Angular 4: Communication with the server using HTTP and WebSocket

Yakov Fain, Farata Systems

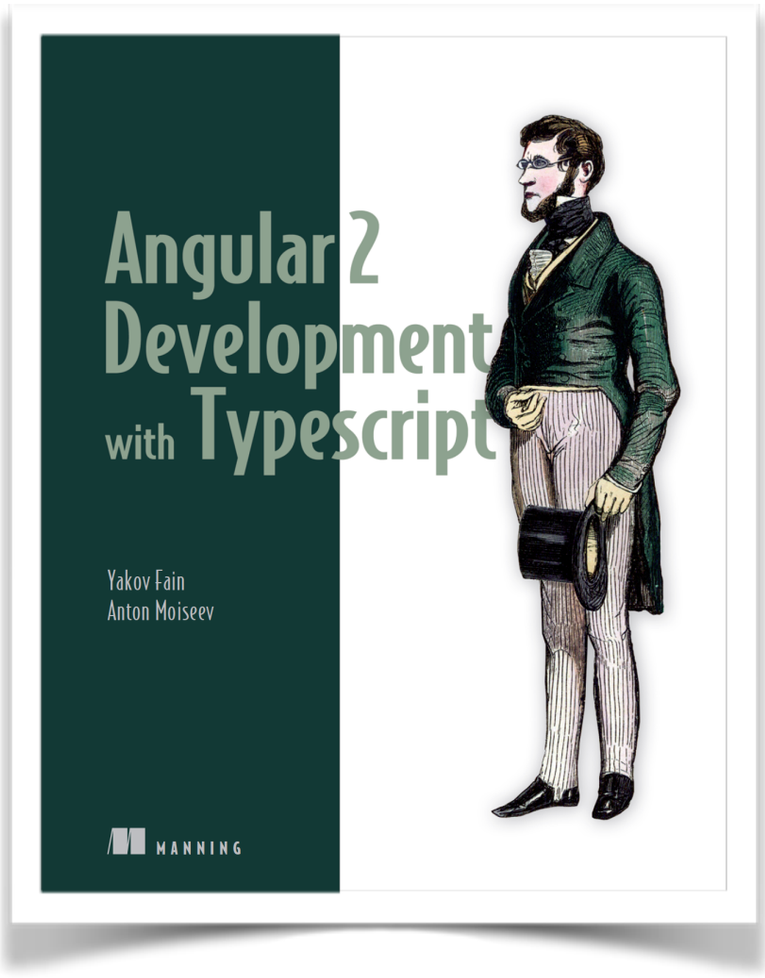
@yfain

About myself

- Work for Farata Systems
  Angular practice lead

- Java Champion

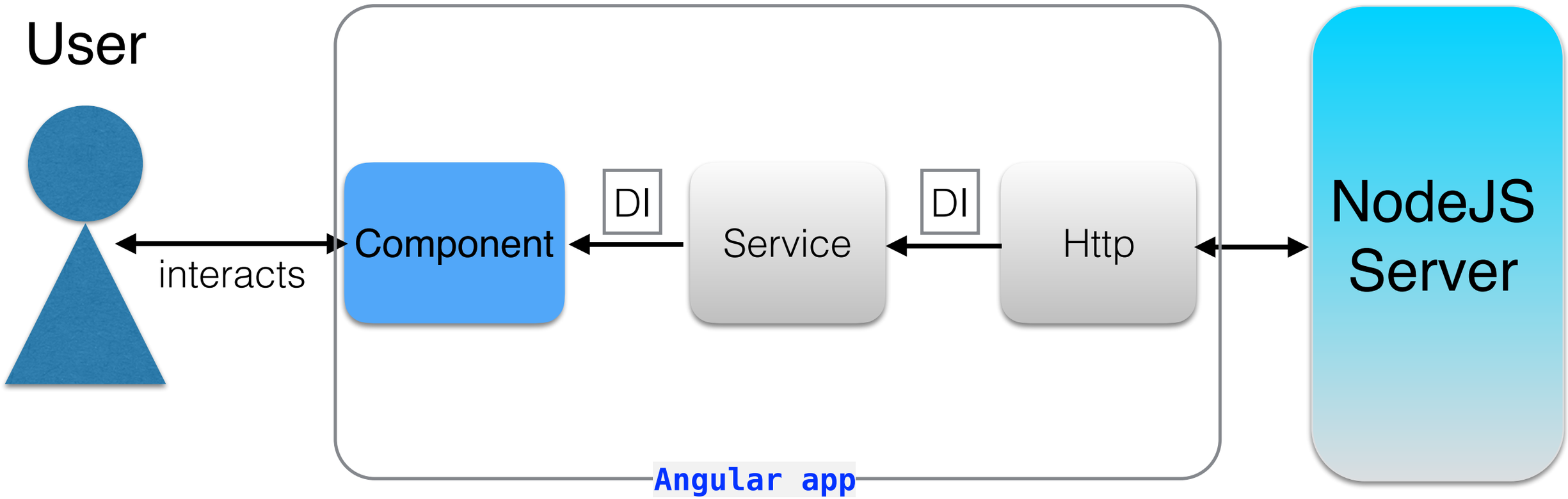
- Latest book:
  “Angular Development with TypeScript”



Agenda

- Creating a simple HTTP server with Node.js
- Working with the Angular Http object
- Deploying apps on Node server with Angular CLI
- Adding custom npm scripts for deployment
- Code review of the sample auction app

Let's develop an app



NodeJS can be replaced with Java, .Net, or other technology.

The server

Node.js

package.json (server)

To run local tsc

Live monitoring

```
{
  "name": "node-samples",
  "description": "Node and Express samples",
  "private": true,
  "scripts": {
    "tsc": "tsc"
  },
  "dependencies": {
    "express": "^4.14.0"
  },
  "devDependencies": {
    "@types/compression": "0.0.33",
    "@types/es6-shim": "0.31.32",
    "@types/express": “^4.0.34",
    "@types/node": "^6.0.57",
    "compression": "^1.6.2",
    "nodemon": "^1.11.0",
    "typescript": "^2.1.0"
  }
}
```

Creating an HTTP server in Node

hello-server.ts

```
import * as http from 'http';

const server = http.createServer((request, response)=> {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World!\n');
});

const port = 8000;

server.listen(port);
console.log('Listening on http://localhost:' + port);
```

tsconfig.json for the server

```
{
  "compilerOptions": {
    "module": "commonjs",
    "outDir": "build",
    "sourceMap": true,
    "target": "es5"
  },
  "exclude": [
    "node_modules"
  ]
}
```

As per commonjs format

```
import * as http from 'http';
```

becomes

```
var http = require('http');
```

Compile the server’s code with tsc and then start it.

Adding Express framework

- Express is a web framework for Node
- Express has simple API for handling client's requests

Install Express and type definitions:

```
npm install express --save

npm i @types/express --save
```

Http server with Express

my-express-server.ts

```
import * as express from "express";
const app = express();

app.get('/', (req, res) => res.send('Hello from Express'));

app.get('/products', (req, res) => res.send('Got a request for products'));

app.get('/reviews', (req, res) => res.send('Got a request for reviews'));

const server = app.listen(8000, "localhost", () => {

    const {address, port} = server.address();

    console.log('Listening on %s %s', address, port);
});
```

Demo

Starting Node and Express servers

```
node build/hello-server.js
```

```
node build/my-express-server.js
```

Live recompilation and reload of the server

- In a separate Terminal window, run tsc in the watch mode:
  tsc -w

- Node doesn't pick up changes after recompile - **nodemon** does.
  npm install nodemon --save-dev

- Add a command to npm scripts to run nodemon:

```
"scripts": {
  "dev": "nodemon build/my-express-server.js”
}
```

- npm run dev

Adding RESTful API to the server

rest-server.ts

```
import * as express from "express";

const app = express();

class Product {
    constructor(public id: number, public title: string, public price: number){}
}

const products = [
    new Product(0, "First Product", 24.99),
    new Product(1, "Second Product", 64.99),
    new Product(2, "Third Product", 74.99) ];

function getProducts(): Product[] {
    return products;
}
```

```
 app.get('/', (req, res) => {
     res.send('The URL for products is http://localhost:8000/api/products');
 });

app.get('/api/products', (req, res) => {
    res.json(getProducts());
});

function getProductById(productId: number): Product {
    return products.find(p => p.id === productId);
}

app.get('/api/products/:id', (req, res) => {
    res.json(getProductById(parseInt(req.params.id)));
});

const server = app.listen(8000, "localhost", () => {
    const {address, port} = server.address();
    console.log('Listening on %s %s', address, port);
});
```

Demo

Get products in JSON format on the client

1. "devRest": "nodemon build/rest-server.js”

2. npm run devRest

3. http://localhost:8000/api/products

Angular client

Working with Http object

From http.d.ts

```
import { Observable } from '@angular/core';
...
export declare class Http {

    constructor(_backend: ConnectionBackend, _defaultOptions: RequestOptions);

    request(url: string | Request, options?: RequestOptionsArgs): Observable<Response>;

    get(url: string, options?: RequestOptionsArgs): Observable<Response>;

    post(url: string, body: string, options?: RequestOptionsArgs): Observable<Response>;

    put(url: string, body: string, options?: RequestOptionsArgs): Observable<Response>;
...
}
```

Angular HTTP Support

Declare `HttpModule` in `@NgModule` and inject an `Http` object into the constructor of a component or a service

```
import {HttpModule} from '@angular/http';
…
@NgModule({
    imports:      [ BrowserModule,
                    HttpModule],
    declarations: [ AppComponent],
    bootstrap:    [ AppComponent ]
})
class AppModule {
```

```
import {Http} from '@angular/http';

class AppComponent {

…

 constructor(private http: Http) {
   …
 }
 ngOnInit(){
  this.http.get('products').subscribe(...);
 }
}
```

Subscribing to Http object's observable

```
@Component({
    selector: 'http-client',
    template: `<h1>All Products</h1>
  <ul>
    <li *ngFor="let product of products">
       {{product.title}}
    </li>
  </ul>
  `})
class AppComponent {

    products: Array<string> = [];

    theDataSource: Observable;

    constructor(private http: Http) {

        this.theDataSource = this.http.get('/api/products')
            .map(res => res.json());
    }

    ngOnInit(){
        // Get the data from the REST server
        this.theDataSource.subscribe(
            data => {
                this.products=data;
            },
            err =>
                console.log("Can't get products. Error code: %s, URL: %s ",  err.status, err.url),
            () => console.log('Product(s) are retrieved')
        );
    }
}
```

3

1

2

Dev mode: two servers

- In dev mode you can continue running the dev server for the client on port 4200 with **ng serve**
- But our REST server runs on port 8000
- If the client will do `http.get('http://localhost:8000/api/products')`, it'll get this:

```
XMLHttpRequest cannot load http://localhost:8000/api/products. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:4200' is therefore not allowed access.
```

Due to the **same-origin policy** we can configure a proxy on the client or add the header **Access-Control-Allow-Origin**: * on the server

Configuring proxy for Angular client

proxy-conf.json

```
{
  "/api": {
    "target": "http://localhost:8000",
    "secure": false
  }
}
```

The REST server runs here

goes to 4200

Angular client: `http.get('/api/products');`

from package.json

```
"scripts": {

  "start": "ng serve --proxy-config proxy-conf.json",

  …
}
```

Run the client with npm start.

Demo

Angular client with a proxy

1. ng serve --app rest -o

2. Browser console shows 404

3. ng serve --app rest --proxy-config proxy-conf.json -o

Building apps for prod deployment

JiT vs AoT compilation

- **Just-in-Time** compilation: your app includes Angular's compiler and is dynamically compiled in the browser.

- **Ahead-of-Time** compilation: Angular components and templates are precompiled into JS with the **ngc** compiler.

- The AoT-compiled apps don't include the Angular compiler

AOT doesn't always result in smaller bundles, but they load faster in the browser.

ng build for dev and prod

the output goes into the **dist** dir

- ng build

```
5.3K Mar 12 15:56 favicon.ico
609B Mar 12 15:56 index.html
5.6K Mar 12 15:56 inline.bundle.js
5.7K Mar 12 15:56 inline.bundle.js.map
9.2K Mar 12 15:56 main.bundle.js
6.6K Mar 12 15:56 main.bundle.js.map
163K Mar 12 15:56 polyfills.bundle.js
200K Mar 12 15:56 polyfills.bundle.js.map
9.8K Mar 12 15:56 styles.bundle.js
 13K Mar 12 15:56 styles.bundle.js.map
1.9M Mar 12 15:56 vendor.bundle.js
2.3M Mar 12 15:56 vendor.bundle.js.map
```

- ng build -prod

performs AoT by default

```
5.3K Mar 12 15:53 favicon.ico
700B Mar 12 15:53 index.html
1.4K Mar 12 15:53 inline.d3157cd62ad827642481.bundle.js
9.4K Mar 12 15:53 main.12703d3090c159b3c72a.bundle.js
 57K Mar 12 15:53 polyfills.bc16906638c50c8e5423.bundle.js
  0B Mar 12 15:53 styles.d41d8cd98f00b204e980.bundle.css
284K Mar 12 15:53 vendor.0d2ee48c6f11d0a326d7.bundle.js
```

Demo: ng build

1. Go to Terminal window in the directory unit7/client
2. ng build

3. Review the content of the **dist** directory and note the files sizes
4. ng build -prod

5. Review the content of the **dist** directory and note the files sizes
6. ng build -prod -aot=false

7. Review the content of the **dist** directory and note the files sizes

Adding custom npm scripts

```
"scripts": {
  "start": "ng serve --app rest --proxy-config proxy.conf.json",
  "build": "ng build -prod --app rest",
  "postbuild": "npm run deploy”,
  "predeploy": "rimraf ../server/build/public
                && mkdirp ../server/build/public”,
  "deploy": "copyfiles -f dist/** ../server/build/public"
}
```

post runs after

pre runs before

static resources

Include these devDependencies in package.json

Serving Angular app from the Node server

- Angular client's code is "static resources"
- Use Node's `path.join()`:

```
const app = express();

app.use('/',                express.static(path.join(__dirname, 'public')));
```

- Node will serve index.html unless you'll explicitly send another file:

```
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, './public/another.html'));
});
```

server: rest-server-angular.ts

```
let express = require("express");

let path = require("path");
let compression = require("compression");

const app = express();
app.use(compression()); // serve gzipped files

app.use('/', express.static(path.join(__dirname, 'public')));

class Product {
    constructor(
        public id: number,
        public title: string,
        public price: number){}
}

const products = [
    new Product(0, "First Product", 24.99),
    new Product(1, "Second Product", 64.99),
    new Product(2, "Third Product", 74.99)
];

function getProducts(): Product[] {
    return products;
}

app.get('/api/products', (req, res) => {
    res.json(getProducts());
});

function getProductById(productId: number): Product {
    return products.find(p => p.id === productId);
}

app.get('/api/products/:id', (req, res) => {
    res.json(getProductById(parseInt(req.params.id)));
});

const server = app.listen(8000, "localhost", () => {
    const {address, port} = server.address();
    console.log('Listening on %s %s', address, port);
});
```

Angular app deployed here

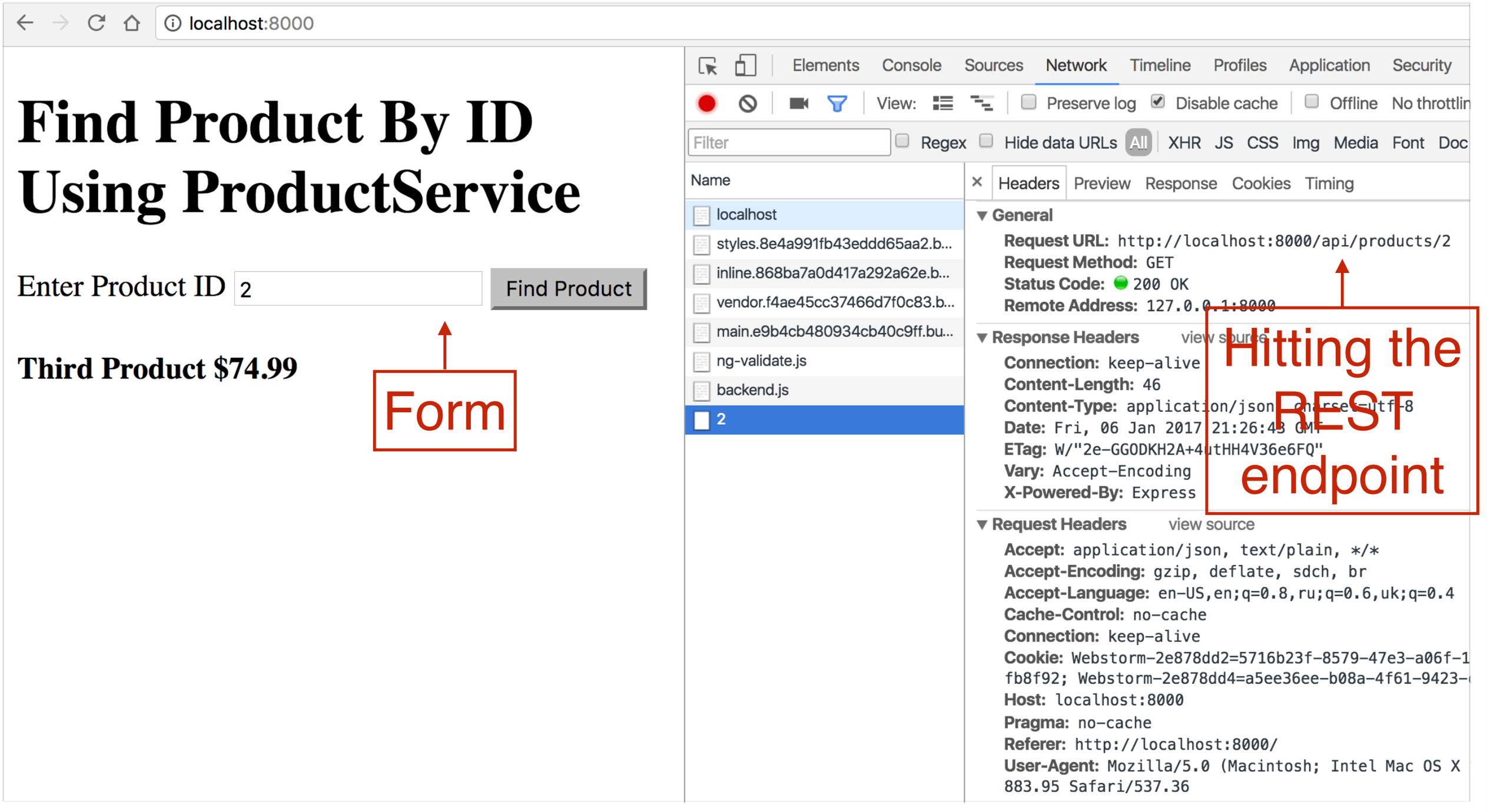
all products

product by id

Demo

1. npm run build
2. "devRestAngular": "nodemon build/rest-server-angular.js"
3. npm run devRestAngular

A form sends HTTP Get request



Making Http request from a form

```
// other imports go here
import {ProductService} from ‘./product.service';

@Component({
  selector: 'http-client',
  providers: [ ProductService ],
  template: `<h1>Find Product By ID Using ProductService</h1>
     <form #f="ngForm" (ngSubmit) = "getProductByID(f.value)" >
       <label for="productID">Enter Product ID</label>
       <input id="productID" type="number" name="productID" ngModel>
       <button type="submit">Find Product</button>
     </form>

     <h4>{{ productTitle}} {{productPrice}}</h4>
  `})
class AppComponent {

  productTitle: string;
  productPrice: string;

  constructor(private productService: ProductService) {}

  getProductByID(formValue){
      this.productService.getProductByID(formValue.productID)
        .subscribe(
            data => {
              this.productTitle = data.title;
              this.productPrice =`$` + data.price;
            },
            err => console.log("Can't get product details. Error code: %s, URL: %s ",  err.status, err.url),
            () =>    console.log( 'Done')
        );
  }
}
```

DI

```
import { Http} from '@angular/http';
import { Injectable} from "@angular/core";
import { Observable} from "rxjs/Observable";
import 'rxjs/add/operator/map';

@Injectable()
export class ProductService{

    constructor( private http: Http){}

   getProductByID(productID: string): Observable<any>{
       return this.http.get(`/api/products/${productID}`)
           .map(res => res.json());
    }
}
```

DI

Demo

1. Modify the build script to build rest-form app
2. npm run build
3. http://localhost:8000.
4. Searching for product returns 404 error. Let’s fix it.

Online Auction

Architecture and code review

The landing page of the Auction

Online Auction About Services Contact

Product title:

Title

Product price:

Price

Product category:

Search

HTTP request

800×300

320×150

First Product 24.99

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

4.3 stars

320×150

Second Product 64.99

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

3.5 stars

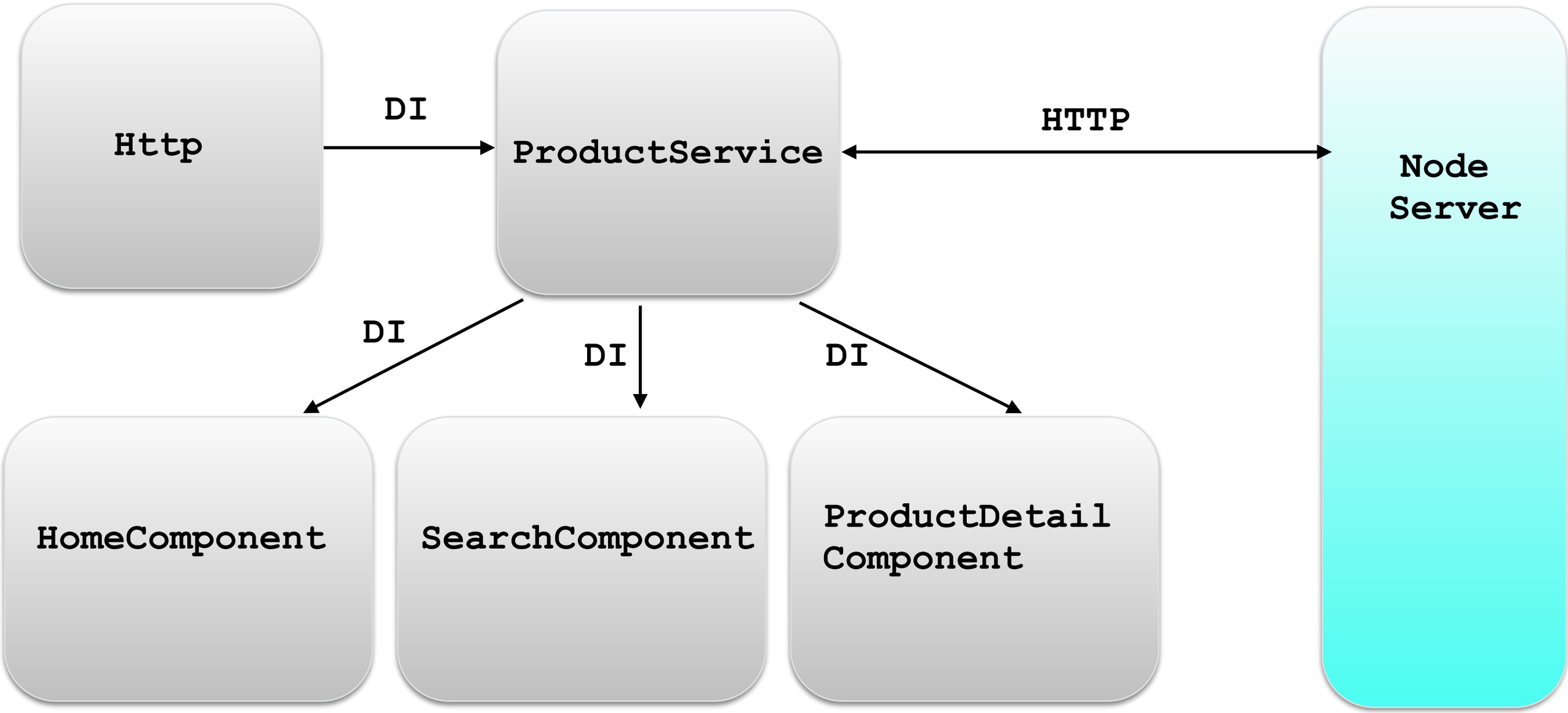
320×150

Third Product 74.99

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

4.2 stars

Product search using HTTP



The product detail page of the Auction

Online Auction About Services Contact

Product title:

Title

Product price:

Price

Product category:

Search

820×320

First Product **USD24.99**

This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

☆☆☆☆☆ 6 reviews

Watch **Current bid: USD24.99**

Bids are pushed via WebSocket

Leave a Review

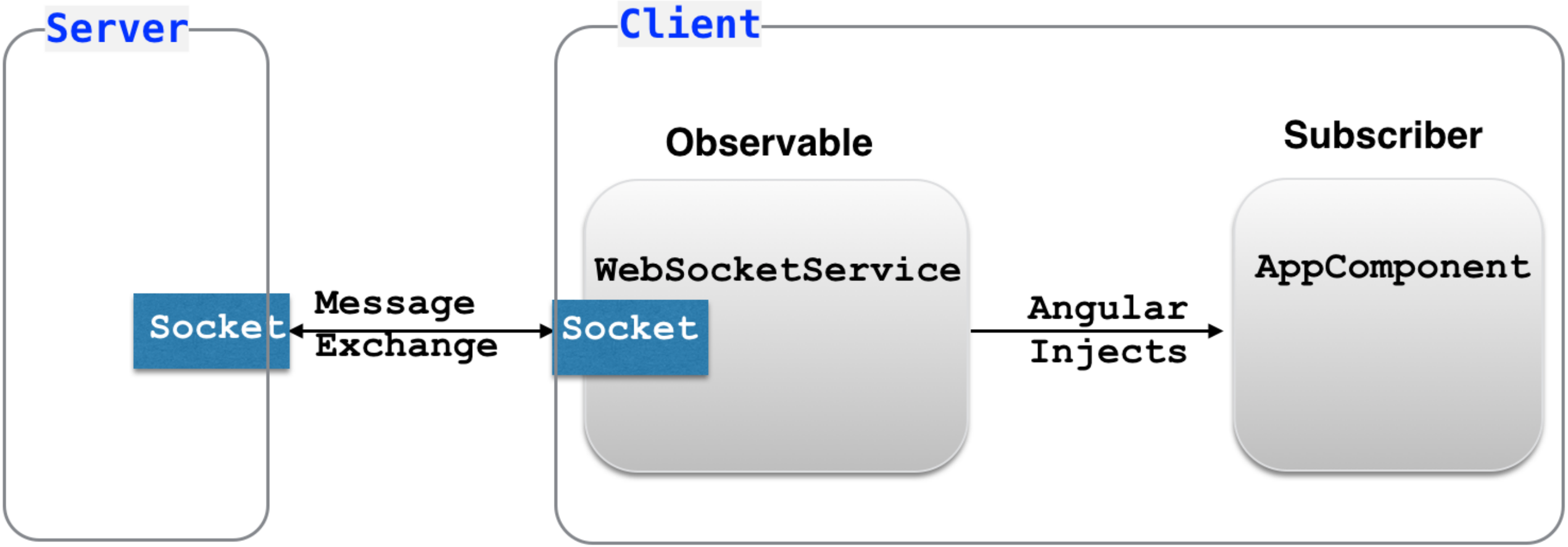
★★★★★ 5 stars

User 1 5/19/2014

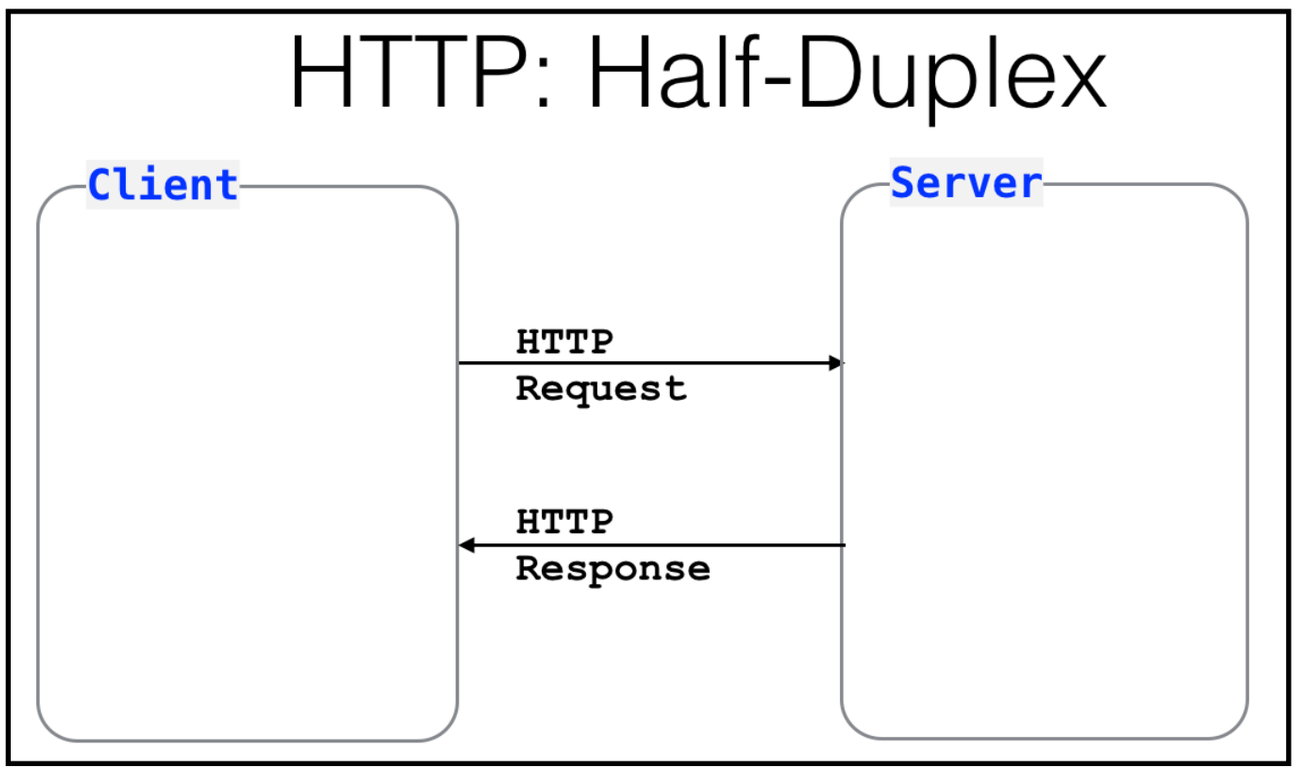
Aenean vestibulum velit id placerat posuere. Praesent placerat mi ut massa tempor, sed rutrum metus rutrum. Fusce lacinia blandit ligula eu cursus. Proin in lobortis mi. Praesent pellentesque auctor dictum. Nunc volutpat id nibh quis malesuada. Curabitur tincidunt luctus leo, quis condimentum mi aliquet eu. Vivamus eros metus, convallis eget rutrum nec, ultrices quis mauris. Praesent non lectus nec dui venenatis pretium.

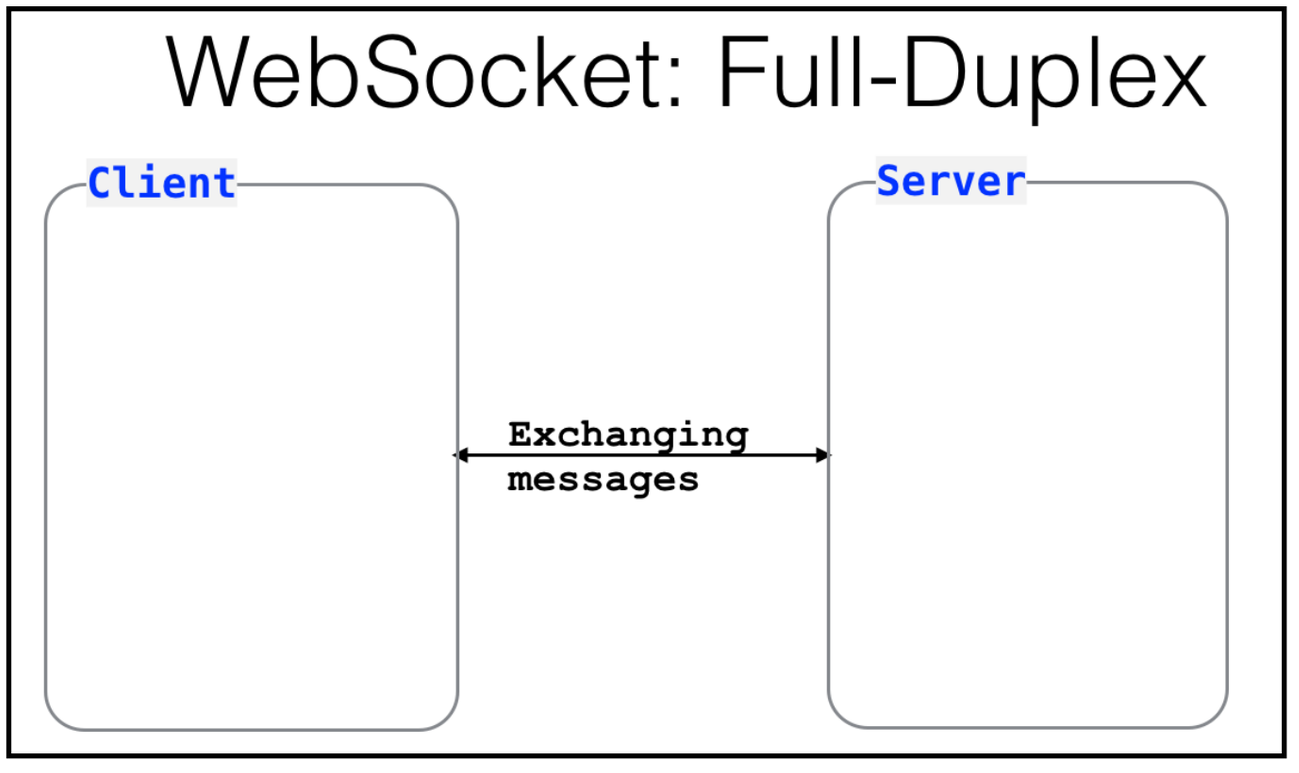
★★★☆☆ 3 stars

Pushing bid notifications via WebSockets



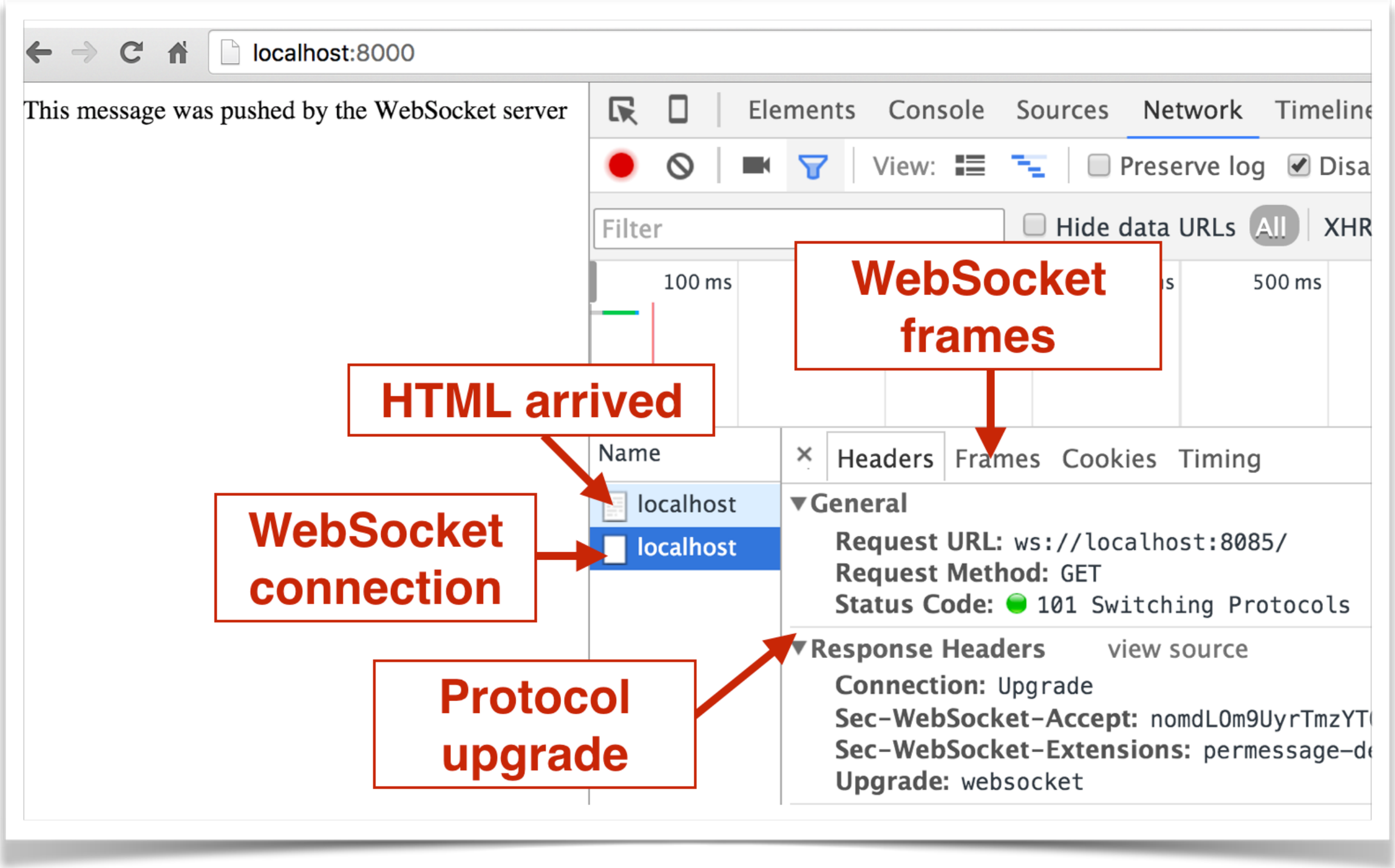
HTTP vs WebSocket





WebSocket docs and demos: http://www.websocket.org

Switching protocols from HTTP to WebSocket



localhost:8000

Angular subscriber to WebSocket service

This message was pushed by the WebSocket server

Send msg to Server

Elements Console Sources Network Timeline Profiles Res

View: Preserve log Disable cache No t

Filter Hide data URLs All XHR JS CSS Img

100 ms 200 ms 300 ms 400 ms 5

Click

Name × Headers Frames Cookies Timing

localhost

Data	Length	Time
This message was pushed by the WebSocket server	47	14:38:33.
Hello from client	17	14:42:36.

Demo

- Server:
- npm run tsc
- npm run startServer
- Client:
- npm run build
- http://localhost:8000

What have we learned

- HTTP requests are treated as observable data streams
- How to create a simple Web server with NodeJS
- You can deploy an Angular app on any Web server
- Deployment comes down to preparing code bundles and copying them to a document root directory of the Web server
- How to build a prod version of the Angular app with and without AoT
- How to automate the build process with npm scripts
- When to use WebSockets
- Reviewed the code and deployed a sample app on your computer

Thank you!

- Code samples:
  http://bit.ly/2oBDazf

- Training inquiries:
  training@faratasystems.com

- My blog:
  yakovfain.com

- Twitter: @yfain