Angular 2. HTTP clients. WebSocket

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Agenda

1. Creating Angular 2 Hello World Application
2. Web components and data binding
3. Differences between Angular 1 and Angular 2
4. Advantages of Angular 2
5. Data architecture in Angular 2 – overview, main types of components: Module, Component, Template, Metadata, Data Binding, Service, Directive, Dependency Injection.
6. Component controllers, views & templates
7. Using external template and style files.
8. Building custom directives, pipes, and validators
9. Ng2 by example – Tour of Heroes official Angular 2 tutorial

Creating Angular 2 Hello World Application

- 5 min Quickstart: https://angular.io/docs/ts/latest/quickstart.html
  - install **node** and **npm**
  - create an application project folder
  - add a **tsconfig.json** to guide the TypeScript compiler
  - add a **typings.json** that identifies missing TypeScript definition files
  - add a **package.json** that defines the packages and scripts we need
  - **install the npm packages and typings** files

Angular 2 Hello World Project Structure

- angular2-quickstart
  - app
    - app.component.ts
    - main.ts
  - node_modules ...
  - typings ...
  - index.html
  - package.json
  - styles.css
  - tsconfig.json
  - typings.json

app/app.component.ts

```
import {Component} from 'angular2/core';

@Component({
    selector: 'my-app',
    template: '<h1>My First Angular 2 App</h1>'
})

export class AppComponent { }
```

app/main.ts

```
import {bootstrap}    from 'angular2/platform/browser';
import {AppComponent} from './app.component';

bootstrap(AppComponent);
```

index.html: Configure SystemJS

```
<html><head>... <script>
      System.config({
        packages: {
          app: {
            format: 'register',
            defaultExtension: 'js'
          }
        }});
      System.import('app/main').then(null, console.error.bind(console));
    </script>
  </head>
  <body>
      <my-app>Loading...</my-app>
  </body>
</html>
```

Web Components

- Make it possible to build widgets …which can be reused reliably …and which won't break pages if the next version of the component changes internal implementation details.

[http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/]

4 emerging W3C specifications:

- **Custom elements** – provide a way for authors to build their own fully-featured DOM elements.
- **Shadow DOM** – combining multiple DOM trees in one hierarchy

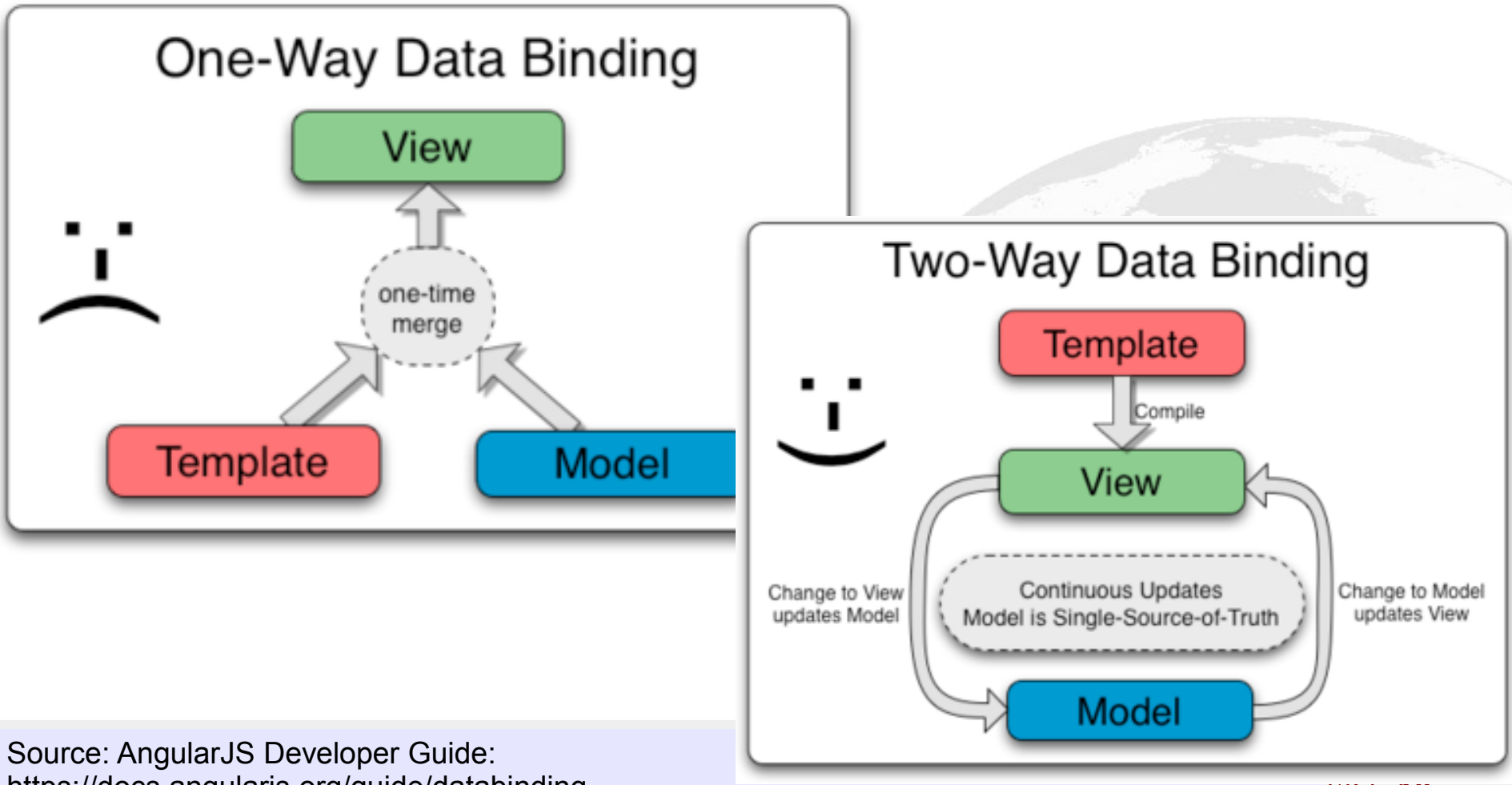
[http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom/]

- **Template** – declare fragments of HTML that can be cloned and inserted in the document by script
- **HTML imports** – <link rel="import" href="my-custom-cmp.html">

Web Components (3)

```
<template id="custom-tag-tpl">
  <style>
     h1 { color: blue; }
  </style>
  <h1>My Custom Component</h1>
</template>
var CustomCmpProto = Object.create(HTMLElement.prototype);
CustomCmpProto.createdCallback = function() {
        var template = document.querySelector('#custom-tag-tpl');
        var clone = document.importNode(template.content, true);
        this.createShadowRoot().appendChild(clone);
};
var MyCmp = document.registerElement('custom-cmp', {prototype: CustomCmpProto});
document.body.appendChild(new MyCmp());
```

Data binding



Source: AngularJS Developer Guide: https://docs.angularjs.org/guide/databinding
License: CC BY-3.0

Differences between Angular 1 and Angular 2

- Component-based:
  - Angular 2 is component based.
  - controllers and $scope - no longer used
  - replaced by components and directives.
  - components are directives with a template
- Directives - specification for directives is considerably simplified, @Directive declarative annotation (decorator)
- Dependency Injection – 3 parts: the injector, bindings, and actual dependencies to be injected
- Forms and Validations - FormBuilder and ControlGroup

Advantages of Angular 2

[http://angularjs.blogspot.bg/2015/11/highlights-from-angularconnect-2015.html]

- **Speed** – Angular 2 is dramatically faster than Angular 1 with support for fast initial loads through server-side pre-rendering, offline compile for fast startup, and ultrafast change detection and view caching for smooth virtual scrolling and snappy view transitions.
- **Browsers** – Angular 2 supports IE 9, 10, 11, Microsoft Edge, Safari, Firefox, Chrome, Mobile Safari, and Android 4.1+.
- **Cross-platform** – By learning Angular 2, you'll gain the core knowledge you'll need to build for a full range of platforms including desktop and mobile web, hybrid and native UI mobile installed apps, and even installable desktop applications.

Data Architecture in Angular 2

Angular 2 Developers Guide:

https://angular.io/docs/ts/latest/guide/

- Data architecture in Angular 2 – overview, main types of components: Module, Component, Template, Metadata, Data Binding, Service, Directive, Dependency Injection.
- Component controllers, views & templates
- Using external template and style files.
- Ng2 by example – Tour of Heroes official Angular 2 tutorial

@Directive

- Directives add behaviour to an existing DOM element. One example use case for @Directive would be to add a highlight on mouse hover. Example – **log-on-click.directive.ts :**

```
import {Directive} from 'angular2/core';

@Directive({
    selector: "[myLogOnClick]",
  host: {
        '(click)': 'onClick()'
  }
})
export class LogOnClickDirective {
    onClick() { console.log('Element clicked!'); }
}
```

Even Better @Directive
[https://github.com/mgechev/angular2-style-guide]

```
import {Directive, HostListener} from 'angular2/core';

@Directive({
    selector: "[myLogOnClick]",
})
export class LogOnClickDirective {
    @HostListener('click')
    onClick() { console.log('Element clicked!'); }
}
```

- Prefer **@HostListener** and **@HostBinding** instead of the **host** property of the **@Directive** and **@Component** decorators

Tooltip @Directive

```
import {Directive, HostListener, Input} from 'angular2/core';
@Directive({
    selector: '[myTooltip]',
    host: {
        '(mouseover)': 'show()'
    }
})
export class TooltipDirective {
    @Input('myTooltip') text: string;
    show() {
        alert(this.text);
    }
}
```

- `<h1 myLogOnClick myTooltip="Enter new hero.">Hero Form</h1>`

Custom pipes

```
import {Pipe, PipeTransform} from 'angular2/core';

@Pipe({ name: 'exponentialStrength' })
export class ExponentialStrengthPipe implements PipeTransform
{
    transform(value: number, [exponent]): number {
        var exp = parseFloat(exponent);
        return Math.pow(value, isNaN(exp) ? 1 : exp);
    }
}
```

- `Super power boost: {{2 | exponentialStrength: 10}}`

Custom Validators

[https://github.com/daviddt/angular2-form-validation-example/]

```
import { Control } from "angular2/common";
interface ValidationResult {
  [key: string]: boolean;
}
export class UsernameValidator {
  static startsWithNumber(control: Control):ValidationResult {
    if (control.value != "" && !isNaN(control.value.charAt(0))) {
       return { "startsWithNumber": true };
    }
       return null;
    }
```

Custom Validators Async (Promise)

[https://github.com/daviddt/angular2-form-validation-example/]

```
constructor(private builder: FormBuilder) {

    this.username = new Control(
        "",
        Validators.compose([Validators.required,
                    UsernameValidator.startsWithNumber]),
                    UsernameValidator.usernameTaken
    );

    this.form = builder.group({
        username: this.username
    });
}
```

Custom Validators Async (Promise)

[https://github.com/daviddt/angular2-form-validation-example/]

```
static usernameTaken(control: Control):Promise<ValidationResult>
{
      return new Promise((resolve, reject) => {
          setTimeout(() => {
              if (control.value === "David") {
                  resolve({ "usernameTaken": true })
              } else {
                  resolve(null);
              };
          }, 1000);
      });
}
```

Angular 2 HTTP client: Injecting HTTP_PROVIDERS services in app.component.ts

```
import {Component, provide} from 'angular2/core';
import { RouteConfig, ROUTER_DIRECTIVES, ROUTER_PROVIDERS }
                                                from 'angular2/router';
import {HeroService} from './hero.service';
import {HeroesComponent} from './heroes.component';
import {HeroDetailComponent} from './hero-detail.component';
import {DashboardComponent} from './dashboard.component';
import {HTTP_PROVIDERS, XHRBackend}    from 'angular2/http';
import {InMemoryBackendService, SEED_DATA}
          from 'a2-in-memory-web-api/core'; //in-memory web api
import {HeroData} from './hero-data';
```

Angular 2 HTTP client: Injecting HTTP_PROVIDERS services in app.component.ts

```
@Component({
    selector: 'my-app',
    directives: [ROUTER_DIRECTIVES],
    providers: [HeroService, ROUTER_PROVIDERS, HTTP_PROVIDERS,
      // in-mem server
      provide(XHRBackend, { useClass: InMemoryBackendService })
      provide(SEED_DATA, { useClass: HeroData }) // in-mem data
    ],
    styleUrls: ['app/app.component.css'],
    Template: `...  `
})
export class AppComponent{
    public title = 'Tour of Heroes';
}
```

Angular 2 HTTP Client (1): Setup

```
import {Injectable} from 'angular2/core';
import {Http, Response, Headers, RequestOptions} from
'angular2/http';
import {Observable}     from 'rxjs/Observable';
import {Hero} from './hero';
import 'rxjs/Rx';

@Injectable()
export class HeroService {
    private _heroesUrl = 'app/heroes';  // URL to web api
    private _heroes: Hero[] = [];
    constructor(private _http: Http) {}
```

Angular 2 HTTP Client (2): Read

```
getHeroes(): Promise<Hero[]> {
    return this._http.get(this._heroesUrl)
        .map(response => <Hero[]>response.json().data)
        .do(heroes => this._heroes = heroes)
        .do(data => console.log(data))
        .catch(this.handleErrorObservable).toPromise();
}
private handleErrorObservable(error: Response) {
    // in real app we may send error to remote logging
    console.error(error);
    return Observable.throw(
                    error.json().error || 'Server error');
}
```

Angular 2 HTTP Client (3) :Cache

```
private getCachedHeroes(): Promise<Hero[]> {
    return this._heroes ? Promise.resolve(this._heroes)
                        : this.getHeroes();
}
getHero(id: number): Promise<Hero> {
    return Promise.resolve(this.getCachedHeroes())
        .then(
   heroes => heroes.filter(hero => hero.id === id)[0]
        );
}
private handleErrorPromise(error: any) {
    console.error(error);
    return Promise.reject(error.message || error.json().error
                                        || 'Server error');
}
```

Angular 2 HTTP Client (4): Create

```
addHero(name: string): Promise<Hero> {
  return this.getCachedHeroes()
      .then(heroes => {
          let nextHeroId = heroes.reduce((prevMaxId, next) =>
              next.id > prevMaxId ? next.id : prevMaxId, 0) + 1;
          let newHero = new Hero(nextHeroId, name);
          let body = JSON.stringify({ name });
          let headers = new Headers(
                        { 'Content-Type': 'application/json' });
          let options = new RequestOptions(
                        { headers: headers });
          return this._http.post(this._heroesUrl, body, options)
              .toPromise()
              .then(res => <Hero>res.json().data)
              .catch(this.handleErrorPromise);
      });
}
```

Angular 2 HTTP Client (5): Update

```
editHero(hero: Hero): Promise<void> {
  let body = JSON.stringify( hero );
  let headers=new Headers({'Content-Type': 'application/json'});
  let options = new RequestOptions({ headers: headers });
  return this._http.put(this._heroesUrl + "/" + hero.id, body,
                                                    options)
      .toPromise().then(response => {
          console.log(response);
          if(response.status == 204) // No content
              return Promise.resolve();
          else
              return Promise.reject('Error updating  hero '
                  + hero.id + ":" + hero.name + ' - status code:'
                  + response.status
              );
      }).catch(this.handleErrorPromise);
}
```

Cross-Domain Requests Using JSONP (1)

[https://angular.io/docs/ts/latest/guide/server-communication.html]

```
import {Component}         from 'angular2/core';
import {JSONP_PROVIDERS}   from 'angular2/http';
import {Observable}        from 'rxjs/Observable';
import {Subject}           from 'rxjs/Subject';
import {WikipediaService} from './wikipedia.service';
@Component({
    selector: 'my-wiki-smart',
    template: `<h1>Smarter Wikipedia Demo</h1>
  <p><i>Fetches when typing stops</i></p>
  <input #term (keyup)="search(term.value)"/>
  <ul>
   <li *ngFor="#item of items | async">{{item}}</li>
  </ul>` ,
  providers: [JSONP_PROVIDERS, WikipediaService]
})
```

Cross-Domain Requests Using JSONP (2)
[https://angular.io/docs/ts/latest/guide/server-communication.html]

```
export class WikiSmartComponent {
    constructor(private _wikipediaService: WikipediaService) { }
    private _searchTermStream = new Subject<string>();
    search(term: string) { this._searchTermStream.next(term); }
    items: Observable<string[]> = this._searchTermStream
        .debounceTime(300)
        .distinctUntilChanged()
        .switchMap(
         (term: string) => this._wikipediaService.search(term));
}
```

Reactive Programming. Functional Programing

- **Reactive Programming** [Wikipedia]: a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow. Ex: **a := b + c**
- **Functional Programming** [Wikipedia]: a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm. Eliminating side effects can make it much easier to understand and predict the program behavior. Ex: **book -> book.getAuthor().fullName()**

Reactive Programming = Programming with Asynchronous Data Streams

- **Functional Reactive Programming (FRP)** [Wikipedia]: a programming paradigm for reactive programming (asynchronous dataflow programming) using the building blocks of functional programming (e.g. map, reduce, filter). FRP has been used for programming graphical user interfaces (GUIs), robotics, and music, aiming to simplify these problems by explicitly modeling time. Example (RxJava):

```
Observable.from(new String[]{"Reactive", "Extensions", "Java"})
    .take(2).map(s -> s + " : on " + new Date())
    .subscribe(s -> System.out.println(s));
```

```
Result: Reactive : on Wed Jun 17 21:54:02 GMT+02:00 2015
        Extensions : on Wed Jun 17 21:54:02 GMT+02:00 2015
```

Good intro tutorial in RP using RxJS by Andre Staltz see: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
JS Fiddle of the demo: http://jsfiddle.net/staltz/8jFJH/48/

Definitions of Reactive Programming

- Microsoft® opens source polyglot project ***ReactiveX*** (Reactive Extensions) [http://reactivex.io]:

  Rx = Observables + LINQ + Schedulers :)

  - Supported Languages – Java: RxJava, JavaScript: RxJS, C#: Rx.NET, C#(Unity): UniRx, Scala: RxScala, Clojure: RxClojure, C++: RxCpp, Ruby: Rx.rb, Python: RxPY, Groovy: RxGroovy, JRuby: RxJRuby, Kotlin: RxKotlin, Swift: RxSwift
  - ReactiveX for platforms and frameworks: RxNetty, RxAndroid, RxCocoa

- ***Reactive Streams*** Specification [http://www.reactive-streams.org/] used by ***Project Reactor*** [http://projectreactor.io/, https://github.com/reactor/reactor]

RxJS – JS ReactiveX (Reactive Extensions)

[http://reactivex.io, https://github.com/ReactiveX]

- **ReactiveX** is a polyglot library for composing asynchronous and event-based programs by using observable sequences.
- It extends the **observer pattern** to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety, concurrent data structures, and non-blocking I/O.
- Allow composing flows and sequences of asynchronous data.
- Consuming and manipulating a stream of events as collections that can be modified and transformed by various operations
- Follows the **onNext(), onError(), onCompleted()** flow
- Aims to solve the "callback hell" problem

Resources: RxMarbles and RxJS Coans

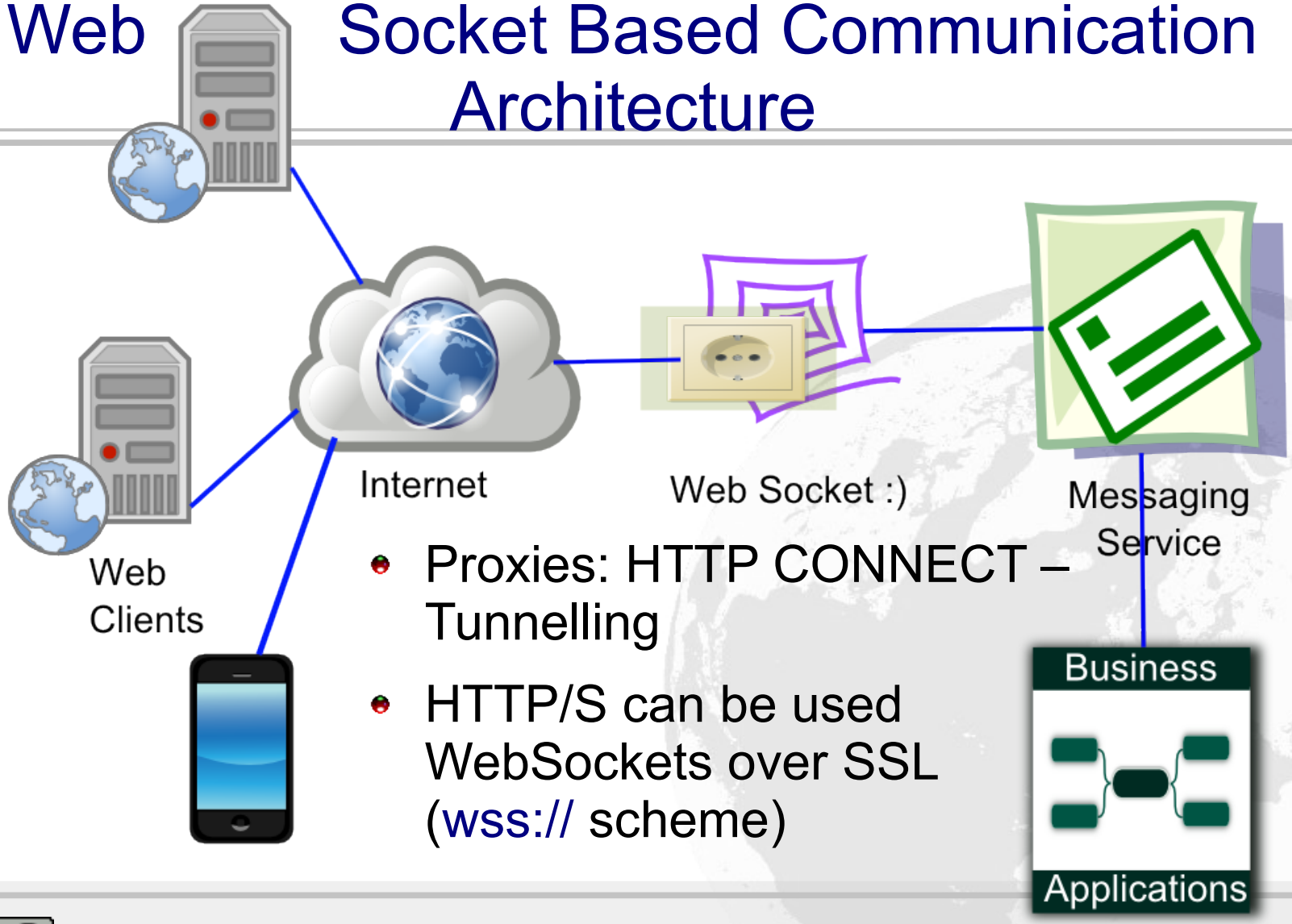
RxMarbles:

http://rxmarbles.com/

RxJS Coans:

https://github.com/Reactive-Extensions/RxJSKoans

Web Socket Based Communication Architecture



- Proxies: HTTP CONNECT – Tunnelling
- HTTP/S can be used WebSockets over SSL (wss:// scheme)

WebSocket Main Applicatin Areas

- Massively multiplayer online role-playing game (MMORPG)
- Online trading – large scale auctions, stock tickers
- Interactive synchronous communication – chat, audio- & video-conferencing
- Collaborative authoring, groupware & social applications - including modelling and art
- Dynamic data monitoring and control – e.g. management dashboards presenting SLA, KPI and BI data in real time
- Remote observation and control of devices and services – e.g. remote monitoring of home security, energy consumption, data center services and devices performance visualizations

IETF WebSocket protocol (RFC 6455) (1)

- Official IETF standard - RFC 6455
- TCP-based full-duplex protocol
- Starts as standard HTTP /HTTPS connection to web server port **80/443** (**handshake phase**) – easy firewall and proxy traversal without the overhead connected with polling
- Uses HTTP protocol upgrade mechanism (Upgrade: websocket + Connection: Upgrade) – communication is immediately upgraded to more efficient WebSocket protocol (**data transfer phase**)
- Allows bidirectional streaming of data (partial messages)

IETF WebSocket protocol (RFC 6455) (2)

- Designed with security and extensibility in mind: Origin validation, sub-protocols & extensions negotiation (through standardized HTTP headers exchanged in handshake phase)
- **WebSocket API in Web IDL** is being standardized by **W3C**
- Supported by latest versions of all major web browsers –

Implementation status

Protocol	Draft date	Internet Explorer	Firefox[17] (PC)	Firefox (Android)	Chrome (PC, Mobile)	Safari (Mac, iOS)	Opera (PC, Mobile)	Android Browser
hixie-75	February 4, 2010				4	5.0.0		
hixie-76 hybi-00	May 6, 2010 May 23, 2010		4.0 (disabled)		6	5.0.1	11.00 (disabled)	
7 hybi-07	April 22, 2011		6[18]1					
8 hybi-10	July 11, 2011		7[19]1	7	14[20]			
13 RFC 6455	December, 2011	10[21]	11	11	16[22]	6	12.10[23]	4.4

[1] Gecko-based browsers versions 6–10 implement the WebSocket object as "MozWebSocket",[24] requiring extra code to integrate with existing WebSocket-enabled code.

Source: Wikipedia, [http://en.wikipedia.org/wiki/WebSockets]

WebSocket Request Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:
GET /ipt-present/ws HTTP/1.1
Host: localhost:8080
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: 3RhAwlJCs7wbj3xUdeDTXA==
Sec-WebSocket-Protocol: epresentation, ipt_present
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits, x-webkit-deflate-frame
Origin: http://localhost:8080

WebSocket Response Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:

HTTP/1.1 **101 Switching Protocols**

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: QIMZj0lbIv1TM+JMx/JsoSKwYb8=

Sec-WebSocket-Protocol: epresentation

Server: GlassFish Server Open Source Edition 4.0

X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.0 Java/Oracle Corporation/1.7)

W3C JavaScript WebSocket API [Web IDL]

```
WebSocket WebSocket(
  in DOMString url,
  in optional DOMString protocols
);

OR

WebSocket WebSocket(
  in DOMString url,
  in optional DOMString[] protocols
);
```

W3C WebSocket API - Methods [Web IDL]

```
void send(
  in DOMString data
);

void send(
  in ArrayBuffer data
);

void send(
  in Blob data
);
```

```
void close(
  in optional unsigned short code,
  in optional DOMString reason
);
```

Constant	Value	Description
CONNECTING	0	Connection is not open yet
OPEN	1	Connection is open and ready to communicate
CLOSING	2	Connection is in the process of closing
CLOSED	3	Connection is closed / can not be opened

WebSocket JS API Example (1)

- Example 1:

```
connection = new WebSocket('ws://h2j.org/echo', ['soap', 'xmpp']);
```

- Example 2:

```
var rootWsUri = "ws://" + (document.location.hostname.length >0 ?
    document.location.hostname : "localhost") + ":" +
    (document.location.port.length > 0 ?
        document.location.port : "8080") + "/ipt-present/ws";

var websocket = new WebSocket( rootWsUri );
```

WebSocket JS API Example (2)

```
websocket.onopen = function (event) {
    onOpen(event);
};
websocket.onmessage = function (event) {
    onMessage(event)
};
websocket.onerror = function (event) {
    onError(event)
};
```

WebSocket JS API Example (3)

```
function onMessage(evt) {
        var jso = JSON.parse(evt.data);
        switch(jso.type){
          case "login-resp":
            conversationId = jso.cid;
            $(".logged-name").html(" - " + userName);
            $("#button-login").hide();
            $("#button-logout").show();
            showToster(jso.data.message, "info");
            break;
          ( - continues in next slide - )
```

Java™ EE 7 WebSocket support: Java API for WebSocket (JSR-356)

- What about the server-side support?
- Now easier than ever – Java™ EE 7 added two new APIs:
  - JSR-356: Java API for WebSocket
  - JSR-353: Java API for JSON Processing
- The new APIs facilitate rapid developemnt of WebSocket client and server applications (endpoints):
  - programmatic – by extending standard class Endpoint;
  - using annotations – @ServerEndpoint, @ClientEndpoint, @OnOpen, @OnClose, @OnMessage, @OnError, @PathParam

Programmatic Implementation of WebSocket Server Endpoints

```
import javax.websocket.*;
import javax.websocket.server.*;
public class MyEchoEndpoint extends Endpoint {
    @Override
    public void onOpen(final javax.websocket.Session session,
            EndpointConfig config) {
        session.addMessageHandler(new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                try {
                    session.getBasicRemote().sendText(message);
                } catch (IOException e) { }
            }
        });
    }
}
```

Implementing WebSocket Using Annotations (1)

```
@ServerEndpoint(value = "/ws",
            decoders = {PresentationMessageDecoder.class},
           encoders = {PresentationMessageEncoder.class})
public class IPTPresentationEndpoint {

private static Set<Session> sessions =
        Collections.newSetFromMap(
             new ConcurrentHashMap<Session, Boolean>());

   @OnOpen
   public void onOpen(Session session) {
      sessions.add(session);
   }
```

Implementing WebSocket Using Annotations (2)

```
@OnClose
public void onClose(Session session) {
    sessions.remove(session);
}
@OnMessage
public void onMessage(PresentationMessage message, Session session) {
    String name;
    try{
        switch (message.getType()) {
            case LOGIN_ACTION:
                name = message.getPayload().getString("name", "Anonimous");
                session.getUserProperties().put("name", name);
                 ...
```

Implementing WebSocket Using Annotations (4)

```
    private void sendMessageToOthers(Session currentSession, String
        messageType, String message) throws EncodeException, IOException {
        for(Session s: sessions){
            if(!s.getId().equals(currentSession.getId())){
                s.getBasicRemote().sendObject(
                    new PresentationMessage(messageType, s.getId(),
                        Json.createObjectBuilder().add("message", message).build()));
            }
        }
    }
}
```

- Alternative approach – use **session.getOpenSessions()** to access all sessions to the same endpoint

Java API for WebSocket Details (1)

- Types of messages:
  - Text (String, Reader, custom Object decoded using provided Decoder.Text or Decoder.TextStream)
  - Binary (ByteBuffer, byte[], InputStream, custom Object decoded using provided Decoder.Binarry or Decoder.BinarryStream)
  - Ping – echo response handled automatically, no custom handling needed
  - Pong (PongMessage)
- Main annotations: @ServerEndpoint, @ClientEndpoint, @OnOpen, @OnClose, @OnMessage, @OnError, @PathParam

Java API for WebSocket Details – Example (1)

```
@ServerEndpoint("/topic/{id}")
public class MyEchoEndpoint {
    @OnMessage
    public void textMessage(@PathParam ("id") String id, Session
        session, String message) {
        System.out.println("Topic " + id + " - Text message: " + message);
    }
    @OnMessage
    public void binaryMessage(@PathParam ("id") String id, Session
        session, ByteBuffer message) {
        System.out.println("Topic " + id + " - Binary message: " +
            message.toString());
    }
```

Asynchronous processing of messages - **RemoteEndpoint.Async**

- void setSendTimeout(long timeoutmillis)
- sendText(String text, SendHandler handler)
- Future<Void> sendText(String text)
- void sendBinary(ByteBuffer data, SendHandler handler)
- Future<Void> sendBinary(ByteBuffer data)
- void sendObject(Object data, SendHandler handler)
- Future<Void> sendObject(Object data)

Example: Using Custom Decoder (1)

```
public class PresentationMessageDecoder implements
Decoder.Text<PresentationMessage> {
    @Override
    public PresentationMessage decode(String jsonData) throws
DecodeException {
        JsonObject obj;
        PresentationMessage message = null;
        try (JsonReader jsonReader = Json.createReader(
            new StringReader(jsonData))) {
            obj = jsonReader.readObject();
            if (obj.containsKey("type") && obj.containsKey("sid") &&
                obj.containsKey("data")) {
                    ( - continues in next slide - )
```

Example: Using Custom Decoder (2)

```
            message = new PresentationMessage(
                    obj.getString("type"), obj.getString("sid"),
                    obj.getJsonObject("data"));
        } else {
            throw new DecodeException(jsonData, "Invalid json string");
        }
    }
    return message;
}
```

(- continues in next slide -)

Example: Using Custom Decoder (3)

```
@Override
public boolean willDecode(String jsonData) {
    JsonObject obj;
    try (JsonReader jsonReader = Json.createReader(new
       StringReader(jsonData))) {
        obj = jsonReader.readObject();
        if (obj.containsKey("type") && obj.getString("type").length() > 0) {
            String type = obj.getString("type");
            switch (type) {
                case LOGIN_ACTION:
                case  … :
                    return true;
            } } }    return false;
}
```

Example: Using Custom Encoder (1)

```
public class PresentationMessageEncoder implements
                                        Encoder.Text<PresentationMessage> {
    @Override
    public String encode(PresentationMessage message) throws
EncodeException {
        JsonObject obj = message.getPayload();
        JsonObject response = Json.createObjectBuilder()
            .add("type", message.getType())
            .add("sid", message.getSessionId())
            .add("data", obj).build();
        return response.toString();
    }
...
}
```

References

- Internet Engineering Task Force (IETF) WebSocket Protocol Specification – http://tools.ietf.org/html/rfc6455
- W3C The Web Sockets API – http://www.w3.org/TR/2009/WD-websockets-20091029/
- Mozilla Developer Network (MDN) – https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications
- Internet of Things (IoT) in Wikipedia – http://en.wikipedia.org/wiki/Internet_of_Things
- JSR 356: JavaTM API for WebSocket – https://jcp.org/en/jsr/detail?id=356
- JSR 353: Java API for JSON Processing - Reference Implementation – https://jsonp.java.net/

Thanks for Your Attention!

Questions?