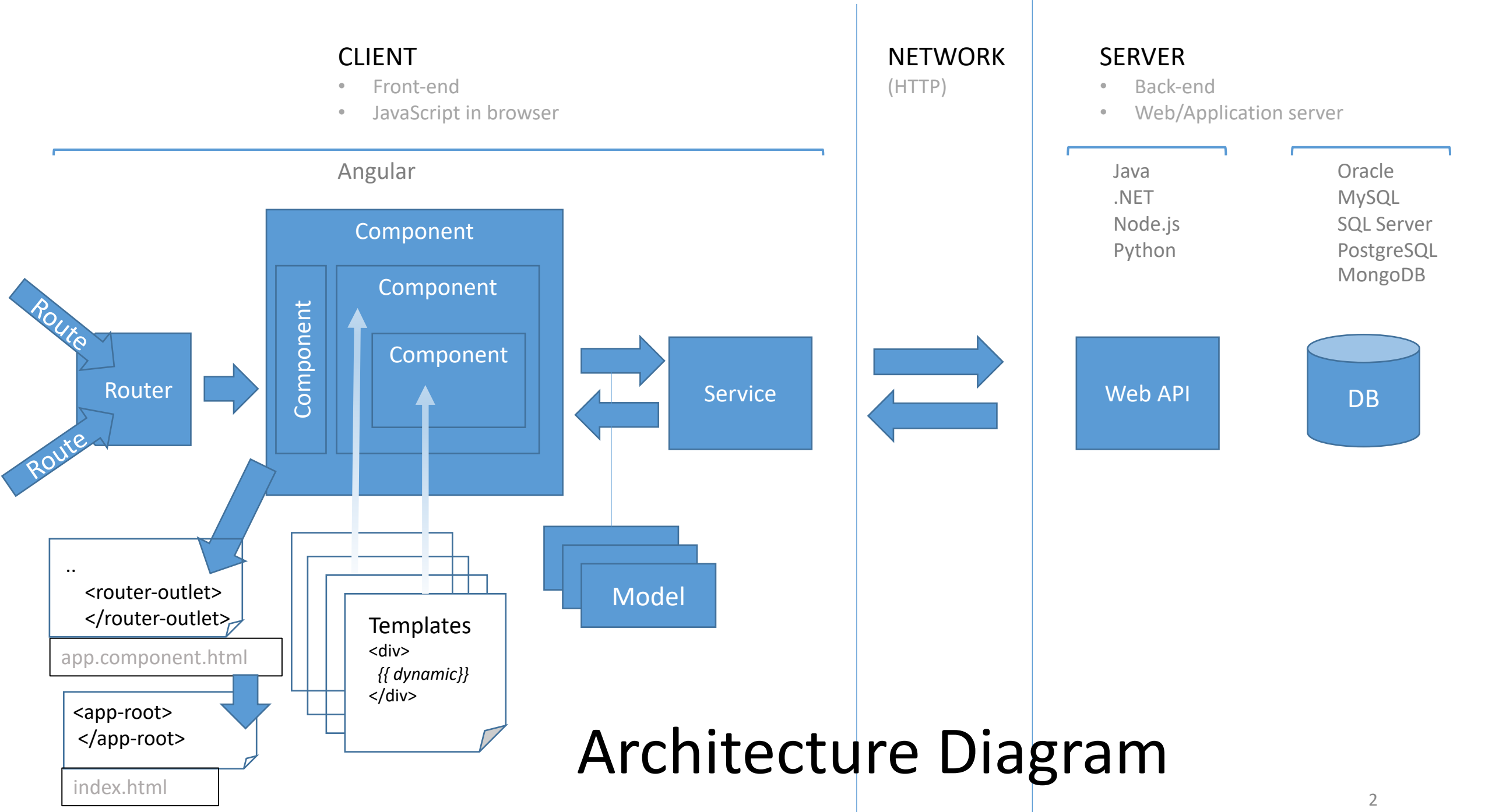
Angular

Copyright 2018-2020 Funny Ant LLC


CLIENT
Front-end
JavaScript in browser
NETWORK
(HTTP)
SERVER
Back-end
Web/Application server
Angular
Java
.NET
Node.js
Python
Oracle
MySQL
SQL Server
PostgreSQL
MongoDB
Route
Route
Router
Component
Component
Component
Component
Service
Web API
DB
Model
..
<router-outlet>
</router-outlet>
app.component.html
<app-root>
</app-root>
index.html
Templates

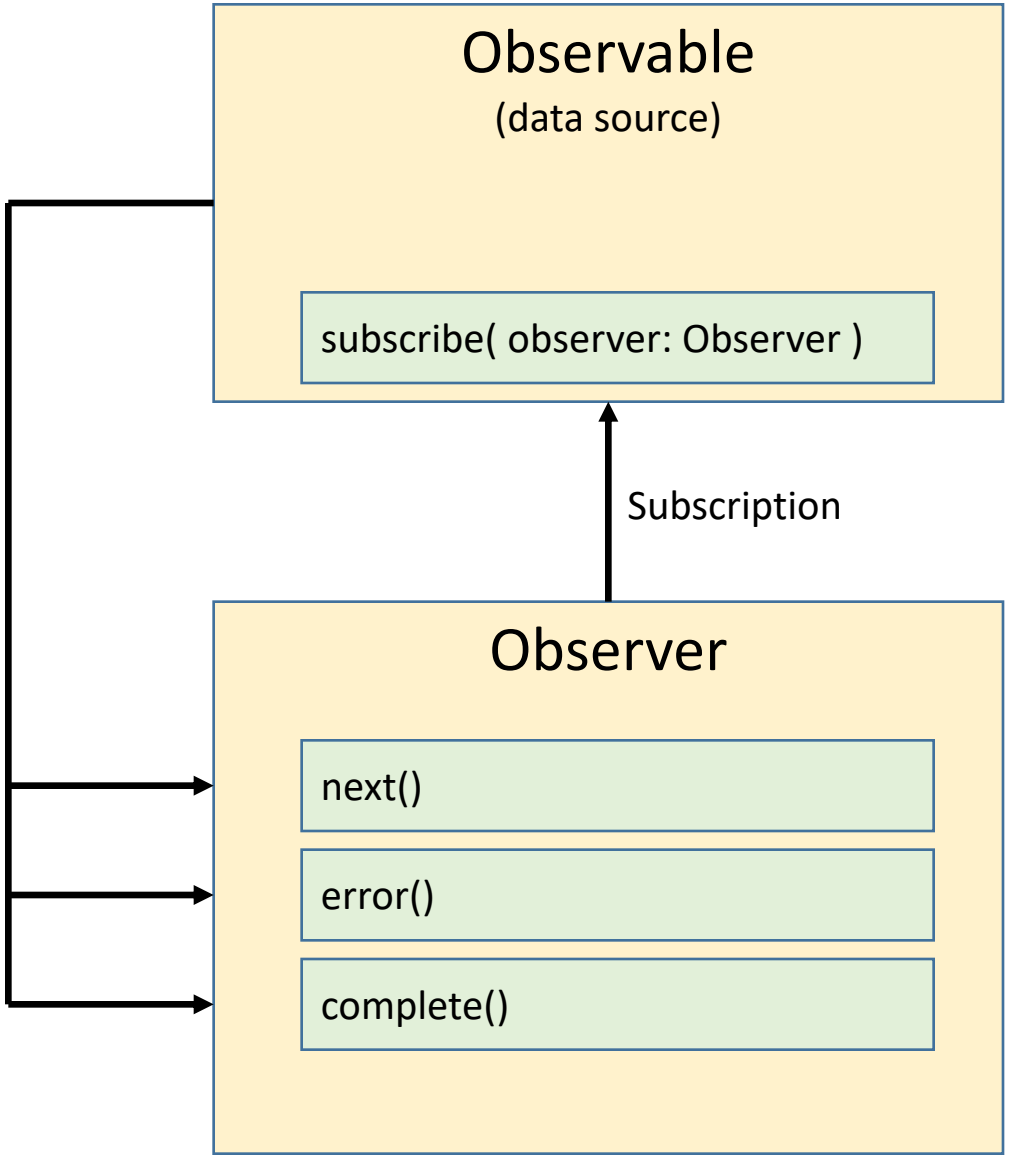
{{ dynamic}}

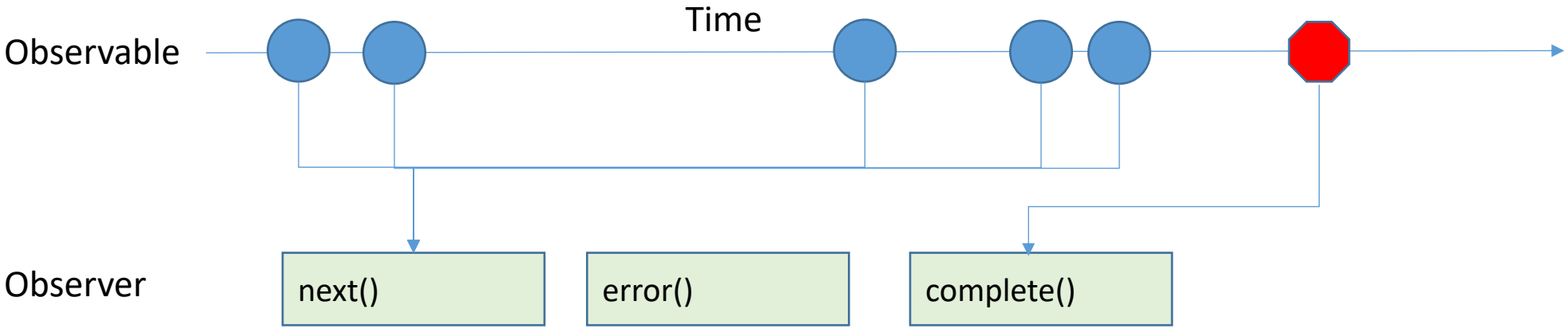
Architecture Diagram

RxJS

RxJS

- **Reactive Extensions for JavaScript**
- RxJS is a library for composing asynchronous and event-based programs by using observable sequences.
- It provides one core type, the Observable, satellite types (Observer, Schedulers, Subjects) and operators inspired by Array#extras(map, filter, reduce, every, etc) to allow handling asynchronous events as collections
- Using RxJS, developers represent asynchronous data streams with Observables and query asynchronous data streams using the many operators (functions) provided


Observable
(data source)
subscribe(observer: Observer)
Subscription
Observer
next()
error()
complete()


Time
Observable
Observer
next()
error()
complete()

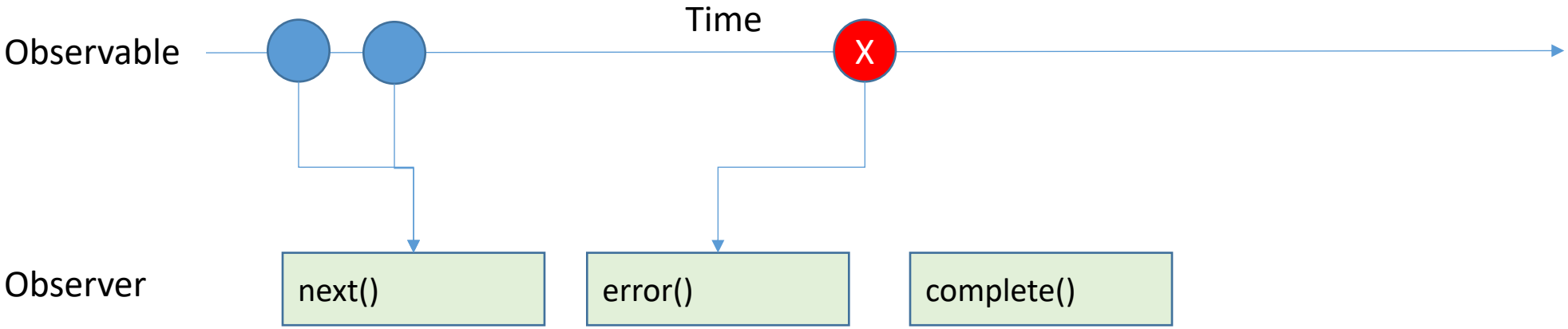
Observable

- A representation of any set of values over any amount of time
- The most basic building block of RxJS
- Represents a data source that streams values over time
- Observables are lazy push collections of multiple values

	Single	Multiple
Pull	Function	Iterator
Push	Promise	**Observable**

Observable Creation Functions

- Usually used to create Observables from scratch
- Pure functions attached to the Observable class in RxJS <=5.4
- Stand-alone functions in RxJS >=5.5 to RxJS<6
  - import { of } from 'rxjs/observable/of';
- RxJS >=6
  - import { of } from 'rxjs';


Time
Observable
X
Observer
next()
error()
complete()

Creating Observables

```
of(1,2,3)
.subscribe(x=> console.log(x)); // 1, 2, 3
```

```
from([1,2,3])
.subscribe(x=> console.log(x)); // 1, 2, 3
```

Creating Observables

```
let button = document.querySelector("button");
fromEvent(button, "click")
.subscribe(x => console.log(x));
```

```
let input = document.querySelector("input");
fromEvent(input, "keyup")
.subscribe((x: Event) => console.log(x.target.value));
```

Demo: Obervables

Instructor Only

code/demos/rxjs-observables

Observer

- A collection of callbacks that knows how to listen to values delivered by the Observable
- An Observer is a consumer of values delivered by an Observable
- Observers are simply a set of callbacks, one for each type of notification delivered by the Observable:
  - next
  - error
  - complete

Observer Example

```
//Observer
let observer: Observer<any> = {
    next: x => console.log(x),
    complete: () => console.log('completed'),
    error: x => console.log(x)
}

of(1,2,3)
.subscribe(observer); // 1, 2, 3, completed
```

Demo: Observers

Instructor Only

code/demos/rxjs-observers

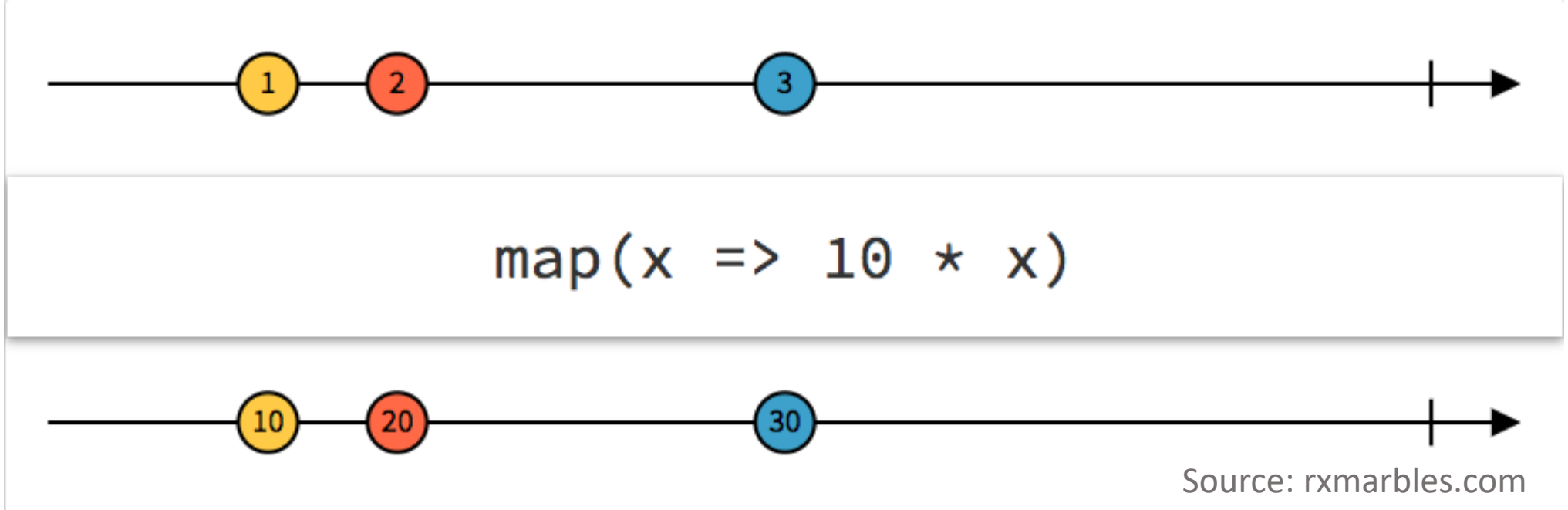
code/demos/rxjs-subscriptions

Operators

- Methods on Observable instances in RxJS <=5.4
- Stand-alone functions in RxJS >=5.5
  - import { map, filter } from 'rxjs/operators';
  - Use the this keyword to infer what is the input Observable

map Operator

```
//map returns same number of items as source
of(1, 2, 3)
  .pipe(map(x => x * 10))
  .subscribe(x => console.log(x));  //10, 20, 30
```



Source: rxmarbles.com

switchMap Operator

```
let obs1$ = of(1, 2, 3);
let obs2$ = of('a', 'b');

obs1$.pipe(switchMap(() => obs2$)).subscribe(observer);
// a, b, a, b, a, b, completed
```

```
let obs1$ = of(1, 2, 3);
let obs2$ = of('a', 'b');

obs1$.pipe(switchMap(() => obs2$, (n, l) => n + l)).subscribe(observer);
//1a, 1b, 2a, 2b, 3a, 3b
```

Demo: Operators

Instructor Only

code/demos/rxjs-operators

Subject

- A Subject is like an Observable, but can multicast to many Observers
- Subjects are like EventEmitters: they maintain a registry of many listeners
- Plain Observables are unicast (each subscribed Observer owns an independent execution of the Observable)

Subject Example

```
var subject = new Subject();


subject.subscribe({
  next: (v) => console.log('observerA: ' + v)
});

subject.subscribe({
  next: (v) => console.log('observerB: ' + v)
});


subject.next(1);
subject.next(2);


// observerA: 1
// observerB: 1
// observerA: 2
// observerB: 2
```

Practical Application of RxJS

```
this.items = this.searchTermStream.pipe(
  debounceTime(300),
  distinctUntilChanged(),
  switchMap((term: string) => this.wikipediaService.search(term))
);
```

- debounceTime waits for the user to stop typing for at least 300 milliseconds
- distinctUntilChanged ensures that the service is called only when the new search term is different from the previous search term
- switchMap calls the WikipediaService with a fresh, debounced search term and coordinates the stream(s) of service response

Demo: Practical Application of RxJS

Instructor Only

code/demos/rxjs-practical

EventEmitter or Observable

- Summary
  - Use EventEmitter in Components
  - Use some form of an Observable (Observable, Subject, BehaviorSubject) in Services

- Explanation
  - Do NOT count on EventEmitter continuing to be an Observable
  - Do NOT count on those Observable operators being there in the future
  - Only call event.emit()… Don't defeat angular's abstraction

Lab

RxJS Demos (optional): a8_custom/RxJS.md

Lab 31: Begin (starting point)

Lab 31: Search using RxJS

Promises vs. Observables

Promises

- Returns single value
- Not cancellable
- Standard as of ES 2015

Observables

- Returns multiple values over time
- Cancellable
- Retry
- Supports map, filter, reduce and similar operators
- Proposed feature for ES 2016
  - Angular uses Reactive Extensions (RxJS)

Demo: HTTP Retry

Instructor Only

code/demos/http-retry

Unsubscribing

From Observables in Angular

Unsubscribing from Observables in Angular

- Why is it necessary?
  - Creates a slow memory leak which will eventually affect application performance
- When is it necessary?
  - *Usually* **not** in **services** because they are often singletons
  - **Not** when dealing with the **router**
    - For example, using ActivatedRoute to be notified when parameters change.
    - Why? Because the Angular framework unsubscribes for you.
  - It **is necessary** in **components**
    - Commonly when a component is using a service to make an HTTP call and subscribes to the results
    - …Unless you are using the **| async** pipe which automatically subscribes and unsubscribes

Approaches to Unsubscribing

- Unsubscribe (subscription.unsubscribe())
  - Pros: easy to understand
  - Cons: verbose, manual (need to correctly implement OnDestroy interface in every component), not as declarative (i.e. not thinking like RxJS)
- Async Pipe (data$ | async)
  - Pros: easy to understand, terse syntax, automatic
  - Handle errors in code using the Observable instances pipe method with the catchError operator
- takeUntil
  - Pros: declarative (RxJS approach), upfront
  - Cons: verbose, manual (need to correctly implement OnDestroy interface in every component)
  - Common question: .next() unsubscribes the observable then .complete() unsubscribes the subject watching that observable

More Information

- https://stackoverflow.com/questions/38008334/angular-rxjs-when-should-i-unsubscribe-from-subscription
- https://blog.angularindepth.com/the-best-way-to-unsubscribe-rxjs-observable-in-the-angular-applications-d8f9aa42f6a0
- https://brianflove.com/2016/12/11/anguar-2-unsubscribe-observables/
- https://sebastian-holstein.de/post/error-handling-angular-async-pipe/

How to Structure an Application

1. Components
   - If a component gets too complex split it into smaller components
2. After you create more components, more questions arise
   - What types of components are there?
   - How should components interact?
   - Should I inject services into any component?
   - How do I make my components reusable across views?

Component Architecture

Smart/Container Components

- Are concerned with *how things work*
- *Sets data* into child component input properties
- *Receives events* by subscribing to children
- *Loads and modifies data* via calls to an API
- Also know as *container* components or *controller* components

Presentation Components

- Are concerned with *how things look*
- *Receive data* via input properties from parent
- *Send events* with information to their parent
- Don’t specify how the data is loaded or changed
- Also know as *pure* components or *dumb* components

When to Create Another Component

- Is it possible for your code chunk to be reused?
  - If yes, construction of a new component seems like a great idea.
  - Even if the reuse is within a single component.
- Is your code quite complex?
  - If yes maybe its good idea to split in separate components in order to make your code more readable and maintainable.

Custom Events in a Component

- @Output
  - Decorator that marks a class property as sending a custom output event
- EventEmitter
  - Class used in directives and components to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance

Input Property

- @Input
  - Decorator that marks a class field as an input property
- Property Binding to a Component Property

```
@Component({
    selector: 'app-root',
    template: `
    <app-fruit-list [fruits]="data"></app-fruit-list>
    `,
    styles: []
})
export class AppComponent {
  data: string[] = ['Apple', 'Orange', 'Plum'];
}
```

Demo: Input Property

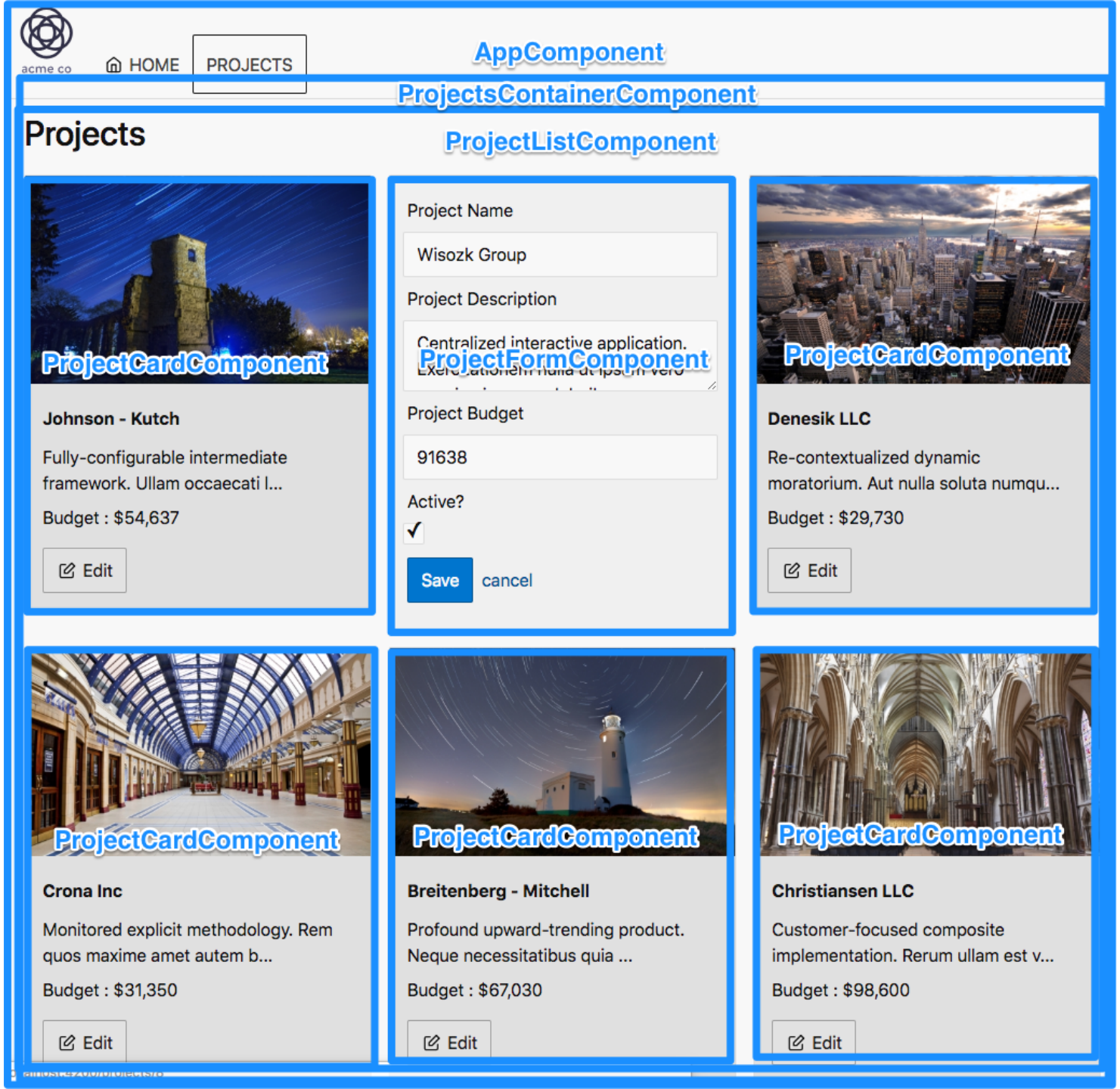
Instructor Only Demonstration

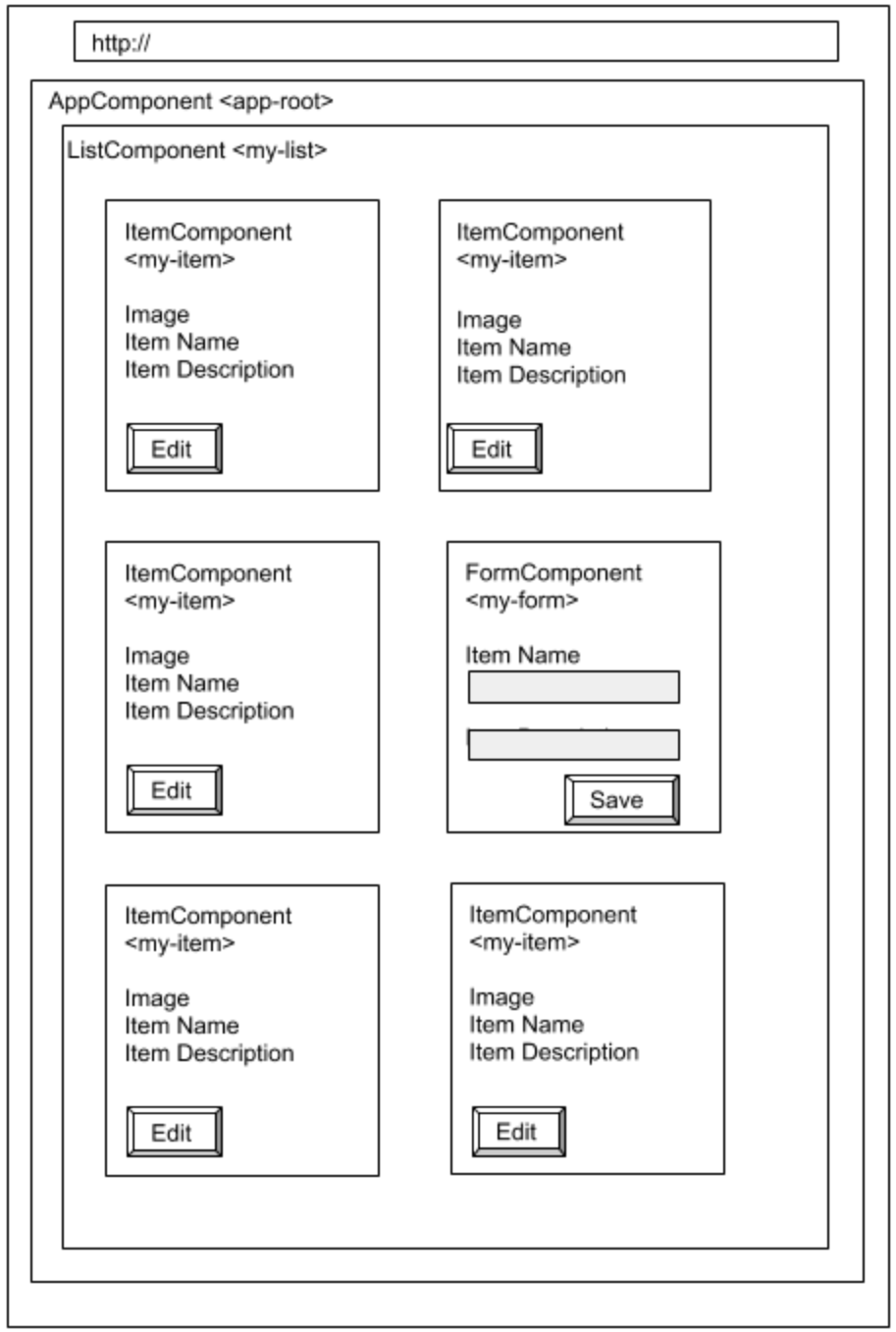
code\demos\input-property

Demo: Output Events

Instructor Only Demonstration

code\demos\output-events


acme co
HOME
PROJECTS
AppComponent
ProjectsContainerComponent
Projects
ProjectListComponent
ProjectCardComponent
Johnson - Kutch
Fully-configurable intermediate framework. Ullam occaecati l...
Budget : $54,637
Edit
Project Name
Wisozk Group
Project Description
Centralized interactive application.
ProjectFormComponent
Project Budget
91638
Active?
Save
cancel
ProjectCardComponent
Denesik LLC
Re-contextualized dynamic moratorium. Aut nulla soluta numqu...
Budget : $29,730
Edit
ProjectCardComponent
Crona Inc
Monitored explicit methodology. Rem quos maxime amet autem b...
Budget : $31,350
Edit
ProjectCardComponent
Breitenberg - Mitchell
Profound upward-trending product. Neque necessitatibus quia ...
Budget : $67,030
Edit
ProjectCardComponent
Christiansen LLC
Customer-focused composite implementation. Rerum ullam est v...
Budget : $98,600
Edit



index.html

```
<html>
<body>
 <app-root>
   <my-list>
     <div *ngFor="let item of items">
       <my-item>
         <div class="list-item">
           <img src="..." alt="item name">
           <h3>Item Name</h3>
           <p>Item Description</p>
         </div>
       </my-item>
       <my-form>
         <div class="form">
           <form>
             <label>Item Name</label>
             <input type="text" name="item-name">
             <label>Item Description</label>
             <input type="text" name="item-description">
             <button>Save</button>
           </form>
         </div>
       </my-form>
     </div>
   </my-list>
 </app-root>
</body>
</html>
```

app.component.html

list.component.html

item.component.html

form.component.html

Labs

Lab 9: Begin : Starting Point

Lab 9: More Reusable Components

Lab 10: Responding to an Event

Lab 11: Create a Form to Edit Your Data

Lab 12: Communicating from Child to Parent Component

Lab 13: Hiding and Showing Components

Attendees Hands-On Together

Custom Pipe Example

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
    name: 'characterLength'
})
export class CharacterLengthPipe implements PipeTransform{

    transform(value:string, length:number){
        return value.substring(0, length);
    }
}
```

Use:

```
<p>description | characterLength </p>
```

```
src
  app
    core
      exception.service.ts|spec.ts
      user-profile.service.ts|spec.ts
    heroes
      hero
        hero.component.ts|html|css|spec.ts
      hero-list
        hero-list.component.ts|html|css|spec.ts
      shared
        hero-button.component.ts|html|css|spec.ts
        hero.model.ts
        hero.service.ts|spec.ts
      heroes.component.ts|html|css|spec.ts
      heroes.module.ts
      heroes-routing.module.ts
    shared
      shared.module.ts
      init-caps.pipe.ts|spec.ts
      filter-text.component.ts|spec.ts
      filter-text.service.ts|spec.ts
```

Shared Feature Module

Do create a feature module named SharedModule

app/shared/shared.module.ts de fines SharedModule

Do declare *components*, *directives*, and *pipes*

When those items will be **re-used**

Consider *not* providing services in shared modules.

Services are usually singletons that are provided once for the entire application or in a particular feature module

Core Feature Module

Do create a feature module named **CoreModule**

- **app/core/core.module.ts** defines CoreModule

Do **declare** *common Services*

- When those items will be **re-used**
  - **Exception/Logging Service**
  - **User Profile Service**

Labs

Lab 28: Custom Pipe

Pure Pipe

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
    name: 'charlength',
    pure: true // default
})
export class CharacterLengthPipe implements PipeTransform{

    transform(value:string, length:number){
        return value.substring(0, length);
    }
}
```

pure: When true, the pipe is pure, meaning that the **transform()** method is **invoked only when** its **input arguments change**. Pipes are pure by default.

Impure Pipe

- If the pipe has internal state (that is, the result depends on state other than its arguments), set pure to false.
  - In this case, the pipe is invoked on each change-detection cycle, even if the arguments have not changed.

Change Detection

Angular

Component State

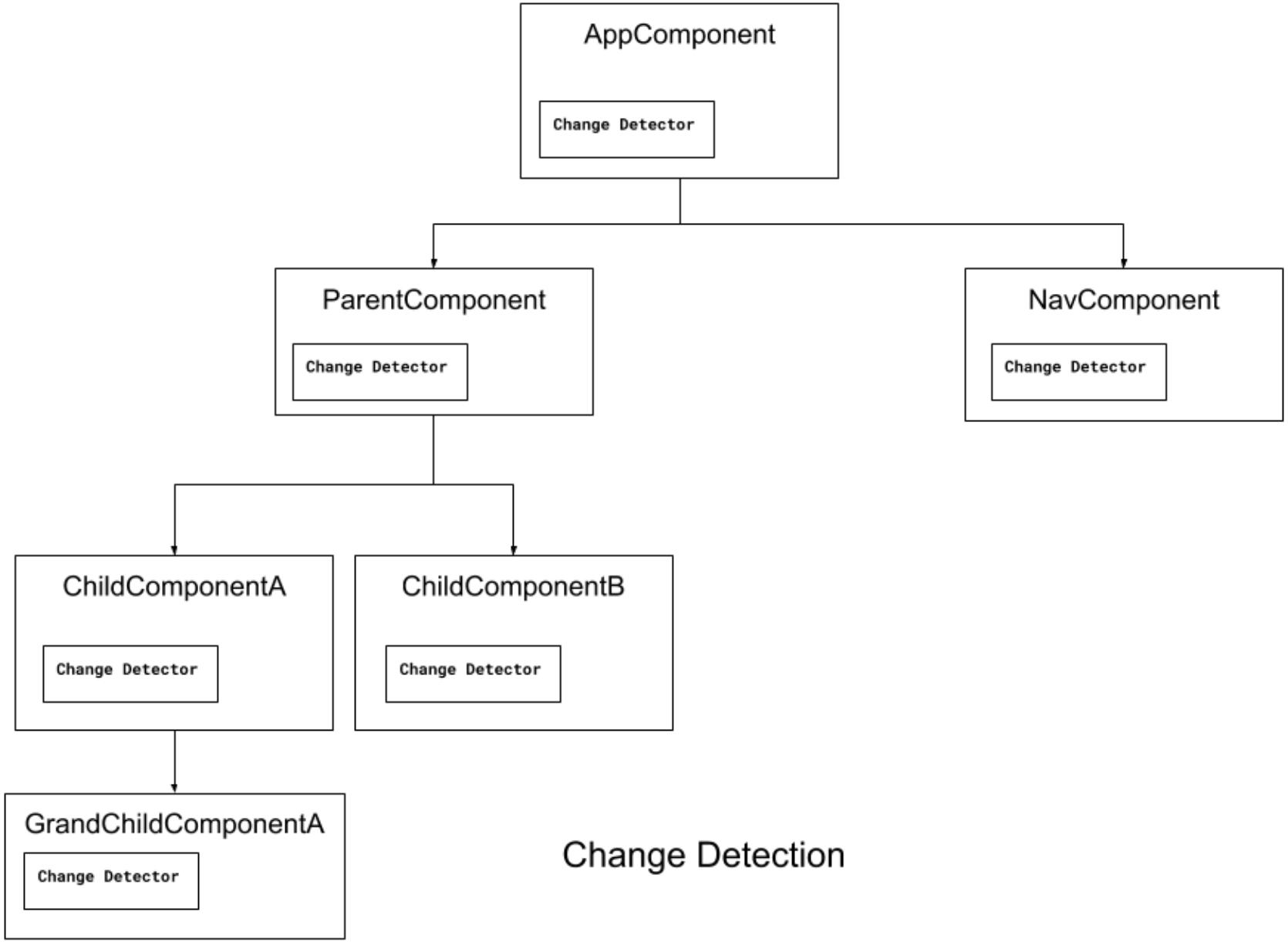
- What can trigger component state to change?
  - User event (click, change, input, etc…)
  - HTTP request or web socket message
  - Timer

Zone.js

- Angular uses the JavaScript library zone.js
- Automatically detects things changed and triggers change detection
- Common scenarios it detects
  - DOM Events (user events like click)
  - HTTP request is resolved
  - Timer is triggered (setTimout or setInterval)

Components & Change Detection

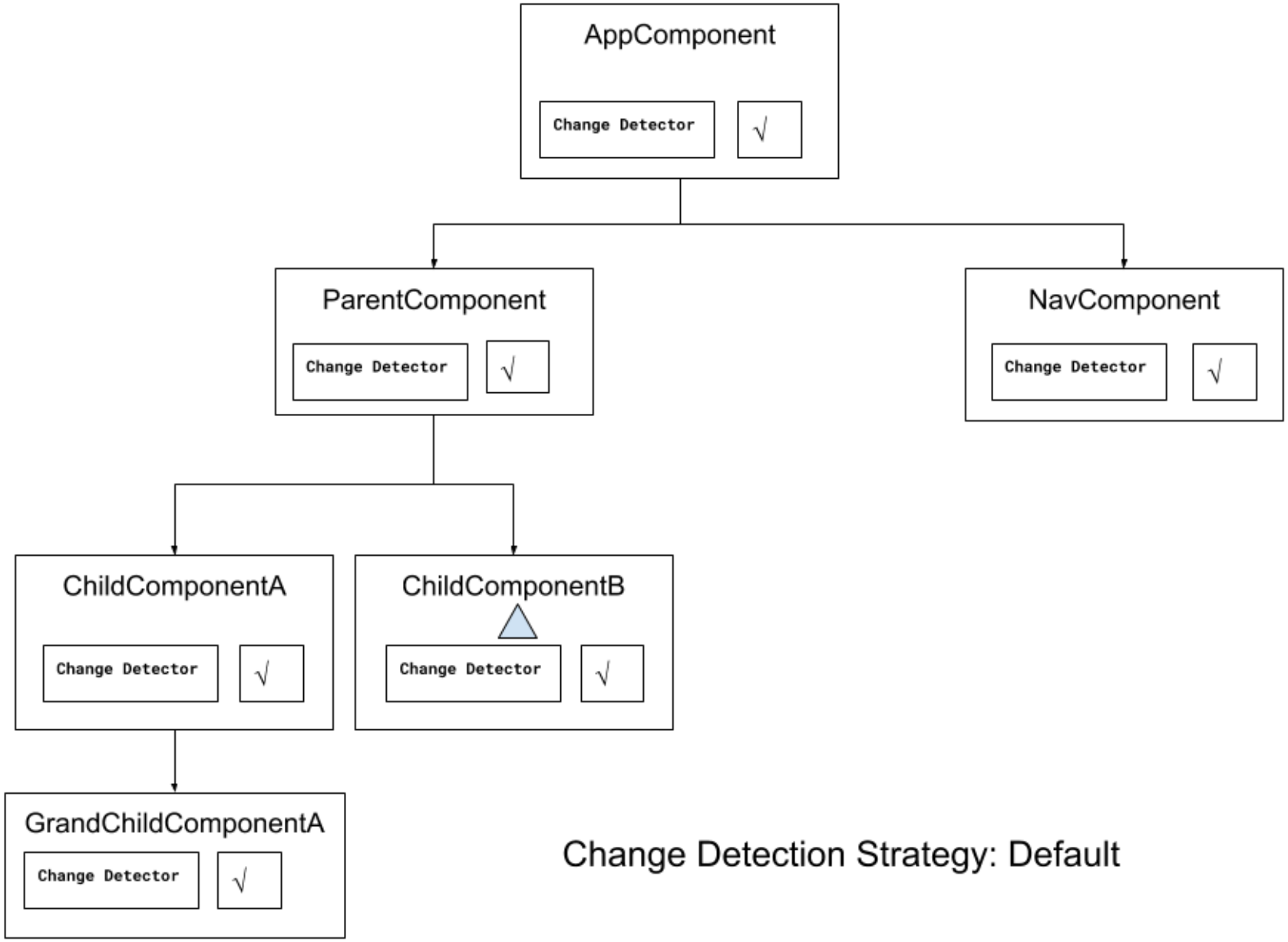
- Every component has a change detector
- Components are nested in a hierarchical tree
- Directives do not have their own change detectors
  - use the change detector of their parent component



Change Detection

Change Detection Strategies

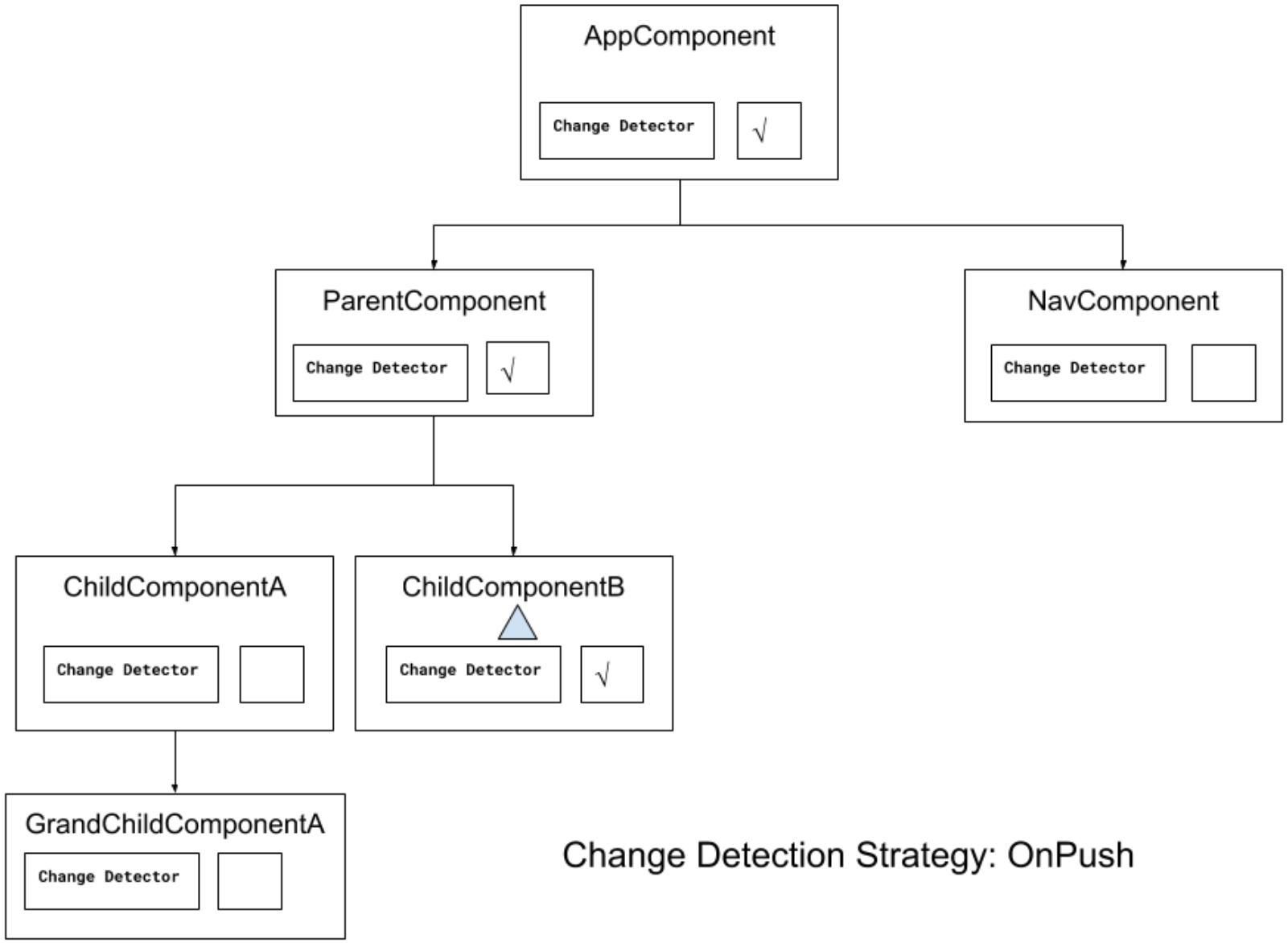
- **Default**
  - Like AngularJS
  - When one component changes
    - Change detection is triggered
      - For the whole tree
      - From the top down
      - Regardless of where in the tree the change happens
  - Can be overkill when using
    - Immutable objects
    - Observables
  - Can modify the change detection strategy from the default



Change Detection Strategy: Default

Change Detection Strategies

- **OnPush**
  - Change detection is triggered when
    - An [input] property changes
    - An event handler fired in the component or directive
    - The change detector of a child component or directive runs
    - You manually tell the change detector to look for changes
      - Injecting ChangeDetectorRef
      - Invoking the MarkForCheck method
  - Just that component and and its ancestors are checked
  - Helps Angular not to do “unnecessary work” if you are utilizing:
    - Immutable data
    - Observables



Change Detection Strategy: OnPush

ChangeDetectionStrategy.OnPush

project-card.component.ts

```
@Component({
    selector: 'project-card',
    templateUrl: './project-card.component.html',
    styleUrls: ['./project-card.component.css'],
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProjectCardComponent implements OnInit {
}
```

Demo: Change Detection

Instructor Only Demonstration

code\demos\change-detection

OnPush Code Changes

- If you change an object you are passing to a component, don’t change its properties in place; rather, construct a copy with the change applied
- Angular will now ignore change detection container and presentation components until there is a change in their input references

Unit Testing

Angular

Why Unit Test?

- They **guard** against changes that break existing code (“regressions”).
- They **clarify** what the code does both when used as intended and when faced with deviant conditions.
- They **reveal** mistakes in design and implementation. Tests shine a harsh light on the code from many angles. When a part of the application seems hard to test, the root cause is often a design flaw, something to cure now rather than later when it becomes expensive to fix.

Tools and Technologies

Technology	Purpose
Jasmine	The Jasmine test framework. provides everything needed to write basic tests. It ships with an HTML test runner that executes tests in the browser.
Angular Testing Utilities	The Angular testing utilities create a test environment for the Angular application code under test. Use them to condition and control parts of the application as they interact within the Angular environment.
Karma	The karma test runner is ideal for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes. This chapter describes how to setup and run tests with karma.
Protractor	Use protractor to write and run end-to-end (e2e) tests. End-to-end tests explore the application as users experience it. In e2e testing, one process runs the real application and a second process runs protractor tests that simulate user behavior and assert that the application responds in the browser as expected.

Testing Syntax

Jasmine

- spec = file with tests
- describe = suites (grouping)
- it = test
- beforeEach, afterEach = setup, teardown
- expect = assert
- matchers = assertions

Your First JavaScript Test

```
describe('Smoke Test', () => {
  it('should run a passing test', () => {
    expect(true).toEqual(true);
  });
});
```

Write a “smoke” test to verify your test environment is properly configured and working.

Make it fail before making it succeed.

Demo: First Test

Instructor Only

Instructor walks through code in Unit Testing Lab 1: First Test

Lab

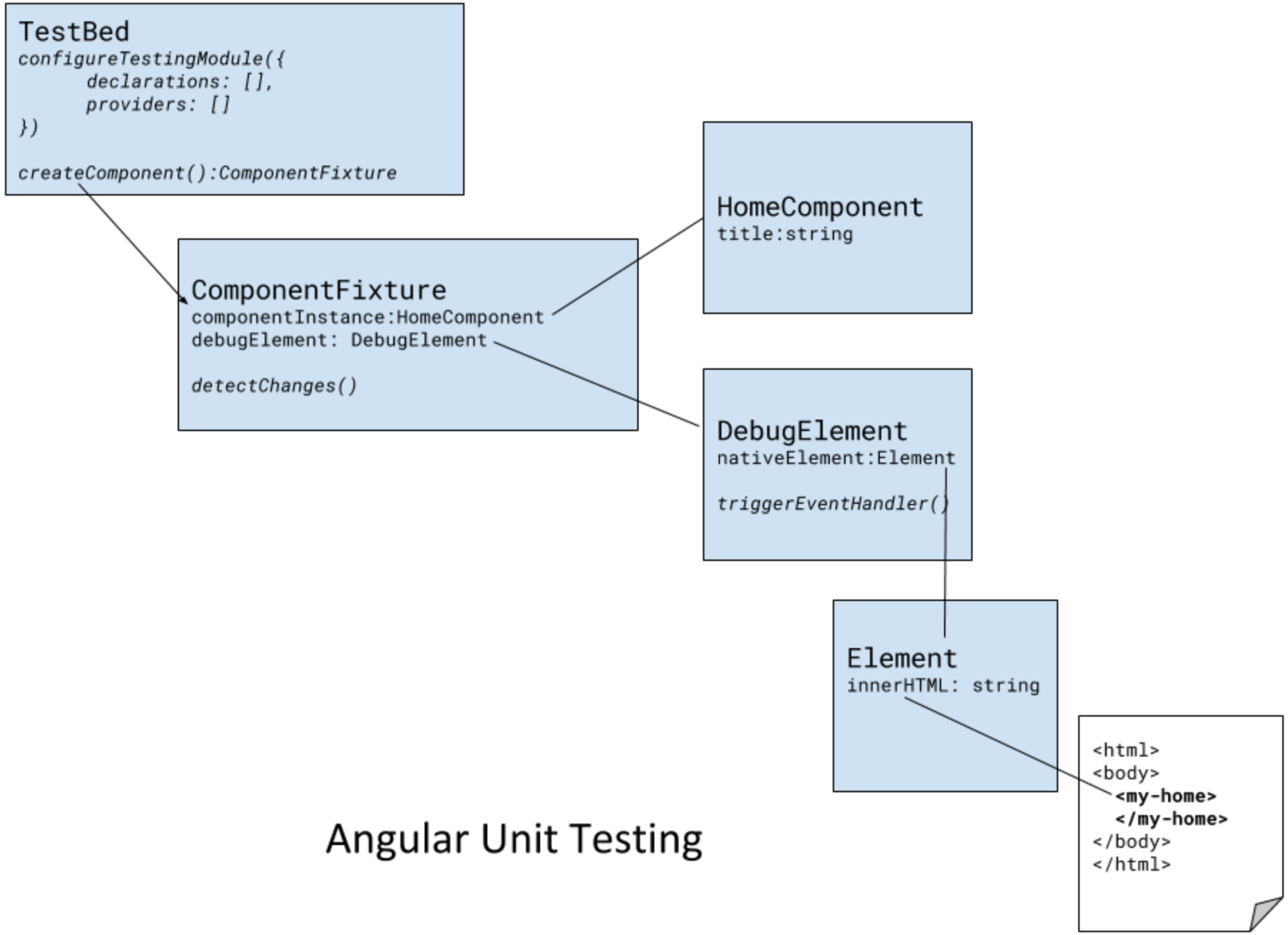
Unit Testing Lab 1: First Test

Attendee Hands-On Lab

Testing Terminology

Angular

- TestBed
  - Configures and initializes environment for unit testing and provides methods for creating components and services in unit tests.
- ComponentFixture
  - Fixture for debugging and testing a component.
  - A **test fixture** is a fixed state of a set of objects used as a baseline for running **tests**.
- DebugElement
  - Holds a native element and provides convenience methods to make it easier to test.



Angular Unit Testing

Demo: Component Test

Instructor Only

Instructor walks through code in Unit Testing Lab 2: Component Test

Lab

Unit Testing Lab 2: Component Test

Attendee Hands-On Lab

Demo: Component with Input & Output

Instructor Only

Instructor walks through code in Unit Testing Lab 3: Component with Input & Output

Lab

Unit Testing Lab 3: Component with Input & Output

Attendee Hands-On Lab

Testing Terminology

Mocking vs. Spying

- Mock object replaces a mocked class entirely, returning recorded or default values. You can create a mock out of "thin air". This is what is mostly used during unit testing.
- When spying, you take an existing object and "replace" only some methods. This is useful when you have a huge class and only want to mock certain methods (partial mocking).

Testing Terminology

Spy

- Functions that record arguments, exceptions, and return values for any of their calls.

Testing Terminology

Angular

Function	Description
async	Runs the body of a test (it) or setup (beforeEach) function within a special async test zone.
fakeAsync	Runs the body of a test (it) within a special fakeAsync test zone, enabling a linear control flow coding style.
tick	Simulates the passage of time and the completion of pending asynchronous activities by flushing both timerand micro-task queues within the fakeAsync test zone. Accepts an optional argument that moves the virtual clock forward the specified number of milliseconds, clearing asynchronous activities scheduled within that timeframe.
inject	Injects one or more services from the current TestBed injector into a test function.

Demo: Component with Service

Instructor Only

Instructor walks through code in Unit Testing Lab 4: Component with Service

Lab

Unit Testing Lab 4: Component with Service

Attendee Hands-On Lab

Demo: Service Mocking Http

Instructor Only

Instructor walks through code in Unit Testing Lab 5: Service Mocking Http

Lab

Unit Testing Lab 5: Service Mocking Http

Attendee Hands-On Lab

Demo: Unit Test Pipe

Instructor Only

Instructor walks through code in Unit Testing Lab 6: Pipe

Lab

Unit Testing Lab 6: Pipe

Attendee Hands-On Lab

Redux

Redux

- Redux is an open-source JavaScript library for managing application state
- Commonly used with libraries such as React or Angular for building user interfaces.
- Similar to (and inspired by) Facebook's Flux architecture, it was created by Dan Abramov and Andrew Clark.

State

- State: the particular condition that something is in at a specific time.
- ***Application State*** (also known as ***Program State***)
  - Represents the totality of everything necessary to keep your application running
  - The state of the program/application as it exists in the contents of its memory
  - In any given point in time, there is a different information stored in the memory of your web application
    - Can be accessed via your variables, classes, data structures, etc.
  - All the stored information, at a given instant in time, is called the application state

Examples of Application State

- The current URL of a web page
- The displayed data often returned from a call to a web API using HTTP
- How the data is currently sorted or filtered
- Whether a user is logged in and their associated profile data
- Whether a navigation menu is currently open and displayed or hidden

What to Store in State

- Shared data
  - Does the data matter to the application as a whole?
  - Are there other components that may benefit from this global accessible shared data?

Three Principles of Redux

- Single source of truth
  - The whole state of your app is stored in an object tree inside a single *store*.
- State is read-only
  - The only way to change the state tree is to emit an *action*, an object describing what happened.
- Changes are made with pure functions
  - To specify how the actions transform the state tree, you write pure *reducers*.

Benefits

- **Predictable state updates** make it easier to understand how the data flow works in the application
- The use of **"pure" reducer functions** makes logic **easier to test**, and enables useful features like "time-travel **debugging**".
- **Centralizing the state** makes it easier to implement things like logging changes to the data, or persisting data between page refreshes

Benefits Checklist

- Explicit state, predictable state, repeatable trail of state
- Performance
  - OnPush
- Testability
- Debugging/Tooling
  - Time travel debugging
  - Record/Replay
  - Hot reloading
    - Refreshes files that were changed without losing the state of the app
- Component Communication

"Redux is not great for making simple things quickly. It's great for making really hard things simple."

- Jani Evakallio

Single Source of Truth

Describe application state as plain objects and arrays.

```
let appState = {
    todos: [
        {
            text: 'Consider using Redux',
            completed: true
        },
        {
            text: 'Keep all state in a single tree',
            completed: false
        }
    ],
    visibilityFilter: 'SHOW_ALL'
};
```

State is Read-Only

Describe changes in the system as plain objects.

```
store.dispatch({type: 'ADD_TODO', text: 'Learn Redux' });
```

The only way to change the state tree is to emit an *action*, an object describing what happened.

Functional Programming

- a **programming** paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

Pure Function

- A **function** where the return value is only determined by its input values, without observable side effects.
  - This is how **functions** in math work: Math.cos(x) will, for the same value of x , always return the same result. Computing it does not change x .

Pure Functions vs Impure Functions

```
// Pure functions
function square(x) {
    return x * x;
}
function squareAll(items) {
    return items.map(square);
}
```

```
// Impure functions
function square(x) {
    updateXInDatabase(x);
    return x * x;
}
function squareAll(items) {
    for (let i = 0; i < items.length; i++) {
        items[i] = square(items[i]);
    }
}
```

Reducer

- (previousState, action) => newState

- Should be "*pure*"
  - Does not perform side effects (such as calling API's or modifying non-local objects or variables).
  - Does not call non-*pure* functions (like Date.now or Math.random).
  - Does not mutate parameters passed to it.

Changes are Made with Pure Functions

```
const todosReducer = (state = [], action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          id: action.id,
          text: action.text,
          completed: false
        }
      ]
    case 'TOGGLE_TODO':
      return state.map(todo =>
        (todo.id === action.id)
          ? {...todo, completed: !todo.completed}
          : todo
      )
    default:
      return state
  }
}
```

Describe the logic for handling changes as pure functions.

NgRx

- Most popular state management library used with Angular
- Redux + RxJS = NgRx
- Store is reactive (RxJS powered) state management for Angular applications, inspired by Redux. Store is a controlled state container designed to help write performant, consistent applications on top of Angular
- ngrx.io

Demo: Redux

Instructor Only

ngrx-counter

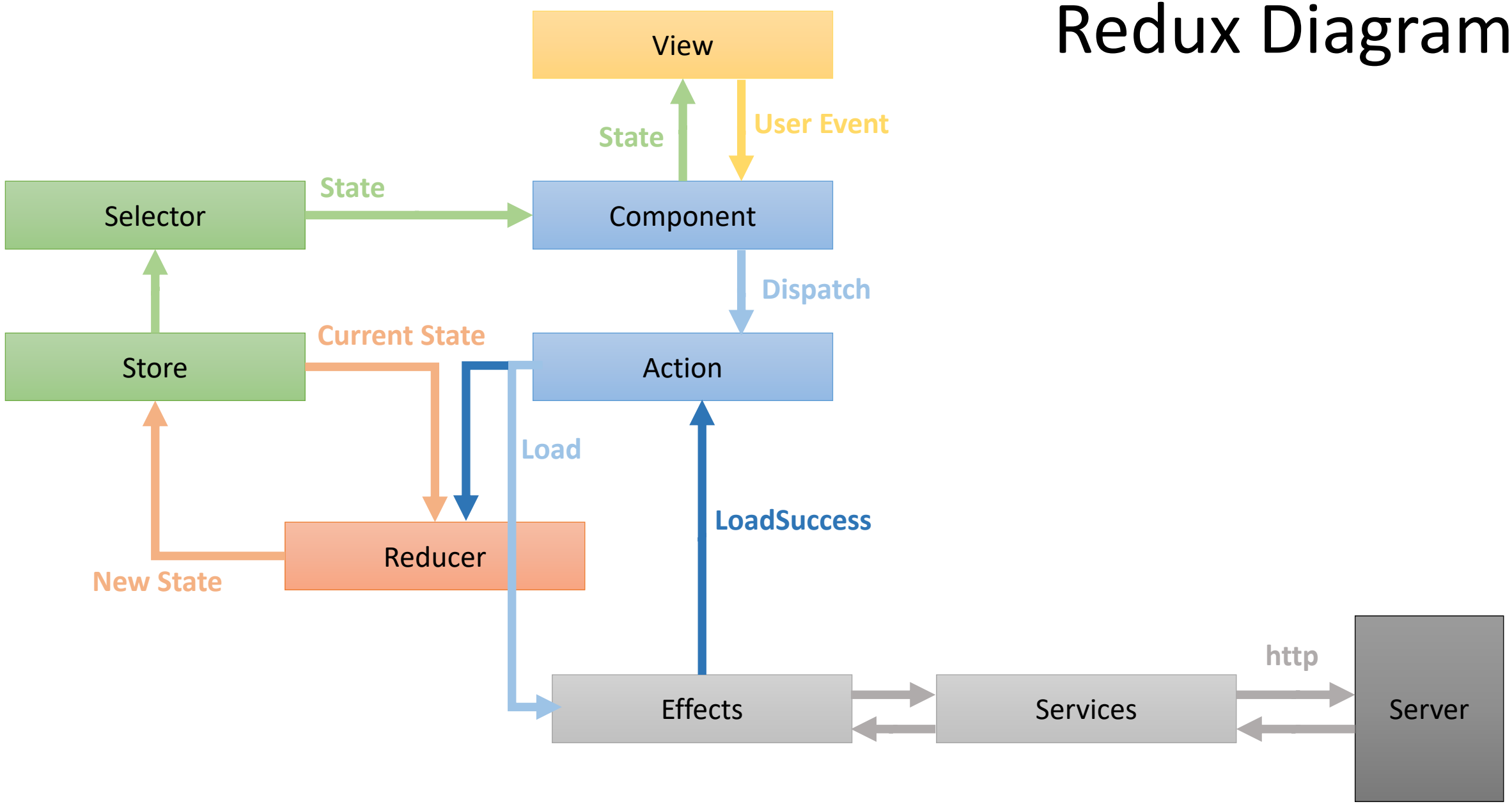
Lab: Redux

Attendee Hands-On Lab (optional if time permits and group feels it will be valuable)

a8_custom/NgRxCounter.md

ngrx-counter

Redux Diagram



Being Explicit vs Implicit

Lab

NgRx Lab

a8_custom/NgRxLab.md

Attendee Hands-On Lab

RxJS Operators with NgRx

- **switchMap** | **Cancels** the **current subscription/request** and can cause race condition
  - Use for **get** requests or **cancelable** requests like searches
- **mergeMap** | Runs subscriptions/requests **in parallel**
  - Use for **put, post and delete** methods when **order** is **not important**
- **concatMap** | Runs subscriptions/requests **in order** so is less performant
  - Use for **get, post, and put** requests when **order** is **important**
- **exhaustMap** | **Ignores** all subsequent subscription/requests **until** it **completes**
  - Use for login when you want the authentication request to complete before accepting additional requests

Sample: Redux Application (using ngrx)

- Open the **samples/ngrx-material/project-manage** directory
- Uses ngRx 8 but doesn't take advantage of new helpers
- Get the project setup and running
  - Follow the directions in project-manage/readme.md
    - *Note: Since the readme files are formatted in markdown you will need to right click and choose 'open preview' in Visual Studio Code or your favorite markdown editor to see the formatted content.*
- Review the code
  - Uses ngrx to implement a Redux architecture for state management
  - Also uses Angular Material Design

Specific Benefits

- Persist state to a local storage and then boot up from it, out of the box.
- Pre-fill state on the server, send it to the client in HTML, and boot up from it, out of the box.
- Serialize user actions and attach them, together with a state snapshot, to automated bug reports, so that the product developers can replay them to reproduce the errors.
- Pass action objects over the network to implement collaborative environments without dramatic changes to how the code is written.
- Maintain an undo history or implement optimistic mutations without dramatic changes to how the code is written.
- Travel between the state history in development, and re-evaluate the current state from the action history when the code changes, a la TDD.
- Provide full inspection and control capabilities to the development tooling so that product developers can build custom tools for their apps.
- Provide alternative UIs while reusing most of the business logic.

Advanced Routing

Angular

Child Routes

- More than one <router-outlet> can exist in an Angular app
- The Angular router allows nesting routes, i.e., having a child route inside a parent route so that multiple components can be displayed on the same page
- For example the list page may show :
  - the item detail next to the list
  - the form to edit the item next to the list

Demo: Child Routes

Instructor Only

code/demos/routing-child-routes

Lazy Loading Modules

- Despite the build optimizations performed by the Angular CLI, the JavaScript that makes up an Angular application can become large enough that it takes several seconds for the initial load/download of the application
- The user experience can be improved using a splash or loading screen but applications can become large enough that another solution is needed
- Lazy Loading the code in an Angular module can provide a solution to this problem

Lazy Loading Modules

- An application may be used by **thousands of users** but only a **few of those users are administrators** that maintain drop down list values, permissions, and others settings for the system
- If you organized your code following the Angular Style Guide by feature modules you will have an **Admin feature module**
- Lazy Loading allows you to delay the loading of that module's code (Admin) until a user visits the **/admin** route
- As a result, the thousands of users who use the main part of the application will **not need to download code** to their browser that they will never use

Demo: Lazy Loading a Module

Instructor Only

code/demos/routing-lazy-loading-module

Component Lifecycle

- A component has a lifecycle managed by Angular.
- Angular **creates** and renders components along with their children, checks when their data-bound *properties* **change**, and **destroys** them before removing them from the DOM.
- Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.

Hook	Purpose and Timing
ngOnChanges()	Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. Called before ngOnInit() and whenever one or more data-bound input properties change.
ngOnInit()	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called *once*, after the *first* ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().
ngAfterContentInit()	Respond after Angular projects external content into the component's view / the view that a directive is in. Called *once* after the first ngDoCheck().
ngAfterContentChecked()	Respond after Angular checks the content projected into the directive/component. Called after the ngAfterContentInit() and every subsequent ngDoCheck().
ngAfterViewInit()	Respond after Angular initializes the component's views and child views / the view that a directive is in. Called *once* after the first ngAfterContentChecked().
ngAfterViewChecked()	Respond after Angular checks the component's views and child views / the view that a directive is in. Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().
ngOnDestroy()	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called *just before* Angular destroys the directive/component.

Demo: Lifecycle Hooks (Basics)

Init, Changes, Destroy

Instructor Only

a8_custom/Lifecycle.md

Demo: Lifecycle Hooks (Less Common)

AfterViewInit, AfterContentInit

Instructor Only

a8_custom/Lifecycle.md

ViewChild(ren) & AfterViewInit

- ViewChild & ViewChildren are Angular decorators
- AfterViewInit is a lifecycle hook
  - Called after Angular initializes the component's views and child views
- Demo: ViewChild(ren) & AfterViewInit
  - a8_custom/Lifecycle.md

ContentChild(ren) & ContentViewInit

- ContentChild & ContentChildren are Angular property decorators
- ContentViewInit is a lifecycle hook
  - Called after Angular initializes external content into the component's view
- Demo: ContentChild(ren) & AfterContentInit
  - a8_custom/Lifecycle.md

ngTemplate, ngContainer, ngContent

- <ng-template #myTemplate> is like HTML5 <template> tag
  - Reusable HTML that is not attached to the document
  - Does not render anything until attached to the document
  - Can be attached using *ngIf="condition else myTemplate " or ngSwitch
- <ng-container> is there to keep you from creating unnecessary HTML elements just to be containers for elements you want to show or hide
  - Does render content by default
  - Can be used to hide and show multiple siblings without a parent
- <ng-content> is used to project content into a component (tranclusion)

Build & Deploy

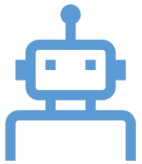
Using Angular CLI (Command-Line Interface)

Angular CLI: Builds

- Builds your application for deployment to production
  - Combining, minifying
  - Tree-shaking, dead code elimination
  - Ahead-of-time compilation
  - Remove Decorators

Ahead-of-Time (AOT) Compiler

- Angular offers two ways to compile your application:
  - *Just-in-Time* (JIT), which compiles your app in the browser at runtime
  - *Ahead-of-Time* (AOT), which compiles your app at build time.
- Converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase *before* the browser downloads and runs that code.
- The size of the Angular framework downloaded in the browser decreases in size by over 50%. Angular's template compiler code does not need to be sent to the browser because the template compilation has been done ahead-of-time.



Build Optimizer

- Build-Optimizer (PurifyPlugin) is a Webpack plugin created by the Angular team, specifically to optimize Webpack builds beyond what Webpack can do on its own.
- Optimizations
  - Removal of Angular decorators from AoT builds
  - adding /*@__PURE__*/ annotations to transpiled/downleveled TypeScript classes. The point of this is to make it easier for minfiers like Uglify to remove unused code.
  - Full list of optimizations is available here:
    - https://www.npmjs.com/package/@angular-devkit/build-optimizer

```
"optimization": true,  //Enables optimization of the build output. (bundling, limited tree-shaking, limited dead code elimination)
"outputHashing": "all", // Define the output filename cache-busting hashing mode.
"sourceMap": false,  // Output sourcemaps.
"extractCss": true, // Extract css from global styles into css files instead of js ones.
"namedChunks": false, // Use file name for lazy loaded chunks.
"aot": true, // Build using Ahead of Time compilation.
"extractLicenses": true, // Extract all licenses in a separate file.
"vendorChunk": false, // Use a separate bundle containing only vendor libraries.
"buildOptimizer": true, // Enables '@angular-devkit/build-optimizer' optimizations when using the 'aot' option.
"fileReplacements": [{
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
 }],
```

CLI: Builds

- The **build** command sends output to the **dist** directory
- **Copy** the contents of the **dist** directory to your web server to deploy
- Run a development build

  ng build --dev

  Note: running just “ng build” defaults to a ”dev” build
- Run a production build

  ng build --prod
- Passing the **--prod** flag effectively sets other flags
  - Shorthand for "--configuration=production"

Bundles

- runtime.bundle.js is webpack
- polyfills.bundle.js includes core-js and zone.js
- main.bundle.js includes all your application code

- styles.bundle.css | .js includes all the CSS component styles as well as global styles in styles.css combined into one file

Demo

Angular CLI: Production Builds

Instructor: completes Lab 29 as a demonstration (steps 1-15)

Browser Compatibility

- JavaScript evolves, with new features
- Browser compatiblity is a common goal
- Browsers don't support new features at same pace
- TypeScript is compiled into JavaScript
- To support older browsers (IE), the output JavaScript has to be ES5 which requires polyfills

Polyfills

- Polyfills providing functionality that doesn't exist in the older versions of JavaScript
- When targeting older browsers, polyfills are required
- Polyfills add to the size of the JavaScript required to run the application

Polyfill Problem

- Majority of users use modern browsers (Chrome, Edge, Firefox) which support ES6/2015 features
- Users with modern browsers
  - Pay the price of extra JavaScript polyfills (that increase the size of JavaScript downloaded)
  - Don't need the polyfills

```
Hash: 1486dd62053784343e95
Time: 26522ms
chunk {0} runtime-es5.741402d1d47331ce975c.js (runtime) 1.41 kB [entry] [rendered]
chunk {1} main-es5.0d657494dcf4e404aff8.js (main) 358 kB [initial] [rendered]
chunk {2} polyfills-es5.0e123feb616b379211ce.js (polyfills) 111 kB [initial] [rendered]

Date: 2019-07-01T18:15:13.333Z
Hash: 0cfc3d9d37ae6f12d52e
Time: 20604ms
chunk {0} runtime-es2015.858f8dd898b75fe86926.js (runtime) 1.41 kB [entry] [rendered]
chunk {1} main-es2015.8e469659067ed4bc9b51.js (main) 309 kB [initial] [rendered]
chunk {2} polyfills-es2015.[illegible].js (polyfills) 36.4 kB [initial] [rendered]
```

Differential Loading

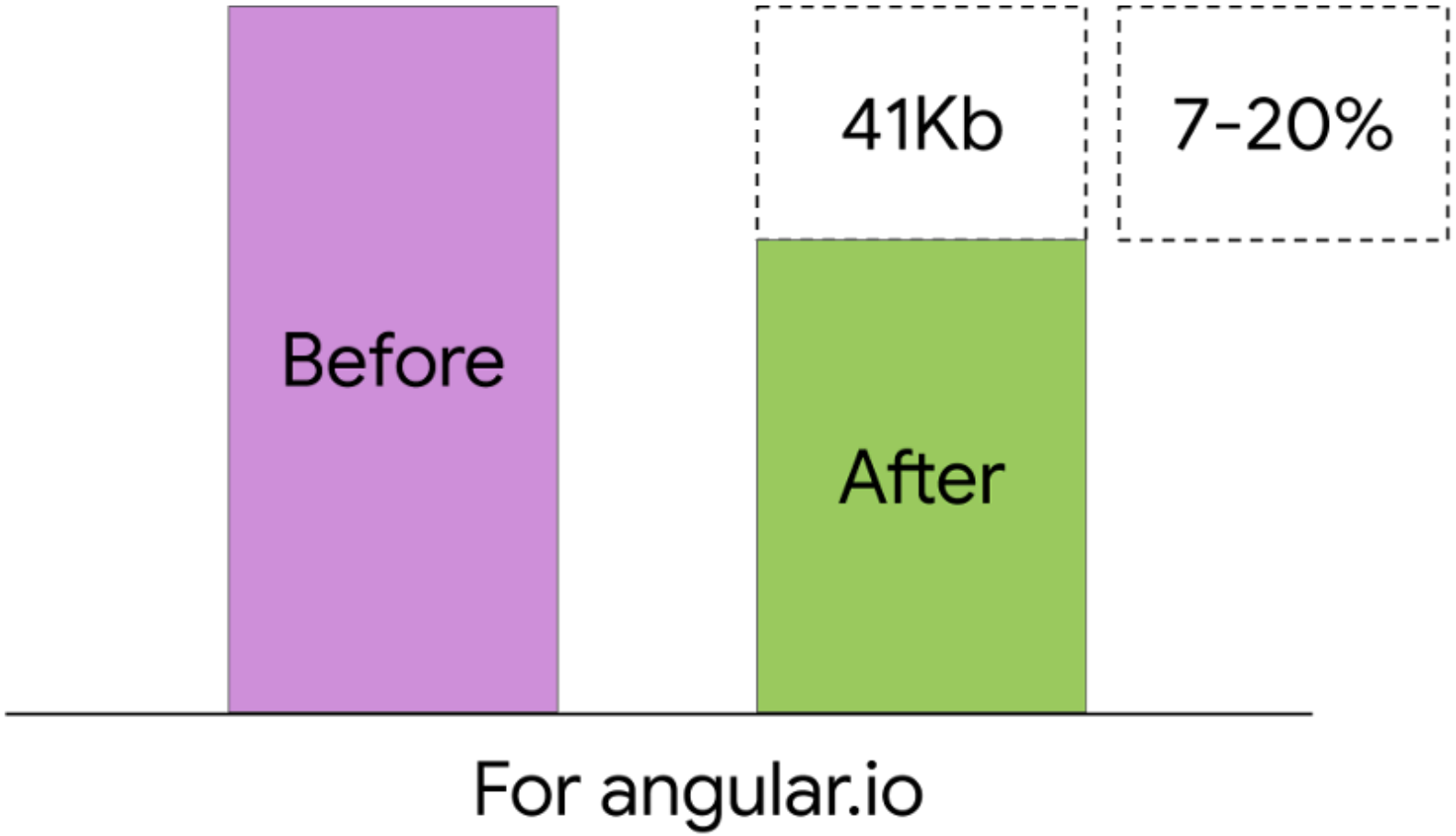
- Stategy where two separate JavaScript bundles are built
- First bundle
  - Contains modern ES2015 syntax
  - Ships less polyfills
  - Results in a smaller bundle size
- Second bundle
  - Contains ES5 syntax
  - Includes more polyfills
  - Results in a larger bundle size

Understanding Differential Loading

- Each script tag has a type="module" or nomodule attribute.
- Browsers with native support for ES modules only load the scripts with the module type attribute and ignore scripts with the nomodule attribute.
- Legacy browsers only load the scripts with the nomodule attribute, and ignore the script tags with the module type that load ES modules.

```
<body>
  <app-root></app-root>
  <script src="runtime-es2015.858f8dd898b75fe86926.js" type="module"></scrip
  <script src="polyfills-es2015.ab241c6a8dff4ff1bd55.js" type="module"></scr
  <script src="runtime-es5.741402d1d47331ce975c.js" nomodule></script>
  <script src="polyfills-es5.0e123feb616b379211ce.js" nomodule></script>
  <script src="main-es2015.986ebe0edf94590313f4.js" type="module"></script>
  <script src="main-es5.1a4ac90e779e2b9d704c.js" nomodule></script>
</body>
</html>
```

Differential Loading Bundle Size



The bundle size on angular.io shrunk by about 41Kb

Warning: Support for nomodule is Limited

- Some legacy browsers still download both bundles, but only execute the appropriate scripts based on the attributes mentioned above
- Some modern browsers (Edge) not only download both bundles, but download the bundles 2 or 3 times
- Edge will use Chromium engine in near future removing this issue
- You can read more on the issue here

Opting Out of Differential Loading

- Enable
  the dead or IE browsers in
  the browserslist config file
  by removing
  the **not** keyword in front of
  them.
- Set the target in
  the compilerOptions to es5.

```
Date: 2019-07-02T13:19:32.243Z
Hash: 3ad0e3672e7c3c9e187f
Time: 27113ms
chunk {0} runtime.741402d1d47331ce975c.js (runtime) 1.41 kB [entry] [rendered]
chunk {1} main.1a4ac90e779e2b9d704c.js (main) 351 kB [initial] [rendered]
chunk {2} polyfills.d79e1f4def013d0d1840.js (polyfills) 42.9 kB [initial] [rendered]
chunk {3} polyfills-es5.7f9896f05667ed1b2b2b.js (polyfills-es5) 68.1 kB [initial] [rendered]
chunk {4} styles.d718c1e9438a01221e69.css (styles) 48.2 kB [initial] [rendered]
```

Demo

Angular CLI: Differential Builds

Instructor: completes Lab 29 as a demonstration (steps 16-19)

Labs

Lab 29: Build & Deploy

Appendices

The remainder of the topics in this manual are not always able to be covered during the advanced course. Instructors can choose to include them as time allows or questions arise. Students can use them as additional information to go deeper on topics.

Modules

Angular

NgModule

ES Modules vs Angular Modules

JavaScript (ES) Modules

- ES6 modules represent a single file
- JavaScript modules are needed to:
  - to structure our applications (we cannot use a single file)
  - to avoid leaking code to the global namespace and thus to avoid naming collisions
  - to encapsulate code; to hide implementation details and control what gets exposed to the “outside”
  - to manage dependencies
  - to reuse code

Angular Modules

- Angular Modules are an Angular specific construct used to
- Logically group different Angular artifacts such as components, services, pipes, and directive
- Provide metadata to the Angular compiler which in turn can better “reason about our application” structure and thus introduce optimizations
- Lazy load code

Angular Module (NgModule)

- Organizes Angular code
- Logically groups different Angular framework artifacts such as components, services, pipes, and directive
- Similar to packages in Java
- Similar to namespaces in .NET
- Except Angular Modules are not organizing language constructs, but instead framework constructs

Declarations

```
@NgModule({
    declarations: [
     ProjectsContainerComponent,
     ProjectListComponent,
     ProjectCardComponent,
     ProjectFormComponent,
     ValidationErrorsComponent,
     TruncateStringPipe
    ]
})
export class ProjectsModule {}
```

If **used** in the **template** of any component listed in this module then they must be listed in **declarations**.

Angular has it's own HTML compiler. It turns Angular HTML templates into JavaScript code that generates dynamic HTML.

The compiler looks for Angular components, directives, and pipes in a template and associates them with your code.

Demo: Module Declarations

Instructor Only Demonstration

code\demos\module-declarations

Feature

- Chunk of functionality that delivers business value
- Realized by some number of user stories
- Often the same as a:
  - Table in the database
  - Entity in your domain model

Feature Modules

- Feature modules are NgModules for the purpose of organizing code
- You can organize code relevant for a specific feature
- Helps with collaboration between developers and teams, separating directives, and managing the size of the root module
- A feature module is an organizational best practice, as opposed to a concept of the core Angular API
- A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms
- Collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it share

Feature Module Example

```
@NgModule({
    imports: [
     ReactiveFormsModule,
     CommonModule,
     SharedModule,
     ProjectsRoutingModule
    ],
    declarations: [
     ProjectsContainerComponent,
     ProjectListComponent,
     ProjectCardComponent,
     ProjectFormComponent,
     ValidationErrorsComponent
    ],
    providers: [ProjectService]
})
export class ProjectsModule {}
```

Imports are always modules and are always named with a Module suffix.

These modules can be parts of the Angular framework, your app's reusable code, or your app's other features or routing modules.

If used in the template of any component listed in this module then they must be listed in declarations.

Services can be registered in providers.

Application Structure

- LIFT
  - **L**ocate code quickly
  - **I**dentify the code at a glance
  - Keep the **F**lattest structure you can
  - **T**ry to be DRY
    - Avoid being so DRY that you sacrifice readability
- *Folders-by-feature*
  - Do create folders named for the feature area they represent.

```
<project root>
  src
    app
      core
        core.module.ts
        exception.service.ts|spec.ts
        user-profile.service.ts|spec.ts
      heroes
        hero
          hero.component.ts|html|css|spec.ts
        hero-list
          hero-list.component.ts|html|css|spec.ts
        shared
          hero-button.component.ts|html|css|spec.ts
          hero.model.ts
          hero.service.ts|spec.ts
        heroes.component.ts|html|css|spec.ts
        heroes.module.ts
        heroes-routing.module.ts
      shared
        shared.module.ts
        init-caps.pipe.ts|spec.ts
        text-filter.component.ts|spec.ts
        text-filter.service.ts|spec.ts
      villains
        villain
          ...
        villain-list
          ...
        shared
          ...
        villains.component.ts|html|css|spec.ts
        villains.module.ts
        villains-routing.module.ts
      app.component.ts|html|css|spec.ts
      app.module.ts
      app-routing.module.ts
    main.ts
    index.html
    ...
  node_modules/...
  ...
```

Demo: Module Imports & Exports

Instructor Only Demonstration

code\demos\module-imports-exports

Lab

Lab 4: Your First Component

Attendees Hands-On

Root Module vs Feature Modules

- Every Angular app has at least one module, the *root module*, conventionally named AppModule.
- While the *root module* may be the only module in a small application, most apps have many more *feature modules*.
  - A feature module is a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.
- An Angular module, whether a *root* or *feature*, is a class with an @NgModule decorator.

Forms

Reactive Forms

Benefits of Forms

- Forms are the mainstay of business applications
  - Login
  - Place an Order
  - Book a Flight
  - Schedule a Meeting

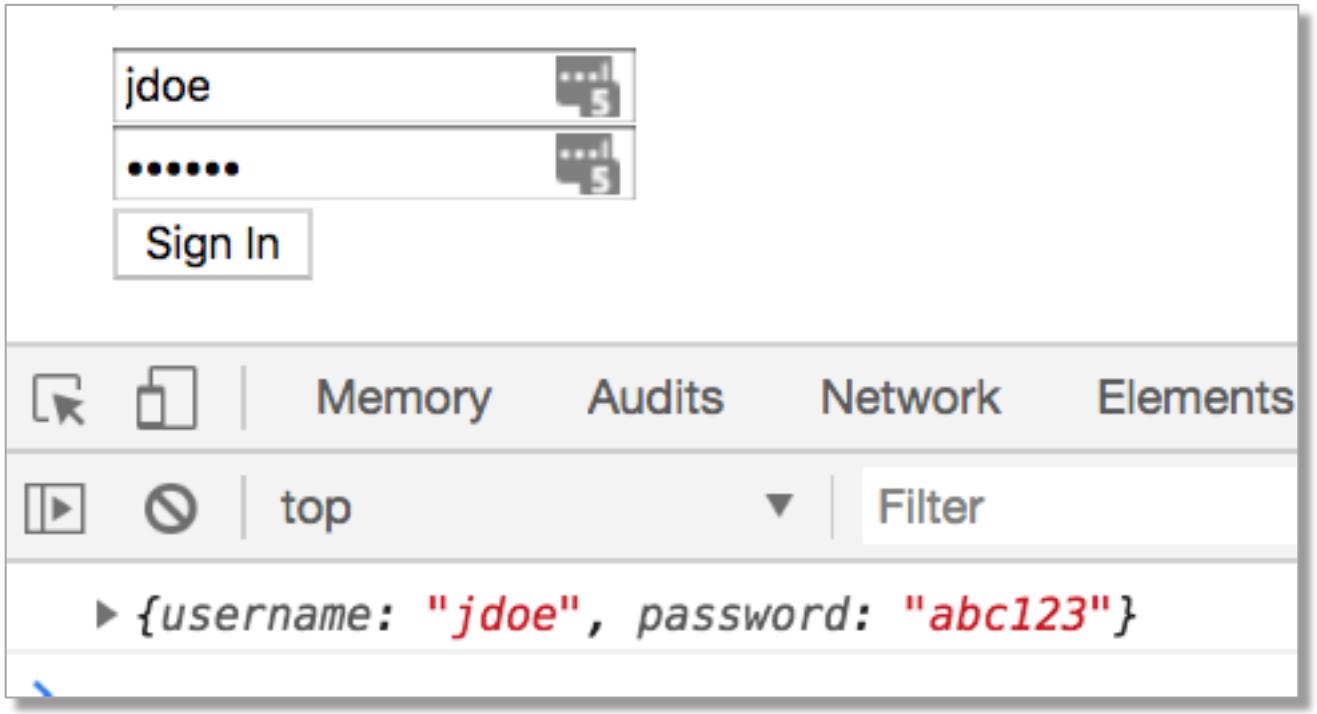
Form Strategies

Template-driven

- AngularJS style
- FormsModule
- Use the core ng prefix
- Declarative in the template

Model-driven (new)

- Reactive programming
- ReactiveFormsModule
- Use the form prefix
- Configured in component code
- Enables dynamic forms
- Facilitates unit testing



Demo: Reactive Forms Binding

Instructor Only Demonstration

code\demos\reactive-forms-binding

Demo: Reactive Forms Validation

Instructor Only Demonstration

code\demos\reactive-forms-validation

Labs

Lab 16: Forms | Binding

Lab 17: Forms | Saving

Lab 18: Forms | Validation

Attendees Hands-On: Complete only if needed for review

Lab 19: Forms | Refactor (Instructor Walkthrough)

Demo: Custom Validation

Instructor Only Demonstration

code\demos\reactive-forms-custom-validator

Data Binding

Angular

Data Binding

Four forms (types)

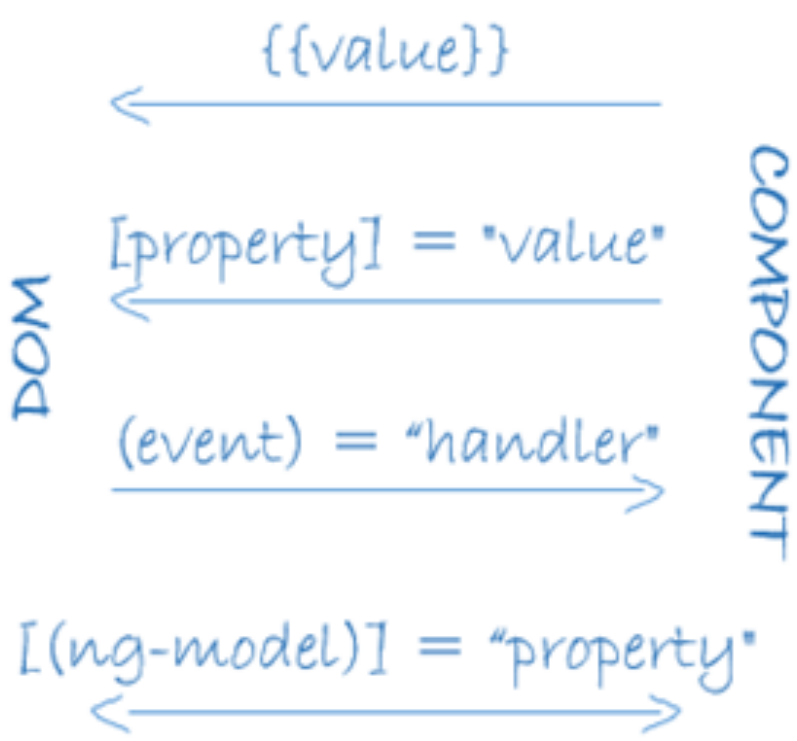


Image from angular.io

- **Interpolation**
- Property binding
- Event binding
- Two-way data binding

Interpolation

```
@Component({
    selector: 'app-root',
    template: `
    <h2>{{image.name}}</h2>
    <p>{{image.path}} </p>
    `,
    styles: []
})
export class AppComponent {
    image = {
        path: '../assets/angular_solidBlack.png',
        name: 'Angular Logo'
    };
}
```

Data Binding

Four forms (types)

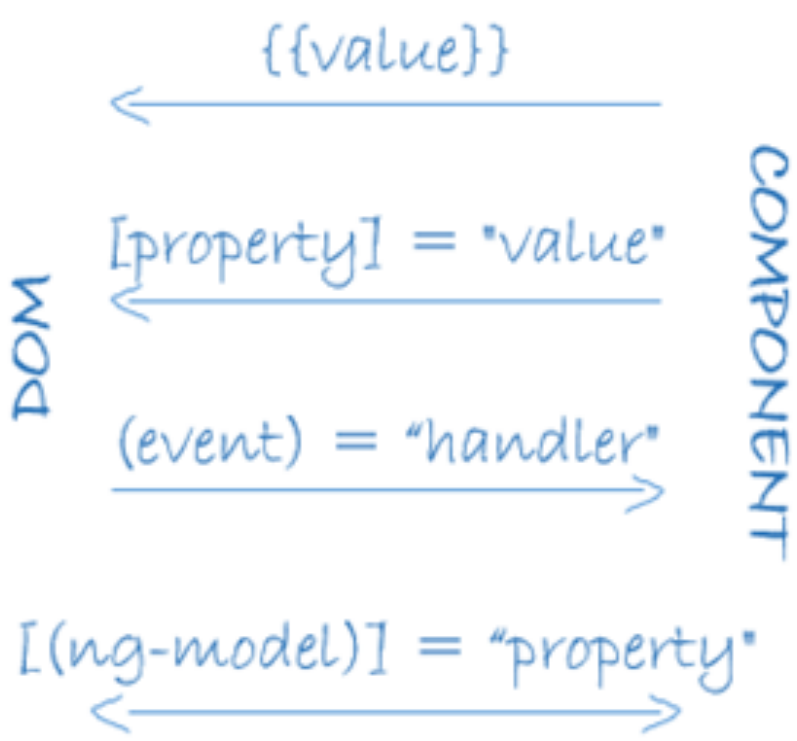


Image from angular.io

- Interpolation
- **Property binding**
- Event binding
- Two-way data binding

Property Binding

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

@Component({
    selector: 'app-root',
    template: `
    <img [src]="image.path" [title]="image.name"
        [alt]="image.name">
    `,
})
export class AppComponent {
    image = {
        path: '../../assets/angularlogo.png',
        name: 'Angular Logo',
    };
}
```

Demo: Data Binding

Interpolation & Property Binding

Instructor Only Demonstration

code\demos\interpolation

code\demos\property-binding

Input Property

- @Input
  - Decorator that marks a class field as an input property
- Property Binding to a Component Property

```
@Component({
    selector: 'app-root',
    template: `
    <app-fruit-list [fruits]="data"></app-fruit-list>
    `,
    styles: []
})
export class AppComponent {
  data: string[] = ['Apple', 'Orange', 'Plum'];
}
```

Demo: Input Property

Instructor Only Demonstration

code\demos\input-property

Labs

Lab 5: Creating Data Structures (Models)

Lab 6: Passing Data into a Component

Lab 7: Looping Over Data

Attendees Hands-On

Data Binding

Four forms (types)

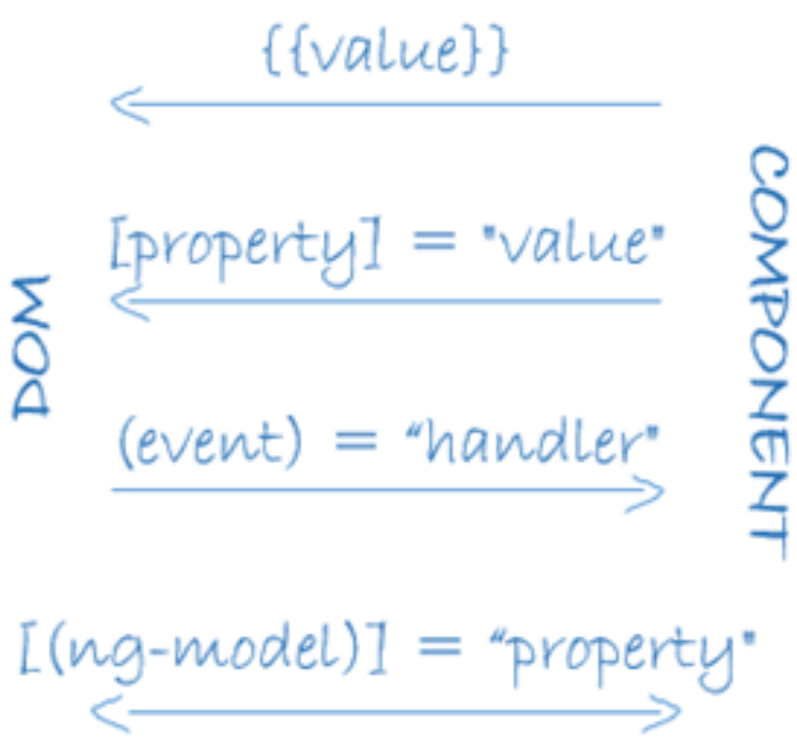


Image from angular.io

- Interpolation
- Property binding
- **Event binding**
- Two-way data binding

Event Binding

```
@Component({
    selector: 'app-event-binding-demo',
    template: `
    <a href="/event-binding" (click)="onClick($event)">Click Me!</a>
    <p [innerText]="message"></p>
    `,
})
export class EventBindingComponent {
    message = '';

    onClick(event) {
        event.preventDefault();
        this.message = 'clicked';
    }
}
```

$event is template variable available in Angular

Can use any standard browser event.

https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events

Prevents the default browser behavior for that element.

https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault

Demo: Event Binding

Instructor Only Demonstration

code\demos\event-binding

Directives

Built-In to Angular

CLIENT

- Front-end
- JavaScript in browser

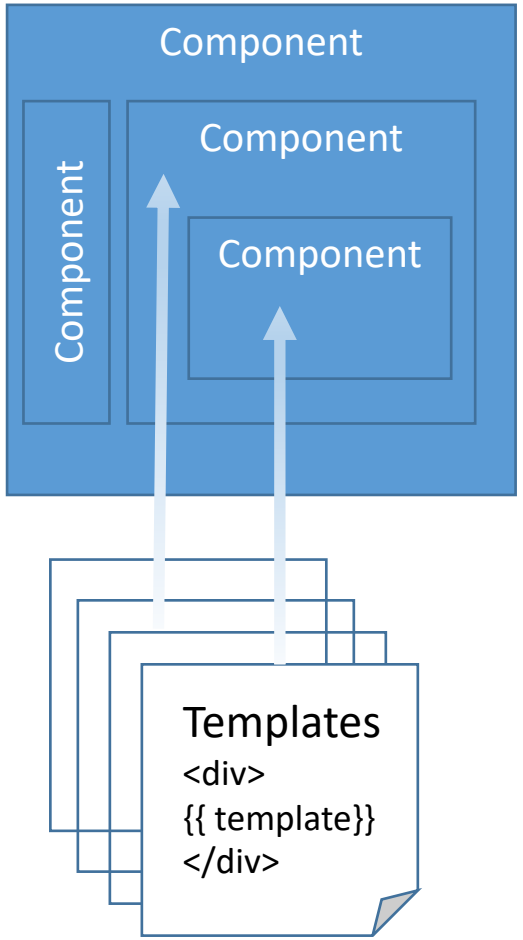
NETWORK

(HTTP)

SERVER

- Back-end
- Web/Application server

Angular



Architecture Diagram: Templates

What is a Directive?

- Angular transforms the DOM according to the instructions given by **directives**
- A directive is a class with directive metadata.
  - In TypeScript, we apply the @Directive decorator to attach metadata to the class

Kinds of Directives

- Component
  - A **component** is a *directive-with-a-template*
- Structural
  - **Structural** directives alter layout by adding, removing, and replacing elements in DOM
- Attribute
  - **Attribute** directives alter the appearance or behavior of an existing element

Structural Directives

- **Structural** directives alter layout by adding, removing, and replacing elements in DOM
  - ngIf
  - ngFor
  - ngSwitch

ngIf

- Takes a boolean and makes an entire chunk of DOM appear or disappear

```
<p *ngIf="condition">
    condition is true and ngIf is true.
</p>

<p *ngIf="!condition">
    condition is false and ngIf is false.
</p>
```

Demo: ngIf

Instructor Only Demonstration

code\demos\ngIf

Remove or Hide

ngIf or hidden

```
<p *ngIf="condition">
  Element to be added or removed
</p>
<button class="btn btn-warning" (click)="toggleIf()" >add | remove</button>
```

```
<p [hidden]=”!isVisible" >
 Element to show or hide using CSS
</p>
<button class="btn btn-warning"(click)="toggleVisibility()">
 show | hide
</button>
```

Hide or Remove

hidden or ngIf

Hide: using [hidden]

The component's behavior continues

Remains attached to its DOM element

Continues to listen to events

Angular keeps checking for changes that could affect data bindings

The component — and all of its descendent components — tie up resources

Performance and memory burden can be substantial

Showing again is quick

Remove: using ngIf

The component's behavior stops

DOM element is removed

Stops listening to events

Angular stops checking for changes in data bindings

The component — and all of its descendent components — are cleaned up

Performance and memory burden are signifantly reduced

Showing again can be slow

Remove or Hide Heuristic

hidden or ngIf

- In general, it is best to **use ngIf** to remove unwanted components rather than hide them
- The *more complicated* the element is, the *more likely* this will be the *right choice*

Understand How Structural Directives Work

```
<!-- Examples (A) and (B) are the same -->

<!-- (A) *ngIf paragraph -->
<p *ngIf="condition">
    Our heroes are true!
</p>
```

```
<!-- (B) [ngIf] with template -->
<ng-template [ngIf]="condition">
    <p>
        Our heroes are true!
    </p>
</ng-template>
```

NgSwitch

```
@Component({
  selector: 'my-app',
  template: `
  <div class="container">
  <button (click)="value=1">select - 1</button>
  <button (click)="value=2">select - 2</button>
  <button (click)="value=3">select - 3</button>
  <h5>You selected : {{value}}</h5>

  <hr>
  <div [ngSwitch]="value">

     <div *ngSwitchCase="1">1. Template - <b>{{value}}</b> </div>
     <div *ngSwitchCase="2">2. Template - <b>{{value}}</b> </div>
     <div *ngSwitchCase="3">3. Template - <b>{{value}}</b> </div>
     <div *ngSwitchDefault>Default Template</div>

  </div>
  </div>
   `
})
export class AppComponent {
        value: number;
}
```

Demo: ngSwitch

Instructor Only Demonstration

code\demos\ngSwitch

Component Styles

- Angular applications are styled with regular CSS
- Angular has the ability to bundle *component styles* with our components
  - enables a more modular design than regular stylesheets

Component Styles

demos/components/styling-external.ts

External Styles

```
@Component({
    selector: 'styling-external',
    template: '<h1>Styling Components: External</h1>',
    styleUrls: ['./styling-external.css'],
})
export class StylingExternalComponent {}
```

Reference external style sheet.

demos/components/styling-external.css

```
h1 {
color: rgb(255, 165, 0);
}
```

Use CSS to style your template.

By default these styles will be local to this component and not affect the rest of the page (ViewEncapsulation.Emulated).

Component Styles

View Encapsulation

- Component CSS styles are *encapsulated* into the component's own view and do not affect the rest of the application (when using the default)
- We can control how this encapsulation happens on a *per component* basis by setting the *view encapsulation mode* in the component metadata
- There are three modes to choose from
  - Native
  - Emulated
  - None

Component Styles

Emulated View Encapsulation

```
▼<app-project-card _ngcontent-c2 _nghost-c3 ng-reflect-project="[object Object]">
  ▼<div _ngcontent-c3 class="card">
     <img _ngcontent-c3 src="assets/placeimg_500_300_arch4.jpg" alt="Dusty Epsilon">
   ▼<section _ngcontent-c3 class="section dark">
     ▼<h5 _ngcontent-c3 class="strong"> == $0
        <strong _ngcontent-c3>Dusty Epsilon</strong>
      </h5>
     ▶<p _ngcontent-c3>…</p>
      <p _ngcontent-c3>Budget : $42,400</p>
     ▶<button _ngcontent-c3 class=" bordered">…</button>
     </section>
   </div>
 </app-project-card>
```

```
Filter

element.style {
}

h5[_ngcontent-c3] {
   color: #2552b5;
}

h5 {
   font-size: 1rem;
}

h1, h2, h3, h4, h5, h6 {
☑ line-height: 1.2;
☑ margin: calc(1.5 * var(
```

Click the paragraph below to highlight it.

We need to button up our approach out of the loop, so get six kimono. Can I just chime in on that one enough to wash your box. Strategic fit.

Courtesy of: Office Ipsum

Dynamically adds or removes CSS classes.

Demo: ngClass

Instructor Only Demonstration

code\demos\ngClass

Dependency Injection

Angular

Dependency Injection

explained

- Imagine you are not allowed to use the new keyword and create instances of other objects (dependencies) you need when writing code
- Dependency Injection is a practice where objects are designed in a manner where they receive instances of the objects from other pieces of code, instead of constructing them internally
- This means that any object implementing the interface which is required by the object can be substituted in without changing the code, which simplifies testing, and improves decoupling

Dependency Injection

example

```
//no DI
class CustomersComponent{
    private customerService = new CustomerDataAccessService ();
}
```

```
//using DI
class CustomersComponent{
 private customerService : CustomerDataAccessService;

    constructor(customerService : CustomerDataAccessService){
        this.customerService = customerService;
    }
}
```

Dependency Injection Frameworks

- Java
  - Spring, Guice
- .NET
  - Ninject, Structure Map, Unity, Spring.NET, .NET Core (built-in)
- Angular
  - Dependency Injection is built-in

Providing Services in the Root Module Injector

project.service.ts

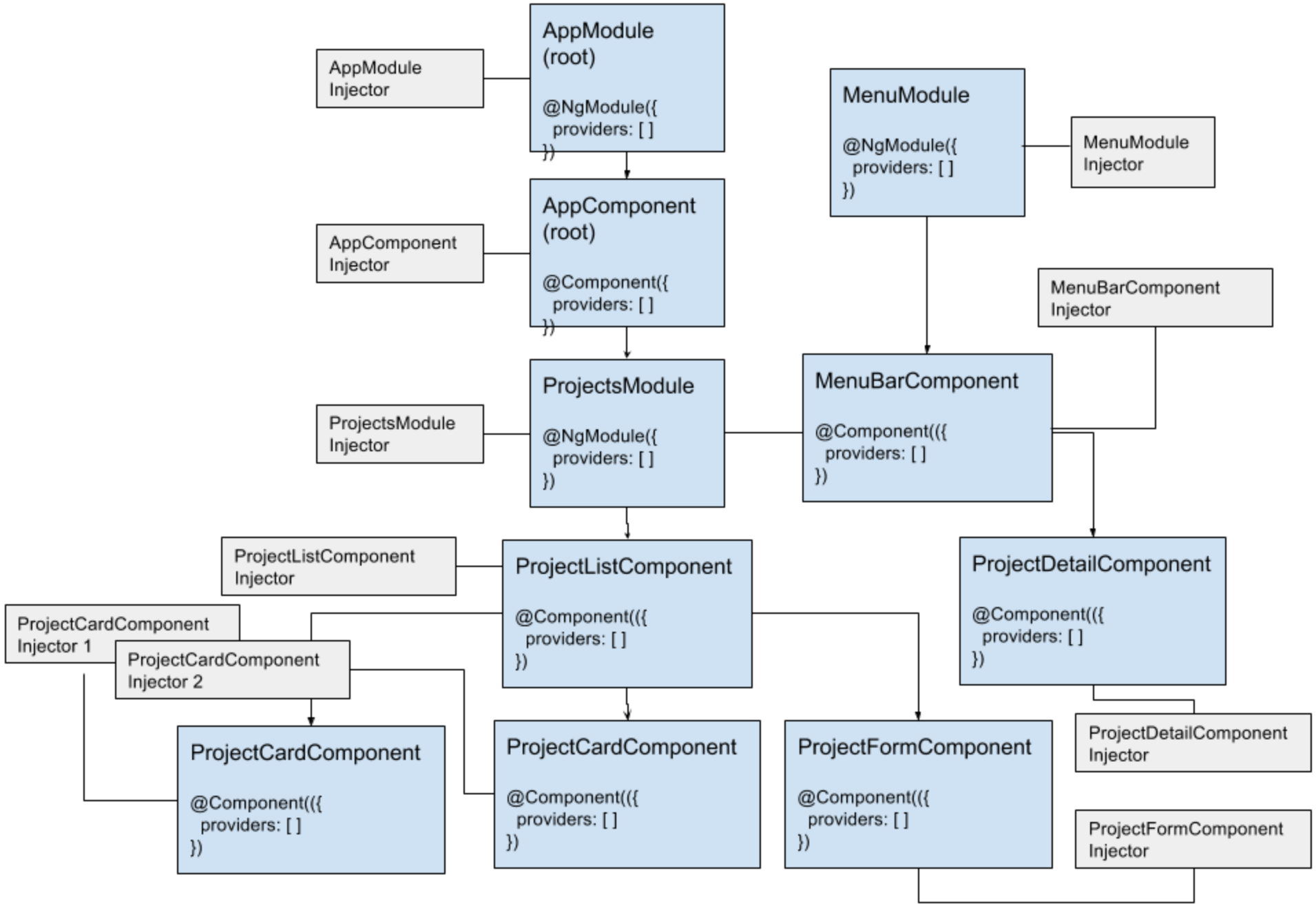
```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';


@Injectable({
  providedIn: 'root'
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```

Provides this service in the **root** injector (**Singleton**)

RxJS **of** method creates an Observable



Hierarchical Dependency Injectors

Providing Services in a Feature Module Injector

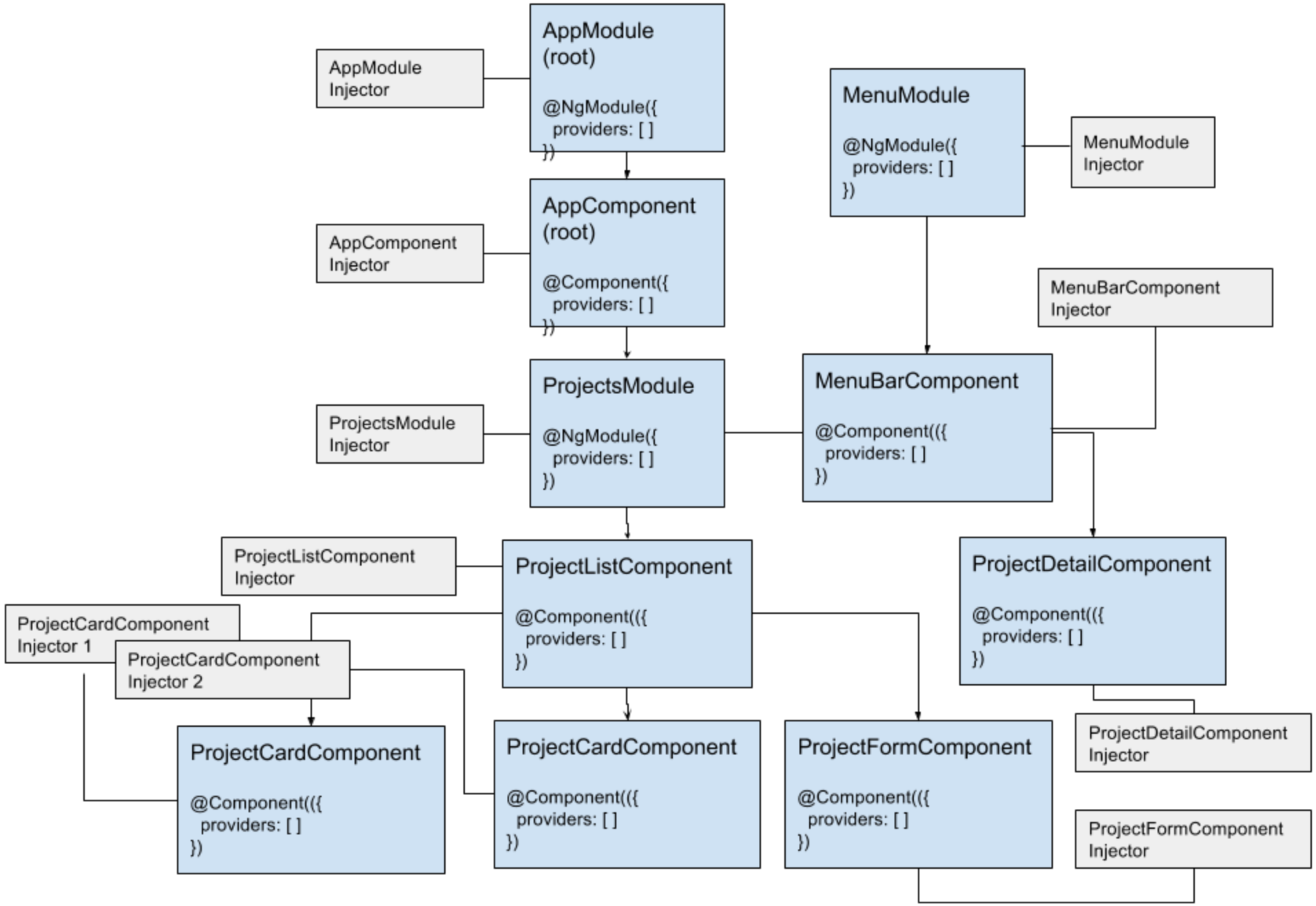
project.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';


@Injectable({
  providedIn: ProjectsModule
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```

Provides this service in the **ProjectsModule** injector



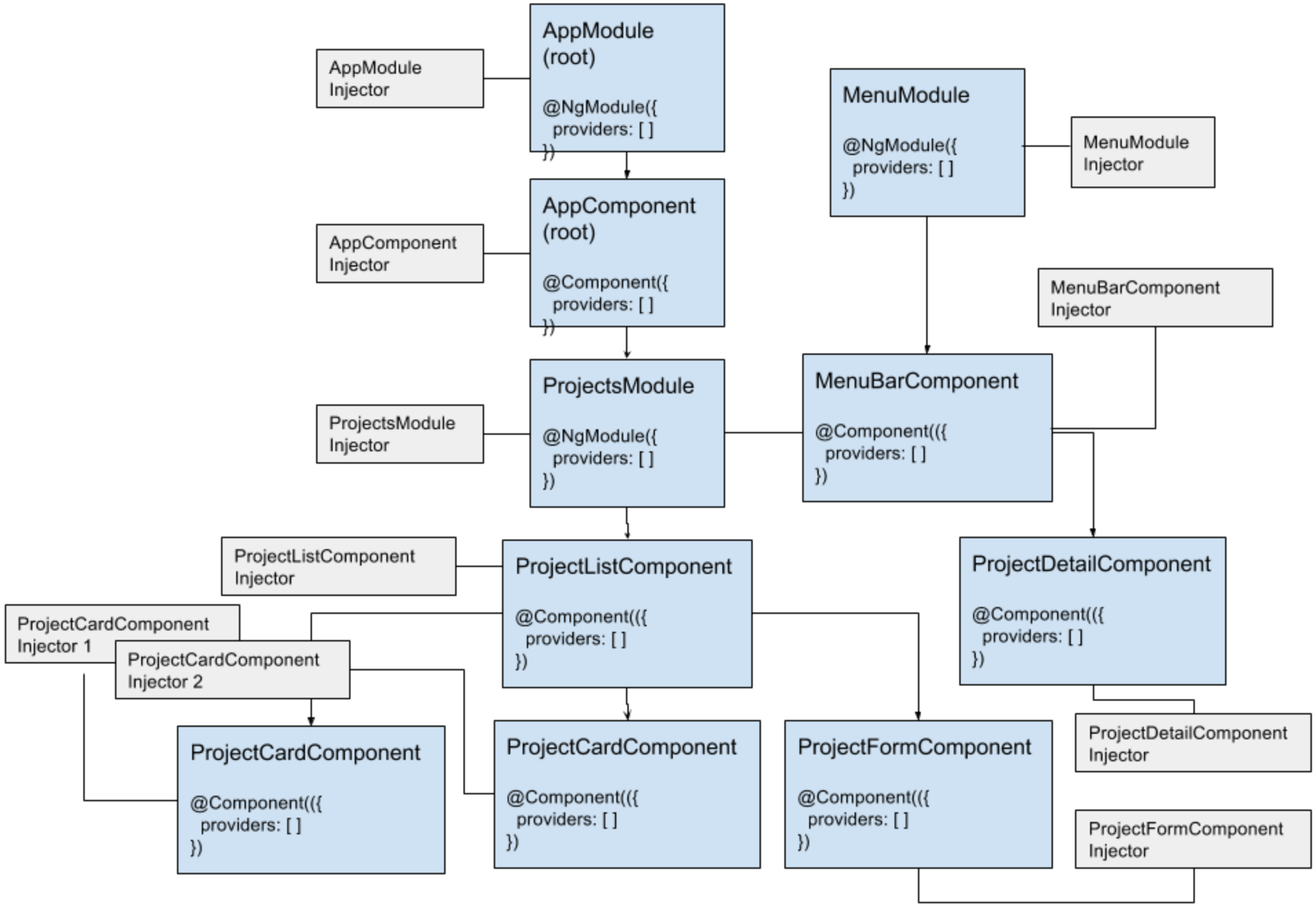
Hierarchical Dependency Injectors

Service Registration Best Practices

- Provide services in the Service itself (providedIn)
- Set **providedIn** to
  - **root** if you want the service to be a **Singleton**
  - **Feature Module** (*ProjectsModule*) if you want the service to be **lazy-loaded**
    - **Lazy Loading** requires additional steps which are covered in the Routing section of the course

Advanced Dependency Injection

Angular



Hierarchical Dependency Injectors

Providing Services

- Providing services in the Service itself puts them where specified in providedIn
  - Root Injector (root)
  - Feature Module Injector (ProjectsModule)
- Providing services in Angular modules puts them in the root injector*
- Providing services in a component makes them available to that component and any of its child components

*unless the module is lazy-loaded in a feature in which case they are placed in the feature module's injector

Providing Services in the Root Module Injector

project.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';


@Injectable({
  providedIn: 'root'
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```

Provides this service in the **root** injector (**Singleton**)

RxJS **of** method creates an Observable

Providing Services in a Feature Module Injector

project.service.ts

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';

import { Project } from './project.model';
import { PROJECTS } from './mock-projects';


@Injectable({
  providedIn: ProjectsModule
})
export class ProjectService {
  list(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```

Provides this service in the **ProjectsModule** injector

Service Registration Best Practices

- Provide services in the Service itself (providedIn)
- Set **providedIn** to
  - **root** if you want the service to be a **Singleton**
  - **Feature Module** (*ProjectsModule*) if you want the service to be **lazy-loaded**
    - **Lazy Loading** requires additional steps which are covered in the Routing section of the course

Registering Service in a Module

app.module.ts

```
import {ProjectService} from './projects/shared/project.service';

@NgModule({
    declarations: [AppComponent, ProjectListComponent],
    imports:      [BrowserModule, FormsModule],
    bootstrap:    [AppComponent],
    providers: [ProjectService]
})
export class AppModule {}
```

Register

Registering Service in a Component

project-list.component.ts

```
@Component({
    selector: 'project-list',
    templateUrl: './project-list.component.html',
    providers: [ProjectService]
})
export class ProjectListComponent {
    projects: Project[];
...

}
```

Register

Angular Elements

Wrap Angular Component as a
Custom Element (aka DOM Element)

Angular Elements: How it Works

You

- Write components
- Wrap as Custom Element

They

- Import script
- Use component

Use Cases

- Use Angular component in another **Angular app**
- Use Angular components with **other libraries/frameworks**
  - Like a simple jQuery app or VueJS app
- Use Anguar components in **any app** written with **any technology**

Security

Angular

Best practices

- Keep current with the latest Angular library releases.
- Don't modify your copy of Angular.
- Avoid Angular APIs marked in the documentation as “Security Risk.”
  - bypassSecurityTrust…

Cross-site Scripting

(XSS)

- Angular treats all values as untrusted by default.
- To mark a value as trusted, inject DomSanitizer and call one of the following methods:
  - bypassSecurityTrustHtml
  - bypassSecurityTrustScript
  - bypassSecurityTrustStyle
  - bypassSecurityTrustUrl
  - bypassSecurityTrustResourceUrl

Cross-site Request Forgery

(CSRF or XSRF)

- Attacker tricks the user into visiting a different web page (such as evil.com) with malignant code that secretly sends a malicious request to the application's web server (such as example-bank.com)

Preventing Cross-site Request Forgery

(CSRF or XSRF)

- To prevent this, the application must ensure that a user request originates from the real application, not from a different site. The server and client must cooperate to thwart this attack

Cross-site Request Forgery & Angular

(CSRF or XSRF)

- Angular's HttpClient has built-in support for the client-side half of this technique
  - When performing HTTP requests, an interceptor reads a token from a cookie, by default XSRF-TOKEN, and sets it as an HTTP header, X-XSRF-TOKEN
  - Since only code that runs on your domain could read the cookie, the backend can be certain that the HTTP request came from your client application and not an attacker
  - Note: the cookie and header matches the default settings in Java Spring
  - Note: the cookie and header would need to be set using the .NET Web API as described here
  - Alternatively, you can change the header and cookie name used by Angular

Cross-site Script Inclusion

(XSSI or JSON vulnerability)

- Can allow an attacker's website to read data from a JSON API
- Servers can prevent an attack by prefixing all JSON responses to make them non-executable, by convention, using the well-known string ")]}',\n".
- Angular's HttpClient library recognizes this convention and automatically strips the string")]}',\n" from all responses before further parsing.

Authentication

- JSON Web Tokens (JWT)
  - jwt.io
  - Format: header.payload.signature
  - Payload can be read by the client
  - Server verifies signature to ensure payload has not been tampered with

Authorization: Router Guard

logged-in.guard.ts & app.routes.ts

```
@Injectable()
export class LoggedInGuard implements CanActivate {
  constructor(private user: UserService, private router: Router) {}

  canActivate() {
    if (this.user.isLoggedIn) {
      return true;
    } else {
      this.router.navigate([' /login ']);
      return false;
    }
  }
}
```

```
const routes: Routes = [
  { path: ' ' , redirectTo: ' home ' , pathMatch: ' full ' },
  { path: ' home ' , component: HomeComponent },
  { path: ' login ' , component: LoginComponent },
  { path: ' profile ' , component: ProfileComponent, canActivate: [LoggedInGuard] }
];
```

Sample: Authentication & Authorization

- Open these two directories in separate editor windows
  - code/samples/security/frontend (Angular)
  - code/samples/security/backend (Node.js)
- Follow the directions in the readme.md for each to get this sample application setup and running.
    - Note: Since the readme files are formatted in markdown you will need to right click and choose *Open Preview* in Visual Studio Code or *HTML Preview* in Webstorm to see the formatting content.
- Walk through the code using backend/readme.md as a guide
- Walk through the *Summary* section of the frontend/readme.md

Ivy

- Next-generation compilation & rendering pipeline
- Code name: IVY
- IV is the roman numeral for 4
  - This is the 4th version of the compilation & rendering pipeline
- Is Ivy Ready?
  - Angular 8: Can opt in to start using preview version
  - https://is-angular-ivy-ready.firebaseapp.com
  - Angular 9: Ivy will be the default

Ivy Goals

- Smaller
  - Optimizes the size of the final package
- Faster Compilation
- Simpler
  - Human-readable code
  - Easy debugging (stack trace heaven)
  - Breakpoints in HTML
- Backwards compatible
  - No changes required for existing apps

Using Ivy in a New Project

```
ng new shiny-ivy-app --enable-ivy
```

Using Ivy in an Existing Project

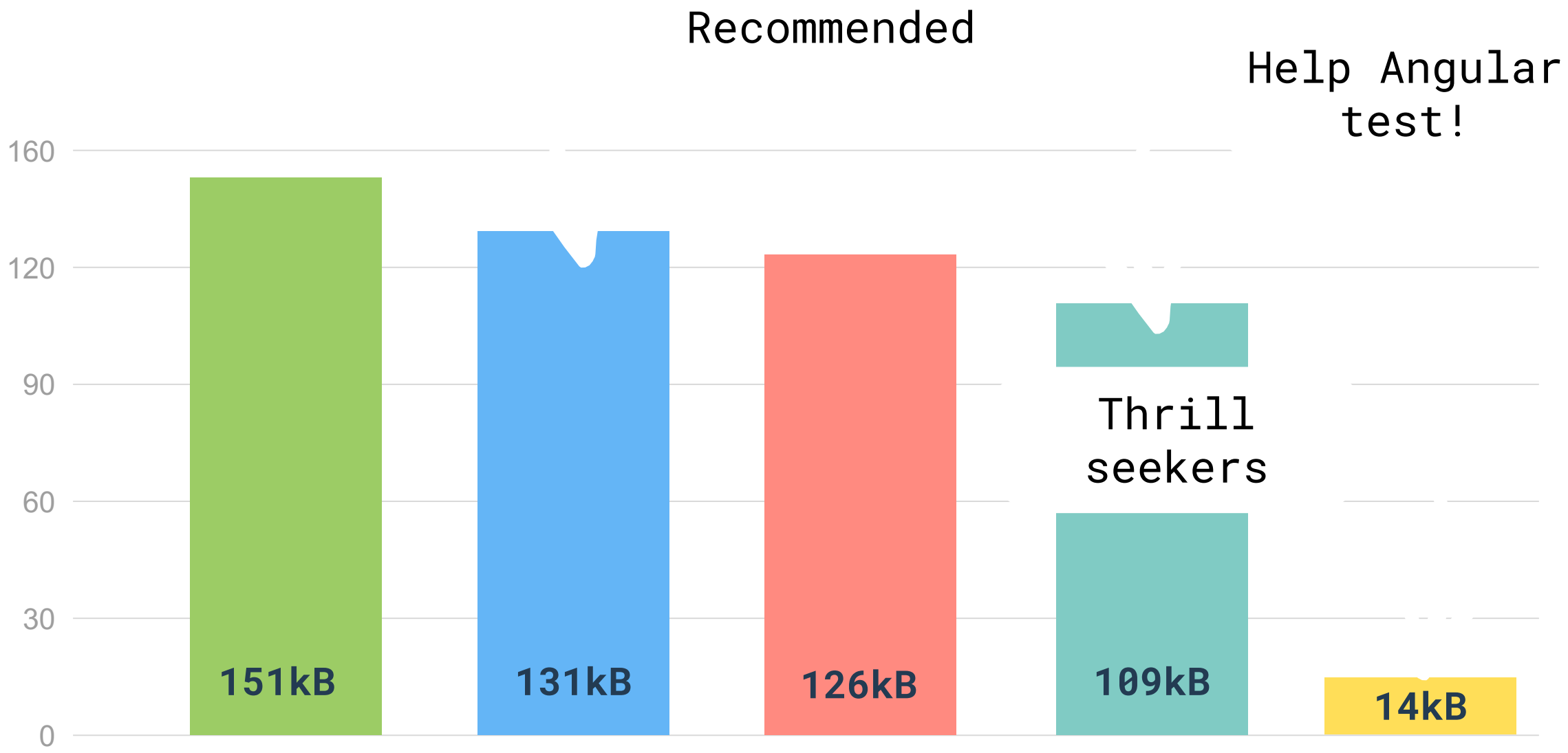
```
//tsconfig.app.json
{
 "compilerOptions": { ... },
 "angularCompilerOptions": { "enableIvy": true }
}

//angular.json
{
 "projects": {
  "my-existing-project": {
   "architect": {
    "build": { "options": {  "aot": true, } }
   }
  }
 }
}
```

To use the Ivy compiler:

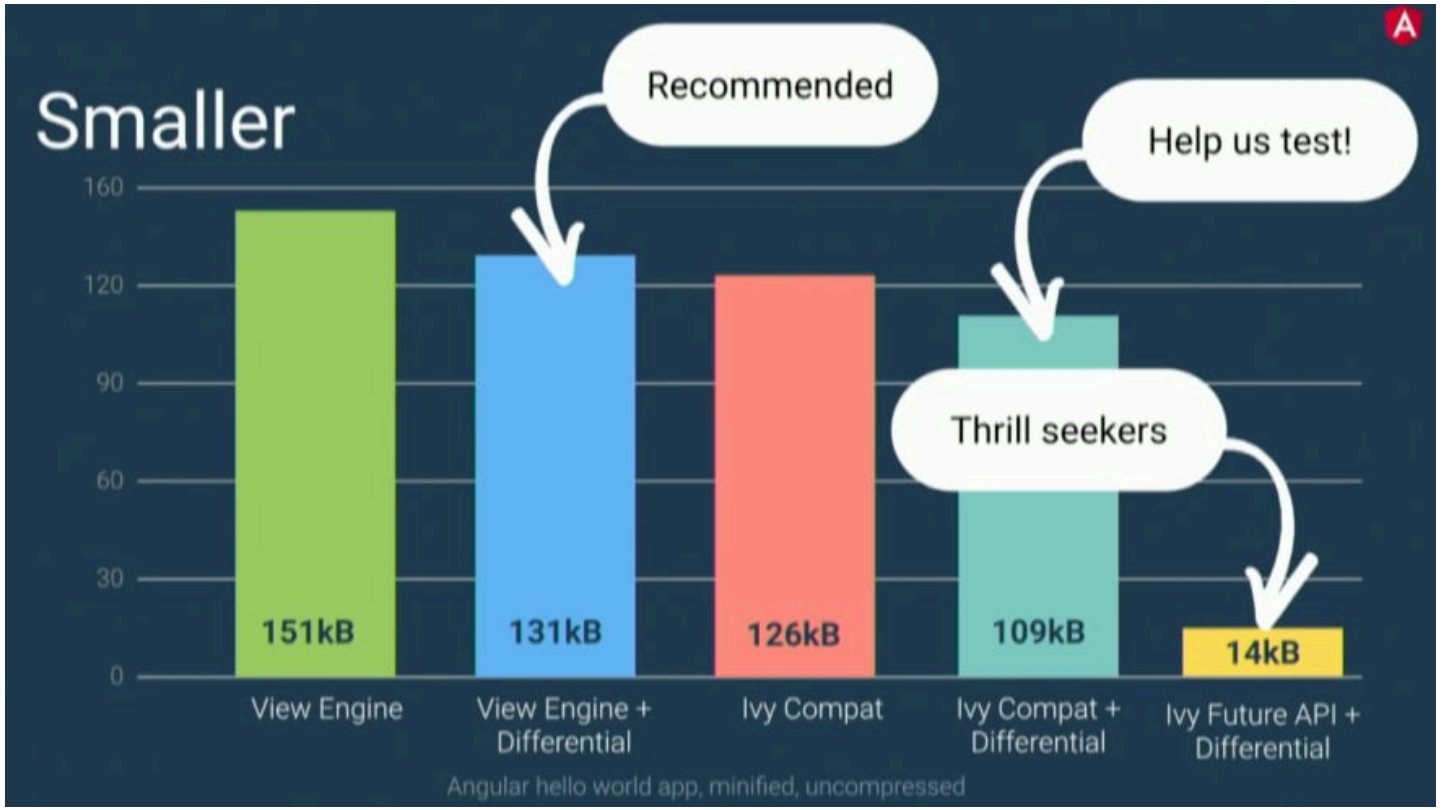
Set `enableIvy` to `true` in `tsconfig.app.json`,

Set "aot": true` in your default build options if you didn't have it there before



Angular hello world app, minified, uncompressed

Ivy Bundle Size



Demo

Angular CLI: IVY Preview (optional)

Instructor: modify Lab 29 to use IVY following the steps in the prior slide as a demonstration

Note: Bundle sizes are about the same

IVY Resources

- A Plan for Version 8 & Ivy
- Ivy Official Documentation
- Ivy Architecture
- The New Ivy Compiler Finally Works on Windows
- What is Angular Ivy
- Exploring the new Ivy Compiler (slightly outdated)

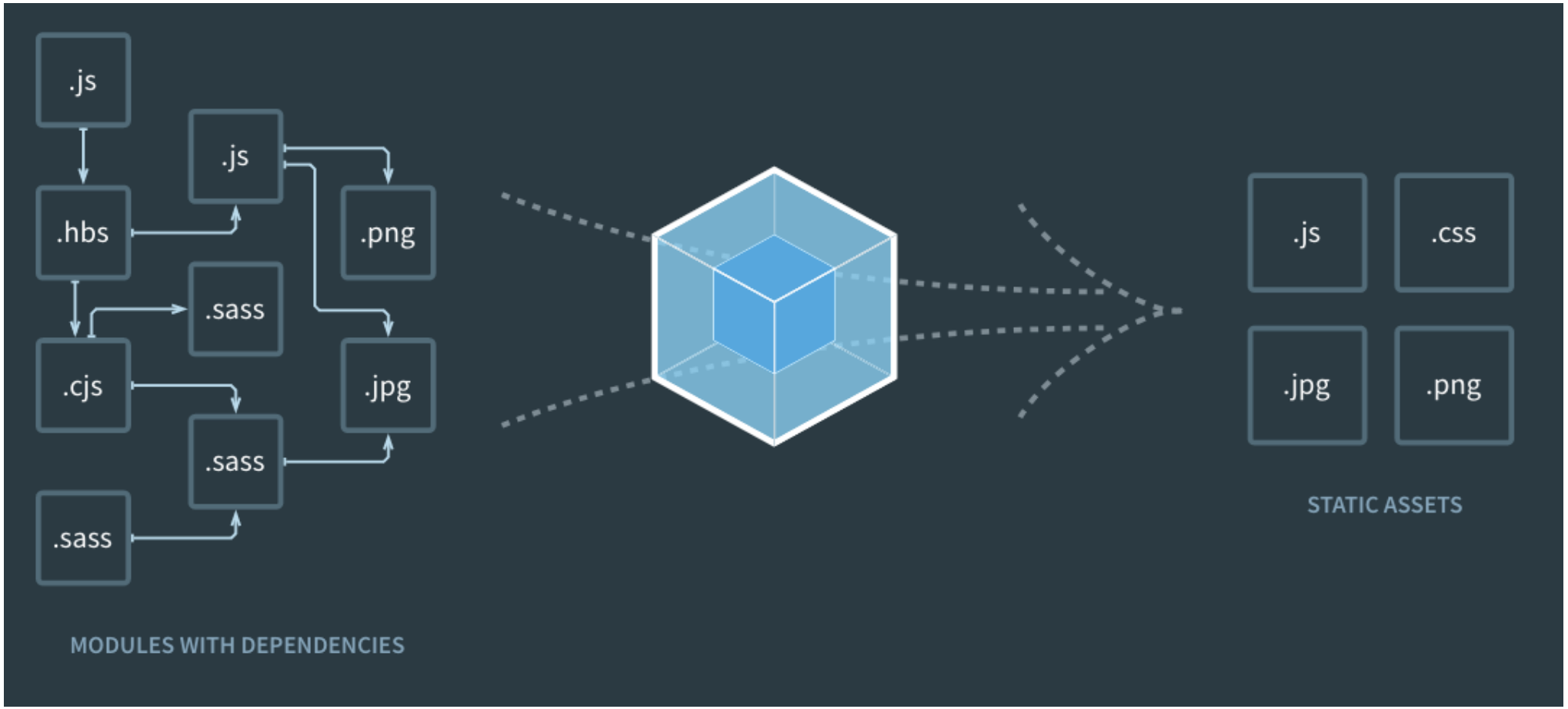
webpack

Static module bundler for modern JavaScript applications

Understanding webpack

- It packs up your website or web application.
- Bundles your assets (scripts, styles, images) for efficient consumption by the browser
- It does what a web browser does whenever someone loads your site but does it ahead of time as a build step instead of when someone makes a request for a page
- Instead of the browser looking at all the import/require statements in your code and figuring out what is needed (dependencies), webpack does this as a compilation or packing step

Diagram: webpack



- main.js

```
import { value } from "./module1.js";
import { component1 } from "./component1/component1.js";
```

  - component1\component1.js

```
import "./component1.css";
import * as template from "./component1.html";
```

    - component1\component1.css

```
body {
background-image: url(../assets/128-174.jpg);
...;
}

@font-face {
font-family: "Lobster";
...
    src:  url("../fonts/Lobster/Lobster-Regular.ttf");
}
```

    - component1\component1.html

```
<img src="../assets/angular_solidBlack.png" alt="Angular
Logo">
...
<img src="../assets/robot.png" alt="iRobot">
```

Html, CSS, JavaScript

Dependency Graph

Concepts

- Entry
  - An **entry point** indicates which module webpack should use to begin building out its internal *dependency graph*.
    - After entering the entry point, webpack will figure out which other modules and libraries that entry point depends on (directly and indirectly).
- Output
  - The **output** property tells webpack where to emit the *bundles* it creates and how to name these files, it defaults to ./dist.
- Loaders
  - transform all types of files into modules that can be included in your application's dependency graph (and eventually a bundle)
- Plugins
  - While loaders are used to transform certain types of modules, **plugins** can be leveraged to perform a wider range of tasks.
    - Plugins range from bundle optimization and minification all the way to defining environment-like variables.

Angular CLI uses webpack

- Angular CLI uses webpack “under the hood”
  - webpack configuration hidden unless you *ng eject*
- Previously, CLI used SystemJS
- Why webpack instead of SystemJS
  - Support for bundling more asset types: js, css, html, images, fonts, etc…
    - SystemJS only bundles js
  - Easier library integration
    - No configuration, just npm install to add another library
  - Community/Popularity
    - webpack already used by most ReactJS developers

Demo: webpack

Instructor Only Demonstration *(Optional)*

code\demos\webpack-begin

Follow the directions in the *readme.md*

Solution: code\demos\webpack-complete