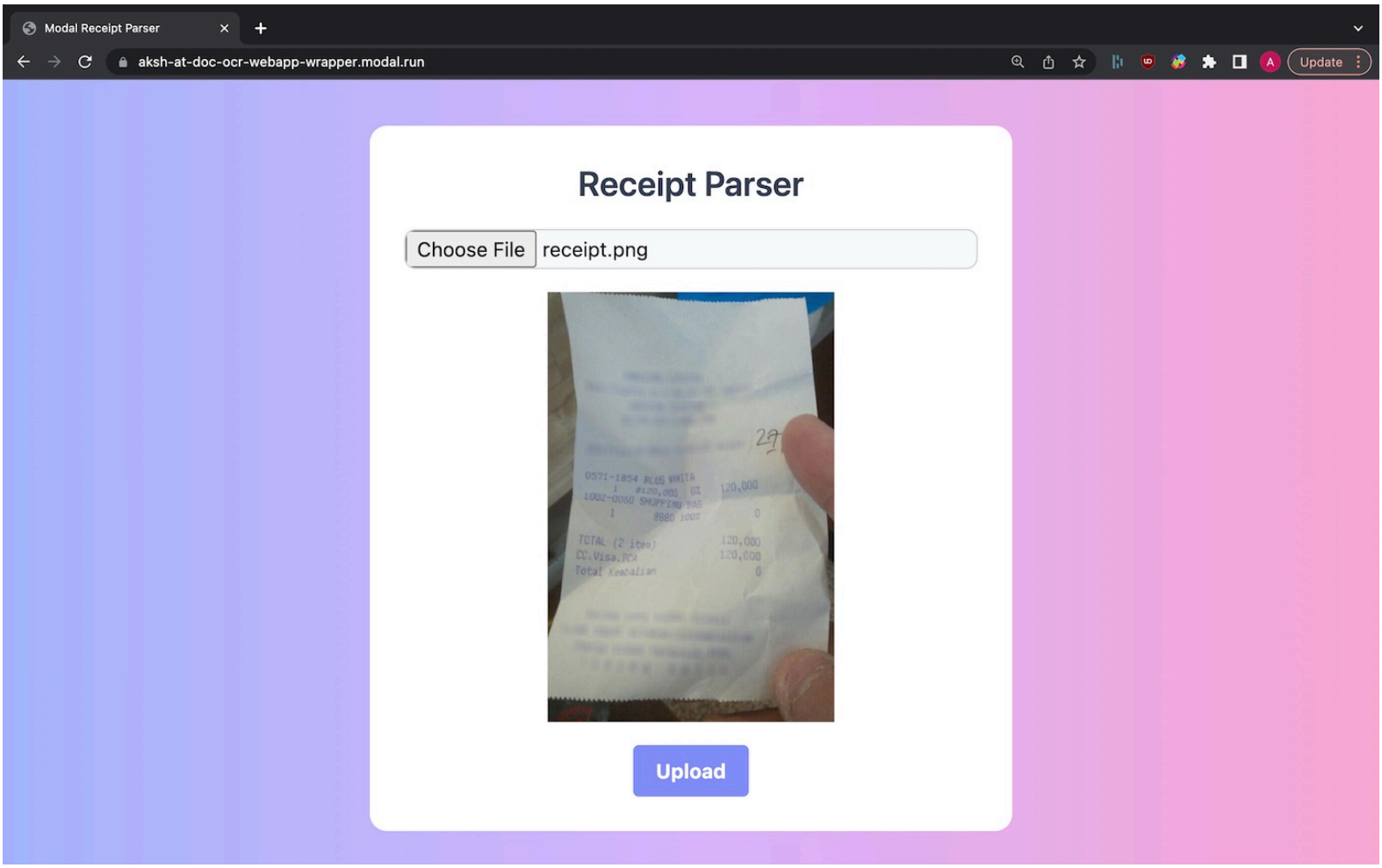
Document OCR web app

View on GitHub

This tutorial shows you how to use Modal to deploy a fully serverless React + FastAPI application. We're going to build a simple "Receipt Parser" web app that submits OCR transcription tasks to a separate Modal app defined in the Job Queue tutorial, polls until the task is completed, and displays the results. Try it out for yourself here.



Basic setup

Let's get the imports out of the way and define a `App`.

```
from pathlib import Path

import fastapi
import fastapi.staticfiles
from modal import App, Function, Mount, asgi_app

app = App(
    "example-doc-ocr-webapp"
)  # Note: prior to April 2024, "app" was called "stub"
```

Modal works with any ASGI or WSGI web framework. Here, we choose to use FastAPI.

```
web_app = fastapi.FastAPI()
```

Define endpoints

We need two endpoints: one to accept an image and submit it to the Modal job queue, and another to poll for the results of the job.

In `parse`, we're going to submit tasks to the function defined in the Job Queue tutorial, so we import it first using `Function.lookup`.

We call `.spawn()` on the function handle we imported above, to kick off our function without blocking on the results. `spawn` returns a unique ID for the function call, that we can use later to poll for its result.

```
@web_app.post("/parse")
async def parse(request: fastapi.Request):
    parse_receipt = Function.lookup("example-doc-ocr-jobs", "parse_receipt")

    form = await request.form()
    receipt = await form["receipt"].read()  # type: ignore
    call = parse_receipt.spawn(receipt)
    return {"call_id": call.object_id}
```

`/result` uses the provided `call_id` to instantiate a `modal.FunctionCall` object, and attempt to get its result. If the call hasn't finished yet, we return a `202` status code, which indicates that the server is still working on the job.

```
@web_app.get("/result/{call_id}")
async def poll_results(call_id: str):
    from modal.functions import FunctionCall

    function_call = FunctionCall.from_id(call_id)
    try:
        result = function_call.get(timeout=0)
    except TimeoutError:
        return fastapi.responses.JSONResponse(content="", status_code=202)

    return result
```

Finally, we mount the static files for our front-end. We've made a simple React app that hits the two endpoints defined above. To package these files with our app, first we get the local assets path, and then create a modal `Mount` that mounts this directory at `/assets` inside our container. Then, we instruct FastAPI to serve this static file directory at our root path.

```
assets_path = Path(__file__).parent / "doc_ocr_frontend"


@app.function(mounts=[Mount.from_local_dir(assets_path, remote_path="/assets")])
@asgi_app()
def wrapper():
    web_app.mount(
        "/", fastapi.staticfiles.StaticFiles(directory="/assets", html=True)
    )

    return web_app
```

Running

You can run this as an ephemeral app, by running the command

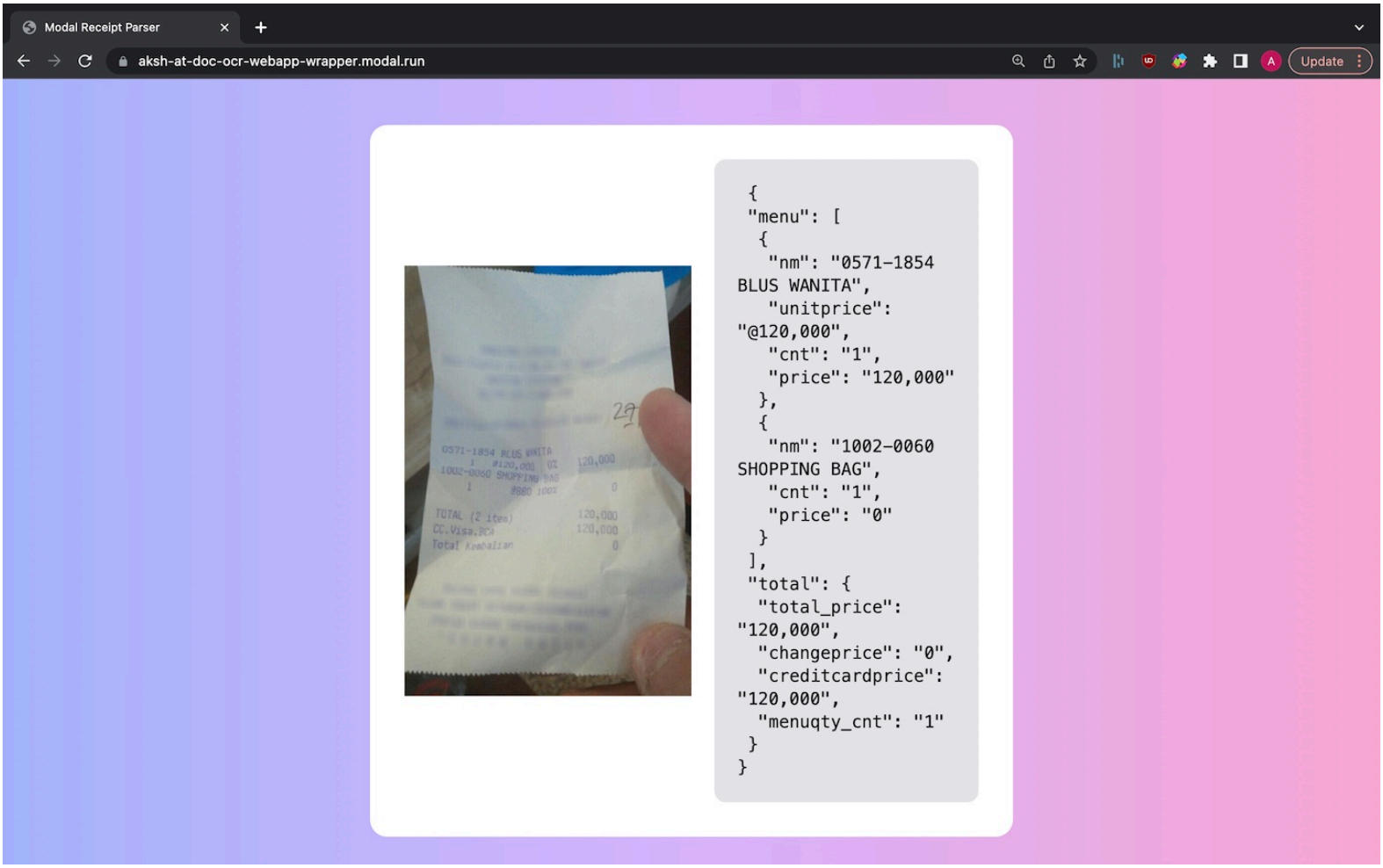
```
modal serve doc_ocr_webapp.py
```

Deploy

That's all! To deploy your application, run

```
modal deploy doc_ocr_webapp.py
```

If successful, this will print a URL for your app, that you can navigate to from your browser 🎉 .



Developing

If desired, instead of deploying, we can serve our app ephemerally. In this case, Modal watches all the mounted files, and updates the app if anything changes.