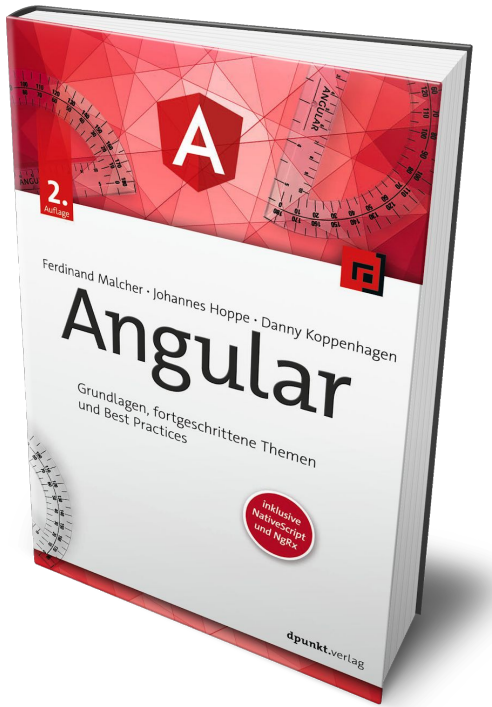
# Server-Side Rendering

## and Pre-Rendering with Angular Universal

**Ferdinand Malcher**

Angular.Schule

# Sponsors and Supporters


# Book raffle

Send a tweet with hashtag
**#ngLeipzig**

**Be the lucky winner!**

# Official Hashtag

#ngLeipzig

# Icebreaker Round

## Introduce yourself and your 3 hashtags

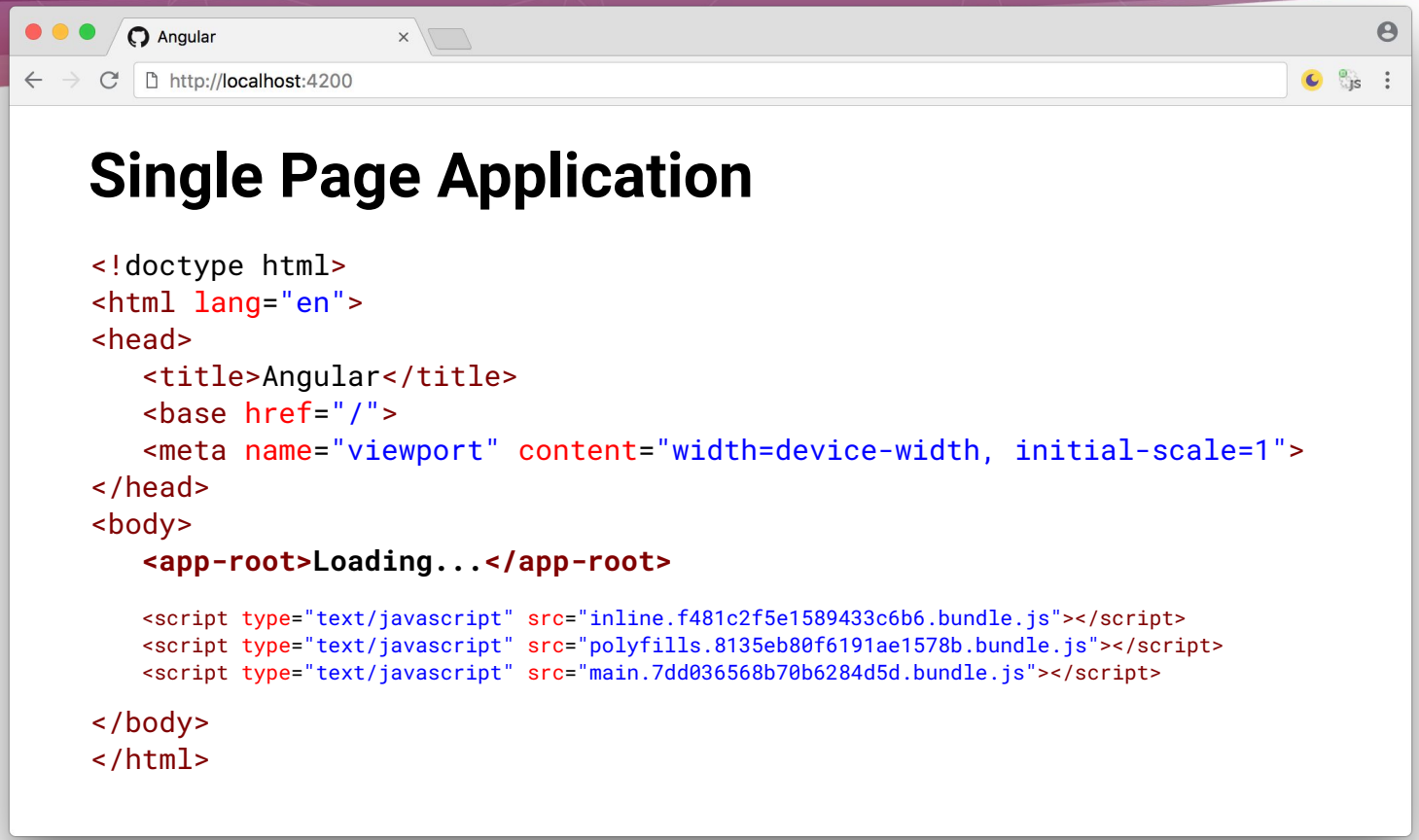
# Server-Side Rendering

## and Pre-Rendering with Angular Universal

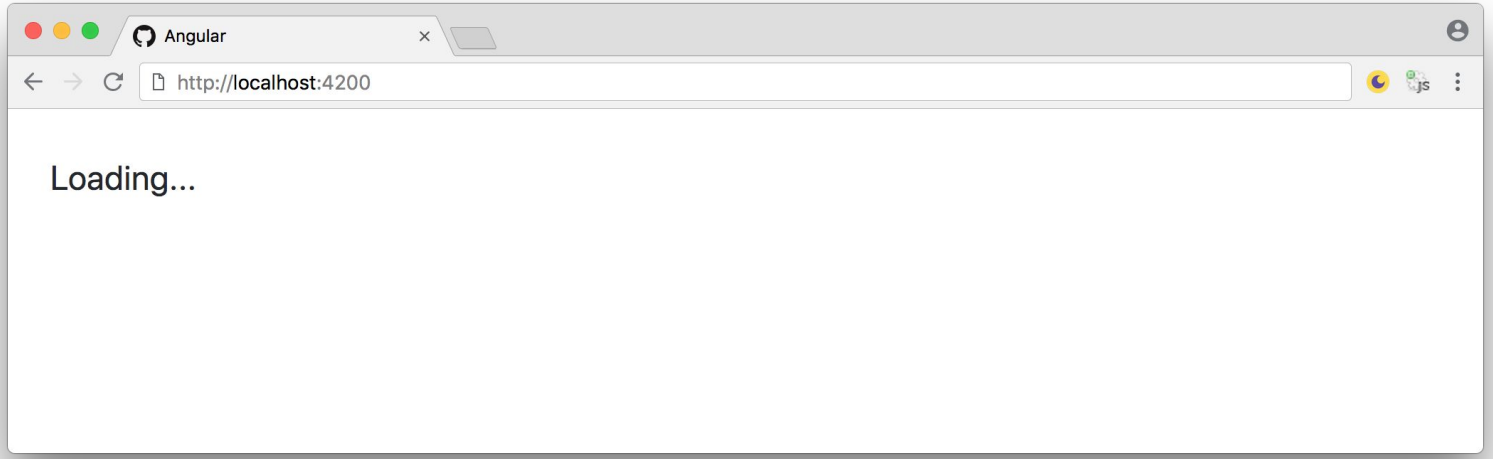
**Ferdinand Malcher**

Angular.Schule

# Single Page Application


Angular
http://localhost:4200
Single Page Application
<!doctype html>
<html lang="en">
<head>
<title>Angular</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<app-root>Loading...</app-root>
<script type="text/javascript" src="inline.f481c2f5e1589433c6b6.bundle.js"></script>
<script type="text/javascript" src="polyfills.8135eb80f6191ae1578b.bundle.js"></script>
<script type="text/javascript" src="main.7dd036568b70b6284d5d.bundle.js"></script>
</body>
</html>

# Search engines don’t like rendering JavaScript



# Site Preview

**angular.schule** @angular_schule · Jun 2

📖 Blogged: "5 useful NgRx effects that don't rely on actions" by @fmalcher01 🔎 Did you know that effects do not need to take actions as their source? 💥
angular.schule/blog/2018-06-5 …
#angular #ngrx #rxjs #effects #redux

**Angular.Schule → 5 useful NgRx effects that don't rely on actions**
🚀 In this article we will discuss how we can leverage the power of Effects in NgRx. We will use observable streams other than the usual
angular.schule

12 11

**Angular.Schule**
Published by Johannes Hoppe · 15 June at 12:38 ·

📖 Blogged: "Generating #Angular API clients with #Apollo and #GraphQL code generator" by Johannes Hoppe

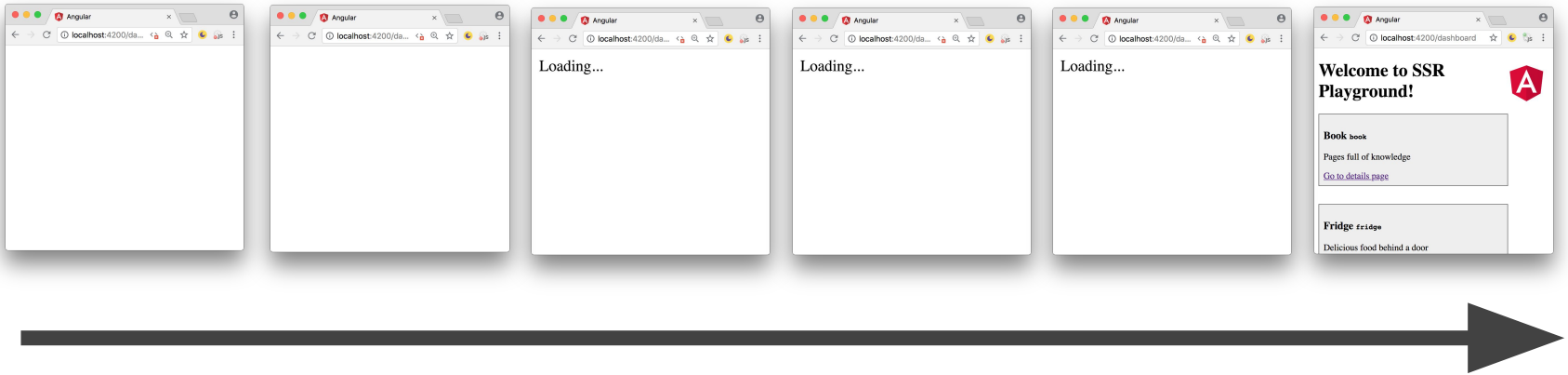
🔎 https://angular.schule/.../2018-06-apollo-graphql-code-genera...
#TypeScript

More on this next week at #ejs18!

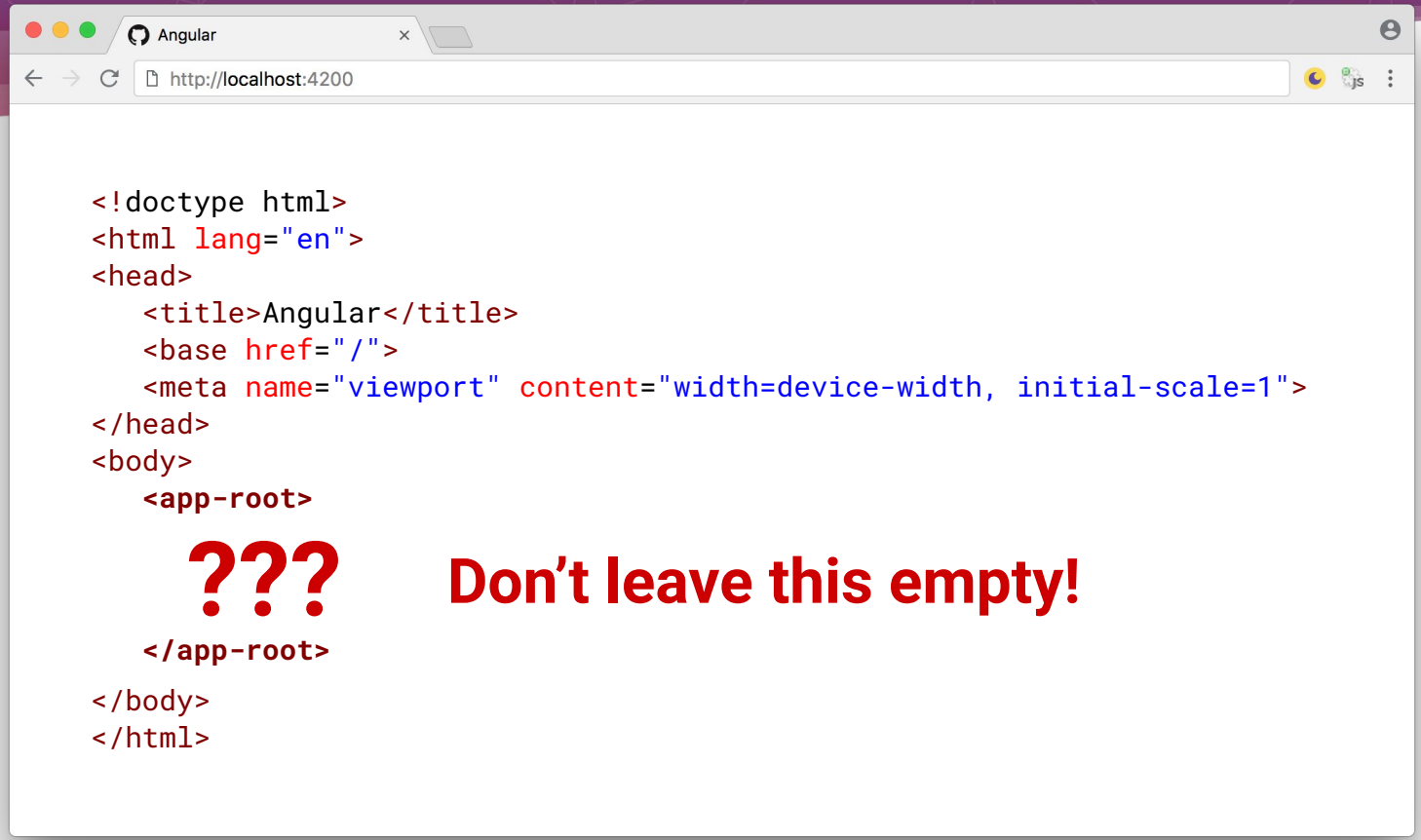

Angular.Schule → Generating Angular API clients with Apollo and GraphQL code generator
🚀 In this article, I will give a short introduction to GraphQL and then we shall look at at Apollo Angular and the GraphQL code generator. We will combine...
ANGULAR.SCHULE

100 people reached Boost Post

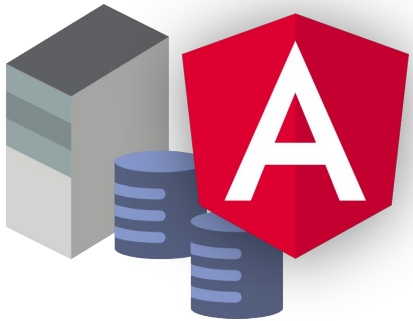
Like Comment Share




# First Meaningful Paint


Angular
http://localhost:4200
<!doctype html>
<html lang="en">
<head>
<title>Angular</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
<app-root>
???
Don’t leave this empty!
</app-root>
</body>
</html>

# Server-Side Rendering



> There's a pretty cool mechanism to extract static trees from components into an optimized format – so that the user can enjoy the non-interactive parts before the rest of the app logic loads.
> *It's called Server-Side Rendering.*

— Dan Abramov

# Strategies

Pre-Rendering with a real browser

Dynamic Server-Side Rendering

Static Pre-Rendering

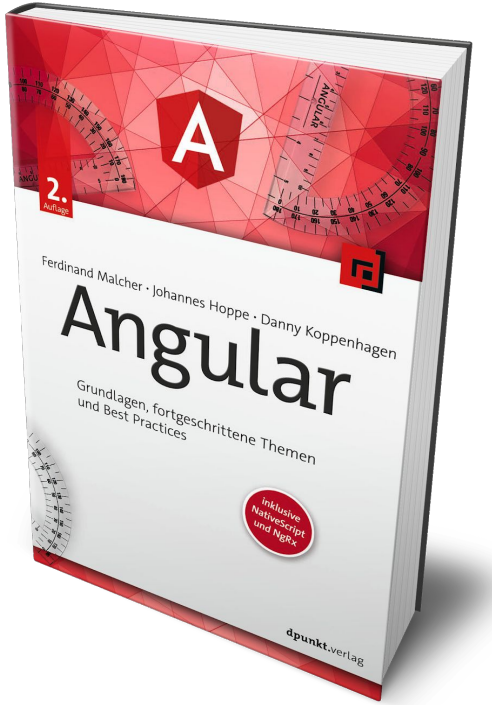
# Hallo, mein Name ist Ferdinand.

Google Developer Expert (GDE)
Angular Book Author
Conference Speaker
Trainer and Co-founder of Angular.Schule

fmalcher01

Google Developers Experts

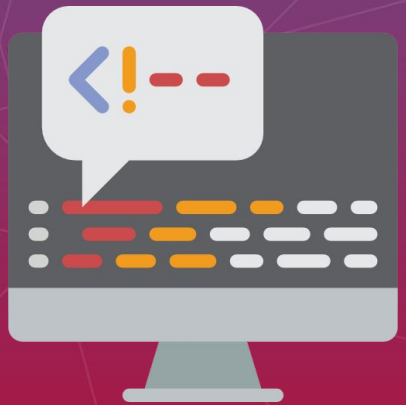
Angular. Schule

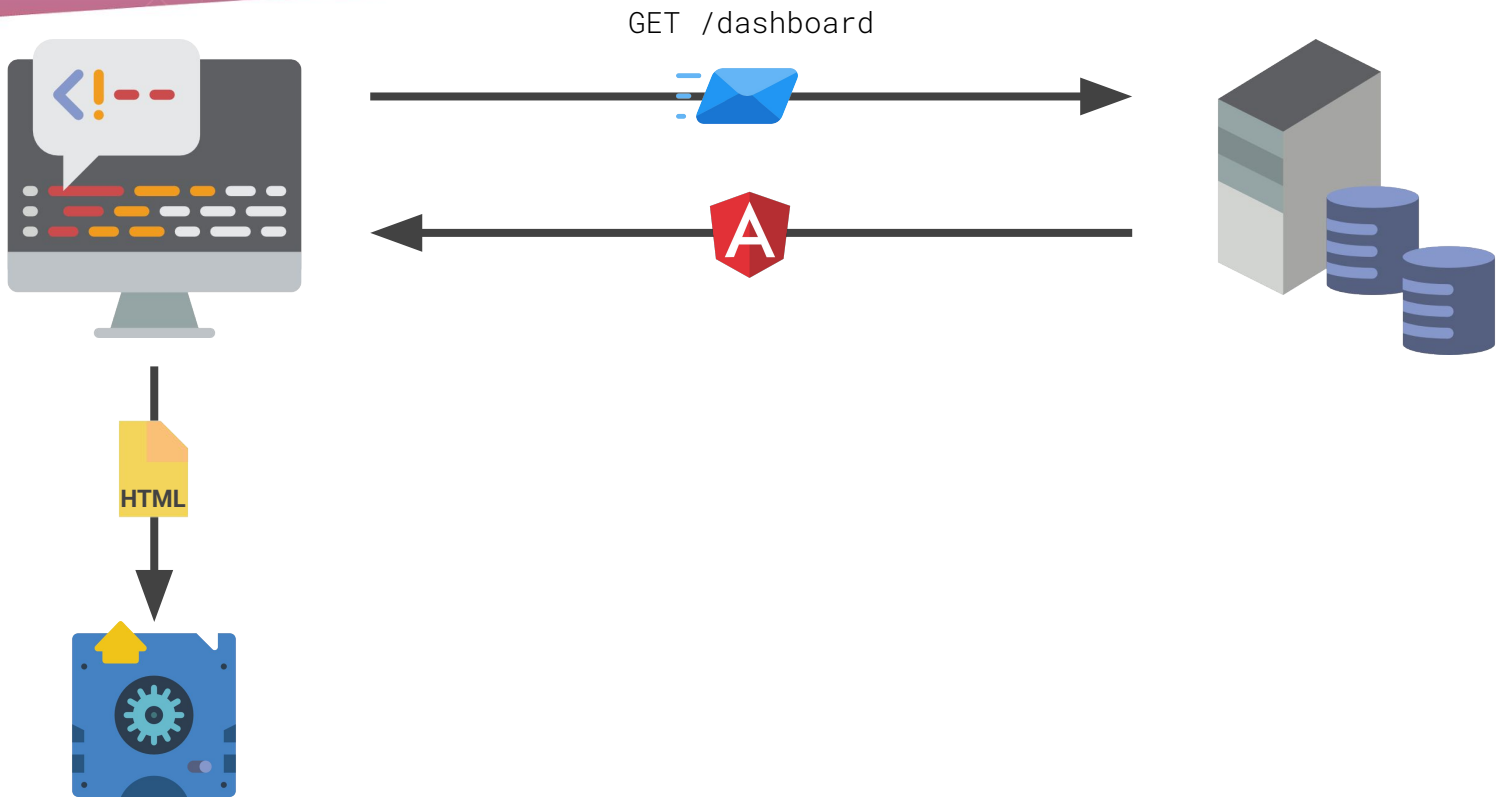



# Angular. Schule

# Pre-Rendering

## with a real browser

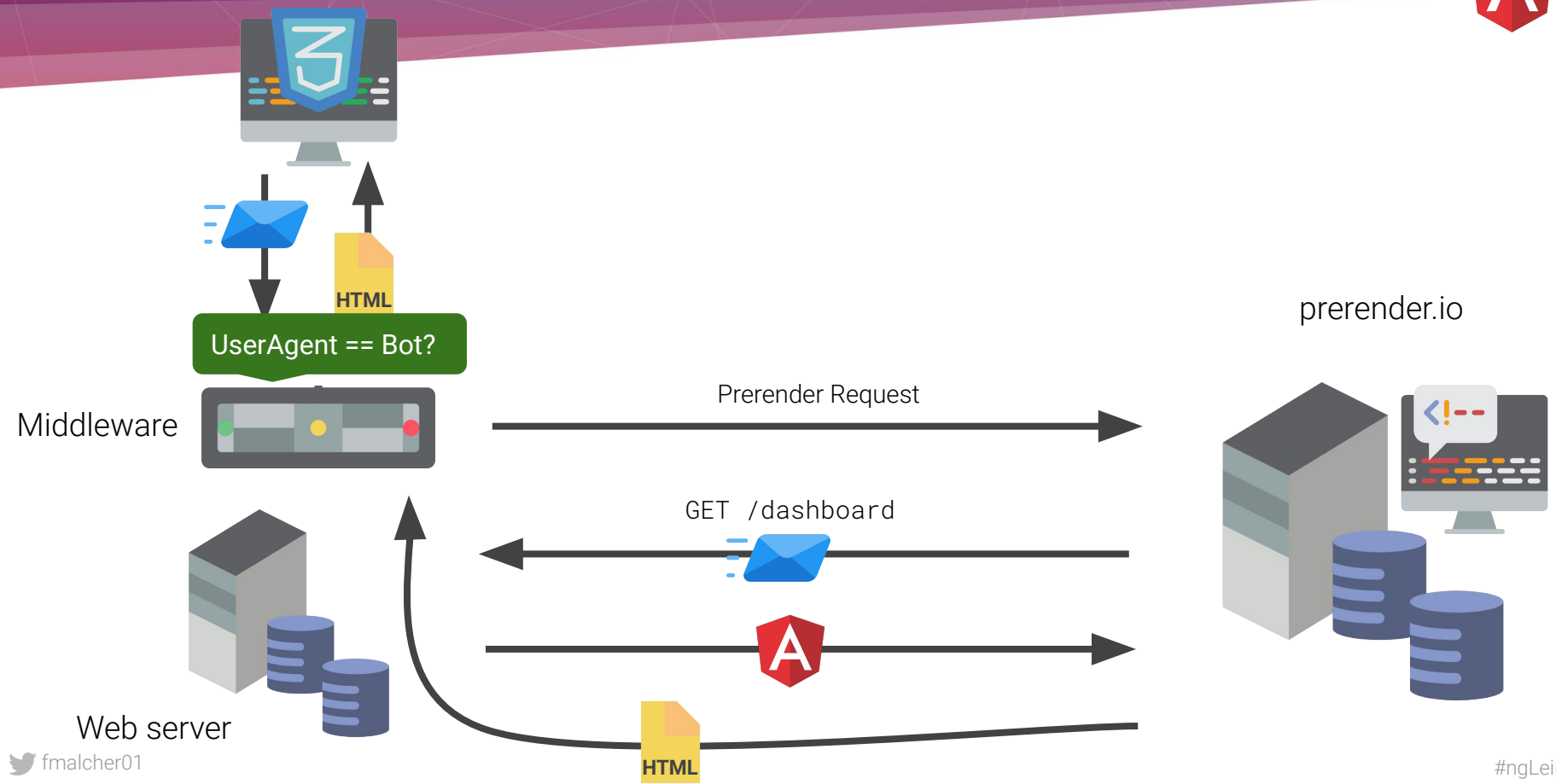



GET /dashboard
HTML

PRERENDER.IO

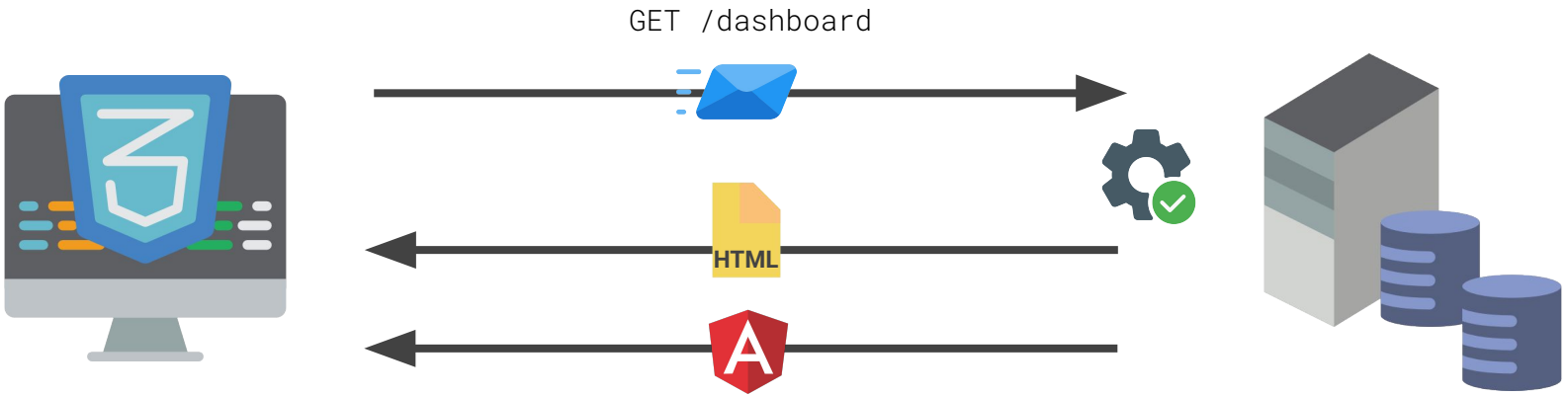
# ROCKET SCIENCE *for* JAVASCRIPT SEO

# Prerender.io: Middleware for the web server



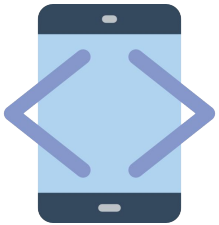
# Server-Side Rendering

# Server-Side Rendering



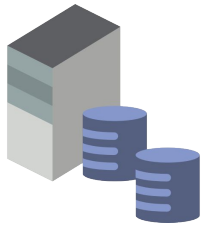
# Angular on all platforms

Browser



Mobile

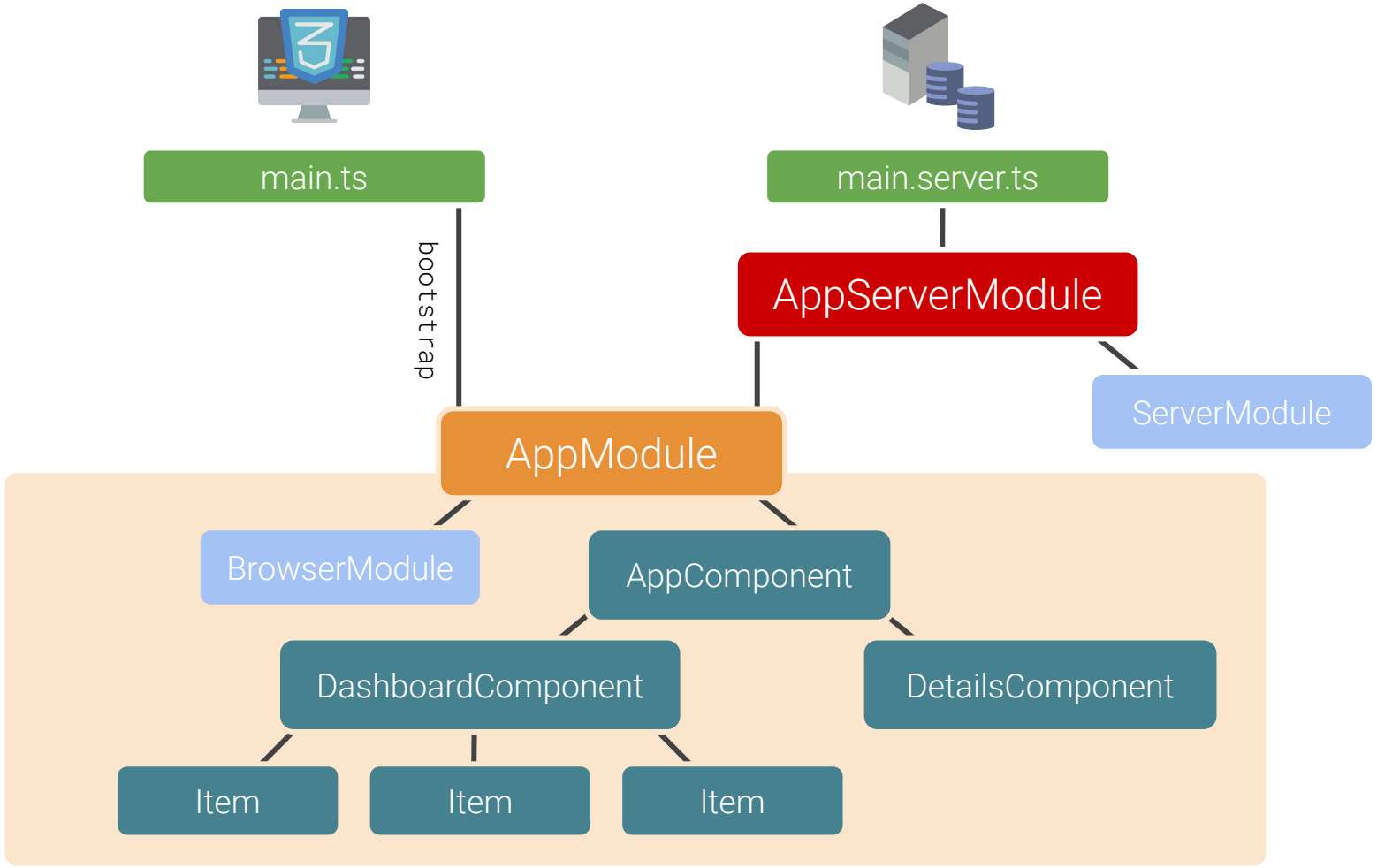
IoT



Server

# Angular Universal

`@angular/platform-server`


main.ts
main.server.ts
bootstrap
AppServerModule
ServerModule
AppModule
BrowserModule
AppComponent
DashboardComponent
DetailsComponent
Item
Item
Item

```
// ...
import { ServerModule } from '@angular/platform-server';

@NgModule({
  imports: [
    AppModule,
    ServerModule
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule {}
```

app.server.module.ts



AppServerModule encapsulates whole application

```
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';

if (environment.production) {
 enableProdMode();
}

export { AppServerModule } from './app/app.server.module';
```

main.server.ts

main.server.ts: Entry point for the server build

```
@NgModule({
  imports: [
    BrowserModule.withServerTransition({ appId: 'serverApp' })
  ],
  // ...
})
export class AppModule { }
```

app.module.ts

Prepare server transition in client application

```
"server": {
  "builder": "@angular-devkit/build-angular:server",
    "options": {
      "outputPath": "dist/ssr-playground/server",
      "main": "server.ts",
      "tsConfig": "tsconfig.server.json"
    },
    "configurations": {
      "production": {
        "sourceMap": false,
        "optimization": true
      }
    }
  }
}
// ...
```

angular.json

# Build target in CLI config

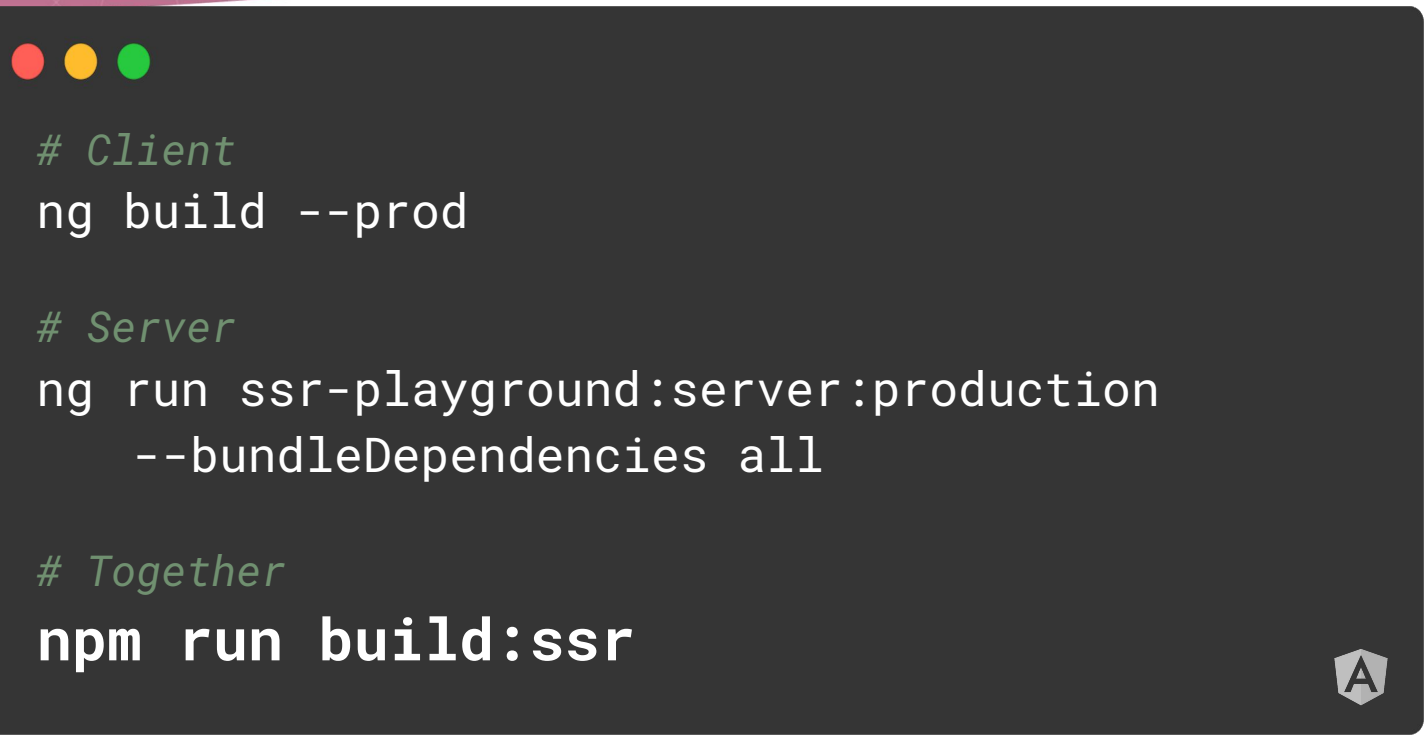
# Other necessary changes

| | |
|---|---|
| `package.json` | Add packages for Angular Universal |
| `tsconfig.server.json` | TypeScript config for the server build |

# Set up Angular Universal


Add @next if you want to work with the latest RC version!

# Build the application

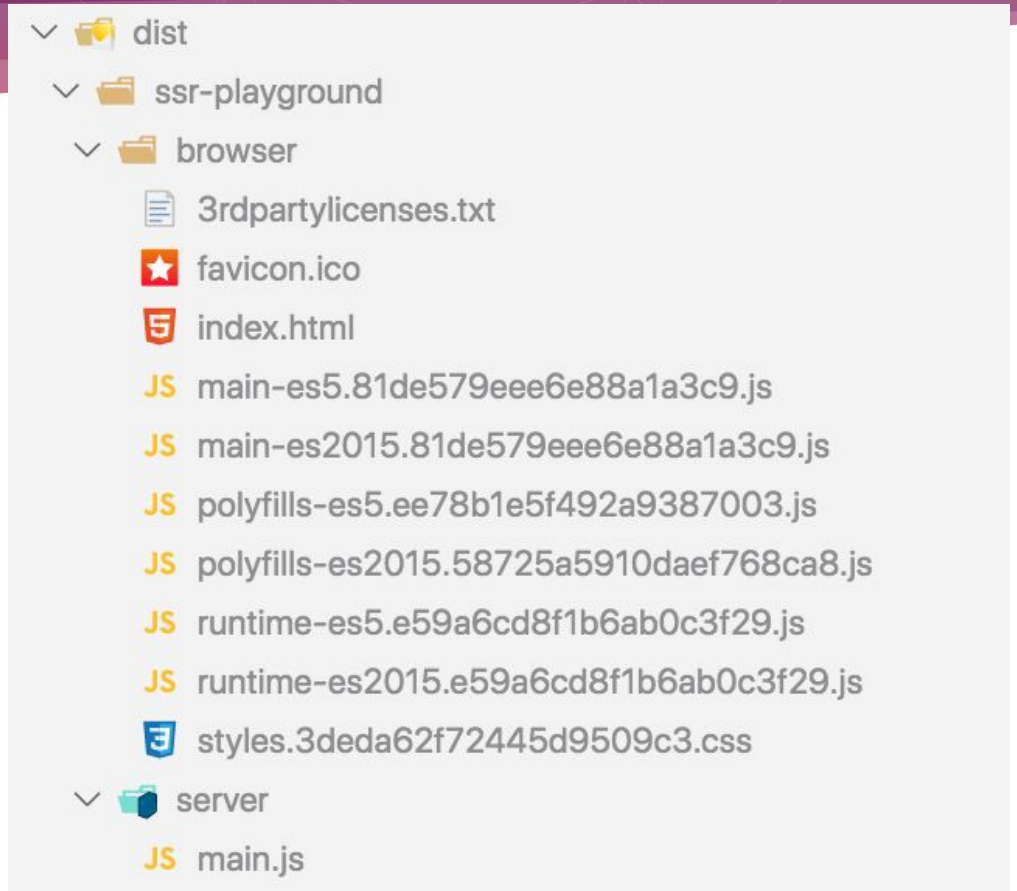


```
# Client
ng build --prod

# Server
ng run ssr-playground:server:production
    --bundleDependencies all

# Together
npm run build:ssr
```

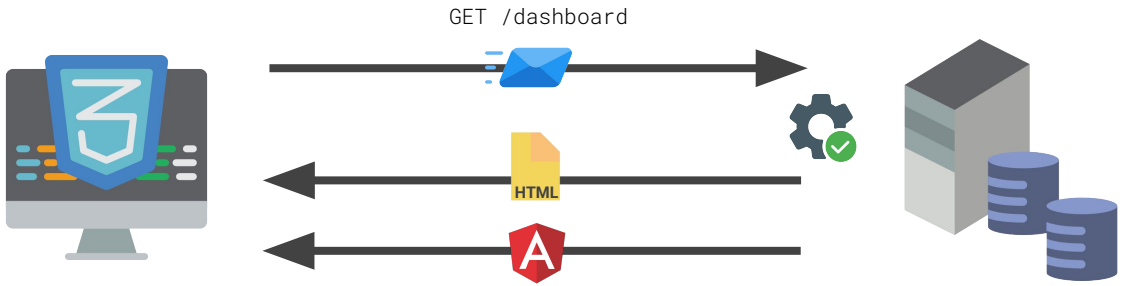
# Application bundles after build




# Duties of the server



- Render application and deliver generated HTML
- Provide application bundles via web server




node
JS


.NET
Core


spring
boot

# The Node.js server (server.ts) 1/4

```
// Express server
const server = express();

// ...

server.listen(4000, () => {
 console.log(`Listening on http://localhost:4000`);
});
```

Simple Express server setup

# The Node.js server (server.ts) 2/4

```
// Serve static files from /dist/browser
server.get('*.*', express.static(distFolder, {
  maxAge: '1y'
}));
```

Serve static files via web server

# The Node.js server (server.ts) 3/4

```
import { ngExpressEngine } from '@nguniversal/express-engine';
import { AppServerModule } from './src/main.server';

// Our Universal express-engine
server.engine('html', ngExpressEngine({
   bootstrap: AppServerModule,
}));
```

Register Express engine for Angular Universal

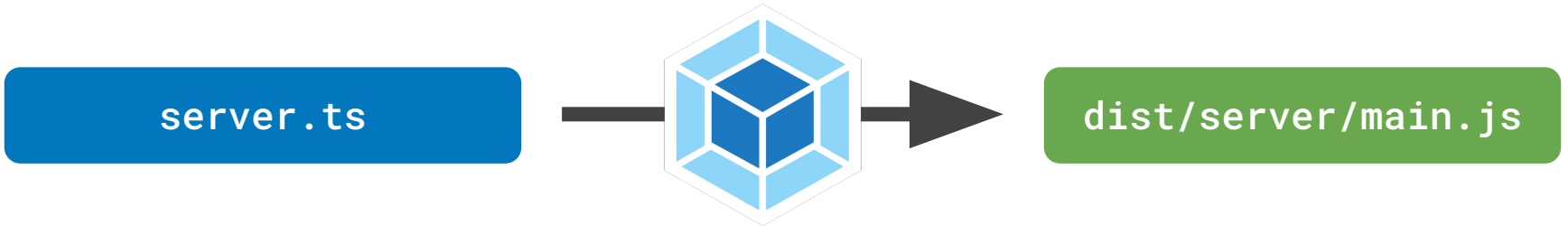
# The Node.js server (server.ts) 4/4

```
server.set('view engine', 'html');
server.set('views', distFolder);

// All regular routes use the Universal engine
app.get('*', (req, res) => {
  res.render('index', { req });
});
```

Use engine to render Angular into index.html

# Compile and bundle the server process



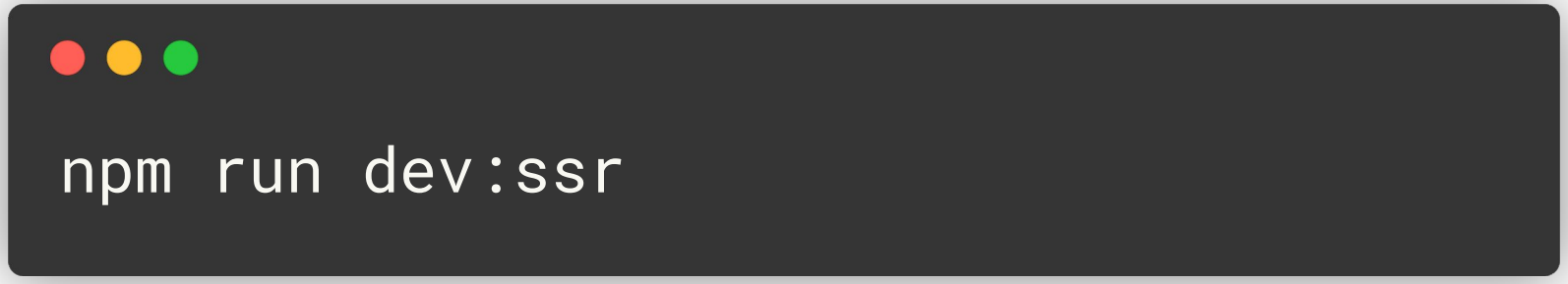

The server process has already been bundled with the application.

# Start server with SSR

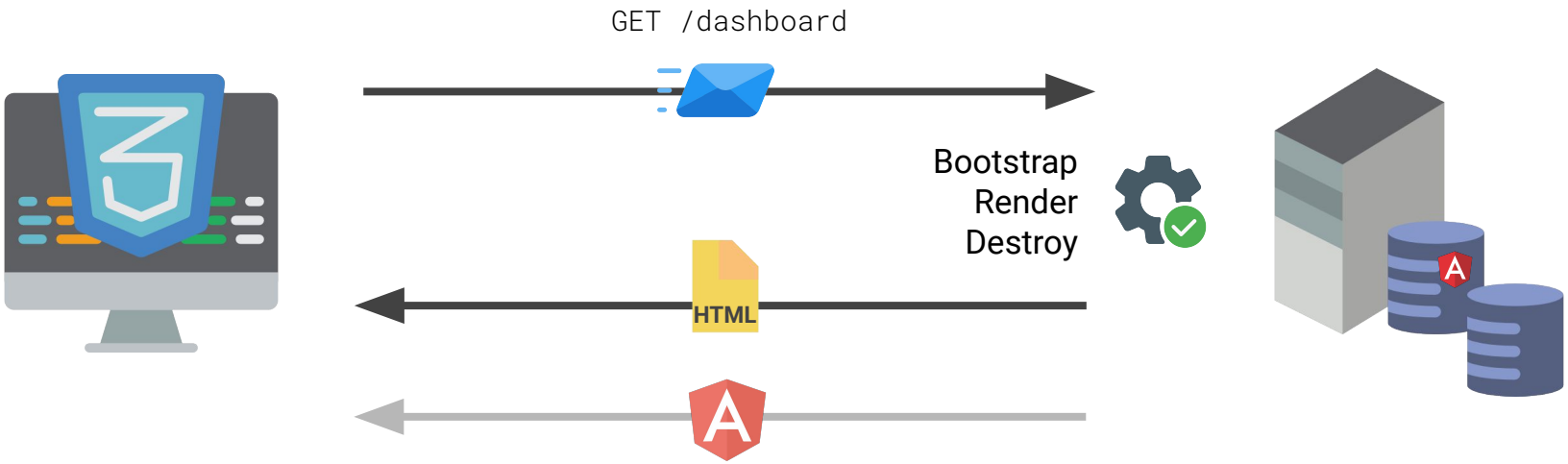
```
node dist/ssr-playground-server/main

# OR
npm run serve:ssr
```

# Dev server with SSR and live reload

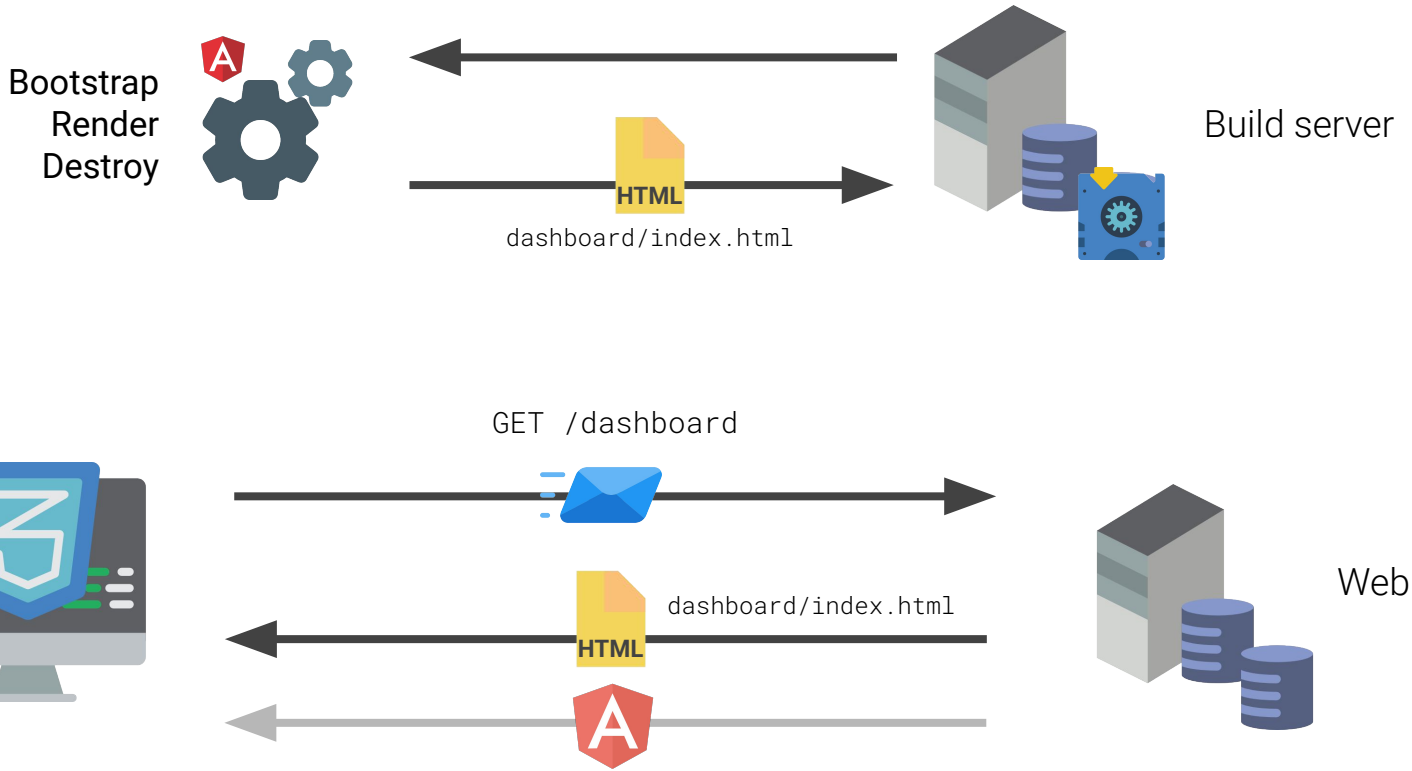


# Server-Side Rendering



# Static Pre-Rendering

# Static Pre-Rendering



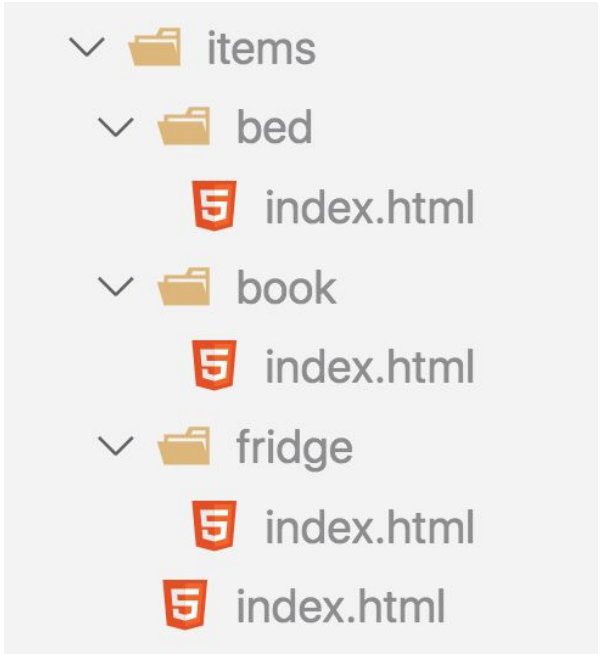
# Pre-Rendering

- 😌 Less computing work at runtime
- 😌 No Node.js needed on the server
- 😕 No dynamic content
- 😕 Routes need to be known at build time

# Hybrid approach

- Pre-Rendering for static content
- Server-Side Rendering for dynamic parts

```
const routes = [
    '/items',
    '/items/bed',
    '/items/book',
    '/items/fridge'
];
```



# Returns rendered HTML for one route

```
import { renderModule } from '@angular/platform-server';
await renderModule(
    AppServerModule,
    {
      document: '<app-root></app-root>',
      url: '/items'
    });
```

# Pre-Rendering: Manual Setup 1/2

```
// ...
const distFolder = path.join(process.cwd(), 'dist', 'ssr-playground');
const indexHtml = fs.readFileSync(path.join(distFolder, 'index.html'))
  .toString();

const routes = ['/', '/items', '/items/bed', /* ... */];
```

# Pre-Rendering: Manual Setup 2/2

```
routes.forEach(async (route) => {
 const html = await renderModule(
   AppServerModule,
   { document: indexHtml, url: route }
 );

 const folder = path.join(distFolder, route);
 mkdirp.sync(folder);
 fs.writeFileSync(path.join(folder, 'index.html'), html);
});
```

**for each route**

**start and render**

**save to file**

## From version 9:

# No manual setup required anymore!

- Add the list of routes (`angular.json`)
- run!

```
npm run prerender
```

# Summary

- Initial content for SEO and Social Media
- Increase perceived performance
- Simple tooling through CLI
- Node.js is always required

- Pre-rendering at build time for static pages
- Runtime SSR for dynamic content

**Only necessary for public-facing pages!**

# Code examples on GitHub

https://github.com/

## angular-schule/demo-ssr-playground

# @fmalcher01

@angular_schule