

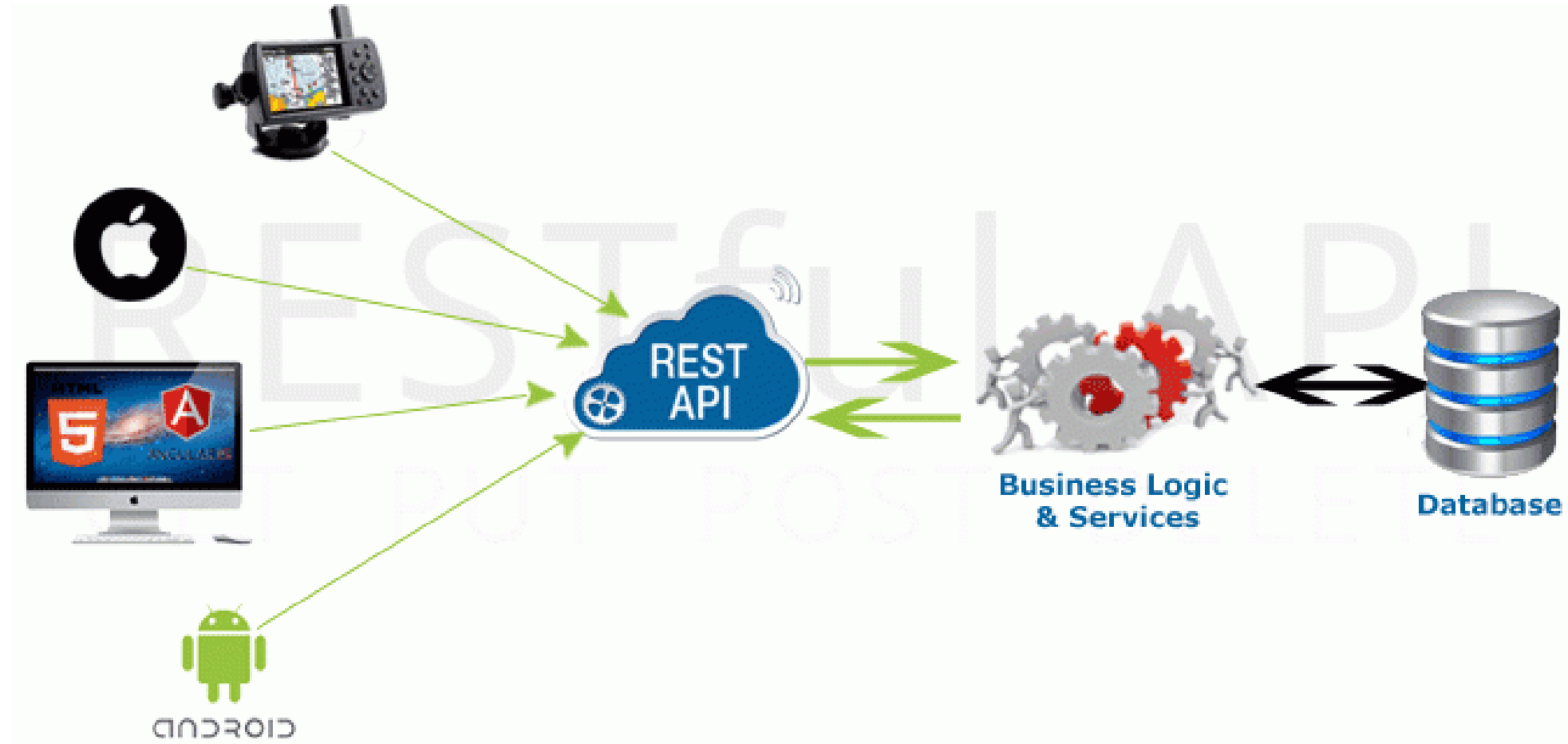


# REST-webservices : Java

Veerle Ongenae



# REST-webservice



# REST-webservice in de praktijk

## Client

- AJAX (XMLHttpRequest)
- Fetch API

## Service in Java

- JAX-RS (Java API for RESTful Web Services)
- Spring REST



# Spring

Framework Professionele Java-programma's

JVM (=container voor Java-objecten)

- Consoleprogramma's
- Testen
- Webserver
- Databank
- ...

Verschillende bibliotheken om sneller toepassingen te bouwen

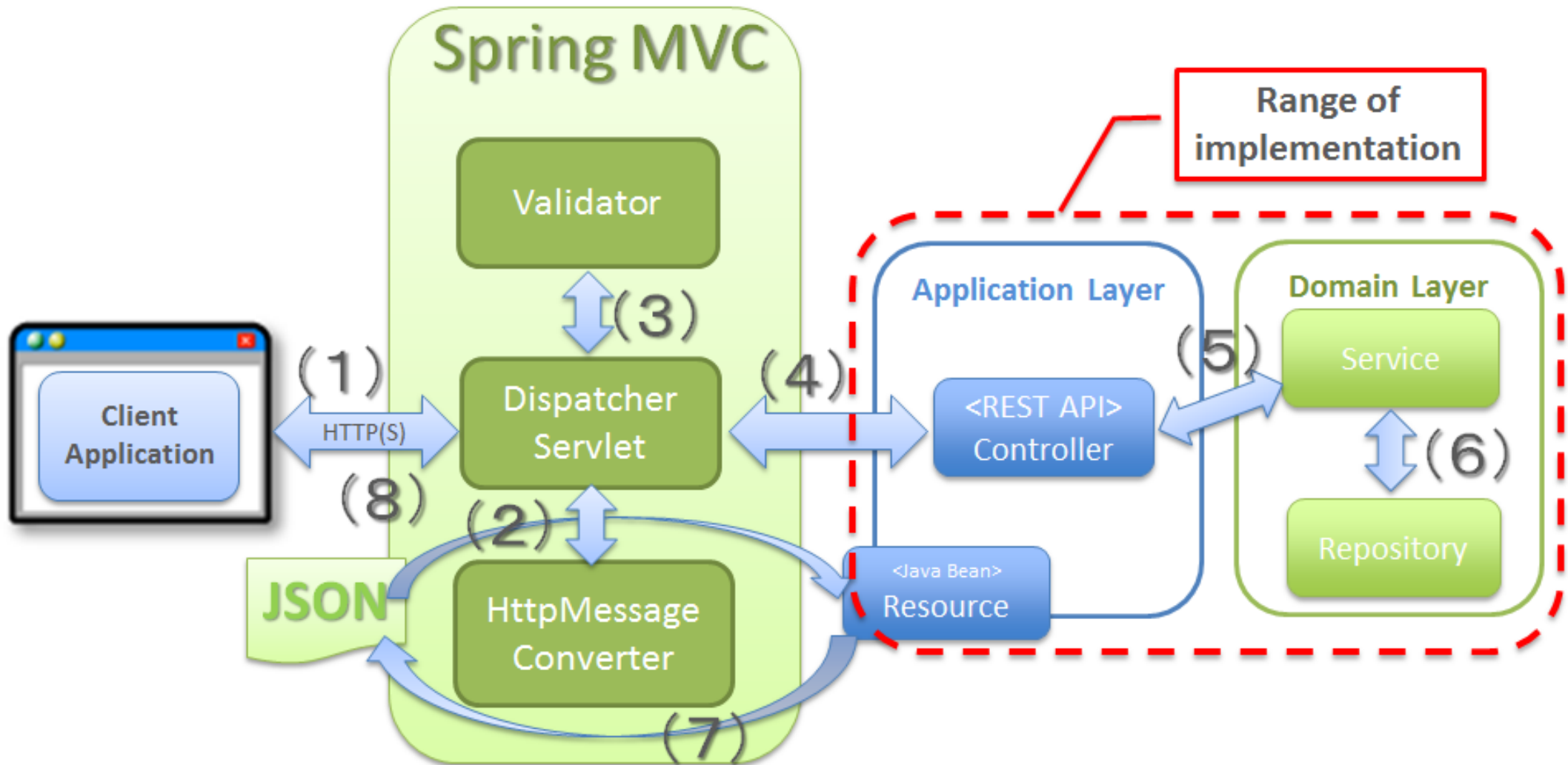


# Springboot

- Sneller projecten maken → templates om nieuwe projecten te genereren (<https://start.spring.io/>)
- Veel configuratie vermijden
- Makkelijk afhankelijkheden beheren



# Architectuur REST API in Spring



# Architectuur REST API in Spring

Spring framework

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
5. In de controller worden van de achterliggende logica opgeroepen
6. De logica interageert met de data laag
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

Implementatie



# 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





# Voorbeeld REST - GET

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

```
[  
  {  
    "id": 3,  
    "onderwerp": "Bjork",  
    "startDatum": "2019-11-13T17:00:00.000+0000",  
    "eindDatum": "2019-11-13T23:00:00.000+0000",  
    "locatie": "Vorst, Brussel"  
  },  
  {  
    "id": 2,  
    "onderwerp": "Film",  
    "startDatum": "2019-11-20T19:00:00.000+0000",  
    "eindDatum": "2019-11-20T20:00:00.000+0000",  
    "locatie": "Gent"  
  },  
  {  
    "id": 1,  
    "onderwerp": "Brunch",  
    "startDatum": "2019-11-23T10:00:00.000+0000",  
    "eindDatum": "2019-11-23T11:00:00.000+0000",  
    "locatie": "Lokeren"  
  }  
]
```



# Voorbeeld REST – GET - RestController

- localhost:8080/agenda/

Deze klasse is een RestController

Het pad van deze REST API begint met “agenda”

Deze methode handelt de GET-aanvraag af

```
import org.springframework.web.bind.annotation.*;

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

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

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



# 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; }
```

Constructor injection

```
@Component
public class Agenda {
```

Managed components:

- Beheerde componenten
- Kunnen geïnjecteerd worden
- Meer specifieke componenten
  - @Service (logicalaag)
  - @Repository (datalaag, persistentie)



# Voorbeeld REST – data - Agenda

```
@Component
public class Agenda {

    private final Map<Integer, AgendaItem> agendaItems;
    int huidigeId = 1;
    private final DateFormat datumFormaat;

    public Agenda() {...}

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

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

    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 opvullen() {...}
}
```



# Voorbeeld REST – data - AgendaItem

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



# Spring REST: Klasse annoteren

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

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



# Spring REST: methode annoteren

## - @GetMapping

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

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

```
[  
  {  
    "id": 3,  
    "onderwerp": "Bjork",  
    "startDatum": "2019-11-13T17:00:00.000+0000",  
    "eindDatum": "2019-11-13T23:00:00.000+0000",  
    "locatie": "Vorst, Brussel"  
  },  
  {  
    "id": 2,  
    "onderwerp": "Film",  
    "startDatum": "2019-11-20T19:00:00.000+0000",  
    "eindDatum": "2019-11-20T20:00:00.000+0000",  
    "locatie": "Gent"  
  },  
  {  
    "id": 1,  
    "onderwerp": "Brunch",  
    "startDatum": "2019-11-23T10:00:00.000+0000",  
    "eindDatum": "2019-11-23T11:00:00.000+0000",  
    "locatie": "Lokeren"  
  }  
]
```



# 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





# Voorbeeld REST - GET

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

```
{  
  "id": 2,  
  "onderwerp": "Film",  
  "startDatum": "2019-11-20T19:00:00.000+0000",  
  "eindDatum": "2019-11-20T20:00:00.000+0000",  
  "locatie": "Gent"  
}
```

```
[  
  {  
    "id": 2,  
    "onderwerp": "Film",  
    "startDatum": "2019-11-20T19:00:00.000+0000",  
    "eindDatum": "2019-11-20T20:00:00.000+0000",  
    "locatie": "Gent"  
  },  
  {  
    "id": 1,  
    "onderwerp": "Brunch",  
    "startDatum": "2019-11-23T10:00:00.000+0000",  
    "eindDatum": "2019-11-23T11:00:00.000+0000",  
    "locatie": "Lokeren"  
  }  
]
```



# Voorbeeld REST – GET - RestController

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

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

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



# Methodes annoteren

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

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

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



# Bronnen filteren: @RequestParam

## paginering

```
GET /devices?startIndex=0&size=20  
GET /configurations?startIndex=0&size=20  
GET /orders?limit=25&offset=50
```

querystring

## filteren

```
GET /calendar/2022-08-08/events?duration=30  
GET /calendar/2014-08-08/events?category=appointments  
  
GET /employees?lastName=Smith&age=30
```

```
@GetMapping("employees")  
public List<Employee> searchEmployees(@RequestParam("lastname") String name,  
    @RequestParam("age") int age) {  
    ...  
}
```



# @RequestParam: optioneel - default

```
@GetMapping("employees")
public List<Employee> searchEmployees(
    @RequestParam(name="lastname",required="false") String name,
    @RequestParam(name="age", defaultValue="21") int age) {
    ...
}
```

naam parameter  
vereist? standaard true  
standaardwaarde

```
@GetMapping("employees")
public List<Employee> searchEmployees(
    @RequestParam("lastname") Optional<String> name,
    @RequestParam("age") int age) {
    ...
}
```

optioneel



# 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



# Voorbeeld REST - POST

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

2	Film	2019-11-20T19:00:00.000+0000	2019-11-20T20:00:00.000+0000	Gent
1	Brunch	2019-11-23T10:00:00.000+0000	2019-11-23T11:00:00.000+0000	Lokeren
3	Bjork	2019-11-13T17:00:00.000+0000	2019-11-13T23:00:00.000+0000	Vorst, Brussel

Start:

Einde:

Onderwerp:

Locatie:



# 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  
Meegegeven als parameter van de methode





# 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;
}
```



# Voorbeeld REST - PUT

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

2	Film	2019-11-20T19:00:00.000+0000	2019-11-20T20:00:00.000+0000	Gent
1	Brunch	2019-11-23T10:00:00.000+0000	2019-11-23T11:00:00.000+0000	Lokeren
3	Bjork	2019-11-13T17:00:00.000+0000	2019-11-13T23:00:00.000+0000	Vorst, Brussel

Start:

Einde:

Onderwerp:

Locatie:

Id:



# Voorbeeld REST – PUT - RestController

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

Deze methode van de RestController  
handelt de PUT-aanvraag af

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

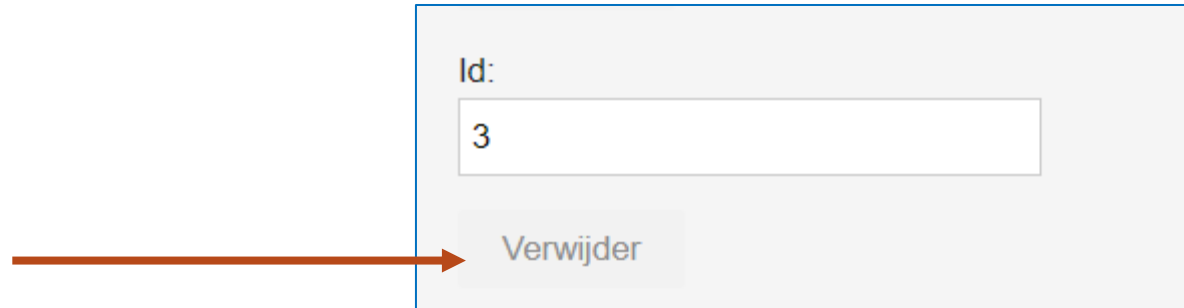
Laatste deel pad wordt geïnterpreteerd als  
een variabele,  
Kan meegegeven worden als parameter van  
de methode

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



# Voorbeeld REST - DELETE

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



Id:  
3

Verwijder



# 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



# Spring REST: methode annoteren

- **@GetMapping, PostMapping, PutMapping, DeleteMapping**
  - Annotatie methode
  - Methode handelt HTTP-berichten met methode ... af
  - Resultaat methode automatisch geconverteerd naar JSON
- Parameters methode
  - Annotatie bepaalt oorsprong parameter
  - **@RequestBody**
    - Body HTTP-request → Java-object
      - Let op namen properties/eigenschappen JSON-object
  - **@PathVariable**
    - Parameter(s) uit deelpad → tussen { ... }
  - **@RequestParam**
    - Parameter(s) uit querystring → URL?{...}



# REST-webservice maken

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



# 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





# 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

Foutklasse koppelen aan HTTP-statuscode

Enum-type met verschillende statuscodes

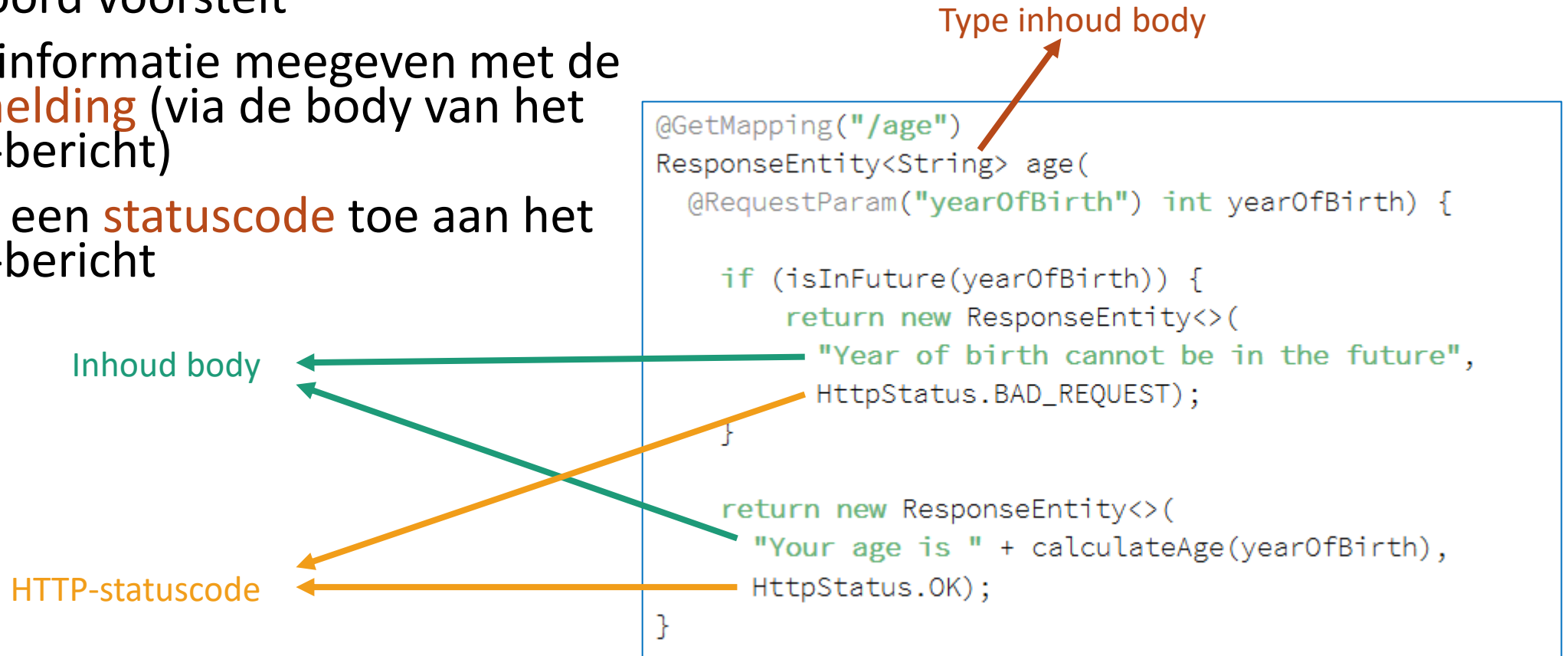
```
@ResponseStatus(code = HttpStatus.BAD_REQUEST)  
class CustomException extends RuntimeException {}
```

bron: <https://www.baeldung.com/spring-response-status>



# Statuscode via ResponseEntity

- **ResponseEntity** = object dat HTTP-antwoord voorstelt
- Extra informatie meegeven met de **foutmelding** (via de body van het HTTP-bericht)
- Voegt een **statuscode** toe aan het HTTP-bericht



bron: <https://www.baeldung.com/spring-response-entity>




# Statuscode koppelen aan methode

- **Statuscode** koppelen aan methode RestController

Alternatief @PostMapping

HTTP-statuscode

```
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>



# 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**



# 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



# URI (link) toevoegen in antwoordbericht



URI aanmaken  
op basis van het huidig pad

Location-header toevoegen aan  
HTTP-antwoord

```
@PostMapping("/jpa/users/{username}/todos")
public ResponseEntity<Void> createTodo(
    @PathVariable String username, @RequestBody Todo todo){

    todo.setUsername(username);

    Todo createdTodo = todoJpaRepository.save(todo);

    //Location
    //Get current resource url
    ///{id}
    URI uri = ServletUriComponentsBuilder.fromCurrentRequest()
        .path("/{id}").buildAndExpand(createdTodo.getId()).toUri();

    return ResponseEntity.created(uri).build();
}
```

```
@RequestMapping(method = RequestMethod.POST)
public ResponseEntity<> createCustomer(UriComponentsBuilder b) {

    UriComponents uriComponents =
        b.path("/customers/{id}").buildAndExpand(id);

    return ResponseEntity.created(uriComponents.toUri()).build();
}
```

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-spring/44751260>



# 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

