IN4MATX 133: User Interface Software

Lecture:
Components in Angular

Goals for this lecture

By the end of this lecture, you should be able to…

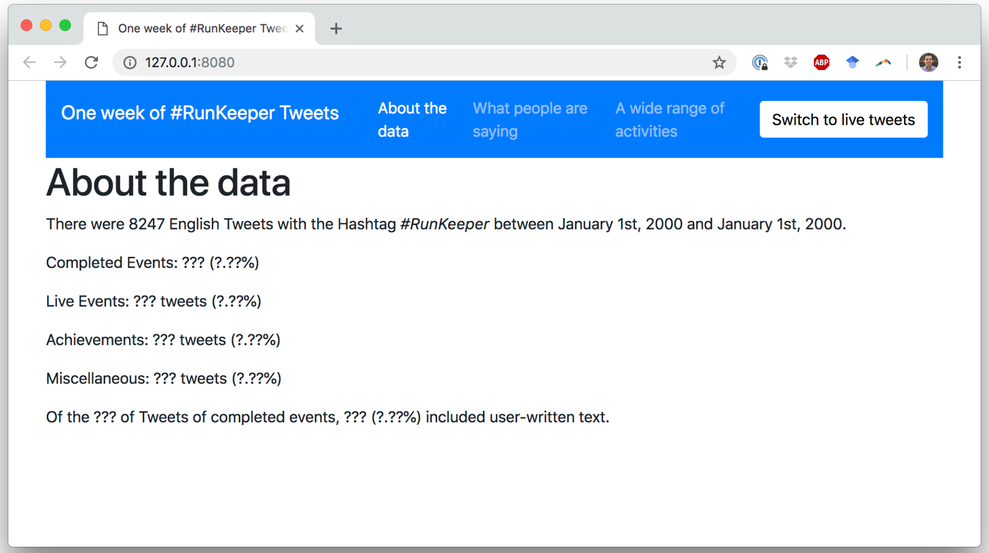
- Explain a Model-View-Controller Architecture and how Angular implements the architecture
- Describe the role of an Angular component

On Friday we'll cover:

- OAuth 2.0
- Implement an Angular component which follows the MVC architecture

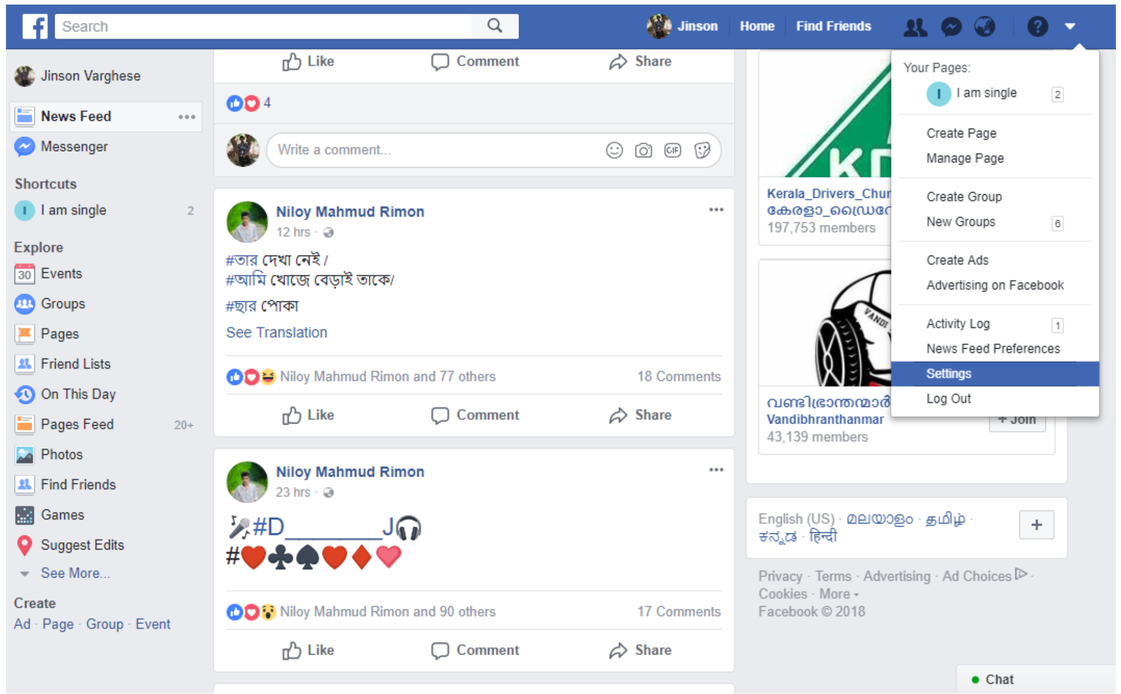
A “small” client interface

- 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

- All support the same overall goal

Angular

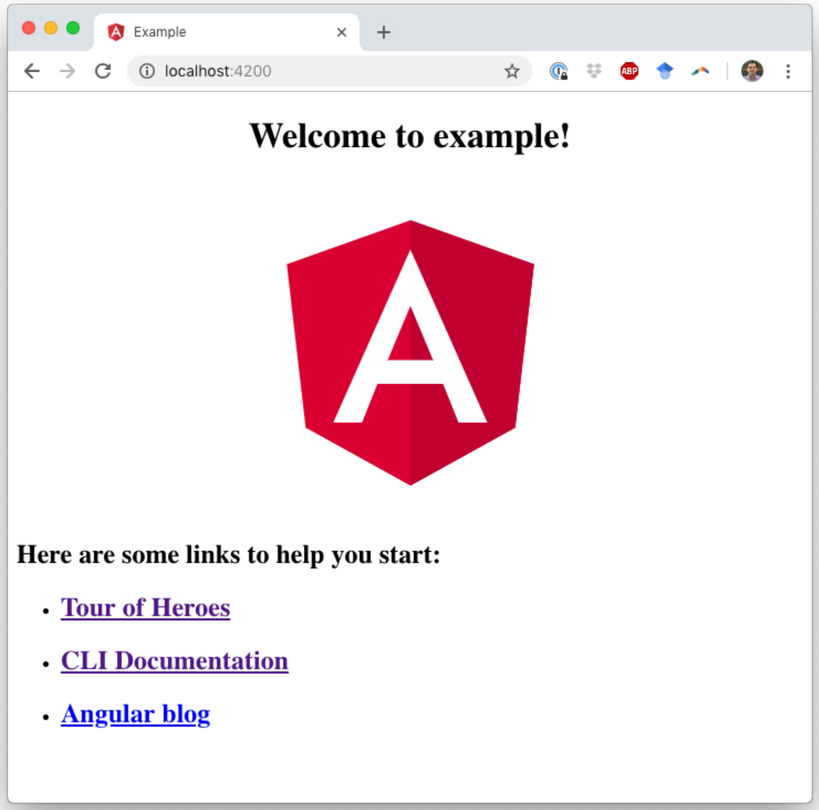
- First released in 2009
- Uses TypeScript, HTML, and CSS
- Does not dictate what framework is used server-side
- Last major release was version 10 in June 2020

Angular != AngularJS

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

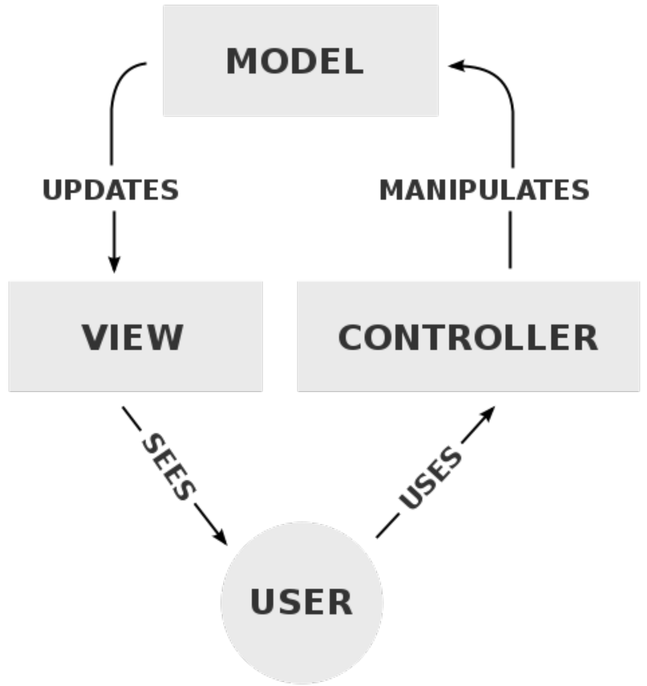
Angular installation

- `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)



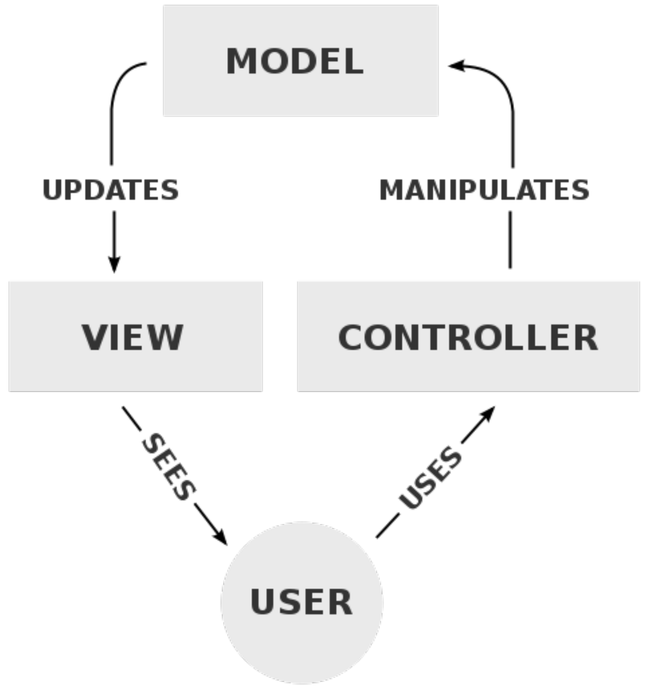
Angular architecture

- 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



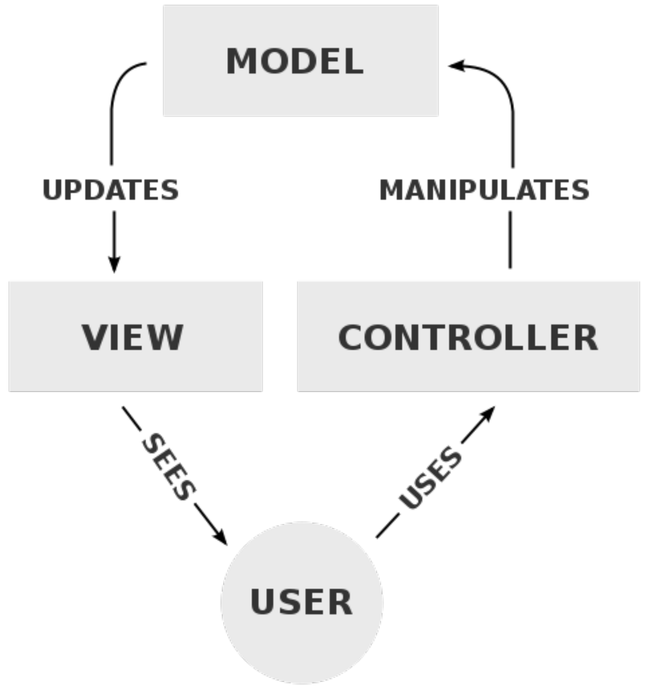
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

- 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



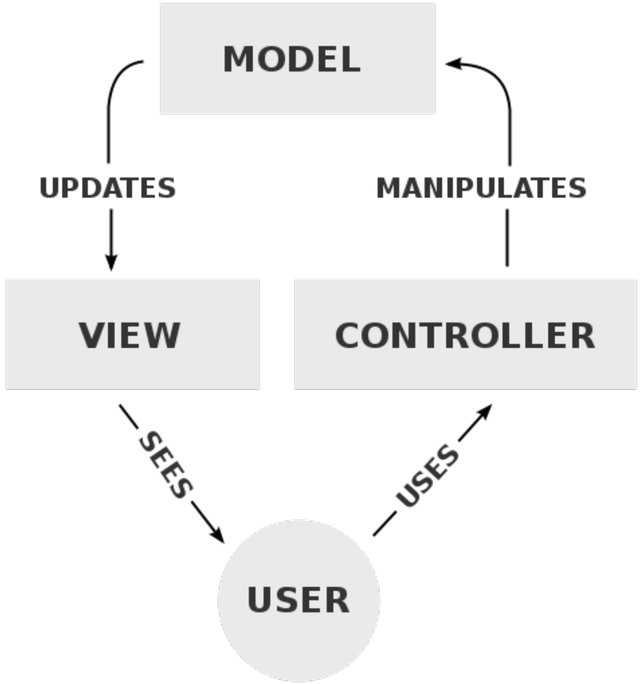
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

- 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 it’s 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



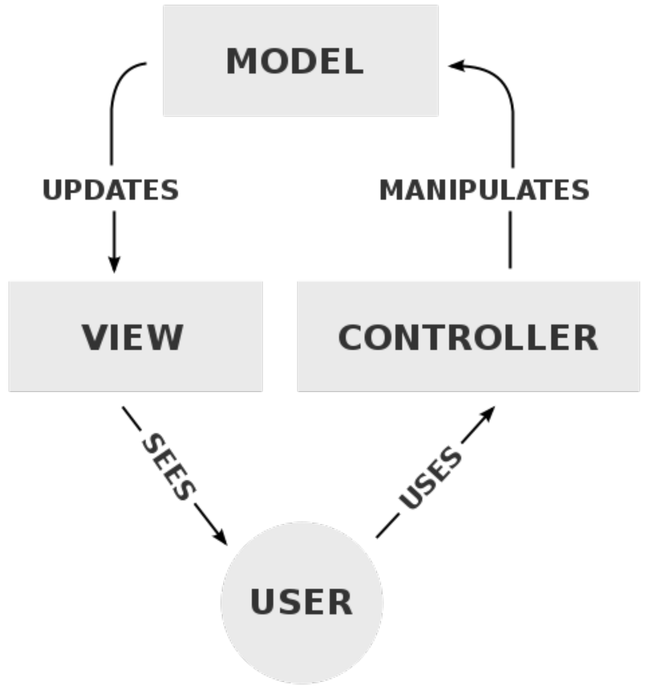
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

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



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

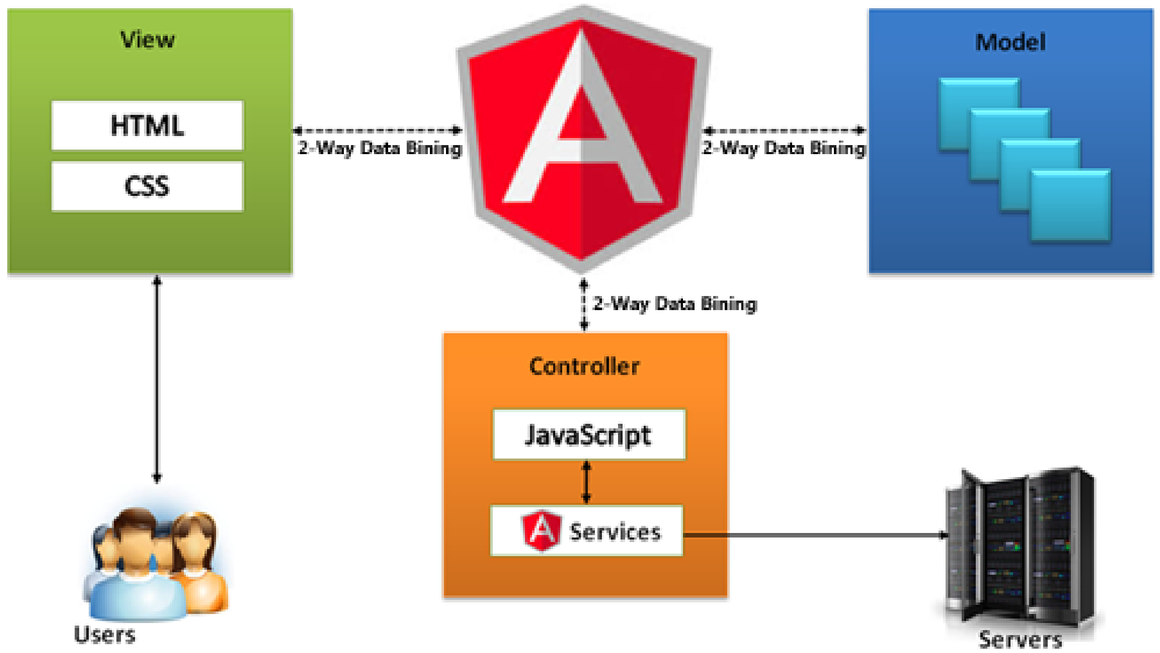
- Model: JavaScript for loading, parsing, and manipulating data
- View: HTML and CSS to specify layout
- Controller: event handlers for buttons and inputs in JQuery



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

MVC in Angular

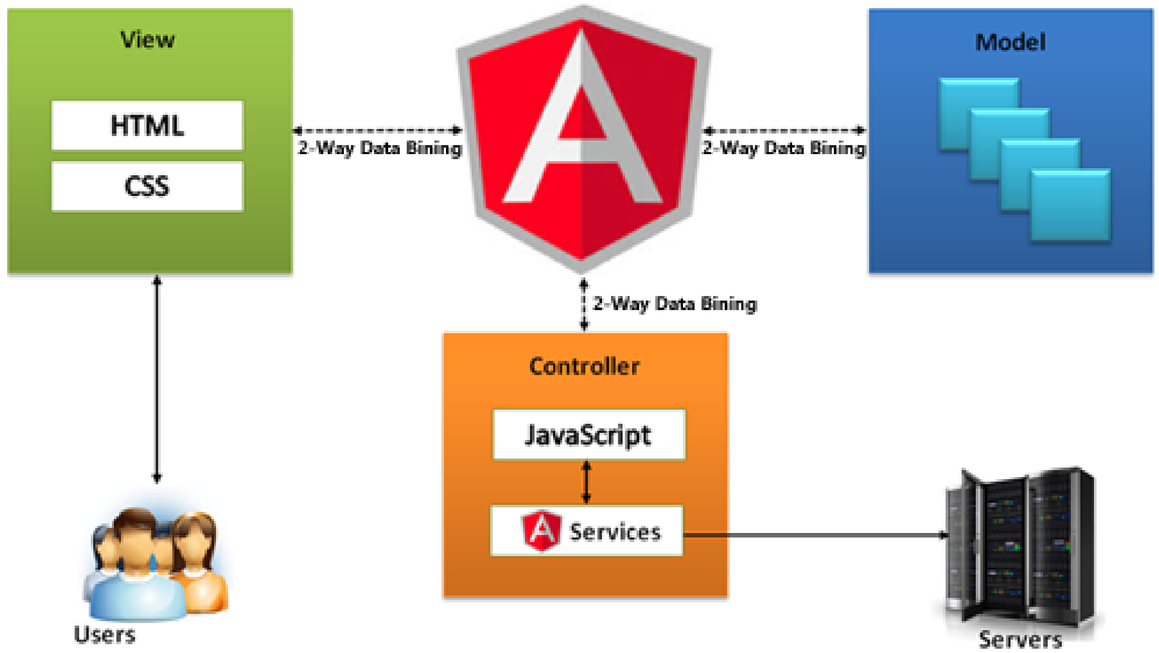
- View: HTML and CS
- Model & Controller: TypeScript
- Angular functionality serves as the glue between the three



https://codeburst.io/angularjs-4-101-6675076784aa

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 controller bound for that input



https://codeburst.io/angularjs-4-101-6675076784aa

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.(html/css/ts/spec.ts)`
  - The “main” page in your app, essentially

Root template

app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/...">
</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>
```

app.component.html

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

Root template

app.component.html

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/...">
</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>
```

Binds text to the variable `title`

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 style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/...">
</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';
}
```

Four types of binding

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

https://angular.io/guide/template-syntax

Interpolation binding {{ }}

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

```
<h3>
   {{title}}
   <img src="{{heroImageUrl}}" style="height:30px">
</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>
```

https://angular.io/guide/template-syntax

Property binding [ ]

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

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

https://angular.io/guide/template-syntax

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>
```

https://angular.io/guide/template-syntax

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)

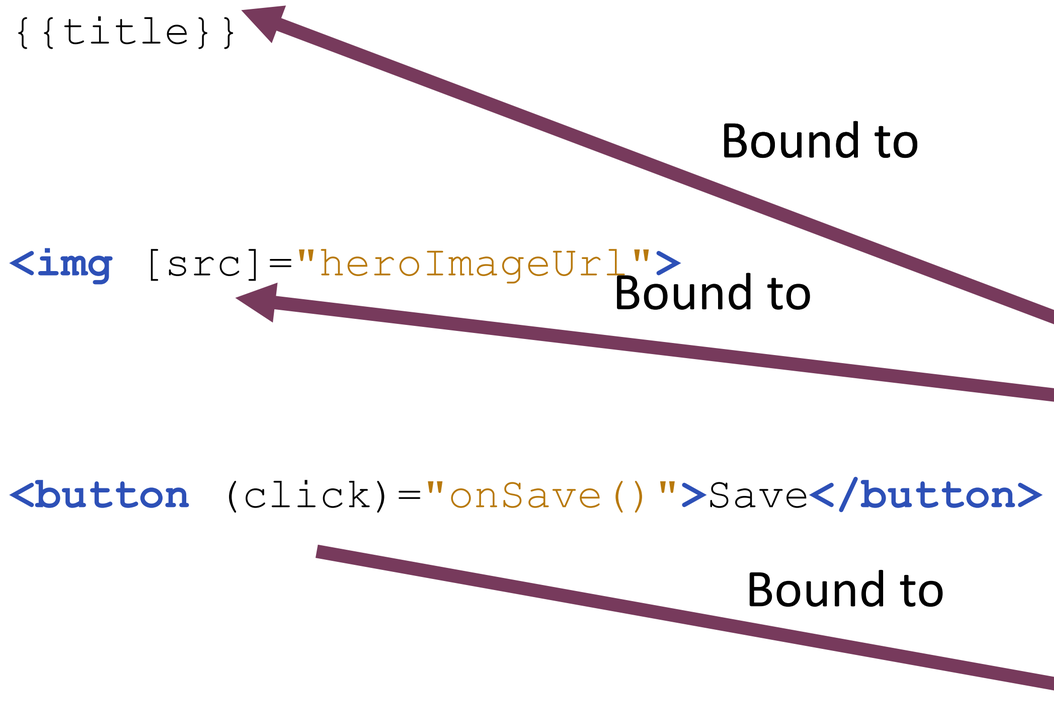
Data direction	Syntax

One-way
from data source
to view target
	`{{expression}}`
	`[target]="expression"`
	`bind-target="expression"`

One-way from view target to data source	`(target)="statement"` `on-target="statement"`
Two-way	`[(target)]="expression"` `bindon-target="expression"`

https://angular.io/guide/template-syntax

One-way binding




```
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';

  constructor() { }

  ngOnInit() {
  }

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

}
```

https://angular.io/guide/template-syntax

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()">
```

https://angular.io/guide/template-syntax

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-->
  <img [src]="heroImageUrl">
</h3>
```

Socrative Quiz!

New Link Today

Enter your UCI Email when prompted for name!!!
e.g.,

xxxxx@uci.edu

https://api.socrative.com/rc/52QwBu

Directives

*ngIf

- Render a tag if condition is true

```
<p *ngIf="isHalloween">
  Spooky!
</p>
```

- Can use ternary operator

```
<div [style.display]="isSpecial ? 'block' : 'none'">Show
with style</div>
```

Directives

*ngFor

- Repeat an item multiple times

```
<ul>
  <li *ngFor="let day of days">
    {{day}}
  </li>
</ul>
```

- Can optionally specify index

```
<ul>
  <li *ngFor="let day of days; let i=index">
    {{i+1}}: {{day}}
  </li>
</ul>
```

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

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

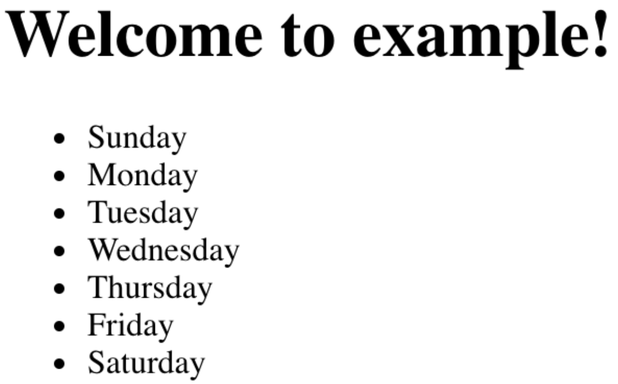
```
//In component.ts
  days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
```

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>
```



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;

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

  constructor() { }
}
```

Input

```
<ul>
  <li *ngFor="let day of days">
    {{day}}
    <strong *ngIf="day == today">Today!</strong>
  </li>
</ul>
```

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!");
}
```

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
  - `*ngIf` and `*ngFor` help determine whether and how many components to create

Goals for this lecture

By the end of this lecture, you should be able to…

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