Angular 2+

Workshop. Angular Universal.

Contents

Explanation of Colors

Blue color is a snippet of code that you need to fully use to create a new file.

Black color in combination with green or red, means the snippet of code that was added earlier.

Green color is the snippet of code that needs to be added.

Red color is the snippet of code that needs to be removed.

Task 01. Install Dependencies

1. Run the following command from the command line:

```
$> npm install --save @angular/platform-server @nguniversal/express-engine
@nguniversal/module-map-ngfactory-loader express

$> npm install --save-dev @types/express
```

Task 02. Angular Universal Project

1. Create Angular Universal Application. Run the following command from the command line:

```
$> ng g universal --client-project angular-ssr
```

2. Make changes to **angular.json** file. Use the following snippet of code:

```
// 1
"build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/browser",
…
}

// 2
"server": {
   "builder": "@angular-devkit/build-angular:server",
   "options": {
          "outputPath": "dist/server",
          "outputPath": "dist/angular-ssr-server",
          "main": "src/main.server.ts",
          "tsConfig": "src/tsconfig.server.json"
       }
}
```

3. Make changes to **AppServerModule**. Use the following snippet of code:

```
// 1
import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-loader';

// 2
imports: [
    …
    ModuleMapLoaderModule // <-- *Important* to have lazy-loaded routes work
],
```

4. Run the following command for Angular Universal Application. This command should create **dist/server/main.js file**.

```
$> ng run angular-ssr:server
```

5. Make changes to **package.json** file. Add the following snippet of code to the script section:

```
"build:server-app:prod": "./node_modules/.bin/ng run angular-ssr:server",
```

After that you can simply run the following code

```
$> npm run build:server-app:prod
```

instead of

```
$> ng run angular-ssr:server
```

Task 03. Prerendering with renderModuleFactory

1. Create file **prerender.ts** in the root folder of the application. Use the following snippet of code:

```
import 'zone.js/dist/zone-node';
import 'reflect-metadata';

import { writeFileSync, readFileSync, existsSync, mkdirSync } from 'fs';
import { join } from 'path';

import { enableProdMode } from '@angular/core';
// Faster server renders w/ Prod mode (dev mode never needed)
enableProdMode();

// Import module map for lazy loading
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-loader';
import { renderModuleFactory } from '@angular/platform-server';

const ROUTES = ['/home', '/users', '/admin'];
// * NOTE :: leave this as require() since this file is built Dynamically from webpack
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } = require('./dist/server/main');
const BROWSER_FOLDER = join(process.cwd(), join('dist', 'browser'));
// Load the index.html file containing referances to your application bundle.
const index = readFileSync(join('dist', 'browser', 'index.html'), 'utf8');

let previousRender = Promise.resolve();

// Iterate each route path
ROUTES.forEach(route => {
  const fullPath = join(BROWSER_FOLDER, route);

  // Make sure the directory structure is there
  if (!existsSync(fullPath)) {
    mkdirSync(fullPath);
  }

  // Writes rendered HTML to index.html, replacing the file if it already exists.
  previousRender = previousRender
    .then(_ =>
      renderModuleFactory(AppServerModuleNgFactory, {
        document: index,
        url: route,
        extraProviders: [provideModuleMap(LAZY_MODULE_MAP)]
      })
    )
    .then(html => writeFileSync(join(fullPath, 'index.html'), html));
});
```

2. Install utility **http-server**. Run the following command from the command line:

```
$> npm i -g http-server
```

3. Make changes to **package.json** file. Add the following snippet of code to the script section:

```
"prerender": "./node_modules/.bin/ts-node ./prerender.ts",
"start:http-server": "http-server ./dist/browser"
```

4. Run the following command from the command line:

```
$> npm run prerender
```

5. Run the following command from the command line:

```
$> npm run http
```

Task 04. NgUniversal Express Engine

1. Create new file **server.ts** in the root folder of the project. Use the following snippet of code:

```
import 'zone.js/dist/zone-node';
import 'reflect-metadata';
import { enableProdMode } from '@angular/core';
import { join } from 'path';

// Express Engine
import * as express from 'express';
import { ngExpressEngine } from '@nguniversal/express-engine';

// Import module map for lazy loading
import { provideModuleMap } from '@nguniversal/module-map-ngfactory-loader';

// Faster server renders w/ Prod mode (dev mode never needed)
enableProdMode();

// * NOTE :: leave this as require() since this file is built Dynamically from webpack
const {
  AppServerModuleNgFactory,
  LAZY_MODULE_MAP
} = require('./dist/server/main');

const app = express();
const PORT = process.env.PORT || 4000;
const BROWSER_FOLDER = join(process.cwd(), 'dist', 'browser');

// Our Universal express-engine (found @
https://github.com/angular/universal/tree/master/modules/express-engine)
// Define new rendering engine. We call it 'html'.
app.engine(
  'html',
  ngExpressEngine({
    // Here we specify AppServerModuleNgFactory
    bootstrap: AppServerModuleNgFactory,
    // Configure Angular Universal specific providers for server, not for client
    providers: [
      provideModuleMap(LAZY_MODULE_MAP)
      // In case you want to use an AppShell with SSR and Lazy loading
      // you'd need to uncomment the below. (see: https://github.com/angular/angular-
cli/issues/9202)
      // {
      //   provide: NgModuleFactoryLoader,
      //   useClass: ModuleMapNgFactoryLoader,
      //   deps: [
      //     Compiler,
      //     MODULE_MAP
      //   ],
      // },
    ]
  })
);

// Any view which is rendered by express should be rendered by 'html' engine.
app.set('view engine', 'html');
```

```
// Configure the place of view files
app.set('views', BROWSER_FOLDER);

// Example Express Rest API endpoints
// app.get('/api/**', (req, res) => { });

// Server static files from /dist/browser
app.get(
  '*.*', // *.js and *.css files
  express.static(BROWSER_FOLDER, {
    maxAge: '1y' // cache these files up to 1 year
  })
);

// All regular routes use the Universal Express Engine
// /, /home, /users, ...
app.get('*', (req, res) => {
  res.render('index', { req }); // index - is the view that we want to render
});

// Start up the Node server
app.listen(PORT, () => {
  console.log(`Node Express server listening on http://localhost:${PORT}`);
});
```

2. Make changes to **package.json** file. Add the following snippet of code to the script section:

```
"start:express-server": "./node_modules/.bin/ts-node ./server.ts"
"universal": "npm run build & npm run build:server-app:prod & npm run start:express-
server",
```

3. Run the following command from the command line:

```
$> npm run universal
```

4. Open http://localhost:4000

Task 05. SEO and Social Media Crowlers

1. Make changes to the **AppComponent**. Use the following snippet of code:

```
private setPageTitlesAndMeta() {
…

.subscribe(data => {
        if (data['title']) {
          this.titleService.setTitle(data['title']);
          this.metaService.addTags(data['meta']);
        }
      });
}
}
```

2. Goto the link http://localhost:4000/edit/1 open Elements tab in console. You can see empty <title> tag. Press F5 to reload app and you should see the "Angular SSR" title.
3. Make changes to the **TaskFormComponent**. Use the following snippet of code:

```
// 1
import { Title, Meta } from '@angular/platform-browser';

// 2
constructor(
    …
    private titleService: Title,
    private metaService: Meta
  ) {}

// 3
ngOnInit(): void {

.subscribe(
        // when Promise.resolve(null) => task = null => {...null} => {}
        task => (this.task = { ...task }),
       task => {
                 this.task = { ...task };
                 this.titleService.setTitle(`Task: ${this.task.action}`);
                 this.metaService.addTag({
                   name: 'description',
                   content: `Task: ${this.task.action}, hours: ${this.task.estHours}`
                 });
               },
}
```

4. Make changes to the **TasksRoutingModule.** Use the following snippet of code:

```
// 1
const metaTags = [
  {
    name: 'description',
    content: 'Task Manager Application. This is an ASP application'
  },
  {
    name: 'keywords',
    content: 'Angular 7 tutorial, SPA Application, Routing'
  },
  {
    name: 'twitter:title',
    content: 'Angular 7 tutorial, SPA Application, Routing'
  },
  {
    name: 'twitter:image',
    content:

'https://upload.wikimedia.org/wikipedia/commons/thumb/c/cf/Angular_full_color_logo.svg/1
200px-Angular_full_color_logo.svg.png'
  },
  {
    name: 'twitter:url',
    content: 'https://my-site.my-hosting.com'
  },
  {
    name: 'twitter:description',
    content: 'Task Manager Application. This is an ASP application'
  },
  {
    name: 'twitter:card',
    content: 'summary'
  },
  {
    name: 'twitter:site',
    content: '@Tutor'
  }
];

// 2
{
    path: 'home',
    component: TaskListComponent,
    data: {
      title: 'Task Manager',
      meta: metaTags
      meta: [{
        name: 'description',
        content: 'Task Manager Application. This is an ASP application'
      },
      {
        name: 'keywords',
        content: 'Angular 4 tutorial, SPA Application, Routing'
      }]
    }
  },
```

5. Run the following command from the command line:

```
$> npm run universal
```

6. Goto the link http://localhost:4000/ and http://localhost:4000/edit/1. Open View Page Source. You can see title tag and meta description tag with populated data. Press F5 to reload app and you should see the same value rendered on the server.

Task 06. Application Shell

1. Make changes to **AppComponent Template**. Use the following snippet of HTML:

```
<div class="col-md-10">
     <router-outlet
             (activate)='onActivate($event)'
             (deactivate)='onDeactivate($event)'>
     </router-outlet>
     <!-- Routed views go here -->

     <div class="spinner-container" *appShellRender>
          <app-spinner></app-spinner>
     </div>
</div>
```

2. Make changes to the file **app.component.css.** Use the following snippet of CSS:

```
.spinner-container {
  display: flex;
  justify-content: center;
  align-content: center;
}
```

3. Make changes to **TaskListComponent Template**. Use the following snippet of HTML:

```
<div *appShellNoRender>
    <button class="btn btn-primary"
        (click)="onCreateTask()">New Task</button>
    <br><br>
…
</div>
```

4. Create directives ***appShellNoRnder** and ***appShellRender**. Run the following commands from the command line:

```
$> ng g d shared/directives/shell-no-render --export true -m shared --spec false
$> ng g d shared/directives/shell-render --export true -m shared --spec false
```

5. Make changes to **TasksModule**. Use the following snippet of code:

```
// 1
import { SharedModule } from '../shared/shared.module';

// 2
imports: [
    CommonModule,
    FormsModule,
    SharedModule,
    …
],
```

6. Make changes to **AppShellNoRenderDirective**. Use the following snippet of code:

```
import { Directive, ViewContainerRef, TemplateRef, PLATFORM_ID, OnInit, Inject } from
'@angular/core';
import { isPlatformServer } from '@angular/common';

@Directive({
  selector: '[appShellNoRender]'
})
export class AppShellNoRenderDirective implements OnInit {

  constructor(
    private viewContainer: ViewContainerRef,
    private templateRef: TemplateRef<any>,
    @Inject(PLATFORM_ID) private platformId
  ) { }

  ngOnInit() {
    if (isPlatformServer(this.platformId)) {
      this.viewContainer.clear();
    } else {
      this.viewContainer.createEmbeddedView(this.templateRef);
    }
  }
}
```

7. Make changes to **AppShellRenderDirective**. Use the following snippet of code:

```
import { Directive, ViewContainerRef, TemplateRef, PLATFORM_ID, OnInit, Inject } from
'@angular/core';
import { isPlatformServer } from '@angular/common';

@Directive({
  selector: '[appShellRender]'
})
export class AppShellRenderDirective implements OnInit {

  constructor(
    private viewContainer: ViewContainerRef,
    private templateRef: TemplateRef<any>,
    @Inject(PLATFORM_ID) private platformId
  ) { }

  ngOnInit() {
    if (isPlatformServer(this.platformId)) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }
}
```

Task 07. TransferState

1. Make changes to **AppModule**. Use the following snippet of code:

```
// 1
import { BrowserModule, BrowserTransferStateModule } from '@angular/platform-browser';

// 2
imports: [
…
    BrowserTransferStateModule
]
```

2. Make changes to **AppServerModule**. Use the following snippet of code:

```
// 1
import { ServerModule, ServerTransferStateModule } from '@angular/platform-server';

// 2
imports: [
…
    ServerTransferStateModule
]
```

3. Make changes to TaskFormComponent. Use the following snippet of code:

```
// 1
import { Component, OnInit, Inject, PLATFORM_ID } from '@angular/core';
import { Title, Meta, TransferState, makeStateKey } from '@angular/platform-browser';
import { of } from 'rxjs';
import { isPlatformServer } from '@angular/common';

// 2
constructor(
    …
    @Inject(PLATFORM_ID) private platformId: any,
    private transferState: TransferState
) {}

// 3 ngOnInit method
switchMap((params: Params) => {
          return params.get('taskID')
            ? this.taskPromiseService.getTask(+params.get('taskID'))
            : Promise.resolve(null);
          // edit form
          if (params.has('taskID')) {
            const taskId = params.get('taskID');
            const transferKey = makeStateKey<TaskModel>(`task-${taskId}`);

            // if there is something in transfer state use it
            if (this.transferState.hasKey(transferKey)) {
              // take it. second param is a default value
              const task = this.transferState.get<TaskModel>(transferKey, null);
              // remove it from transfer state
              this.transferState.remove(transferKey);
              // return result
              return of(task);
            }
            // if there is nothing in transfer state call server api
            else {
              return this.taskPromiseService
                .getTask(+params.get('taskID'))
                .then(task => {
                  // if platform is server, additionally save data to transfer state
                  if (isPlatformServer(this.platformId)) {
                    this.transferState.set(transferKey, task);
                  }

                  return task;
                });
            }
          }

          // add form
          else {
            return Promise.resolve(null);
          }
})
```

4. Run application. Use the following command from the command line:

```
$> npm run universal
```

5. Goto http://localhost:4200/edit/1 Open Network Tab. Set up filter /1. Refresh the page and calculate the number of requests to /1 It should be one.