Java

Les 10

jeroen.devos01@ap.be

AP HOGESCHOOL ANTWERPEN

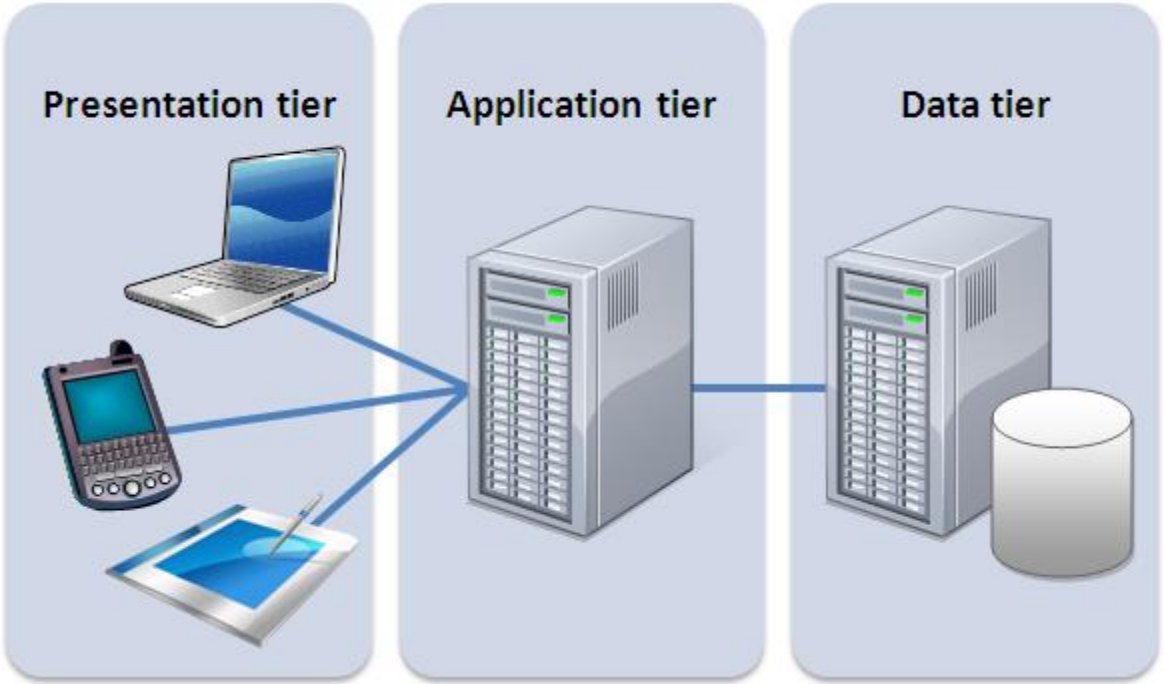
AP.BE

Lesdoelen

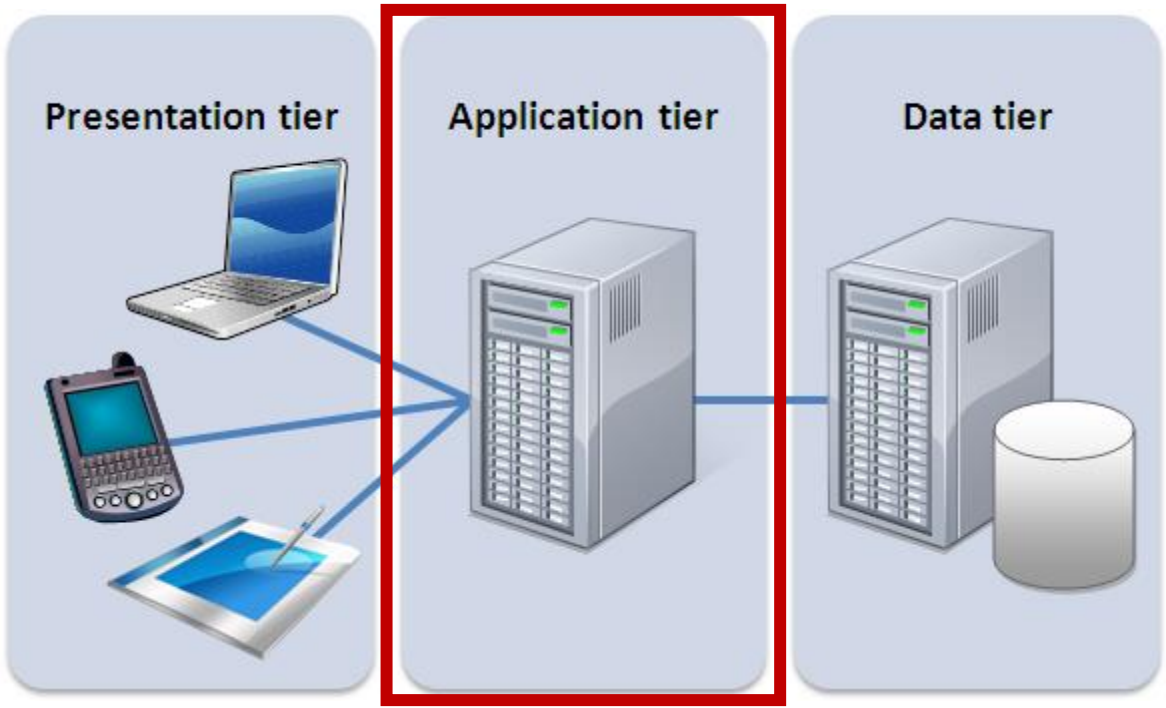
- API & REST
- JAX-RS
- Jersey
- JSON & Jackson
- Jetty

Advanced Java™

Java in het werkveld – three tier architecture

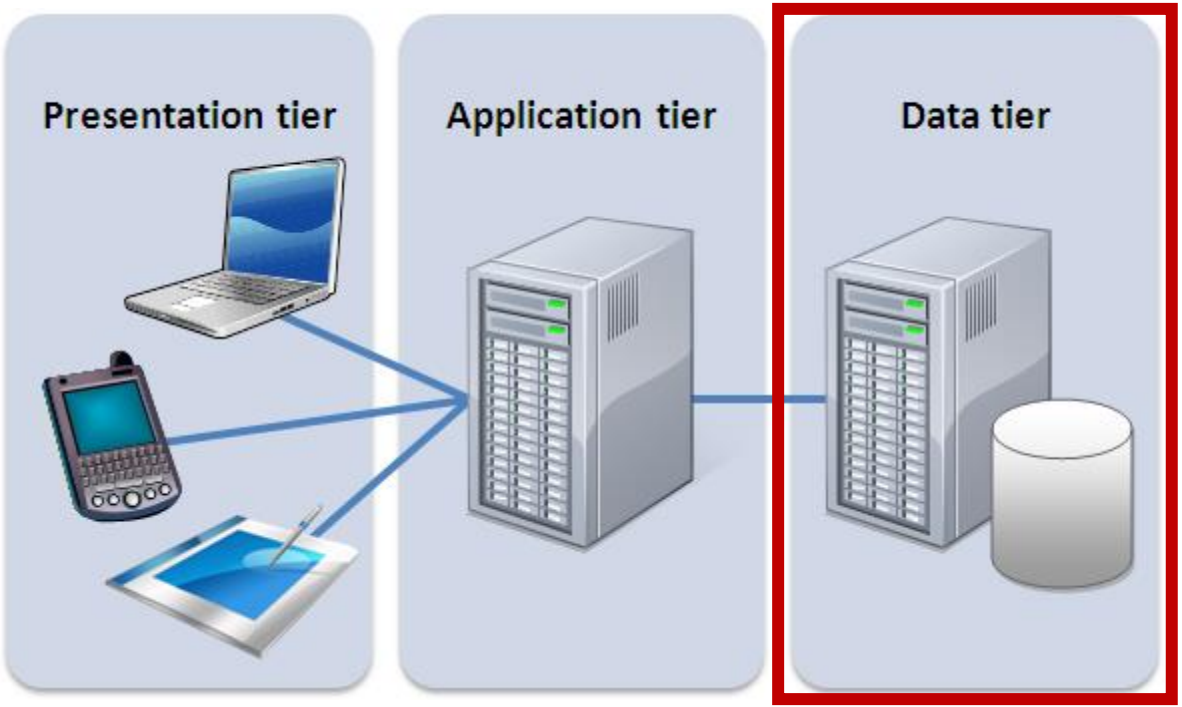


Java in het werkveld – three tier architecture



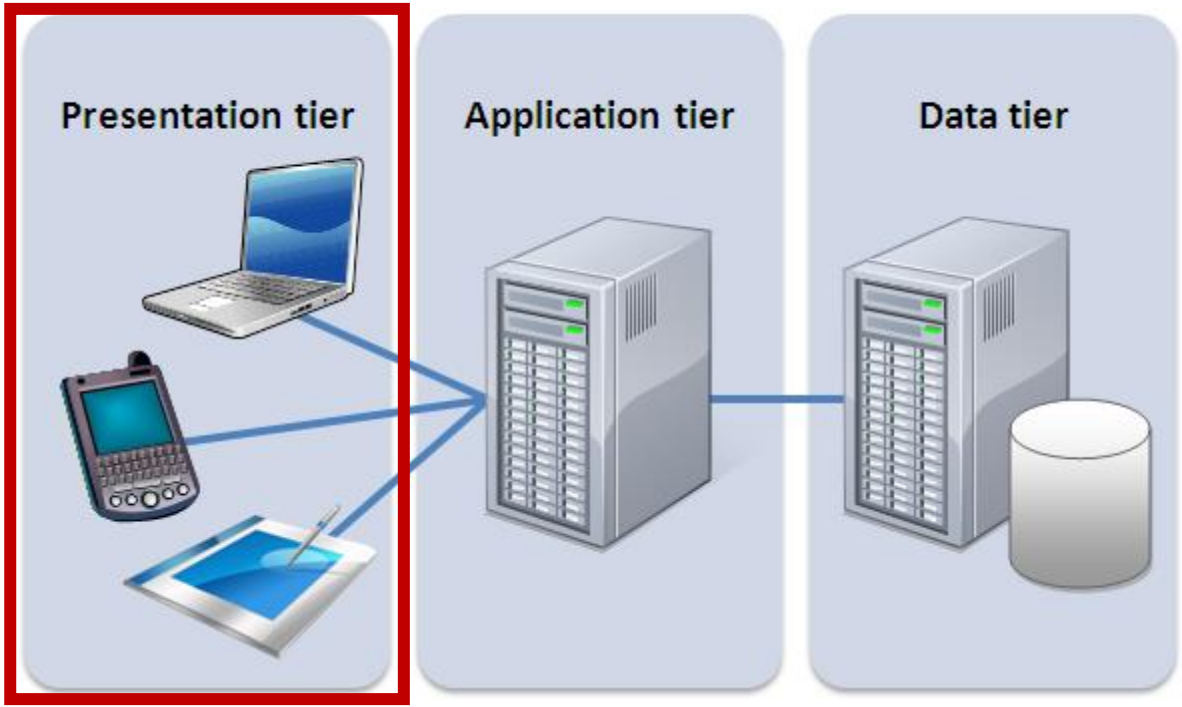
Java server

Java in het werkveld – three tier architecture



Java server Database

Java in het werkveld – three tier architecture

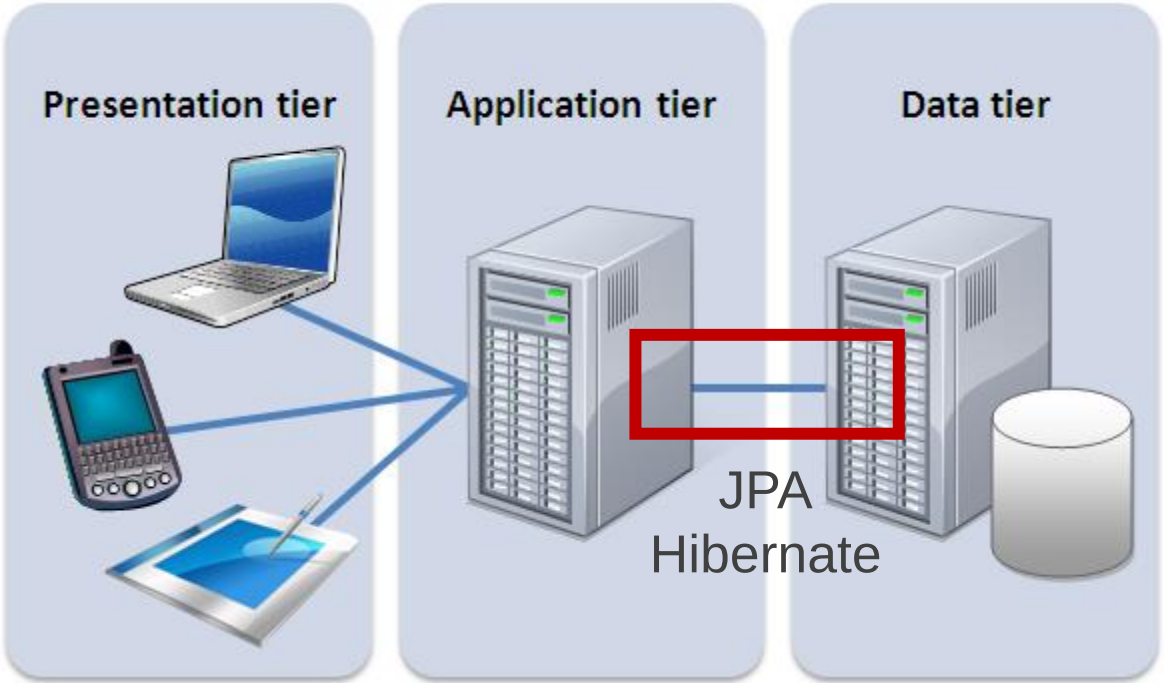


Web/mobile framework

Java server

Database

Java in het werkveld – three tier architecture

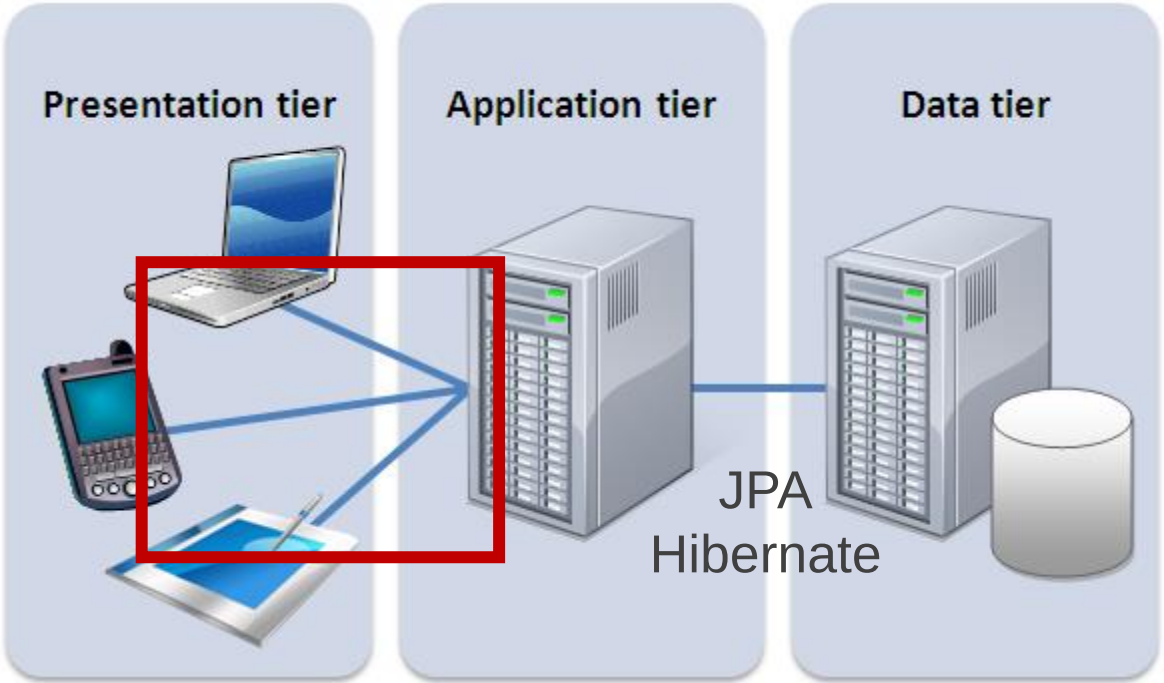


Web/mobile framework

Java server

Database

Java in het werkveld – three tier architecture



Web/mobile framework

Java server

Database

Theorie – API

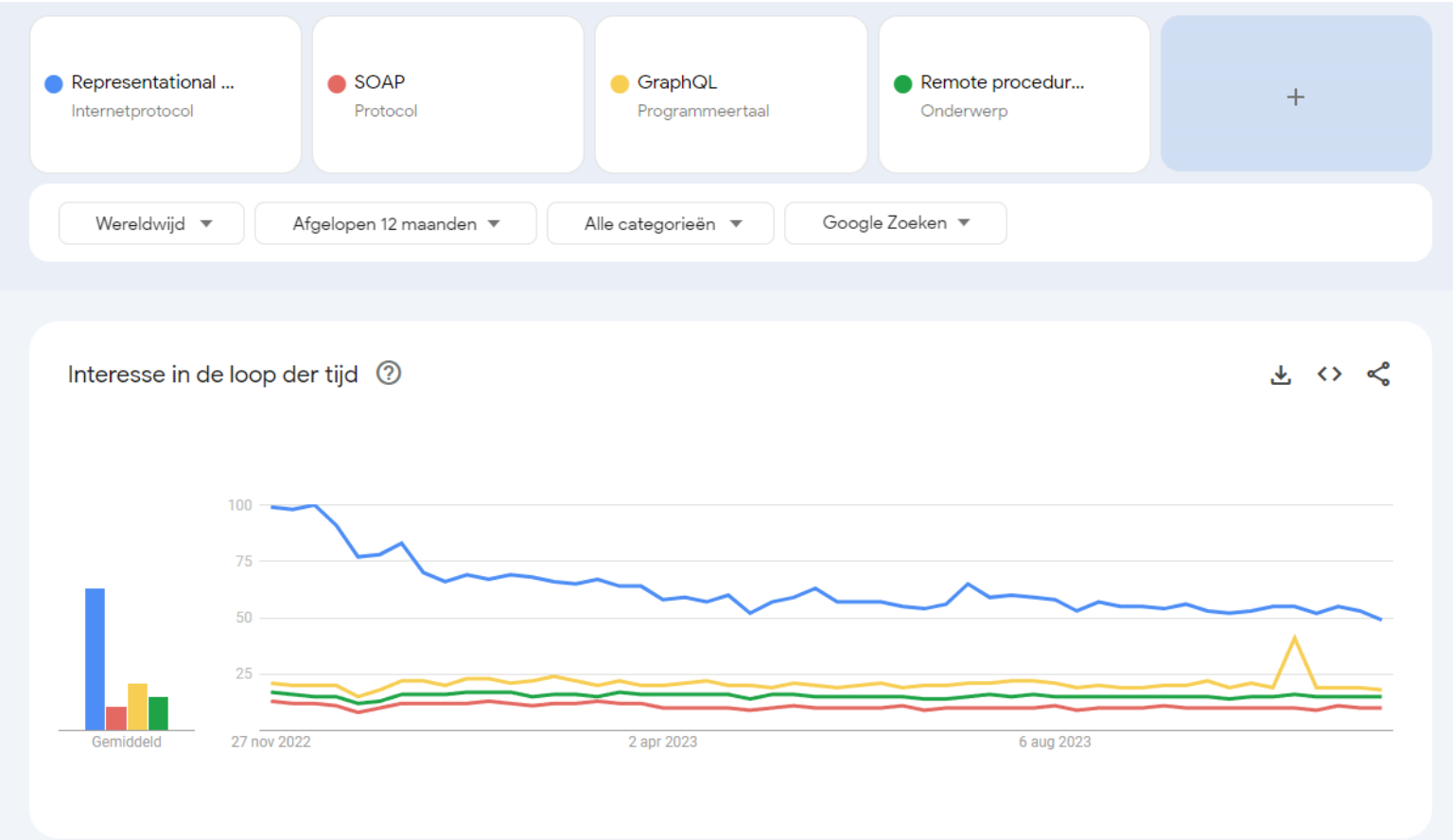
Application **P**rogramming **I**nterface

- Bepaalt hoe een systeem met een ander systeem communiceert.

- < > **UI** / **U**ser **I**nterface : bepaalt hoe een gebruiker met een systeem communiceert.

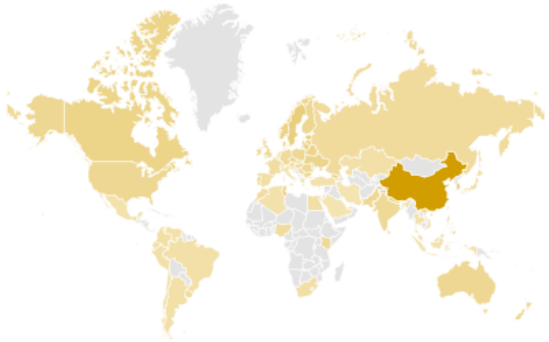
- Geen programmeertaal of architectuur!
- Verzamelnaam voor de software die zorgt voor de communicatie.

- API architecturen : REST, SOAP, GraphQL, RPC …


Representational ...
Internetprotocol
SOAP
Protocol
GraphQL
Programmeertaal
Remote procedur...
Onderwerp
+
Wereldwijd
Afgelopen 12 maanden
Alle categorieën
Google Zoeken
Interesse in de loop der tijd
100
75
50
25
Gemiddeld
27 nov 2022
2 apr 2023
6 aug 2023

GraphQL

Interesse per regio | Regio



Regio's met laag zoekvolume opnemen

Gerelateerde zoekopdrachten | Stijgend

1	graphql query is unauthorized	Snelle stijger
2	graphql query is unauthorized facebook	Snelle stijger
3	graphql query unauthorised	Snelle stijger
4	la consulta de graphql no está autorizada	Snelle stijger
5	nieautoryzowane zapytanie graphql facebo...	Snelle stijger

1-5 van 18 zoekopdrachten tonen

Theorie – REST

REpresentational **S**tate **T**ransfer

- Architectuur
- Wordt gebruikt om API's te bouwen -> REST API of RESTful API
- API voldoet aan REST design principes
  - Client-server
  - Uniform interface
  - Stateless
  - Cacheable
  - Layered application
  - Code On Demand (optional)

Theorie – REST design principles

Client-server

Client en server moeten onafhankelijk van elkaar zijn. De client moet enkel maar weten hoe het de server moet contacteren (URL).

Uniform interface

Er moet een standaard set van operaties voorzien worden zoals bv HTTP GET , POST , PUT en DELETE

Theorie – REST design principles

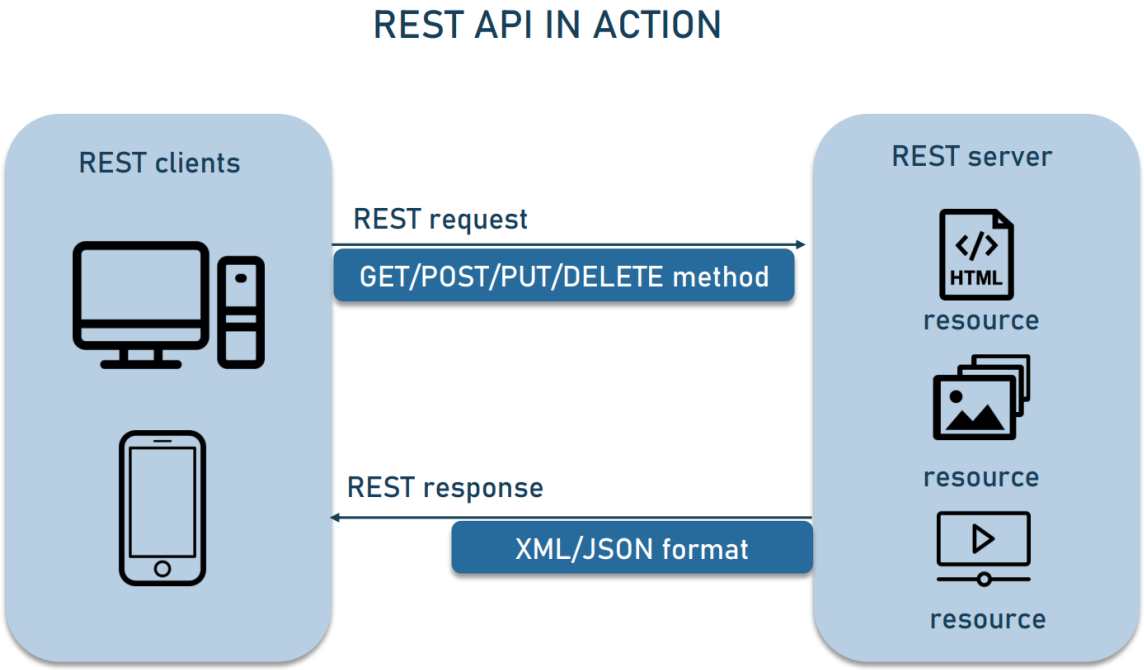
Stateless

Elke vraag (request) aan de server staat op zichzelf, de server mag geen ‘state’ bijhouden tussen requests.

Cacheable

De client kan een antwoord cachen (typisch gedrag van een web browser). Het antwoord (response) van de server moet kunnen aangeven of dit mag gecached worden of niet.

Theorie – REST request - response



altexsoft

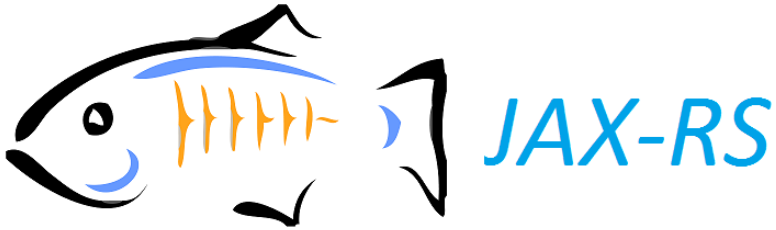
Theorie – JAX-RS

Java **A**PI for **R**ESTful Web **S**ervices

Nieuwe naam : Jakarta RESTful Web Services

= specificatie, geen implementatie !

- Beschrijft REST voor Java Objecten
- Maakt vrijwel uitsluitend gebruik van annotations

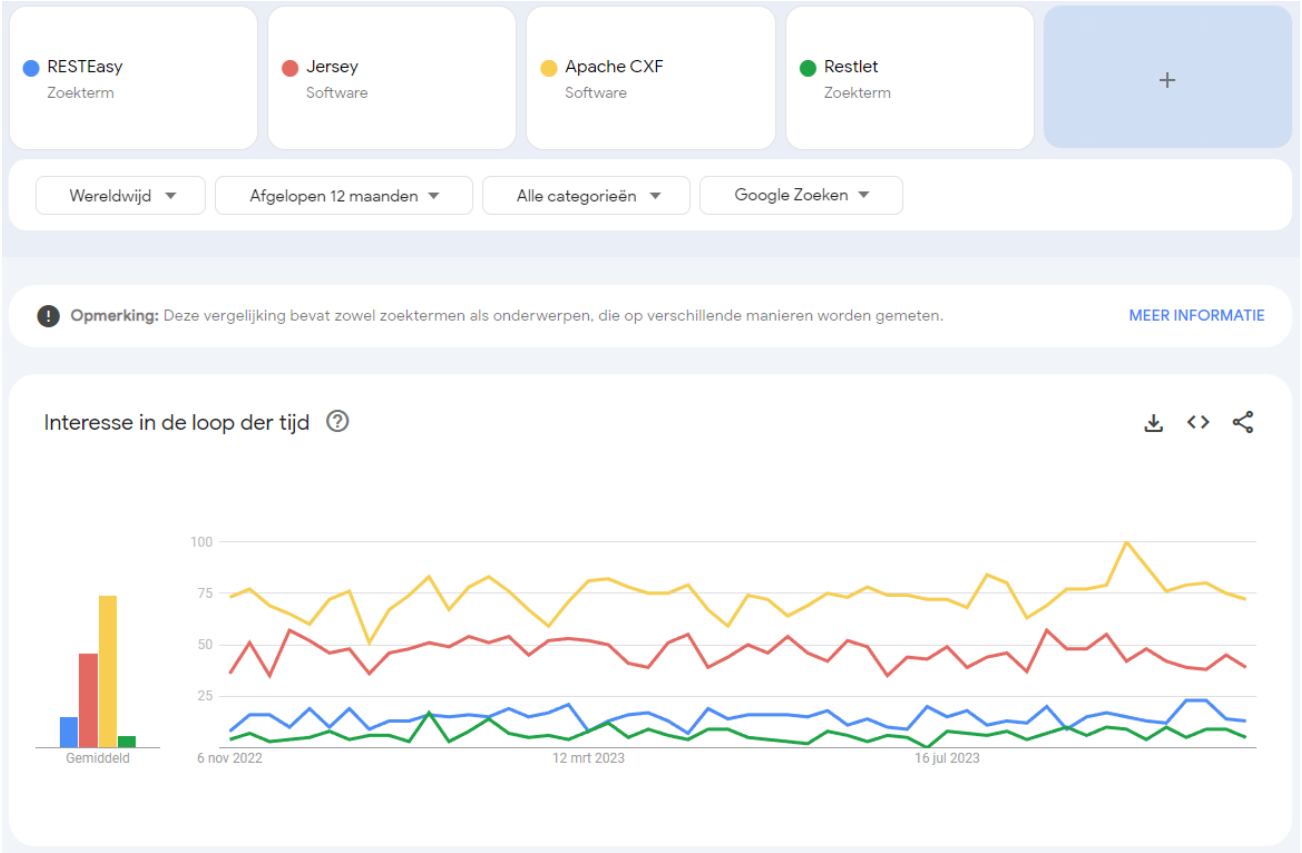


Theorie – JAX-RS implementaties

Bekende JAX-RS implementaties:

- RESTEasy (JBoss)
- Jersey (Eclipse)
- Apache CXF (= framework met meer dan JAX-RS)
- Restlet

Theorie – JAX-RS implementaties



Jersey

Jersey

- Open source
- Eclipse Foundation
- Implementeert JAX-RS

→ API's schrijven

https://eclipse-ee4j.github.io/jersey/

Jersey Maven dependency

```
<dependency>
   <groupId>org.glassfish.jersey.core</groupId>
   <artifactId>jersey-server</artifactId>
   <version>3.1.3</version>
</dependency>
```

- Alle functionaliteiten om RESTful server code te schrijven

@Path

```
@Path(“/demo”)
public class DemoController {

}
```

- @Path annotation maakt van een **class** een REST resource
- Heeft één parameter: relatieve URL -> https://localhost:8080/demo

@GET

```
@Path(“/demo”)
public class DemoController {
   @GET
   public String helloWorld() {
      return “hello world”;
   }
}
```

- @GET annotation maakt van een **method** een REST GET request
- HTTP GET https://localhost:8080/demo

@GET + @Path

```
@Path(“/demo”)
public class DemoController {
   @GET
   @Path(“/hello”)
   public String helloWorld() {
      return “hello world”;
   }
}
```

- @Path annotation op method : relatieve URL
- HTTP GET https://localhost:8080/demo**/hello**

@GET + @Path + @PathParam

```
@Path(“/demo”)
public class DemoController {
   @GET
   @Path(“/hello/{name}”)
   public String hello(@PathParam(“name”) String name) {
      return “hello ” + name;
   }
}
```

- @PathParam annotation in method : parameter van URL lezen / in combinatie met @Path waar de parameter tussen { } moet staan
- HTTP GET https://localhost:8080/demo/hello/**jeroen**

@GET + @QueryParam

```
@Path(“/demo”)
public class DemoController {
   @GET
   @Path(“/hello”)
   public String hello(@QueryParam(“firstname”) String first, @QueryParam(“lastname”) String last) {
      return “hello ” + first + “ “ + last;
   }
}
```

- @QueryParam annotation in method : query parameter van URL lezen
- HTTP GET https://localhost:8080/demo/hello**?firstname=jeroen&lastname=devos**

@GET + @Produces

```
@Path(“/demo”)
public class DemoController {
   @GET
   @Path(“/hello”)
   @Produces(MediaType.TEXT_PLAIN)
   public String helloWorld() {
      return “hello world”;
   }
}
```

- @Produces annotation op method : datatype van response
  - MediaType.TEXT_PLAIN
  - MediaType.APPLICATION_JSON

JSON

Java**S**cript **O**bject **N**otation

- Standaard formaat voor uitwisseling van data
- Heeft niets met JavaScript te maken…
- Taal-onafhankelijk tekstformaat
- Bestaat uit key-value koppels
- Datatype : string, number, boolean
- Ondersteunt ook arrays

JSON

```
{
   “firstName” : “Luke”,
   “lastName” : “Skywalker”,
   “homePlanet” : “Tatooine”,
   “age” : 19,
   “jedi” : true,
   “friends” : [“Han”, “Chewie”, “Leia”]
}
```

Jersey media Maven dependency

```
<dependency>
   <groupId>org.glassfish.jersey.media</groupId>
   <artifactId>jersey-media-json-jackson</artifactId>
   <version>3.1.3</version>
</dependency>
```

- Voegt ondersteuning voor JSON toe

Jackson annotations

- **@JsonProperty(“myPropertyName”)**

De key van de JSON zal deze waarde hebben. Geef je deze annotation niet mee, dan is de key de naam van het attribuut.

Tip : schrijft steeds de annotation zelfs indien de naam hetzelfde blijft.

- **@JsonIgnore**

Dit attribuut van het Java object zal niet meegegeven worden in de JSON.

Gebruik dit om attributen voor intern gebruik intern te houden.

Jackson voorbeeld

```
public class StarWarsCharacterDTO {
    @JsonProperty(“firstName”)
    private String firstName;
    @JsonProperty(“lastName”)
    private String lastName;
    @JsonProperty(“homePlanet”)
    private String homePlanet;
    @JsonProperty(“age”)
    private Integer age;
    @JsonProperty(“jedi”)
    private Boolean jedi;
   @JsonProperty(“friends”)
    private List<String> friends;
    @JsonIgnore
    private String father;
}
```

@POST + @Path + @Consumes

```
@Path(“/star-wars”)
public class StarWarsController {
   @POST
   @Path(“/”)
   @Consumes(MediaType.APPLICATION_JSON)
   public void setCharacter(StarWarsCharacterDTO dto) {
      System.out.println(dto);
   }
}
```

- @POST stuurt data door in de body van de request.
- Jackson zal JSON automatisch uit de body halen en op het Java object mappen gebruik makkende van de @JsonProperty annotations.

Return void?

```
@Path(“/star-wars”)
public class StarWarsController {
   @POST
   @Path(“/”)
   @Consumes(MediaType.APPLICATION_JSON)
   public void setCharacter(StarWarsCharacterDTO dto) {
      System.out.println(dto);
   }
}
```

- Geen return value?
- Een HTTP request zonder response?

Response

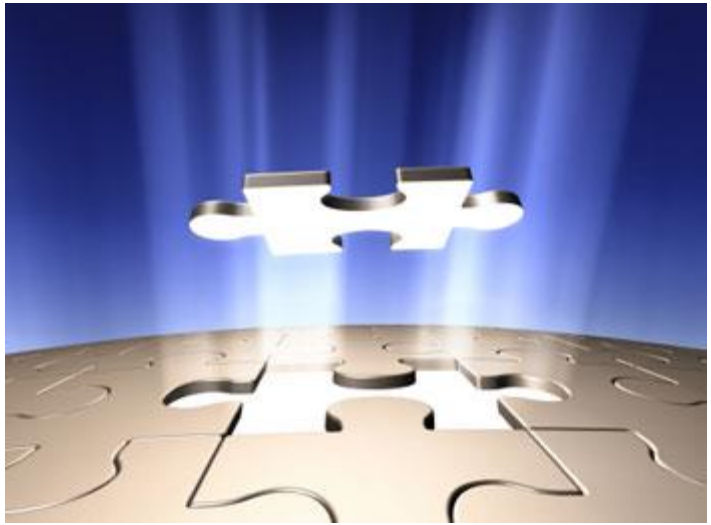
```
@Path(“/star-wars”)
public class StarWarsController {
   @POST
   @Path(“/”)
   @Consumes(MediaType.APPLICATION_JSON)
   public Response setCharacter(StarWarsCharacterDTO dto) {
      System.out.println(dto);
      return Response.ok().build();
   }
}
```

- Response object geef een antwoord terug aan de request
  - ok() = HTTP status 200 ok

ResponseBuilder

- Response…. geeft een ResponseBuilder terug
- Met de builder kan je een volledig eigen response samenstellen:
  - ok() : shortcut voor status(200)
  - status() : een HTTP statuscode meegeven
  - entity() : een object meegeven (bv JSON)
  - cookie() : cookies die geplaatst moeten worden
  - allow() : welke domeinen deze response mogen krijgen
  - cacheControl() : bepaalt hoe de browser moet cachen
  - expires() : tijd hoe lang gecached moet worden

Hoe onze API beschikbaar maken?



Java web server

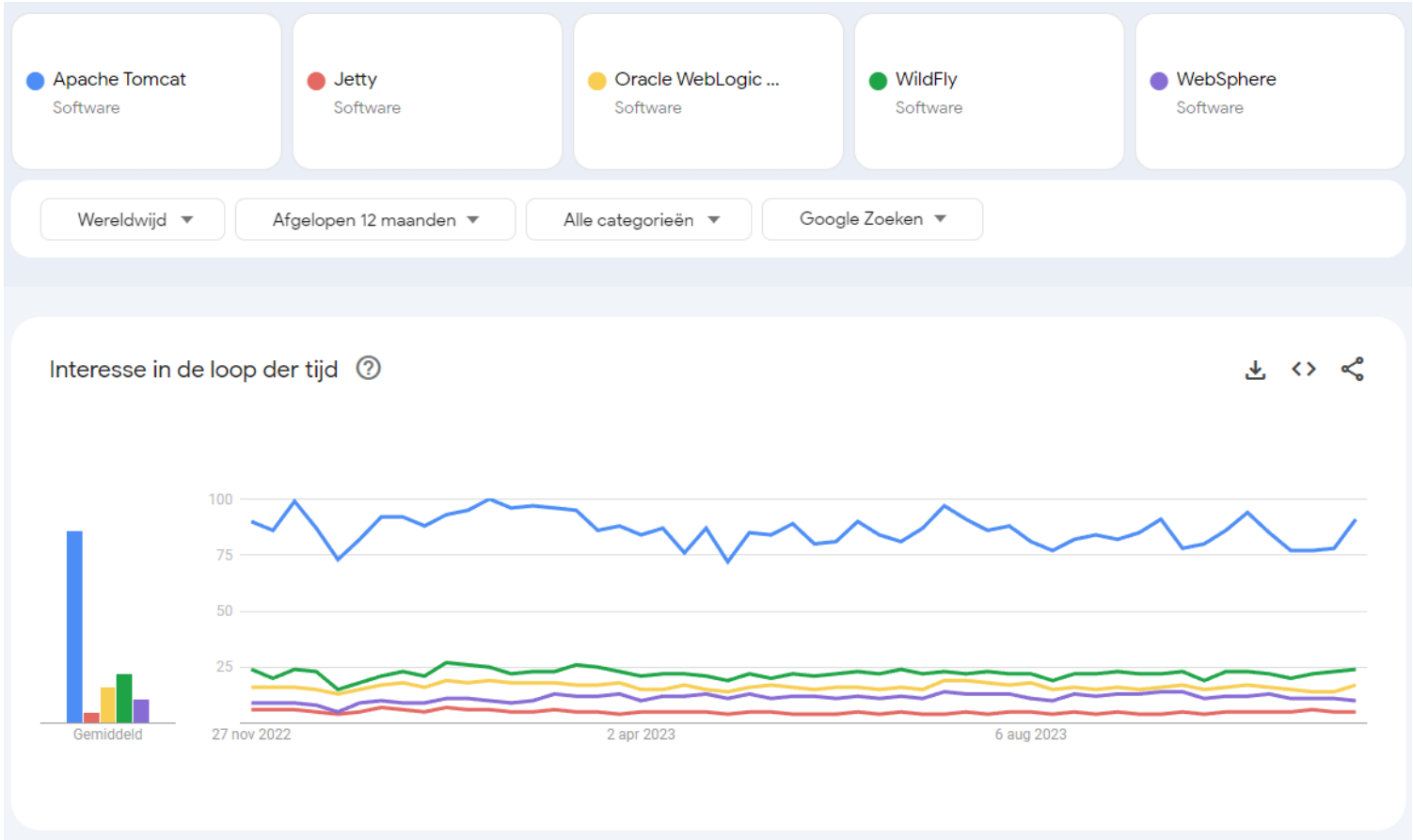
- HTTP communicatie opzetten
- Hosting o.a. van API
- Kan Java Servlets uitvoeren

Een servlet is een Java class die een request kan aanvaarden en een response terugstuurt.

Java web server

Populaire servers:

- Tomcat (Apache)
- Jetty (Eclipse)
- WebLogic (Oracle)
- WebSphere (IBM)
- WildFly (Red Hat)


Apache Tomcat
Software
Jetty
Software
Oracle WebLogic ...
Software
WildFly
Software
WebSphere
Software
Wereldwijd
Afgelopen 12 maanden
Alle categorieën
Google Zoeken
Interesse in de loop der tijd
100
75
50
25
Gemiddeld
27 nov 2022
2 apr 2023
6 aug 2023

Jetty web server

- Open source
- Eclipse foundation

- Lightweight
- Embedded

https://eclipse.dev/jetty/

Jetty Maven dependency

```
<dependency>
   <groupId>org.eclipse.jetty</groupId>
   <artifactId>jetty-server</artifactId>
   <version>11.0.18</version>
</dependency>
<dependency>
   <groupId>org.eclipse.jetty</groupId>
   <artifactId>jetty-servlet</artifactId>
   <version>11.0.18</version>
</dependency>
```

- Activeert de Jetty Server voor je code

Server aanmaken

```
public class JettyServer {
   private Server server;

   public void start() {
      //create server listening on localhost port 8080
      server = new Server(8080);
      //start server
      try {
         server.start();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

En nu alles samen…

jetty://

Extra Maven dependencies

```
<dependency>
   <groupId>org.glassfish.jersey.containers</groupId>
   <artifactId>jersey-container-servlet-core</artifactId>
   <version>3.1.3</version>
</dependency>
<dependency>
   <groupId>org.glassfish.jersey.containers</groupId>
   <artifactId>jersey-container-jetty-http</artifactId>
   <version>3.1.3</version>
</dependency>
<dependency>
   <groupId>org.glassfish.jersey.inject</groupId>
   <artifactId>jersey-hk2</artifactId>
   <version>3.1.3</version>
</dependency>
```

Servlet aanmaken

```
public class JettyServer {
   private Server server;

   public void start() {
      …
      //register API
      ResourceConfig config = new ResourceConfig();
      config.register(HelloWorldController.class); //register more if needed
      //create servlet
      ServletContextHandler handler = new ServletContextHandler();
      handler.addServlet(new ServletHolder(new ServletContainer(config)), “/api/*”);
      server.setHandler(handler);
      …
   }
}
```

ResourceConfig

```
//register API
ResourceConfig config = new ResourceConfig();
config.register(HelloWorldController.class); //register more if needed
```

ResourceConfig : bevat alle REST resources.

Gebruik .register( ) om extra resources toe te voegen.

Opgelet:

- De resource met geldig zijn, dwz minstens @Path en één operatie
- Nieuwe resource aanmaken, vergeet de .register( ) niet

ServletContextHandler

```
//create servlet
ServletContextHandler handler = new ServletContextHandler();
handler.addServlet(new ServletHolder(new ServletContainer(config)), “/api/*”);
server.setHandler(handler);
```

ServletContextHandler : bevat alle servlets, minstens één.

Gebruik .addServlet( ) om servlets aan de handler toe te voegen én je moet het path opgeven waar de servlet op luistert.

Opgelet :

- elke servlet heeft en eigen path nodig!
- Vergeet de server.setHandler( ) niet

Uitvoering

```
public class Main {

    public static void main(String[] args) {
        JettyServer server = new JettyServer();
        server.start();
    }

}
```

Test via browser : http://localhost:8080/api/hello

→ Default gedrag van browser = GET request

Uitvoering – deel 2


FOR TODAY
ANYWAY...