Progresivní technologie v informatice I

Webové aplikace

České vysoké učení technické v Praze
Fakulta informačních technologií

Osnova

Webové služby

- REST
  - Požadavky
  - Klíčové elementy
  - Rozhraní
  - Návratové kódy
- GraphQL

Webové aplikace

- Single Page Application
- Rapid Application Development
- Moderní přístupy
- Progressive Web Application
- WebSocket

Webové služby

- komunikace dvou a více aplikací
- vzájemné volání => **API**
- standardizované

Standardy

RPC
: výměna zpráva popisujících volání jednotlivých metod

SOAP + WSDL
: komplexní model webové služby a její definice

REST
: zaměřeno na CRUD operace se zdroji

GraphQL
: soustředěno na retrieval (*mutations*)

RPC

- remote procedure call

Různé varianty

- RPC (nativní podpora, př. Java)
- XML-RPC
- gRPC (binární, vhodné pro micro-services)

Ukázka SOAP

```
<soap:Envelope
  xmlns:soap="http://schemas.xml.soap.org/soap/envelope/
  <soap:Body>
    <GetAccountTransactionsRequest
      xmlns="http://bank.example/api">
      <AccountId>b43dae20-5f7c-11ee-8c99-0242ac120002</A
    </GetAccountTransactionsRequest>
  </soap:Body>
</soap:Envelope>
```

Základy REST

- Representational State Transfer
- softwarová architektura pro distribuované hypermediální systémy
- dizertační práce R. Fieldinga
- popis fungování webu - (HTTP/1.0 -> HTTP/1.1)

REST - cíle

- obecné rozhraní
- škálovatelnost
- nezávislé nasazení modulů
- middleware (mezi-komponenty)

REST - požadavky

- klient – server
- vrstvená architektura
- uniformní rozhraní
- bezestavovost
- cachovatelnost
- code on demand *(optional)*

Klient – Server

- separace zodpovědností

Vrstvená architektura

Klient není schopen rozlišit:

- load-balancing
- proxy server
- autorizační server
- skutečný server

Snadná rozšiřitelnost.

Uniformní rozhraní

- založené na zdrojích (resources)
- manipulace pomocí reprezentace
- samopopisné zprávy
- **HATEOAS (Hypermedia as the Engine of Application** State)

Bezestavovost

- požadavek obsahuje všechny informace
- klient je pasivní, střídají se fáze
  - odpočívá (nevyužívá síť, server, . . .)
  - přechodu (mezi stavy aplikace, odeslání požadavku a jeho vyřízení)

Cachovatelnost

- většina požadavků je získání dat => **cache**
- `Cache-Control: no-store` (bez cache) :(
- `private` vs `public` (proxy)
- `max-age=<seconds>` (životnost cache)
- `must-revalidate` (validace)

- `ETag: W/"<hash>"` vs `If-None-Match: W/«hash»"`
- `ETag: "<hash>"` vs `If-None-Match: «hash»"`
- `Last-Modified: <time>` vs `If-Modified-Since: <time>`

Code on Demand

- není povinné
- poskytovat část kódu (např. JS)
- vytváří black-box

Elementy

zdroj (resource)
: konceptuální entita či skupina entit

resource identifier
: URL, URN

reprezentace
: HTML, XML, JSON, JPEG, PNG

metadata reprezentace
: media type, čas poslední úpravy

metadata zdroje
: zdrojový odkaz, `Alternates`, `Vary`

Identifikace zdroje

- výstižný název
- **unikátní identifikace zdroje**
- hierarchická struktura (`/book/[id]/authors`)
- dokumentace
- singulár (`/book/[id]`) vs. plurál (`/books/[id]`)

Rozhraní

- implementováno pomocí metod
- URL pouze identifikuje zdroj, metody reprezentují akci
- z formulářů známe GET a POST
- kolekce vs. konkrétní instance

Metody

bezpečné (safe) nemění stav zdroje
většinou pouze pro čtení, lze zacachovat

idempotentní volání metody má na zdroji stejný efekt
nastav hodnotu na 10 vs *inkrementuj hodnotu o 1*

GET je *idempotent* i **safe**

PUT je *idempotent*

DELETE je *idempotent* (a co vícenásobné volání?)

POST není ani safe ani idempotent

PATCH není ani safe ani idempotent *(proč?)*

Rozhraní (konkrétní instance)

GET	získání (200 OK, 400 Not Found, . . .)
POST	*nepoužívá se* (405 Method Not Allowed)
PUT	náhrada prvku, vytvoření prvku (200 OK, 201 Created, 204 No Content, . . .)
DELETE	smazání prvku (200 OK, 204 No Content, . . .)
PATCH	úprava části prvku (200 OK, 204 No Content, . . .)

Rozhraní (kolekce)

GET	získání (200 OK)
POST	vytvoření prvku (201 Created)
PUT	*nepoužívá se, příp. nahrazuje celou kolekci* (405 Method Not Allowed)
DELETE	*nepoužívá se, příp. smaže celou kolekci (ne obsah)* (405 Method Not Allowed)
PATCH	*nemá standardizované chování*

Návratové kódy (2xx)

200 OK

- GET zdroj získán, reprezentace v těle zprávy
- PUT úspěch, v těle zprávy je výsledek akce
- POST

201 Created (typicky POST, občas PUT)

204 No Content (*Proč žádný obsah?*)

Návratové kódy (3xx)

300 Multiple Choice
více možných odpovědí, server nedokáže rozhodnout

301 Moved Permanently
URL se změnila trvale, měli bychom používat novou

302 Found
URL se změnila pouze dočasně, použij GET metodu (moderní prohlížeče -> 303)

303 See Other
HTTP/1.1, zdroj lze nalézt na jiné URL

304 Not Modified
cachování – zdroj se nezměnil, použij cache

307 Temporary Redirect
podobné 302, ale respektuje použitou metodu

308 Permanent Redirect
podobné 301, ale respektuje použitou metodu

Návratové kódy (4xx)

400 Bad Request

401 Unauthorized
neautentizovaný

403 Forbidden
neautorizovaný

404 Not Found

405 Method Not Allowed

406 Not Acceptable
klient akceptuje jen HTML, server poskytuje jen JSON

410 Gone
permanentně odstraněno

412 Precondition Failed
konkurenční editace s cache

Návratové kódy (5xx)

500 Internal Server Error

501 Not Implemented

503 Service Unavailable
něco předělláváme, zopakuj `Retry-After`

505 HTTP Version Not Supported

GraphQL

- architektura
- dotazovací jazyk
- zaměřeno na "variabilitu"uživatele služby
- podpora pro notifikací (ale...)
- založeno na typech

Problémy:

- cachování
- verzování
- modifikace
- vlastní formát

Typy

```
type Movie {
    id: ID
    title: String
    actors: [ Actor ]
}

type Actor {
    id: ID
    name: String
    movies: [ Movie ]
}
```

Dotazy

```
type Query {
    movie(id: ID!): Movie
}

...
GET /graphql?query={movie(id:
"12345"){title,actors{id,name}}}
```

Dotazy

```
type Query {
    movie(id: ID!): Movie
}

...

POST /graphql HTTP/1.1
Content-Type: application/json

{
    "query": "{
        movie(id: \"12345\") {
            title,
            actors {
                id,
                name
            }
        }
    }"
}
```

Mutations I

```
type Mutation {
  addActorToMovie(id: ID!, actor: Actor!): Movie
}
```

Mutations II

```
POST /graphql HTTP/1.1
Content-Type: application/json

{
  "query": "mutation AddActorToMovie($movie: ID, $actor:
    addActorToMovie(movie: $movie, actor: $actor) {
      actors
    }
  }",
  "variables": {
    "movie": \"12345\",
    "actor": { ... }
  }
}
```

Dynamické HTML

- interaktivní webové stránky

HTML	informace
CSS	prezentace
JavaScript	skripty na straně klienta
DOM	model dokumentu

- dynamické webové stránky
  - změna při načtení (PHP, ...)

Dynamické HTML

- nekompatibilita napříč prohlížeči
  - dnes už "minulost"?
  - standardizace
- unobtrusive JavaScript, DOM scripting
  - oddělení funkcionality od webové stránky a prezentace
  - best practices
    - používání event modelu podle W3C specifikace
    - detekce schopností prohlížeče
    - zapozdření, abstrakce, konvence, návrhové vzory, testování, ...
  - progressive enhancement
    - základní obsah je k dispozici vždy
    - dále dle možností klienta

Standardy

W3C	DOM Level 3, Web IDL
ECMA	ECMAScript

ECMAScript

- standardizovaný jazyk pro skriptování na straně klienta
  - počátky v roce 1996
  - ECMA-262, ISO/IEC 16262
- dialekty
  - JavaScript, JScript, ActionScript, ...
  - často poskytující funkcionalitu nad rámec normy
  - -> nekompatibilita

ECMAScript

- imperativní, známá syntaxe - Java, C
- dynamicky typovaný (run-time evalutation)
- funkcionální (vnořené funkce, closures, lambda)
- prototypovaný
- run-time prostředí - DOM

JavaScript

- implementace ECMAScript
- `https://developer.mozilla.org/en/JavaScript`
  - referenční příručka
  - učebnice

Propojení s HTML

- linkování

```
<script src="scripts/modal.js"></script>
```

- klasicky (blokující)
  - <head>
  - <body>
- async
  - stáhne JS asynchronně
  - spustí hned po stáhnutí
- defer
  - stáhne JS asynchroně
  - spuštění počká na načtení dokumentu

Propojení s HTML

- linkování
- script v HTML

```
<script>
  window.addEventListener("load", () => {
    console.log("Page is fully loaded.")
  })
</script>
```

Propojení s HTML

- linkování
- script v HTML
- inlinování

```
<button onclick="console.log("Clicked!")"></button>
```

Propojení s HTML

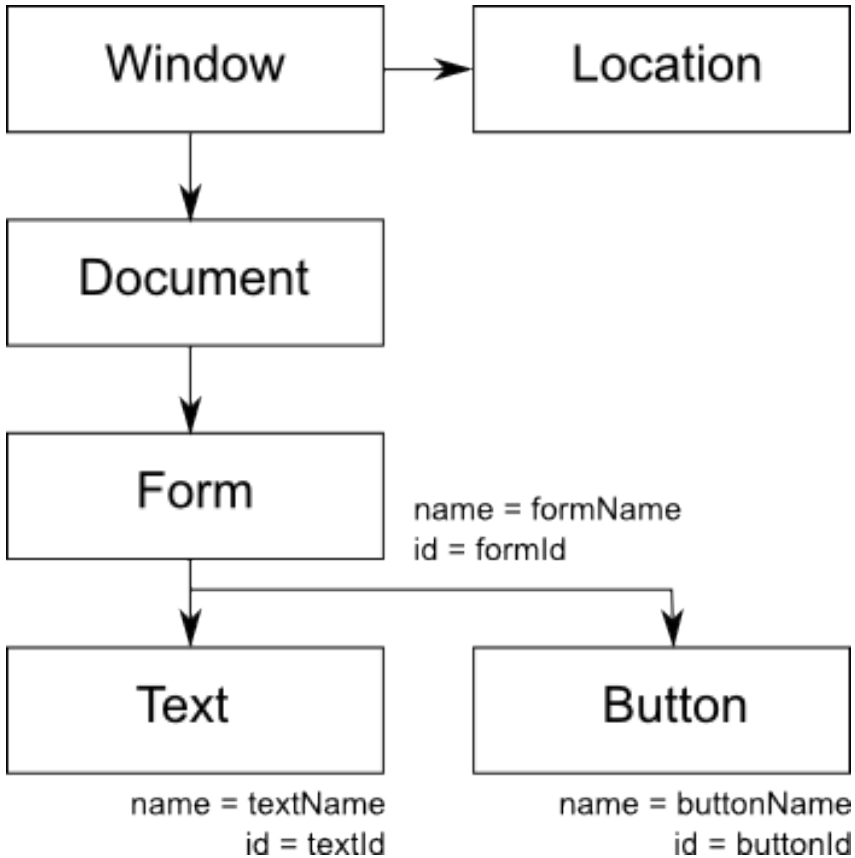
- linkování
- script v HTML
- inlinování
- `<noscript>`

Document Object Model

- model reprezentující HTML/XHTML/XML dokument
  - -> API
- nezávislý na platformě/jazyku
- standardizace - W3c

  1998 DOM Level 1
  2000 DOM Level 2 - getElementById()
  2004 DOM Level 3 - XPath, keyboard events, serializace do XML
- `http://www.w3.org/TR/DOM-Level-3-Core/core.html`
- `http://www.w3schools.com/jsref/dom_obj_document.asp`

DOM - ukázka

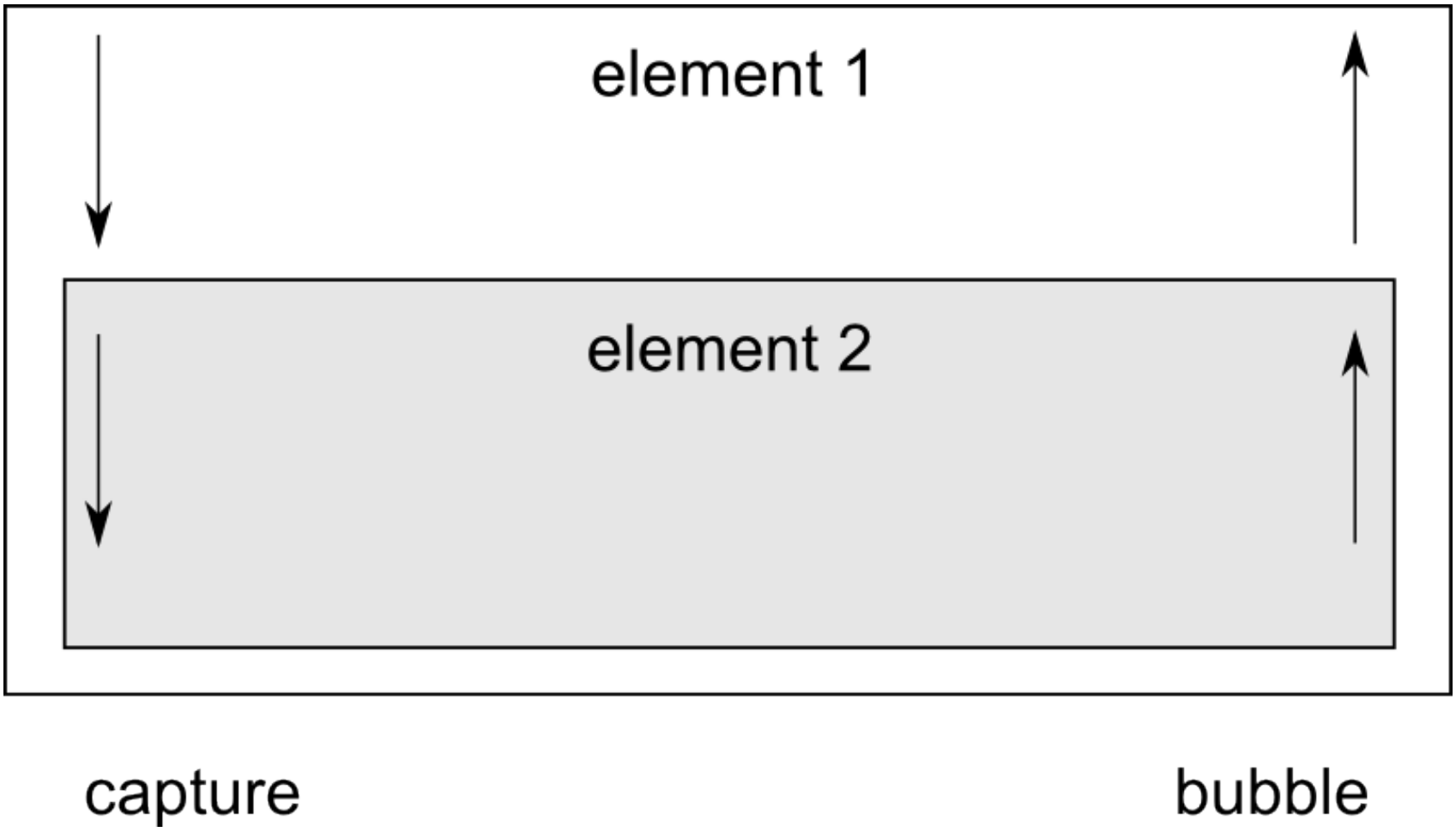


Obrázek: Ukázka DOM modelu

Události

- myš, klávesnice, UI, ...
- formulář
  - `submit, reset, change, select, focus, blur`
- W3C standardizované metody pro registraci posluchačů
  - `addEventListener, removeEventListener, dispatchEvent`
- tok událostí
  - podle W3C (capture -> bubble)
  - IE8 (pouze bubble)
  - Netscape (pouze capture)
  - dnes již standard

Capture & Bubble



Obrázek: Porovnání capture a bubble fáze.

Registrace - inline

```
1  <a href="link.html"
2     onClick="alert('Kliknuto!')">
3
4  <form onSubmit="validate()">
```

- nevhodné
- mixuje se HTML a JS

Registrace - programová (DOM)

```
1  document.anchors[0].onclick = function()
2    { alert('Kliknuto!'); }
3
4  form.onsubmit = validate;
```

- lze vyvolat programově (je to funkce)
- nelze vícenásobná registrace (poslední přepíše předchozí)

Registrace - programová (DOM2)

```
1  form.addEventListener('submit',
2     cleanValues, false);
3  form.addEventListener('submit', validate, false);
4
5  form.removeEventListener('submit',
6     cleanValues, false);
```

- vícenásobná registrace
- lze určit fázi (*capture*/`true` vs *bubble*/`false`)
- event handler dostane událost jako argument - lze zastavit propagaciu

Kdy registrovat?

`document::DOMContentLoaded`

- HTML dokument byl rozparsován
- `defered` skripty a moduly byly načteny a spuštěny
- nečeká na `async` skripty, obrázky, videa, ...
- vhodné pro registrací událostí

`window::load`

- veškerý obsah (vč. obrázků, videí, ...) byl načten
- vhodné pro spuštění dynamického obsahu závislého na všech zdrojích

Vývoj webu

- základní technologie se nemění
- mění se vývoj a použití
- postupná evoluce

Web 1.0

- podle návrhu Tima Berners-Lee, standardizace W3C
- původní myšlenka *read/write web*
- obsah je vytvářen několika málo entitami
- naopak odebírán velkým množstvím uživatelů
- nevhodný jako platforma pro vývoj aplikací
  - příliš jednoduchý
  - vznikají doplňkové technologie (Flash, Java, ActiveX)

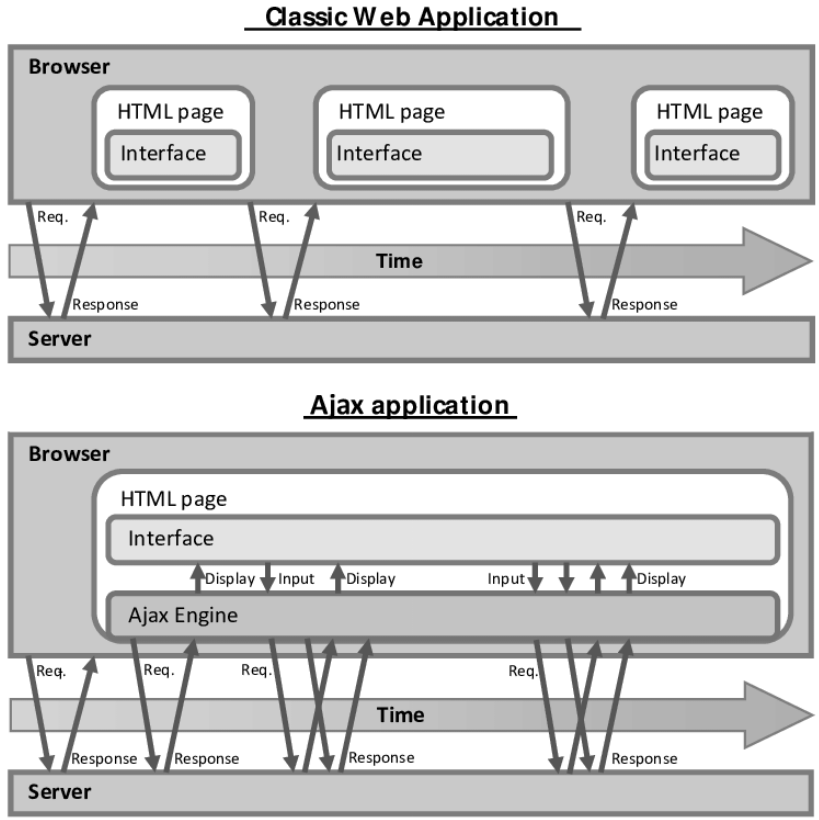
Web 2.0

- důvody pro změnu vývoje a použití webu
  - roste důležitost webu jako média
  - objevují se interaktivní aplikace
- buzzword
  - přesná definice pojmu neexistuje
- platforma
- uživatelé vytváření obsah a mají nad ním kontrolu
- služby, škálovatelnost, spojení datových zdrojů
- software nezávislý na zařízení
- kolektivní inteligence
- nové formáty (RSS, Atom, ...)

AJAX

- Asynchronous JavaScript and XML
- není technologie, ale množina technologií
  - prezentace HTML a CSS
  - Document Object Model
  - výměna dat ve st. formátu - XML, XSLT, JSON
  - JavaScript
    - `XMLHttpRequest` dnes `FetchAPI`
    - spojení částí dohromady

AJAX - ilustrace



Obrázek: Ukázka fungování AJAXu[1]

[1] Zdroj: Kluge, Jonas & Kargl, Frank & Weber, Michael. (2007). The effects of the ajax technology on web application usability.

AJAX - výhody

- zkrácení odezvy aplikace, uživatel méně čeká
  - načítá se pouze změněná část
- menší množství přenášených dat v jednom požadavku
- úspora zdrojů na serveru
  - přepočítává se pouze změna

AJAX - nevýhody

- komplikovanější práce s historií
  - nutno použít fragment url, history push api
- problematičtější bookmarky
- crawleři nevykonávají JavaScript
- uživatel nemusí vykonávat JavaScript
- vyšší počet požadavků
- komplexní kód

AJAX - komunikace

- short polling
- long polling
- streaming
- chunked response

Single Page Application

- větší část logiky se vykonává na straně klienta
- klient obdrží téměř celou aplikaci
- přístup k datům pomocí API

Frameworky:

- AngularJS
- ReactJS
- EmberJS
- BackBoneJS
- VueJS
- ...

SPA - výhody

- jen jedno prvotní načtení statických dat
- nižší přenosy dat
- rychlejší responzitivta
  - přijdou jen potřebná data
  - není potřeba překreslovat celou stránku
  - stránku lze standardně používat
- přívětivější pro uživatele

SPA - nevýhody

- většinou JS-only
  - některé frameworky nabízí "server-side rendering"
  - emulace PhantomJS (discontinued)
- při špatné implementaci pomalé
- nevhodné pro komplikované systémy
- nemožnost aplikaci upravovat za běhu
  - v případě tradičního přístupu, se uživateli při dalším dotazu pošle úplně jiný požadavek
  - v SPA je nutné provést přenačtení aplikace, které se u SPA běžně nevolá
- typické problémy JS/AJAX
  - většinou řeší frameworky
  - browser history (pushState a replaceState)
  - security scanning

Frameworky

- routování
- tracking historie
- persistenci dat na straně klienta
- komponenty
- PWA (později)

Ukázka

- JSX (HTML in JS)

```
() => <Header>
    <Logo icon="app_logo" />
    <Title text="The Best App" />
    <Action label="Subscribe" onClick={ () => subscribe(
</Header>
```

- `Header`, `Logo`, `Title`, `Action` definovány jako znovupoužitelné komponenty
- velké množství existujících komponent

Rapid Application Development

- pracovat s minimálním úsilím
- vývoj frontend a backend $\rightarrow$ 2x tolik práce
- věnovat čas logice, ne kódu

Příklady:

- Class-less CSS framework
- JS supports
- Hotwire

Class-less CSS framework

- focus HTML-first
- CSS se vytvoří později
- může být "demotivující", že nevidíme skutečný výsledek

Další druhy CSS frameworků:

- obecné (Bootstrap, Bulma, Foundation)
- utility (Tailwind, Materialize)
- lightweight (Pure, Miligram, Skeleton)
- specializované

JS support

- promítají JS do HTML
- usnadňují typické problémy v HTML
- př. *Alpine.js*

```
<div x-data={ open: false }>
    <button @click="open = true">Expand</button>

    <span x-show="open">
        ... obsah ...
    </span>
</div>
```

Hotwire

- prakticky JS support
- Hotwire

Turbo	Drive, Frames, Streams, Native
Stimulus	definice komponent
Strada	abstrakce mezi native a web přístupem

Hotwire Turbo

Drive	nahrazuje HTTP requesty za AJAX volání a následně prohodí obsah stránky
Frames	umožňuje přenačítání menších bloků
Streams	WebSockets, SSE, ... real-time zpracování
Native	podpora pro nativní aplikace

Hotwire Turbo: Frames

```
<body>
  <h1>Article #1</h1>
  <!-- ... -->

  <turbo-frame id="comments">
    <div id="comment-1">...</div>
    <div id="comment-2">...</div>

    <form action="/comments" method="POST">...</form>
  </turbo-frame>

  <!-- ... -->
  <turbo-frame id="comments"
                src="/comments" loading="lazy">
    <img src="spinner.gig">
  </turbo-frame>
</body>
```

Hotwire Stimulus I

```
<div data-controller="collapsible">
  <button data-action="click->hello#open">Expand</button

  <span data-collapsible-target="content">
    ... obsah ...
  </span>
</div>
```

Hotwire Stimulus II

```
// src/controllers/collapsible_controller.js
import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
    static targets = [ "content" ]

    initialize() {
        this.opened = false
    }

    open() {
        this.opened = ! this.opened;
        this.updateContentView()
    }

    updateContentView() {
        this.contentTargets.forEach(element) => {
            element.hidden = this.opened
        }
    }
}
```

Moderní přístupy

- **PWA**
- WebGL, WebAudio
- WebSocket
- WASM

Progresivní webové aplikace

- aplikace dodávaná přes web
- multi-platformní
- tvářící se jako **nativní** aplikace
  - možnost nainstalovat
  - není nutné používat browser
  - browser je schovaný na pozadí
- kombinace technologií
- problém s podporou (ale rychle se vyvíjí)
  - Chromium - vše až na iOS
  - Firefox - téměř vůbec
  - Safari - trochu

PWA - Manifest

Definuje:

- název aplikace
- link na ikonku aplikace
- preferovaná URL pro start aplikace
- konfigurační data
- defaultní orientace aplikace
- display mode (př. full-screen)

Service Workers

- proxy mezi aplikací a sítí
- nemají přístup k DOMu
- cachování `fetch`
- notifikace
- výpočty

Data Storage

- WebStorage
  - perzistentní
  - klíč-hodnota
  - cookies (max. 4kB) vs WebStorage (max. 5-10 MB)
    - local storage (stejná appka, sdílí mezi okny)
    - session storage (stejná appka, více oken, omezený lifetime)

```
localStorage.setItem('key', 'value');
localStorage.getItem('key');

sessionStorage.setItem('key', 'value');
sessionStorage.getItem('key');
```

Data Storage

- WebSQL
  - deprecated
- Indexed Database API
  - perzistentní
  - NoSQL databáze
  - ukládání JSON objektů
  - téměř neomezená velikost
  - cachování, off-line perzistence
  - komplikovanější práce

Další API

- Push Notifications
  - Push API
  - Notification API
- on-line/off-line detection
- Geolocation API
- Barcode Detection API
- Bluetooth API
- Battery API
- herní
  - Gamepad API
  - Web Audio API
  - WebGL, Web GPU, WebVR, WebXR
- `...https://developer.mozilla.org/en-US/docs/Web/API`

WebSockets

- velký boom web her
- HTTP-first $\rightarrow$ WS protocol upgrade
  - `101 Switching Protocols`
- hlavičky se posílají jen jednou
- trvalé obousměrné spojení (statefull)
- nutno implementovat vlastní "aplikační protokol"
- vyžaduje speciální server, pro zpracování

Využití

real-time aplikace

- hry
- chaty
- burzovní aplikace
- monitoring
- streaming

Události

open	spojení navázáno
message	přijata zpráva
close	ukončené spojení
error	nastala chyba, spojení bylo ukončeno

Ukázka - server

```
server = new WSServer("0.0.0.0", 9001)

server.on("open", (conn) => {
    SendToAll(UserConnectedMessage( conn.user ))
})

server.on("message", (frame) => {
    SendToAll(UserSendMessage( frame.user, frame.data ))
})

server.on(["close", "error"], (conn) => {
    SendToAll(UserDisconnectedMessage( conn.user ))
})

server.run()
```

Ukázka - klient

```
messages = ...

chat = new WebSocket('ws://example.local:9001');
chat.onmessage = (event) => {
    messages.append(new DOMMessage(
        event.data.author,
        event.data.message
    ))
}

const messageInput = document.getElementById("message")
document.getElementById("send").addEventListener(
    "click", function (event) {
        const text = messageInput.value
        chat.send(new ChatMessage(text))
    }
)
```