Web patterns in Svelte 5

Table of Contents

1. Patterns

Component Library

publish components as npm package to reuse them in other projects.

Finite State Machine

model complicated flows and keep track of state.

Internationalization (i18n)

allow your site to support multiple languages and locales.

Validation

check if user input is correct and provide errors informations.

Webhook

listen events from outside sources.

2. Component library

2.1. ObjectTable

Render any object as a human readable table.

Installation

```
npm install svelte-object-table
```

Usage :

```
<script lang="ts">
import {ObjectTable} from "svelte-object-table";
import  "svelte-object-table/object-table.css";
// You can also use your own CSS (every css class is global)
</script>

<ObjectTable data={{"foo":"bar","baz":"moo"}}></ObjectTable>
```

Purpose

Quick visualization of data which structure is unknown.

- Showcase how to publish a component library

Patterns used

- Embrace the cascade (CSS)
- Approval testing (snapshot)
- Design system documentation (Storybook)
- Eslint but no Prettier

Publish component library

Content in `src/lib` is published.

It publishes files in `dist` folder.

To simulate publication to npm :

```
npm publish --dry-run
```

To actually publish :

```
npm publish --dry-run
```

Embrace the cascade

Even if it is not idiomatic to Svelte 5 and most Frontend frameworks, I will not use scoped style in this component library.

It means I will not use `<style>` in Svelte components.

This will allow to use CSS cascade to overwrite CSS class of the components.

A pro is I can use vanilla css style without the need of any preprocessor.

Example in object-table/src/lib/ObjectTable.svelte

```
{#snippet table(headers: any, rows: any, summarizeInner: any)}
    <table class="object-table-table">
        <thead>
        <tr>
            {#each headers as header }
                <th>{header}</th>
            {/each}
        </tr>
        </thead>
        <tbody>
        {#each rows as row }
            <tr>
                {#each headers as header}
                    <td>
                        <ObjectTable data={row[header]} summarize={summarizeInner}/>
                    </td>
                {/each}
            </tr>
        {/each}
        </tbody>
    </table>
{/snippet}
```

Style can be then implemented is a plain css file.

object-table/src/lib/object-table.css

```
/* ObjectTable */

table.object-table-table,
.object-table-table td,
.object-table-table th {
    border: lightslategrey solid;
}

table.object-table-table {
    border-collapse: collapse;
}

.object-table-table td {
    min-width: 12rem;
    padding: 1rem;
    overflow-wrap: break-word;
    overflow-x: auto;
}

.object-table-table th {
    padding: 1rem;
}
```

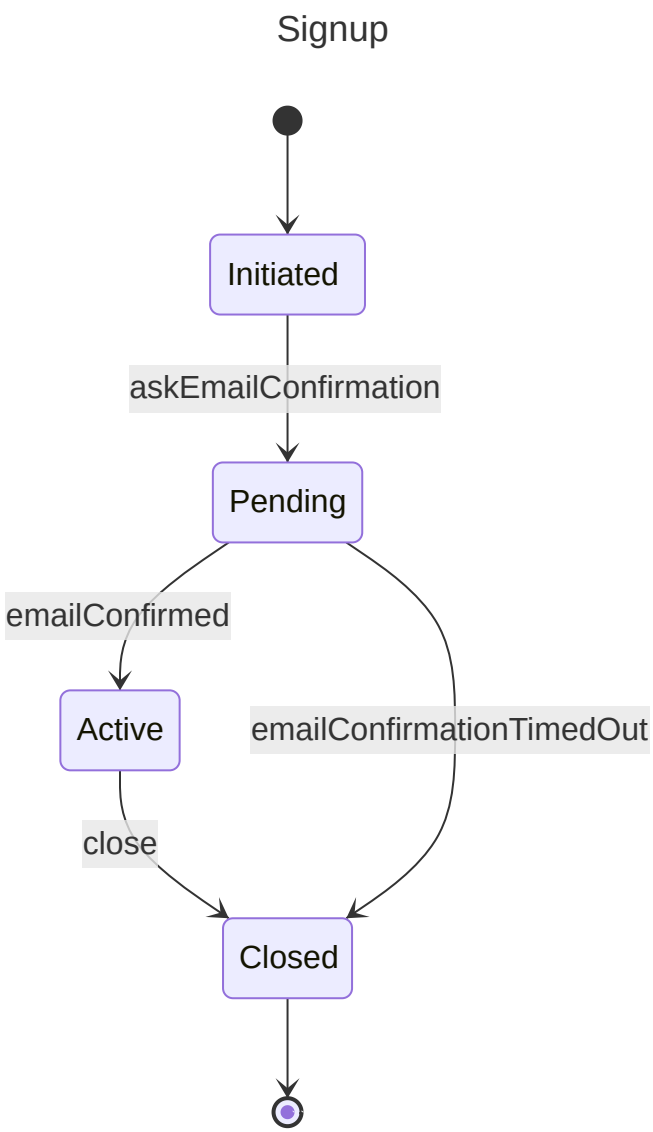
And imported in the object-table/src/routes/+page.svelte

```
<script lang="ts">
    // Use bundled style :
    // import  "svelte-object-table/object-table.css";
    // Or your own :
    import "../object-table.css"
    import {ObjectTable} from "$lib/index.js";
</script>
```

3. Finite state machine

Example of a finite state machine for a signup workflow as a Mermaid diagram.

```
---
title: Signup
---
stateDiagram-v2
  [*] --> Initiated
  Initiated --> Pending: askEmailConfirmation
  Pending --> Active: emailConfirmed
  Active --> Closed: close
  Pending --> Closed: emailConfirmationTimedOut
  Closed --> [*]
```



Appendix A: Nodejs ecosystem

A.1. NVM

Node version manager

Use node version specified in a `.nvmrc` file

```
nvm use
```

Example of `.nvmrc` file for node version 22

```
22
```

A.2. npm workspaces

Run command in sub workspace from parent :

```
npm run dev --workspace=full-demo
```

Configuration

Find all workspaces in `package` folder.

In `package.json`

```
{
  "workspaces": {
    "packages": [
      "packages/*"
    ]
  }
}
```

Example of workspaces in `package` folder

```
.
└── packages
    ├── some-module
    └── full-demo
```

Appendix B: Living documentation

B.1. Core Principles of Living Documentation

- Reliable
- Low-Effort
- Collaborative
- Insightful

Living Documentation Core Principles

Awesome Living Documentation

B.2. Examples

- Linters
- Diagram as text/code (ex. [Mermaid](mermaid-diagram.md))
- doctests
- ubiquitous language (Domain-Driven-Design)
- Domain Specific Languages (DSL)
- Living Glossary

B.3. Mermaid diagrams

Mermaid is a tool to generate diagrams as code.

https://mermaid.js.org/

Useful to generate living diagrams (Living Documentation).

B.4. Approval testing

Approval testing.

Implementation for nodejs

Generate living diagrams with mermaid

Approvals can generate text and [Mermaid Diagrams](mermaid-diagram.md) are text. So can generate living diagrams (Living Documentation) with approvals.

Usually approvals implementation can support custom file extension (). But unless it is pushed, it is not native yet in approvals for nodejs.

While waiting for the evolution to be merged, you can use existing API to verify with custom file extensions.

An example of such utility :

```
import * as approvals from 'approvals'
import StringWriter from 'approvals/lib/StringWriter'
import path from 'node:path'
import { Namer } from 'approvals/lib/Core/Namer'

export function verifyWithExtension(
    dirName: string,
    testName: string,
    text: string,
    extensionWithoutDot: string = 'txt'
) {
    approvals.verifyWithControl(
        namerWithExtension(dirName, testName),
        new StringWriter(approvals.getConfig(), text, extensionWithoutDot),
        null,
        approvals.getConfig()
    )
}

function namerWithExtension(dirName: string, testName: string): Namer {
    return {
        getApprovedFile(ext: string): string {
            return path.join(dirName, testName + '.approved.' + ext)
        },
        getReceivedFile(ext: string): string {
            return path.join(dirName, testName + '.received.' + ext)
        }
    }
}
```

Usage to generate a Living Diagram in mermaid.

```
    it('Draws living documentation mermaid diagram', async () => {
        const stateMachine = new SignupFsm(Initiated, callbacks)

        const mermaidDiagram = stateMachine.toMermaid('Signup')
        verifyWithExtension('docs/diagrams', 'signup-fsm', mermaidDiagram, 'mermaid')
    })
```

Will generate and verify against files : - `docs/diagrams/signup-fsm.approved.mermaid` - `docs/diagrams/signup-fsm.received.mermaid`

Configuration override

`~/.approvalsConfig`

```
{
  "reporters": [
    "BeyondCompare",
    "diffmerge",
    "p4merge",
    "tortoisemerge",
    "nodediff",
    "opendiff",
    "gitdiff"
  ],
  "normalizeLineEndingsTo": false,
  "failOnLineEndingDifferences": false,
  "appendEOL": true,
  "errorOnStaleApprovedFiles": true,
  "stripBOM": false,
  "forceApproveAll": false,
  "blockUntilReporterExits": false,
  "maxLaunches": 10
}
```