**Technology Comparison Sheet.**

|  |  |  |
| --- | --- | --- |
| What type of technology are you comparing? | | |
| Server Backends | | |
| Product / Technology | Pros | Cons |
| ASP.NET | Reasonably scalable | Made by Microsoft |
| Type safe | C#, which is made by Microsoft |
| Easy-to-use routers | Doesn’t have support for the more modern JS frameworks |
|  |  |
| Flask (/ other python servers) | Easy routers | Its python (ew)  i.e. not very fast or scalable |
| Fast to develop | Again, very little support for the modern js frameworks (react) |
|  | Very janky SSR/templating |
|  | Code duplication between client/server because |
| Next.js | Suitable integration for modern web frameworks e.g. react | Js, so may be slower |
| Easy routers, with support for passing information to react |  |
| Supported by cloudflare for free hosting |  |
| ServerSide Rendering, beneficial for client performance  ---------------------------------------------------- Type safe using typescript  Using same language on client/server allows for some code deduplication in shared code |  |
| NodeJS http.server | Low-level | Slow development |
| Powerful for any needed low-level packet management | Manual routing and handling |
|  | Code duplication |
|  | Hard to scale |

|  |
| --- |
| Which technology will you be selecting? |
| Next.js |
| What are the reasons for your choice? |
| Supported by cloudflare, allowing for free hosting during development, including DDOS protection and protection from other common vulnerabilities, for free. NextJS and Cloudflare are both incredibly scalable for any future expansion or large web traffic flow. |

**Technology Comparison Sheet.**

|  |  |  |
| --- | --- | --- |
| What type of technology are you comparing? | | |
| Frontend frameworks | | |
| Product / Technology | Pros | Cons |
| Svelte | Componentwise for code deduplication | Uses non-vanilla-js-style, unfamiliar, syntax, slowing development |
| Simplicity – simpler than react | Lack of support by community and IDEs slowing development |
| Simple abstractions | Less scalable compared to react |
|  |  |
| React | Componentwise for code deduplication | Less extensible to web/native |
| Easily abstractable |  |
| Can use Server Components to take advantage of SSR for faster client render |  |
|  |  |
| React Native Web | Component-based rendering allowing for code deduplication |  |
| Can compile to native for mobile support using common code |  |
| Reduced code deduplication allowing for faster development and more focus on functionality |  |
| Easy abstractions for code deduplication and extensibility for scalability and faster development |  |
| Vanilla (HTML, CSS, JS) | Small code size for faster load times | Incredibly unscalable when it comes to large projects |
| Static serving – can host on simple hosts like github/cloudflare pages | Much code duplication – no templating |
|  | Browser support varies |
|  | Little/no support for libraries outside of fully switching to nodejs (at which point you might as well use a framework) |

|  |
| --- |
| Which technology will you be selecting? |
| React Native Web |
| What are the reasons for your choice? |
| Gives an incredibly clean and flexible development process, while upholding a highly scalable and maintainable product with support for future expansion (e.g. to PWAs or Mobile apps (using Expo)). Build process can fully integrate with nextjs and expo to produce a quickly deployable build result. |

**Technology Comparison Sheet.**

|  |  |  |
| --- | --- | --- |
| What type of technology are you comparing? | | |
| UI Frameworks | | |
| Product / Technology | Pros | Cons |
| None | Less dependencies, decreasing project bundle size (which could affect hosting prices) | Requires custom-built UI elements, increasing development time |
|  | More likely to contain bugs, as the code will not have been as extensively tested |
|  | Code may need to be written for each platform independently |
|  |  |
| Tamagui | Fully type safe with typescript support | Doesn’t fully support all events cross-platform (although writing platform-specific overrides is easy) |
| Naturally targets both mobile and web | Lacking some documentation, slightly increasing development time |
| Has many prebuilt GUI components such as dialogs, buttons and menus, SVG icons, as well as theming support |  |
| Comes with a choice of three animation drivers to support all platforms  Compiles to CSS for performant web support  Very frequent updates and bug-fixes |  |
| Tailwind css | Extensive support for CSS styles | Doesn’t naturally support native/mobile (without external libs) |
| IDE support for type-hinting | Hard-to-use, everything is acronyms that developers must simply remember |
| Well documented, large community |  |
|  |  |

|  |
| --- |
| Which technology will you be selecting? |
| Tamagui |
| What are the reasons for your choice? |
| It integrates nicely with react native web, allowing for future extension into native (with tauri) or mobile (with react-native). Highly extensible and abstractable using styled(). Covers most UI use-cases with cross-platform code, but has extensive support for platform-dependance as well. Compiles to CSS on web, making very performant for web users. Also compiles to native android/ios UI elements when available, increasing interopability with their system’s native UI |