
skip_validation	If set to True, skips validation. Defaults to False. **TYPE:** bool **DEFAULT:** False

func insert_web_component

Inserts a web component into the current component tree.

PARAMETER	DESCRIPTION
name	The name of the web component. This should match the custom element name defined in JavaScript. **TYPE:** str
events	A dictionary where the key is the event name, which must match a web component property name defined in JavaScript. The value is the event handler (callback) function. Keys must not be "src", "srcdoc", or start with "on" to avoid web security risks. **TYPE:** dict[str, Callable[[WebEvent], Any]] | None **DEFAULT:** None
properties	A dictionary where the key is the web component property name that's defined in JavaScript and the value is the property value which is plumbed to the JavaScript component. Keys must not be "src", "srcdoc", or start with "on" to avoid web security risks. **TYPE:** dict[str, Any] | None **DEFAULT:** None
key	A unique identifier for the web component. Defaults to None. **TYPE:** str | None **DEFAULT:** None

class WebEvent dataclass

Bases: MesopEvent

An event emitted by a web component.

ATTRIBUTE	DESCRIPTION
value	The value associated with the web event. **TYPE:** Any
key	key of the component that emitted this event. **TYPE:** str

func slot

This function is used when defining a content component to mark a place in the component tree where content can be provided by a child component.

Web Components

> Note: Web components are a new experimental feature released under labs and may have breaking changes.

Mesop allows you to define custom components with web components which is a set of web standards that allows you to use JavaScript and CSS to define custom HTML elements.

Use cases

- **Custom JavaScript** - You can execute custom JavaScript and have simple bi-directional communication between the JavaScript code running in the browser and the Python code running the server.
- **JavaScript libraries** - If you want to use a JavaScript library, you can wrap them with a web component.
- **Rich-client side interactivity** - You can use web components to deliver stateful client-side interactions without a network roundtrip.

Anatomy of a web component

Mesop web component consists of two parts:

- **Python module** - defines a Python API so that your Mesop app can use the web component seamlessly.
- **JavaScript module** - implements the web component.

Next steps

Learn how to build your first web component in the quickstart page.

Quickstart

> Note: Web components are a new experimental feature released under labs and may have breaking changes.

You will learn how to build your first web component step-by-step, a counter component.

Although it's a simple example, it will show you the core APIs of defining your own web component and how to support bi-directional communication between the Python code running on the server and JavaScript code running on the browser.

Python module

Let's first take a look at the Python module which defines the interface so that the rest of your Mesop app can call the web component in a Pythonic way.

counter_component.py

```
from typing import Any, Callable

import mesop.labs as mel


@mel.web_component(path="./counter_component.js")
def counter_component(
  *,
  value: int,
  on_decrement: Callable[[mel.WebEvent], Any],
  key: str | None = None,
):
  return mel.insert_web_component(
    name="quickstart-counter-component",
    key=key,
    events={
      "decrementEvent": on_decrement,
    },
    properties={
      "value": value,
    },
  )
```

The first part you will notice is the decorator: `@mel.web_component` . This annotates a function as a web component and specifies where the corresponding JavaScript module is located, relative to the location of this Python module.

We've defined the function parameters just like a regular Python function.

> Tip: We recommend annotating your parameter with types because Mesop will do runtime validation which will catch type issues earlier.

Finally, we call the function `mel.insert_web_component` with the following arguments:

- `name` - This is the web component name and must match the name defined in the JavaScript module.
- `key` - Like all components, web components accept a key which is a unique identifier. See the component key docs for more info.
- `events` - A dictionary where the key is the event name. This must match a property name, defined in JavaScript. The value is the event handler (callback) function.
- `properties` - A dictionary where the key is the property name that's defined in JavaScript and the value is the property value which is plumbed to the JavaScript component.

> Note: Keys for events and properties must not be "src", "srcdoc", or start with "on" to avoid web security risks.

In summary, when you see a string literal, it should match something on the JavaScript side which is explained next.

JavaScript module

Let's now take a look at how we implement in the web component in JavaScript:

counter_component.js

```
import {
  LitElement,
  html,
} from 'https://cdn.jsdelivr.net/gh/lit/dist@3/core/lit-core.min.js';

class CounterComponent extends LitElement {
  static properties = {
    value: {type: Number},
    decrementEvent: {type: String},
  };

  constructor() {
    super();
    this.value = 0;
    this.decrementEvent = '';
  }

  render() {
    return html`
      <div class="container">
```

```
        <span>Value: ${this.value}</span>
        <button id="decrement-btn" @click="${this._onDecrement}">
          Decrement
        </button>
      </div>
    `;
  }

  _onDecrement() {
    this.dispatchEvent(
      new MesopEvent(this.decrementEvent, {
        value: this.value - 1,
      }),
    );
  }
}

customElements.define('quickstart-counter-component', CounterComponent);
```

In this example, we have used Lit which is a small library built on top of web standards in a simple, secure and declarative manner.

> Note: you can write your web components using any web technologies (e.g. TypeScript) or frameworks as long as they conform to the interface defined by your Python module.

Properties

The static property named `properties` defines two kinds of properties:

- **Regular properties -** these were defined in the `properties` argument of `insert_web_component`. The property name in JS must match one of the `properties` dictionary key. You also should make sure the Python and JS types are compatible to avoid issues.
- **Event properties -** these were defined in the `events` argument of `insert_web_component`. The property name in JS must match one of the `events` dictionary key. Event properties are always type `String` because the value is a handler id which identifies the Python event handler function.

Triggering an event

To trigger an event in your component, let's look at the `_onDecrement` method implementation:

```
this.dispatchEvent(
  new MesopEvent(this.decrementEvent, {
    value: this.value - 1,
  }),
);
```

`this.dispatchEvent` is a standard web API where a DOM element can emit an event. For Mesop web components, we will always emit a `MesopEvent` which is a class provided on the global object (`window`). The first argument is the event handler id so Mesop knows which Python function to call as the event handler and the second argument is the payload which is a JSON-serializable value (oftentimes an object) that the Python event handler can access.

Learn more about Lit

I didn't cover the `render` function which is a standard Lit method. I recommend reading through Lit's docs which are excellent ahd have interactive tutorials.

Using the component

Finally, let's use the web component we defined. When you click on the decrement button, the value will decrease from 10 to 9 and so on.

counter_component_app.py

```
from pydantic import BaseModel

import mesop as me
import mesop.labs as mel
from mesop.examples.web_component.quickstart.counter_component import (
  counter_component,
)


@me.page(
  path="/web_component/quickstart/counter_component_app",
)
def page():
  counter_component(
    value=me.state(State).value,
    on_decrement=on_decrement,
  )


@me.stateclass
class State:
  value: int = 10


class ChangeValue(BaseModel):
  value: int


def on_decrement(e: mel.WebEvent):
  # Creating a Pydantic model from the JSON value of the WebEvent
  # to enforce type safety.
```

```
    decrement = ChangeValue(**e.value)
    me.state(State).value = decrement.value
```

Even though this was a toy example, you've learned how to build a web component from scratch which does bi-directional communication between the Python server and JavaScript client.

Next steps

To learn more, read the API docs or look at the examples.

Web Security

Mesop by default configures its apps to follow a set of web security best practices.

How

At a high-level, Mesop is built on top of Angular which provides built-in security protections and Mesop configures a strict Content Security Policy.

Specifics:

- Mesop APIs do not allow arbitrary JavaScript execution in the main execution context. For example, the markdown component sanitizes the markdown content and removes active HTML content like JavaScript.
- Mesop's default Content Security Policy prevents arbitrary JavaScript code from executing on the page unless it passes Angular's Trusted Types polices.

Iframe Security

To prevent clickjacking, Mesop apps, when running in prod mode (the default mode used when deployed), do not allow sites from any other origins to iframe the Mesop app.

> Note: pages from the same origin as the Mesop app can always iframe the Mesop app.

If you want to allow a trusted site to iframe your Mesop app, you can explicitly allow list the sources which can iframe your app by configuring the security policy for a particular page.

Example

```
import mesop as me


@me.page(
  path="/allows_iframed",
  security_policy=me.SecurityPolicy(
    allowed_iframe_parents=["https://google.com"],
  ),
)
def app():
  me.text("Test CSP")
```

You can also use wildcards to allow-list multiple subdomains from the same site, such as: `https://*.example.com`.

Why Mesop?

Mesop is a new UI framework that enables Python developers to quickly build delightful web apps in a scalable way.

Many Python UI frameworks are easy to get started with, but customizing beyond the defaults often requires diving into JavaScript, CSS, and HTML — a steep learning curve for many developers.

Mesop provides a different approach, offering a framework that's both easy to learn and enables flexible UI building, all within Python.

I want to share a couple concrete ways in which Mesop achieves this.

Build UIs with Functions (i.e. Components)

Mesop embraces a component-based philosophy where the entire UI is composed of reusable, building blocks which are called components. Using a component is as simple as calling a Python function. This approach offers several benefits:

- **Simplicity:** You can use your existing Python knowledge to build UIs quickly and intuitively since components are just functions.
- **Maintainability:** Complex UIs become easier to manage and understand by breaking them down into smaller, focused components.
- **Modularity:** Components are self-contained, enabling easy reuse within a project or across different projects.

Here's an example of a reusable icon button component:

```
def icon_button(*, icon: str, label: str, tooltip: str, on_click: Callable):
  """Icon button with text and tooltip."""
  with me.content_button(on_click=on_click):
    with me.tooltip(message=tooltip):
      with me.box(style=me.Style(display="flex")):
        me.icon(icon=icon)
        me.text(
          label, style=me.Style(line_height="24px", margin=me.Margin(left=5))
        )
```

Flexibility through Layered Building Blocks

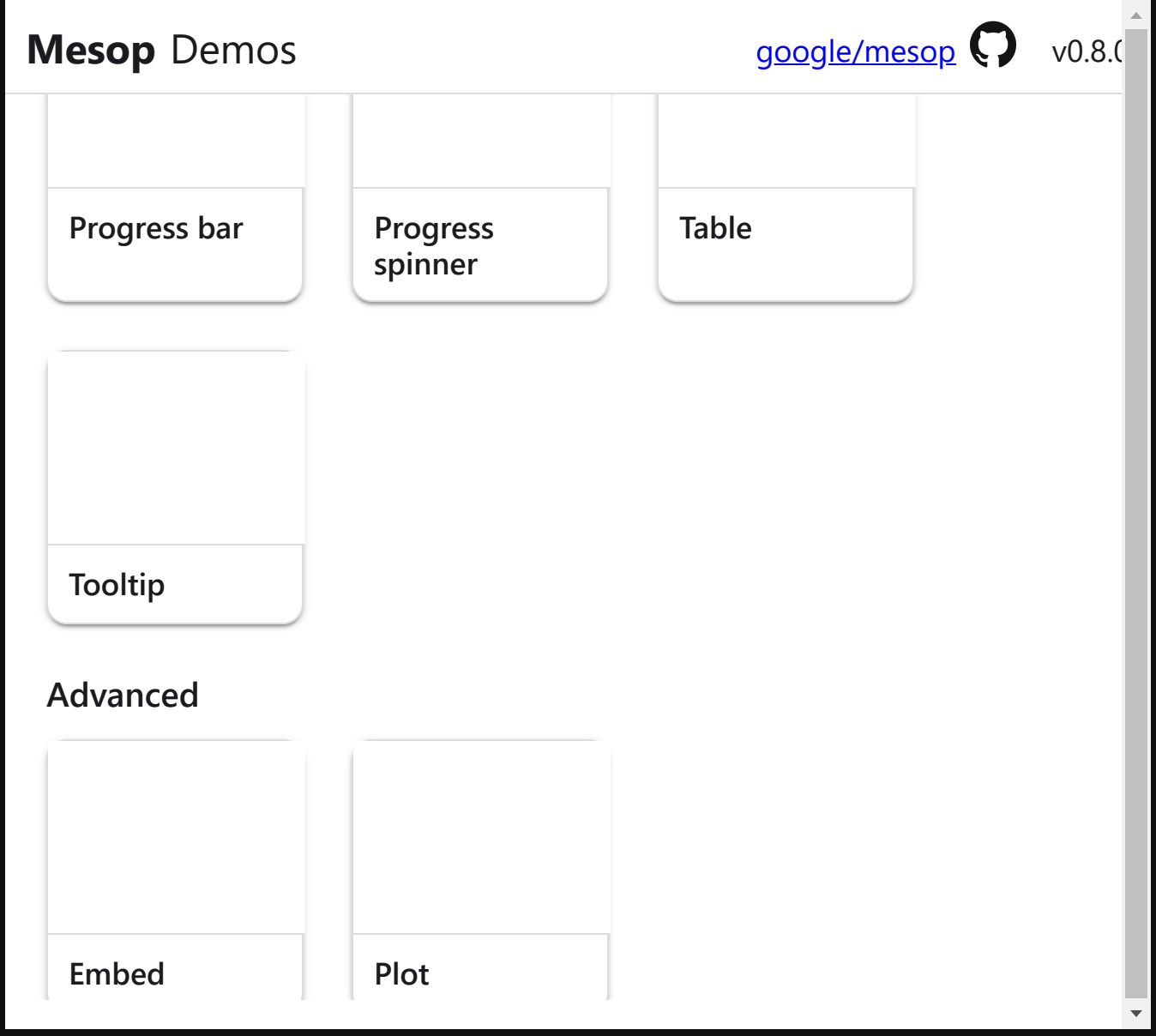
Mesop provides a range of UI building blocks, from low-level native components to high-level components.

- Low-level components: like box, offer granular control over layout and styling. They empower you to create custom UI elements through flexible layouts like flexbox and grid.
- High-level components: like chat, are built from low-level components and provide ready-to-use elements for common use cases, enabling rapid development.

This layered approach makes deep customization possible. This means that if you want to customize the chat component, you can fork the chat implementation because it's written entirely in Python using Mesop's public APIs.

See Mesop in Action

To demonstrate the range of UIs possible with Mesop, we built a demo gallery to showcase the types of applications you can build and the components that are available:



The demo gallery itself is a Mesop app and implemented in a few hundred lines of Python code. It demonstrates how Mesop can be used to create polished, custom UIs in a maintainable way.

Try Mesop

If this sounds intriguing, read the Getting Started guide and try building your own Mesop app. Share your feedback and contribute as we continue developing Mesop.