Web Development

COMP 431 / COMP 531

Lecture 11: Angular Services and Routes

Instructor: Mack Joyner

Department of Computer Science, Rice University

mjoyner@rice.edu

http://www.clear.rice.edu/comp431

Recap

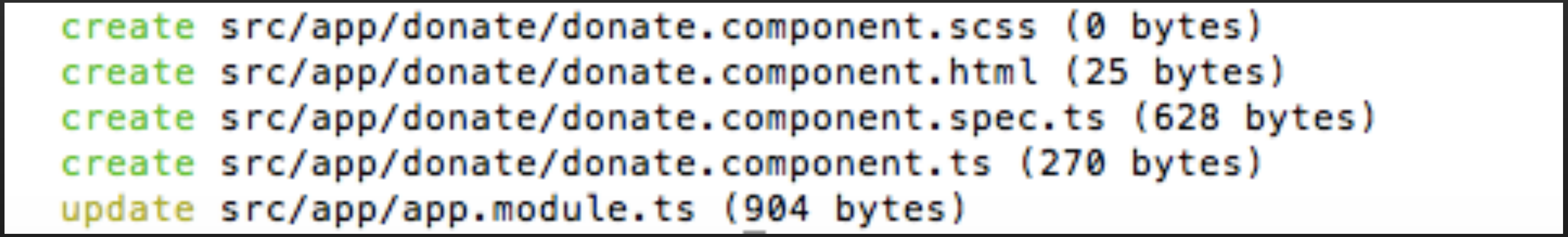
- HTML and HTML5, Storage, Canvas
- JavaScript and Scope
- Forms, CSS, Events
- jQuery, AJAX, and fetch
- Modern JS
- MVC

Homework Assignment 4
(Draft Front-end)
Due Thursday 10/12

Generating Angular Components

Generate new components is fast:

```
>> ng generate component donate
```



```
  create src/app/donate/donate.component.scss (0 bytes)
  create src/app/donate/donate.component.html (25 bytes)
  create src/app/donate/donate.component.spec.ts (628 bytes)
  create src/app/donate/donate.component.ts (270 bytes)
  update src/app/app.module.ts (904 bytes)
```

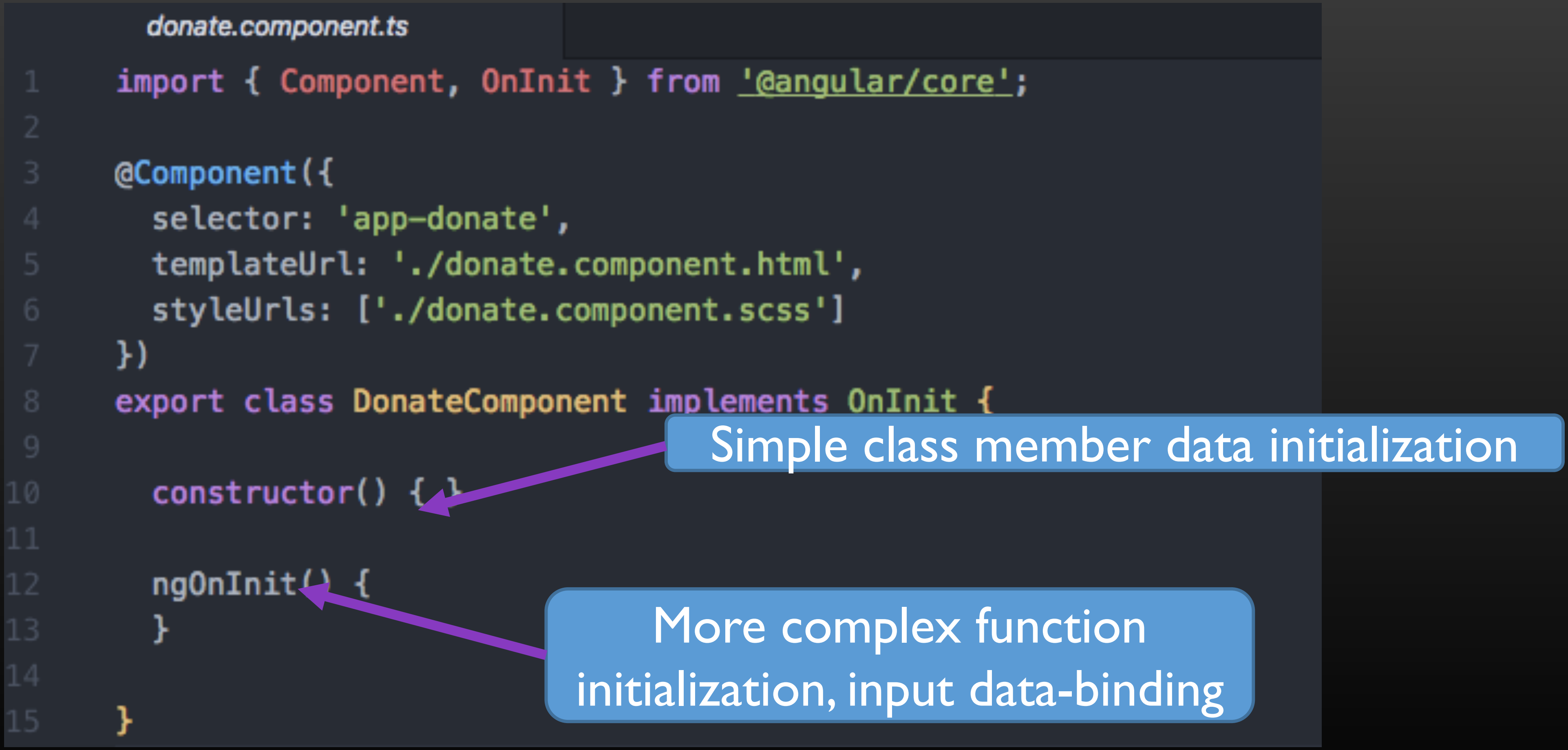
In-Class Exercise: Hello World

git commit /inclass-10/…

- Install Angular CLI: npm install –g @angular/cli
  - May need to use sudo
- Create a new application in git comp431_531 repo:

  ng new helloWorld –dir ./inclass-10
- Build and serve the application (view on http://localhost:4200)
  - cd to inclass-10
  - ng serve --open
- Change page to say "Hello, World!" (keep template, get started links)
- Enclose the interpolation ({{title}}) with a basic toolbar (md-toolbar)
  - Hint: May need to import another module
- Commit all files except node_module directory
  - Use git add, git commit –m "your descriptive message", and git push

Generating Angular Components


donate.component.ts

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

@Component({
  selector: 'app-donate',
  templateUrl: './donate.component.html',
  styleUrls: ['./donate.component.scss']
})
export class DonateComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Simple class member data initialization
More complex function initialization, input data-binding

Structural Directive

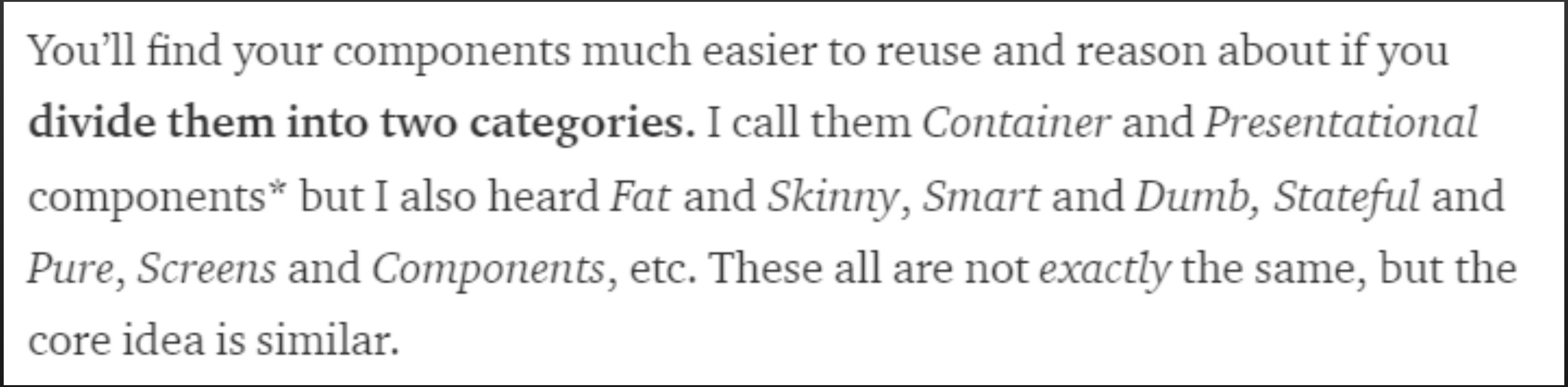
- Alter layout: add, remove, replace DOM elements
- <div *ngIf=“selectedProfile”> …<div>

Add div if not null

- <md-list-item *ngFor=”let pField of profile”>

List item for each field

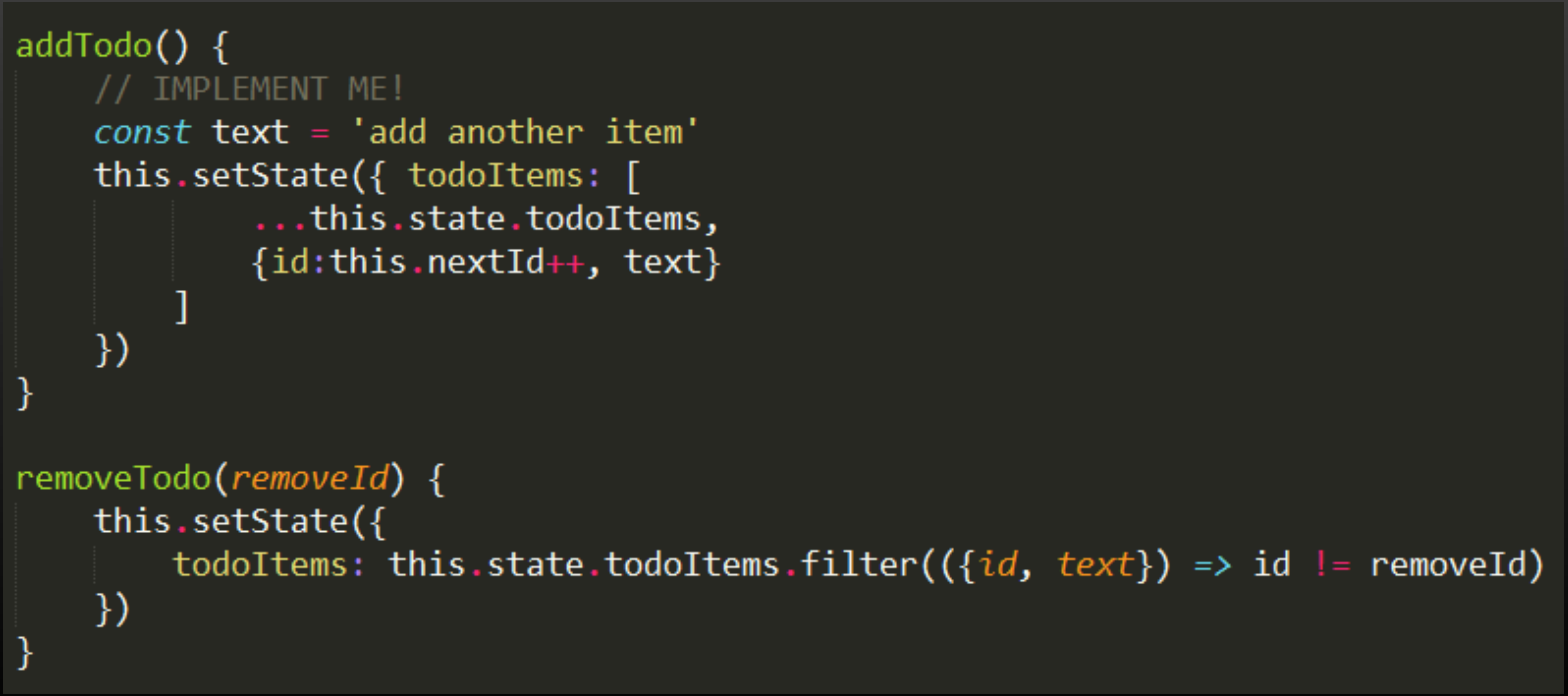
Separation of Concerns



You'll find your components much easier to reuse and reason about if you **divide them into two categories.** I call them *Container* and *Presentational* components* but I also heard *Fat* and *Skinny*, *Smart* and *Dumb*, *Stateful* and *Pure*, *Screens* and *Components*, etc. These all are not *exactly* the same, but the core idea is similar.

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.z20p1v3je

The Fat Component



```
addTodo() {
    // IMPLEMENT ME!
    const text = 'add another item'
    this.setState({ todoItems: [
            ...this.state.todoItems,
            {id:this.nextId++, text}
        ]
    })
}

removeTodo(removeId) {
    this.setState({
        todoItems: this.state.todoItems.filter(({id, text}) => id != removeId)
    })
}
```

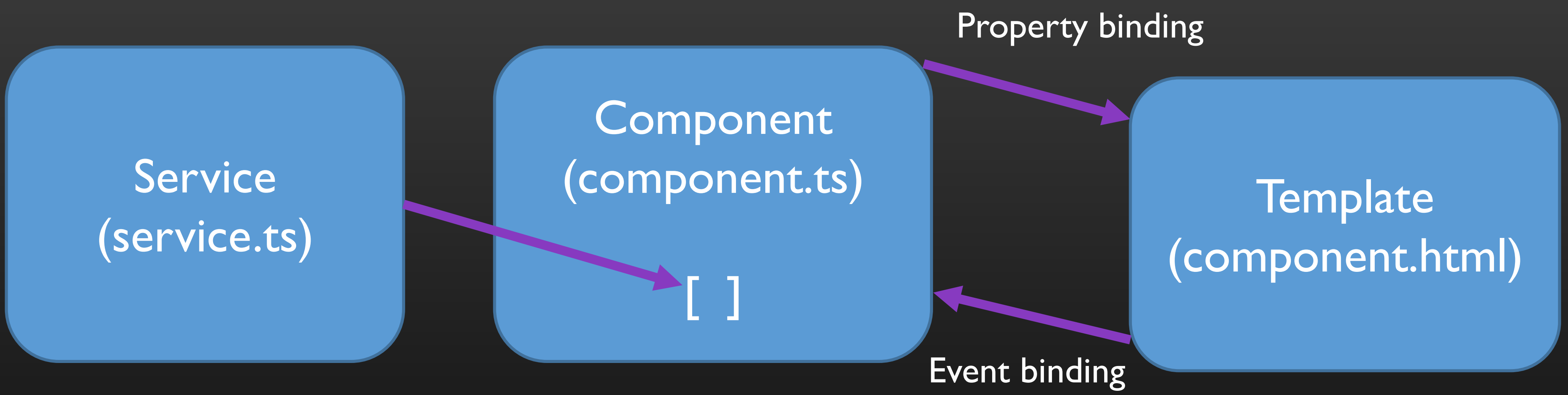
Ideal Angular Components

- Components are presentational
- Data comes in through services
- Components have little if any state
- Components generate actions to update “global” state
- “global” state trickles down as services to Components

Angular Services

- Components should be kept lean
- Fetching data, user input validation, logging – service
    - Factor out application logic
- Dependency injection, promises, reactive JavaScript

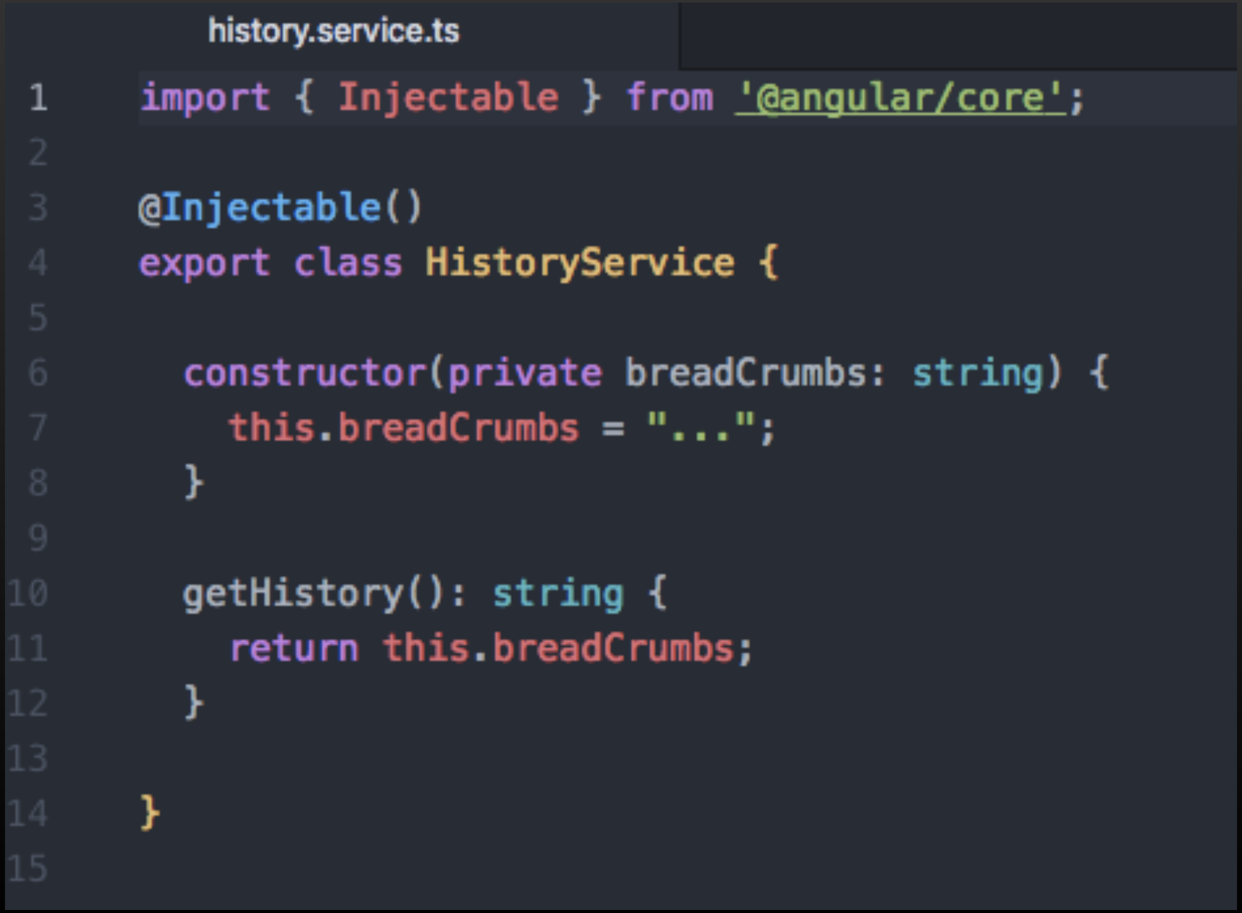
Angular Services



Dependency Injection

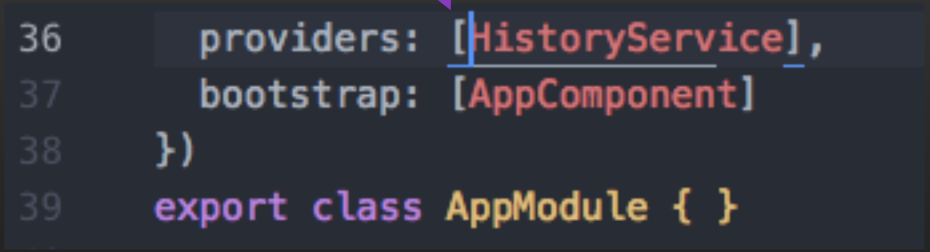
Add service to providers in app.module.ts

```
>> ng generate service history
```

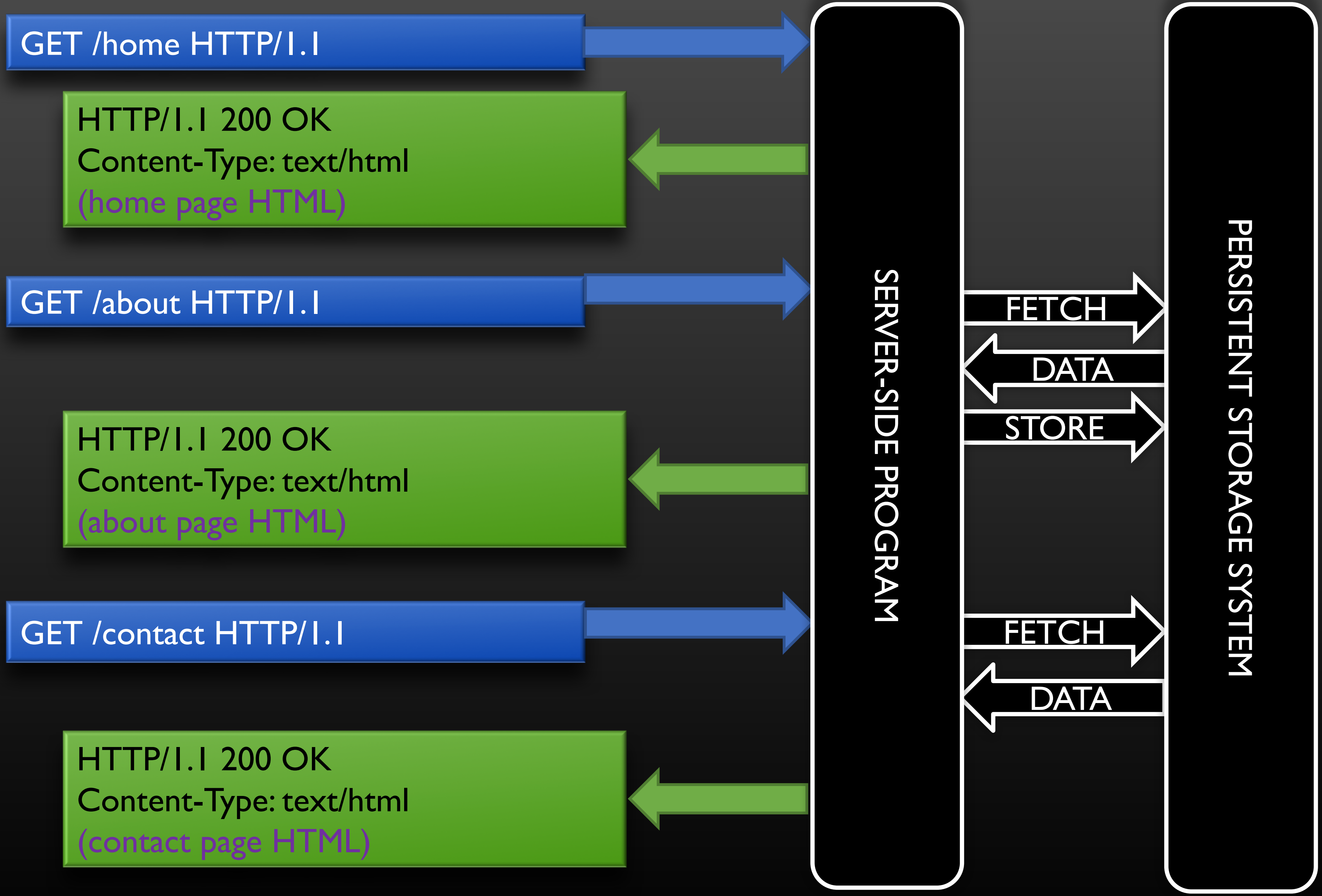


history.service.ts

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

@Injectable()
export class HistoryService {

  constructor(private breadCrumbs: string) {
    this.breadCrumbs = "...";
  }

  getHistory(): string {
    return this.breadCrumbs;
  }

}
```



```
36      providers: [HistoryService],
37      bootstrap: [AppComponent]
38    })
39    export class AppModule { }
```

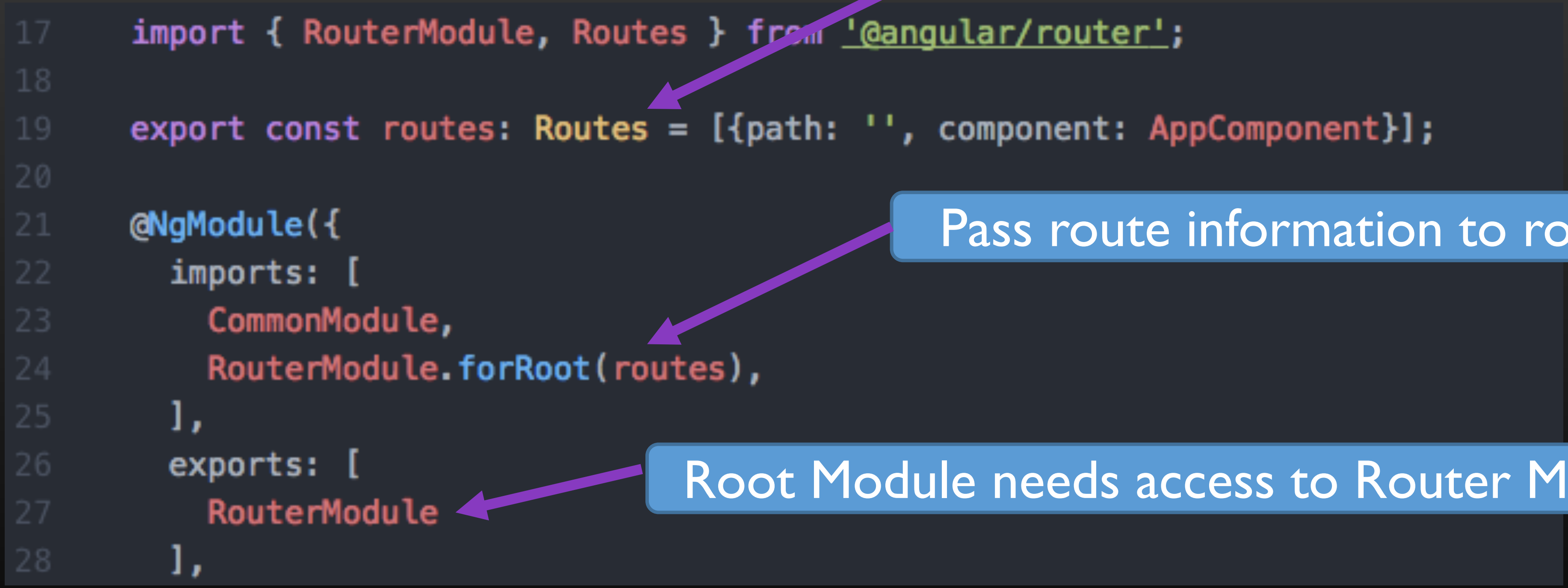

GET /home HTTP/1.1
HTTP/1.1 200 OK
Content-Type: text/html
(home page HTML)
GET /about HTTP/1.1
HTTP/1.1 200 OK
Content-Type: text/html
(about page HTML)
GET /contact HTTP/1.1
HTTP/1.1 200 OK
Content-Type: text/html
(contact page HTML)
SERVER-SIDE PROGRAM
PERSISTENT STORAGE SYSTEM
FETCH
DATA
STORE
FETCH
DATA

Angular Routing

- Navigate between components, render different views
- Uses a browser URL to navigate to client-gen view

- HTML5 History
  - Modify website URL without causes a refresh
  - Normally go to server, change so that we change view without refresh
  - pushState(): Add history entry
  - replaceState(): Modify history entry
  - Configure base href = "/"
  - http://example.com/home to http://example.com/menu without refresh

Angular Routing



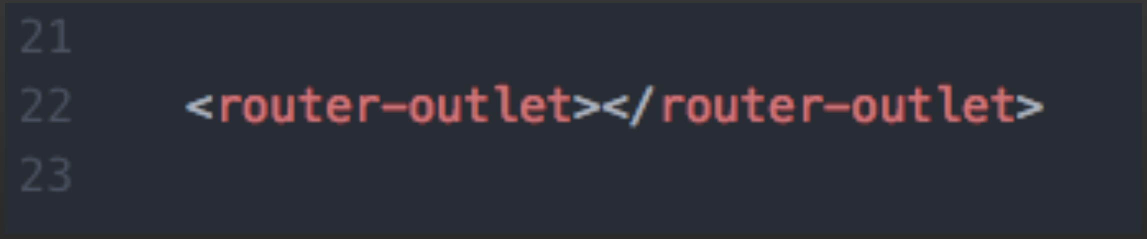
```
import { RouterModule, Routes } from '@angular/router';

export const routes: Routes = [{path: '', component: AppComponent}];

@NgModule({
  imports: [
    CommonModule,
    RouterModule.forRoot(routes),
  ],
  exports: [
    RouterModule
  ],
```

Define routes relative to base URL

Pass route information to route module

Root Module needs access to Router Module

Routing in Root Component Template



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

Uses routes to determine which template view to display