

Frameworks voor serverapplicaties Industrieel Ingenieur

Veerle Ongenae

Bachelor in de Industriële Wetenschappen: Informatica
Academiejaar 2025–2026



Frameworks voor serverapplicaties (E761038)

Cursusomvang *(nominale waarden; effectieve waarden kunnen verschillen per opleiding)*

Studiepunten 6.0

Studietijd 180 u

Aanbodsessies en werkvormen in academiejaar 2025-2026

A (semester 1)

Nederlands

Gent

werkcollege

groepswerk

hoorcollege

Lesgevers in academiejaar 2025-2026

Ongenae, Veerle

TW05

Verantwoordelijk lesgever

Aangeboden in onderstaande opleidingen in 2025-2026

stptn

aanbodsessie

[Bachelor of Science in de industriële wetenschappen\(afstudeerrichting informatica\)](#)

6

A

[Voorbereidingsprogramma tot Master of Science in de industriële wetenschappen:
informatica](#)

6

A

Onderwijsstalen

Nederlands

Trefwoorden

Webapplicaties, .NET-platform, J2EE, MVC, REST, AJAX, nodeJS, HTTP, websockets, server-sent events, Databanktoegang vanuit applicaties, ADO.NET, JDBC, API contracts, ORM, Beveiliging webapplicaties, Computerwetenschappen (P170), Informatica (P175), Computertechnologie (T120)

Situering

De doelstelling van dit opleidingsonderdeel is om de basisprincipes van de architectuur en de werking van de backend van een mobiele of webapplicaties te begrijpen. De studenten leren webapplicaties en webservices ontwikkelen die gegevens manipuleren in een databank.

Inhoud

- Communicatie met de server
 - De werking van het HTTP-protocol en de structuur van HTTP-berichten
 - Afhandelen HTTP-berichten
 - Tweewegscommunicatie via websockets
 - Events op de server: server-sent events
 - Interface definities: specificatie via contracten (OAS) en schema's
 - REST API en services: concepten en ontwikkeling (synchroon en asynchroon)
 - Beveiligingsaspecten bij webapplicaties: OAuth, SQL-injectie, XSS, CSRF
 - Architectuur van webapplicaties: meerlagen model, MVC server side: principes o. a. request routing en handling dynamisch genereren van webpagina's, dependency injection
 - ORM: concepten, basisprincipes en werking van ORM-frameworks
 - Crosscutting concerns zoals logging, authenticatie, monitoring: principes en realisatie
- Raamwerken: .NET focus op ASP.NET MVC en Web API, Spring: JPA, REST, gateways, nodeJS

Begincompetenties

- Op een doorgedreven niveau kunnen objectgeoriënteerd programmeren en ontwerpen in Java en C#
- Basiskennis over gegevensbanken

- De werking en principes van user interfaces, die gebruik maken van een backend, beheersen

Eindcompetenties

- 1 Een interface contract voor services ontwerpen, uitschrijven en valideren.
- 2 REST-webservices te ontwikkelen m.b.v verschillende frameworks
- 3 Een webapplicatie met dynamische gegenereerde pagina's ontwikkelen.
- 4 De architectuur en de basisprincipes van een backend kunnen toelichten en een overzicht geven van de hierbij gebruikte protocollen
- 5 Gebruik maken van dependency injection en annotaties om makkelijk onderhoudbare, testbare, ... backends te realiseren
- 6 Het MVC-principe uitleggen aan de hand van een voorbeeld en implementeren in een webapplicatie (serverside)
- 7 De basisprincipes van ORM toelichten en illustreren met een voorbeeld en hiermee een datalaag ontwerpen en implementeren.

Creditcontractvoorraarde

Toelating tot dit opleidingsonderdeel via creditcontract is mogelijk na gunstige beoordeling van de competenties

Examencontractvoorraarde

Dit opleidingsonderdeel kan niet via examencontract gevuld worden

Didactische werkvormen

Groepswerk, Werkcollege, Hoorcollege

Toelichtingen bij de didactische werkvormen

- Hoorcollege (24u)
- Werkcollege (36u): PC-klassoefeningen (zelfstandig werk aan een individuele PC) en groepswerk (uitgebreid computerlabo)

Studiemateriaal

Type: Slides

Naam: Kennisclips, slides, voorbeeldprogramma's en oefeningen met oplossingen zijn beschikbaar via het elektronisch leerplatform.
Richtprijs: € 7
Optioneel: ja
Taal: Nederlands
Beschikbaar op Ufora : Ja
Online beschikbaar : Nee
Beschikbaar in de bibliotheek : Nee
Beschikbaar via studentenvereniging : Ja

Type: Software

Naam: Visual Studio .NET
Richtprijs: Gratis of betaald door opleiding
Optioneel: nee
Beschikbaar op Athena : Nee
Online beschikbaar : Ja
Beschikbaar in de bibliotheek : Nee
Beschikbaar via studentenvereniging : Nee
Gebruik en levensduur binnen het opleidingsonderdeel : regelmatig
Gebruik en levensduur binnen de opleiding : regelmatig
Gebruik en levensduur na de opleiding : niet

Type: Software

Naam: IntelliJ IDEA Ultimate Edition
Richtprijs: Gratis of betaald door opleiding
Optioneel: nee
Beschikbaar op Athena : Nee
Online beschikbaar : Ja
Beschikbaar in de bibliotheek : Nee
Beschikbaar via studentenvereniging : Nee
Gebruik en levensduur binnen het opleidingsonderdeel : regelmatig
Gebruik en levensduur binnen de opleiding : intensief
Gebruik en levensduur na de opleiding : niet

Referenties

- "Programming Web Applications with Node, Express and Pug", Jörg Krause, Apress, 2017
- "Securing PHP Apps", Ben Edmunds, Apress, 2016
- "Professional ADO.NET 3.5 with LINQ and the Entity Framework", Roger Jennings, Wrox, 2009
- "XML in a Nutshell - A Desktop Quick Reference", Harold Elliotte Rusty, Means W Scott, O'Reilly, 2001
- Create a web API with ASP.NET Core, <https://docs.microsoft.com/en-1us/aspnet/core/tutorials/first-web-api6>
- ASP.NET documentation, <https://docs.microsoft.com/en-us/aspnet/core/>
- Building a RESTful Web Service, <https://spring.io/guides/gs/rest-service>
- Accessing Relational Data using JDBC with Spring, <https://spring.io/guides/gs relational-data-access/>
- Accessing Data with JPA, <https://spring.io/guides/gs/accessing-data-jpa/>
- Open API specification, <https://oai.github.io/Documentation/>
- OAuth 2.0, <https://oauth.net/2/>
- Top 10 Web Application Security Risks, <https://owasp.org/www-project-top-ten/>
- The WebSocket Protocol, <https://tools.ietf.org/html/rfc6455>

Vakinhoudelijke studiebegeleiding

Lesgevers zijn ter beschikking voor extra uitleg tijdens de labo's, voor of na de theorielessen en eventueel op andere ogenblikken na afspraak.

Evaluatiemomenten

periodegebonden en niet-periodegebonden evaluatie

Evaluatievormen bij periodegebonden evaluatie in de eerste examenperiode

Vaardigheidstest, Schriftelijke evaluatie

Evaluatievormen bij periodegebonden evaluatie in de tweede examenperiode

Vaardigheidstest, Schriftelijke evaluatie

Evaluatievormen bij niet-periodegebonden evaluatie

Vaardigheidstest, Werkstuk

Tweede examenkans in geval van niet-periodegebonden evaluatie

Examen in de tweede examenperiode is niet mogelijk

Toelichtingen bij de evaluatievormen

Tijdens het werkcollege worden één of meerdere testen aan de computer georganiseerd. De score op de NPE is de combinatie van één of meerdere testen over de labo's (3/4) en de groepsopdracht (1/4).

Het examen bestaat uit een schriftelijke deel en een deel dat bestaat uit oefeningen op de computer. De oefeningen op de computer zijn open boek.

Eindscoreberekening

Examen: 60% (schriftelijk examen en computeroefeningen)

Labo's: 40% (één of meerdere testen en groepsopdracht)

In de tweede examenperiode: score = maximum(E; 40% L + 60% E), waarbij L de score van het labo (NPE) is en E de nieuwe score van het examen uit tweede zit.

Inhoudsopgave

| | |
|--|-----------|
| 1 REST web services | 1 |
| 1.1 Herhaling HTTP | 2 |
| 1.2 REST web services | 4 |
| 1.3 REST web services in Java | 9 |
| 1.4 Webservices testen | 15 |
| 1.5 Webservices beveiligen | 19 |
| 1.6 Reactive webservices in Java | 23 |
| 1.7 REST web services in C# | 29 |
| 1.8 OpenAPI specificatie en Swagger | 35 |
| 2 Object Relational Mapping | 41 |
| 2.1 The object-relational impedance mismatch | 41 |
| 2.2 ORM-principes | 43 |
| 2.3 JPA en Hibernate | 46 |
| 2.4 JPA in Spring | 47 |
| 2.5 Objecten afbeelden | 49 |
| 2.5.1 Eigenschappen | 49 |
| 2.5.2 Overerving | 54 |
| 2.5.3 Relaties | 59 |
| 2.6 Overerving in ORM | 64 |
| 2.7 Value-objecten in ORM | 66 |
| 2.8 Relaties in ORM | 69 |

| | |
|--|-----------|
| 2.9 Werking ORM | 74 |
| 2.10 Zoekopdrachten in ORM | 79 |
| 3 NodeJS | 81 |
| 4 Websockets | 93 |
| 5 Java Database Connectivity (JDBC) | 97 |
| 5.1 Wat is JDBC? | 97 |
| 5.1.1 Verschillende versies | 97 |
| 5.2 JDBC-drivers | 98 |
| 5.2.1 Verschillende types JDBC-drivers | 98 |
| 5.2.2 Laden van de driver | 100 |
| 5.2.3 Verschillende versies van de JDBC API | 100 |
| 5.3 Verbindingen met een gegevensbank | 101 |
| 5.4 SQL-opdrachten | 102 |
| 5.4.1 DDL-opdrachten, insert, delete en update | 102 |
| 5.4.2 Zoekopdrachten | 103 |
| 5.5 Prepared Statements | 105 |
| 5.5.1 Aanmaken van een <i>prepared statement</i> | 105 |
| 5.5.2 Parameters toekennen | 106 |
| 5.5.3 Gegevensconversie | 106 |
| 5.5.4 SQL-injectie | 107 |
| 5.6 Callable Statements | 108 |
| 5.6.1 Aanmaken <i>callable statement</i> | 108 |
| 5.6.2 Invoerparameters | 109 |
| 5.6.3 Uitvoerparameters | 110 |
| 5.6.4 Invoer- en uitvoerparameters | 111 |
| 5.7 Transacties | 112 |
| 5.7.1 De JNDI API | 113 |

| | |
|--|------------|
| INHOUDSOPGAVE | ix |
| 5.7.2 Een <i>DataSource</i> -object aanmaken | 114 |
| 5.7.3 <i>Connection Pooling</i> | 115 |
| 5.7.4 <i>Gedistribueerde transacties</i> | 116 |
| 5.8 Slides JDBC | 117 |
| 5.9 ORM versus JDBC | 129 |
| 6 ADO.NET: een inleiding | 131 |
| 6.1 Wat is ADO.NET? | 131 |
| 6.1.1 DataSet | 131 |
| 6.1.2 .NET Data Provider | 132 |
| 6.2 Een verbinding maken | 134 |
| 6.3 Opdrachten uitvoeren | 136 |
| 6.3.1 Het commando-object | 136 |
| 6.3.2 Zoekopdrachten | 137 |
| 6.3.3 Gegevens wijzigen, toevoegen of verwijderen | 138 |
| 6.3.4 Parameters | 138 |
| 6.4 DataSet en DataAdapter | 140 |
| 6.4.1 DataSet | 140 |
| 6.4.2 DataAdapter | 141 |
| 6.4.3 <i>DataReader</i> versus <i>DataSet</i> | 143 |
| 6.5 Transacties | 144 |
| 7 ASP.NET Core MVC | 145 |
| 7.1 Basisconcepten | 146 |
| 8 Webapplicaties | 165 |
| 8.1 Authenticatie | 166 |
| 8.2 Beveiligingsrisico's | 173 |
| 8.3 Verschillende types webapps | 176 |

Hoofdstuk 1

REST web services

1.1. HERHALING HTTP

1.1 Herhaling HTTP

Herhaling HTTP | Veerle Ongenaee

1

HOOFDSTUK 1. REST WEB SERVICES

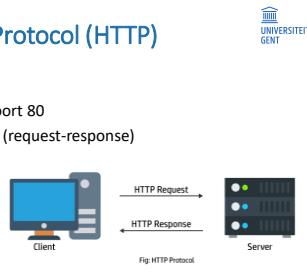
Wat is HTTP?

- <https://padlet.com/vongenae/HTTP>



Hypertext Transfer Protocol (HTTP)

- Client-Server protocol
- Transportprotocoll: TCP op poort 80
- Aanvraag-antwoord protocol (request-response)
- Web
- Verschillende versies
 - HTTP 1.0
 - HTTP 1.1
 - HTTP/2
 - HTTP/3



HTTP-aanvraag (request)

```

Erste headerlijn
POST /localhost:8080/httpv8Servlets/toonwaarden_HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/httpv8Servlets/
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cookie: JSESSIONID=42061473951661
Content-Length: 41
naam=veerle+0K27haese&adres=Kerkstraat=10
  
```

Volgende headerlijnen

Lege lijn

Body (optioneel, afhankelijk van de aanvraagmethode)

Aanvraag-methodes

| | |
|--------|---------|
| GET | HEAD |
| POST | PUT |
| DELETE | OPTIONS |
| ... | ... |

HTTP-antwoord (response)

```

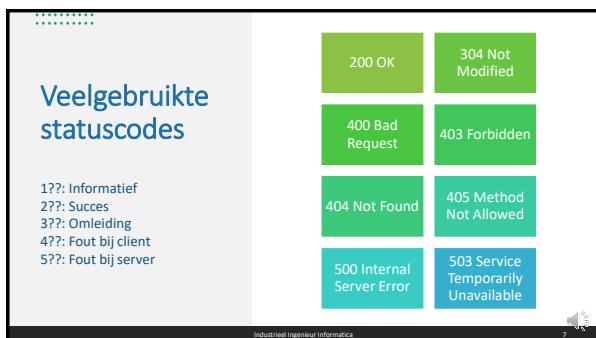
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition 4.1
Date: Fri, 30 Sep 2016 10:01:01 GMT
Content-Type: text/html
Content-Language: nl
Content-Length: 562
Last-Modified: Fri, 06 Oct 2014 08:15:41 GMT
ETag: "w"/"562-1412583341652"
Content-Security-Policy: default-src 'self'
Content-Security-Policy-Report-Only: 'self'
Content-Type: text/html; charset=UTF-8
Content-Language: nl
Content-Length: 562
  
```

Eerste headerlijn

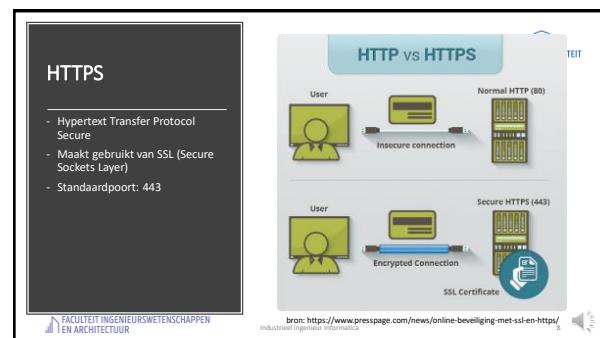
Volgende headerlijnen

Lege lijn

Body (optioneel, bepaald door MIME-type)

HOOFDSTUK 1. REST WEB SERVICES1.1. HERHALING HTTP

7



8

1.2. REST WEB SERVICESHOOFDSTUK 1. REST WEB SERVICES

1

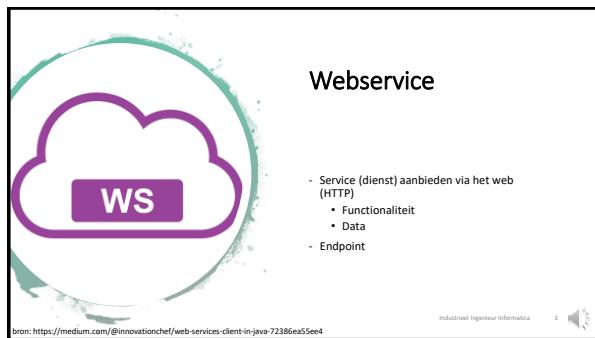
Wat is een REST API?

- <https://www.youtube.com/watch?v=SlwpgD8n3d0>

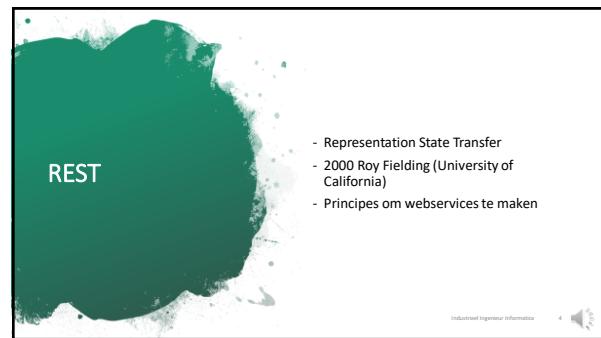
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

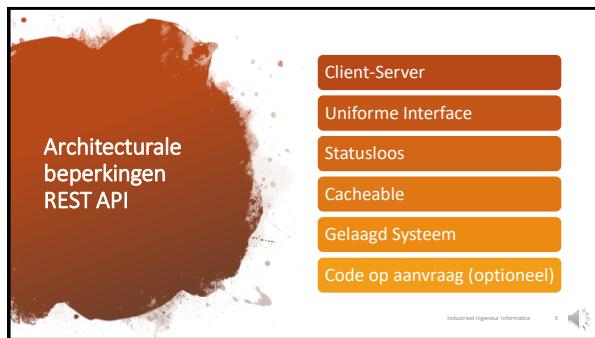
2



3



4



5



6

HOOFDSTUK 1. REST WEB SERVICES**1.2. REST WEB SERVICES**

Uniforme interface

- Uniforme manier om te communiceren met de server
 - Onafhankelijk van het type client
- Richtlijnen
 - Bron-georiënteerd
 - Gebruik de HTTP-methodes expliciet
 - Links om makkelijk zelf bronnen te ontdekken

Industriel Ingenieur Informatica

7

Bron-georiënteerd

- Webservice
 - Bron
 - Geïdentificeerd door URI
 - Representatie

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

8

Gebruik de HTTP-methodes expliciet

The diagram illustrates the mapping of CRUD operations to HTTP methods. It shows four vertical arrows pointing from the letters C, R, U, D to their corresponding actions: Create, Read, Update, and Delete. To the right, a bracket groups these actions under the heading 'HTTP Methods'. Within this bracket, the methods POST, GET, PUT, and DELETE are listed in a vertical column. The word 'edureka!' is written vertically next to the bracket.

bron: <https://www.edureka.co/blog/what-is-rest-api/>

Industriel Ingenieur Informatica

9

Gebruik HTTP-methodes expliciet

- HTTP-methodes corresponderen met CRUD (create, read, update, delete)
 - POST: een bron toevoegen op de server
 - GET: een bron ophalen van server
 - Geen neveneffecten (veranderingen op server)
 - Idempotent
 - PUT: een bron updaten op server (idempotent)
 - DELETE: een bron verwijderen (idempotent)

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

10

Voorbeeld: GET

- Gebruiker ophalen

```
GET /users/3 HTTP/1.1
Host: myserver
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
  "id": 3,
  "name": "Robert",
  "color": "red"
}
```

Industriel Ingenieur Informatica

11

Voorbeeld POST

- Gebruiker toevoegen

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/json
```

```
{
  "name": "John",
  "color": "green"
}
```

```
HTTP/1.1 201 Created
Location: http://myserver/users/10
```

URL nieuwe bron

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur

12

1.2. REST WEB SERVICES

HOOFDSTUK 1. REST WEB SERVICES

Voorbeeld PUT

- Gebruiker vervangen

```
PUT /users/10 HTTP/1.1
Host: myserver
Content-type: application/json

{
  "name": "John",
  "color": "blue"
}

HTTP/1.1 204 No Content
...
```



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur

13



13

Voorbeeld DELETE

- Gebruiker verwijderen

```
DELETE /users/3 HTTP/1.1
Host: myserver
```

```
HTTP/1.1 204 No Content
...
```



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur

14



14

URI's bij REST

- URI
 - Zelfstandig naamwoord
 - Bron
 - Waarop moet actie toegepast worden
- HTTP-methode
 - Actie
 - Werkwoord

POST `/users` **HTTP/1.1**

↳ Bron? Waarop actie toepassen?

↳ Welke actie? Wat doen?



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

15



15

URI's ~ directory-structuur

- URI's hebben een structuur analoog aan directory's
- Intuitieve URI
 - Eenvoudig
 - Voorspelbaar
 - Verstaanbaar
- Structuur zoals een directory
 - Hiërarchisch
 - Boomstructuur met één startpunt

```
http://www.myservice.org/discussions/topics/{topic}
http://www.myservice.org/discussions/{year}/{day}/{month}/{topic}
```



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

16



16

URI's: extra richtlijnen

- <https://restfulapi.net/resource-naming/>
- Gebruik een zelfstandig naamwoord in het meervoud
- Verberg extensies (.jsp, .php, .asp, ...)
- URI's blijven bij veranderende technologie
- Gebruik kleine letters
- Vervang spaties door -
- Gebruik "querystrings" om te filteren, te sorteren, ...

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices?region=USA
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ&sort=installation-date
```

- Vermijd "404 Not Found"
 - Statisch (onveranderlijke) links → bladwijzers



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

17



17

Berichten

- Representatie bron
- Wissel XML of JSON uit
- HTTP-body
 - Eenvoudig
 - Leesbaar
 - XML of JSON



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

18

HOOFDSTUK 1. REST WEB SERVICES**1.2. REST WEB SERVICES**

Statusloos

- Wees Statusloos
 - Elke aanvraag staat op zichzelf
 - Server houdt geen informatie
 - > Gebruiker
 - > Application context
 - Client moet status bijhouden
- Voordelen
 - Aanvraag doorgeven aan andere server(s)
 - Load-balancing → verhogen schaalbaarheid
 - Failover → verhogen betrouwbaarheid

Industriel Ingenieur Informatica

bron: https://codeburst.io/load-balancers-an-analogy-cc64d9430db07g=75f92b6189d4

19

Cacheable

- Antwoord bevat
 - * Cacheable?
 - * Hoelang?
- Verbetert performantie

Industriel Ingenieur Informatica

20

20

Gelaagd systeem

- De client weet niet met welke laag hij geconnecteerd is

Industriel Ingenieur Informatica

bron: https://www.chakray.com/rest-web-services-y-apis-design-restful/

21

Code op aanvraag

- Code on demand
- Optioneel
- De inhoud van het bericht mag ook code zijn (bv. javascript)

Industriel Ingenieur Informatica

22

22

Architecturale beperkingen REST API

- Client-Server
- Uniforme Interface
- Statusloos
- Cacheable
- Gelaagd Systeem
- Code op aanvraag (optioneel)

Industriel Ingenieur Informatica

23

23

Glory of REST

Glory of REST

Level 0: The Swamp of POX

Level 1: Resources

Level 2: HTTP Verbs

Level 3: Hypermedia Controls

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

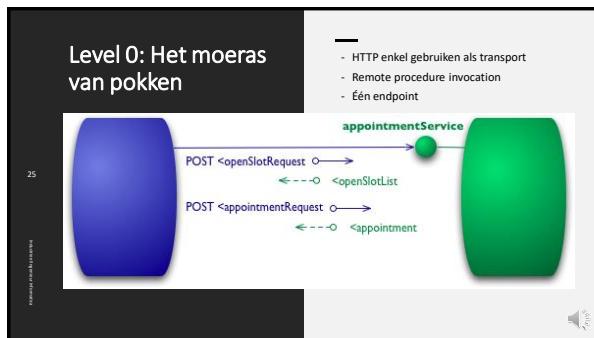
bron: https://martinfowler.com/articles/richardsonMaturityModel.html

Industriel Ingenieur Informatica

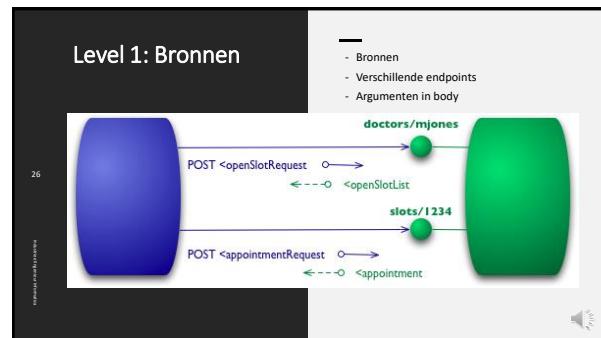
24

1.2. REST WEB SERVICES

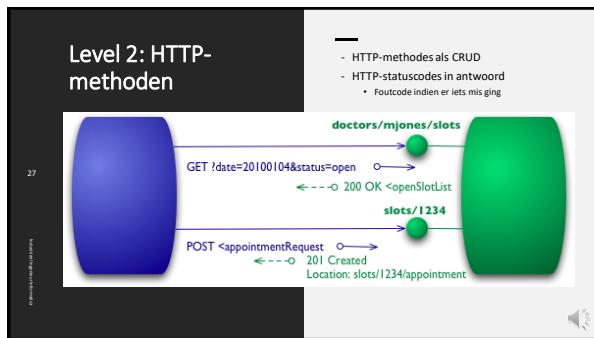
HOOFDSTUK 1. REST WEB SERVICES



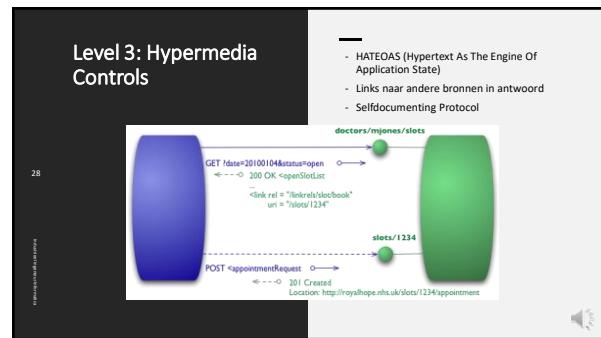
25



26



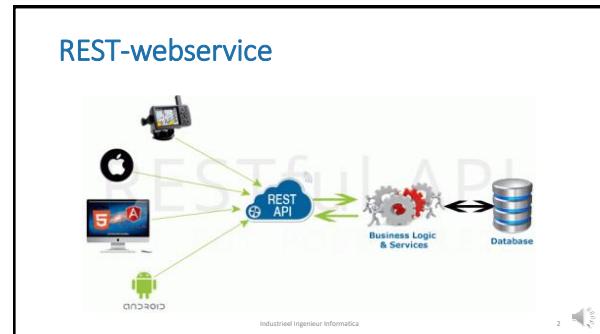
27



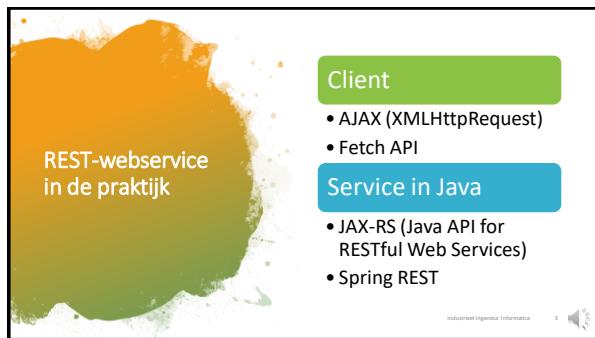
28

HOOFDSTUK 1. REST WEB SERVICES1.3. REST WEB SERVICES IN JAVA

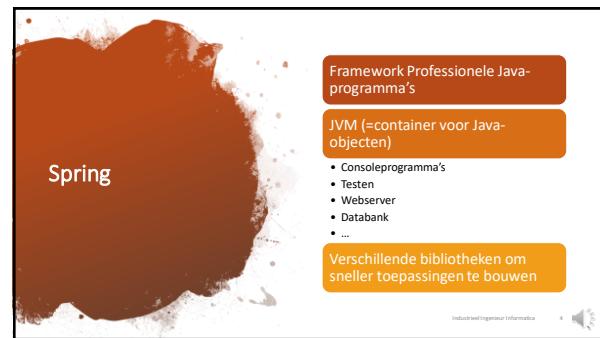
1



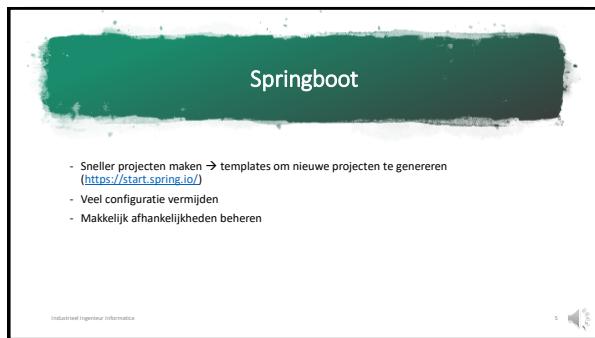
2



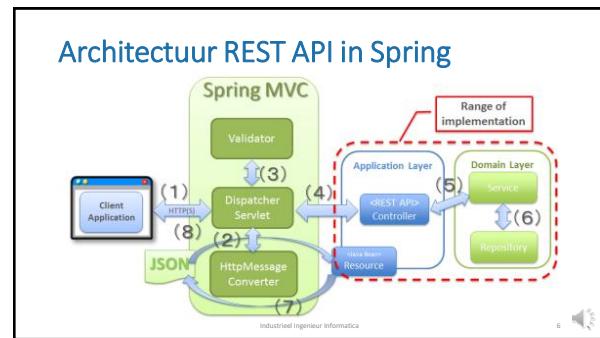
3



4



5



6

1.3. REST WEB SERVICES IN JAVA

HOOFDSTUK 1. REST WEB SERVICES

Architectuur REST API in Spring

1. Spring-framework ontvangt een HTTP-aanvraag en bepaalt voor welke methode van welke controller opgeroepen moet worden
2. (Optioneel) JSON in de body converteren naar een specifiek object
3. (Optioneel) validatie voor de inputwaarden van het object
4. Oproepen methode van controller met object als parameter Implementatie
5. In de controller worden van de achterliggende logica opgeroepen
6. De logica interageert met de datalaag
7. Het resultaat van de methode van de controller wordt geserialiseerd naar JSON (of XML of ...)
8. Het HTTP-antwoord wordt aangemaakt. De (eventuele) de body bevat het JSON-antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique



Spring framework

7

Implementatie

- HTTP-aanvragen afbeelden op methodes RestController
 - Berichten converteren
 - Data wegschrijven naar body HTTP-bericht
- Parameters in pad
 - Data uit body HTTP-bericht ophalen
 - Antwoordstatus bepalen
 - URI's in antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique



8

Voorbeeld REST - GET

- URL: localhost:8080/agenda/
- Body HTTP-antwoord

```
{
  "id": 3,
  "onderwerp": "Bjork",
  "startDatum": "2019-11-23T18:00:00+0000",
  "eindDatum": "2019-11-23T20:00:00+0000",
  "locatie": "Merk, Brussel"
},
{
  "id": 2,
  "onderwerp": "File",
  "startDatum": "2019-11-20T19:00:00+0000+0000",
  "eindDatum": "2019-11-20T20:00:00+0000+0000",
  "locatie": "Gent"
},
{
  "id": 1,
  "onderwerp": "Bruno",
  "startDatum": "2019-11-23T18:00:00+0000+0000",
  "eindDatum": "2019-11-23T21:00:00+0000+0000",
  "locatie": "Lokeren"
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique, UGent

9

Voorbeeld REST – GET - RestController

- localhost:8080/agenda/

```
import org.springframework.web.bind.annotation.*;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping(value = "/agenda", method = RequestMethod.GET)
public class AgendaController {
    private Agenda agenda;
    public AgendaController(Agenda agenda) { this.agenda = agenda; }

    @RequestMapping(method = RequestMethod.GET)
    @ResponseBody
    public Collection<AgendaItem> agendaVanAanstaaf() {
        return agenda.getAgendaItems(new Date());
    }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique, UGent



10

Dependency Injection

- Afhankelijkheden injecteren
- Contexts and Dependency Injection → J2EE
- Spring
 - @Component
 - Framework zoekt automatisch parameters constructor

```
@RestController
@RequestMapping("agenda")
public class AgendaController {
    private Agenda agenda;

    public AgendaController(Agenda agenda) { this.agenda = agenda; }
```

```
Component
public class Agenda {
```

- Managed components:
- Beheerde componenten
 - Kunnen geïnjecteerd worden
 - Meer specifieke componenten
 - @Service (logicaal)
 - @Repository (datalaag, persistentie)

Constructor injection

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique



11

Voorbeeld REST – data - Agenda

```
Komponent
public class Agenda {
    private final Map<Integer, AgendaItem> agendaItems;
    int huidigeId = 1;
    private final DateFormat dateFormat;

    public Agenda() { ... }

    private synchronized int getHuidigeId() { return huidigeId++; }

    public SortedSet<AgendaItem> getAgendaItems(Date startDate) { ... }

    public AgendaItem getAgendaItem(int id) { return agendaItems.get(id); }

    public synchronized AgendaItem addAgendaItem(AgendaItem item) { ... }

    public synchronized boolean veranderAgendaItem(int id, AgendaItem item) { ... }

    public synchronized boolean deleteAgendaItem(int id) { ... }

    private void opvolgen() { ... }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

12

10

HOOFDSTUK 1. REST WEB SERVICES

1.3. REST WEB SERVICES IN JAVA

Voorbeeld REST – data - Agendalitem

```
public class Agendalitem {
    protected int id;
    protected String onderwerp = "Geen onderwerp";
    protected Date startDatum;
    protected Date endDatum;
    protected String locatie = "Geen locatie gespecificeerd";
    public Agendalitem() { this(new Date()); }
    public Agendalitem(int id) {...}
    public Agendalitem(Date startDatum) {...}
    public Agendalitem(Date startDatum, Date endDatum) {...}
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



13

Spring REST: Klasse annoteren

- **@RestController**
 - Gewone klasse
 - Wordt gewraapt in een servlet
 - REST-services
- **@RequestMapping**
 - Pad afgehandeld door REST-service

```
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("agenda")
public class AgendaController {
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

14

13

14

Spring REST: methode annoteren

- @GetMapping

- Methode handelt HTTP-berichten met methode GET af
- Resultaat methode automatisch geconverteerd naar JSON

```
@GetMapping
public Collection<Agendalitem> agendaVanNu() {
    return agenda.getAgendaItems(new Date());
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



15

Implementatie

- HTTP-aanvragen afbeelden op methodes RestController
 - Berichten converteren
 - Data wegschrijven naar body HTTP-bericht
- **Parameters in pad**
 - Data uit body HTTP-bericht ophalen
 - Antwoordstatus bepalen
 - URI's in antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica



16

15

16

Voorbeeld REST - GET

- localhost:8080/agenda/231119
- localhost:8080/agenda/id/2

```
{
    "id": 2,
    "onderwerp": "T1le",
    "startdatum": "2019-11-20T19:00:00+0000",
    "enddatum": "2019-11-20T20:00:00+0000",
    "locatie": "Gent"
},
{
    "id": 1,
    "onderwerp": "Brunch",
    "startdatum": "2019-11-21T18:00:00+0000",
    "enddatum": "2019-11-21T21:00:00+0000",
    "locatie": "Lokeren"
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



17

17

18

Voorbeeld REST – GET - RestController

- localhost:8080/agenda/231119
- localhost:8080/agenda/id/2

```
@GetMapping("{datum}")
public Collection<Agendalitem> agendaVanNu(@PathVariable("datum") String datumString) {
    DateFormat dateFormat = new SimpleDateFormat(pattern="ddMMyy");
    Date datum = dateFormat.parse(datumString, new ParsePosition( index: 0));
    return agenda.getAgendaItems(datum);
}

@GetMapping("/{id}/{id}")
public Agendalitem getAgendalitemResource(@PathVariable("id") int id) {
    return agenda.getAgendalitem(id);
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

18

1.3. REST WEB SERVICES IN JAVA

HOOFDSTUK 1. REST WEB SERVICES

Methodes annoteren

- Deelpad → tussen { ... }
- Dient vaak ook als parameter van methode
 - Parameters annoteren
 - @PathVariable

```
@GetMapping("/{datum}")
public Collection<AgendaItem> agendaItems(@PathVariable("datum") String datumString) {
    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    Date date = dateFormat.parse(datumString, new ParsePosition(0));
    return agenda.getItems(date);
}

@GetMapping("/{id}")
public AgendaItem getAgendaItemResource(@PathVariable("id") int id) {
    return agenda.getItems(id);
}
```

19



Implementatie

- HTTP-aanvragen afbeelden op methodes RestController
 - Berichten converteren
 - Data wegschrijven naar body HTTP-bericht
- Parameters in pad
 - Data uit body HTTP-bericht ophalen
- Antwoordstatus bepalen
- URI's in antwoord

FACULTEIT
EN ARCHITECTUUR

Industriel Ingenieur Informatica



20

Voorbeeld REST - POST

- URL: <http://localhost:8080/agenda>
- Agenda-item toevoegen

| | 2 | 3 | 4 | 5 | 6 |
|---|--------|---------------------------|---------------------------|---------------|---|
| 2 | Film | 2019-11-20T19:00:00+00:00 | 2019-11-20T21:00:00+00:00 | Gent | |
| 1 | Brunch | 23/11/2019 20:00+00:00 | 23/11/2019 22:00+00:00 | Lokatie | |
| 3 | Bork | 2019-11-13T17:00:00+00:00 | 2019-11-13T22:00:00+00:00 | Veld, Brussel | |

21



Voorbeeld REST – POST – RestController

- <http://localhost:8080/agenda>

Deze methode van de RestController
handelt de POST-aanvraag af

```
@PostMapping
public AgendaItem voegItemToe(@RequestBody AgendaItem item) {
    AgendaItem itemMetId = agenda.addAgendaItem(item);
    return itemMetId;
}
```

Body HTTP-vraag: JSON → object
Meergegeven als parameter van de methode



22

21

22

Spring REST: methode annoteren

- **@PostMapping**
 - Methode handelt HTTP-berichten met methode POST af
 - Resultaat methode automatisch geconverteerd naar JSON
- **@RequestBody**
 - Body HTTP-request → Java-object
 - Namen properties/eigenschappen moeten zelfde zijn

```
@PostMapping
public AgendaItem voegItemToe(@RequestBody AgendaItem item) {
    AgendaItem itemMetId = agenda.addAgendaItem(item);
    return itemMetId;
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

23



Voorbeeld REST - PUT

- URL: <http://localhost:8080/agenda/3>

| | 2 | 3 | 4 | 5 | 6 |
|---|--------|---------------------------|---------------------------|---------------|---|
| 2 | Film | 2019-11-20T19:00:00+00:00 | 2019-11-20T21:00:00+00:00 | Gent | |
| 1 | Brunch | 2019-11-20T20:00:00+00:00 | 2019-11-20T22:00:00+00:00 | Lokatie | |
| 3 | Bork | 2019-11-13T17:00:00+00:00 | 2019-11-13T22:00:00+00:00 | Veld, Brussel | |



24

23

24

HOOFDSTUK 1. REST WEB SERVICES**1.3. REST WEB SERVICES IN JAVA**

Voorbeeld REST – PUT - RestController

- http://localhost:8080/agenda/3

Deze methode van de RestController handelt de PUT-aanvraag af

```
@PutMapping("{id}")
public void verschijfVroegToe(@PathVariable("id") int id, @RequestBody AgendaItem item) {
    agenda.veranderAgendaItem(id, item);
}
```

Laatste deel pad wordt geïnterpreteerd als een variabele, kan meegegeven worden als parameter van de methode

Body HTTP-vraag: JSON → object
Meegegeven als parameter van de methode

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 25

25

Voorbeeld REST - DELETE

- URL http://localhost:8080/agenda/3

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR 26

26

Voorbeeld REST – DELETE - RestController

- http://localhost:8080/agenda/3

Deze methode van de RestController handelt de DELETE-aanvraag af

```
@DeleteMapping("/{id}")
public void delete(@PathVariable("id") int id) {
    agenda.deleteAgendaItem(id);
}
```

Laatste deel pad wordt geïnterpreteerd als een variabele, kan meegegeven worden als parameter van de methode

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 27

27

Spring REST: methode annoteren

- **@GetMapping, PostMapping, PutMapping, DeleteMapping**
 - Methode handelt HTTP-berichten met methode ... af
 - Resultaat methode automatisch geconverteerd naar JSON
- **@RequestBody**
 - Body HTTP-request → Java-object
 - Let op namen properties/eigenschappen JSON-object
- **Deelpad → tussen { ... }**
 - Dient als parameter van methode
 - Parameters annoteren
 - @PathVariable

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 28

28

REST-webservice maken

- Klasse annoteren
 - Pad
- Methodes annoteren
 - Deelpad
 - HTTP-Methode
 - Invoer-Uitvoer
 - Parameters annoteren

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 29

29

Implementatie

- HTTP-aanvragen afbeelden op methodes RestController
 - Berichten converteren
 - Data wegschrijven naar body HTTP-bericht
- Parameters in pad
 - Data uit body HTTP-bericht ophalen
- **Antwoordstatus bepalen**
 - URI's in antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 30

30

1.3. REST WEB SERVICES IN JAVA

HOOFDSTUK 1. REST WEB SERVICES

Statuscode koppelen aan exceptie

- Fout in het verwerken van de HTTP-aanvraag
 - Bv.
 - > Klant met onbestaande id opgevraagd
 - > Onbestaand agenda-item geüpdatet
 - > ...
- Fout die opgegooid wordt, koppelen aan een HTTP-statuscode

Enum-type met verschillende statuscodes
 Foutklasse koppelen aan HTTP-statuscode
 bron: <https://www.baeldung.com/spring-response-status>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



31

Statuscode via ResponseEntity

- Extra informatie meegeven met de **foutmelding** (via de body van het HTTP-bericht)
- Voegt een **statuscode** toe aan het HTTP-bericht

Inhoud body
 Type inhoud body
 HTTP-statuscode
 bron: <https://www.baeldung.com/spring-response-entity>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



32

31

Statuscode koppelen aan methode

- **Statuscode** koppelen aan methode RestController

Alternatief @PostMapping
 1 @RequestMapping(method = RequestMethod.POST)
 2 @ResponseStatus(HttpStatus.CREATED)
 3 public void storeEmployee(@RequestBody Employee employee) {
 4 }
 5 ...
 bron: <https://www.javacodegeeks.com/2019/05/using-responsestatus-http-status-spring.html>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



33

Implementatie

- HTTP-aanvragen afbeelden op methodes RestController
 - Berichten converteren
 - Data wegschrijven naar body HTTP-bericht
- Parameters in pad
- Data uit body HTTP-bericht ophalen
- Antwoordstatus bepalen
- URI's in antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica



34

33

Uniforme interface

- Uniforme manier om te communiceren met de server
 - Onafhankelijk van het type client
- Richtlijnen
 - Bron-georiënteerd
 - Gebruik de HTTP-methodes expliciet
 - Links om makkelijk zelf bronnen te ontdekken

Industriel Ingenieur Informatica

35

URI (link) toevoegen in antwoordbericht

URI aanmaken op basis van het huidige pad
 Location-header toevoegen aan HTTP-antwoord
 bron: <https://www.programcreek.com/java-api-examples/?api=org.springframework.web.servlet.support.ServletUriComponentsBuilder>
 bron: <https://stackoverflow.com/questions/3318912/what-is-the-preferred-way-to-specify-an-http-location-response-header-in-spring44751260>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR



36

35

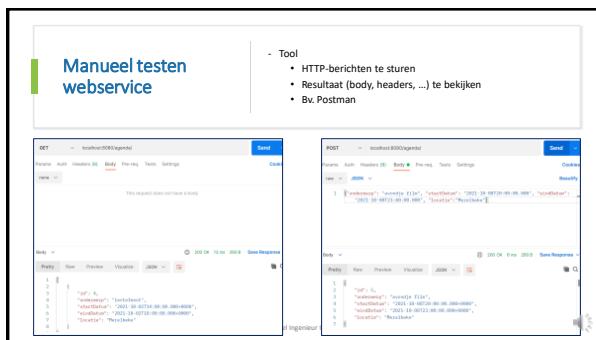
HOOFDSTUK 1. REST WEB SERVICES1.4. WEBSERVICES TESTEN

1

Webservices testen

- Manueel (algemeen)
- Herhaling Junit (Java)
- WebTestClient (Java – Spring)

2



3

Webservices testen

- Manueel (algemeen)
- Herhaling Junit (Java)
- WebTestClient (Java – Spring)

4



5

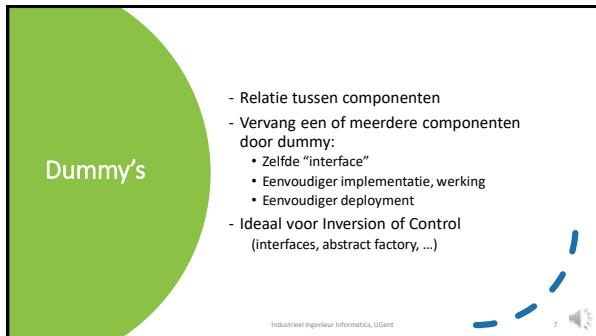
Fixture en fases

- Fixture
 - = vaste context/beginstoend voor testen
 - = baseline state
 - bv. lege databank, geformateerde schijf, geen gebruikers, etc.
- Vier fases:
 1. Set up: test fixture klaarzetten
 2. Exercise: interageer met het te onderzoeken systeem
 3. Verify: controleer of verwacht resultaat bereikt is
 4. Tear down: test fixture afbreken naar originele toestand
- Test suite: (onafhankelijke) unit tests met zelfde fixture

6

1.4. WEBSERVICES TESTEN

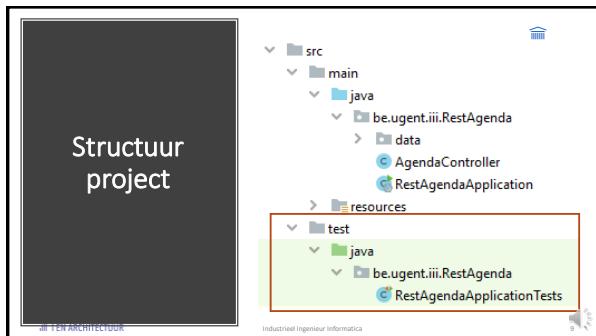
HOOFDSTUK 1. REST WEB SERVICES



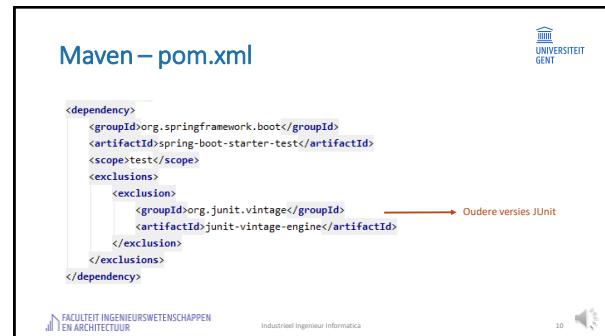
7



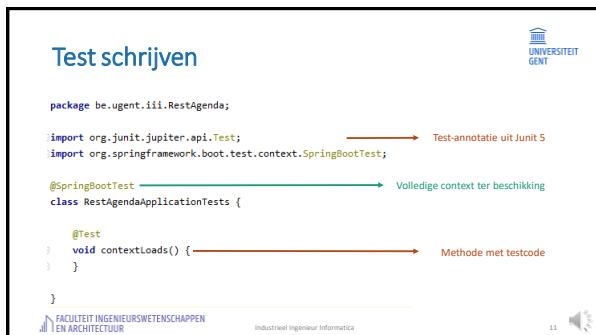
8



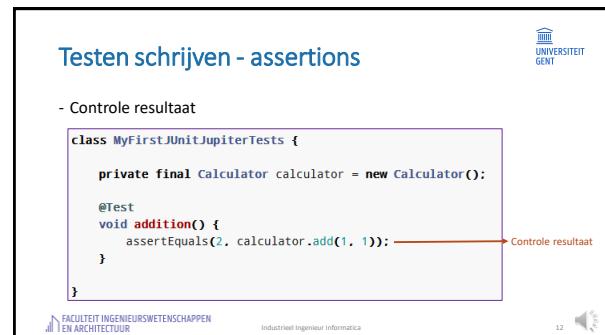
9



10



11



12

HOOFDSTUK 1. REST WEB SERVICES

1.4. WEBSERVICES TESTEN

Assertions

- <https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html>

```

static void assertEquals(double[] expected, double[] actual, String message)
    Asserts that expected and actual double arrays are equal.

static void assertEquals(double expected, double actual, String message)
    Asserts that expected and actual are equal.

static void assertFalse(boolean condition, String message)
    Asserts that the supplied condition is not true.

static void assertNotEquals(Object unexpected, Object actual, String message)
    Asserts that expected and actual are not equal.

static void assertNull(Object actual, String message)
    Asserts that actual is not null.
  
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 13

13

Nog meer assertions met AssertJ

- <https://www.baeldung.com/introduction-to-assertj>

- Meer mogelijkheden
- Verschillende testen op één object

```

1 assertEquals(frodo)
2 .isNotEqualTo(sauron)
3 .isIn(fellowshipOfTheRing);
4
5 assertEquals(frodo.getName())
6 .startsWith("Fro")
7 .endsWith("do");
8 .isEqualToIgnoringCase("frodo");
9
10 assertEquals(fellowshipOfTheRing)
11 .hasSize(9)
12 .contains(frodo, sam)
13 .doesNotContain(sauron);
  
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 14

14

Voor en na een test

`@BeforeAll`
`static void setup() {
 log.info("@BeforeAll - executes once before all test methods in this class");
}`

`@BeforeEach`
`void init() {
 log.info("@BeforeEach - executes before each test method in this class");
}`

`@AfterEach`
`void tearDown() {
 log.info("@AfterEach - executed after each test method.");
}`

`@AfterAll`
`static void done() {
 log.info("@AfterAll - executed after all test methods.");
}`

bron: <https://www.baeldung.com/junit-5>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 15

15

Webservices testen

- Manueel (algemeen)
- Herhaling Junit (Java)
- WebTestClient (Java – Spring)

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 16

16

REST-webservice testen: Client nodig

- Client nodig om HTTP-berichten te sturen

```

@ExtendWith(SpringExtension.class) → Junit 5 combineren met Spring
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT) → niet nodig bij recente Springversies
@AutoConfigureWebTestClient → willekeurige poort gebruiken voor te testen REST-service (conflicten vermijden)

class RestAgendaApplicationTests {
    @Autowired
    private WebTestClient webClient;
  
```

Injecteer WebTestClient

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 17

17

HTTP-antwoord testen met WebTestClient

- Juiste inhoud? Juiste headers? Juiste statuscode?

```

@Test
public void test2GetAllGithubRepositories() {
    webTestClient.get().uri("/api/repos") → HTTP-methode + URI instellen
        .accept(MediaType.APPLICATION_JSON_UTF8) → Headers instellen
        .exchange() → Bericht versturen
        .expectStatus().isOk() → Controle statuscode
        .expectHeader().contentType(MediaType.APPLICATION_JSON_UTF8) → Controle headers
        .expectBodyList(GithubRepo.class); → Controle Inhoud body
  
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 18

17

1.4. WEBSERVICES TESTEN

HOOFDSTUK 1. REST WEB SERVICES

WebTestClient – test API

- Headers aanvraag instellen [https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/WebTestClient.RequestHeadersSpec.html](https://www.javadoc.io/static/org.springframework/spring-test/6.0.12/org/springframework/test/web/reactive/server/WebTestClient.RequestHeadersSpec.html)
- HTTP-status controleren <https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/StatusAssertions.html>
- HTTP-headers controleren <https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/HeaderAssertions.html>
- HTTP-body controleren <https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/WebTestClient.ResponseSpec.html>
- Inhoud body opvragen (één object) <https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/WebTestClient.BodySpec.html>
- Inhoud body opvragen (lijst) <https://www.javadoc.io/static/org.springframework/test/6.0.12/org/springframework/test/web/reactive/server/WebTestClient.ListBodySpec.html>

19

Profielen gebruiken op te testen

- Andere beans (componenten, services, ...) gebruiken afhankelijk van een profiel (test, ontwikkeling, productie, ...)
- Werkwijze
 - Bean annoteren met een profiel
 - Profiel toekennen aan klasse met programma, test, ...



20

Bean - @Profile

```
@Component
@Profile("dev")
public class DummyDao implements Dao {...}
```

Bean hoort bij een welbepaald profiel


```
@Component
@Profile("!dev")
public class RealDao implements Dao {...}
```

Bean hoort bij alle profielen behalve het opgegeven profiel

- Afhankelijk van profiel zal het Springframework een ander type object injecteren
- Geen **@Profile** → standaardprofiel (**default**)

21

Profiel instellen

- Bij testen via **@ActiveProfiles**

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureWebTestClient
@ActiveProfiles("test")
public class TestsClass {...}
```

- Uitvoeren toepassing
 - In configuratie, omgevingsvariabele, maven-profiel, commandolijn-optie, ...



22

Webservices testen

- Manueel (algemeen)
- Herhaling Junit (Java)
- WebTestClient (Java – Spring)

23

HOOFDSTUK 1. REST WEB SERVICES

1.5. WEBSERVICES BEVEILIGEN

1.5 Webservices beveiligen

Webservices beveiligen

Veerle Ongenae

1

Webservice beveiligen

- Authentificatie: gebruiker herkennen (bv. login – wachtwoord)
 - PasswordEncoder: wachtwoord veilig bewaren
- Autorisatie: rechten instellen
- CSRF: Cross Site Request Forgery
- HTTPS gebruiken

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

Universiteit Gent

2

Authenticatie

- Gebruikers configureren

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public UserDetailsService userDetailsService(BCryptPasswordEncoder bCryptPasswordEncoder) {
        InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
        manager.createUser(User.withUsername("user")
            .password(bCryptPasswordEncoder.encode("userPass"))
            .roles("USER"));
        manager.createUser(User.withUsername("admin")
            .password(bCryptPasswordEncoder.encode("adminPass"))
            .roles("USER", "ADMIN"));
        return manager;
    }
}
```

Configuratiebestand
Standaardbeveiling uitgeschakeld en vervangen

Methode die bean levert → gebruikt bij Dependency Injection

Encrypteren wachtwoorden

Gebruikers aanmaken en aanpassen in het geheugen → testen

Interface om op basis van logingegevens op te halen

Industrieel Ingenieur Informatica

3

Webservice beveiligen

- Authentificatie: gebruiker herkennen (bv. login – wachtwoord)
 - PasswordEncoder: wachtwoord veilig bewaren
- Autorisatie: rechten instellen
- CSRF: Cross Site Request Forgery
- HTTPS gebruiken

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

Universiteit Gent

4

Autorisatie

- Rechten bepalen

- Toegang beperken

bron: <https://docs.spring.io/spring-security/reference/service/architecture.html>

5

Autorisatie

- Beveiliging configureren

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests((auth) -> auth
                .anyRequest().authenticated())
            .httpBasic(withDefaults());
        return http.build();
    }
}
```

Bean voor reeks beveiligingsfilters

Configuratie beveiling HTTP-berichten

Beperken toegang

Basis HTTP-authenticatie via headers

SecurityFilterChain maken

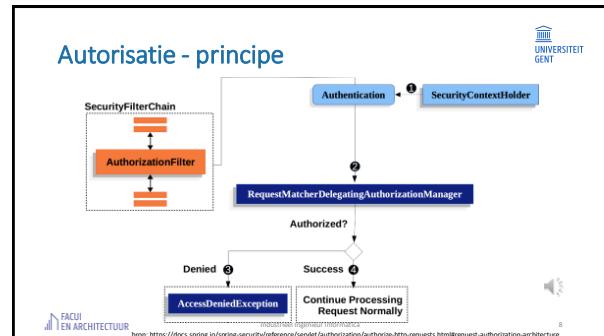
6

1.5. WEBSERVICES BEVEILIGEN

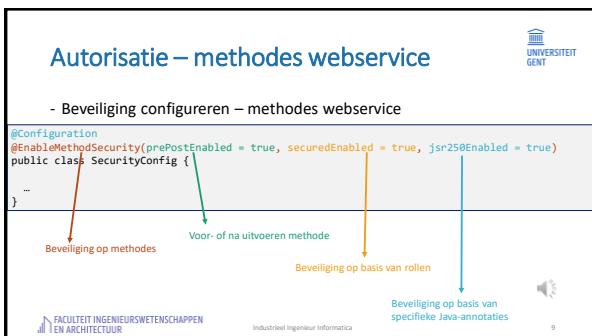
HOOFDSTUK 1. REST WEB SERVICES



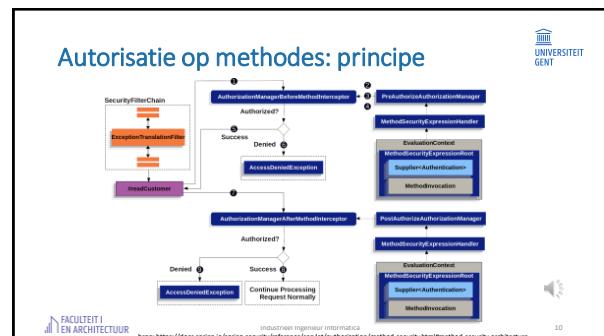
7



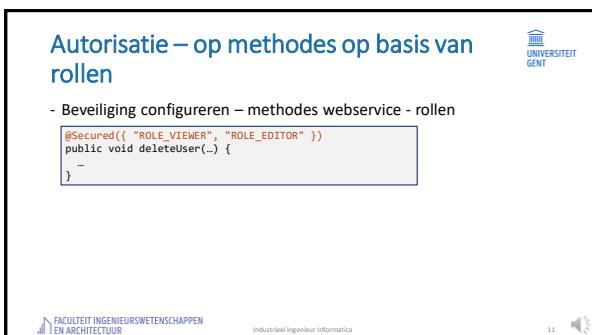
8



9



10



11



12

HOOFDSTUK 1. REST WEB SERVICES

1.5. WEBSERVICES BEVEILIGEN

Autorisatie – voor of na uitvoeren methode

- Beveiliging configureren – conditie voor of na uitvoeren methode

```
@PreAuthorize("hasRole('ROLE_VIEWER')")
public String getUsernameInUpperCase() {
    ...
}

@PreAuthorize("#username == authentication.principal.username")
public String getMyRoles(String username) {
    ...
}
```

Voor uitvoeren methode Uitdrukking in Spring Expression Language

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 13

13

Autorisatie op requests versus methodes

| | Op request/bericht | Op methode |
|-----------------------|--------------------------|----------------------------|
| Type autorisatie | Grofmazig | Fijnmazig |
| Plaats configuratie | In configuratieklasse | Op een specifieke methode |
| Type configuratie | Met lambda-uitdrukkingen | Met annotations |
| Definitie autorisatie | In code | Spring Expression Language |

bron: <https://docs.spring.io/spring-security/reference/servlet/authorization/method-security.html>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 14

14

Webservice beveiligen

- Authenticatie: gebruiker herkennen (bv. login – wachtwoord)
 - PasswordEncoder: wachtwoord veilig bewaren
- Autorisatie: rechten instellen
- CSRF: Cross Site Request Forgery
- HTTPS gebruiken

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 15

15

Wat is CSRF?

Website bank Hackers site

```
<form method="post"
      action="/transfer">
<input type="text"
      name="amount"/>
<input type="text"
      name="routingNumber"/>
<input type="text"
      name="account"/>
<input type="submit"
      value="Transfer"/>
```

POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=Randomid
Content-Type: application/x-www-form-urlencoded
amount=100.00&routingNumber=12345&account=9876

<form method="post"
 action="https://bank.example.com/transfer">
<input type="hidden"
 name="amount"
 value="100.00"/>
<input type="hidden"
 name="routingNumber"
 value="evilRoutingNumber"/>
<input type="hidden"
 name="account"
 value="evilAccountNumber"/>
<input type="submit"
 value="Transfer"/>

informatica 16

16

Beveiliging tegen CSRF

- Hoe onderscheid maken tussen beide HTTP-aanvragen? (bank site vs hackers site)
- **Synchronizer Token Pattern**
 - Naast cookie ook een willekeurig token in HTTP-request

```
<form method="post"
      action="/transfer">
<input type="text"
      name="amount"
      value="4fd1579-3ad1-4d21-96c7-4ef2d9ff8721"/>
<input type="text"
      name="routingNumber"/>
<input type="text"
      name="routingNumber"/>
<input type="text"
      name="account"/>
<input type="submit"
      value="Transfer"/>
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 17

17

Beveiliging tegen CSRF

- Hoe onderscheid maken tussen beide HTTP-aanvragen? (bank site vs hackers site)
- **Synchronizer Token Pattern**
- **SameSite-attribuut op cookie**

```
Set-Cookie: JSESSIONID=Randomid; Domain=bank.example.com; Secure; HttpOnly; SameSite=Lax
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR Industriel Ingenieur Informatica 18

21

1.5. WEBSERVICES BEVEILIGENHOOFDSTUK 1. REST WEB SERVICES**CSRF in Spring Security**

- Standaard aan

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            // ...
            .csrf((csrf) -> csrf
                .ignoringRequestMatchers("/api/**")
            );
        return http.build();
    }
}
```



Uitschakelen CSRF voor bepaalde webservices

19

Webservice beveiligen

- Authenticatie: gebruiker herkennen (bv. login – wachtwoord)
 - PasswordEncoder: wachtwoord veilig bewaren
- Autorisatie: rechten instellen
- CSRF: Cross Site Request Forgery
- HTTPS gebruiken

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR Industriel Ingenieur Informatica



20

20

22

4

HOOFDSTUK 1. REST WEB SERVICES1.6. REACTIVE WEBSERVICES IN JAVA

1.6 Reactive webservices in Java

Reactive REST webservices

Veerle Ongenaé

1

Overzicht

- Herhaling reactive programming

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



2

Waarom Reactive Programming ?

Traditional Data Processing

Stream Processing

bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armenia-1/>

Industriel Ingenieur Informatica, UGent

3

Observer - push

If use bounded buffer, drop messages

Fast Publisher → Subscriber

Slow Subscriber (10 op/sec)

Fast Publisher (100 op/sec) → Slow Subscriber (10 op/sec)

use buffer for pending events

Push Event

If use unbounded buffer, out of memory

Fast Publisher (100 op/sec) → Slow Subscriber (10 op/sec)

Slow Subscriber (10 op/sec)

Industriel Ingenieur Informatica, UGent

bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armenia-1/>

4

Observer - pull

Request as many as we need!

Fast Publisher (100 op/sec) → Slow Subscriber (10 op/sec)

request(10)

Backpressure (tegendruk) in softwaresystemen is het vermogen om de communicatie te overbelasten. Met andere woorden, zenders van informatie overstelpen consumenten met gegevens die zij niet kunnen verwerken. Men past deze term ook toe als het mechanisme om dit te controleren en af te handelen.

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armenia-1/>

5

Reactive Programming - principe

Handling events by observer

Observable → Subscription

next function
error function
complete function

Subscribers: Subscriber 1, Subscriber 2, Subscriber 3

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

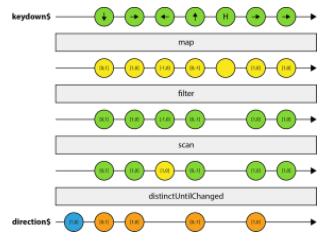
bron: <https://dev.to/sagar/reactive-programming-in-javascript-with-rxjs-4j0m>

6

1.6. REACTIVE WEBSERVICES IN JAVA

HOOFDSTUK 1. REST WEB SERVICES

Reactive Programming - pipeline



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

<https://blog.thoughttram.io/rxjs/2017/08/24/taming-snakes-with-reactive-streams.html>



7

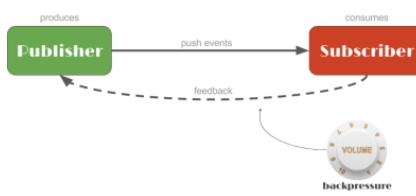
Overzicht

- Herhaling reactive programming
- Reactive Streams



8

Reactive Streams - concept



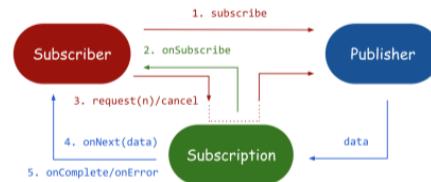
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



9

Reactive Streams API (Java)



bron: <https://engineering.linecorp.com/en/blog/reactive-streams-armenia-1/>

Industriel Ingenieur Informatica, UGent



10

Overzicht

- Herhaling reactive programming
- Reactive Streams
- Reactor



11

Reactor



FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

<https://projectreactor.io/>

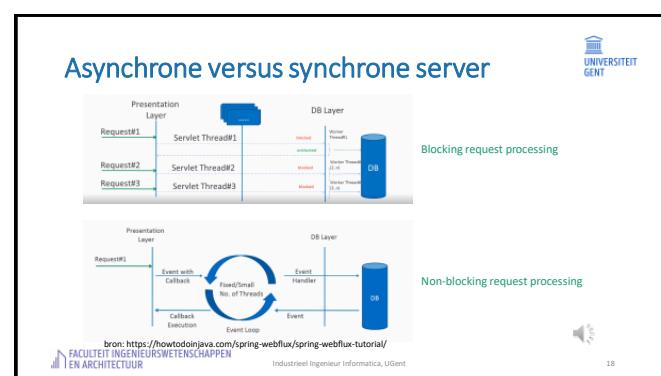
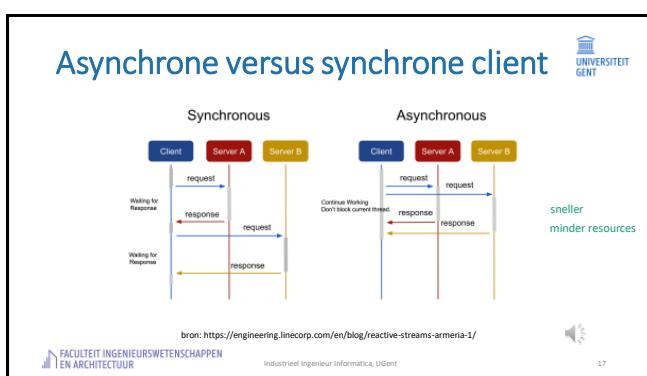
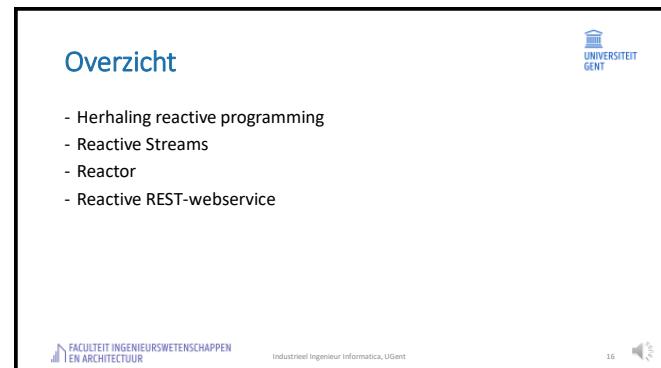
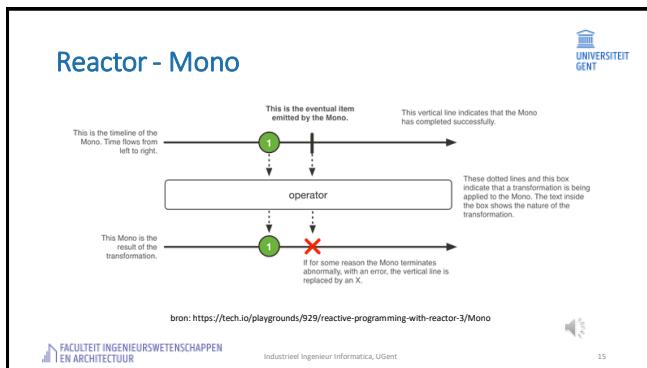
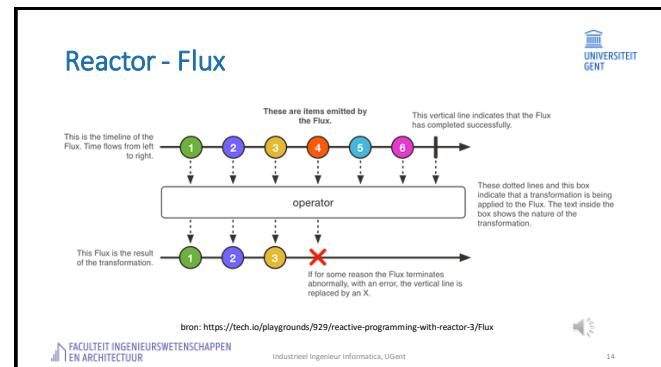
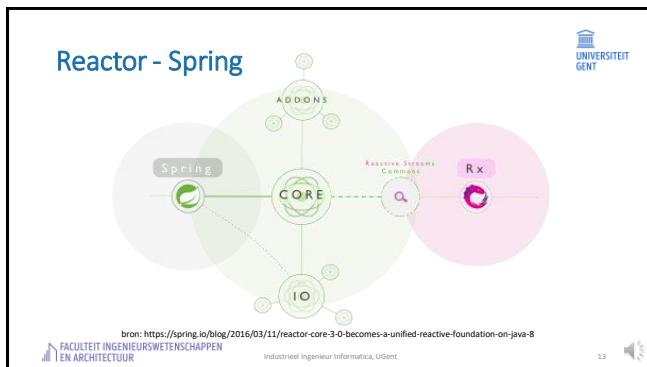


12

11

24

2

HOOFDSTUK 1. REST WEB SERVICES1.6. REACTIVE WEBSERVICES IN JAVA

17

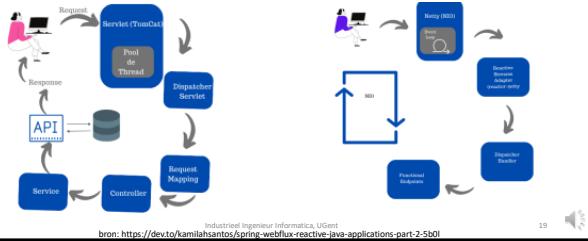
25

3

1.6. REACTIVE WEBSERVICES IN JAVA

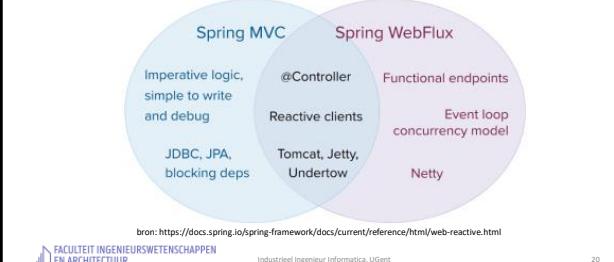
HOOFDSTUK 1. REST WEB SERVICES

Spring MVC versus Spring WebFlux - Architectuur

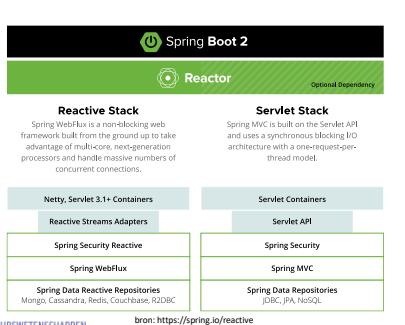


19

Spring MVC versus Spring WebFlux



20



21

Spring WebFlux

- Twee programmeermodellen
 - Reactive components met **annotations**
 - Zelfde annotaties als Spring MVC @RestController, ...
 - Resultaat methodes: Mono of Flux
 - **Functional Routing and Handling**
 - Klasse met methodes die bepalde aanvragen afhandelen (Handler)
 - Parameter: request-object
 - Resultaat: Mono met response-object
 - Configuratie: routing
 - Pad ~ methode handler

22

Reactive DAO

```
@Service
public class BoorputDAO {
    final Random random = new Random();
    String[] ids = {"BP1", "BP2", "BP3"};

    public Flux<Boorput> geefMetingen() {
        return Flux.interval(Duration.ofSeconds(1)).take(10)
            .map(pulse -> geefMeting());
    }

    private Boorput geefMeting() {
        Boorput boorput = new Boorput();
        boorput.setId(ids[random.nextInt(ids.length)]);
        boorput.setTijdstipDebiet(LocalDateTime.now());
        boorput.setTijdstipPeil(LocalDateTime.now());
        boorput.setPeil(28 + 5 * random.nextDouble());
        boorput.setDebiet(5 + 2 * random.nextDouble());
        return boorput;
    }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

23

Reactive REST-webservice

```
@RestController
@RequestMapping("boorputten")
public class BoorputController {

    private BoorputDAO dao;

    public BoorputController(BoorputDAO dao) {
        this.dao = dao;
    }

    @GetMapping("metingen")
    public Flux<Boorput> haalMetingen() {
        return dao.geefMetingen();
    }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

24

23

26

4

HOOFDSTUK 1. REST WEB SERVICES

1.6. REACTIVE WEBSERVICES IN JAVA

25



Webclient: niet blokkerend (non blocking) client

```
System.out.println("start test");
ClientTestwebflux client = new ClientTestWebflux();
client.testServer();
System.out.println("na test");
```

```
public class ClientTestWebflux {

    public void testServer() {
        webClient client = WebClient.create("http://localhost:8080");

        Flux<Boorput> boorputFlux = client.get()
            .uri("/boorput/metingen")
            .retrieve()
            .bodyToFlux(Boorput.class);

        boorputFlux.subscribe(System.out::println);
        System.out.println("in test webflux");
    }
}
```

Start test
in test webflux
na test
111.volleflux.web.Boorput@7f21ac6
111.volleflux.web.Boorput@7f21e4a
111.volleflux.web.Boorput@7f21ec2
111.volleflux.web.Boorput@7f21e22
111.volleflux.web.Boorput@7f21e20
111.volleflux.web.Boorput@6dfe530
111.volleflux.web.Boorput@7647bf
111.volleflux.web.Boorput@997f233
111.volleflux.web.Boorput@32463fF
111.volleflux.web.Boorput@9e2a0
111.volleflux.web.Boorput@644b258

26



UNIVERSITEIT
GENT

Spring WebFlux

- Twee programmeermodellen
 - Reactive components met **annotaties**
 - Zelfde annotaties als Spring MVC @RestController, ...
 - Resultaat methodes: Mono of Flux
 - GEEN blocking methodes gebruiken
 - **Functional Routing and Handling**
 - Klasse met methodes die bepaalde aanvragen afhandelen (Handler)
 - Parameter: request-object
 - Resultaat: Mono met response-object
 - Configureert: routing
 - Pad ~ methode handler

 FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatique, Ugent

27

27

Reactive REST-webservice - handler



```
@Component
public class BoorputHandler {
    private BoorputDAO dao;

    public BoorputHandler(BoorputDAO dao) {
        this.dao = dao;
    }
    public Mono<ServerResponse> metingenVoorBoorputten(ServerRequest request) {
        return ServerResponse.ok().contentType(MediaType.APPLICATION_JSON)
            .body(dao.geefMetingen(), Boorput.class);
    }
}

@FunctionalInterface
public interface HandlerFunction<T extends ServerResponse> {
    Mono<T> handle(ServerRequest request);
}
```

28



Reactive REST-webservice - routing

```
@Configuration
public class BoorputRouter {
    @Bean
    public RouterFunction<ServerResponse> routeBoorput(BoorputHandler boorputHandler) {
        return RouterFunctions.route(RequestPredicates.GET("/boorputten/metingenAnders")
            .and(RequestPredicates.accept(MediaType.APPLICATION_JSON)),
            boorputHandler::metingenVoorBoorputten);
    }
}

@FunctionalInterface
public interface RouterFunction<T extends ServerResponse> {
    Mono<HandlerFunction<T>> route(ServerRequest request);
    // ...
}

public static <T extends ServerResponse> RouterFunction<T> route(
    RequestPredicate predicate,
    HandlerFunction<T> handlerFunction)
    hulpfunctie
```

29

Spring Data Reactive



- MongoDB
 - Document-georiënteerde databank (JSON)
- Cassandra
 - Mix tussen key-value store en databank gebruikmakend van tabellen (wide column store)
- Redis
 - In-memory key-value database
- NoSQL databases

30

Overzicht

- Herhaling reactive programming
- Reactive Streams
- Reactor
- Reactive REST-webservice



31

Industriel Ingenieur Informatica, UGent

31

HOOFDSTUK 1. REST WEB SERVICES1.7. REST WEB SERVICES IN C#

1.7 REST web services in C#

REST- services in .NET: Web API

Veerle Ongenaé

Speaker icon

1

Overzicht

- Wat is Web API?

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Universiteit Gent

Industrieel Ingenieur Informatica

Speaker icon

2

Weservice

- Service (dienst) aanbieden via het web (HTTP)
 - Functionaliteit
 - Data
- Endpoint

bron: <https://medium.com/@innovationschef/web-services-client-in-java-72386ea55e4>

Industrieel Ingenieur Informatica

Speaker icon

3

Architecturale beperkingen REST API

- Client-Server
- Uniforme Interface
- Statusloos
- Cacheable
- Gelaagd Systeem
- Code op aanvraag (optioneel)

Industrieel Ingenieur Informatica

Speaker icon

4

Web API

- Restful webservices met ASP.NET Core

https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industrieel Ingenieur Informatica

Speaker icon

5

Overzicht

- Wat is Web API?
- Een nieuw project

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Universiteit Gent

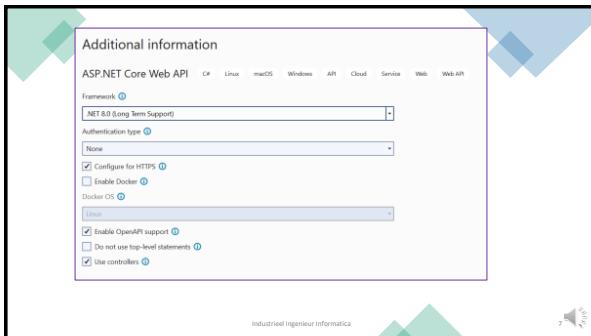
Industrieel Ingenieur Informatica

Speaker icon

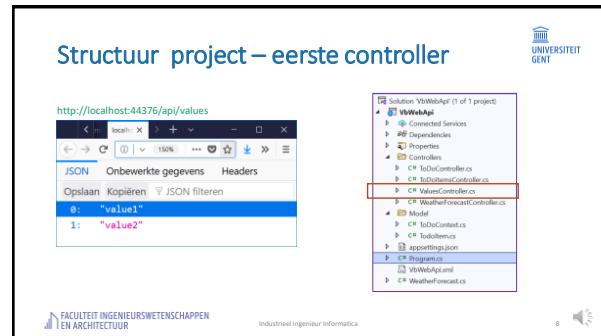
6

1.7. REST WEB SERVICES IN C#

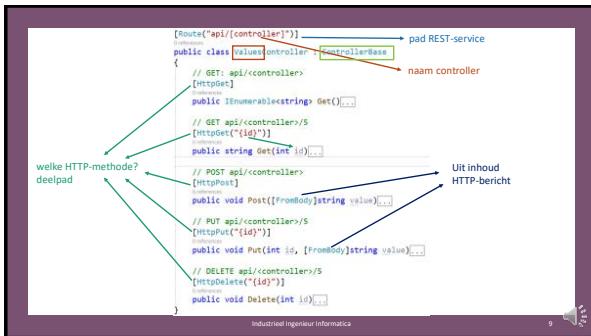
HOOFDSTUK 1. REST WEB SERVICES



7



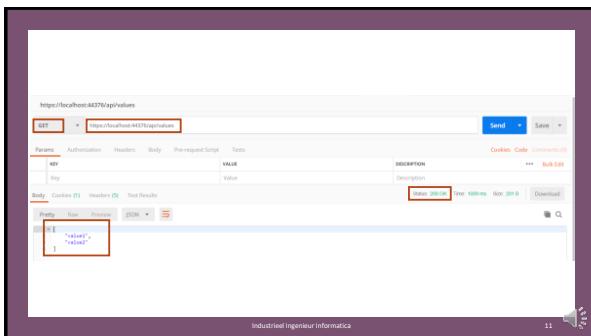
8



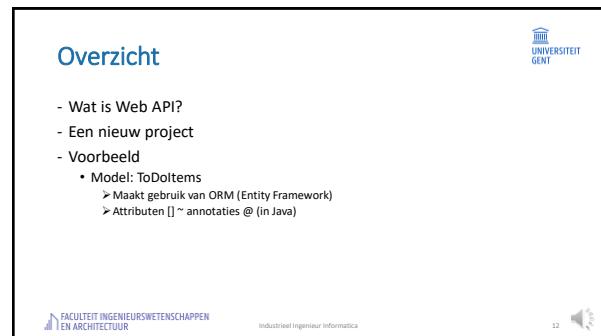
9



10



11



12

HOOFDSTUK 1. REST WEB SERVICES**1.7. REST WEB SERVICES IN C#**

```

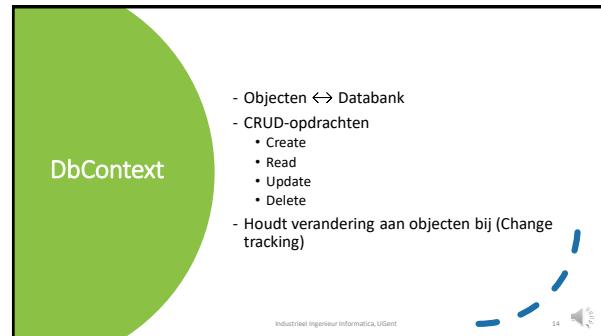
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace VbWebAPI.Models
{
    public class ToDoItem
    {
        public long Id { get; set; } → unieke id  
[Required] → NOT NULL - kolom
        public string Name { get; set; }
        [DefaultValue(false)] → standaardwaarde
        public bool IsComplete { get; set; }
    }
}

```

Model

13



14

```

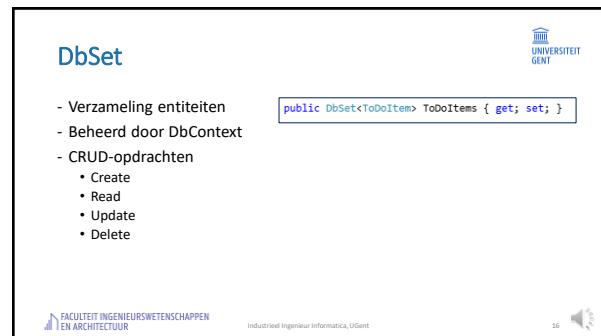
using Microsoft.EntityFrameworkCore;
namespace VbWebAPI.Models
{
    public class ToDoContext : DbContext
    {
        public ToDoContext(DbContextOptions<ToDoContext> options)
            : base(options)
        {
        }

        public DbSet<ToDoItem> ToDoItems { get; set; }
    }
}

```

Voorbeeld DbContext

15



16

Toevoegen item

```

private readonly ToDoContext _context;
public ToDoController(ToDoContext context)
{
    _context = context;
    if (_context.ToDoItems.Count() == 0)
    {
        _context.ToDoItems.Add(new ToDoItem { Name = "item1" });
        _context.SaveChanges();
    }
}

```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

17

17

Ophalen items

```

_context.ToDoItems.ToList() → List<ToDoItem>
var item = _context.ToDoItems.FirstOrDefault(t => t.Id == id); → ToDoItem

```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

18

1.7. REST WEB SERVICES IN C#

HOOFDSTUK 1. REST WEB SERVICES


 UNIVERSITEIT
GENT

Aanpassen item

```

var id = ...; // unieke identificatie
var item = ...; // nieuwe waarden voor todo
var todo = _context.ToDoItems.FirstOrDefault(t => t.Id == id);
if (todo != null)
{
    // item aanpassen
    todo.IsComplete = item.IsComplete;
    todo.Name = item.Name;
    _context.ToDoItems.Update(todo); Item aanpassen in DbContext
    _context.SaveChanges(); Aanpassingen naar database
}
  
```

19

Configuratie



UNIVERSITEIT
GENT

```
Program.cs
```

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<ToDoContext>(opt => opt.UseInMemoryDatabase("ToDoList"));
builder.Services.AddControllers();
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

20 

20



UNIVERSITEIT
GENT

Overzicht

- Wat is Web API?
- Een nieuw project
- Voorbeeld
 - Model
 - Controller
 - GET

 FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur informatica

21

21

22

```
namespace VoorbeeldWebAPI.Controllers
{
    [Route("api/{controller}")] → api/ToDoItems
    [ApiController] → controller voor REST API
    [Controller]
    public class ToDoItemsController : [ControllerBase] → methodes overerven
    {
        private readonly ToDoContext _context;
        ...
        public ToDoItemsController([ToDoContext context])... → dependency injection
        ...
        // GET: api/ToDoItems
        [HttpGet]
        public async Task<ActionResult<IEnumerable<ToDoItem>>> GetToDoItems()...
        ...
        // GET: api/ToDoItems/{id}
        [HttpGet("{id}")]
        public async Task<ActionResult<ToDoItem>> GetToDoItem(long id)...
```

23



UNIVERSITEIT
GENT

GET – alle items

```
// GET: api/ToDoItems
[HttpGet]           → GET-aanvraag naar api/todoitems afhandelen
public async Task<ActionResult<IEnumerable<ToDoItem>> GetToDoItems() → asynchrone methode
{
    return await _context.ToDoItems.ToListAsync();
}
```

Geserialiseerd naar JSON

HTTP 200-antwoord

Inhoud = JSON

Bij fout 400-antwoord

 FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

24

24

HOOFDSTUK 1. REST WEB SERVICES**1.7. REST WEB SERVICES IN C#**

GET – één item

```
// GET: api/ToDoItems/{id}
[HttpGet("{id}")] → api/todoitems/{id}
public async Task<ActionResult<ToDoItem>> GetToDoItem(long id)
{
    var ToDoItem = await _context.ToDoItems.FindAsync(id);

    if (ToDoItem == null)
    {
        return NotFound();           HTTP 404 antwoord, overgeerfd
    }

    return ToDoItem;             HTTP 200 antwoord
                                Inhoud = JSON
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

25

Overzicht

- Wat is Web API?
- Een nieuw project
- Voorbeeld
 - Model
 - Controller
 - GET
 - POST

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

26

request

POST → https://localhost:44376/api/todoitems

Body: { "name": "Tadaarts", "isComplete": false }

response

| Body | Cookies | Headers | Test Results |
|---------------------------------|---------|---------|--------------|
| {} 201 Created [29 ms 281 B] | | | |
| | | | |
| | | | |

KEY VALUE

- Transfer-Encoding: chunked
- Content-Type: application/json; charset=UTF-8
- Location: https://localhost:44376/api/todoitems/1
- Server: Microsoft-IIS/10.0
- X-Powered-By: ASP.NET
- Date: Thu, 03 Nov 2022 10:28:34 GMT

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

27

POST – item toevoegen

```
// POST: api/ToDoItems
[HttpPost]
public async Task<ActionResult<ToDoItem>> PostToDoItem([ToDoItem] ToDoItem toDoItem)
{
    _context.ToDoItems.Add(toDoItem);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetToDoItem", new { id = toDoItem.Id }, toDoItem);
}
```

Uit inhoud
HTTP-bericht

201 HTTP-antwoord
Location-header toegevoegd

URI bepaald door actie "GetToDoItem"

data voor de route (URI)

data voor de body

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

28

Overzicht

- Wat is Web API?
- Een nieuw project
- Voorbeeld
 - Model
 - Controller
 - GET
 - POST
 - PUT

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

29

request

PUT → https://localhost:44376/api/todoitems/1

Body: { "name": "Tandarts beker", "isComplete": true }

response

Status: 204 No Content Time: 190 ms Size: 2710

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica

30

1.7. REST WEB SERVICES IN C#

HOOFDSTUK 1. REST WEB SERVICES

```
// PUT: api/ToDoItems/{id}
[HttpPut("{id}")]
public async Task<ActionResult> PutToDoItem(long id, ToDoItem toDoItem)
{
    if (id != toDoItem.Id)
    {
        return BadRequest();
    }
    _context.Entry(toDoItem).State = EntityState.Modified;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ToDoItemExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return NoContent();
}
```

Industriel ingenieur informatica

31

Overzicht

- Wat is Web API?
- Een nieuw project
- Voorbeeld
 - Model
 - Controller
 - GET
 - POST
 - PUT
 - DELETE

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel ingenieur informatica



32

32

http://localhost:44376/api/todoitems

DELETE https://localhost:44376/api/todoitems/2

Body: { "id": 2, "name": "Tanden wassen", "isComplete": true }

http://localhost: 44376/api/todoitems/2

DELETE http://localhost:8022/api/todoitems/2

Body: { "id": 2, "name": "Tanden wassen", "isComplete": true }

33

Industriel Ingenieur Informatica

33

DELETE – item verwijderen

```
// DELETE: api/ToDoItems/{id}
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteToDoItem(long id)
{
    var toDoItem = await _context.ToDoItems.FindAsync(id);
    if (toDoItem == null)
    {
        return NotFound();
    }

    _context.ToDoItems.Remove(toDoItem);
    await _context.SaveChangesAsync();

    return toDoItem;
}
```

34

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel ingenieur informatica



34

34

HOOFDSTUK 1. REST WEB SERVICES1.8. OPENAPI SPECIFICATIE EN SWAGGER

1.8 OpenAPI specificatie en Swagger

OpenAPI specificatie | Veerle Ongenaee

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur Informatica, UGent

1

Open API Specification

- Formaat om REST-webservices te beschrijven
 - Welke endpoints zijn er? (paden, bv. /users)
 - Welke methodes ondersteunen ze? (GET, POST, ...)
 - Parameters: input, output?
 - Welke authenticatie-methodes?
 - Informatie: contact, licentie, ...
- In JSON of YAML (YAML Ain't Markup Language)

UNIVERSITEIT GENT
OpenAPI 3.0

- info
- servers
- security
- paths
- tags
- externalDocs
- components

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur Informatica, UGent

<https://www.openapis.org/blog/2017/03/20/openapi-spec-3-implementers-guide-released>

2

Waarom OpenAPI?

- Beschrijving interface onafhankelijk van een programmeertaal
- De "consumer" (gebruiker REST-webservices) kent de functionaliteit van de webservice zonder de broncode te kennen
- De "consumer" weet hoe hij kan interageren met de REST-webservice
- Ontkoppelen API en implementatie
- API first approach → consistentere API
- Leesbaar voor "machines" → handig voor veel tools: postman, mocks genereren, ...

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur Informatica, UGent

3

Swagger

- Drie tools
 - Swagger Editor: specificatie schrijven (<https://editor.swagger.io/>)
 - Swagger UI: specificatie genereren, interactieve documentatie
 - Swagger Codegen: specificatie → code

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur Informatica

4

YAML

Visuele voorstelling

Swagger Petstore

pet

Swashbuckle.AspNetCore

Industriel ingenieur Informatica, UGent

5

Swagger

- Drie tools
 - Swagger Editor: specificatie schrijven
 - **Swagger UI: specificatie genereren, interactieve documentatie**
 - Swagger Codegen: specificatie → code
- Toevoegen aan project via NuGet
- <https://docs.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-3.1&tabs=visual-studio>

Swashbuckle.AspNetCore by Swashbuckle.AspNetCore, 67.5M downloads

NSwag.AspNetCore by Rico Suter, 8.09M downloads

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel ingenieur Informatica

6

1.8. OPENAPI SPECIFICATIE EN SWAGGER

HOOFDSTUK 1. REST WEB SERVICES

Swagger UI

- Genereert JSON
 - Formele beschrijving REST-service
 - <https://localhost:44376/swagger/v1/swagger.json>
- Genereert webpagina
 - Leesbare beschrijving REST-service
 - <https://localhost:44376/swagger/index.html>

7

Gegeneerde JSON

<https://localhost:44376/swagger/v1/swagger.json>

```
openapi: "3.0.1"
  info: ...
  paths: ...
  components: ...
```

8

Gegeneerde JSON - info

<https://localhost:44376/swagger/index.html>

9

Gegeneerde JSON - paths

<https://localhost:44376/swagger/index.html>

10

Gegeneerde JSON - GET

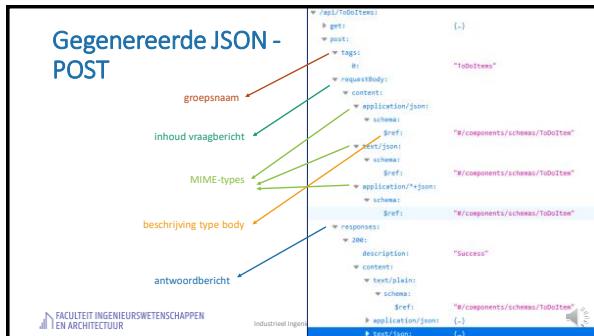
groepsnaam
HTTP-statuscode
inhoud antwoordbericht
MIME-types
beschrijving type antwoord

11

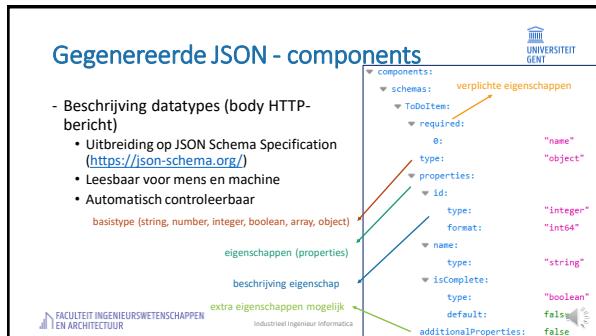
Gegeneerde JSON – GET met parameter

groepsnaam
parameters
Info parameter (naam, uit pad te halen, ...)
beschrijving type parameter

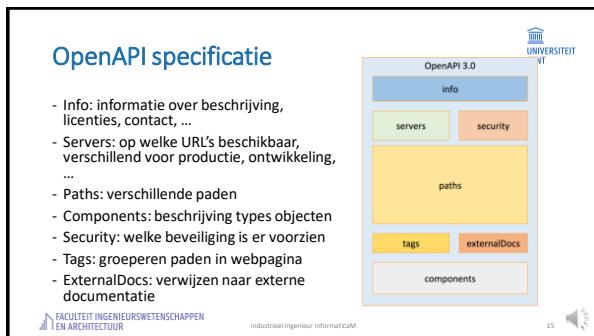
12

HOOFDSTUK 1. REST WEB SERVICES**1.8. OPENAPI SPECIFICATIE EN SWAGGER**

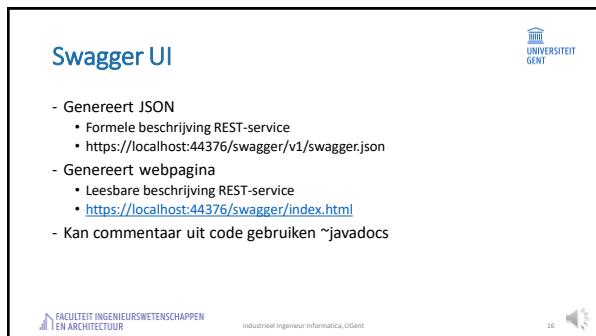
13



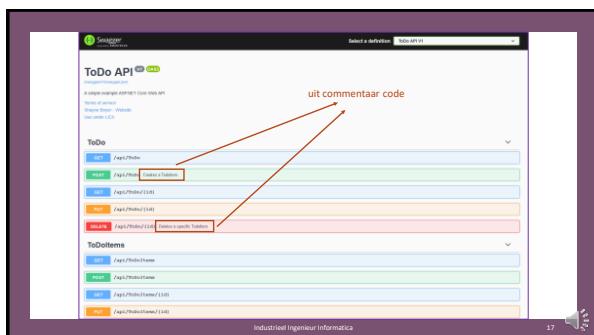
14



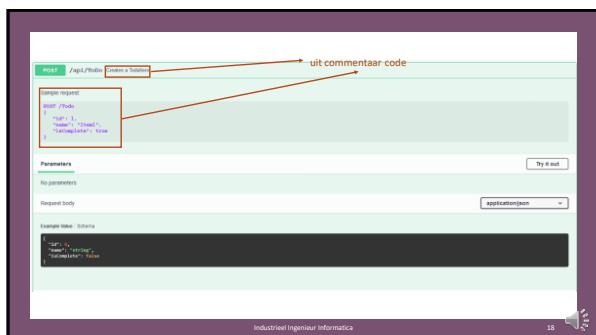
15



16



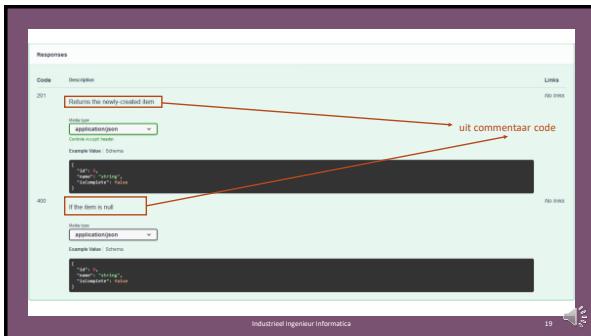
17



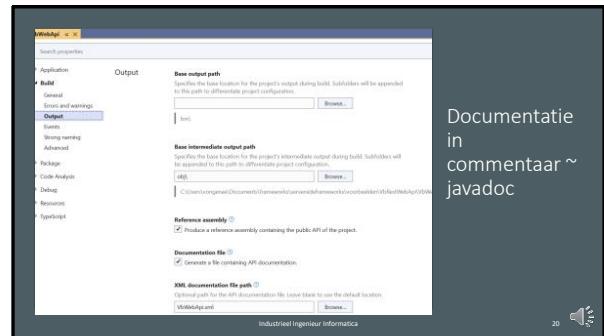
18

1.8. OPENAPI SPECIFICATIE EN SWAGGER

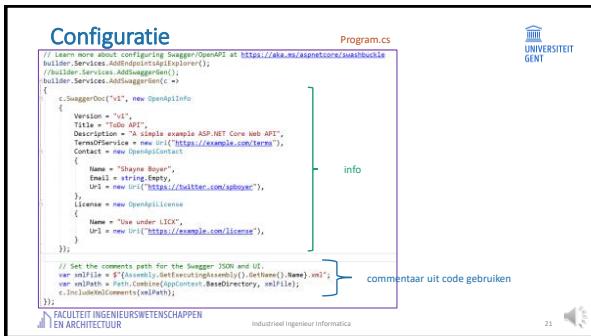
HOOFDSTUK 1. REST WEB SERVICES



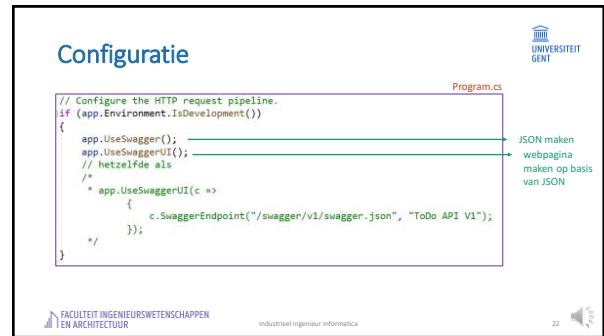
19



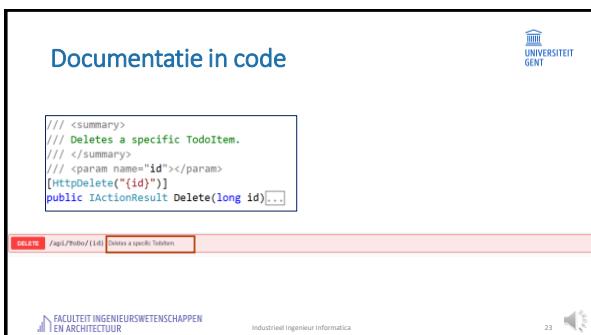
20



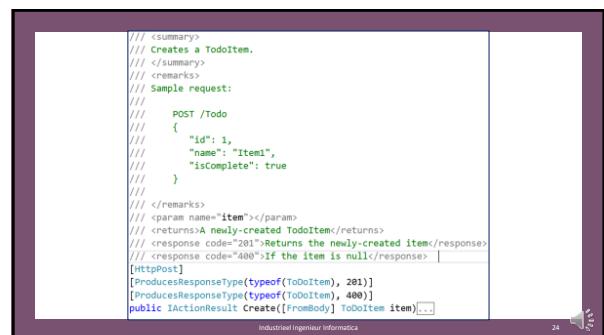
21



22



23



24

HOOFDSTUK 1. REST WEB SERVICES**1.8. OPENAPI SPECIFICATIE EN SWAGGER**

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

25

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

26

Swagger UI

UNIVERSITEIT GENT

- Genereert JSON
 - Formele beschrijving REST-service
 - <https://localhost:44376/swagger/v1/swagger.json>
- Genereert webpagina
 - Leesbare beschrijving REST-service
 - <https://localhost:44376/swagger/index.html>
- Kan commentaar uit code gebruiken ~javadocs
- Informatie uit data-attributen

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

27

Informatie in Data Annotaties

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace VbWebAPI.Models
{
    public class ToDoItem
    {
        [Required]
        public long Id { get; set; }

        [Required]
        public string Name { get; set; }

        [DefaultValue(false)]
        public bool IsComplete { get; set; }
    }
}
```

Industriel ingenieur Informatica

28

Gegenereerde JSON – info uit data-attributen

verplichte eigenschappen

standaardwaarde

```
components:
  schemas:
    ToDoItem:
      required:
        - id
        - name
        - isComplete
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        isComplete:
          type: boolean
          default: false
          additionalProperties: false
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

29

Swagger

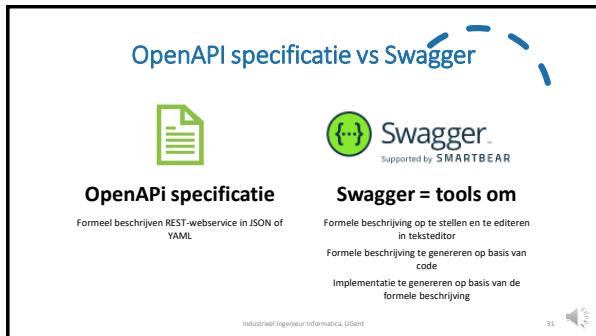
Drie tools

- Swagger Editor: specificatie schrijven
- Swagger UI: specificatie genereren, interactieve documentatie
- Swagger Codegen: specificatie → code (meer info op <https://github.com/swagger-api/swagger-codegen>)
 - Ondersteunt meerdere talen

Ada, C# (ASP.NET Core, NancyFx), C++ (Pistache, Restbed), Erlang, Go, Haskell (Servant), Java (MSF4J, Spring, Undertow, JAX-RS: CDI, CXF, Inflector, RestEasy, Play Framework, PKMST), Kotlin, PHP (Lumen, Slim, Silex, Symfony, Zend Expressive), Python (Flask), NodeJS, Ruby (Sinatra, Rails5), Rust (rust-server), Scala (Finch, Lagom, Scalatra)

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

30

1.8. OPENAPI SPECIFICATIE EN SWAGGERHOOFDSTUK 1. REST WEB SERVICES

31

Hoofdstuk 2

Object Relational Mapping

In dit hoofdstuk bespreken we de basisprincipes van ORM (*Object Relational Mapping*, [AA13a]). In de voorbeelden gebruiken we JPA ([JPAa]) om deze principes te illustreren.

Vele applicaties worden ontwikkeld in een object-georiënteerde taal zoals Java of C# en gebruiken een relationele gegevensbank om informatie te bewaren. In een object-georiënteerde taal bestaan applicaties uit objecten die data bevatten. De methodes van de objecten bepalen het gedrag van het programma. In een relationele gegevensbank worden gegevens opgeslagen in tabellen en gemanipuleerd met *stored procedures* en SQL-opdrachten. Het is duidelijk dat de principes van deze twee types technologieën, object-georiënteerd programmeren en relationele databanken, sterk uiteen lopen. Dit verschil in aanpak werd in de jaren '90 de “object-relational impedance mismatch” genoemd. ([AA13b])

2.1 The object-relational impedance mismatch

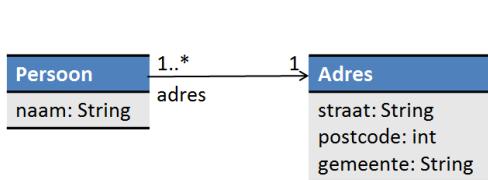
“The object-relational impedance mismatch” is een verzamelnaam voor de problemen die opduiken bij het gebruik van een relationele gegevensbank in software geschreven in een object-georiënteerde taal. Het object-georiënteerd paradigma is gebaseerd op succesvolle softwareontwikkelingsprincipes en stelt data voor als een graaf van objecten. Het relationele paradigma daarentegen heeft als basis een aantal wiskundige principes (predicatenlogica en waarheidsuitspraken) en stelt data voor in tabelformaat. Het verschil in aanpak wordt onder andere duidelijk in het ophalen van informatie. In een object-georiënteerd programma maak je gebruik van de relaties tussen objecten om data te verkrijgen terwijl in een SQL-opdracht de datarijen (*records*) van tabellen gecombineerd worden (*joins*). Een aantal andere verschillen worden hieronder beschreven.

Identiteit

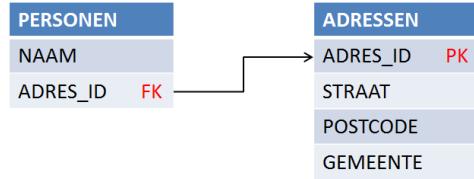
Eén van de verschillen in beide paradigma's is het concept identiteit, m.a.w. wanneer stellen twee entiteiten hetzelfde voor. In een gegevensbank wordt de uniciteit van een rij bepaald door een primaire sleutel. Twee objecten daarentegen zijn gelijk als ze dezelfde referentie hebben (en dus verwijzen naar dezelfde geheugenplaats). Daarnaast is het ook mogelijk om na te gaan of twee objecten hetzelfde voorstellen. Dit is het geval als het resultaat van de `equals`-methode de waarde `true` is. In een object-georiënteerde taal is er dus een verschil tussen objectidentiteit en objectgelijkheid.

Relaties

Object-georiënteerde talen realiseren relaties tussen objecten met referenties. Een relatie kan een éénrichtingsrelatie (*unidirectional*) zijn: het ene object kent het andere maar niet omgekeerd (zie figuur 2.1); of een tweerichtingsrelatie (*bidirectional*). In het laatste geval zijn er eigenlijk twee éénrichtingsrelaties. In relationele gegevensbanken worden relaties bepaald door verwijssleutels (*foreign keys*) (zie figuur 2.2).



Figuur 2.1: Eénrichtingsrelatie tussen objecten



Figuur 2.2: Relatie tussen tabellen

Overerving

Eén van de concepten van een object-georiënteerde taal is overerving. In de meeste gegevensbanken bestaat dit concept echter niet. Er bestaan verschillende oplossingen om overerving af te beelden op een tabelstructuur. (zie §2.5.2)

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING**2.2. ORM-PRINCIPES**

2.2 ORM-principes

ORM (Object Relational Mapping) | Veerle Ongenae

1

ORM (Object Relational Mapping)

- **Applicatie**
 - Software
 - Objecttechnologie (bv. Java, C#, ...)
 - Data
 - Relationale gegevensbanken
- **Afbeelding (mapping) tussen**
 - Data in objecten
 - Data in tabellen
- **Hoe realiseren?**

2

Verschil OO-concepten en relationele databanken: types versus tabellen

Klassen ↔ Tabellen
of

| Person | | Adres | | Person | |
|-------------------|----------------|-----------------|---------------------|----------|----------|
| +naam: String | +adres: Adres | +straat: String | +huisnummer: String | +naam | +straat |
| +straat: String | +postcode: int | +postcode | +gemeente: String | straat | postcode |
| +gemeente: String | | gemeente | | adresID | PK |
| | | | | naam | adresID |
| | | | | adresID | FK |
| | | | | gemeente | |

3

Object-relational impedance mismatch

- Verschil OO-concepten en relationele databanken
 - Identiteit
 - Relaties
 - Overerving

4

Verschil OO-concepten en relationele databanken: identiteit

- Identiteit?
 - Wanneer zijn twee entiteiten hetzelfde?
- DB
 - Bepaald door primaire sleutel
- OO
 - Zelfde referentie (verwijzen naar dezelfde geheugenplaats)
 - Resultaat equals-methode
 - Bv. een aantal attributen zijn dezelfde

5

Identiteit

- Primaire sleutel (DB)
 - Geen corresponderende eigenschap in domeinmodel
 - Besmetten domeinmodel met extra attribuut
- Automatische gegenereerde sleutel
 - Vergelijken objecten enkel mogelijk indien toegevoegd aan DB
- Object-identiteit is moeilijk te mappen naar record-identiteit

6

2.2. ORM-PRINCIPES

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

Schaduw informatie

- Shadow Information
- Data
 - Nodig om informatie te bewaren (in DB)
 - Bv. Primaire sleutel
 - Geen deel van business model

FACULTEIT INGEGENIERSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



7

ORM in Java

- JPA (Java Persistence API)
- Mapping tussen klassen en tabellen met annotaties

Industriel Ingenieur Informatica, UGent



8

Dataklassen

- Moeten JavaBeans zijn
 - Defaultconstructor
 - Getters en setters voor eigenschappen

FACULTEIT INGEGENIERSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



9

Entity

- Entiteit
- Beheerd door JPA-framework
- Klasse ↔ Tabel
 - Attribuut name: tabelnaam
 - Optioneel
- Object ↔ Record/rij

```
import javax.persistence.*;
@Entity
@Table(name = "sportclub")
public class Sportclub { ... }
```

FACULTEIT INGEGENIERSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



10

ID

- ID vereist
 - Samengestelde sleutels kan ook → weinig gebruikt
- name-attribuut → kolom in tabel

FACULTEIT INGEGENIERSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



11

ID

- Generator
 - Hoe id aangemaakt bij nieuwe objecten?
 - AUTO
 - IDENTITY
 - SEQUENCE
 - TABLE

FACULTEIT INGEGENIERSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



12

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING**2.2. ORM-PRINCIPES**

Object↔ Tabel ??



`///@Basic
///@Column(name = "NAAM")
public String getNaam() { ... }
public void setNaam(String naam) { ... }`

- Eigenschap v.e. object → nul of meerdere kolommen
 - Nul?
 - Sommige informatie moet niet bewaard worden
 - Vb. Gemiddelde
 - Berekend op basis van informatie uit DB
 - Meer dan één?
 - Eigenschap = object
 - Heeft op zijn beurt eigenschappen
 - Vb. Persoon-object heeft als eigenschap een Adres-object

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industrial Ingenieur Informatica, UGent 13

13

Eigenschap ↔ kolom



`///@Basic
///@Column(name = "NAAM")
public String getNaam() { ... }
public void setNaam(String naam) { ... }`

- **@Basic**
 - Primitieve types (of wrappers)
 - Optioneel
- **@Column**
 - name-attribuut → kolom in tabel
 - Optioneel
- Niet bewaren
 - **@Transient**

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industrial Ingenieur Informatica, UGent 14

14

ORM-framework



- API basisbewerkingen

- CRUD (Create Read Update Delete)

- Taal queries

- Maakt gebruik van attributen en klassen

- Metadata om mapping te definiëren

- Technieken om transactionele objecten te gebruiken

- Dirty checking: object gewijzigd?
- Lazy association fetching: attributen pas ophalen als ze nodig zijn
- Andere optimalisaties

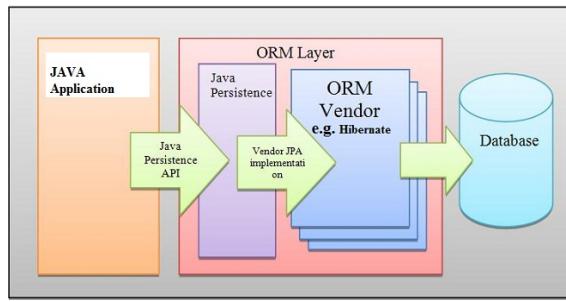
Industrial Ingenieur Informatica, UGent 15

15

2.3 JPA en Hibernate

JPA

JPA (Java Persistence API, [JPAA] en [Das08]) is een gestandariseerde Java-API om een datalaag te ontwikkelen volgens het ORM-principe waarbij de gegevens in een relationele gegevensbank bewaard worden (*persist*). Deze API specificeert de toegang tot, het bewaren van en het beheren van gegevens tussen Java-objecten en een relationele databank. Net zoals JDBC (Hoofdstuk 5) is JPA een specificatie die vooral bestaat uit interfaces. Het equivalent van een JDBC-driver is hier een JPA-implementatie. Eén mogelijke JPA-implementatie is Hibernate. (zie figuur 2.3)



Figuur 2.3: ORM-laag (bron [JPAC])

Hibernate

Hibernate is een ORM-framework dat in 2001 ontwikkeld werd door Gavin King als reactie op de complexiteit van Container Managed Persistence in de EJB(Enterprise JavaBeans)-specificatie.

Het framework is gepubliceerd als een Open Source framework dat valt onder de GNU Lesser General Public License (LGPL 2.1). Dit betekent dat Hibernate *Open Source* software is, maar dat software die gebruik maakt van Hibernate zelf geen *Open Source* hoeft te zijn. In 2003 werd Hibernate deel van JBoss Inc2 dat op zijn beurt in 2006 onderdeel werd van RedHat Inc.

Onder invloed van het succes van ORM-frameworks zoals Hibernate is ook de specificatie van de CMP EJB's sterk gewijzigd. JPA ontstond als onderdeel van EJB 3.0. Hibernate, vanaf versie 3.5, volgt de JPA 2.0-specificatie (gepubliceerd in 2009). In deze cursus gebruiken we Hibernate 4.x.

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.4. JPA IN SPRING

2.4 JPA in Spring

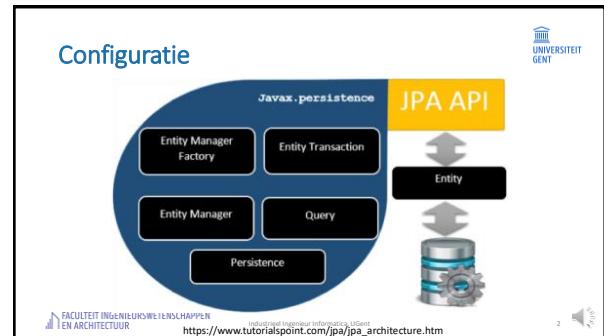
JPA in Spring

Veerle Ongenae

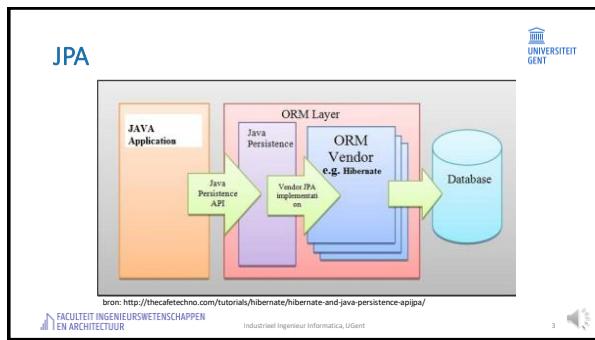
FACULTEIT INGEEIERSWETENSCHAPPEN EN ARCHITECTUUR
Industrial Ingénieur Informatique, UGent

bron: https://www.tutorialspoint.com/jpa/jpa_architecture.htm

1



2



3

Configuratie JPA in Spring (pom.xml)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

FACULTEIT INGEEIERSWETENSCHAPPEN EN ARCHITECTUUR
Industrial Ingénieur Informatique, UGent

4

Entiteiten: klassen annoteren

- Annotaties
 - Boven getter
 - Boven attribuut
 - Consistent (beide werkwijzen niet combineren)

```
import javax.persistence.*;
@Entity
@Table(name = "sportclub")
public class Sportclub {
    private Long id;
    private String naam;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    @Basic
    @Column(name = "NAAM")
    public String getNaam() { return naam; }
    public void setNaam(String naam) { this.naam = naam; }
}
```

FACULTEIT INGEEIERSWETENSCHAPPEN EN ARCHITECTUUR
Industrial Ingénieur Informatique, UGent

5

Repository

- CRUD-operaties voor entiteiten

```
public interface SportclubRepository extends JpaRepository<Sportclub, Long > { }
```

- Interface → Spring genereert de implementatie
- JpaRepository (<https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>)
- Methodes om entiteiten op te halen, te bewaren, aan te passen, ...

FACULTEIT INGEEIERSWETENSCHAPPEN EN ARCHITECTUUR
Industrial Ingénieur Informatique, UGent

6

2.4. JPA IN SPRING

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

JpaRepository

- Objecten bewaren

```
SportclubRepository repository; // geïnjecteerd
...
repository.save(new Sportclub("Mijn favoriete sportclub"));
```

- Alle objecten opvragen

```
SportclubRepository repository; // geïnjecteerd
...
repository.findAll().forEach(club -> log.info(club.getNaam()));
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

7

JpaRepository

- Een object ophalen op id

```
SportclubRepository repository; // geïnjecteerd
...
Sportclub club = repository.findById(1L);
log.info(club.getNaam());
```

- Andere methodes
 - flush → aanpassingen doorsturen naar de database
 - saveAll → meerdere entiteiten bewaren of aanpassen
 - count → aantal entiteiten
 - delete, deleteAll → één of meerdere entiteiten verwijderen
 - existById → bestaat er een entiteit met de gegeven id?

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

8

Extra zoekmethodes

- Repository – interface

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {
    List<Customer> findByLastName(String lastName);
```

- Definieert welke zoekmethodes er beschikbaar zijn
- <https://docs.spring.io/spring-data/data-jpa/docs/1.5.x/reference/html/jpa.repositories.html>

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

9

Configuratie databank

- application.properties
 - Naam/waarde-paren

```
spring.datasource.url=jdbc:mysql://localhost:3306/iiidb?serverTimezone=UTC
spring.datasource.username=iis
spring.datasource.password=iipwd
spring.jpa.properties.javax.persistence.schema-generation.database.action=create
spring.jpa.properties.javax.persistence.schema-generation.scripts.action=create
spring.jpa.properties.javax.persistence.schema-generation.scripts.create-target=create.sql
spring.jpa.properties.javax.persistence.schema-generation.scripts.create-source=metadata
```

resources
application.properties

Locatie databank
Login en wachtwoord
Databankstructuur genereren
Scripts genereren

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

10

Databankstructuur genereren

- Standaard **create**

```
spring.jpa.properties.javax.persistence.schema-generation.database.action=create
```

Niet nodig

- Alternatieven

| Instelling | Betekenis |
|-----------------|---|
| none | Niets aanmaken of verwijderen |
| create | De eerste keer databankstructuur aanmaken |
| drop-and-create | Telkens structuur verwijderen en opnieuw aanmaken |
| drop | Structuur verwijderen |

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

11

2.5 Objecten afbeelden

In deze paragraaf bekijken we hoe je informatie uit objecten kan afbeelden op tabellen en omgekeerd zonder zelf JDBC(Hoofdstuk 5)- of ADO.NET(Hoofdstuk 6)-code te schrijven.

Om aan te geven hoe je objecten vertaald (*mapped*) naar één of meerdere rijen in gegevensbank zijn er twee opties: ofwel beschrijf je de afbeelding (*mapping*) in een XML-bestand ofwel voeg je annotaties toe aan je Java-klassen. Enkel het gebruik van annotaties behoort tot de JPA-specificatie. Hibernate ondersteunt beide werkwijzen.

2.5.1 Eigenschappen

Objecten die ORM-frameworks beheren, noemen we entiteiten. Het framework zal deze objecten kunnen aanmaken, bewaren, ophalen, aanpassen, ... Om informatie in de objecten om te zetten naar gegevens in de gegevensbank en omgekeerd, zal er in de *mapping* onder andere beschreven worden hoe klassen worden afgebeeld op tabellen en hoe eigenschappen van klassen worden vertaald naar kolommen.

De attributen of eigenschappen van een klasse worden afgebeeld op één of meerdere kolommen van een tabel in een relationele gegevensbank.

Niet alle eigenschappen hoeven bijgehouden worden in de databank. Sommigen kunnen het resultaat van een berekening zijn, bijvoorbeeld de oppervlakte en de omtrek van een rechthoek wordt berekend op basis van de lengte en de breedte. (zie figuur 2.4)

Eigenschappen van een primitief type of van het type *String* corresponderen met één kolom uit een tabel.

| Rechthoek | RECHTHOEKEN |
|--------------------------|-------------|
| getBreedte(): double | BREEDTE |
| getHoogte(): double | HOOGTE |
| getOppervlakte(): double | |
| getOmtrek(): double | |

Figuur 2.4: Eigenschappen afbeelden op kolommen

Hierbij worden SQL-types omgezet in Java-types en omgekeerd (zie ook §5.4.2).

De klassen van de entiteiten, beheerd door een ORM-framework, moeten JavaBeans voorstellen. Dit betekent

- dat de klasse private instantievariabelen heeft waarvoor getters en setters voorzien worden;
- dat de klasse een defaultconstructor heeft;
- en dat de klasse serialiseerbaar is (implementeert de interface `Serializable`)

De defaultconstructor is nodig omdat Hibernate objecten aanmaakt worden via Java Reflection. Voor Hibernate mogen constructoren, getters en setters private zijn.

Unieke identificatie

Verder moet de klasse één eigenschap (*property*) hebben die de unieke identificatie van de entiteit voorstelt. Deze eigenschap correspondeert met de primaire sleutel in de gegevensbank. Soms behoort deze eigenschap niet echt tot het business-model van de applicatie en wordt dan schaduwinformatie genoemd

(*shadow information*). Indien de unieke identificatie automatisch gegenereerd wordt en dus geen deel uitmaakt van het domeinmodel, dan geeft Hibernate een nieuwe entiteit pas een unieke identificatie als het object bewaard wordt. Aangezien we de unieke identificatie van een entiteit best niet veranderen, is de setter hiervan private.

Voorbeeld

In het onderstaande voorbeeld illustreren we hoe een unieke identificatie `id` (type `int`) en een eigenschap `naam` van het type `String` van een klasse `Sportclub` (zie listing 2.1) afgebeeld worden op een tabel `SPORTCLUBS` met twee kolommen: `ID` en `NAAM` (zie figuur 2.5).

De *mapping* kan op twee manieren gerealiseerd worden:

- ofwel door JPA-annotaties ([JPAb]) toe te voegen aan de Java-klasse (zie listing 2.1)
- ofwel door een XML-bestand `Sportclub.hbm.xml` te maken waarin de afbeelding beschreven wordt.

| SPORTCLUBS | |
|------------|----|
| ID | PK |
| NAAM | |

Figuur 2.5: Tabel SPORTCLUBS in gegevensbank

```

package voorbeeldORMAnnotaties.data;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Table(name = "sportclubs")
public class Sportclub implements Serializable {

    private int id;
    private String naam;

    public Sportclub() {
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    //@Basic
    //@Column(name = "NAAM")
    public String getNaam() {
        return naam;
    }

    public void setNaam(String naam) {
        this.naam = naam;
    }
}

```

Codevoorbeeld 2.1: Mapping met annotaties (JPA)

Klassen

In het voorbeeld wordt de klasse `Sportclub` afgebeeld op de tabel `SPORTCLUBS`. Met annotaties realiseer je dit zo:

- De annotaties behoort tot de package `javax.persistence.*`. Die moet je dus in je klasse importeren.
- De annotatie `@Entity` geeft aan dat objecten van de klasse `Sportclub` beheerd moeten worden door het ORM-framework.
- De annotatie `@Table(name = ``sportclubs``)` bepaalt dat de informatie van de `Sportclub`-objecten bewaard wordt in de tabel `SPORTCLUBS`. Deze annotatie mag weggelaten worden als de naam van de klasse en de naam van de tabel dezelfde zijn. Merk op dat de namen van tabellen en kolommen niet hoofdlettergevoelig zijn.

Een alternatief voor annotaties is het gebruik van XML-bestanden om de *mapping* vast te leggen. Dit kan enkel in Hibernate en niet in JPA, m.a.w. de extra-abstractielag die JPA biedt, valt dan weg.

De naam van het *mapping*-bestand is dezelfde als de naam van de klasse plus de extensie `.hbm.xml`.

Mapping van de unieke identificatie

Voor elke entiteit moet er een eigenschap zijn die de unieke identificatie voorstelt en correspondeert met de primaire sleutel in de gegevensbank. Eenmaal toegekend mag deze eigenschap niet meer gewijzigd worden. Dit kan je realiseren door de setter private te declareren.

De waarde van deze eigenschap kan gegenereerd worden door de databank of kan een bestaande eigenschap zijn uit het domeinmodel. Als deze eigenschap automatisch aangemaakt wordt, dan zijn er verschillende strategieën mogelijk, afhankelijk van de gekozen databank (zie tabel 2.1). De standaardwaarde voor de generator is `assigned`. Dit betekent dat in de applicatie een unieke identificatie moet toegekend worden voordat de entiteit bewaard wordt en dat deze eigenschap dus niet automatisch aangemaakt wordt.

| JPA | Omschrijving |
|----------|---|
| Identity | De unieke identificatie correspondeert met een kolom van het type <code>IDENTITY</code> of <code>AUTO_INCREMENT</code> . In dit geval wordt de waarde van deze eigenschap gegenereerd wanneer de rij bewaard wordt. De waarde is een geheel getal (<code>long</code> , <code>short</code> of <code>int</code>). |
| Sequence | De gehele waarde (<code>long</code> , <code>short</code> of <code>int</code>) van de unieke identificatie wordt gegenereerd door een gegevensbank- <code>SEQUENCE</code> . |
| Table | De unieke identificatie (<code>long</code>) wordt aangemaakt op basis van een tabel en een kolom. |

Tabel 2.1: Strategieën automatische aanmaken id's

De mapping van de unieke identificatie verloopt als volgt. (zie ook listing 2.1)

- Voeg de annotatie `@Id` toe aan de eigenschap die de unieke identificatie voorstelt. Deze eigenschap hoeft NIET de naam `id` te hebben.
- Indien de unieke identificatie automatisch gegenereerd wordt, voeg dan ook de annotatie `@GeneratedValue` toe waarbij je het attribuut `strategy` op de juiste waarde (zie tabel 2.1) instelt.

Ook samengestelde primaire sleutels zijn mogelijk. Meer informatie hierover vind je in de JPA Tutorial ([JPAa]).

Mapping van eenvoudige eigenschappen

Met eenvoudige eigenschappen bedoelen we eigenschappen van een primitief type, van het type `String`, van het type `Date`, van het type `BigInteger`, ... Dit zijn Java-types die eenvoudig om te zetten zijn naar gegevensbanktypes.

Elke eigenschap van een entiteit die bewaard wordt in de gegevensbank, moet een getter en een setter hebben.

- Bij het gebruik van JPA wordt de annotatie toegevoegd aan de getter van de eigenschap. Het is ook mogelijk om niet de getters, maar de instantievariabelen zelf te annoteren.
- De annotatie `@Basic` geeft aan dat de geannoteerde eigenschap een eenvoudige eigenschap is. Deze annotatie is optioneel en mag weggetallen worden.
- Met de `@Column`-annotatie koppel je de eigenschap aan een specifieke kolom, bepaald door het attribuut `name`, in de tabel. Als je deze annotatie weglaat dan zijn de naam van de kolom en de eigenschap hetzelfde.
- Eigenschappen die niet bewaard worden in de gegevensbank worden geannoteerd met `@Transient`.

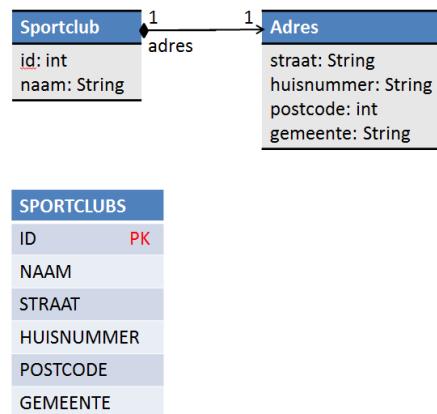
Value-objecten

Als een eigenschap van een entiteit niet van een eenvoudig type is (een primitief type, `String`, `Date`, `BigInteger`, ...), dan zijn er twee opties. Ofwel is de eigenschap op zijn beurt ook een entiteit, en bestaat dus als apart item in het domeinmodel (zie §2.5.3). Ofwel stelt de eigenschap een object voor dat alleen bestaat binnen de entiteit (compositie). In dat geval spreken we van een *value*- of *embedded*-object. Er zijn drie criteria om onderscheid te maken tussen een entiteit en een *value*-object:

1. Een *value*-object hoort bij juist één entiteit. Een andere entiteit kan onmogelijk een verwijzing hebben naar hetzelfde *value*-object.
2. Een *value*-object wordt verwijderd als de entiteit verwijderd wordt. Het kan dus nooit op zichzelf bestaan.
3. Een entiteit heeft een unieke identificatie nodig. Een *value*-object wordt altijd opgevraagd via de entiteit waartoe het behoort en heeft geen unieke identificatie.

In het volgend voorbeeld gaan we ervan uit dat een `Adres`-object niet gedeeld wordt tussen verschillende `Sportclub`-entiteiten. In tegenstelling tot eenvoudige eigenschappen kan deze eigenschap niet afgebeeld worden op één kolom, maar zullen er meerdere kolommen nodig zijn. (zie figuur 2.6)

- Een klasse waarvan de instanties *value*-objecten zijn, krijgt de annotatie `@Embeddable` ipv de annotatie `@Entiteit`. In het voorbeeld zijn de eigenschappen van het type `Adres` *value*-objecten. (zie listing 2.3).
- Aan de eigenschap die een *value*-object voorstelt, wordt de annotatie `@Embedded` toegekend. Deze annotatie mag weggetallen worden omdat op basis van het type (de klasse) reeds duidelijk is dat het om een *value*-object gaat. (zie eigenschap `adres` in listing 2.2)

Figuur 2.6: Voorbeeld *value-object*

```

package voorbeeldORMAnnotaties.data;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Table(name = "sportclubs")
public class Sportclub implements Serializable {

    private int id;
    private String naam;
    private Adres adres;

    ...

    // @Embedded
    public Adres getAdres() {
        return adres;
    }

    public void setAdres(Adres adres) {
        this.adres = adres;
    }

}
  
```

Codevoorbeeld 2.2: JavaBean Sportclub met *value-object* Adres

```

package voorbeeldORMAnnotaties.data;

import java.io.Serializable;
import javax.persistence.Embeddable;

@Embeddable
public class Adres implements Serializable {
    private String straat;
    private String huisnummer;
    private int postcode;
    private String gemeente;

    public String getGemeente() {
        return gemeente;
    }

    public void setGemeente(String gemeente) {
        this.gemeente = gemeente;
    }

    public String getHuisnummer() {
        return huisnummer;
    }

    public void setHuisnummer(String huisnummer) {
  
```

```

    this.huisnummer = huisnummer;
}

public int getPostcode() {
    return postcode;
}

public void setPostcode(int postcode) {
    this.postcode = postcode;
}

public String getStraat() {
    return straat;
}

public void setStraat(String straat) {
    this.straat = straat;
}
}

```

Codevoorbeeld 2.3: Klasse voor *value*-objecten Adres

2.5.2 Overerving

De meeste relationele gegevensbanken hebben geen ingebouwde ondersteuning voor overerving en dus moet je een expliciete *mapping* tussen de klassen in een klassenhiërarchie en tabellen definiëren. In deze paragraaf lichten we verschillende mogelijke opties toe om deze *mapping* te realiseren.

- De volledige klassenhiërarchie wordt afgebeeld op één tabel.
- Elke klasse wordt afgebeeld op één tabel.
- Elke niet-abstracte klasse wordt afgebeeld op één tabel.

We illustreren de principes van elke *mapping* aan de hand van de volgende klassenhiërarchie (zie figuur 2.7).

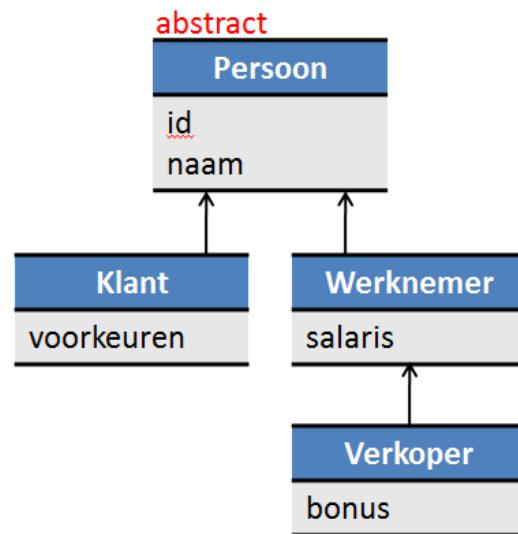
- De klasse `Persoon` is een abstracte klasse met een uniek attribuut `id` en een attribuut `naam`.
- De klasse `Klant` is een afgeleide klasse van `Persoon` met één attribuut `voorkeuren`.
- De klasse `Werknemer` is een afgeleide klasse van `Persoon` met één attribuut `salaris`.
- De klasse `Verkoper` is een afgeleide klasse van `Werknemer` met één attribuut `bonus`.

Volledige klassenhiërarchie naar één tabel

De eerste optie is om alle attributen van alle klassen in de hiërarchie in één tabel te bewaren. (zie figuur 2.8) In dit geval heeft de tabel één kolom voor alle eenvoudige eigenschappen van alle klassen in de hiërarchie en één extra kolom die het type van het object bepaalt, de *discriminator*-kolom genoemd. Zo zal bijvoorbeeld de letter `K` betekenen dat de rij de gegevens van een `Klant`-object bevat, de letter `W` die van een `Werknemer`-object en de letter `V` die van een `Verkoper`-object.

Deze methode is zeer efficiënt voor zoekopdrachten (*queries*) over meerdere types in de hiërarchie omdat er geen *joins* nodig zijn.

Een nadeel van deze methode is dat sommige kolommen `NULL`-waarden bevatten. Bijvoorbeeld de waarden voor de kolommen `SALARIS` en `BONUS` zullen steeds `NULL` zijn voor een rij die de gegevens van een `Klant`-object



Figuur 2.7: Voorbeeld overerving

| PERSOON | |
|--------------|----|
| PERSOON_ID | PK |
| PERSOON_TYPE | |
| NAAM | |
| VOORKEUREN | |
| SALARIS | |
| BONUS | |

Figuur 2.8: Volledige klassenhiërarchie naar één tabel

voorstelt. Dit betekent ook dat het niet mogelijk is om te eisen dat sommige attributen van de afgeleide klassen verplicht zijn.

Een ander nadeel is dat bij uitbreiding van de klassenhiërarchie met een extra klasse, er extra kolommen aan de tabel toegevoegd moeten worden.

- Aan de bovenste klasse in de klassenhiërarchie, `Persoon` in het voorbeeld, worden twee annotaties toegevoegd: `@Inheritance` en `@DiscriminatorColumn`.
- Het attribuut `strategy` van de annotatie `@Inheritance` bepaalt hoe de klassenhiërarchie afgebeeld wordt in de gegevensbank. Het type van dit attribuut is de `enum InheritanceType`. Deze opsomming heeft drie mogelijke waarden: `JOINED`, `SINGLE_TABLE` en `TABLE_PER_CLASS`. De standaardwaarde voor het attribuut `strategy` is `SINGLE_TABLE`.
- Het attribuut `DiscriminatorColumn` definieert de kolom in de tabel die de *discriminator*-waarde bevat. De attributen `name` en `discriminatorType` leggen de naam en het type van de kolom vast. Mogelijke types zijn `DiscriminatorType.CHAR`, `DiscriminatorType.INTEGER` en `DiscriminatorType.STRING` (standaardwaarde).
- Voor elke afgeleide klasse in de hiërarchie moet je enkel nog de waarde van de *discriminator* specificeren met de annotatie `@DiscriminatorValue`.

```
package voorbeeldORMAnnotaties.TabelPerHierarchie;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "PERSOON_TYPE",
    discriminatorType= DiscriminatorType.STRING)
public abstract class Persoon implements Serializable {
    ...
}
```

```
package voorbeeldORMAnnotaties.TabelPerHierarchie;

import javax.persistence.*;

@Entity
@DiscriminatorValue("K")
public class Klant extends Persoon {
    ...
}
```

```
package voorbeeldORMAnnotaties.TabelPerHierarchie;

import javax.persistence.*;

@Entity
@DiscriminatorValue("W")
public class Werknemer extends Persoon {
    ...
}
```

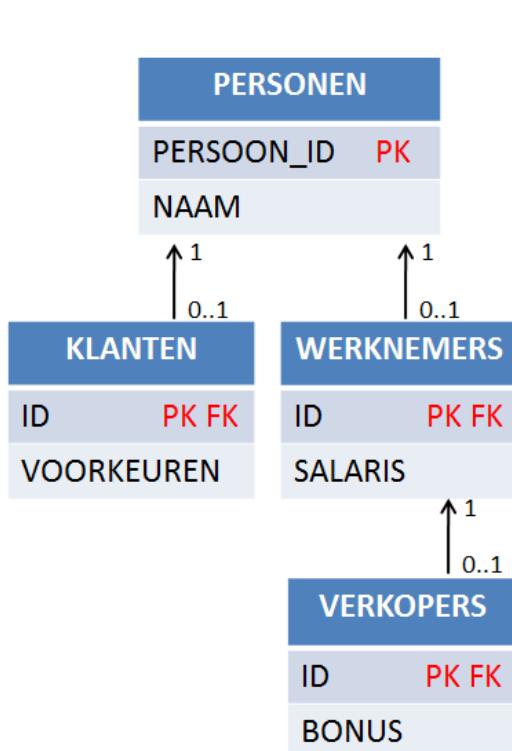
```
package voorbeeldORMAnnotaties.TabelPerHierarchie;

import javax.persistence.*;

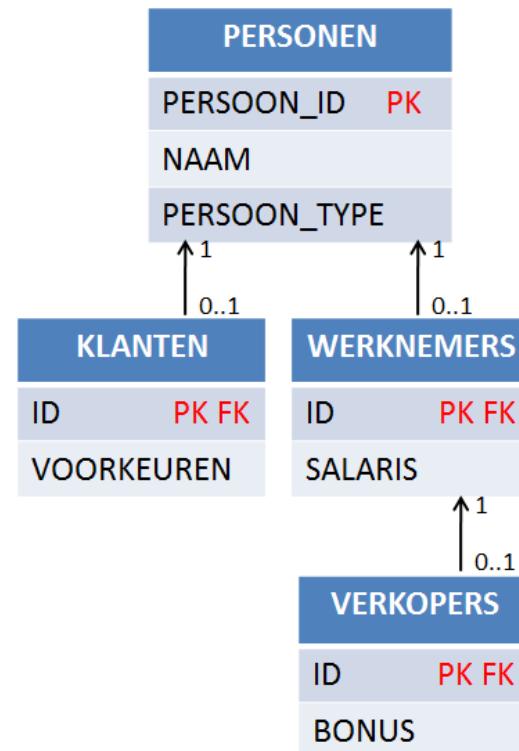
@Entity
@DiscriminatorValue("V")
public class Verkoper extends Werknemer {
    ...
}
```

Elke klasse naar één tabel

Een tweede optie is om elke klasse in de hiërarchie af te beelden op één tabel. Attributen van de superklasse worden dan bewaard in een tabel die de superklasse voorstelt. Enkel attributen specifiek voor een afgeleide klasse worden bewaard in een tabel die het afgeleid type voorstelt. De gegevens van een object waarvan het type een afgeleide klasse is, worden dus bewaard in meerdere tabellen (overgeërfd attributen worden bewaard in de tabel van de superklasse, specifieke attributen worden bewaard in de tabel van het afgeleid type), waarbij de unieke identificatie van het object respectievelijk als primaire sleutel en verwijzingssleutel herhaald wordt in beide tabellen. Dit wordt ook wel *vertical mapping* genoemd. (zie figuur 2.9)



Figuur 2.9: Elke klasse naar één tabel



Figuur 2.10: Elke klasse naar één tabel met discriminator

Sommige JPA-providers verwachten om performantieredenen een extra *discriminator*-kolom in de tabel horende bij de basisklasse. Deze kolom geeft het afgeleid type van de instanties aan (zie figuur 2.10). Dit is niet nodig in Hibernate, maar bv. wel voor de standaardprovider van de GlassFish Server.

Bij uitbreiding van de klassenhiërarchie met een extra klasse, moet bij deze methode enkel een nieuwe tabel toegevoegd worden aan de databank. Het aanpassen van een bestaand type in de hiërarchie heeft enkel invloed op de corresponderende tabel en niet op de andere tabellen.

Een mogelijk nadeel van deze strategie is dat elke zoekopdracht één of meerdere *joins* gebruikt. Dit kan een negatief effect hebben op de performantie in het geval van grote tabellen of grote klassenhiërachieën.

- De annotatie `@Inheritance` van de basisklasse van de klassenhiërarchie heeft in dit geval de waarde `InheritanceType.JOINED` voor het attribuut `strategy`.
- Voor elke afgeleide klasse in de hiërarchie moet je de naam van de kolom met de primaire sleutel specificeren met de annotatie `@PrimaryKeyJoinColumn` en het attribuut `name`.

- Indien een *discriminator*-kolom gewenst is, dan moet aan de basisklasse ook de annotatie `@Discriminator` toegevoegd worden.

```
package voorbeeldORMAnnotaties.TabelPerKlasse;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name="PERSONEN")
public abstract class Persoon implements Serializable {
    ...
}
```

```
package voorbeeldORMAnnotaties.TabelPerKlasse;

import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="WERKNEMERS")
@PrimaryKeyJoinColumn(name="ID")
public class Werknemer extends Persoon {
    ...
}
```

```
package voorbeeldORMAnnotaties.TabelPerKlasse;

import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="VERKOPERS")
@PrimaryKeyJoinColumn(name="ID")
public class Verkoper extends Werknemer {
    ...
}
```

Elke concrete klasse naar één tabel

Een derde optie is om enkel de niet-abstracte klassen uit een klassenhiërarchie af te beelden op een tabel. Elke tabel bevat dan alle gegevens (inclusief de overgeërfde gegevens van de superklassen) van de objecten van dat subtype.

| KLANT | WERKNEMER | VERKOPER |
|------------|-----------|-----------|
| PERSOON_ID | PERSON_ID | PERSON_ID |
| NAAM | NAAM | NAAM |
| VOORKEUREN | SALARIS | SALARIS |
| | | BONUS |

Figuur 2.11: Elke concrete klasse naar één tabel

Deze manier van werken wordt *horizontal mapping* genoemd en is performanter dan *vertical mapping*, maar is dan weer nadeliger als men refactoring moet toepassen en bijvoorbeeld attributen wil toevoegen aan het supertype. Niet alle JPA-providers ondersteunen deze methode.

Een andere moeilijkheid bij deze strategie is dat de primaire sleutel gedeeld moet worden over verschillende tabellen. In voorbeeld 2.7 is elk `Klant`, `Werknemer` en `Verkoper`-object ook een `Persoon`-object. Dit betekent dat ze een unieke identificatie als `Persoon`-object hebben. In de gegevensbank (zie figuur 2.11) moeten de tabellen `KLANT`, `WERKNEMER` en `VERKOPER` dus een primaire sleutel delen.

- De annotatie `@Inheritance` van de basisklasse van de klassenhiërarchie heeft in dit geval de waarde `InheritanceType.TABLE_PER_CLASS` voor het attribuut `strategy` (zie listing 2.4).
- Aangezien de primaire sleutel tussen verschillende tabellen gedeeld moet worden, kan hier niet gebruik gemaakt worden van een kolom van het type `IDENTITY` of `AUTO_INCREMENT`. Als de gegevensbank `sequences` ondersteunt, is dat een mogelijk alternatief. De annotatie `@GeneratedValue` zal dus een andere waarde moeten hebben voor het attribuut `strategy` (zie listing 2.4).
- De afgeleide klassen hebben geen extra annotatie nodig naast `@Entity` (zie listing 2.5).

```
package voorbeeldORMAnnotaties.tabelPerConcreteKlasse;

import java.io.Serializable;
import javax.persistence.*;

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Persoon implements Serializable {
    ...
    @Id
    @Column(name="PERSOON_ID")
    @GeneratedValue(strategy=GenerationType.SEQUENCE)

    public int getId() { ... }
    ...
}
```

Codevoorbeeld 2.4: Mapping één tabel per concrete klasse: annotatie superklasse

```
package voorbeeldORMAnnotaties.tabelPerConcreteKlasse;

import javax.persistence.Entity;

@Entity
public class Klant extends Persoon {
    ...
}
```

Codevoorbeeld 2.5: Mapping één tabel per concrete klasse: annotatie afgeleide klasse

2.5.3 Relaties

De laatste *mapping* die we bekijken is de afbeelding van relaties naar een tabelstructuur. In de objectgeoriënteerde wereld zijn er drie types relaties: associaties, aggregaties en composities. De onderliggende tabelstructuur is in alle drie de gevallen dezelfde, alhoewel de eigenschappen van de *mapping* kunnen verschillen.

In een objectgeoriënteerde taal realiseren instantievariabelen van een “niet-eenvoudig” type, relaties: een object heeft een referentie naar een ander object. Het type van deze referentie kan een andere klasse of een collectietype zijn. JPA ondersteunt de volgende collectietypes:

- `java.util.Collection`
- `java.util.Set`
- `java.util.List`

- `java.util.Map`

Relaties kunnen verder nog opgedeeld worden aan de hand van twee categorieën: multipliciteit en richting. In het geval van multipliciteit is er de volgende opdeling:

één-op-één In een één-op-éénrelatie is elke instantie van de klasse verwant met één instantie van een andere klasse. Bijvoorbeeld een werkzoekende heeft juist één CV en een CV kan maar van één werkzoekende zijn. Hier heeft de klasse `Werkzoekende` een één-op-éénrelatie met de klasse `CV`.

één-op-veel In een één-op-veelrelatie is elke instantie van de klasse verwant met meerdere instanties van een andere klasse. Bijvoorbeeld, een afdeling in een bedrijf heeft meerdere werknemers, maar een werknemer kan maar tot één afdeling behoren.

veel-op-één Dit is het omgekeerde van een één-op-veelrelatie, m.a.w. de relatie bekijken vanuit de andere richting. In een veel-op-éénrelatie zijn meerdere instanties van de klasse verwant met juist één instantie van een andere klasse. Zo kan bijvoorbeeld een werknemer behoren tot juist één afdeling, maar kunnen meerdere werknemers deel uitmaken van dezelfde afdeling.

veel-op-veel In een veel-op-veelrelatie zijn meerdere instanties van de klasse verwant met meerdere instanties van een andere klasse. Bijvoorbeeld een student kan meerdere opleidingsonderdelen volgen. En elk opleidingsonderdeel wordt opgenomen door meerdere studenten.

Eén-op-veel- en veel-op-veelrelaties worden geïmplementeerd gebruik makend van een collectieobject. De één-op-één- en veel-op-éénrelatie wordt gerealiseerd met een referentie naar een ander object.

De indeling volgens richting heeft twee types:

unidirectioneel In een unidirectionele of eenrichtingsrelatie heeft slechts één object in de relatie een attribuut of eigenschap die verwijst naar het andere object in de relatie. Het ene object kent het andere, maar niet omgekeerd. Bijvoorbeeld, een bestelling kan een verwijzing naar een product (dat besteld wordt) hebben, maar niet omgekeerd.

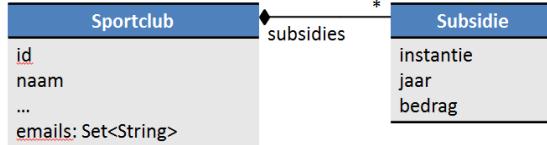
bidirectioneel In een bidirectionele of tweerichtingsrelatie hebben beide objecten uit de relatie een attribuut of eigenschap die naar elkaar verwijst: beide objecten kennen elkaar. Bijvoorbeeld, een `Werknemer`-object heeft een referentie naar een `CV`-object en omgekeerd.

Een relatie tussen twee objecten kan een relatie tussen twee entiteiten zijn of een relatie tussen een entiteit en een *value*-object. Ter herinnering *value*-objecten zijn objecten die enkel bestaan binnen een entiteit en die dus niet tot het domeinmodel behoren. In dat geval noemen we de relatie een compositie. De één-op-éénrelatie tussen een entiteit en een *value*-object werd besproken in §2.5.1.

Compositie met meerdere *value*-objecten

In deze paragraaf bespreken we hoe je een compositie met meerdere *value*-objecten kan afbeelden op een tabelstructuur. Op basis van het type van de objecten in de compositie onderscheiden we twee gevallen:

- een primitief type of `String`
- een referentietype, maar niet `String`



Figuur 2.12: Voorbeeld één-op-veel-compositie

| SPORTCLUBS | |
|------------|----|
| ID | PK |
| NAAM | |
| STRAAT | |
| HUISNUMMER | |
| POSTCODE | |
| GEMEENTE | |

| EMAILADRESSEN | |
|---------------|----|
| SPORTCLUB | FK |
| EMAIL | |

| SUBSIDIES | |
|-----------|----|
| SPORTCLUB | FK |
| INSTANTIE | |
| JAAR | |
| BEDRAG | |

Figuur 2.13: Tabelstructuur één-op-veel-compositie

In beide gevallen wordt de compositie gerealiseerd met een collectieobject. In voorbeeld 2.12 heeft de klasse `Sportclub` een eigenschap `emails` van het type `Set<String>` en een eigenschap `subsidies` van het type `Set<Subsidie>`. Aangezien er meerdere emails en subsidies zijn voor één sportclub zullen deze elk in een aparte tabel bijgehouden worden. Naast de kolommen om de gegevens bij te houden zal er ook één kolom (`SPORTCLUB`) zijn met een verwijssleutel die de sportclub identificeert (zie 2.13).

Algemeen geldt dat een één-op-veelrelatie gerealiseerd wordt met een verwijssleutel in de tabel die gelinkt is met de “veel”-kant van de relatie.

- Met de annotatie `@ElementCollection` duid je aan dat de eigenschap een collectie van *value-objecten* voorstelt.
- De annotatie `CollectionTable` laat je toe om de naam van de tabel (attribuut `name`) en de naam van de kolom met de verwijssleutel, naar de primaire sleutel van de tabel van de entiteit (attribuut `joinColumns`), te specifiëren. De standaardwaarden zijn respectievelijk `KLASSE_EIGENSCHAP` en `KLASSE_IDKOOLM`. In het voorbeeld (listing 2.6) zouden de standaardwaarden dus `SPORTCLUB_EMAILS` voor de tabel en `SPORTCLUB_ID` voor de kolom met verwijssleutel zijn.
- De annotatie `@JoinColumn` specificeert een kolom met verwijssleutel die een associatie of een compositie realiseert.
- Met de annotatie `@AttributeOverrides` groepeer je een aantal `@AttributeOverride`-annotaties. De `@AttributeOverride`-annotatie gebruik je om de (al dan niet standaard) *mapping* van een *value-object* te overschrijven. In het voorbeeld (listing 2.6) definieert de annotatie `@AttributeOverrides` eigenlijk de standaardwaarden. Je zou deze annotatie dus kunnen weglaten en enkel de klasse `Subsidie` de notatie `@Embeddable` geven.

```

package voorbeeldORMAnnotaties.data;

import java.io.Serializable;
import java.util.Set;
import javax.persistence.*;

@Entity
@Table(name = "sportclubs")
public class Sportclub implements Serializable {

    ...
    @ElementCollection
    @CollectionTable(name = "emailadressen",
        joinColumns = @JoinColumn(name = "sportclub"))
    @Column(name = "email")
}
  
```

```

public Set<String> getEmails() {...}

public void setEmails(Set<String> emails) {...}

@ElementCollection
@CollectionTable(name = "subsidies",
    joinColumns = @JoinColumn(name = "sportclub"))
@AttributeOverrides({
    @AttributeOverride(name = "jaar", column = @Column(name = "jaar")),
    @AttributeOverride(name = "bedrag", column = @Column(name = "bedrag")),
    @AttributeOverride(name = "instantie",
        column = @Column(name = "instantie"))
})
public Set<Subsidie> getSubsidies() {...}
...
}

```

Codevoorbeeld 2.6: Klasse met *value*-objecten

Eén-op-veel en veel-op-één relaties

Unidirectionele veel-op-één relatie We starten met een unidirectionele veel-op-éénrelatie. In figuur 2.14 behoort een lid tot juist één sportclub. Een sportclub kan meerdere leden hebben. De klasse `Lid` heeft een eigenschap `club`, maar de klasse `Sportclub` heeft geen verwijzing naar zijn leden. Op tabelniveau wordt de relatie gerealiseerd door de kolom `SPORTCLUB` van de tabel `LEDEN` die een verwijssleutel is naar de kolom `ID` van de tabel `SPORTCLUBS`.



Figuur 2.14: Unidirectionele veel-op-éénrelatie

Om de unidirectionele veel-op-éénrelatie vast te leggen kunnen we een aantal zaken specifiëren:

- de eigenschap van de “veel”-kant die de relatie realiseert.
In het voorbeeld is dat de eigenschap `club` van de klasse `Lid`.
Plaats de annotatie `@ManyToOne` bij de getter van deze eigenschap. (zie listing 2.7)
- de kolom in de tabel van de “veel”-kant met de verwijssleutel naar de primaire sleutel van de “één-kant”.
In het voorbeeld bevat de kolom `SPORTCLUB` een verwijssleutel naar de `ID`-kolom van de tabel `SPORTCLUBS`. De standaardnaam van de kolom is “de naam van de eigenschap” + “_” + “kolomnaam primaire sleutel”. In het voorbeeld zou dit dus `CLUB_ID` zijn.
Om te kolom een eigen naam te geven, voeg je de annotatie `@JoinColumn` toe aan de getter van de eigenschap. Het attribuut `name` specificeert dan de naam van de kolom.
- het type van de “één”-kant.
In JPA bepaalt het type van de eigenschap dit type.

```

@Entity
@Table(name = "leden")

```

```
public class Lid implements Serializable {  
    ...  
  
    @ManyToOne  
    @JoinColumn(name="sportclub")  
    public Sportclub getClub() {...}  
}
```

Codevoorbeeld 2.7: Veel-op-éénrelatie met annotaties

Unidirectionele één-op-veel relatie

2.6. OVERERVING IN ORM

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.6 Overerving in ORM

ORM overerving | Veerle Ongenae

This slide illustrates inheritance mapping. It shows a class hierarchy where 'Person' is the base class, and 'Client', 'Worker', and 'Salesperson' are subclasses. Each subclass has its own specific attributes: 'voorkieuren' for Client, 'salaris' for Worker, and 'bonus' for Salesperson. Arrows indicate the inheritance relationship from the subclasses back to the base class.

1

Object ↔ Tabel ??

- Klasse → tabel
 - Vaak, maar niet altijd!
 - Soms anders o.w.v. performantie

This slide compares object-oriented concepts with relational database tables. It shows a class 'Person' with attributes 'id' and 'naam'. Below it, a table 'Person' is shown with columns 'PERSOON_ID' (primary key) and 'NAAM'. Other columns like 'VOORKEUREN', 'SALARIS', and 'BONUS' are also listed. A red arrow points from 'PERSOON_ID' to 'id' in the class diagram, labeled 'primaire sleutel' (primary key). A green arrow points from 'NAAM' to 'naam' in the class diagram, labeled 'type object' (type object).

2

Overerving

- Hoe vertaal je overerving naar een gegevensbank?

This slide provides examples of inheritance mapping. It shows three subclasses: 'Klant' (Client), 'Werknemer' (Worker), and 'Verkoper' (Salesperson), all derived from a common base class 'Person'. Each subclass has its own specific attributes: 'voorkieuren' for Klant, 'salaris' for Werknemer, and 'bonus' for Verkoper.

3

Tabel per klassenhiërarchie

This slide compares object-oriented concepts with relational database tables for inheritance mapping. It shows a class 'Person' with attributes 'id' and 'naam'. Below it, a table 'Person' is shown with columns 'PERSOON_ID' (primary key) and 'NAAM'. Other columns like 'VOORKEUREN', 'SALARIS', and 'BONUS' are also listed. A red arrow points from 'PERSOON_ID' to 'id' in the class diagram, labeled 'primaire sleutel' (primary key). A green arrow points from 'NAAM' to 'naam' in the class diagram, labeled 'type object' (type object). A note below states: 'Nadeel: verplichte velden niet mogelijk' (Disadvantage: mandatory fields not possible).

4

Annotaties

```

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "PERSOON_TYPE",
discriminatorType= DiscriminatorType.STRING)
public abstract class Persoon implements Serializable { ... }

@Entity
@DiscriminatorValue("K")
public class Klant extends Persoon { ... }

@Entity
@DiscriminatorValue("W")
public class Werknemer extends Persoon { ... }

@Entity
@DiscriminatorValue("V")
public class Verkoper extends Werknemer { ... }

```

This slide shows Java annotations for inheritance mapping. It defines an abstract base class 'Persoon' and three subclasses: 'Klant', 'Werknemer', and 'Verkoper'. Annotations include 'Entity', 'Inheritance', 'DiscriminatorColumn', and 'DiscriminatorValue'. Arrows point from the annotations to their corresponding descriptions: 'Type opslag overerving' (Storage mapping inheritance) and 'Waarde ~ objecttype' (Value ~ object type).

5

Elke klasse naar een tabel

This slide illustrates vertical mapping for inheritance mapping. It shows the same class hierarchy as before. To the right, a 'VERTICALE MAPPING' diagram shows each class mapped to a separate table. 'Person' maps to 'Person' (pk: 'persoonID'), 'Klant' maps to 'Klant' (fk: 'ID', pk: 'voorkieuren'), 'Werknemer' maps to 'Werknemer' (fk: 'ID', pk: 'salaris'), and 'Verkoper' maps to 'Verkoper' (fk: 'ID', pk: 'bonus'). Relationships are indicated by arrows between the tables.

6

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.6. OVERERVING IN ORM

Annotations

```

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name="PERSONEN")
public abstract class Persoon implements Serializable { ... }

@Entity
@Table(name="KLANTEN")
@PrimaryKeyJoinColumn(name="ID")
public class Klant extends Persoon { ... }

@Entity
@Table(name="WERKNEMERS")
@PrimaryKeyJoinColumn(name="ID")
public class Verkoper extends Werknemer { ... }

```

Type opslag overerving
Tabel

Kolom foreign key
- primary key

FACULTEIT INGENIEURSWESEN
EN ARCHITECTUUR

7

Efficiënter?

- In basistabel
 - Extra kolom ~ subtype

Persoon
PERSOON_ID PK
NAAM
PERSOON_TYPE

Klant
ID FK
VOORKEUREN

Werknemer
ID FK
SALARIS

Verkoper
ID FK
BONUS

FACULTEIT INGENIEU
EN ARCHITECTUUR

8

Annotations

```

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name="PERSONEN_EFFICIENT")
@DiscriminatorColumn(name = "PERSOON_TYPE", discriminatorType=DiscriminatorType.STRING)
public abstract class Persoon implements Serializable { ... }

@Entity
@DiscriminatorValue("K")
@Table(name="KLANTEN_EFFICIENT")
@PrimaryKeyJoinColumn(name="ID")
public class Klant extends Persoon { ... }

```

Type opslag overerving
Tabel

Kolom objecttype

Waarde objecttype

Kolom foreign key
- primary key

FACULTEIT INGENIEURSWESEN
EN ARCHITECTUUR

9

Elke concrete klasse naar eigen tabel

- Refactoring moeilijker
 - bv. attributen supertype toevoegen

Klant
PERSOON_ID → primaire sleutel
naam
voorkeuren

Werknemer
PERSOON_ID → primaire sleutel
naam
salaris

Verkoper
PERSOON_ID → primaire sleutel
naam
salaris
bonus

HORIZONTALE MAPPING

FACULTEIT INGENIEURSWESEN
EN ARCHITECTUUR

10

Annotations

```

@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Persoon implements Serializable {
    @Id
    @Column(name="PERSOON_ID")
    @GenericGenerator(name="generator", strategy="increment")
    @GeneratedValue(generator="generator")
    public int getId() { return id; }
}

@Entity
public class Klant extends Persoon { ... }

```

Type opslag overerving

Primaire sleutel gedeeld over verschillende tabellen

FACULTEIT INGENIEURSWESEN
EN ARCHITECTUUR

11

2.7. VALUE-OBJECTEN IN ORM

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.7 Value-objecten in ORM

ORM associaties: value-objecten

Veerle Ongenaé

1

Verschil OO-concepten en relationele databanken: associaties

- OO: Referenties naar objecten
- DB: Foreign key

| Person | Adres |
|---|--|
| +id: String +naam: String +adres: Adres | +adresID: String +straat: String +postcode: int +gemeente: String |

| Person | Adres |
|-----------------------------------|---|
| ID PK naam adresID FK | adresID PK straat postcode gemeente |

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industrial Ingenieur Informatica, UGent

2

Associaties: richting

- Richting (OO, niet DB)
 - Unidirectioneel
 - Bidirectioneel
 - Moeilijk om actueel te houden
- Multipliciteit
 - 1-op-1
 - 1-op-veel
 - Veel-op-veel
- Hoe vertalen?

unidirectioneel

bidirectioneel

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industrial Ingenieur Informatica, UGent

3

Opvragen geassocieerde gegevens

```

FacturatieGegevens fg = persoon.getFacturatieGegevens();
- Versus
SELECT nummer FROM facturatiegegevens f JOIN gebruikers g ON f.user_id =
g.user_id WHERE g.naam='gebruiker';
- Facturatiegegevens van n personen
  • Erst personen opvragen, dan facturatiegegevens
    ➤ (1+n) selects
  • Persoonsgegevens en facturatiegegevens tegelijk opvragen
    ➤ 1 select
  
```

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industrial Ingenieur Informatica, UGent

4

Relaties

- Value-objecten
- Relaties tussen entiteiten

5

Entiteiten en value-objecten

- Entiteit
 - Apart item in domeinmodel
 - Worden bewaard in de databank
 - Mapping vastleggen
 - Bv. Gebruiker, Factuur, ...
- Atributen van een entiteit
 - Andere entiteiten
 - Value-objecten
 - Deel van een entiteit (compositie)
 - Beteenisloos indien entiteit verwijderd is
 - Bv. emailadres van een persoon

6

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.7. VALUE-OBJECTEN IN ORM

Entiteiten en value-objecten

- Criteria
 - Value-object hoort bij **exact één** entiteit
 - Entiteit verdwijnt → value-object verdwijnt
 - Een entiteit heeft een **identifier**
 - Identificatie om entiteit op te vragen
 - Value-object wordt opgevraagd via entiteit
- UML: compositie

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

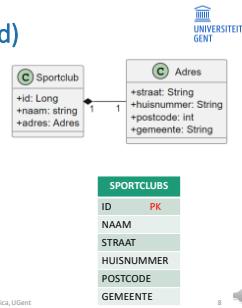


7

Value-objecten (embedded)

- Instantiërievarele primitief type of String → één kolom in tabel ("klasse")
- Value-object niet zinvol om te bewaren in aparte tabel → meerdere kolommen in tabel

- Klasse value-object:
@Embeddable



8

```
@Entity
@Table(name = "sportclubs")
public class Sportclub {
    private Long id;
    private String naam;
    private Adres adres;
    ...
    //@Embedded
    public Adres getAdres() { return adres; }
    public void setAdres(Adres adres) {
        this.adres = adres;
    }
}
```

```
@Embeddable
public class Adres {
    private String straat;
    private String huisnummer;
    private Integer postcode;
    private String gemeente;
    ...
    public String getGemeente() {
        return gemeente;
    }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

9

Eigenschappen zonder annotaties

| Type eigenschap | Default annotatie |
|---|---|
| Primitief type of String | @Basic toegepast |
| Klasse met annotatie @Embeddable | @Embedded toegepast voor eigenschap |
| Serializable | @Basic toegepast <small>Let op met collecties!</small> |
| java.sql.Clob of java.sql.Blob | @Lob toegepast |

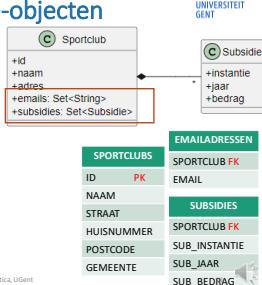
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

10

Mapping associaties value-objecten

- Instantiërievarele
 - Collection primitief type of string
 - Collection ander type
- Bestaat enkel in huidige klasse/object

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

11

Mapping associaties value-objecten

- Javatype
 - Collection
 - Set
 - List
 - Map
- Type item in collectie
 - Primitief
 - Embeddable
- Annotations
 - JPA 2.0
 - **@ElementCollection**
 - **@CollectionTable**
 - **@Column**
 - **@AttributeOverrides**

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

12

2.7. VALUE-OBJECTEN IN ORM

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

`public class Sportclub {
 private int id;
 private String naam;
 private Adres adres;
 private Set<String> emails;
 @ElementCollection
 @CollectionTable(name = "emailadressen",
 joinColumns = @JoinColumn(name = "sportclub"))
 @Column(name="email")
 public Set<String> getEmails() {
 return emails;
 }
 ...
}`

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

13

Mapping associaties value-objecten

- Collection primitief type of string
 - Set, List \leftrightarrow `@ElementCollection`
 - `@CollectionTable`
 - Koppling tussen tabel object en tabel instantievariabele
 - name: naam tabel
 - joinColumns: kolom(men) foreign key
 - * `@JoinColumn`: naam kolom met foreign key
 - `@Column`
 - Mapping element uit collection
 - name: kolom waarde uit collection

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent



14

`public class Sportclub {
 private int id;
 private String naam;
 private Adres adres;
 private Set<String> emails;
 private Set<Subsidie> subsidies;
 @ElementCollection
 @CollectionTable(name = "subsidies",
 joinColumns = @JoinColumn(name = "sportclub"))
 @AttributeOverrides(
 @AttributeOverride(name = "jaar",
 column = @Column(name = "sub_jaar")),
 @AttributeOverride(name = "bedrag"),
 @AttributeOverride(name = "instantie",
 column = @Column(name = "sub_instance"))
)
 public Set<Subsidie> getSubsidies() {
 return subsidies;
 }
}`

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent

15

Mapping associaties value-objecten

- Collection geen primitief type of string
 - Set, List \leftrightarrow `@ElementCollection`
 - `@CollectionTable`
 - Koppling tussen tabel object en tabel instantievariabele
 - name: naam tabel
 - joinColumns: kolom(men) foreign key
 - * `@JoinColumn`: naam kolom met foreign key
 - `@AttributesOverrides`
 - Mapping eigenschappen element uit collection op kolommen
 - `@AttributeOverride`
 - Eigenschap element \leftrightarrow kolom
 - name: eigenschap element uit collection
 - column: corresponderende kolom in tabel

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR
Industriel Ingenieur Informatica, UGent



16

2.8. RELATIES IN ORM

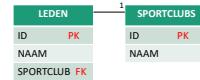
HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

Mapping van associaties van entiteiten

- 1-1
 - Unidirectioneel
 - Bidirectioneel
- 1- veel
 - Unidirectioneel
 - Bidirectioneel
- Veel – veel
 - Unidirectioneel
 - Bidirectioneel

1- veel bidirectioneel

- Met foreign key



Bidirectioneel in code

```

Sportclub maakSportclubMetLeden(String naam) {
    Sportclub club = maakSportclub(naam);
    club.setLeden(maakLeden());
    club.getLeden().forEach(lid -> lid.setClub(club));
    return club;
}

// verander van club
Lid lid = club.getLeden().get(index);
lid.setClub(club2);
club2.getLeden().add(lid);
club1.getLeden().remove(lid);
  
```

Aanpassing annotatie Sportclub

```

@ManyToOne
@JoinColumn(name="sportclub")
public Sportclub getClub() {
    return club;
}

@OneToMany(mappedBy="club")
public Set<Lid> getLeden() {
    return leden;
}
  
```

1-veel bidirectioneel: foreign key

```

@OneToMany(mappedBy="club")
public Set<Lid> getLeden() {
    return leden;
}

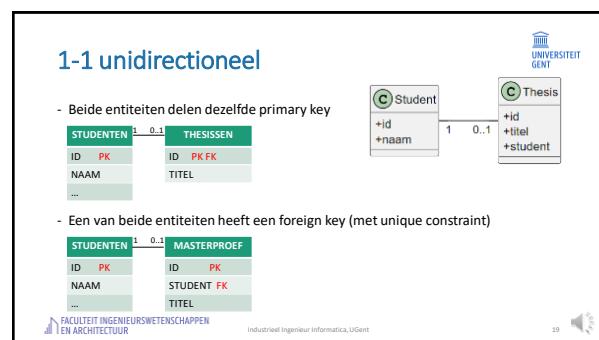
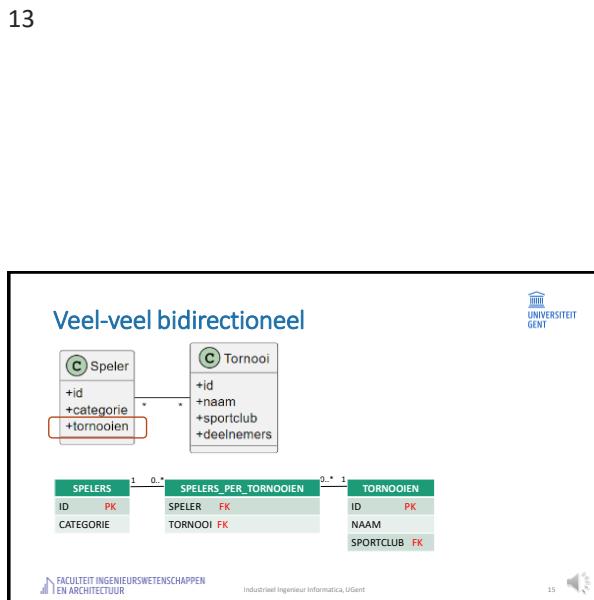
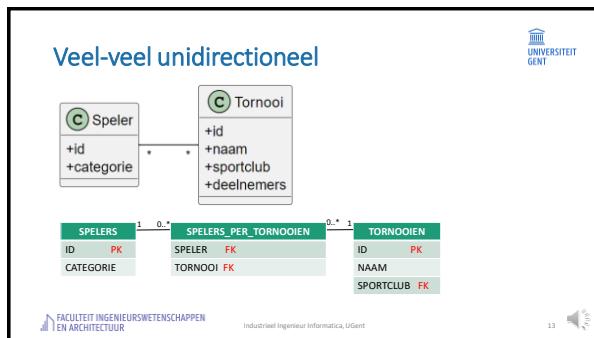
- 1-veel relatie
- Naam van de eigenschap van Lid die de relatie beschrijft
- Altijd bij @OneToMany niet bij @ManyToOne
  
```

Mapping van associaties van entiteiten

- 1-1
 - Unidirectioneel
 - Bidirectioneel
- 1- veel
 - Unidirectioneel
 - Bidirectioneel
- Veel – veel
 - Unidirectioneel
 - Bidirectioneel

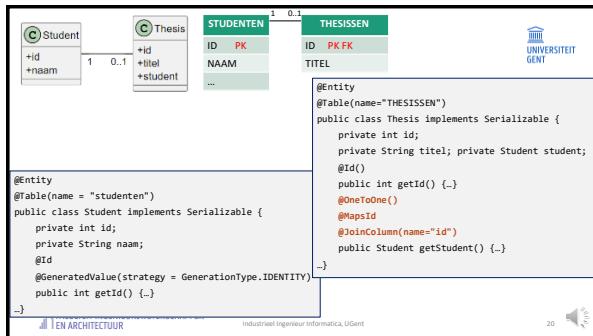
HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.8. RELATIES IN ORM

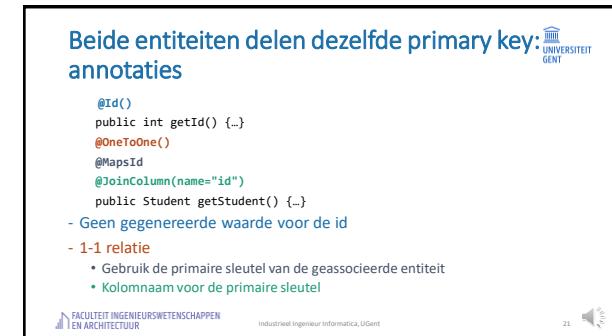


2.8. RELATIES IN ORM

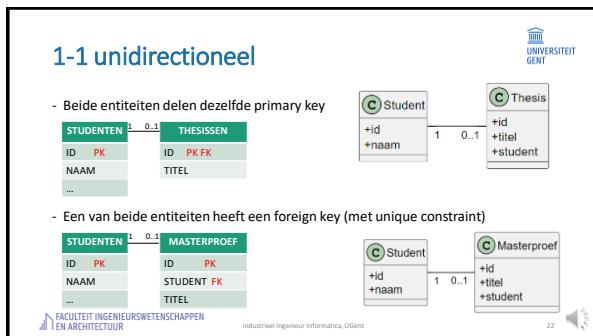
HOOFDSTUK 2. OBJECT RELATIONAL MAPPING



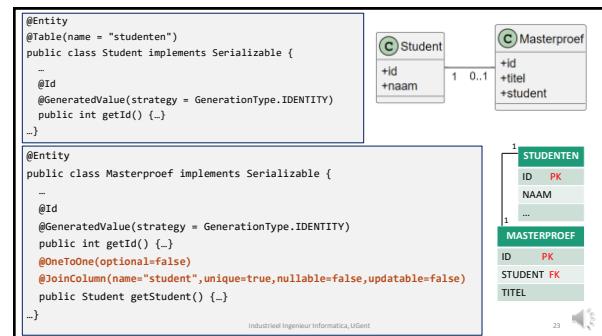
20



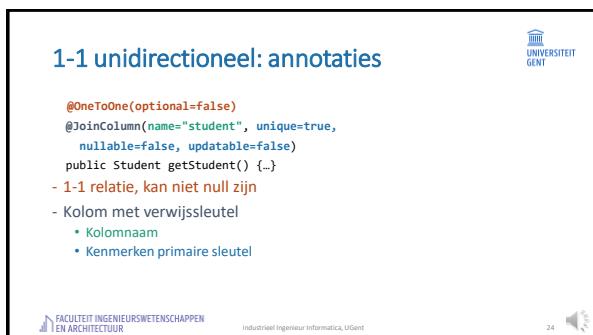
21



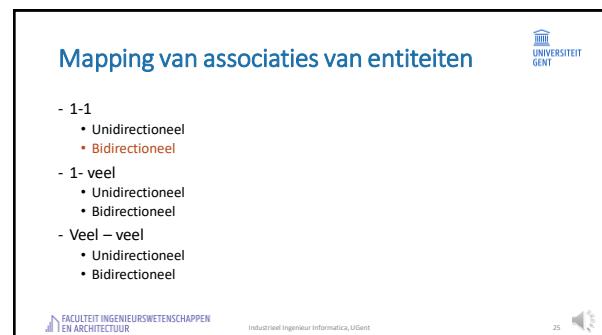
22



23



24



25

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

2.8. RELATIES IN ORM

1-1 bidirectioneel

- Beide entiteiten delen dezelfde primary key

| STUDENTEN | 1 .. 0..1 | THESISSEN |
|-----------|-----------|------------|
| ID PK | ID PK FK | NAAM TITEL |
| NAAM | | TITEL |
| ... | | |

- Een van beide entiteiten heeft een foreign key (met unique constraint)

| STUDENTEN | 1 .. 0..1 | MASTERPROEF |
|-----------|-----------|-------------|
| ID PK | ID PK | STUDENT FK |
| NAAM | | TITEL |
| ... | | |

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

26

Aanpassen klasse Student en Masterproef

- Eigenschap masterproef toevoegen aan Student
- Eenduidigheid toevoegen

```
List<Masterproef> maakMasterproeven(List<Student> studenten) {
    List<Masterproef> masterproeven = new ArrayList<>();
    String[] titels = {"Hibernate", "ORM", "Linq", "JDBC",
        "ADO.NET", "JSF", "JPA", "JAXB", "Webservices"};
    for (int i = 0; i < titels.length; i++) {
        Masterproef masterproef = new Masterproef();
        masterproef.setTitel(titels[i]);
        masterproef.setStudent(studenten.get(i));
        studenten.get(i).setMasterproef(masterproef);
        masterproeven.add(masterproef);
    }
    return masterproeven;
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

27

1-1 bidirectioneel: annotations

- Beide entiteiten delen dezelfde primary key

| STUDENTEN | 1 .. 0..1 | THESISSEN |
|-----------|-----------|------------|
| ID PK | ID PK FK | NAAM TITEL |
| NAAM | | TITEL |
| ... | | |

- Thesis

```
@Id() public int getId() {...}
@OneToOne(optional=false)
@JoinColumn(name="id")
public Student getStudent() {...}
```

- Student

```
@OneToOne(mappedBy="student") public Thesis getThesis() {...}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

28

1-1 bidirectioneel: foreign key

- Een van beide entiteiten heeft een foreign key (met unique constraint)

| STUDENTEN | 1 .. 0..1 | MASTERPROEF |
|-----------|-----------|-----------------|
| ID PK | ID PK | NAAM STUDENT FK |
| NAAM | | STUDENT |
| ... | | TITEL |

- Masterproef

```
@OneToOne(optional=false)
@JoinColumn(name="student", unique=true, nullable=false,
updateable=false)
public Student getStudent() {...}
```

- Student

```
@OneToOne(mappedBy="student") public Masterproef getMasterproef() {...}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

29

Relaties en cascade

- Actie op entiteit → actie op entiteit in relatie?
 - Bv. bestaan entiteit afhankelijk bestaan andere entiteit
- Enum javax.persistence.CascadeType
- Attribuut cascade
 - ALL
 - DETACH
 - MERGE
 - PERSIST
 - REFRESH
 - REMOVE

Customer WEG → orders WEG

```
@OneToMany(cascade=CascadeType.REMOVE, mappedBy="customer")
public Set<CustomerOrder> getOrders() { return orders; }
```

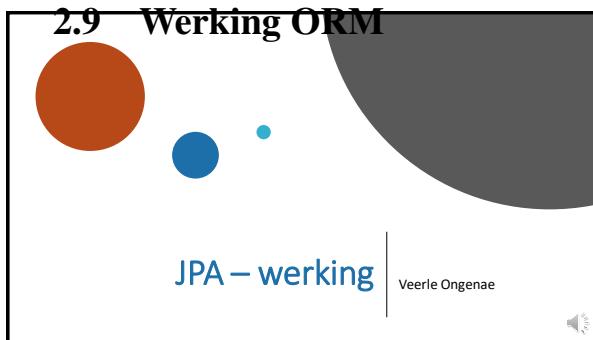
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

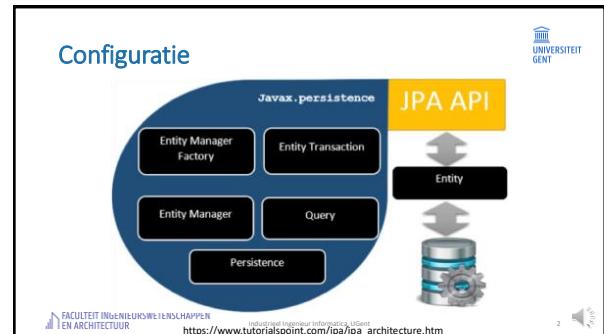
30

2.9. WERKING ORM

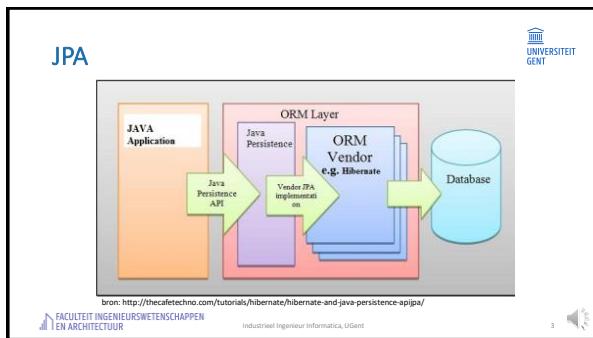
HOOFDSTUK 2. OBJECT RELATIONAL MAPPING



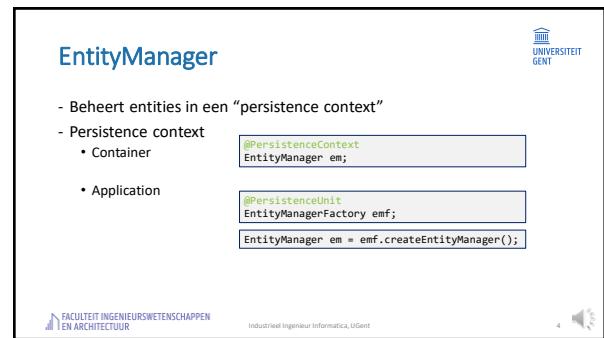
1



2



3



4

EntityManager

```
String[] clubs = {"EIKENLO", " - ", "LANDEGEM BC FV"};
EntityManagerFactory entityManagerFactory =
    Persistence.createEntityManagerFactory("BadmintonJPAPU");
EntityManager entityManager = entityManagerFactory.createEntityManager();
entityManager.getTransaction().begin();
for (String club : clubs) {
    Sportclub sportclub = new Sportclub();
    sportclub.setNaam(club);
    entityManager.persist(sportclub);
}
entityManager.getTransaction().commit();
entityManager.close();
```

FACULTEIT INGEGENieursWETenschappen
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent

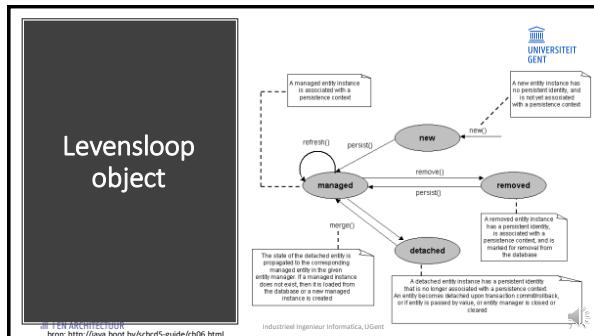
5

-
- Werken met objecten**
- Toestanden van een object
 - Objecten persistent maken
 - Objecten opvragen
 - Objecten verwijderen
 - Objecten wijzigen
- FACULTEIT INGEGENieursWETenschappen
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent

6

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

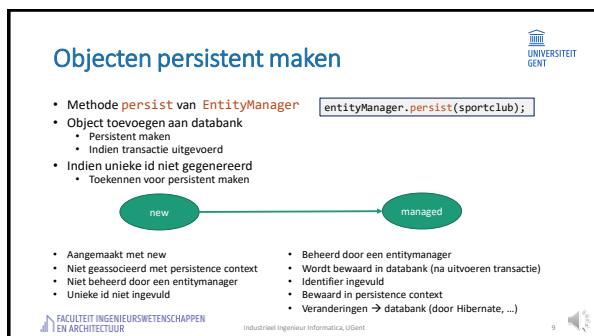
2.9. WERKING ORM



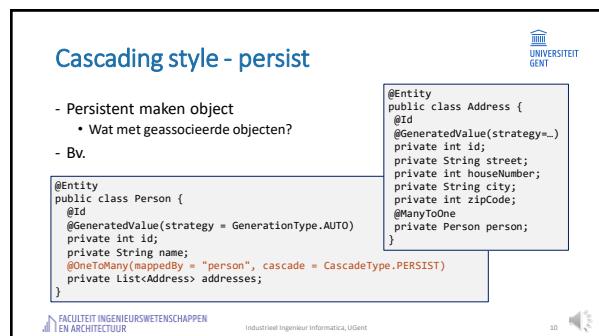
7



8



9



10



11



12

2.9. WERKING ORM

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

Opvragen op basis van identifier

```
public Sportclub getSportclub(Long id) {
    EntityManager em = emf.createEntityManager();
    Sportclub sportclubById = em.find(Sportclub.class, id);
    em.close();
    return sportclubById;
}
```

- Methode `find` van EntityManager
 - `T find(Class<T> entityClass, Object primaryKey)`
 - null indien onbestand persistent object
 - Meermaals opvragen object metzelfde identifier → verschillende referenties naarzelfde object (in cache)



Industriel Ingenieur Informatica, UGent



13

Lazy fetching

- Opvragen object
 - Geassocieerde objecten niet mee opgevraagd
 - Opgevraagd indien nodig
 - Enkel als sessie nog open is
- Onmiddellijk mee opvragen

```
@ManyToOne(fetch=FetchType.EAGER)
```

```
@Entity
public class Address {
    @Id
    @GeneratedValue(strategy=...)
    private int id;
    private String street;
    private int houseNumber;
    private String city;
    private int zipCode;
    @ManyToOne(fetch = FetchType.LAZY)
    private Person person;
}
```

Industriel Ingenieur Informatica, UGent



14

13

14

Voorbeeld lazy

```
public List<Lid> getLeden(Long id) {
    EntityManager em = emf.createEntityManager();
    Sportclub sportclubById = em.find(Sportclub.class, id);

    // lazy, ophalen en kopiëren
    List<Lid> leden = new ArrayList<>(sportclubById.getLeden());
    em.close();
    return leden;
}
```



Industriel Ingenieur Informatica, UGent



15

Werken met objecten

- Toestanden van een object
- Objecten persistent maken
- Objecten opvragen
- Objecten verwijderen
- Objecten wijzigen

FACULTEIT INGENIEURSWETENSCHAPPEN

Industriel Ingenieur Informatica, UGent



16

15

16

Objecten verwijderen

```
EntityManager em = ...
Employee employee = em.find(Employee.class, 1);
em.getTransaction().begin();
em.remove(employee);
em.getTransaction().commit();
```

- Methode `remove` van EntityManager
- Ook voor geassocieerde entiteiten?
 - Instellen met `cascade=CascadeType.REMOVE` op associatie (@)



- Beheerd door een entitymanager
- Wordt bewaard in databank (na uitvoeren transactie)
- Identifer ingevuld
- Bewaard in persistence context
- Veranderingen → databank (door Hibernate, ...)
- Gemarkerd om verwijderd te worden
- Pas verwijderd bij uitvoeren transactie



Industriel Ingenieur Informatica, UGent



17

Werken met objecten

- Toestanden van een object
- Objecten persistent maken
- Objecten opvragen
- Objecten verwijderen
- Objecten wijzigen

FACULTEIT INGENIEURSWETENSCHAPPEN

Industriel Ingenieur Informatica, UGent



18

17

18

HOOFDSTUK 2. OBJECT RELATIONAL MAPPING**2.9. WERKING ORM****Objecten wijzigen**

- Persistent objecten
 - Aangemaakt via **persist**
 - Opgehaald via **find** of een zoekopdracht
- Wijzigen tijdens een transactie
 - Methode **flush**
 - Wijzigingen doorvoeren naar databank
 - Wordt ook impliciet opgeroepen bij **commit** van de transactie

```
EntityManager em = ...;
Employee employee = em.find(Employee.class, 1);
em.getTransaction().begin();
employee.setNickname("Joe the Plumber");
em.getTransaction().commit();
```

19

Merge

- Methode **merge** van EntityManager
 - Het meegegeven object in de huidige persistentiecontext kopiëren
 - Resultaat = object uit persistentiecontext
 - Enkel nodig bij doorgeven objecten tussen contexten
- ```
EntityManager em = createEntityManager();
Employee detached = em.find(Employee.class, id);
em.close();

em = createEntityManager();
em.getTransaction().begin();
Employee managed = em.merge(detached);
em.getTransaction().commit();
```

20

**Objecten wijzigen**

- Detached objecten
  - Object aangemaakt door andere/niet meer bestaande EntityManager
  - Object wijzigen
  - Wijzigingen bewaren met nieuwe EntityManager
    - Methode **merge**
      - Na methode is het object persistent
      - Eventueel wijzigingen kopiëren in een bestaand object met zelfde id
- Geassocieerde entiteiten ook?
  - Instellen op associatie: **cascade=CascadeType.MERGE**
- Persistent object
  - Niet gekoppeld aan persistence-context omdat entitymanager gesloten is
  - Niet beheerd door een entitymanager
- Beheerd door een entitymanager
  - Wordt bewaard in databank (na uitvoeren transactie)
  - Unieke id ingevuld
  - Bewaard in persistence context
  - Veranderingen → databank (door Hibernate, ...)

21

20

22

22

**Persistence context**

- Deel van
  - JPA-EntityManager
- Nuttig voor
  - Automatisch checken dirty objecten
  - Dirty objecten
    - Gewijzigd
    - Wijzigingen nog niet doorgevoerd op database
    - Doorvoeren wijzigingen zo lang mogelijk uitgesteld, bv. commit transactie
  - First level cache
    - Entiteiten cachen
    - Meermalen oproveren → referentie naarzelfde object
- Bestaat zolang de EntityManager geopend is

23



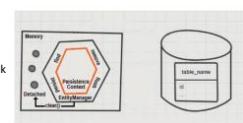
Industriel Ingenieur Informatica, UGent

23

**Object verwijderen uit persistence context**

- EntityManager sluiten
- Methode **detach** van EntityManager
  - Verwijderd uit session cache
    - Aanpassingen niet meer doorgevoerd in databank
    - Toestand = detached
  - Kan nog gebruikt worden in programma
  - Ook op geassocieerde objecten?
    - **cascade=CascadeType.DETACH** (annotaties)
  - Alle persistenten objecten verwijderen uit session cache (= detached maken)
    - Methode **clear**
    - Nog niet "gefuslused" opdrachten worden niet uitgevoerd op de database

24



Industriel Ingenieur Informatica, UGent

24

## 2.9. WERKING ORM

## HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

**Entity Locking/Concurrency**

- Gelijktijdige toegang?
  - Dataintegriteit bewaren?
- Standaard
  - Optimistic locking
    - Voor aanpassen kijken of data niet veranderd is a.d.h.v. versiekolom
  - Pessimistic locking
    - Transactie legt lock op data zolang de transactie loopt

The diagram consists of two parts. The top part is a flowchart titled 'Entity Locking/Concurrency' with the University of Ghent logo. It starts with 'Record retrieved', leading to 'Read value in database'. This leads to a decision diamond 'Check old value and new value'. If 'Not changed', it goes to 'Commit value'. If 'Changed', it goes to 'Reject transaction'. The bottom part is a timeline titled 'Simple Locking Mechanism' showing two transactions, Transaction 1 and Transaction 2, interacting with a resource. Transaction 1 locks the resource from start to finish. Transaction 2 waits until Transaction 1 has released the lock.

Bron: <https://www.c-sharpcorner.net/uploadfile/mihirprasad/2-ways-of-doing-optimistic-locking-in-net/>

Bron: <https://tech.urbancompany.com/pessimistic-locking-for-a-distributed-system-part-1-cdb5765d37>

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingénieur Informatica, UGent

25

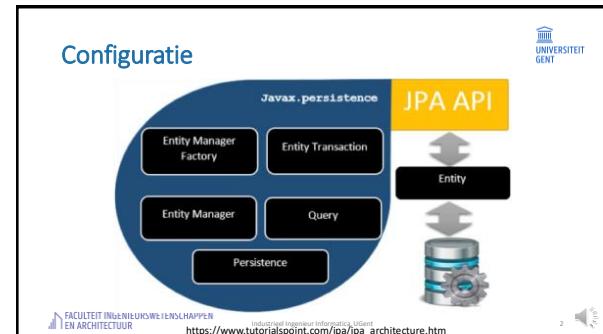
25

## 2.10 Zoekopdrachten in ORM

JPA – query

Veerle Ongenaee

1



2

### Objecten opvragen

- Op basis van identifier
- Zonder identifier
  - Java Persistence Query Language
  - SQL

3

### Java Persistence Query Language

- Lijkt op SQL
  - Niet hoofdlettergevoelig (behalve klassennamen en attributen)
  - Klassennamen, namen properties ↔ tabellenamen, kolomnamen
- Aanmaken zoekopdracht (**Query**)
- **javax.persistence.EntityManager**

```
Query createQuery(String queryString)
```

4

### JPQL-zoekopdracht

- Voorbeeld
 

```
EntityManager entityManager = ...;
List<Sportclub> clubs = entityManager
 .createQuery("SELECT s FROM Sportclub s", Sportclub.class)
 .getResultList();
```

klasse
- Alle objecten van het type Sportclub ophalen en bewaren in een lijst
- Zoekopdracht uitvoeren
  - Meestal: oproepen methode getResultList()
  - Lijst van persistente objecten

5

### JPQL-zoekopdracht met parameters

```
Query opdracht
 = entityManager.createQuery("SELECT l from Lid l where l.club.naam = ?1");
String naam = "EIKENLO";
opdracht.setParameter(1, naam);
List<Lid> leden = opdracht.getResultList();
```

alias      eigenschappen object  
             ↑                                ↑  
               volgnummer

```
opdracht
 = entityManager.createQuery("SELECT l from Lid l where l.club.naam = :naam");
naam = "EIKENLO";
opdracht.setParameter("naam", naam);
leden = opdracht.getResultList();
```

parameter

- Alternatief
- Parameter
  - ?n (telt vanaf 1) of :naam (een naam voor de parameter)

6

## 2.10. ZOEKOPDRACHTEN IN ORM

## HOOFDSTUK 2. OBJECT RELATIONAL MAPPING

## Zoekopdracht met Spring-repository

```
public interface UserRepository extends JpaRepository<User, Long> {
 @Query("select u from User u where u.emailAddress = ?1")
 User findByEmailAddress(String emailAddress);
}
```



parameter

## JPQL

- distinct, order by

```
List<String> namen = entityManager
 .createQuery("select distinct s.naam from Sportclub s order by s.naam")
 .getResultList();
```

- COUNT, SUM, MIN, MAX, AVG, GROUP BY

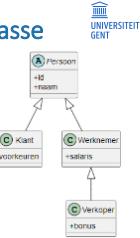
```
List<overzicht> entitymanager
 .createQuery("select s.naam, count(s) from Sportclub s group by s.naam having count(s) >= 2")
 .getresultlist();
System.out.println("Sportclubs met een naam die minstens 2 keer voorkomt");
for (Object result : overzicht) {
 Object[] temp = (Object[]) result;
 System.out.println(temp[0] + " " + temp[1]);
}
```

- Joins: inner join, left join, right join, full join



## JPQL zoekopdracht met abstracte klasse

```
Query zoekopdracht
 = entityManager.createQuery("select p from Persoon");
List<Persoon> personen = zoekopdracht.getResultList();
System.out.println("Personen");
for (Object object : personen) {
 Persoon persoon = (Persoon) object;
 System.out.println(persoon.getNaam());
}
```



## Objecten opvragen

- Op basis van identifier
- Zonder identifier
  - Java Persistence Query Language
  - SQL



## SQL-opdrachten – JPA

- Aanmaken zoekopdracht ([Query](#))

- [javax.persistence.EntityManager](#)

```
Query createNativeQuery(String queryString, Class resultClass)
```

- Eerste param: zoekopdracht
- Tweede param: klasse resultaat (resultaten)

```
List<Sportclub> sportclubs = entityManager
 .createNativeQuery("select id, naam from sportclubs", Sportclub.class)
 .getResultList();
```



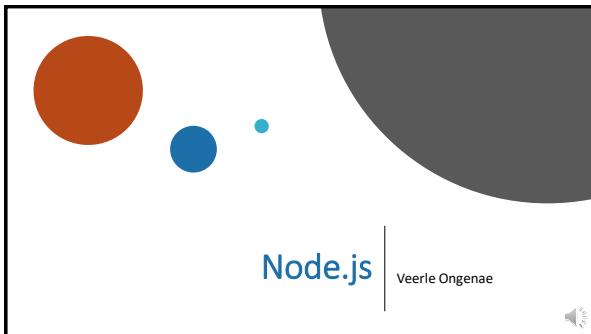
## SQL-zoekopdracht met Spring-repository

```
public interface UserRepository extends JpaRepository<User, Long> {
 @Query(value = "SELECT * FROM USERS u WHERE u.status = 1", nativeQuery = true)
 Collection<User> findAllActiveUsersNative();
}
```



## **Hoofdstuk 3**

### **NodeJS**

**HOOFDSTUK 3. NODEJS**

1

**Overzicht**

- Node.js en npm

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica

UNIVERSITEIT  
GENT

2

**Wat is Node.js?**

- Javascript buiten de browser
  - Commandoolijn programma's
  - Serverside platform
- Gebouwd op Chromes Javascript Runtime
- <https://nodejs.org/en/download>

node.js

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica

3

3

**npm**

- Node package manager
- Pakketbeheerder voor Node.js
  - Repository: publiceren open-source Node.js projecten (=packages)
  - Beheren van packages
    - Interageren met repository
    - Packages installeren
    - Versiebeheer
    - Beheer afhankelijkheden
- Bv. HTTP-module → webserver
- Deel van de Node.js-installatie
- Alternatief: yarn (<https://classic.yarnpkg.com/en/>)

https://devopedia.org/package-manager

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

4

4

**package.json**

- Informatie over het project
- In de hoofdmap

```
{
 "name": "MyApp",
 "version": "1.0.0",
 "dependencies": {
 "sax": "^0.3.x",
 "nano": "0.0.0",
 "request": ">0.2.0"
 }
}
```

afhankelijkheden: gebruikte packages

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica

5

5

**Applicatie maken**

```
C:\Users\veongenee\Documents\webtech\mettechnologie\2027-2028\voorbereiden\11_nodejs\mkdir book-project
C:\Users\veongenee\Documents\webtech\mettechnologie\2027-2028\voorbereiden\11_nodejs\cd book-project
C:\Users\veongenee\Documents\webtech\mettechnologie\2027-2028\voorbereiden\11_nodejs\book-project>npm init
```

```
{
 "name": "book-project",
 "version": "1.0.0",
 "description": "Voorbeeld NodeJS",
 "main": "server.js",
 "repository": {
 "type": "git",
 "url": "..."
 },
 "dependencies": {
 "express": "latest",
 "mongoose": "latest"
 },
 "author": "Joerg Krause",
 "license": "MIT",
 "homepage": "http://www.joergkrause.de"
}
```

nuttige scripts voor het programma
 

- testen
- build
- ...

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica

6

## HOOFDSTUK 3. NODEJS

**Applicatie starten**

```
npm start main in package.json uitvoeren
npm run <task-name> package.json
{
 "scripts": {
 "start-dev": "node lib/server-development",
 "start": "node lib/server-production"
 }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica 7

7

**Afhankelijkheden**

```
npm install <naampakket> -g -global
npm install <naampakket> --save toegevoegd in package.json
> In map node_modules
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica 8

8

**Overzicht**

- Node.js en npm
- HTTP-server

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica 9

9

**Webservers: statische pagina's**

bron: <http://www.resultantsys.com/index.php/general/what-is-a-web-application-server/>

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 10

10

**Dynamische webpagina's**

- Webpagina's genereren
  - Op server
  - Door "programma's"
    - Geïnterpreteerd (PHP, node.js, ...)
    - Gecompileerd (Java Spring, ASP.NET core, ...)

bron: <http://www.resultantsys.com/index.php/general/what-is-a-web-application-server/>

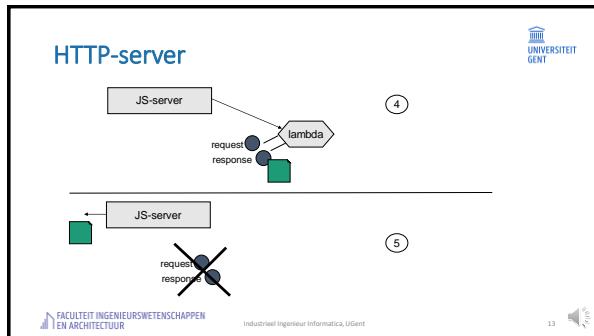
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 11

11

**HTTP-server**

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 12

12

**HOOFDSTUK 3. NODEJS**

**HTTP-server in Node.js**

```
let http = require('http');
let fs = require('fs');
let port = process.env.PORT || 1337;

http.createServer((req, res) => {
 console.log("Aanvraag op poort 1337");
 res.writeHead(200, {
 'Content-Type': 'text/html',
 'Access-Control-Allow-Origin': '*'
 });
 let read = fs.createReadStream(__dirname + '/index.html');
 read.pipe(res);
}).listen(port);
```

modules, https kan ook poort  
HTTP-server maken op poort  
request en response-object  
headers instellen  
afhandelen aanvraag  
map huidige module/app bestand inlezen en wegschriften naar body response

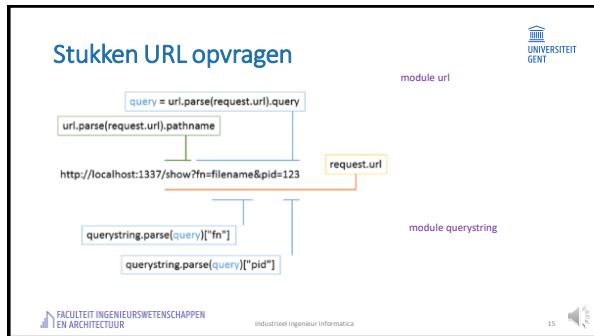
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel ingenieur Informatica

14

13

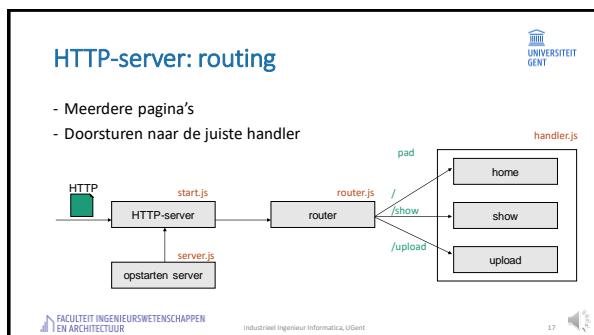
14



- Overzicht**
- Node.js en npm
  - HTTP-server
  - Routing
- FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR
- Industrieel ingenieur Informatica
- 16

15

16



17

18

### HOOFDSTUK 3. NODEJS

**Voorbeeld routing – opstarten server**

```
server.js
let server = require('./start');
let router = require('./router');
let requestHandlers = require("./handlers");

let handler = {};
handler['/'] = requestHandlers.home;
handler['/show'] = requestHandlers.show;
handler['/upload'] = requestHandlers.upload;

server.start(router.route, handler);
```

zelfgeschreven modules  
link pad = functie  
server starten  
routes en handlers meegeven

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

19

**Voorbeeld – handler-functies**

```
handlers.js
let fs = require('fs');
function home(response) {
 respond(response, 'views/home.html');
 return true;
}

function show(response) {...}
function upload(response) {...}
function respond(response, file) {
 fs.readFile(file, (err, data) => {
 response.writeHead(200, {"Content-Type": "text/html"});
 response.write(data);
 response.end();
 });
}

exports.home = home;
exports.show = show;
exports.upload = upload;
```

module filesystem  
handler-functies  
kopieert file naar body HTTP-bericht  
functies exporteren

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

20

19

20

**Voorbeeld - router**

```
router.js
function route(pathname, handler, response) {
 console.log("Request for " + pathname);
 if (handler[pathname] !== undefined) {
 return handler[pathname](response);
 } else {
 console.log("No Method found for " + pathname);
 return null;
 }
}
exports.route = route;
```

bepaalt de juiste handler-functie voor het gegeven pad en voert die uit  
functie exporteren

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

21

**Voorbeeld - server**

```
start.js
let http = require("http");
let url = require("url");

function start(route, handler) {
 console.log("Starting...");
}

function onRequest(request, response) {
 let pathname = url.parse(request.url).pathname;
 let content = route(pathname, handler, response);
 if (!content) {
 response.writeHead(404, {"Content-Type": "text/plain"});
 response.end();
 }
}

let port = process.env.port || 1337;
http.createServer(onRequest).listen(port);
console.log("Has been started.");
}

exports.start = start;
```

modules  
server opstarten  
afhandelen aanvraag pad bepalen inhoud voor specifiek pad headers bij fout instellen body bij fout otschrijven antwoord doorsturen  
HTTP-server aanmaken functie exporteren

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

start.js

UNIVERSITEIT  
GENT

22

21

22

**HTTP-methode beperken**

```
if (request.method !== 'GET') {
 response.writeHead("405");
 response.end();
}

if (request.method !== 'POST') {
 response.writeHead("405");
 response.end();
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica

23

**Overzicht**

- Node.js en npm
- HTTP-server
- Routing
- Node.js versus Threads

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

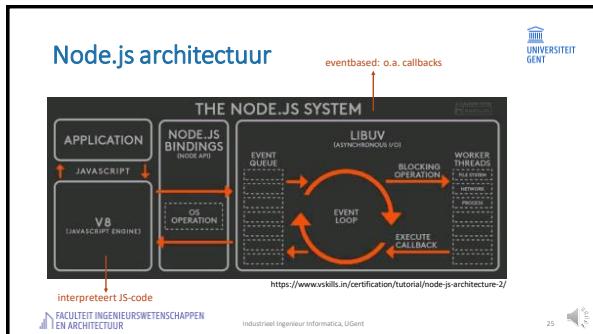
Industrieel Ingenieur Informatica

24

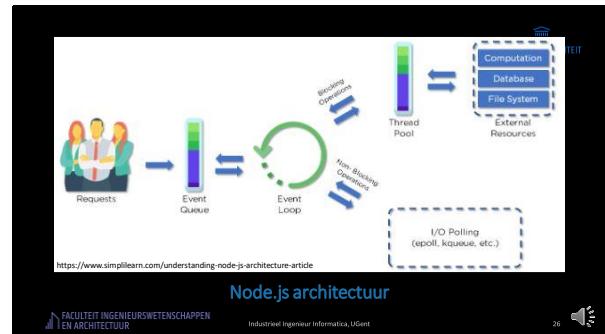
23

24

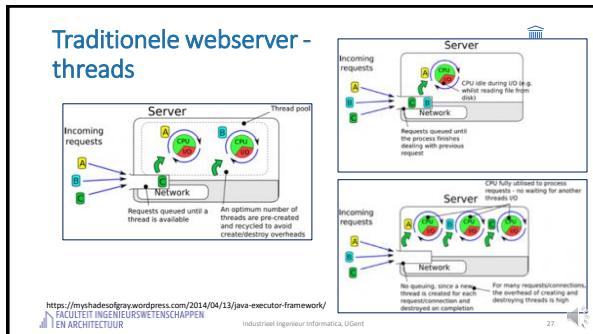
## HOOFDSTUK 3. NODEJS



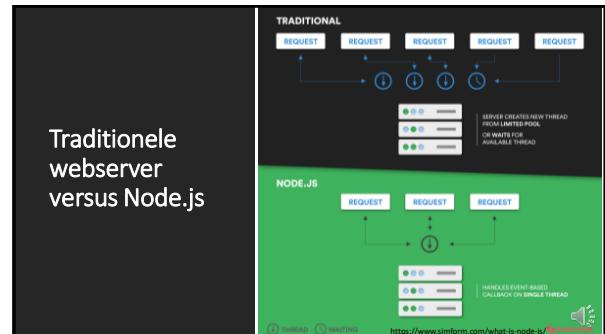
25



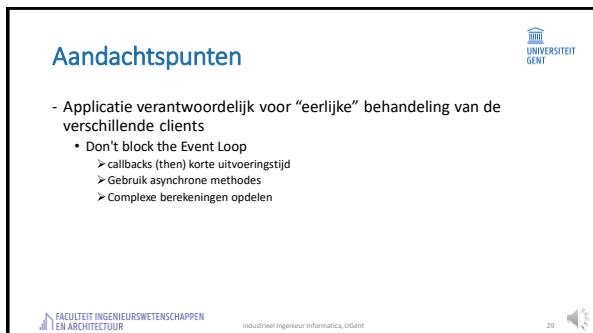
26



27



28

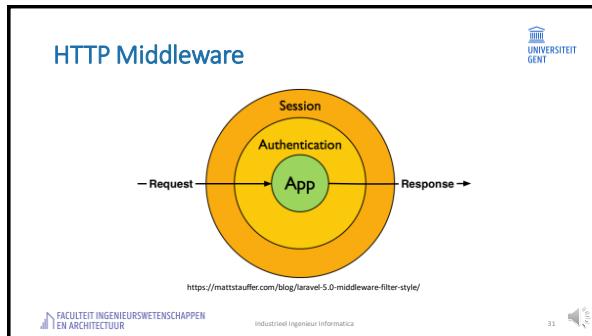


29

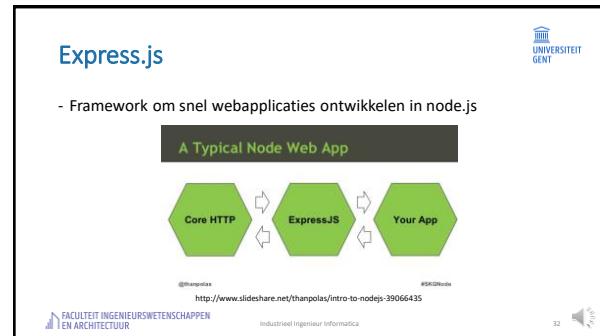


30

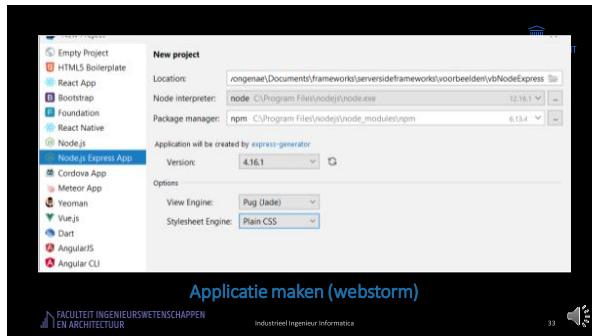
## HOOFDSTUK 3. NODEJS



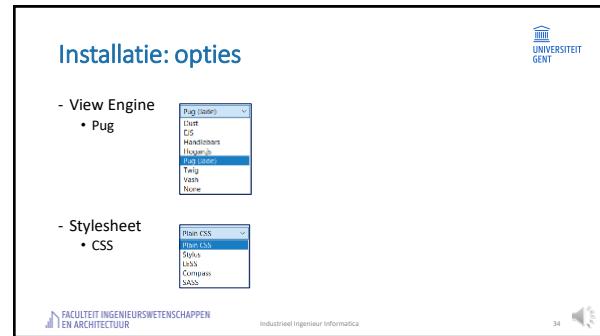
31



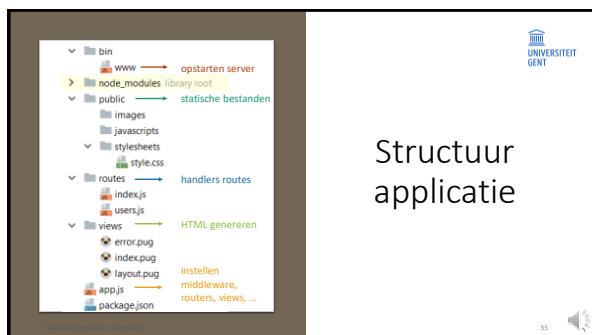
32



33

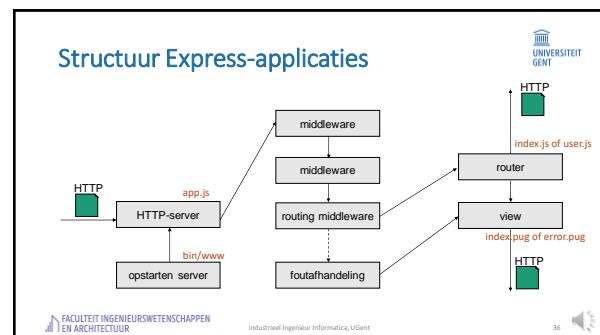


34



35

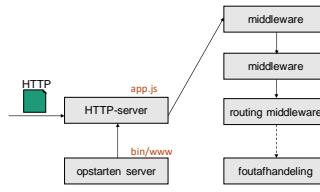
## Structuur applicatie



36

**HOOFDSTUK 3. NODEJS****Overzicht**

- Node.js en npm
- HTTP-server
- Routing
- Node.js versus Threads
- Express
  - Inleiding
  - app.js

**Structuur Express-applicaties****app.js - modules**

```

let express = require('express');
let path = require('path');
let favicon = require('serve-favicon');
let logger = require('morgan'); // logging
let cookieParser = require('cookie-parser');
let bodyParser = require('body-parser');

let routes = require('./routes/index');
let users = require('./routes/users');

let app = express();

```

gebruikte modules  
zelfgeschreven routes

**app.js – middleware**

```

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// uncomment after placing your favicon in /public
// app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);

```

views instellen  
middleware  
routes instellen

**app.js – foutafhandeling**

```

// catch 404 and forward to error handler
app.use(function(req, res, next) {
 let err = new Error('Not Found');
 err.status = 404;
 next(err);
});

```

enkel opgereden indien nog niet afgehandeld  
doorsturen volgende middleware

**app.js – foutafhandeling**

```

// error handler
app.use(function(err, req, res, next) {
 // set locals, only providing error in development
 res.locals.message = err.message;
 res.locals.error =
 req.app.get('env') === 'development' ? err : {};
 // render the error page
 res.status(err.status || 500);
 res.render('error');
});

module.exports = app;

```

HTTP-statuscode instellen  
doorsturen naar view  
applicatie-object exporteren

## HOOFDSTUK 3. NODEJS

### Overzicht

- Node.js en npm
- HTTP-server
- Routing
- Node.js versus Threads
- Express
  - Inleiding
  - app.js
  - Routing

FACULTEIT INGENIEURSWETENSCHAPPEN

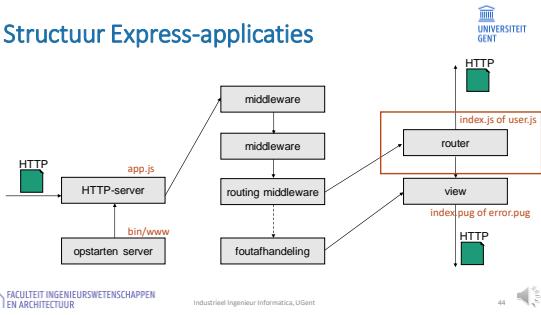
EN ARCHITECTUUR

Industrieel Ingenieur Informatica



43

### Structuur Express-applicaties



FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent

44

43

44

### routes/index.js

```
let express = require('express');
let router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
 res.render('index', { title: 'Express' });
});

module.exports = router;
```



FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica

45

### routes/users.js

```
let express = require('express');
let router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
 res.send('respond with a resource');
});

module.exports = router;
```



router ophalen  
route voor get naar / instellen (binnen huidig pad)  
pad  
functie die aanvraag afhandelt  
doorsturen naar view  
view  
data voor view

REST-service

FACULTEIT INGENIEURSWETENSCHAPPEN

Industrieel Ingenieur Informatica

46

46

45

### Routing

- URL ↔ methode, module
- Single Page Applications (SPA)
  - URL ↔ REST call
- Non-SPA
  - URL ↔ volledige webpagina
- Module Express Router

```
let router = express.Router();
```



FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica

47

### Pad ↔ router

```
let express = require('express');
let adminRouter = require('./routes/adminRouter');
let app = express();
app.use('/admin', adminRouter);

let express = require('express');
let router = express.Router();

// The Admin-Site (http://localhost:3000/admin)
router.get('/', function(req, res) {
 res.send('Homepage of admin area!');
});
// The article-Site (http://localhost:3000/admin/article)
router.get('/article', function(req, res) {
 res.send('Show all articles!');
});
module.exports = router;
```



app.use(SUB\_PATH, ROUTER\_MODULE)

adminRouter.js

FACULTEIT INGENIEURSWETENSCHAPPEN

Industrieel Ingenieur Informatica

48

48

47

**HOOFDSTUK 3. NODEJS****Express routing**

- URI → actie
- Basis routing

```
app.METHOD(PATH, HANDLER)
```

```
let app = express();
app.get('/', (req, res) => {});
```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica



49

**Verschillende handlers - parameters**

```
router.get('/user/:id', function (req, res, next) {
 let id = req.params['id'];
 console.log('CALLED ONLY ONCE');
 next();
});

router.get('/user/:id', function (req, res, next) {
 console.log('although this matches');
 next();
});

router.get('/user/:id', function (req, res) {
 console.log('and this matches too');
 res.end();
});
```

GET /user/42  
 CALLED ONLY ONCE  
 although this matches  
 and this matches too

route voor get  
 parameter in pad  
 request-object  
 response-object  
 volgende middleware  
 parameters  
 afsluiten antwoord

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica

50

**Functie route()**

- Eén route (URL)
  - Verschillende handlers per HTTP-methode
    - GET
    - POST
    - PUT
    - DELETE

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica



51

```
let routes = express.Router();
router.route('/users/:user_id')
 .all(function(req, res, next) {
 // runs for all HTTP verbs first
 req.user = {
 id: req.user_id,
 name: 'TJ'
 };
 next();
 })
 .get(function(req, res, next) {
 res.json(req.user);
 })
 .put(function(req, res, next) {
 req.user.name = req.params.name; // just an example of maybe updating the user
 // save user ... etc
 res.json(req.user);
 })
 .post(function(req, res, next) {
 next(new Error('not implemented'));
 })
 .delete(function(req, res, next) {
 next(new Error('not implemented'));
 });
});
```

route voor pad  
 voor alle HTTP-methodes  
 info toevoegen aan request  
 handler voor HTTP-methode

51

52

**Routing - samenvatting**

- Path ~ handler
 

```
router.METHOD(PATH, HANDLER)
```
- Deelpath ~ module
 

```
app.use(SUB_PATH, ROUTER_MODULE)
```
- Router-object
 

```
let router = express.Router();
```
- Verschillende handlers voor één aanvraag: functie route
  - all(...), get(...), post(...), put(...), delete(...)
  - Doorsturen met next()

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industrieel Ingenieur Informatica



53

**Overzicht**

- Node.js en npm
- HTTP-server
- Routing
- Node.js versus Threads
- Express
  - Inleiding
  - app.js
  - Routing
  - Views

FACULTEIT INGENIEURSWETENSCHAPPEN

Industrieel Ingenieur Informatica

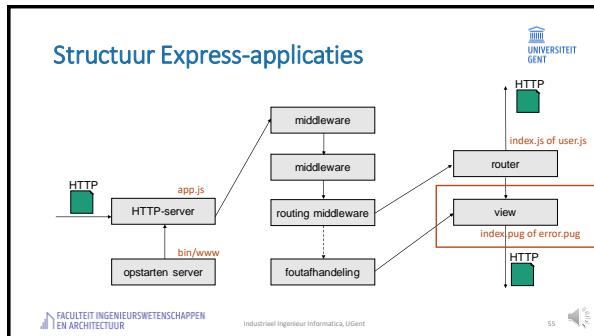


54

53

90

## HOOFDSTUK 3. NODEJS



**Views: index.pug**

```

extends layout
block content
 h1= title
 p Welcome to #{title}
<!DOCTYPE html>
<html>
 <head><title>Express</title>
 <link rel="stylesheet" href="/stylesheets/style.css">
 </head>
 <body>
 <h1>Express</h1>
 <p>Welcome to Express</p>
 </body>
</html>

```

makelt gebruik van layout  
inhoud voor block uit layout  
parameter meegegeven met render-methode  
Gegenererde HTML

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel ingenieur Informatica

57

**Views: error.pug**

```

extends layout
block content
 h1= message
 h2= error.status
 pre #{error.stack}

```

lokale omgevingsparameters

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel ingenieur Informatica

58

**Overzicht**

- Node.js en npm
- HTTP-server
- Routing
- Node.js versus Threads
- Express
  - Inleiding
  - app.js
  - Routing
  - Views

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel ingenieur Informatica

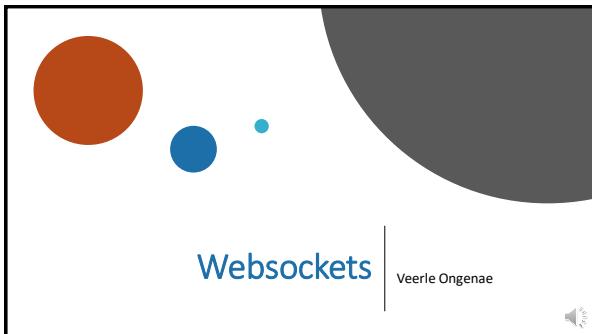
59



## **Hoofdstuk 4**

### **Websockets**

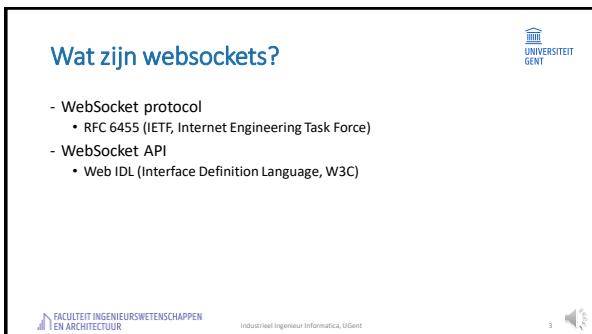
## HOOFDSTUK 4. WEBSOCKETS



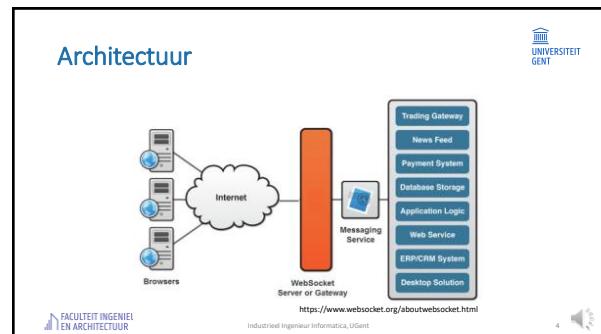
1



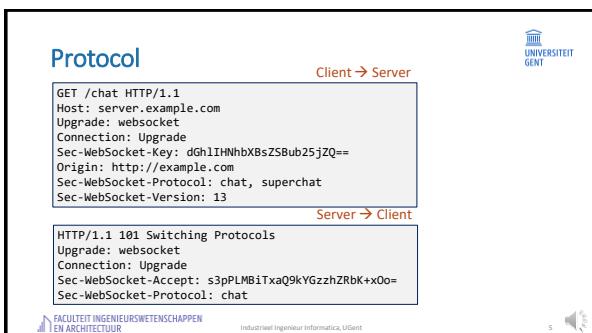
2



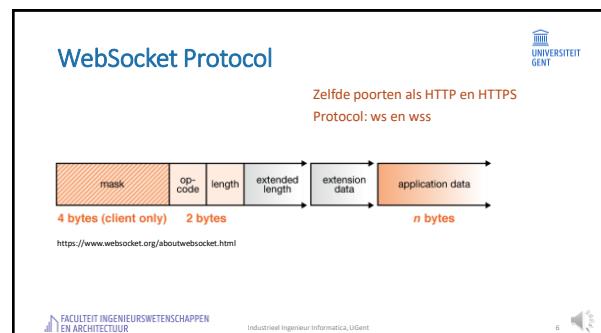
3



4



5



6

## HOOFDSTUK 4. WEBSOCKETS

### Overzicht

- Inleiding
- API



### API (deels)

```
[Constructor(in DOMString url, optional in DOMString protocol)]
interface WebSocket {
 readonly attribute DOMString URL;
 // ready state
 const unsigned short CONNECTING = 0;
 const unsigned short OPEN = 1;
 const unsigned short CLOSING = 2;
 const unsigned short CLOSED = 3;
 readonly attribute unsigned short readyState;
 readonly attribute unsigned long bufferedAmount;

 // networking
 attribute Function onopen;
 attribute Function onmessage;
 attribute Function onclose;
 boolean send(in DOMString data);
 void close();
};

WebSocket implements EventTarget;
```

### Voorbeeld - client

```
let myWebSocket = new WebSocket("ws://www.websocket.org");
// connetie geopend
myWebSocket.addEventListener('open', function (event) {
 myWebSocket.send('Hello Server!');
});
// luisteren naar berichten
myWebSocket.addEventListener('message', function (event) {
 console.log('Message from server ', event.data);
});
// server sluit verbindin
myWebSocket.addEventListener('close', function(event) {
 alert("Connection closed.");
});
myWebSocket.send("Hello WebSockets!");
myWebSocket.close();
```

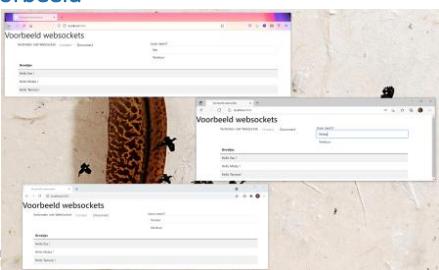


### Overzicht

- Inleiding
- API
- Server



### Voorbeeld



### Java Spring – aanmaken websockethandler

```
public class GreetingHandler extends TextWebSocketHandler {
 // thread safe
 // https://www.beeldung.com/java-copy-on-write-arraylist/
 // https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html
 List<WebSocketSession> sessions = new CopyOnWriteArrayList();

 @Override
 public void handleTextMessage(WebSocketSession session, TextMessage message) {
 // the message will be forwarded to all users
 Map<String, TextMessage> forwardMessage = new HashMap<String, TextMessage>();
 for(WebSocketSession webSocketSession : sessions){
 webSocketSession.sendMessage(new TextMessage("Hello " + value.get("name") + " !"));
 }
 }

 @Override
 public void afterConnectionEstablished(WebSocketSession session) throws Exception {
 sessions.add(session);
 }

 @Override
 public void afterConnectionClosed(WebSocketSession session, CloseStatus status) throws Exception {
 sessions.remove(session);
 }
}
```



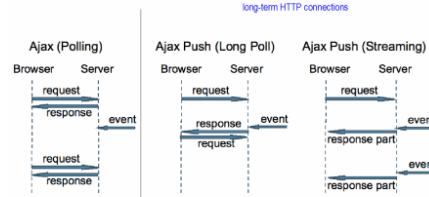
## HOOFDSTUK 4. WEB SOCKETS

### Overzicht

- Inleiding
- API
- Server
- Voordelen websockets



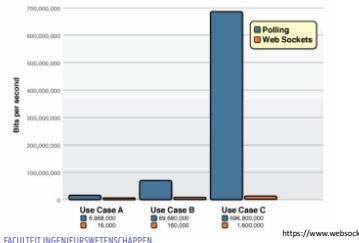
### Voor websockets



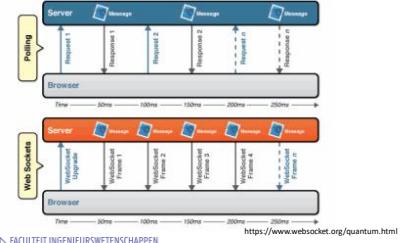
13

14

### Waarom websockets?



### Waarom websockets?



15

16

## **Hoofdstuk 5**

# **Java Database Connectivity (JDBC)**

### **5.1 Wat is JDBC?**

JDBC is de afkorting van Java Database Connectivity. De JDBC API biedt de mogelijkheid om vanuit Java-programma's op een eenvoudige manier toegang te krijgen tot relationele gegevensbanken. Deze API bestaat uit een aantal klassen en interfaces die het mogelijk maken om SQL-opdrachten uit te voeren op een gegevensbank en de resultaten van deze opdrachten te verwerken. Daarvoor voorziet de JDBC API een aantal algemene methoden om toegang te verkrijgen tot SQL-compatibele relationele gegevensbanken. Deze methoden zijn productonafhankelijk en voorzien de meeste standaardacties voor een gegevensbank. Dit maakt het mogelijk om eenzelfde applicatie te gebruiken met gegevensbanken van verschillende producenten.

JDBC bestaat uit twee pakketten: *java.sql* dat de basisfunctionaliteiten bevat en *javax.sql* met de extensies.

#### **5.1.1 Verschillende versies**

De JDBC API bestaat in drie verschillende versies, waarbij elke versie meer mogelijkheden biedt dan de vorige. Uiteraard omvat elke versie de vorige. Deze tekst behandelt hoofdzakelijk de basisfunctionaliteiten van de verschillende versies van de JDBC API.

##### **JDBC 1.0**

De JDBC 1.0 API bevat de basisklassen en interfaces om een verbinding te maken met een gegevensbank, een aantal SQL-opdrachten uit te voeren op die gegevensbank en de resultaten van die opdrachten te verwerken. Deze API maakt deel uit van de JDK 1.1.x.

##### **JDBC 2.0**

JDBC 2.0 bestaat uit twee delen: een basispakket en een optioneel pakket. Het basispakket werd opgenomen vanaf de JDK 1.2. Het optionele deel is intussen deel van het J2SE-platform, versie 1.4. Het

basispakket omvat JDBC 1.0 en voegt er een aantal nieuwe functionaliteiten aan toe. Deze nieuwe functionaliteiten zijn o.a. aanpasbare *results sets*, wijzigingen via batch, programmatisch aanpassen van de gegevensbank, ondersteuning van de SQL3-gegevenstypes.

Het optioneel deel maakt deel uit van het J2EE-platform waarmee professionele serverapplicaties ontwikkeld kunnen worden. Het omvat o.a. transacties over verschillende gegevensbanken, *connection pooling*, JNDI voor gegevensbanken, ...

### JDBC 3.0

Versie 3.0 is een uitbreiding van 2.0 en bevat meteen ook alle extensies. Ze beperkt zich in principe niet tot gegevensbanken, maar kan ook gebruikt worden voor andere gegevensstructuren in tabelvorm, zoals rekenbladen en gewone bestanden (*flat files*). Deze versie wordt standaard bij Java 1.4 gebundeld. Een overzicht van de nieuwe functionaliteiten vind je op de JDBC-website van Sun Microsystems JDBC Technology Guide: Getting Started - Appendix A: Summary of New Features

## 5.2 JDBC-drivers

Om client-applicaties toegang te geven tot een gegevensbankserver voorzien producenten een stel van (productafhankelijke) API's. Vanuit programma's (bv. in C++) kan dan gebruik gemaakt worden van deze API's om gegevens uit de gegevensbank op te halen of aan te passen. De JDBC API biedt een alternatief voor deze productafhankelijke API's, namelijk een API die voor elke SQL-compatibele relationele gegevensbank gebruikt kan worden, onafhankelijk van het gekozen product. Dit betekent dat het mogelijk zou moeten zijn om van gegevensbank te veranderen zonder het programma aan te passen.

De opdrachten van deze JDBC API moeten echter nog vertaald worden naar opdrachten in de productafhankelijke API. Met andere woorden er moet implementaties voorzien worden voor de interfaces uit de JDBC API. Deze implementatie noemt men een JDBC *driver*. Deze drivers vertalen de algemene JDBC-opdrachten naar productafhankelijke opdrachten. Deze JDBC driver wordt meestal ter beschikking gesteld door de producent van de gegevensbank, maar soms ook door een onafhankelijke firma.

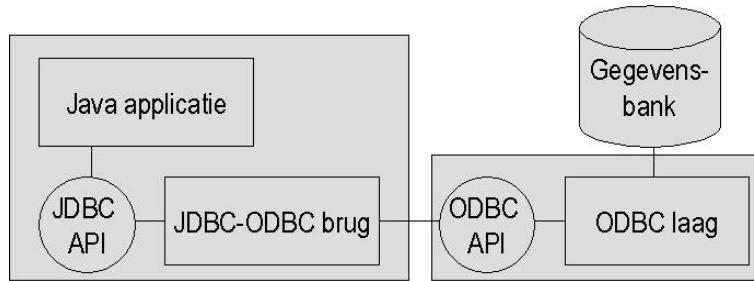
### 5.2.1 Verschillende types JDBC-drivers

Er bestaan vier soorten JDBC-drivers om een applicatie te verbinden met een gegevensbankserver.

**JDBC/ODBC-brug** ODBC (*Open Database Connectivity*) is de Microsoft's algemene API voor gegevensbanken. De ODBC API voorziet een aantal methodes om toegang te krijgen tot een gegevensbank. Deze methodes worden op zich geïmplementeerd door een ODBC-driver. De JDBC/ODBC-brug zorgt voor de vertaling van JDBC-opdrachten naar ODBC-opdrachten die op hun beurt door de ODBC-laag vertaald worden naar gegevensbankopdrachten.

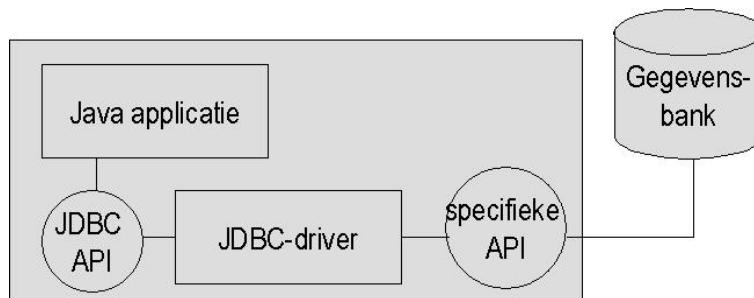
Deze werkwijze is nogal inefficiënt en dus niet bruikbaar voor applicaties die een zeer performante gegevensbanktoegang nodig hebben. Bovendien wordt de functionaliteit van de JDBC API beperkt tot de functionaliteit van de ODBC-driver.

De JDBC/ODBC-brug maakt deel uit van de JDK1.1 (Solaris of Windows-versie). De client-computer moet echter ook de ODBC-driver installeren en configureren.



Figuur 5.1: Structuur XML-document

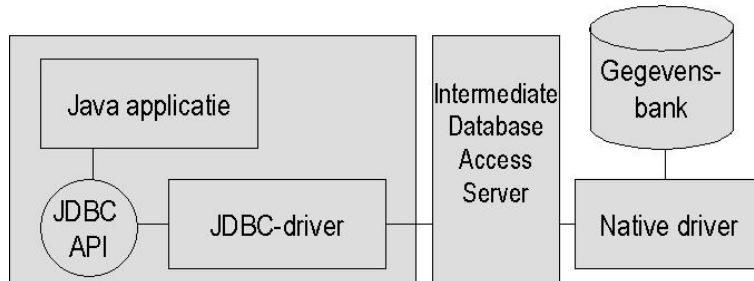
**Deels Java, deels productafhankelijk** De JDBC-drivers die onder deze categorie vallen, vertalen de methoden uit de JDBC-API naar methoden van de productafhankelijke API. Deze driver is efficiënter dan de JDBC/ODBC-brug en de applicatie kan gebruik maken van de volledige functionaliteit van de API voorzien door de producent. Ook hier moet een stuk binaire code specifiek voor het gegevensbanksysteem geïnstalleerd worden op de client-computer.



Figuur 5.2: Structuur XML-document

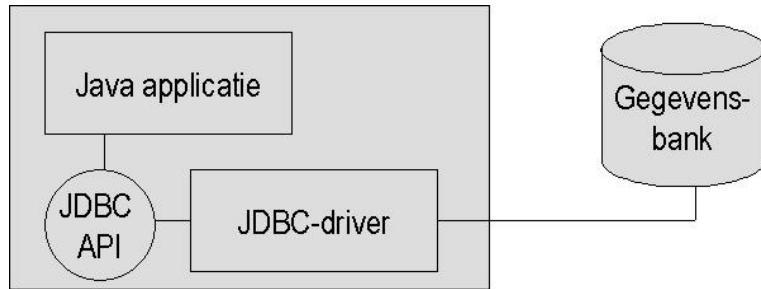
**JDBC-Net** JDBC-drivers van dit type maken gebruik van een tussenliggende gegevensbankserver. Java-clients kunnen nu met verschillende gegevensbanken connecteren via deze tussenliggende server. De tussenliggende server kan gebruik maken van verschillende productafhankelijke protocollen om verbindingen te maken met de verschillende gegevensbanken. De JDBC-opdracht van de Java-applicatie wordt nu via de JDBC-driver doorgestuurd naar de tussenliggende gegevensbankserver, die de aanvraag dan verder afhandelt.

Dit type driver is uitermate geschikt wanneer een applicatie gebruik moet maken van verschillende gegevensbanken.



Figuur 5.3: Structuur XML-document

**Pure Java** Deze drivers zijn volledig in Java geschreven en maken directe netwerkverbindingen met de gegevensbank (met behulp van de *Socket API*). Hierbij wordt gebruik gemaakt van productafhankelijke netwerkprotocols. Deze werkwijze is de meest efficiënte wat betreft performantie, tijd om de applicatie te ontwikkelen en installatie.



Figuur 5.4: Structuur XML-document

### 5.2.2 Laden van de driver

Om gebruik te kunnen maken van de JDBC-drivers die horen bij de gegevensbanken die een applicatie nodig heeft, moeten deze drivers worden ingeladen. Dit verloopt als volgt:

---

```

import java.sql.*;
...
private final static String PAR_DRIVER = "org.postgresql.Driver";
...
try {
 Class.forName (PAR_DRIVER);
} catch (ClassNotFoundException e) {
 ... // driver niet gevonden
}

```

---

De constante *PAR\_DRIVER* stelt de volledige klassennaam van de JDBC-driver voor. In het voorbeeld wordt een PostgreSQL-gegevensbank gebruikt. Merk op dat in een echte applicatie de driverklasse niet hardgedeclareerd wordt, maar wordt opgenomen in een configuratiebestand. De driverklasse wordt geleverd door de gegevensbankproducent en maakt uiteraard geen deel uit van de standaard Java API. De naam van de klasse vind je terug in de driverdocumentatie. Om de klassen van het driverpakket te kunnen gebruiken moet het jar-bestand dat de driverklasse bevat zich in het ‘classpath’ bevinden. Elke driver implementeert de *java.sql.Driver*-interface.

De methode *forName* laadt de driverklasse in de virtuele machine. Tijdens het laden van de driverklasse moet die een instantie van zichzelf aanmaken en registeren bij de *DriverManager*. De klasse *java.sql.DriverManager* vormt een gemeenschappelijke toegangslaag tot de verschillende JDBCdrivers van een applicatie. Merk op dat het perfect mogelijk is om tegelijkertijd verschillende drivers actief te hebben. Je toepassing kan op die manier bijvoorbeeld tegelijkertijd gebruik maken van verschillende gegevensbanken van verschillende leveranciers.

### 5.2.3 Verschillende versies van de JDBC API

Niet elke versie van de JDBC API wordt ondersteund door alle JDBC-drivers. De driverdocumentatie specificeert de JDBC-versie. In de online JDBC API wordt bij elke klasse en bij elke interface vermeld tot

welke JDBC-versie ze behoort. Dit maakt het mogelijk om enkel die klassen en interfaces te gebruiken die ondersteund worden door de gekozen driver.

## 5.3 Verbindingen met een gegevensbank

Eenmaal de gepaste JDBC-driver is ingeladen, kan je een verbinding met een gegevensbank opzetten. Een dergelijke verbinding wordt voorgesteld door een object van het type *Connection*. *Connection* is een interface die behoort tot het pakket *java.sql* (net zoals de andere JDBC-interfaces en -klassen die we hier bespreken). De eigenlijke implementatie van een verbinding is afhankelijk van de gebruikte driver.

Het volgende voorbeeld illustreert hoe een verbinding met de gegevensbank geopend en gesloten kan worden.

---

```
private final static String
 PAR_JDBC_URL = "jdbc:postgresql:voorbeelden";
private final static String PAR_LOGIN = "iiii4";
private final static String PAR_PASWORD = "iiii4pwd";
...
try {
 Connection conn =
 DriverManager.getConnection
 (PAR_JDBC_URL, PAR_LOGIN, PAR_PASWORD);
 try {
 ... // gegevens ophalen, toevoegen, veranderen, ...
 } finally {
 conn.close();
 }
} catch (SQLException e) {
 ... // gegevensbankproblemen
}
```

---

Om de verbinding aan te maken, gebruik je de methode *getConnection* van *DriverManager*. Deze methode heeft drie argumenten: een JDBC-URL, een gebruikersnaam en een wachtwoord. De gebruikersnaam en het wachtwoord zijn de gebruikersnaam en het wachtwoord van het DBMS (*Database Management System*). De JDBC-URL bestaat uit drie stukken.

**het protocol** In een JDBC-URL is het protocol altijd `jdbc`:

**het subprotocol** Het subprotocol identificeert de gegevensbankdriver en wordt bepaald door de gegevensbankproducent. In het voorbeeld is het subprotocol `postgresql:`, bij een JDBC/ODBC-brug is dit bijvoorbeeld `odbc:`.

**de ‘subname’** Dit is het laatste gedeelte van de JDBC-URL en het identificeert de gebruikte gegevensbank op een manier die kan verschillen van driver tot driver. De subname kan bijvoorbeeld de naam van de gegevensbank, de server en de poort waarop de gegevensbank draait bevatten. Bij een PostgreSQL-gegevensbank kan dit één van de volgende vormen aannemen:

```
naamdatabase
//host/naamdatabase
//host:poort/naamdatabase
```

Hierbij is `naamdatabase` de naam van de gegevensbank, `host` de naam van de server waarop de gegevensbank draait en `poort` de (netwerk)poort waarop de gegevensbank luistert naar aanvragen.

De verbinding zelf wordt gelegd met de methode *getConnection* die automatisch de juiste driver bepaalt aan de hand van de JDBC-URL. De methode *close* (van *Connection*) sluit de verbinding af.

Merk op dat de meeste methoden uit de JDBC API uitzonderingen kunnen genereren van het type *SQLException*. In bovenstaand fragment worden die opgevangen door de buitenste **try/catch**-blok. De **try/finally**-blok binnendien zorgt ervoor dat de verbinding met de databank zeker wordt afgesloten, zelfs wanneer er een fout optreedt. We hebben hier twee vernestelde **try**-opdrachten nodig omdat ook de *close* een uitzondering kan veroorzaken.

Verder merken we op dat de JDBC-URL, de loginnaam en het paswoord normaal niet hardgedeeld worden, maar bv. worden ingelezen uit een configuratiebestand.

## 5.4 SQL-opdrachten

Wanneer de verbinding is gemaakt, kan je de gegevensbank vanuit een Java-programma een aantal SQL-opdrachten laten uitvoeren. Om dit te realiseren heb je een object van het type *Statement* nodig. Dit object stuurt SQL-opdrachten naar de gegevensbank en haalt de resultaten ervan op. *Statement* is een interface, de eigenlijke implementatie wordt bepaald door de JDBC-driver. Een *Statement*-object wordt op de volgende wijze aangemaakt en terug afgesloten:

---

```
try {
 Statement stmt = conn.createStatement ();
 try {
 ... // SQL-opdrachten uitvoeren
 } finally {
 stmt.close();
 }
} catch (SQLException e) {
 ...
}
```

---

Het object *conn* is van het type *Connection* en stelt een verbinding met de gegevensbank voor. De methode *close* zorgt ervoor dat de gereserveerde geheugenruimte in de gegevensbank en in de virtuele machine terug wordt vrijgegeven.

Met behulp van een *Statement*-object kan elke SQL-opdracht die de gegevensbank ondersteunt uitgevoerd worden. Als de applicatie met verschillende gegevensbanksystemen moet kunnen werken, dan moet de SQL-opdracht door al die systemen ondersteund worden. De gegevensbankafhankelijkheid van de Java-applicatie wordt dus sterk bepaald door de gekozen SQL-opdrachten.

Bij het gebruik van het *Statement*-object wordt een onderscheid gemaakt tussen SQL-opdrachten die rijen of records van de databank als resultaat hebben (zoekopdrachten) en andere opdrachten.

### 5.4.1 DDL-opdrachten, insert, delete en update

Niet-zoekopdrachten zijn opdrachten die deel uitmaken van de *Data Definition Language* (DDL) en opdrachten die rijen in de gegevensbank toevoegen, verwijderen of aanpassen. De eerste reeks opdrachten wordt gebruikt om tabellen, gebruikers, en dergelijke, aan te maken en te verwijderen. De tweede reeks gebruikt de SQL-opdrachten *INSERT*, *DELETE* en *UPDATE* uit de *Data Manipulation Language* (DML). In beide gevallen wordt de methode *executeUpdate* van *Statement* opgeroepen om de opdracht uit te voeren. De signatuur van deze methode is de volgende.

---

```
public int executeUpdate (String sqlOpdracht)
 throws SQLException;
```

---

Het argument van deze methode is een *String* die de SQL-opdracht voorstelt. De SQL-opdracht bevat geen afsluitteken omdat dat verschillend is voor verschillende gegevensbanksystemen. De waarde is het aantal aangepaste rijen. In het geval van een DDL-opdracht is dit uiteraard 0.

In het volgende voorbeeld wordt een tabel `temp` aangemaakt die uit één kolom bestaat. De naam van deze kolom is `nr` en bevat een getal. Vervolgens wordt een rij aan deze tabel toegevoegd die bestaat uit het cijfer 1. Daarna wordt in alle rijen de waarde 1 vervangen door de waarde 2. Tot slot worden alle rijen met waarde 2 verwijderd.

---

```
// SQL-opdrachten (DDL)
String maakTabel = "create table temp (nr numeric)";
// SQL-opdrachten (DML)
String voegRijToe = "insert into temp values (1)";
String pasRijkenAan = "update temp set nr=2 where nr=1";
String verwijderRijken = "delete from temp where nr=2";

// uitvoeren SQL-opdrachten
stmt.executeUpdate (maakTabel);
stmt.executeUpdate (voegRijToe);
int updateCnt = stmt.executeUpdate (pasRijkenAan);
stmt.executeUpdate (verwijderRijken);
```

---

De waarde van de variabele `updateCnt` in dit voorbeeld is 1.

## 5.4.2 Zoekopdrachten

### **ResultSet bepalen**

Zoekopdrachten zijn SELECT-opdrachten die gegevens selecteren uit één of meerdere tabellen in de gegevensbank en hun resultaat teruggeven als een aantal rijen met gegevens in één of meerdere kolommen. Deze opdrachten worden uitgevoerd met behulp van de methode `executeQuery` van *Statement*.

Het resultaat van de zoekactie is een object van het type *ResultSet*. Je kan dan later de inhoud van dit object (dat eigenlijk een soort tabel is met rijen en kolommen) opvragen met behulp van gepaste methoden.

---

```
public ResultSet executeQuery (String sqlOpdracht)
 throws SQLException;
```

---

De *StringsqlOpdracht* stelt de SELECT-opdracht voor.

### **Het ResultSet-object overlopen**

Elk *ResultSet*-object bevat een wijzer (ook *cursor* genoemd). Deze wijzer staat na het aanmaken van het object voor de eerste rij. Met deze wijzer wordt het *ResultSet*-object rij na rij overlopen.

Om de cursor één rij naar beneden te schuiven, gebruik je de methode `next` (zonder parameters). Deze functie geeft **true** terug als er nog rijen zijn en **false** in het andere geval. Ze wordt vaak gebruikt in een lus van de volgende vorm:

---

```
ResultSet res = stmt.executeQuery (...);
while (res.next ()) {
 ... // haal de gegevens op van de huidige rij
}
```

---

## Data ophalen

Uit de rij waar de cursor naar wijst, kunnen gegevens opgevraagd worden. De interface *ResultSet* biedt verschillende *getter*-methoden om gegevens op te vragen. Bijvoorbeeld,

---

```
public boolean getBoolean (int columnIndex)
 throws SQLException;
public double getDouble (int columnIndex)
 throws SQLException;
public float getFloat (int columnIndex)
 throws SQLException;
public int getInt (int columnIndex)
 throws SQLException;
public String getString (int columnIndex)
 throws SQLException;
```

---

De methode *getXxx* geeft de waarde van de kolom met volgnummer *columnIndex* terug als een object of variabele van het type *xxx*. (Merk op dat de eerste kolom volgnummer 1 heeft.) Men raadt aan om de gegevens van een bepaalde rij zoveel mogelijk op te vragen in stijgende volgorde van kolomnummers.

Daarnaast bestaan er ook gelijknamige (iets minder efficiënte) methoden waarbij je de gewenste kolom mag aanduiden met behulp van zijn naam. Bijvoorbeeld,

---

```
public boolean getBoolean (String columnName)
 throws SQLException;
public double getDouble (String columnName)
 throws SQLException;
public float getFloat (String columnName)
 throws SQLException;
public int getInt (String columnName)
 throws SQLException;
public String getString (String columnName)
 throws SQLException;
```

---

In beide gevallen probeert de JDBC-driver de onderliggende data te converteren naar het Java-type gespecificeerd door de *getter*-methode. Deze methode geeft dan de overeenkomstige Java-waarde weer. Een overzicht van welke SQL-types in welke Java-types omgezet kunnen worden en omgekeerd vind je in de JDBC specificaties.

De implementatie van *ResultSet* kan verschillen van driver tot driver: in sommige gevallen wordt de inhoud van dit object meteen opgevuld wanneer *executeQuery* wordt opgeroepen, in andere gevallen wordt de gegevensbank slechts gecontacteerd op het moment dat de gegevens met de *getXxx*-methoden wordt opgevraagd. Dit verklaart waarom elk van deze methoden een *SQLException* kan veroorzaken.

## Voorbeeld

In het volgende fragment maken we een lijst aan van alle boeken die zich in een gegevensbank bevinden. Een boek wordt gekenmerkt door een uniek ISBN-nummer, een titel en een prijs. In de gegevensbank worden de boeken bewaard in een tabel boeken met kolommen *isbn*, *titel* en *prijs*. In het geheugen stellen we de boekenlijst voor als een *ArrayList* van *Boek*-objecten.

---

```
static final String QUERY_BEPAL_BOEKEN
 = "select * from boeken";
static final String PARAM_ISBN = "isbn";
static final String PARAM_TITEL = "titel";
static final String PARAM_PRIJS = "prijs";
...
List boeken = new ArrayList ();
try {
 Statement stmt = conn.createStatement ();
 try {
 ResultSet rs = stmt.executeQuery (
```

---

```

 QUERY_BEPAAL_BOEKEN);
while (rs.next ()) {
 Boek boek = new Boek (rs.getString (PARAM_ISBN),
 rs.getString (PARAM_TITEL),
 rs.getDouble (PARAM_PRIJS));
 boeken.add (boek);
}
} finally {
 stmt.close ();
}
} catch (SQLException e) {
...
}

```

---

Hierbij stelt *conn* een verbinding met de gegevensbank voor die in een ander gedeelte van het programma wordt geopend en gesloten. De constructor van *Boek* heeft drie parameters: het unieke ISBN-nummer (een string), de titel (een string) en de prijs (een reëel getal) van het boek.

### Sluiten van een *ResultSet*-object

Een *ResultSet*-object wordt automatisch afgesloten door het *Statement*-object dat het genereerde wanneer dit *Statement*-object gesloten wordt of opnieuw uitgevoerd wordt. Het kan eventueel expliciet gesloten worden met de methode *close*.

## 5.5 Prepared Statements

Als een SQL-opdracht meermaals uitgevoerd moet worden of verschillende keren uitgevoerd moet worden met andere parameters, dan is het efficiënter om een *PreparedStatement*-object te gebruiken. *PreparedStatement* is een interface afgeleid van de interface *Statement*. Tijdens het aanmaken van een prepared statement wordt reeds een SQL-opdracht meegegeven. Wanneer de driver dit ondersteunt wordt deze opdracht onmiddellijk naar de gegevensbank doorgestuurd en daar gecompileerd. Dit zorgt ervoor dat de SQL-opdracht sneller zal worden uitgevoerd.

### 5.5.1 Aanmaken van een *prepared statement*

Om een prepared statement aan te maken wordt de methode *prepareStatement* van een *Connection*-object opgeroepen. Het argument van deze methode is de SQL-opdracht, waarin eventuele parameters nog niet zijn ingevuld. Deze parameters worden door vraagtekens voorgesteld.

In het volgend voorbeeld wordt een prepared statement gecreëerd waarmee een rij toegevoegd kan worden aan een tabel *bestelling*. Deze tabel heeft drie kolommen: *klant*, *boek* en *aantal*. Op deze manier wordt bijgehouden hoeveel exemplaren een klant van één of meerdere boeken bestelt. Het object *conn* is een *Connection*-object dat op een andere plaats in de applicatie aangemaakt en afgesloten wordt.

---

```

static final String BEWAAR_BESTELLING_OPDRACHT
 = "insert into bestelling (klant, boek, aantal)"
 + " values (?,?,?)";
...
try {
 PreparedStatement bewaarBestellingStmt =
 conn.prepareStatement (BEWAAR_BESTELLING_OPDRACHT);
 try {
 ... // bestelling bewaren (zie verder)
 } finally {
 bewaarBestellingStmt.close();
 }
}

```

```

} catch (SQLException e) {
 ...
}

```

---

In dit voorbeeld werd een *PreparedStatement*-object aangemaakt dat zal gebruikt worden om rijen toe te voegen aan een tabel *bestelling*. De vraagtekens in de SQL-opdracht van *bewaarBestellingStmt* stellen parameters voor die later een waarde krijgen. Hetzelfde *PreparedStatement*-object kan opnieuw gebruikt worden met verschillende waarden voor elke parameter.

### 5.5.2 Parameters toekennen

Vooraleer je het aangemaakte *prepared statement* kan uitvoeren moet je nog waarden toekennen aan elke parameter. We illustreren dit eerst met een voorbeeld.

De volgende lijnen Java-code werken het bovenstaande voorbeeld verder uit. Voor een klant, gekenmerkt door de variabele *klantcode*, worden de bestelde boeken bewaard. Er wordt ook bijgehouden hoeveel exemplaren de klant van elke boek wil. Hiervoor wordt een lijst *artikels* overlopen. De variabele *artikels* is van het type *java.util.List*. Deze lijst bevat een reeks *Artikel*-objecten. Een *Artikel* bestaat uit een boek en het aantal aangekochte exemplaren van dit boek. Deze gegevens verkrijg je via de methoden *getBoek* en *getHoeveelheid*. De klasse *Boek* heeft de methode *getISBN* om het ISBN-nummer van het boek op te vragen.

```

// bestelling bewaren
bewaarBestellingStmt.setString (1, klantcode);
Iterator loperArtikels = artikels.iterator ();
while (loperArtikels.hasNext ()) {
 Artikel artikel = (Artikel) loperArtikels.next ();
 bewaarBestellingStmt.setString(2, artikel.getBoek().getISBN());
 bewaarBestellingStmt.setInt(3, artikel.getHoeveelheid());
 bewaarBestellingStmt.executeUpdate();
}

```

---

De parameters van een prepared statement worden ingevuld met methoden van de volgende vorm:

```

public void setXxx (int indexParameter, xxx waardeParameter)
 throws SQLException;

```

---

Hierbij is *xxx* het Java-type van de parameter. Het eerste argument van de methode bepaalt het volgnummer van de parameter die de meegegeven waarde *wardeParameter* moet krijgen (parameters worden genummerd vanaf 1). Alle parameters moeten op deze manier een waarde krijgen vooraleer de SQL-opdracht uitgevoerd kan worden. De methode *setXxx* converteert de waarde van de parameter naar een waarde van een gegevensbanktype. Zo wordt een *String* bijvoorbeeld omgezet in een *VARCHAR*.

De methoden *executeUpdate* en *executeQuery* voeren dan uiteindelijk de opdracht uit. Net zoals bij *Statement* gebruik je voor zoekopdrachten de methode *executeQuery* en in het andere geval de methode *executeUpdate*. Merk op dat deze methoden hier geen argument hebben omdat de SQL-opdracht reeds werd opgegeven bij constructie van het *prepared statement*.

### 5.5.3 Gegevensconversie

Hieronder illustreren we nog een andere reden om *PreparedStatement* boven *Statement* te verkiezen als de SQL-opdracht parameters bevat. Veronderstel dat we in een tabel met studentengegevens alle studenten met een gegeven familienaam wensen op te zoeken. Deze familienaam wordt gegeven in de vorm van een variabele *naam*. We zouden dit op de volgende manier kunnen doen:

---

```
Statement stmt = conn.createStatement ();
ResultSet rs = stmt.executeQuery
 ("select * from studenten where name = \'" + naam + "\'");
...

```

---

of anders, met behulp van een prepared statement:

---

```
PreparedStatement pstmt = conn.prepareStatement
 ("select * from studenten where name = ?");
pstmt.setString (1, naam);
ResultSet rs = pstmt.executeQuery ();
...

```

---

Het nut van een prepared statement lijkt bij dit voorbeeld op het eerste zicht niet groot. We merken echter op dat wanneer *naam* een aanhalingsteken bevat, enkel het tweede fragment zal werken. Stel dat de variabele *naam* de waarde *D'haese* heeft, dan wordt de SQL-opdracht in het eerste fragment

```
select * from studenten where name = 'D'haese'
```

Aangezien dit geen correcte SQL-opdracht is, treedt er een fout op. Het tweede fragment werkt wel omdat hier een onderscheid gemaakt wordt tussen de SQL-opdracht en zijn parameters. Bij aanmaak van het *prepared statement* wordt bij de meeste drivers de SQL-opdracht gecompileerd in de gegevensbank. De parameters voor deze opdracht worden later doorgestuurd als parameters voor de reeds gecompileerde SQL-opdracht. Bovendien zorgt de methode *setXxx* waarmee de parameters ingesteld worden voor de conversie van de parameterwaarden. Speciale tekens worden dan omgezet.

#### 5.5.4 SQL-injectie

SQL-injectie kan voorkomen als een gebruiker bv. via een webapplicatie parameters kan ingegeven die gebruikt worden in een SQL-opdracht. Een malaïde gebruiker kan dan samen met die parameters een aantal willekeurige SQL-opdrachten meegegeven om voor hem onbeschikbare gegevens te zien, om gegevens te verwijderen, ... Met andere woorden hij ‘injecteert’ een aantal SQL-opdrachten samen met de parameter die hij ingeeft.

We illustreren dit met een voorbeeld. Veronderstel dat we uit een tabel met studentengegevens de gegevens van een student met een opgegeven studentennummer willen zien. Dit nummer wordt bv. ingegeven via een HTML-formulier. De variabele *studentNummer* bevat de ingegeven waarde. We zouden de studentengegevens kunnen opvragen met het volgende stukje code. *conn* is een *Connection*-object.

---

```
Statement stmt = conn.createStatement ();
ResultSet rs = stmt.executeQuery
 ("select * from studenten where studnr =
 + studentNummer");
...

```

---

Indien *studentNummer* de waarde *200200234* dan wordt bij het uitvoeren van de methode *executeQuery* de volgende SQL-opdracht uitgevoerd.

```
select * from studenten where studnr = 200200234
```

Een gebruiker met slechte bedoelingen zou echter ipv *200200234* bijvoorbeeld *200200234 OR 1=1* kunnen invoeren. De uitgevoerde SQL-opdracht is dan

```
select * from studenten where studnr = 200200234 OR 1=1
```

en de gegevens van alle studenten worden getoond. Nog slechter zou de invoer `200200234; DROP TABLE studenten` zijn. Dit zou betekenen dat er twee SQL-opdrachten worden uitgevoerd.

```
select * from studenten where studnr = 0
```

en

```
DROP TABLE studenten
```

De eerste opdracht zal geen resultaat geven, maar de tweede verwijdert mogelijk de tabel `studenten`.

Een ander voorbeeld van niet-gewenste SQL-injectie is bv. de invoer `0 UNION SELECT username, password FROM users`. Dit kan eventueel de gebruikersnamen en wachtwoorden vrijgeven.

Het gebruik van *prepared statements* voorkomt de bovenstaande SQL-injecties. We kunnen de bovenstaande code als volgt vervangen.

---

```
PreparedStatement pstmt = conn.prepareStatement
 ("select * from studenten where studnr = ?");
pstmt.setString (1, studentNummer);
ResultSet rs = pstmt.executeQuery ();
...
```

---

Bij aanmaak van het *PreparedStatement*-object wordt de SQL-opdracht

```
select * from studenten where studnr = ?
```

gecompileerd in de gegevensbank. De methode `setString` converteert de parameterinvoer naar een `VARCHAR` die als argument aan de gecompileerd SQL-opdracht wordt gegeven. Deze `String` wordt niet meer gecompileerd.

Aangezien een klein aantal drivers *prepared statements* niet voorcompileren is het nodig om dit expliciet uit te testen.

## 5.6 Callable Statements

*Callable Statements* dienen om *stored procedures* van een gegevensbanksysteem uit te voeren. Een *stored procedure* is een functie of procedure die bewaard wordt in een gegevensbank en die door clientapplicaties kan opgeroepen worden. Deze functie of procedure haalt informatie op uit de gegevensbank of manipuleert zijn data.

Er is geen standaardsyntax of -taal om *stored procedures* aan te maken of op te roepen. Bovendien wordt deze functionaliteit niet aangeboden door alle gegevensbanksystemen. Ondanks het gebrek aan standaardisering biedt de JDBC API een uniforme manier om *stored procedures* op te roepen vanuit een Java-applicatie.

### 5.6.1 Aanmaken *callable statement*

Om een *stored procedure* te kunnen oproepen moet je zijn naam en eventuele parameters kennen. In eerste instantie bekijken we hoe we een *stored procedure* zonder parameters kunnen uitvoeren.

Veronderstel dat we een *stored procedure* *SELECTEER\_BOEKEN* willen uitvoeren die ons een tabel met de gegevens van alle beschikbare boeken teruggeeft. Dan moeten we eerst een *CallableStatement*-object aanmaken dat een oproep voor de procedure *SELECTEER\_BOEKEN* bevat. Merk op dat dit object niet de procedure zelf bevat, maar enkel welke procedure het moet oproepen. De interface *CallableStatement* is een afgeleide interface van *PreparedStatement* en dus ook van *Statement*. Net zoals bij een *prepared statement* of een *statement* wordt een *CallableStatement*-object gecreëerd met behulp van een *Connection*-object (de variabele *conn* in het voorbeeld). In dit geval wordt de methode *prepareCall* gebruikt.

---

```
static final String SELECTEER_BOEKEN_OPDRACHT
 = "{call SELECTEER_BOEKEN}";
...
try {
 CallableStatement selecteerBoekenStmt =
 conn.prepareCall (SELECTEER_BOEKEN_OPDRACHT);
 try {
 ResultSet rs = selecteerBoekenStmt.executeQuery ();
 ...
 } finally {
 selecteerBoekenStmt.close ();
 }
} catch (SQLException e) {
 ...
}
```

---

Merk op dat de opdracht die de procedure *SELECTEER\_BOEKEN* oproept tussen accolades staat. Dit noemen ze de *escape-syntax*. De *escape-syntax* in JDBC wordt gebruikt voor opdrachten die niet in de standaard SQL-syntaxis kunnen geformuleerd worden. Deze opdrachten worden geformuleerd in de *escape-syntax* en tussen accolades geplaatst. Op deze manier is het toch mogelijk om op een gestandaardiseerde wijze *stored procedures* uit te voeren. De driver moet deze opdrachten dan vertalen naar opdrachten in de SQL-syntaxis van het gegevensbanksysteem. In het voorbeeld wordt de opdracht *call SELECTEER\_BOEKEN* vertaald naar de specifieke syntax van de gegevensbank om de *stored procedure* *SELECTEER\_BOEKEN* op te roepen.

Aangezien de procedure *SELECTEER\_BOEKEN* een zoekopdracht is, wordt de methode *executeQuery* gebruikt om de procedure uit te voeren. In het andere geval wordt uiteraard de methode *executeUpdate* opgeroepen. Indien de procedure meerdere opdrachten bevat en er dus eventueel meerdere resultaten kunnen zijn, moet de methode *execute* gebruikt worden. Meer informatie vind je in de Java API.

## 5.6.2 Invoerparameters

Aangezien de interface *CallableStatement* afgeleid is van de interface *PreparedStatement* is het gebruik van invoerparameters volledig analoog. De parameters zijn in dit geval de argumenten van de procedure en worden voorgesteld door vraagtekens. Met de methodes

---

```
public void setXxx (int indexParameter, xxx waardeParameter)
 throws SQLException;
```

---

krijgen de parameters een waarde, vooraleer de procedure uitgevoerd wordt. We illustreren dit met een voorbeeld.

Voor een klant, gekenmerkt door de variabele *klantcode*, worden de bestelde boeken bewaard. Er wordt ook bijgehouden hoeveel exemplaren de klant van elke boek wil. De procedure *BEWAAR\_BESTELLING* bewaart deze gegevens in de gegevensbank. Deze procedure heeft drie argumenten: de unieke code van de klant, het ISBN-nummer van het boek en het aantal bestelde exemplaren. De bestelling van de klant wordt in het programma bijgehouden in een lijst *artikels*. De variabele *klantcode* bevat de unieke identificatie van de klant. De variabele *artikels* is van het type *java.util.List*. Deze lijst bevat een reeks *Artikel*-objecten. Een *Artikel* bestaat uit een boek en het aantal aangekochte exemplaren

van dit boek. Deze gegevens verkrijg je via de methoden *getBoek* en *getHoeveelheid*. De klasse *Boek* heeft de methode *getISBN* om het ISBN-nummer van het boek op te vragen.

---

```
static final String BEWAAR_BESTELLING_OPDRACHT
 = "{call BEWAAR_BESTELLING(?, ?, ?)}";
...
try {
 CallableStatement bewaarBestellingStmt =
 conn.prepareCall(BEWAAR_BESTELLING_OPDRACHT);
 try {
 ...
 bewaarBestellingStmt.setString(1, klantcode);
 Iterator loperArtikels = artikels.iterator();
 while (loperArtikels.hasNext()) {
 Artikel artikel = (Artikel) loperArtikels.next();
 bewaarBestellingStmt.setString(2,
 artikel.getBoek().getISBN());
 bewaarBestellingStmt.setInt(3,
 artikel.getHoeveelheid());
 bewaarBestellingStmt.executeUpdate();
 }
 } finally {
 bewaarBestellingStmt.close();
 }
} catch (SQLException e) {
 ...
}
```

---

### 5.6.3 Uitvoerparameters

Naast invoerparameters is het voor *callable statements* ook mogelijk om uitvoerparameters te specificeren. Deze parameters corresponderen dan met de uitvoerparameters van de procedure in de gegevensbank. Een uitvoerparameter is een parameter die een waarde krijgt of die verandert tijdens het uitvoeren van de procedure. Na het oproepen van de procedure kan de waarde van de parameter opgevraagd worden.

Uitvoerparameters worden net zoals invoerparameters aangeduid met een vraagteken in de opdracht in de *escape*-syntax. Vooraleer het *callable statement* uitgevoerd kan worden moet de uitvoerparameter geregistreerd worden met de methode *registerOutParameter*.

---

```
public void registerOutParameter(int parameterIndex,
 int sqlType) throws SQLException
```

---

Hierbij is *parameterIndex* het volgnummer van de parameter (beginnend bij 1), m.a.w. met welk vraagteken uit de opdracht de parameter correspondeert. De variabele *sqlType* staat voor het generieke SQL-type van de parameter. Deze types zijn de constanten van de klasse *Types*, die zoals alle klassen en interfaces in dit hoofdstuk behoort tot het pakket *java.sql*. Indien het SQL-type van de parameter *DECIMAL* of *NUMERIC* is moet ook het aantal cijfers na de komma opgegeven worden.

---

```
public void registerOutParameter(int parameterIndex, int sqlType,
 int schaal) throws SQLException
```

---

De derde parameter *schaal* stelt het aantal cijfers na de komma voor.

Als de procedure uitgevoerd is, kunnen de waarden van de uitvoerparameters opgehaald worden. De interface *CallableStatement* voorziet hiervoor een reeks *getter*-methodes van de volgende vorm.

---

```
public xxx getXxx(int parameterIndex) throws SQLException
```

---

Het type *xxx* is het Java-type dat correspondeert met het SQL-type van de parameter. (Meer informatie over de conversie tussen Java- en SQL-types in de JDBC specificaties). *parameterIndex* stelt het volgnummer van de parameter voor.

In het volgend voorbeeld wordt een *stored procedure* *SELECTTEER\_BOEKEN* uitgevoerd. Deze procedure haalt niet alleen alle beschikbare boeken op maar heeft bovendien nog een uitvoerparameter die aangeeft hoeveel boeken er beschikbaar zijn. De uitvoerparameter kan het resultaat van een functie zijn of een argument van een procedure. De *String* die de opdracht voorstelt is dus

---

```
final static String SELECT_BOEKEN_OPDR =
 "{ ? = call SELECTTEER_BOEKEN }";
```

---

of

---

```
final static String SELECT_BOEKEN_OPDR =
 "{ call SELECTTEER_BOEKEN(?) }";
```

---

afhankelijk van hoe de procedure gedeclareerd is in de gegevensbank. Het aanmaken van het *CallableStatement*-object, het registeren van de uitvoerparameter, het uitvoeren van de procedure en het ophalen van de boeken en het aantal beschikbare boeken verloopt als volgt. De variabele *conn* is een *Connection*-object.

---

```
CallableStatement cstmt = conn.prepareCall(SELECT_BOEKEN_OPDR);
try {
 cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
 ResultSet rs = cstmt.executeQuery();
 ... // boeken uit de ResultSet halen
 int aantalBoeken = cstmt.getInt(1);
 ...
} catch (SQLException e) {
 ...
} finally {
 cstmt.close();
```

---

Het is aan te raden om de uitvoerparameters pas te bepalen als alle gegevens uit de *ResultSet* zijn opgehaald. Dit omwille van de beperkingen van sommige gegevensbanksystemen.

Het volgend voorbeeld illustreert het samen voorkomen van invoer- en uitvoerparameters. De procedure in de gegevensbank selecteert de boeken die tot een bepaalde categorie (de variabele *categorieCode*) behoren en bepaalt ook het aantal geselecteerde boeken.

---

```
final static String SELECT_BOEKEN_VR_CAT_OPDR =
 "{ ? = call SELECTTEER_BOEKEN_VR_CAT(?) }";
...
CallableStatement cstmt = conn.prepareCall(
 SELECT_BOEKEN_VR_CAT_OPDR);
try {
 cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
 cstmt.setString(2,categorieCode);
 ResultSet rs = cstmt.executeQuery();
 ... // boeken uit de ResultSet halen
 int aantalBoeken = cstmt.getInt(1);
 ...
} catch (SQLException e) {
 ...
} finally {
 cstmt.close();
```

---

Merk op dat uit de opdracht in de *escape*-syntax niet blijkt welke parameter een invoerparameter is en welke een uitvoerparameter.

## 5.6.4 Invoer- en uitvoerparameters

Sommige parameters zijn zowel invoer- als uitvoerparameters. Dat betekent dat de parameter een waarde heeft die meegegeven wordt aan de *stored procedure* en dat de waarde van de parameter verandert tijdens het uitvoeren van de procedure. Om dit te realiseren krijgt de parameter een waarde via een *setter*-methode

en wordt geregistreerd met de methode *registerOutParameter*. Na het oproepen van de procedure en even-tueel het ophalen van de *ResultSet* kan dan de waarde van de parameter opgevraagd worden.

Veronderstel een procedure *BEWAAR\_BESTELLING* in een gegevensbank. Deze procedure bewaart voor een bepaalde klant hoeveel hij van een welbepaald boek bestelt. Bovendien past deze procedure het totaal aantal bestelde boeken van deze klant aan. De procedure krijgt mee hoeveel boeken de klant tot nu toe reeds bestelde. Het totaal aantal parameters van deze procedure is dus vier: de klantcode, het ISBN-nummer van het boek, het aantal bestelde exemplaren van dit boek en het totaal aan bestelde boeken van de klant. De eerste drie parameters zijn invoerparameters. De laatste parameter is zowel een invoer- als een uitvoerparameter. Het stukje Java-code die voor één klant en één boek de bestelling bewaart ziet er dan zo uit.

---

```
final static String BEWAAR_BESTELLING_OPDR =
 "{call BEWAAR_BESTELLING(?, ?, ?, ?)}";
...
CallableStatement cstmt = conn.prepareCall(
 BEWAAR_BESTELLING_OPDR);
try {
 ...
 // parameterwaarden toekennen
 cstmt.setString(1,klantCode);
 cstmt.setString(2,ISBNNummer);
 cstmt.setInt(3,aantalBoeken);
 cstmt.setInt(4,totaalAantalBoeken);
 // uitvoerparameter registreren
 cstmt.registerOutParameter(4, java.sql.Types.INTEGER);
 // procedure oproepen
 cstmt.executeUpdate();
 // uitvoerparameter ophalen
 int totaalAantalBoeken = cstmt.getInt(4);
 ...
} catch (SQLException e) {
 ...
} finally {
 cstmt.close();
```

---

## 5.7 Transacties

Sommige applicaties vereisen dat een reeks SQL-opdrachten ofwel allemaal succesvol zijn ofwel allemaal niet uitgevoerd worden. Het is niet toegelaten dat een deel van de opdrachten wel en een ander deel niet uitgevoerd worden. Als alle opdrachten uitgevoerd konden worden, dan mogen ze permanent gemaakt worden in de databank (*commit* in het Engels), in het andere geval moeten alle reeds uitgevoerde opdrachten ongedaan gemaakt worden (*rollback* in het Engels). Een transactie is een reeks opdrachten die samen uitgevoerd moeten worden.

De interface *Connection* voorziet de volgende methoden om transacties te implementeren.

---

```
public void setAutoCommit(boolean autoCommit)
 throws SQLException;
public void commit() throws SQLException;
public void rollback() throws SQLException;
```

---

Een *Connection*-object wordt standaard aangemaakt in de mode ‘auto-commit’. Dit betekent dat de geslaagde uitvoering van een SQL-opdracht onmiddellijk permanent is. Als de methode *executeQuery* of *executeUpdate* van een *Statement*-object succesvol is, dan is zijn resultaat permanent.

Om meerdere opdrachten samen te kunnen uitvoeren moet de auto-commit-mode uitgeschakeld worden. De methode *setAutoCommit* schakelt de auto-commit-mode aan of uit afhankelijk van het argument (respectievelijk **true** of **false**). Indien de auto-commit-mode uitgeschakeld is, wordt het resultaat van één of

meerdere SQL-opdrachten uitgevoerd over de verbinding pas permanent na het oproepen van de methode *commit*.

Om het resultaat van een aantal SQL-opdrachten ongedaan te maken roep je de methode *rollback* aan. Alle opdrachten sinds de laatste *commit* worden dan verwijderd.

In het voorbeeld uit §5.5 kunnen we eisen dat de bestelling van één klant volledig ingevoerd moet worden of als dat niet kan niets ingevoerd mag worden. De code wordt dan als volgt aangepast.

---

```

try {
 // bestelling bewaren
 bewaarBestellingStmt.setString(1,klantcode);
 Iterator loperArtikels = artikels.iterator();
 conn.setAutoCommit (false);
 while (loperArtikels.hasNext()) {
 Artikel artikel = (Artikel) loper.next ();
 bewaarBestellingStmt.setString(2,
 artikel.getBoek().getId());
 bewaarBestellingStmt.setInt(3,
 artikel.getHoeveelheid());
 bewaarBestellingStmt.executeUpdate ();
 }
 conn.commit ();
} catch (SQLException e) {
 conn.rollback ();
} finally {
 conn.setAutoCommit (true);
}

```

---

In het bovenstaande wordt de methode *executeUpdate* van *bewaarBestelling* nul of meerdere keren opgeroepen afhankelijk van het aantal verschillende boeken dat de klant bestelde. Indien er iets misloopt bij de uitvoering van één van de methodes *executeUpdate* dan worden de resultaten van alle uitvoeringen van deze methode voor de huidige klant ongedaan gemaakt. Het oproepen van de methode *rollback* in het *catch*-blok zorgt daarvoor. Konden alle methodes *executeUpdate* wel goed uitgevoerd worden dan worden de wijzigingen permanent gemaakt met de methode *commit*. In beide gevallen moet nadien de verbinding weer in de mode ‘auto-commit’ geplaatst worden. Zo kan het *Connection*-object opnieuw correct gebruikt worden op een andere plaats.

Een ander effect van het gebruik van transacties is dat andere gebruikers van de gegevensbank het resultaat van de aanpassingen pas merken als de methode *commit* is uitgevoerd.

Merk op dat bij gelijktijdige transacties elke transactie een ander *Connection*-object moet hebben. Verschillende transactie kunnen uiteraard niet gelijktijdig over hetzelfde *Connection*-object verlopen.

### 5.7.1 De JNDI API

De JNDI API voorziet in een aantal interfaces en klassen om in een Java-applicatie gebruik te maken van *naming* en *directory services*. Deze API is zo opgebouwd dat hij onafhankelijk is van een specifieke *service*. Op deze manier wordt het aanspreken van bestaande *naming* en *directory services* in een Java-programma mogelijk. Bovendien kan van deze API gebruik gemaakt worden om Java-objecten die gedeeld worden door verschillende applicaties, te bewaren en op te zoeken.

De basispakketten van de JNDI API zijn *javax.naming* en *javax.naming.directory*. In deze paragraaf bespreken we geen klassen of interfaces van de JNDI API. Hun functionaliteit wordt besproken in §5.7.2.

### Naming service

Het verbinden van verstaanbare namen met computerobjecten is de taak van een *naming service*. Het bestandssysteem en het *Domain Name System* zijn voor de hand liggende voorbeelden van een *naming service*. Bestandsnamen zijn verbonden met *handlers* die toegang verschaffen tot het eigenlijke bestand. DNS-namen zijn gelinked aan IP-adressen van computers. De *naming service* bewaart niet alleen de verbindingen tussen de namen en de objecten, maar biedt bovendien de mogelijkheid om een object op te zoeken of terug te vinden op basis van zijn naam.

De *naming service* bepaalt de syntax waaraan de gebruikte namen moeten voldoen. Dit wordt ook de naamconventie van de *naming service* genoemd.

Een *naming service* bestaat uit een aantal contexten met dezelfde naamconventie. Een *context* bestaat uit een aantal naam/object-verbindingen en heeft een eigen naamconventie. Een context heeft een opzoekactie om object terug te vinden aan de hand van zijn naam. Een naam kan verbonden zijn met een andere context met dezelfde naamconventie. Deze context noemen we dan de subcontext van de huidige context. Op deze manier kan een *naming service* opgebouwd worden uit verschillende contexten. In een bestandssysteem zijn de mappen (directory's) de contexten.

### Directory service

Een *naming service* kan uitgebreid worden met een *directory service*. Die associeert attributen met de objecten in de *naming service*. Een attribuut bestaat uit een naam en mogelijke waarden.

Een *directory service* maakt het mogelijk om objecten te zoeken op basis van hun attributen ipv op naam. Uiteraard voorziet deze service ook in acties om de attributen te veranderen, aan te maken en te verwijderen.

De hiërarchische structuur van de *directory service* correspondeert met de contexten in de *naming service*.

## 5.7.2 Een *DataSource*-object aanmaken

Een *DataSource*-object is een fabriek (*factory*) voor *Connection*-objecten. De gegevensbron waarmee de verbindingen gelegd worden, kan zowel een gegevensbankserver zijn als bijvoorbeeld een bestand. De interface *DataSource* behoort tot het pakket *javax.sql* en maakt deel uit van de JDBC 2.0 API.

Een *DataSource*-object dat we willen gebruiken om een verbinding te maken met een gegevensbank moet geregistreerd zijn bij een *naming service*. Met behulp van de JNDI API kan dit object dan opgevraagd worden.

Dit betekent dat het Java-framework (ook wel *application server* genoemd) waarin jouw applicatie draait een *naming service* moet hebben. Bovendien moet er een *DataSource*-object geregistreerd zijn bij die *naming service*. Dat is de taak van de systeembeheerder van de *application server*.

Deze configuratie bestaat uit de volgende stappen:

**DataSource-klasse instellen** Je hebt een JDBC 2.0 driver nodig die *data sources* ondersteund. De klasse-naam van de *data source* van de gekozen driver moet opgegeven worden zodat een object van deze klasse aangemaakt kan worden door een tool van de *application server*.

**Eigenschappen *DataSource* instellen** De eigenschappen (*properties*) van het *DataSource*-object moet ingesteld worden. Dit kan onder andere het volgende zijn: de naam van de gegevensbank, de naam van de gegevensbankserver, een beschrijving, ... Al deze eigenschappen corresponderen normaal met *get/set*-methodes van de *DataSource*-klasse van de driver.

**DataSource registreren** Tot slot moet het aangemaakte *DataSource*-object geregistreerd worden bij de *naming service* gebruikt door de *application server*. De naam die de systeembeheerder hier opgeeft kan dan in de Java-applicatie gebruikt worden om het *DataSource*-object op te halen.

Na het uitvoeren van de bovenstaande stappen is het *DataSource*-object beschikbaar voor de applicatie. Het opzoeken van dit object en het opvragen van een verbinding met de gegevensbank verloopt nu als volgt:

---

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("jdbc/mijnDB");
Connection conn = ds.getConnection("login", "wachtwoord");
```

---

De interface *Context* en de klasse *InitialContext* maken deel uit van JNDI API en behoren tot het pakket *javax.naming*. De interface *Context* stelt een context van de *naming service* voor (zie §5.7.1). De klasse *InitialContext* stelt de context voor waarin de opdrachten van de *naming service* zoals opzoeken, registreren, ... starten. Deze context wordt bepaald omgevingsvariabelen of configuratiebestanden.

De string "*jdbc/mijnDB*" is de naam waarmee de systeembeheerder het *DataSource*-object registreerde. Met de methode *getConnection* van de interface *javax.sql.DataSource* kan dan een verbinding met de gegevensbron opgevraagd worden. Er zijn twee versies van deze methode: één zonder argumenten en één waarmee de loginnaam en het wachtwoord van de gegevensbankgebruiker meegegeven worden.

```
public Connection getConnection() throws SQLException;
public Connection getConnection(String loginnaam,
 String wachtwoord) throws SQLException;
```

Het voordeel van deze werkwijze op het gebruik van de *DriverManager* is dat er veel eenvoudiger van *datasource*, van JDBC-driver, van gegevensbank, ... veranderd kan worden. Deze veranderingen hebben namelijk geen invloed op de applicatiecode. Enkel de configuratie moet aangepast worden.

Een tweede voordeel is dat op deze manier *connection pooling* en gedistribueerde transacties mogelijk zijn. Producenten van JDBC-drivers hebben keuze uit drie types implementaties van de interface *DataSource*.

**basis** De verbindingen met de gegevensbank ondersteunen geen *connection pooling* en geen gedistribueerde transacties. Deze verbindingen zijn gelijkaardig als die verkregen via een *DriverManager*.

**ondersteunt *connection pooling*** De verbindingen met de gegevensbank maken automatisch gebruik van *connection pooling*.

**ondersteunt gedistribueerde transacties** De verbindingen kunnen gebruik maken van gedistribueerde transacties en meestal maken ze ook gebruik van *connection pooling*.

### 5.7.3 *Connection Pooling*

Het herhaaldelijk openen en sluiten van verbindingen met de gegevensbank kan in grote applicaties een overlast zijn en dus performantieverlies betekenen. Een oplossing hiervoor is *connection pooling*. Dit betekent dat er een aantal verbindingen met de gegevensbank beschikbaar zijn en dat als de applicatie een

verbinding nodig heeft ze één van de beschikbare verbindingen kan gebruiken. Als de opdrachten in de gegevensbank uitgevoerd zijn wordt de verbinding weer ter beschikking gesteld.

Voor de ontwikkelaar van de applicatie verandert er echter nauwelijks iets. Het volgende stukje code illustreert dit.

---

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("jdbc/mijnDB");
try {
 Connection conn = ds.getConnection("login", "wachtwoord");
 try {
 ...
 } catch (...) {
 ...
 } finally {
 if (conn != null)
 conn.close();
 }
} catch (SQLException e) {
 ...
}
```

---

De methode *getConnection* maakt in het geval dat het *DataSource*-object *connection pooling* ondersteund, geen nieuwe verbinding aan met de gegevensbank maar geeft een referentie naar één van de beschikbare verbindingen uit de *pool*. Het oproepen van de methode *close* sluit de verbinding niet af, maar stelt ze weer ter beschikking.

Opdat *connection pooling* mogelijk zou zijn, moet de configuratie zoals beschreven in §5.7.2 uitgebreid worden. Niet alleen moeten de klassenaam van de *data source*, zijn eigenschappen en zijn naam in de *naming service* opgegeven worden, de configuratie moet ook een *ConnectionPoolDataSource*-object beschrijven en registreren. Dit object maakt de verbindingen van de *connection pool* aan. Meer informatie vind je in [Fis99].

#### 5.7.4 Gedistribueerde transacties

Gedistribueerde transacties zijn transacties die data manipuleren in verschillende gegevensbronnen. De acties op die verschillende gegevensbronnen vormen een eenheid die ofwel volledig uitgevoerd moet worden of wel niet uitgevoerd. In de JDBC 2.0 API is het mogelijk om *Connection*-objecten aan te maken die deel zijn van een gedistribueerde transactie. Dit kan enkel als er gebruik gemaakt wordt van een *DataSource*-object.

Applicaties die gebruik maken van gedistribueerde transacties scheiden de code die de acties op de gegevensbank voorstellen van de code die de transactie maakt, beheert en uitvoert. Dit betekent dat in de JDBC-code geen spoor van transacties te vinden zal zijn. Verschillende methodes met JDBC-code zullen samen genomen worden in een transactie.

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

## 5.8. SLIDES JDBC

## 5.8 Slides JDBC

JDBC | Veerle Ongenaee

1

### Overzicht

- Wat is JDBC?
- Basiswerking
  - Statements
  - PreparedStatements
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - DataSource
  - Scrollable en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes

2

### Java Database Connectivity (JDBC) API

- Toegang tot relationele data vanuit Java
- SQL-compatibele relationele gegevensbanken
- Een API om SQL-statements uit te voeren en de verkregen resultaten te verwerken
- Abstractie van productafhankelijke details
  - Veralgemengd meest voorkomende gegevensbankfuncties
  - Applicatie bruikbaar met verschillende gegevensbanken

3

### JDBC drivers - types

4

### JDBC

- JDBC Core API: **java.sql**
  - Gegevensbankverbinding via JDBC-drivers
  - Basis SQL-operaties uitvoeren
  - Bij alle versies van Java
- JDBC Optional Package API: **javax.sql**
  - Voor application servers
    - Datasource ipv driver
  - Beheer en pooling van JDBC-connecties, gedistribueerde transacties, rowsets, ...
  - Standaard bij JSR vanaf versie 1.4

5

### Overzicht

- Wat is JDBC?
- Basiswerking
  - Statements
  - PreparedStatements
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - DataSource
  - Scrollable en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes

6

## 5.8. SLIDES JDBC

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

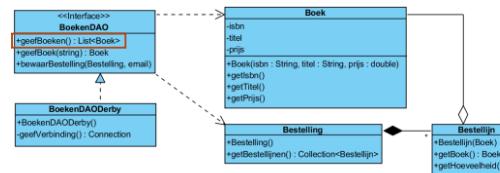
## Voorbeeld

| Tabel BOEKEN       |                            |       |
|--------------------|----------------------------|-------|
| ISBN               | TITEL                      | PRIJS |
| isbn19789043025256 | jQuery - de basis          | 12    |
| isbn2              | aan de slag met HTML5      | 15    |
| isbn3              | Head First Design Patterns | 45    |
| isbn1              | Thinking in Java           | 58    |
| isbn4              | Java Servlet Programming   | 36    |
| isbn5              | Java Server Programming    | 81    |
| isbn6              | The Java Tutorial          | 65    |
| isbn7              | Java Swing                 | 55    |

Industriel Ingenieur Informatica, UGent

7

## Voorbeeld



Industriel Ingenieur Informatica, UGent

8

## Verschillende stappen

- Configuratiegegevens inlezen
- Verbinding aanmaken
- Statement aanmaken
  - “gewoon”
  - PreparedStatement (parameters)
  - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



9

## Configuratie

```

public class BoekenDAO Derby implements BoekenDAO {
 private final ResourceBundle sqlOpdrachten;
 private final ResourceBundle databaseConfig;

 public BoekenDAO Derby() throws ClassNotFoundException {
 sqlOpdrachten = ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.sql");
 databaseConfig = ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.database");
 // enkel voor oudere versies van Java (voor JDK 1.6)
 // Class.forName(databaseConfig.getString("DRIVER"));
 }
}

```

Industriel Ingenieur Informatica, UGent

10

9

10

## Configuratie

- .properties-bestand

```

boek opvragen
Q_BOEKEN = select * from boeken
commentaar
BOEK_ISBN = isbn
sleutel-waarde paren
TITEL = titel
PRIJS = prijs

```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



11

## Configuratie

- Bestand gebruiken in programma

```

ResourceBundle databaseConfig =
ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.database");
+ Naam klasse of naam bestand
+ PropertyResourceBundle
String klasseDriver = databaseConfig.getString("DRIVER")

```

```

info_database
DRIVER = org.apache.derby.jdbc.ClientDriver

```

Industriel Ingenieur Informatica, UGent

12

11

12

**HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)****5.8. SLIDES JDBC**

### Configuratie - Spring

opdrachten.properties (in map resources)

```
boek opvragen
Q_BOEKEN = select * from boeken
BOEK_ISBN = isbn
```

Klasse die properties gebruikt

```
@PropertySource("classpath:opdrachten.properties")
public class BoekenDAO implements BoekenDAO {
 @Value("${Q_BOEKEN}")
 String haalBoeken;
 ...
}
```

In deze klasse gebruik maken van een properties-bestand  
Locatie properties-bestand  
Sleutel in properties-bestand  
Waarde uit properties-bestand injecteren bij aanmaak object

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 13

13

### Driverklasse toevoegen aan CLASSPATH

- De driverklasse behoort niet tot de standaard Java API
- Meegeleverd met de gegevensbank
  - jar-bestand
- Toevoegen aan CLASSPATH
  - Als omgevingsvariabele
  - Als optie bij het java-commando
  - IntelliJ: External Libraries
  - Als dependency in pom.xml
- CLASSPATH
  - Waar zoeken naar klassen?
  - Standaard: huidige map

```
<dependency>
<groupId>org.postgresql</groupId>
<artifactId>postgresql</artifactId>
<scope>runtime</scope>
</dependency>
```

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 15

15

### Verschillende stappen

- Configuratiegegevens inlezen
- Verbinding aanmaken
- Statement aanmaken
  - "gewoon"
  - PreparedStatement (parameters)
  - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 16

16

### Verbinding aanmaken

- DriverManager
  - Toegang tot verschillende beschikbare drivers

```
private Connection geefVerbinding() throws SQLException {
 return DriverManager.getConnection(
 databaseConfig.getString("URL"),
 databaseConfig.getString("LOGIN"),
 databaseConfig.getString("PASWORD"));
}
```

database.properties

```
URL = jdbc:derby://localhost:1527/boekenwinkel
LOGIN = iii
PASWORD = iipwd
```

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 17

17

### JDBC URL

- Vorm: jdbc:<subprotocol>:<subname>
- Subprotocol: identificeert gegevensbankdriver
  - MySql: mysql
  - Derby: derby
  - PostgreSQL: postgresql
  - ...
- Subname: Afhankelijk van driver
  - MySql, Derby, PostgreSQL:
    - host:poort/database

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 18

18

### Verschillende stappen

- Configuratiegegevens inlezen
- Verbinding aanmaken
- Statement aanmaken
  - "gewoon"
  - PreparedStatement (parameters)
  - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 19

19

## 5.8. SLIDES JDBC

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

### Opdracht aanmaken

```
public List<Boek> geefBoeken() throws BoekenNietBeschikbaarException {
 ArrayList<Boek> boeken = new ArrayList<>();

 try (Connection conn = geefVerbinding();
 Statement stmt = conn.createStatement()) {
 // opdracht uitvoeren
 } catch (SQLException e) {
 throw new BoekenNietBeschikbaarException(e);
 }
 return boeken;
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



20

### Voorbeeld

- Voorbeeld uitvoer?

```
try (Connection conn = ...;
 Statement stmt = conn.createStatement()) {
 System.out.println(conn.getClass());
 System.out.println(stmt.getClass());
} catch (SQLException e) {
 Logger.getLogger(KlassenDB.class.getName())
 .log(Level.SEVERE, null,
 "probleem met ophalen info uit databank: " + e.getMessage());
}
```

```
class org.postgresql.jdbc.PgConnection
class org.postgresql.jdbc.PgStatement
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

21

20

### Select-opdracht uitvoeren

```
try (Connection conn = geefVerbinding();
 Statement stmt = conn.createStatement()) {
 // opdracht uitvoeren
 ResultSet rs = stmt.executeQuery(sqlOpdrachten.getString("Q_BOEKEN"));
 while (rs.next()) {
 boeken.add(new Boek(
 rs.getString(sqlOpdrachten.getString("BOEK_ISBN")),
 rs.getString(sqlOpdrachten.getString("TITEL")),
 rs.getDouble(sqlOpdrachten.getString("PRIJS"))));
 }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



22

### Resultaten ophalen

- Interface ResultSet

- Resultaat één keer van boven naar beneden overlopen
- Kolommen tellen vanaf 1 !!!!
- Resultaten worden mogelijk niet allemaal tegelijk opgehaald
- Conversie SQL-type naar Java-type

```
public boolean next() throws SQLException;
public xxx getXXX(int columnIndex) throws SQLException
public xxx getXXX(String columnName) throws SQLException
```



Industriel Ingenieur Informatica, UGent

23

22

### JDBC Types

- Java ↔ SQL
  - String ↔ CHAR, VARCHAR
  - double ↔ DOUBLE
  - int ↔ INTEGER
  - float ↔ REAL
  - ...

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



24

### Andere opdrachten uitvoeren

- delete, update, insert, DDL (Data Definition Language)

```
try (Connection conn = geefVerbinding();
 Statement stmt = conn.createStatement()) {
 // opdracht uitvoeren
 String opdrn = "create table temp (nr numeric)";
 stmt.executeUpdate(opdrn);

 String insertSql = "insert into temp values (1)";
 String updateSql = "update temp set nr=2 where nr=1";
 stmt.executeUpdate(insertSql);
 int updateCnt = stmt.executeUpdate(updateSql);
}
```



Industriel Ingenieur Informatica, UGent

25

24

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

## 5.8. SLIDES JDBC

## Overzicht Statement

```
try {
 Connection conn = DriverManager.getConnection(..., ...);
 Statement stmt = conn.createStatement();
}

// delete-, insert-, update-opdracht of DDL
int aantalJenaAangepast = stmt.executeUpdate(...);

// select-opdracht
ResultSet rs = stmt.executeQuery(...);
while (rs.next()) {
 ... = rs.getXXX(kolomNaam);
 ... = rs.getXXX(kolomNummer);
}
```

## Verschillende stappen

- Configuratiegegevens inlezen
- Verbinding aanmaken
- Statement aanmaken
  - "gewoon"
  - PreparedStatement (parameters)
  - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen

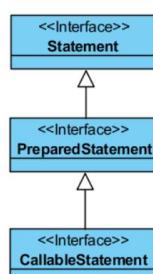
## Overzicht

- Wat is JDBC?
- Basiswerking
  - Statements
    - PreparedStatements
    - CallableStatements
    - Transacties
- JDBC uitgebreid
  - DataSource
  - Scrollbare en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes

## Prepared Statements

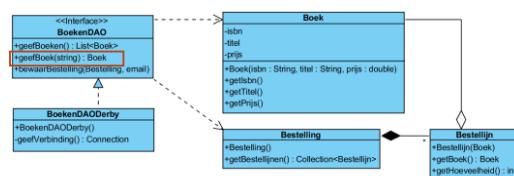
- SQL-opdrachten met parameters
- Voorgecompileerd in gegevensbank
- Opdrachten die verschillende keren uitgevoerd moeten worden
- Reductie uitvoertijd
- Veiligheid
  - Gegevensconversie
  - Voorkomen SQL-injectie

## Prepared Statements



- Werkwijze
  - SQL-opdracht: parameters = ?
  - PreparedStatement aanmaken
  - Parameters invullen
  - Uitvoeren, eventueel resultaat ophalen

## Voorbeeld



## 5.8. SLIDES JDBC

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

## Voorbeeld

| Tabel BOEKEN       |                            |       |
|--------------------|----------------------------|-------|
| ISBN               | TITEL                      | PRIJS |
| isbn19789043025256 | jQuery - de basis          | 12    |
| isbn2              | aan de slag met HTML5      | 15    |
| isbn3              | Head First Design Patterns | 45    |
| isbn1              | Thinking in Java           | 58    |
| isbn4              | Java Servlet Programming   | 36    |
| isbn5              | Java Server Programming    | 81    |
| isbn6              | The Java Tutorial          | 65    |
| isbn7              | Java Swing                 | 55    |

32

## Voorbeeld PreparedStatement

```
public Boek geefBoek(String isbn) throws BoekenNietBeschikbaarException {
 Boek boek = null;

 try (Connection conn = geefVerbinding();
 PreparedStatement stmt = conn.prepareStatement(sqlOpdrachten.getString("Q_BOEK")) {
 stmt.setString(1, isbn);
 ResultSet rs = stmt.executeQuery();
 if (rs.next()) {
 boek = new Boek(
 rs.getString(sqlOpdrachten.getString("BOEK_ISBN")),
 rs.getString(sqlOpdrachten.getString("TITEL")),
 rs.getDouble(sqlOpdrachten.getString("PRIJS")));
 }
 } catch (SQLException e) { ... }
 return boek;
}
```

Industriel Ingenieur Informatica, UGent

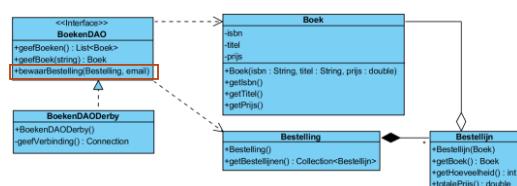
33

## Voorbeeld PreparedStatement

```
sql.properties
boek opvragen aan de hand van isbnnummer
Q_BOEK = select * from boeken where isbn = ?
kolomnamen
BOEK_ISBN = isbn
TITEL = titel
PRIJS = prijs
```

34

## Voorbeeld



35

## Voorbeeld PreparedStatement

| Tabel BESTELLINGEN |                  |          |
|--------------------|------------------|----------|
| AANTAL             | KLANT            | ISBNBOEK |
| 4                  | veerle@hogent.be | isbn2    |
| 1                  | veerle@hogent.be | isbn3    |
| 1                  | test@test.be     | isbn2    |
| 1                  | vo@gent.be       | isbn5    |
| 2                  | vo@gent.be       | isbn3    |
| 2                  | veerle@tiwi.be   | isbn4    |
| 1                  | veerle@tiwi.be   | isbn7    |

36

## SQL-opdrachten

```

Q_BESTELLING = select aantal from bestellingen where klant = ? and isbnboek = ?
AANTAL = aantal
I_BESTELLING = insert into bestellingen (klant, isbnboek, aantal) values (?, ?, ?)
U_BESTELLING = update bestellingen set aantal = ? where klant = ? and isbnboek = ?

```

Industriel Ingenieur Informatica, UGent

37

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

### 5.8. SLIDES JDBC

```
public void bewaarBestelling(Bestelling bestelling, String email) throws BestellingMisluktException {
 try (Connection conn = geefVerbinding();
 PreparedStatement bestaatBestel = conn.prepareStatement(sqlOpdrachten.getString("Q_BESTELLING")));
 {
 PreparedStatement voegBestelToe = conn.prepareStatement(sqlOpdrachten.getString("I_BESTELLING"));
 PreparedStatement pasBestelAan = conn.prepareStatement(sqlOpdrachten.getString("U_BESTELLING"));

 bestaatBestel.setString(1, email);
 voegBestelToe.setString(1, email);
 pasBestelAan.setString(2, email);
 for (Bestellijn lijn : bestelling.getBestellijnen()) {
 bestaatBestel.setString(2, lijn.getBoek().getisbn());
 ResultSet bestaandeBestelling = bestaatBestel.executeQuery();
 if (bestaandeBestelling.next()) {
 bestelling.setAantal(bestelling.getAantal() + bestaandeBestelling.getInt("AANTAL"));
 pasBestelAan.setString(3, lijn.getBoek().getisbn());
 pasBestelAan.setInt(1, aantal + lijn.gethoeveelheid());
 pasBestelAan.executeUpdate();
 } else {
 voegBestelToe.setString(2, lijn.getBoek().getisbn());
 voegBestelToe.setInt(3, lijn.gethoeveelheid());
 voegBestelToe.executeUpdate();
 }
 }
 } catch (Exception e) {
 System.out.println(e.getMessage());
 throw new BestellingMisluktException(e);
 }
}
```

38

### Parameter

- SQL-opdracht met parameter uitvoeren
- Herhaaldelijk uitvoer van dezelfde opdracht met andere waarden
- Gegevensconversie
- Voorkomen SQL-injectie



Industriel Ingenieur Informatica, UGent



39

39

### Gegevensconversie

- Aanhalingsteken in naam

```
String naam = "D'Haese";
```

- SQL-opdracht met knippen en plakken

```
String opdracht = "select * from studenten where naam = \'" + naam + "\'";
```

- SQL-opdracht met parameter

```
String opdracht = "select * from studenten where naam = ?";
```



Industriel Ingenieur Informatica, UGent

40



40

### Gegevensconversie

- Aanhalingsteken in naam

- Knippen en plakken geeft fout

- Gebruik Parameter

- Opdracht wordt gecompileerd met parameter
- Bij uitvoeren
  - Parameters doorsturen
  - Parameters worden geconverteerd naar juiste type
  - Parameters zijn argumenten voor gecompileerde functie



Industriel Ingenieur Informatica, UGent



41

41

### SQL-injectie

- Meestal in een webapplicatie

- Gebruiker

- Parameter ingeven
- Combineert parameter met SQL-opdracht
  - Injecteert SQL-opdrachten
    - Niet-toegankelijke gegevens bekijken
    - Gegevens verwijderen
    - ...



Industriel Ingenieur Informatica, UGent

42



42

### SQL-injectie: voorbeeld

- Ingave gebruiker: studentnummer

- SQL-opdracht

```
String opdracht = "select * from studenten where studnr = " + nummer;
```

- Malafide ingave

- 200200234 OR 1=1
- 200200234; DROP TABLE studenten
- 0 UNION SELECT username, password FROM users



Industriel Ingenieur Informatica, UGent



43

43

## 5.8. SLIDES JDBC

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

## SQL-injectie voorkomen → gebruik parameters!

- Opdracht wordt gecompileerd  

```
select * from studenten where studnr = ?
```
- Instellen parameter converteert invoer naar een VARCHAR
- Deze VARCHAR is een parameter voor de voorgecompileerde SQL-opdracht

## Overzicht

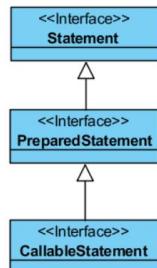
- Wat is JDBC?
- Basiswerkings
  - Statements
  - PreparedStatements
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - DataSource
  - Scrollable en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes

44

45

## Prepared Statements

- Uitvoeren stored procedure
- Stored procedure
  - Functie of procedure
  - In gegevensbank
  - Geen standaardsyntax
  - Niet door alle systemen ondersteund
- JDBC API
  - Uniforme manier



## Callable Statement

- Nodig
  - Naam procedure (bv. SELECT\_BOEKEN)
  - Parameters
- Aanmaken CallableStatement
 

```
String opdracht = "{call SELECT_BOEKEN}";
CallableStatement cstmt = conn.prepareCall(opdracht);
```
- Escape-syntaxis
  - Geen standaard SQL
  - Driver moet interpreteren of omzetten
- Verwijzing naar procedure in de databank

46

47

## Voorbeeld CallableStatement

```

String opdracht = "{call BEWAAR_BESTEL(?, ?, ?)}";
try (Connection conn = geefVerbinding();
 CallableStatement cstmt = conn.prepareCall(opdracht)) {
 // parameters instellen
 cstmt.setString(1, ...);
 cstmt.setString(2, ...);
 cstmt.setInt(3, ...);
 cstmt.executeUpdate();
}

```

## Uitvoerparameters

- Corresponderen met uitvoerparameters van de stored procedure
- Methode
  - ? in de SQL-opdracht
  - Registreren
- Voorbeeld Opdracht
  - ?: aantal boeken

```
String opdracht = "? = call SELECT_BOEKEN";
String opdracht = "{call SELECT_BOEKEN(?)}";
```

48

49

## Registreren uitvoerparameter

- Opmerking
  - Eerst ResultSet overlopen
  - ?: bepaalt NIET het type parameter (in- of uit-)

```
CallableStatement cstmt = conn.prepareCall(opdracht);
cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
ResultSet rs = cstmt.executeQuery();
... // boeken ophalen
int aantalBoeken = cstmt.getInt(1);
```



## Overzicht

- Wat is JDBC?
- Basiswerkings
  - Statements
  - PreparedStatements
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - DataSource
  - Scrollable en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes



## Transacties

- Verschillende rijen en/of tabellen tegelijkertijd aanpassen
- Consistente gegevens
- Data integrity
  - Andere gebruikers zien pas aangepaste waarden als ze "definitief" zijn



## Transacties

- Principe
  - Auto-commit-mode uitschakelen
  - Opdrachten uitvoeren
    - Geslaagd: commit
    - Mislukt: rollback
  - Auto-commit-mode inschakelen
- Auto-commit-mode
  - Standaard aan



## Transacties

```
Connection conn = ...;
PreparedStatement stmt = ...;
try {
 stmt.setString(1,email);
 conn.setAutoCommit(false);
 for (Bestelling lijn : bestelling.geefBestellijnen()) {
 stmt.setInt(2,lijn.getBoek().getId());
 stmt.setInt(3,lijn.gethoeveelheid());
 stmt.executeUpdate();
 }
 conn.commit();
} catch (SQLException e) {
 conn.rollback();
} finally {
 conn.setAutoCommit(true);
}
```



## Opmerkingen

- Let op: Connection-object lokaal
- Verschillende transacties
  - Verschillende Connection-objecten



## 5.8. SLIDES JDBC

## HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

### Overzicht

- Wat is JDBC?
- Basiswerkings
  - Statements
  - PreparedStatements
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - **DataSource**
  - Scrollbare en aanpasbare ResultSet
  - RowSets
  - SQL3-gegevenstypes

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent



56

### Driver versus DataSource

- **java.sql**
  - Gebruik Driver
- **javax.sql**
  - Gebruik DataSource
- **DataSource**
  - Applicatieservers of frameworks zoals Spring
  - Representeert een DBMS
  - Aangemaakt en ter beschikking gesteld via configuratie
  - Beschikbaar via JNDI (Java Naming and Directory Interface) of Dependency Injection

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

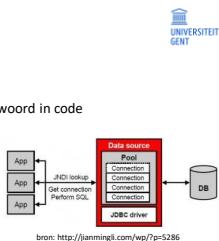
Industrieel Ingenieur Informatica, UGent



57

### DataSource

- Beter alternatief voor JDBC-driver
  - Geen driverklasse, gebruikersnaam en wachtwoord in code
  - Gebruik connection pool
  - Ondersteunt gedistribueerde transacties
- Steunt op ofwel
  - JNDI: Java naming en directory services
    - Naming service: naam ~ object
    - Directory service uitbreiding naming service
    - Object heeft attributen
    - Zoeken op objecten mogelijk
  - Dependency Injection

bron: <http://janningi.com/wp/?p=5286>

58

### Gebruik DataSource

- In applicatieserver bv. Glassfish
- Via framework bv. Spring
- Zorgen voor connection pool

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent



59

### Configuratie DataSource

- In configuratiebestand (Spring: application.properties)
    - JDBC-URL
    - Login
    - Wachtwoord
- ```
spring.datasource.url=jdbc:postgresql://localhost:5432/iii
spring.datasource.username=iii
spring.datasource.password=iiipwd
#spring.datasource.driver-class-name=org.postgresql.Driver
```
- Nodige drivers ter beschikking stellen bv. via pom.xml

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent



60

Gebruik DataSource

- Ophalen via JNDI
 - of Dependency Injection
- ```
private DataSource dataSource;
@.Autowired
public void setDataSource(DataSource dataSource) {
 this.dataSource = dataSource;
}
```
- ```
private DataSource dataSource;
public MijnConstructor(DataSource dataSource) {
    this.dataSource = dataSource;
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent



61

60

126

10

HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)5.8. SLIDES JDBC**JdbcClient in Spring**

- JDBC: veel boilerplate code
- JdbcClient:
 - Vermijdt boilerplate code
 - Ondersteunt meeste JDBC-acties
 - Geen ondersteuning voor Stored Procedures of Batch

Voorbeeld met JdbcClient - configuratie

```
application.properties (in map resources)
spring.datasource.url=jdbc:postgresql://localhost:5432/iii
spring.datasource.username=iii
spring.datasource.password=iipwd
```

pom.xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
```

Voorbeeld met JdbcClient – dependency injection

```
Dependency injection JdbcClient
public class BoekenDAO implements BoekenDAO {
    JdbcClient jdbcClient;

    public BoekenDAO(JdbcClient jdbcClient) {
        this.jdbcClient = jdbcClient;
    }
    ...
}
```

Voorbeeld met JdbcClient – opdracht

```
@Override
public List<Boek> geefBoeken() {
    return jdbcClient.sql(haalBoeken).query(Boek.class).list();
}
```

```
public record Boek(String isbn, String titel, double prijs) {}
```

SQL-opdracht instellen
Tekst met de SQL-opdracht

Zoekopdracht uitvoeren
Type resultaat
Resultaat bewaren in een lijst

Data-klasse (final) – niet aanpasbaar

Voorbeeld met JdbcClient – opdracht met "named" parameters

```
@Value("${BOEK_ISBN}")
String isbn;                                parameter instellen
                                                naam parameter
                                                waarde parameter
                                                resultaat bewaren

@Override
public Boek geefBoek(String isbn) {
    Optional<Boek> boek =
        jdbcClient.sql(zoekBoek).param(pisbn, isbn).query(Boek.class).optional();

    if (boek.isEmpty()) {
        return null;
    } else {
        return boek.get(); # boek oproepen aan de hand van isbnnummer
    }
}                                                 Q_BOEK = select * from boeken where isbn = :isbn
#kolomnamen
BOEK_ISBN = isbn
```

Voorbeeld met JdbcClient – opdracht met parameters (zonder naam)

```
parameter instellen      waarden parameters
Optional<Integer> lijnDb = jdbcClient.sql(zoekBestellingen)
    .params(List.of(email, lijn.boek().isbn()))
    .query(Integer.class).optional();

#bestellingen opvragen
Q_BESTELLING = select aantal from bestellingen where klant = ? and isbnboek = ?

public record Bestellijn(Boek boek, int aantal) {}
public record Boek(String isbn, String titel, double prijs) {}
```

5.8. SLIDES JDBC

HOOFDSTUK 5. JAVA DATABASE CONNECTIVITY (JDBC)

Opdracht met parameters

- Parameters instellen kan op naam of op volgorde
 - Methode **params**
 - Meerdere parameters instellen
 - Methode **param**
 - Eén parameter instellen
- ```

jdbcClient.sql(nieuweBestelling)
 .params(list.of(email, lijn.boek().isbn(), lijn.aantal()))
 .param("lijn", email)
 .param("lijn", lijn.boek().isbn())
 .param("aantal", lijn.aantal())
 .update();

jdbcClient.sql(updateBestelling)
 .params(list.of(aantal + lijn.aantal(), email, lijn.boek().isbn()))
 .param("aantal + lijn.aantal()")
 .param("email")
 .param("lijn.boek().isbn()")
 .update();

```
- FACULTEIT INGEGEN  
EN ARCHITECTUUR



## Overzicht

- Wat is JDBC?
- Basiswerkings
  - Statements
  - PreparedStatement
  - CallableStatements
  - Transacties
- JDBC uitgebreid
  - DataSource
    - JdbcClient API



68

69

69

## 5.9 ORM versus JDBC

ORM vs JDBC | Veerle Ongenaé

1

### JDBC/ADO.NET ↔ ORM

UNIVERSITEIT GENT

- JDBC/ADO.NET
  - Tijdsintensief
  - Low-level dataatoegang
    - SQL-opdrachten als tekst in Java/C#
- ORM
  - Mapping vastleggen met annotaties
  - Minder performant
  - Minder ontwikkelingstijd

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industrial Ingénieur Informatique, UGent

3



# Hoofdstuk 6

## ADO.NET: een inleiding

### 6.1 Wat is ADO.NET?

ADO.NET ([ADO]) is de technologie van het .NET-platform die het mogelijk maakt om vanuit een .NET-applicatie toegang te verkrijgen tot data. Het is de opvolger van ADO, wat staat voor Activex Data Objects. Met ADO.NET is het mogelijk om data van verschillende types databronnen op te halen, te manipuleren en aan te passen. ADO.NET voorziet niet alleen toegang tot data in gegevensbanken, maar ook tot data in XML-files of XML-streams.

Eén van de principes van ADO.NET is dat het een losse koppeling voorziet tussen de applicatie en de databron. In tegenstelling tot wat gebruikelijk is bij een standaard Client-Server-architectuur is er geen constante verbinding met de gegevensbank en kan een deel van de data lokaal in de applicatie bewaard en gemanipuleerd worden. ADO.NET ondersteunt dus probleemloos het bouwen van applicaties volgens de meerlagenarchitectuur.

De ADO.NET architectuur bestaat uit twee grote pijlers: de *DataSet* en .NET Data Providers. De *DataSet* maakt het mogelijk om gegevens lokaal in een programma te bewaren en te manipuleren, los van een gegevensbron. De .NET Data Providers voorzien in toegang tot databanken buiten de applicatie, rechtstreeks of via OLEDB of ODBC.

De ADO.NET klassen vormen de namespace System.Data in de klassenbibliotheek van het .NET-platform.

#### 6.1.1 DataSet

Een *DataSet* is een verzameling van *DataTable*-objecten en hun onderliggende relaties en beperkingen, zoals primaire sleutels, verwijssleutels, .... Een *DataSet* kan opgevuld worden met gegevens uit een database of een xml-bestand, maar heeft geen informatie over de onderliggende gegevensbron. Het is dus een alleenstaande component met informatie in het geheugen. Een *DataTable* is een collectie rijen van een bepaalde tabel.

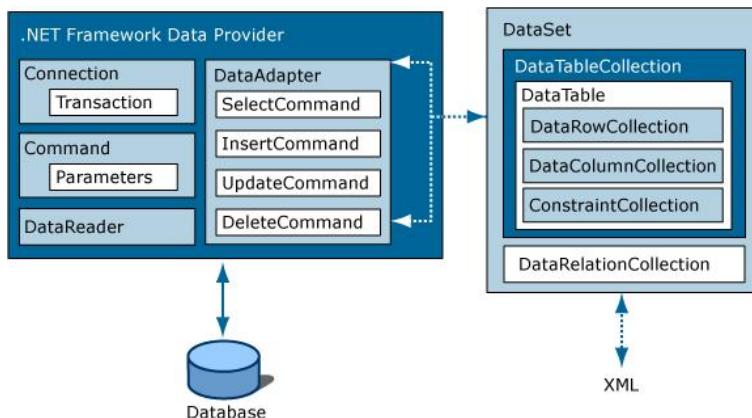
Tussen *DataSets* en XML is er een hechte band. Een *DataSet* kan gegevens inlezen uit een XML-bestand of een XML-stream en kan de data die het bevat uitschrijven in XML-formaat, ook als de gegevens oorspronkelijk uit een gegevensbank kwamen. Het gebruik van *DataSets* wordt uitgebreider besproken in §6.4.1.

### 6.1.2 .NET Data Provider

De toegang tot gegevensbanken in het .NET-platform verloopt via .NET Data Providers. Ze hebben een vergelijkbare rol als JDBC-drivers, namelijk het vertalen van de algemene opdrachten uit ADO.NET API naar gegevensbankspecifieke opdrachten. Ze bieden dus ook de mogelijkheid om een verbinding te maken met de databank, opdrachten uit te voeren in de databank en gegevens op te halen uit de databank. Om dit te verwezenlijken maken ze gebruik van vier sleutelobjecten: *Connection*, *Command*, *DataReader* en *DataAdapter*.

- *Connection* maakt een verbinding met de gegevensbank
- *Command* voert opdrachten uit in de gegevensbank: gegevens ophalen, gegevens veranderen, *stored procedures* uitvoeren en informatie over parameters uitwisselen
- *DataReader* leest gegevens uit een databank met een hoge performantie
- *DataAdapter* is de brug tussen een databank en een *DataSet*-object; hij kan een *DataSet* opvullen met gegevens uit een gegevensbank of deze aanpassen op basis van de gegevens in de *DataSet*; hiervoor gebruikt hij verschillende *Command*-objecten

De relatie tussen een .NET Data Provider en een *DataSet* wordt geïllustreerd in de volgende figuur.



Figuur 6.1: ADO.NET architectuur (bron: <http://msdn.microsoft.com/>)

De namespace *System.Data.Common* bevat de abstracte klassen van de sleutelobjecten waarvan de verschillende providers moeten afleiden, namelijk *DbConnection*, *DbCommand*, *DbDataReader* en *DbDataAdapter*. Deze abstracte klassen voorzien reeds een gedeeltelijke implementatie van de functionaliteit voorzien in de interfaces van de vier kernobjecten *IDbConnection*, *IDbCommand*, *IDataReader* en *IDbDataAdapter* uit de namespace *System.Data*.

Het gebruik van de abstracte klassen in je applicatie laat toe om je programma compatibel te maken met verschillende .Net Data Providers. Applicaties die vaste databaseservers hebben hoeven dit niet te doen. Ze kunnen de concrete klassen van een specifieke provider gebruiken. De namespace horende bij een provider is een subnamespace van *System.Data*.

Het .NET-platform bevat een aantal .NET Data Providers: ‘.NET Framework Data Provider for SQL Server’, ‘.NET Framework Data Provider for OLE DB’, ‘.NET Framework Data Provider for ODBC’, ‘.NET Framework Data Provider for Oracle’, ...

Verschillende producenten van databases voorzien hun eigen provider om op een efficiëntere manier gegevens uit te wisselen met hun gegevensbank. In de rest van dit hoofdstuk zullen we de principes van ADO.NET illustreren aan de hand van de abstracte klassen uit de namespace *System.Data.Common*. Het gebruik van de specifieke klassen van bv. SQL Server .NET Data Provider of de OLE DB .NET Data Provider verloopt analoog.

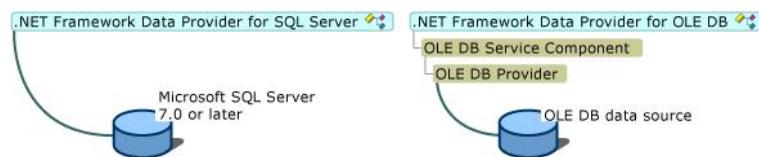
### **.NET Framework Data Provider for SQL Server**

De SQL Server .NET Data Provider is ontworpen voor SQL Server 7.0 en latere versies. Deze provider communiceert direct via het data-transfer-protocol van SQL Server en is daardoor zeer performant en efficient. De klassen van deze provider vind je in de namespace *System.Data.SqlClient*. Voor oudere versies van SQL Server maak je gebruik van de .NET Framework Data Provider for OLE DB.

### **.NET Framework Data Provider for OLE DB**

De .NET Framework Data Provider for OLE DB is gebouwd voor OLE DB API van versie 2.6 (OLE DB=Object Linking and Embedding for Databases). De OLE DB API voegt een extra laag toe tussen de provider en de gegevensbank, die een uniforme toegang tot verschillende gegevensbronnen biedt, maar iets minder efficient is.

Deze provider is iets minder efficient dan de vorige omdat hij via een OLE DB Service Component en een OLE DB Provider communiceert met de database. Het voordeel van deze provider is dat hij een uniforme toegang biedt tot verschillende databronnen.



Figuur 6.2: SQL Server .NET Data Provider versus OLE DB .NET Data Provider (bron: <http://msdn.microsoft.com/>)

Het gebruik van ADO.NET is getest voor de volgende OLE DB drivers: SQLOLEDB (Microsoft OLE DB provider for SQL Server), MSDAORA (Microsoft OLE DB provider for Oracle) en Microsoft.Jet.OLEDB.4.0 (OLE DB provider for Microsoft Jet). Let op! De Microsoft.Jet.OLEDB.4.0 driver, die o.a. toegang geeft tot Access databases, is niet bruikbaar voor multithreaded applicaties zoals een webapplicatie in ASP.NET (zie cursus Webtechnologieën).

De klassen van deze provider vind je in de namespace *System.Data.OleDb*.

### **.NET Framework Data Provider for ODBC**

ODBC staat voor **Open Database Connectivity** en is net zoals JDBC of OLE DB een API om te connecteren met verschillende gegevensbanksystemen. Ook hier zorgt een driver voor de vertaling van de (ODBC-)opdrachten naar specifieke opdrachten van het gegevensbanksysteem. ODBC is het oudere broertje van OLE DB. De ‘.NET Framework Data Provider for ODBC’ maakt gebruik van ODBC om te communiceren met de databank. De bijhorende namespace is *System.Data.Odbc*.

### .NET Framework Data Provider for Oracle

Om met een Oracle gegevensbank te interageren gebruik makend van ADO.NET is de ‘.NET Framework Data Provider for Oracle’ beschikbaar op voorwaarde dat de Oracle client software (versie 8.1.7 en later) geïnstalleerd is. De klassen van deze provider vind je in de namespace *System.Data.OracleClient*.

## 6.2 Een verbinding maken

Het eerste stuk dat we bespreken van ADO.NET is zeer gelijkaardig aan JDBC. Net zoals JDBC biedt ADO.NET de mogelijkheid om

- een verbinding te maken met een gegevensbank
- SQL-opdrachten uit te voeren in een gegevensbank
- het resultaat van een zoekopdracht binnen te halen in een programma
- SQL-opdrachten met parameters uit te voeren
- ‘stored procedures’ uit te voeren
- SQL-opdrachten te groeperen in een transactie

Daarnaast kent ADO.NET ook het gebruik van datasets en data-adapters. In de volgende tabel vind je de interfaces van JDBC en de corresponderende abstracte klassen van ADO.NET die een gelijkaardige functie hebben als de interfaces in JDBC.

| ADO.NET      | JDBC       |
|--------------|------------|
| DbConnection | Connection |
| DbCommand    | Statement  |
| DbDataReader | ResultSet  |

Om een verbinding te maken met een gegevensbank maak je gebruik van een *DbConnection*-object. Het eigenlijke type van het *Connection*-object is een afgeleide klasse van *DbConnection* en verschilt van provider tot provider. In het configuratiebestand van de applicatie (*web.config* voor een webapplicatie en *App.config* voor andere applicaties) wordt de provider en de connectiestring gespecificeerd. Een factory, horende bij de provider, levert dan het *Connection*-object. De volgende stukken code illustreren hoe een verbinding met een gegevensbank verkregen kan worden.

---

```
<configuration>
 ...
 <connectionStrings>
 <add name="bestellingDB"
 providerName="System.Data.OleDb"
 connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
 User ID=Admin;
 Data Source=C:\\...\\vbAdo\\klantorders.mdb"/>
 </connectionStrings>
</configuration>
```

---

Elk kindelement *add* van het element *connectionStrings* bepaalt een combinatie bestaande uit een provider (attribuut *providerName*) en een connectiestring (attribuut *connectionString*). Deze combinatie krijgt een naam a.d.h.v. het attribuut *name*.

---

```

ConnectionStringSettings connStringSet
 = ConfigurationManager.ConnectionStrings["bestellingDB"];
DbProviderFactory factory
 = DbProviderFactories.GetFactory(connStringSet.ProviderName);

DbConnection connection = factory.CreateConnection();
connection.ConnectionString = connStringSet.ConnectionString;

```

---

Een *ConfigurationManager* leest de combinatie provider-connectiestring uit het configuratiebestand. Hier voor gebruik je de eigenschap *ConnectionStrings* met als index de waarde van het attribuut *name* uit het configuratiebestand. De combinatie provider-connectiestring wordt gekenmerkt door het type *ConnectionStringSettings*. Dit type heeft twee eigenschappen:

**ProviderName** een string die de provider specificeert; vaak is dit de namespace waartoe de providerklassen behoren

**ConnectionString** de connectiestring

Op basis van de providernaam wordt een factory geselecteerd. Deze factory maakt alle objecten die je nodig hebt om te communiceren met de gegevensbank aan. Dankzij de factory hoef je de specifieke types van deze objecten, die verschillen van provider tot provider, niet kennen.

De connectiestring initieert een aantal eigenschappen van de verbinding met de gegevensbank en bestaat uit naam/waarde-paren gescheiden door een punt-komma's. De eigenschap *ConnectionString* van het *Connection*-object kan enkel ingesteld worden als de verbinding met de gegevensbank niet geopend is. De volgende tabel geeft een overzicht van een aantal in te stellen eigenschappen. In het geval van een *OleDbConnection* is de *Provider*-eigenschap vereist. De syntax en de eigenschappen kunnen variëren van provider tot provider.

| Eigenschap                                              | Stand.       | Omschrijving                                                                                                                                                                                                                              |
|---------------------------------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Provider</i>                                         |              | De driver die gebruikt wordt om een verbinding te maken met de gegevensbank. Dit attribuut is niet nodig voor de SQL Server .NET Data Provider omdat die geen tussenliggende laag gebruikt.                                               |
| <i>Connect Timeout</i> of <i>Connection Timeout</i>     | 15           | Het aantal seconden dat er gewacht wordt op een verbinding met de server. Daarna wordt de poging gestaakt en wordt een fout gegenereerd.                                                                                                  |
| <i>Data Source of Server</i>                            |              | De naam of het adres van de gegevensbankserver of gegevensbron.                                                                                                                                                                           |
| <i>Initial Catalog of Database</i>                      |              | De naam van de gegevensbank.                                                                                                                                                                                                              |
| <i>Integrated Security</i> of <i>Trusted_Connection</i> | <i>false</i> | Indien de waarde van deze eigenschap <i>false</i> is moeten de eigenschappen <i>User Id</i> en <i>Password</i> gespecificeerd worden. De waarde <i>true</i> impliceert dat de huidige Windowsgebruiker gebruikt wordt voor authenticatie. |
| <i>Password of Pwd</i>                                  |              | Paswoord om in te loggen op de gegevensbank.                                                                                                                                                                                              |
| <i>User ID</i>                                          |              | De gebruiker waarmee ingelogd wordt op de gegevensbank.                                                                                                                                                                                   |

Een alternatief voor de factory is het gebruik van de expliciete providerklassen. Bijvoorbeeld:

---

```

using System.Data.SqlClient;
...
String connString
 = "server=(local)\\NetSDK;database=pubs;Trusted_Connection=yes";
SqlConnection conn = new SqlConnection(connString);

```

---

---

```
using System.Data.OleDb;
...
String connString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
 "User ID=Admin;" +
 "Data Source=C:\\vongenae\\gegevensbanken\\vbn.mdb";
OleDbConnection conn = new OleDbConnection(connString);
```

---

of

---

```
using System.Data.SqlClient;
...
String connString
 = "server=(local)\\NetSDK;database=pubs;Trusted_Connection=yes";
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connString;
```

---



---

```
using System.Data.OleDb;
...
String connString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
 "User ID=Admin;" +
 "Data Source=C:\\vongenae\\gegevensbanken\\vbn.mdb";
OleDbConnection conn = new OleDbConnection();
conn.ConnectionString = connString;
```

---

De methoden *Open* en *Close* zorgen voor het openen en sluiten van de verbinding met de gegevensbank. Het is aangewezen om na gebruik steeds de verbinding te sluiten. Dit kan je realiseren met een try/catch/finally-blok of met de *using*-opdracht van C#.

---

```
conn.Open();
try {
 ... // gegevens uitwisselen met de gegevensbank
} catch (Exception e) {
 ... // foutafhandeling
} finally {
 conn.Close();
}

using System.Data.SqlClient;
...
using (SqlConnection conn = new SqlConnection(connString)) {
 conn.Open();
 ... // gegevens uitwisselen met de gegevensbank
}
```

---

## 6.3 Opdrachten uitvoeren

### 6.3.1 Het commando-object

Om opdrachten te kunnen uitvoeren op de gegevensbank moet men na het aanmaken van een verbinding (zie §6.2) een commando-object aanmaken. Het type van dit object is afgeleid van de abstracte klasse *DbCommand* en is afhankelijk van de gebruikte provider.

Een commando-object kan aangemaakt worden met behulp van een connectie-object. Verder is er ook een *string* die een SQL-opdracht voorstelt nodig. Deze opdracht zal gecompileerd en uitgevoerd worden in de gegevensbank. De volgende code illustreert het aanmaken van een commando-object aan de hand van een connectie-object en een string die een SQL-query voorstelt. De SQL-opdracht wordt opgegeven in het configuratiebestand.

---

```
<configuration>
 ...

```

```

<appSettings>
 <add key="SELECT_BESTELLINGEN"
 value="select * from Bestellingen"/>
 ...
</appSettings>
...
</configuration>

DbConnection conn = ... ;
// commando-object aanmaken
DbCommand command = conn.CreateCommand();
command.CommandText = ConfigurationManager.AppSettings["SELECT_BESTELLINGEN"];

```

Met de methode *CreateCommand* van het connectie-object wordt het commando-object aangemaakt. De SQL-opdracht wordt ingesteld met de eigenschap *CommandText*.

Het is ook mogelijk om de specifieke providerklassen te gebruiken.

```

using System.Data.SqlClient;
...
SqlConnection conn = ...
string query = "select voornaam, achternaam from personen";
SqlCommand bepPers = new SqlCommand(query, conn);

```

De *IDbCommand*-interface bevat een aantal methodes om opdrachten uit te voeren, o.a. *ExecuteReader* en *ExecuteNonQuery*. De methode *ExecuteReader* dient om zoekopdrachten uit te voeren (zie §6.3.2) en met de methode *ExecuteNonQuery* kunnen gegevens toegevoegd, gewijzigd of verwijderd worden (zie §6.3.3).

Om SQL-opdrachten met parameters aan te maken wordt de methode *CreateParameter* gebruikt (zie §6.3.4).

### 6.3.2 Zoekopdrachten

Om het resultaat van zoekopdrachten op te halen kan je gebruik maken van een *DataReader*. In het geval van de SQL Server .Net Data Provider, respectievelijk de OLE DB .Net Data Provider, zijn dat objecten van het type *SqlDataReader* en *OleDbDataReader*. Met een *DataReader* kan je de gegevensstroom van de databank éénmaal lezen van het begin naar het einde. De leesoperatie haalt de informatie rij per rij op.

Een *DataReader* wordt verkregen als het resultaat van het oproepen van de methode *ExecuteReader* van een commando-object (zie §6.3.1). Met de methode *Read* kan je een rij van het resultaat van de uitgevoerde zoekopdracht inlezen. Deze methode verplaatst de positie van de *DataReader* naar de volgende rij van het resultaat. Voor de eerste oproep van de methode *Read* is de positie van de *DataReader* voor de eerste rij van het verkregen resultaat. Het resultaat van de methode *Read* is een logische grootheid (*bool*) die aangeeft of er nog rijen in het verkregen resultaat zijn of dat de positie van de *DataReader* na de laatste rij is.

De interface *IDataReader*, en dus ook *DbDataReader*, heeft een indexer die zowel een string als een geheel getal als argument aanvaardt. Je kan deze indexer gebruiken om de gegevens van de verschillende kolommen van de huidige rij in het verkregen resultaat op te halen. Als argument van de indexer gebruik je dan de naam van de kolom of het volgnummer van de kolom (0, 1, 2, ...). Een alternatief zijn de methodes *GetXxx*. Hierbij stelt *Xxx* het type van de kolomgegevens voor. Deze methodes hebben het volgnummer van de kolom als argument. De laatste methode om gegevens uit één rij op te halen is performanter onder andere omdat er dan minder conversies gebeuren.

De volgende voorbeelden illustreren hoe de gegevens van een zoekopdracht op het scherm getoond kunnen worden.

```

// verbinding met de gegevensbank
DbConnection conn = ... ;

```

---

```
// commando-object aanmaken
DbCommand command = conn.CreateCommand();
command.CommandText = ConfigurationManager.AppSettings["SELECT_BESTELLINGEN"];
// zoekopdracht uitvoeren
conn.Open();
try {
 DbDataReader reader = command.ExecuteReader();
 while (reader.Read()) {
 Console.WriteLine("BestelID: " + reader["BestelID"]);
 Console.WriteLine("Hoeveelheid: " + reader[1]);
 Console.WriteLine("Firma: " + reader.GetString(2));
 }
} catch (Exception e) {
 ... // foutafhandeling
 Console.WriteLine(e.StackTrace);
} finally {
 conn.Close();
}
```

---

Het gebruik van de *DataReader* kan de performantie van de applicatie verhogen omdat hij data ophaalt van zodra ze beschikbaar is en omdat er slechts één rij tegelijk in het geheugen bewaard wordt.

Na het ophalen van het resultaat van een zoekopdracht moet de *DataReader* afgesloten worden met de methode *Close* om andere opdrachten te kunnen uitvoeren over dezelfde verbinding met de databank. Het afsluiten van de verbinding sluit ook de *DataReader* af.

### 6.3.3 Gegevens wijzigen, toevoegen of verwijderen

Om gegevens te wijzigen, toe te voegen of te verwijderen wordt opnieuw een commando-object (zie §6.3.1) aangemaakt. Het enige verschil met het voorbeeld in §6.3.1 is dat de eigenschap *CommandText* nu een UPDATE, INSERT of DELETE SQL-opdracht voorstelt. Vervolgens wordt de methode *ExecuteNonQuery* van de interface *IDbCommand* opgeroepen. Het resultaat van deze methode is het aantal aangepaste rijen.

Het volgende voorbeeld illustreert de uitvoering van een INSERT SQL-opdracht.

---

```
// verbinding met de gegevensbank
DbConnection conn = ... ;
// commando-object aanmaken
DbCommand command = conn.CreateCommand();
command.CommandText
 = "insert into personen(achternaam,voornaam) values ('De Ridder','Jan')";
// opdracht uitvoeren
conn.Open();
try {
 command.ExecuteNonQuery();
} catch (Exception e) {
 ... // foutafhandeling
 Console.WriteLine(e.StackTrace);
} finally {
 conn.Close();
}
```

---

Op een analoge manier is het mogelijk om DDL-opdrachten (Data Definition Language) of stored procedures uit te voeren.

### 6.3.4 Parameters

Parameters zijn handig om herhaaldelijk eenzelfde SQL-opdracht uit te voeren met andere waarden. Het commando-object (zie §6.3.1) heeft een eigenschap *Parameters* die de parameters bijhoudt. De eigenschap *Parameters* is van het type *IDataParameterCollection* (respectievelijk *SqlParameterCollection* of

*OleDbParameterCollection*). De parameterklasse is afgeleid van *DbParameter* die de *IDbDataParameter*-interface implementeert. De klassen die parameters voor stellen bij de SQL Server .Net Data Provider en de OLE DB .Net Data Provider zijn respectievelijk *SqlParameter* en *OleDbParameter*.

Om een parameter aan te maken moeten er twee dingen gebeuren: de plaats van de parameter aanduiden in de string die de SQL-opdracht voorstelt en de parameter registreren bij het commando-object.

De manier waarop de plaats van de parameter in de SQL-opdracht wordt aangegeven hangt af van provider tot provider. Bij de .NET Framework Data Provider for SQL Server wordt een parameter gekenmerkt door een string van de vorm *@parameter* waarbij *parameter* de naam van de parameter voorstelt. Het volgend voorbeeld toont een INSERT SQL-opdracht waarbij twee parameters gebruikt worden, namelijk *@naam* en *@voornaam*.

---

```
string opdracht
= "insert into personen(naam,voornaam) values (@naam,@voornaam);"
```

---

In de volgende tabel vind je een overzicht van hoe de verschillende providers parameters aangeven in een SQL-opdracht.

| provider namespace | parameter syntax  |
|--------------------|-------------------|
| SqlClient          | <i>@parameter</i> |
| OleDb              | ?                 |
| Odbc               | ?                 |
| OracleClient       | <i>:parameter</i> |

Met behulp van een *DbProviderFactory* kan je een parameter aanmaken. Daarna stel je een aantal eigenschappen van de parameter in:

**naam** *ParameterName*

**type** *DbType*

**waarde** *Value*

In het volgend voorbeeld worden een parameter aangemaakt van het type *int*.

---

```
DbProviderFactory factory = ... ;
DbCommand opdracht = ... ;
const string MIN = "@min";
int minimumAantal = ... ;
DbParameter parameter = factory.CreateParameter();
parameter.ParameterName = MIN;
parameter.DbType = DbType.Int32;
opdracht.Parameters.Add(parameter);
...
opdracht.Parameters[MIN].Value = minimumAantal;
```

---

Het type van een parameter is afhankelijk van de gebruikte provider. De verschillende types bij de SQL Server .Net Data Provider en de OLE DB .Net Data Provider worden bepaald door de opsommingen *SqlDbType* en *OleDbType*. De opsomming *DbType* biedt ook de mogelijkheid om op een meer generieke wijze types instellen.

De methode *Add* voegt de parameters dan toe aan de parameterverzameling horende bij het commando-object. In tegenstelling tot wat men zou verwachten moeten de parameters in de volgorde van voorkomen in de SQL-opdracht toegevoegd worden aan de parameterverzameling.

De *IDataParameterCollection* heeft een indexer met een *string*-argument die een *DbParameter*-object teruggeeft. Er is ook een indexer van het type *int* beschikbaar.

## 6.4 DataSet en DataAdapter

In §6.3.1 werd besproken hoe opdrachten uitgevoerd kunnen worden op de gegevensbank vanuit een C#-programma. Om de gegevens op te halen uit de gegevensbank werd gebruik gemaakt van een *DataReader*. Met behulp van deze *DataReader* kan het resultaat van een query rij voor rij overlopen en opgehaald worden. In deze sectie bekijken we een alternatieve manier om gegevens op te halen uit de gegevensbank. De opgehaalde gegevens worden in het geheugen bewaard in een *DataSet*-object en kunnen in het geheugen gemanipuleerd worden. Achteraf kunnen de aangebrachte wijzigingen dan teruggestuurd worden naar de gegevensbank.

### 6.4.1 DataSet

Een *DataSet* is een voorstelling van gegevens in het geheugen zoals in een relationele databank. Deze gegevensvoorstelling is niet verbonden met een bestaande gegevensbron en kan de volgende elementen bevatten: gegevens, tabellen, beperkingen (bv. *primary keys*) en relaties tussen tabellen zoals bv. verwijssleutels (*foreign keys*). De gegevens in een *DataSet* kunnen data aangemaakt door de huidige applicatie zijn of data van één of meerdere externe databronnen. De communicatie met de bestaande databanken wordt bepaald door een *DataAdapter*.

Het volgende voorbeeld toont hoe je een *DataSet*-object kan aanmaken. Elk *DataSet*-object heeft een naam, die o.a. gebruikt wordt om dit object om te zetten naar XML. Indien er geen naam gespecificeerd wordt dan krijgt de *DataSet* de standaardnaam *NewDataSet*. Dit is het geval in het eerste voorbeeld. In het tweede voorbeeld is de naam van de *DataSet* wel opgegeven en is die *CustomerOrders*.

```
DataSet gegevens = new DataSet();
DataSet custDS = new DataSet("CustomerOrders");
```

Een *DataSet* kan één of meerdere *DataTable*-objecten bevatten. Bovendien kan je op bepaalde kolommen van deze *DataTable*-objecten beperkingen plaatsen zoals *primary keys* en *unique*. In het volgende voorbeeld wordt een *DataSet* aangemaakt met één *DataTable*-object. De tabel heeft drie kolommen *OrderID*, *OrderQuantity* en *CompanyName*. De kolom *OrderID* is de primaire sleutel van de tabel.

---

```
DataSet custDS = new DataSet("CustomerOrders");

DataTable ordersTable = custDS.Tables.Add("Orders");

 DataColumn pkCol = ordersTable.Columns.Add("OrderID",
 typeof(Int32));
ordersTable.Columns.Add("OrderQuantity", typeof(Int32));
ordersTable.Columns.Add("CompanyName", typeof(string));

ordersTable.PrimaryKey = new DataColumn[] {pkCol};
```

---

Uit de bovenstaande code leren we dat een *DataSet* de eigenschap *Tables* heeft van het type *DataTableCollection*. De methode *Add* van *DataTableCollection* voegt een tabel toe aan de *DataSet*. Het argument van deze methode is de naam van de tabel, het resultaat is een verwijzing naar de aangemaakte tabel. De klasse *DataTableCollection* heeft ook een indexer met als argument een *string* of een *int*. Het resultaat van de indexer is een *DataTable*-object met de opgegeven naam of het opgegeven volgnummer.

Een *DataTable* heeft onder andere de eigenschappen *Columns* van het type  *DataColumnCollection* en *PrimaryKey* (een tabel van  *DataColumn*). In het voorbeeld wordt de methode *Add* van  *DataColumnCollection* gebruikt om een kolom toe te voegen aan de tabel. De argumenten van deze methode kunnen onder andere de naam en het type van de kolom zijn. Het resultaat is een verwijzing naar de aangemaakte kolom.

Om het type van een klasse te bepalen kan het sleutelwoord *typeof* gebruikt worden. Een alternatief voor *Type t = typeof(string);* is bijvoorbeeld de volgende uitdrukking.

---

```
using System;
...
string mijnString = "Een String";
Type t = mijnString.GetType();
```

---

Het volgend voorbeeld illustreert hoe een rij toegevoegd kan worden aan de bovenstaande tabel.

---

```
// een rij aanmaken
DataRow ordersRow = ordersTable.NewRow();
ordersRow[0] = 3145;
ordersRow[1] = 3;
ordersRow["CompanyName"] = "Mijn Bedrijf";

// een rij toevoegen aan de tabel
ordersTable.Rows.Add(ordersRow);
```

---

De methode *NewRow* van *DataTable* maakt een nieuwe rij aan op basis van de structuur van het *DataTable*-object. De gegevens van deze rij kunnen ingevuld worden met de indexer van *DataRow*. Deze indexer is van het type *object*. Het argument van de indexer kan onder andere een *string*, een *int*, een  *DataColumn*, ... zijn die de kolom aanduidt.

Om de rij toe te voegen aan de tabel wordt gebruik gemaakt van de methode *Add* van de klasse *DataRowCollection*. Het *DataRowCollection*-object wordt verkregen met de eigenschap *Rows* van *DataTable*.

Met behulp van de methode *Find* van het *DataRowCollection*-object kunnen rijen gevonden worden aan de hand van hun primaire sleutel. In het volgend voorbeeld wordt de hierboven toegevoegde rij geselecteerd.

---

```
DataRow selectRow = ordersTable.Rows.Find(3145);
```

---

## 6.4.2 DataAdapter

Om een *DataSet* op te vullen met gegevens uit een databank wordt gebruik gemaakt van een *DataAdapter*. Een *DataAdapter* vormt de brug tussen de *DataSet* en de onderliggende databank. De *DataAdapter* wordt achteraf ook gebruikt om de wijzigingen door te voeren op de gegevensbronnen. De *DataAdapter*-klasse is afgeleid van *DbDataAdapter* en implementeert dus de interface *IDbDataAdapter*. De klassen die *DataAdapters* voorstellen bij de SQL Server .Net Data Provider en de OLE DB .Net Data Provider zijn respectievelijk *SqlDataAdapter* en *OleDbDataAdapter*.

Een *DataAdapter* kan vier commando-objecten hebben die respectievelijk zorgen voor het ophalen van de gegevens uit de gegevensbank, het toevoegen van gegevens aan de gegevensbank, het wijzigen van gegevens in de gegevensbank en het verwijderen van gegevens uit de gegevensbank. Deze vier commando-objecten worden ingesteld, gewijzigd en geselecteerd met de eigenschappen *SelectCommand*, *InsertCommand*, *UpdateCommand* en *DeleteCommand*. Verder beschikt de *DataAdapter* ook over een connectie-object (zie §6.2) via zijn commando-objecten.

Om een *DataSet* op te vullen wordt de *Fill*-methode van de *DataAdapter* gebruikt. Deze methode heeft twee argumenten: de op te vullen *DataSet* en de tabel die het resultaat van de query, voorgesteld door de eigenschap *SelectCommand*, zal bevatten. Het argument dat naar de tabel verwijst kan een *DataTable*-object zijn of een *string* die de naam van de tabel is. In het onderstaande voorbeeld wordt een *DataSet* opgevuld met een tabel die het resultaat van een SQL-zoekopdracht *query* bevat. De tabel krijgt de naam *Personen*.

---

```
// verbinding met de gegevensbank
DbProviderFactory factory = ...;
```

---

```

DbConnection conn = ... ;
// comando-object aanmaken
DbCommand command = conn.CreateCommand();
command.CommandText
 = ConfigurationManager.AppSettings["SELECT_BESTELLINGEN"];

// DataAdapter aanmaken
DbDataAdapter adapter = factory.CreateDataAdapter();
adapter.SelectCommand = command;

// DataSet aanmaken
DataSet klantDS = new DataSet(DS_ORDERS);

// DataSet opvullen
adapter.Fill(klantDS, TABEL_BESTELLING);

```

---

Als je gebruik maakt van de concrete providerklassen, dan kan je de SQL-zoekopdracht en de connectie-string met de constructor meegeven.

```

string connString = "...";
string query = "select personenID, voornaam, achternaam from personen";

OleDbConnection conn = new OleDbConnection(connString);
OleDbDataAdapter adapter = new OleDbDataAdapter(query, conn);
DataSet ds = new DataSet();
adapter.Fill(ds, "Personen");

```

---

Merk op dat de verbinding met de gegevensbank niet expliciet geopend en gesloten wordt. Dit gebeurt impliciet in de *Fill*-methode. Als de verbinding al geopend was bij het oproepen van de *Fill*-methode dan wordt de verbinding niet geopend en gesloten tijdens het uitvoeren van de methode.

Bij het opvullen van de *DataSet* worden geen primaire sleutels aangemaakt tenzij dit expliciet wordt aangegeven op de volgende manier.

```
adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
```

---

Hierbij is de eerste *MissingSchemaAction* een eigenschap van *DbDataAdapter* en de tweede *MissingSchemaAction* is een opsomming uit de bibliotheek *System.Data*.

Nu we de *DataSet* opgevuld hebben met de tabel *Personen* kunnen we hieraan gegevens toevoegen, wijzigen en verwijderen. Het volgend stukje code geeft een voorbeeld van mogelijke wijzigingen.

```

DataRow rij = ds.Tables["Personen"].NewRow();
rij[2] = "Dierick";
rij[1] = "Dirk";
ds.Tables["Personen"].Rows.Add(rij);

rij = ds.Tables["Personen"].Rows.Find(12);
rij[2] = "Danneels";

rij = ds.Tables["Personen"].Rows.Find(17);
rij.Delete();

```

---

De methode *Delete* van *DataRow* markeert de rij als te verwijderen. Deze actie kan nog ongedaan gemaakt worden door het oproepen van de methode *RejectChanges* van *DataRow*. De bovenstaande code voert wijzigingen uit aan de *DataSet ds*. Om deze wijzigingen ook door te voeren aan de gegevensbank moeten de eigenschappen *InsertCommand*, *UpdateCommand* en *DeleteCommand* van de bijhorende *DataAdapter* ingevuld worden en moet de methode *Update* van die adapter worden opgeroepen. In het volgend stuk code worden deze comando-objecten aangemaakt en toegekend aan de adapter.

```

// comando's aanmaken
// insert
adapter.InsertCommand = new OleDbCommand(insert, conn);
OleDbParameter paramNaam
 = adapter.InsertCommand.Parameters.Add("@achternaam", DbType.String);

```

```

paramNaam.SourceColumn = "achternaam";
OleDbParameter paramVNaam
 = adapter.InsertCommand.Parameters.Add("@voornaam", DbType.String);
paramVNaam.SourceColumn = "voornaam";

// update
adapter.UpdateCommand = new OleDbCommand(update, conn);
paramNaam = adapter.UpdateCommand.Parameters.Add("@achternaam", DbType.String);
paramNaam.SourceColumn = "achternaam";
paramNaam.SourceVersion = DataRowVersion.Current;
OleDbParameter paramID
 = adapter.UpdateCommand.Parameters.Add("@personenID", DbType.Int64);
paramID.SourceColumn = "personenID";
paramID.SourceVersion = DataRowVersion.Original;

// delete
adapter.DeleteCommand = new OleDbCommand(delete, conn);
paramID = adapter.DeleteCommand.Parameters.Add("@personenID", DbType.Int64);
paramID.SourceColumn = "personenID";
paramID.SourceVersion = DataRowVersion.Original;

// wijzigingen doorvoeren op gegevensbank
adapter.Update(ds, "Personen");

```

---

In het bovenstaande voorbeeld wordt een alternatieve methode (anders dan in §6.3.4) gebruikt om een parameter aan te maken. Deze methode kan je enkel gebruiken indien je de provider kent omdat de methode *Add* met meerdere parameters niet bestaat voor een *IDataParameterCollection*, maar wel voor een *OleDbParameterCollection*.

Een ander verschil met §6.3.4 is dat de parameter geen waarde wordt toegekend. Hier wordt met de eigenschap *SourceColumn* de kolom aangeduid die de waarde voor de parameter bevat. Als je een rij verandert dan worden zowel de oorspronkelijk als de huidige inhoud van de rij bijgehouden. Bovendien wordt ook de status (toegevoegd, veranderd, verwijderd, ...) van de rij aangepast. De eigenschap *SourceVersion* van *DbDataParameter* geeft aan welke versie van de kolom van een rij gebruikt moet worden bij het uitvoeren van de opdracht. De opsomming *DataRowVersion* bevat de mogelijke versies van een rij. De mogelijke statussen van een rij worden bepaald door de opsomming *DataRowState*.

Bij het uitvoeren van de methode *Update* wordt nagegaan welke rijen toegevoegd, veranderd of verwijderd zijn. Vervolgens wordt het corresponderende commando voor deze rij uitgevoerd. Dit betekent dat als er een rij toegevoegd is, het *InsertCommand* wordt uitgevoerd, dat als er een rij veranderd is het *UpdateCommand* wordt uitgevoerd en dat als er een rij verwijderd is, het *DeleteCommand* wordt uitgevoerd.

De waarde van de parameters in het commando-object zijn dan de waarden voor de huidige rij. Welke versie van de kolom er gebruikt wordt, hangt af van de eigenschap *SourceVersion* voor die parameter. Als in het bovenstaande voorbeeld voor een bepaalde rij zowel het *personenID* als de *achternaam* veranderd zijn, dan wordt bij het uitvoeren van het *UpdateCommand* de nieuwe *achternaam*, maar het oude *personenID* gebruikt omdat de waarde van de eigenschap *SourceVersion* respectievelijk *DataRowVersion.Current* en *DataRowVersion.Original* zijn.

#### 6.4.3 DataReader versus DataSet

De keuze tussen het gebruik van een *DataReader* en een *DataAdapter* in combinatie met een *DataSet* wordt bepaald door een aantal factoren. Het gebruik van een *DataReader* is performanter omdat er veel minder gegevens in het geheugen bewaard worden. Anderszijds is een *DataSet* handig in combinatie met ASP.NET-pagina's (zie cursus Webtechnologieën) en biedt het de mogelijkheid om data te manipuleren los van de gegevensbank en pas later wijzigingen toe te brengen.

Om te beslissen of jouw applicatie gebruik zal maken van een *DataReader* of een *DataSet* bekijk je best de vereiste functionaliteit van jouw applicatie. Gebruik een *DataSet* om

- lokaal gegevens te bewaren zodat ze veranderd kunnen worden. Als je alleen de resultaten van een zoekopdracht nodig hebt, kies je beter voor een *DataReader*.
- gegevens door te geven tussen verschillende lagen of te verkrijgen van een webservice.
- gegevens dynamisch te binden aan component bv. in een GUI-applicatie (*Windows Forms*) of om gegevens uit verschillende bronnen te combineren.
- gegevens te verwerken zonder een verbinding te maken met de gegevensbank.

Als de functionaliteit die een *DataSet* voorziet niet vereist is, dan kan je de performantie van je applicatie verhogen door het gebruik van een *DataReader*. De performantie wordt verbeterd omdat je geheugen spaart, dat anders ingenomen wordt door de *DataSet*, en omdat er minder verwerkingsstappen zijn: het aanmaken en opvullen van de *DataSet* valt weg.

## 6.5 Transacties

Een transactie is een groep van gegevensbankopdrachten die samen uitgevoerd moeten worden. Als er een fout optreedt bij het uitvoeren van de opdrachten dan worden alle opdrachten terug ongedaan gemaakt. Het uitvoeren van een transactie in ADO.NET bestaat uit een aantal stappen:

1. Om een transactie te starten wordt de methode *BeginTransaction* van *IDbConnection* uitgevoerd. Het resultaat van deze methode is een transactie-object van het type *IDbTransaction*.
2. Vervolgens wordt dit transactie-object toegekend aan alle commando-objecten die tot de transactie behoren. Hierbij wordt gebruik gemaakt van de eigenschap *Transaction* van *IDbCommand*.
3. Daarna worden alle opdrachten van de transactie uitgevoerd.
4. Tot slot wordt de methode *Commit* van het transactie-object opgeroepen als alle opdrachten gelukt zijn, in het andere geval wordt de methode *Rollback* uitgevoerd.

Het bovenstaande principe wordt geïllustreerd in het volgend voorbeeld.

---

```

string connString = "Data Source=localhost;Initial Catalog=Northwind;...";
SqlConnection myConnection = new SqlConnection(connString);
myConnection.Open();

// Start a local transaction.
SqlTransaction myTrans = myConnection.BeginTransaction();

// Enlist the command in the current transaction.
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = myConnection;
myCommand.Transaction = myTrans;

try {
 myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) "
 + "VALUES (100, 'Description')";
 myCommand.ExecuteNonQuery();
 myCommand.CommandText = "Insert into Region (RegionID, RegionDescription) "
 + "VALUES (101, 'Description')";
 myCommand.ExecuteNonQuery();
 myTrans.Commit();
 Console.WriteLine("Both records are written to database.");
} catch (Exception e) {
 myTrans.Rollback();
 Console.WriteLine(e.ToString());
 Console.WriteLine("Neither record was written to database.");
} finally {
 myConnection.Close();
}

```

---

## **Hoofdstuk 7**

# **ASP.NET Core MVC**

## 7.1. BASISCONCEPTEN

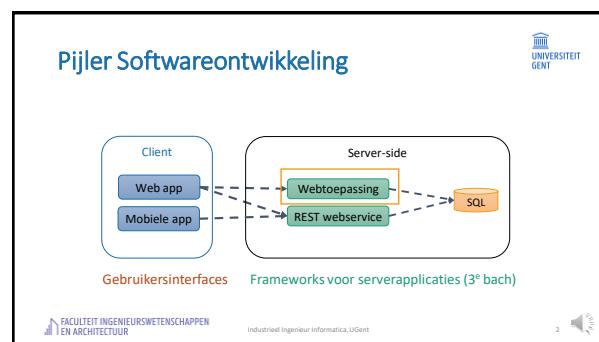
## HOOFDSTUK 7. ASP.NET CORE MVC

**7.1 Basisconcepten**

ASP.NET Core MVC

Veerle Ongenaee

1

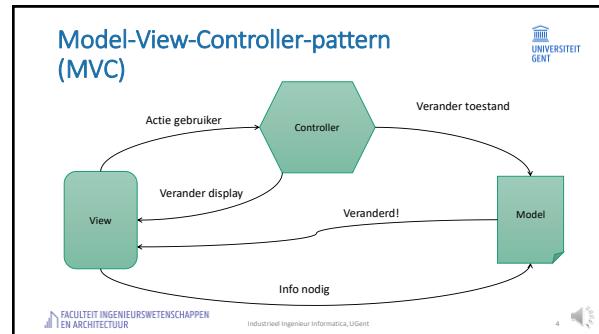


2

**Overzicht**

- MVC

3



4

**MVC in GUI's**

- Oorsprong SmallTalk
- GUI
  - Tabellen
  - Tekstcomponenten
    - M: document
    - V: tonen
    - C: editeermogelijkheden
  - ...
- Klok
  - Mathematisch model
  - Tonen
  - Aanpassen

5



6

## HOOFDSTUK 7. ASP.NET CORE MVC

### 7.1. BASISCONCEPTEN

**MVC in webapplicaties**

- Drie modules
  - Applicatiemodel: gegevensvoorstelling en applicatielogica (Model)
  - Presentatie gegevens en gebruikersinput (View)
  - Gedrag applicatie (flow) – doorsturen aanvragen (Controller)

bron: <https://www.jeremymorgan.com/blog/programming/what-is-mvc/>

7

**Overzicht**

- MVC
- Voorbeeld

8

**Informatie**

- ASP.NET Core MVC
  - Webapplicaties
  - REST-webservices (Web API)
    - Kan gecombineerd worden met een Angular-client of React-client
- Nodig?
  - Visual Studio 2022, Visual Studio for Mac, Visual Studio Code
- Kenmerken
  - Windows – Linux – OSX
  - HTML5 – CSS – Javascript
  - NuGet – npm – Bower – Gulp
  - Microsoft Azure

9

**ASP.NET Core MVC**

- Template MVC-applicatie gebruikt

10

**Voorbeeld**

http://localhost:44381/Steden/

Overzicht Steden

- Brussel
- Gent

Steden

Naam

Brussel

Gent

http://localhost:44381/Steden/Toon/Brussel

Foto's Brussel

Atomium

Manneken Pis

Stadhuis

UNIVERSITEIT GENT

11

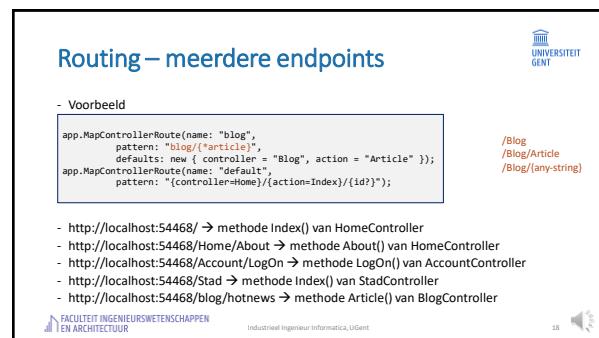
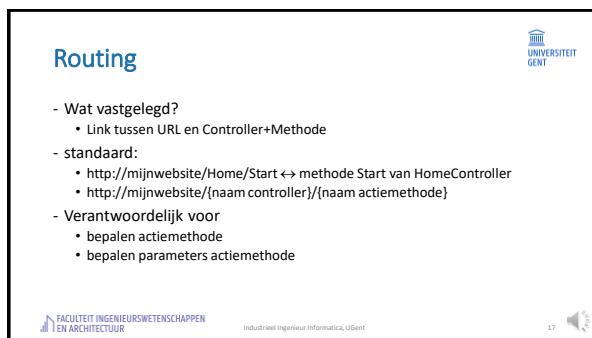
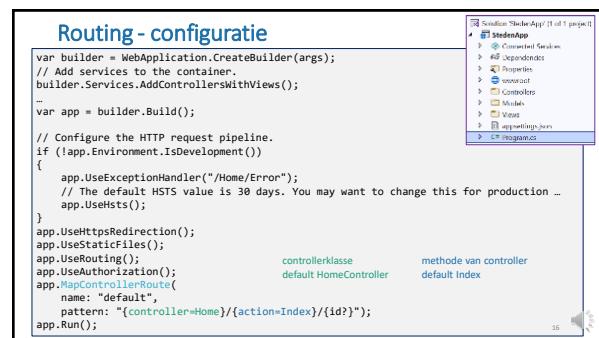
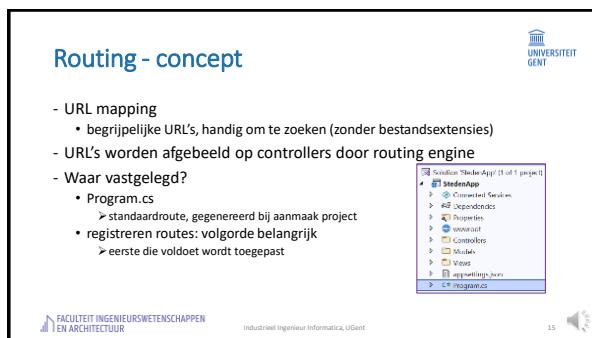
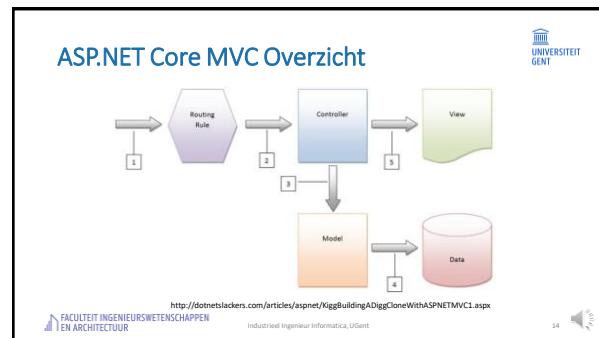
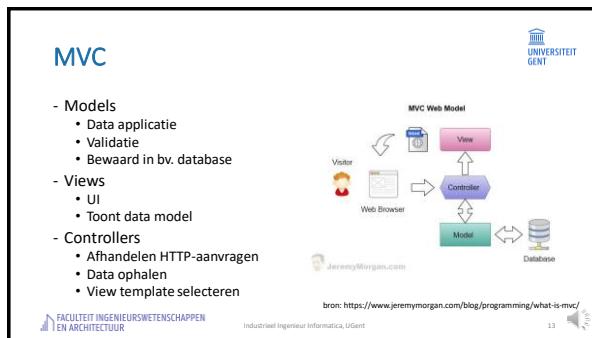
**Overzicht**

- MVC
- Voorbeeld
- Routing

12

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC



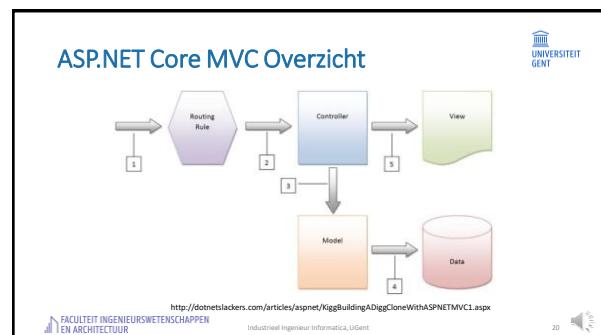
HOOFDSTUK 7. ASP.NET CORE MVC

## 7.1. BASISCONCEPTEN

# Overzicht

- MVC
- Voorbeeld
- Routing
- Controller

19



20

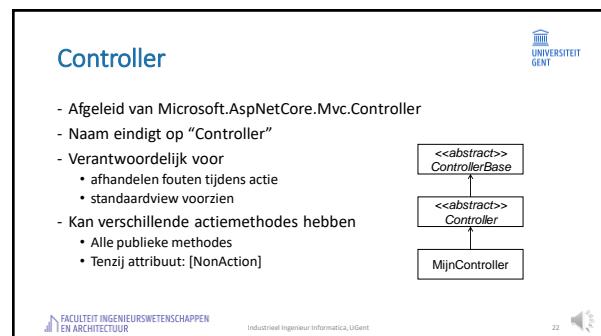
The screenshot shows the Solution Explorer window with the following project structure:

- Solution: StedenApp (1 of 1 project)
  - StedenApp
    - Connected Services
    - Properties
    - Resources
    - wwwroot
      - css
      - fonts
      - images
      - js
    - Views
      - Home
      - Shared
      - Steden
      - Imports
        - ViewImports.cshtml
        - ViewStart.cshtml
    - Models
      - ErvaringModel.cs
      - Gemeente.cs
      - StadModel.cs
      - StadFabrik.cs
      - StedenFabrik.cs
    - Views
      - Home
      - Shared
      - Steden
      - Imports
    - appsettings.json
    - Program.cs

**Structuur ASP.NET Core MVC applicatie**

  - Aparte map voor modellen, views en controllers
  - Model
    - gewone klassen
    - map Models

21



22

## Controller - voorbeeld

- <http://localhost:3250/Home/Privacy>

```
public class HomeController : Controller
{
 public IActionResult Privacy()
 {
 return View();
 }
}
```

StadApp Home Privacy

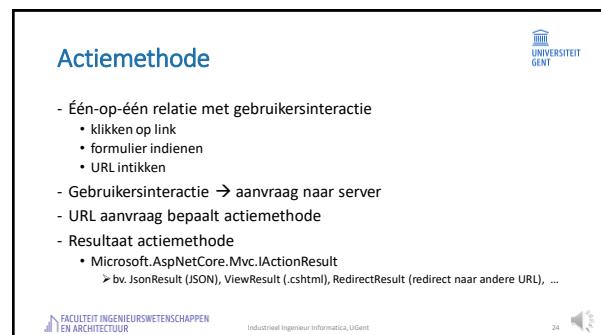
### Privacy Policy

Use this page to detail your site's privacy policy.

- Methode `View()`

- stuurt door naar `naamactiemethode.cshtml`
- Hier: `Privacy.cshtml`

23



24

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**(I) ActionResult**

| Action Result         | Helper Method    | Description                                                                                         |
|-----------------------|------------------|-----------------------------------------------------------------------------------------------------|
| ViewResult            | View             | Renders a view as a Web page.                                                                       |
| PartialViewResult     | PartialView      | Renders a partial view, which defines a section of a view that can be rendered inside another view. |
| RedirectResult        | Redirect         | Redirects to another action method by using its URL.                                                |
| RedirectToRouteResult | RedirectToAction | Redirects to another action method.                                                                 |
| RedirectToRouteResult | RedirectToRoute  | Redirects to another action method.                                                                 |
| ContentResult         | Content          | Returns a user-defined content type.                                                                |
| JsonResult            | Json             | Returns a serialized JSON object.                                                                   |
| JavaScriptResult      | JavaScript       | Returns a script that can be executed on the client.                                                |
| FileResult            | File             | Returns binary output to write to the response.                                                     |
| EmptyResult           | (None)           | Represents a return value that is used if the action method must return a null result (void).       |

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 25

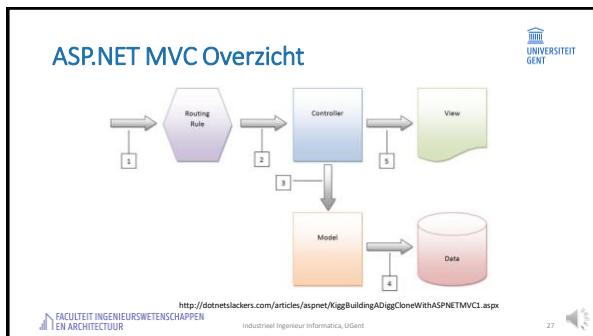
25

**Overzicht**

- MVC
- Voorbeeld
- Routing
- Controller
- View

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 26

26



27

**View**

- Razor view engine
- .cshtml-bestanden
  - HTML genereren gebruik makend van C#
- @
  - C#-code
- Parser kent C#-semantiek en kan onderscheid maken tussen code en HTML

```

@foreach (var item in Model)
{
 <tr>
 <td> URL Methode Controller Parameter(s) </td>
 @Html.DisplayFor(modelItem => item.Naam)
 </td>
 </tr>
}

```

http://localhost:3250/Steden/Toon/Brussel  
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 28

28

**View**

- Notatie voor codefragmenten
- Uitvoer naar HTML
  - @varNaam
  - schrijft naar uitvoer
  - encodeert eventueel naar HTML (bescherming tegen HTML-injectie en cross site scripting)
    - alle invoer (mogelijk malafide) die getoond wordt in een pagina omzetten naar tekst
    - https://www.youtube.com/watch?v=cbmBDiR6WaY

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 29

29

**Cross site scripting (XSS)**

- Kwaadaardige code injecteren in een webpagina
- Invoer gebruiker toevoegen in HTML van de webpagina

https://blogs.sap.com/2015/12/17/xss-cross-site-scripting-overview-with-contexts/  
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR Industriel Ingenieur Informatica, UGent 30

30

## HOOFDSTUK 7. ASP.NET CORE MVC

### 7.1. BASISCONCEPTEN

#### View HTML-helpers

- Methodes om HTML te genereren
  - ~ webcomponenten
- Voorbeeld
  - @Html.DisplayNameFor(model => model.Naam)
    - Genereert een HTML-string voor de naam van de eigenschap
  - @Html.DisplayFor(modelItem => item.Naam)
    - Genereert een HTML-string voor de eigenschap die het resultaat is van de lambda-uitdrukking
- Gegenereerde HTML
  - Naam
  - Brussel // Gent

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



31

#### Tag-helpers

- Extra attributen om HTML-teks te genereren

```
<a asp-action="Toon" asp-route-id="@item.Naam">
 @Html.DisplayFor(modelItem => item.Naam)

```

```
Brussel
```

```
<a asp-controller="Home" asp-action="About">
 About

```

```
About
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



32

31

32

#### Layout - Sjablonen

- Layout-pagina's
  - sjabloon voor webpagina's
  - .cshtml-bestand
- Default: \_Viewstart.cshtml

```
ViewStart.cshtml
@{
 Layout = "_Layout";
}
```



```
_Layout.cshtml
----- TEMPLATE -----
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>@ViewData["Title"] | Steden</title>
 <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
 <link rel="stylesheet" href="/css/site.css" />
 </head>
 <body>
 <header></header>
 <div class="container">
 <main role="main" class="pb-3">
 @RenderBody()
 </main>
 </div>
 ...
 <script src="/lib/jquery/dist/jquery.min.js"></script>
 <script src="/lib/bootstrap/dist/js/bootstrap.bundle.min.js" asp-append-version="true"></script>
 @RenderSection("Scripts", required: false)
 </body>
</html>
```

33

#### Layout - Sjablonen

- Verschillende (in te vullen) secties in sjablonen

```
<!DOCTYPE html>
<html>
 <head>
 <title>Simple Site</title>
 </head>
 <body>
 <div id="header">
 Home
 About
 </div>
 <div id="left-menu">
 <renderSection("Menu", required: false) />
 </div>
 <div id="body">
 <renderBody() />
 </div>
 <div id="footer">
 <renderSection("Footer", required: false) />
 </div>
 </body>
</html>
```

(hi-about This Site:)  
 This is some content that will make up the "about" page of the website. It is generated from a template with a layout template. The content you are seeing here comes from the Home.layout file.  
 (p)  
 And obviously I can have code in here too. Here is the current datetime: @DateTime.Now

```
<section menu>
 <ul id="sub-menu">
 About Item 1
 About Item 2

</section>
```

```
<section footer>
 <p>This is my custom footer for Home</p>
```

Industriel Ingenieur Informatica, UGent



34

33

34

#### Overzicht

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



35

#### Data van Controller naar View

- Via ViewData
  - Map
- Via de parameter van de methode View
  - objecten van modelklassen

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



36

35

36

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**Gebruik ViewData in Controller**

```
public IActionResult Contact()
{
 ViewData["Message"] = "Your contact page.";
 return View();
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

37

**Gebruik ViewData in View**

```
@{ ViewData["Title"] = "Contact";
}
<h2>@ViewData["Title"]</h2>
<h3>@ViewData["Message"]</h3>
```

UNIVERSITEIT GENT  
Namen Steden Steden  
Contact.  
Your contact page.  
One Microsoft Way  
Redmond, WA 98052-6399  
P: 425.555.0100  
Support: Support@example.com  
Marketing: Marketing@example.com  
© 2016 - StedenWeb  
Industrial Ingenieur Informatica, UGent

38

**Voorbeeld Gebruik ViewData**

- /Steden/ of /Steden/Index
- Methode Index van StedenController

```
StedenController CONTROLLER
public IActionResult Index(){
 ViewData["Stadsnamen"] = stadFabriek.Stadsnamen;
 return View();
}

@{
 ViewData["Title"] = "Steden";
}
<h2> Overzicht Steden </h2>

 @foreach (string naam in (string[])ViewData["Stadsnamen"])
 {
 @naam
 }

```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
StadApp Home Privacy  
Overzicht Steden  
• Brussel  
• Gent  
Index.cshtml  
VIEW

39

**Data van Controller naar View**

- Via eigenschap ViewData
- Via de parameter van de methode View (in Controller)
  - objecten van modelklassen
  - Type model specifiëren in view

UNIVERSITEIT GENT  
Controller → View  
Controller → Model  
Model → Data  
Model → View  
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industrial Ingenieur Informatica, UGent

40

**Data van Controller Naar View**

```
StedenController CONTROLLER
public IActionResult Overzicht()
{
 Gemeente[] steden = stadFabriek.Steden;
 return View(steden);
}
```

Overzicht.cshtml VIEW

```
@model IEnumerable<StadApp.Models.Gemeente>
...
@foreach (var item in Model) {
 <tr> <td>
 Toon" asp-route-id="@item.Naam">@Html.DisplayFor(modelItem => item.Naam)
 </td> </tr>
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Steden  
Naam  
Brussel  
Gent  
Overzicht.cshtml  
eventueel gegenereerd door VS  
VIEW

41

**Overzicht**

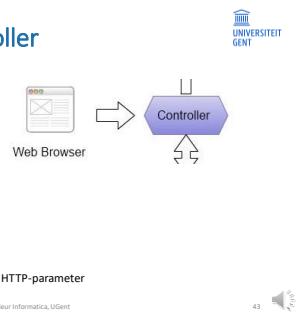
- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View
- Informatie van de client

UNIVERSITEIT GENT  
Controller  
Web Browser  
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industrial Ingenieur Informatica, UGent

42

**HOOFDSTUK 7. ASP.NET CORE MVC****7.1. BASISCONCEPTEN****Informatie voor Controller**

- HTTP-parameters
  - Querystring  
/Products/Detail?id=3
  - Formulier
- Parameters in URL  
/Products/Detail/3
- Toegang
  - via Request-eigenschap
  - via parameters actiemethode
    - naam parameter in methode = naam HTTP-parameter



43

**HTTP-Parameters in QueryString**

- Voorbeeld /Products/Detail?id=3
- via Request-eigenschap

```
Methode Controller
```

```
public IActionResult Detail() {
 int id = Convert.ToInt32(Request["id"]);
 ...
}
```

- via parameters actiemethode

```
Methode Controller
```

```
public IActionResult Detail(int id) {
 int identificatie = id;
 ...
}
```

Industriel Ingenieur Informatica, UGent

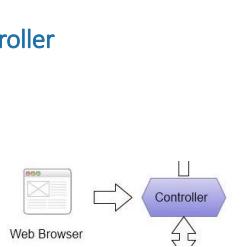
43



44

**Informatie voor Controller**

- HTTP-parameters
  - Querystring  
/Products/Detail?id=3
  - Formulier
- Parameters in URL  
/Products/Detail/3



Industriel Ingenieur Informatica, UGent

45

**Parameters in URL**

- Parameters opgehaald uit URL
  - via parameters actiemethode

- Voorbeeld

- URL: /Products/Detail/3
- Routing: /(controller)/(action)/(id)

```
Methode Controller
```

```
public IActionResult Detail(int id) {
 int identificatie = id;
 ...
}
```

45



46

**Voorbeeld Parameters**

- /Steden/Toon/Brussel
  - Methode Toon van StedenController
  - Stad.cshtml



47

**Voorbeeld - Controller**

```
StadController
```

```
// GET: /Steden/Toon/id
public IActionResult Toon(string id)
{
 Gemeente stad = stadFabriek.geefStad(id);
 return View("Stad", stad);
}
```

Naam cshtml-bestand      Model doorgeven aan View

Industriel Ingenieur Informatica, UGent



48

47

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**Voorbeeld - View**

Stad.cshtml Model doorgeven aan View

```
@model StedenApp.Models.Gemeente
@{
 ViewData["Title"] = "Foto's " + Model.Naam;
}

<h2>Foto's @Html.DisplayFor(model => model.Naam)</h2>

@foreach (Foto foto in Model.Fotos)
{
 <figure>
 <figcaption>@foto.Omschrijving</figcaption>

 </figure>
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

49

49

**Methode View**

- View()
  - viewengine gebruikt `naammethode.cshtml`
- View("string")
  - viewengine gebruikt `string.cshtml`
- View(object)
  - object beschikbaar als variabele Model in `.naammethode.cshtml`-pagina
- View("string", object)
  - object beschikbaar als variabele Model in `string.cshtml`

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

50

50

**Namenruimten in View**

- `@using` → gebruik namenruimten
- In bestand `_ViewImports`

`using StedenApp`  
`using StedenApp.Models`  
`@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`

All helpers uit de gegeven assembly

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

51

51

**Informatie voor Controller**

- HTTP-parameters
  - Querystring  
`/Products/Detail?id=3`
  - Formulier
- Parameters in URL  
`/Products/Detail/3`

Web Browser

Controller

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

52

52

**Data van formulieren**

`/Schaak/Details/TORGE`

`/Schaak/Create/TORGE`

`Schaakpartij toevoegen`

`Partij`

`Speler Wit`  
`Veldle`

`Speler Zwart`  
`Lid`

`Winnaar`  
`Zwart`

`Bewaren`

`Brug meer partij`

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

53

53

**Controller en formulier**

welke Create?

```
// GET: Schaak/Create/ID
public ActionResult Create(string id)
{
 TornooiMetPartijen tornooi = tornooiDAO.GetTornooi(id);
 Partij partij = new Partij();
 partij.Tornooi = tornooi;
 return View(partij);
}

// POST: Schaak/Create/ID
[HttpPost]
public ActionResult Create(string ID, Partij partij)
{
 try {
 TornooiMetPartijen tornooi = tornooiDAO.GetTornooi(ID);
 partij.Tornooi = tornooi;
 tornooiDAO.AddPartij(partij);
 return RedirectToAction("Details", new { id = tornooi.ID });
 } catch {
 return View(partij);
 }
}
```

`Partij`

`Tornooi`

`SpelerWit`

`SpelerZwart`

`Winnaar`

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

54

154

## HOOFDSTUK 7. ASP.NET CORE MVC

### 7.1. BASISCONCEPTEN

**View en formulier**

overzicht foutboodschappen validatie

```
<form asp-action="Create">
 <div asp-validation-summary="ModelOnly" class="text-danger"></div>
 <div class="form-group">
 <label>SpelerWit</label>
 <input asp-for="SpelerWit" class="form-control" /> tektvak

 </div>
 <div class="form-group">
 <label>SpelerZwart</label>
 <select asp-for="SpelerZwart" asp-items="Html.GetEnumSelectList<SpelerZwart>(); " class="form-control" />

 </div>
 <div class="form-group">
 <input type="submit" value="Bewaar" class="btn btn-primary" />
 </div>
</form>
```

55

**View en formulier**

```
@model SchaakApp.Models.Partij
@{
 ViewData["Title"] = "Create";
}

Terug naar tornooi

@section Scripts {
 @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Universiteit Gent  
Industriel Ingenieur Informatica, UGent

56

55

56

**HTML - helpermethodes**

- ActionLink
- BeginForm
- ValidationSummary
- LabelFor
- EditorFor
- ValidationMessageFor
- Lambda uitdrukking

model => model.SpelerWit

argument = functie

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Universiteit Gent  
Industriel Ingenieur Informatica, UGent

57

**Alternatief**

```
<div class="editor-label">
 @Html.LabelFor(model => model.SpelerWit)
</div>
<div class="editor-field">
 @Html.EditorFor(model => model.SpelerWit)
 @Html.ValidationMessageFor(model => model.SpelerWit)
</div>

<div class="form-group">
 <label asp-for="Title" class="col-md-2 control-label"></label>
 <div class="col-md-10">
 <input asp-for="Title" class="form-control" />

 </div>
</div>
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Universiteit Gent  
Industriel Ingenieur Informatica, UGent

58

57

58

**Overzicht**

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View
- Informatie van de client
- Dependency Injection

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Universiteit Gent  
Industriel Ingenieur Informatica, UGent

59

**Dependency Injection**

- Afhankelijkheden
- Parameter in constructor (constructor injection)
- Interface
- Container ("runtime") levert concrete implementatie

```
public class StedenController : Controller
{
 IStedenFabriek stadFabriek;

 public StedenController(IStedenFabriek stadFabriek)
 {
 this.stadFabriek = stadFabriek;
 }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Universiteit Gent  
Industriel Ingenieur Informatica, UGent

60

59

60

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**Dependency Injection**

- Transient: telkens nieuw object
- Singleton: één
- Scoped: één per request

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
//services.AddTransient<IOperationTransient, Operation>();
//services.AddScoped<IOperationScoped, Operation>();
builder.Services.AddSingleton<StedenApp.Models.IStedenFabriek, StedenApp.Models.StedenFabriek>();

var app = builder.Build();

// Configure the HTTP request pipeline.
...
```

61

**Overzicht**

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View
- Informatie van de client
- Dependency Injection
- Validatie

62

**Validatie**

<https://localhost:44332/Schaak/Create/ITR02>

63

**Validatie – Model - View**

**Model**

```
public class Partij {
 public Turnooi Turnooi { get; set; }

 [Display(Name = "White Player")]
 [Required(ErrorMessage = "White player required")]
 public string SpelerWit { get; set; }
}
```

**View**

64

**Validatie - View**

View.cshtml

```
<div class="form-group">
 <label asp-for="SpelerWit" class="col-md-2 control-label"></label>
 <div class="col-md-10">
 <input asp-for="SpelerWit" class="form-control" />

 </div>
</div>
```

Gegenerateerde HTML

```
<div class="form-group">
 <label class="control-label" for="SpelerWit">Speler Wit</label>
 <input class="form-control" type="text" data-val="true" data-val-required="Witte speler vereist" id="SpelerWit" name="SpelerWit" value="" />

</div>
```

Bepaald door Display-attribut  
data-\*: attributen met info die in JS-scripts gebruikt kan worden

65

**Validatie in HTML met jQuery**

- jQuery = javascript bibliotheek

Gegenerateerde HTML

```
<div class="form-group">
 <label class="control-label" for="SpelerWit">Speler Wit</label>
 <input class="form-control" type="text" data-val="true" data-val-required="Witte speler vereist" id="SpelerWit" name="SpelerWit" value="" />

</div>
```

Valideert voor welk formulierelement (id)? Bevat foutbericht?

validatie instellen op dit element  
type validatie

66

**HOOFDSTUK 7. ASP.NET CORE MVC****7.1. BASISCONCEPTEN****Client-side validatie**`_ValidationScriptsPartial.cshtml`

```
<script src="/lib/jquery-validation/dist/jquery.validate.min.js"></script>
<script src="/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js">
</script>
```

Industriel Ingenieur Informatica, UGent



67

**Validatie - Controller**

```
// POST: /Schaaktornooien/Create/ID
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(string ID,[Bind("SpelerWit,SpelerZwart,Winnaar")]Partij partij)
{
 TornooiMetPartijen tornooi = tornooiDAO.GetTornooi(ID);
 partij.Tornooi = tornooi;
 if (ModelState.IsValid)
 {
 tornooiDAO.AddPartij(partij);
 return RedirectToAction("Details", new { id = tornooi.ID });
 }
 return View(partij);
}
```



Industriel Ingenieur Informatica, UGent

68

**Mogelijke validaties****- Required****EnumDataType**

```
[EnumDataType(typeof(Winnaar), ErrorMessage = "White, Black or Remise!!")]
public Winnaar Winnaar { get; set; }
```

```
<div class="form-group">
 <label asp-for="Winnaar" class="col-md-2 control-label"></label>
 <div class="col-md-10">
 @*input asp-for="Winnaar" class="form-control" />@
 <select asp-for="Winnaar"
 asp-items="Html.GetEnumSelectList<Winnaar>()" class="form-control">
 </select>

 </div>
</div>
```



69

**Mogelijke validaties****- CreditCard**

```
[CreditCard]
public string CreditCardNumber { get; set; }

[CreditCard(
 AcceptedCardTypes=CreditCardAttribute.CardType.Visa |
 CreditCardAttribute.CardType.MasterCard)]
public string CreditCardNumber { get; set; }
```



Industriel Ingenieur Informatica, UGent

70

**Mogelijke validaties****- EmailAddress**

```
[EmailAddress(ErrorMessage = "Invalid Email Address")]
public string Email { get; set; }
```

**- MaxLength****- MinLength****- StringLength**

```
[Required]
[StringLength(1000)]
public string Description { get; set; }
```

Industriel Ingenieur Informatica, UGent



71

**Mogelijke validaties****- Phone**

```
[Phone]
public string Phone { get; set; }
```

**- Range**

```
[Required]
[Range(0, 999.99)]
public decimal Price { get; set; }
```

**- RegularExpression**

```
[RegularExpression(@"^([A-Z]+[a-zA-Z'-'\s]*$")]
[Required]
[StringLength(30)]
public string Genre { get; set; }
```



Geen \ nodig

Industriel Ingenieur Informatica, UGent

72

71

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**Remote Validation**

```
public class User
{
 [Remote(action: "VerifyEmail", controller: "Users")]
 public string Email { get; set; }
}

[AcceptVerbs("Get", "Post")]
public IActionResult VerifyEmail(string email)
{
 if (!_userRepository.VerifyEmail(email))
 {
 return Json(data: $"Email {email} is already in use.");
 }

 return Json(data: true);
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

UNIVERSITEIT GENT

73

73

**Overzicht**

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Informatie van de client
- Dependency Injection
- Validatie
- Overposting en Cross-site Request Forgery
- Data van Controller naar View

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

UNIVERSITEIT GENT

Industrial Ingenieur Informatica, UGent

74

74

**Controller – Bind - overposting**

```
// POST: /Schaaktornooien/Add/ID
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Add(string ID, [Bind("SpelerWit,SpelerZwart,Winnaar")]Partij partij)
{
 TornooiMetPartijen tornooi = tornooiDAO.GetTornooi(ID);
 partij.Tornooi = tornooi;
 if (ModelState.IsValid)
 {
 tornooiDAO.AddPartij(partij);
 return RedirectToAction("Details", new { id = tornooi.ID });
 }
 return View(partij);
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrial Ingenieur Informatica, UGent

75

75

**Bind voorkomt Overposting**

```
public class Student
{
 public int ID { get; set; }
 public string LastName { get; set; }
 public string FirstName { get; set; }
 public DateTime EnrollmentDate { get; set; }
 public string Secret { get; set; }
}

public virtual ICollection<Enrollment> Enrollments { get; set; }
```

Niet op website  
→ Voorkomen toch meegeven

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

UNIVERSITEIT GENT

Industrial Ingenieur Informatica, UGent

76

76

**Cross-site request forgery**

- Browser stuurt authenticatie token bij elke request
  - Bv. gecached cookie
  - Hacker → js-script met call naar zelfde site → cookie meegestuurd
    - <https://www.youtube.com/watch?v=9JrzPX1pVjs>
- Preventie
  - Twee tokens
    - In cookie
    - In form
    - Moeten gelinkt zijn op server

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrial Ingenieur Informatica, UGent

77

77

**Controller – ValidateAntiForgeryToken cross-site request forgery voorkomen**

```
// POST: /Schaaktornooien/Add/ID
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Create(string ID, [Bind("SpelerWit,SpelerZwart,Winnaar")]Partij partij)
{
 TornooiMetPartijen tornooi = tornooiDAO.GetTornooi(ID);
 partij.Tornooi = tornooi;
 if (ModelState.IsValid)
 {
 tornooiDAO.AddPartij(partij);
 return RedirectToAction("Details", new { id = tornooi.ID });
 }
 return View(partij);
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

UNIVERSITEIT GENT

Industrial Ingenieur Informatica, UGent

78

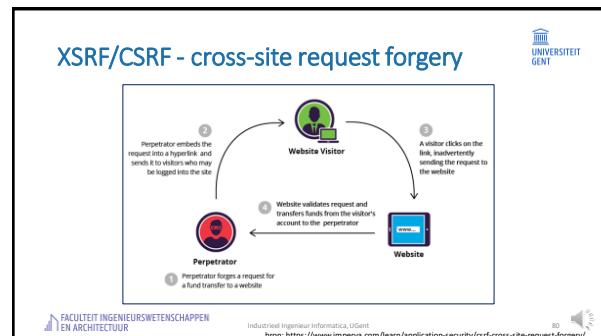
HOOFDSTUK 7. ASP.NET CORE MVC

## 7.1. BASISCONCEPTEN

# Controller – ValidateAntiForgeryToken cross-site request forgery voorkomen

```
<input name="__RequestVerificationToken" type="hidden" value="CfD8B8C792b9B153984c1e70e-274
ge610XQ4mM1L17MqpcGkRyX1t+g=MfT1y+-138f7eby_Agk13y17cr+h0BNWp0vA7x117a3SMw0t3kFv
KzMD-AdJ5t7hLqHtQyuop" />
</form>
```

79



80

# Overzicht



UNIVERSITEIT  
GENT

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View
- Informatie van de client
- Dependency Injection
- Validatie
- Overposting en Cross-site Request Forgery
- Globalization en Localization

81

# Globalization en Localization



- Globalization

- Applicaties zo ontwikkelen dat ze verschillende talen en culturen ondersteunen

  
- Localization

- Applicatie (die globalization ondersteunt) aanpassen voor één specifieke taal/cultuur

82

# Globalization en Localization



UNIVERSITEIT  
GENT

- Drie stappen
  - Maak de inhoud aanpasbaar
  - Voorzie inhoud per taal en cultuur
  - Selecteer de taal en cultuur

83

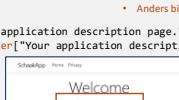
# Maak inhoud aanpasbaar - Controller



```
public class HomeController : Controller {
 private readonly IStringLocalizer<HomeController> _localizer;
 public HomeController(IStringLocalizer<HomeController> localizer)
 {
 _localizer = localizer;
 }
 public IActionResult About()
 {
 //ViewData["Message"] = "Your application description page.";
 ViewData["Message"] = _localizer["Your application description page."];
 return View();
 }
 -
}
```

Resultaat?

- Key indien geen bron beschikbaar
- Anders bijhorende waarde



FACULTEIT INGENIEURSWETENSCHAPPEN  
AN ARCHITECTUUR

Industrial

© 2020 - Universiteit Gent

84

84

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC

**Maak inhoud aanpasbaar - Controller**

```
Program.cs
```

```
builder.Services.AddLocalization(options => options.ResourcesPath = "Resources");
...
// DAO
builder.Services.AddSingleton();
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

85

**Maak inhoud aanpasbaar - View**

```
@using Microsoft.AspNetCore.Mvc.Localization
@inject IViewLocalizer Localizer
@{
 ViewData["Title"] = "Home Page";
}

<p>@ViewData["Message"]</p>

<address>
 @Localizer["Support"]:
 Support@example.com

 @Localizer["Marketing"]:
 Marketing@example.com
</address>
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

86

**Maak inhoud aanpasbaar - View**

```
Program.cs
```

```
builder.Services.AddLocalization(options => options.ResourcesPath = "Resources");
builder.Services.AddControllersWithViews()
 .AddViewLocalization()
 .AddDataAnnotationsLocalization();

// DAO
builder.Services.AddSingleton();
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

87

**Maak inhoud aanpasbaar - Model**

```
public class Partij {
 public Tornooi Tornooi { get; set; }

 [Display(Name = "White player")]
 [Required(ErrorMessage = "White player required")]
 public string SpelerWit { get; set; }
}
```

Schaakpartij toevoegen

Speler Wit  
White player vereist  
Speler Zwart  
Zwart  
Winaar  
Wit  
Toevoegen  
Tornaai terug

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

88

**Maak inhoud aanpasbaar - Model**

```
public class Partij {
 public Tornooi Tornooi { get; set; }

 [Display(Name = "White Player")]
 [Required(ErrorMessage = "White player required")]
 public string SpelerWit { get; set; }

 [Display(Name = "Black Player")]
 [Required(ErrorMessage = "Black player required")]
 [PlayersVerschillend("SpelerWit", ErrorMessage = "White and black are different players")]
 public string SpelerZwart { get; set; }

 [Required]
 [EnumDataType(typeof(Winnaar)), ErrorMessage = "White, Black or Remise!!"]
 public Winnaar Winnaar { get; set; }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

89

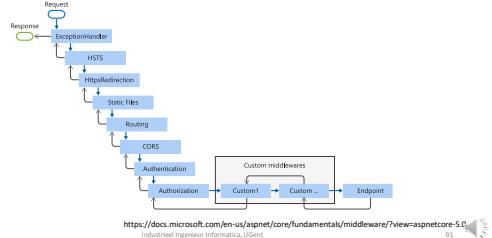
**Voorzie inhoud per taal en cultuur**

- .resx-bestanden
  - XML
  - Naam/waarde-paren
  - Culture (US, NL, BE, ...)
  - Date, time, measurement unit
  - UI culture (en, nl, ...)
  - taal
- Mappenstructuur idem project

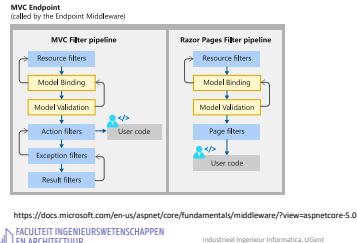
| Name      | Value         |
|-----------|---------------|
| Support   | Ondersteuning |
| Marketing | Verkoop       |

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

90

HOOFDSTUK 7. ASP.NET CORE MVC7.1. BASISCONCEPTEN**Middleware ASP.NET core MVC**

91

**Endpoint Middleware**

92

**Localization Middleware**

- Op basis van instelling browser
- Optioneel: expliciet instellen voor welke taal+cultuur-combinatie de webapp ondersteuning biedt
  - Meegegeven via cookies, in de querystring, headers

Program.cs

```
var supportedCultures = new[] { "en-US", "fr" };
var localizationOptions
 = new RequestLocalizationOptions().SetDefaultCulture(supportedCultures[0])
 .AddSupportedCultures(supportedCultures)
 .AddSupportedUICultures(supportedCultures);
```

app.UseRequestLocalization(localizationOptions);

93

93

**Overzicht**

- |                                                                                                                                                                                |                                                                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>- MVC</li> <li>- Voorbeeld</li> <li>- Routing</li> <li>- Controller</li> <li>- View</li> <li>- Data van Controller naar View</li> </ul> | <ul style="list-style-type: none"> <li>- Informatie van de client</li> <li>- Dependency Injection</li> <li>- Validatie</li> <li>- Overposting en Cross-site Request Forgery</li> <li>- Globalization en Localization</li> <li>- Authentication en Authorization</li> </ul> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



94

**Beveiliging**

- Afschermen delen van een website
- Twee stappen
  - Authentificatie
    - Gebruiker "herkennen"
    - De gebruiker is wie hij zegt te zijn
    - Inloggen
  - Autorisatie
    - Bepaalde rechten (niet) toekennen aan bepaalde gebruikers
      - Leden kunnen meer pagina's bekijken dan niet-leden
      - Administrators kunnen nog andere pagina's raadplegen

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

95

95

**Authenticatie - ASP.NET Core Identity**

- API
- Ondersteunt inlogfunctionaliteit voor de gebruikersinterface
- Beheert gebruikers, wachtwoorden, profielgegevens, rollen, claims, tokens, e-mailbevestiging en meer
- Loginfunctionaliteit
  - Login/wachtwoord
    - Databank (SQL Server)
    - Storage in cloud (Azure Table Storage)
  - Externe login provider: facebook, google, ...

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

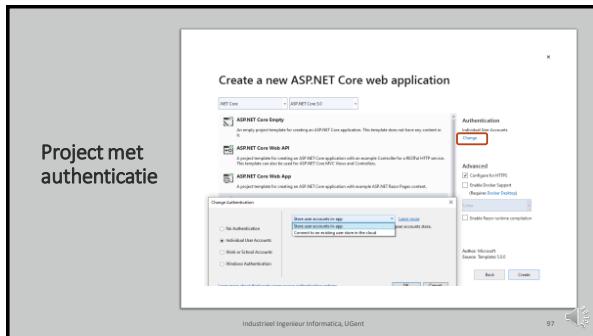
Industriel Ingenieur Informatica, UGent



96

## 7.1. BASISCONCEPTEN

## HOOFDSTUK 7. ASP.NET CORE MVC



97

**Webapp met authenticatie**

98



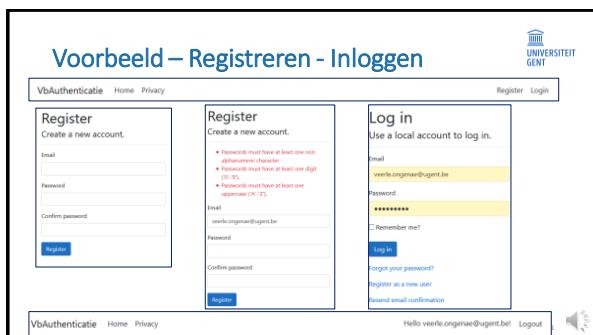
99

**Configuratie (gegenereerd)**

```
- app.UseAuthentication();
app.UseAuthorization();
-
```

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

100



101

**Autorisatie**

- Bepalen of een gebruiker al dan niet toegang heeft tot een bepaalde pagina
  - Wat mag hij/zij doen?
- Twee opties
  - Rollen
  - Policy

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

Industrial Ingenieur Informatica, UGent

102

HOOFDSTUK 7. ASP.NET CORE MVC7.1. BASISCONCEPTEN**Controller: enkel toegankelijk indien ingelogd**

AccountController.cs

```
[Authorize]
public class AccountController : Controller
{
 [AllowAnonymous]
 public ActionResult Login()
 {
 }

 public ActionResult Logout()
 {
 }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

103



AccountController.cs

**Actie: enkel toegankelijk indien ingelogd**

AccountController.cs

```
public class AccountController : Controller
{
 public ActionResult Login()
 {
 }

 [Authorize]
 public ActionResult Logout()
 {
 }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

104



103

104

**Autorisatie: rollen**UNIVERSITEIT  
GENT

```
[Authorize(Roles = "Administrator")]
public class AdministrationController : Controller
{ }

[Authorize(Roles = "HRManager,Finance")]
public class SalaryController : Controller
{ }

[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller
{
 public ActionResult SetTime()
 { }

 [Authorize(Roles = "Administrator")]
 public ActionResult ShutDown()
 { }
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

105

**Autorisatie: policy**UNIVERSITEIT  
GENT

Program.cs

```
- builder.Services.AddAuthorization(options =>
{
 options.AddPolicy("RequireAdministratorRole",
 policy => policy.RequireRole("Administrator"));
});

[Authorize(Policy = "RequireAdministratorRole")]
public IActionResult Shutdown()
{
 return View();
}
```

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

106

105

106

**Overzicht**UNIVERSITEIT  
GENT

- MVC
- Voorbeeld
- Routing
- Controller
- View
- Data van Controller naar View
- Informatie van de client
- Dependency Injection
- Validatie
- Overposting en Cross-site Request Forgery
- Globalization en Localization
- Authentication en Authorization

FACULTEIT INGENIEURSWETENSCHAPPEN

EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

107

107

163



## **Hoofdstuk 8**

# **Webapplicaties**

## 8.1. AUTHENTICATIE

## HOOFDSTUK 8. WEBAPPLICATIES

### 8.1 Authenticatie

Webapplicaties – authenticatie

Veerle Ongenaee

1

### Beveiliging

Authentication      Authorization

<https://www.tantoo.com/blog/authentication-vs-authorization-whats-the-difference/>

- Afslermen delen van een website
- Twee stappen
  - Authenticatie
    - Gebruiker "herkennen"
    - De gebruiker wie hij zegt te zijn
    - Inloggen
  - Autorisatie
    - Beperkte rechten (niet) toekennen aan bepaalde gebruikers
      - Leden kunnen meer pagina's bekijken dan niet-leden
      - Administrators kunnen nog andere pagina's raadplegen

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR      Industriel Ingenieur Informatica, UGent

2

### Verschillende vormen van authenticatie

- Authenticatie met cookies (sessies)
- Authenticatie met tokens
- OAuth2
- Single Sign On (SSO)
- Two Factor Authentication

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR      Industriel Ingenieur Informatica, UGent

3

### Authenticatie met cookies (sessies)

The diagram illustrates the session-based authentication process:

- Client sends login details (username, password) to the Server.
- Server stores session data in memory.
- Client sends (session\_id) to cookies.
- Client gets user profile (authenticated request) (session\_id and from cookie).
- Server compares session\_id with stored data.
- Server sends user profile on success.

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR      https://devtothesecacher/what-really-is-the-difference-between-session-and-token-based-authentication-2639      Industriel Ingenieur Informatica, UGent

4

### Authenticatie met cookies: voor- en nadelen

- Maar één keer inloggen
- Geldigheid van de cookie kan uitgeschakeld worden
- Geldt maar voor één domein
- Gevoelig voor XSS- en CSRF-aanvallen
- (Tijdelijke) informatie van de gebruiker bewaard op de server
  - Niet statusloos

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR      Industriel Ingenieur Informatica, UGent

5

### Verschillende vormen van authenticatie

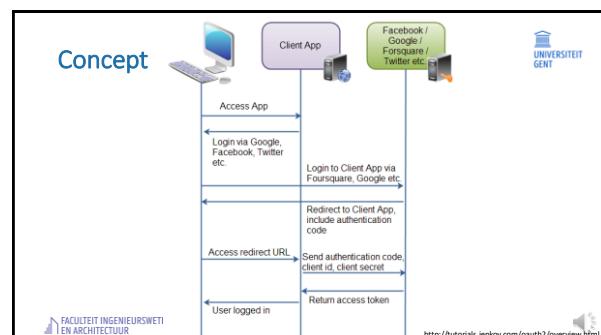
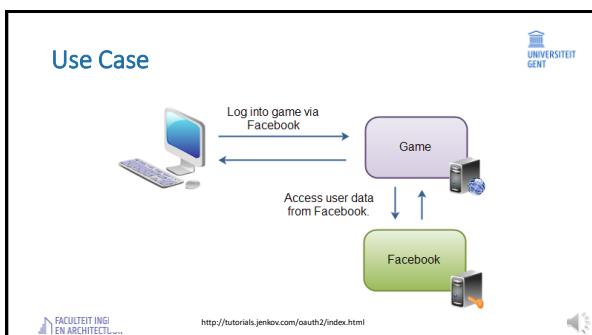
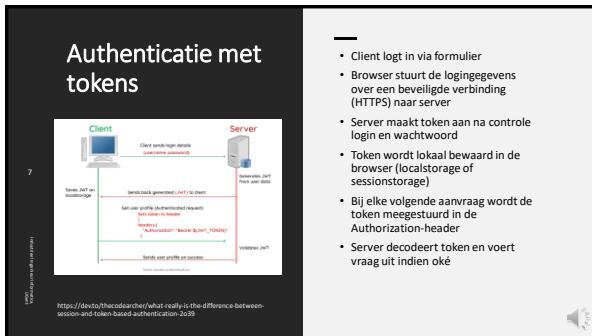
- Authenticatie met cookies (sessies)
- **Authenticatie met tokens**
- OAuth2
- Single Sign On (SSO)
- Two Factor Authentication

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR      Industriel Ingenieur Informatica, UGent

6

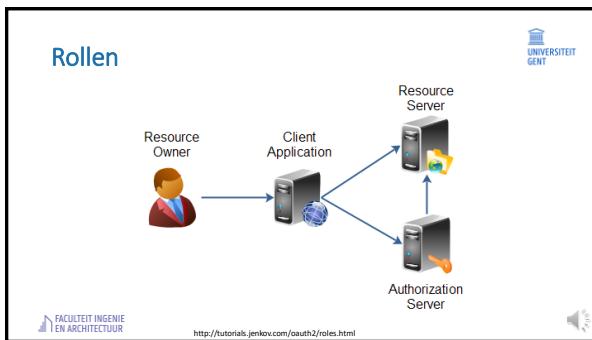
## HOOFDSTUK 8. WEBAPPLICATIES

### 8.1. AUTHENTICATIE



## 8.1. AUTHENTICATIE

## HOOFDSTUK 8. WEBAPPLICATIES



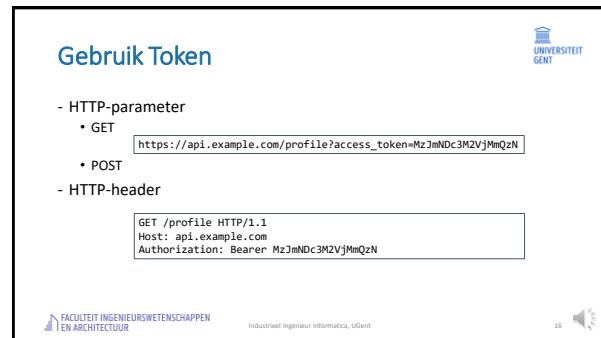
13



14



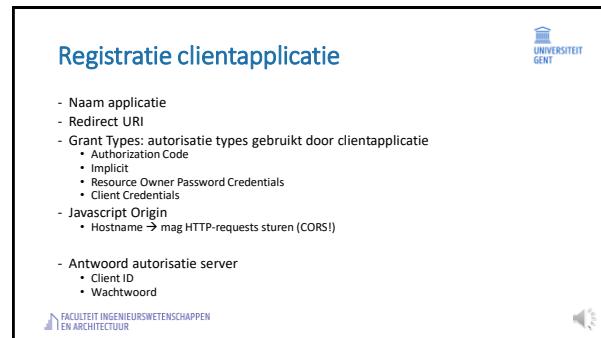
15



16



17



**HOOFDSTUK 8. WEBAPPLICATIES****8.1. AUTHENTICATIE****OAuth 2**

- Basisprincipes
- Clientapplicatie registreren
- **Grant Types**

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Inductie Ingenuier Informatica, UGent



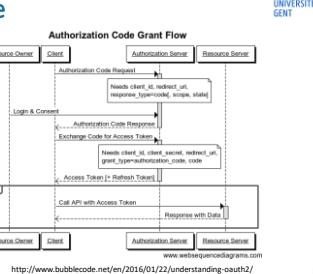
19

**OAuth 2.0 Requests and Responses**

- HTTPS
- Afhankelijk authorization grant type
  - Authorization Code Grant
  - Implicit Grant
  - Resource Owner Password Credentials Grant
  - Client Credentials Grant

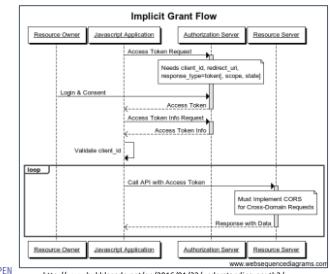


20

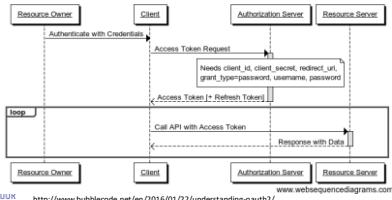
**Authorization Code**Wanneer?  
client = webserverFACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

www.websequencediagrams.com

21

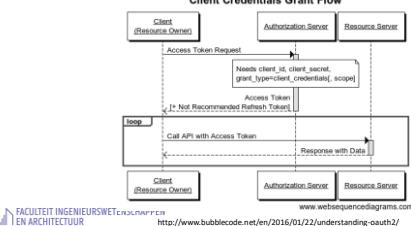
**Implicit**Wanneer?  
client = JS-applicatie  
in browser

22

**Resource Owner Password Credentials**Wanneer?  
client en autorisatie server ontwikkeld door zelfde "authority"**Resource Owner Password Credentials Grant Flow**FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

www.websequencediagrams.com

23

**Client Credentials**Wanneer?  
client = resource owner**Client Credentials Grant Flow**

24

## 8.1. AUTHENTICATIE

## HOOFDSTUK 8. WEBAPPLICATIES

### OAuth 2

- Basisprincipes
- Clientapplicatie registreren
- Grant Types
- Kwetsbaarheid

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



25

### Kwetsbaarheid Implicit

- Minst veilige optie
  - Access token beschikbaar in javascript in browser
- Hacker
  - Maakt website waarmee je kan inloggen via facebook
  - Slachtoffer logt in op deze site → Access Token
  - Gebruikt deze token voor een andere website, indien deze niet voldoende beveiligd is → toegang tot account
- Oplossing
  - Autorisatieserver voorziet informatie over token
    - Clientapplicatie controleert of de client-id wel de juiste is
- Afergeraden (<https://medium.com/oauth-2/why-you-should-stop-using-the-oauth-implicit-grant-2436ced1c926>)
  - Beter: Authorization Code + Proof Key for Code Exchange

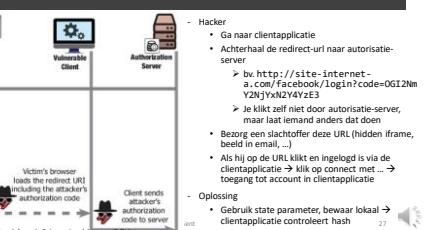
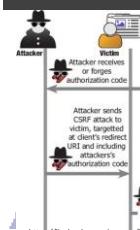
FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



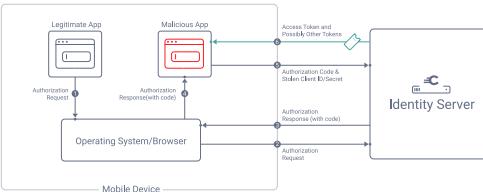
26

### Kwetsbaarheid Authorization Code

<https://livebook.manning.com/book/oauth-2-in-action/chapter-7/24>

27

### Kwetsbaarheid Authorization Code

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

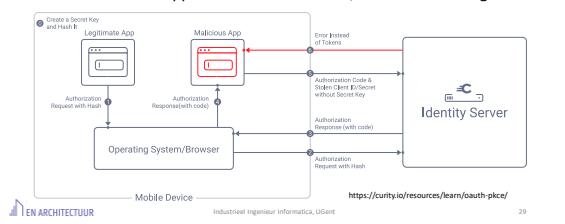
Industriel Ingenieur Informatica, UGent



28

### Proof Key for Code Exchange

- Verzekeren dat de applicatie die de flow start, ook de flow eindigt



EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

29

### Clickjacking

- Autorisatiepagina verbergen achter een transparant iframe
- Gebruiker klikt op link over "allow" knop
- Oplossing
  - Autorisatieserver stuurt "X-Frame-Options" header
    - Waarde DENY, SAMEORIGIN
    - Kan niet in iframe getoond worden (DENY)
    - Domain pagina en iframe moeten dezelfde zijn (SAMEORIGIN)



malicious website  
Welcome to my website!  
Follow me on GitHub!  
click here  
z-index: -1

<http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/>

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent



30

## HOOFDSTUK 8. WEBAPPLICATIES

### 8.1. AUTHENTICATIE

#### Verschillende vormen van authenticatie

- Authenticatie met cookies (sessies)
- Authenticatie met tokens
- OAuth2
- Single Sign On (SSO)
- Two Factor Authentication



FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

31

#### Single Sign On



- Een keer inloggen voor verschillende webapps
- Maar één keer security implementeren
- Single point of failure
- Inbraak op beveiliging veel apps en data getroffen
- Kan gerealiseerd worden met OAuth of SAML

32

31

#### Security Assertion Markup Language (SAML)

1. Gebruiker vraagt webpagina op + controle door service provider
2. Indien geen geldige beveiligingscontext, doorturen naar identity provider (IDP), doorsturen met XML-formaat
3. Browser stuurt aanvraag naar IDP
4. Controle door IDP
5. Eventueel inloggen
6. Antwoord IDP bevat XML-formulier met parameters
7. Browser stelt de vraag opnieuw aan de service provider, nu met parameter uit formulier
8. Na controle parameter, wordt beveiligingscontext aangemaakt en de browser doorgestuurd naar de gebruiker

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

<https://habr.adarsoft.com/index.php/2018/11/05/single-sign-on-with-saml-standards/>

33

#### Verschillende vormen van authenticatie

- Authenticatie met cookies (sessies)
- Authenticatie met tokens
- OAuth2
- Single Sign On (SSO)
- Two Factor Authentication



FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industriel Ingenieur Informatica, UGent

35

#### Two Factor Authentication



- Authenticatie in twee of meerder stappen op meerdere manieren
- Op basis van
  - iets wat de gebruiker weet, zoals een wachtwoord of pincode
  - iets wat de gebruiker heeft, zoals een pasje
  - iets dat de gebruiker is (een eigenschap van de gebruiker), zoals een vingerafdrift
  - waar de gebruiker nu, omdat weet waar de gebruiker toegang probeert te krijgen, of vanaf welk IP-adres,
  - de tijd: op welk tijdstip de gebruiker toegang probeert te krijgen.

36

35

## 8.1. AUTHENTICATIE

## HOOFDSTUK 8. WEBAPPLICATIES

Verschillende vormen van authenticatie

- Authenticatie met cookies (sessies)
- Authenticatie met tokens
- OAuth2
- Single Sign On (SSO)
- Two Factor Authentication

FACULTEIT INGENIEURSWETENSCHAPPEN  
EN ARCHITECTUUR

Industrieel Ingenieur Informatica, UGent

37

37

HOOFDSTUK 8. WEBAPPLICATIES

## 8.2. BEVEILIGINGSRISICO'S

## **8.2 Beveiligingsrisico's**

Webapplicaties -  
beveiligingsproblemen

Veerle Ongenaë



1

## Top 10 beveiligingsrisico's bij webapplicaties



- OWASP Top 10 2021 - The List and How You Should Use It (<https://www.youtube.com/watch?v=hryt-rCUJA>)
  - Open Web Application Security Project
  - Brede consensus meet voorkomende beveiligingsrisico's
  - <https://owasp.org/www-project-top-ten/>



Industrieel Ingenieur Informatica, UGent



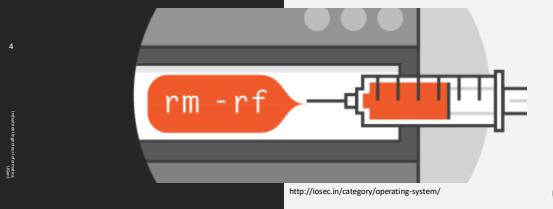
1



3

## Injectie

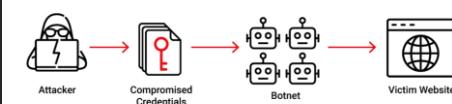
- Injecteren van SQL, NoSQL, OS, LDAP, JS ...
  - Geïnterpreteerd als deel van een opdracht
  - Uitvoeren van ongewilde commando's
  - Toegang toe data zonder de juiste autorisatie
  - Bv. Cross-site Scripting



4

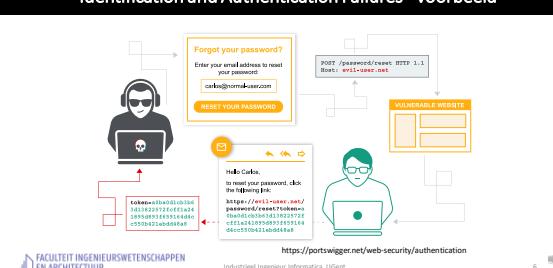
## Identification and Authentication Failures

- Foute implementatie authenticatie en sessiebeheer
  - Wachtwoorden, sleutels of sessietokens besmetten
  - Tijdelijk of permanent zich voordoen als een andere gebruiker
  - Gebruik standaard frameworks



5

Identification and Authentication Failures - voorbeeld



6

## 8.2. BEVEILIGINGSRISICO'S

## HOOFDSTUK 8. WEBAPPLICATIES

### Cryptographic Failures

The diagram shows a user entering a credit card number into a form. A malicious insider steals 4 million credit card numbers from logs. Logs are accessible to IT staff for debugging purposes. The logs contain sensitive information like credit card details because the merchant gateway is unavailable. This leads to a list of risks:

- Gevoelige data niet voldoende afschermen
- Data stelen of aanpassen
- Fraude met kredietkaarten, stelen identiteit, ...
- Oude of zwakke cryptografische algoritmes

<https://www.slideshare.net/appsec/18-owasp-top-10-sensitve-data-exposure>

7

### Insecure Design

The slide discusses common mistakes in design:

- Fouten in ontwerp
- Beveiligingsrisico's niet meenemen in ontwerp
- Niet testen op gekende aanvalsmethodes

Industriel Ingenieur Informatica, UGent

8

### Broken Access Control

The diagram shows a user bypassing restrictions to gain admin privileges. It includes a screenshot of a login page and a database dump showing user credentials.

- Beperkingen voor gebruikers worden niet juist afgedwongen
- Toegang tot andere accounts, gevoelige files, data aanpassen, rechten aanpassen, ...

<https://medium.com/@deepakdramz/bug-bounty-for-beginners-part-2-broken-access-control-775f5fd1927>

Industriel Ingenieur Informatica, UGent

9

### Security Misconfiguration

The diagram shows a central configuration node connected to a web server, database server, and application server. It highlights that misconfiguration happens when no secure configuration has been applied to these components.

- Onveilige standaardconfiguraties, onvolledige configuraties, foutboodschappen met te veel informatie, ...
- Niet gereeld upgraden

FACULTY INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR <https://insights.securecodewarrior.com/coders-conquer-security-share-learn-series-security-misconfigurations/>

Industriel Ingenieur Informatica, UGent

10

### Server-side request forgery (SSRF)

The diagram shows a client sending forged requests to a server. It includes a screenshot of an exploit payload and a list of risks:

- Een onveilige server gebruiken om toegang te krijgen tot interne data
- Bv. URLs als parameter in antwoord

<https://www.mafsee.com/blog/other-blog/microsoft-server-side-request-forgery-takes-down-the-worlds-largest-social-network/>

FACULTY INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR

11

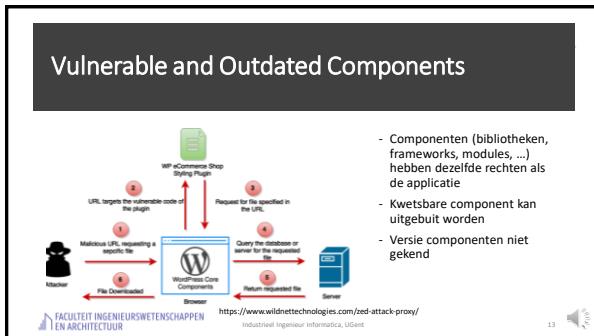
### Software and Data Integrity Failures

The diagram shows a flowchart of a deserialization process. It highlights risks related to using untrusted data and failing to validate inputs.

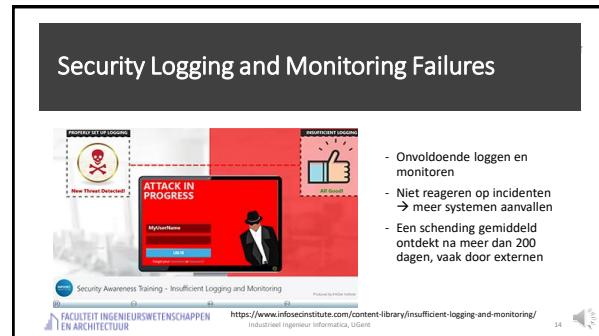
Bibliotheken van niet-vertrouwde bronnen  
Onveilige CI/CD pijplijnen  
Automatische updates die niet voldoende beveiligd zijn  
Onveilig deserialiseren → uitvoeren van code of aanvallen

Industriel Ingenieur Informatica, UGent

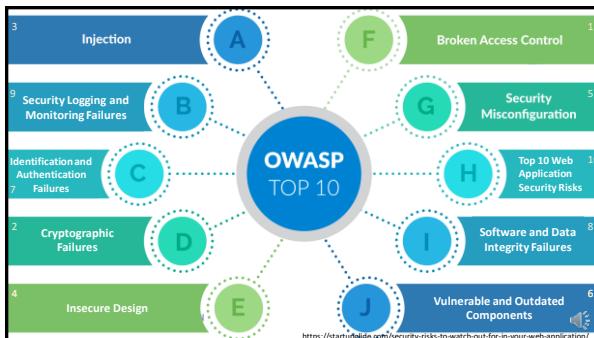
12

**HOOFDSTUK 8. WEBAPPLICATIES****8.2. BEVEILIGINGSRISICO'S**

13



14



15

### 8.3. VERSCHILLEnde TYPES WEBAPPS

### HOOFDSTUK 8. WEBAPPLICATIES

## 8.3 Verschillende types webapps

Webapplicaties - verschillende types webapps

Veerle Ongenaee

1

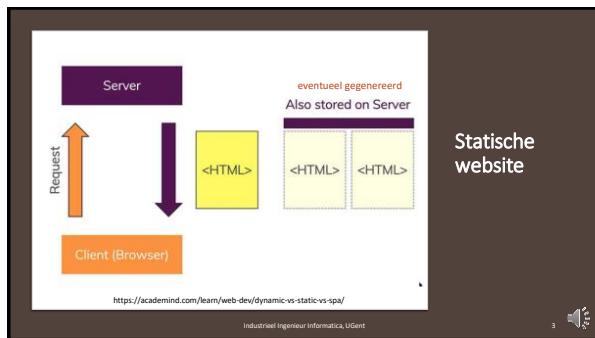
### Verschillende types webapps

- Statische website
- Dynamische websites
- Single Page Applications (SPA)
- Progressive web apps

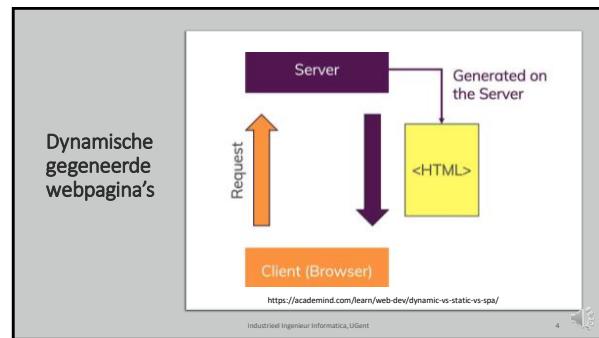
[https://www.youtube.com/watch?v=Kg0Q\\_YaQ3Gk](https://www.youtube.com/watch?v=Kg0Q_YaQ3Gk) Dynamic Websites vs Static Pages vs Single Page Apps (SPAs)  
[https://www.youtube.com/watch?v=F\\_BYg2QGsCO](https://www.youtube.com/watch?v=F_BYg2QGsCO) SPAs vs MPAs/MVC - Are Single Page Apps always better?  
<https://www.youtube.com/watch?v=Z8MjdQGyfA> What is a progressive web app in 60 seconds!

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

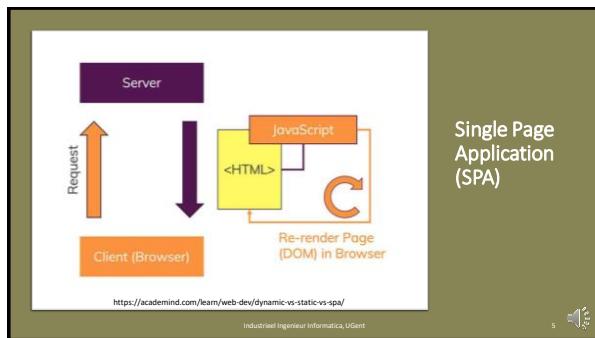
2



3



4



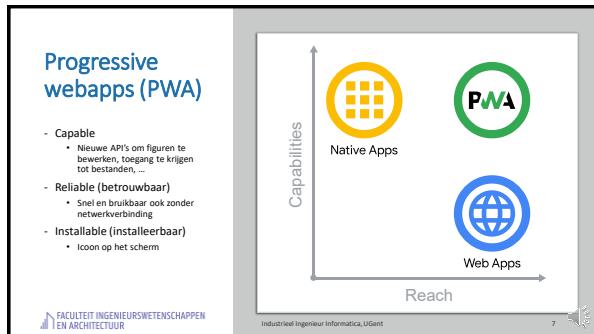
5

| Dynamische webpagina's                                                | SPA                                                                               | Statische webpagina's                                                            |
|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| De volledige HTML-pagina wordt doorgestuurd, ideaal voor zoekmachines | Reageert zeer snel, lijkt op mobiele apps                                         | De volledige HTML-pagina wordt doorgestuurd, ideaal voor zoekmachines            |
| Al het zware werk gebeurt op de server                                | Niet wachten op laden van de pagina                                               | Geen performantieproblemen. De HTML wordt niet gegenereerd.                      |
| Elke nieuwe pagina moet geladen worden                                | De inhoud van de pagina wordt gemaakt in de browser, moeilijker voor zoekmachines | Niet geschikt voor snel veranderende inhoud                                      |
| Moeilijkere splitsing tussen frontend en backend ontwikkeling         | De browser moet meer "zware" werk doen                                            | Het opzetten van een efficient proces om pagina's te genereren kan moeilijk zijn |

P R O C O N T R A

FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR  
Industriel Ingenieur Informatica, UGent

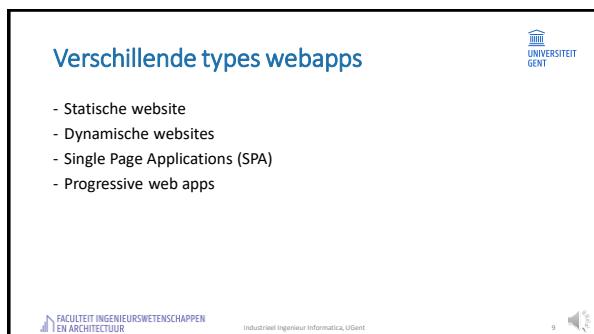
6

**HOOFDSTUK 8. WEBAPPLICATIES****8.3. VERSCHILLELENDE TYPES WEBAPPS**

7



8



9



# Bibliografie

- [AA13a] Scott W. Ambler and Associates. Mapping objects to relational databases: O/r mapping in detail. *Agile Data*, 2002-2013. <http://www.agiledata.org/essays/mappingObjects.html>.
- [AA13b] Scott W. Ambler and Associates. The object-relational impedance mismatch. *Agile Data*, 2002-2013. <http://www.agiledata.org/essays/impedanceMismatch.html>.
- [ADO] Ado.net overview. <http://msdn.microsoft.com/en-us/library/h43ks021%28v=VS.100%29.aspx>.
- [Das08] Aditi Das. Understanding jpa, part 1: The object-oriented paradigm of data persistence. *JavaWorld*, January 2008. <http://www.javaworld.com/article/2077817/java-se/understanding-jpa-part-1-the-object-oriented-paradigm-of-data-persistence.html>.
- [Fis99] Maydene Fisher. The jdbc tutorial: Chapter 3 - advanced tutorial. <http://java.sun.com/developer/Books/JDBCTutorial/index.html>, May 1999.
- [JPAa] The java ee 6 tutorial: Chapter 32 introduction to the java persistence api. <http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>.
- [JPAb] Package javax.persistence. <http://docs.oracle.com/javaee/6/api/javax/persistence/package-summary.html>.
- [JPAc] Hibernate and java persistence api (jpa). <http://thecafetechno.com/tutorials/hibernate/hibernate-and-java-persistence-api-jpa/>.