Client-Server Model

File Protocol

- In browser: file://<path>
- (html) bestand op eigen pc

OF

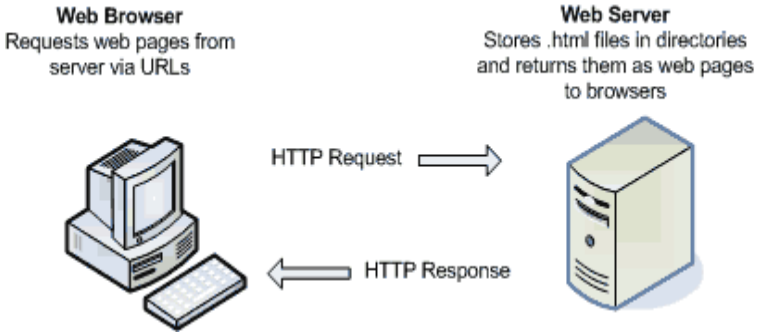
- In browser: file://<host>/<path>
- (html) bestand op vaste plaats op andere PC in zelfde netwerk

HTTP (HyperText Transfer Protocol)

➔ Bestand toegankelijk via internet
- In browser: http(s)://<host>/<path>
- (html) bestand op http server

➔ Werkt volgens Client Server Model: Client=vb Browser, Server=vb Apache
➔ Web applicatie laag protocol, request-response protocol
➔ Beschrijft hoe communicatie over WWW gebeurt
  - Hoe boodschappen gevormd & overgebracht moeten worden
  - Hoe browsers en web servers hiermee moeten omgaan
    - Soorten boodschappen: Get, Post, Put, Delete, …

Client-Server Model



Clients	Server
Needs service	Offers service
Runs client program	Runs server program
Share no resources	Shares resources w client
Starts communication	Waits for request
Asks (request)	Answers (response)

HTTP - GET

GET Request

Link klikken of ingeven in adresbalk -> GET Request
Van de vorm: *http(s)://<host>/<path>*
Vb: *http://localhost:8080/Web_1_Servlet/index.html*

GET /Web_1_Servlet/index.html HTTP/1.1 **-> Eigenlijke Request (path)**
Host: localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:30.0) Gecko/20100101 Firefox/30.0 Accept: text/html,application/ xhtml+xml,application/xml;q=0.9,/*;q=0.8*
Accept-Language: nl-be,en-gb;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: textwrapon=false; wysiwyg=textarea
Connection: keep-alive **-> Headers met extra info**
-> Lege lijn = 'einde bericht'

Response

HTTP/1.1 200 OK **-> Eigenlijke response**
Server: Apache-Coyote/1.1

```
Accept-Ranges: bytes
Etag: W/"398-1403942012000"
Last-Modified: Sat, 28 Jun 2014 07:53:32 GMT
Content-Type: text/html
Content-Length: 398
Date: Sat, 28 Jun 2014 08:54:48 GMT
```

-> Headers met extra info

-> Lege lijn

```
<html>
    <head>
        <title>My first page</title>
        <link rel="stylesheet" href="css/sample.css">
    </head>
    <body>
        <h1>Welcome to the course <em>Web 2</em></h1>
        <h2>Have fun!</h2>
        <img src="images/dog.jpg" width="340" height="367" alt="">
            <p>
                <a href="DiceServlet">Play dice</a>
            </p>
    </body>
</html>
```

-> Response body: html

GET

- Doel: data ophalen
- Afspraak: GEEN side-effect
- Link: http://<host>/<path>
- Inhoud bericht:

GET Request	Response
Request lijn met **URL**	Response lijn met **status code**
Headers met extra info	Headers met extra info
	Body met bv. **HTML**

- **Note that the query string (name/value pairs) is sent in the URL of a GET request:**
- /test/demo_form.php**?name1=value1&name2=value2**
- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

Foutopsporing

HTTP Error codes

Code	Betekenis	Info
1xx	Informational	Request is ontvangen, proces gaat door
2xx	Success	Request is ontvangen, begrepen, geaccepteerd en succesvol verwerkt
3xx	Redirection	Client moet bijkomende actie ondernemen om request af te werken
4xx	Client error	Foutboodschap door client veroorzaakt, zoals verkeerd typen URL
5xx	Server error	Fout opgetreden aan de server kant

Build en Deploy

Build: Java compileren, bestanden in juiste folderstructuur zetten, alle comprimeren -> War

Deploy: War in juiste folder van webcontainer zetten (Tomcat: webapps), unzip

WebContent -> Web bestanden die je zelf schrijft
WEB-INF -> Bestanden niet (rechtstreeks) publiek toegankelijk zijn
- Deployment descriptor file: *web.xml*
- *lib*: Jar bestanden die aan applicatie classpath worden toegevoegd @runtime
- *classes*: klassen die applicatie gebruikt die niet in een jar zitten

HTTP Server vs Web Container

Dynamische websites: http server volstaat niet meer -> Web container nodig die met Java kan werken *(of php interpreter voor php)*

Web Container: kan alles wat HTTP server kan (antwoorden op http requests en html pagina's teruggeven) + kan Java code uitvoeren (met servlets)
Servlet: Klasse die HTML genereert, 'gewone' klasse, wordt door web container herkend en beheerd

Hoe weet Web Container om naar Servlet te gaan ipv bestand ophalen?
Vroeger: Servlet-mapping in web.xml bestand

```
<servlet-mapping>
    <servlet-name>DiceServlet</servlet-name>
    <url-pattern>/Dice</url-pattern>
</servlet-mapping>
```

Nu: Annotations – Container overloopt alle Java klassen en kijkt of een annotatie @WebServlet staat > Container checkt of URL bij annotatie overeenkomt met URL uit request

```
@WebServlet("/Dice")
public class DiceServlet extends HttpServlet
```

HTTPServlet

Bevat en uitgewerkt: init(ServletConfig), Service(request, response) en destroy()

Init(ServletConfig): Eenmaal opgeroepen, door webcontainer, na creatie servlet
Destroy(): Eenmaal opgeroepen, door webcontainer, vernietigd servlet
Service(request, response): Telken opgeroepen, door webcontainer, bij iedere aanroep servlet, roept doGet() of doPost() op, telkens in afzonderlijke thread

Override (zelf schrijven): doGet(request, response), doPost(request, response)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { … }
```

HttpServletRequest: Web container steekt alle gegevens van de request in object
HttpSerlvetResponse: Web container zet alvast een Response-object klaar
Exceptions: voor als er iets fout gaat

! Thread per request aangemaakt door Service() (die doGet, of doPost oproept)

PrintWriter

➔ Om weg te schrijven naar het Response-object

```
PrintWriter out = response.getWriter();
out.println("html code");
```

Automatisatie / Selenium

⇨ Selenium WebDriver, bootst echte gebruiker na

1. Driver

- Zeg Selenium hoe te communiceren met browser
  System.setProperty("webdriver.chrome.driver", "/yourPath/chromedriver");
- Maak driver object om browser te simuleren
  WebDriver driver = new ChromeDriver();
- Opm: driver per type browser

2. Start URL

- get(): navigeer naar URL
  driver.get("url");

3. Elementen vinden

- findElement(): HTML element zoeken
- By: Zoeken op id, name, css selector, …
- *WebElement searchField = driver.findElement(By.id("searchinput"));*
  WebElement welcomeParagraph = driver.findElement(By.tagName("p"));
  WebElement errorDiv = driver.findElement(By.cssSelector(".error"));

4. User input

- sendKeys(): typen
  searchField.sendKeys("KHL");
- click(): klikken
  goButton.click();
- clear(), …

5. Checking the result

- Assert methods
  assertEquals("Kontinental Hockey League" , driver.findElement(By.cssSelector("caption")).getText());

6. Quit the driver

- Quit method (closes browser automatically)
  driver.quit();

7. Extra jar needed

- Selenium Standalone Server toevoegen aan build path (ook JUnit)

JUnit

Voordeel:

- Assert-methodes om resultaat te controleren
- Automatische controle
- Testrapport

Wat testen? - Form

- Alle velden juist ingevuld
- Resultaat voor elk leeg invullen
- Randgevallen (min, max)

Don't repeat yourself (DRY)

- Gemeenschappelijke code isoleren in aparte klasses

- **@Before**: wordt altijd herhaald **VOOR** de uitvoering van **iedere** testmethode
- **@After**: wordt altijd herhaald **NA** de uitvoering van **iedere** testmethode

JSP (Java Server Page)

Nadeel schrijven html in Servlet: - Veel out.println() -> onleesbare code
- geen ondersteuning voor validatie HTML

⇨ Andere aanpak: In plaats van HTML in Java -> Java in HTML -> JSP

! Op basis van JSP pagina -> Web Container genereert servlet (jasper) die deze HTML wegschrijft naar een response object > omzet naar HTTP Response bericht > Terugstuurt naar client *(in workspace/.metadata/.plugins /org.eclipse.wst.server.core/tmp0/work/Catalina/localhost/<project_name>/org/apache/jsp)*

JSP

- Lijkt op html pagina
- Heeft bovenaan een aantal JSP **directives**: info voor de compiler
- Bevat JSP **scriplets**: stukjes Java-code -> tussen **<% %>**

JSP Directives

- **Page** *<%@ page import="java.util.*"%>*
  - import statements
  - character encoding
  - bovenaan pagina gebruiken
- **Include** *<%@ include file="header.jsp"%>*
  - html of jsp bestanden hergebruiken
  - waar je andere pagina wilt tonen gebruiken
- **Taglib** (zie later)

JSP Expression

⇨ Tussen <%= %>

```
<% java.util.Date date = new java.util.Date(); %>
<p>The time is now <strong> <%= date %> </strong></p>
```

GET + Parameters

- GET request (parameters in URL = querystring)
- POST request (parameters in body van request)

➔ Voor elke parameter: key-value paar -> gescheiden door ampersands (&)
➔ Protocol://Host/Path?QueryString
http://localhost:8080/GuessGame/guess.jsp?guess=4&numberOfGuesses=3
- Web Container zet alle gegevens uit HTTP request om in het request-object
-> Parameter opvragen met getParameter("key")

```
<%String guessFromParameter = request.getParameter("guess");
int guessedNumber = Integer.parseInt(guessFromParameter);%>
<p>You guessed <%=guessedNumber%>...</p>
```

JSP declaration

➔ Met: <%! %>
➔ Instantievariabele

- Server maakt maar 1 instantie van servlet aan, die wordt gebruikt voor alle requests van gebruikers. Instantievariabele zal dus gelijk zijn voor alle gebruikers *(in guess game is juiste getal hetzelfde voor alle gebruikers), anders bij elke reload van client zou er een nieuw te raden getal worden gekozen)*

JSP vs Servlet

Nadeel logica in JSP: - Veel Java in JSP
- Presentatie & logica niet gescheiden

RULES: - No HTML in Java -> Presentation: html/jsp
- As little Java in HTML as possible -> Logic: java

➔ Logica naar Servlet

Waar HTML genereren? -> Nog steeds JSP pagina (vanuit Servlet request en response forwarden)
RequestDispatcher view = request.getRequestDispatcher("result.jsp");
view.forward(request, response);

Resultaten uit logica in Servlet doorgeven? Kunnen zelf geen parameters aan request (die wordt geforward naar jsp) meegeven, maar wel **Attributen** -> *request.setAttribute("key", value);*

➔ In JSP: <%=*request.getAttribute("key");*%> voor de in Servlet toegevoegde attributen
<%=*request.getParameter("key"*)%> voor oorspronkelijke parameters
➔ JSP berekent niks meer!

Request Parameters

- Parameters worden altijd door client meegegeven in de request als querystring (bij GET)
- Achteraf **geen extra** parameters toevoegen
- Enkel **String**
- Methodes: *getParameter(name: String)* en *getParamaterNames()*

Request Attributen

- Kunnen **vanuit code** toegevoegd worden
- Niet alleen String, alle **objecten**
- Methodes: *setAttribute(name: String, value: Object), getAttribute(name: String), getAttributeNames(), …*

Request Forwarden

1. RequestDispatcher vragen aan request

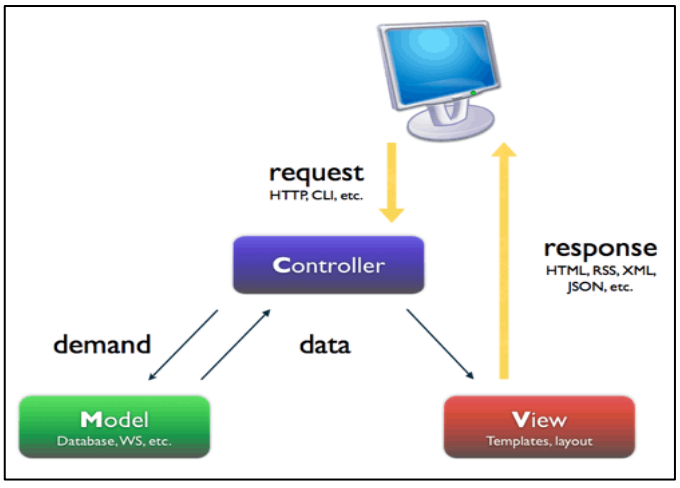
➔ *RequestDispatcher view = request.getRequestDispatcher("result.jsp");*

- Bestaande Java klasse
- Op te vragen aan request met methode *getRequestDispatcher(destination: String)*
- URL meegeven waar men naartoe wil

2. Request en Response forwarden

➔ *view.forward(request, response);*

- Via *forward()* methode van RequestDispatcher



MVC (Model-View-Controller)

➔ **Businesslogica** scheiden van **user interface** > Elk onafhankelijk ontwikkelen, testen & onderhouden

- **Model**: alle data, toestand en businesslogica
- **View**: visuele representatie van model
- **Controller**: reageert op input van gebruikers & stuurt de communicatie tussen view & model

Voordeel: klassen die logica bevatten (kern applicatie) kunnen hergebruikt worden.

Packages

UI > View (in WebContent folder) & Controller
Domain > Service, Model & DB
Test > Selenium testen -> in package ui.view; JUnit testen voor domain in domain.model

POST

HTTP Methode

- Form-attribuut *method*
- *<form method="HTTP methode">*
- Standaard: GET

HTTP POST Request

POST /Study/Course HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:31.0) Gecko/20100101 Firefox/31.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/*;q=0.8*
Accept-Language: nl-be,en-gb;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/Study/courseForm.jsp
Cookie: JSESSIONID=08E34B1AF6FE4D802A0F3B9FC6F164C4; textwrapon=false; wysiwyg=textarea
Connection: keep-alive

title=BOP&code=1234&credits=6&contact=4 **-> Parameters in body van bericht**

- **GET**
  - GEEN side-effect
  - Voor navigatie, ophalen gegevens, …
  - 1 of 100 keer sturen maakt niet uit
    - Idempotent!
- **POST**
  - Wel side-effect
  - Voor toevoegen, wijzigen, verwijderen, …

	GET	POST
Bookmark	V	X
Cache	V	X
Restrictions on length	V	X
Restrictions on type	V	X
Back button	V	X (warning)
Parameters in history	V	X
Security	X	V
Visibility	V	X

Navigatie

- Form-attribuut *action*
- *<form action="URL">*, ***URL=vb path van servlet***
- Indien leeg > zelfde pagina

Web.xml

- Start van web applicatie definieren (standaard index.html/jsp)
  <welcome-file-list>
  <welcome-file>courseForm.jsp</welcome-file>
  </welcome-file-list>
- Moet in WEB-INF map zitten > Web container gebruikt deze om applicatie te starten

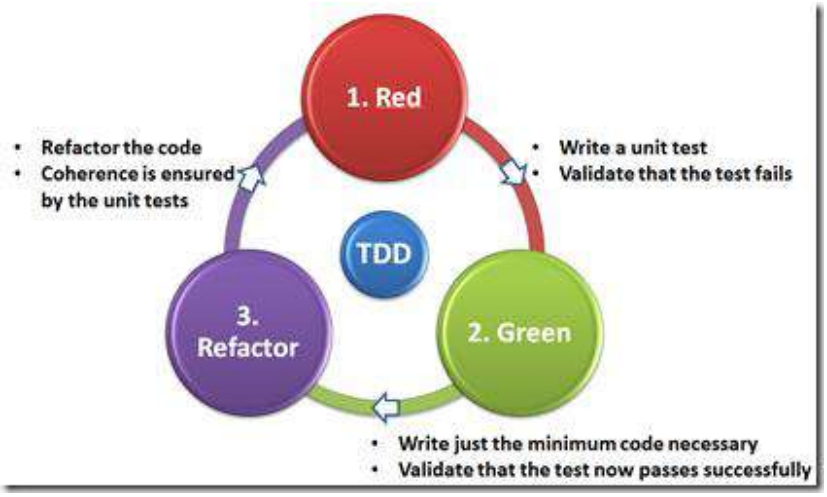
Test Driven Development (TDD)

Refactoring is deel van ontwikkelingscyclus

Nieuwe code ontwikkelen, enkel focussen op dat het werkt, nog niet 'perfect design'.

Wanneer het werkt, reflecteren op de kwaliteit van de code en zo nodig refactoren.

Test klassen zorgen ervoor dat je bestaande functionaliteit niet kapot maakt.



Validation

➔ **!** Controleer input (ook) server-side

Implementatie

- Model
  - Exceptions op de juiste plaatsen
- Controller
  - Exceptions opvangen
  - Error informatie aan view geven
- View
  - Error informatie tonen

Client-side vs Server-side Validation

Client-side Validation

- HTML 5
- Browser support?
- Vertrouw gebruiker nooit: altijd server-side validatie ook!
- Afzetten bij testen server-side validatie!
  - Attribuut *novalidate* aan form toevoegen
- ! Let Op: Date wordt bij FF en Chrome anders doorgegeven aan web server
  - Oplossing: extra attribuut toevoegen aan input element: *pattern="[0-9]{4}-(0[1-9]|1[012])-(0[1-9]|1[0-9]|2[0-9]|3[01])*

Server-side Validation

- Enkel client-side is niet genoeg!
  - Kan gemanipuleerd/afgezet worden
- Server-side validation
  - = nakijken in de controller of de input is wat we verwachten
  - Moet gebruikt worden!

Front Controller

Controller:

- ADD > doPost()
- READ > doGet()
- DELETE > doPost()
- FIND > doPost()

Probleem: Servlet maar 1 doGet en 1 doPost

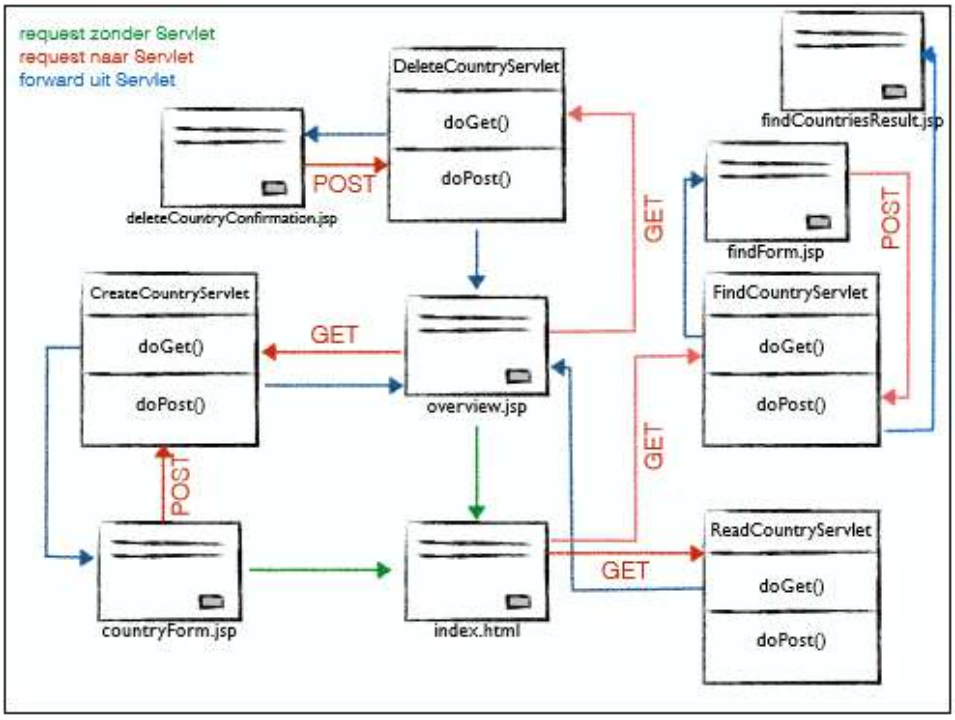
Oplossing 1 – Meerdere Servlets

Problemen:

- View moet veel weten over welke controllers er zijn

Oplossing 2 – Eén Servlet met parameter voor action

- ➔ **Front Controller pattern**
- ➔ **"Single Point of Entry"**
  - Handles all requests
  - Delegates
    - Business processing
    - Navigation



Met waarde:

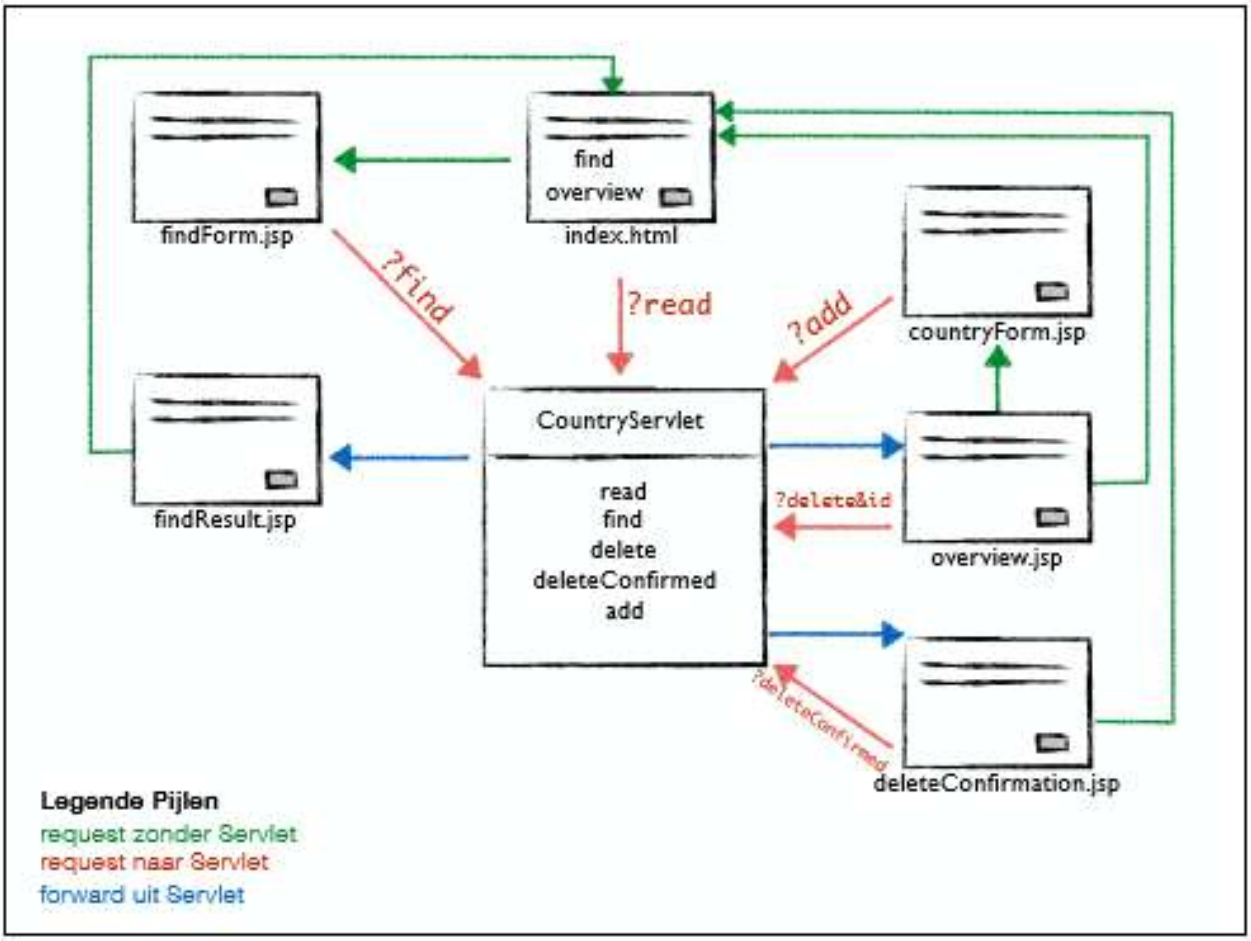
- Create
- Read
- Find
- Delete
  - *<form method="POST" action="Controller?action=create">*
  - <td><a href="Controller?action=delete&id=<%= country.getName()%>">Delete</td>

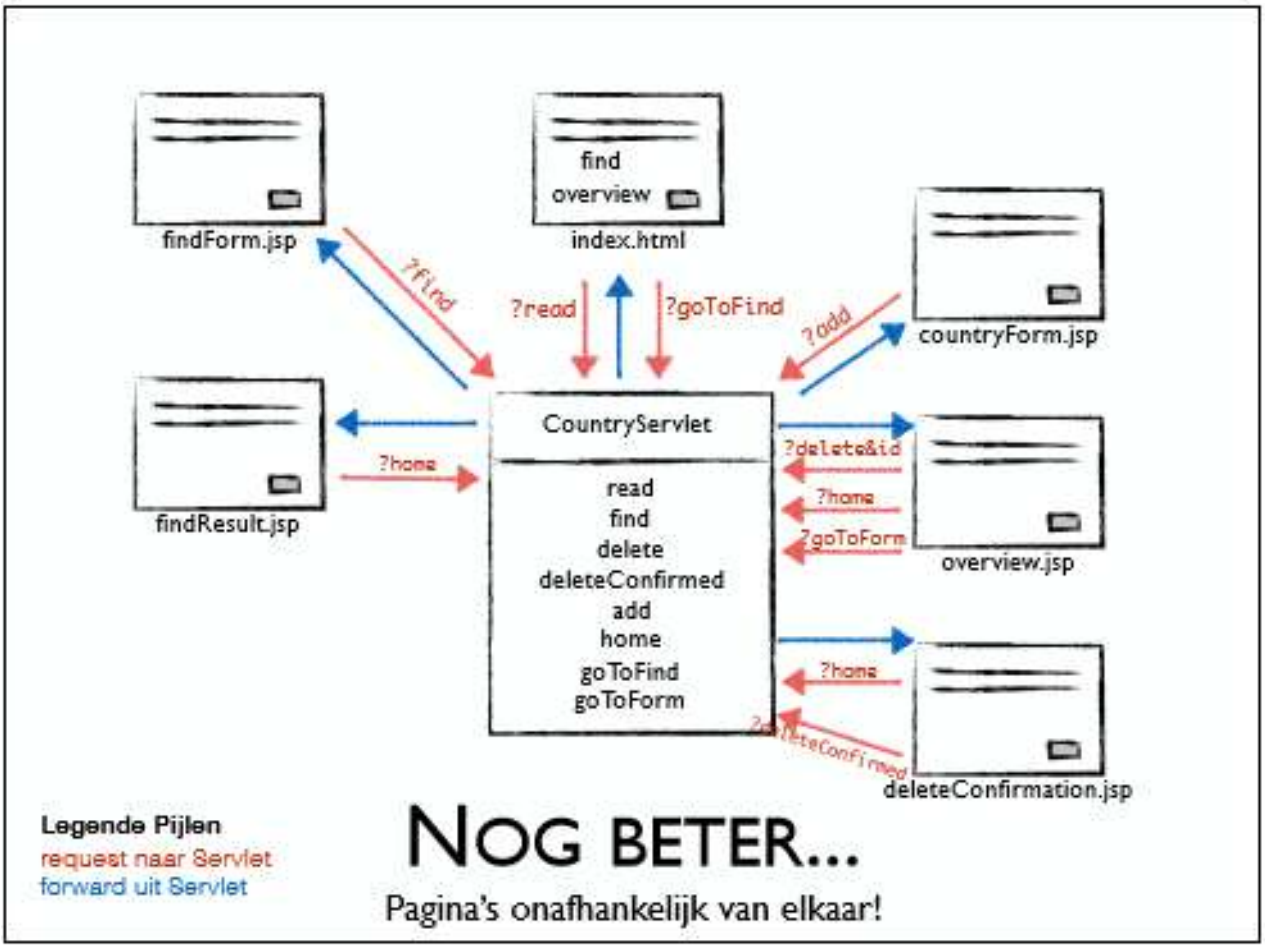
In Controller:

- If structuren wordt onleesbaar -> **SWITCH**
  - String action = request.getParameter("action");

```
switch(action) {
        case "read" :
                destination = showCountriess(request, response);
                break;
        case "delete" :
                destination = deleteCountry(request, response);
                break;
        case "create" :
                destination  = createCountry(request, response);
                break;
        case "find" :
                destination =  findCountry(request, response);
                break;
        default :
                destination = "index.html";

}
RequestDispatcher view = request.getRequestDispatcher(destination);
view.forward(request, response);
```


find
overview
index.html
findForm.jsp
?find
?read
?add
countryForm.jsp
CountryServlet
read
find
delete
deleteConfirmed
add
findResult.jsp
?delete&id
overview.jsp
deleteConfirmation.jsp
Legende Pijlen
request zonder Servlet
request naar Servlet
forward uit Servlet


find
overview
index.html
findForm.jsp
?find
?read
?goToFind
?add
countryForm.jsp
CountryServlet
?home
?delete&id
?goToForm
read
find
delete
deleteConfirmed
add
home
goToFind
goToForm
findResult.jsp
overview.jsp
deleteConfirmation.jsp
Legende Pijlen
request naar Servlet
forward uit Servlet
NOG BETER...
Pagina's onafhankelijk van elkaar!

JSPF (Java Server Page Fragment)

➔ Reuse parts of your html > Don't Repeat Yourself (DRY)

➔ *<%@include file="header.jspf" %>* (jsp directive)

Web container:

- Inserts inhoud van het jsp fragment/segment in de jsp pagina
- Kan ook hele jsp pagina zijn (ook jsp directive)
- Compileerd het naar 1 servlet

➔ Niet dynamisch: insertion gebeurt tijdens vertaling (deel van de JSP cyclus waar JSP wordt omgezet in equivalent servlet) en dus niet @runtime

CRUD

➔ = Create Read Update Delete