Spring Boot

Olivier Liechti & Laurent Prévost
COMEM Web Services

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

What is Spring Boot?

- It is **kind of** a development platform: it provides high-level **integrations** to develop software components. It's an extension of Spring Framework.
- It is an execution platform: it provides an **embedded** environment to **run** these components "to live".
- It is an entreprise platform: it **allows to use different integration to** support distributed transactions, security, integration, etc.
- Separation of concerns: "The developer takes care of the business logic. **Spring Framework** takes care of the systemic qualities".

http://flickr.com/photos/decade_null/427124229/sizes/m/#cc_license

Spring Boot and standards

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Spring Boot is a facilitator to use one or more libraries from Spring Framework.
  - Like Spring Framework, there is no standards nor specifications about Spring Boot
- Spring Boot and more generally Spring Framework is an “umbrella” of technologies
  - Spring Framework can comply on Java EE specifications and technologies but it does not force to use them.
  - At least, the Servlet Specs is brought to you by Spring Boot.
  - Spring Boot brings to you the tools to make the plumbing between your different application parts.

Architecture

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

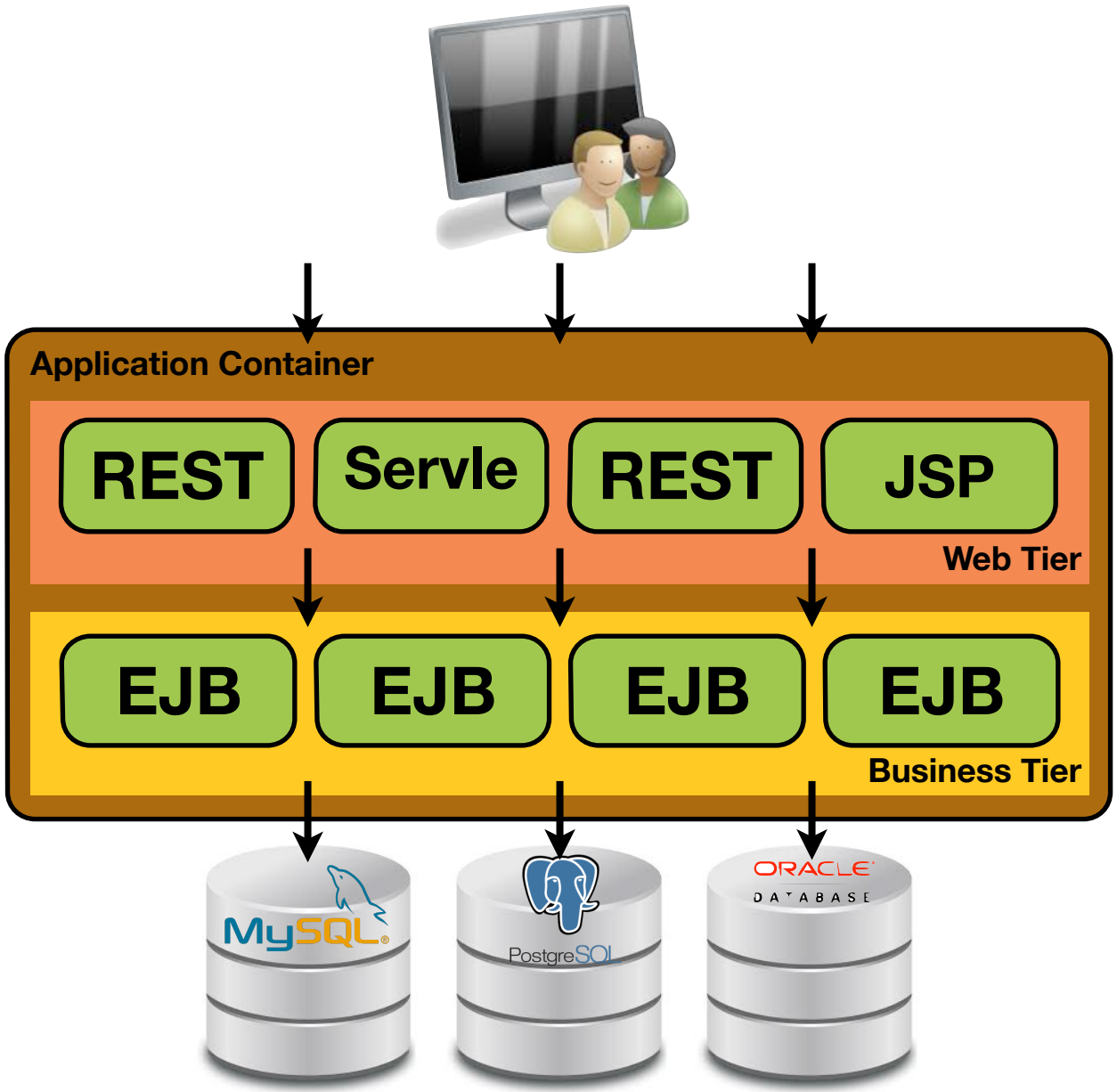
- Spring Boot is a facade for Spring Framework well organized to let you just pick up one Spring Boot starter and then let you build your apps quickly and easily.
  - The application container aka application server is therefore embedded into your application and you have several choice like Tomcat, Jetty or Undertow.
  - Running the application is easy:

```
java -jar jarFile.jar
```

  - Spring Source provide several bindings and integrations with most of the well known technologies:
    - JPA, JMS, Spring Security, ...
- Unlike with Java EE, Spring Boot is the container
  - the container is the **environment** in which we run components;
  - the container **provides services** (transactions, security, etc.) through APIs like Java EE. These libraries are available in the numerous libraries of Spring Framework;
  - Spring Boot offer only kind of a "**web**" container like we find in Java EE.

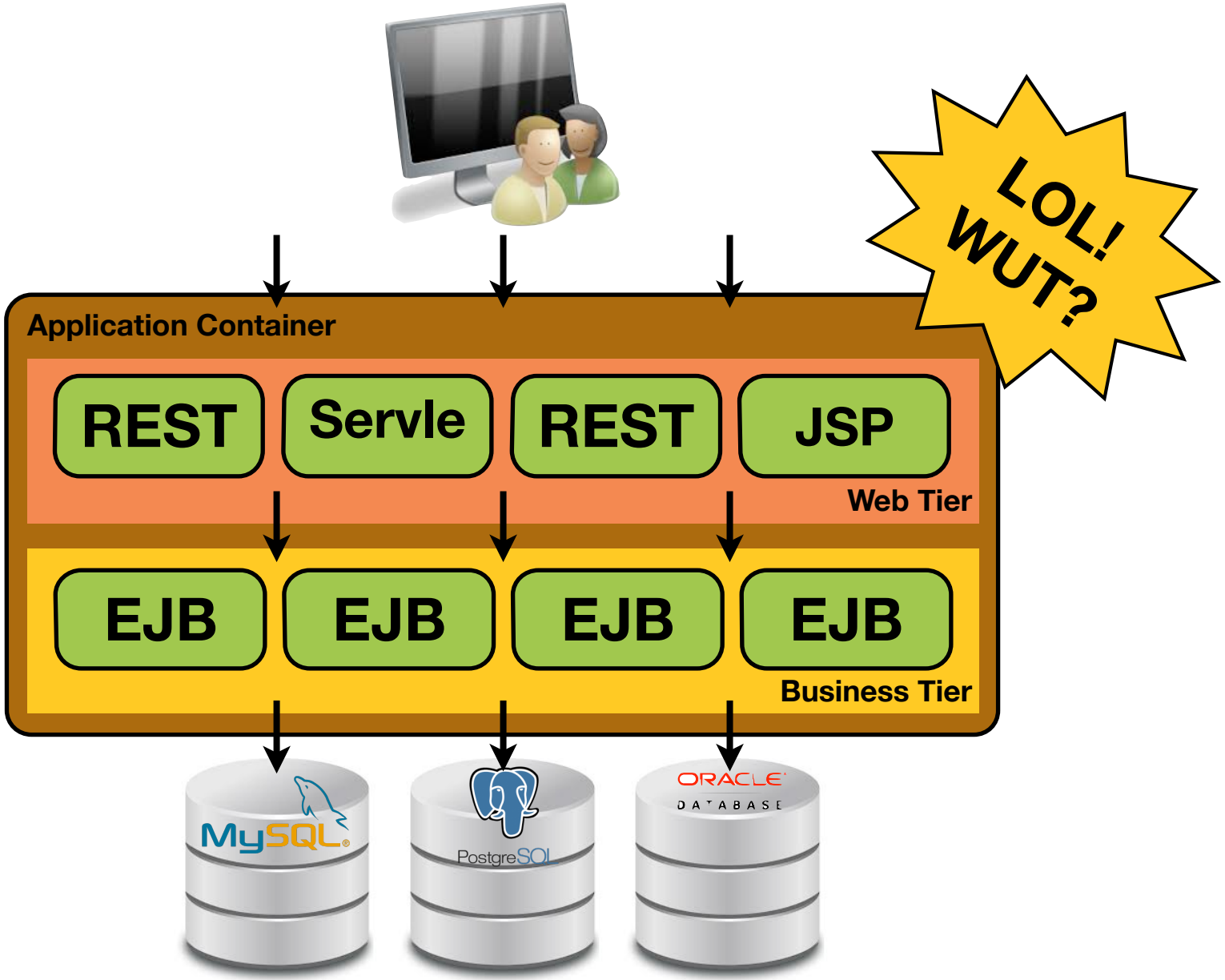
Spring Boot - Tiers

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



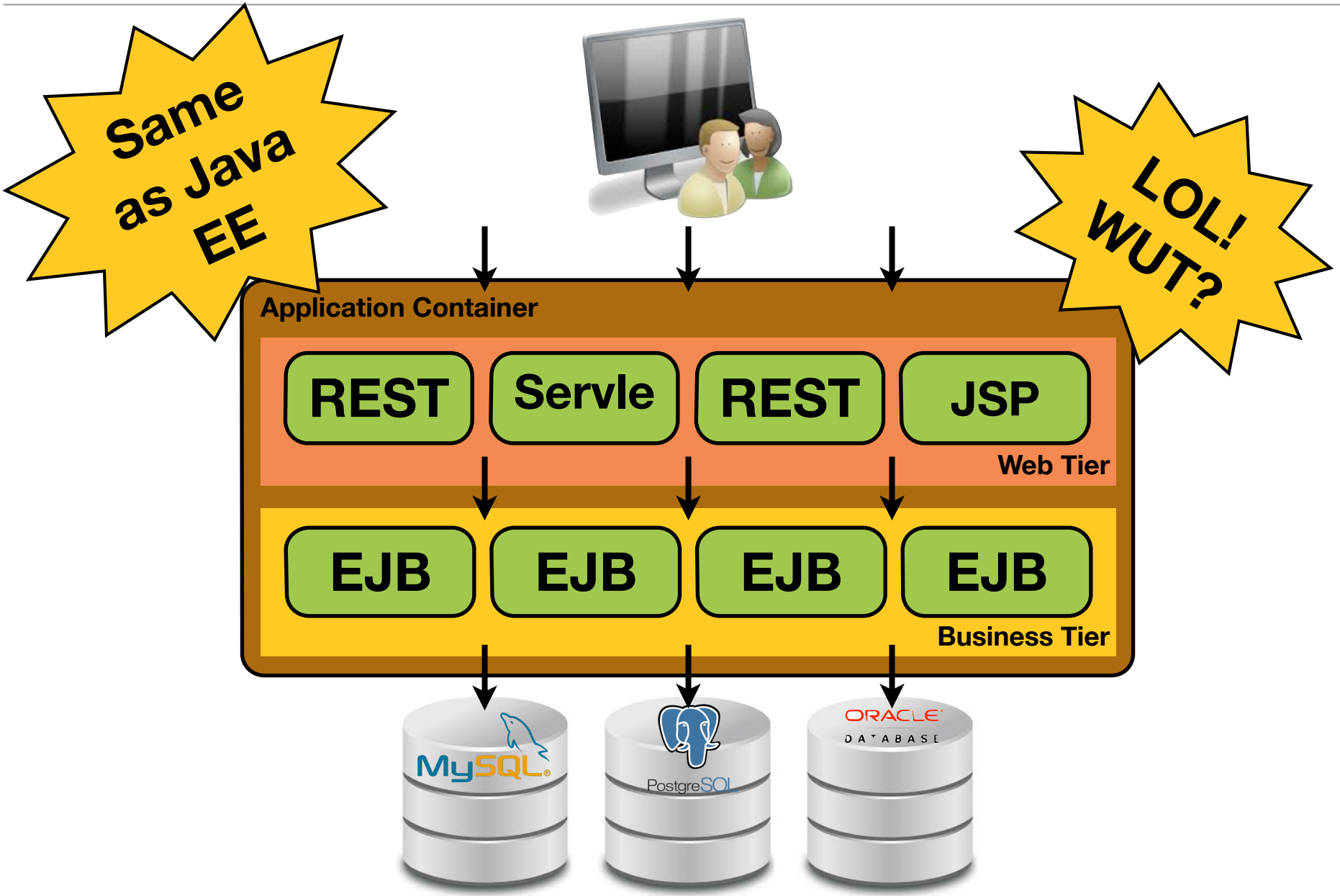
Spring Boot - Tiers

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



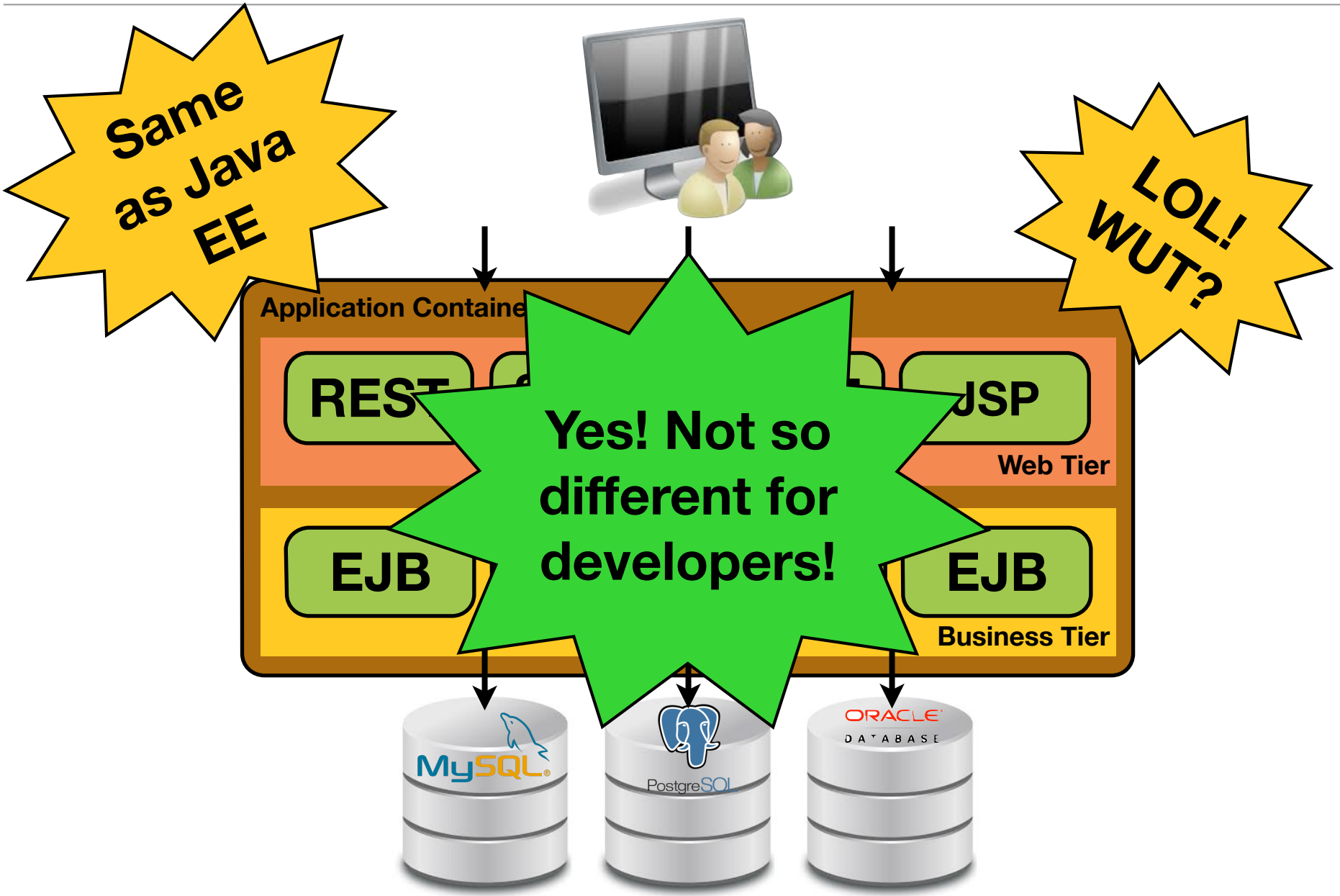
Spring Boot - Tiers

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



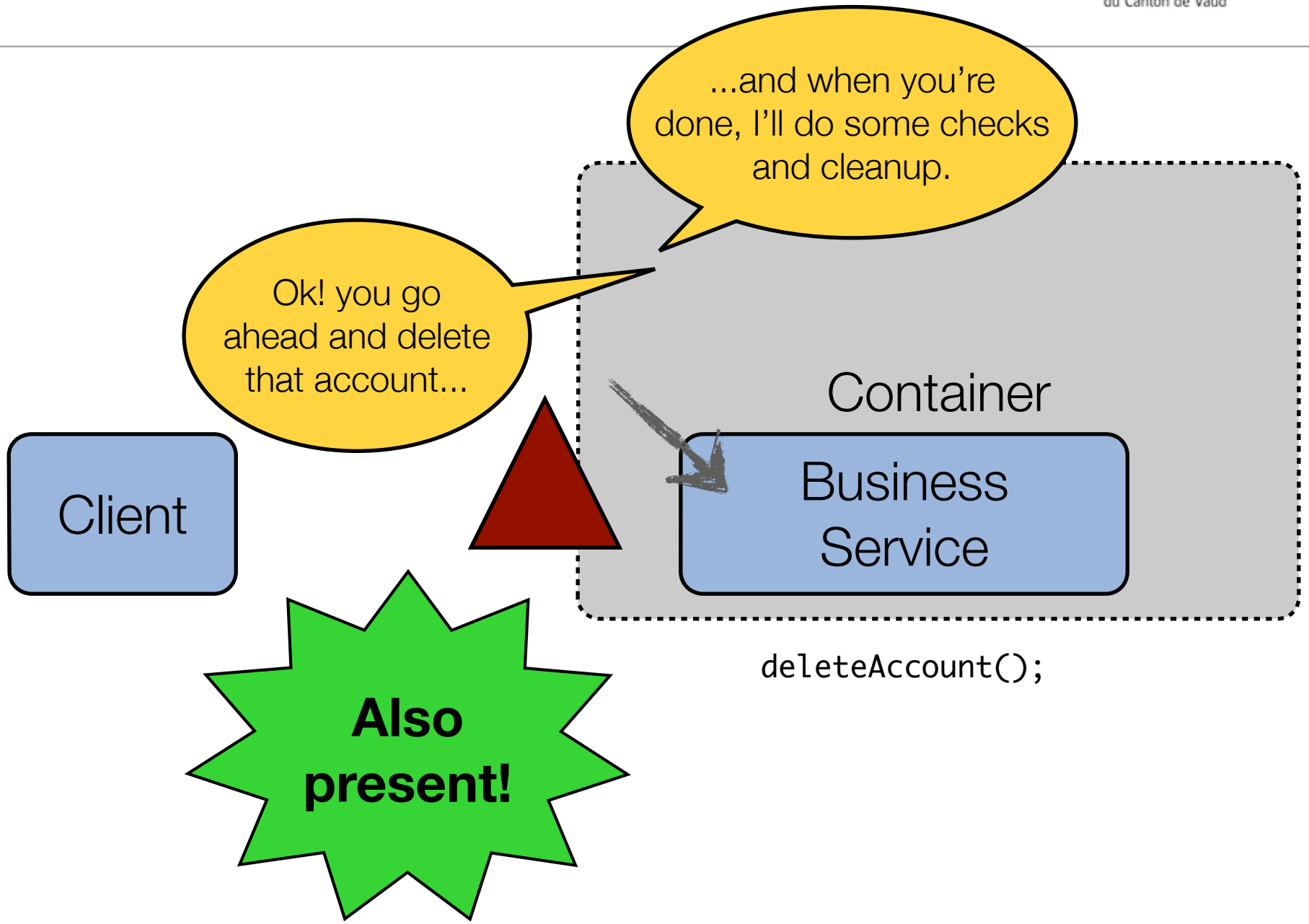
Spring Boot - Tiers

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Mediated access

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud



Why Spring Boot?

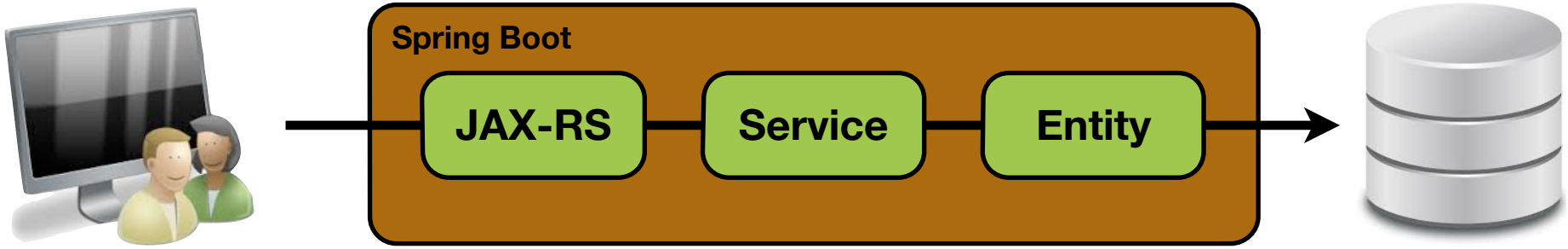
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Java EE need new ideas to enrich his specs
- Developers love alternatives
- Spring Source is defacto a kind standard for various libraries (Spring Framework, Spring Security)
- Offers a lightweight way to develop Web applications
- Allows to integrate various technologies that are not necessary possible when we use Application Servers.
  - Glassfish: Jersey for REST APIs, cannot be changed
  - WildFly: RestEasy for REST APIs, cannot be changed
- Does not require to install and maintain an application server for the devs and ops.
- Really convenient to build POCs.

Spring Boot vs. Java EE

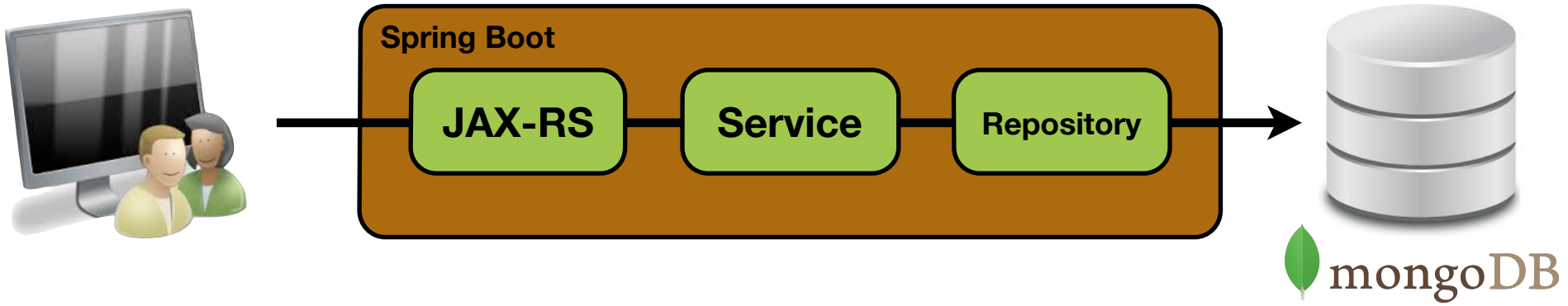
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Are they concurrent or complementary?
  - In fact, they play well together but that really depends what you are building.
  - Example: http://spring.io/blog/2014/11/23/bootiful-java-ee-support-in-spring-boot-1-2
  - Tons of code samples: https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples



Application example

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private String id;

   private String username;

   private Role role;

   @DBRef
   private Address address;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

No more @Entity, welcome @Document specific for MongoDB but same purpose.

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private String id;

   private String username;

   private Role role;

   @DBRef
   private Address address;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

No more @Entity, welcome @Document specific for MongoDB but same purpose.

Also an ID marker to be sure we have a sort of primary key...

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private Str

   private String username;

   private Role role;

   @DBRef
   private Address address;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion

... but not the same package.

No more @Entity, welcome @Document specific for MongoDB but same purpose.

Also an ID marker to be sure we have a sort of primary key...

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private Str

   private String username;

   private Role role;

   @DBRef
   private Address address;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion

... but not the same package.

No more @Entity, welcome @Document specific for MongoDB but same purpose.

Also an ID marker to be sure we have a sort of primary key...

No more way to constrain the length of the field.

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private Str

   private String username;

   private Role role;

   @DBRef
   private Address address;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private Str

   private String username

   private Role role;

   @DBRef
   private Address address;
}
```

... but not the same package.

No more @Entity, welcome @Document specific for MongoDB but same purpose.

Also an ID marker to be sure we have a sort of primary key...

No more way to constrain the length of the field.

And no way to specify how is rendered an enum in the database.

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class User {
   public static enum Role {
    ADMIN,
    MEMBER
   }

   @Id
   private Str

   private String username

   private Role role;

   @DBRef
   private Address address;
}
```

... but not the same package.

No more @Entity, welcome @Document specific for MongoDB but same purpose.

Also an ID marker to be sure we have a sort of primary key...

No more way to constrain the length of the field.

And no way to specify how is rendered an enum in the database.

A way to specify this object refers to another document in MongoDB.

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;

@Document
public class Address {
   @Id
   private String id;
   private String street;
   private String city;
   private int postalCode;
}
```

Entity in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

No additional magic in this model. We retrieve the @Document and @Id markers.

```
package ch.heigvd.ptl.sc.model;

import org.springframework.data.annotation.Id;

@Document
public class Address {
   @Id
   private String id;
   private String street;
   private String city;
   private int postalCode;
}
```

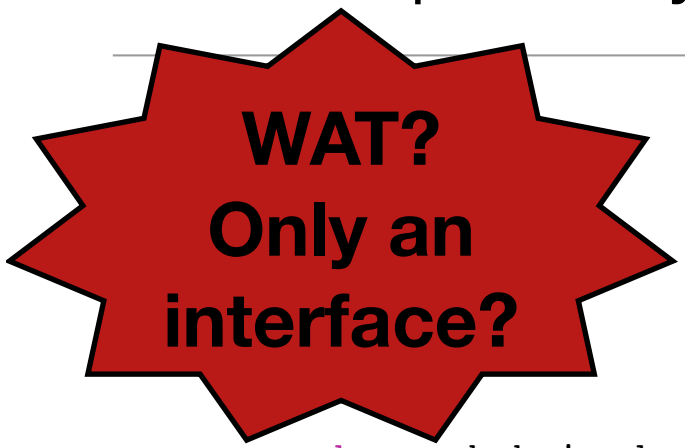
Data Repository (Spring Data) in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.persistence;

import ch.heigvd.ptl.sc.model.User;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface UserRepository extends MongoRepository<User, String> {
   public List<User> findByUsername(String username);
}
```

Data Repository (Spring Data) in code

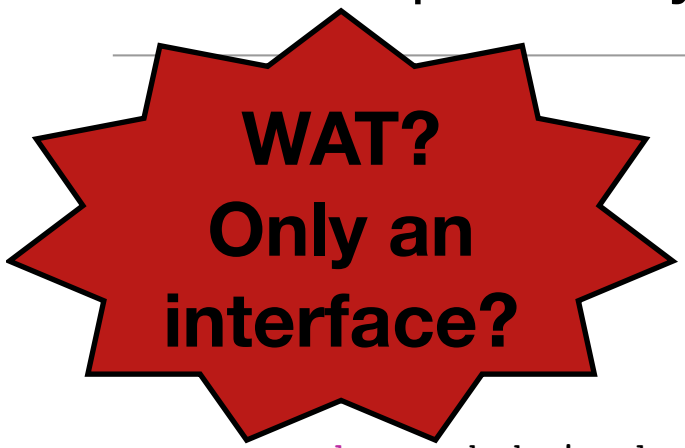
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
package ch.heigvd.ptl.sc.persistence;

import ch.heigvd.ptl.sc.model.User;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface UserRepository extends MongoRepository<User, String> {
   public List<User> findByUsername(String username);
}
```

Data Repository (Spring Data) in code

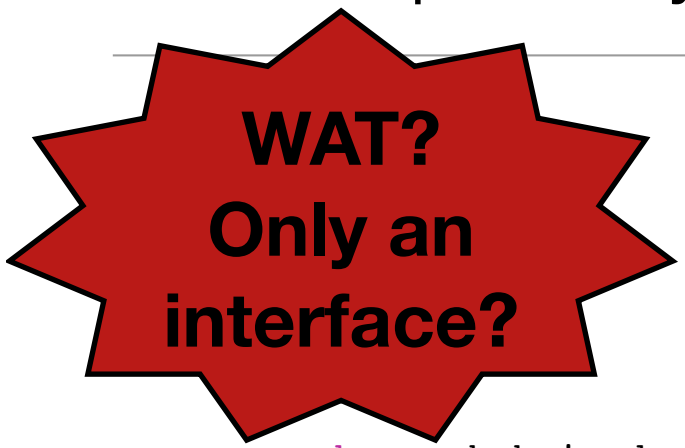
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
package ch.heigvd.ptl.sc.persistence;

import ch.heigvd.ptl.sc.model.User;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface UserRepository extends MongoRepository<User, String> {
   public List<User> findByUsername(String username);
}
```

Data Repository (Spring Data) in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud




```
package ch.heigvd.ptl.sc.persistence;

import ch.heigvd.ptl.sc.model.User;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface UserRepository extends MongoRepository<User, String> {
   public List<User> findByUsername(String username);
}
```

Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.converter;

import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.to.UserTO;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserConverter {
    @Autowired
    private AddressConverter addressConverter;

    public final List<UserTO> convertSourceToTarget(List<User> sources) { ... }
    public final UserTO convertSourceToTarget(User source) { ... }
    public final List<User> convertTargetToSource(List<UserTO> targets) { ... }
    public final User convertTargetToSource(UserTO target) { ... }

    public void fillTargetFromSource(UserTO target, User source) {
     target.setAddress(addressConverter.convertSourceToTarget(source.getAddress()));
     target.setUsername(source.getUsername());
     target.setRole(source.getRole().name());
    }

    public void fillSourceFromTarget(User source, UserTO target) {
     source.setAddress(addressConverter.convertTargetToSource(target.getAddress()));
     source.setUsername(target.getUsername());
    }
}
```

Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.converter;

import ch.heigvd.ptl.sc.model.User;
import ch.heigv
import java.uti
import java.uti
import org.spri                                    utowired;
import org.spri

@Service
public class UserConverter {
    @Autowired
    private AddressConverter addressConverter;

    public final List<UserTO> convertSourceToTarget(List<User> sources) { ... }
    public final UserTO convertSourceToTarget(User source) { ... }
    public final List<User> convertTargetToSource(List<UserTO> targets) { ... }
    public final User convertTargetToSource(UserTO target) { ... }

    public void fillTargetFromSource(UserTO target, User source) {
     target.setAddress(addressConverter.convertSourceToTarget(source.getAddress()));
     target.setUsername(source.getUsername());
     target.setRole(source.getRole().name());
    }

    public void fillSourceFromTarget(User source, UserTO target) {
     source.setAddress(addressConverter.convertTargetToSource(target.getAddress()));
     source.setUsername(target.getUsername());
    }
}
```

Make sure that Spring plumbing see this class as a service that we can inject elsewhere in the application.

Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.converter;

import ch.heigvd.ptl.sc.model.User;
import ch.heigv
import java.uti
import java.uti
import org.spri                                    utowired;
import org.spri

@Service
public class UserConverter {
    @Autowired
    private Addre  Converter addressConverter;

    public final List                                    ) { ... }
    public final User
    public final List                                    ) { ... }
    public final User

    public void fillTargetFromSource(UserTO target, User source) {
     target.setAddress(addressConverter.convertSourceToTarget(source.getAddress()));
     target.setUsername(source.getUsername());
     target.setRole(source.getRole().name());
    }

    public void fillSourceFromTarget(User source, UserTO target) {
     source.setAddress(addressConverter.convertTargetToSource(target.getAddress()));
     source.setUsername(target.getUsername());
    }
}
```

Make sure that Spring plumbing see this class as a service that we can inject elsewhere in the application.

Spring will inject the corresponding service like we are used to with Java EE and @EJB annotation. This is something really similar.

Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.converter;

import ch.heigvd.ptl.sc.model.User;
import ch.heigv
import java.uti
import java.uti
import org.spri                                    utowired;
import org.spri

@Service
public class UserConverter {
    @Autowired
    private Addre  Converter addressConverter;

    public final List                                  ) { ... }
    public final User
    public final List                                  ) { ... }
    public final User

    public void fillTargetFromSource(UserTO target, User source) {
     target.setAddress(addressConverter.convertSourceToTarget(source.getAddress()));
     target.setUsername(source.getUsername());
     target.setRole(source.getRole().name());
    }

    public void fillSourceFromTarget(User source, UserTO target) {
     source.setAddress(addressConverter.convertTargetToSource(target.getAddress()));
     source.setUsern me());
    }
}
```

Make sure that Spring plumbing see this class as a service that we can inject elsewhere in the application.

Spring will inject the corresponding service like we are used to with Java EE and @EJB annotation. This is something really similar.

We use the @Autowired service injected by Spring.

One more Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.service;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.persistence.UserRepository;
import ch.heigvd.ptl.sc.to.UserTO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService {
   @Autowired
   private UserConverter userConverter;

   @Autowired
   private UserRepository userRepository;

   public User register(UserTO userTO) {
    User user = userConverter.convertTargetToSource(userTO);

    user.setRole(User.Role.MEMBER);

    return userRepository.save(user);
   }
}
```

One more Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.service;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.persistence.UserRepository;
import ch.heigvd.ptl.sc.to.UserTO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService {
   @Autowired
   private UserConverter userConverter;

   @Autowired
   private UserRepository userRepository;

   public User register(UserTO userTO) {
    User user = userConverter.convertTargetToSource(userTO);

    user.setRole(User.Role.MEMBER);

    return userRepository.save(user);
   }
}
```

Again, Spring will inject the required services.

One more Service in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.service;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.persistence.UserRepository;
import ch.heigvd.ptl.sc.to.UserTO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserService {
   @Autowired
   private UserConverter userConverter;

   @Autowired
   private UserRepository userRepository;

   public User register(UserTO userTO) {
    User user = userConverter.convertTargetToSource(userTO);

    user.setRole(User.Role.MEMBER);

    return userRepository.save(user);
   }
}
```

Again, Spring will inject the required services.

We can process additional business logic.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

@Component annotation is required to let Spring managing the Resource for injections.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import
import
import
import javax.ws.rs.core.MediaType;
import javax.ws.rs.
import org.springfr
import org.springfr

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

@Component annotation is required to let Spring managing the Resource for injections.

Nothing different than Java EE as we use JAX-RS which is a standard.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import
import
import
import javax.ws.rs.core.MediaType;
import javax.ws.rs.
import org.springfr
import org.springfr

@Component
@Path("/users")
public class UserRes
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

@Component annotation is required to let Spring managing the Resource for injections.

Nothing different than Java EE as we use JAX-RS which is a standard.

Again, Spring magic happens for injection.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import
import
import
import ax.ws.rs.core.MediaType;
import vax.ws.rs.
import org.springfr
import org.springfr

@Component
@Path("/users")
public class UserRes
  @Autowired
  private UserService userService;

  @Autowired
  private Us

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

@Component annotation is required to let Spring managing the Resource for injections.

Nothing different than Java EE as we use JAX-RS which is a standard.

Again, Spring magic happens for injection.

Method register will be called when we do a post on /users

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import
import
import
import javax.ws.rs.core.MediaType;
import javax.ws.rs.
import org.springfr
import org.springfr

@Component
@Path("/users")
public class UserRes
  @Autowired
  private UserService userService;

  @Autowired
  private Us

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response r gister(UserTO userTO) {
    User user = user vice.register(userTO);

    return Response.o
      userConverter.c
    ).status(201).bui
  }
}
```

@Component annotation is required to let Spring managing the Resource for injections.

Nothing different than Java EE as we use JAX-RS which is a standard.

Again, Spring magic happens for injection.

Method register will be called when we do a post on /users

Consumes and **Produces** will define what is accepted as a representation format and what will be rendered as a response.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

Request
POST /users HTTP/1.1
Content-Type: application/json

```
{
  “username”: “fuubar”,
  “address”: {
    “street”: “Somewhere”,
    “city”: “Elsewhere”,
    “postalCode”: 1337
  }
}
```

Response
Content-Type: application/json

```
{
  “username”: “fuubar”,
  “role”: “MEMBER”,
  “address”: {
    “street”: “Somewhere”,
    “city”: “Elsewhere”,
    “postalCode”: 1337
  }
}
```

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

Magic happens there between **JAX-RS** (and the **Jersey** implementation with **Jackson**) and the container.

Once everything is correctly configured, serialization and deserialization is mainly done automatically.

JAX-RS in code

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```java
package ch.heigvd.ptl.sc.rest;

import ch.heigvd.ptl.sc.converter.UserConverter;
import ch.heigvd.ptl.sc.model.User;
import ch.heigvd.ptl.sc.service.UserService;
import ch.heigvd.ptl.sc.to.UserTO;
import javax.ws.rs.Consumes;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
@Path("/users")
public class UserResource {
  @Autowired
  private UserService userService;

  @Autowired
  private UserConverter userConverter;

  @POST
  @Produces(MediaType.APPLICATION_JSON)
  @Consumes(MediaType.APPLICATION_JSON)
  public Response register(UserTO userTO) {
    User user = userService.register(userTO);

    return Response.ok(
      userConverter.convertSourceToTarget(user)
    ).status(201).build();
  }
}
```

Magic happens there between **JAX-RS** (and the **Jersey** implementation with **Jackson**) and the container.

Once everything is correctly configured, serialization and deserialization is mainly done automatically.

Same for the response.

JAX-RS Plumbing

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

```
package ch.heigvd.ptl.sc;

import ch.heigvd.ptl.sc.rest.UserResource;
import javax.ws.rs.ApplicationPath;
import org.glassfish.jersey.server.ResourceConfig;
import org.springframework.stereotype.Component;

@Component
@ApplicationPath("/api")
public class JerseyConfig extends ResourceConfig {
  public JerseyConfig() {
    register(UserResource.class);
  }
}
```

JAX-RS Plumbing

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd

import ch.heigvd.
import javax.ws.rs
import org.glassf
import org.springf

@Component
@ApplicationPath("/api")
public class JerseyConfig extends ResourceConfig {
  public JerseyConfig() {
    register(UserResource.class);
  }
}
```

Well, we define the base path for the all the resource that will be managed by this application class.

In this our example, the user resource will finally be available under /api/users.

JAX-RS Plumbing

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd

import ch.heigvd.
import javax.ws.r
import org.glassf
import org.spring

@Component
@ApplicationPath("/api")
public class JerseyConfig extends ResourceConfig {
  public JerseyConfig() {
    register(UserResource.class);
  }
}
```

Well, we define the base path for the all the resource that will be managed by this application class.

In this our example, the user resource will finally be available under /api/users.

We simply add the user resource class to the set of classes managed by this application.

Therefore, you can imagine having two different REST applications using the same resources.

REMARK: HTTP 404 -> Resource class probably missing there!

JAX-RS Plumbing

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

We need to inform Spring to manage that configuration. Otherwise, no REST resources will be available once the app is started.

Well, we define the base path for the all the resource that will be managed by this application class.

In this our example, the user resource will finally be available under /api/users.

```
heigvd

eigvd.
x.ws.r
glassf
spring
```

```
@Component
@ApplicationPath("/api")
public class JerseyConfig extends ResourceConfig {
  public JerseyConfig() {
    register(UserResource.class);
  }
}
```

We simply add the user resource class to the set of classes managed by this application.

Therefore, you can imagine having two different REST applications using the same resources.

REMARK: HTTP 404 -> Resource class probably missing there!

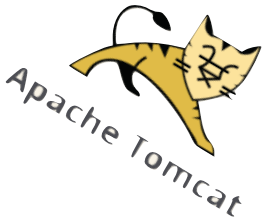
Spring Boot main application

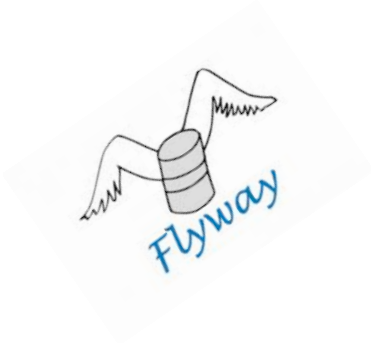
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
package ch.heigvd.ptl.sc;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;

@SpringBootApplication
public class SampleApplication extends SpringBootServletInitializer {
  @Override
  protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
    return application.sources(SampleApplication.class);
  }

  public static void main(String[] args) {
    new SampleApplication().configure(
      new SpringApplicationBuilder(SampleApplication.class)
    ).run(args);
  }
}
```

Spring Boot integrations

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud





Spring Boot in summary

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- Brings the power of Spring Framework and related libraries
- Offers code infrastructure to easily develop and deploy
- Various integrations
- Nothing really new for the core technologies (services, persistence, ...)
- Production ready tools embedded (metrics, health, configuration, ...)
- Big community around Spring technologies
- Not specific to application servers