



Agenda

- **Frontend frameworks**

Topics: traditional vs. modern websites, Why to use a framework and the differences between them, building desktop / mobile apps

- **Basics of real-time network programming**

Topics: Websockets, webRTC, custom encoded payloads

by William Sjökvist
williamsjokvist.se



Why use a frontend framework over vanilla JS?

- Abstracted DOM manipulation
- Unified API - thank jQuery
- Tooling which add support for bundling, minifying, linting, hot-reload, TypeScript



**Awesome
Developer
Experience
(DX)**



</> A Distinction: traditional vs modern web app

- Traditional website

Primarily to serve content, usually delivered through a *Content Management System (CMS)* backend, not interactable-first design, good *Search Engine Optimization (SEO)*.

Examples: blogs, portfolios, documentation sites, product pages, wikis, forums



A Distinction: traditional vs modern web app

- Modern web app

Designed to be interactable, strictly single-page, usually more complex than only serving content

Examples: telecommunication software, software tools, streaming services, eCommerce platforms, social media

Figma, Google Apps, Instagram, Slack, VSCode, Netflix, Amazon

<https://www.geeksforgeeks.org/difference-between-web-application-and-website/>



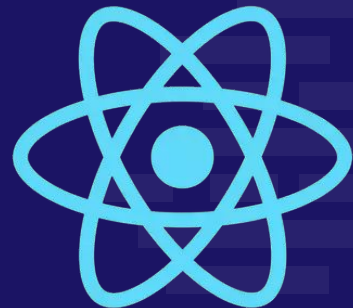
Choosing a Frontend Framework - what type of site/app are you making?

- Research the modern frameworks and try multiple
- Pick an appropriate framework for the project
- **React-like** - React, Preact, Solid
- **Not React-like** - Svelte, Angular, Vue, Astro
- **I'm not a script kid** - Blazor (C#), yew (Rust), Hugo (Go) etc.



Some key differences: React

- More a library than a fully-featured framework
- Widely used, easiest framework to get a job for
- Big community support
- Not the easiest to learn due to confusing API (e.g. useEffect)
- Uses a “virtual DOM” for partial re-rendering
- Poor documentation (the new beta docs are good)





Some key differences: Angular

- Not to be confused with AngularJS
- Enforces an MVC architecture and built-in dependency injection
- TypeScript only
- Good for enterprise solutions
- Steep learning curve
- Good documentation
- Relatively poor community support
- Easy to migrate to new versions, never breaking changes





Some key differences: Vue

- Made in 2014 by Evan You, with the goal of combining the best of both worlds in React and Angular
- Not much widespread enterprise support, although it is financially supported by Alibaba
- Easy to learn
- Clean components





Some frontend terminology

- **SPA** - Only one page, client-side rendering, partial re-rendering
- **MPA** - Rerenders the whole document and all its assets from the server when clicking a link
- **SSR** - The server renders the HTML document
- **SSG** - Generate static HTML documents to serve
- **PWA** - Add native capabilities to your web app, like push notifications, offline support



“Meta” JS frameworks - cuz they're the meta

- Capability to use and combine SSR, SSG, MPA, SPA
- Simplifies development process
- Ability to make backend API endpoints - good for small projects which are not part of a broader system




React => Next, Remix, Gatsby
Vue => Nuxt

Angular => Analog
Svelte => SvelteKit



Live example

- API endpoints
 - Scaffolding Angular app (my first time)
 - Performance difference Astro vs. Next
- 



Using web languages to build desktop apps for Mac, Windows and Linux

Electron - Serve an app by bundling Chromium and Node into the executable. Compatible with any backend and frontend frameworks. JS frontend and Node backend.

Tauri

JS frontend, Rust backend - uses system WebView

Wails

JS frontend, Go backend - uses system Chromium

Chromium Embedded Framework

Supports C++, .NET, Python, Go, Java, Delphi



Apps built with Electron

VSCode, Slack, Discord, Figma, GitHub Desktop, Skype, Signal, Trello, WhatsApp, Twitch, Notion, Microsoft Teams, Mullvad, 1Password

Spotify and Steam are built with Chromium Embedded Framework



https://en.wikipedia.org/wiki/List_of_software_using_Electron

<https://www.electronjs.org/apps/>



Native alternatives, build for Mobile too

- **.NET MAUI** - Build multi-platform native apps with Blazor (C#)
- **Flutter** - Build multi-platform native apps with Dart :(
- **React Native** - Build multi-platform native apps with React



WebAssembly

- A low-level language that can be executed in the browser
- Is a compilation target for C# (Blazor Wasm, Unity), Rust (yew), C++ (Emscripten), Java etc.
- Designed to run alongside JavaScript, does not have direct access to the DOM
- Example: Unity Game Engine in the browser



WA



Topic switch!

Real-time communication



WebSocket protocol



- Utilize the TCP protocol in the browser for real-time duplex communication in between client and server
- Ensures guaranteed delivery (no packet loss)
- Blocking, will wait for late packets to ensure in-order delivery
- Abstracts away low level networking
- Transmit payloads in e.g. JSON, XML or Binary
- Established by sending a GET request with an *Upgrade* header, called a handshake



WebRTC protocol



- Utilize the UDP protocol in the browser
- Not guaranteed delivery
- Does not follow order
- Peer-to-peer

Useful for fast non-critical data

<https://webrtc.dom.se>

```
fps: 118 var: 1.1 ms ping: 0 ms  
loss: 0% choke: 0%  
tick: 64.0 sv: 6.8 +- 1.2 ms var: 1.135 ms
```



Custom Encodings

- When JSON payloads are getting too large, and will use too much bandwidth.
- Example of a payload using byte array vs. JSON

```
Payload UTF-8 string: "{  
  "Player": 1,  
  "Move": {  
    "X": 5,  
    "Y": 12  
  }  
}"
```

42 bytes

```
UInt8Array: [ 1, 5, 12 ]
```

3 bytes



Pseudo code raw TCP socket

```
func sendPlayerMovement() {
    // Player, Move X, Move Y
    var playerMovement = []byte{1, 5, 12}
    var bytesSent, err = TCPConnection.Write(playerMovement)
}

func sendMessage() {
    // length uint16, Player ID
    var payload = []byte{0, 0, 1}
    var text = "Hello from player one"

    binary.BigEndian.PutUint16(payload[0:2], Uint16Array(len(text))
    payload= append(payload, []byte(msg))
    bytesSent, err = TCPConnection.Write(payload)
}
```



Pseudo code raw TCP socket

```
func readPlayerMovement() {  
    data := make([]byte, 3)  
    bytesRead, err := TCPConnection.Read(data)  
  
    playerId := data[0]  
    moveX := data[1]  
    moveY := data[2]  
  
    log.Println(playerId, moveX, moveY)  
}
```



Pseudo code raw TCP socket

```
func readMessage() {  
    // First read the prefix (if there is one, this case 3 bytes), uint16 length and player id  
    prefix := make([]byte, 3)  
    bytesRead, err := TCPConnection.Read(prefix)  
  
    // Create a buffer specifically for this message  
    length := binary.BigEndian.Uint16(prefix[0:2])  
    msg := make([]byte, length)  
  
    // Read the message  
    bytesRead, err := TCPConnection.Read(msg)  
  
    // Log Player ID and text message  
    log.Println(prefix[2], string(msg))  
}
```



Custom Encodings

- JSON for human-readable REST APIs when the client should not have to make their own parser
- Custom encodings when performance is critical, e.g. intra-service communication
- **Protobuf by Google is a great library for serializing data into an efficient binary format** - gRPC uses this for sending messages between services



Live game example





Video recommendations



I built 10 web apps... with 10 different languages

- Fireship



WebSockets in 100 Seconds & Beyond with Socket.io

- Fireship

Is JSON blazingly fast..?

- ThePrimeagen



What is gRPC?

- IBM Technology

