Angular Questions

- Node.js: a runtime environment for running JavaScript outside the browser
  - npm (node package manager): analogous to Maven
- package.json: contains a list of dependencies for node projects (node packages) analogous to pom.xml
  - devDependencies: needed only for development
  - Dependencies: dependencies needed to run the application (included for the entire project)
  - Angular projects ARE node projects

- Databinding: the purpose of databinding is to pass information between the .html template and the .ts file
  - One-way
    - Property binding: to pass information from component class .ts to the attributes of different elements in the .html
      - <img [src]="someVariable" />
      - Parent-to-child communication
        - Passing data from a parent component to a child component
        - <app-my-component [someProperty]="myVar"></app-my-component>
          - @Input("someProperty") let x: string = ""; **IN THE CHILD COMPONENT**
    - Event binding: invokes function that exists in component class
      - From template (.html) to component class (.ts)
      - <button (click)="someFunction()">Click me</button>
      - Child-to-parent communication
        - <app-my-component (someEvent)="anotherFunction($event)"></app-my-component>
          - @Output("someEvent") let ee: EventEmitter<String> = new EventEmitter<String>();
          - this.ee.emit("some string");
    - String interpolation: from component class (.ts) to template (.html)
  - Two-way
    - Connect .html and .ts both ways
    - If the variable changed in the .ts file, then the value over in the .html will change as well
    - And vice versa
    - .html
      - What element? <input type="text" **[(ngModel)]="myVar"** />
    - .ts
      - let myVar: string = "";
    - What do you need to configure?
      - Inside of AppModule, import FormsModule
      - @NgModule({ …., imports: [FormsModule, ...]}

- Observable
- When we’re using Angular, “how do we consume an API”?
    - Sending HTTP Requests and receiving responses
    - HttpClient
    - Constructor(private http: HttpClient) {}
    - this.http.get(‘…’, { withCredentials: true })
        - What does the .get() function return?
        - Returns an observable
    - Drawing an analogy, what does fetch(…) return?
        - Returns a promise
- Promise v. Observable
    - Promise: something that will happen in the future, and will emit only 1 value
    - Observables: something that will happen in the future, but can emit multiple values
        - A stream of values
            - It is optional to have multiple values
            - Does an observable from HttpClient emit multiple values? No, only 1 value
            - So why does Angular even utilize Observables for HttpClient?
                - The developers of Angular just decided to make use of a library called RxJS which introduced observables
                - RxJS is an advanced library with plenty of features that we can make use of throughout an Angular application
        - obs.subscribe((res) => { … });

- A Subject is an object that can have multiple publishers and multiple subscribers
  - Think of a Subject as a central hub through which information passes
  - Whenever the subject receives information from a publisher, the Subject will send that information to EVERY subscriber
- Think about subscribing to an online newsletter
  - You’re going to have many authors for different articles
  - You’re going to have people subscribing people subscribing to the newsletter
  - Whenever an author publishes an article, it will be sent to every single subscriber
- let s: Subject = new Subject();
  - Publishing data:
    - someService.s.next(“some data”); Exists in a service
  - Subscribing to data:
    - someService.s.subscribe((data) => { … }); <- Component 1
    - someService.s.subscribe((data) => { … }); <- Component 2
    - someService.s.subscribe((data) => { … }); <- Component 3

Routing

- We need to create a RoutingModule
  - The routing module contains an array with the different routes

    ```
    const routes: Routes = [ { path: 'heroes', component:
    HeroesComponent } ];
    ```
  - <router-outlet></router-outlet> (typically inside of the AppComponent template is the component that actually gets displayed for a certain route)
  - <a routerLink="/heroes">Heroes</a> as a link to the route
- Why do we use routerLink instead of href?
  - href reloads the page
  - routerLink does not

- SPAs (Single page application)
    - Application that has a single page
    - When you navigate through the website, instead of navigating to different .html files, DOM Manipulation changes the elements being displayed
    - We only have a single page (a single .html file)
        - Index.html
- Directives
    - Structural and attribute
        - Structural: reshaping the DOM by adding, removing elements
            - ngSwitch: displays different element “cases” based on a certain value being met
            - ngFor: it allows you to iterate through a data structure such as an array and display a series of elements over and over for each item in the array
            - ngIf: an element and its child elements will be displayed or not displayed (added or removed) based on a Boolean expression.
                - True: it is displayed
                - False: not displayed
                - <h1 *ngIf=“someVar”>Hello world</h1>
                    - someVar has a value of true: h1 is displayed
                    - someVar has a value of false: h1 is not displayed
        - Attribute:
            - ngClass: used to programmatically add or remove classes from an element
            - ngStyle: used to programmatically add or remove CSS styling from an element
        - Components
            - Are actually directives themselves, “specialized directive”

- Pipe: transforms values inside of the component template to a different value
  - {{ aStringVariable | uppercase }}
    - Transform what is displayed from the original string value to something that appears in all uppercase letters
  - {{ aNumberVariable | currency: ‘GBP’ }}
- We can create custom pipes
  - ng generate pipe <pipe name>
    - Or ng g p <pipe name>
  - Creates a class with the @Pipe decorator on top
- Angular CLI: command line interface for Angular
  - Allows us to create Angular projects using ng new
  - Generate components using ng generate component <component name>
    - Or ng g c <component name>

- Root Component
  - The component that is displayed at all times in the Angular application
  - Directly bootstrapped to the index.html (Single page)
  - Other components would then be rendered within the **App Component**
  - If we have routing, we would use <router-outlet></router-outlet> inside of this component template
- Module
  - A collection of pipes, components, custom directives, etc. contained into one unit
  - The module that we are initially given is called the AppModule whenever we generate the Angular project
  - Whenever we generate new components, pipes, directives, etc., these will be declared inside of the AppModule .ts file (unless we want to have them contained in another module that we create)
  - For small apps, AppModule is enough

- TypeScript interfaces
  - Define a “type” for a variable
- Pokemon.ts

```
export interface Pokemon {
        id: number,
        name: string,
        sprites: { front_default: string }
}

let x: Pokemon = { id: 10, name: ‘something’, sprites: { front_default: ‘http://somelink.com/image-1.jpg’ } }
```

- Component Lifecycle: the process a component goes through when it is created and destroyed
- Lifecycle hooks: functions that are executed at various points during a component’s life
- Most common hooks that we should be using
  - Constructor() {}: Whenever the component is first created, you need to instantiate the component itself
    - The Component class is a blueprint for a component object behind the scenes
    - **Dependency Injection** (we need to have services injected into our components so that we can use those services)
  - ngOnInit(): called shortly after we create a component