



# Java EE 7 in Practice: Blueprints Reborn

Ed Burns

JSF Spec Lead

[edward.burns@oracle.com](mailto:edward.burns@oracle.com)

[@edburns](#)

# My Plan for Your Time Investment

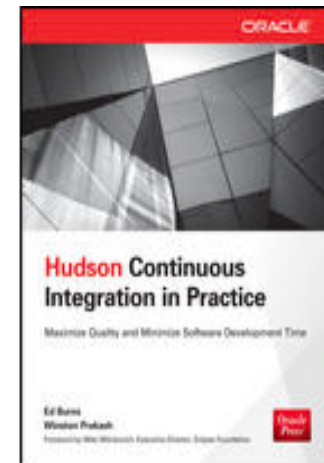
- Java EE 7 Overview
- Cargo Tracker Overview
- Java EE 7 Feature Traversal
- Resources

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

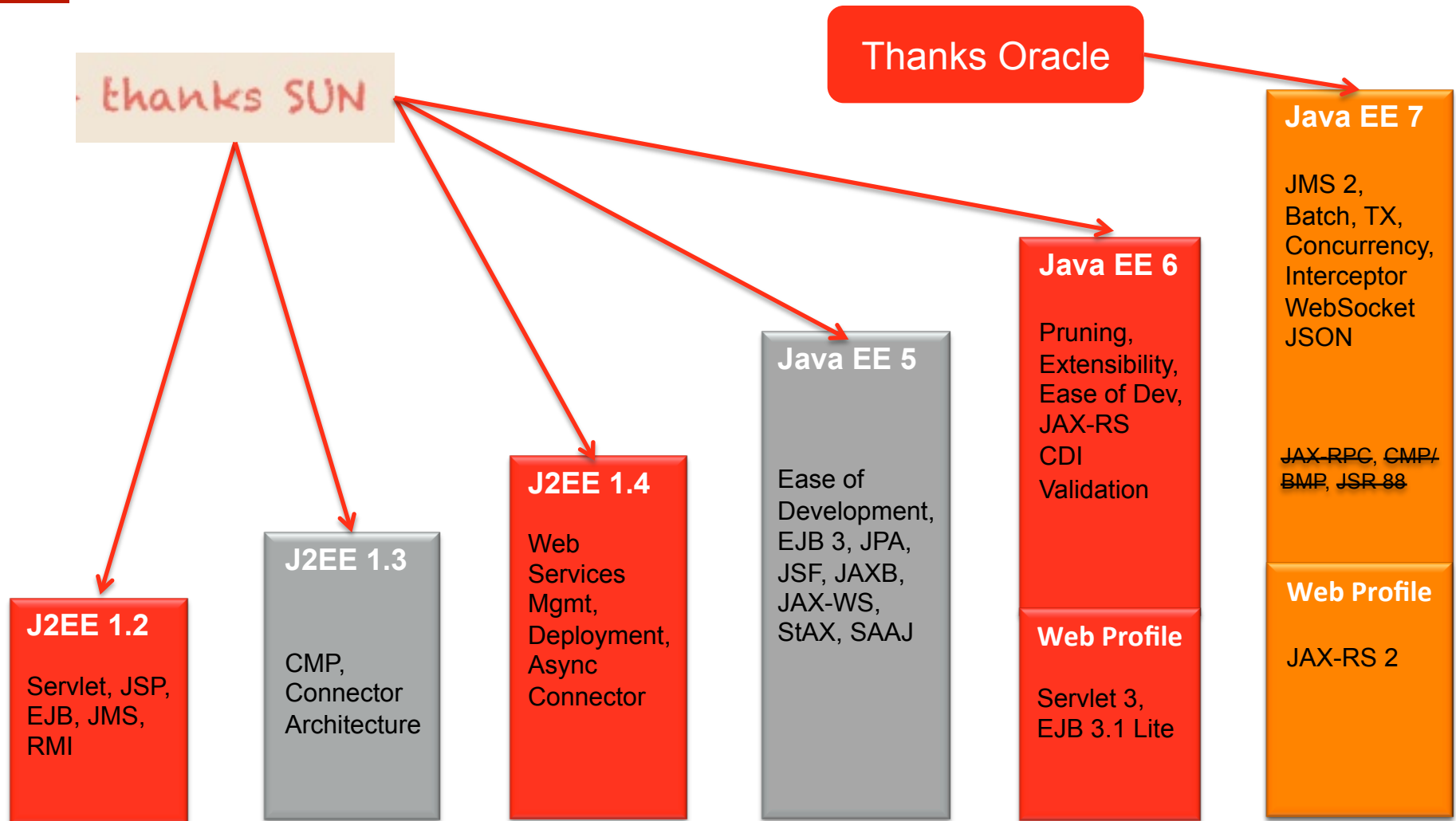
# Speaker Qualifications – Ed Burns

## And non-qualifications

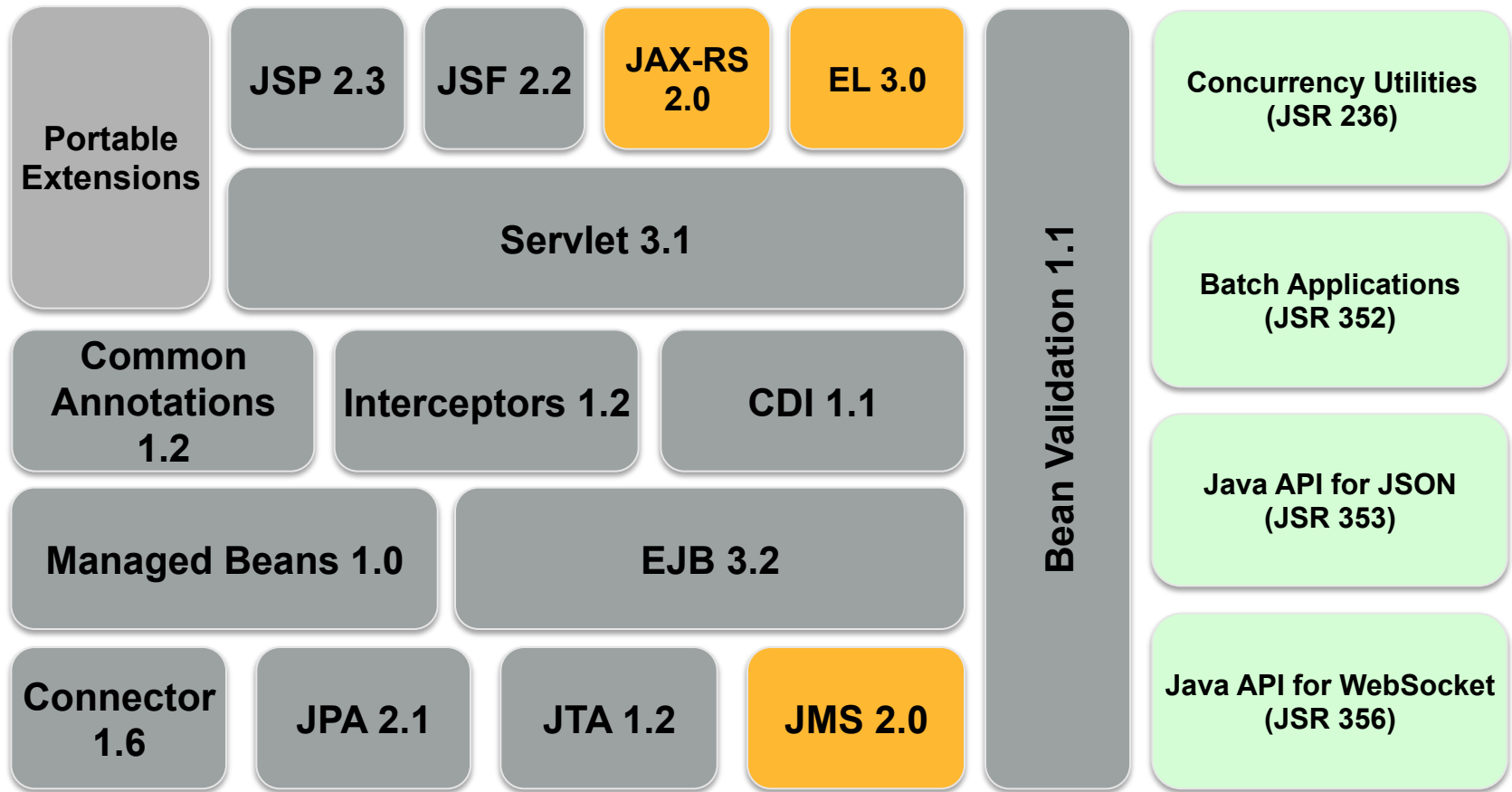
- Involved with JSF since 2002
- Spec lead since 2003
  - Most fun part of the job: cleanly integrating other people's great ideas into JSF (and hopefully improving on the in the process)
  - Not an expert in applying JSF in practice
- Author of four books for McGraw-Hill



# Java EE Past, Present and Future



# Java EE 7



 New  Major Release  Updated

# Java EE in Action

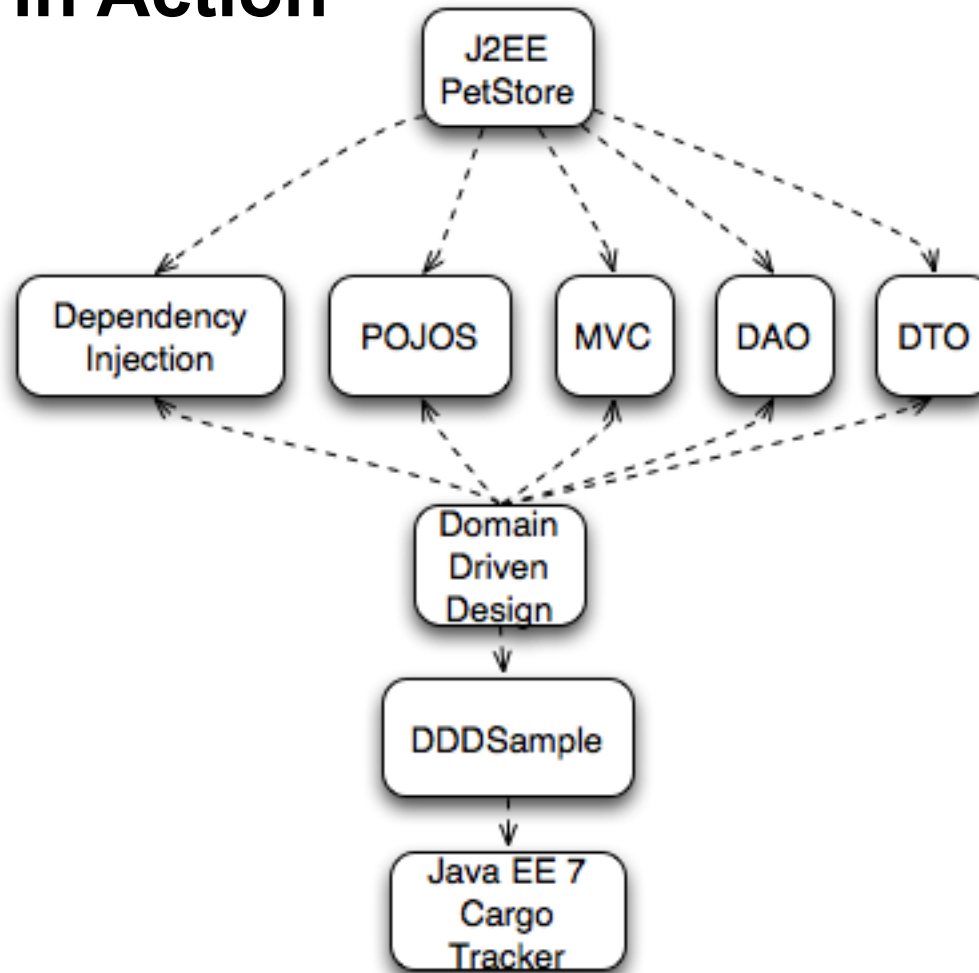


## *Cargo Tracker*

Applied domain-driven design blueprints for Java EE

<http://cargotracker.java.net>

# Java EE in Action



## Cargo Tracker

Applied domain-driven design blueprints for Java EE



CargoTracker: Wiki: Home — Project Kenai

message.html SPEC\_WORKING SPEC Mojarra T GF VPN Mojarra console test jsf-planning ant

Recently Bookmarked Cloning Window... Pearson | ePAT Oracle, Java and... FIRST Newslette... LiveLeak.com - ...

Home Projects Forums People Java User Groups JCP My Projects

Project Features

- Message Forums
- Message Forum
- Mailing Lists
- Commits Archive
- Issues Archive
- Mailing List Archive
- Downloads
- Issue Tracking
- Source Code Repositories
- Cargo Tracker Source
- Code R...
- Wiki
- WikiHomePage

Wiki Controls

- Show Page
- Edit Page
- Show Page as Text
- Show Page Revision History

Jump to Page:

Home

List all Pages

Manage Images

Create New Page:

GO

About this Project

CargoTracker is a subproject of BluePrints, was started in April 2013 and has 13 members. The project administrators are Ed Bratt, shreedhar.ganapathy, and Reza Rahman.

java.net> projects> cargotracker> wiki> Home

Welcome edburns

Last updated 1 week ago, by Reza Rahman

# Cargo Tracker

Applied domain-driven design blueprints for Java EE

## Overview

This project demonstrates how you can develop applications with the Java EE platform using widely adopted architectural best practices like Domain-Driven Design (DDD). The code is intended to mirror a non-trivial application that developers in the real world would work on. It attempts to demonstrate first-hand how you can use Java EE to effectively meet practical enterprise concerns such as productivity, agility, testability, flexibility, maintainability, scalability and security.

The project is directly based on the well known original Java DDD sample (<http://dddsample.sourceforge.net>) application developed by DDD pioneer Eric Evans' company Domain Language and the Swedish software consulting company Citerus. The cargo example actually comes from Eric Evans' seminal book. In the same vein as that project, we certainly are not in the business of prescribing how you should write Java EE applications. The code simply demonstrates ideas that you could adopt if you think they are right for you. We've also tried to show you just some of the many different approaches that you could take to interpret and implement DDD concepts. You are of course warmly welcomed to provide feedback and contribute to this Open Source project.

As such, we will continue to rely heavily on the original DDD sample application as well as the existing knowledge base in the DDD community as invaluable resources on further details on DDD as it applies to this code base (please see the [Resources](#) page for vital details). We also highly recommend you read the [DDD and Java EE](#) section. It is perhaps best to skim it, explore the application/code and revisit the section.

This effort is very grateful for the kind blessings and guidance of the good folks behind the original DDD sample application.

## Goals

- The primary goal of this project is demonstrating well established architectural patterns/blueprints for enterprise development with Java EE using pretty close to a real world application.
- Demonstrating a concrete implementation of DDD concepts.
- Showcase some core Java EE technologies (please see section on [Java EE Features](#)).
- Contrary to beliefs long held by some folks, Java EE does not attempt to be a walled garden. Like all open standards, the goal of Java EE is to provide a reliable core standard foundation that a vibrant ecosystem of plug-ins and extensions can be built around. In that spirit, we will incorporate a select set of representative tools that complement Java EE well such as Maven, JUnit, Cargo, JMeter, soapUI, Arquillian, PrimeFaces and DeltaSpike (is there a reasonable Oracle Open Source project we can add here? Avatar? EclipseLink NoSQL?).

## Non-Goals

- Comparison with other technologies such as Spring, .NET and Ruby on Rails. We suggest you do your own research using the plethora of good information already available. This project is squarely about showing you how you could write good Java EE applications and nothing else.
- Attempting to demonstrate the very wide gamut of Java EE APIs and features. While we do incorporate a pretty representative set of Java EE APIs and features, it is not our goal to serve as a kitchen sink of comprehensive Java EE samples. The [GlassFish samples project](#) serves that purpose far better than us.
- While it is certainly possible to learn certain aspects of Java EE (primarily how to architect good Java EE applications), this project is not intended to teach you the basics of Java EE. The official [Java EE Tutorial](#) and [First Cup](#) starter applications are intended as basic learning tools for Java EE.

## Getting Started

If you simply wish to explore the code, you can download it as a zip from the [downloads](#) page. Alternatively, you can browse the [repository](#) online. The source is a Maven project, so you should be able to easily build it or set it up in your favorite IDE. We have instructions for [NetBeans](#). We welcome a contribution showing how to use the project with Eclipse and other IDEs. You can also run the application directly from the Maven command line using Apache Cargo. All you need to is navigate to the project source root and type:

```
mvn cargo:run
```

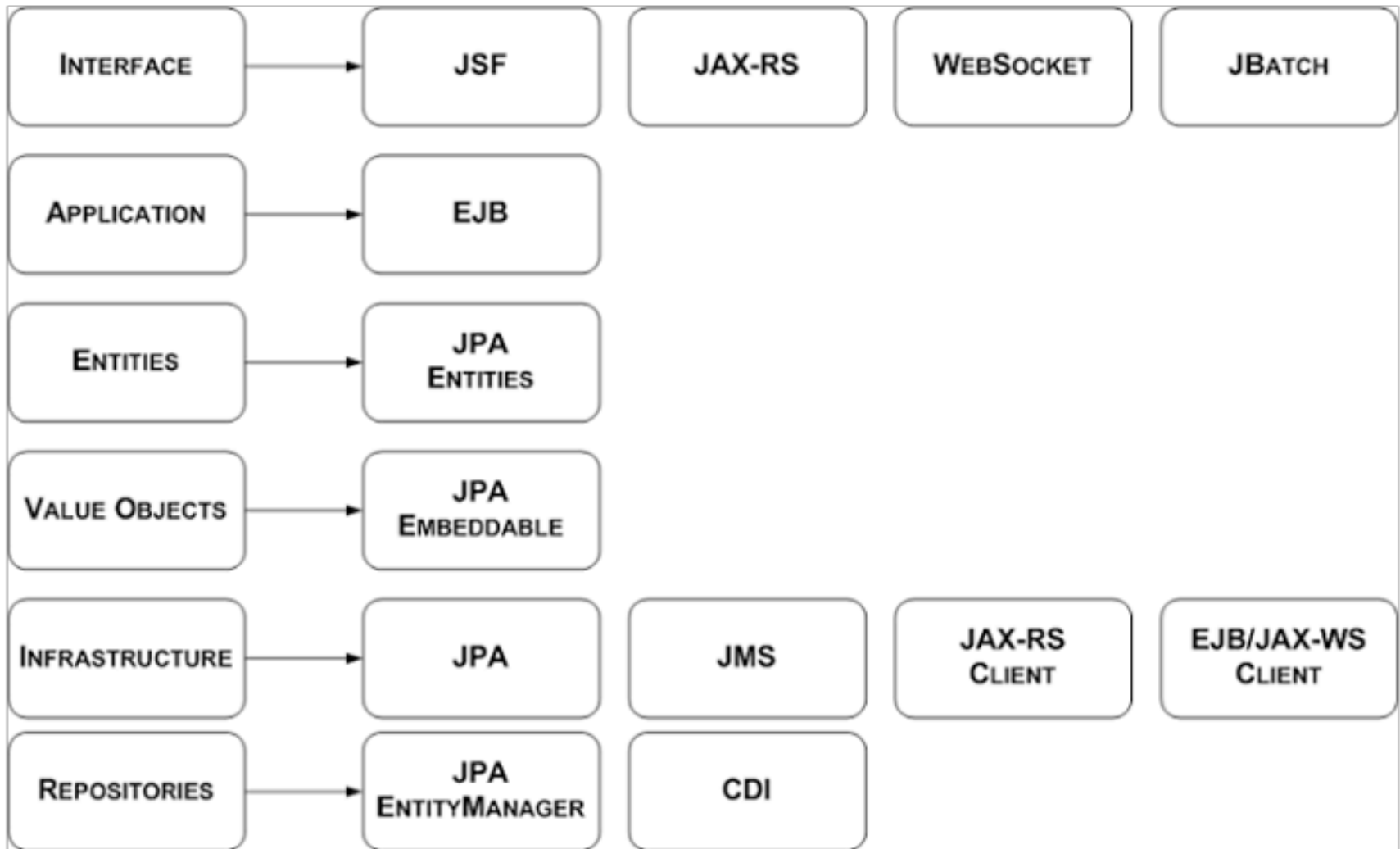
Once the application starts up, just open up a browser and navigate to <http://localhost:8080/cargo-tracker/>.

The project currently runs on Java EE 7/GlassFish 4 and Java SE 7. We welcome contributions for running the application on other Java EE application servers (as you will see in the source code, there are few dependencies on GlassFish so this should be pretty easy to do). Our roadmap (expressed as [JIRA Issues](#)) includes plans to create a Java EE 6

<http://cargotracker.java.net>

ORACLE

# Java EE 7 in Action



# Java EE 7 in Action

## Components in this talk

- JSF 2.2
- Bean Validation 1.1
- JAX-RS 2.0
- WebSocket 1.0
- JSON-P 1.0

# JSF 2.2 HTML5 Friendly Markup

The best part of Wicket comes to JSF

- This is a JSF page

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:myNS="http://xmlns.jcp.org/jsf">
<form myNS:id="form">
  <input name="textField" type="text" myNS:value="#{bean.text1}" />
  <input type="submit" myNS:id="submitButton" value="submit" />
  <p>submitted text: #{bean.text1}.</p>
</form>
</html>
```

# 1090 HTML5 Friendly Markup

Let's get back to basics

- JSF Views are written in a View Declaration Language (VDL).
- The standard Facelet VDL is an XML application with two kinds of elements
  - HTML Markup
  - JSF Components
- HTML Markup is passed through straight to the browser
- JSF Components take some action on the server, during the lifecycle

# 1090 HTML5 Friendly Markup

Let the elegance of HTML shine through

- Before JSF 2.2
  - JSF tags hide complexity of underlying HTML+script+css+images
  - JSF “Renderer”:
    - encode: markup to browser
    - decode: name=value from browser

```
<html>...
```

```
  <my:colorPicker value="#{colorBean.color2}" />
```

```
  <my:calendar value="#{calendarBean.date1}" />
```

```
</html>
```

- Context: Missing feature in browser? Write a JSF component.

# 1090 HTML5 Friendly Markup

Let the elegance of HTML shine through

- With JSF 2.2
  - Pure HTML+script+css+images in the JSF page
  - JSF Renderer handles decode from browser
    - Leverage the strength of the JSF lifecycle
    - Leverage the expressiveness of HTML5

```
<html>...
```

```
  <input type="color" jsf:value="#{colorBean.color2}"/>
```

```
  <input type="date" jsf:value="#{calendarBean.date1}" />
```

```
</html>
```

- Context: New feature in browser? Use “pass through elements”

# JSF 2.2 HTML5 Friendly Markup

Let the elegance of HTML shine through

- DEMO





# Java EE 7 in Action

## Components in this talk

- JSF 2.2
- Bean Validation 1.1
- JAX-RS 2.0
- WebSocket 1.0
- JSON-P 1.0

# Bean Validation – History Lesson

- Just as JSF had managed beans and inversion of control years before CDI, JSF also had validation years before Bean Validation
- Task for JSF 2.0 was to seamlessly integrate Bean Validation into JSF
- Biggest hurdle: conceptual mismatch
  - When does validation occur?
    - JSF: before values are in the model
    - Bean Validation: after values are in the model (but before they are persisted via JPA)

# Bean Validation – History Lesson

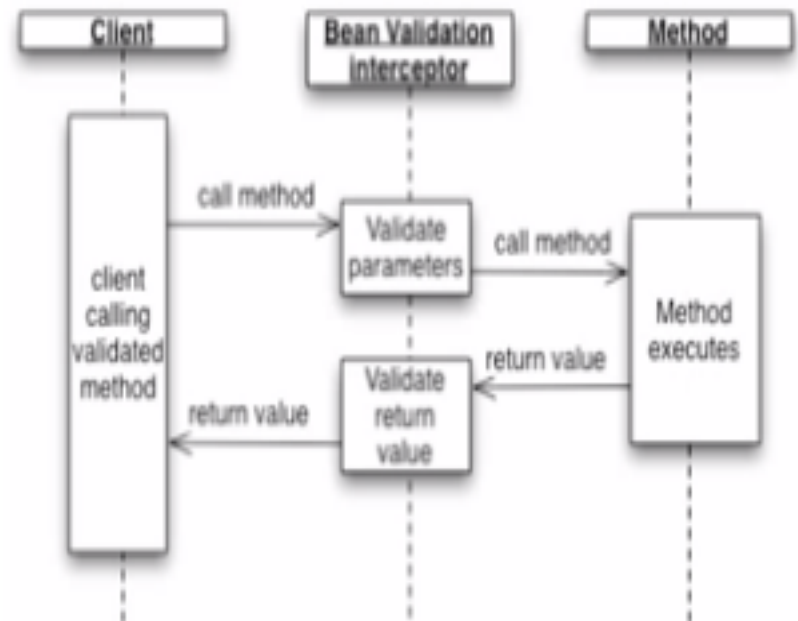
- Answer: define a standard JSF validator  
`javax.faces.Bean`
- Its `validate()` method calls the `validateValue()` method added to the Bean Validator API at the request of JSF.
- This allows the JSF validation reporting system to take action, while allowing Bean Validation be effective as well.

# Bean Validation

- Write once, validate anywhere
- Defines validation behavior for use in all tiers of Java EE (and SE)
- Works well with JSF, JPA, and even Servlet

# Bean Validation 1.1: New Features

- Approaches Programming By Contract (PBC)
  - Validates incoming parameters and outgoing return value
- Can inject with CDI into Validator implementations
- Better message interpolation



# BeanValidation 1.1 PostCondition validation

Programming by Contract comes to Java EE

- DEMO



# Java EE 7 in Action

## Components in this talk

- JSF 2.2
- Bean Validation 1.1
- JAX-RS 2.0
- WebSocket 1.0
- JSON-P 1.0

# JAX-RS 2.0

```
@Stateless // TODO Make this a stateless bean for better scalability.
@Path("/handling")
public class HandlingReportService {

    public static final String ISO_8601_FORMAT = "yyyy-MM-dd HH:mm";
    @Inject
    private ApplicationEvents applicationEvents;

    @POST
    @Path("/reports")
    @Consumes(MediaType.APPLICATION_JSON)
    // TODO Better exception handling.
    public void submitReport(@NotNull @Valid HandlingReport handlingReport) {
        try {

xhr.open("POST", "http://localhost:8080/cargo-tracker/rest/handling/reports", true);
xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8');
xhr.send(JSON.stringify(data));
xhr.onreadystatechange = update;

```



# Java EE 7 in Action

## Components in this talk

- JSF 2.2
- Bean Validation 1.1
- JAX-RS 2.0
- WebSocket 1.0
- JSON-P 1.0

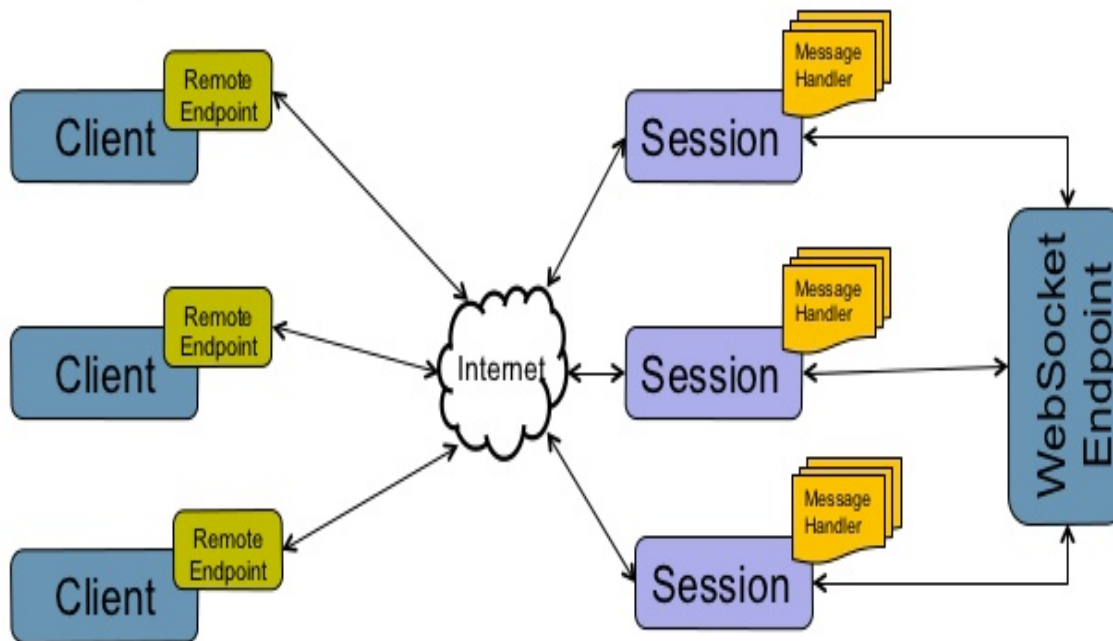
# WebSockets

## What's In a name?

- Several moving parts, each with its own standard!
  - Client: W3C JavaScript API <http://dev.w3.org/html5/websockets/>
  - Transport: IETF RFC 6455 <http://www.ietf.org/rfc/rfc6455.txt>
  - Server: JSR-356: <http://jcp.org/en/jsr/detail?id=356>
- Even with all these parts, it's still very understandable
  - Client: 17 page downs
  - Transport: 86 page downs (about a third of which you can skip)
  - Server: 43 pages

# WebSockets

## Object Model



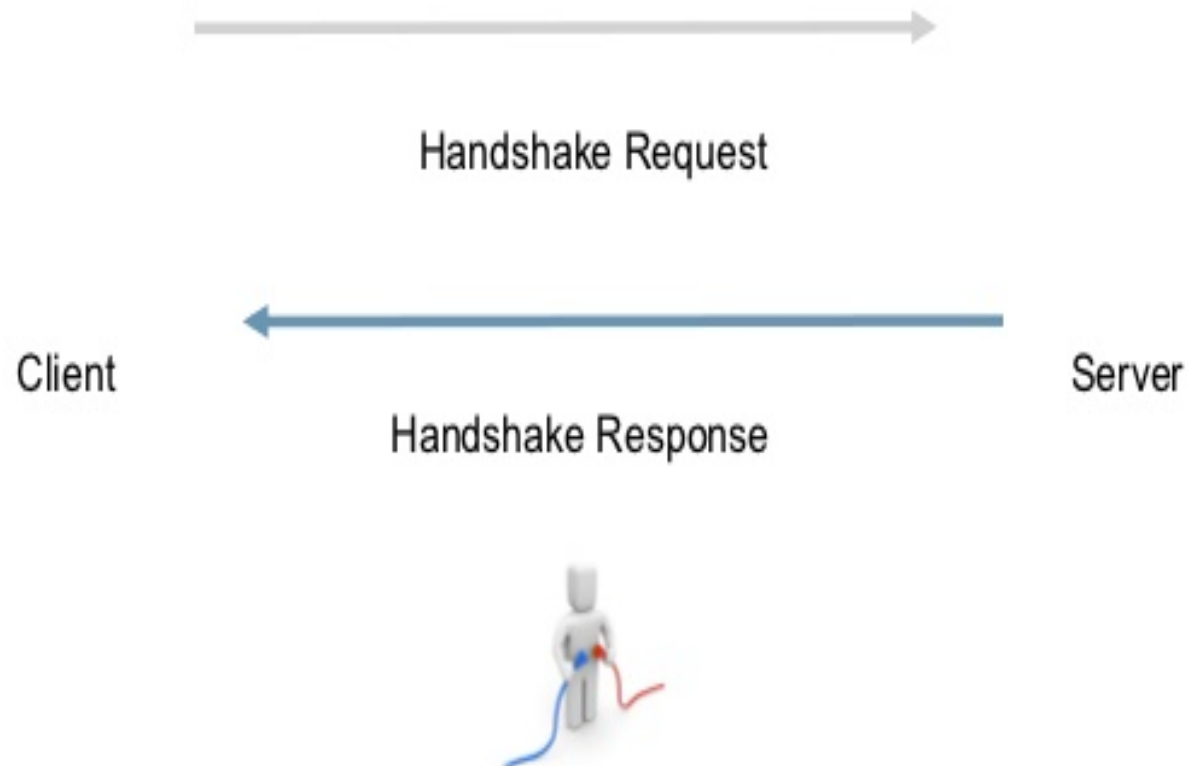
# WebSocket

## Big Picture

- It really is a Socket for the Web
  - It just works over proxies
- Lets you do TCP easily in a web app
  - Establish the connection
  - Send messages both ways
  - Send messages independent of timing
  - Close the connection
- Two basic types: text and binary

# WebSocket

Establish the connection



# WebSocket

TCP over HTTP, use the Upgrade: header

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

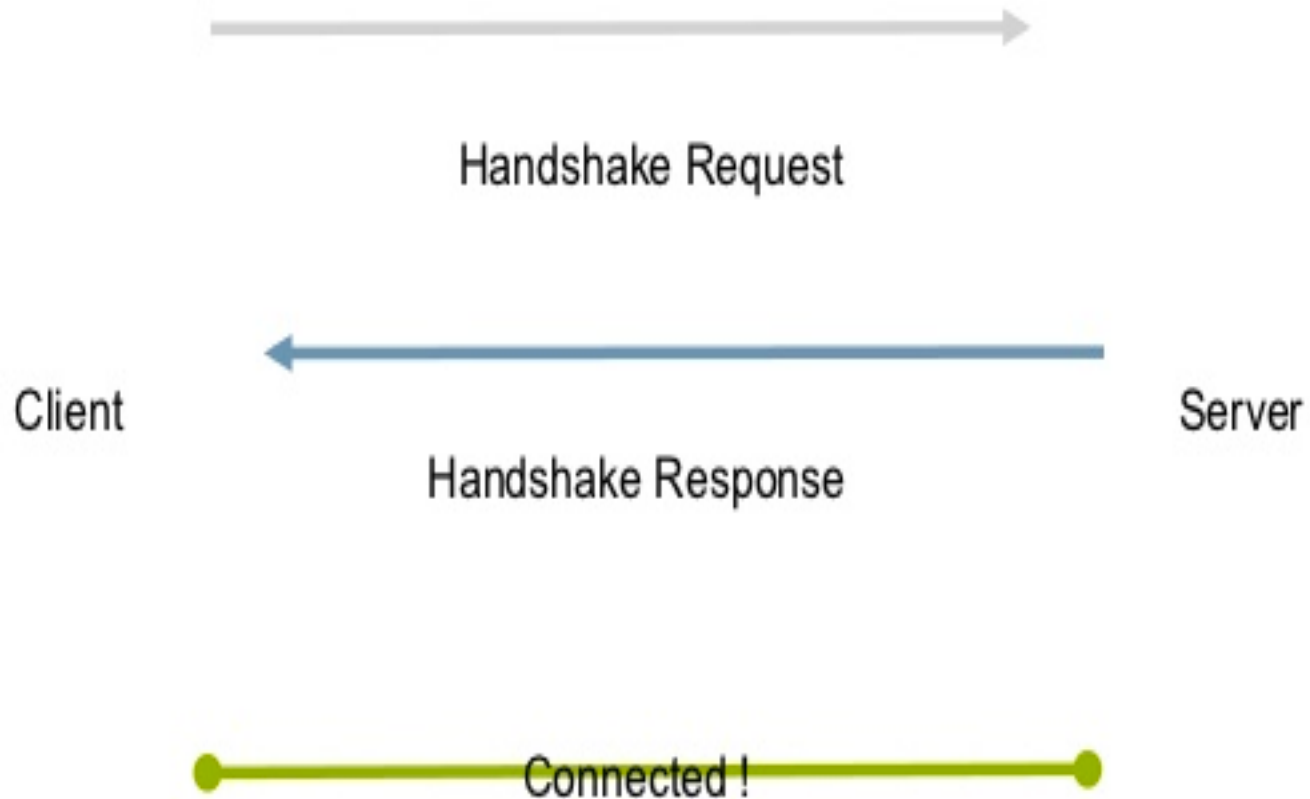
# WebSocket

TCP over HTTP, use the Upgrade: header

```
HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=  
Sec-WebSocket-Protocol: chat
```

# WebSocket

TCP over HTTP, use the Upgrade: header





# WebSocket

## Browser Support – caniuse.com

# Web Sockets - Candidate Recommendation

Bidirectional communication technology for web apps

Usage stats:

Global

Support: 67.46%

Partial support: 2.62%

Total: 70.08%

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
						3.2		2.2		
						4.0-4.1		2.3		
	8.0					4.2-4.3		3.0		
	9.0	23.0	29.0	5.1		5.0-5.1		4.0		
	10.0	24.0	30.0	6.0		6.0-6.1		4.1	7.0	
Current	11.0	25.0	31.0	7.0	17.0	7.0	5.0-7.0	4.2-4.3	10.0	10.0
Near future		26.0	32.0		18.0			4.4		
Farther future		27.0	33.0							

Notes

Known issues (1)

Resources (6)

Feedback

Edit on GitHub

Partial support refers to the websockets implementation using an older version of the protocol and/or the implementation being disabled by default (due to security issues with the older protocol).

# JavaScript/Browser Client

## JavaScript WebSocket Object

- Minimal Lifecycle
  - `new WebSocket(url)`
  - pass in some functions
    - `onopen`
    - `onmessage`
  - call `send()`
  - call `close()`
- Can connect to arbitrary servers, other than the page origin
- Server may enforce use of `Origin` header

# WebSocket

```
@Singleton
@ServerEndpoint("/tracking")
public class RealtimeCargoTrackingService {

    private static final Logger logger = Logger.getLogger(
        RealtimeCargoTrackingService.class.getName());

    private final Set<Session> sessions = new HashSet<>();

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

    @OnClose
    public void onClose(final Session session) {
        sessions.remove(session);
    }

    public void onCargoInspected(@Observes @CargoInspected Cargo cargo) {
        Writer writer = new StringWriter();
    }
}
```

# WebSocket and JSON

```
public void onCargoInspected(@Observes @CargoInspected Cargo cargo) {
    Writer writer = new StringWriter();

    try (JsonGenerator generator = Json.createGenerator(writer)) {
        generator
            .writeStartObject()
            .write("trackingId", cargo.getTrackingId().getIdString())
            .write("origin", cargo.getOrigin().getName())
            .write("destination", cargo.getRouteSpecification().getDestination().getName())
            .write("lastKnownLocation", cargo.getDelivery().getLastKnownLocation().getName())
            .write("transportStatus", cargo.getDelivery().getTransportStatus().toString())
            .writeEnd();
    }

    String jsonValue = writer.toString();

    for (Session session : sessions) {
        try {
            session.getBasicRemote().sendText(jsonValue);
        } catch (IOException ex) {
            logger.log(Level.WARNING, "Unable to publish WebSocket message", ex);
        }
    }
}
```

# WebSocket

```
(function() {  
  
    var ws = new WebSocket("ws://localhost:8080/cargo-tracker/tracking");  
    ws.onopen = function(event) {  
        onConnect(event);  
    };  
    ws.onmessage = function(event) {  
        onMessage(event);  
    };  
  
    onConnect = function(event) {  
    }  
  
    onMessage = function(event) {  
        populateListRouted(event);  
    }  
  
    populateListRouted = function(event) {  
  
        var jsonObject = JSON.parse(event.data);  
        var table = document.getElementById("listRoutedTab");  
  
        for (var count = 1, row; row = table.rows[count]; count++) {  
            if (row.id === jsonObject.trackingId) {  
                row.cells[1].innerHTML = jsonObject.origin;  
                row.cells[2].innerHTML = jsonObject.destination;  
                row.cells[3].innerHTML = jsonObject.lastKnownLocation;  
                row.cells[4].innerHTML = jsonObject.transportStatus;  
                break;  
            }  
        }  
    }  
})
```

# Java EE 7 in Action

## Components in this talk

- JSF 2.2
- Bean Validation 1.1
- JAX-RS 2.0
- WebSocket 1.0
- JSON-P 1.0

There is much more in CargoTracker, to learn more, visit

<http://cargotracker.java.net/>

# Java EE 8

- JSON-B
- JCache
- CDI 2
- More CDI/EJB alignment
- JMS.next()?
- Security
- Testability
- Cloud, PaaS, multitenancy/SaaS
- Action-oriented Web framework/HTML 5 alignment?
- NoSQL?
- Modularity?

<http://glassfish.org/survey>

# Try it Out!

# 4.0



<http://download.java.net/glassfish/4.0/release/glassfish-4.0.zip>

ORACLE



# Come to JavaLand in Brühl, Germany

Programm  
online

*JavaLand*

25.-26. März 2014  
im Phantasialand



<http://javaland.eu/>  
~ 50% English Talks

ORACLE

# Resources

- Java EE Tutorials
  - <http://docs.oracle.com/javaee/7/tutorial/doc/home.htm>
- Digging Deeper
  - <http://docs.oracle.com/javaee/7/firstcup/doc/home.htm>
  - <https://glassfish.java.net/hol/>
  - <https://java.net/projects/cargotracker/>
- Java EE 7 Transparent Expert Groups
  - <http://javaee-spec.java.net>
- Java EE 7 Reference Implementation
  - <http://glassfish.org>
- The Aquarium
  - <http://blogs.oracle.com/theaquarium>