Marker Tools

CS572 Modern Web Applications Programming

Maharishi University of Management

Department of Computer Science

Assistant Professor Asaad Saad

OUTLINE

Search...

Forms in Angular

A lot of applications are very form-intensive, especially for enterprise development.

Forms can end up being really complex:

- Form inputs are meant to **modify data**, both on the page and the server
- Users cannot be trusted in what they enter, so you need to **validate** values
- The UI needs to clearly **state expectations** and errors
- **Dependent fields** can have complex logic
- We want to be able to **test** our forms, without relying on DOM selectors

Form states such as being: **`valid`**, **`invalid`**, **`pristine`**, **`dirty`**, **`untouched`**, **`touched`**, **`disabled`**, **`enabled`**, **`pending..etc`**

PREV NEXT

Two Modules

Angular has the **`@angular/forms`** package with two modules:

`FormsModule` (**Template Driven Forms**)

We create a form by placing directives in the template. We then use data bindings to pass the data from and to that form.

`ReactiveFormsModule` (**Data Driven Forms**)

We define a form in the component class and bind it to elements in the template. Because we use reactive programming to pass data in and out of the form, we call it “reactive”.

PREV NEXT

Importing Form Modules

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
@NgModule({
        declarations: [ ],
        imports: [ BrowserModule,
                   FormsModule,
                   ReactiveFormsModule ]
})
```

By importing `FormsModule` or `ReactiveFormsModule` into our `NgModule` means we can use all directives exported from these modules in our view template.

High-Level Form Programming

Form Model

The form model is a UI-independent representation of a form. There are three building blocks: **`FormControl`**, **`FormGroup`**, and **`FormArray`**. All of these are available in both **`FormsModule`** and **`ReactiveFormsModule`**.

Form Directives

These directives connect the form model to the DOM. **`FormsModule`** and **`ReactiveFormsModule`** provide different sets of these directives.

DOM

These are DOM inputs, checkboxes, and radio buttons.

PREV NEXT

OUTLINE

Search...

Angular Form Models

- **`FormControl`** an indivisible part of the form, an atom. It usually corresponds to a simple UI element, such as an input. It has a **`value`**, **`status`**, and a map of **`errors`**
- **`FormGroup`** A fixed-size collection of controls. The value of a group is just an aggregation of the values of its children.
- **`FormArray`** A collection of the same control type as `FormGroup` of a variable length.
- **`Validator`** give us the ability to validate inputs.
- **`Observers`** let us watch our form for changes and respond accordingly.

Why Form Model?

The form model is a UI-independent way to represent user input consist of simple controls (`FormControl`) and their combinations (`FormGroup` and `FormArray`), where each control has a value, status, validators, errors, it emits events and it can be disabled.

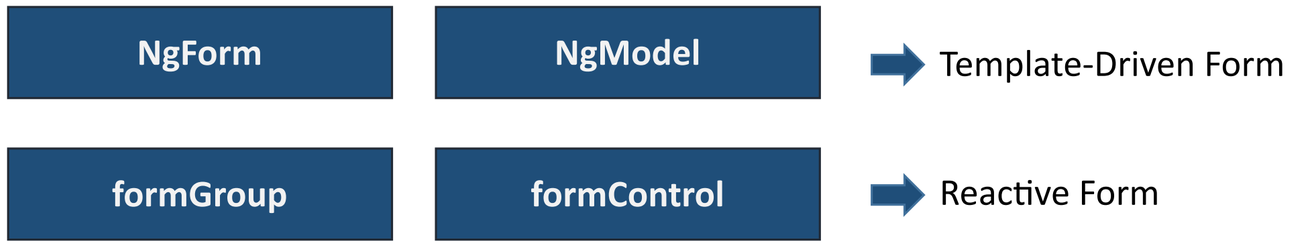
Having this model has the following advantages:

- Form handling is a complex problem. Splitting it into UI-independent and UI-dependent parts makes them easier to manage. And we can test form handling without rendering UI.
- Having the form model makes reactive forms possible.

PREV NEXT

Form Directives

Form Directives connect Form Models to the UI.



9

NgForm Directive

NgForm directive's selector is the **<form>** tag. It will get automatically attached to any **<form>** tags and makes **ngForm** available to our view.

How does it work?

By adding the **<form>** tag in a template, the directive will trigger and creates a new **FormGroup** instance named **ngForm** and it will give us an **(ngSubmit)** output

PREV NEXT

NgModel Directive

NgModel directive has a selector of **ngModel**, This means we can attach it to any form element as an attribute:
ngModel="var" (**var** is property in our component class).

How does it work?

By adding **ngModel** to UI elements, Angular creates a new **FormControl** that is automatically added to the parent **FormGroup** and then binds the new **FormControl** to the DOM element.

Template Driven Form

```
<form #f="ngForm" (ngSubmit)="onSubmit(f)">

    <input type="text"
           name="email"
           ngModel
           required
           pattern="[a-z0-9]+@(?:[a-z0-9]\.[a-z0-9]{2,4}"
           #email="ngModel">
    <div *ngIf="!email.valid"> Invalid email </div>

<button type="submit" [disabled]="!f.valid">Submit</button>
```

```
@Component({...})
export class TemplateDrivenComponent {
       onSubmit(form) {
              console.log(form.value);
       }
}
```

Easy trick for an alias to **ngForm** , to enable/disable the submit button

Easy trick to access ngModel in the view so we can display error messages

`form.value` will return the key/value pairs of this `FormGroup`

Two-way Data-binding (REVISIT)

```
@Component({
        selector: 'comp',
        template: `
            <p>Message: {{message}}</p>
            <input [value]="message" (input)="message=$event.target.value">
            <input [(ngModel)]="message" /> `
})
export class CompComponent {
        public message: string = 'Default Message';
}
```

Bind the input element to **ngModel**:
[] as input to take the value from the Component
() as output to save the value in the Component

ngModel binds a model to a form. ngModel can implement two-way data binding. Remember that two-way data binding is more complicated and difficult to reason about! Use it sparingly.

Data Driven Forms

```
const formgroup = new FormGroup({
        login: new FormControl(''),
        passwords: new FormGroup({
                password: new FormControl('', Validators.required),
                passwordConfirmation: new FormControl('')
        })
});
```

Using `FormBuilder`

`FormBuilder` is a service that helps us build forms and give us a lot of customization options.

Forms are made up of **`FormControl`** and **`FormGroup`** and the **`FormBuilder`** helps us create them (you can think of it as a factory object).

Data Driven Forms Directives

We want to link our **<form>** UI to the **myForm** object that we created in our Component.

When we want to bind an existing **FormGroup** to a **form** we use the directive **formGroup**

When we want to bind an existing **FormControl** to an **input** we use the directive **formControl**

Reactive Forms with FormBuilder

To start building a Data-Driven Form we inject **`FormBuilder`** service in the constructor of our component class.

During injection an instance of **`FormBuilder`** will be created.

```
@Component({...})
export class DataDrivenComponent {
        myForm: FormGroup;
        constructor(private fb: FormBuilder) {
                this.myForm = fb.group({
                        'email': ['asaad@mum.edu'] });
        }
        onSubmit(): void {
                console.log('you submitted value: ', this.myForm.value);
        }
}
```

There are two main methods on FormBuilder :
`control()` - creates a new **`FormControl`**
`group()` - creates a new **`FormGroup`**

Using myForm in the view

```
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
        <input type="text"
               name="email"
               [formControl]="myForm.get('email')">
        <div *ngIf="!myForm.get('email').valid">Invalid email</div>
        <div *ngIf="myForm.get('email').hasError('invalid')">Error</div>
        <button type="submit" [disabled]="!myForm.valid">Submit</button>
</form>
```

formControlName="email"

Inspect the code and see how angular adds all kind of state change classes to the form elements.

Main Points

Template Driven Forms

To create a new **`FormGroup`** and **`FormControl`** implicitly we use the following directives in our template:

- **`ngForm`**
- **`ngModel`**

Data Driven Forms

To bind to an existing **`FormGroup`** and **`FormControl`** from your component to your template we use the following directives:

- **`formGroup`**
- **`formControl`**

Add Validation

Validators change the status of **`FormControl/FormGroup.`**
To assign a validator to a **`FormControl`** object we simply pass it as the **second argument** to our **`FormControl`** constructor:

```
@Component({...})
export class DataDrivenComponent {
    myForm: FormGroup;
    constructor(fb: FormBuilder) {
        this.myForm = fb.group({
            'email': ['asaad@mum.edu', Validators.required] });
    }
}
```

To use more than one validator:

The compose function will execute all the validators and merge the errors.

```
'email': ['asaad@mum.edu', Validators.compose([Validators.required, Validators.email]) ]
```

Custom Validators

A validator is a function that takes a **`FormControl`** as an input and returns a **`StringMap<string, boolean>`** where the key is error code and the value is `true` if it fails

```
exampleValidator(control: FormControl): {[s: string]: boolean} {
        if (control.value === 'Example') {
                return {'invalid': true};
        }
        return null;
}
```

Returning null means validation is valid.
Returning anything else means validation is invalid.

Custom Asynchronous Validators

Asynchronous validator is a function that takes a **`FormControl`** as its input and returns a **`Promise<any>`** or an **`Observable<any>`**

```
asyncValidator(control: FormControl): Promise<any> | Observable<any> {
   const promise = new Promise<any>((resolve, reject) => {
         setTimeout(() => { if (control.value === 'Example') {
                              resolve({'invalid': true});
                           } else {
                              resolve(null);
                           }
                    }, 1500);
   });
   return promise;
}
```

While the promise or the observable being resolved, the status of the form will be `PENDING`

Watching For Changes

Both **`FormGroup`** and **`FormControl`** have an **`EventEmitter`** that we can use to observe changes.

Any time a control updates, it will emit the value.

To watch for changes on form/control we get access to the **`EventEmitter/Observable`** by calling **`valueChanges`** or **`statusChanges`** (valid, invalid, pending).

Since **`valueChanges`** and **`statusChanges`** are RxJS observables, we can use the rich set of RxJS operators to implement powerful user interactions in a just a few lines of code.

Example

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder } from "@angular/forms";

@Component({...})
export class DataDrivenComponent {
        myForm: FormGroup;
        constructor(private formBuilder: FormBuilder) {
                this.myForm = formBuilder.group({});
                this.myForm.statusChanges.subscribe(
                        (data: any) => console.log(data) );
        }
}
```

Main Point

Angular forms can be created via two ways: by using the **template-driven** approach or the **data-driven** approach.

- In the first approach (**template-driven**), Angular will automatically create a form with which it works by identifying the **`<form>`** tag. You will have to assign controls to this form by adding the **`ngModel`** directive to the HTML elements which should be form inputs/controls.
- In the **data-driven** approach, you create the form on your own (in the component body) and then assign it to the HTML code by using **`formGroup`** for the form and **`formControl`** for the individual controls.

PREV NEXT

Authentication vs Authorization

Authentication is the process of identifying that somebody really is who they claim to be.

Authorization is the process of verifying that you have access to something.

The two concepts are completely independent, but both are central to security design, and the failure to get either one correct opens up the avenue to compromise.

Authentication = login + password (who you are)
Authorization = permissions (what you are allowed to do)

OAuth2 - Delegated Authorization

OAuth2 is a standard protocol that solves the delegated authorization problem.

Example: Users may give permission to "Photo Printing App" to access one album on "Google Photos" so the app can print it out or create a collage poster. This is a better way than giving the App our Google username/password to access Google Photos on our behalf.

Unfortunately some apps (Banks, Mint: financial dashboard) are still collecting username/password to access users data.

I trust Google but I kind of trust this new App. I want the App to have access to my contacts only.

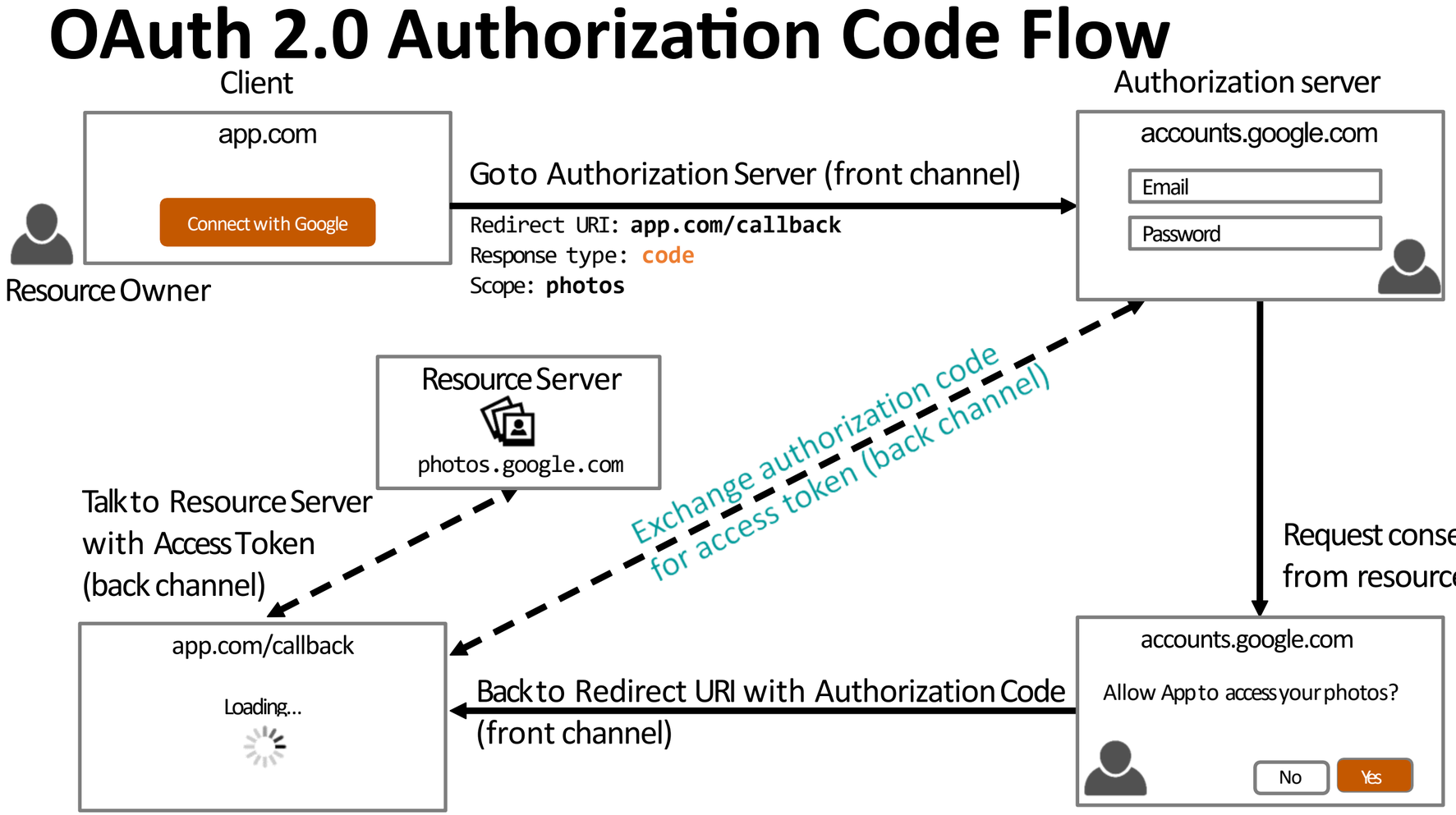
Creating an App and receiving Client ID and Secret

Application must register with the Auth service (application name, website, a logo, etc.). We must also register a **redirect URI** to be used for redirecting users back to the application.

After registering the app, we will receive a **client ID** and a **client secret**. The client ID is considered public information, and is used to initiate the process of the login URL (can be included in JavaScript source code on a page). The client secret must be kept confidential.

Because SPA apps or native apps cannot keep the secret confidential, then the secret is not used.

PREV NEXT



Starting the flow

```
https://accounts.google.com/oauth2/?
        client_id=asaad123&
        redirect_uri=https://app.com/callback&
        scope=photos&
        response_type=code&
        state=foobar
```

The user sees the authorization prompt: because we are sending a **`client_id`**, the auth server will display a message: The **APP_NAME** by **AUTHOR** would like the ability to access your photos, allow or deny? If the user clicks "Allow," the service redirects the user back to your site with an auth code

Calling back

The server returns the authorization code in the query string along with the same state value that is passed

```
https://app.com/callback?
        error=access_denied&
        error_description=The user did not consent.
```

```
https://app.com/callback?
        code=oMsCeLvIaQm6bTrgtp7&
        state=foobar
```

You should first compare this state value to ensure it matches the one you started with. This ensures your redirection endpoint isn't able to be tricked into attempting to exchange arbitrary authorization codes.

Exchange code for an Access token

```
POST accounts.google.com/oauth2/token
Content-Type: application/x-www-form-urlencoded
    code=oMsCeLvIaQm6bTrgtp7&
    client_id=asaad123&
    client_secret=secret123&
    grant_type=authorization_code&
    redirect_uri=https://app.com/callback&
```

Authorization server returns an Access token and expiration time

```
{
   "access_token": "fFAGRNJru1FTz70BzhT3Zg",
   "expires_in": 3920,
   "token_type": "Bearer",
}
```


PREV NEXT

Use the Access token

```
GET photos.google.com/api
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg
```

Resource Server will validate the token and use its scope for authorization

PREV NEXT

OUTLINE

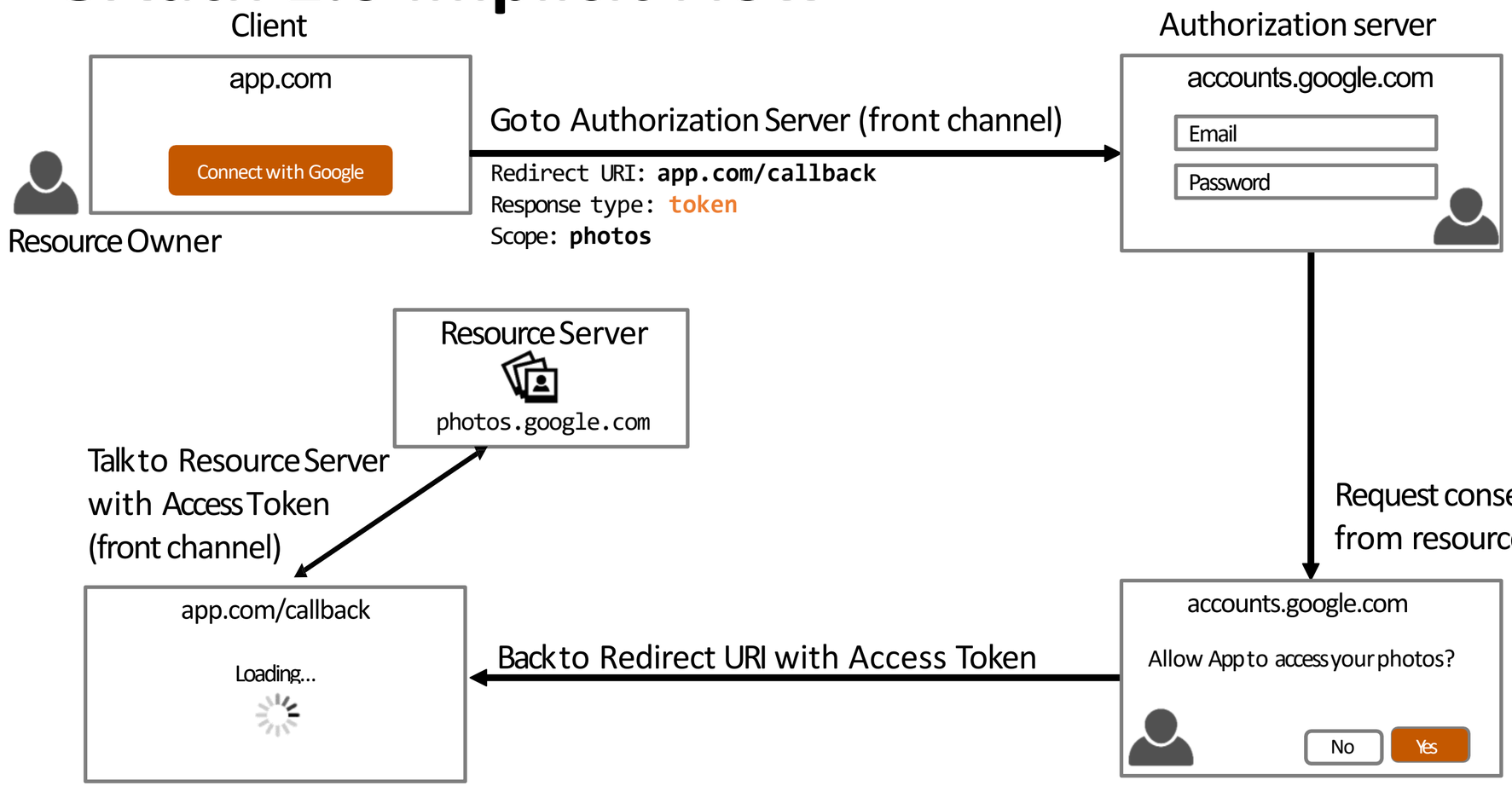
Search...

OAuth 2.0 flows

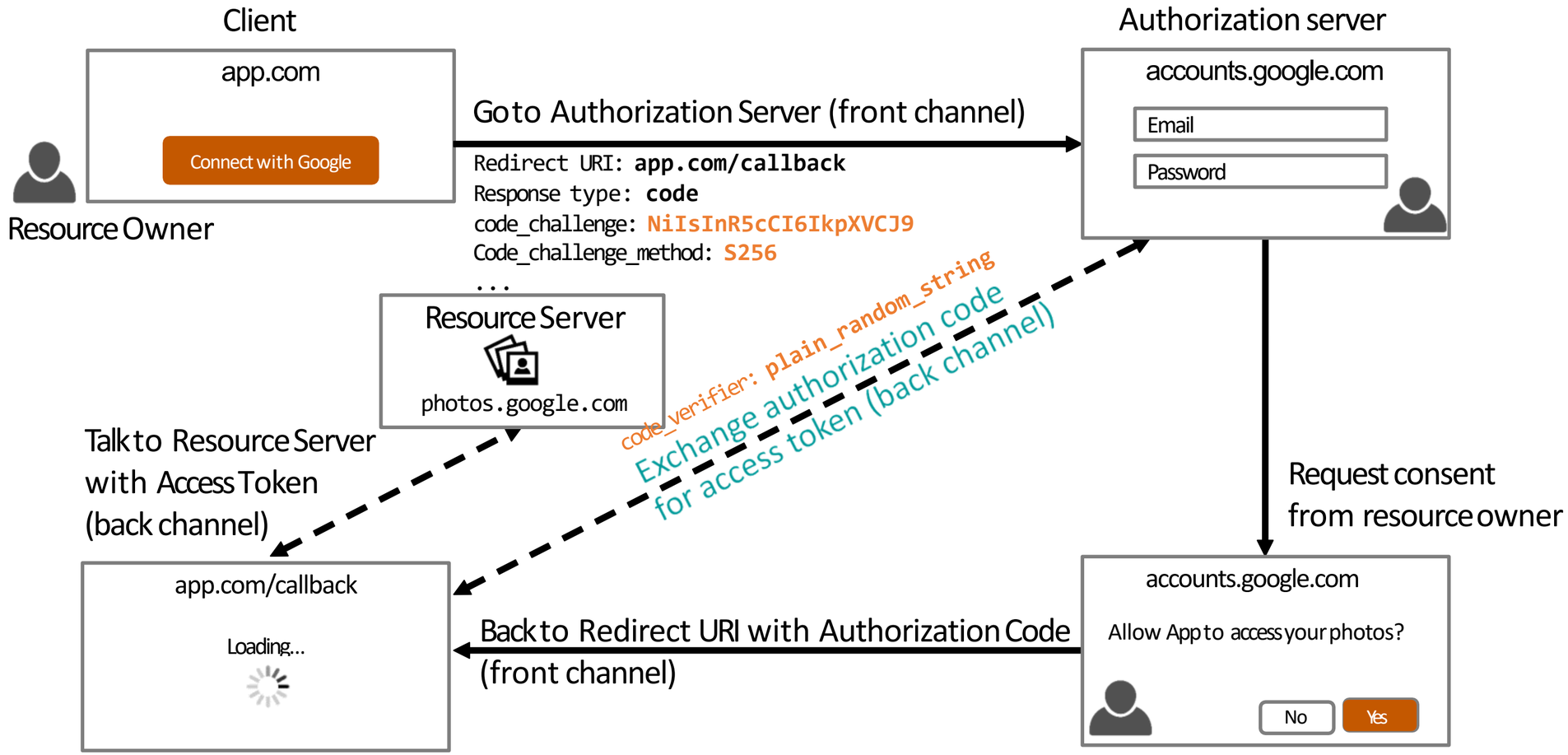
- Authorization code *(front channel + back channel)*
- Implicit *(front channel only)*
- Authorization code flow with PKCE *(Proof Key for Code Exchange)*

PREV NEXT

OAuth 2.0 Implicit Flow



OAuth 2.0 Authorization flow with PKCE



PREV NEXT

OUTLINE

Search...

OAuth 2.0 Authorization flow with PKCE

Clients create a **Code Verifier** which is a random string that the app stores locally.

Send the code challenge along with your first request:

`code_challenge=XXXXXXX` (base64-encoded version of the sha256 hash of the code verifier string)

`code_challenge_method=S256` (hashing method used to compute the challenge)

When you exchange your authorization code with an access token, you need to include:

`code_verifier=CODE_VERIFIER_STRING`

The authorization server will hash the provided plaintext (same client ID) and confirm that the hashed version corresponds with the hashed string that was sent in the initial authorization request. This ensures the security of using the authorization code flow with clients that don't support a secret.

OAuth 2.0 is NOT for Authentication

People often misuse OAuth protocol for simple login or mobile apps login, which is bad, because there is no standard way to know the user's information, OAuth cares about permission and scope, whether your access token is scoped to access certain resource or not, it does not care of who you are.

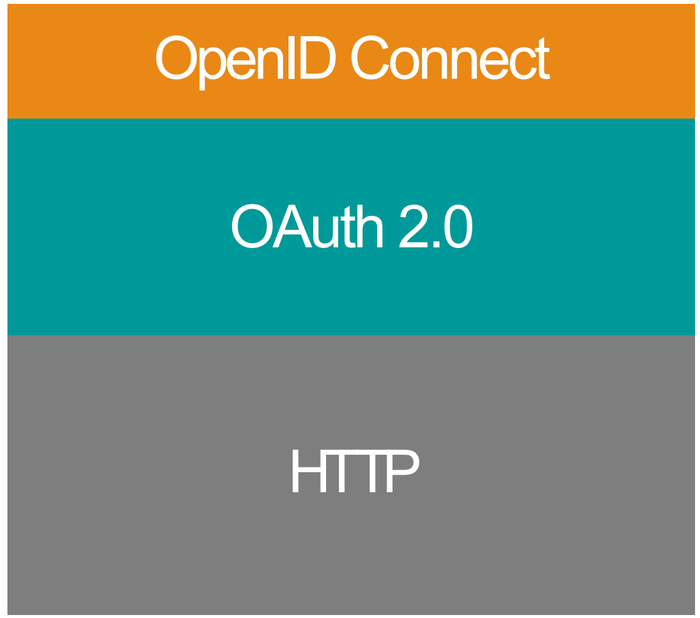
- Companies added their own custom implementation differently
- No common set of scopes

PREV NEXT

OpenID Connect for Authentication

OpenID Connect is not a new protocol, but rather an extension to OAuth2

- Added ID token
- UserInfo endpoints
- Standard set of scope



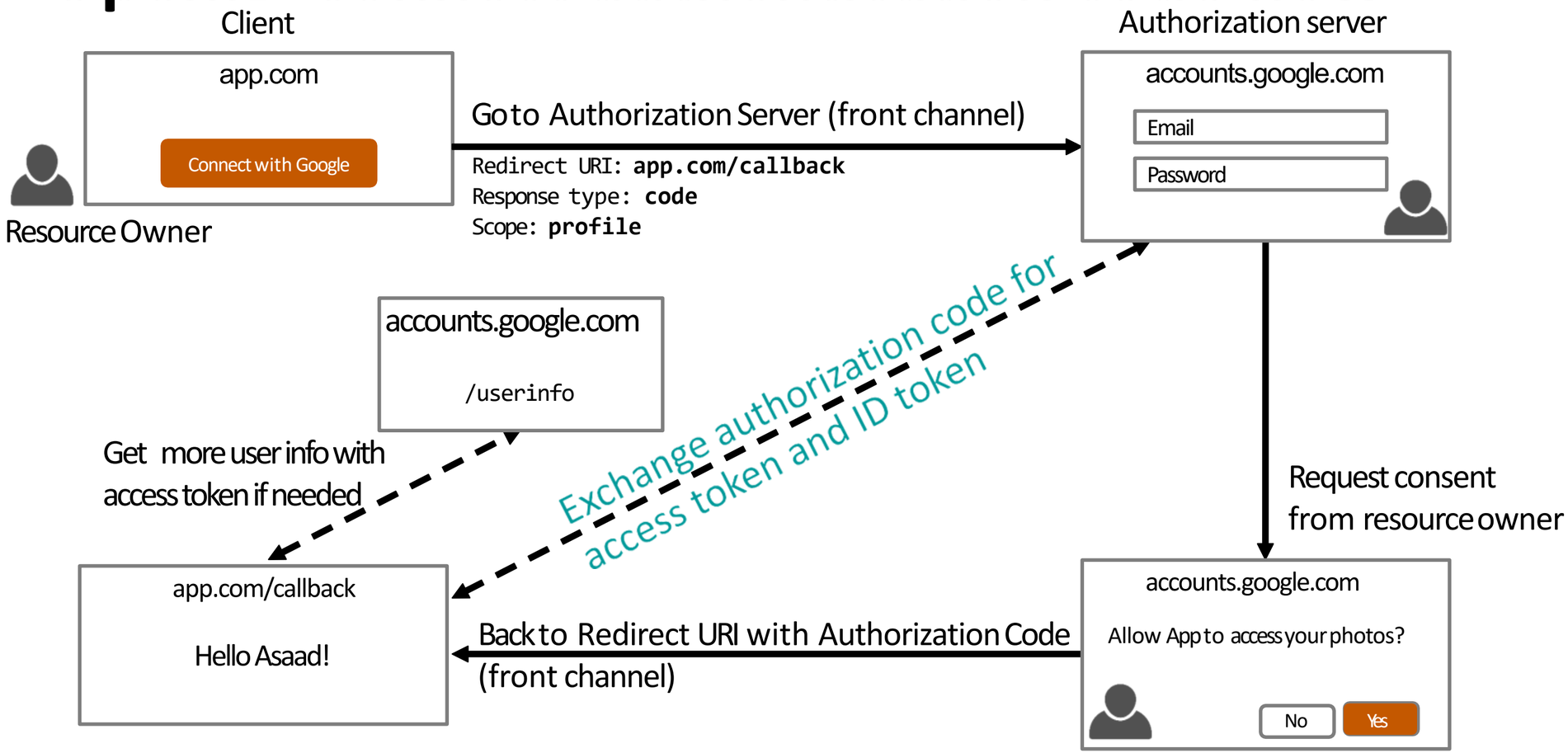
OpenID Connect is for authentication
OAuth 2.0 is for authorization

PREV NEXT

OUTLINE

Search...

OpenID Connect authentication code flow



Exchange code for access token and ID token

```
{
   "access_token": "fFAGRNJru1FTz70BzhT3Zg",
   "id_token": "eyJraB03ds3F..."
   "expires_in": 3920,
   "token_type": "Bearer",
}
```

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IkRNa3Itd0J
qRU1EYnhOY25xaVJISVhuYUxubWI3UUpfWF9rWm
JyaEtBMGMifQ.eyJzdWIiOiIwMHU5bzFuaWtqdk
9CZzVabzBoNyIsInZlciI6MSwiaXNzIjoiaHR0c
HM6Ly9kZXYtMzQxNjA3Lm9rdGFwcmV2aWV3LmNv
bS9vYXV0aDIvYXVzW84d3ZraG9ja3c5VEwwaDci
LCJhdWQiOiJsWFNlbkx4eFBpOGtRVmpKRTVzCIs
ImlhdCI6MTUwOTA0OTg5OCwiZXhwIjoxNTA5MDU
zNDk4LCJqdGkiOiJJRC5oa2RXSXNBSXZTbnBGYV
FHTVRYUGNVSmhhMkgwS2c5Ykl3ZEVvVm1ZZHN3I
iwiYW1yIjpbImtiYSIsIm1mYSIsInB3ZCJdLCJp
ZHAiOiIwMG85bzFuaWpraWpLeGNpbjBoNyIsIm5
vbmNlIjoidWpwMmFzeHlqN2UiLCJhdXRoX3RpbW
UiOjE1MDkwNDk3MTl9.dv4Ek8B4BDee1PcQT_4z
m7kxDEY1sRIGbLoNtlodZcSzHzXU5GkKyl6sAVm
dXOIPUlAIrJAhNfQWQ_XZLBVPjETiZE8CgNg5uq
NmeXMUnYnQmvN5oWlXUZ8GcubAbJ8NQuyBmyec1
j3gmGzX3wemke8NkuI6SX2L4Wj1PyvkknBtbjfi
F9ud1RKbobaFbnjDFOFTzvL6g34SpMmZWy6uc_H
sn4ICx_Ps3FcMwRggCW_o2FpH6rJTOGPZYrOx44
n3ZwAu2dGm6axtPIsqU8b6sw7DaHpogD_hxsXgM
IOzOBMbYsQEiczoGn71ZFz_1O7FiW4dH6g
```

JSON Web Token (JWT)

Calling the /userinfo endpoint

```
GET accounts.google.com/oauth2/userinfo
Authorization: Bearer fFAGRNJru1FTz70BzhT3Zg

200 OK
Content-Type: application/json
{
   "email": "asaad@mum.edu",
   "name": "Asaad Saad"
}
```

What is JSON Web Token?

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

Because of their smaller size, JWTs can be sent through a URL, POST parameter, or inside an HTTP header.

JWT is Self-contained: The payload contains all the required information about the user, avoiding the need to query the database more than once.

45

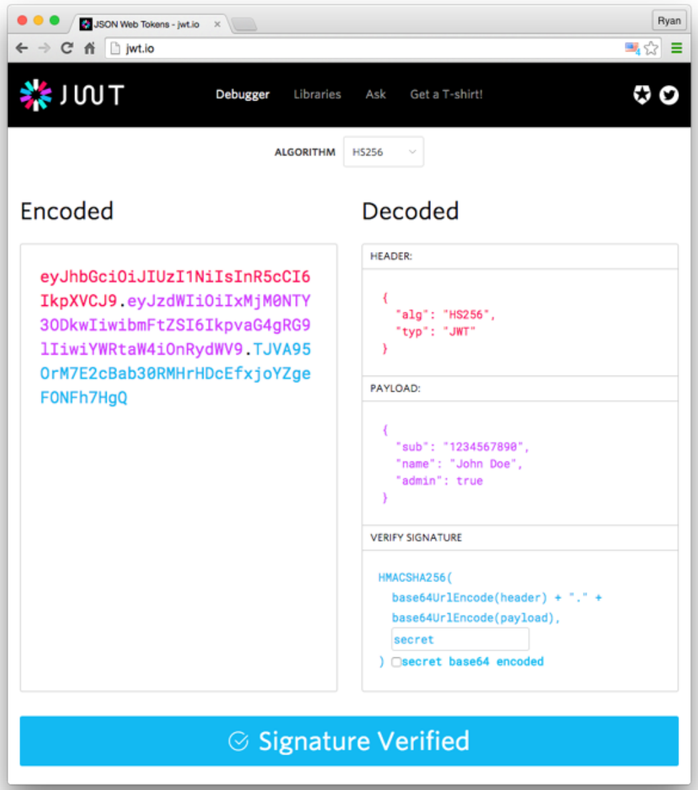
PREV NEXT

What is the JSON Web Token structure?

JSON Web Tokens consist of three parts separated by dots (.)

Header.Payload.Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```



JWT Structure

Header: Where we define the **algorithm** used to sign the token, as well as the token type.

Payload: This is where we keep a JSON object of all the **claims** we want. Claims can include those that are registered in the JWT spec, as well as any arbitrary data we want. Some of the reserved claims are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.

Signature: The signature is where the signing action happens. To get the signature, we take the Base64URL encoded header, add the Base64URL encoded payload next to it, and run that string along with the secret key through the hashing algorithm we've chosen.

PREV NEXT

The Bearer Schema

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the Authorization header using the Bearer schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```

You may save the authorization data for the user in the JWT.

PREV NEXT

How Are JWTs Used to Authenticate Angular Apps?

- Users send their credentials to the server which are verified against a database. If everything checks out, a JWT is sent back to them.
- The JWT is saved in the user's browser by holding it in local storage.
- The presence of a JWT saved in the browser is used as an indicator that a user is currently logged in.
- The JWT's expiry time is continually checked to maintain an authenticated state in the app, and the user's details are read from the payload to populate views such as the their profile.
- Access to protected client-side routes (such as the profile area) are limited to only authenticated users via Guards
- When the user makes XHR requests to the API for protected resources, the request must be intercepted and the JWT gets sent as an Authorization header using the Bearer.
- Middleware on the server, which is configured with the app's secret key checks the incoming JWT for validity and, if valid, sends the response.

PREV NEXT

JWT packages for Angular and Node

Angular

There are several open source libraries for Angular which help us work with JWT:

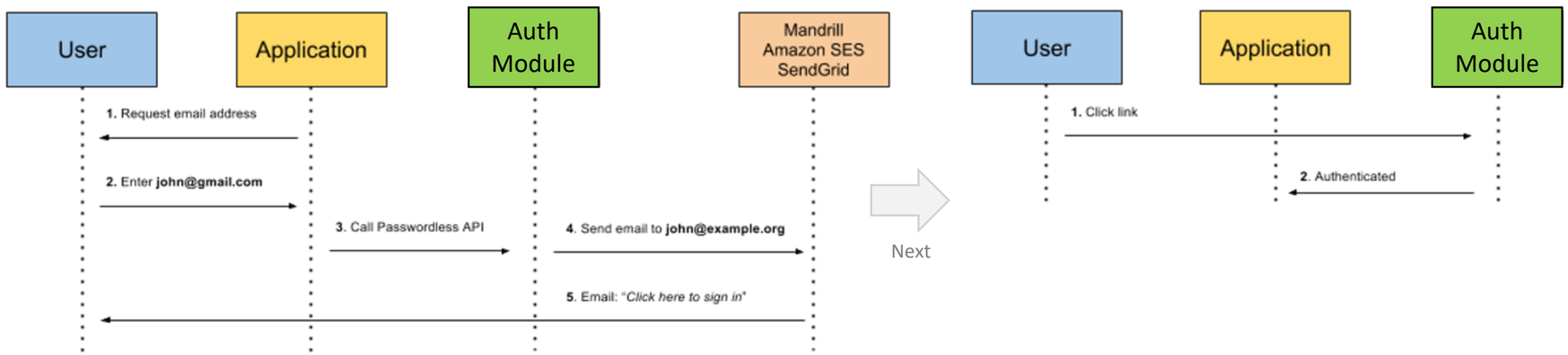
- **`angular2-jwt`**
- **`ng2-ui-auth`**

Node

https://jwt.io/#libraries-io **`npm install jsonwebtoken`**

Passwordless Authentication

Passwordless authentication is a type of authentication where users do not need to login with passwords. Users may log in simply by a magic link, fingerprint, or using a token that is delivered securely via known email or text message.



OTPW One-Time Password

Improve User Experience

The faster users can sign up and use your service, the more users your app tends to attract. Users do not like having to fill out forms and go through a long registration process.

Increase Security

Once you go passwordless, there are no passwords to be hacked.

PREV NEXT

How Passwordless Authentication Works

- Instead of asking users for a password when they try to log in to your app or website, just ask them for their username (or email or mobile phone number).
- Create a temporary authorization code on the backend server and store it in your database.
- Send the user an email or SMS with a link that contains the code.
- The user clicks the link which opens your app or website and sends the authorization code to your server.
- On your backend server, verify that the code is valid and exchange it for a long-lived token, which is stored in your database and sent back to be stored on the client device as well. (There will also be checks on the server to ensure that the link was clicked within a certain period)
- The user is now logged in, and doesn't have to repeat this process again until their token expires or they want to authenticate on a new device.

Why Do We Need a Angular Compiler?

- The TS compiler will convert your TS code (types, decorators and classes) to ES 5/6 friendly code.
- But these function constructors will have templates written in Angular language and need to be converted to JS functionality to handle all the property bindings, change detection and the directives.. etc
- The Angular code **needs to be compiled** into functional JS logic (component factories when called it will generate the DOM elements)
- Finally after compilation the JS code is then sent to V8 to be: parsed into AST, from AST we generate bytecode, this code will be optimized and recompiled before being sent to the CPU.

PREV NEXT

Just-in-time (JIT) compilation

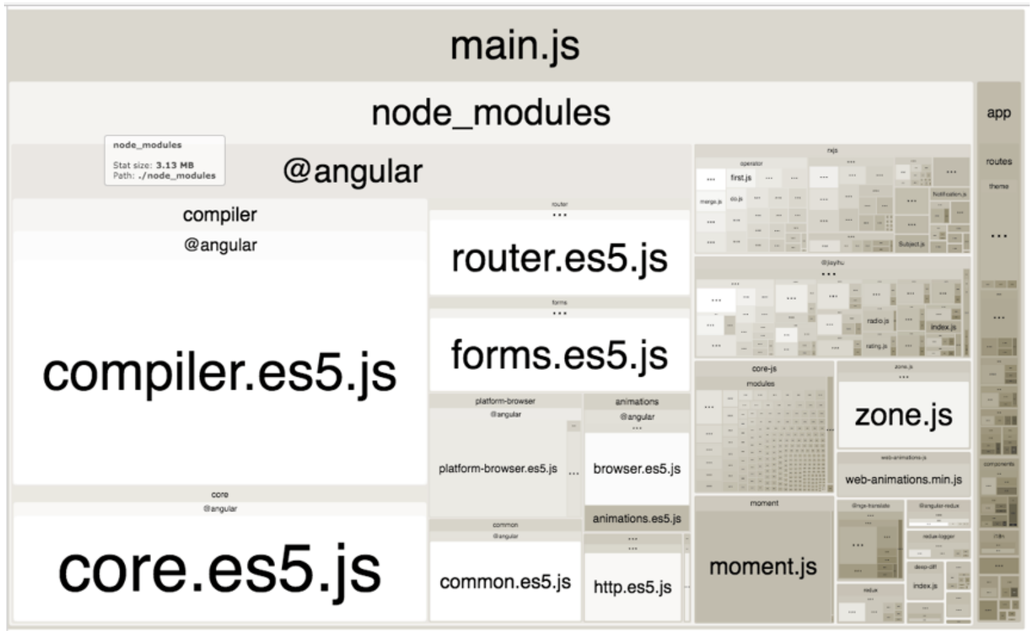
It's known as **dynamic translation**, the compilation is done during execution of a program at runtime. When the user opens the browser, the following steps are performed:

- Download all the JavaScript assets (including the compiler code)
- Angular bootstraps
- Angular goes through the JiT compilation process generating all the JavaScript for each component in the application
- The application gets rendered.

PREV NEXT

Source Code Shipped to Users (JiT)

```
// npm i source-map-explorer -g
source-map-explorer bundle.js bundle.js.map
```



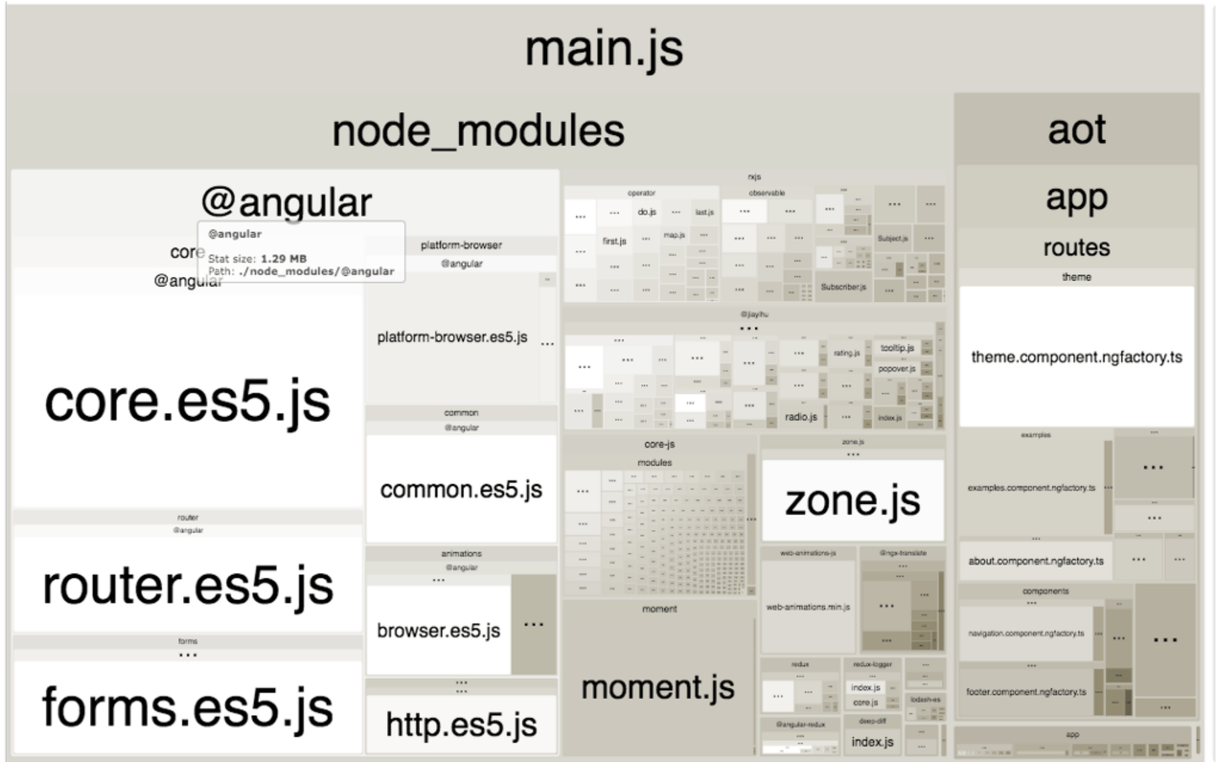
jiayihu.net

Ahead of Time compilation (AOT)

- We no longer have to ship the compiler to the client
- Since the compiled app doesn't have any HTML, and instead has the generated TypeScript code, the TypeScript compiler can analyze it to produce type errors. In other words, your templates are type safe.
- Bundlers can tree shake away everything that is not used in the application. The bundler will figure out which components are used, and the rest will be removed from the bundle.
- Since the most expensive step in the bootstrap of your application is compilation, compiling ahead of time can significantly improve the bootstrap time.

PREV NEXT

Source Code Shipped to Users (AoT)



JiT vs AoT

Name	Status	Type	Initiator	Size	Time	Timeline ▲ Star
localhost	200	docu...	Other	761 B	5 ms	
polyfills.js	200	script	(index):12	1.5 MB	87 ms	
vendor.js	200	script	(index):12	6.7 MB	443 ms	
app.js	200	script	(index):12	641 KB	65 ms	
info?t=1480933562088	200	xhr	zone.js?fad3...	367 B	5 ms	
websocket	101	webs...	VM967:35	0 B	Pendi...	
angular.png	200	png	Other	4.7 KB	18 ms	
ng-validate.js	200	script	content-scri...	(from ...	4 ms	
backend.js	200	script	content-scri...	(from ...	2 ms	

JIT

9 requests | 8.8 MB transferred | Finish: 1.37 s | DOMContentLoaded: 1.31 s | Load: 1.32 s

127.0.0.1	200	docu...	Other	1.7 KB	7 ms
zone.js	200	script	(index):18	57.1 KB	7 ms
app.main.js	200	script	(index):19	437 KB	14 ms
0.app.main.js	200	script	app.main.js:1	3.8 KB	5 ms
angular.png	200	png	Other	4.7 KB	4 ms
ws	101	webs...	VM100:35	0 B	Pendi...
ng-validate.js	200	script	content-scri...	(from ...	2 ms
backend.js	200	script	content-scri...	(from ...	2 ms

AOT

8 requests | 505 KB transferred | Finish: 441 ms | DOMContentLoaded: 302 ms | Load: 437 ms

dzurico.com

87

AOT implications

To make AOT work the application has to have a clear separation of the **static and dynamic data** in the application. And the compiler has to be built in such a way that it **only depends on the static data**.

The information in the decorator is known statically. Angular knows the selector and the template of your component. It also knows that the component has an inputs and outputs.

Since Angular knows all the necessary information ahead of time, it can compile the application components without actually executing any application code, as a build step.

Server Side Rendering

Browsers have to do a heavy work before it can display the first pixel on the screen.

Flow for **server pre-rendering**:

- Generate static HTML with build tool
- Deploy generated HTML to a CDN
- Server view served up by CDN
- Server view to client view transition

https://universal.angular.io

Flow for **server re-rendering**:

- HTTP GET request sent to the server
- Server generates a page that contains rendered HTML and inline JavaScript for Preboot
- Server view to client view transition

SSR: Server View to Client View Transition

Browser receives initial payload from server, user sees **Server View.**

Preboot creates hidden div that will be used for client bootstrap and starts recording events.

Browser makes async requests for additional assets.

Once external resources loaded, Angular client bootstrapping begins.

Client view is rendered to the hidden div created by **Preboot.**

Preboot events replayed in order to adjust the application state to reflect changes made by the user before Angular bootstrapped.

Preboot switches the hidden **Client View** div for the visible **Server View** div and performs some cleanup on the visible **Client View** including setting focus.

The Big Picture

Developer writes code in TS, run **`ng build`**:

- **Just-In-Time**:
  Empty HTML + JS bundle with Angular code (Templates) + compiler
- **Ahead-Of-Time**:
  Empty HTML + JS bundle with pure JS classes (no templates)
- **Server-Side-Rendering**:
  HTML fully rendered component (server view) and fetch client view in background.

Main Points

- The central part of Angular is its compiler.
- The compilation can be done just in time (at runtime) and ahead of time (as a build step).
- The AOT compilation creates smaller bundles, tree shakes dead code, makes your templates type-safe, and improves the bootstrap time of your application.
- The AOT compilation requires certain metadata to be known statically, so the compilation can happen without actually executing the code.

PREV NEXT

Deploy for Production

```
> ng build --prod // When you run the ng build command, it creates a /dist folder.
```

Adding the production flag reduce the bundle size nearly an 83% reduction:

- Removes unwanted white space by minifying files.
- Uglifies files by renaming functions and variable names.
- AoT compilation

The app will work if you're uploading it to the root public folder, such as website.com, but if it's within a sub folder such as website.com/subfolder, you can specify the **`--base-href`** flag during the build process based on the folder structure of where the app will be placed.

If you want to use /app as an application base for router and /public as base for your assets:

```
> ng build --prod --base-href /app --deploy-url /public
```

Differential loading

Provides two groups of bundles: one is based on ES5 and addresses older browsers, the other is based on a ES6 version (ECMAScript 2015), and offers modern browsers with faster code.

1. Set an upper bar for the ECMAScript versions to be supported in the **`tsconfig.json`** as follows: **`"target": "es2015"`**
2. Set a lower bar within **`browserslist`**. It is a file that identifies many browsers to be supported:

```
> 0.5%
last 2 versions
Firefox ESR
not dead
IE 9-11
```

Adding CSS and JavaScript to a Project

Adding external files directly to your index.html file is not a good practice, because files won't get bundled. What we ultimately want to end up with is code that we can ship all as one unit.

If you've already done a build, you should have a folder named "**`dist`**" with all the files you need to run your application. What we want to end up with is ALL of the files we need for our application in that directory all bundled and minified.

When CSS and HTML files get compiled into our bundles, they all end up as JavaScript.

Adding Bootstrap to a Project

```
npm install bootstrap --save-dev
```

Update **angular.json** file to tell the CLI to these files installed:

```
"styles": [
    "./node_modules/bootstrap/dist/css/bootstrap.css",
    "src/styles.css"
],
"scripts": [
    "./node_modules/bootstrap/dist/js/bootstrap.js"
],
```

Alternative Way

An alternate way is to place **`@import`** statements in your application level CSS file (**`styles.css`** in an Angular CLI project). This works best if you are referencing external files (URLs) or files that you can access from your resources directory.

```
@import "~bootstrap/dist/css/bootstrap.min.css";

OR

@import url('https://unpkg.com/bootstrap@3.3.7/dist/css/bootstrap.min.css');
```

Deploy SPA to Github Pages

1. Enable production mode
2. Edit **`.gitignore`** and remove **`"/dist"`** line
3. Run "**`ng build -prod`**" (make sure you have Administrator privilege in your shell)
4. Create a repo on gitHub ("**`repoName`**")
5. Go to "**`dist`**" folder in your project directory and edit file "**`index.html`**"
6. Instead of .. **`<base href="/">`** .. put .. **`<base href="/repoName/">`** .. save & close.
7. From your project directory run commands

```
git add .
git commit -m 'first commit'
git remote add origin REPO-GIT-URL
git subtree push --prefix dist origin gh-pages
```

8. Access your deployed app from **`https://(YourUsername).github.io/(repoName)/`**