Team “Hello World” 04/02/2021
Team Lead: Matthew Caponi
Matthew Caponi - 011161928
Shaan Shariff - 015741750
Ian Macabulos - 016401240
Jakub Koziol - 016523856

JS Framework Approval Request

Introduction

Modern JavaScript frameworks range from a collection of libraries to assist with frontend development, to complete solutions and development environments. In particular, they aim to solve a common issue that makes writing dynamic, client-side code, particularly cumbersome; how to handle state changes.

In Vanilla JavaScript or jQuery code, the developer has to follow an imperative approach to handle dynamic changes in the state of their application. That means that they have to write code that is more concerned with how the UI should change with changes in the state of the application. Modern JavaScript frameworks, on the other hand, abstract away much of what would normally have to be done with vanilla JavaScript, so that the developer can follow a declarative approach; that is, they can declare what they want the UI to render under which conditions, and let the framework figure out how to do it.

While this is one of the core benefits of using JavaScript frameworks, they also offer other benefits, such as helping with the development of single-page or progressive web applications. Many also follow a component-based system that allows for the creation of web pages from a tree of smaller, reusable components.

Business Need

The web application we are building will be moderate in size and complexity, and have many features that require dynamic capabilities. Using a modern JavaScript framework will help us to build our web application at a high enough level, whereby we can focus more on fulfilling the frontend-related business needs, rather than reinventing the wheel.

By using a framework that follows a declarative approach and abstracts away much of the implementation details of changing state, we can add much more dynamic content to our web application without it getting overly complex.

In addition, using a framework that is component-based will support reusability and make the application far more maintainable.

Finally, the solutions provided to enable single-page web applications will help us to meet the requirements of our high-level design, while also offering a more dynamic and seamless user experience.

The Options

Angular[1]

Angular is a declarative, component-based JavaScript framework that follows an architecture whereby an application is split into HTML templates, TypeScript classes that provide data for the templates, and services, which are injected into the Typescript class to offer functionality.

Angular offers a complete solution, with many built in libraries, that cover almost every feature that is needed in developing modern web applications.

React[2]

React is a declarative, component-based JavaScript library that offers functionality that is mainly focused on handling state in a reactive way, for a more efficient UI building experience.

React uses JSX code to give the user the ability to write HTML-like code mixed in with JavaScript code, for a more intuitive and flexible development experience. React maintains a virtual DOM that keeps track of state changes and updates the real DOM in an efficient manner, only changing that which has been changed in the virtual DOM.

Vue[3]

Vue is a declarative JavaScript framework that focuses mainly on the view, connecting data models to html templates. Vue can be used in just specific parts of an application,

[1] https://angular.io/guide/what-is-angular
[2] https://reactjs.org/docs/introducing-jsx.html
[3] https://vuejs.org/v2/guide/

or to handle the rendering of an entire application using components. Vue also uses a virtual DOM to render changes.

License

Angular[4]: Angular uses an MIT license.

React[5]: React uses an MIT license.

Vue[6]: Vue uses an MIT license.

Comparisons

Comparison Criteria

A selection of criteria deemed important for evaluating the differences between the three JavaScript frameworks have been developed. For each criterion, how each framework handles it will be discussed, and then a favorite will be chosen.

The criteria we will evaluate are as follows:

Design Architecture

- What is the design architecture typically used with said framework, and how strictly is it enforced?

Databinding

- How to share data between different parts of a component?

Intra-Component State

- How to respond to changes in the state of the application?
- How to respond to changes in the state of the UI?

Component Communication

- How do components communicate state data to other components?

[4] https://github.com/angular/angular.js/blob/master/LICENSE
[5] https://github.com/facebook/react/blob/master/LICENSE
[6] https://github.com/vuejs/vue/blob/dev/LICENSE

- How do components communicate code to other components?

Component Lifecycle

- What lifecycle methods exist?
- What is the render order of these lifecycle methods?
- How can a developer hook into them?

Ecosystem

- What libraries, packages, and tooling is integrated with the framework?

Design Architecture

Angular[7]

Angular is component-based but has a more complex design architecture than the other two options. An angular component typically consists of the following:

- HTML template
- Component class
- Services

HTML Template

The template in angular uses regular HTML code with extended functionality to allow for dynamic content to be rendered. This makes the syntax feel much closer to regular html syntax, but it results in more restrictive uses of the dynamic JavaScript code.

To use this dynamic code, Angular offers features such as:

- event bindings
- property/attribute bindings
- references
- directives.

Component Class

The component class is a TypeScript class that acts as a code-behind to the HTML template. The purpose of the component class is to store and configure data, and to enable communication between the template and the logic.

[7] **https://angular.io/guide/component-overview**

Services[8]

Services are TypeScript classes that are dependencies of component classes. These services often contain functionality that the component classes need, but that are not directly related to rendering the template view.

Angular has a dependency injection system that allows these dependencies to be abstracted out of the component class and injected by Angular into the classes. This helps to manage the dependencies and also assists in cross-component communication.

Directives[9]

Directives are essentially classes that help the developer to add behavior to their application. They are separated into three types.

- Components
    - Components consist of the user-defined components that can be rendered within a template.
- Attribute Directives
    - Attribute directives helps with changing the behavior or style of other elements. They can be used for
        - Changing styles
        - Changing classes
        - Using two-way binding.
- Structural Directives
    - Structural directives assist in changing the HTML layout, by manipulating elements in a way that specifies how the layout should react to changes in state.
        - Some of the most common structural directives do the following:
            - Add conditional functionality to an element
            - Add loop functionality to an element

Modules[10]

Modules are classes that are meant to be available to either the entire application, or large chunks of the application. Some of the areas they assist in are:

- Registering components.
- Registering directives
- Configuring service providers.

---

[8] https://angular.io/guide/dependency-injection
[9] https://angular.io/guide/built-in-directives
[10] https://angular.io/guide/architecture-modules

- Global Styles

React

React has a much simpler design architecture than Angular. React has a component based architecture that mainly just consists of JavaScript (ES6) files (and optional css files). Typically, each file represents a component, and then these components can be used as children of other components higher up in the component tree.

There are two types of components in React:

- Class-based Components
- Functional Components

Class-based Components[11]

Class-based components use ES6 classes to render data to the virtual DOM. Before recent updates, these were the only types of components that could cause dynamic changes to the state of the data of the application, referred to as stateful components.

Functional Components[12]

Functional components are simply functions that return some data to be rendered by a parent. Prior to recent updates, functional components could only be used as presentational components, that is, they could only be used to display information, not handle state.

With the introduction of React Hooks, functional components can now be used to change state and is actually recommended by the React dev team and the React community as the main way of using handling state, for future projects.

Virtual DOM[13]

The main way in which React renders data is through the use of a virtual DOM. Specifically, it uses two virtual DOMs, one for the current state of the real DOM (the previous virtual DOM) and one for the current state of the virtual DOM, then compares what has changed between the two virtual DOMs, and only applies those specific changes to the real DOM. This allows for much faster and more efficient rendering, as the real DOM is very slow in comparison.

[11] https://reactjs.org/docs/components-and-props.html
[12] https://reactjs.org/docs/hooks-intro.html
[13] https://reactjs.org/docs/introducing-jsx.html

JSX

JSX is a type of syntactic sugar over JavaScript. JSX very closely resembles actual HTML code, thus allowing the developer to write HTML code in a near seamless combination with JavaScript code. As a result, the developer can tie the UI logic with the rendering logic, something that helps to interact with the HTML in a more intuitive and inventive way, enabling more efficient development of dynamic content.

Vue[14]

Vue has a component-based architecture, but unlike the other two frameworks, does not have to be used as a complete application framework. Instead, Vue can just be used to control specific parts of the application that would like to take advantage of its functionality. That being said, Vue can also be used for the entire application, if that is desired.

Vue consists of a single Vue instance that is separated into an HTML-template and an options object.

HTML-template[15]

The HTML template is very similar to the template used in Angular. It is used to declare HTML elements that will be rendered to a virtual DOM, which is then used to manipulate the real DOM.

Options object

The options object is something that is passed to a Vue instance that can contain information that assists in interaction with and manipulation of the HTML-template. The options object consists of the following options, among others:

- Data
    - Properties that can be passed to the HTML-template.
- Methods
    - JavaScript methods that can be used with the HTML-template to enable functionality, such as event handling.
- Props
    - Data that can be received from the parent component.
- Computed

[14] https://vuejs.org/v2/guide/instance.html
[15] https://vuejs.org/v2/guide/syntax.html

    - Methods that act as reactive properties that account for changes in their own dependencies.
- Watch
    - Methods that check for changes in any specified dependency.

Discussion

The Angular architecture is very rigid and overly complex. While it may be good for very large, enterprise applications, it is overkill for our project and would result in overly complex code that is hard to maintain. Vue's architecture is very simple and easy to understand. React also has a very simple architecture. In addition, React's architecture is much more fluid and allows for more inventive, creative, and intuitive frontend development through the use of JSX. The tying together of the UI logic and render logic is very helpful for developing dynamic content, and the lack of a rigid, biased structure will help us to streamline development.

Verdict

React.

Databinding

Angular

Text Interpolation[16]

Angular allows one to access the properties and methods of the component classes from within an HTML template.

```
<h1>{{ someProperty }}</h1>
<h1>{{ someMethod }}</h1>
```

Property Binding[17]

Angular gives the functionality of property binding, which lets the developer directly modify the value of an underlying HTML element using a component property. Doing this requires the property name to be wrapped in brackets.

```
<button [disabled]="someBoolean">Click Me</button>
```

[16] https://angular.io/guide/interpolation
[17] https://angular.io/guide/property-binding

React

JSX-Embedded Properties

React allows the developer to output ES6 defined properties within JSX code.

```
const someProperty = "Hello";
```

```
<h1>{someProperty}</h1>
```

JSX-Embedded Expressions

JavaScript expressions can also be freely embedded within JSX code.

```
const isMorning = false;
```

```
<h2>{isMorning ? "Good Morning" : "Goodnight"}</h2>
```

JSX-Embedded Attributes

JSX attributes can be specified very simply, almost the same as in html.

```
<input type="password" />
```

In the case of attributes that contain two words, they can be specified using camelCase. Additionally, since “class” is a reserved word in JavaScript, “className” is to be used instead.

```
<button className="btn">Click me!</button>
```

Attributes can also be easily set to JavaScript properties.

```
<input type="text" placeholder={somePlaceholder} />
```

Vue

Text Interpolation

Data from the data option can be outputted within an html element.

```
data() {
    return {
      someProperty: 'Prop',
    };
  },
```

```
<h2>{{ someProperty }}</h2>
```

Attribute Binding

Attributes of the html elements can be bound to data properties indirectly using the “v-bind” syntax.

```
<button v-bind:disabled="buttonDisabled">Click Me!</button>
```

Expression Binding

Javascript expressions can be bound inside data bindings as well.

```
<button v-bind:style="{borderColor: isSelected ? 'red' : 'grey'}">Click Me!</button>
```

Discussion

All three frameworks offer very similar capabilities in terms of databinding. For attribute binding, Vue has the most cumbersome syntax, requiring directives. Angular’s lack of JavaScript expression binding makes it less flexible for databinding. React has the most flexible solution, allowing expression’s to be freely integrated in JSX code and having attribute binding that is almost identical to how it would be in regular HTML.

Verdict

React

Intra-Component State

Angular

Event Binding[18]

Angular allows event binding by using a predefined event keyword wrapped in parentheses to be set equal to a user-defined function.

```
<button (click)="onSubmit()">Submit</button>
```

Custom events can also be defined in the component class by exposing an EventEmitter property and using it to “emit” an event to the template. This allows events to be specific to a certain component.

```
  @Output() someEventTrigger = new EventEmitter();
```

Two-way Binding[19]

Angular has easy two-way binding support, which allows for both the UI and the data properties to respond to changes in the other.

```
<input type="text" class="form-control" [(ngModel)]="someProperty">
```

React

React streamlines state changes within a component, with only one main way to do so; setting the state. However, recent updates to React have changed the way in which state is set, leaving two separate ways to set state; one of class-based components and one for functional components.

Class Component-Based State Updates

In the old way of setting state, the developer would set an initial state using this.state, and then when they would want to change the state, all they would have to do is use the setState method.

This would only allow the developer to change object state, and would automatically merge state updates with the old state. This was only possible in class-based components.

[18] https://angular.io/guide/event-binding
[19] https://angular.io/guide/two-way-binding

```
state = {
    someState = ''
  }
```

```
this.setState( {
      someState= 'updatedState'
    } )
```

Functional Component-Based State Updates[20]

With the introduction of React Hooks, React now allows for state to be set in functional components using the useState hook.

```
const [someCurrentState, setSomeCurrentState] = useState("");
```

```
const [someCurrentState, setSomeCurrentState] = useState("");
```

```
<h1>{someCurrentState}</h1
```

Unlike class-based components, the useState is not merged automatically with previous states, however, it allows for any type to be set as a state, not just objects, as well as any many states to be set as one likes.

This is a highly simplified approach of setting states and is currently the way that the React dev team and the React community recommends setting state for new projects going forward. We will use examples from functional components for the rest of this document when discussing React, except in the lifecycle section where both are compared.

Event Handlers[21]

Event handlers in React can be used to facilitate internal application state changes in response to specific changes in the UI. An event can be listened for on a component, and then trigger a function which will change the internal state upon execution, using whichever state change methodology the developer wishes.

```
const [clickState, setClickState] = useState("Click Me!");
```

[20] https://reactjs.org/docs/hooks-state.html
[21] https://reactjs.org/docs/handling-events.html

```
function clickEventHandler() {
  setClickState("I have been clicked!");
}
return (
  <div className="App">
    <button onClick={clickEventHandler}>{clickState}</button>
  </div>
);
```

The onChange event can be used to listen to changes in input, and set up a two-way binding solution, although it is more of a workaround rather than a feature.

```
const [inputState, setInputState] = useState("");

function handleInputChange(event) {
  setInputState(event.target.value);
}
return (
  <div className="App">
    <input onChange={handleInputChange} />
    <h1>{inputState}</h1>
  </div>
```

An example of two-way binding:

```
const [inputState, setInputState] = useState("Default Input");

function handleInputChange(event) {
  setInputState(event.target.value);
}
return (
  <div className="App">
    <input onChange={handleInputChange} value={inputState} />
```

```
    <h1>{inputState}</h1>
  </div>
```

Vue

Event Handling[22]

Vue has similar support for event handling. Vue can use the v-on: directive to listen for specific, Vue-defined events and provide them with methods defined in the methods options.

```
<button v-on:click="increment(1)">Increment</button>
```

```
methods: {
    increment(num) {
      this.counter = this.counter + num;
    },
```

Computed Properties[23]

For a more reactive solution, Vue offers computed properties, which are aware of all their dependencies and will only re-render when it detects changes in its dependencies.

Computed Properties are actually methods that define logic, but are used like properties.

When any one of the dependencies of the method changes, it will re-run, updating the property that is based on the method.

This is a much more reactive and efficient solution when logic is involved in updating properties.

```
<input type="text" v-model="name">
```

```
computed: {
    fullName() {
```

[22] https://vuejs.org/v2/guide/events.html
[23] https://vuejs.org/v2/guide/computed.html

```
      return this.name + 'SomeLastName';
    },
  },
```

```
<p>Name: {{ fullName }}</p>
```

Watchers

Watchers can watch any sort of property and define logic to react to changes in that data. This is a more generic solution and is typically used for non-data related changes, such as http requests.

```
watch: {
    someProperty(someWatchedProperty) {
      if (someWatchedProperty == true) {
        //do something
      }
    }
  },
```

Two-way Binding[24]

Vue offers an easy two-way binding solution using the v-model directive. This allows for changes to the internal state and the ui state based on changes to either.

```
<input v-model="message" v-bind:placeholder="{message}">
<p>Message is: {{ message }}</p>
```

```
 data() {
    return {
      message: "Current Message"
    };
  },
```

Discussion

Both Angular and Vue have easier 2-way binding support, however, React does not require a complex solution to set up 2-way binding. In addition, only allowing for 1 way binding, which is default in React, allows for more predictable code.

[24] https://vuejs.org/v2/guide/forms.html

Both Angular and Vue have a more complex and verbose state handling system and many more ways to handle state. React just has one main way of handling state, which is simple and flexible, and intuitive.

Verdict

React.

Component Communication

Angular

Sharing State Data

Inputs[25]

Input allows for state data to be passed to child components by having the child component expose some property with the @Input decorator. With the property exposed as such, the parent component can then use property binding to pass data to the child component.

```
@Input() name: string;
```

```
<app-input [name]=”someName”></app-input>
```

@ViewChild[26]

@ViewChild allows the developer to pass a local reference to some content in a child component to a parent component, so that they can access the properties of that component. Specifically, this allows the parent component’s class to have access to the child component, as typically, only their template can have this access.

Sharing Code

Output

Angular allows for child components to define certain events which will be emitted to the parent components under certain conditions. They allow this using the @Output decorator on events that are then emitted using the EventEmitter.

[25] https://angular.io/guide/component-interaction
[26] https://angular.io/api/core/ViewChild

```
  @Output someOutput = new EventEmitter<string>();
```

```
addSomeOutput() {
   this.newOutputEvent.emit(value);
 }
```

```
  <button (click)="addSomeOutput()">Click Me!</button>
```

The properties exposed in the component class of a child can also be accessed in the template of that child through property binding.

Services

A parent and child component can also communicate via services, wherein, any child component can emit an event that is defined in a service to a parent component.

In the Service

```
changeEvent = new EventEmitter<string>();
```

‘

In the child component the event can be emitted

```
this.changeService.changeEvent.emit(changeValue);
```

In the parent component, the event can be listened to

```
constructor(private changeService: ChangeService) {
   this.changeService.changeEvent.subscribe(
     (changeValue: string) => alert('Change Happened: ' + changeValue)
   );
 }
```

React

Sharing State Data

React has a simple way of passing data to child components, using the “props” syntax. Props can be added to a parent as an attribute very simply. In fact, to add a prop, all the developer has to do is define a new attribute and set it equal to a value on the

component they wish to add it to, as if it already exists, and React will create a props object behind the scenes that contains all the properties that were added, and pass it to the child component.

All the child component has to do is take “props” as a parameter and then can reference any property within props by using “props.propertyName”.

```
<UserTable rows={users}/>
```

```
{(props.rows.map(user => (
```

In this example, the UserTable component has a props called “rows” which takes in an array called “users”. Within the actual component, it can be used simply by referring to props.rows. In this specific example, it then maps the rows to a table.

Sharing Code

React also allows the developer to pass function references to child components. A common reason for this is so that the parent can pass an event handler to the child so that the child can react to some event happening in the parent, in essence, passing data back up to the parent.

```
const [someInput, setSomeInput] = useState("No Input Yet");


function inputHandler(event) {
  setSomeInput(event.target.value);
}
return (
  <div className="App">
    <TestComponent change={inputHandler} />
    <h2>Hello {someInput}</h2>
  </div>
```

```
<div className="Test">
    <input type="text" onChange={props.change} />
  </div>
```

In this example, the parent component passes an event handler to the child component, which then uses the event handler to change the name of some value with the input that is being entered.

Vue

Sharing State Data

Props

Vue has a similar props functionality to React. The difference is that in Vue, the props must be specifically defined in the child component that wishes to receive them, and the props must be passed from the parent using property-binding if they are to be dynamic.

```
props: {
      someProp: String,
}
```

```
<child-component v-bind:someProp="somePropValue"></child-component>
```

Sharing Code with Parent and Child Components

Events

Vue allows for event emitting, so that a child component can emit an event that can be listened to in the parent component and handled.

In the child component, the event can be emitted and pass some data.

```
someEvent() {
      this.$emit('some-event', this.someData);
    },
```

In the parent template, the event can be listened to using the v-on directive, as an attribute on an html element.

```
@some-event="someEventMethod"
```

The event can be handled in the methods section of the parent component.

```
methods: {
```

```
    someEventMethod(someData) {
     // do something
    },
  },
```

Discussion

Sharing state data in Angular is a little bit arduous. Having to share state data by specifying the decorators or references adds overhead that is unnecessary to our application. While this follows the design philosophy that Angular takes overall, that definitely has pros in larger applications, it is unnecessarily complex for ours.

When it comes to sharing of code, such as emitting events, Angular has a syntax that is very similar to Vue. Both have lots of overhead that just reduce maintainability.

Vue's props system for sharing data is fairly similar to React's, although it requires extra specification of props in the child component, although this is very minor. The main issue is that it requires a v-bind directive for dynamic prop data, which just makes it less intuitive and more rigid.
React's prop system is ideal because the props on the parent component can be written as if they are already predefined html attributes. This just adds to the style that React seems to follow.

When it comes to sharing code, React has less options than the other two, however, being able to pass a function reference is really all that is needed.

Verdict

React.

Component Lifecycle

All three JavaScript frameworks define a specific component lifecycle, which specifies the order in which components are rendered upon mounting, updating, and unmounting. This is important because it allows the developer to be able to "hook" into specific lifecycle events to have functionality run at different times in the component lifecycle.

Angular[27]

Constructor

- The constructor is called before any lifecycle methods are.

ngOnChanges

- Executed after a component is created or if a bound input property changes.

ngOnInit

- Rendered after the component has been initialized.

ngDoCheck

- Runs anytime Angular runs their change detection.
- Only on changes that Angular can't or won't detect.

ngAfterContentInit

- Called whenever ng-content projection has been initialized.

ngAfterContentChecked

- Called after projected content has been checked.

ngAfterViewInit

- Runs after the views have been rendered

ngAfterViewChecked

- Runs after the views have been checked.

ngOnDestroy

- Called right before a component is going to be destroyed.

React

React has two different lifetimes for class-based components vs functional components.

[27] https://angular.io/guide/lifecycle-hooks

Class-Based[28]

Mounting Lifecycle

Constructor

- Runs when a component is created.

getDerivedStateFromProps

- Runs whenever prop is initially mounted.

Render

- Return JSX code
- Child components

componentDidMount

- Called after a component is mounted.
- Can be used for side effects.

Updating Lifecycle

getDerivedStateFromProps

- This is also called on updates to prop changes.

shouldComponentUpdate

- Run before rendering to and receives the props or other state data that is being passed.
- Is often used for optimization, to prevent re-renders for components that don't need to be frequently re-rendered.

Render

- Return JSX code
- Child components

getSnapshotBeforeUpdate

- Is run right before the end of the update render cycle.
- Takes in props and state that can be passed as a parameter to componentDidUpdate, to save data that might want to be configured and then used after the render is committed.

[28] https://reactjs.org/docs/react-component.html

componentDidUpdate

- Renders rights after update has finished.
- Receives the snapshot parameters that were sent from getSnapshotBeforeUpdate.
- Can be used for side effects.

Unmounting Lifecycle

componentWillUnmount

- Runs right before the component is removed.

Functional[29]

With the use of React hooks, functional components can now tap into the React component render lifecycle in a much simpler way.

useState

- Since there is no constructor, the state can first be initialized in the useState.
- This is the first method to run.

React.memo

- Used for optimization in place of shouldComponentUpdate

Render

- Return JSX code
- Child components

useEffect

- useEffect can handle all cases that previously could be handled by componentDidMount, componentDidUpdate, and componentWillUnmount.
- This is used to handle side effects, such as http requests and other asynchronous operations.

[29] https://reactjs.org/docs/hooks-faq.html#how-do-lifecycle-methods-correspond-to-hooks

Vue[30]

App Created

beforeCreate

- Runs before app has initialized

created

- Runs after app has been initialized.
- Options have finished processing.

Template Compilation

- Dynamic data is removed and replaced with concrete values.

beforeMount

- Right before instance has mounted and data rendered to the screen.

Mounted

- After instance has mounted and data rendered to the screen.

Data Changed

beforeUpdate

- Before the updating has fully happened.
- Whenever data is changed.

Updated

- Update has finished.

Vue Instance Unmounted

beforeUnmounted

- Called right before a Vue instance has been unmounted.

Unmounted

- Call when Vue instance has been unmounted.

[30] https://v3.vuejs.org/api/options-lifecycle-hooks.html#mounted

Discussion

The lifecycle and lifecycle methods are very similar for all three.The only main difference is the life cycle for functional components with Hooks in React. This consolidates a lot of the lifecycle methods and makes the stateful logic much more maintainable and thus, reusable.

Verdict

React functional components with React Hooks.

Ecosystem

Angular

Angular has the most comprehensive ecosystem. It is a complete solution that contains almost all that is needed to build a web application, including:

- DOM manipulation
- CLI
- State management
- Routing[31]
- Forms[32]
- HTTP Client[33]
- Testing[34]
- Animations[35]
- Schematics[36]
- Server

React

React has the smallest ecosystem out of the three, which includes:

- DOM Manipulation
- State management
- CLI
- Server

[31] https://angular.io/guide/router
[32] https://angular.io/guide/forms-overview
[33] https://angular.io/guide/http
[34] https://angular.io/guide/testing
[35] https://angular.io/guide/animations
[36] https://angular.io/guide/schematics

React has good integration with extensive and numerous official and community libraries that can handle all other necessary features.

Vue

Vue is a middle ground between Angular and React, with an ecosystem that includes:

- CLI[37]
- State management
- Loader[38]
- Router[39]
- State Management Library[40]
- Server

Discussion

Angular by far has the most comprehensive ecosystem, which is why it is so great for large applications. However, React has the best community libraries. However, since this is criteria is about ecosystem, Angular is the obvious winner.

Verdict

Angular

Pros and Cons

Angular

Pros

- Complete frontend development framework and ecosystem.
- Highly structured approach that is good for large applications.
- Very clean separation of concerns.
- TypeScript helps make the code more predictable and error-free.
- Two-way binding makes updating state fast and easy.

[37] https://cli.vuejs.org/
[38] https://vue-loader.vuejs.org/
[39] https://next.router.vuejs.org/
[40] https://vuex.vuejs.org/

Cons

- Overly complex and bloated code for applications that don't warrant it.
- Has a very biased architecture that can make development more rigid.
- Two-way binding is less maintainable and can have unpredictable consequences.

React

Pros

- The architecture gives the developer lots of freedom.
- JSX syntax makes writing dynamic components much more inventive, intuitive, flexible, and robust.
- Component-based architecture makes props feel like HTML attributes and components feel like HTML elements better than other frameworks.
- Have a direct coupling between stateful logic and the UI makes managing state more efficient and makes the code more maintainable.
- Hooks enable re-usable state logic through custom hooks.
- Hooks reduce complexity in managing lifecycle while still being flexible enough to do everything that class-based components can do.
- One-way binding makes passing data around more predictable.
- Changes in layout can be done with predictable JavaScript code rather than relying on directives.

Cons

- Less structured architecture means that it is easier to use React "wrong".
- Less structured approach also means that very large applications might not be as maintainable.
- Third party libraries required for things like routing.
- Less features in general.
- One-way binding makes it more arduous to pass data around.

Vue

Pros

- Structured but also very simple.
- Closest to using regular HTML and JavaScript.
- Easiest to learn.

- Flexible because you can choose how much of the application to control.

Cons

- Does not have proven stability due to not being backed by large corporations and being around not as long.
- Smaller ecosystem than Angular.
- Architecture is not ideal for larger projects.

Discussion

The main utility of a JavaScript framework to our application is its ability to abstract out stateful logic so that we can build our frontend at a higher level, focusing mainly on what we want our frontend to do rather than how. This was the main criteria when evaluating the different frameworks, which was then broken down into more granular criteria to compare.

Angular is a robust solution that offers a complete ecosystem and many features and functionality. It is great for large-scale applications and has many different ways to facilitate state management and component communication, but we found that it has too much overhead for our medium-size application and that we might spend more time fighting the technology than benefiting from it.

Vue offers a very simple solution to frontend development. It is easy to learn and get started with, but it doesn't offer a very intuitive approach and lacks reliable backing.

React is a highly intuitive and flexible framework. It's use of JSX provides coupling with the stateful logic and UI that enables more reactive and dynamic applications. The introduction of React Hooks makes handling state much more maintainable and reusable. It's implementation of a component-based architecture is also more ideal than the other two frameworks.

JSX helps to create the illusion that components are really just HTML elements and props are really just HTML attributes, much more than the other frameworks which require verbose syntax to achieve the same things.

Changing layout dynamically is also more direct with React, where loops and conditionals can be achieved with regular JavaScript code, without having to use things like directives.

When it comes to sharing state data and event handling, React also offers a more direct way to do this, without having to manage lots of boilerplate code.

The main downside to React is that it does not come with native integration with some common functionalities, like Routing. However, this can also be seen as an upside since it lets React focus on doing one thing well, following the Single Responsibility Principle, which was a major reason from separating the frontend from our ASP.NET Core backend in the first place.

Verdict

React

React offers the best solution for our needs and we would like to request approval to use it for our application.