Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44000 Internet Applications

Week 4 Lecture 1

Conor Sheedy

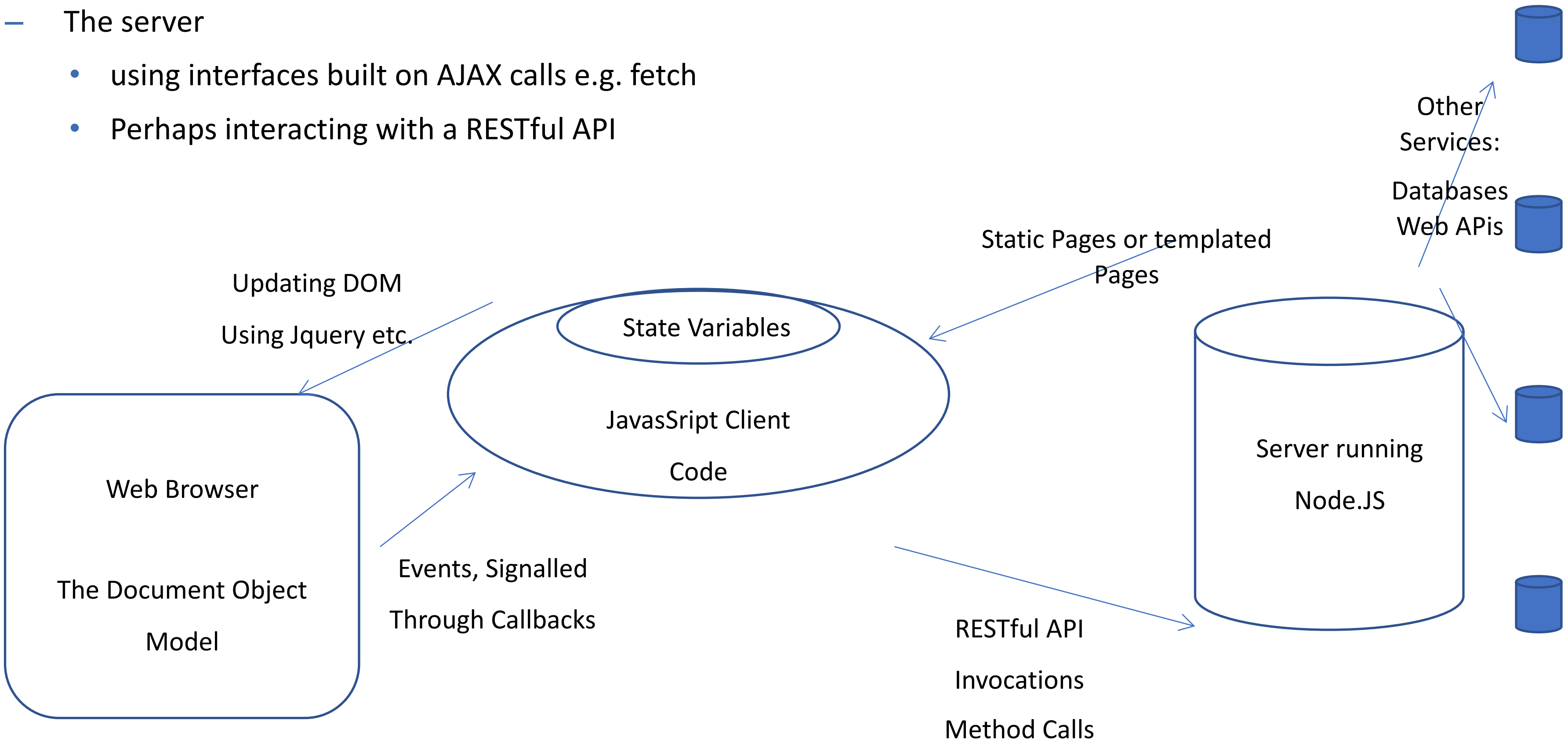
API overview

- **Good API design**
  - RESTful
- **An example of writing a REST API**
  - Using express
- **Using a web API**
  - Tools like postman
- **Learn more**
  - Rest APIs, 100 second overview
    - https://www.youtube.com/watch?v=-MTSQjw5DrM
  - Using web APIs, 2 hour 20 min course
    - https://www.youtube.com/watch?v=GZvSYJDk-us
  - API testing with postman, 2 hour 10 min course
    - https://www.youtube.com/watch?v=VywxIQ2ZXw4

Browser (Client) Side Frameworks

Code running in the browser interacts with:

- The User Interface via the DOM
- The server
  - using interfaces built on AJAX calls e.g. fetch
  - Perhaps interacting with a RESTful API



Reactive Frameworks

- Frameworks are a substantial piece of JavaScript code
- That you import into the browser
- That make implementation easier

- **Borrows an idea from the Spreadsheet concept**
  - Dependencies are updated automatically
- **Variables within JavaScript are ‘linked’ to variables within the HTML document**
- **Simply updating the value of a JavaScript variable will cause the ‘linked’ HTML variables to ‘React’ or automatically update**

Model View Controller Paradigm

- a software architectural pattern
- commonly used for developing user interfaces
- Developed in the context of Smalltalk-79 in the 1970s
- applied to web applications

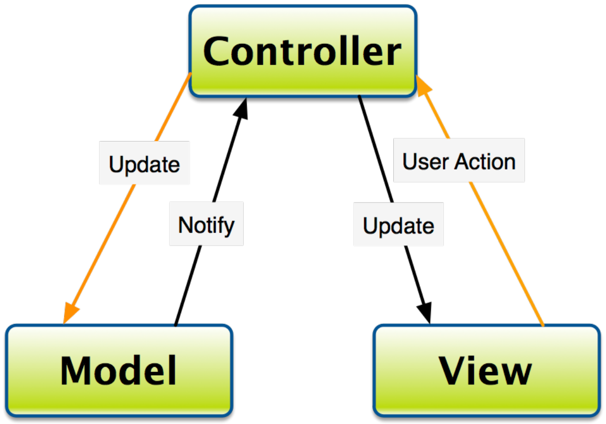
Helps add structure to a Web Application

by dividing functionality into:

– **Model**
  - the basic data that an application is dealing with

– **View**
  - represents something that a user sees which is derived from the underlying data
  - There could be multiple different views of the same model

– **Controller**
  - represents the business logic that controls the application and moves from one view to another



AngularJS

AngularJS uses the Model View Controller Paradigm

- a framework introduced by Google in October, 2010
- Open Source
- Popular

  - Since 2016, there is also an Angular framework which is a ground-up re-write of AngularJS done in Typescript
  - This is the current version of Angular and AngularJS is no longer being supported

React

- **Introduced in May, 2013**
- **developed by Community**
- **supported by Facebook**
- **Used for Facebook Newsfeed in 2011**
- **Used for Instagram in 2012**
- **Used a Virtual DOM**
  - a copy of the DOM
  - so that rendering can be minimized on update

Vue.js

- **Created by Evan You after working for Google using AngularJS**
  - “I figured what if I could just extract the part I really liked about Angular and build something really lightweight”
- **Released February, 2014**
- **Used by:**
  - Facebook for part of its Newsfeed
  - Grammarly
  - GitLab
  - Recommended for smaller projects

Popularity of front end frameworks

The top three:

- React
- Angular
- Vue

There are always trends to read about:

https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190

https://stackdiary.com/front-end-frameworks/

e.g. Svelte voted ‘**most loved**’ front end framework

https://insights.stackoverflow.com/survey/2021#overview

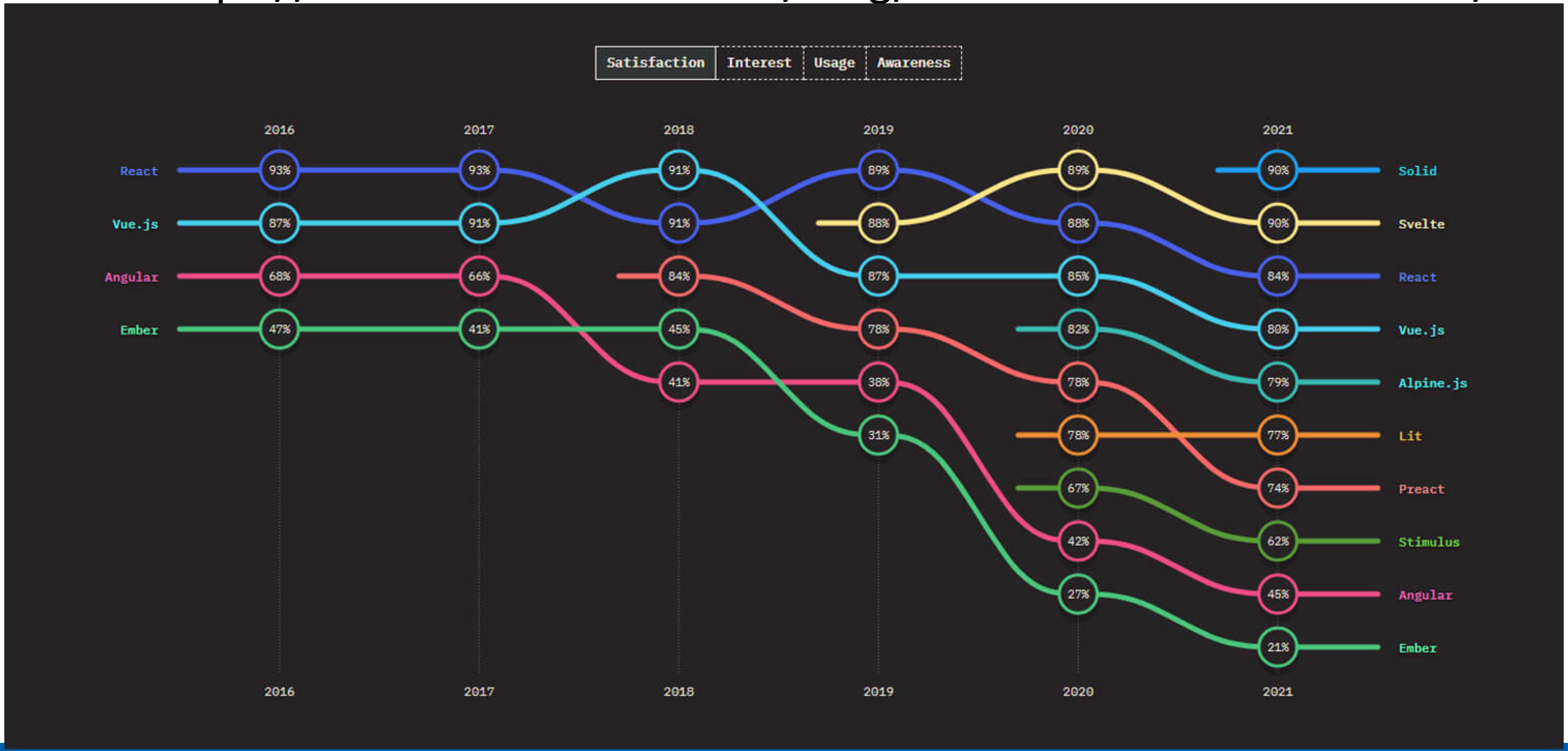
https://svelte.dev/

https://www.infoworld.com/article/3618748/hands-on-with-svelte.html

Popularity of technology stack

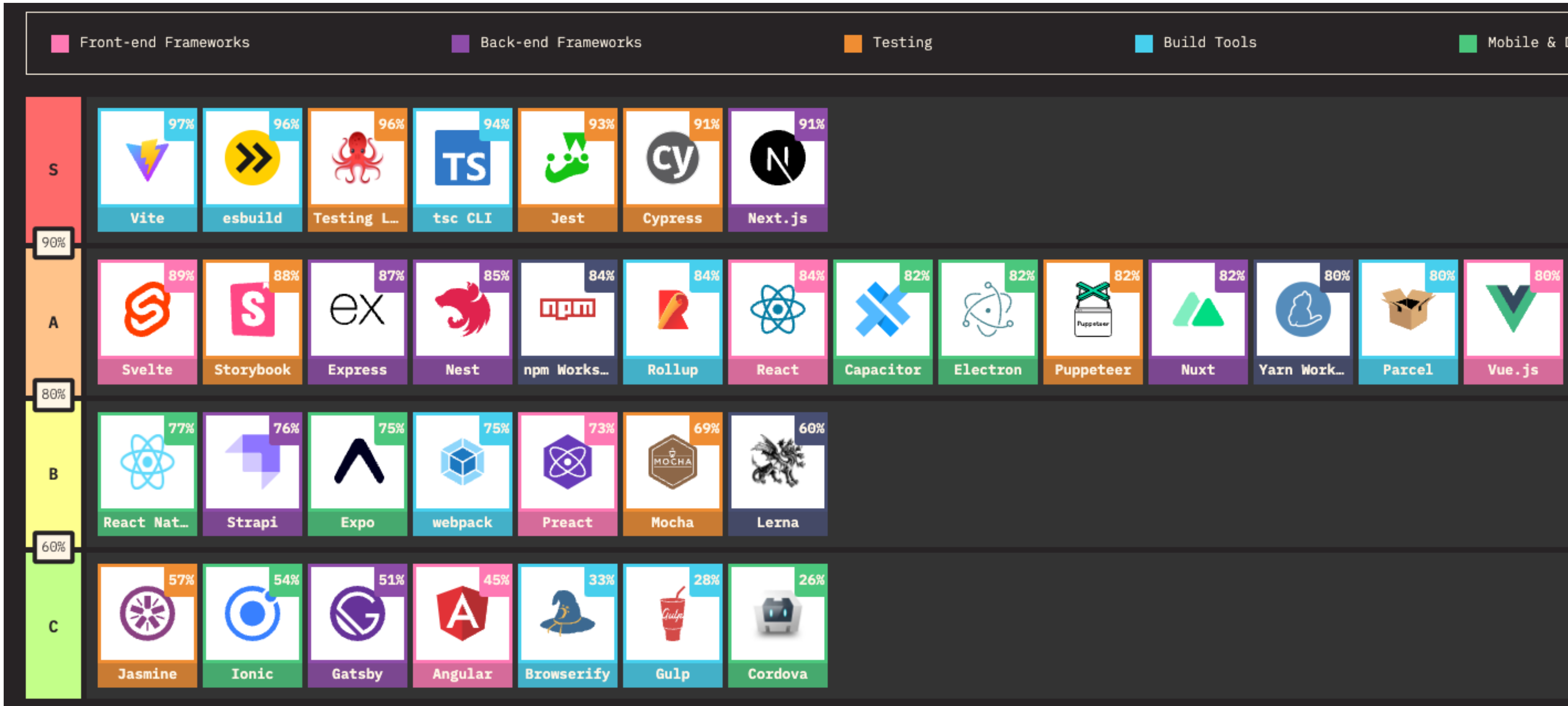
- **There are lots of trends to read about**
  - Choice can depend on the nature of the project
  - The availability of developers

– https://www.monocubed.com/blog/best-front-end-frameworks/



Popularity of js libraries

- **Retention ratio**
  - How likely are you to use it again
    - https://2021.stateofjs.com/en-US/libraries



🟪 Front-end Frameworks 🟪 Back-end Frameworks 🟧 Testing 🟦 Build Tools 🟩 Mobile &

S	Vite 97%	esbuild 96%	Testing L… 96%	tsc CLI 94%	Jest 93%	Cypress 91%	Next.js 91%							
90%														
A	Svelte 89%	Storybook 88%	Express 87%	Nest 85%	npm Works… 84%	Rollup 84%	React 84%	Capacitor 82%	Electron 82%	Puppeteer 82%	Nuxt 82%	Yarn Work… 80%	Parcel 80%	Vue.js 80%
80%														
B	React Nat… 77%	Strapi 76%	Expo 75%	webpack 75%	Preact 73%	Mocha 69%	Lerna 60%							
60%														
C	Jasmine 57%	Ionic 54%	Gatsby 51%	Angular 45%	Browserify 33%	Gulp 28%	Cordova 26%							

Notable Mentions

- **Next.js**
  - back end framework
    - enabling React-based web applications
      - with server-side rendering
      - also generates static websites
- **Svelte**
  - compiles HTML templates
    - manipulates the DOM directly
    - client performance
      - reduce the size of transferred files
      - No virtual DOM
        - React and Vue do at runtime in the browser

Vue

Declarative Rendering

- extends standard HTML
- template syntax
- declaratively describe HTML output
- based on JavaScript state

Reactivity

- automatically tracks JavaScript state changes
- efficiently updates the DOM
  - Using a virtual DOM

Including Vue

Can include all the code in a HTML file drawn from a Content Distribution Network (CDN)

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
```

Or can install it on the development machine and Browserify or Webpack it into the production install

- Allows you to bring in just the parts of Vue.js that you are using.
- And include other libraries if necessary
  - Framework tool often do this ‘behind the scenes’
    - E.g. Vue CLI uses Webpack to bundle

Simple Vue Example

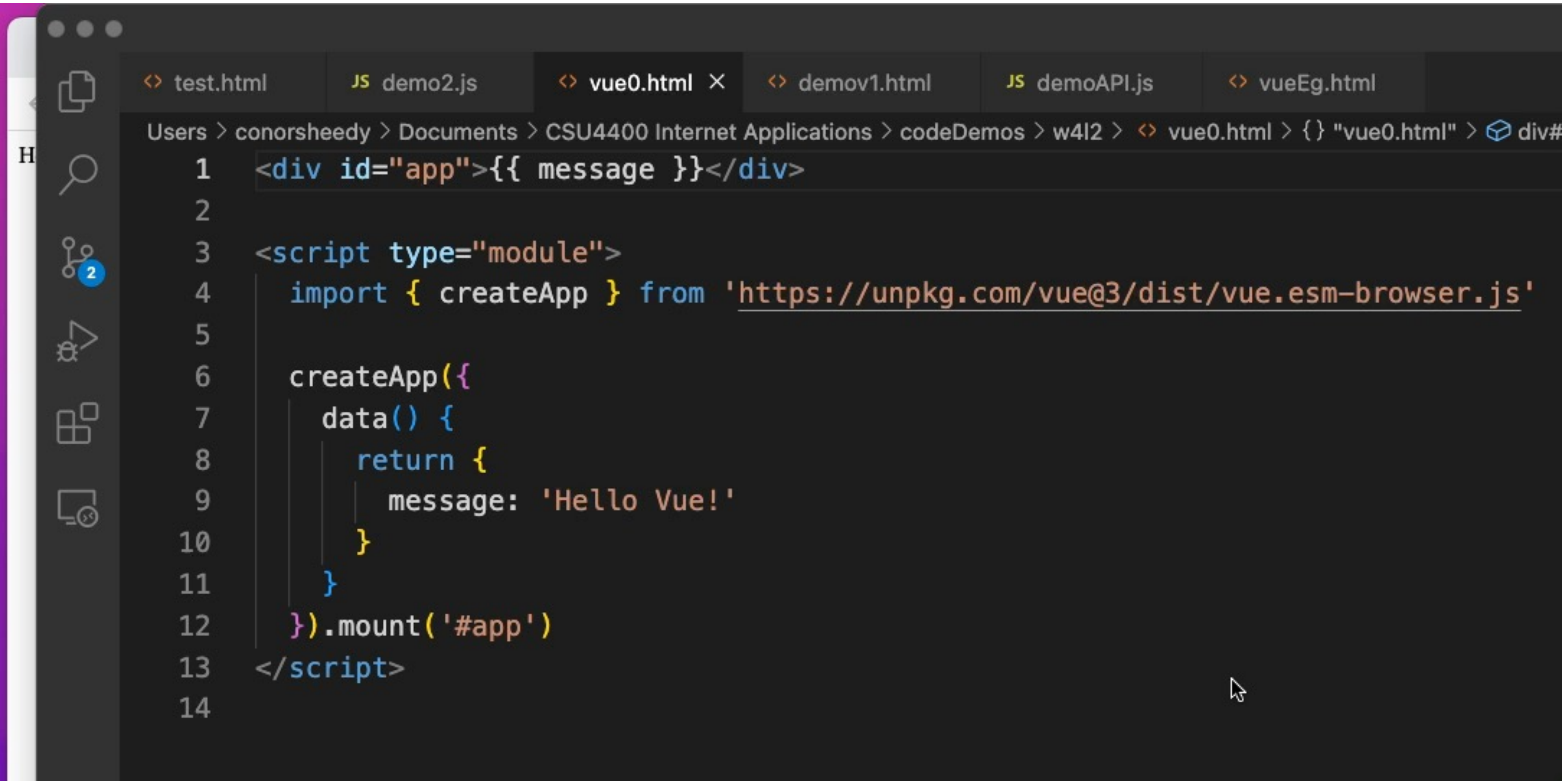
Browsers load Vue code from the web – usually via CDN

The Vue Component has

- A ‘data’ section – all variables in here are made ‘reactive’
- The HTML uses {{ Syntax }}
  - Moustache syntax
  - which allows variable values to be picked up from JavaScript

```
<div id="app">{{ message }}</div>


<script type="module">
  import { createApp } from
'https://unpkg.com/vue@3/dist/vue.esm-browser.js'


  createApp({
    data() {
      return {
        message: 'Hello Vue!'
      }
    }
  }).mount('#app')
</script>
```

Simple Vue Example



test.html
demo2.js
vue0.html
demov1.html
demoAPI.js
vueEg.html
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l2 > vue0.html > {} "vue0.html" > div#

```html
<div id="app">{{ message }}</div>

<script type="module">
  import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    data() {
      return {
        message: 'Hello Vue!'
      }
    }
  }).mount('#app')
</script>
```

Simplifying DOM interaction

JavaScript already has native methods for interacting with the DOM

- Directly adding/removing/modifying elements
- Attaching event handler that are triggered on events
- Libraries like Jquery that make it easier to navigate

Vue (and other frameworks) make this much easier

Vue constructs a virtual DOM

- updates this
- then decides how much of the real DOM to modify
- Determines what is re-rendered

Vue is a Progressive Framework

- **Flexible**
- **incrementally adoptable**
  - We will look at adding elements to html
    - Using ‘Options API’
  - Using html directives
  - There are other approaches
    - e.g. Single-File Component (SFC)
      - .vue files
      - Requires a build setup
      - Allows easier use of Composition API
        - More abstraction
        - Arguably nicer syntax

Vue HTML Directives

See: https://vuejs.org/guide/introduction.html

v-if

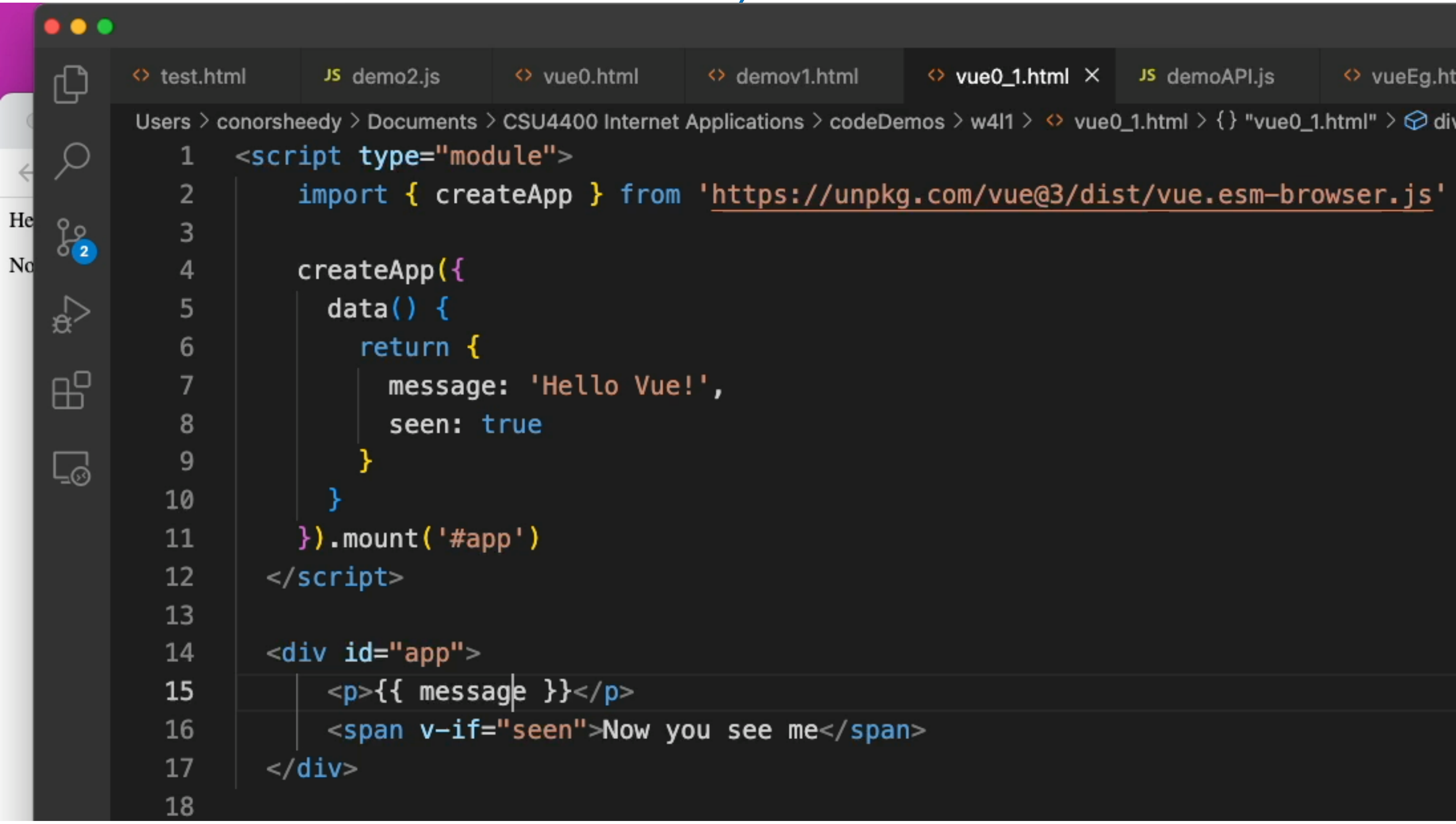
- **toggles the presence of an element**

```
<div id="app">
        <span v-if="seen">Now you see me</span>
</div>
```

Vue HTML Directives, v-if demo

```
<script type="module">
  import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    data() {
      return {
        message: 'Hello Vue!',
        seen: true
      }
    }
  }).mount('#app')
</script>

<div id="app">
    <p>{{ message }}</p>
    <span v-if="seen">Now you see me</span>
</div>
```

Vue HTML Directives, v-if demo


test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
demoAPI.js
vueEg.ht
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > vue0_1.html > {} "vue0_1.html" > div

```
<script type="module">
    import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

    createApp({
      data() {
        return {
          message: 'Hello Vue!',
          seen: true
        }
      }
    }).mount('#app')
  </script>

  <div id="app">
    <p>{{ message }}</p>
    <span v-if="seen">Now you see me</span>
  </div>
```

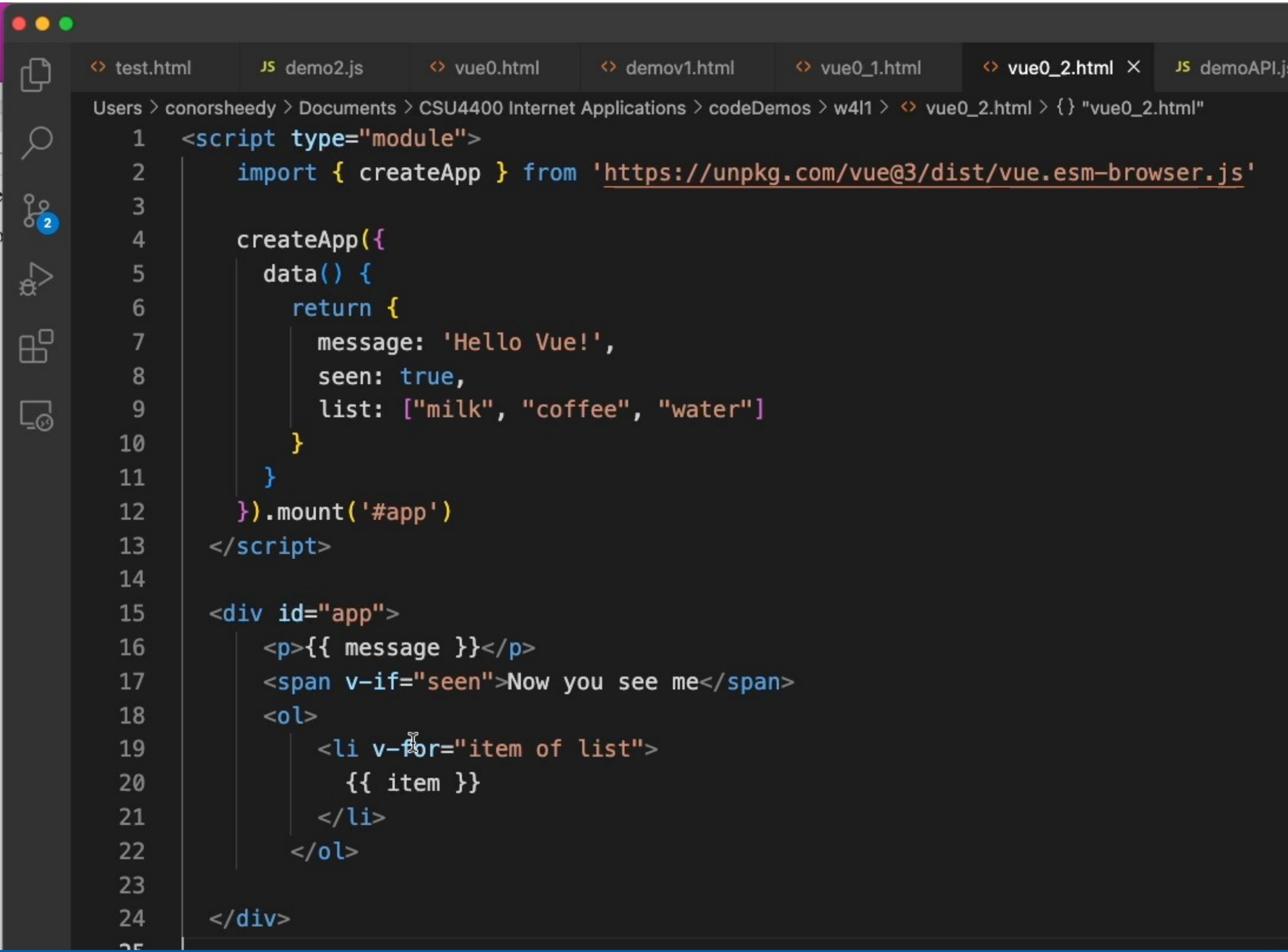
Vue HTML Directives

v-for **allows iteration within an element**

```
<ol>
  <li v-for=”item of list">
   {{ item }}
  </li>
 </ol>
```

Vue HTML Directives, v-for demo

```
<script type="module">
 import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

 createApp({
  data() {
   return {
    message: 'Hello Vue!',
    seen: true,
    list: ["milk", "coffee", "water"]
   }
  }
 }).mount('#app')
</script>

<div id="app">
  <p>{{ message }}</p>
  <span v-if="seen">Now you see me</span>
  <ol>
    <li v-for="item of list">
     {{ item }}
    </li>
   </ol>

</div>
```


test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoAPI.j
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > vue0_2.html > {} "vue0_2.html"

```html
<script type="module">
    import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

    createApp({
      data() {
        return {
          message: 'Hello Vue!',
          seen: true,
          list: ["milk", "coffee", "water"]
        }
      }
    }).mount('#app')
  </script>

  <div id="app">
      <p>{{ message }}</p>
      <span v-if="seen">Now you see me</span>
      <ol>
          <li v-for="item of list">
            {{ item }}
          </li>
        </ol>

  </div>
```

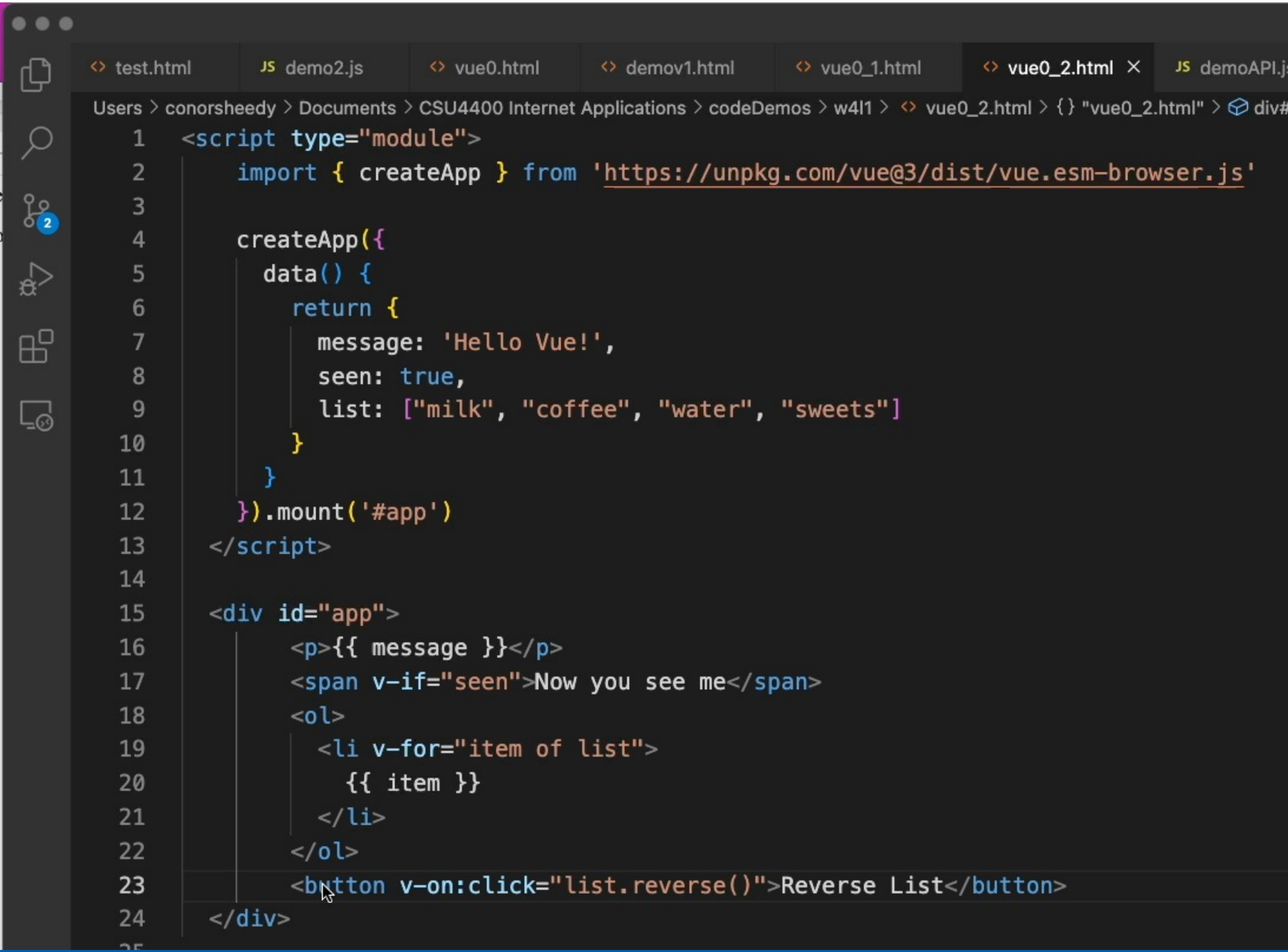
Vue HTML Directives, v-on

**v-on	allows event handlers to be connected**

- e.g. calls a function on the click event
  - Two versions of syntax

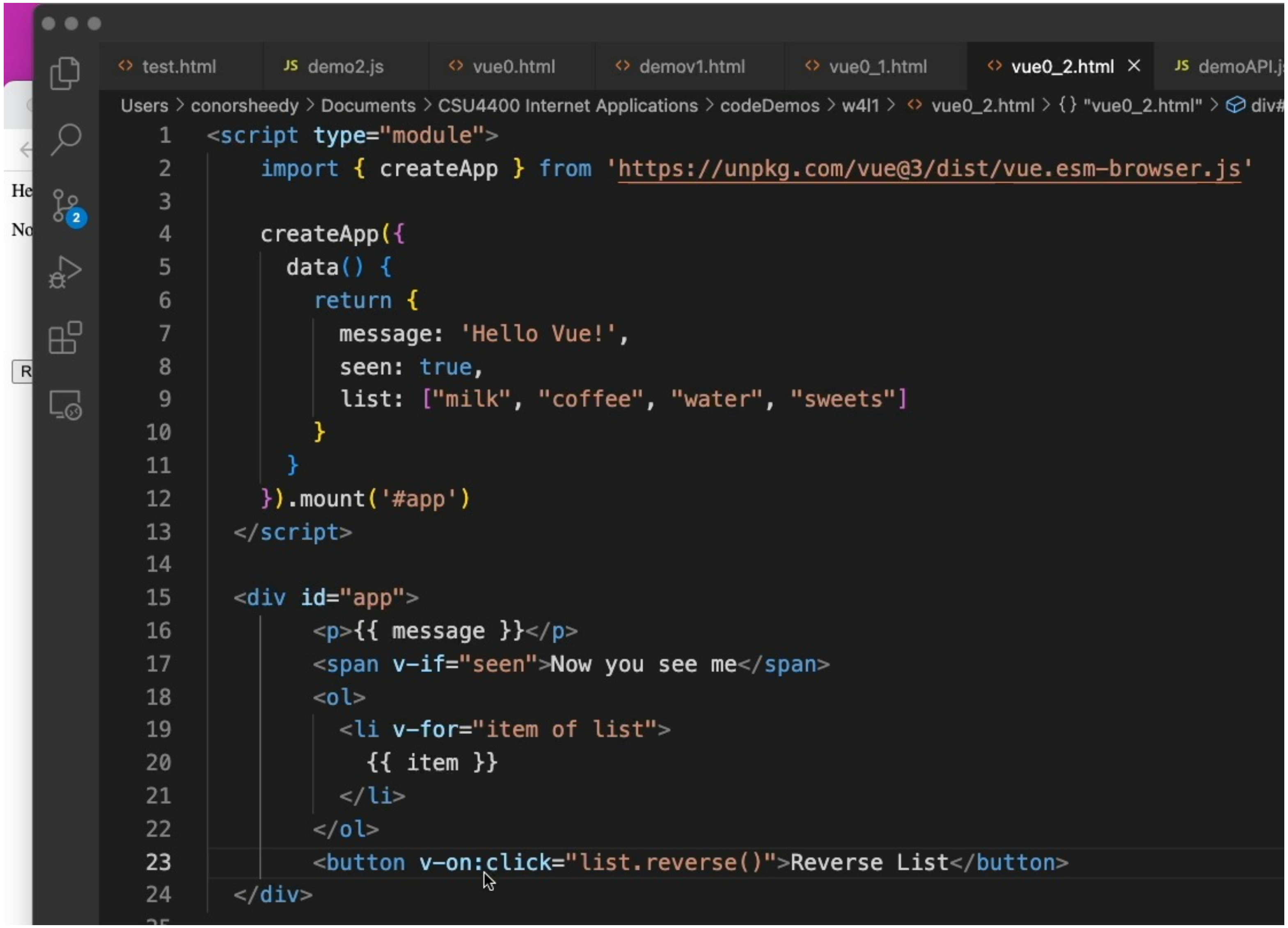
```
  <button v-on:click="list.reverse()">Reverse List</button>
```

```
<button @click="list.reverse()">Reverse List</button>
```


test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoAPI.j
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > vue0_2.html > {} "vue0_2.html" > div#

```
<script type="module">
    import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

    createApp({
      data() {
        return {
          message: 'Hello Vue!',
          seen: true,
          list: ["milk", "coffee", "water", "sweets"]
        }
      }
    }).mount('#app')
  </script>

  <div id="app">
        <p>{{ message }}</p>
        <span v-if="seen">Now you see me</span>
        <ol>
          <li v-for="item of list">
            {{ item }}
          </li>
        </ol>
        <button v-on:click="list.reverse()">Reverse List</button>
  </div>
```

Vue HTML Directives, v-on, @


test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoAPI.j
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > vue0_2.html > {} "vue0_2.html" > div#

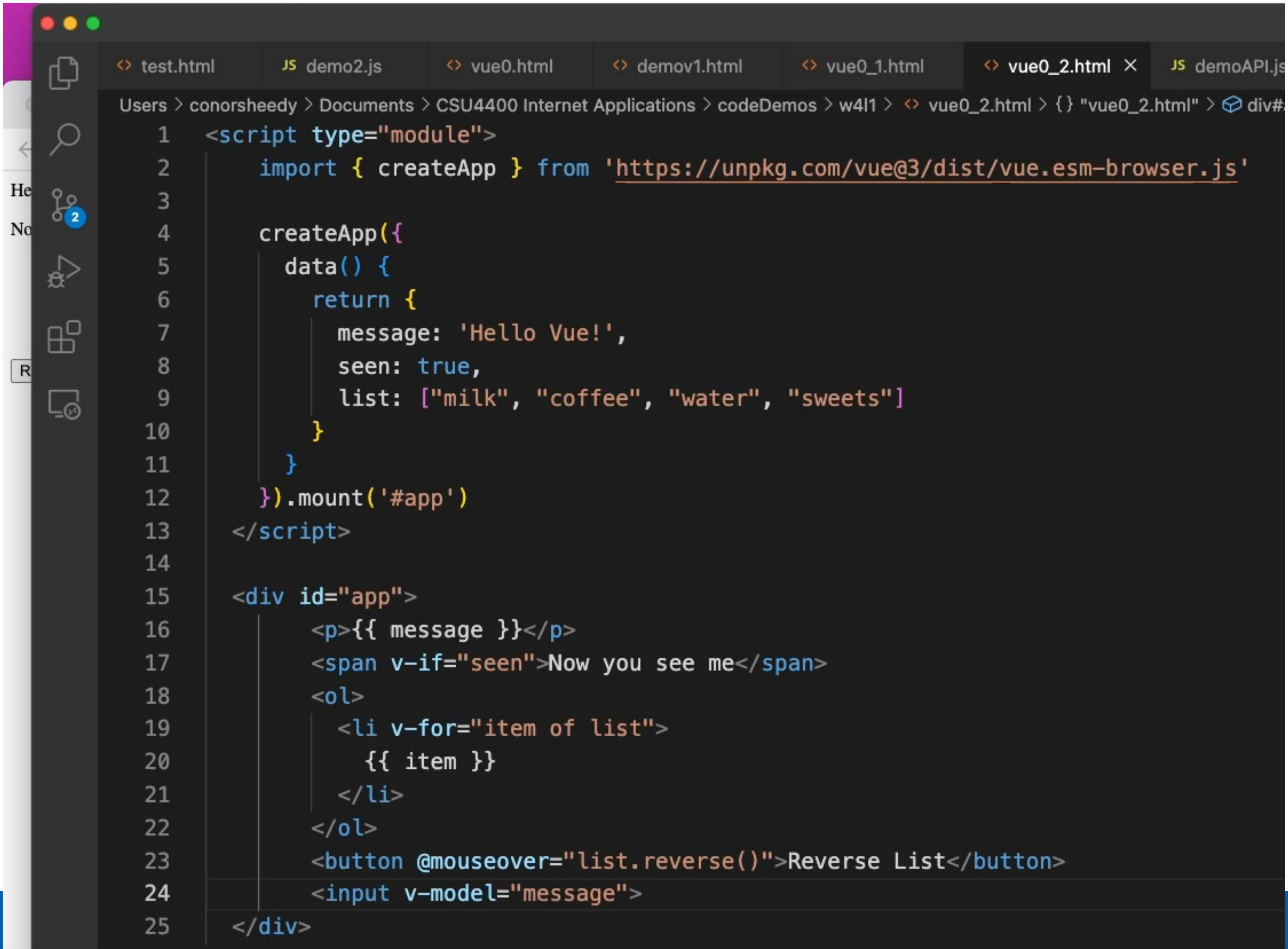
```
<script type="module">
    import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

    createApp({
      data() {
        return {
          message: 'Hello Vue!',
          seen: true,
          list: ["milk", "coffee", "water", "sweets"]
        }
      }
    }).mount('#app')
  </script>

  <div id="app">
        <p>{{ message }}</p>
        <span v-if="seen">Now you see me</span>
        <ol>
          <li v-for="item of list">
            {{ item }}
          </li>
        </ol>
        <button v-on:click="list.reverse()">Reverse List</button>
  </div>
```

Two-Way binding, v-model

- **We have seen one-way binding between JavaScript (reactive) variables and the rendered HTML**
  - using {{   variable }}
  - called ‘mustache syntax’
- **When it comes to INPUT elements**
- **we can do two-way binding with V-model**
  - Allow input to update variable value
  - Which updates the message

```
<input v-model="message">
```

Two-Way binding, v-model



test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoAPI.js
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > vue0_2.html > {} "vue0_2.html" > div#

```
<script type="module">
    import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

    createApp({
      data() {
        return {
          message: 'Hello Vue!',
          seen: true,
          list: ["milk", "coffee", "water", "sweets"]
        }
      }
    }).mount('#app')
  </script>

  <div id="app">
        <p>{{ message }}</p>
        <span v-if="seen">Now you see me</span>
        <ol>
          <li v-for="item of list">
            {{ item }}
          </li>
        </ol>
        <button @mouseover="list.reverse()">Reverse List</button>
        <input v-model="message">
  </div>
```

Methods in Vue

We can add functions to the Vue 'methods' object when creating the app, e.g. methods: { DoSomething:doSomething}

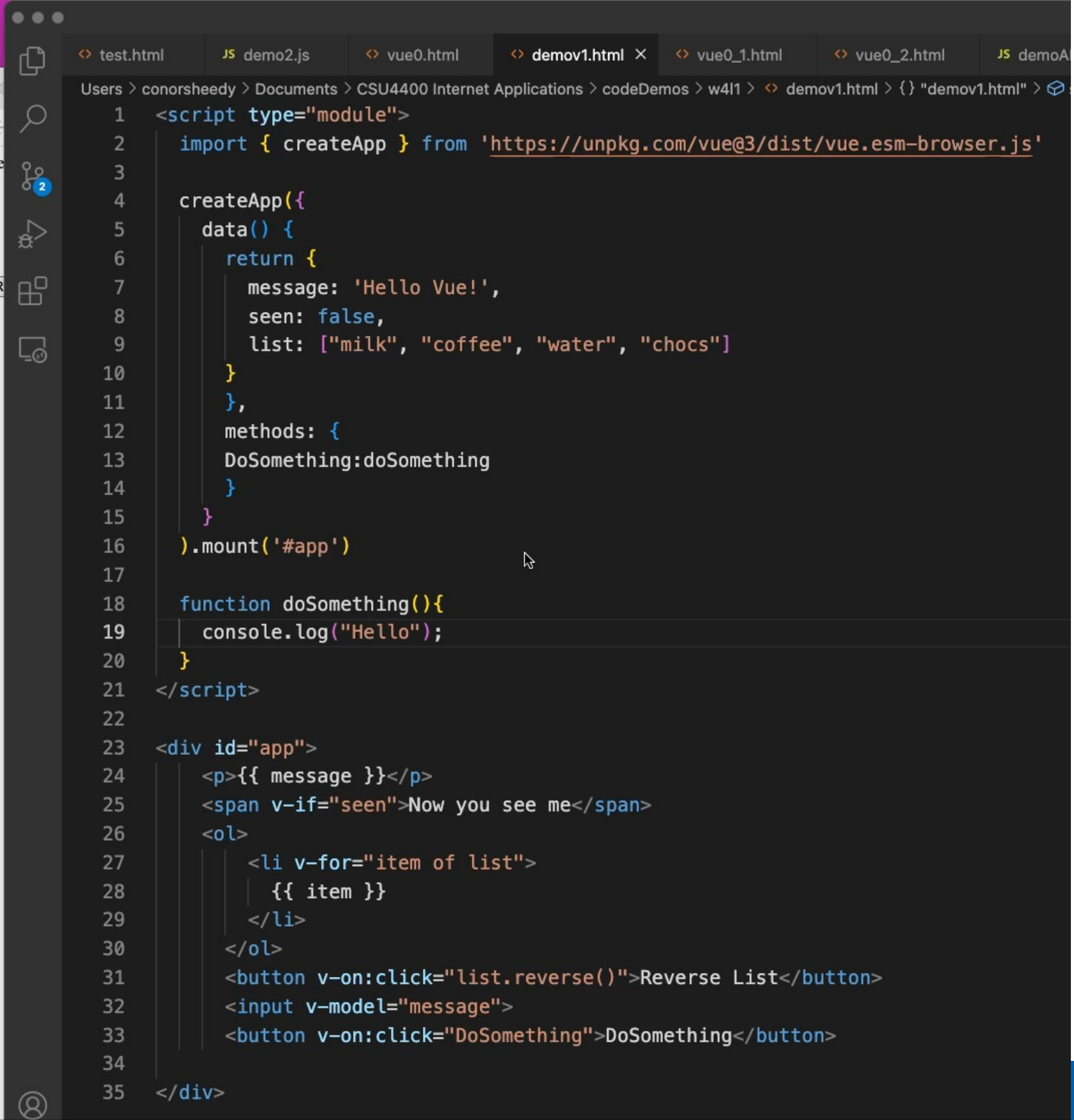
```
<script type="module">
  import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    data() {
      return {
        message: 'Hello Vue!',
        seen: true,
        list: ["milk", "coffee", "water"]
      }
      },
      methods: {
      DoSomething:doSomething
      }
    }
  ).mount('#app')

  function doSomething(){
    console.log("Hello");
  }
</script>
```

```
<div id="app">
    <p>{{ message }}</p>
    <span v-if="seen">Now you see me</span>
    <ol>
        <li v-for="item of list">
          {{ item }}
        </li>
      </ol>
      <button v-on:click="list.reverse()">Reverse
List</button>
      <input v-model="message">
      <button v-
on:click="DoSomething">DoSomething</button>


</div>
```

M

He

R


test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoA
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l1 > demov1.html > {} "demov1.html" >

```
<script type="module">
  import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    data() {
      return {
        message: 'Hello Vue!',
        seen: false,
        list: ["milk", "coffee", "water", "chocs"]
      }
      },
      methods: {
      DoSomething:doSomething
      }
    }
  ).mount('#app')

  function doSomething(){
    console.log("Hello");
  }
</script>

<div id="app">
    <p>{{ message }}</p>
    <span v-if="seen">Now you see me</span>
    <ol>
        <li v-for="item of list">
          {{ item }}
        </li>
      </ol>
      <button v-on:click="list.reverse()">Reverse List</button>
      <input v-model="message">
      <button v-on:click="DoSomething">DoSomething</button>

</div>
```

Our first Internet Application

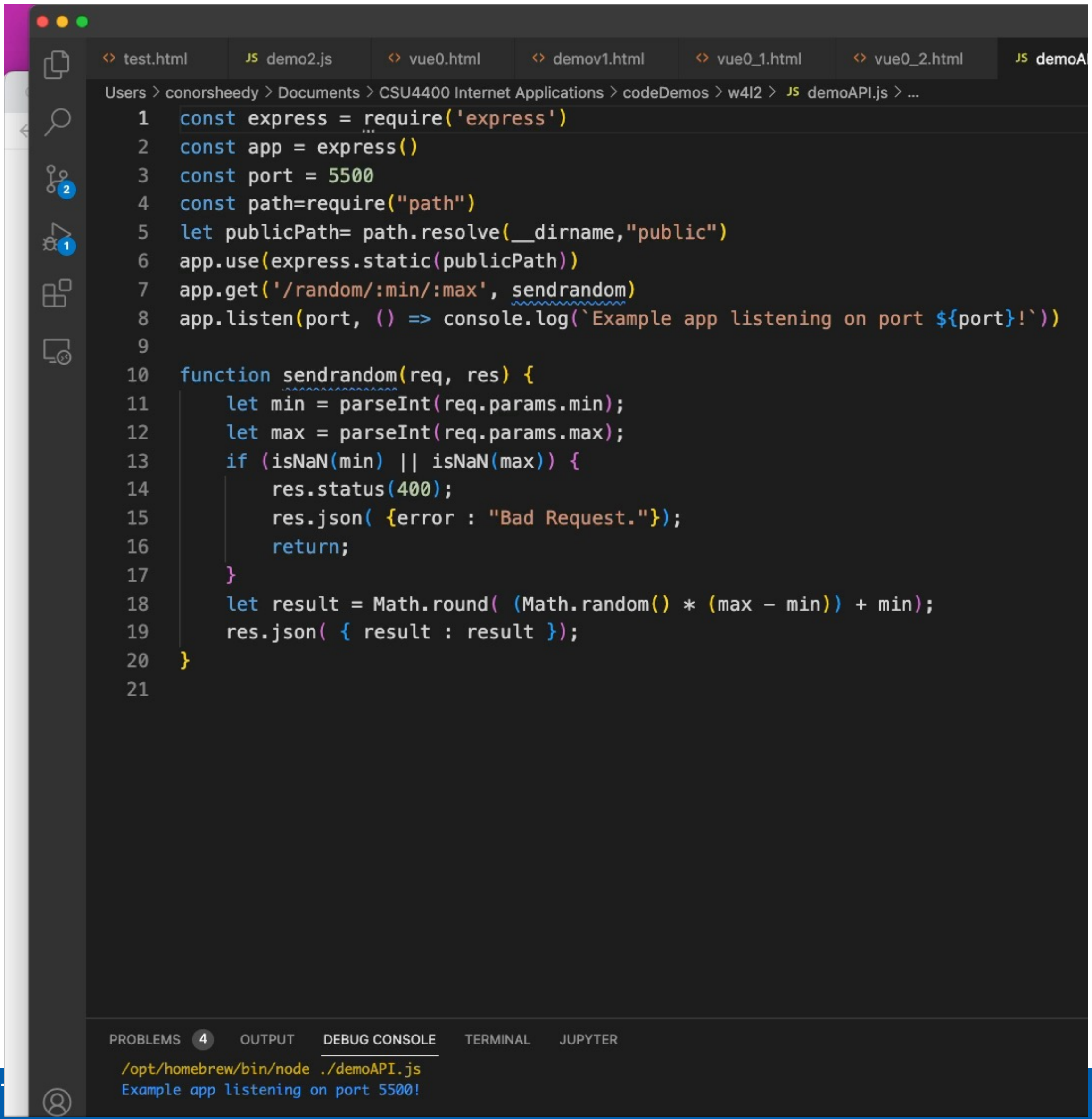
- **We’ve now seen enough examples to create our first Internet Application with:**
  - A server with a RESTful API
    - Express
      - Random number server example
      - We’ve seen how to serve static html from our local directory
        - We can also serve files with Vue directives in it.
  - A client with a reactive front end framework
    - Vue

Example – Server Side

- **Serves static files – including index.html which contains the client code**
- **Also generates a random numer in response to GET /random/x/y**
- **Result in JSON fmt**
- **{ result: 86}**

```
const express = require('express')
const app = express()
const port = 3000

const path=require("path")
let publicPath= path.resolve(__dirname,"public")
app.use(express.static(publicPath))

app.get('/random/:min/:max', sendrandom)

app.listen(port, () => console.log(`Example app listening on port ${port}!`))

function sendrandom(req, res) {
   let min = parseInt(req.params.min);
   let max = parseInt(req.params.max);
   if (isNaN(min) || isNaN(max)) {
      res.status(400);
      res.json( {error : "Bad Request."});
      return;
   }
   let result = Math.round( (Math.random() * (max - min)) + min);

   res.json( { result : result });
}
```

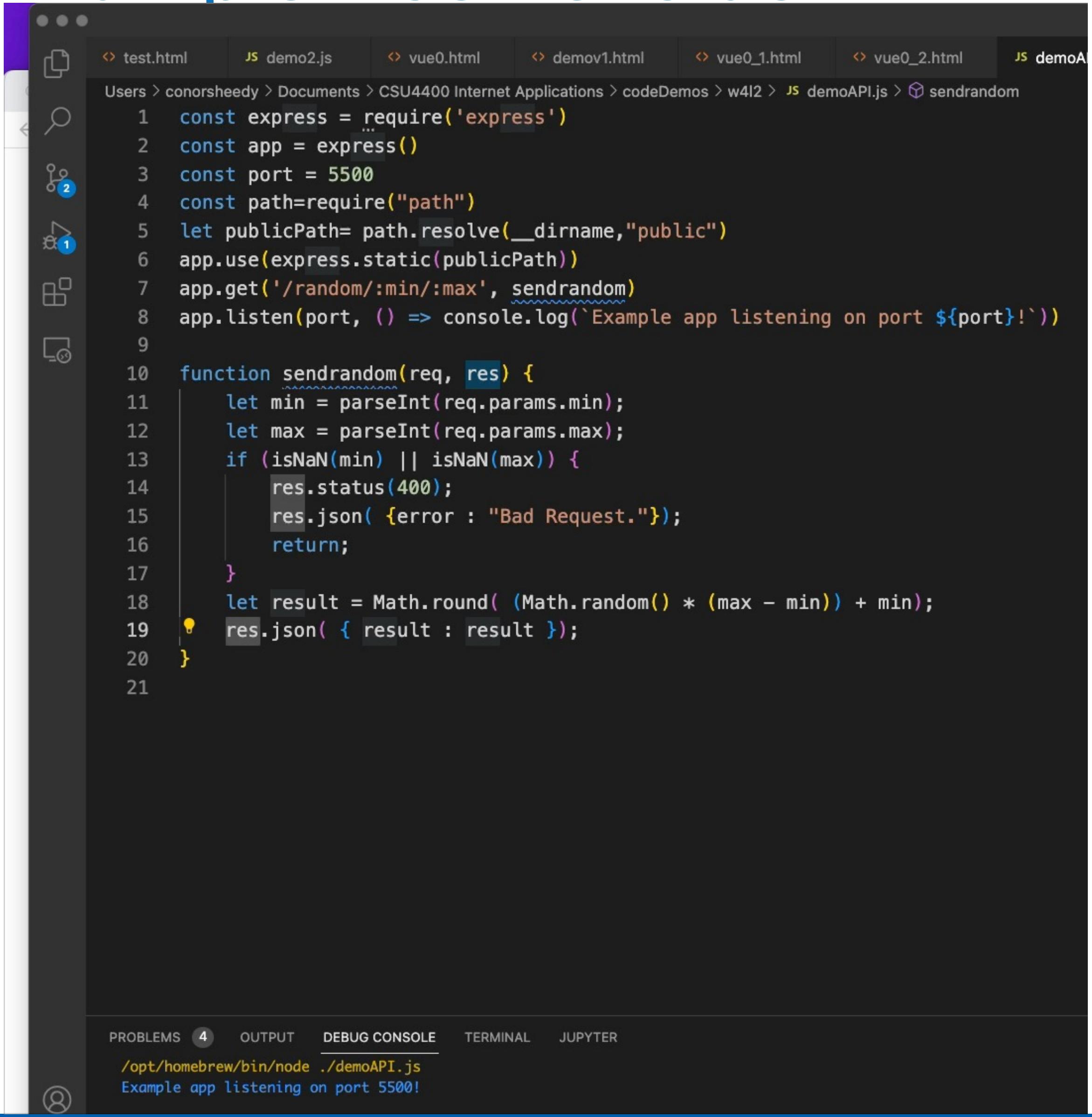
Example – Server Side



```javascript
const express = require('express')
const app = express()
const port = 5500
const path=require("path")
let publicPath= path.resolve(__dirname,"public")
app.use(express.static(publicPath))
app.get('/random/:min/:max', sendrandom)
app.listen(port, () => console.log(`Example app listening on port ${port}!`))

function sendrandom(req, res) {
    let min = parseInt(req.params.min);
    let max = parseInt(req.params.max);
    if (isNaN(min) || isNaN(max)) {
        res.status(400);
        res.json( {error : "Bad Request."});
        return;
    }
    let result = Math.round( (Math.random() * (max - min)) + min);
    res.json( { result : result });
}
```

```
/opt/homebrew/bin/node ./demoAPI.js
Example app listening on port 5500!
```

Example – Server Side



test.html
demo2.js
vue0.html
demov1.html
vue0_1.html
vue0_2.html
demoA
Users > conorsheedy > Documents > CSU4400 Internet Applications > codeDemos > w4l2 > demoAPI.js > sendrandom
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER

```
const express = require('express')
const app = express()
const port = 5500
const path=require("path")
let publicPath= path.resolve(__dirname,"public")
app.use(express.static(publicPath))
app.get('/random/:min/:max', sendrandom)
app.listen(port, () => console.log(`Example app listening on port ${port}!`))

function sendrandom(req, res) {
    let min = parseInt(req.params.min);
    let max = parseInt(req.params.max);
    if (isNaN(min) || isNaN(max)) {
        res.status(400);
        res.json( {error : "Bad Request."});
        return;
    }
    let result = Math.round( (Math.random() * (max - min)) + min);
    res.json( { result : result });
}
```

```
/opt/homebrew/bin/node ./demoAPI.js
Example app listening on port 5500!
```

Client Side

Shows

- V-IF
- V-for
- V-Model
- Mustache syntax

```
 <!-- development version, includes helpful console warnings -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<div id="app">

<h1>Welcome to the Random Number Generator</h1>
Please enter a minimum and a maximum as the range of the Random Number
<br>
Minimum: <input v-model="randMin"> <input v-model="randMax"> <br><br>
<button v-on:click="GetRand">Generate Random Number</button>
 between {{randMin}} and {{randMax}}
 <br> Latest Random Number is {{random}}
 <span v-if="history.length>0">
<hr>
 <h1>History of Numbers generated</h1>
 <ul>
    <li v-for="num in history">{{num}}</li>
 </ul>
 </span>
</div>

 <script> ...............................
```

Vue App in HTML Page

App contains reactive pool

Fetch

- Only available in browser
- Returns a promise
- Resolves to a stream
- .json() extracts result from stream in the form of a JSON obj
- Extract the random number and put into reactive variables
- Subject to same-origin policy:
- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

```
 <script>
  var app = new Vue({
   el:'#app',
   data:{
   randMin: 1,
   randMax: 10,
   random: -1,
   history :[]},
   methods:{
      GetRand : getRandom }

  })

function getRandom (){
      console.log("getRandom called")
      let prom = fetch("random/"+this.randMin+"/"+this.randMax)
      prom.then( response => response.json())
         .then (response =>
         { this.random = response.result
           this.history.push(response.result)
         })}

</script>
```

Other Vue Features

Loads of other features

Hierarchical system of Vue Components

Vue-cli

- a command line interface that makes it easy to build a Vue project
- Facilitates scaffolding
- Not needed for the project to ensure depth of understanding

Vue-Router

- allows client-side routing

Vue-Devtools

- a browser extension making it easy to debug

Making Vue Pages look nice

- **Can use HTML, CSS**
- **Also many different toolkits with pretty UI Elements**
- **15 of the most interesting UI Component Libraries for 2022:** https://www.codeinwp.com/blog/vue-ui-component-libraries/
- Easy to pick a style that you like
- Import a library
  - E.g. Vue Material kit
  - Live preview

Course – Part II - Packaging

- **Inspiration: The Story of the Shipping Container**
  - https://www.youtube.com/watch?v=0MUkgDIQdcM
  - The advantages of standardisation
  - Scalability
  - Cost reduction

Attributes of the Shipping Container

Aluminium Box

- Same shape and size everywhere
- don’t have to weight the same though

Easy interfaces to the box

- lugs at the corners to lift it and fasten it down

Make everything else to fit the box

- –Trucks, Rail Cars, Ship’s holds, Canal sizes…..

Harmonize costs associated with moving the box over distance

Make easy to understand rules on how to handle the box (customs etc)

Invented in 1956

- Today 95% of all products moved are in shipping containers

Analogy to Cloud Computing

Standardisation of computing

- Turns computing into a commodity
  - Compute
  - Memory