



SAMENVATTING WEBONTWIKKELING 3

Martijn Meeldijk



9 JANUARI 2020
UCLL

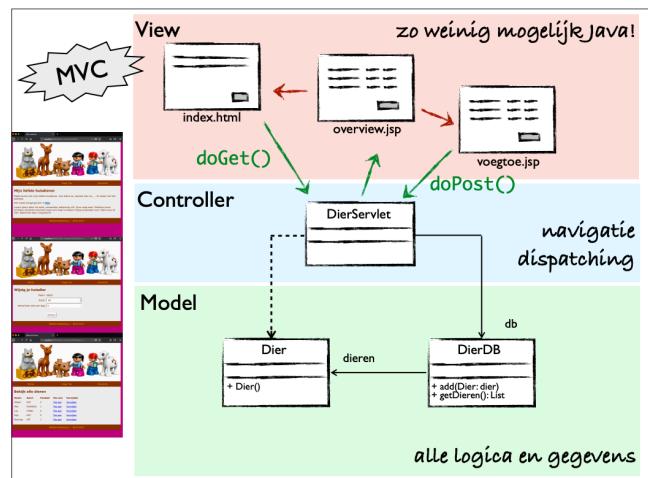
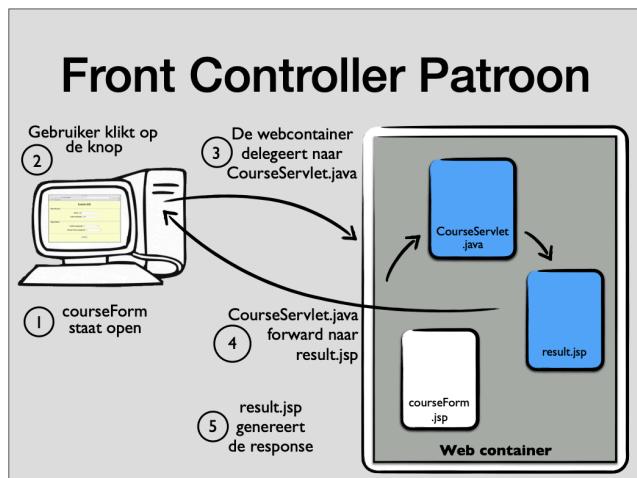
INHOUD

Les 1	2
<i>MVC en Front Controller</i>	2
<i>Cookies</i>	2
<i>Sessions</i>	3
<i>Expression language</i>	4
Les 2	5
<i>JDBC</i>	5
<i>Exception Handling</i>	6
Les 3	7
<i>Database Design</i>	7
<i>Error page</i>	9
Les 4	10
<i>Cross-site-scripting</i>	10
Les 5	12
<i>Configuratie</i>	12
<i>Hashing</i>	15
<i>Salt</i>	16
Les 6	17
<i>Post/Redirect/Get</i>	17
<i>Front controller continued</i>	19
<i>Authenticatie</i>	20
Les 7	21
<i>Authorisatie</i>	21
Les 8-10	22
<i>Javascript</i>	22

Volg mij op SoundCloud: <https://soundcloud.com/kingmarti>

Les 1

MVC en Front Controller



Cookies

HTTP is stateless protocol. Als de client een response ontvangt, gaat de informatie van de voorafgaande request verloren.

Hoe zorgen we ervoor dat die informatie behouden kan blijven? (stateful)

Cookies!

Stap 1: cookie maken

Om een cookie aan te maken:

- web container
 - leest informatie uit request object
 - maakt cookie-object aan
 - voegt cookie toe aan response object
- webserver stuurt instructie "set-cookie" mee met http response
- client bewaart gegevens van cookie op harde schijf

Stap 2: cookie gebruiken

Als client een request stuurt naar host waarvoor een cookie bestaat:

- client voegt cookie toe aan header van http request
- web container leest cookies uit request object
- web container verwerkt request, rekening houdend met de waarde van de ontvangen cookies

Sessions

Hoe kan de webcontainer complexere informatie over de client bewaren? Bijvoorbeeld lijst van producten, zoekgeschiedenis, informatie over de client die niet bewaard kan worden als string.

Stap 1: sessie maken

Om een sessie aan te maken:

- web container
 - leest informatie uit request object
 - maakt sessie object aan (request.getSession())
 - bewaart de nodige informatie in dit object (session.setAttribute())
- web server
 - stuurt de id van de sessie naar de client in de response header als cookie

Stap 2: sessie gebruiken

Als er een sessie bestaat voor een host:

- client voegt cookie met session id toe aan de request header
- server gebruikt deze id om de juiste sessie te vinden in de web container
- web container verwerkt de request, rekening houdend met de gegevens die de sessie bevatten

```
request.getSession()  
    - geeft bestaande sessie OF  
    - maakt nieuwe sessie aan als er nog geen bestaat  
  
session.setAttribute(name, object)  
    - voegt informatie toe aan sessie  
    - als attribuut met gegeven naam bestaat, wordt bestaande vervangen  
  
session.getAttribute(name)  
    - leest informatie uit de sessie
```

Expression language

Implicit Objects

`${attribuut.property}`

`${implicitObject.keyOrProperty}`

↓
pagescope
requestScope
sessionScope
applicationScope
param
paramvalues
header
headervalues
cookie
initParam
pageContext

`${header["host"]}`

`${param.id}`

`${initParam.mainEmail}`

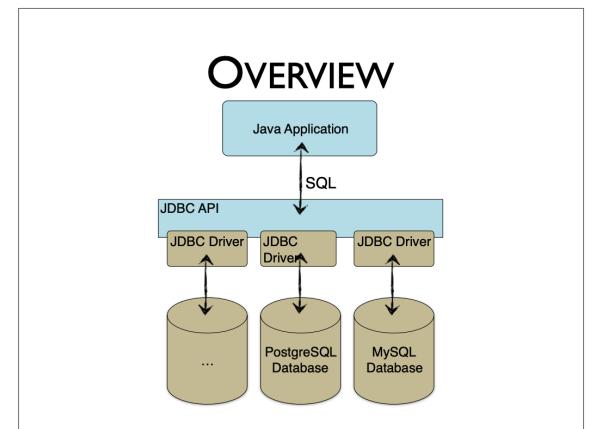
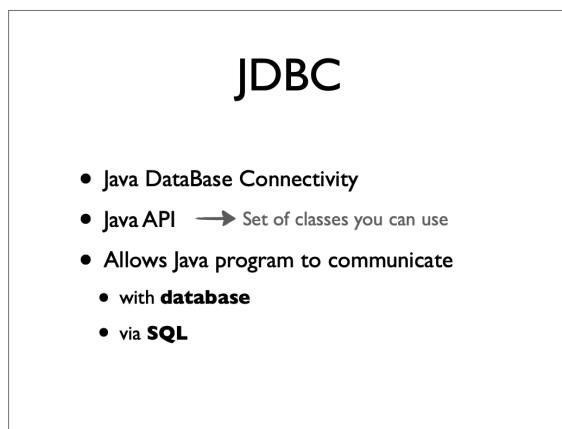
`${cookie.JSESSIONID.value}`

Les 2

JDBC

Mogelijke examenvragen:

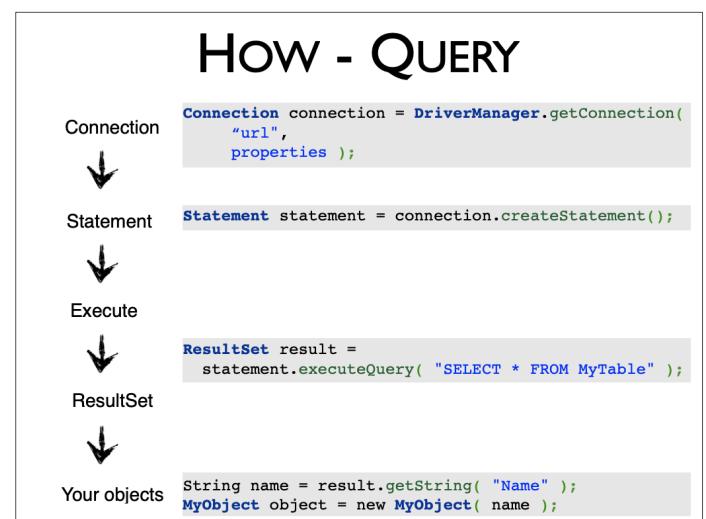
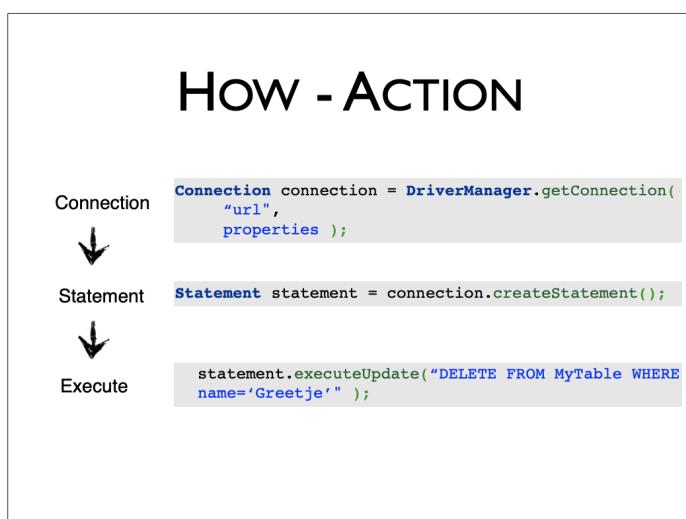
- Waar staat JDBC voor?
- Wat is het voordeel van JDBC?
- Gegeven wat code uit een databaseklasse: leg uit wat deze regels betekenen



Waarom JDBC?

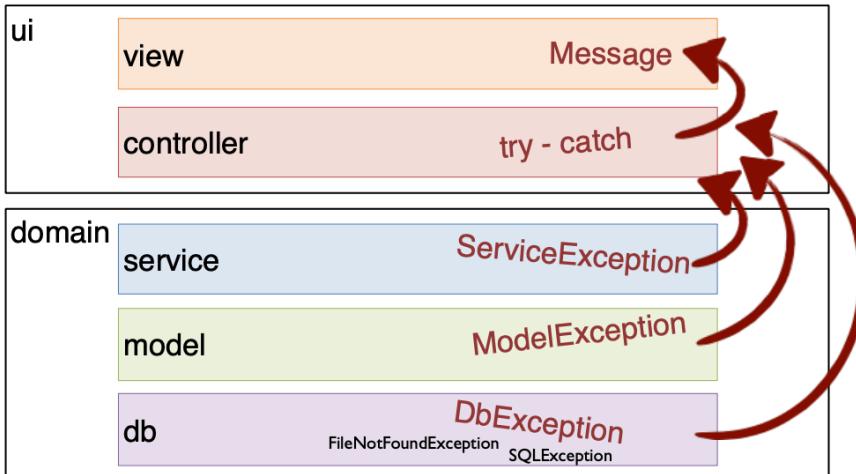
- "Write Once, Run Anywhere"
- Communicatie met relationele databanken (postgreSQL, Java DB, MySQL)
- Communicatie met spreadsheets

Hoe?



Exception Handling

EXCEPTION HANDLING



CUSTOM EXCEPTIONS

Tells me where it went wrong

```
package db
public class DbException extends RuntimeException {
    public DbException() {
        super();
    }
    public DbException(String message, Throwable exception) {
        super(message, exception);
    }
    public DbException(String message) {
        super(message);
    }
    public DbException(Throwable exception) {
        super(exception);
    }
}
```

Unchecked

you can pass the original exception

Vang de exceptions die de code gooit met een try/catch blok en gooi dan een zelfgemaakte runtime exception op.

Les 3

Database Design

Examenvragen

- Wordt JDBC gebruikt in View, Model of Controller? (**model**)
- Als ik gebruik maak van try-with-resources, waar is de verbinding dan gesloten?
- Wanneer moet ik een foutpagina gebruiken?
- Wat maakt een foutpagina anders dan een normale JSP-pagina
- Hoe lang moet je je databaseverbinding openhouden voor een webapplicatie?
Waarom?
- (wil je het antwoord weten op deze boeiende, tot nadenken stemmende vragen?
Lees dan zeker verder)

ADD

```
public void add(Country country){  
    if(country == null){  
        throw new DbException("Nothing to add.");  
    }  
  
    //create insert query based on properties of country  
    String sql = "INSERT INTO country (name, capital, inhabitants, votes)"  
    + "VALUES ("'  
    + country.getName() + ", '" + country.getCapital() + "', "  
    + country.getNumberInhabitants() + ", " + country.getVotes() + ")";  
  
    try {  
        //use statement object to execute the action  
        Statement statement = connection.createStatement()  
        statement.executeUpdate(sql);  
    } catch (SQLException e) {  
        throw new DbException(e);  
    }  
}
```

open connection?

close connection?

Voorbeeld: in elke methode maken we een statement en voeren die uit.

Probleem: om een statement te maken, hebben we een connectie nodig. Waar maken we de connectie? Waar sluiten we de connectie?

Voor een webserver is 1 connectie te weinig. Stel je voor dat je 10000 gebruikers hebt, en elke aanvraag van elke gebruiker moet via die ene connectie.

Je moet dus in elke call naar de methode een connectie openen, en op het einde van die call de connectie weer sluiten.

Door gebruik te maken van try-with-resources wordt de verbinding vanzelf gesloten als het codeblok is uitgevoerd.

REQUEST SCOPE JAVA 8

TRY WITH RESOURCES

```
public String add(Country country) {  
    try {  
        Connection connection = DriverManager.getConnection(url, properties);  
        Statement statement = connection.createStatement()  
        {  
  
            String sql = "INSERT INTO country (name, capital, inhabitants, votes)"  
            + "VALUES ("'  
            + country.getName() + ", '" + country.getCapital() + "', "  
            + country.getNumberInhabitants() + ", " + country.getVotes() + ")";  
            statement.executeUpdate(sql);  
        } catch (SQLException e) {  
            throw new DbException(e.getMessage(), e);  
        }  
    }  
}
```

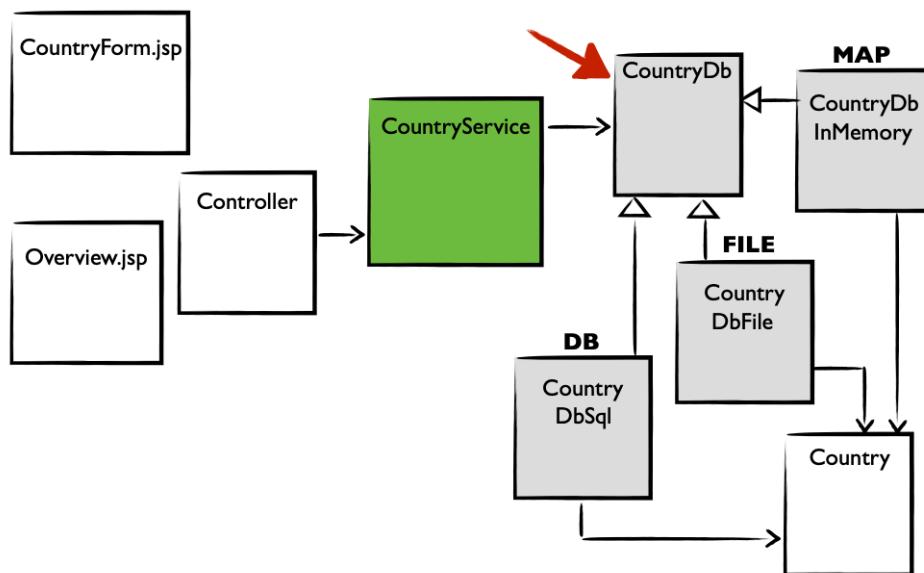
each resource is closed at the end of the try-catch

You do not have to close the connection and statement, it is done for you

REQUEST SCOPE

- 1 connection per method (thread)
- 100 simultaneous users:
 - how many connections ? → 100
 - time lost creating connections ? → a lot !
 - how many statements per connection ? → 1
 - scalable ? → Yes

→ dit is een performance issue, je kan beter doen aan connection pooling.



→ omdat je view en controllen best zo weinig mogelijk weten over de werking van de databank, kan je best de 'facade' design pattern toepassen.

→ 'strategy' kan ook nuttig zijn, zodat je makkelijk van soort databank kan wisselen

Error page

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<%@page isErrorPage="true"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Something wrong</title>
<link rel="stylesheet" href="css/sample.css">
</head>
<body>
<main>
<article>
<h1>Oh dear !</h1>
<p>You caused a ${pageContext.exception} on the server!</p>
<p>
<a href="Controller">Home</a>
</p>
</article>
</main>
</body>
</html>
```

I. ERROR.JSP

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://
java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" version="3.0">
<welcome-file-list>
<welcome-file>Controller</welcome-file>
</welcome-file-list>
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.jsp</location>
</error-page>
<error-page>
<error-code>404</error-code>
<location>/error404.jsp</location>
</error-page>
</web-app>
```

In case of an exception
navigate to error.jsp

2. WEB.XML

Ni hope da ik ga uitlegge wa een error pagina is he

- Je moet in de JSP aangeven dat het een errorpagina is
- in de web.xml moet je aangeven dat er naar de errorpagina moet genavigeerd worden als er een exception wordt opgegooid

De foutpagina vervangt niet het systeem dat je eerder hebt geschreven om een foutmelding te tonen wanneer een gebruiker een verkeerde invoer invoert.

Het doel is om fouten te tonen die niet de fout van de gebruiker zijn. Als er een fout in je code zit, of als er een probleem is met de database verbinding, de gebruiker kan het niet helpen, dus het heeft geen zin om op het formulier te blijven staan. In dat geval toon je de foutmeldingspagina.

Het betekent dat je moet differentiëren tussen de uitzonderingen die worden gegooid: sommige uitzonderingen vang je op en laat je zien als foutmelding, anderen vang je niet op zodat de foutpagina automatisch wordt getoond.

Les 4

Cross-site-scripting

Examenvragen

- Verklaar XSS. Gebruik een voorbeeld en beschrijf in detail de oplossing om XSS te vermijden.
- Waar staat XSS voor?

NOK

```
<form method="post" action="">
  <fieldset>
    <legend>Login</legend>
    <p>
      <label for="username">Username</label>
      <input type="text" id="username" name="username"
             value="${param.username}">
    </p>
  </fieldset>
  <p>
    <input type="submit" id="save" value="Log in">
  </p>
</form>
```

OK

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<form method="post" action="">
  <fieldset>
    <legend>Login</legend>
    <p>
      <label for="username">Username</label>
      <input type="text" id="username" name="username"
             value=">">
    </p>
  </fieldset>
  <p>
    <input type="submit" id="save" value="Log in">
  </p>
</form>
```

LOGIN.JSP

```
<input ... value="${param.username}" />  
  
<input ... value="Mieke" /> → no problem  
  
<input ... value="" /><script>alert("lol");</script>" />
```

↓
problem!

Dit spreekt voor zich.

Les 5

Configuratie

Examenvragen

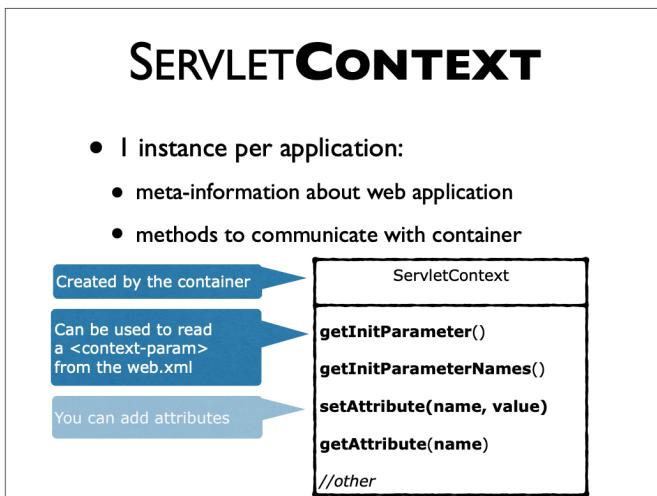
- Waar sla je properties op in een webapplicatie?
- Wat is de ServletContext?

```
<welcome-file-list>
    <welcome-file>Controller</welcome-file>
</welcome-file-list>
<context-param>
    <param-name>url</param-name>
    <param-value>jdbc:postgresql://databanken.ucll.be:51819/webontwerp</param-value>
</param-value>
</context-param>
<context-param>
    <param-name>user</param-name>
    <param-value>local_u0082726</param-value>
</context-param>
<context-param>
    <param-name>password</param-name>
    <param-value>MyVerySecretPassword</param-value>
</param-value>
</context-param>
<context-param>
    <param-name>currentSchema</param-name>
    <param-value>0082726</param-value>
</param-value>
</context-param>
<context-param>
    <param-name>ssl</param-name>
    <param-value>true</param-value>
</param-value>
</context-param>
<context-param>
    <param-name>sslfactory</param-name>
    <param-value>org.postgresql.ssl.NonValidatingFactory</param-value>
</param-value>
</context-param>
```

WEB.XML

how can you access them?

Je kan properties opslaan in de web.xml van je webproject.



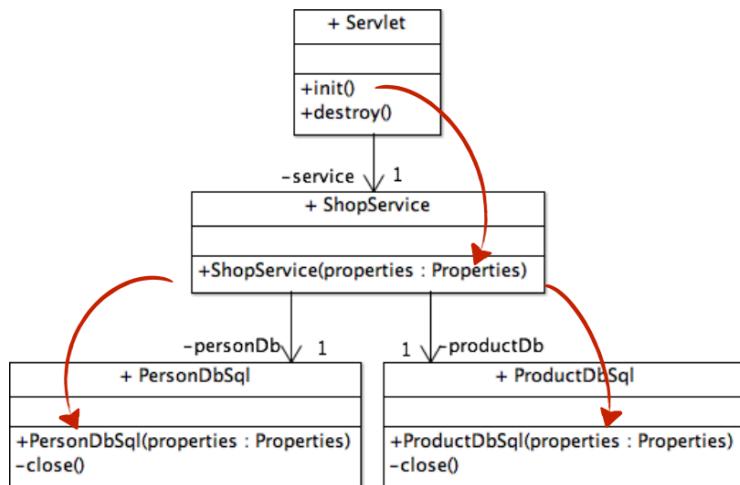
Wanneer een applicatie wordt gedeployed, maakt de container 1 ServletContext-object aan om meta-info op te slaan over de Toepassing.

De contextparameters van het web.xml worden automatisch opgeslagen in dit object.

Wanneer lezen we deze contextparameters in? - getServletContext()

We kunnen de methode getInitParameter gebruiken om de volgende eigenschap te lezen

CLASS DIAGRAM



Via `ServletContext` kent `Servlet` de waarden van de `Properties`. Die moeten op hun beurt doodgegooid worden naar de `Db` klassen

CREATE MODEL: FACADE

METHOD INIT() IN THE SERVLET

```

private ShopService service;

@Override
public void init() throws ServletException {
    super.init();
    ServletContext context = getServletContext();

    Properties properties = new Properties();
    properties.setProperty("user", context.getInitParameter("user"));
    properties.setProperty("password", context.getInitParameter("password"));
    properties.setProperty("ssl", context.getInitParameter("ssl"));
    properties.setProperty("sslfactory", context.getInitParameter("sslfactory"));
    properties.setProperty("sslmode", context.getInitParameter("sslmode"));
    properties.setProperty("url", context.getInitParameter("url"));

    service = new ShopService(properties);
}
  
```

A red curved arrow points from the `Properties` assignment in the code to the `create facade object with properties` annotation below.

`create facade object with properties`

Nu kun je in elk van de constructoren van de klassen die `Properties` nodig hebben een `Properties` object meegeven.

Oh ja dit nog

WHY IN INIT() METHOD AND NOT IN CONSTRUCTOR?

Because we have to wait for the **ServletContext-**
object:

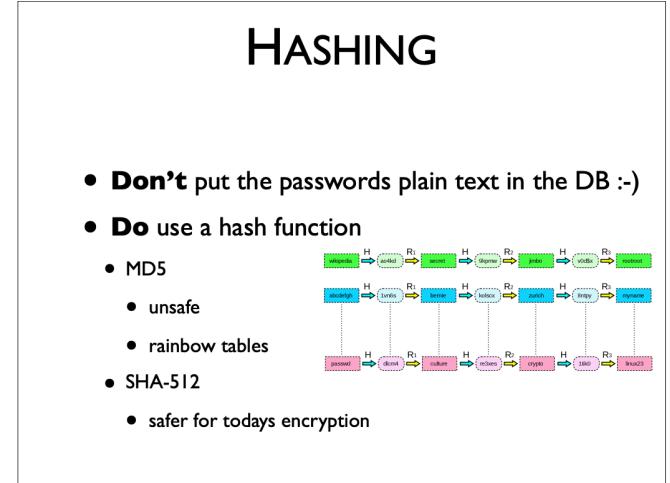
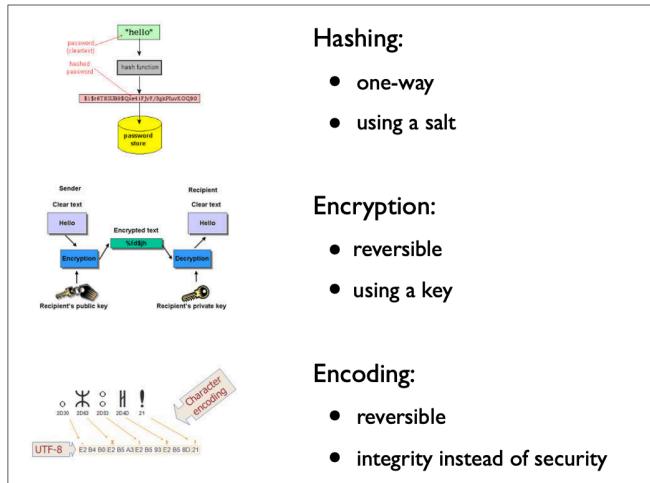
1. Container creates servlet
2. Container creates/has access to:
 - ServletConfig
 - ServletContext
3. Container calls init()



Hashing

Examenvragen

- Leg uit waarom we onze wachtwoorden moeten hashen?
- Wat is de reden om een SALT toe te voegen voordat we een wachtwoord hashen?



Zo doe je dat



Salt

USING A SALT

- appending or prepending a random string, called a **salt**, to the password before hashing
 - sha1(salt + password)
 - save both salt and hash in the DB

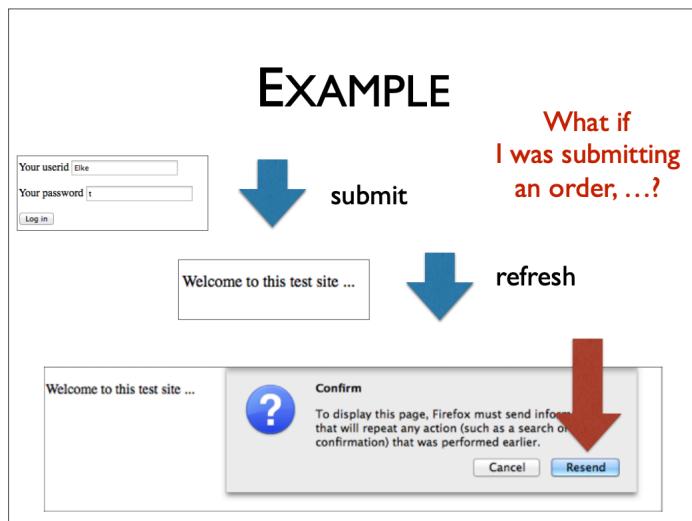
Dit is dus veiliger dan alleen een hash. Je moet weten wat het is, maar je hoeft het niet toe te passen.

Les 6

Post/Redirect/Get

Examenvragen

- Leg uit wat het probleem is dat Post/Redirect/Get oplost?
- Leg de figuren uit in deze slides.
- Hoe moet je Post/Redirect/Get implementeren?
- Wat is het verschil tussen forward en sendRedirect?



Als je deze pagina ververst

- Je kunt de POST opnieuw sturen
- Als dit een bevestiging is van een bestelling die je bent aan het plaatsen...
- je zult het twee keer of meer bestellen...

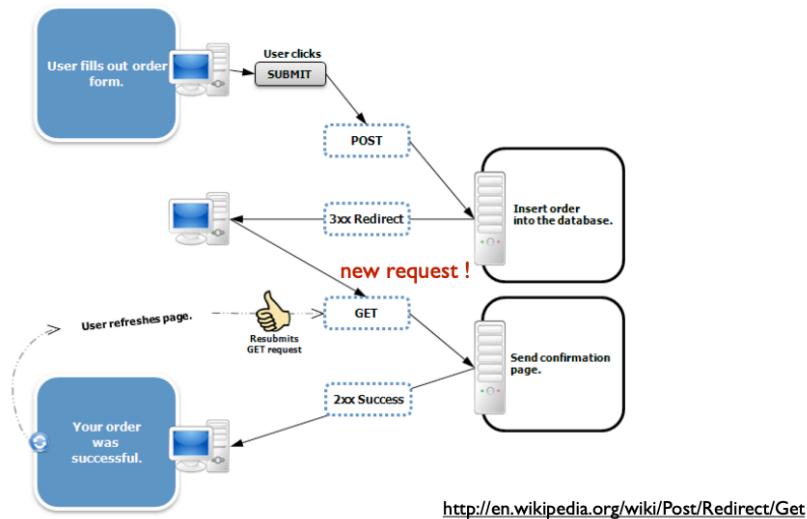
SOLUTION POST/REDIRECT/GET

After submitting a form, ...

- Do NOT **forward** the **request**
- But **redirect** the **response**
- = PRG

```
response.sendRedirect("Controller?action=welcome");
ipv request.getRequestDispatcher("welcome.jsp").forward(request, response)
```

SOLUTION POST/REDIRECT/GET



DIFFERENCES?

request.forward()

- The forward() method is executed on the server side.
- It can be used within server.
- information in the request is still available
- ...

response.sendRedirect()

- The sendRedirect() method is executed on the client side.
- It can be used within and outside the server.
- information in the request is lost
- ...

Front controller continued

Examenvragen:

- Leg het Front Controller patroon uit.
- Verschillende stukjes code kunnen uitleggen van de verschillende opties die er hier in de slides worden getoond.
- Voor- en nadelen van de verschillende opties kunnen uitleggen. (factory met reflection, factory met if-statements, factory met config file)
- Option 4 moet je NIET kunnen implementeren! (Requestmapping annotations)

Essentie van deze les

HANDLERFACTORY

```
public class HandlerFactory {  
    private Map<String, RequestHandler> handlers = new HashMap<>();  
  
    public HandlerFactory(Properties handlerNames, ShopService model) {  
        for(Object key : handlerNames.keySet()) {  
            RequestHandler handler = null;  
            String handlerName = handlerNames.getProperty((String) key);  
            try {  
                Class<?> handlerClass = Class.forName(handlerName);  
                Object handlerObject = handlerClass.newInstance();  
                handler = (RequestHandler) handlerObject;  
            } catch (ClassNotFoundException e) {  
                ...  
            }  
            handler.setModel(model);  
            handlers.put((String)key, handler);  
        }  
    }  
  
    public RequestHandler getRequestHandler(String key) {  
        return handlers.get(key);  
    }  
}
```

Je maakt een HandlerFactory, die voor elk request een controller aanmaakt die dat request afhandelt. Welke controller er wordt aangemaakt, wordt beslist via de string die je meegeeft aan de factory. Je moet dus voor elk verschillend request een klasse maken die overerft van RequestHandler.

Authenticatie

Examenvragen:

- Wat wordt bedoeld met “Authentication”?
- Geef de nodige stappen in model, view en controller

Authentication is the process of verifying the identity of a person or device.

Dat wordt dus bedoeld met authenticatie.

View:

- Login form

Model:

- Model checkt wachtwoord
- Db checkt user

Controller:

- De authenticatie verwerken

Les 7

Authorisatie

Examenvragen:

- What is the difference between authentication and authorization?

AUTHENTICATION

AUTHORIZATION

- Check whether a person that logs in can be authenticated.
- means that it uses the right credentials (username and password for example)



- Check whether a person has the right role to access particular pages of the web application, if this person authorized to see this page
- means that an administrator can do more on the web application than a normal user for example

Bedankt Greetje

Wat moet ik doen om autorisatie werkende te maken in mijn project?

1. Rollen geven aan mensen
2. Inloggen
3. Toegang verlenen aan bepaalde rollen
4. Toegan tot paginas verlenen aan bepaalde rollen

Hoe?

1. Maak een klasse 'utility' aan die een statische methode bevat die checkt of een gebruiker een van de rollen heeft die jij aan de methode mee hebt gegeven.
2. Laat deze klasse een exception opgooien als de persoon geen van deze rollen heeft.
3. Zet de methode om de rol te checken in de requesthandlers waar dit nodig is.
4. Zet in de Servlet het handleRequest blok in een try/catch, waar hij de bovengenoemde exception opvangt en indien nodig een actie onderneemt.

SHOW PAGES TO SPECIFIED ROLES

```
// header.jsp

<nav>
    <ul>
        <li><a href="Controller">Home</a></li>
        <li><a href="Controller?action=everyone">Everyone</a></li>

        <c:if test="${user.role=='ADMIN' }">
            <li><a href="Controller?action=admin">Admin</a></li>
        </c:if>

        <c:if test="${user.role=='ADMIN' || user.role=='CUSTOMER' }">
            <li><a href="Controller?action=allRoles">All Roles</a></li>
        </c:if>
    </ul>
</nav>
```

The screenshots show three levels of navigation:

- Home:** Shows links for "Home" and "Everyone".
- Admin:** Shows links for "Home", "Everyone", and "Admin".
- All Roles:** Shows links for "Home", "Everyone", "Admin", and "All Roles".

In the "Home" and "Admin" versions, there is a "Please log in." message. In the "All Roles" version, there is a "Welcome, CustoMer" message at the top.

← Je kan dit ook
Best wel cool he

Les 8-10

Javascript

Examenvragen:

- webapplicatie maken met JavaScript
- elementen uit de DOM aanspreken en wijzigen
- elementen verwijderen en toevoegen
- events definiëren
- events toevoegen aan een formulier

Maak de oefeningen man

Hier zijn opl

<https://github.com/martijnmeeldijk/web3-javascript-by-kingmarti>