App Organization and History API

Michael Chang

Spring 2023

Full stack topics

(Today) Web app organization

(Thursday) Authentication and users

CSS techniques and mobile styling

Accessibility

Week 10

Frontend frameworks, deployment, advanced techniques, wrap-up

Plan for today

Web app structure

- Definition: single page application
- Different strategies and their tradeoffs

location and history API

- Changing URL without reloading page

Storing data across HTML pages

Note: route ordering

Express checks routes one by one

- First added -> first checked
- Uses first match

Example

```
api.post("/binky", ...);
api.use(bodyParser.json());
```

- `req.body` not available in POST /binky route

```
api.all("/*", ...); /* handle all requests */
api.get("/", ...);
```

- Second handler never called

Web app structure

How many HTML pages?

Traditional websites: one page per "view"

- Pages might have dynamic content filled in by server

Single page application (SPA)

- Only one HTML file; elements shown/hidden via JS
- Dynamic content via fetch
- Popular with frontend frameworks

Modern non-SPAs

- Use fetch for dynamic content
- Some showing/hiding to respond to events (user) or data (server)
- If page sufficiently different, can make a new HTML file

Browser URL bar

Current model

- URL shows exact file name the server will return
  - E.g. `http://localhost:1930/social.html`
- Links point to specific HTML pages
  - E.g. `<a href="social.html">Part 3</a>`

Limitations

- No dynamic URLs, sharable links
  - E.g. `http://localhost:1930/profile/mchang`
- Changing URL -> reload from server
  - No smooth/fancy transitions, loading bars...
- When page is an SPA, no "deep links"
  - Can't link to parts of app; can't use back button

Multi-step approach

0. Stick to current organization. It works fine.

For informational pages, pointing at HTML files is fine

If only a couple views, no need for deep links

0.5. Use URL hash

Hash part never sent to server; changing won't reload

E.g. `http://localhost:1930/social.html#mchang`

1. Server maps multiple URLs -> same HTML

Use `location` in frontend to fetch/display content

2. Use `history` API to change URL

Change URL without reloading page

Frontend: location

`window.location`: info about the loaded URL

`.href: "http://localhost:1930/index.html?binky=winky#foo"`

- `.protocol`: "http:"
- `.host`: "localhost:1930"
- `.pathname`: "/index.html"
- `.search`: "?binky=winky"
- `.hash`: "#foo"

`.assign(url)`

Navigate to this URL (loaded from server)

`.replace(url)`

Replace URL (load from server, can't "Back" to current URL)

Backend: sending files

```
const PUBLIC_PATH = path.join(process.cwd(), "public");
app.get("/profile/:id", (req, res) => {
  res.sendFile("profile.html", { root: PUBLIC_PATH });
});
```

`res.sendFile(path, options)`

Send the file as a response

`path` must be absolute, or `options.root` must be set

- For security, probably best to always use root

Careful: if path is coming from user, could introduce security issues

- E.g. "send me the file `../../../secret.txt`"
- Using `root` helps, but still should be careful

Frontend: history API

`window.history`: interact with browser's URL bar and history

`.pushState(state, title, url)`

- Change URL bar without loading page
- No one uses title (pass "")
- Can pass "state" that isn't shown in URL
  - Accessed through `history.state`

`.replaceState(state, title, url)`

- Replace history entry with new one (no load, can't Back to current page)

`.back, .forward, .go`

- Programmatically move through browser history

Frontend: history API

`popstate` event on `window`

Fired when user (or program) moves through history (back, forward, etc.)

NOT fired when pushState or replaceState

Example

```
window.addEventListener("popstate", (event) => {
  console.log(`Location: ${location.pathname}`);
  loadProfile();
});
```

Saving state across pages

`window.sessionStorage`

- Allows you to save data on one page and read it on another
- Or save state while refreshing the page
- Reminder: frontend only, backend is stateless

Example

```
window.sessionStorage.setItem("user", "mchang");

let user = window.sessionStorage.getItem("user");

window.sessionStorage.removeItem("user");
```

Summary

Today

- Assorted things that may be relevant to your project

Before next time

- assign3.2 ongoing

Next time: auth and users

- Tracking who's logged in, API tokens
- Using 3rd-party auth (like Google), Oauth
- Security considerations