Angular Developer 6

example code

Tasks

base repo

Convert showing/hiding to routing

1. Intro page (smart component, route)
    - handles navigation
    - composed from:
        - Intro text
        - Player form component (dumb component)
2. Game page (smart component, route)
    - handles navigation (going back, Location service)
    - composed from:
        - Personalized welcoming tex
        - Game info: status, points (dumb component)
        - tetris game code (library)
        - Controller (dumb component)

App should be broken now - no data being passed from Intro to Game page

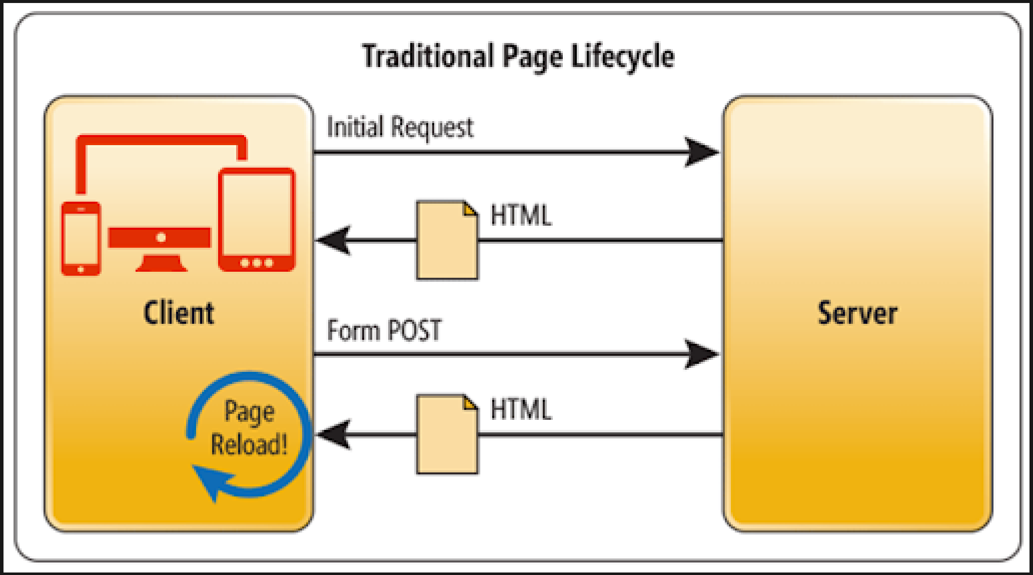
Store player data in a service

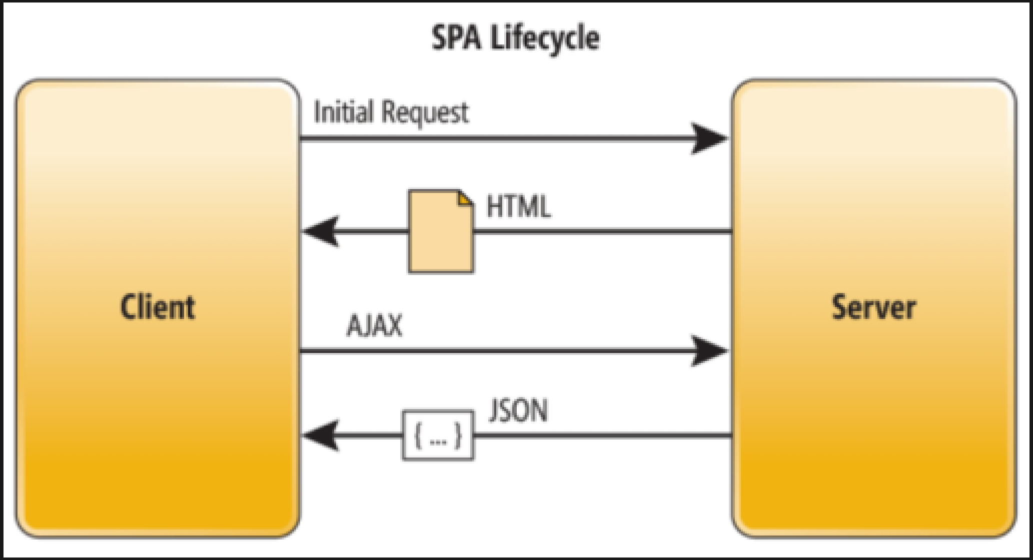
1. Create service for storing player data
2. Intro page - puts player data to store
3. Game page - reads player data from store

MPAs

vs

SPAs


Traditional Page Lifecycle
Client
Initial Request
HTML
Form POST
HTML
Page Reload!
Server


SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

Old school way (MPA way)

- Faster initial page load
- Server generates everything
- **SERVER** decides what to show

SPA way

- More data to load at first
- Server just serves data
- Browser renders everything
- **BROWSERs JS** decides what to show

Browsing websites

NAVIGATION

IN ANGULAR WORLD...

Navigation

=

Showing / hiding content

```
export class AppComponent {
    show = 'A';

    change(): void {
        this.show = this.show === 'A' ? 'B' : 'A';
    }
}
```

```
<app-a
  *ngIf="show === 'A'">
</app-a>

<app-b
  *ngIf="show === 'B'">
</app-b>

<button (click)="change()">change</button>
```

Ok, this works...

... but there is a smarter way :P

ROUTING

What is it?

- Natural way to keep application state
- Mapping browser url to state

```
https://shop.com

https://shop.com/sign-in

https://shop.com/sign-up

https://shop.com/products

https://shop.com/product/12

https://shop.com/cart
```

Routing in SPAs

showing/hiding content

based on browser URL

ROUTING IN ANGULAR

Rendering components based on url

Defining routes

```
// app.module.ts
@NgModule({
  ...
  imports: [
      BrowserModule,
  ],
  ...
})
export class AppModule { }
```

Defining routes

```
// app.module.ts
import { RouterModule } from '@angular/router';
@NgModule({
  ...
  imports: [
      BrowserModule,
      RouterModule.forRoot([
        ...
      ]),
  ],
  ...
})
export class AppModule { }
```

Defining routes

```
// app.module.ts
import { RouterModule } from '@angular/router';
@NgModule({
  ...
  imports: [
      BrowserModule,
      RouterModule.forRoot([
          {  path: 'A', component: AComponent },
          {  path: 'B', component: BComponent },
          {  path: '**', redirectTo: 'A' },
      ]),
  ],
  ...
})
export class AppModule { }
```

Where to render?

Router outlet

```html
<!-- app.component.html -->
<app-a
  *ngIf="show === 'A'">
</app-a>

<app-b
  *ngIf="show === 'B'">
</app-b>
```

Router outlet

```html
<!-- app.component.html -->

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

What about links?

```html
<!-- app.component.html -->

<router-outlet></router-outlet>
<a [routerlink]="['/A']">Open A</a>
<br>
<a [routerlink]="['/B']">Open B</a>
```

Buttons?

```html
<!-- app.component.html -->

<router-outlet></router-outlet>
<button (click)="openA()">Open A</button>
<br>
<a [routerlink]="['/B']">Open B</a>
```

```typescript
// app.component.ts
import { Router } from '@angular/router';
export class AppComponent {
    constructor(private _router: Router) { }

    openA() {
        this._router.navigate(['/A']);
    }
}
```

Routing - Summary

- Routes - components paired with URLs

```
RouterModule.forRoot([
    {  path: 'A', component: AComponent }
]),
```

- Based on defined routes and current path renders our application

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

- Navigation through links

```
<a [routerlink]="['/A']">Open A</a>
```

- Navigation from code

```
constructor(private _router: Router) { }
openA() {
  this._router.navigate(['/A']);
}
```

Basic Routing!

```
constructor(private _router: Router) { }
openA() {
  this._router.navigate(['/A']);
}
```

Services

WHAT IS A SERVICE?

- **Class**
- value, object, function

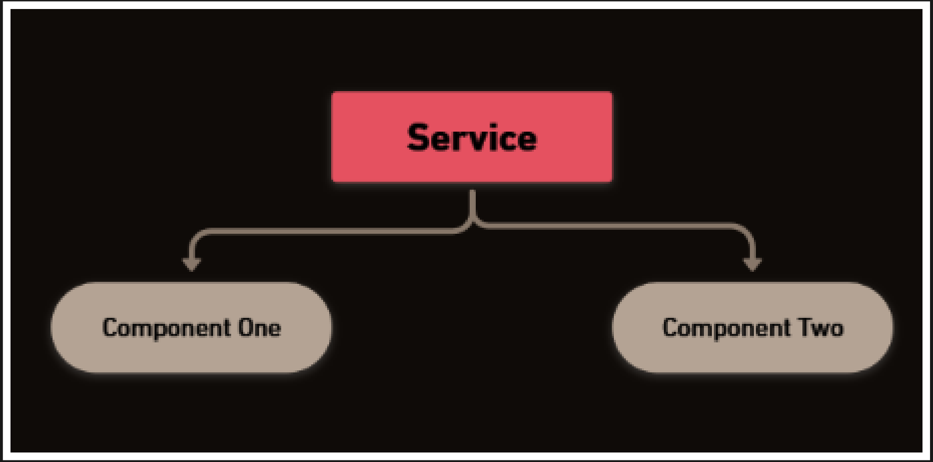
reusable

something like component?

COMPONENT	vs	SERVICE
.ts + .html + .scss		.ts
User interaction layer		Business logic layer
Presenting data		Storing and processing data
Multiple instances		One instance (not always)

`@Component()` `@Injectable()`

COMPONENTS & SERVICE



Why we need services?

- Share logic
- Share data

As most of them are singletons

Built in services

- Router
- HttpClient - making http requests
- Location - interaction with URL

USING SERVICES

Inject service to your component

```
// b.component.ts
import { Location } from '@angular/common';
@Component(...)
export class BComponent implements OnInit {

  constructor(private _location: Location) { }
}
```

Class constructor is basically the list of dependencies

Use it in your code

```typescript
// b.component.ts
import { Location } from '@angular/common';
@Component(...)
export class BComponent implements OnInit {

  constructor(private _location: Location) { }
  goBack() {
    this._location.back();
  }
}
```

```html
<!-- b.component.html -->
<button (click)="goBack()">back</button>
b works!
```

DEPENDENCY INJECTION

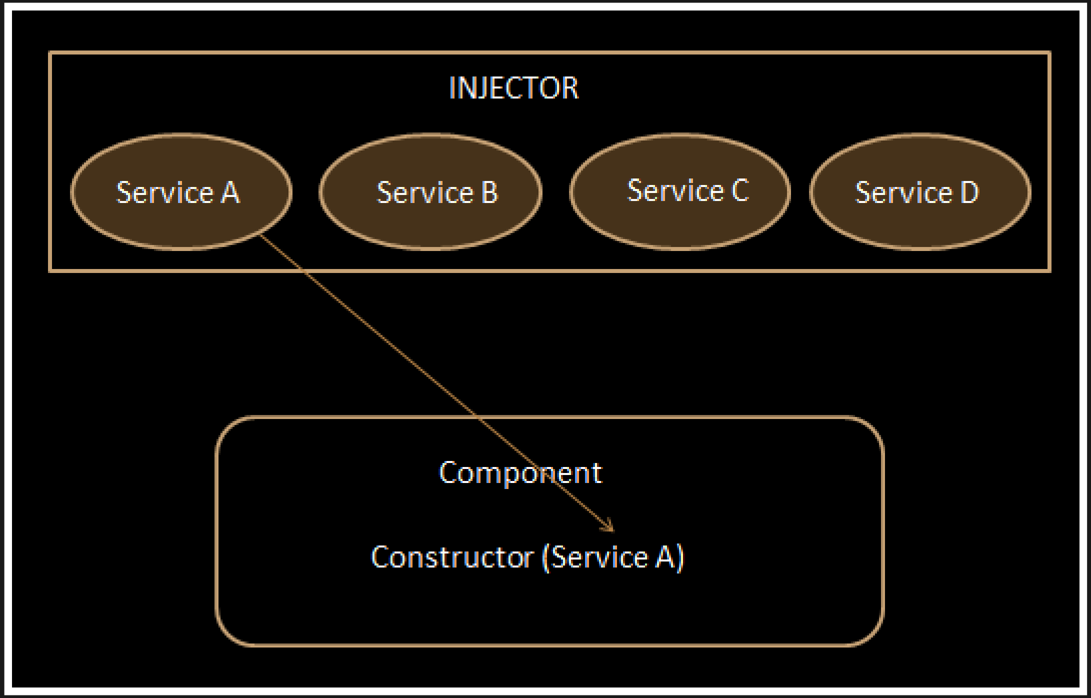
SOLI D

Dependency Inversion Principle

- High-level modules should not depend on low-level modules.
  Both should depend on abstractions.
- Abstractions should not depend upon details.
  Details should depend upon abstractions.

DEPENDENCY INJECTION IN ANGULAR(LINK)

- Out of the box
- Way to provide dependencies
- Introduces new entity: INJECTOR
- There can be many injectors
- Always at least one 'app root' injector

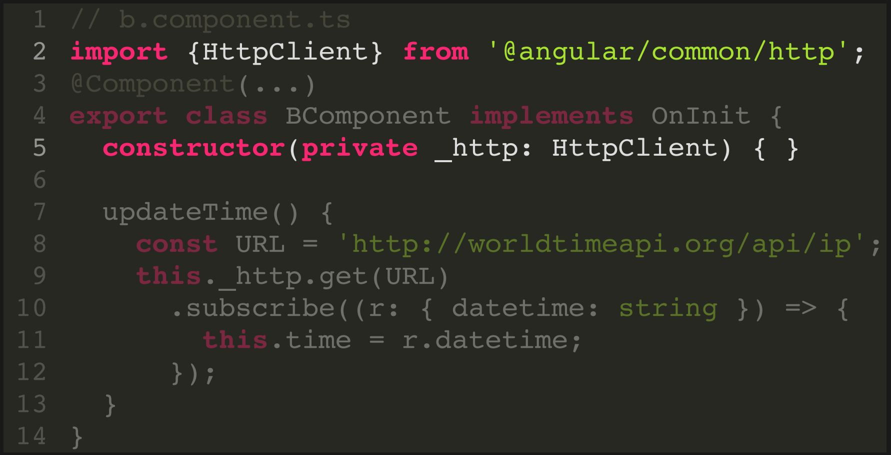

INJECTOR
Service A
Service B
Service C
Service D
Component
Constructor (Service A)

Crucial in testing

Super useful in modularization

How to use?

Define what you need in your class and use it!



```
// b.component.ts
import {HttpClient} from '@angular/common/http';
@Component(...)
export class BComponent implements OnInit {
  constructor(private _http: HttpClient) { }

  updateTime() {
    const URL = 'http://worldtimeapi.org/api/ip';
    this._http.get(URL)
      .subscribe((r: { datetime: string }) => {
        this.time = r.datetime;
      });
  }
}
```

Works with: components, pipes, directives, other services...

PS

Sometimes services come from separate modules

```
// app.module.ts
import { RouterModule } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  ...
  imports: [
      BrowserModule,
      RouterModule.forRoot(...),
      HttpClientModule
  ],
  ...
})
export class AppModule { }
```

We need to import them!

OUR OWN SERVICES

Class

```
// storage.service.ts
export class StorageService {

  constructor() { }
}
```

... class will be pretty versatile

Decorate your class

```
// storage.service.ts
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StorageService {

  constructor() { }
}
```

…or just generate with CLI

```
$ ng generate service storage
```

Business logic

```typescript
// storage.service.ts
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StorageService {
  private _secret = 'some secret string';

  readSecret () {
      return this._secret;
  }
}
```

Using

```
// b.component.ts
import {StorageService} from '../storage.service';

@Component(...)
export class BComponent implements OnInit {
  public text;

  constructor(private _storage: StorageService) {
    this.text = this._storage.readSecret();
  }
}
```

Summary

- Create class for your business logic and data
- Decorate it with @Injectable

```
@Injectable({
  providedIn: 'root'
})
```

- Request your new service in a component or other service by properly typing constructor parameters

```
import {StorageService} from '../storage.service';

@Component(...)
export class SomeClass {
  constructor(private _myService: MyService) { }
}
```

- Use it!

```
export class SomeClass {
  async initialize() {
    this.data = await this._myService.loadData();
  }
}
```

PS

Lets stick to that for now

```
@Injectable({
  providedIn: 'root'
})
```

there are other ways... but they are pretty advanced

What it is actually doing?

```
@Injectable({
  providedIn: 'root'
})
```

- Converts your class into a service
- Makes it 'usable' by angulars DI mechanism
- Creates injection token from decorated class
- Registers your service in global INJECTOR

read more

APPLICATION STRUCTURE

Types of components

(framework agnostic)

- Smart components
- Dumb components

Dumb components

- No interaction with routing
- No access to services
- Stateless
- Focused on UI
- Colors and shapes (styling)
- Receive data through @Input(s)
- Communicate through @Outpus(s)

Examples: lists, list item details, forms...

Smart components

- State aware
- Work with routing
- Utilize services
- Communicate with servers
  (through services)
- Know how to handle data
- Pass data to dumb components

How smart and dumb components communicate?

- @Input
- @Output

Why?

- Easier to work with and focus on a task
- Easier to harness change detection
- Better separation of concerns