Inclusive Architecture Technologies

Introduction

The key to developing an inclusive architecture is to adopt technologies that:

- Separate concerns (such as interface design, business logic, and API integration)
- Hide complexity behind more accessible abstractions (correct-by-default)
- Resist bespoke solutions in favor of standard community-generated solutions
- Connect the team to a wider network of documentation and training content
- Enable team members of all experience levels to contribute meaningfully based on their individual talents without needing to be micromanaged

In this document, we'll look at some of the classes of technologies that are used to build robust front-end applications. In each class of technologies, we'll look at some of the libraries and software that have impressed us and embody the principles of the PAM stack towards creating a more inclusive architecture that enables developers of all experience levels to be fully successful and makes whole teams more efficient.

Frameworks

We wholeheartedly promote and endorse the use of frontend frameworks such as React, Angular, Vue, Svelte, AMP, etc. in the creation of front-end applications. Frameworks do an excellent job of hiding the complexity of modern web development and allow your team to focus specifically on building the features that best serve your customers. As your team grows through experience, personal learning, and mentorship, they can and should discover how the frameworks are developed and how they solve problems related to performance, security, accessibility, responsive design, and more!

It cannot be overstated how much frameworks help make it easy to make the right decisions. Frameworks help junior developers (and senior developers as well!) deliver robust web experiences to clients even while they continue to learn about vanilla JavaScript and web platform approaches to performance, accessibility, etc.

There are many successful and popular frameworks. You cannot go wrong with any of the major frameworks. We have known developers and inclusive teams have success with any of the following frameworks:

- Angular
- React

- Vue
- Preact
- Svelte
- Ember
- AMP
- LitElement

However, another objective of making more inclusive architectural choices is considering how much support, documentation, and training materials exist. For these reasons, we strongly recommend either Angular, React, or Vue due to the top-notch documentation, large communities of users, and great third-party libraries and training resources.

Command Line Interface (CLI) Tools

Another source of complexity on projects are in the bespoke project scaffolding and build systems that exist. Often, even within a single company, different teams might use wildly different approaches to project construction, building, and deployment.

To make teams more successful, we want to standardize the ways that we start and build projects. This helps developers transition to new teams and spin-up faster. Even better, we want to standardize how we initialize projects with the rest of the community, so we have access to support, documentation, and training.

This is where command line interface (CLI) tools come in. They help us start up projects with good, performant defaults. Furthermore, we can use the CLI tools to automate and assist with repetitive tasks that might themselves be error prone.

There is very little risk associated with using CLI tools as well, because most CLI tools allow users to eject their project if they ever outgrow the choices made for them by the CLI. This leaves those teams with a project that looks exactly as they would if they had built the structure themselves from scratch.

We love the following CLI tools:

- Angular CLI (Angular)
- Nrwl Nx (Angular)
- Create React App (React)
- Gatsby CLI (React)
- Next (React)
- Vue CLI (Vue)
- Nuxt (Vue)
- Preact CLI (Preact)
- Ember CLI (Ember)

Graphical CLI Utilities

The fact that most web development tools rely on the terminal (or Linux/Mac machines) is a major inclusivity challenge to bring in the next set of successful developers. We advocate that terminal mastery should not be a basic requirement to be a successful web developer. There exist graphical user interfaces (GUI) for most terminal tools such as Git, which help new developers deal with the intimidating set of Git commands.

Luckily, GUI tools are increasingly being developed to help with framework and front-end development as well. These tools help users start new projects, analyze bundle sizes, install new libraries, run tests, serve the applications, and even build assets for production.

Wherever possible, we recommend that teams use technologies that are compatible with these newly developed GUI CLI tools. Here are some GUI tools for frameworks that we know and love:

- Vue CLI GUI
- Angular Console
- GuiJS

Deployment and DevOps

The complexities of web development don't stop after you run your production build. It can be complicated to integrate a front-end with APIs, to set up continuous integration and continuous deployment pipelines, to run back-end processing, perform authentication, and manage data.

For all the reasons we love any technology, we're a fan of service providers that help teams easily deploy and manage their applications. There are tons of amazing products out there, and more are being developed every day. That said, here are some of the tools that we recommend to teams and have used alongside successful teams in the past:

- Netlify
- Firebase
- Zeit
- Gatsby
- Amplify

State Management

We are big fans of using state management libraries such as NgRx (Angular), Vuex (Vue), and Redux (React). When used correctly, these libraries greatly simplify components and isolate business logic complexity into the state management layer. This creates ways for senior developers and junior developers to split up work on new features and reduce regressions due to what should otherwise be visual changes.

If a team has no developer with experience setting up and using a state management solution, then we would not recommend using one of these libraries. The costs of using one of these libraries incorrectly can often outweigh the benefits they provide. However, when used on a team with a senior developer who can properly manage the state management architecture, these tools can greatly speed up new development.

As mentioned before, we’re big fans of the following state management libraries:

- NgRx (Angular)
- Vuex (Vue)
- Redux (React)

State Machines

Another great tool for reducing complexity of front-end applications are state machines. Not only do state machines help to control and manage the various states of your application and the transitions between them, but they also function as documentation for current and future team members about how the application is meant to function independent of the technology you implement it in.

Furthermore, state machines are proving to be an excellent adapter for the automatic generation of accompanying both unit tests and end-to-end tests. This makes them an even more powerful tool for improving quality.

In 2019 and 2020, state machines took off in popularity in front-end applications, thanks in large part to the success of the XState library. In a year or two, they will be used as ubiquitously on projects of all sizes as state management libraries like Redux are now.

Due to its rapid adoption rate and ever-growing amount of training resources, we fully recommend the use of the XState library for implementing state machines in any framework.

Component Libraries and Design Systems

Another issue that large teams face is how to promote consistency and reuse of markdown and styling across their applications. Large applications tend to have tons of repeated CSS and reimplemented component patterns that lead to inconsistent experiences, slow maintenance, and frequent visual regressions.

The goal is to create a shared visual language across all parts of your team: design, development, test, and management. This shared language enables everyone on the team to be responsible for the quality of the visual quality of the application, because everyone is aware of what is expected.

The two tools we advocate for to address these concerns are component libraries and design systems.

Design systems help you define the tone, content, approach, and structure of your website and your copy. They help your designers and developers create with the same vision. They help your team clearly define and understand what visual components are desirable and which are not. Design systems can often include design files for your designers, component libraries for your developers, and guidelines for your copywriters and testers. Often it can be best for each team to develop their own design system, but there are several off-the-shelf design systems available for reuse:

- Shopify's Polaris
- Google's Material Design
- IBM Carbon
- Stack Overflow Stacks
- Microsoft's Fluent
- Atlassian's AtlasKit

Component libraries are collections of basic components for oft-required visual elements like inputs, buttons, autocomplete boxes, modals, and more. These libraries are often implementations of a design system in a particular framework. Beyond the benefits created from standardization, they help your team build new features faster by focusing on the new, complex components instead of spending all their time reskinning basic HTML elements. Furthermore, the component libraries are great at hiding complexity by managing things such as performance, accessibility, and responsive design by default.

Examples of great component libraries include things like:

- Shopify Polaris
- Google's Material Design (Open-source implementations available for all frameworks)
- Reach UI (React)
- AWS Amplify

Conclusion

The above list is not inclusive of all the great software and libraries that exist out there. It's not even inclusive of all the various classes of tools that teams will need to build robust, inclusive front-end applications. Instead, apply the reasoning for using the above tools when making changes to your technology stack. All other things being equal, prefer using technologies that make your teams more inclusive to developers of all experience levels and make your whole team more productive and efficient.