Web Technologies in Java EE

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

$ whoami

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

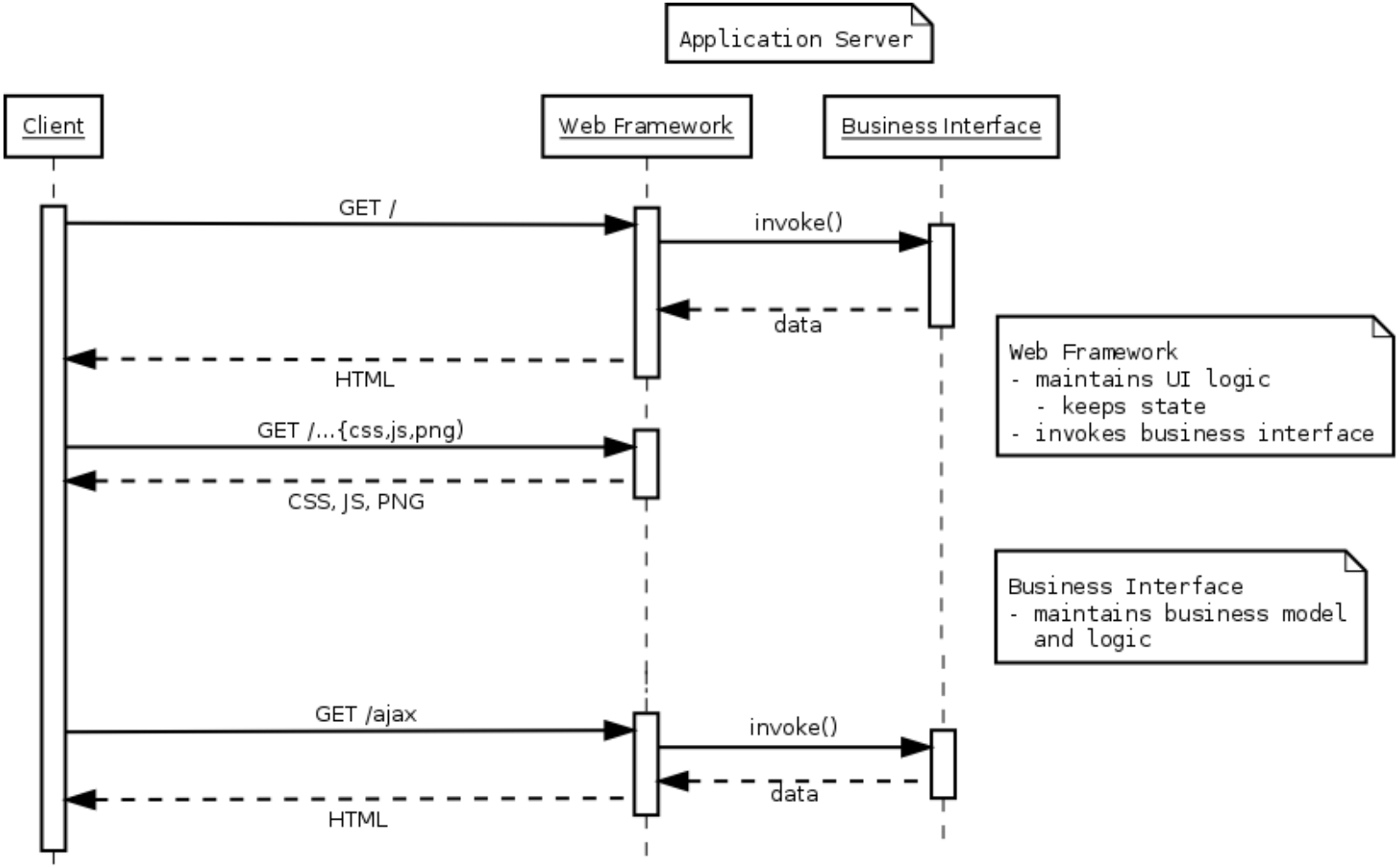
Server-Side

Web Architecture

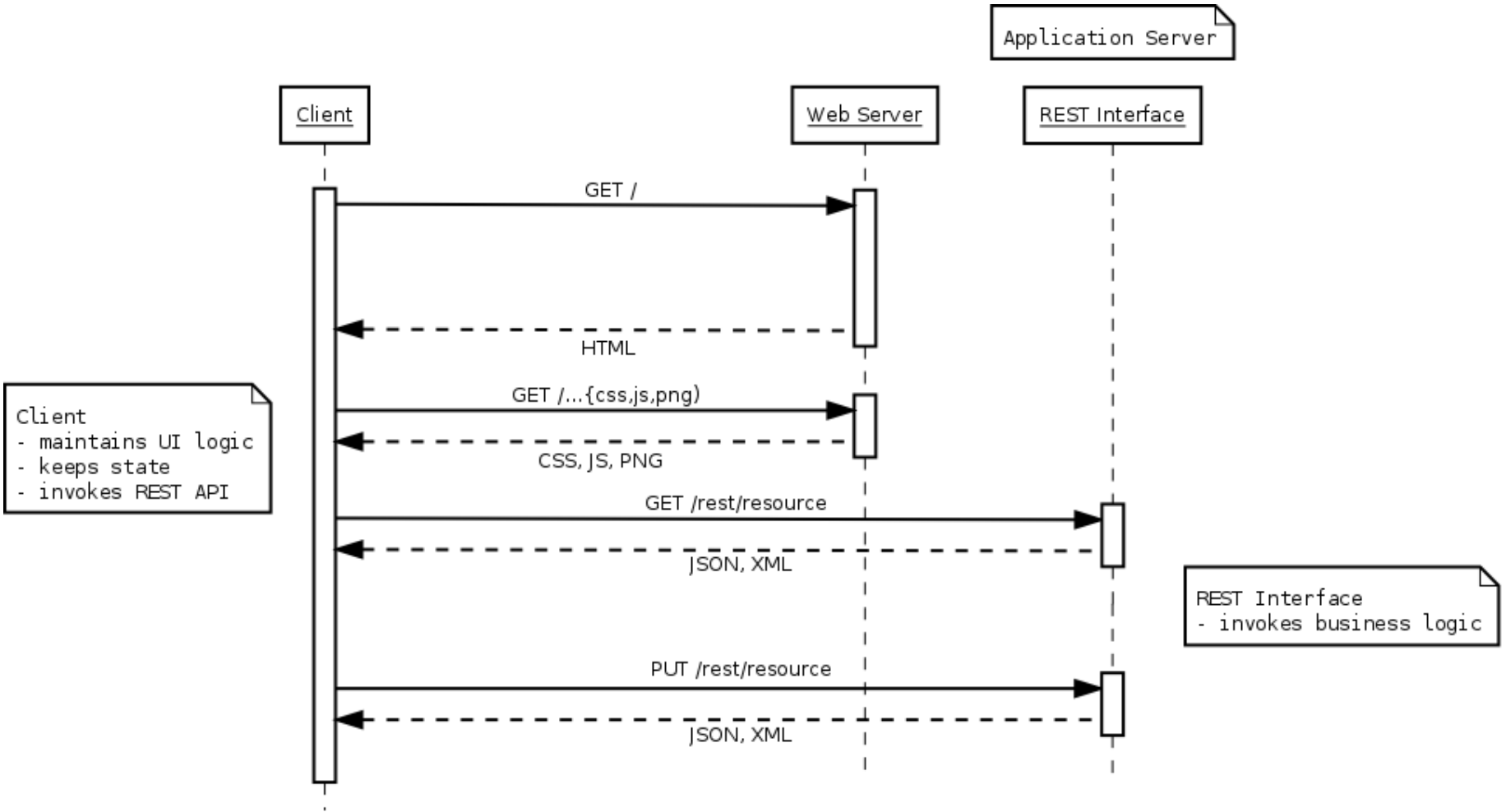
Client- vs. Server-Side Web

- Server-Side Web (Thin Client)
  - Well-established approach
  - 90's, 00's
- Client-Side Web (Thick Client)
  - Moden approach
  - SPA (Single Page Applications)
  - 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
  - Stateless, Scalable
- Client-Side Frameworks
  - Angular, React, Ember, Backbone, ...
- Standards improvements
  - HTML5 + Protocols
- REST interfaces
  - Data-oriented, presentation independent

Server-Side Advantages

- Well-Known
- No need for another layer of abstraction

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

- Application Resources
  - @ApplicationPath (or web.xml)
  - @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 **with** a known ID (modify/update)
@PUT	Modify or create **without** a known ID (create/modify)
@DELETE	Remove
@HEAD	GET with no response
(@PATCH)	json-patch

http://stackoverflow.com/questions/630453/put-vs-post-in-rest
http://stackoverflow.com/questions/25253899/how-to-implement-patch-requests-in-resteasy

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(“query”)	GET /consumer;name=???/orders

http://stackoverflow.com/questions/2048121/url-matrix-parameters-vs-request-parameters

New in JAX-RS 2.0

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

Demo

JAX-RS – Client API

Filters and Interceptors

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

Filters

- Non-wrapping extension points
  - Pre: `RequestFilter`
  - Post: `ResponseFilter`
- Each filter decides to proceed or break chain
  - FilterAction.NEXT, FilterAction.STOP

Interceptors

- Wrapping extensions points
  - ReadFrom: `ReaderInterceptor`
  - WriteTo: `WriterInterceptor`
- 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
- Client API support
  - Future<T>, InvocationCallback<T>

Demo

JAX-RS – Asynchronous

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
  - generate, parse

JSON-P APIs

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

Demo

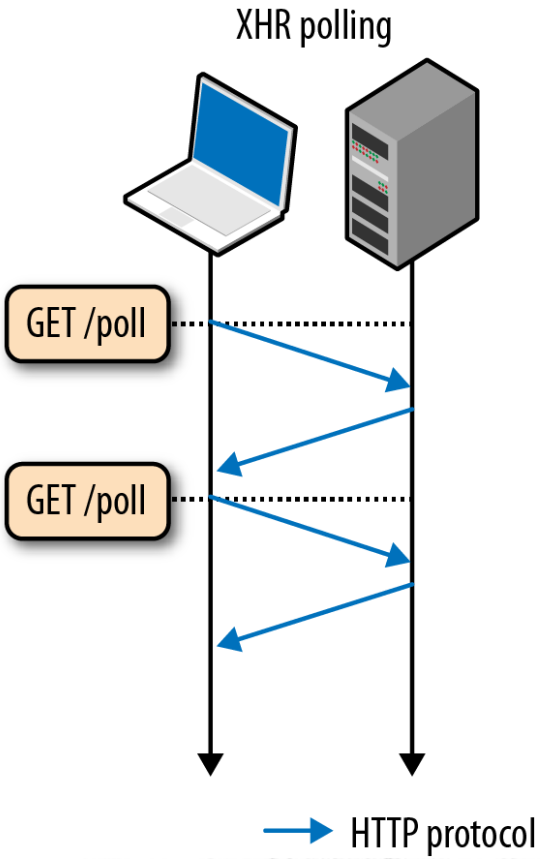
JSON Object Model API

Java API for WebSocket

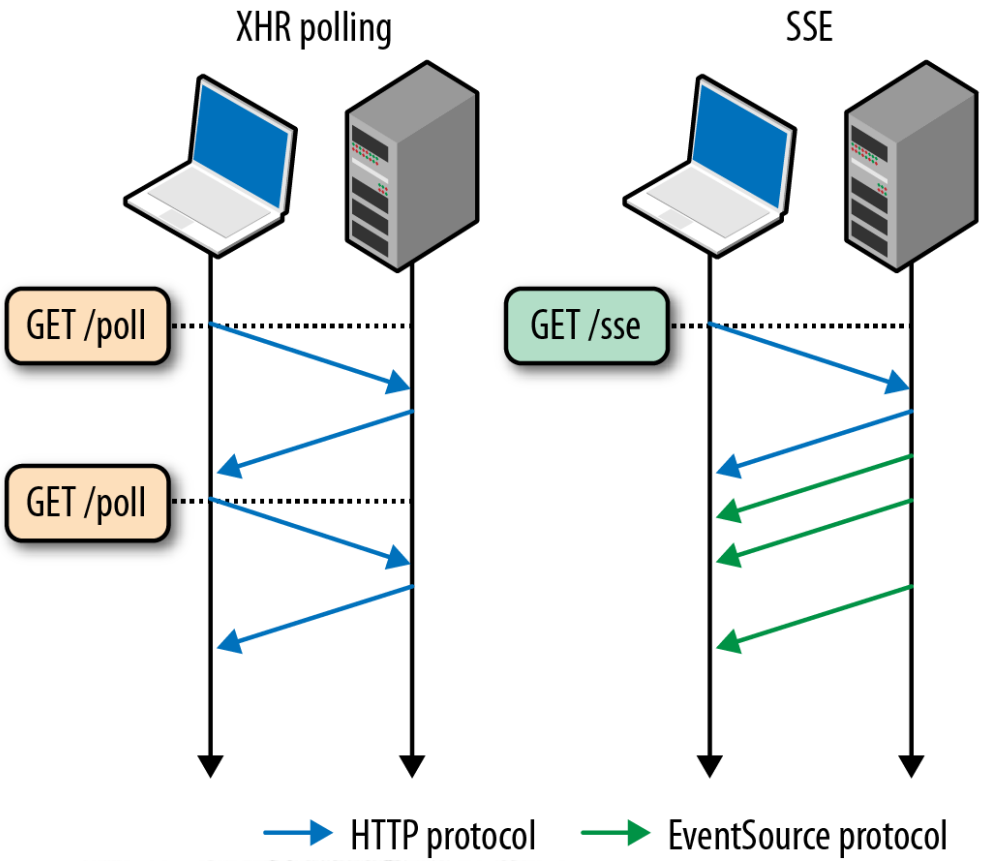
Motivation

- HTTP is half-duplex
- HTTP is inefficient
- HTTP hacked to achieve Push

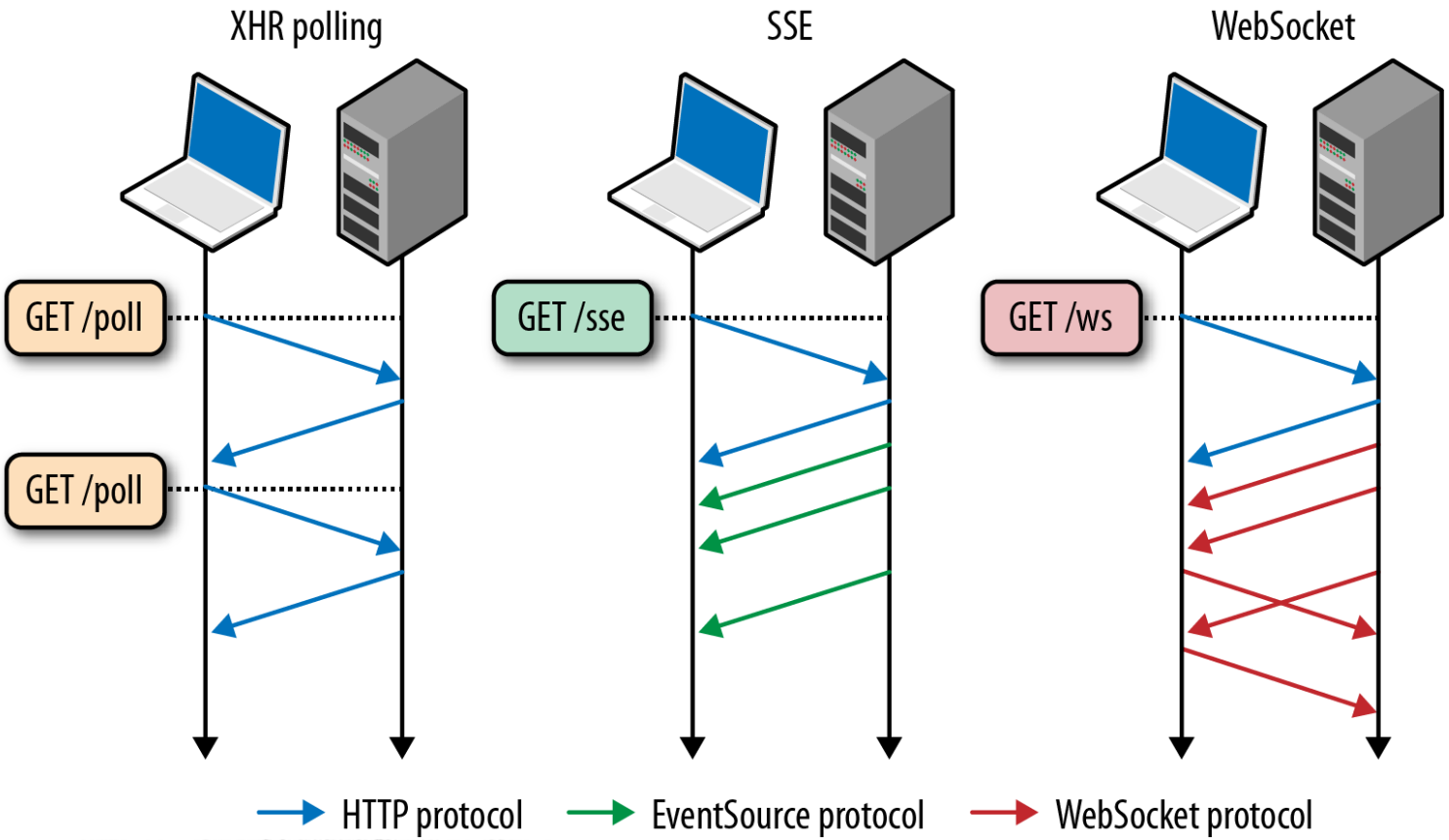
Server Push - Polling



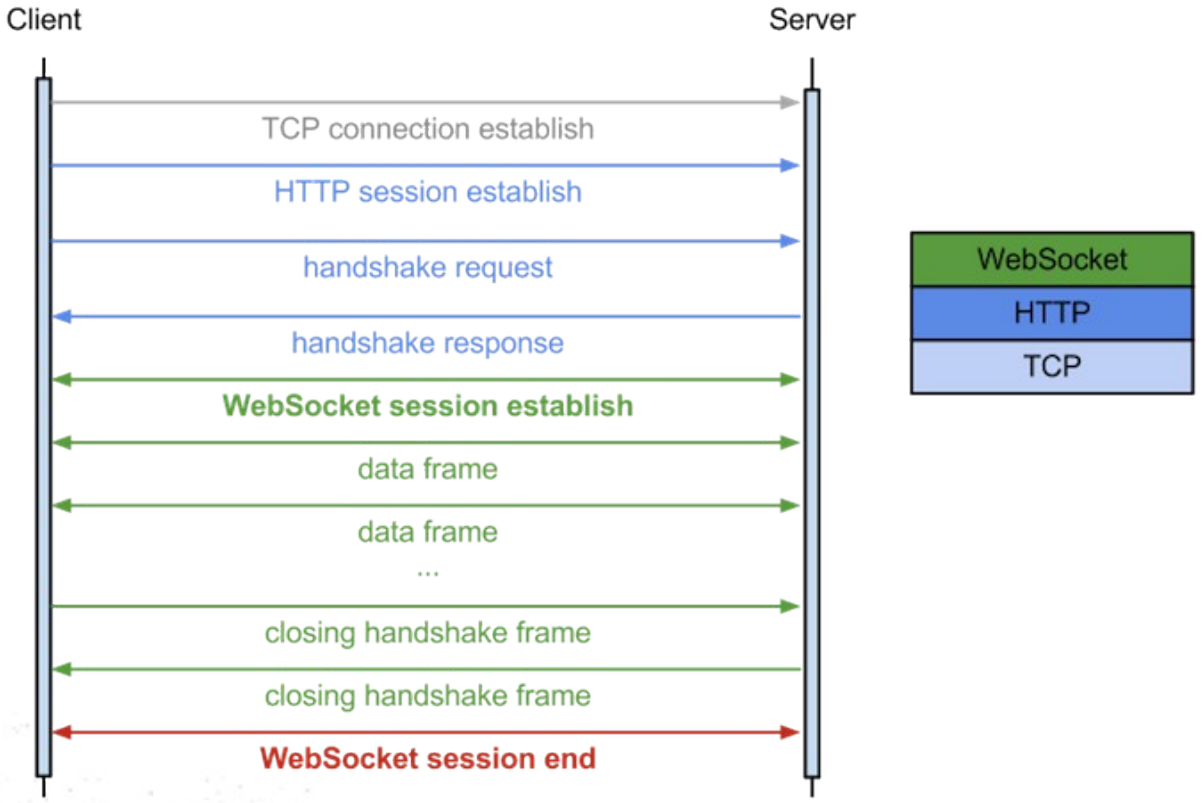
Server Push – SSE



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

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

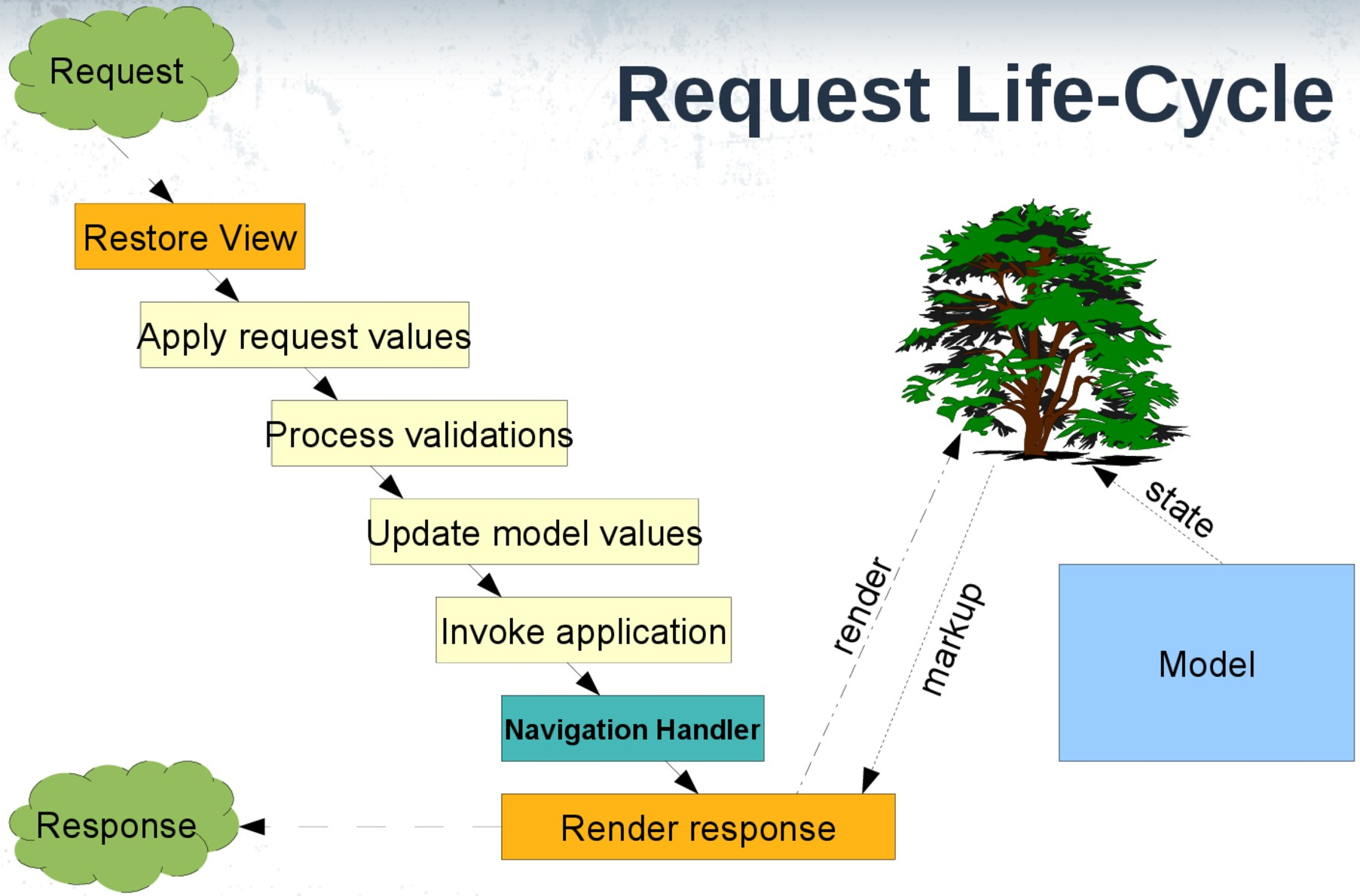
Component Libraries

- Rich components
  - PrimeFaces
  - RichFaces
  - ICEFaces
- Functionality
  - PrettyFaces – Pretty URLs, SEO, Bookmarks
  - OmniFaces – Nice features

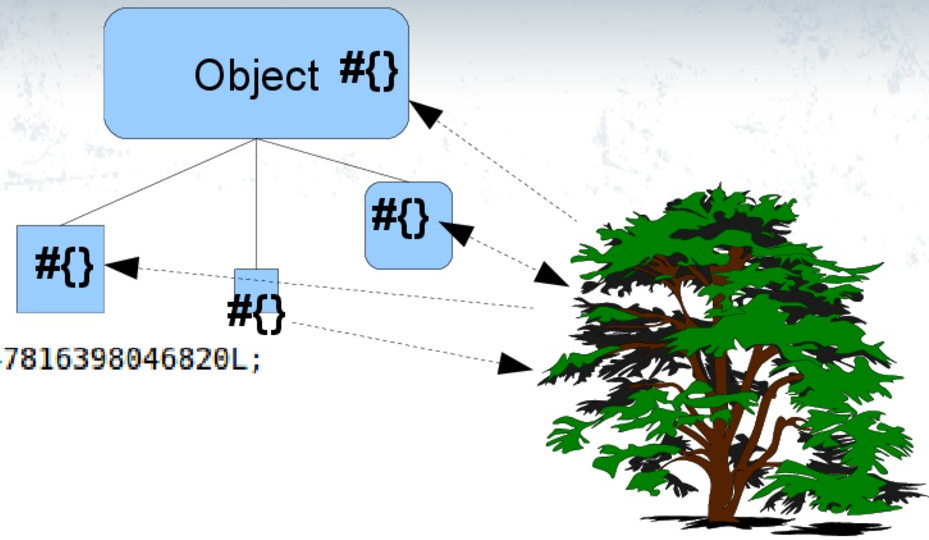
Component Tree

```
<!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



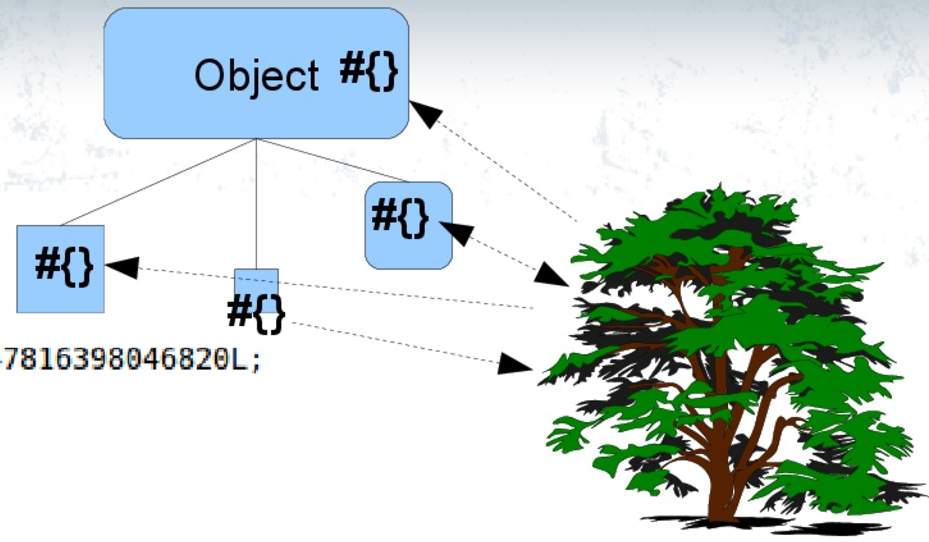
Model & View




```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

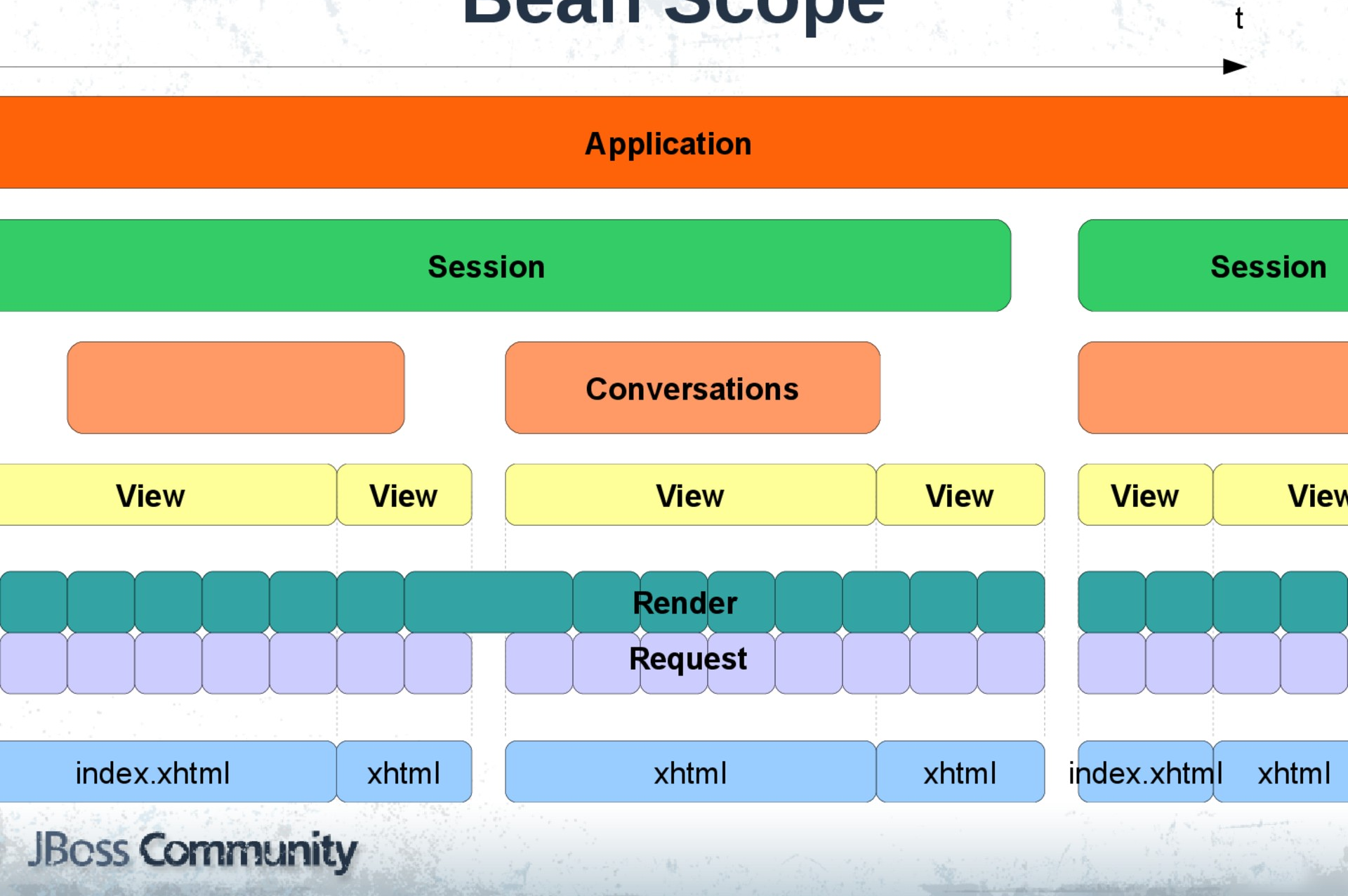



```
@Named
@SessionScoped
public class User implements Serializable {

    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 2.x Goals

- Standardize on top of Community Innovation
  - Ease of Use
  - Compelling Features
- Gradually Improve
  - Stateless
  - Security
  - HTML5 friendly

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>
```

That's it

Summary

- JSF
  - Fully-featured web framework core
- JAX-RS
  - RESTful endpoints, SPA, stateless
- WebSocket
  - Realtime bi-directional communication
- JSON-P
  - Standardization of JSON processing

Thank you

Links

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