Web Technologies in Java EE

JAX-RS 2.0, JSON-P, WebSocket, JSF 2.2

$ whoami

- Lukáš Fryč
  - Software Engineer, JBoss, Red Hat
    - AeroGear, RichFaces, Arquillian
  - Interests
    - Living and advocating Java EE (6 yrs)
    - HTML5, Web Components, AngularJS
    - Running, Hiking
    - Spending time with my family

Agenda

- **Client-Side** vs **Server-Side Web**
- **JAX-RS 2.0** RESTful Services
  - Origins + News in 2.0
- **JSON-P** Java API for JSON Processing
- Java API for **WebSocket**
- **JSF 2.2** **JavaServer Faces**
  - Origins + News in 2.2

Client-Side

vs

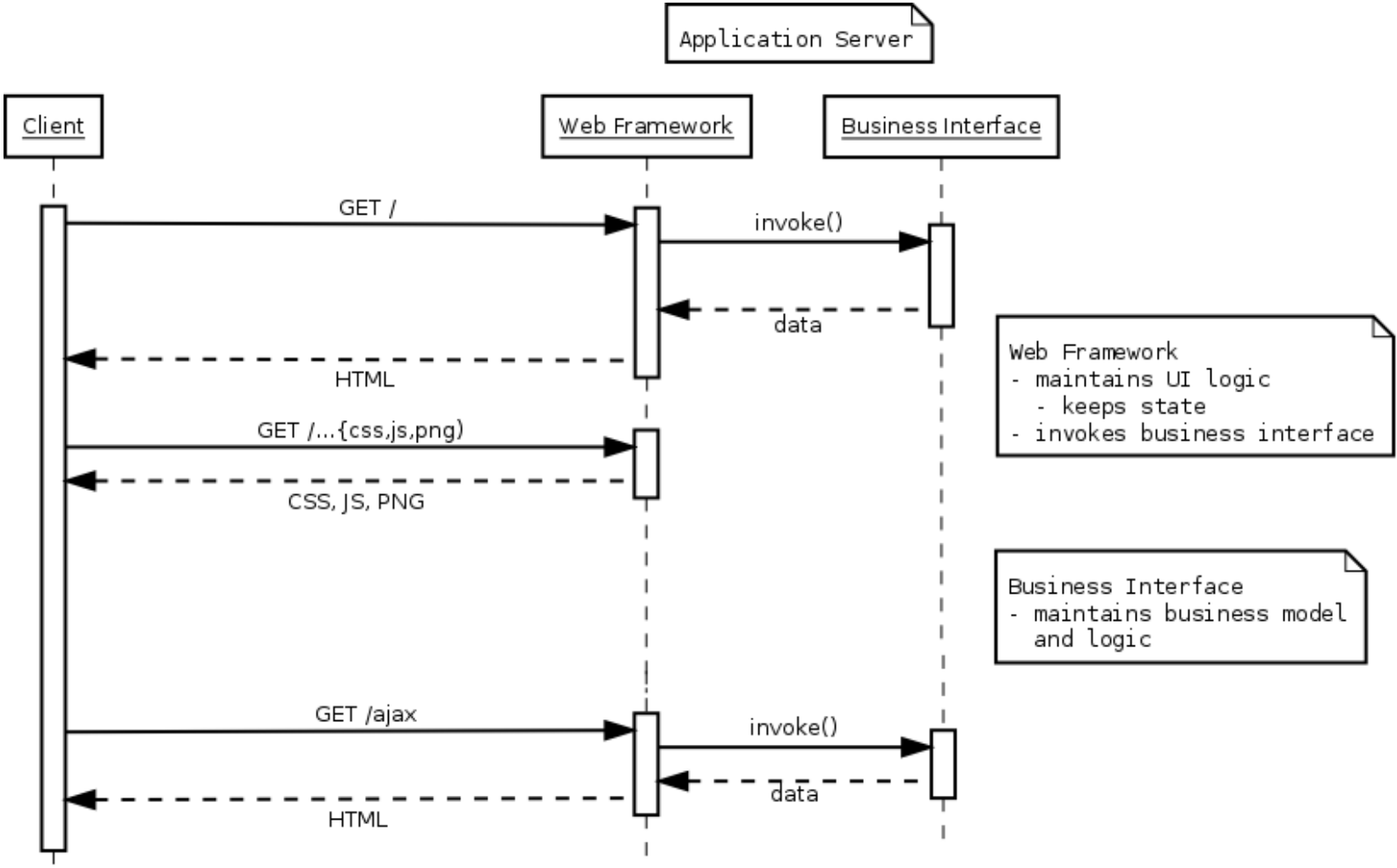
Server-Side

Web Architecture

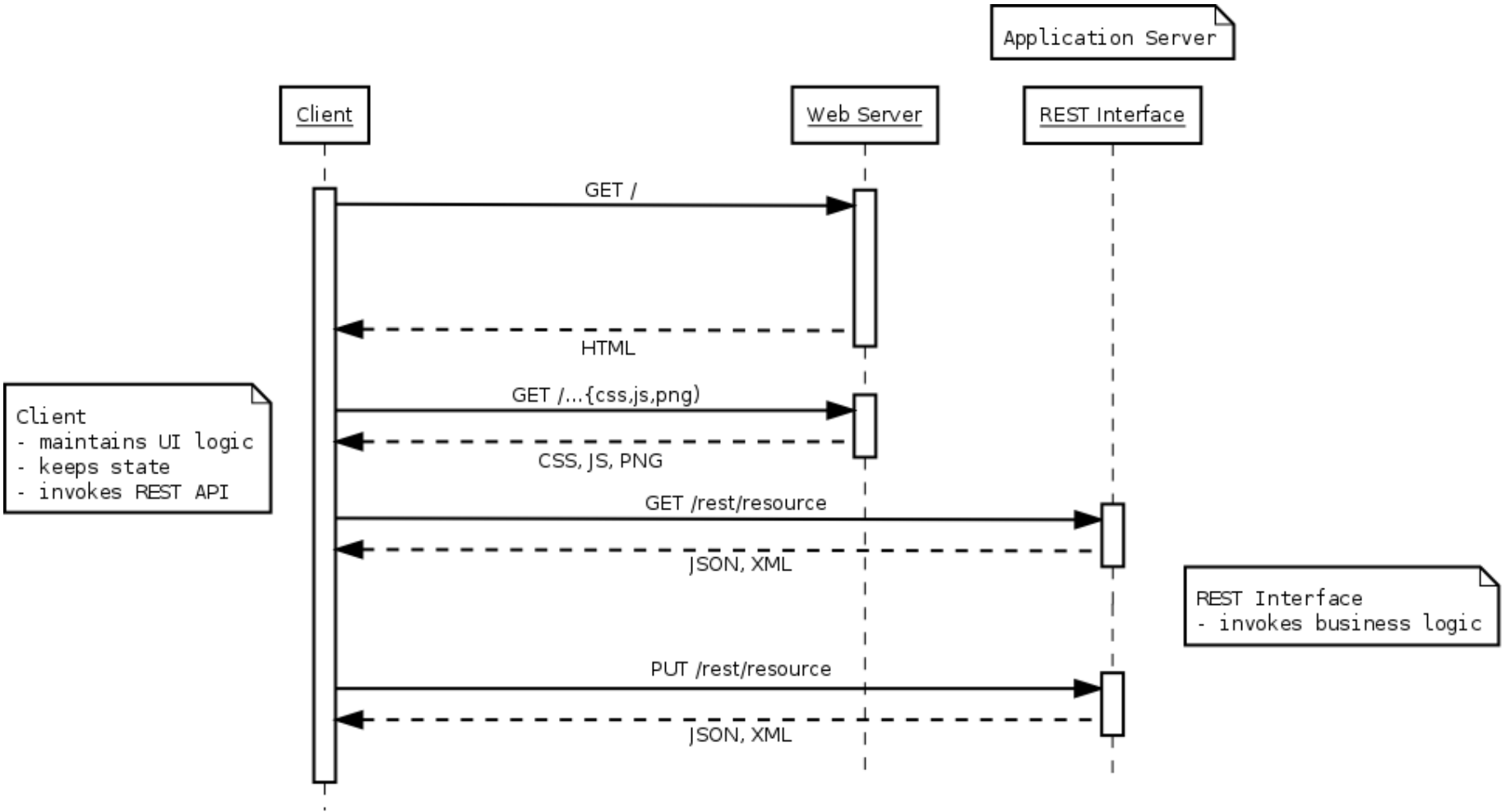
Client- vs. Server-Side Web

- Server-Side Web (Thin Client)
  - Well-established approach
  - 90's, 00's
- Client-Side Web (Thick Client)
  - Modern approach
  - Fully leverages enhancements in web standards and protocols
  - 10's

Server-Side


Application Server
Client
Web Framework
Business Interface
GET /
invoke()
data
HTML
GET /...{css,js,png)
CSS, JS, PNG
GET /ajax
invoke()
data
HTML
Web Framework
- maintains UI logic
- keeps state
- invokes business interface
Business Interface
- maintains business model
and logic

Client-Side


Application Server
Client
Web Server
REST Interface
GET /
HTML
GET /...{css,js,png)
CSS, JS, PNG
GET /rest/resource
JSON, XML
PUT /rest/resource
JSON, XML
Client
- maintains UI logic
- keeps state
- invokes REST API
REST Interface
- invokes business logic

Client-Side Web Approach

- Off-loading server
  - Scalability, Stateless
- Client-Side Frameworks
  - AngularJS, Ember, Backbone, Errai, Polymer, ...
- Standards improvements
  - HTML5 + Protocols
- REST interfaces
  - Data-oriented, presentation independent

Java API for RESTful Services

JAX-RS 2.0

JAX-RS Origins

- RESTful Principles
  - Assign everything an ID
  - Link things together
  - Use common methods (GET, POST, ...)
  - Stateless communication

JAX-RS 1.0 Goals

- POJO-Based API
- HTTP Centric
- Format Independence
  - plain/text
  - XML
  - HTML
  - JSON

JAX-RS API

- Resources
  - @Path
- HTTP methods
  - @GET / @POST / @PUT / @DELETE / ...
- Parameters
  - @PathParam / @QueryParam / ...
- Media-Type
  - @Consumes / @Produces

Demo

JAX-RS Endpoint

http://javaee-samples.github.io/

HTTP Method Purpose

Method	Meaning
@GET	Read, possibly cached
@POST	Modify or create **without** a known ID (modify/update)
@PUT	Modify or create **with** a known ID (create/modify)
@DELETE	Remove
@HEAD	GET with no response
@OPTIONS	Supported methods

http://stackoverflow.com/questions/630453/put-vs-post-in-rest

Parameter Injection

Annotation	Example
@PathParam(“id”)	@Path(“/consumer/{id}”)
@QueryParam(“query”)	`GET /consumer/search?query=???`
@CookieParam(“username”)	`Cookie: ...`
@HeaderParam(“Authorization”)	`Header: Authorization: ...`
@FormParam(“inputName”)	@Consumes("multipart/form-data")
@MatrixParam	GET /consumer/search;query=???

New in JAX-RS 2.0

- New Features
  - Client API
  - Filters and Interceptors
  - Asynchronous API
  - Hypermedia
- Improvements
  - Content-Type Negotiation
  - Validation Alignments

Client API

- HTTP client libraries too low-level
- Need for standardization

Demo

JAX-RS – Client API

Filters and Interceptors

- Customize JAX-RS
  - via well-defined extension points
- Use cases:
  - Logging
  - Compression
  - Security
- Shared between server & client

Filters

- Non-wrapping extension points
  - Pre: `RequestFilter`
  - Post: `ResponseFilter`
- Part of a filter chain
- Do not call the next filter directly
- Each filter decides to proceed or break chain
  - FilterAction.NEXT, FilterAction.STOP

Interceptors

- Wrapping extensions points
  - ReadFrom: `ReaderInterceptor`
  - WriteTo: `WriterInterceptor`
- Part of an interceptor chain
- Call the next handler directly
- Each handler decides to proceed or break chain
  - By calling `ctx.proceed();`

Asynchronous

- Let “borrowed” threads run free!
  - Suspend and resume connections
    - Suspend while waiting for an event (`@Suspended AsyncResponse`)
    - Resume when event arrives
- Leverages Servlet 3.x async support
  - HttpServletRequest.upgrade(ProtocolHandler)
- Client API support
  - Future<T>, InvocationCallback<T>

Demo

JAX-RS – Asynchronous

Validation

- Constraint annotations
  - Fields and properties
  - Parameters
    - Including request entity
  - Methods
    - Response entities
  - Resource classes
- Custom constraints

Demo

JAX-RS – Bean Validation

Hypermedia

- Link types
  - Structural links
    - <customer>http://.../customers/1234</customer>
  - Transitional links
    - Links: <http://.../cancel>; rel=cancel

Java API for JSON Processing

JSON-P

Motivation: JSON

- JavaScript Object Notation
  - The format of the Web
    - Comes from JavaScript object syntax
    - Human-Readable
  - Language independent
    - Standard parsers in many languages
  - Key-value Pair Format

```
{ “firstName”: “John”, “lastName”: “Smith” }
```

Motivation: Java API for JSON

- Lot of vendor-dependent APIs
  - Need for standardization
- Standard API for JSON processing
  - parse, generate, transform
- Binding? Querying?

JSON-P APIs

- Streaming API
  - Similar to XML DOM
- Object Model API
  - Similar to StAX

JSON-P APIs

- Streaming API
  - `JsonParser, JsonGenerator`
- Object Model API
  - `JsonObject, JsonArray`
  - `JsonBuilder`
  - `JsonReader, JsonWriter`

Object Model - `JsonReader`

- Reads `JsonObject` and `JsonArray`
  - I/O Reader, InputStream
- Uses pluggable JsonParser

```
// Reads a JSON Object
try (JsonWriter reader = Json.createReader(io)) {
    JsonObject obj = reader.readObject();
}
```

Object Model - `JsonWriter`

- Writes `JsonObject` and `JsonArray` to output source
  - I/O Writer, OutputStream
- Uses pluggable JsonGenerator
- Allows pretty-printing and other features

```
// Writes a JSON Object
JsonObject obj;

try (JsonWriter writer = Json.createWriter(io)) {
    Writer.writeObject(obj);
}
```

Object Model – `Json*Builder`

- Chained API
  - For bulding `JsonObject` and `JsonArray`
    - Can consume `JsonObject` and `JsonArray`
  - Type-safe (no mixing arrays/objects)
- `Json.createObjectBuilder()`
- `Json.createArrayBuilder()`

Demo

JSON Object Model API

Streaming - `JsonParser`

- Parses JSON in a streaming way from input sources
  - Similar to StAX's XMLStreamReader
  - a pull parser
- `Json.createParser()`

Streaming - `JsonGenerator`

- Generates JSON in a streaming way to output sources
  - Similar to StAX's XMLStreamWriter
- `Json.createGenerator()`

Demo

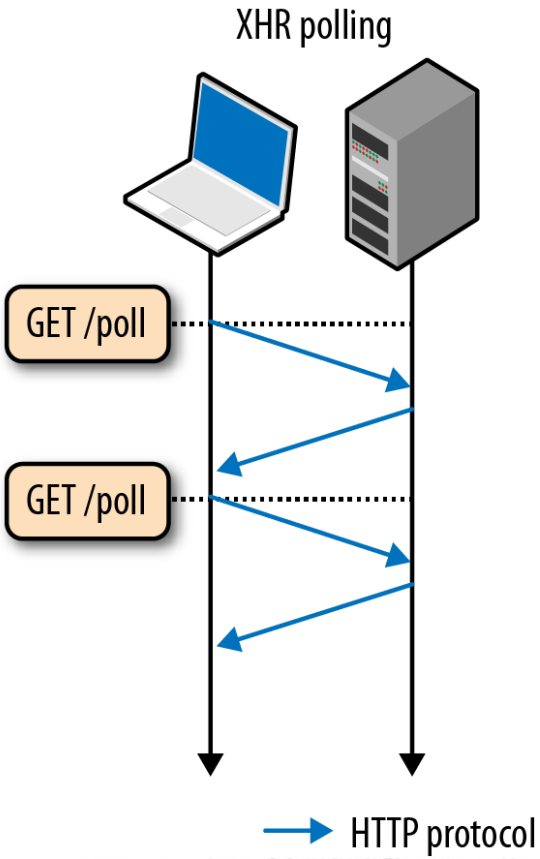
JSON Streaming API

Java API for WebSocket

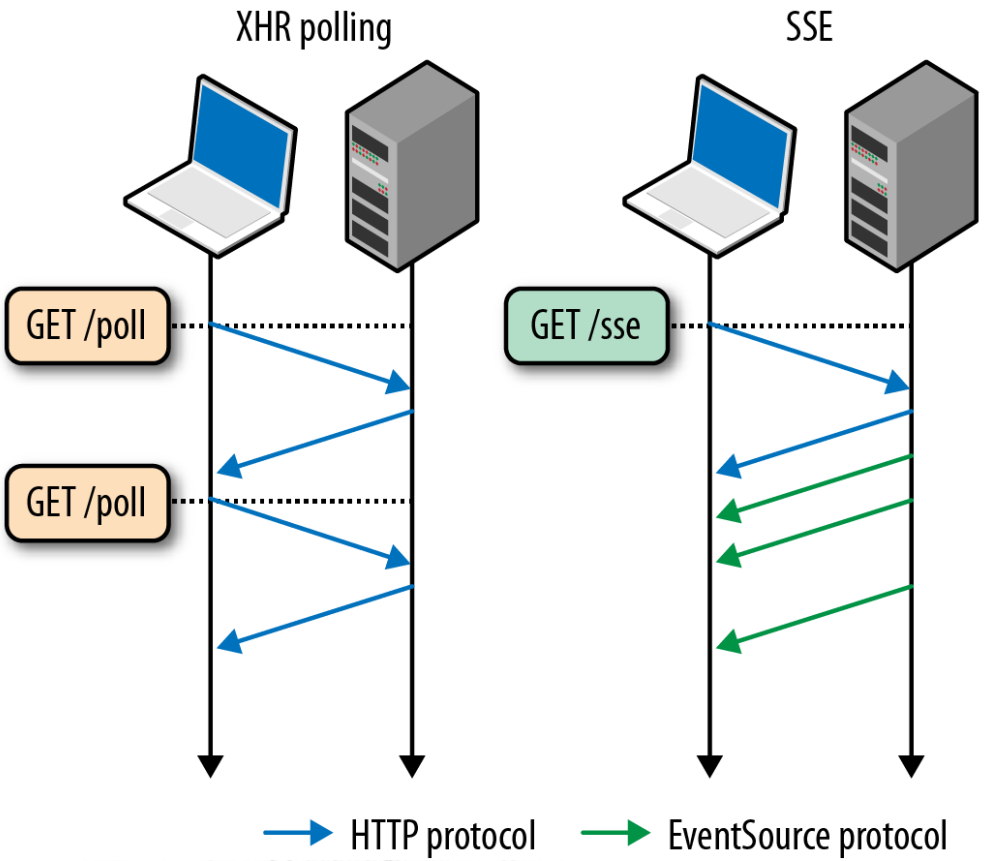
Motivation

- HTTP is half-duplex
- HTTP is inefficient
- HTTP hacked to achieve Push
  - HTTP Polling
  - HTTP Long-Polling (Comet)
  - Server Sent Events

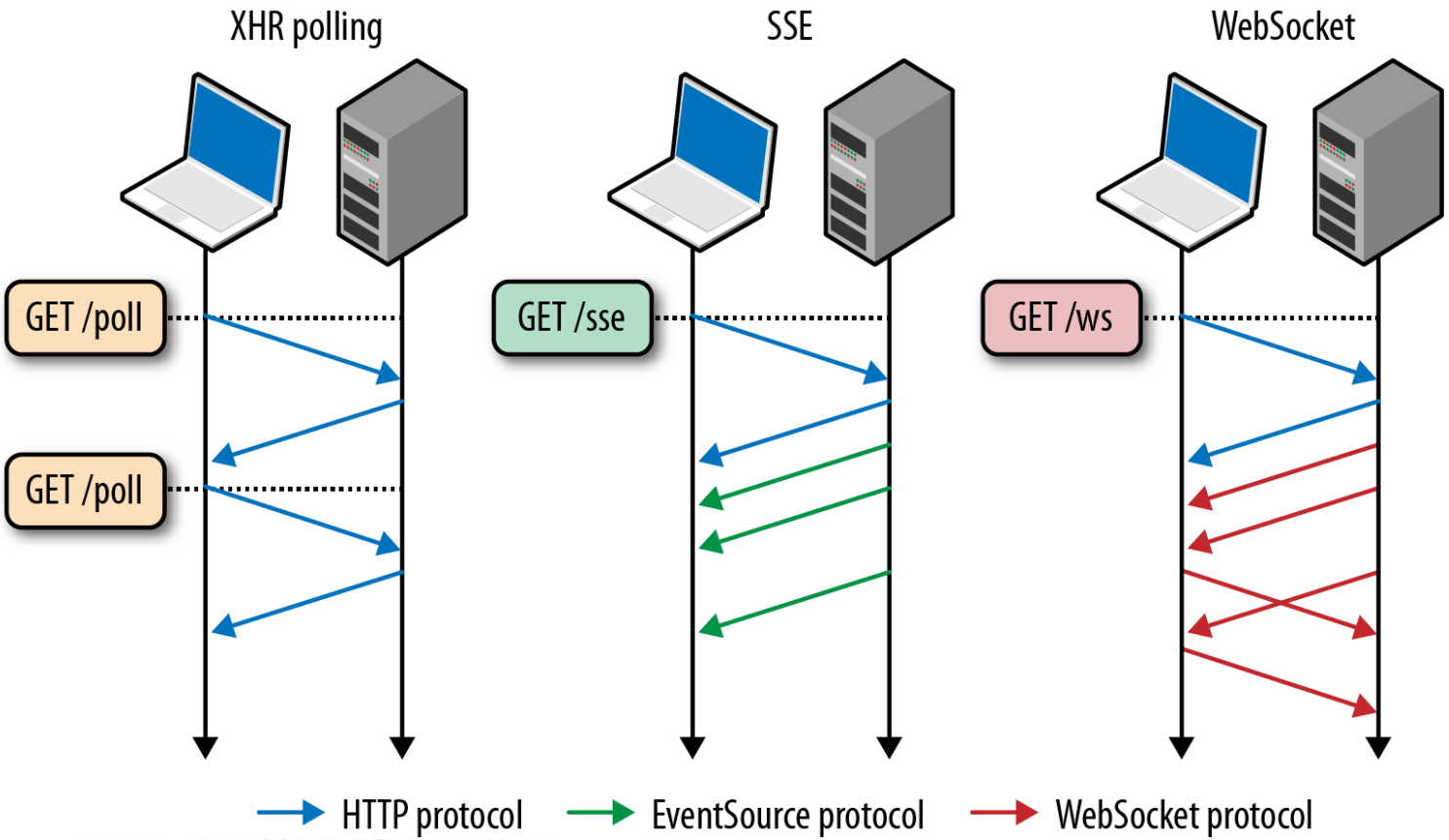
Server Push - Polling



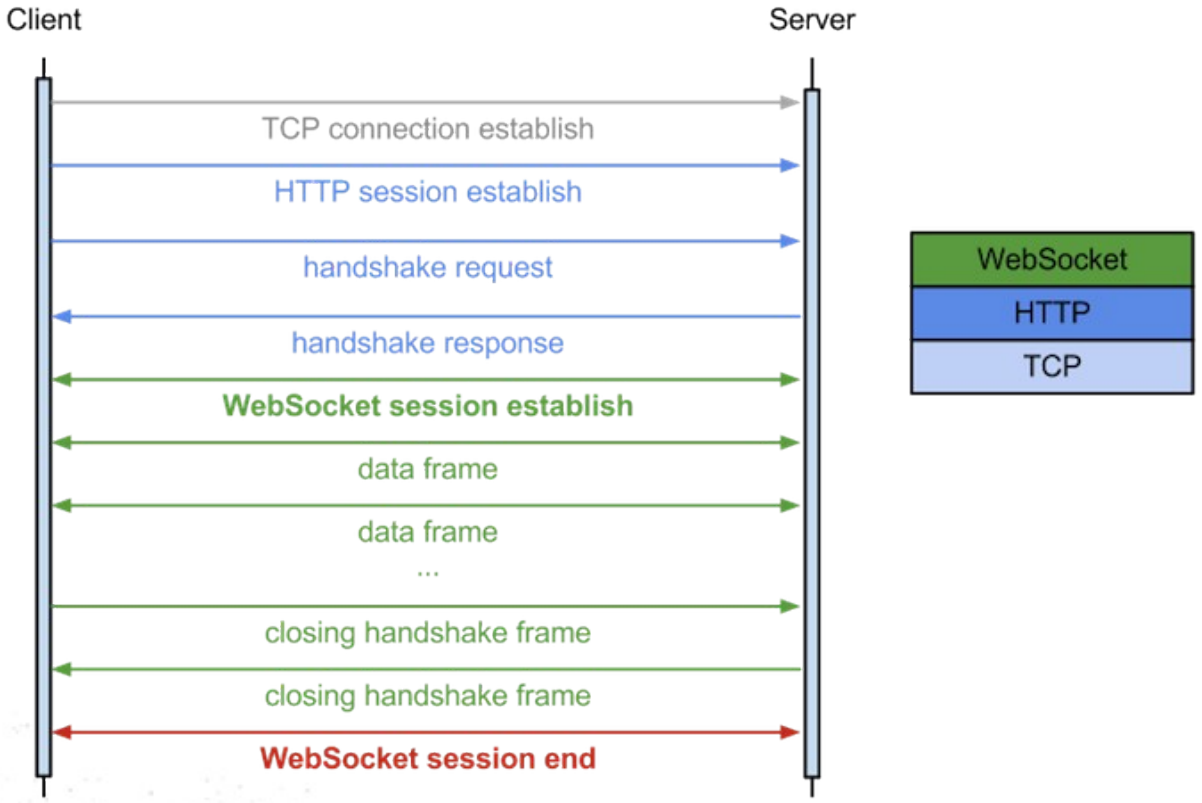
Server Push – SSE



WebSocket


XHR polling
SSE
WebSocket
GET /poll
GET /poll
GET /sse
GET /ws
HTTP protocol
EventSource protocol
WebSocket protocol

Handshake


Client
Server
TCP connection establish
HTTP session establish
handshake request
handshake response
WebSocket session establish
data frame
data frame
...
closing handshake frame
closing handshake frame
WebSocket session end
WebSocket
HTTP
TCP

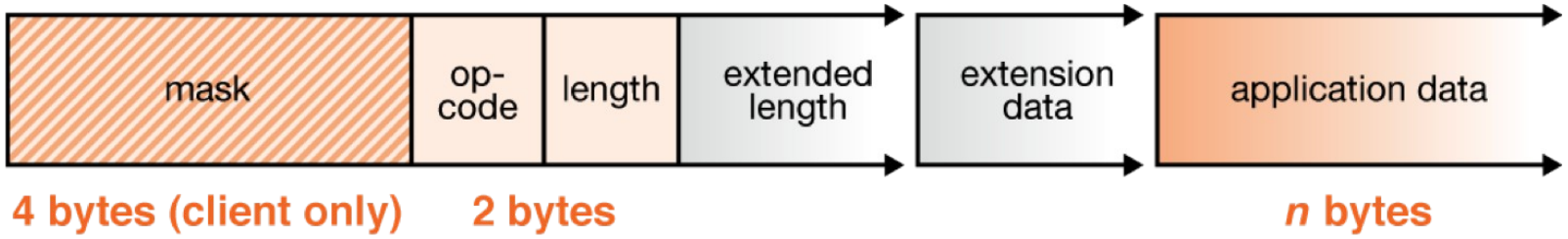
HTTP Upgrade - Request

```
GET /socket/updates HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: www.sample.org
```

HTTP Upgrade - Response

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
```

WebSocket Frames



WebSocket

- Full duplex & efficient communication
- A component of HTML5
  - JavaScript API under W3C
  - Protocol under IETF
- Wide support for browsers
  - http://caniuse.com/#feat=websockets

WebSocket: Limitations

- Use of existing infrastructure
  - Proxies doesn't have to handle connection upgrade
- Fallback mechanisms
  - Atmosphere

WebSocket: Trade-offs

- WebSocket
  - Low efforts to maintain TCP connection
  - Limited by number of available ports
  - **Highly interactive applications**

- HTTP
  - Resource-consuming protocol
  - **Fairly interactive applications**

WebSocket: Use Cases

- Realtime, truly low latency
  - Chat applications
  - Live sports ticker
  - Realtime updating social streams
  - Multiplayer online games
- Requires architecture shift to
  - Non-blocking IO
  - Event queues

Java API for WebSocket

- Programmatic
- Annotation-based
  - our focus

WebSocket Annotations

- `@ServerEndpoint`
  - `@OnOpen`
  - `@OnMessage`
  - `@OnClose`

Demo

WebSocket - Whiteboard

Method Parameters

- `Session`
- Implicitly supported types
  - `String, byte[]`
  - `JsonArray, JsonObject`
- More types supported by Encoders

Integration to Java EE 7

- Relation to Servlet 3.1
  - `HttpServletRequest.upgrade(ProtocolHandler)`
- Dependency Injection
  - CDI beans
  - EJB beans
- Security
  - `ws://... vs. wss://...`
  - web.xml: `<security-constraint>`

JavaServer Faces

JSF 2.2

JSF Origins

- MVC Framework
  - Component-oriented
  - Server-Side
  - Extensible
- Component Libraries

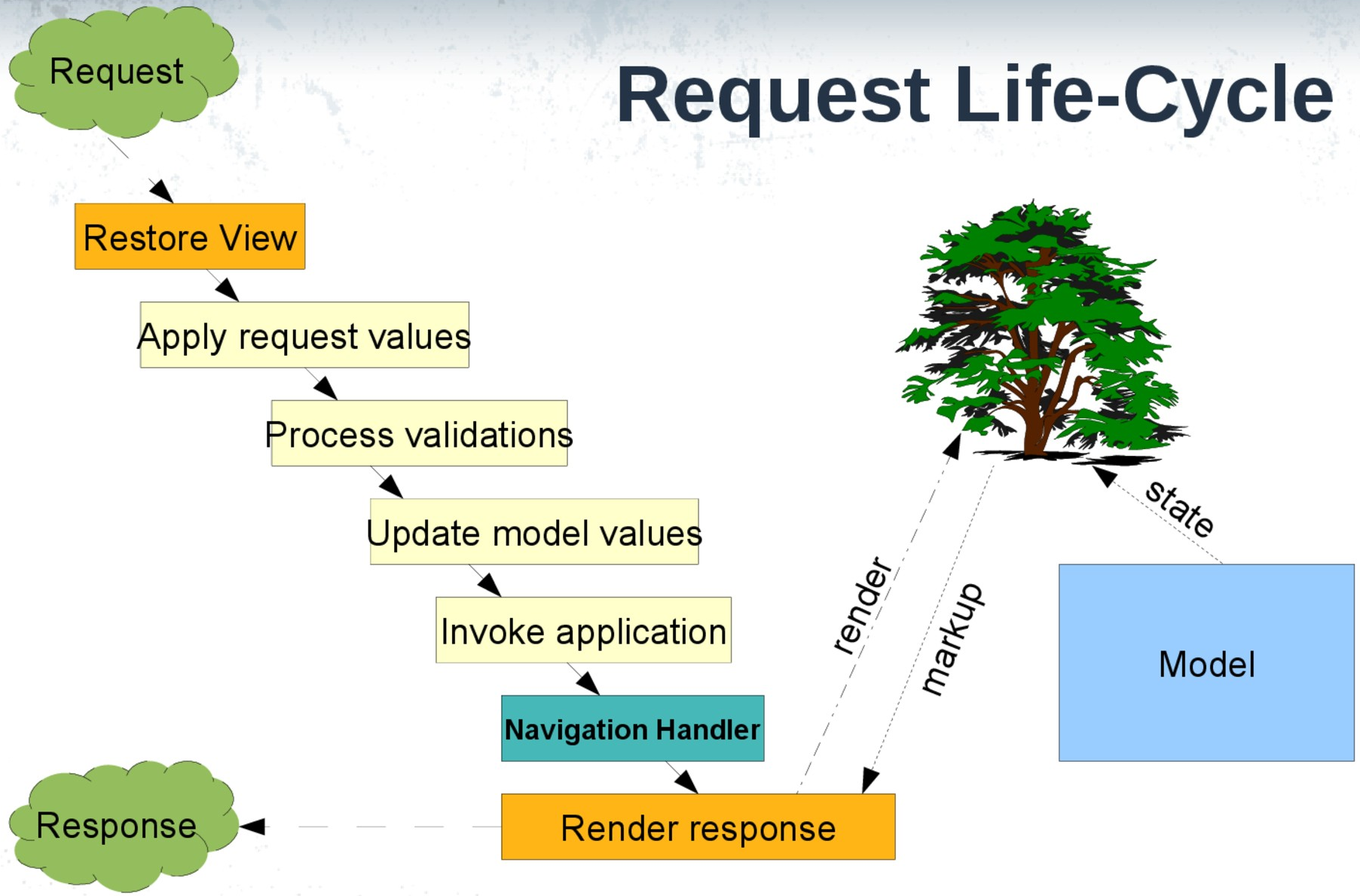
Component Libraries

- Rich components
  - RichFaces
  - PrimeFaces
  - ICEFaces
- Functionality
  - PrettyFaces
  - OmniFaces

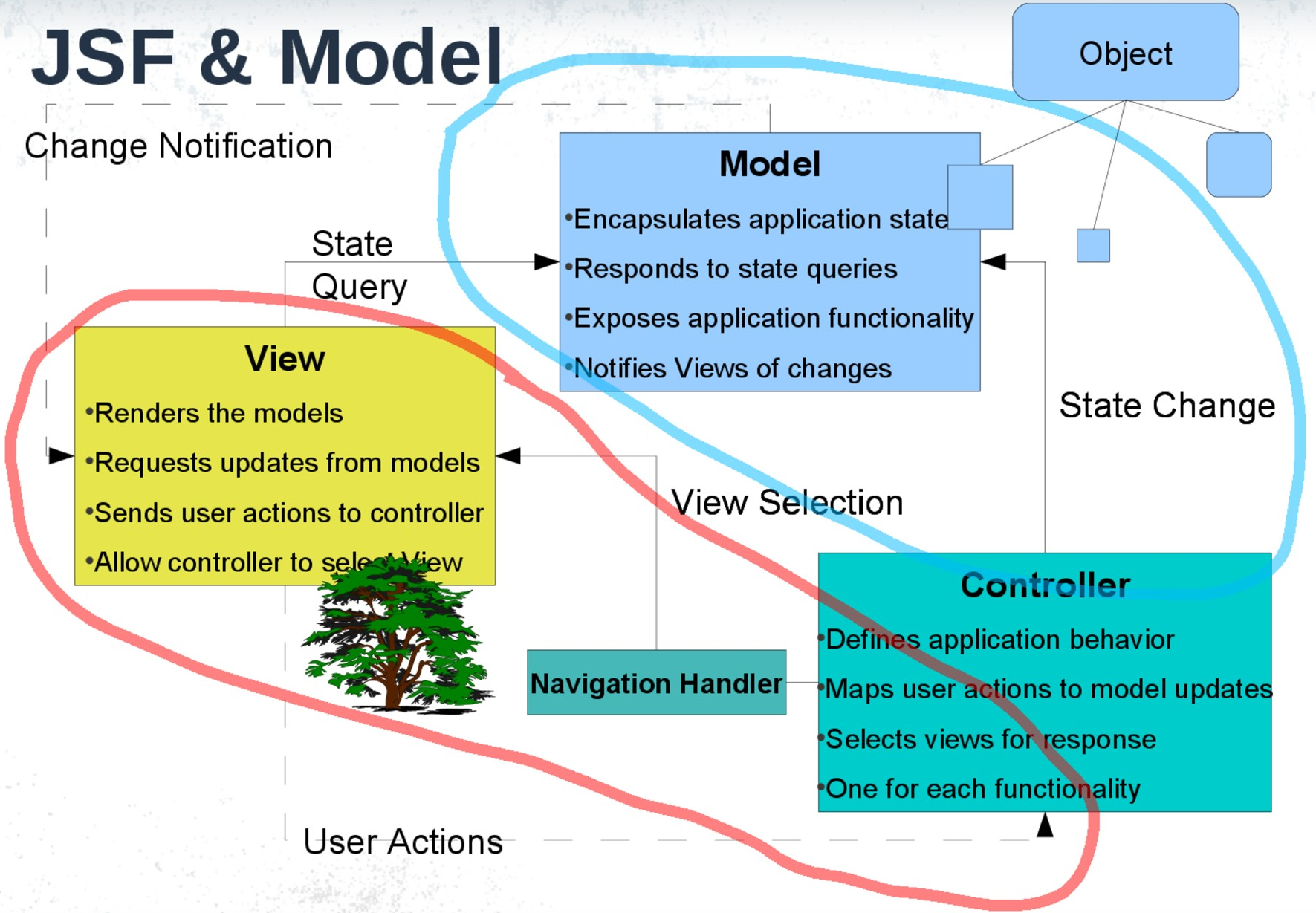
Component Tree

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets">

    <h:head>
        ...
    </h:head>

    <h:body>
        <h:form ...>
            <h:selectOneMenu ... >
                <f:selectItems ... />
            </h:selectOneMenu>
        </h:form>

        <ui:repeat ...>
            <h:outputText ... />
        </ui:repeat>
    </h:body>

</html>
```

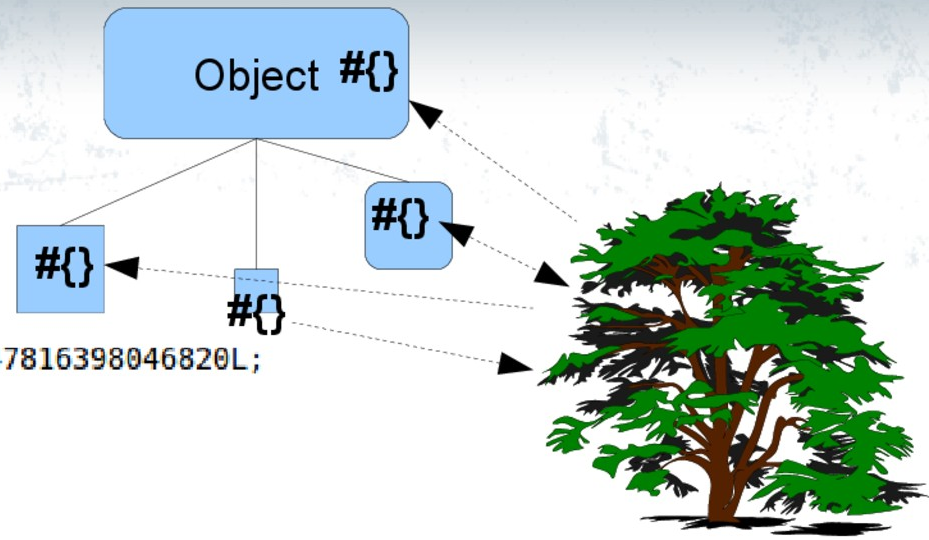
Request Life-Cycle



JSF & Model


Object
Change Notification
Model
•Encapsulates application state
•Responds to state queries
•Exposes application functionality
•Notifies Views of changes
State
Query
State Change
View
•Renders the models
•Requests updates from models
•Sends user actions to controller
•Allow controller to select View
View Selection
Controller
•Defines application behavior
•Maps user actions to model updates
•Selects views for response
•One for each functionality
Navigation Handler
User Actions

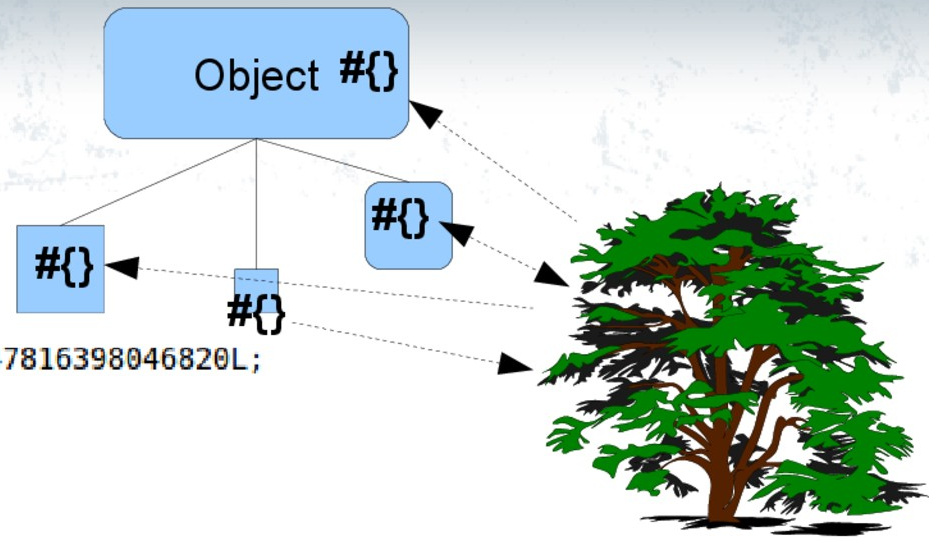
Model & View




```java
@Named
@SessionScoped
public class User implements Serializable {

    private static final long serialVersionUID = -8082147816398046820L;

    private String name = null;

    public String getName() {
        return name;
    }

    public boolean isLogged() {
        return name != null;
    }

    public void login(String name) {
        System.out.println("login: " + name);
        this.name = name;
    }

    public void logout() {
        this.name = null;
    }
}
```

```xml
<h:form>
    <h:panelGrid columns="1" style="width: 95%">
        <h:outputText value="#{user.name}" rendered="#{user.logged}
        <h:inputText id="username" value="#{username}"
            rendered="#{not user.logged}"
            validatorMessage="Uzivatelske jmeno musi byt neprazdne
            <f:validateRequired />
            <f:validateRegex pattern="[a-z]+" />
        </h:inputText>
    </h:panelGrid>

    <h:commandButton id="loginButton" value="Prihlasit se"
        action="#{user.login(username)}" rendered="#{not user.logge

    <h:commandButton id="logoutButton" value="Odhlasit se"
        action="#{user.logout}" rendered="#{user.logged}" />
```

Model & View

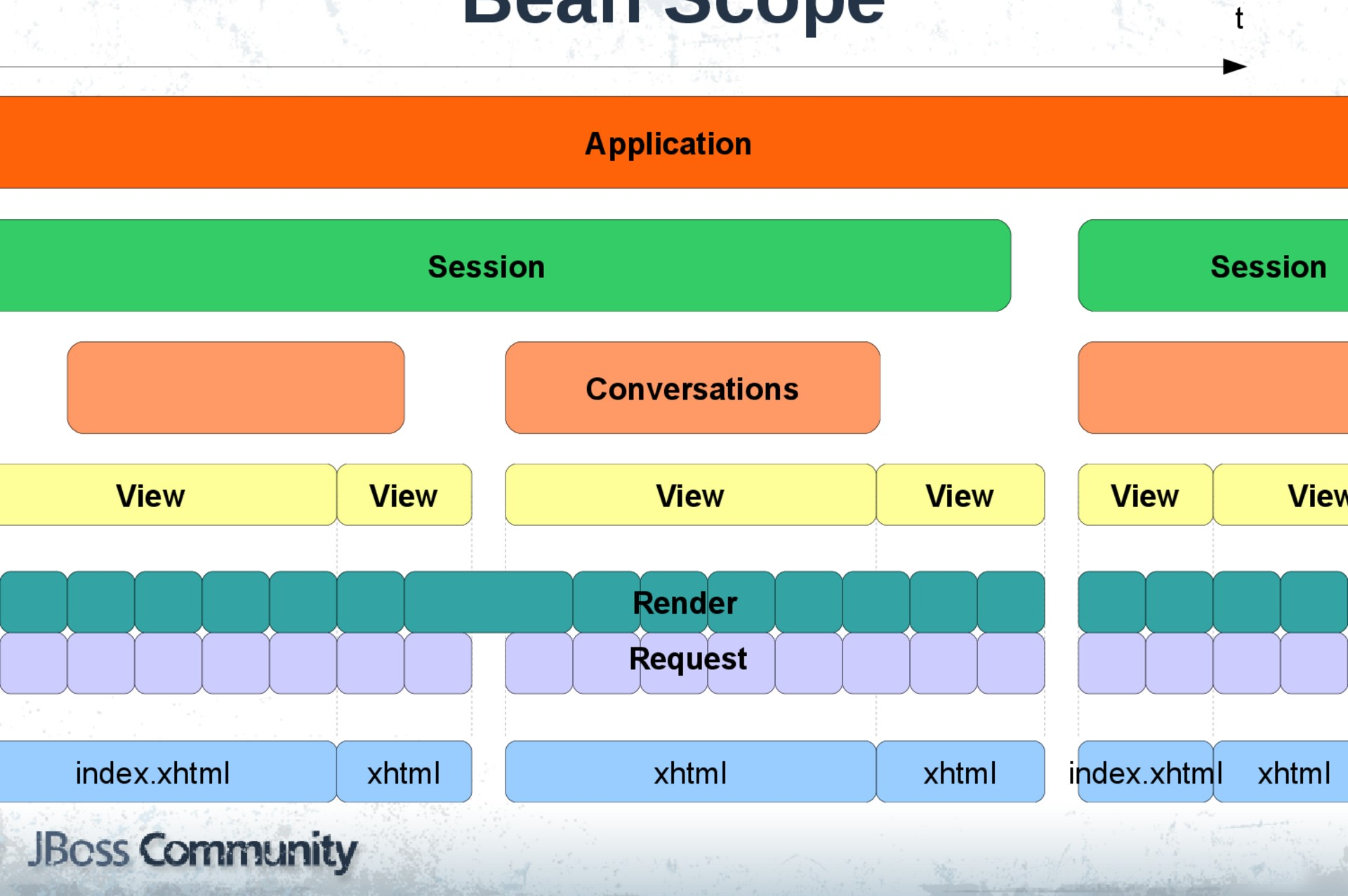



```
@Named
@SessionScoped
public class User implements Serializable {

    private static final long serialVersionUID = -8082147816398046820L;

    private String name = null;

    public String getName() {
        return name;
    }

    public boolean isLogged() {
        return name != null;
    }

    public void login(String name) {
        System.out.println("login: " + name);
        this.name = name;
    }

    public void logout() {
        this.name = null;
    }
}
```

```
<h:form>
    <h:panelGrid columns="1" style="width: 95%">
        <h:outputText value="#{user.name}" rendered="#{user.logged}
        <h:inputText id="username" value="#{username}"
            rendered="#{not user.logged}"
            validatorMessage="Uzivatelske jmeno musi byt neprazdne
            <f:validateRequired />
            <f:validateRegex pattern="[a-z]+" />
        </h:inputText>
    </h:panelGrid>

    <h:commandButton id="loginButton" value="Prihlasit se"
        action="#{user.login(username)}" rendered="#{not user.logge

    <h:commandButton id="logoutButton" value="Odhlasit se"
        action="#{user.logout}" rendered="#{user.logged}" />
```

@Named ↔ Expression Language

Bean Scope


t
Application
Session
Session
Conversations
View
View
View
View
View
View
Render
Request
index.xhtml
xhtml
xhtml
xhtml
index.xhtml
xhtml

JSF 1.0 Goals

- What it adds over other frameworks?
  - Maintainability
  - Tooling Support
  - I18N

JSF 1.0 Goals

- What it adds over other frameworks?
  - Maintainability
  - Tooling Support
  - I18N

JSF 1.0

- Components
- Renderers
- Managed beans (CDI)
- Converters / Validators
- Navigation
- Request lifecycle
- Error handling

JSF 2.0

- Facelets (as default VDL)
- Composite Components
- AJAX
- Resource Libraries
- Behaviors
- GET support - <f:viewParam>
- Project Stage

JSF 2.2

- Big Tickets
  - Performance, Markup, Multi-tenancy
- Small Features
  - <f:viewAction>
  - CSRF protection
  - ClientWindow
  - Favours CDI
- Many smaller improvements

JSF 2.2

- Stateless Views
  - (Performance)
- HTML5 Friendly Markup
  - (Modern Markups)
- Flows, Resource Library Contracts
  - (Multi-Tenancy)

Stateless JSF

- What is state?
  - UI Components, Model, Persistence
  - Developer's concern
- Implementation
  - <f:view transient=”true”>

HTML5 Friendly Markup

JSF Components

```
<html>
<my:colorPicker value="#{colorBean.color2}" />
<my:calendar value="#{calendarBean.date1}" />
</html>
```

HTML5 Markup

```
<html>
<input type="color" j:value="#{colorBean.color2}" />
<input type="date" j:value="#{calendarBean.date1}" />
</html>
```

Multitenant Capability

- JSF app as a collection of modules
  - Faces Flows
    - Modularize ***behavior***
    - Builds on navigation
  - Resource Library Contracts
    - Modularize ***appearance***
    - Builds on Facelets
- Well defined contract for each

Demo

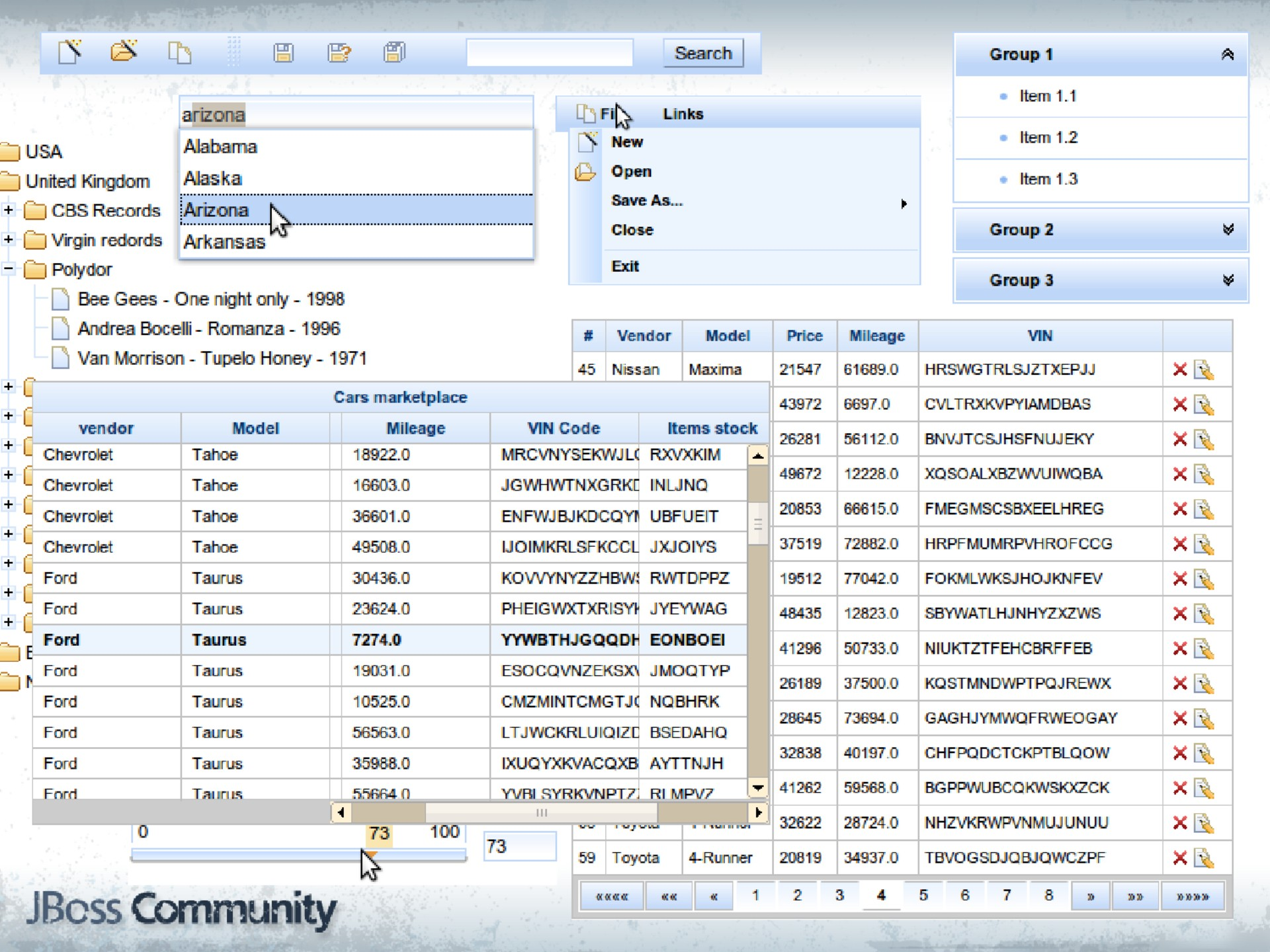
JSF – Declarative Flows,
Resource Contracts

JSF Component Libraries

RichFaces

RichFaces 4.3

- Rich & interactive components
- Client-Side Validation
- Push


Search
arizona
Alabama
Alaska
Arizona
Arkansas
Links
New
Open
Save As...
Close
Exit
Group 1
Item 1.1
Item 1.2
Item 1.3
Group 2
Group 3
USA
United Kingdom
CBS Records
Virgin redords
Polydor
Bee Gees - One night only - 1998
Andrea Bocelli - Romanza - 1996
Van Morrison - Tupelo Honey - 1971
Cars marketplace
vendor
Model
Mileage
VIN Code
Items stock
Chevrolet Tahoe 18922.0 MRCVNYSEKWJLC RXVXKIM
Chevrolet Tahoe 16603.0 JGWHWTNXGRKD INLJNQ
Chevrolet Tahoe 36601.0 ENFWJBJKDCQYN UBFUEIT
Chevrolet Tahoe 49508.0 IJOIMKRLSFKCCL JXJOIYS
Ford Taurus 30436.0 KOVVYNYZZHBWS RWTDPPZ
Ford Taurus 23624.0 PHEIGWXTXRISYH JYEYWAG
Ford Taurus 7274.0 YYWBTHJGQQDH EONBOEI
Ford Taurus 19031.0 ESOCQVNZEKSXV JMOQTYP
Ford Taurus 10525.0 CMZMINTCMGTJC NQBHRK
Ford Taurus 56563.0 LTJWCKRLUIQIZD BSEDAHQ
Ford Taurus 35988.0 IXUQYXKVACQXB AYTTNJH
Ford Taurus 55664.0 YVBLSYRKVNPTZ RLMPVZ
0
73
100
73
Vendor Model Price Mileage VIN
45 Nissan Maxima 21547 61689.0 HRSWGTRLSJZTXEPJJ
43972 6697.0 CVLTRXKVPYIAMDBAS
26281 56112.0 BNVJTCSJHSFNUJEKY
49672 12228.0 XQSOALXBZWVUIWQBA
20853 66615.0 FMEGMSCSBXEELHREG
37519 72882.0 HRPFMUMRPVHROFCCG
19512 77042.0 FOKMLWKSJHOJKNFEV
48435 12823.0 SBYWATLHJNHYZXZWS
41296 50733.0 NIUKTZTFEHCBRFFEB
26189 37500.0 KQSTMNDWPTPQJREWX
28645 73694.0 GAGHJYMWQFRWEOGAY
32838 40197.0 CHFPQDCTCKPTBLQOW
41262 59568.0 BGPPWUBCQKWSKXZCK
32622 28724.0 NHZVKRWPVNMUJUNUU
59 Toyota 4-Runner 20819 34937.0 TBVOGSDJQBJQWCZPF
«««« «« « 1 2 3 4 5 6 7 8 » »» »»»»
JBoss Community

Push

```
@Qualifier
@JmsDestination(jndiName = "jms/chat")
public @interface ChatTopic {}
```

```
@Inject @ChatTopic
TopicPublisher chat;

void sendMessage() {
    chat.publish(new Message(...));
}
```

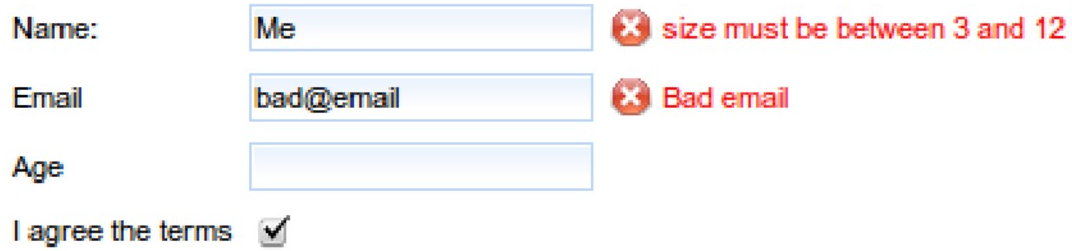
```
<a4j:push address="chat" render="chatPanel" />
```

Bean Validation Integration (JSR-303)

```
@Entity
public class Person {

    @Size(min=3, max=12)
    private String name = null;
    @Email
    private String email = null;
    @Min(value = 18)
    @Max(value = 99)
    private Integer age;

    ...

}
```

Client-Side Bean Validation

```
<h:outputText value="Name:" />
<h:inputText value="#{person.name}" id="name">
  ◆ <rich:validator />
</h:inputText>
<rich:message for="name" />
```



RichFaces 5.0

- Sneak peek
  - Visual Improvements
    - 3rd-party widget libraries
    - Skinnability
      - LESS
      - Bootstrap 3
  - New Components
    - Charts

That's it

Summary

- JSF
  - Fully-featured web framework
- JAX-RS
  - RESTful services
- WebSocket
  - Efficient bi-directional communication
- JSON-P
  - Standardization of JSON processing

What's next on EE Web?

- Improving integration
  - JSON-B (JSON Binding)
- Simplicity of Use
- Aligning with upcoming Web standards
- Java EE standardizes, not innovates
  - Innovation is driven by community

Disclaimer: This is my personal view

That's it

Summary

- JAX-RS
  - RESTful services
- JSON-P
  - Standardization of JSON processing
- WebSocket
  - Efficient bi-directional communication

What's next on EE Web?

- Improving integration
  - JSON-B (JSON Binding)
  - WebSocket – automatic en-/decoding
  - WebSocket + JMS
- Simplicity of Use
- Aligning with upcoming Web standards
- Java EE standardizes, not innovates
  - Innovation is driven by community

Disclaimer: This is my personal view

Thank you

Links

- http://javaee-samples.github.io/