Communication with the server using HTTP and WebSockets

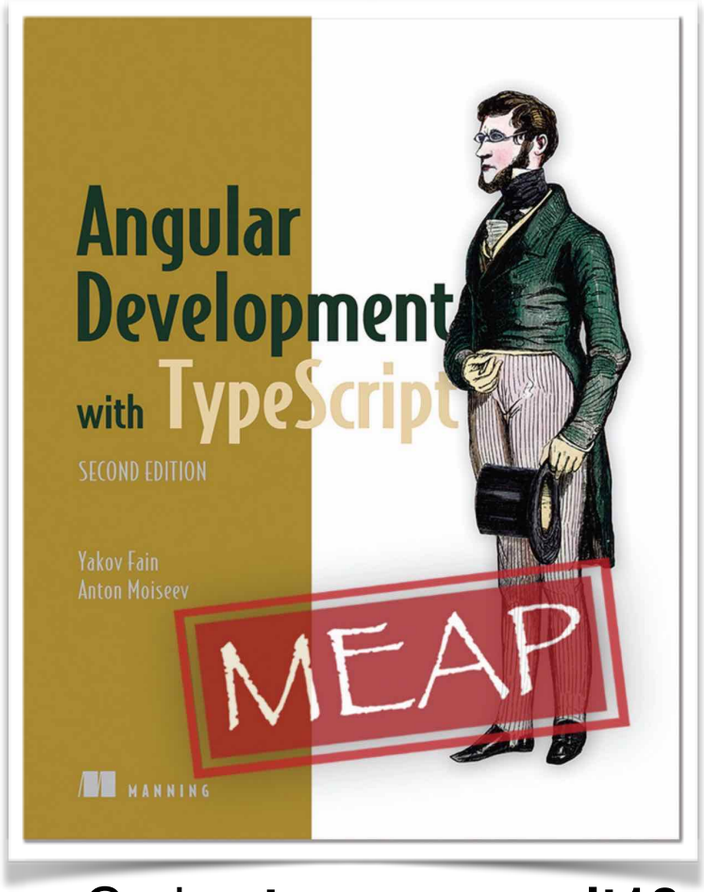
Yakov Fain, Farata Systems

@yfain

@yfain

About myself

- Work for Farata Systems
- Angular consulting and training
- Java Champion
- Co-authored two editions of the book
  "Angular Development with TypeScript"
- Working on the book
  "Get Programming with TypeScript"



Code **ctwangsummit18**
40% off all books
at manning.com

@yfain

Agenda

- Working with the Angular HttpClient object

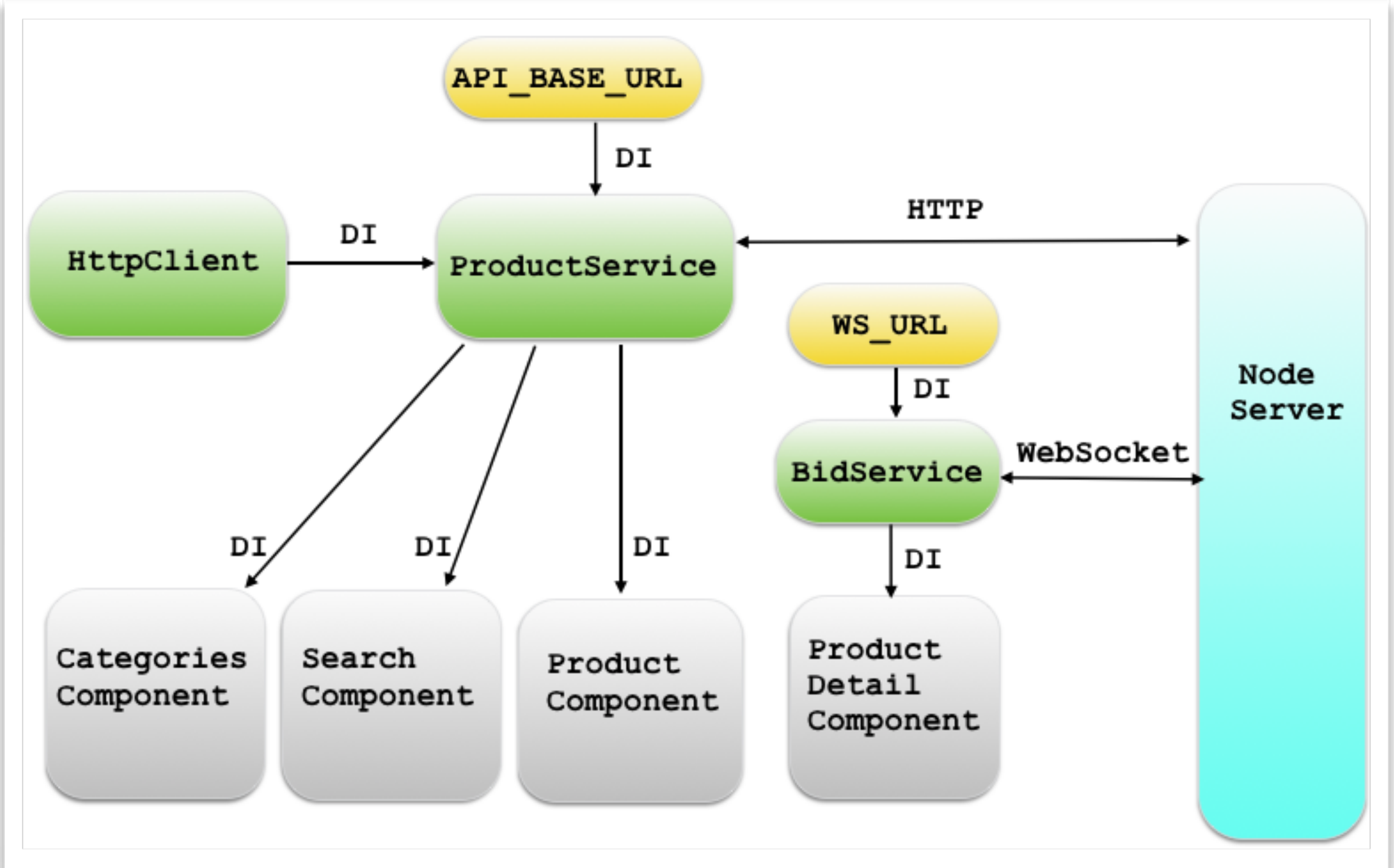
- HTTP Interceptors and progress events

- Deploying apps on the server with npm scripts and Angular CLI

- Pushing the data from the server via WebSockets

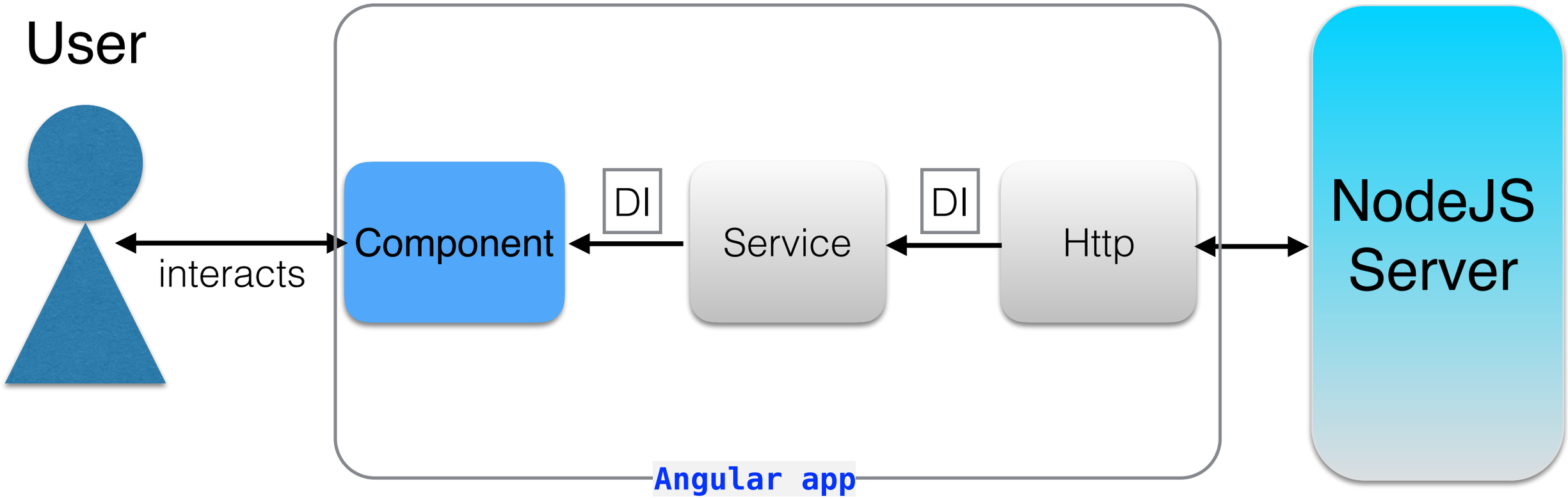
- Code review of the sample auction app that uses HTTP and WebSockets

The final demo app



@yfain

but let's start with this



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

@yfain

The server

Node.js

@yfain

package.json (server)

```
{
  "name": "node-server",
  "description": "Node and Express samples",
  "private": true,

  "dependencies": {
    "express": "^4.16.2"
  },
  "devDependencies": {
    "@types/express": "^4.0.39",
    "@types/node": "^6.0.57",
    "typescript": "2.6.2"
  }
}
```

@yfain

Node-Express server

```
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);
});
```

rest-server.ts

@yfain

Start the server

1. Compile
   tsc

2. Start server
   node build/rest-server

3. Get JSON with products
   http://localhost:8000/api/products

Angular client

@yfain

Angular HttpClient

- Offers API for all standard HTTP methods,
  e.g. `get()`, `post()`, `put()`, `delete()`, `request()`
- All these methods return `Observable`
- Interceptors - the code that's invoked on each HTTP request or response
- Supports progress events for request uploads and response downloads

Using HttpClient

Import `HttpClientModule` in `@NgModule` and inject an `HttpClient` object into the constructor of a service (or component)

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

```
import {HttpClient} from '@angular/common/http';
...
class ProductService {

 constructor(private httpClient: HttpClient){}

 getProducts(){
  this.httpClient.get<Product[]>('/api/products')
      .subscribe(...);
 }
}
```

Server endpoint

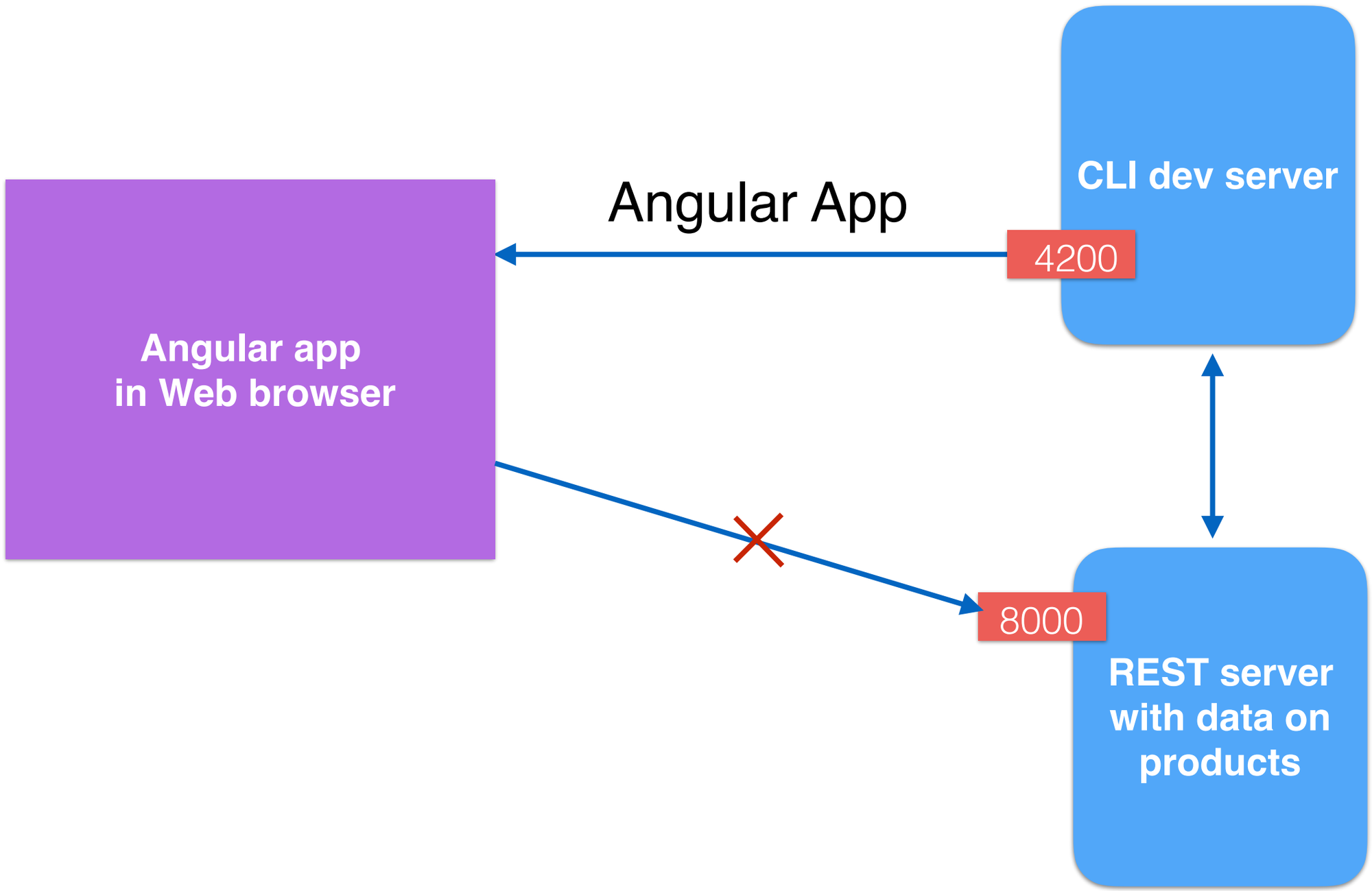
@yfain

Explicitly subscribing to HttpClient

```
@Component({
  selector: 'app-root',
  template: `<h1>All Products</h1>
  <ul>
    <li *ngFor="let product of products">        ←3
       {{product.title}} {{product.price}}
    </li>
  </ul>
  {{error}}
  `})
export class AppComponent implements OnInit, OnDestroy{

  products: any[] = [];
  theDataSource$: Observable<any[]>;
  productSubscription: Subscription;
  error:string;

  constructor(private httpClient: HttpClient) {
    this.theDataSource$ = this.httpClient.get<any[]>('/api/products');←1
  }

  ngOnInit(){
    this.productSubscription = this.theDataSource$
      .subscribe(
      data => {
          this.products=data;          ←2
      },
      err =>
        this.error = `Can't get products. Got ${err.status} from ${err.url}`
    );
  }

  ngOnDestroy(){
    this.productSubscription.unsubscribe();        ←4
  }
}
```

@yfain

Configuring a proxy for dev mode



@yfain

Same origin error

- 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 Angular 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 and gets redirected

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

```
ng serve --proxy-config proxy-conf.json
```

@yfain

Demo

Angular client with a proxy

1. ng serve --app restclient -o
2. http://localhost:4200 returns 404
3. ng serve --app restclient --proxy-config proxy-conf.json -o
4. http://localhost:4200 serves data
   from the server running on port 8000

Subscribing to HttpClient with async pipe

```
@Component({
  selector: 'app-root',
  template: `<h1>All Products</h1>
  <ul>
    <li *ngFor="let product of products$ | async">
      {{product.title }} {{product.price}}
    </li>
  </ul>
  {{error}}
  `})
export class AppComponentAsync{

  products$: Observable<any[]>;
  error:string;
  constructor(private httpClient: HttpClient) {
    this.products$ = this.httpClient.get<any[]>('/api/products')
      .catch( err => {
          this.error = `Can't get products. Got ${err.status} from ${err.url}`;
          return Observable.of([]);    // empty observable array
      });
  }
}
```

2

Best practice

1

@yfain

Building apps for prod deployment

@yfain

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.

@yfain

ng build for dev

the output goes into the **dist** dir

- ng build

```
136B Jan 29 12:46 data
5.3K Jan 29 12:46 favicon.ico
609B Jan 29 12:46 index.html
5.7K Jan 29 12:46 inline.bundle.js
5.8K Jan 29 12:46 inline.bundle.js.map
9.0K Jan 29 12:46 main.bundle.js
6.2K Jan 29 12:46 main.bundle.js.map
212K Jan 29 12:46 polyfills.bundle.js
259K Jan 29 12:46 polyfills.bundle.js.map
 11K Jan 29 12:46 styles.bundle.js
 15K Jan 29 12:46 styles.bundle.js.map
2.4M Jan 29 12:46 vendor.bundle.js
2.9M Jan 29 12:46 vendor.bundle.js.map
```

ng build for dev and prod

the output goes into the **dist** dir

- ng build

```
136B Jan 29 12:46 data
5.3K Jan 29 12:46 favicon.ico
609B Jan 29 12:46 index.html
5.7K Jan 29 12:46 inline.bundle.js
5.8K Jan 29 12:46 inline.bundle.js.map
9.0K Jan 29 12:46 main.bundle.js
6.2K Jan 29 12:46 main.bundle.js.map
212K Jan 29 12:46 polyfills.bundle.js
259K Jan 29 12:46 polyfills.bundle.js.map
 11K Jan 29 12:46 styles.bundle.js
 15K Jan 29 12:46 styles.bundle.js.map
2.4M Jan 29 12:46 vendor.bundle.js
2.9M Jan 29 12:46 vendor.bundle.js.map
```

- ng build --prod

performs AoT by default

```
3.2K Jan 29 12:51 3rdpartylicenses.txt
136B Jan 29 12:51 data
5.3K Jan 29 12:51 favicon.ico
589B Jan 29 12:51 index.html
1.4K Jan 29 12:51 inline.d483d84aa7d8440978f5.bundle.js
175K Jan 29 12:51 main.8522776bac4edaecdaad.bundle.js
 64K Jan 29 12:51 polyfills.47853ebf6acf9efe05b4.bundle.js
 79B Jan 29 12:51 styles.9c0ad738f18adc3d19ed.bundle.css
```

Demo: ng build

1. Go to Terminal window in the directory client. After running each of the following commands check the content of the **dist** dir
2. ng build ← not optimized
3. ng build --prod ← AoT, optimized

@yfain

Base Href

- If you deploy at myserver.com use default: `<base href="/">`
- If you deploy at myserver.com/shipping use `<base href="/shipping">`
- ng build --base-href /shipping/

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>NgAuction</title>
  <base href="/shipping/">

  <meta name="viewport" conte
  <link rel="icon" type="imag
</head>
<body>
  <nga-root></nga-root>
<script type="text/javascript
</html>
```

@yfain

Adding custom npm scripts

```
"scripts": {
  "start": "ng serve --proxy-config proxy.conf.json",
  "build": "ng build -prod”,
  "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

@yfain

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 serves index.html

@yfain

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

@yfain

Demo

1. Build and deploy the Angular app:
   npm run build

2. Start the server
   node build/rest-server-angular

3. In the browser:
   http://localhost:8000

HTTP interceptors

@yfain

Why intercepting?

- For pre- and post-processing of all HTTP requests and responses, e.g. adding certain headers
- For implementing cross-cutting concerns, e.g. logging, global error handling, authentication

The original HttpRequest is immutable, but you can clone it and update the clone

Intercepting HTTP requests

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

  constructor (private auth: MyAuthService) {}

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

    const modifiedRequest =
        req.clone({ req.headers.set('Authorization', this.auth.getToken()) });

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

```
@NgModule({
  ...
  providers: [[ { provide: HTTP_INTERCEPTORS, useClass: MyInterceptor, multi: true } ]],
})
```

@yfain

Intercepting HTTP responses

- `next.handle()` returns an observable of `HttpEvent`'s
- Use RxJS operators to handle the response
- No need to subscribe as the original `HttpClient` already has a subscriber

```
@Injectable()
export class MyInterceptor implements HttpInterceptor {
   ...
    return next.handle(modifiedRequest).pipe(
          tap(event = > {
            if (event instanceOf HttpResponse) {
              // clone and change event here if need be
                console.log (event);
            }
            return event;
           });
}
```

@yfain

Demo: An interceptor for logging errors

1. Start the server

```
node build/rest-server-angular-post-errors
```

2. Start the client

```
ng serve --app interceptor -proxy-config proxy-conf.json
```

3. Add some products. The server will randomly generate errors, which will be handled in the interceptor

Progress Events

- If an uploading or downloading takes time, report the progress to the user
- Use `HttpRequest` if you want to handle progress events

An HTTP GET with progress events

```
const req = new HttpRequest('GET',
                            './data/48MB_DATA.json',
                            {reportProgress: true}
                            );

  httpClient.request(req)
  .subscribe(data => {

   if (data.type === HttpEventType.DownloadProgress) {

     console.log(`Read ${this.percentDone}% of ${data.total} bytes`);

   } else {

    this.mydata = data

  }
});
```

@yfain

Demo: progress events

```
ng serve --app progressevents
```

Using WebSocket protocol

@yfain

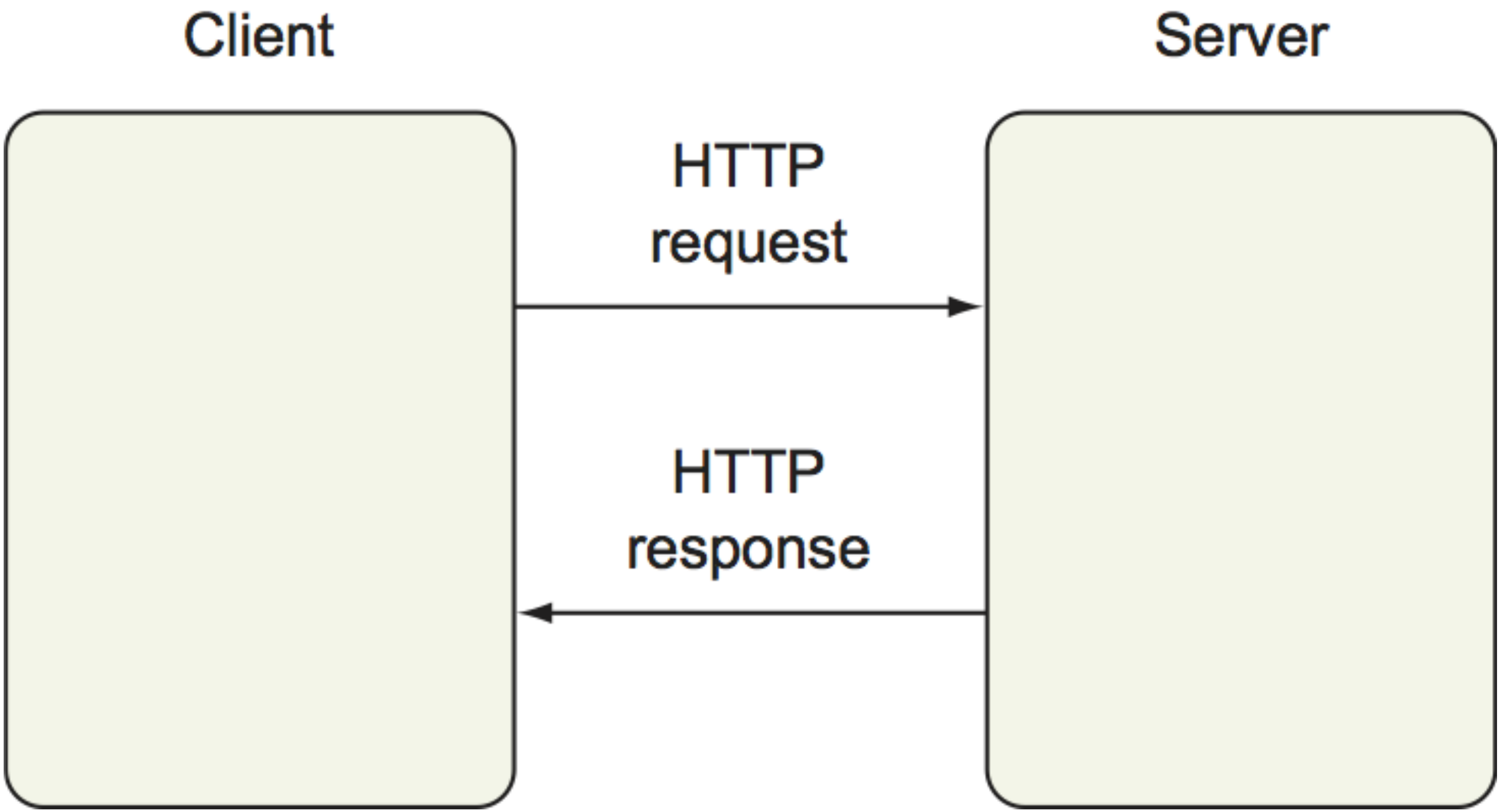
What’s WebSocket protocol

- Low-overhead binary protocol
- Not a request/response based
- Supported by all modern browsers and servers
- Allows bidirectional message-oriented streaming of text and binary data between browsers and web servers

Where the server push is needed

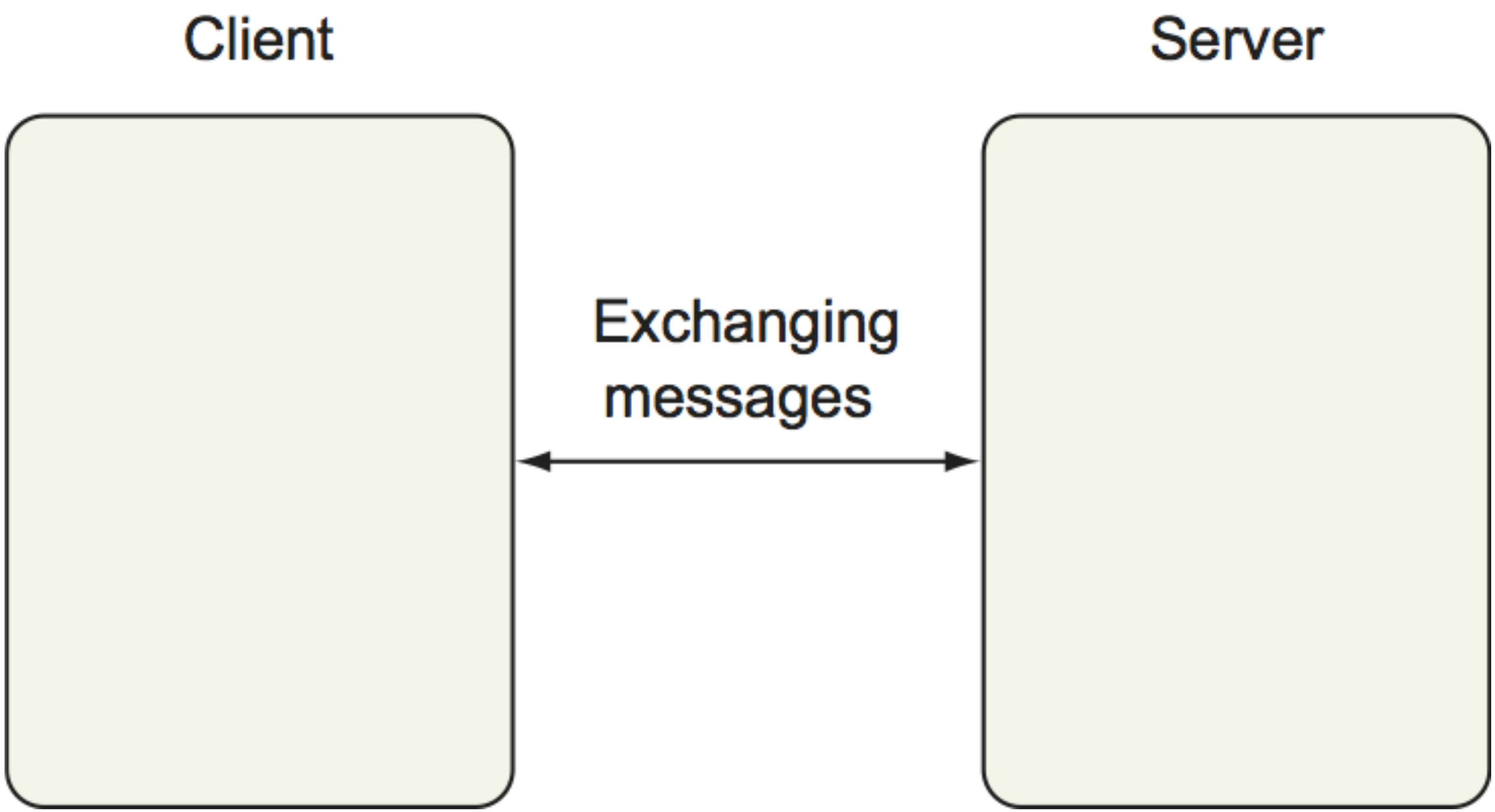
- Live trading/auctions/sports notifications
- Controlling medical equipment over the web
- Chat applications
- Multiplayer online games
- Real-time updates in social streams
- Live charts

HTTP: half duplex



@yfain

WebSocket: full duplex



@yfain

WebSocket in a plain JavaScript client

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
<span id="messageGoesHere"></span>

<script type="text/javascript">

    var ws = new WebSocket("ws://localhost:8085");

    ws.onmessage = function(event) {
        var mySpan = document.getElementById("messageGoesHere");
        mySpan.innerHTML=event.data;
    };

    ws.onerror = function(event){
        ws.close();
        console.log(`Error ${event}`);
    }
</script>
</body>
</html>
```

simple-websocket-client.html

@yfain

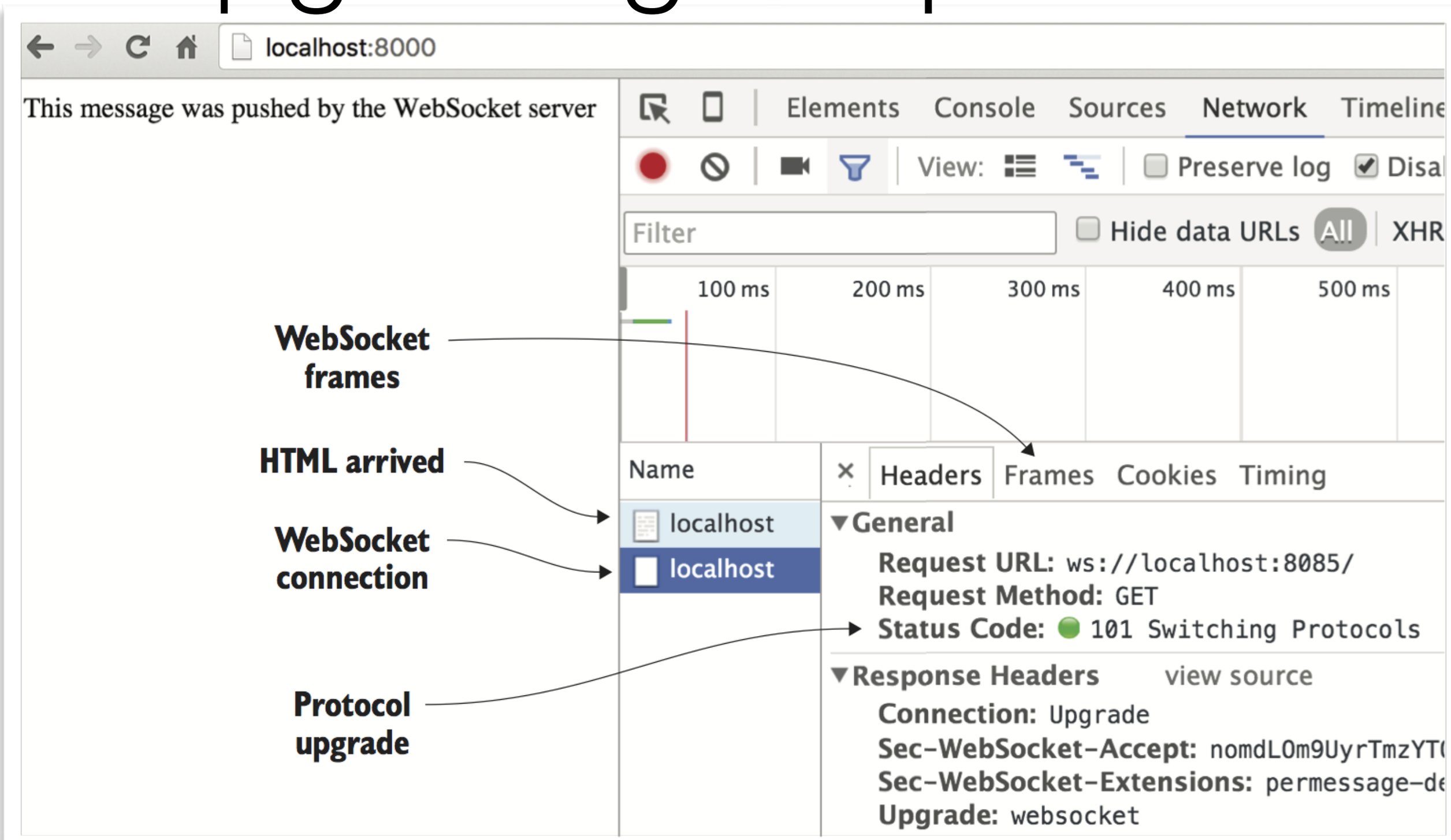
Simple Node.js Websocket server

```
import * as express from "express";
import * as path from "path";
import {Server} from "ws";

const app = express();

// HTTP Server
app.get('/', (req, res) =>
    res.sendFile(path.join(__dirname, '../simple-websocket-client.html')));

const httpServer = app.listen(8000, "localhost", () => {
    const {port} = httpServer.address();
    console.log(`HTTP server is listening on ${port}`);
});

// WebSocket Server
const wsServer = new Server({port:8085});
console.log('WebSocket server is listening on port 8085');

wsServer.on('connection',
            wsClient => {
                wsClient.send('This message was pushed by the WebSocket server');

                wsClient.onerror = (error) =>
                    console.log(`The server received: ${error['code']}`);
            }
);
```

simple-websocket-server.ts

@yfain

Upgrading the protocol



@yfain

Demo: Pushing to a JavaScript client

- Go to server dir
- tsc
- node build/simple-websocket-server
- Open the browser at http://localhost:8000

Two ways of using WebSockets in Angular

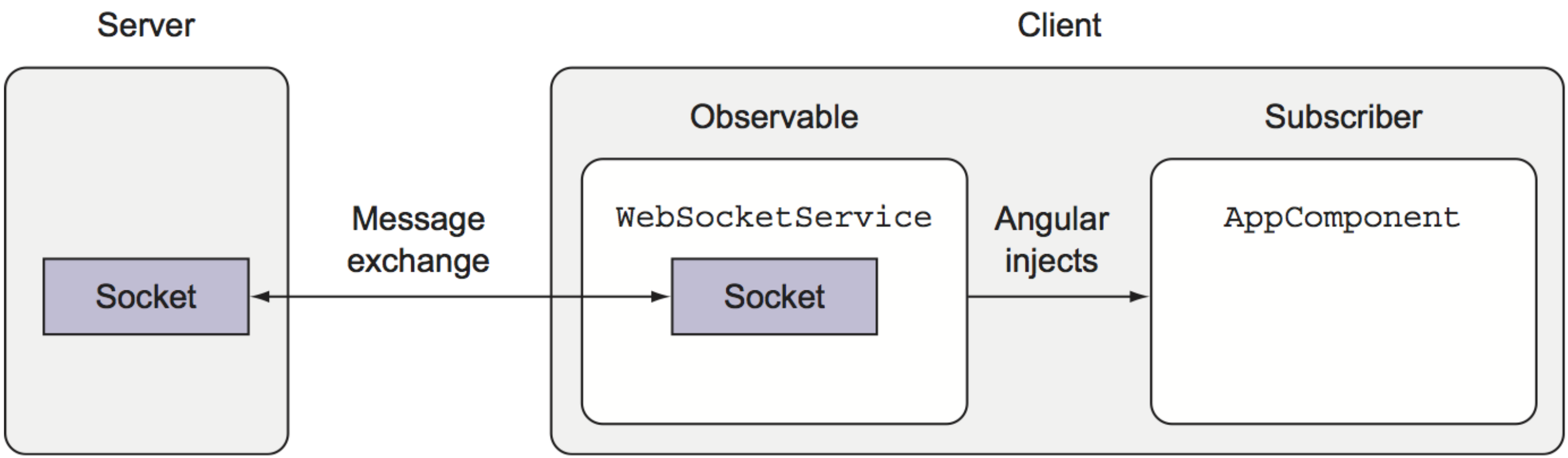
1. Manually create an instance of the `WebSocket` object
2. Use RxJS `WebSocketSubject`

Wrapping a WebSocket object in a service

@yfain

WebSocket in Angular service

Think of WebSocket as a data producer for an Observable stream



@yfain

Wrapping WebSocket in Angular service

```
export class WebSocketService{

  ws: WebSocket;
  socketIsOpen = 1; // WebSocket's open

  createObservableSocket(url:string): Observable<any>{
    this.ws = new WebSocket(url);

    return new Observable(
      observer => {
        this.ws.onmessage = (event) => observer.next(event.data);

        this.ws.onerror = (event) => observer.error(event);

        this.ws.onclose = (event) => observer.complete();


        // a callback invoked on unsubscribe()
      return () => this.ws.close(1000, "The user disconnected");
      }
    );
  }

  sendMessage(message: string): string{
    if (this.ws.readyState === this.socketIsOpen) {
      this.ws.send(message);
      return `Sent to server ${message}`;
    } else {
      return 'Message was not sent - the socket is closed';
    }
  }
}
```

Emit data from server

Send data to server

websocket-service.ts

Using a WebSocket service in a component

```
export class AppComponent implements OnDestroy {

  messageFromServer: string;
  wsSubscription: Subscription;
  status;

  constructor(private wsService: WebSocketService) {

    this.wsSubscription = this.wsService.createObservableSocket("ws://localhost:8085")
      .subscribe(
        data => this.messageFromServer = data,  // Receiving
        err => console.log( 'err'),
        () =>  console.log( 'The observable stream is complete')
      );
  }

  sendMessageToServer(){
    this.status = this.wsService.sendMessage("Hello from client”);  // Sending
  }

  closeSocket(){
    this.wsSubscription.unsubscribe();  // Closing
    this.status = 'The socket is closed';
  }

  ngOnDestroy() {
    this.closeSocket();
  }
}
```

websocket-service.ts

Node server

```
import {Server} from "ws";

var wsServer = new Server({port:8085});

console.log('WebSocket server is listening on port 8085');

wsServer.on('connection',
    websocket => {

        websocket.send('Hello from the two-way WebSocket server');

        websocket.onmessage = (message) =>
            console.log('The server received:', message['data']);

        websocket.onerror = (error) =>
            console.log(`The server received: ${error['code']}`);

        websocket.onclose = (why) =>
            console.log(`The server received: ${why.code} ${why.reason}`);
    }
);
```

two-way-websocket-server.ts

Demo: Angular/Node send/receive

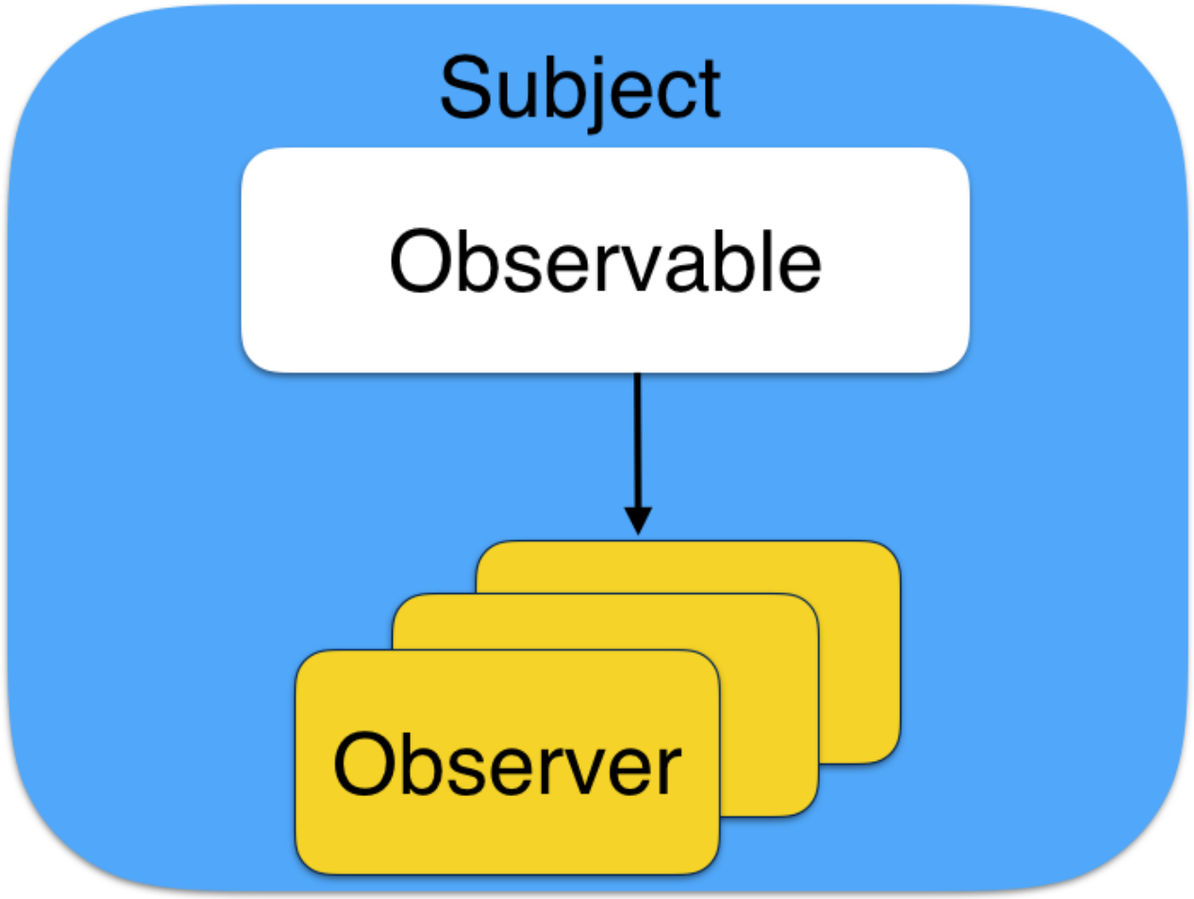
- Server:
  node build/two-way-websocket-server
- Client:
  ng serve --app wsservice

Using RxJS WebSocketSubject

@yfain

RxJS Subject

Rx.Subject is both an observable and observer




```
const mySubject = new Subject();
...
subscription1 = mySubject.subscribe(...);
subscription2 = mySubject.subscribe(...);
...
mySubject.next(123); // each subscriber gets 123
```

@yfain

RxJS WebSocketSubject

- A ready-to-use wrapper around the browser's `WebSocket`
- Accepts either a string with the WebSocket endpoint or a `WebSocketSubjectConfig`
- On subscribe, it uses either an existing connection or creates a new one
- On unsubscribe, it closes connection if there are no other subscribers

When a server pushes data

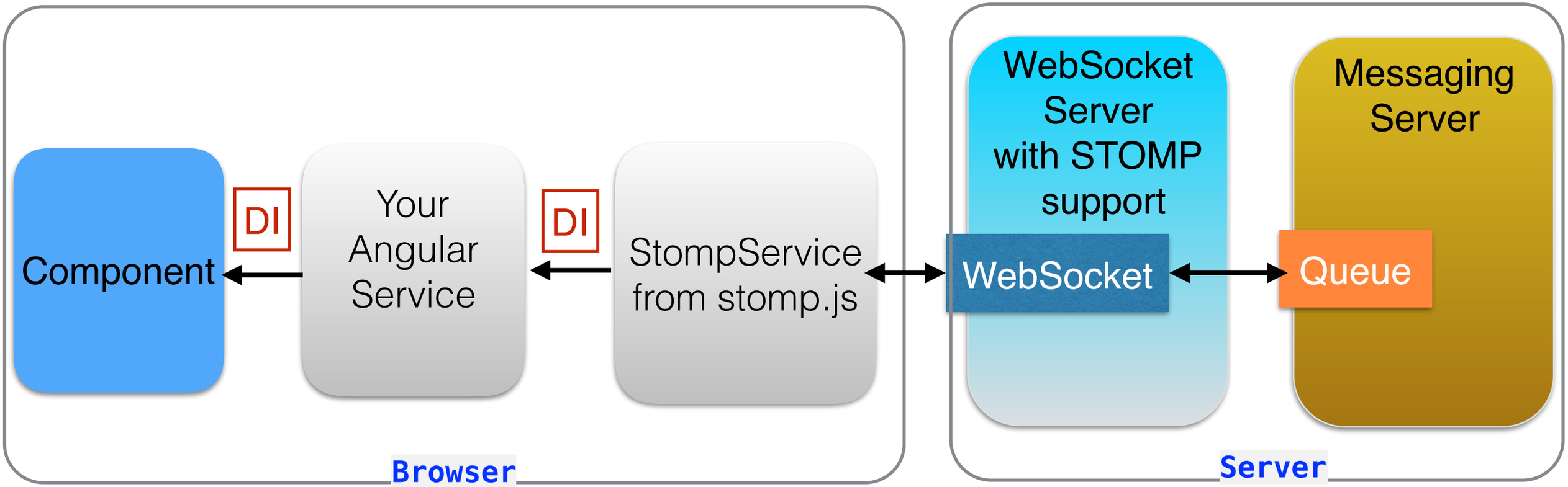
- `WebSocketSubject` emits the data into observable stream
- In case of a socket error, `WebSocketSubject` emits an error
- If there are no subscribers, `WebSocketSubject` buffers the value

BidService with WebSocketSubject

```
import { WebSocketSubject } from 'rxjs/observable/dom/WebSocketSubject';
...

export interface BidMessage {
  productId: number;
  price: number;
}

@Injectable()
export class BidService {
  private _wsSubject: WebSocketSubject<any>;

  private get wsSubject(): WebSocketSubject<any> {
    const closed = !this._wsSubject || this._wsSubject.closed;
    if (closed) {
      this._wsSubject = WebSocketSubject.create(this.wsUrl);
    }
    return this._wsSubject;
  }

  get priceUpdates$(): Observable<BidMessage> {
    return this.wsSubject.asObservable();
  }

  constructor(@Inject(WS_URL) private readonly wsUrl: string) {}

  placeBid(productId: number, price: number): void {
    this.wsSubject.next(JSON.stringify({ productId, price }));
  }
}
```

@yfain

Integrating with server-side messaging systems using STOMP protocol



STOMP docs: http://stomp.github.io/stomp-specification-1.2.html

stomp.js: https://github.com/stomp-js/ng2-stompjs

@yfain

Online Auction

@yfain

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

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

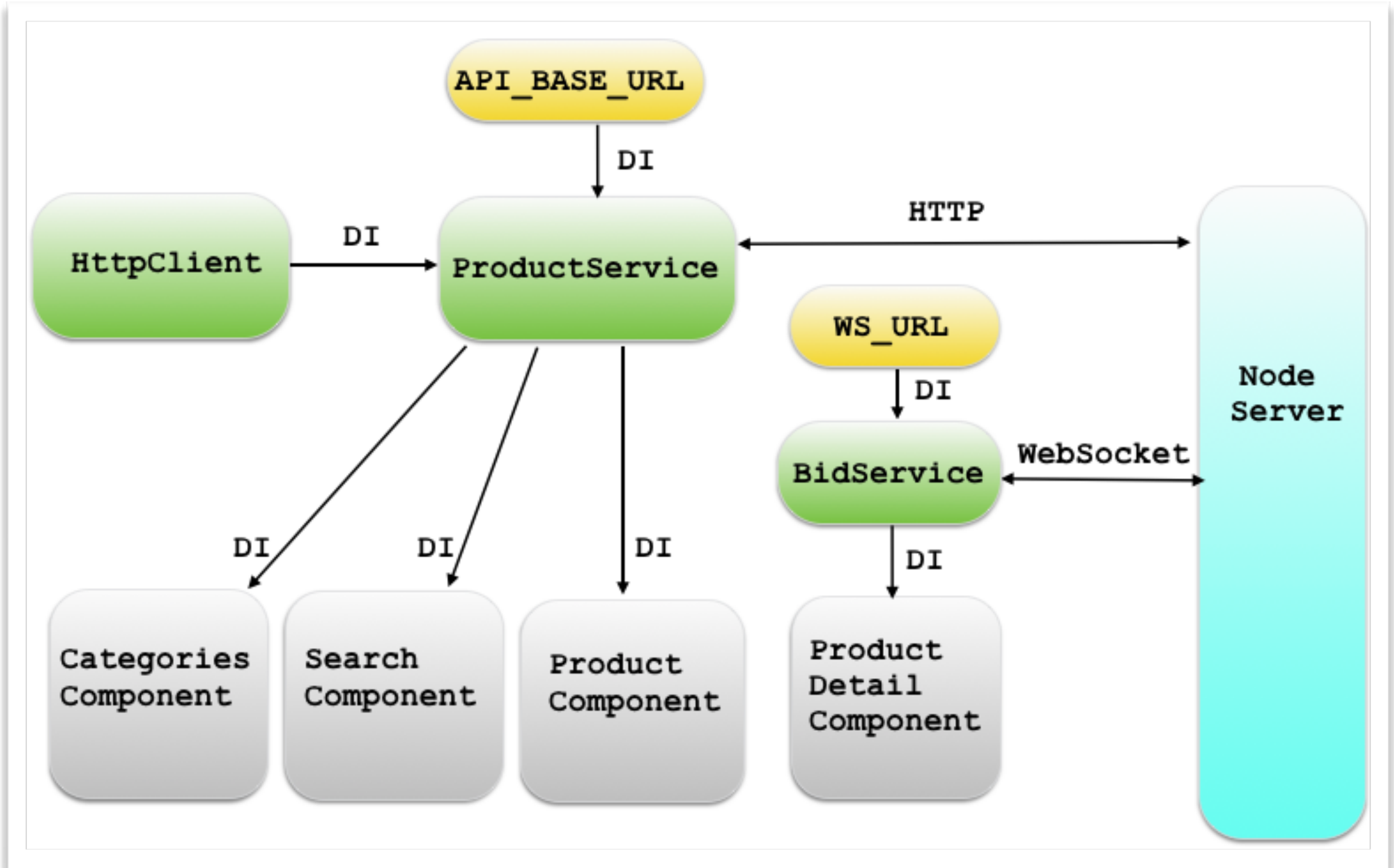
★★★★★ 5 stars

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

Building blocks of ngAuction



@yfain

Demo of ngAuction

- Server (go to ng-auction/server dir):
- tsc
- node build/main
- Client (go to ng-auction/client)
- ng serve -o

@yfain

What have we learned

- HTTP requests are treated as observable data streams
- How to create a simple Web server with NodeJS
- How to build and deploy an Angular app on the server
- How to automate your build process with npm scripts
- What are interceptors and progress events
- How to work with WebSockets

Thank you!

- HTTP code samples:
  https://bit.ly/2rvHuTP
- WebSocket code samples:
  https://bit.ly/2vhP35J
- Email: yfain@faratasystems.com
- Blog: yakovfain.com

@yfain