Additional Considerations

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

Functions vs. pipes

Using a memo decorator

HttpClient and RxJS operators

Security considerations

HTTP interceptors

Functions vs. Pipes

Function Calls in Templates

Function calls made from a template are invoked every time a change occurs (no caching)

Calling a Function from a Template

Name	Total
Basketball	$32.39
XBox	$269.99
Nintendo Switch	$269.99
Bat	$32.39

```
{{ addTax(product.price) | currency }}
```

Replacing Functions with Pipes

A pure pipe returns the same result given the same inputs

Only called when inputs are changed

Using a Pipe in a Template

Name	Total
Basketball	$32.39
XBox	$269.99
Nintendo Switch	$269.99
Bat	$32.39

```
{{ product.price | addtax | currency }}
```

Functions and Pipes in Action

Using a Memo Decorator

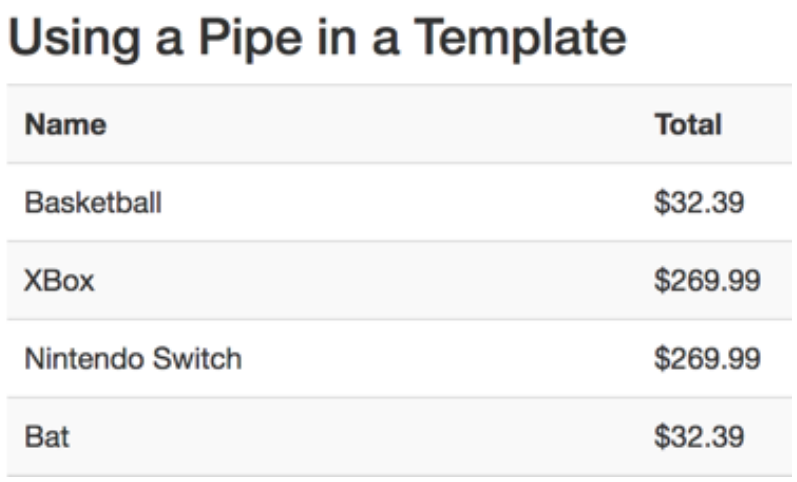
```
import memo from 'memo-decorator';

@Pipe({ name: 'addtaxmemo' })
export class AddTaxMemoPipe implements PipeTransform {

  @memo()
  transform(value: any, args?: any): any {
    // return product.price + tax
  }

}
```

The Memo Decorator

Use the Memo Decorator to enhance caching of a pipe's transform() function when a primitive value is passed

Pipe with a Memo Decorator

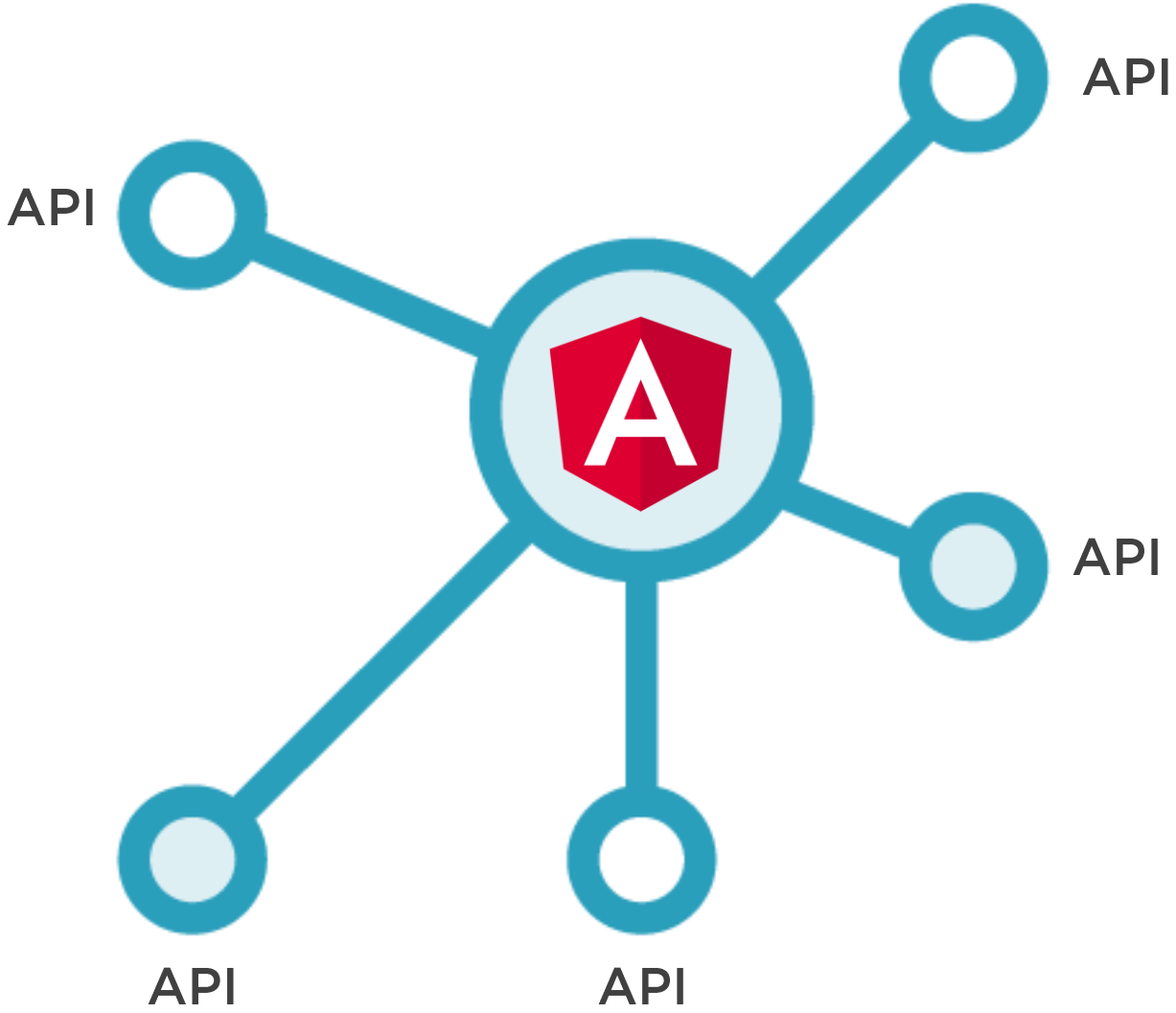
Cache result based on inputs passed into pipe transform()

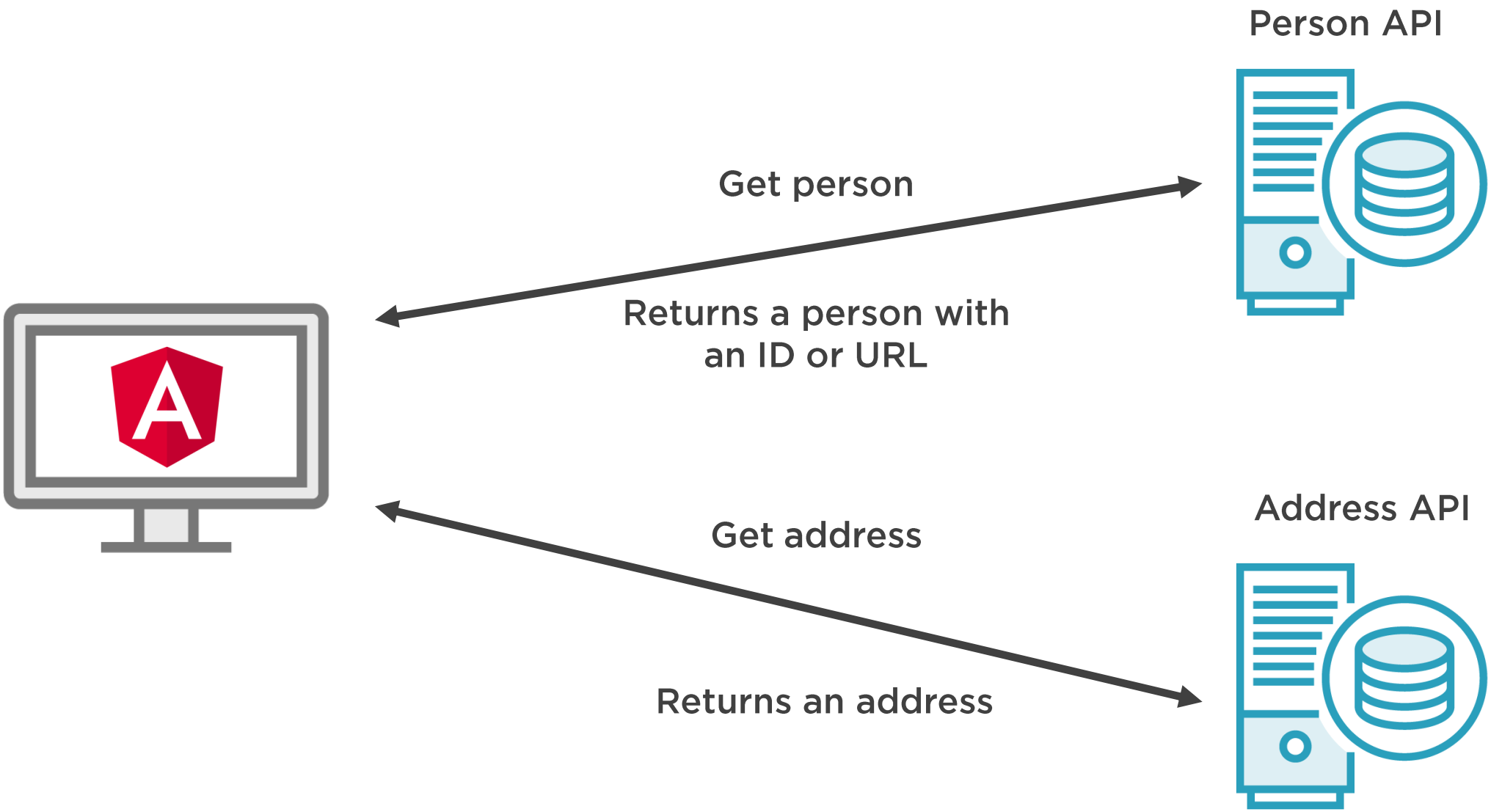



```
{{ product.price | addtaxmemo | currency }}
```

HttpClient and RxJS Operators

API Calls




Person API
Get person
Returns a person with
an ID or URL
Address API
Get address
Returns an address

HttpClient and RxJS Operators

switchMap

mergeMap

forkJoin

Key Security Considerations

Security Considerations

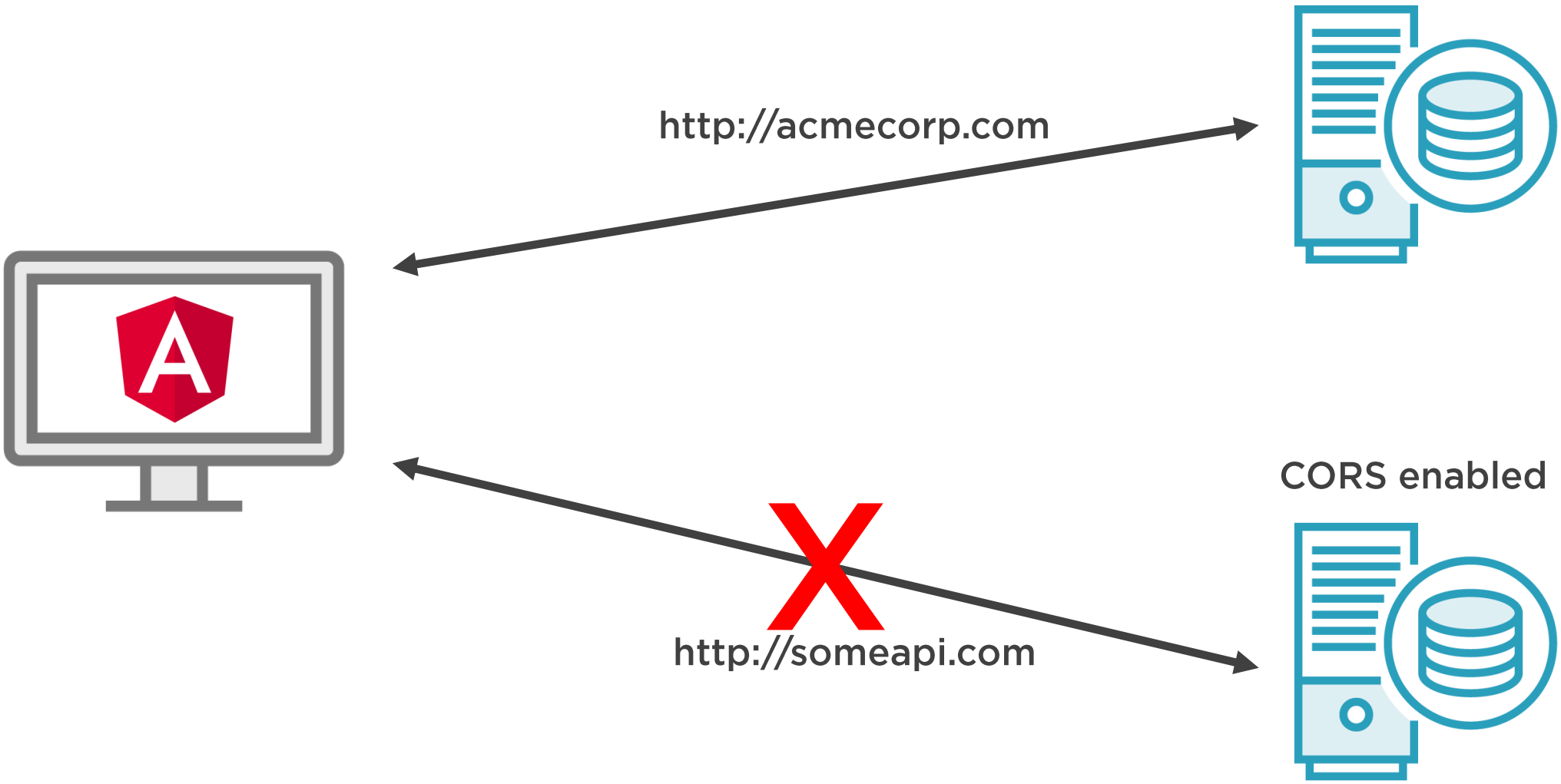
CORS

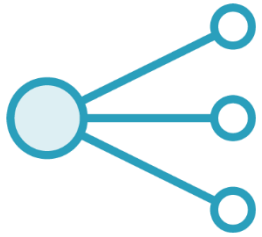
CSRF

Route Guards

Sensitive Data

Cross-origin Resource Sharing (CORS)





CORS Considerations

CORS allows a browser to call a different domain or port

Enable on the server as needed

Limit allowed domains, headers, and methods

Security Considerations

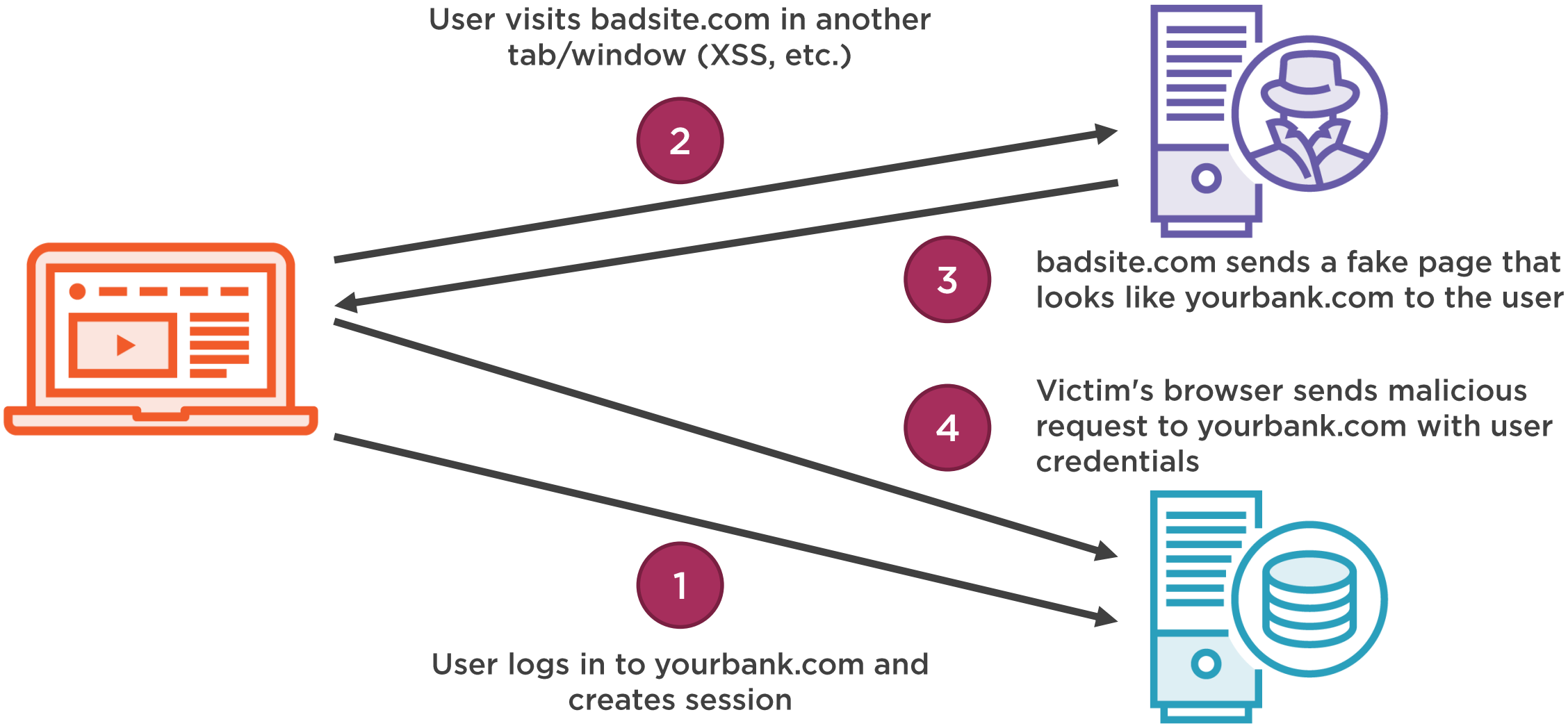
CORS

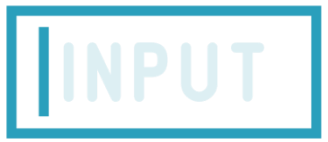
CSRF

Route Guards

Sensitive Data

Cross-site Request Forgery (CRSF)





CSRF Considerations

Enable CSRF on the server if using cookie authentication

Angular will read a token from a cookie set by the server and add it to the request headers

Change the cookie/header name as appropriate for your server

Server will validate the header value

Security Considerations

CORS

CSRF

Route Guards

Sensitive Data

Route Guards

Define route guards needed by application based on user or group/role

Keep in mind that route guards don't "secure" an application

Rely on the server to secure data, APIs, etc.

Security Considerations

CORS

CSRF

Route Guards

Sensitive Data

Sensitive Data

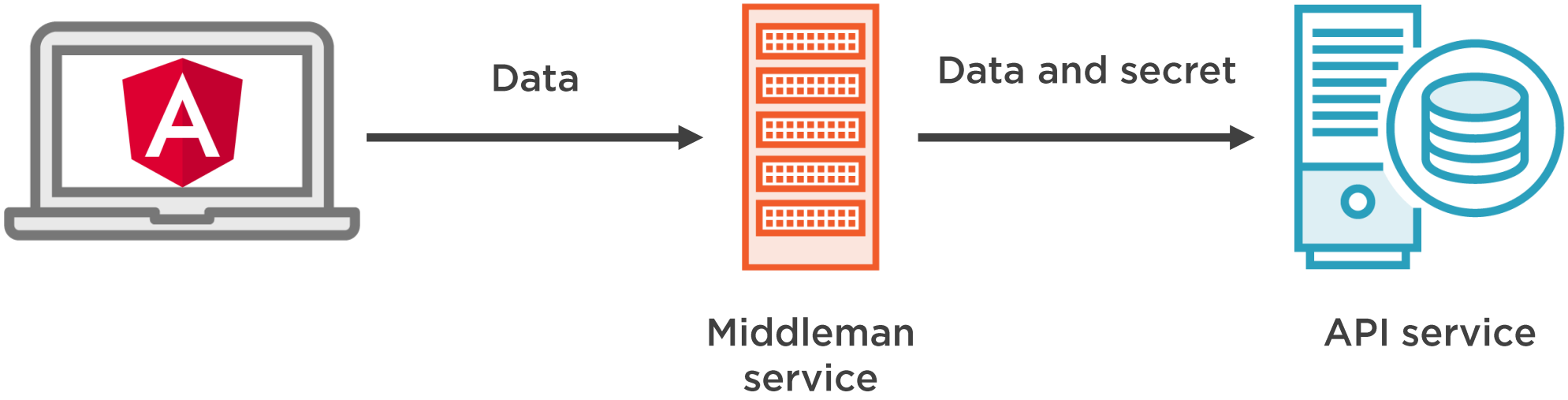
Anyone can access the browser developer tools to view variables, local/session storage, cookies, etc.

Do not store sensitive data (secrets, keys, passwords, etc.) in the browser

If an API requires a ”secret” to be passed, consider calling it through a “middle-man” service that you own

Use JWT tokens where possible for server authentication (set appropriate TTL expiration for tokens)

Using a Middleman Service



Additional Security Considerations

Authentication

Authorization

HTTPS

HTTP Interceptors

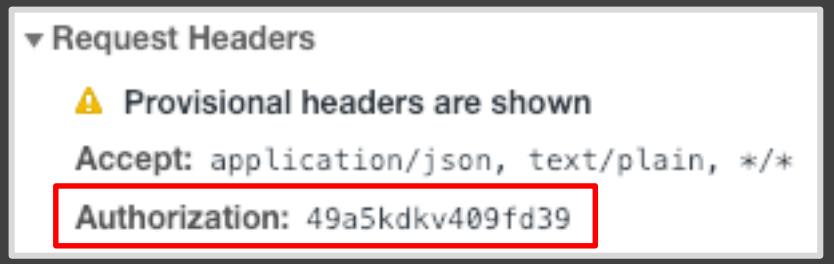
```
@Injectable()
export class CorsInterceptor implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    const authReq = req.clone({
      withCredentials: true,
      headers: req.headers.set('X-Requested-With', 'XMLHttpRequest')

    });

    return next.handle(authReq);
  }
}
```

HTTP Interceptors and CORS

HTTP Interceptors provide a centralized place to hook into requests/responses

Add withCredentials when using cookies and calling via CORS

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    // Get the auth header (fake value is shown here)
    const authHeader = '49a5kdkv409fd39'; // this.authService.getAuthToken();
    const authReq = req.clone({
        headers: req.headers.set('Authorization', authHeader)

    });

    return next.handle(authReq);
  }
}
```



HTTP Interceptors and Tokens

Interceptors can be used to pass tokens required by services

```
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { AuthInterceptor } from './interceptors/auth.interceptor’;

@NgModule({

  providers: [{
    provide: HTTP_INTERCEPTORS,
    useClass: AuthInterceptor,
    multi: true
  }]

}
export class CoreModule {}
```

Registering an HTTP Interceptor

Interceptors can be provided in the Core module

Since more than one interceptor can be used, set *multi* to *true*

Summary

- Use pipes over functions in views (when possible)
- Leverage RxJS operators when making HttpClient calls to the server
- Use HTTP interceptors to modify requests and access responses in a centralized place
- Take time to carefully evaluate the security needs of the application