1 Historical Overview

Web Applications

- http://www.evolutionoftheweb.com/

2 Frontend Technologies

2.1 Java World

Servlet API

- (HTTP-specific) classes for request/response processing
- Response written directly into output stream sent to the client
- Able to process requests concurrently

```
public class ServletDemo extends HttpServlet{

  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException{
    PrintWriter out = response.getWriter();
       out.println("<html>");
       out.println("<body>");
       out.println("<h1>Hello World!</h1>");
       out.println("</body>");
       out.println("</html>");
  }
}
```

Java Server Pages

- JSPs are text-based files containing:
  - Static data, usually HTML markup or XML
  - JSP technology elements for creating dynamic content
- JSPs are compiled into Servlets and returned to the client in response
- JSP Standard Tag Library (JSTL) - a library of common functionalities – e.g. forEach, if, out

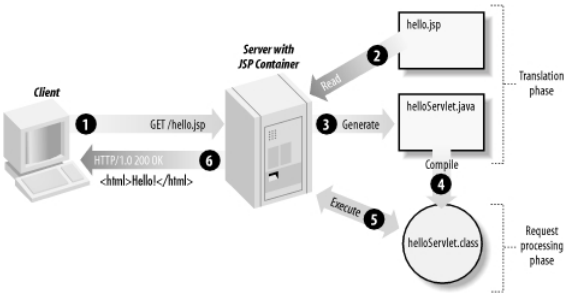


Figure 1: JSP processing. From `http://www.onjava.com/2002/08/28/graphics/Jsp2_0303.gif`

JSP Example

```
<html>
<head>
  <title>JSP Example</title>
</head>
<body>
  <h3>Choose a hero:</h3>
  <form method="get">
    <input type="checkbox" name="hero" value="Master Chief">Master Chief
    <input type="checkbox" name="hero" value="Cortana">Cortana
    <input type="checkbox" name="hero" value="Thomas Lasky">Thomas Lasky
    <input type="submit" value="Query">
  </form>

  <%
  String[] heroes = request.getParameterValues("hero");
  if (heroes != null) {
  %>
    <h3>You have selected hero(es):</h3>
    <ul>
  <%
      for (int i = 0; i < heroes.length; ++i) {
  %>
        <li><%= heroes[i] %></li>
  <%
      }
  %>
    </ul>
    <a href="<%= request.getRequestURI() %>">BACK</a>
  <%
  }
  %>
</body>
</html>
```

Java Server Faces

- Component-based framework for server-side user interfaces
- Two main parts:
  - An API for representing UI components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features
  - Custom JSP tag libraries for expressing UI components
- Components make it easier to quickly develop complex applications

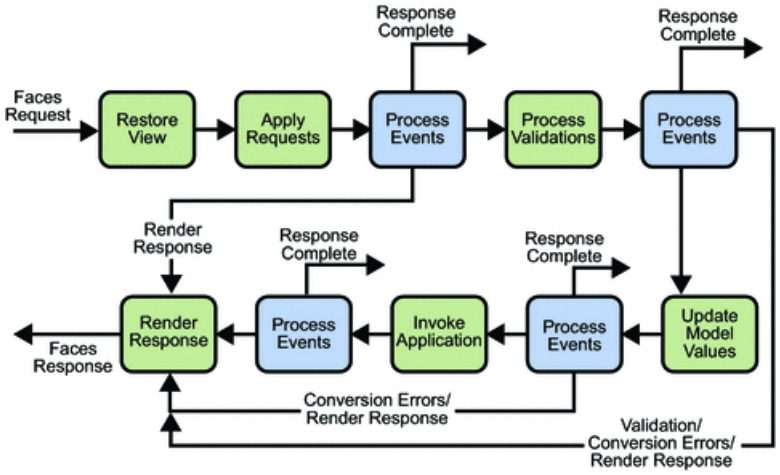


Figure 2: JSF lifecycle. From `http://docs.oracle.com/javaee/5/tutorial/doc/figures/jsfIntro-lifecycle.gif`

- Many component libraries - PrimeFaces, RichFaces, IceFaces
- Custom components add support for Ajax

JSF Lifecycle

JSF Example

```
<f:view>
    <h:head>
        <title>Book store – Users</title>
    </h:head>
    <h:body>
        <h1 class="title ui-widget-header ui-corner-all"><h:outputText value="#{msg['user-list.title']}"/></h1>
        <p:panel>
            <h:form>
                <p:growl />
                <p:dataTable var="user" value="#{usersBack.users}">
                    <p:column headerText="User">
                        <p:commandLink action="#{selectedUser.setUserById('user')}" ajax="false">
                            <h:outputText value="#{user.userName}" />
                            <f:param name="userid" value="#{user.id}" />
                        </p:commandLink>
                    </p:column>
                    <sec:ifAllGranted roles="ROLE_ADMIN">
                        <p:column headerText="Delete User">
                            <p:commandButton value="Delete" actionListener="#{usersBack.deleteUser(user.id)}"
                              update="@form" />
                        </p:column>
                    </sec:ifAllGranted>
                    <p:column headerText="Age">
                        <h:outputText value="#{user.age}"/>
                    </p:column>
                </p:dataTable>
                <p:link outcome="book-store-welcome-page" value="Home"/>
            </h:form>
        </p:panel>
        <p:commandLink action="#{loginBean.logout()}" value="Logout" />
    </h:body>
</f:view>
```

JSF Example II

```
@Component("usersBack")
@Scope("session")
public class UsersBack {

    @Autowired
    private UserService userService;

    public List<UserDto> getUsers() {
        return userService.findAllAsDto();
    }

    public void deleteUser(Long userId) {
        userService.removeById(userId);
        FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("User was sucessfully deleted"));
    }
}
```

Other Popular Frameworks

Google Web Toolkit (GWT) Write components in Java, GWT then generates JavaScript from them

Vaadin Built on top of GWT

Wicket Pages represented by Java class instances on server

Spring MVC Servlet-based API with various UI technologies, originally JSP, but also Thymeleaf, FreeMarker, Groovy Markup

Caveats of JSP/JSF?

- JSP, JSF are based on request/response, which requires frequent page reloads
- Almost everything happens on server
  - Requires a lot of resources (esp. given the JSF lifecycle)
  - Possible performance issues
- Limited support for AJAX
- Limited support for mobile devices
- Difficult to add new or extend existing components

2.2 JavaScript-based UI

JavaScript-based UI

- Client-side interface generated completely or partially by JavaScript
- Based on AJAX
  - Dealing with asynchronous processing

  - Events – user, server communication
  - Callbacks, Promises
  - When done wrong, it is very hard to trace the state of the application
- Enables dynamic and fluid user experience

No jQuery

- Plain jQuery use discouraged
- It is a collection of functions and utilities for dynamic page manipulation/rendering
- But building a complex web application solely in jQuery is difficult and the code easily becomes messy

JS-based UI Principles

- Application mostly responds by manipulating the DOM tree of the page
- Fewer refreshes/page reloads
- Server communication happens in the background
- Single-threaded (usually)
- Asynchronous processing

JS-based UI Classification

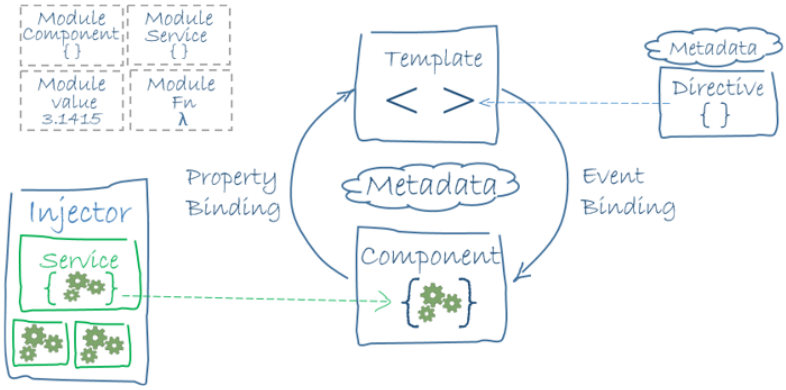
Declarative "HTML" templates with bindings, e.g. Angular.

```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<hero-detail *ngIf="selectedHero" [hero]="selectedHero"></hero-detail>
```

JS-based UI Classification

"Procedural" View structure is defined as part of the JS code, e.g. React.

```
class HelloMessage extends React.Component {
  render() {
    return <h1>Hello {this.props.message}!</h1>;
  }
}

ReactDOM.render(<HelloMessage message="World" />, document.getElementById('root'));
```



Angular

- Developed by Google, current version – Angular 7.0.x
- Encourages use of MVC with two-way binding
- HTML templates enhanced with hooks for the JS controllers
- Built-in routing, AJAX
- `https://angularjs.org/`

Angular Example

```
import { Component } from '@angular/core';
import { Hero } from '../hero';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent {
  hero: Hero = {
    id: 117,
    name: 'Master Chief'
  };

  constructor() { }
}
```

```
<h2>{{hero.name}} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div><span>name: </span>{{hero.name}}</div>
```

```
<div>
 <label>Name:</label>
 {{input type="text" value=name placeholder="Enter your name"}}
</div>
<div class="text">
 <h3>My name is {{name}} and I want to learn Ember!</h3>
</div>
```

React

A JavaScript library for building user interfaces.

- Created and developed by Facebook
- Used by Facebook and Instagram
- React Native for developing native applications for iOS, Android and UWP in JS
- Easy to integrate into legacy web applications
- Let's take a look at the EAR e-shop UI
- `https://facebook.github.io/react/`

Other JS-based Alternatives

Ember

- Open source framework,
- Templates using Handlebars,
- Encourages MVC with two-way binding,
- New components created using Handlebars templates + JS,
- Built-in routing, AJAX,
- `http://emberjs.com/`.

Other JS-based Alternatives

BackboneJS

- Open source framework,
- Provides models with key-value bindings, collections,
- Views with declarative event handling,
- View rendering provided by third-party libraries - e.g., jQuery, React,
- Built-in routing, AJAX,
- `http://backbonejs.org/`.

And many others...

```
var Todo = Backbone.Model.extend({


  defaults: function() {
    return {
      title: "empty todo...",
      order: Todos.nextOrder(),
      done: false
    };
  },


  toggle: function() {
    this.save({done: !this.get("done")});
  }

});
```

```
export function loadCategories() {
    const action = {
        type: ActionType.LOAD_CATEGORIES
    };
    return (dispatch) => {
        dispatch(asyncActionRequest(action));
        return axios.get('rest/categories')
            .then(resp => dispatch(loadCategoriesSuccess(resp.data)))
            .catch(error => {
                if (error.response.data.message) {
                    dispatch(publishMessage({message: error.response.data.message, type: 'danger'}));
                }
                return dispatch(asyncActionFailure(action, error.response.data));
            });
    };
}
```

3 Integrating JavaScript-based Frontend with Backend

Frontend – Backend Communication

- JS-based frontend communicates with REST web services of the backend
- Usually using JSON as data format
- Asynchronous nature
  - Send request
  - Continue processing other things
  - Invoke callback/resolve Promise when response received

Frontend – Backend Communication Example

↓

```
GET /eshop/rest/categories HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Accept: application/json
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36
```

```java
@RestController
@RequestMapping("/categories")
public class CategoryController {

    private static final Logger LOG = LoggerFactory.getLogger(CategoryController.class);

    private final CategoryService service;

    private final ProductService productService;

    @Autowired
    public CategoryController(CategoryService service, ProductService productService) {
        this.service = service;
        this.productService = productService;
    }

    @RequestMapping(method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
    public List<Category> getCategories() {
        return service.findAll();
    }
```

```javascript
export function loadCategories() {
    const action = {
        type: ActionType.LOAD_CATEGORIES
    };
    return (dispatch) => {
        dispatch(asyncActionRequest(action));
        return axios.get('rest/categories')
            .then(resp => dispatch(loadCategoriesSuccess(resp.data)))
            .catch(error => {
                if (error.response.data.message) {
                    dispatch(publishMessage({message: error.response.data.message, type: 'danger'}));
                }
                return dispatch(asyncActionFailure(action, error.response.data));
            });
    };
}
```

Frontend – Backend Communication Example II

```
HTTP/1.1 200 OK
Date: Sat, 17 Nov 2018 16:12:46 GMT
Server: Apache/2.4.10 (Debian)
Content-Type: application/json

{
  // JSON response body
}
```

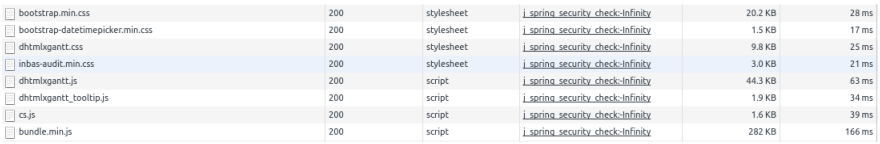
Frontend – Backend Communication Example III

4 Single Page Applications

Single vs. Multi Page JS-based Web Applications

Multi Page Web Applications Individual pages use JS, but browser navigation still occurs – browser URL changes and page reloads. Example: GitHub, FEL GitLab

Single Page Web Applications No browser navigation occurs, everything happens in one page using DOM manipulation. Example: Gmail, YouTube



bootstrap.min.css	200	stylesheet	j spring_security_check-Infinity	20.2 KB	28 ms
bootstrap-datetimepicker.min.css	200	stylesheet	j spring_security_check-Infinity	1.5 KB	17 ms
dhtmlxgantt.css	200	stylesheet	j spring_security_check-Infinity	9.8 KB	25 ms
inbas-audit.min.css	200	stylesheet	j spring_security_check-Infinity	3.0 KB	21 ms
dhtmlxgantt.js	200	script	j spring_security_check-Infinity	44.3 KB	63 ms
dhtmlxgantt_tooltip.js	200	script	j spring_security_check-Infinity	1.9 KB	34 ms
cs.js	200	script	j spring_security_check-Infinity	1.6 KB	39 ms
bundle.min.js	200	script	j spring_security_check-Infinity	282 KB	166 ms

Single Page Applications

- Provide more fluid user experience
- No page reloads
- View changes by modifications of the DOM tree
- Most of the work happens on the client side
- Communication with the server in the background
- Client architecture becomes important – a lot of code on the client

Single Page Application Specifics

- Almost everything has to be loaded when page opens
  - Framework
  - Application bundle
  - Most of CSS
- Different handling of security
- Different way of navigation
- Difficult support for bookmarking

Single Page Application Drawbacks

- Navigation and *Back* support
- Scroll history position
- Event cancelling (navigation)
- Bookmarking
- SEO
- Automated UI testing

4.1 Client Architecture

Client Architecture

- JS-based clients are becoming more and more complex
  - → neccessary to structure them properly
- Plus the asynchronous nature of AJAX
- Several ways of structuring the client

Model View Controller (MVC)

- Classical pattern applicable in client-side JS, too
- Controller to control user interaction and navigation, **no business logic**
- Frameworks often support MVC

Client Architecture II

Model View View-Model (MVVM)

- Originally developed for event-driven programming in WPF and Silverlight
- View-Model is an abstraction of the View
- Let the framework bind UI components to View-Model attributes (two-way binding)
- Controllers still may be useful

Flux

- Unidirectional flow
- Originated in React
- Simplifies reasoning about application state
- Separate business logic from UI components

Flux

The End

Thank You

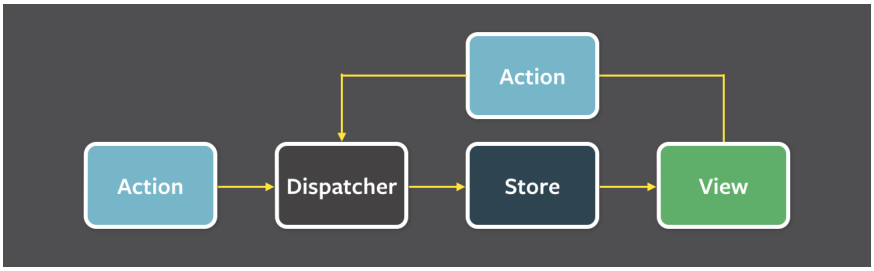


Figure 3: Flux architecture.

Resources

- M. Fowler: Patterns of Enterprise Application Architecture,
- `https://dzone.com/articles/java-origins-angular-js`,
- `https://msdn.microsoft.com/en-us/magazine/dn463786.aspx`,
- `http://singlepageappbook.com/index.html`,
- `http://adamsilver.io/articles/the-disadvantages-of-single-page-applications/`,
- `http://www.oracle.com/technetwork/articles/java/webapps-1-138794.html`.