Ryan Cooke
Posted on Jan 27, 2023 • Updated on Aug 27 • Originally published at jumpwire.ai

Svelte without Kit

#svelte #javascript #programming #tutorial

updated August 2024 with node 22, express 4.19 and html-express-js 3.1

I love Svelte so much that I want to use it in all of my projects! Often these projects are micro apps that don't need a full-fledged app framework - they are a few HTML pages with a couple of server-side API endpoints. SvelteKit ends up being overkill for this use case, as it doesn't need routing, preloading, customized rendering, or pretty much all of the great features provided by SvelteKit.

The Svelte docs don't give a lot of guidance on how to deploy an app without SvelteKit, in fact there's only one paragraph in the docs that talks about how this might work -

> If you don't need a full-fledged app framework and instead want to build a simple frontend-only site/app, you can also use Svelte (without Kit) with Vite by running npm init vite and selecting the svelte option. With this, npm run build will generate HTML, JS and CSS files inside the dist directory.

deployment, as we use slightly different build steps for each of them.

The setup

Express is a great web framework, it's super simple for setting up HTTP endpoints and processing requests/responses. There are a variety of middleware plugins to handle things like authnz and rendering. JSON request bodies are natively supported in the latest version of Express. While middleware can significantly extend the capabilities of Express, in this setup we are looking for absolute minimalism. We do need a rendering engine so that we can inject data as properties into our Svelte components from routes on the server. But that's about it.

Also critical to get working is the build step for using Svelte. Since Svelte is pre-compiled, it's not as simple as serving Svelte files directly through Express. Instead those assets must be built by Vite before they can be served to the browser client. Vite will package all of our Svelte components into minified Javascript and CSS files, along with any JS dependencies used by components *as well as* CSS libraries like Tailwindcss with Postcss.

To manage the compilation step, we are going to use some scripting in our local dev environment. It will trigger a build command whenever a `.svelte` file is created/modified/deleted, and copy the generated files for Express to serve. (Unfortunately, there's no hot-reload in the browser). For deployment, we'll compile Svelte as a step in building a Docker container to run on a hosting provider.

Express with html-express-js

Our app is a simple chatbot that receives a message from the Svelte front-end and echos back a random response. It stores chat sessions with a random identifier in memory. It's not a super complicated app, but we need to route a chat session based on an ID in the HTTP request path. This ID is an example of server-side state that is loaded as properties by Svelte.

We configure Express with a bit of middleware - `cors` and `html-express-js`, as well as built-in middleware `express.static` and `express.json`.

`express.static` will serve the compiled Svelte js/css from a local `./public` directory. `json` and `cors` are necessary to allow for fetching data from our Svelte components.

Here's the basic Express backend, as `server.js` -

```
// ./server.js
import express, { response } from 'express';
import htmlExpress, { renderView } from 'html-express-js';
import cors from 'cors';
import { resolve } from 'path';
import fs from 'fs';

const port = process.env.PORT || 3000;
const hostname = process.env.HOSTNAME || 'http://localhost'

const __dirname = resolve();

const app = express();
// Configure express to parse json and allow CORS requests
app.use(express.json());
app.use(cors());

// Configure express to serve static assets from
// the local directory ./public. This is where
// compiled Svelte files and images will get copied into.
app.use(express.static('public'));

// Store chats in local memory
const chats = {};

// Configure html-express-js as the rendering engine.
// Templates will be in the local directory ./views
const { engine, staticIndexHandler } = htmlExpress({
  viewsDir: `${__dirname}/views`
})
app.engine('js', engine)
app.set('view engine', 'js')

// POST endpoint for the front-end to send chats
// The front-end sends a list of all chats on each request
app.post('/:cid', function(req, res) {
  chats[req.params.cid] = req.body;
  res.send("Ok");
});

// GET endpoint for the front-end to load chats by identifier
// The list of chats is sent as a JSON array
```

```
    cachedchats = chats[req.params.cid]
  }

  res.send(cachedchats);
})

// GET endpoint for loading HTML from the template in views/homepage.js
// If a chat identifier is in the path, parse as :cid and render in template
app.get(['/', '/:cid'], function (req, res, next) {
  renderView('homepage', req, res, {
    title: 'Very Cool Chatbot',
    cid: req.params.cid
  });
});

app.listen(port, () => console.log(`Node app listening on port ${port}!`));
```

Take note of the GET endpoint for the root path - it also parses a `cid` parameter from the path and passes that into the `html-express-js` template. You'll see that parameter used in both the template as well as the root Svelte component below.

Svelte standalone

We set up the Svelte project in a subdirectory `./svelte` using Vite, as recommended in the docs -

> you can also use Svelte (without Kit) with Vite by running `npm create vite@latest` and selecting the Svelte option.

After initializing and running a build using `npm run build`, we can take a peek at the generated assets under `./svelte/dist`. Of note is the HTML file that gets created, it's actually pretty simple -

```
<!-- ./svelte/dist/index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- meta tags -->
    <script type="module" crossorigin src="/assets/index-acdd3185.js"></script>
    <link rel="stylesheet" href="/assets/index-972eb3c3.css">
  </head>
  <body>
    <div id="app"></div>
```

The HTML references a `.js` and `.css` file that are created during the build step. And there is a single `<div id="app"></div>` that is used by Svelte to inject the compiled HTML/JS/CSS components that serve as the main web app. This means we can use the same compiled JS/CSS files in any HTML file, as long as there is a div with `id="app"` somewhere in the template.

html-express-js

We'll create a similar HTML file as our template that is rendered by Express. It will load the `.js` and `.css` file created by building Svelte, and include a single div with `id="app"`. It also will have data rendered from the Express server backend, remember that path parameter called `cid` from above?

```
// views/homepage.js
import { html } from 'html-express-js';

export const view = (data, state) => html`
  <!DOCTYPE html>
  <html lang="en">
    <head>
      <title>${data.title}</title>
      <meta charset="UTF-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      <!-- Precompiled JS/CSS files by Svelte during build step -->
      <script type="module" crossorigin src="/assets/index-acdd3185.js"></script>
      <link rel="stylesheet" href="/assets/index-972eb3c3.css">
    </head>

    <body>
      <!-- data.cid is the parsed path variable from Express -->
      <div id="app" data-chat-id="${data.cid}"></div>
    </body>
  </html>
`
```

Wiring it all together

How do we access the data from the template in our Svelte component? Here's the App.svelte that loads the root of our front-end app. It uses plain JS/HTML DOM to read the value of the data attribute out of the main app div, selecting by `id="app"`. Super easy!

```
  import Chatbox from "./lib/Chatbox.svelte";

  // Lookup main app div
  const app = document.getElementById("app");
  // Access `data-chat-id` attribute from div
  let chatId = app.dataset.chatId;

</script>

<main class="font-mono text-primary h-full">
    <!-- other markup -->

    <Chatbox {chatId} />
</main>
```

In my testing, this worked great for simple parameters such as strings. For loading more complex JSON data, I found it didn't always render correctly in the `html-express-js` template, especially for embedded string properties that contained quotes. In that case, I would fetch JSON data when mounting the Svelte component instead.

Finally we need to copy the generated JS/CSS files from `./svelte/dist/assets` into `./public/assets` so that they can be served by Express. Given the file names include a random postfix for browser cache busting, we'll add a bit of code to read JS/CSS file names from the filesystem and render them in the template:

```
//./server.js
// ...other includes

import fs from 'fs';

const __dirname = resolve();

function readAssets() {
  const js = fs.readdirSync(`${__dirname}/public`).filter((f) => f.endsWith('.js'));
  const css = fs.readdirSync(`${__dirname}/public`).filter((f) => f.endsWith('.css')

  return { js, css };
}

// ...other setup and endpoints

app.get(['/', '/:cid'], function (req, res, next) {
```

```
    cid: req.params.cid,
    js,
    css
  });
});



// views/homepage.js
import { html } from 'html-express-js'

const renderJs = (js) => {
  return js.map((j) => `<script type="module" crossorigin src="/${j}"></script>`).jo
}

const renderCss = (css) => {
  return css.map((c) => `<link rel="stylesheet" href="/${c}" />`).join('\n');
}

export const view = (data, state) => html`
  <!DOCTYPE html>
  <html lang="en">
    <head>
      <title>${data.title}</title>
      <meta charset="UTF-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      ${renderJs(data.js)}
      ${renderCss(data.css)}
    </head>

    <body data-theme="forest">
      <div id="app" data-chat-id="${data.cid}"></div>
    </body>
  </html>
`
```

DevEx

In review so far - we've set up an Express app to serve static assets, created a stand-alone Svelte app, and added a template that initializes the Svelte app from Vite compiled assets as well as renders data from the server. However this still requires building the Svelte app and copying the compiled asset files into a directory for

First, a simple Makefile to run the Vite build and copy the compiled files into Express asset directory -

```
# ./Makefile
SHELL := /bin/bash

clean:
    rm -rf ./public/* \
    && rm -rf ./svelte/dist/*

build:
    mkdir -p ./public \
    && pushd svelte \
    && npm install \
    && npm run build \
    && popd \
    && rm -rf ./public/* \
    && cp ./svelte/dist/assets/* ./public/ \
    && find ./svelte/dist -name \*.svg -exec cp {} public/ \;

run:
    npm install \
    && node server.js
```

We also create a small shell script called `./watch-svelte` to watch for changes to `*.svelte` files and run the build automatically. This saves us from having to run a command every time we modify a Svelte component -

```
# ./watch-svelte
#!/bin/bash

inotifywait -q -m -r -e modify,create,delete ./svelte | while read DIRECTORY EVENT F
    if [[ $FILE == *.svelte ]]; then
        make build
    fi
done
```

Production build

hosting providers, for this example we deployed to fly.io

The Dockerfile uses a multi-stage build, which helps to keep the resulting container size smaller as we won't include any of the Vite or Svelte dev dependencies.

```
FROM node:22-alpine AS svelte-build

WORKDIR /app

ADD svelte ./
RUN npm install
RUN npm run build

FROM node:22-alpine

WORKDIR /app

ADD public ./public
ADD views ./views
COPY --from=svelte-build /app/dist/assets/* ./public/
COPY --from=svelte-build /app/dist/*.svg ./public/

COPY server.js ./
COPY package.json ./
RUN npm install

EXPOSE 3000

CMD node server.js
```

And that's it! Hopefully this is a useful reference for how you might use Svelte in other micro apps where SvelteKit's capabilities are extraneous.

Happy coding,

- Ryan

Header image generated with the assistance of DALL·E 2

Top comments (2)

DeadpixelDesign • Jul 15 '23

Wow. Thanks for the information! I've been racking my brain as to how to grab the static files as I'm beginning to learn the language, but the focus of most authors is to have Svelte apps run on a node server. You'd think they would highlight this content in a separate area of the docs, but instead it is just listed as an afterthought. sigh.

As a node/svelte nood, I very much appreciate this information. Hopefully, I can also give something back to the dev community as my skills improve.

Ryan Cooke • Sep 5 '23

Ryan Cooke

LOCATION
New York, NY

EDUCATION
Stanford University

WORK
CEO at JumpWire

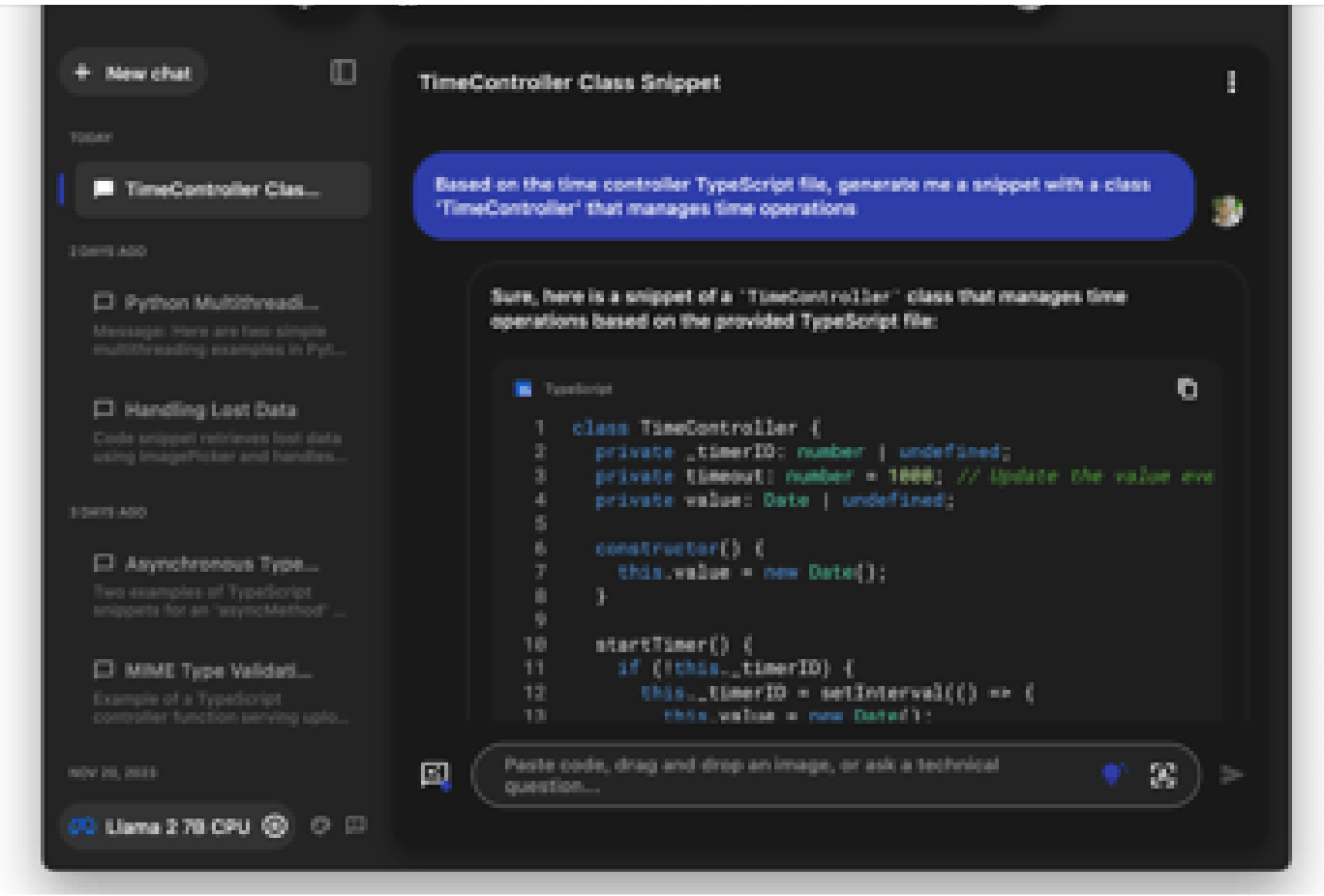
JOINED
Sep 29, 2021

More from Ryan Cooke

State management in Svelte apps
#svelte #javascript #webdev #database

Breaking IndexedDB consistency to explore its transactions
#webdev #idb #javascript #database

E2E Reactivity using Svelte with Phoenix LiveView
#webdev #elixir #svelte #tutorial



Our desktop app, with its intelligent copilot, streamlines coding by generating snippets, extracting code from screenshots, and accelerating problem-solving.

Read the docs