JavaScript
Frameworks

Bjarte Wang-Kileng

HVL

September 5, 2025

Western Norway University of Applied Sciences

Outline

Why framework?

- Tried and tested tools for building scaleable, interactive web applications.
- Tools to develop advanced web applications, especially single page applications.
- Create applications that work across different versions of browsers.

Multi Page Applications (MPA)

- Traditional web application.
- Browser will load a new page from webserver on user navigation.
  - Navigation e.g. by links (tag *A*) and submit of form (tag *FORM*).
- If form data is submitted, web server returns a new web page.
  - Can be constructed using data received from *FORM* element.
- Current page is removed from browser, and replaced with new page.
- Each page has a unique URL.
- Require network to load new page.

Single Page Applications (SPA)

- Uses a single web page for all of the application.
- JavaScript is used to modify the view.
- Data can be fetched from server using Ajax or websockets.
- Navigation to application pages will run JavaScript code.
  - Default navigation action is captured and aborted.
- By default, the same URL is used for all of application.
  - A *router* though allows multiple URLs, one for each application state.

Multi- and Single Page Applications

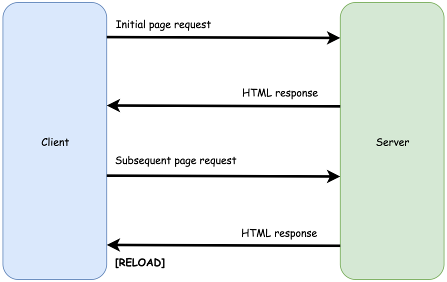


Figure: Multi Page Application

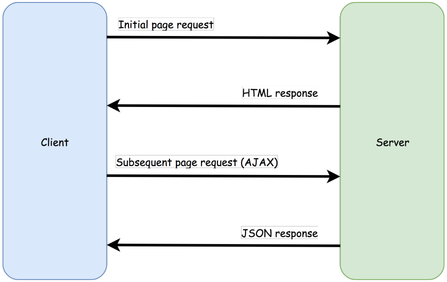
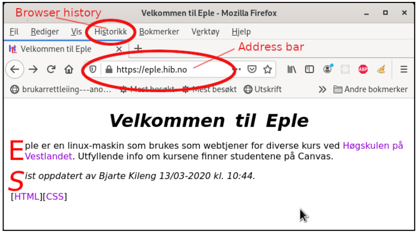


Figure: Single Page Application

Router library

- Mimics browser navigation.
- Matches URL to application state.
- Manages address bar of browser.
- Manages browser history.


Browser history
Velkommen til Eple - Mozilla Firefox
Fil Rediger Vis Historikk Bokmerker Verktøy Hjelp
Velkommen til Eple
Address bar
https://eple.hib.no
brukarrettleiing---ano… Mest besøkt Mest besøkt Utskrift
Andre bokmerker

Velkommen til Eple

Eple er en linux-maskin som brukes som webtjener for diverse kurs ved Høgskulen på Vestlandet. Utfyllende info om kursene finner studentene på Canvas.

Sist oppdatert av Bjarte Kileng 13/03-2020 kl. 10:44.

[HTML][CSS]

Figure: History and address bar

Routing

- Each SPA state should have its own URL.
  - Different URLs represent different states of same page.
- When URL in address bar is changed, view is modified by JavaScript.
  - No new web page is loaded from web server.
- Supported by browsers using URLs with hash sign “#”.
- Possible also with normal URLs, i.e. no hash sign.
  - Require JavaScript for routing on client, and preferably also
  - server side tools for routing on web server.
- Lift routing out of component if accessing external elements.
  - E.g. if modifying *document.title*.

SPA pages

- Some GUI components and HTML are common for all routing URLs.
- Other GUI components are replaced on a new URL.
- An SPA page is a component that can be fetched from web server.
  - The component has its own URL.
  - Fetched by SPA from web server using Ajax or websockets.
  - Not to be confused with a web page!
- See e.g. the Next.js definition of Pages.

Routing using hash sign “#”

- A hash appended string is not considered part of web page address.
  - No page loading if only hash part of URL in address bar is modified.
- Event **hashchange** fires when hash part in address bar changes.
- Examples of URLs that cause no navigation from page **index.html**:
  - <a href="`#`">Goto main page<\a>
  - <a href="`index.html#`">Goto main page<\a>
  - <a href="`#pageOne`">Goto page one<\a>
  - <a href="`index.html#pageOne`">Goto page one<\a>
  - <a href="`#\user\ole`">Get info on user<\a>
- Links above will change the address bar URL, but not load page.
  - If hash-part equals an HTML ID, browser will scroll to that element.
- Observer, browser will always navigate if no “#”, even if same URL.

Routing on normal URLs

- Navigation must be aborted on URLs to application pages.

```
event.preventDefault();
```

  - E.g. click event on *A* tags, and submit event on *FORM* elements.

- JavaScript of event handler must modify the view.
- Event handler must update browser address bar with new URL.

```
history.pushState("","",newURL);
```

  - Does not load page from server.

- All URLs must point to same resource on server.
  - Web page is only loaded on first visit, with data set to current state.
  - Data for other states can be fetched on demand by client using Ajax.
- Manual update of address bar will inevitably load a page from server.

Benefits of single page applications

- Improved user experience (UX).
  - No page refresh on navigation.
  - HTML, CSS and JavaScript are loaded only once.
  - Only loads data that changes between application states.
- Capability to work offline.
  - Application can work with local data stored in cache.
  - Data is copied to server when online.
- For offline support and caching, see also the Service Worker API.

Drawbacks of single page applications

- Difficult search engine optimization.
- Can have a large initial download size.
- Require custom loading states and error messages.
  - No 404 status message to user when Ajax fails.
- Link sharing capability, navigation history and bookmarking require a routing module.

Are frameworks necessary?

- Can make development smoother.
- Many front-end development jobs require framework experience.
- Frameworks have a performance price.
  - Wasted re-renders, redundant listeners, unnecessary and large DOM manipulations.
- Vanilla JavaScript has everything needed for client.
  - But, can require a not to old web browser.

Vanilla JavaScript

Vanilla JavaScript refers to plain, native JavaScript.

- No use of frameworks or external libraries.
- See the Vanilla JS page (page is a joke).

What do frameworks provide?

- Reusable components.
- A template system.
- Synchronization of state and view.
- Routing.

Reusable components

- Covered by previous lectures.
- Available in Vanilla JavaScript with shadow DOM and custom tags.
  - Firefox from November 2018.
  - Chrome from September 2016.
  - Edge from January 2020.
  - Opera from October 2016.
  - Safari from November 2016.

Template system

- A template engine replaces variables in template with actual values.
- A template example using Vanilla JavaScript:

```
const course = {
  "course": "DAT152",
  "start": new Date(2025,7,20)
};

const startday = course.start.toLocaleDateString("en-GB",{weekday: 'long'});

const infoString = `Course ${course.name} starts on ${startday}`;
```

- Vanilla JS will will inject actual values when the template is defined.
  - Template is not altered if variable is changed later.
- A template engine should inject variables when template is used.
  - With vanilla JS, can use a function to produce the string.
- For more info on JS template literals, see e.g. Template literals.

State and view

- State refers to the state of the application.
- Determined by data stored in JavaScript objects and class instances.
  - The *Model* of MV* patterns.
- View is HTML, an HTML fragment or a reusable web component.
  - The *View* of MV* patterns.

Synchronization of state and view

- Synchronization of view from state:
  - Any change in state (*Model*) will update view.
- Synchronization of state from view:
  - Any change in view will update state (*Model*).
- One-way data binding (e.g. MVC):
  - Synchronization of view from state only.
- Two-way (bidirectional) data binding (e.g. MVVM):
  - Both synchronization of view from state, and
  - synchronization of state from view.

One-way data binding

- Example with synchronization of view from state:
  - Changes in a JavaScript Array automatically updates an HTML table.
- Made easy by modern frameworks, e.g. React, Vue.js, Angular.

Two-way (bidirectional) data binding

- Bidirectional (two-way) data binding example:
  - Changes in a JavaScript Array automatically updates an HTML table.
  - Changes in the HTML table updates the JavaScript Array.
- Supported by some frameworks, e.g. Angular and AngularJS.
- React can have two-way data binding on FORM control elements.

Synchronization in Vanilla JavaScript

- Synchronization of view from state:
  - Possible using e.g. Proxy instances or Decorators.
- Synchronization of state from view:
  - Possible using e.g. MutationObserver (or IntersectionObserver).

Synchronize view from state in Vanilla JavaScript

- Proxy instances can trap changes on JavaScript objects.
  - Trap can update the DOM.
  - Caveat, object must be accessed through the proxy.
- Decorators can wrap class instances and class fields in functions.
  - Wrapper function can update the DOM.
  - Yet no support in any browser.
  - Supported by transcompilers, e.g. Babel.

Synchronize state from view in Vanilla JavaScript

- MutationObserver can monitor DOM objects.
  - Can modify state of a JavaScript object on DOM object change.
- IntersectionObserver monitor intersect with display, or DOM object.
  - E.g. trigger when some portion of DOM element is visible in browser.

Routing and frameworks

- Routing by Vanilla JavaScript on client only.
- Need routing support also on server:
  - Server must provide specific state of SPA on first load, by URL.
  - Server must provide difference between states on Ajax requests.
- Examples of frameworks for server side routing:
  - Next.js for React.
  - NuxtJS and Express for Vue.
  - FastBoot for Ember.

Types of frameworks

- Client side frameworks.
- Static Site Generators (SSG).
- Server-Side Rendering (SRR).

Client side frameworks

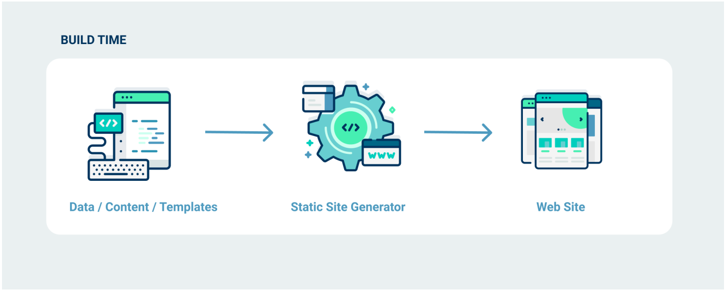
- Client side rendering of SPA views.
- JavaScript library that runs in web browser.
  - Can require a build process to assemble the JavaScript code.
- Avoids code collisions.
  - E.g. adds random characters to CSS class names to avoid collisions.
- Typically no true isolation of components.
  - True isolation possible only through shadow DOM.
- Examples include React, Vue, Angular, AngularJS, Ember.

Static- and dynamic server pages

- Static pages are delivered from server exactly as stored.
    - All clients are served the same view (HTML, CSS, JS).
    - Independent of user input and data from databases and other sources.
        - Can use database and other sources when building the static content.
- Dynamic pages generate view (HTML, CSS, JS) on user request.
    - View generated by server can differ between clients.

Static Site Generators (SSG)

- Applies data and content to templates to build static pages.
  - View (HTML, CSS, JS) is built in advance, prior to client requests.
- Generates all possible application views at build time.
- Pages does not depend on data from request.
- E.g. Hugo, Jekyll, Eleventy, VuePress, Gatsby.



Server-Side Rendering (SSR)

- Traditional approach for dynamic web pages.
    - Multi Page Application.
- View (HTML, CSS, JS) is built on server.
- View is generated per request.
    - Can have dynamic content tailored to client.

SSG and SSR for SPA

- SPA pages are built on server, and fetched using Ajax or websockets.
- Both SSG and SSR can be used for SPA pages.
  - SPA page is created at build time.
  - Page with SSR also includes content created when page is loaded.
- Both SSG and SSR pages can be dynamic on client:
  - JavaScript on client can update the client view.
- Application can include both SSG and SSR pages.
  - Page is either SSG or SSR.
  - For performance, default and recommended in e.g. Next.js is SSG.
- Can combine client side rendering with SSG and SSR pages.
- E.g. Next.js (React), Nuxt (Vue), Express (Vue), Angular Universal (Angular), FastBoot (Ember), Jamstack.

Learn Vanilla JavaScript first!

- Evolution will eventually kill any existing JavaScript framework.
  - True for most JavaScript frameworks that has been taught at HVL.
- More modern frameworks have always replaced older.
- New JavaScript features can make frameworks obsolete.
- Browser incompatibilities was a much bigger problem before.
- Frameworks can have a significant performance cost.
  - See e.g. The Cost of JavaScript Frameworks.

Learn Vanilla JavaScript first

Do not let your JavaScript knowledge depend on a specific framework.

Choosing a JavaScript framework

- React is the most used, but Vue will probably pass React in 2025.
  - Web framework rankings.
- Many comparisons of frameworks can be found on the net, e.g.:
  - Introduction to client-side frameworks.
- We will meet React (next lecture).