

IN4MATX 133: User Interface Software

Lecture 10: Components in Angular

Announcements

- Discussion Friday (2/7) will offer more depth into Angular
- You're now over the hardest part of the learning curve in this course
- And for me, we're starting to reach the more fun parts
- Start to take a look at A3!
- #assignment3 on Slack
- You have until 2/17 to declare a partner on Canvas

Announcements

- We're not going to create pairs for A3 because there's a long list of properties which go into creating a good match, most of which we don't know
 - Programming background/interest (frontend, backend, user experience, etc.)
 - Schedule and other commitments

Announcements

- Good questions to ask a potential assignment partner:
 - When do you normally work (morning, afternoon, evening, late evening)?
 - Do you want to meet up in-person or work on the assignment remotely?
 - Are you interested in frontend development, backend development, user experience, etc.?
 - What's the minimum grade you would be comfortable receiving on this assignment?

Today's goals

By the end of today, you should be able to...

- Explain a Model-View-Controller Architecture and how Angular implements the architecture
- Describe the role of an Angular component
- Implement an Angular component which follows the MVC architecture

Client-side

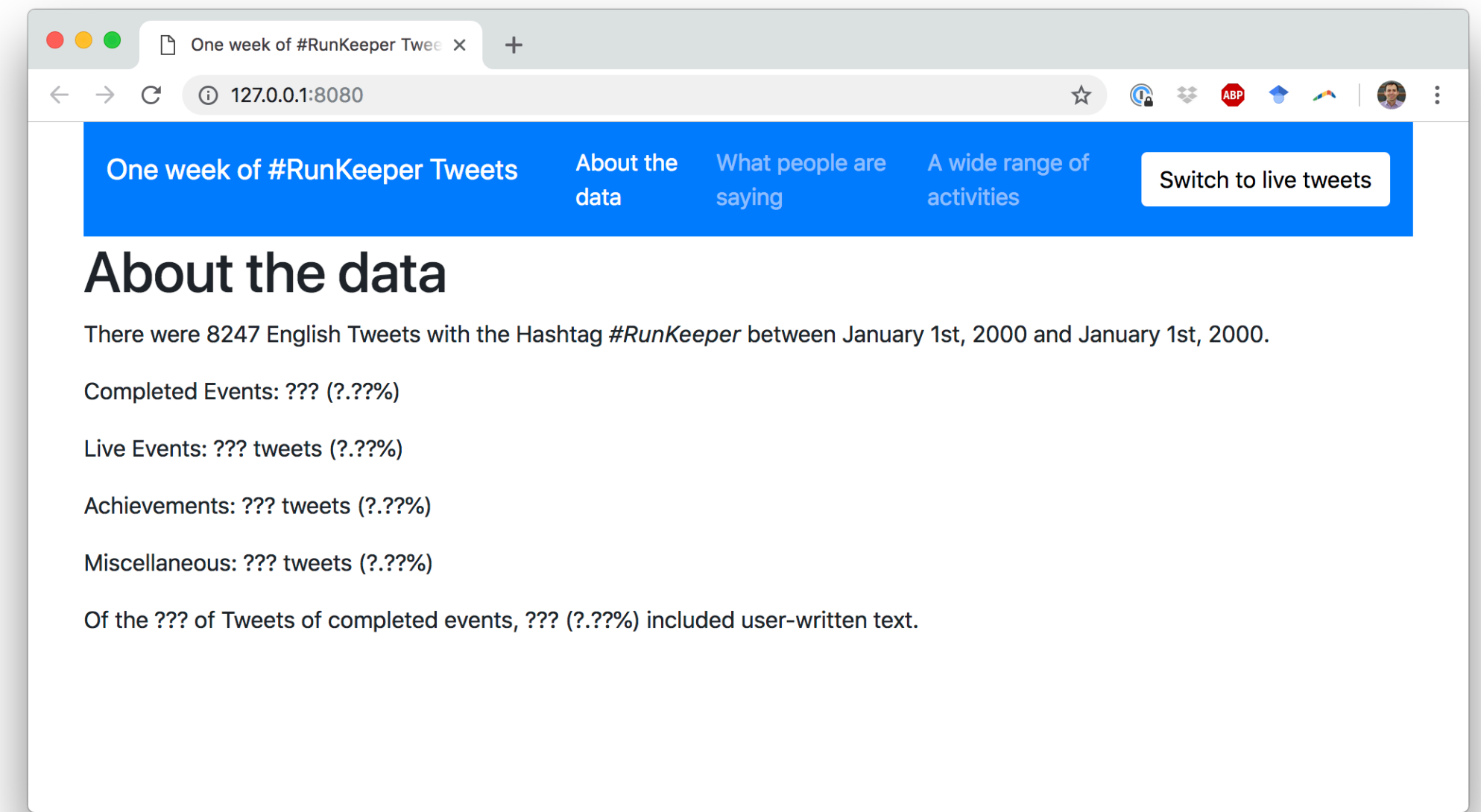
- Runs in the browser
- Changes happen in real-time in the browser
- Cannot make HTTP requests to many APIs
- Examples: AJAX, Angular, React, Vue.js

Server-side

- Runs in the command line, etc. (but maybe can still be accessed from the browser)
- Changes happen in response to HTTP requests
- Can make HTTP requests to most APIs
- Examples: Node, ASP.NET

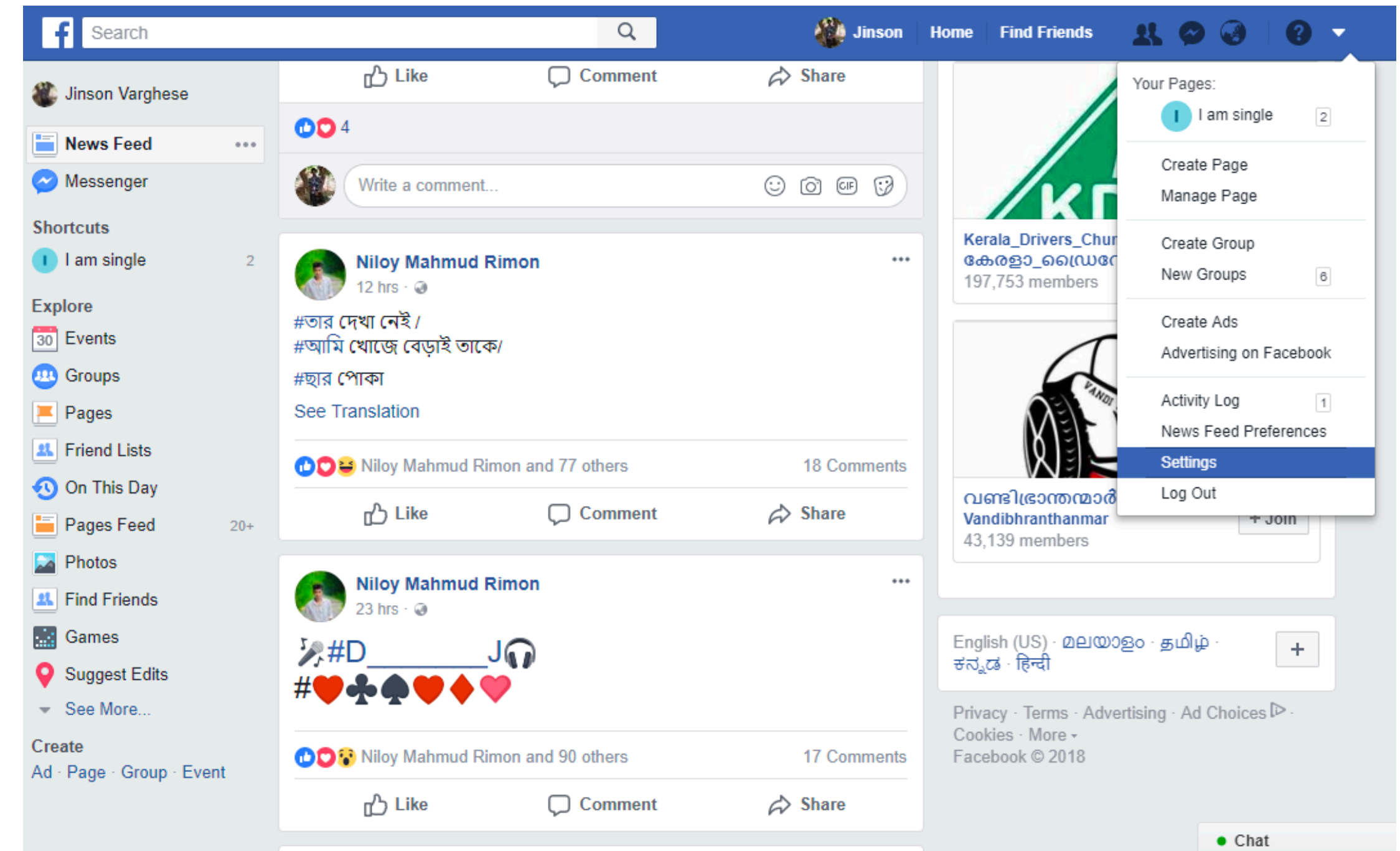
A “small” client interface

- 3 pages
- No interactivity between pages
- Data is dynamic, but UI is fairly static
- A lot of computation behind each page, but each page was very self-contained



A “large” client interface

- Hundreds of pages and ways to navigate between pages
- Repeated UI components (status updates)
- Different content, links, etc. displayed for each person



**How do we develop
large client applications?**

Frameworks for large clients

- Add structure and organization
- Make UI components reusable
- Support modularity
 - Import packages, UIs, etc. when needed

Frameworks for large clients

- Angular
- React
- Vue.js
- (Insert your favorite other framework here)
- All support the same overall goal
- All have commonalities in how they function



Angular

- First released in 2009
- Uses TypeScript, HTML, and CSS
- Does not dictate what framework is used server-side
- Major releases yearly



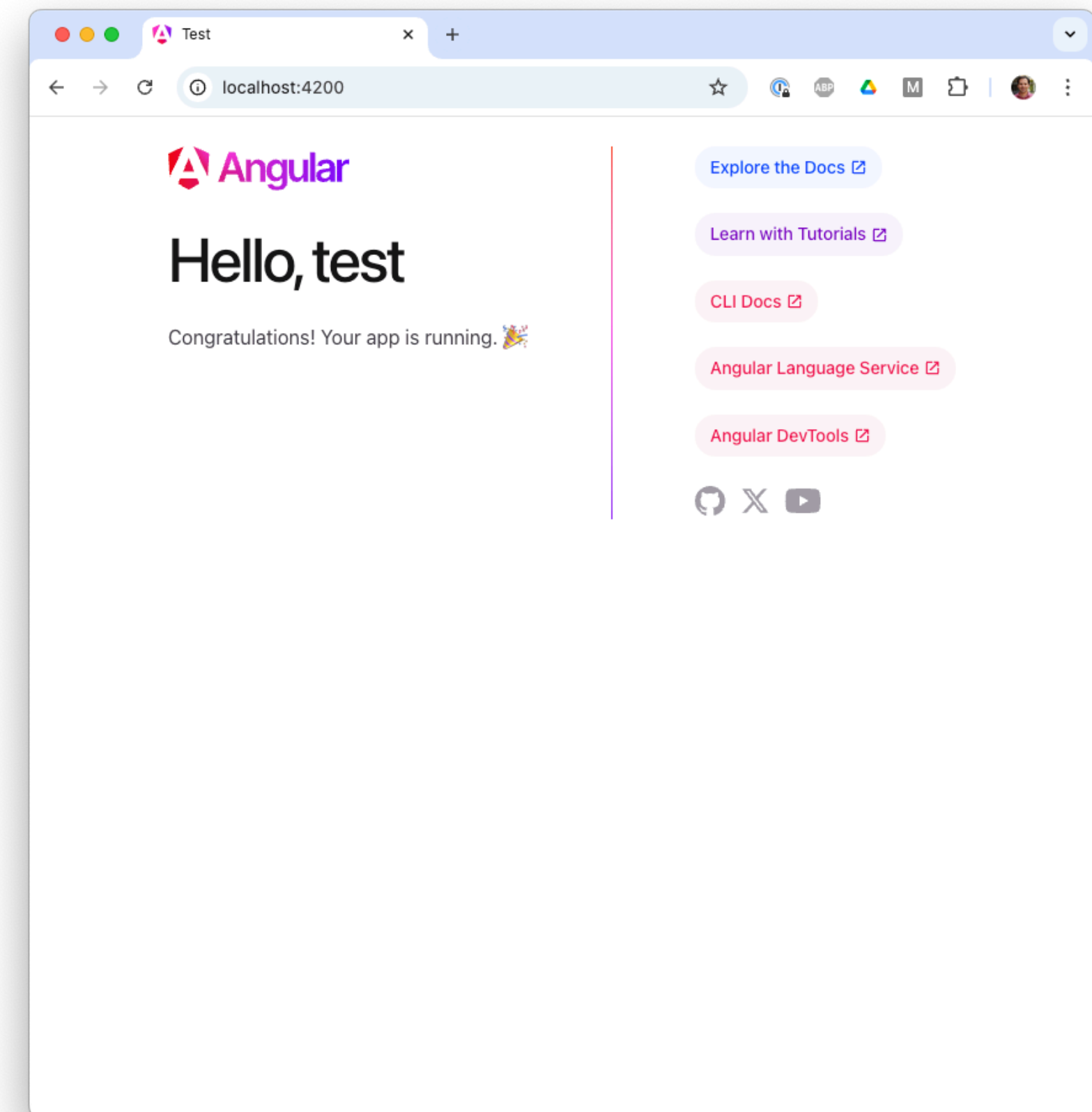
Angular != AngularJS

- Major rewrite in 2016
 - Move to TypeScript from JavaScript
 - Not backwards-compatible
- Recent syntax changes in 2024
- Makes searching StackOverflow a bit of a pain
 - But the syntax maps over reasonably okay



Angular installation

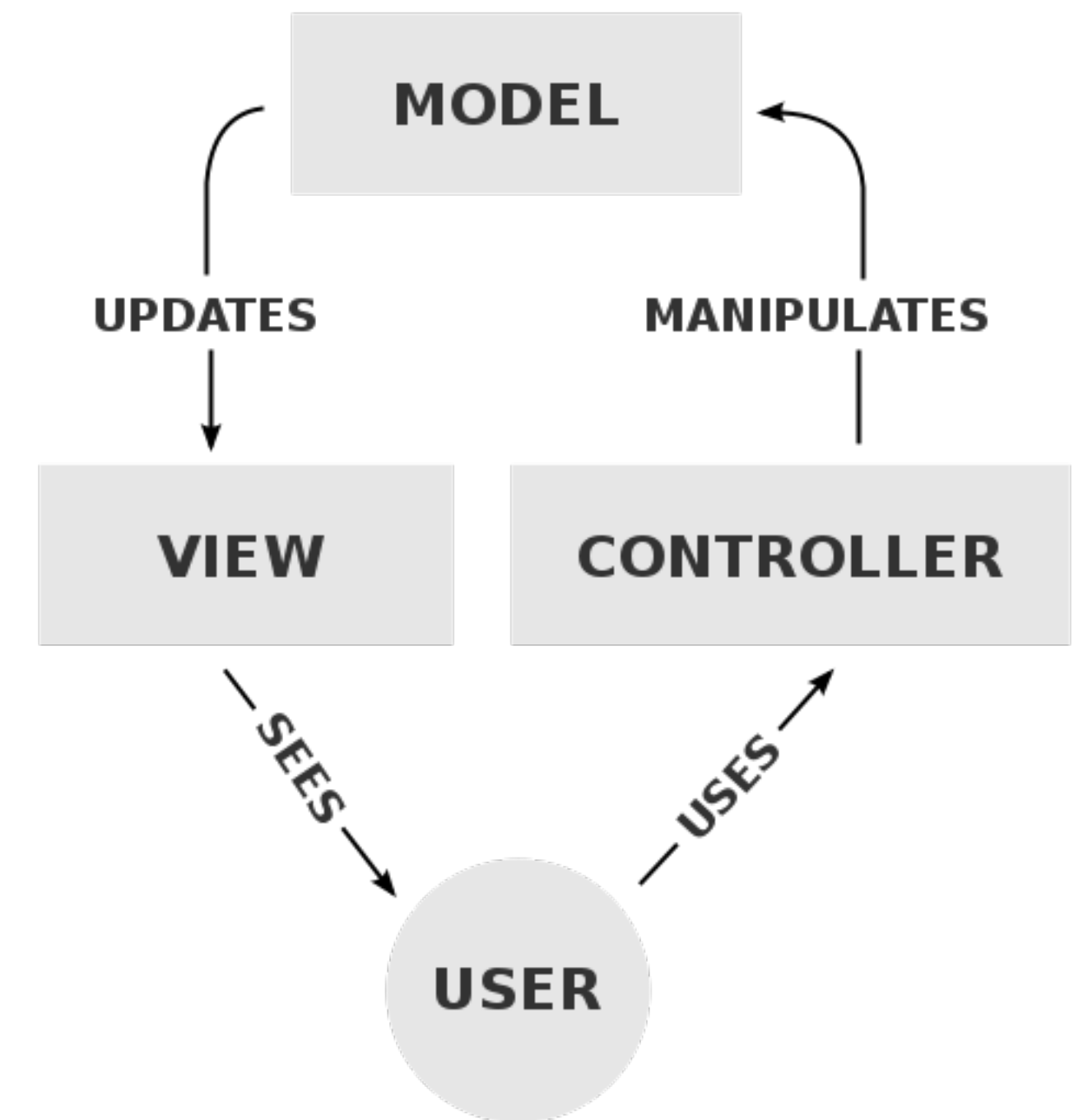
- `npm install -g @angular/cli`
- Create a new app with name `example`
 - `ng new example`
- Run app
 - `cd example`
 - `ng serve (--open)`
 - (Runs on localhost:4200 by default)



Angular architecture

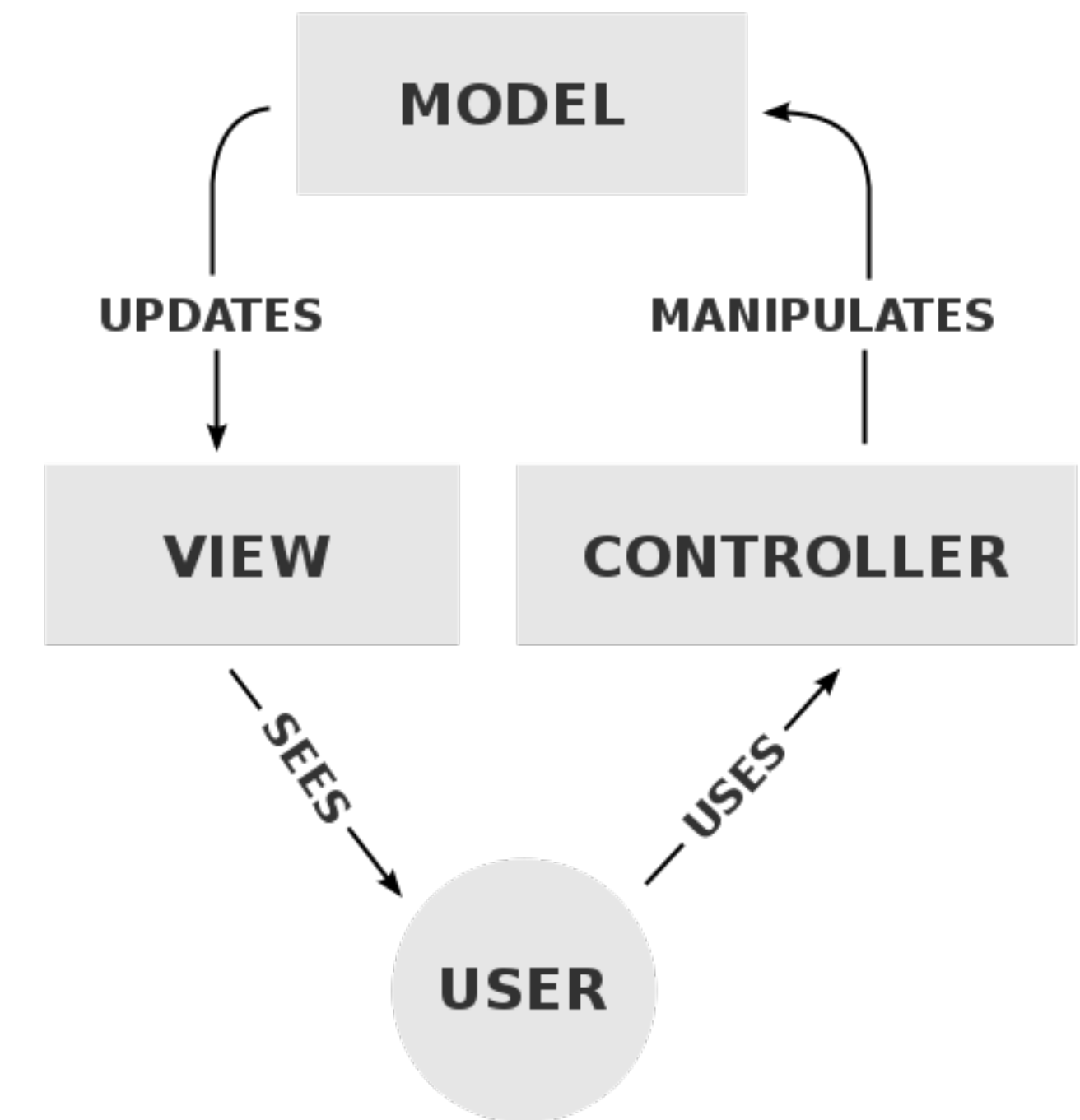
Model-View-Controller

- Approach for structuring the code behind interfaces
- Model: the data behind an app
- View: the visual interface of an app
- Controller: the interaction with an app



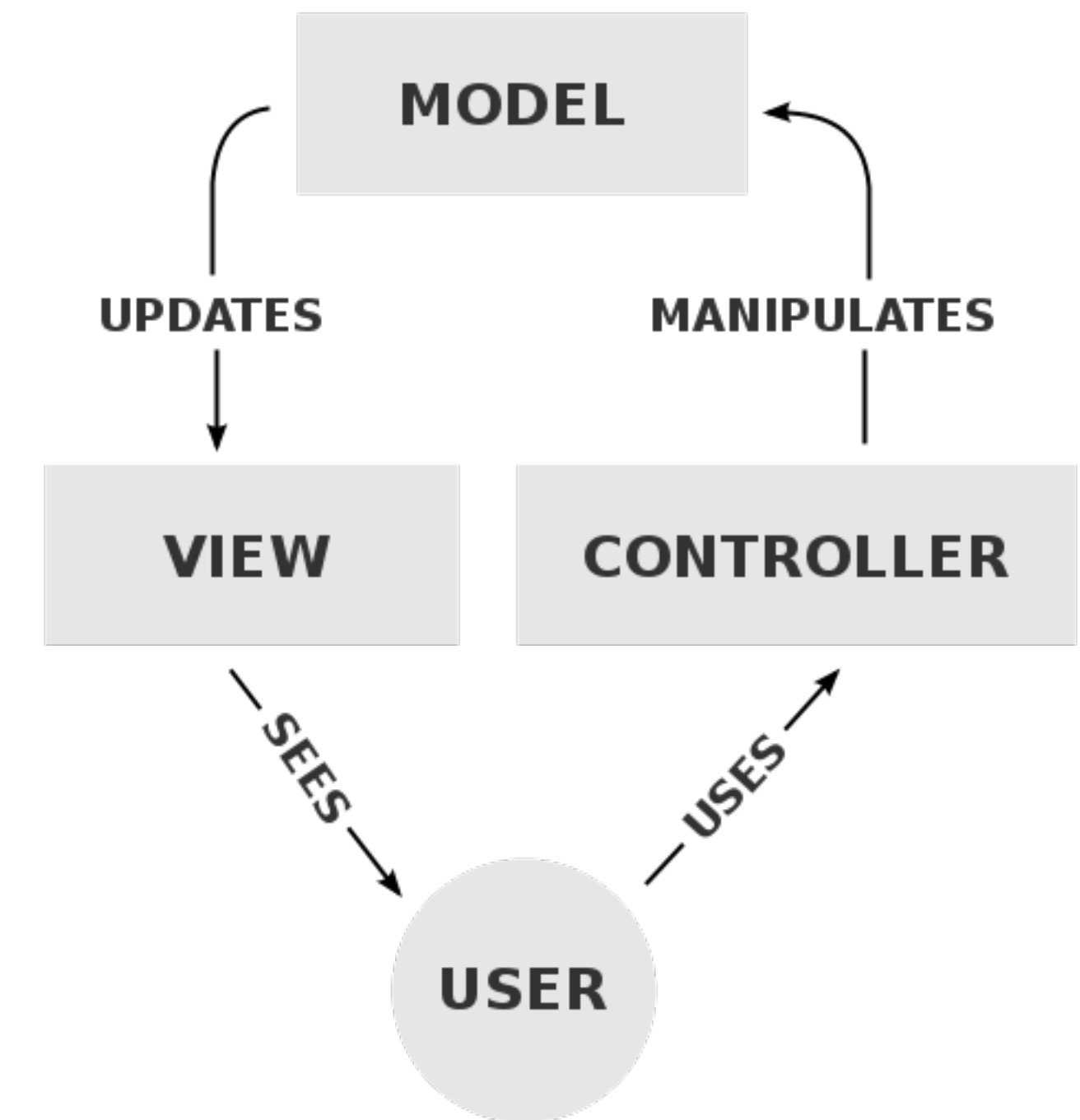
Model-View-Controller

- Model: the data behind an app
 - Notifies views when it changes
 - Enables views to query the model for data
 - Allows the controller to manipulate data in the model



Model-View-Controller

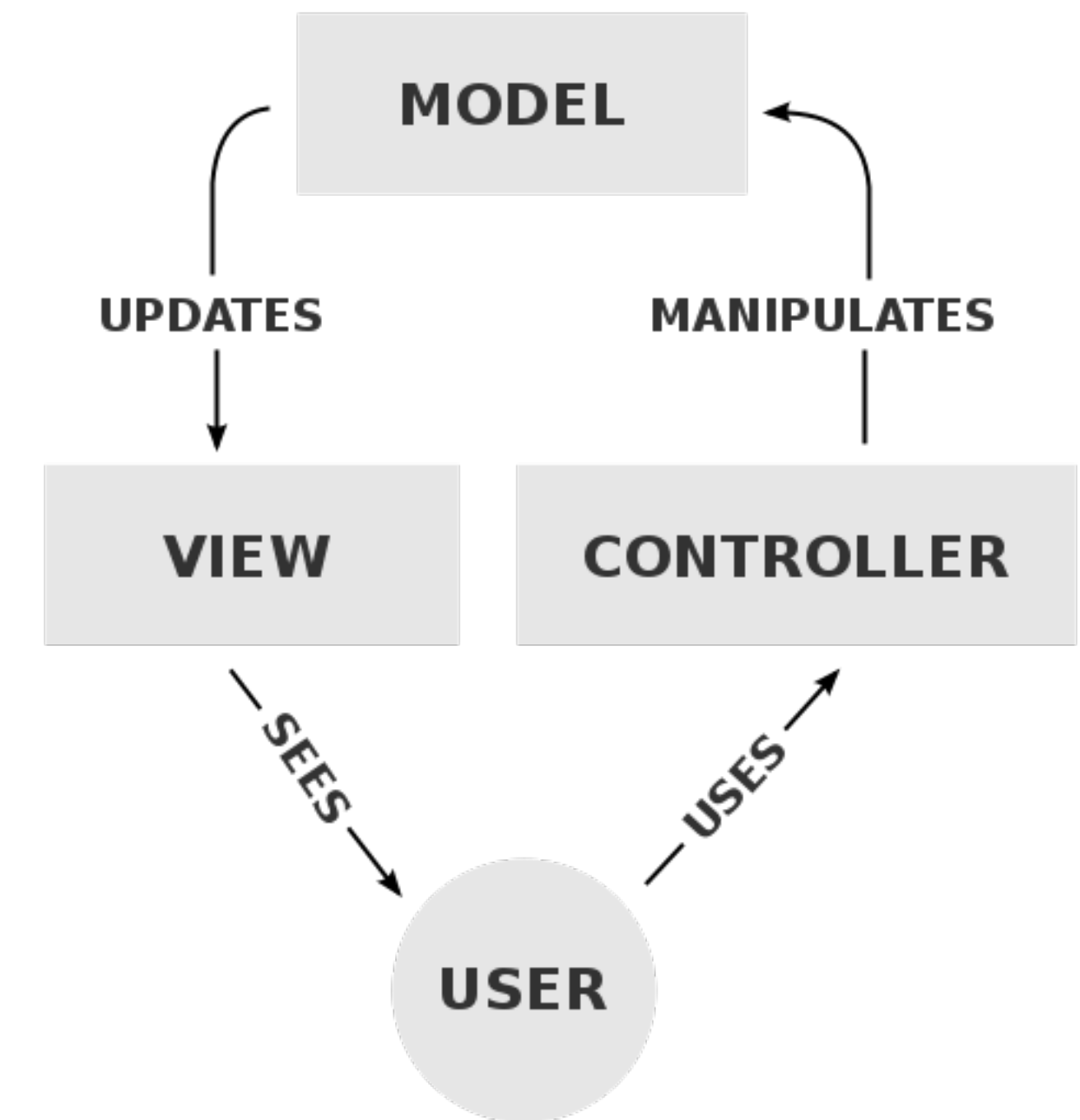
- View: the visual interface of an app
 - Renders the contents of the model
 - Specifies how the model data should be presented
 - When the model changes, the view must update its presentation
 - “Push” approach: the view waits for change notifications (live updating feed)
 - “Pull” approach: the view must ask when it wants new data (pull to refresh)
 - Forwards input to the controller



<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

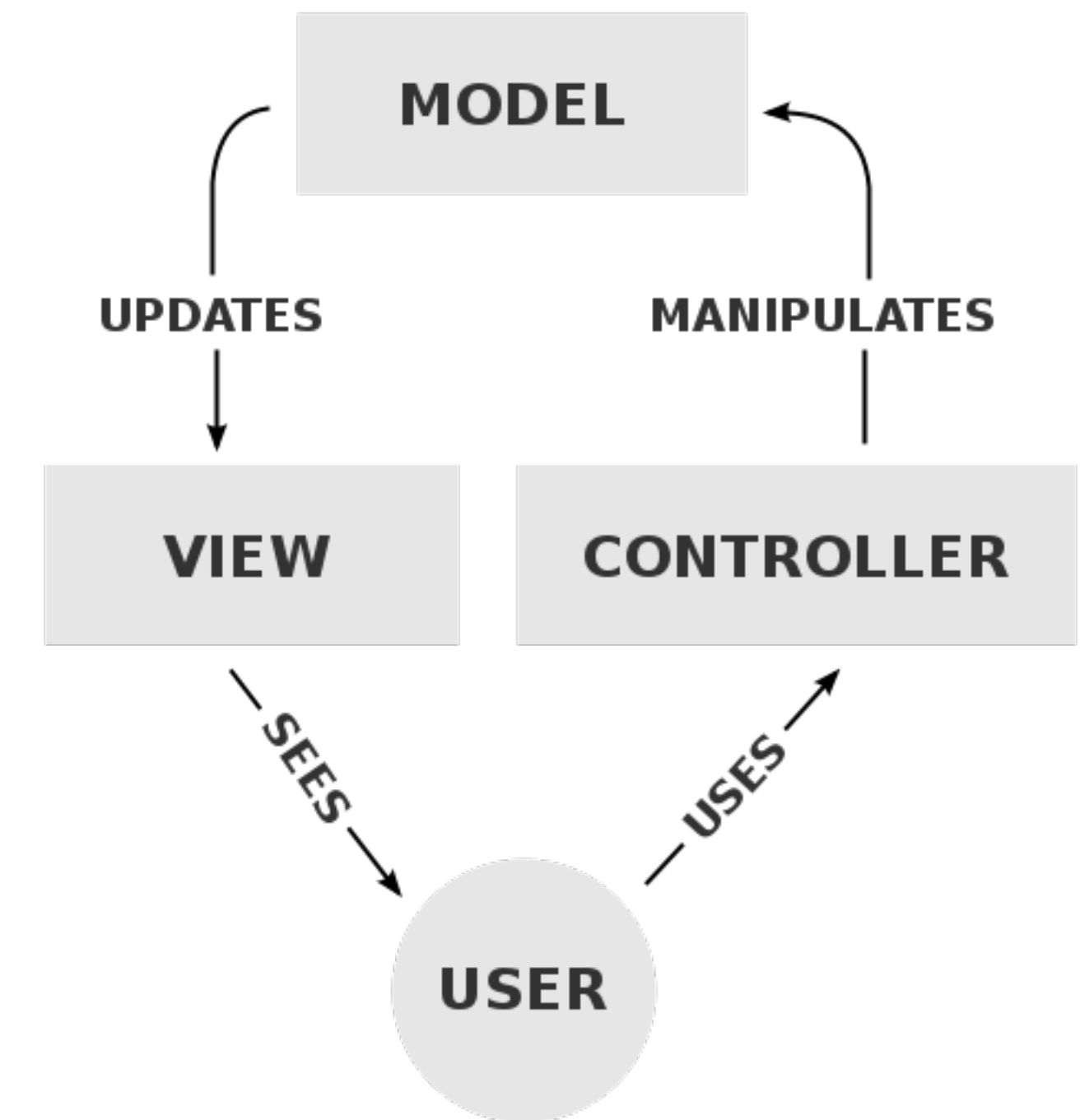
Model-View-Controller

- Controller: the interaction with an app
 - Interprets user input and maps them to actions
 - Tells the model what actions to perform
 - Indirectly tells the view (through the model) if page should be rendered differently



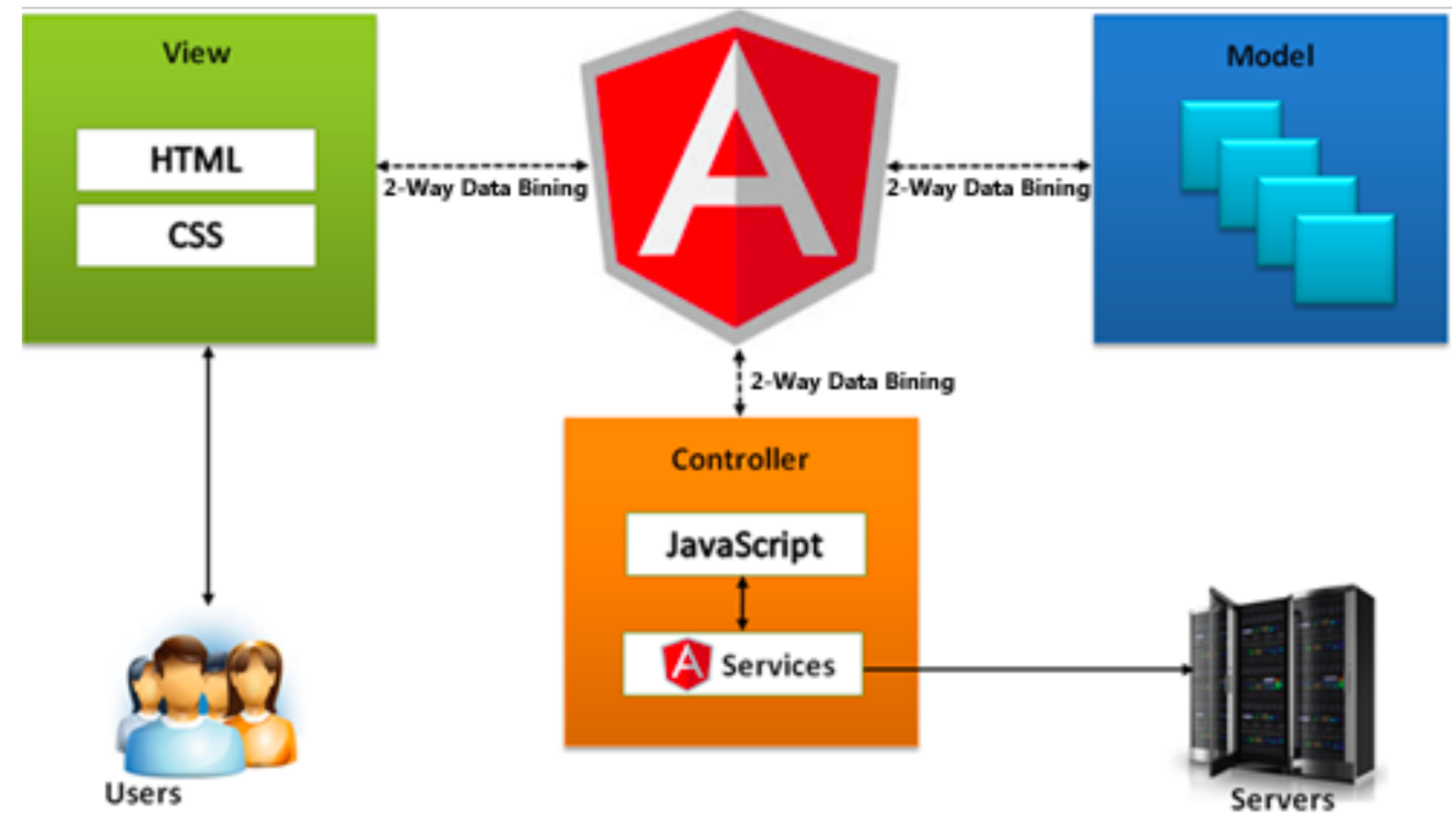
Model-View-Controller

- Model: JavaScript for loading, parsing, and manipulating data
- View: HTML and CSS to specify layout
- Controller: event handlers for buttons and inputs in JavaScript



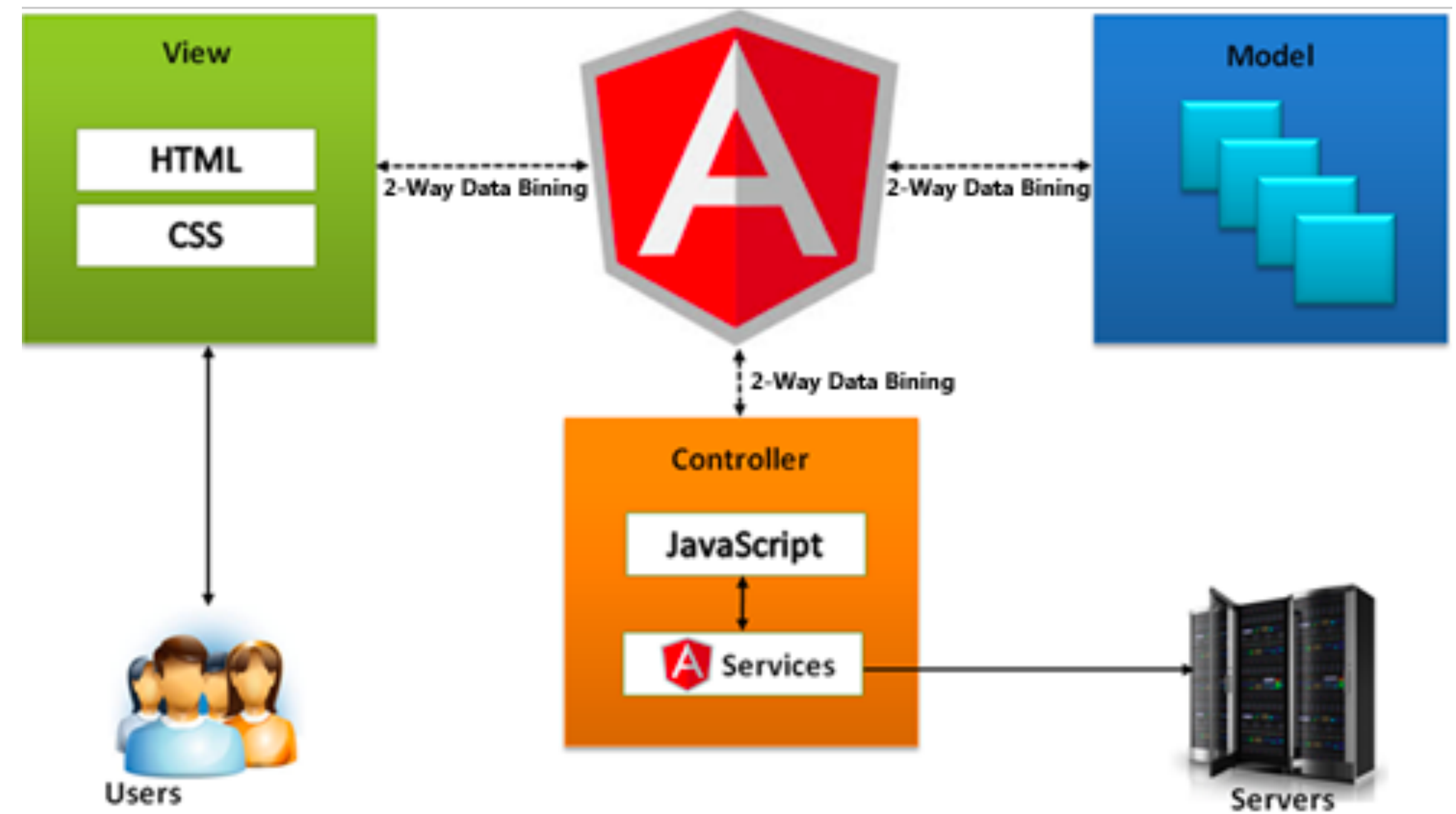
MVC in Angular

- View: HTML and CSS
- Model & Controller: TypeScript
- Angular functionality serves as the glue between the three



MVC in Angular

- **Binding:** key term
 - Variables in a view can be *bound* to variables and functions in a model or controller
 - When a variable in the *model* changes, any references to it in the *view* will also change (“push” model)
 - When a view receives input from a user, it passes it to the controller bound for that input



Following MVC in Angular

Angular components

- A component is an interface element
 - Usually larger than “a button”, but smaller than “a page”
 - Usually one which repeats across the interface

Angular components

- Component terms
 - template: the HTML file representing the view
 - style: the CSS file(s) which indicate how the component should be styled
 - selector: a CSS selector that Angular will use to all instances with this component

Angular components

- Defines the model, view, and controller for any interface element
- Make a new component: `ng generate component hello`
- Each component makes a folder consisting of four files:
 - `hello.component.css` (view)
 - `hello.component.html` (view)
 - `hello.component.spec.ts` (for automated testing; we'll mostly ignore)
 - `hello.component.ts` (model and controller)

Angular components

- Every app has at least one component
- “Root” component
 - By default, in `app.component.ts` (`html/css/ts/spec.ts`)
 - The “main” page in your app, essentially

Root template

app.component.html

```
<div>
  <h1>
    Welcome to {{ title }}!
  </h1>
  
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://github.com/angular/angular-cli/wiki">CLI
Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>

<router-outlet></router-outlet>
```

Root template

`app.component.html`

- Looks like any other HTML page
- Only difference: page can support data binding

Root template

app.component.html

```
<div>
  <h1>
    Welcome to {{ title }}! ← Binds text to the variable title
  </h1>
  
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://github.com/angular/angular-cli/wiki">CLI
Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>

<router-outlet></router-outlet>
```

Root component

app.component.ts

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
)  
export class AppComponent {  
  title = 'example';  
}
```

Import Angular component library

Designates class as a component

Replace all occurrences of `app-root` with this

Designate template

Designate style(s)

All code for the component

Root component and template

app.component.html

```
<div>
  <h1>
    Welcome to {{ title }}!
  </h1>
  
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://github.com/angular/angular-cli/wiki">CLI Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>

<router-outlet></router-outlet>
```

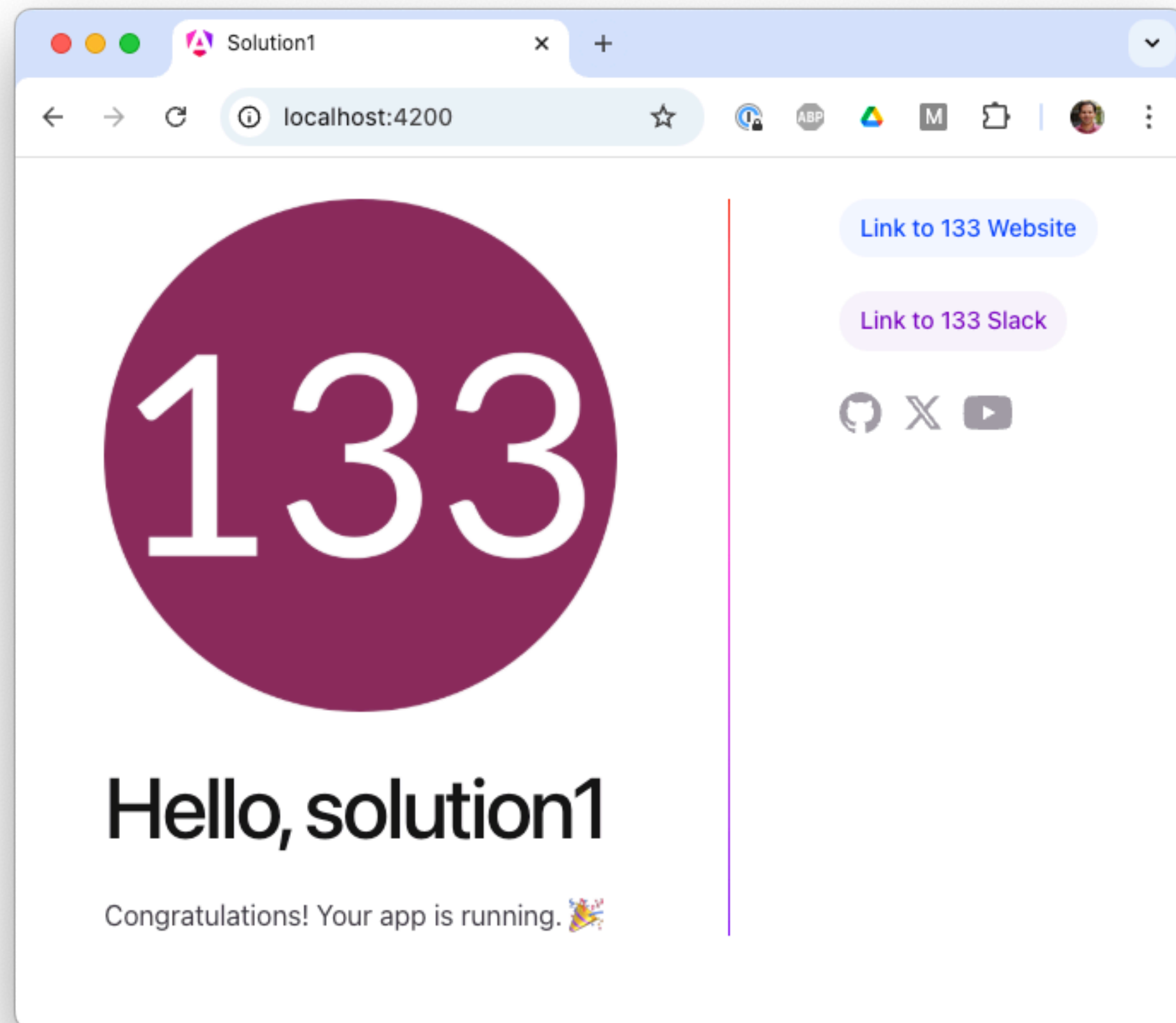
Bound to

app.component.ts

```
import { Component } from
 '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './
app.component.html',
  styleUrls: ['./
app.component.css']
})
export class AppComponent {
  title = 'example';
}
```


Hello Angular



Four types of binding

- Interpolation: {{ }}
- Property: []
- Event: ()
- Two-way: [()]

Interpolation binding {{ }}

- “Weave calculated strings into the text between HTML element tags and within attribute assignments”

```
<h3>
  {{title}}
  
</h3>
```

- Can also be used to calculate values

```
<!-- "The sum of 1 + 1 is 2" -->
<p>The sum of 1 + 1 is {{1 + 1}}</p>
```

Property binding []

- “Set an element property to a component property value”

```
<img [src]="heroImageUrl">
```

Event binding ()

- “Listen for certain events such as keystrokes, mouse movements, clicks, and touches”

`<!--When clicked, will run the onSave() function in component.ts file-->`

`<button (click)="onSave()">Save</button>`

One-way binding

- Interpolation, property, and event are all one-way, or *read-only* binding
- For interpolation `{{ }}` and property `[]`, binding goes from data source (.ts) to view target (.html)
- For event `()`, binding goes from view target (.html) to data source (.ts)

Data direction	Syntax
One-way from data source to view target	<pre>{{expression}} [target]="expression" bind-target="expression"</pre>
One-way from view target to data source	<pre>(target)="statement" on-target="statement"</pre>
Two-way	<pre>[(target)]="expression" bindon-target="expression"</pre>

One-way binding

{{title}}

Bound to

Bound to

<button (click)="onSave()">Save</button>

Bound to

```
import { Component, OnInit } from
 '@angular/core';

@Component({
  selector: 'app-hello',
  templateUrl: './hello.component.html',
  styleUrls: ['./hello.component.css']
})
export class HelloComponent implements
 OnInit {
  title = 'example';
  heroImageUrl = 'hero.jpg';

  onSave() {
    console.log('File saved!');
  }
}
```

Question

Which will bind the link's href to the link is the variable `url`?

☐ A `link`

☐ B `<a (href)="url">link`

☐ C `<a [href]="url">link`

☐ D A and B

☐ E A and C

Which will bind the link's href to the link is the variable url?

A `link`

B `<a (href)="url">link`

C `<a [href]="url">link`

D A and B

E A and C

A

0%

B

0%

C

0%

D

0%

E

0%

Question

Which will bind the link's href to the link is the variable `url`?

☐ A `link`

☐ B `<a (href)="url">link`

☐ C `<a [href]="url">link`

☐ D A and B

☒ E A and C

Two-way binding [()]

- “You often want to both display a data property and update that property when the user makes changes”
 - Most common use: binding to user-generated input
 - ngModel directive enables two-way binding to input fields
- ```
<!--enteredText variable contains inputted text-->
<!--textChanged() is called after every keystroke-->
<input [(ngModel)]="enteredText" (change)="textChanged()">
```

# Binding

```
<!--enteredText variable contains inputted text-->
<!--textChanged() is called after every keystroke-->
<input [(ngModel)]="enteredText" (change)="textChanged()" >

<!--When clicked, will run the onSave() function in component.ts
file-->
<button (click)="onSave()" >Save</button>

<h3>
 <!--will display the title-->
 {{title}}
 <!--will display the image at heroImageUrl-->

</h3>
```

# Control flows

- @if and @for, with parentheses to identify what should be covered
- Switched from \*ngIf and \*ngFor in 2024, many online examples will use the old syntax

# @if

- Render a tag if condition is true

```
@if(isHalloween) {
 <p>
 Spooky!
 </p>
} @else {
 <p>
 Just another boring day in the neighborhood.
 </p>
}
```

# @for

- Repeat an item multiple times

```

 @for(day of days; track day) {

 {{day}}

 }

```

- “Track” is for performance, should be a unique property

- Can optionally specify index

```

 @for(day of days; track day; let i = $index) {

 {{i+1}}: {{day}}

 }

```

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday

- 1: Sunday
- 2: Monday
- 3: Tuesday
- 4: Wednesday
- 5: Thursday
- 6: Friday
- 7: Saturday

```
//In component.ts
days =
["Sunday",
 "Monday",
 "Tuesday",
 "Wednesday",
 "Thursday",
 "Friday",
 "Saturday"];
```

<https://angular.dev/api/core/@for>

# Using components

- Components can import other components
  - Follow the selector defined in the component's `.ts` file
- In `app.component.html`:

```
<div>
 <h1>
 Welcome to {{ title }}!
 </h1>
 <app-day></app-day>
</div>

<router-outlet></router-outlet>
```

## Welcome to example!

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday



# Using components

- Components can specify inputs

```
import { Component, OnInit, Input }
from '@angular/core';

@Component({
 selector: 'app-day',
 templateUrl: './day.component.html',
 styleUrls: ['./day.component.css']
})
export class DayComponent {
 @Input() today:string; ← Input

 days = ["Sunday", "Monday",
 "Tuesday", "Wednesday", "Thursday",
 "Friday", "Saturday"];
}
```

```

 @for(let day of days){

 {{day}}
 @if(day == today) {
 Today!
 }

 }

```

Input referenced in .html

# Using components

- Inputs are then passed:
  - As properties if they're dynamic
  - Like any other attribute if they're static

```
<div>
 <h1>
 Welcome to {{ title }}!
 </h1>
 <app-day [today]="dayOfWeek"></app-day>
</div>
```



Sets day property  
to dayOfWeek variable

```
<div>
 <h1>
 Welcome to {{ title }}!
 </h1>
 <app-day today="Friday"></app-day>
</div>
```



Sets day property  
static value Friday

# Using components

- Can also specify output properties

```
@Output('myClick') clicks = new EventEmitter<string>();
```

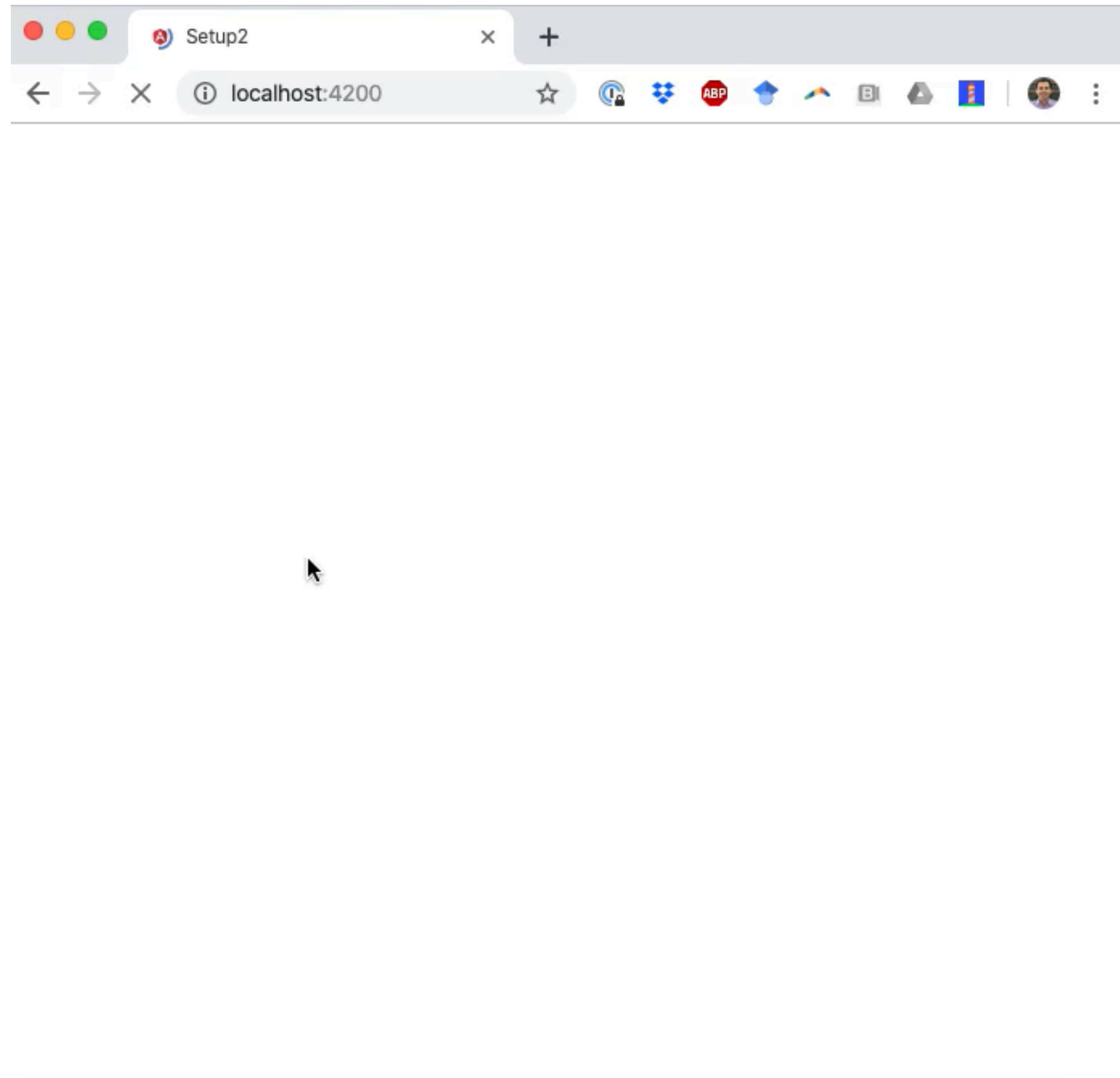
- When adding component, can specify an event to trigger when `clicks()` is called

```
<app-button (myClick)="clickMessage">click with myClick</app-button>
```

- The event will be triggered in the parent component

```
clickMessage() {
 console.log("clicked!");
}
```

# Rolling Dice



# Summary

- Angular is made up of components, which are UI elements which each follow a Model-View-Controller framework
  - `.html` and `.css` file define the view
  - `.ts` defines the model and controller
- Data moves between model and view through binding
  - Components can pass data by binding `Inputs` and `Outputs`
- Components can include other components
  - `@if` and `@for` help determine whether and how many components to create

# Today's goals

By the end of today, you should be able to...

- Explain a Model-View-Controller Architecture and how Angular implements the architecture
- Describe the role of an Angular component
- Implement an Angular component which follows the MVC architecture

# IN4MATX 133: User Interface Software

## Lecture 10: Components in Angular