Lecture 3: business tier & REST APIs

Olivier Liechti
AMT

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

Project - 1 feature

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

As an admin, I have a REST API to do CRUD operations on user accounts

but the data does not have to be stored in a database. It can be stored "in memory"

Feature 1 tasks

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

Understand what EJBs are and how to use them

Understand the difference between @Singleton and @Stateless

Refactor existing services to use EJBs

Learn how to implement a REST API with JAX-RS

Validate dependency injection in servlet

Experiment with JMeter

Define the REST API for user accounts (specify how **passwords** are handled!)

Update the docker-compose environment

Test the REST API with Postman (save and share collection of requests)

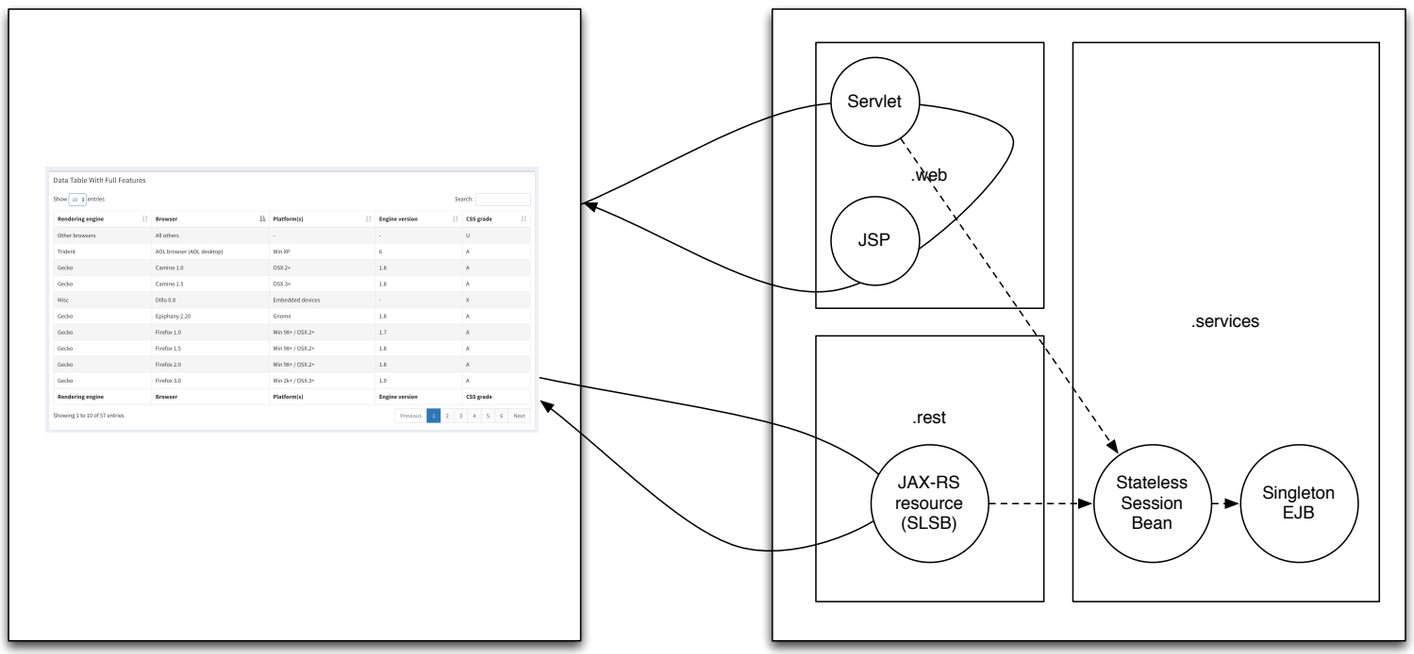
Validate the acceptance criteria.

Validate that an authenticated user can log out and not access protected pages anymore.

Webcasts

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

1 MVC - the browser asks for a an HTML page and its assets (css, js, etc.)



2 SPA - the data tables script invokes the REST API to get data (AJAX)

Tasks

1. Create a new project
- 1.1. the code deployed in Glassfish and Wildfly will be slightly different
- 1.2. for this reason, we will work in 2 branches: fb-rest-glassfish, fb-rest-wild

2. Implement the business services layer with EJBs
- 2.1. Implement a singleton EJB
- 2.2. Implement a stateless session bean
- 2.3. Inject the stateless session bean in a servlet

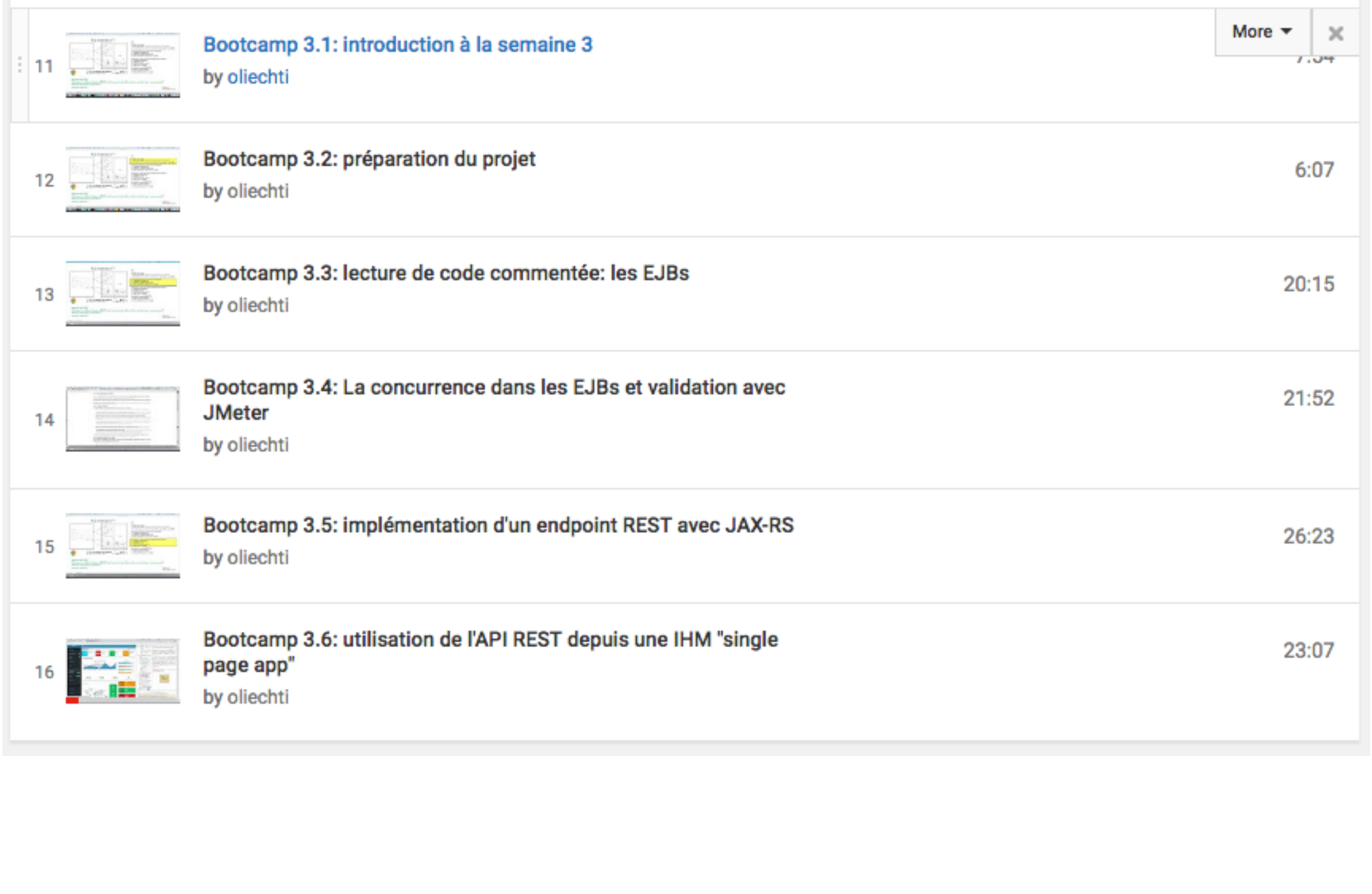
3. Implement a REST API with JAX-RS (Jersey and Jackson)
- 3.1. Configure the framework
- 3.2. Implement DTOs
- 3.3. Implement a REST endpoint
- 3.4. Test the REST endpoint

4. Build a UI on top of the REST API
- 4.1. Select and study a template
- 4.2. Discover jquery datatables
- 4.3. Integrate the template in the project

What is an EJB?
What are the different types of EJBS and how are they different from servlets (e.g. concurrency)?
What is dependency injection?
What is JAX-RS?

Webcasts

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

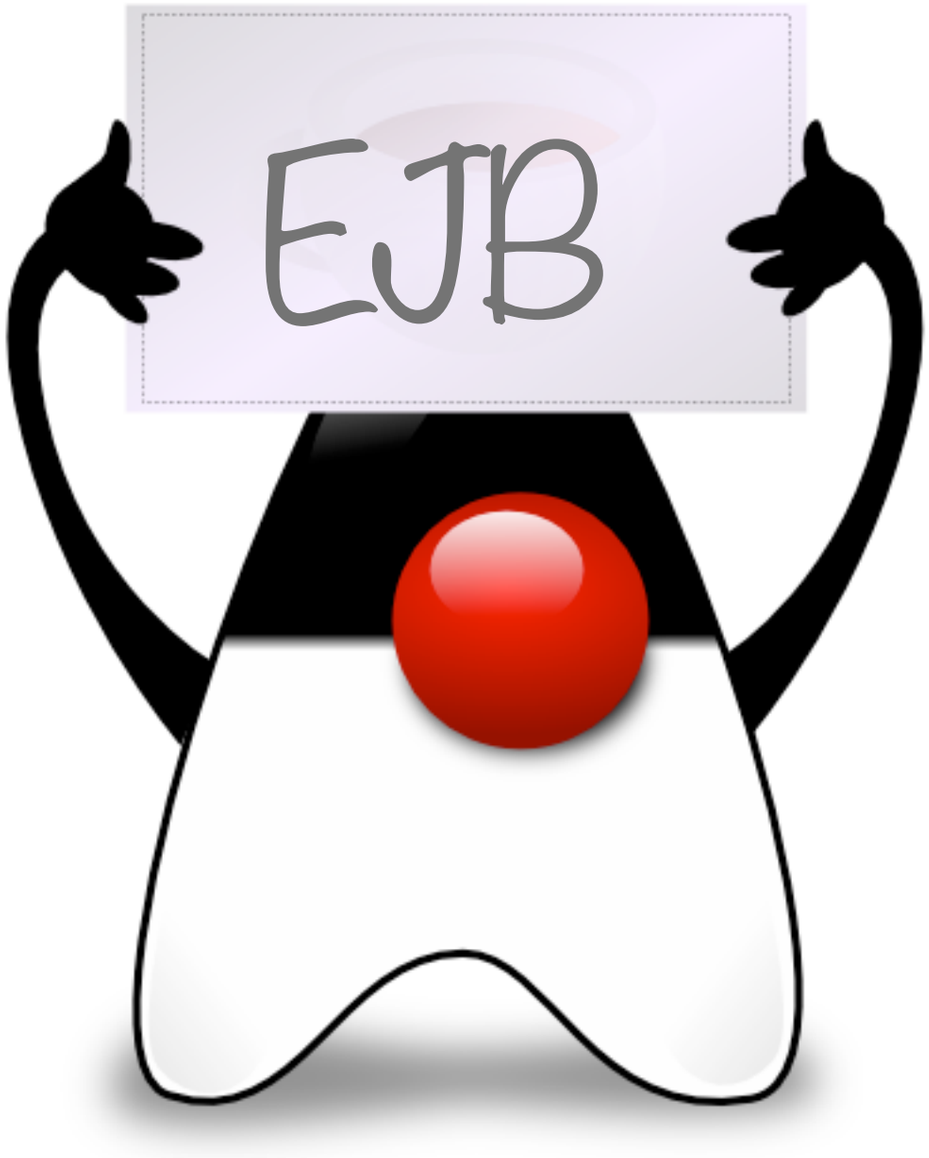


Today's agenda

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

13:15 - 13:40	**25'**	**The business tier & the EJBs**
13:40 - 14:00	20'	Exercise: counters with EJB and servlets
14:00 - 14:15	15'	Introduction to JMeter
14:15 - 14:45	30'	Exercise: load test the counters
14:45 - 15:15	30'	Introduction to JAX-RS
15:15 - 15:40	10'	Individual work

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



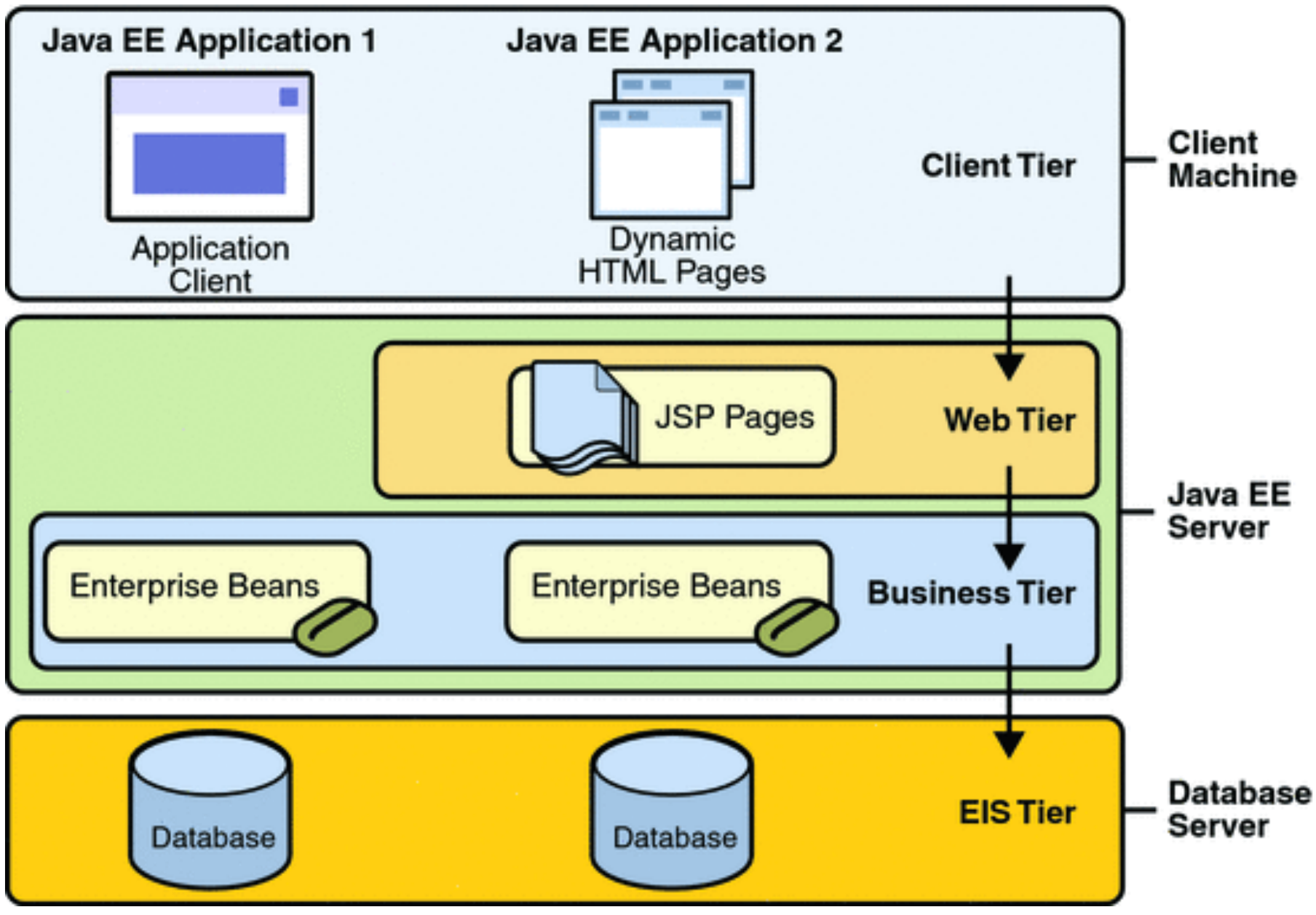
Business Services & EJB

Services in a Java EE application

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

- Last week, we implemented a very simple Java EE application.
- When we implemented the MVC pattern, we implemented a service as a **Plain Old Java Object** (POJO).
- **The POJO was not a managed component**. We created the instance(s) of the service (*in the web container*).
- This week, we will see an **alternative solution** for implementing Java EE services: Enterprise Java Beans (EJBs).

What is the best way to implement services, POJOs or EJBs?
There is not a single right answer to this question! There are pros and cons in both approach.



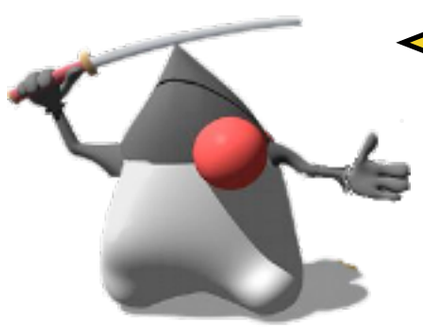
http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html

What is an **Enterprise Java Bean** (EJB)?

- An EJB is a **managed component**, which implements **business logic** *in a UI agnostic way*.
- The EJB container manages the **lifecycle** of the EJB instances.
- The EJB container also **mediates the access** from clients (i.e. it is an “invisible” intermediary) to EJBs.
- This allows the EJB container to perform technical operations (especially related to **transactions** and **security**) when EJBs are invoked by clients.
- The EJB container manages a **pool** of EJB instances.
- Note: the EJB 3.2 API is **specified is JSR 345**.

What are the **4 types** of EJBs used today?

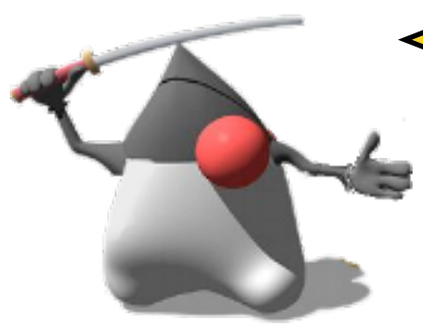
- **Stateless Session Beans** are used to implement business services, where every client request is independent.
- **Stateful Session Beans** are used for services which have a notion of conversation (e.g. shopping cart).
- **Singleton Session Beans** are used when there should be a single service instance in the app.
- **Message Driven Beans** are used together with the Java Message Service (JMS). Business logic is not invoked when a web client sends a request, but when a message arrives in a queue. We will see that later.



When you implement a stateful application in Java EE, **you have the choice to store the state in different places**. One option is to do it in the web tier (in the HTTP session). Another option is to use **Stateful Session Beans**. Many (most) developers use HTTP sessions.

In older versions of Java EE (before Java EE 5), there was another type of EJBs: **Entity Beans**.

Entity Beans were used for **accessing the database**. They were a nightmare to use and raised a number of issues. You might find them in legacy applications.



Entity Beans (as a legacy type of EJB) are **not the same thing** as **JPA Entities**, which are now widely used!

A first example

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

```
package ch.heigvd.amt.lab1.services;
import javax.ejb.Local;

@Local
public interface CollectorServiceLocal {

  void submitMeasure(Measure measure);

}
```

```
package ch.heigvd.amt.lab1.services;
import javax.ejb.Stateless;

@Stateless
public class CollectorService implements CollectorServiceLocal {

  @Override
  public void submitMeasure(Measure measure) {
    // do something with the measure (process, archive, etc.)
  }

}
```

These **annotations** are processed by the application server at **deployment time**.

They are an **declaration** that the service must be handled as a **managed component**!

How does a “client” find and use an EJB?

- By “**client**”, we refer to a **Java component** that wants to get a reference to the EJB and invoke its methods.
- In many cases, the client is a **servlet** or **another EJB** (i.e. a service that delegates part of the work to another service).
- The application server is providing a **naming and directory service** for managed components. Think of it as a “white pages” service that keeps track of component names and references.
- Remember that we mentioned **Dependency Injection** earlier today?

The Java Naming and Directory Interface (JNDI) provides an API to access directory services. It can be used to access an LDAP server. It can also be used to lookup components in a Java EE server.

The **first method** to find an EJB is to do an **explicit lookup**, with JNDI.

```
@WebServlet(name = "FrontController", urlPatterns = {"/FrontController"})
public class FrontController extends HttpServlet {

  private CollectorServiceLocal collectorService;

  @Override
  public void init() throws ServletException {
    super.init();
    try {
     Context ctx = new InitialContext();
      collectorService = (CollectorServiceLocal) ctx.lookup("java:module/CollectorService");
    } catch (NamingException ex) {
      Logger.getLogger(FrontController.class.getName()).log(Level.SEVERE, null, ex);
    }
  }

}
```

This gives me access to the app server's naming service

I am using the app server's naming service

Warning! These 2 JNDI operations are **costly** (performance-wise). You don't want to re-execute them for every single HTTP request!!!!
It is much better to do it once and to **cache the references** to the services.

The **second method** is to ask the app server to **inject a dependency** to the service.

```
@WebServlet(name = "FrontController", urlPatterns = {"/FrontController"})
public class FrontController extends HttpServlet {

  @EJB
  private CollectorServiceLocal collectorService;

}
```

With the @EJB annotation, **I am declaring a dependency** from between my servlet and my service. The servlet *uses* the service.

With the @EJB annotation, I am also giving instructions to the app server. The servlet and the service are **managed components**. When the app server instantiates the servlet, it **injects a value** into the **collectorService** variable.

EJBs in the MVCDemo project

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

```
@Singleton
public class BeersDataStore implements BeersDataStoreLocal {

 private final List<Beer> catalog = new LinkedList<>();

 public BeersDataStore() {
    catalog.add(new Beer("Cardinal", " Feldschlösschen", "Switzlerland", "Lager"));
    catalog.add(new Beer("Punk IPA", " BrewDog", "Scotland", "India Pale Ale"));
 }
...
}
```

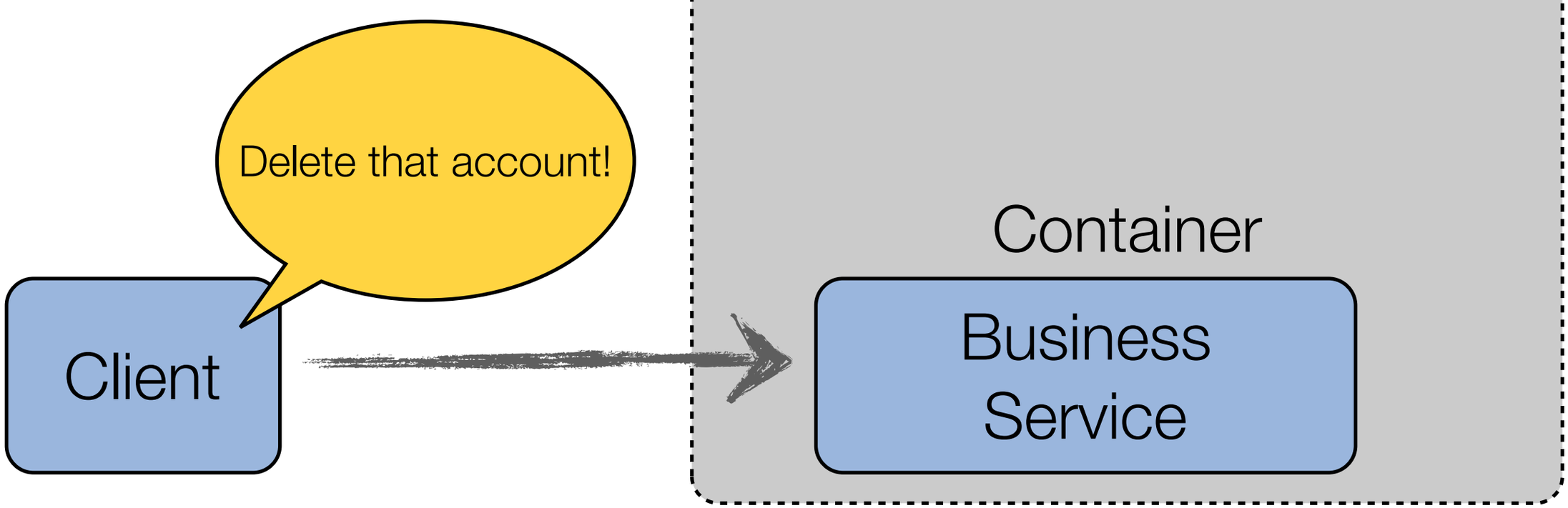
```
@Stateless
public class BeersManager implements BeersManagerLocal {

  @EJB
  BeersDataStoreLocal beersDataStore;

  @Override
  public List<Beer> getAllBeers() {
    simulateDatabaseDelay();
    return beersDataStore.getAllBeers();
  }
...
}
```

EJBs in the MVCDemo project

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

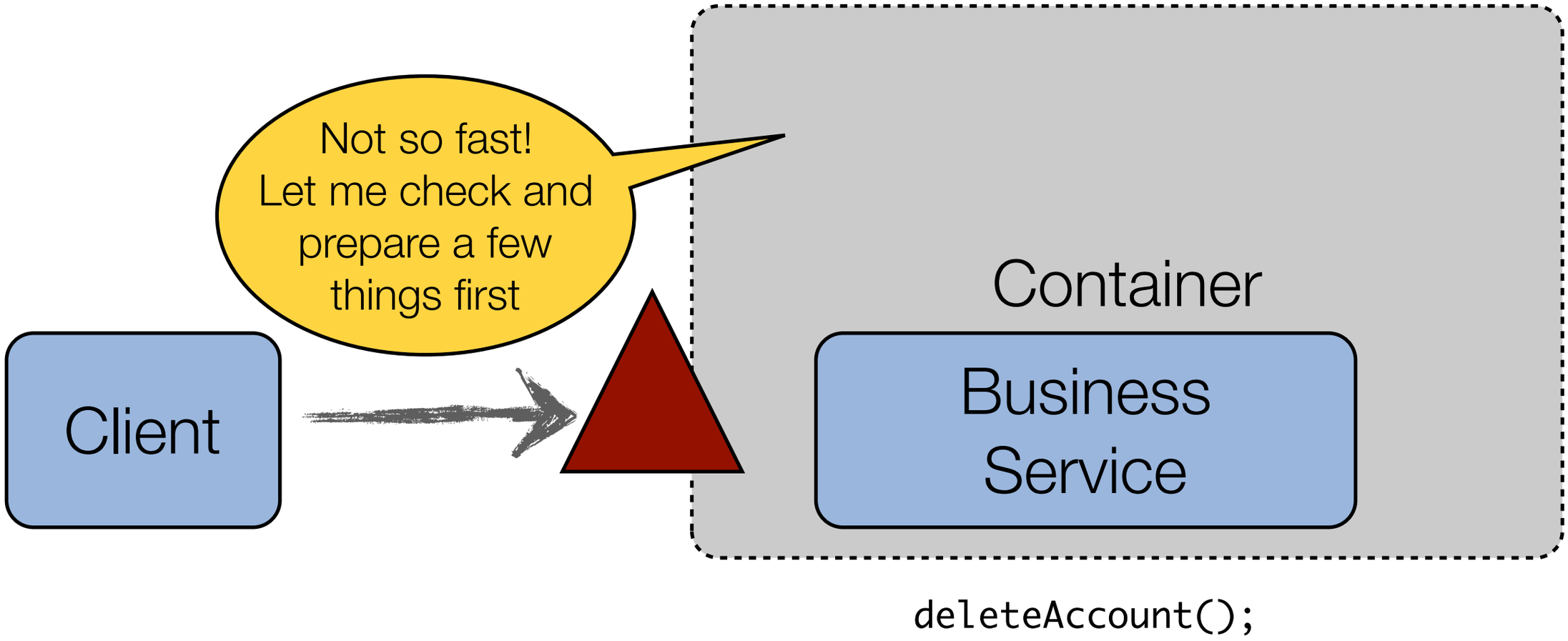
```
public class BeersServlet extends HttpServlet {

  @EJB
  BeersManagerLocal beersManager;

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
     /*
      Firstly, we need to get a model. It is not the responsibility of the servlet
      to build the model. In other words, you should avoid to put business logic
      and database access code directly in the controller. In this example, the
      beersManager takes care of the model construction.
      */
     Object model = beersManager.getAllBeers();
     ...
   }
   ...
}
```

The app server **mediates** the access between clients and EJBs. What does it mean?



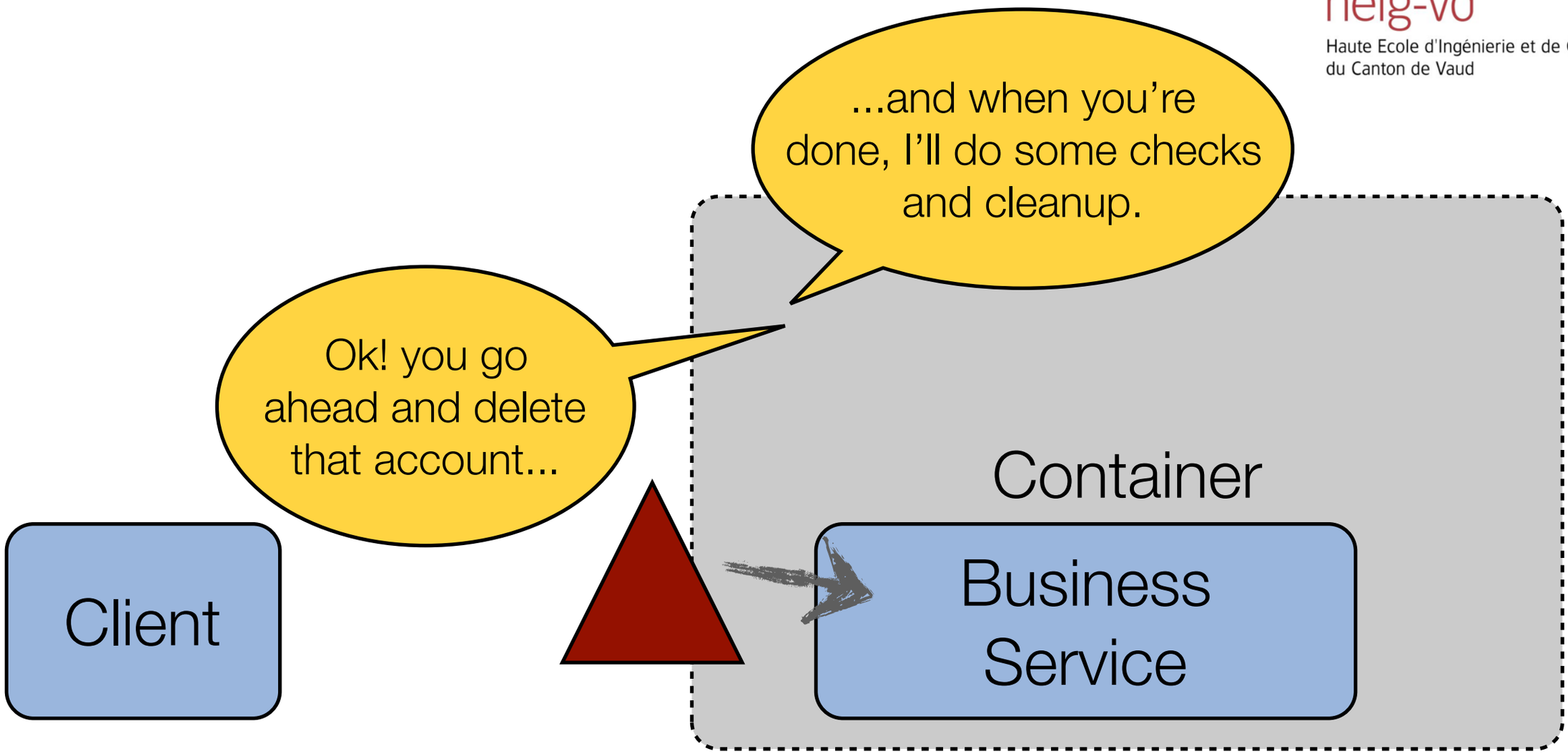

```
deleteAccount();
```

The business service, implemented as a Stateless Session Bean, is a **managed component**.
The client ***thinks*** that he has a direct reference to a Java object.
He is ***wrong***.




```
deleteAccount();
```

In reality, when the client invokes the deleteAccount() methods, the call is going **through the container**.
The container is in a position to **perform various tasks** (security checks, transaction demarcation, etc.)



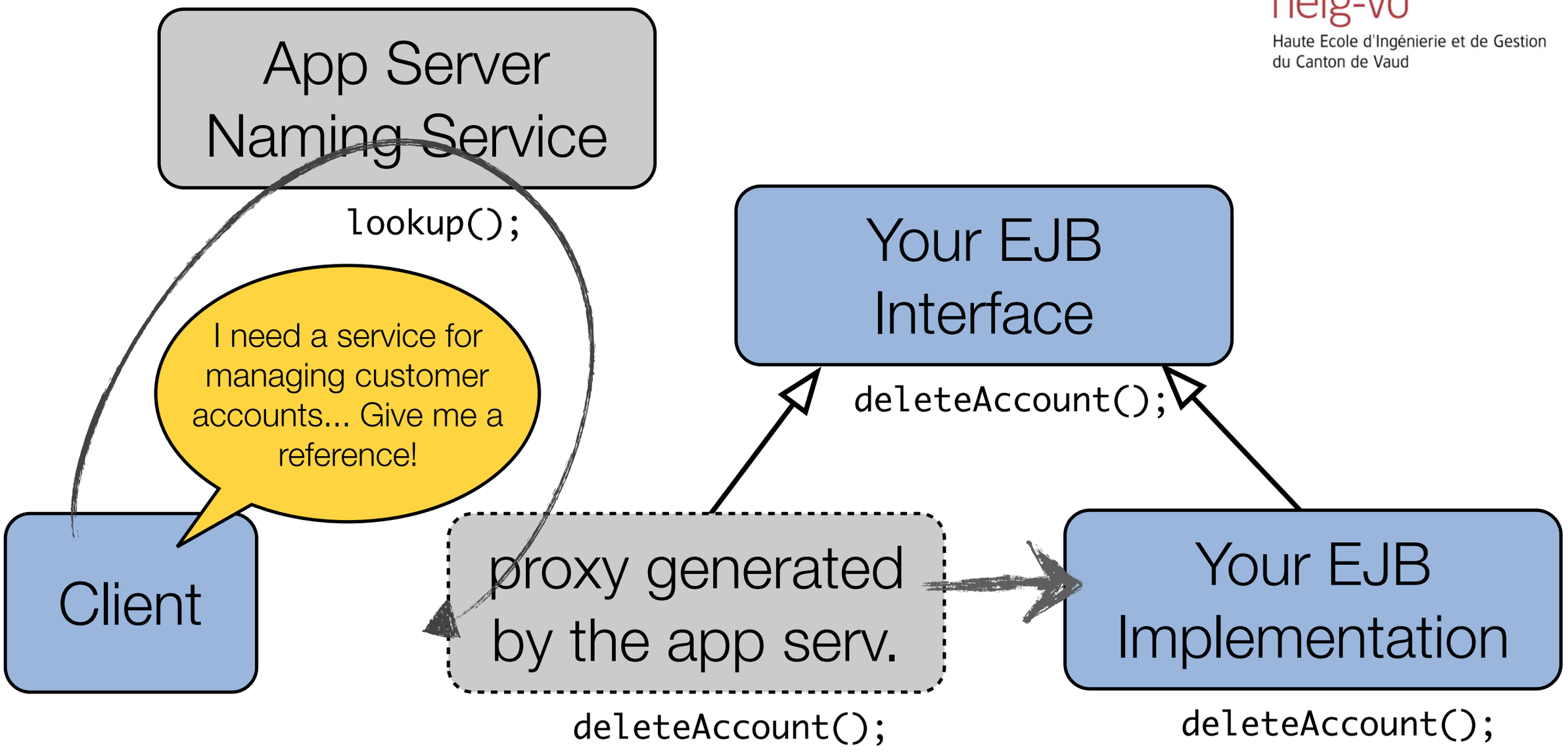

```
deleteAccount();
```

When done, the container can forward the method call to the business service (your implementation).
On the way back, the response also goes back **via the container**.

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

How is that possible?
How does it work?

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



Your service implementation implements your interface.
The container dynamically generates a class, which implements the same interface. This class performs the technical tasks and invokes your class (proxy).

Scientific Method

Formulate a **question**

"How is the container able to **intercept** method calls on EJBs?"

Propose an **hypothesis**

"I believe that it generates **dynamic proxies**"

Predict the **consequences**

"**If** I am right, then I should see these proxies in the call stack."

Experiment and **measure**

"Let's **throw an exception** in my stateless session bean and have a look at the stack trace"

Analyze

"In the stack trace, I can confirm that the servlet is not directly calling my Stateless Session Bean implementation class."

```
Caused by: java.lang.RuntimeException: just kidding
     at ch.heigvd.amt.lab1.services.CollectorService.submitMeasure(CollectorService.java:15)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:483)
     at org.glassfish.ejb.security.application.EJBSecurityManager.runMethod(EJBSecurityManager.java:1081)
     at org.glassfish.ejb.security.application.EJBSecurityManager.invoke(EJBSecurityManager.java:1153)
     at com.sun.ejb.containers.BaseContainer.invokeBeanMethod(BaseContainer.java:4786)
     at com.sun.ejb.EjbInvocation.invokeBeanMethod(EjbInvocation.java:656)
     at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
     at com.sun.ejb.EjbInvocation.proceed(EjbInvocation.java:608)
     at
org.jboss.weld.ejb.AbstractEJBRequestScopeActivationInterceptor.aroundInvoke(AbstractEJBRequestScopeActivationInte
ceptor.java:46)
     at org.jboss.weld.ejb.SessionBeanInterceptor.aroundInvoke(SessionBeanInterceptor.java:52)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMeth
     at java.lang.reflect.Method.i
     at com.sun.ejb.containers.int
     at com.sun.ejb.containers.int
     at com.sun.ejb.EjbInvocation.
     at com.sun.ejb.containers.int
     at com.sun.ejb.containers.int
     at sun.reflect.NativeMethodAc
     at sun.reflect.NativeMethodAc
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:483)
     at com.sun.ejb.containers.interceptors.AroundInvokeInterceptor.intercept(InterceptorManager.java:883)
     at com.sun.ejb.containers.interceptors.AroundInvokeChainImpl.invokeNext(InterceptorManager.java:822)
     at com.sun.ejb.containers.interceptors.InterceptorManager.intercept(InterceptorManager.java:369)
     at com.sun.ejb.containers.BaseContainer.__intercept(BaseContainer.java:4758)
     at com.sun.ejb.containers.BaseContainer.intercept(BaseContainer.java:4746)
     at com.sun.ejb.containers.EJBLocalObjectInvocationHandler.invoke(EJBLocalObjectInvocationHandler.java:212)
     ... 34 more
```

```
@Stateless
public class CollectorService implements CollectorServiceLocal {

  @Override
  public void submitMeasure(Measure measure) {
    throw new RuntimeException("just kidding");
  }
```

Scientific Method

Formulate a **question**

"How is the container able to **intercept** method calls on EJBs?"

Propose an **hypothesis**

"I believe that it generates **dynamic proxies**"

Predict the **consequences**

"**If** I am right, then I should see these proxies in the call stack."

Experiment and **measure**

"Let's **inspect the call stack in the debugger**"

Analyze

"In the debugger console, I can confirm that there are container generated classes between my servlet and my service implementation."

Projects | Files | Services | Debugging

'http-listener-1(5)' at line breakpoint Collecto

- **CollectorService.submitMeasure:15**
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
- Method.invoke:483
- Hidden Source Calls
  - AroundInvokeInterceptor.intercept:883
  - AroundInvokeChainImpl.invokeNext:82
  - InterceptorManager.intercept:369
  - BaseContainer.__intercept:4758
  - BaseContainer.intercept:4746
  - EJBLocalObjectInvocationHandler.invok
  - EJBLocalObjectInvocationHandlerDeleg
  - $Proxy347.submitMeasure
  - FrontController.processRequest:86
  - FrontController.doGet:103
- Hidden Source Calls
- Thread.run:745

At some point, the method call is forwarded to my implementation.

The reference actually points to a proxy generated by the container. The container performs tasks that are visible in a **long call stack**!

My servlet invokes the method on its **reference** to the EJB.

*An HTTP request has arrived; GF invokes the doGet callback on my servlet (**IoC**). GF has also **injected** a **reference** to the EJB into the servlet.*

Today's agenda

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

13:15 - 13:40	25'	The business tier & the EJBs
13:40 - 14:00	**20'**	**Exercise: counters with EJB and servlets**
14:00 - 14:15	15'	Introduction to JMeter
14:15 - 14:45	30'	Exercise: load test the counters
14:45 - 15:15	30'	Introduction to JAX-RS
15:15 - 15:40	10'	Individual work

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Exercise (20')

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

- **Create a new project for the experiment**
- **Implement a Singleton Session Bean with a single business method:**
  - public long **incrementAndGetSingletonCounter()**, which updates an instance variable and returns its value to the client.
- **Implement a Servlet:**
  - inject a reference to the Singleton Session Bean
  - implement a private method **incrementAndGetServletCounter()**, which updates an instance variable and returns its value to the client.
  - in the doGet method, invoke both **incrementAndGetSingletonCounter** and **incrementAndGetServletCounter**.

Today's agenda

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

13:15 - 13:40	25'	The business tier & the EJBs
13:40 - 14:00	20'	Exercise: counters with EJB and servlets
14:00 - 14:15	**15'**	**Introduction to JMeter**
14:15 - 14:45	30'	Exercise: load test the counters
14:45 - 15:15	30'	Introduction to JAX-RS
15:15 - 15:40	10'	Individual work

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

Introduction to JMeter

JMeter

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

- Open source project, apache foundation
- http://jmeter.apache.org/index.html

"The Apache JMeter™ desktop application is open source software, a 100% pure Java application designed to ***load test functional behavior*** *and* ***measure performance****.*
It was originally designed for ***testing Web Applications*** *but has since* ***expanded to other*** *test functions."*

JMeter

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

*"Apache JMeter may be used to **test performance** both on static and dynamic resources (files, Servlets, Perl scripts, Java Objects, Data Bases and Queries, FTP Servers and more).*

*It can be used to **simulate a heavy load** on a server, network or object to test its strength or to analyze overall performance under **different load types**. You can use it to make a **graphical analysis of performance** or to test your server/script/object behavior under heavy **concurrent** load."*

Types of tests (1)

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

- **Functional tests**
  - Is the system doing what it is supposed to do?
  - Does its behavior comply with functional requirements (use cases)?
  - Selenium is a tool for automating functional testing of web applications (http://seleniumhq.org/)
- **Performance, load and stress tests**
  - What is the response time? What is the consumption of resources? Are there issues (e.g. concurrency issues) that happen under load?
  - Relevant both for interactive and batch use cases.

What to install?

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

- **Main project**
  - http://jmeter.apache.org/download_jmeter.cgi
- **Add-ons**
  - http://jmeter-plugins.org/

Standard Set

Basic plugins for everyday needs. Does not require additional libs to run.

Download | Installation | Package Contents

Extras Set

Additional plugins for extended and complex testing. Does not require additional libs to run.

Download | Installation | Package Contents

Extras with Libs Set

Additional plugins that *do require* additional libs to run.

Download | Installation | Package Contents

WebDriver Set

Selenium/WebDriver testing ability.

Download | Installation | Package Contents

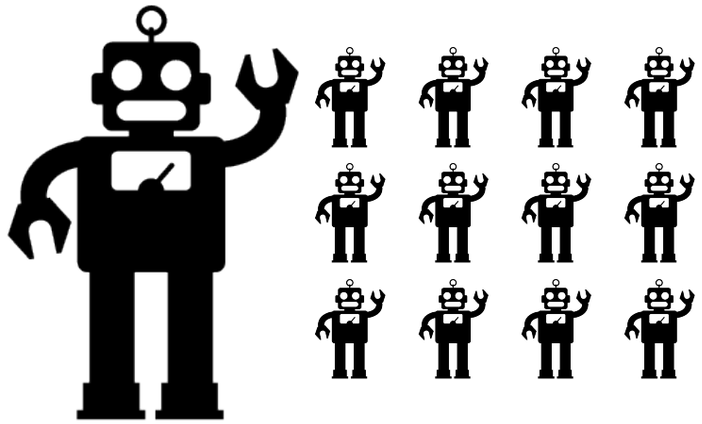
Hadoop Set

Hadoop/HBase testing plugins.

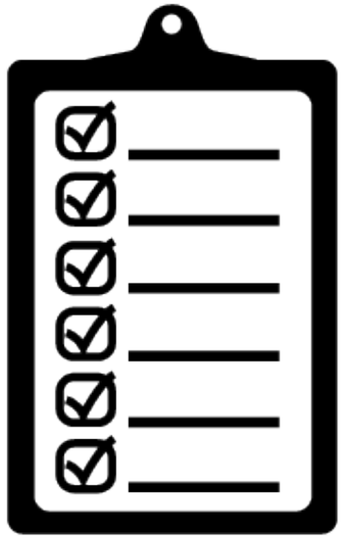
Download | Installation | Package Contents

JMeter Building Blocks

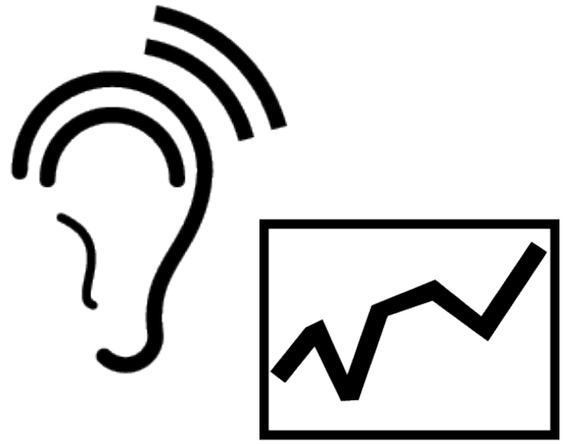
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



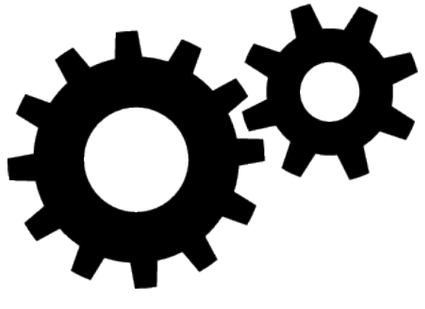
ThreadGroup



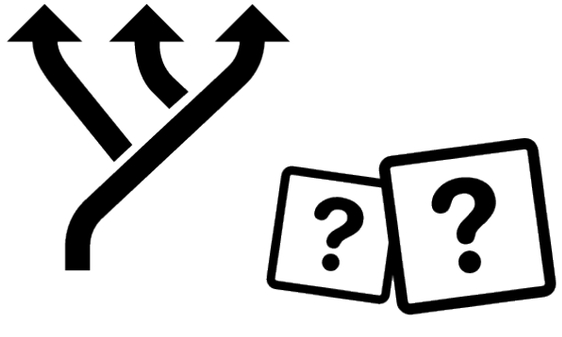
Test Plan



Listeners
(results & stats)



Samplers
(actions)



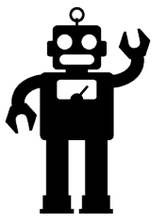
Logic Controllers
& Assertions

Concepts

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

- Test Plan
- ThreadGroup
- Samplers
- Logic Controllers
- Listeners
- Timers
- Assertions
- Configuration Elements
- Pre-Processor Elements
- Post-Processor Elements

Thread Group

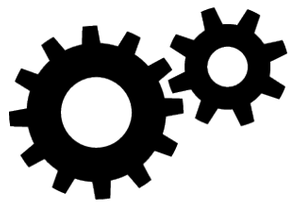


heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

*"Thread group elements are the **beginning points of any test plan**. All controllers and samplers must be under a thread group. [...]. As the name implies, the thread group element controls the number of threads JMeter will use to execute your test. The controls for a thread group allow you to:*

- *Set the **number of threads***
- *Set the **ramp-up** period*
- *Set the **number of times** to execute the test*

*Each thread will execute the test plan in its entirety and completely independently of other test threads. **Multiple threads are used to simulate concurrent connections to your server application.**"*

Sampler 

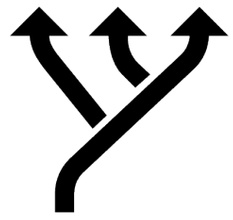
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

*"Samplers tell JMeter to **send requests to a server and wait for a response**. They are processed in the order they appear in the tree. Controllers can be used to modify the number of repetitions of a sampler.*

- *FTP Request*
- ***HTTP Request***
- *JDBC Request*
- *Java object request*
- *LDAP Request*
- *SOAP/XML-RPC Request*
- *WebService (SOAP) Request*

*Each sampler has several **properties** you can set. You can further customize a sampler by adding one or more Configuration Elements to the Test Plan."*

Logic Controllers



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

"Logic Controllers let you customize ***the logic that JMeter uses to decide when to send requests****. Logic Controllers can change the order of requests coming from their child elements. They can modify the requests themselves, cause JMeter to repeat requests, etc."*

- *Loop Controller*
- *Once Only Controller*
- *Interleave Controller*
- *Random Controller*
- *Random Order Controller*
- *Throughput Controller*
- *Runtime Controller*
- *If Controller*
- *etc.*

Listeners

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

"Listeners provide ***access to the information JMeter gathers about the test cases*** *while JMeter runs. The* ***Graph Results*** *listener plots the response times on a graph. The "****View Results Tree****" Listener shows details of sampler requests and responses, and can display basic HTML and XML representations of the response. Other listeners provide* ***summary or aggregation information****.*

Additionally, listeners can ***direct the data to a file*** *for later use.*

Listeners can be added anywhere in the test, including directly under the test plan. They will collect data only from elements at or below their level."

Timers

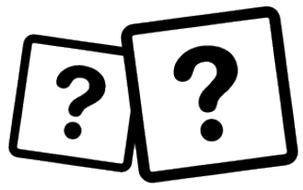
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

"By default, a JMeter thread sends requests without pausing between each request. We recommend that you specify a delay by adding one of the available timers to your Thread Group. If you do not add a delay, JMeter could ***overwhelm your server*** *by making too many requests in a very short amount of time.*
The timer will cause JMeter to delay a certain amount of time before each sampler which is in its scope .

If you choose to add more than one timer to a Thread Group, JMeter takes the sum of the timers and pauses for that amount of time before executing the samplers to which the timers apply. Timers can be added as children of samplers or controllers in order to restrict the samplers to which they are applied.

To provide a pause at a single place in a test plan, one can use the ***Test Action Sampler.****"*

Assertions



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

---

*"Assertions allow you to **assert facts about responses received** from the server being tested.*

*Using an assertion, you can essentially **"test" that your application is returning the results you expect it to**.*

*For instance, you can assert that the response to a query will **contain some particular text**. The text you specify can be a Perl-style regular expression, and you can indicate that the response is to contain the text, or that it should match the whole response.*

*You can add an assertion to any Sampler. For example, you can add an assertion to a HTTP Request that checks for the text, "</HTML>". JMeter will then check that the text is present in the HTTP response. If JMeter cannot find the text, then it will **mark this as a failed request.**"*

Configuration Elements

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

"A configuration element works closely with a Sampler. Although it does not send requests (except for HTTP Proxy Server), it can add to or modify requests.

A configuration element is accessible from only inside the tree branch where you place the element. For example, if you place an ***HTTP Cookie Manager*** *inside a Simple Logic Controller, the Cookie Manager will only be accessible to HTTP Request Controllers you place inside the Simple Logic Controller"*

- *HTTP Authorization Manager*
- *HTTP Cache Manager*
- *HTTP Cookie Manager*
- *HTTP Request Defaults*
- *HTTP Header Manager*

How to Create Test Scenarios?

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

- **Option 1 : manually**
  - Create a Test Plan
  - Add a Thread Group
  - Add HTTP samplers and specify HTTP request parameters
- **Option 2 : recording with JMeter configured as an HTTP proxy**
  - http://jmeter.apache.org/usermanual/jmeter_proxy_step_by_step.pdf
  - Do manual adjustments

Advanced Usage

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

- **Use variables:**
  - Very often, it is needed to use parts of an HTTP response into follow-up requests (e.g. session ids).
  - `http://jmeter.apache.org/usermanual/test_plan.html#properties`
- **Use more than one machine:**
  - With JMeter running on a single machine, it is common to “exhaust” the client before the server (especially when testing a “real” infrastructure with multiple nodes).
  - For real performance tests, it is therefore recommended to use multiple machines for injecting load into the network. One way to do it is use use virtual machines in a cloud environment (e.g. on Amazon EC2).
  - Multiple JMeter clients can be coordinated by a master and results can be collected and aggregated (`http://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.pdf`)

Today's agenda

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

13:15 - 13:40	25'	The business tier & the EJBs
13:40 - 14:00	20'	Exercise: counters with EJB and servlets
14:00 - 14:15	15'	Introduction to JMeter
14:15 - 14:45	**30'**	**Exercise: load test the counters**
14:45 - 15:15	30'	Introduction to JAX-RS
15:15 - 15:40	10'	Individual work

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Exercise (20')

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

- **Install JMeter**
  - http://jmeter.apache.org/
- **Install the JMeter plugin manager**
  - https://jmeter-plugins.org/install/Install/
- Create a test plan, with at the minimum:
  - An **ultimate thread group** (plugin to install with the plugin manager)
  - An **HTTP request sampler**
  - A **constant timer**
  - **Listeners** to display the results (play with "summary report", "response time graph", "view results in tree").

Exe

et de Gestion

- Ins
- Ins

JMeter Plugins Manager

Installed Plugins | Available Plugins | Upgrades

- 3 Basic Graphs
- 5 Additional Graphs
- BM.Sense Uploader
- Command-Line Graph Plotting Tool
- Composite Timeline Graph
- Custom JMeter Functions
- Custom Thread Groups
- Distribution/Percentile Graphs
- Dummy Sampler
- FTP Protocol Support
- Flexible File Writer
- Graphs Generator Listener
- HTTP Protocol Support
- Inter-Thread Communication
- JDBC Support
- JMS Support
- JMeter 'Monitors' (Deprecated)
- JMeter Core
- JSON Plugins
- JUnit Support
- Java Components
- LDAP Protocol Support
- Mail/SMTP Support
- MongoDB Support
- OS Process Support
- PerfMon (Servers Performance Monitoring)
- Plugins Manager
- Selenium/WebDriver Support
- TCP Protocol Support
- Throughput Shaping Timer
- Various Core Components
- jpgc - Standard Set

Custom Thread Groups

Vendor: *JMeter-Plugins.org*

Adds new Thread Groups:

- Stepping Thread Group
- Ultimate Thread Group
- Concurrency Thread Group
- Arrivals Thread Group
- Free-Form Arrivals Thread Group

Documentation: https://jmeter-plugins.org/wiki/ConcurrencyThreadGroup/

jp@gc - Concurrency Thread Group

Name: jp@gc - Concurrency Thread Group

Comments:

Help on this plugin v1.4.0

Action to be taken after a Sampler error

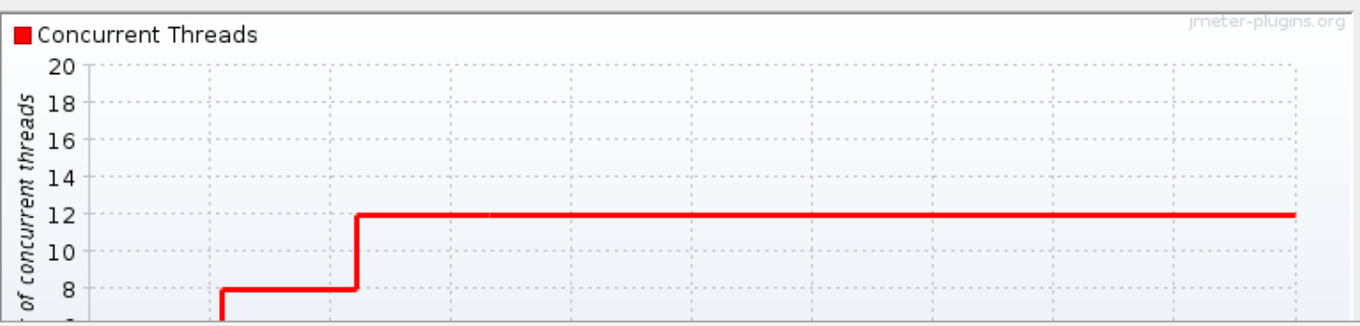
(•) Continue ( ) Start Next Thread Loop ( ) Stop Thread ( ) Stop Test ( ) Stop Test Now

Target Concurrency: 12

Ramp Up Time (sec): 60

Ramp-Up Steps Count: 3

Hold Target Rate Time (sec): 120



Version: 2.1

Review Changes

Uninstall plugin: jpgc-standard 2.0

Apply Changes and Restart JMeter

Today's agenda

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

13:15 - 13:40	25'	The business tier & the EJBs
13:40 - 14:00	20'	Exercise: counters with EJB and servlets
14:00 - 14:15	15'	Introduction to JMeter
14:15 - 14:45	30'	Exercise: load test the counters
14:45 - 15:15	**30'**	**Introduction to JAX-RS**
15:15 - 15:40	10'	Individual work

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

RESTful APIs

The REST Architectural Style

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

- REST: **REpresentational State Transfer**
- REST is an **architectural style** for building distributed systems.
- REST has been introduced in **Roy Fielding's Ph.D. thesis** (Roy Fielding has been a contributor to the HTTP specification, to the apache server, to the apache community).
- The WWW is **one example** for a distributed system that exhibits the characteristics of a REST architecture.

Principles of a REST Architecture

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

- The state of the application is captured in a **set of resources**
  - Users, photos, comments, tags, albums, etc.
- Every resource can be **identified with a standard format** (e.g. URL)
- Every resource can have **several representations**
- There is one **unique interface for interacting** with resources (e.g. HTTP methods)
- The communication protocol is:
  - client-server
  - stateless
  - cacheable
- These properties have a positive impact on systemic qualities (scalability, performance, availability, etc.).
  - Reference: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

HTTP is a protocol for interacting with "**resources**"

Resource vs. Representation

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

- A "resource" can be something intangible (stock quote) or tangible (vending machine)
- The HTTP protocol supports the exchange of data between a client and a server.
- Hence, what is exchanged between a client and a server is **not** the resource. It is a **representation** of a resource.
- Different representations of the same resource can be generated:
  - HTML representation
  - XML representation
  - PNG representation
  - WAV representation
- **HTTP provides the content negotiation mechanisms!!**

How Do We Interact With Resources?

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

- The HTTP protocol defines the standard methods. These methods enable the interactions with the resources:
  - **GET**: retrieve whatever information is identified by the Request-URI
  - **POST**: used to request that the origin server accept the entity enclosed in the request as a new subordinate of the ressource identified by the Request-URI in the Request-Line
  - **PUT**: requests that the enclosed entity be stored under the supplied Request-URI.
  - **DELETE**: requests that the origin server delete the ressource identified by the Request-URI.
  - HEAD: identical to GET except that the server MUST NOT return a message-body in the response
  - TRACE: used for debugging (echo)
  - CONNECT: reserved for tunneling purposes
  - **PATCH**: used for partial updates

How should I design and specify my REST API?

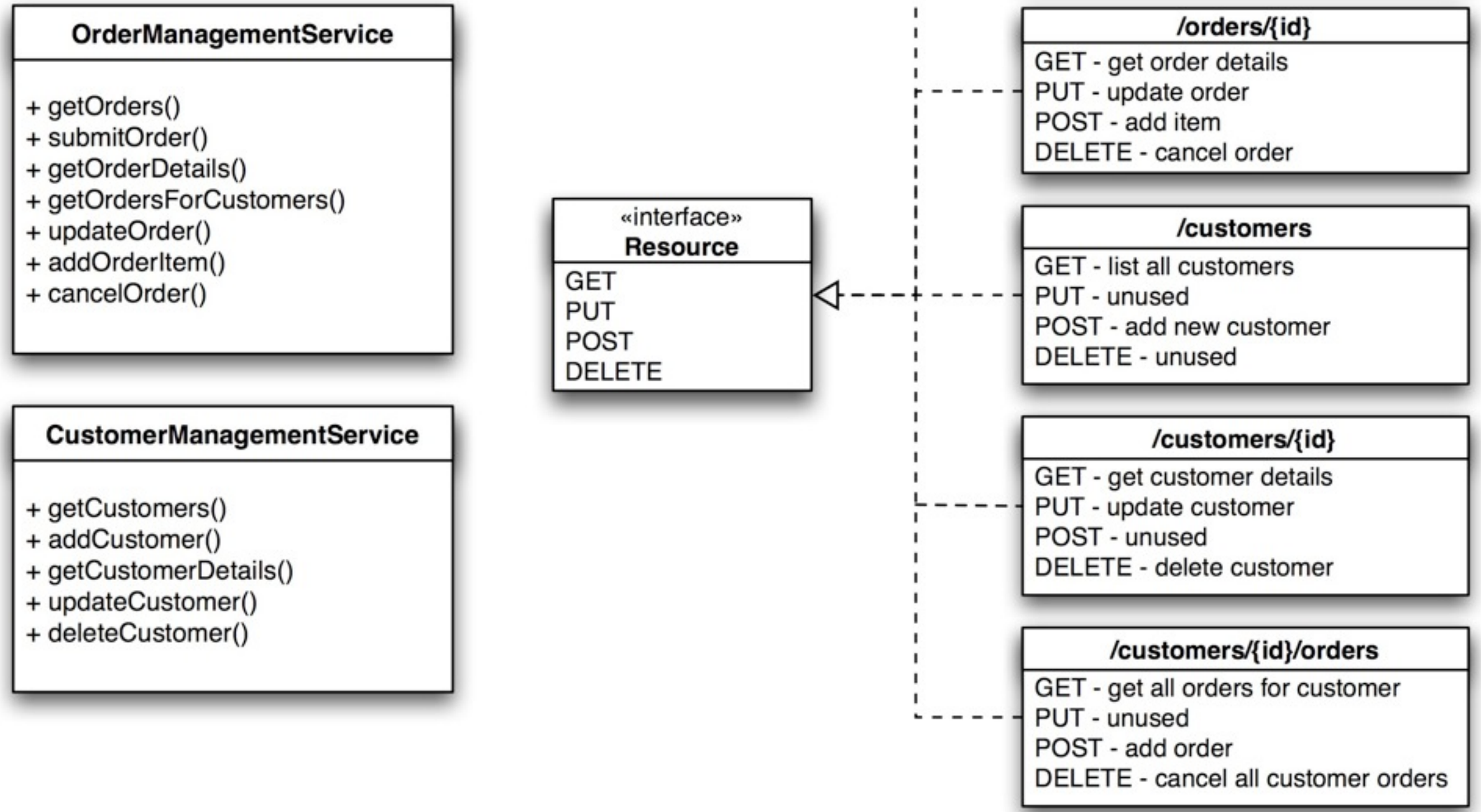
Design a RESTful API

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

- Start by identifying the **resources** - the **NAMES** in your system.
- Define the **structure of the URLs** that will be mapped to your resources.
- Define the **semantic of the operations** that you want to support on all of your resources (you don’t want to support GET, POST, PUT, DELETE on all resources!).

- Some examples:
  - http://www.photos.com/users/oliechti identifies a resource of type "user". A client can do a "HTTP GET" to obtain a representation of the user or a “HTTP PUT” to update the user.
  - http://www.photos.com/users identifies a resource of type "collection of users". A client can do a "HTTP POST" to add users, or an "HTTP GET" to obtain the list of users.

RPC vs REST

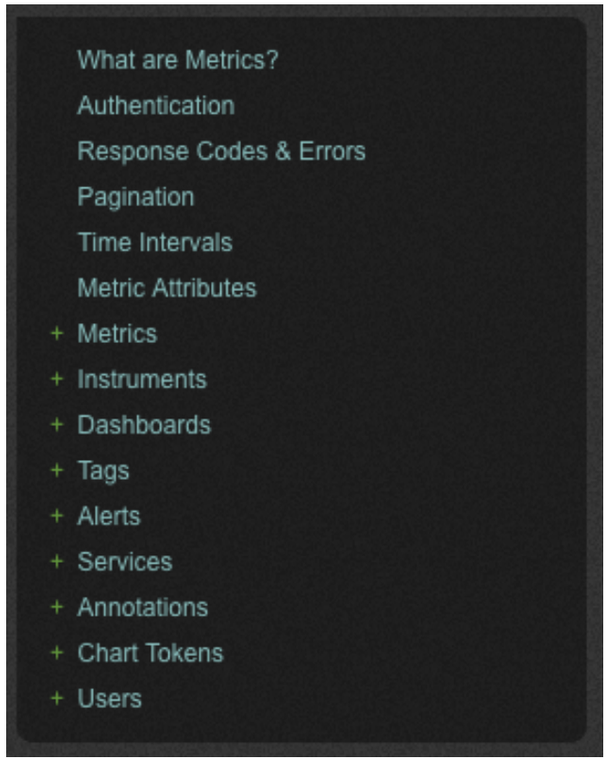

OrderManagementService
+ getOrders()
+ submitOrder()
+ getOrderDetails()
+ getOrdersForCustomers()
+ updateOrder()
+ addOrderItem()
+ cancelOrder()
CustomerManagementService
+ getCustomers()
+ addCustomer()
+ getCustomerDetails()
+ updateCustomer()
+ deleteCustomer()
«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders

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

If you have a verb in your URI, you are probably doing something wrong!

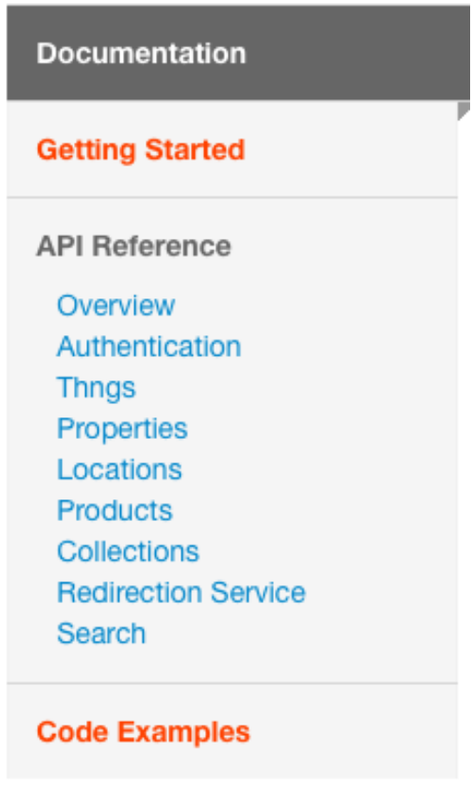
/api/students/create	POST /api/students/ HTTP/1.1
/api/students/22/delete	DELETE /api/students/22 HTTP/1.1

Look at Some Examples



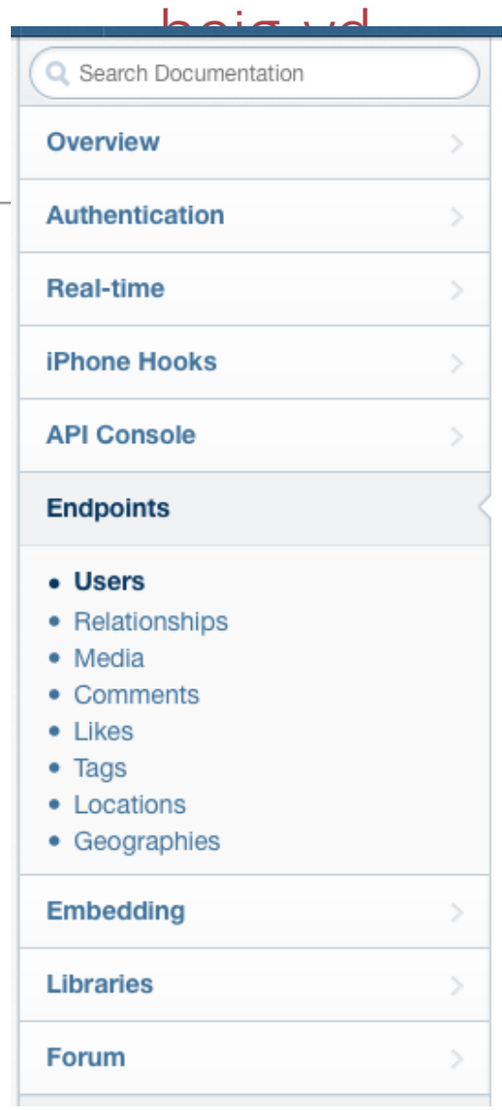
http://dev.librato.com/v1

librato



https://dev.evrythng.com/documentation/api

EVRYTHNG™



http://instagram.com/developer/endpoints/

Instagram

Short description of the resource (domain model)

heig-vd

What Are Metrics?

Metrics are custom measurements stored in Librato's Metrics service. These measurements are created and may be accessed programatically through a set of RESTful API calls. There are currently two types of metrics that may be stored in Librato Metrics, `gauges` and `counters`.

Gauges

Gauges capture a series of measurements where each measurement represents the value under observation at one point in time. The value of a gauge typically varies between some known minimum and maximum. Examples of gauge measurements include the requests/second serviced by an application, the amount of available disk space, the current value of $AAPL, etc.

Counters

Counters track an increasing number of occurrences of some event. A counter is unbounded and always monotonically increasing in any given *run*. A new *run* is started anytime that counter is reset to zero. Examples of counter measurements include the number of connections made to an app, the number of visitors to a website, the number of a times a write operation failed, etc.

Metric Properties

Some common properties are supported across all types of metrics:

`name`

Each metric has a name that is unique to its class of metrics e.g. a gauge name must be un
gauges. The name identifies a metric in subsequent API calls to store/query individual meas
can be up to 63 characters in length. Valid characters for metric names are 'A-Za-z0-9.:-_'.

`period`

The `period` of a metric is an integer value that describes (in seconds) the standard reportin
metric. Setting the period enables Metrics to detect abnormal interruptions in reporting and

Examples & payload structure

Examples

Return the metric named *cpu_temp* with up to four measurements at resolution 60.

```
curl \
  -u <user>:<token> \
  -X GET \
  https://metrics-api.librato.com/v1/metrics/cpu_temp?resolution=60&count=4
```

Response Code

```
200 OK
```

Response Headers

```
** NOT APPLICABLE **
```

Response Body

```
{
  "type": "gauge",
  "display_name": "cpu_temp",
  "resolution": 60,

  : {
  e.com": [

    1,
  _time": 1234567890,
    84.5

    1,
  _time": 1234567950,
    86.7

    1,
  _time": 1234568010,
    84.6

    1,
  _time": 1234568070,
    89.7

{
": 0,
nsform": null,
ts_short": "&#176;F",
ua": "librato-metrics/0.7.4 (ruby; 1.9.3p194; x86_64-linux) direct-faraday/0.8.4
": null,
ts_long": "Fahrenheit",
cked": true

  "Current CPU temperature in Fahrenheit",
emp"
```

CRUD method description

GET /v1/metrics/:name

API VERSION 1.0

Description

Returns information for a specific metric. If time interval search parameters are specified will also include a set of metric measurements for the given time span.

URL

```
https://metrics-api.librato.com/v1/metrics/:name
```

Method

```
GET
```

Measurement Search Parameters

If optional time interval search parameters are specified, the response includes the set of metric measurements covered by the time interval. Measurements are listed by their originating *source* name if one was specified when the measurement was created. All measurements that were created without an explicit source name are listed with the source name `unassigned`.

`source`

Deprecated: Use `sources` with a single source name, e.g `[mysource]`.

`sources`

If `sources` is specified, the response is limited to measurements from those sources. The `sources` parameter should be specified as an array of source names. The response is limited to the set of sources specified in the array.

navigator

- What are Metrics?
- Authentication
- Response Codes & Errors
- Pagination
- Time Intervals
- Metric Attributes
- \- Metrics
  - GET /metrics
  - POST /metrics
  - DELETE /metrics
  - GET /metrics/:name
  - PUT /metrics/:name
  - DELETE /metrics/:name
- \+ Instruments
- \+ Dashboards
- \+ Tags
- \+ Alerts
- \+ Services
- \+ Annotations
- \+ Chart Tokens
- \+ Users

heig-vd

Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Short description of the whole domain model

Overview

The central data structure in our engine are `Thngs`, which are data containers to store all the data generated by and about any physical object. Various `Properties` can be attached to any Thng, and the content of each property can be updated any time, while preserving the history of those changes. Thngs can be added to various `Collections` which makes it easier to share a set of Thngs with other `Users` within the engine.

`Thng`

An abstract notion of an object which has location & property data associated to it. Also called Active Digital Identities (ADIs), these resources can model real-world elements such as persons, places, cars, guitars, mobile phones, etc.

`Property`

A Thng has various properties: arbitrary key/value pairs to store any data. The values can be updated individually at any time, and can be retrieved historically (e.g. "Give me the values of property X between 10 am and 5 pm on the 16th August 2012").

`Location`

Each Thng also has a special type of Properties used to store snapshots of its geographic position over time (for now only GPS coordinates - latitude and longitude).

`User`

Each interaction with the EVRYTHNG back-end is authenticated and a user is associated with each action. This dictates security access.

`Collection`

A collection is a grouping of Thngs. Coll
one collection.

CRUD method description

Creating a new Product

To create a new `Product`, simply POST a JSON document that describes a product to the `/products` endpoint.

```
POST /products
Content-Type: application/json
Authorization: $EVRYTHNG_API_KEY
```

```
{
  *"fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
    ... },
  "tags": [<String>, ...]
}
```

Mandatory Parameters

`fn`

<String> The functional name of the product.

Optional Parameters

`description`

<String> An string that gives more details about the product, a short description.

More details about the Product resource (domain model) & payload structure

Products

Products are very similar to thngs, but instead of modeling an individual object instance, products are used to model a class of objects. Usually, they are used for general classes of thngs, usually a particular model with specific characteristics. Let's take for example a specific TV model (e.g. this one), which has various properties such as a model number, a description, a brand, a category, etc. Products are useful to captor the properties that are common to a set of thngs (so you don't replicate a property "model name" or "weight" for thousands of thngs that are individual instances of a same product category).

The Product document model used in our engine has been designed to be compatible with the hProduct microformat, therefore it can easily be integrated with the hProduct data model and applications supporting microformats.

The Product document model is as follows:

```
<Product>={
  "id": <String>,
  "createdAt": <timestamp>,
  "updatedAt": <timestamp>,
  "fn": <String>,
  "description": <String>,
  "brand": <String>,
  "categories": [<String>, ...],
  "photos": [<String>, ...],
  "url": <String>,
  "identifiers": {
    <String>: <String>,
    ... },
  "properties": {
    <String>: <String>,
        ... },
  "tags": [<String>, ...]
}
```

Cross-cutting concerns

Pagination

Requests that return multiple items will be paginated to 30 items by default. You can specify further pages with the `?page` parameter. You can also set a custom page size up to 100 with the `?per_page` parameter.

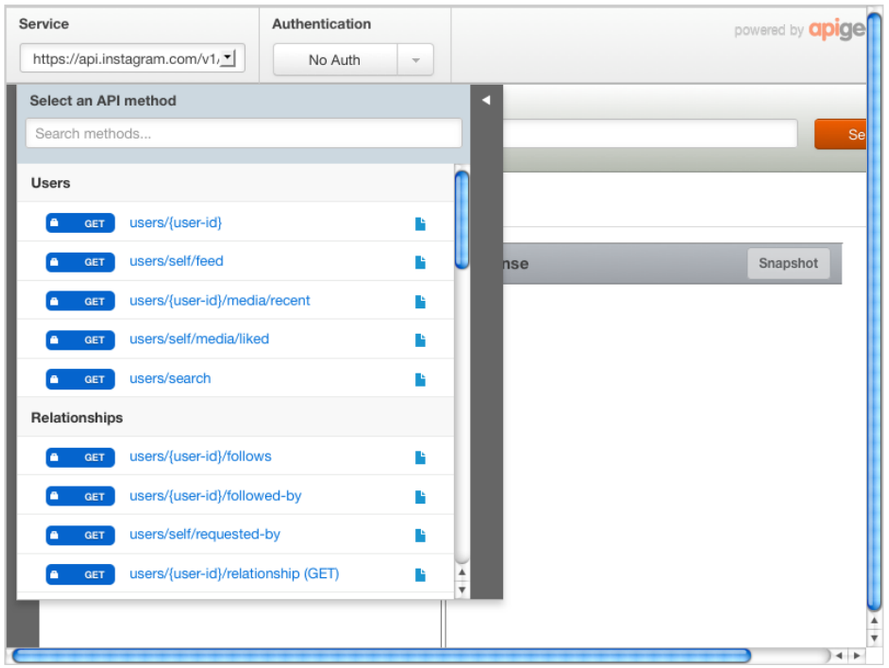
Authentication

Access to our API is done via HTTPS requests to the `https://api.evrythng.com` domain. Unencrypted HTTP requests are accepted (`http://api.evrythng.com` for low-power device without SSL support), but we **strongly** suggest to use only HTTPS if you store any valuable data in our engine. Every request to our API must include an API key using `Authorization` HTTP header to identify the user or application issuing the request and execute it if authorized.

Interactive test console

API Console

Our API console is provided by Apigee. Tap the Lock icon, select OAuth, and you can experiment with making requests to our API. See it in full screen →



List of supported CRUD methods for each resource (R, RW)

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

User Endpoints

GET	/users/ **user-id**	Get basic information about a user.
GET	/users/self/feed	See the authenticated user's feed.
GET	/users/ **user-id** /media/recent	Get the most recent media published by a user.
GET	/users/self/media/liked	See the authenticated user's list of liked media.
GET	/users/search	Search for a user by name.

Comment Endpoints

GET	/media/ **media-id** /comments	Get a full list of comments on a media.
POST	/media/ **media-id** /comments	Create a comment on a media. Please email apide…
DEL	/media/ **media-id** /comments/ **comment-id**	Remove a comment.

GET /media/ **media-id** /comments

https://api.instagram.com/v1/media/555/comments?access_token=ACCESS-TOKEN — RESPONSE


```
{
    "meta": {
        "code": 200
    },
    "data": [
        {
            "created_time": "1280780324",
            "text": "Really amazing photo!",
            "from": {
                "username": "snoopdogg",
                "profile_picture": "http://images.instagram.com/profiles/profile_16_75sq_1305612434.jpg",
                "id": "1574083",
                "full_name": "Snoop Dogg"
            },
            "id": "420"
        },
        ...
    ]
}
```


Get a full list of comments on a media.

Required scope: comments

CRUD method description

Cross-cutting concerns

Limits

Be nice. If you're sending too many requests too quickly, we'll send back a 503 error code (server unavailable).

You are limited to 5000 requests per hour per access_token or client_id overall. Practically, this means you should (when possible) authenticate users so that limits are well outside the reach of a given user.

PAGINATION

Sometimes you just can't get enough. For this reason, we've provided a convenient way to access more data in any request for sequential data. Simply call the url in the next_url parameter and we'll respond with the next set of data.

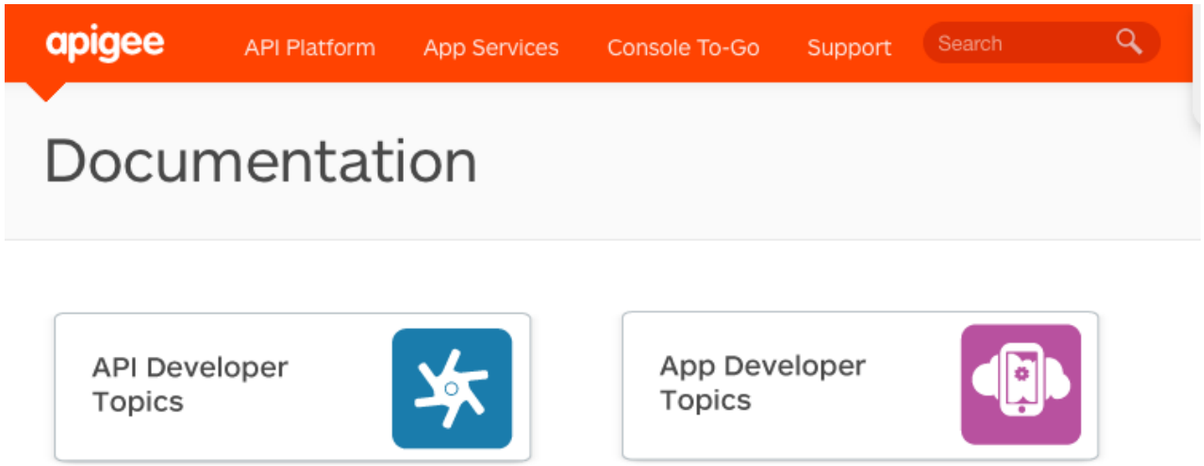
The Envelope

Every response is contained by an envelope. That is, each response has a predictable set of keys with which you can expect to interact:

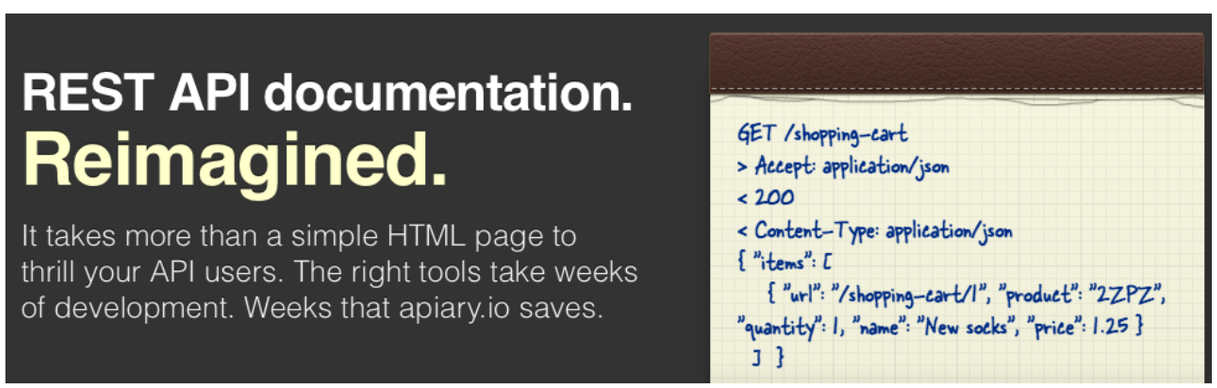

```
{
    "meta": {
        "code": 200
    },
    "data": {
        ...
    },
    "pagination": {
        "next_url": "...",
        "next_max_id": "13872296"
    }
}
```

Some Tools that Might Help/Inspire You

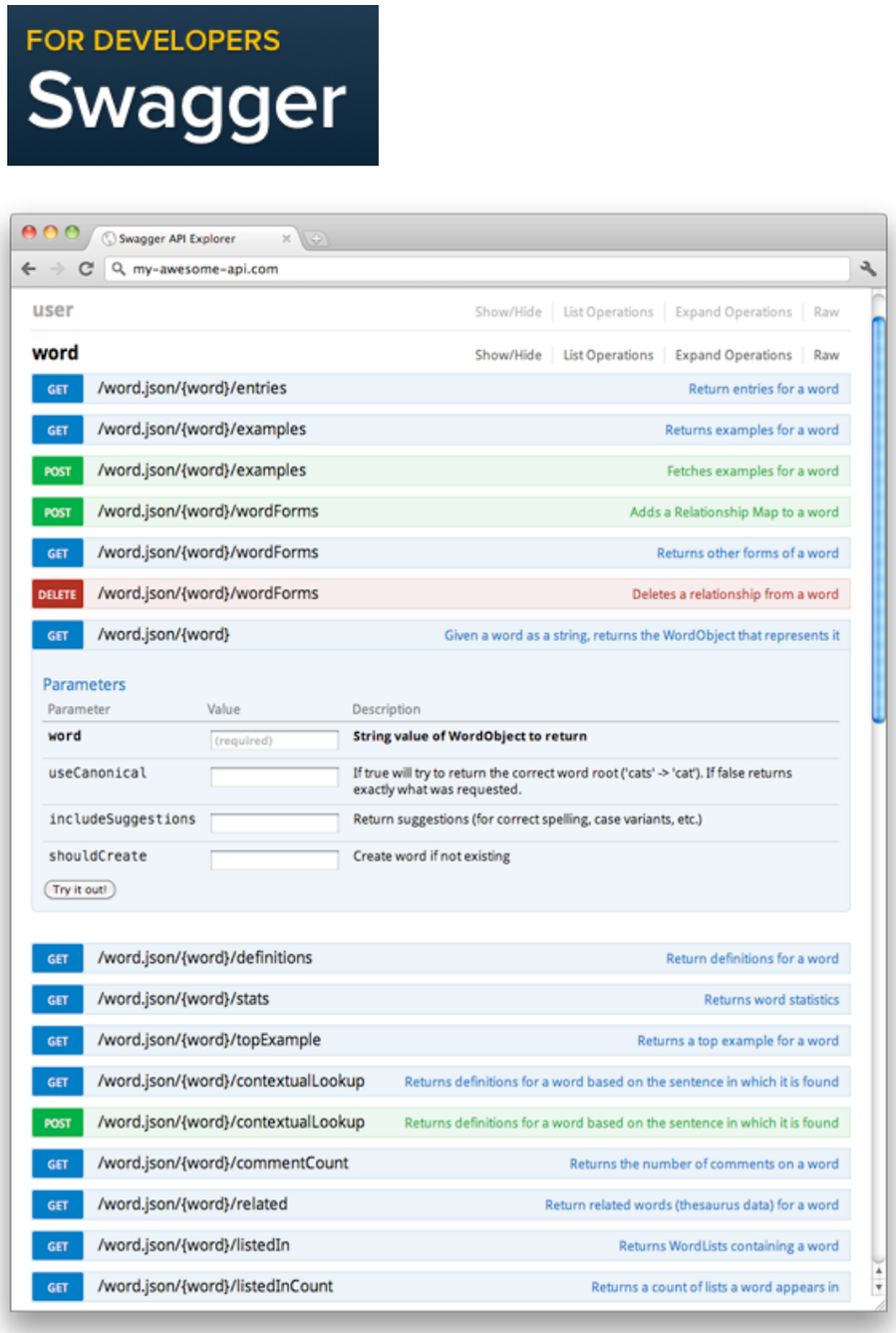
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



http://apigee.com/docs/



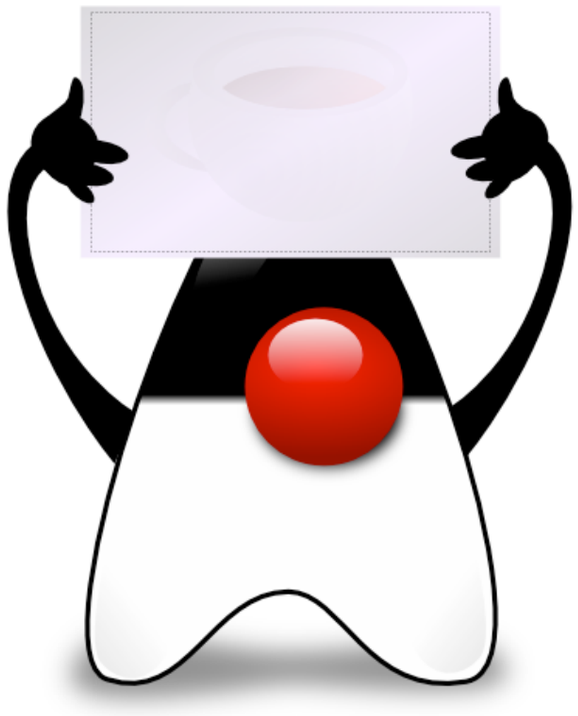
http://apiary.io/



https://developers.helloreverb.com/swagger/

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



RESTful APIs with JAX-RS

How to implement a REST API?

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

- On the **server side**, one could do everything in a **FrontController** servlet:
  - Parse URLs
  - Do a mapping between URLs and Java classes that represent resources
  - Generate the different representations of resources
  - etc.
- But of course, there are frameworks that do exactly that for us.
- It is true for nearly every platform and language, including Java.
- There is even a JSR for that: **JAX-RS** (JSR 311, JSR 339).
  - Oracle provides the reference implementation, in the **Jersey project** (open source). This is the implementation that you get with **Glassfish**.

Java EE, JSRs and implementations

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

IBM WebSphere

JBoss by Red Hat

PostgreSQL driver

OpenJPA

Apache CXF

WildFly

ORACLE driver

HIBERNATE

REST Easy

Glassfish

MySQL driver

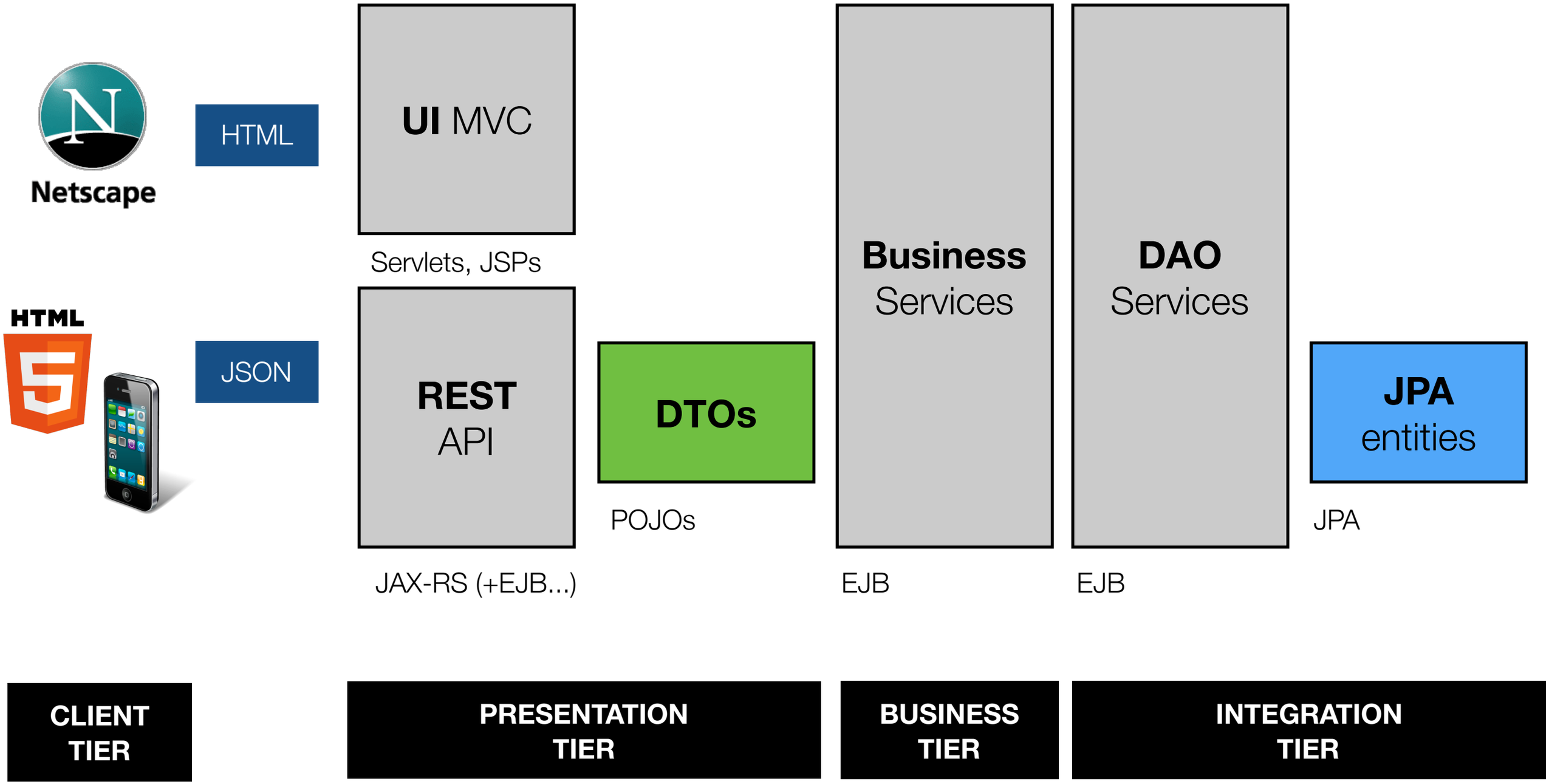
eclipselink

Jersey
RESTful Web Services in Java.

Java EE | **JDBC** | **JPA** | **JAX-RS**

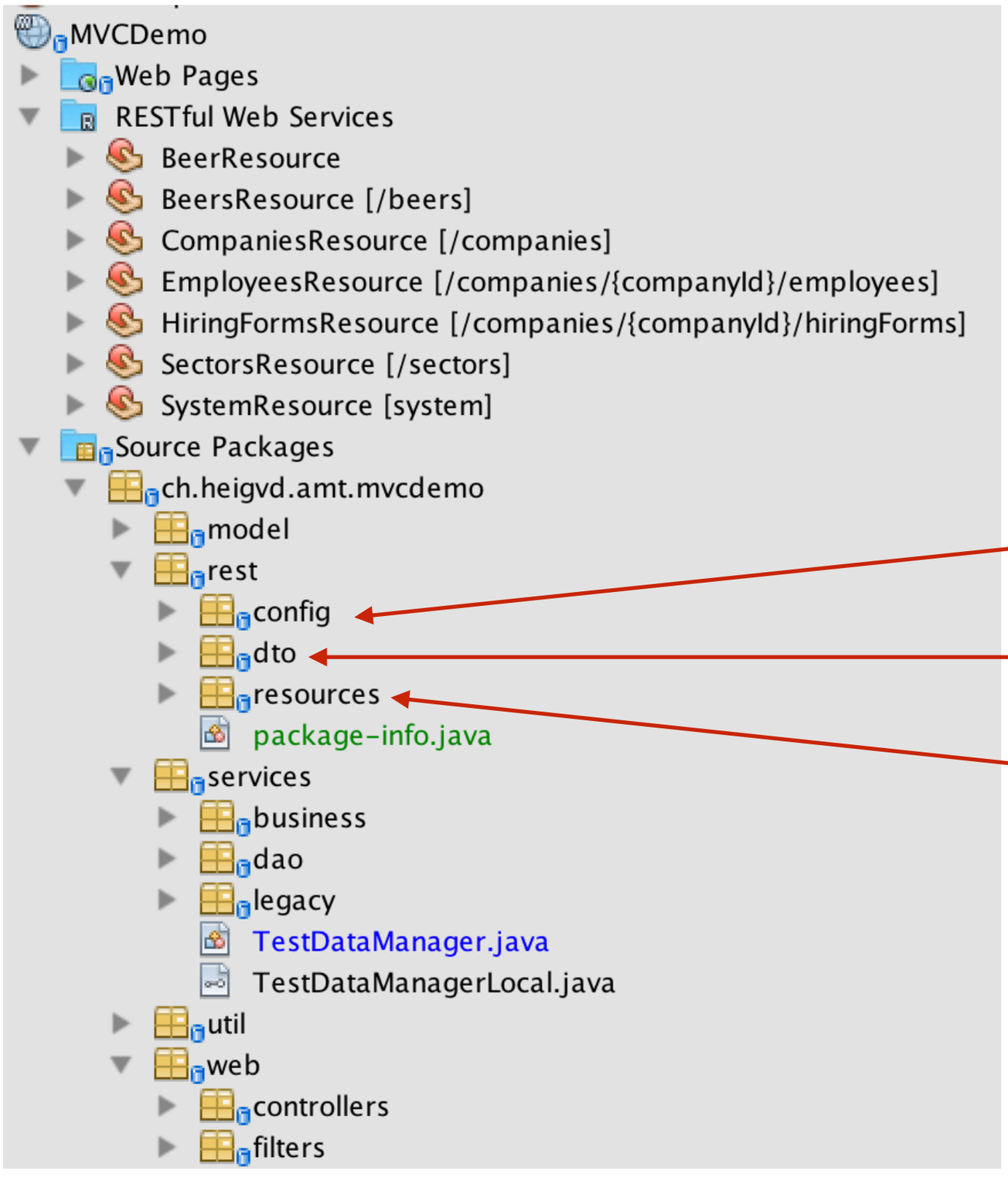
End-to-end Reference Architecture

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



End-to-end Reference Architecture

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



JAX-RS plumbing...

Objects **received from** / **sent to** clients via the REST API (JSON)

JAX-RS resource classes

JAX-RS 101

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

- JAX-RS is another example of applying **Inversion of Control** (IoC) in Java EE.
- As a developer, you create **Resource Classes** for your API endpoints:

`GET /users/ HTTP/1.1`	`UsersResource.java`
`POST /students/23/grades HTTP/1.1`	`StudentsResource.java`

- With various **annotations**, you tell the application server which classes and which methods should be invoked when HTTP requests are sent by clients:
  - Depending on the **URL**
  - Depending on the **HTTP method**
  - Depending on the **Accept** or **Content-type header**
- This is very similar what we have seen with Servlets in the past (with web.xml and @WebServlet)

See https://docs.oracle.com/javaee/7/tutorial/jaxrs002.htm#GILIK

JAX-RS 101

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

```
@Stateless
@Path("/beers")
public class BeersResource {

  @EJB
  BeersManagerLocal beersManager;

  @GET
  @Produces("application/json")
  public List<Beer> getBeers() {
    return beersManager.getAllBeers();
  }

  @POST
  @Consumes("application/json")
  public long addBeer(Beer beer) {
    return beersManager.add(beer);
  }
}
```

Even if we are conceptually in the presentation tier, we declare our JAX-RS resource class as a Stateless Session Bean.

This allows us to inject EJBs in the class. This also facilitates the use of the JPA persistence context (transaction starts when we enter the JAX-RS method)

IN THIS CASE, **DO NOT DEFINE A LOCAL INTERFACE** (it makes things a bit more complicated...)

```
GET /beers/ HTTP/1.1
Host: localhost:8080
Accept: application/json
```

```
POST /beers/ HTTP/1.1
Host: localhost:8080
Content-type: application/json

{
  "name" : "Cardinal",
  "country" : "Switzerland"
}
```

JAX-RS 101

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

- When you implement a REST API, you have to:
  - **serialize** Java objects into JSON (for GET methods)
  - **deserialize** JSON into Java objects (for POST/PUT/PATCH)
  - **JAX-RS** (with the help of **Jackson**, **JAXB** and other friends...) handles the serialization and deserialization for you.

```
@GET
@Produces("application/json")
public List<Beer> getBeers() {}

@POST
@Consumes("application/json")
public long addBeer(Beer beer)
```

See https://jersey.java.net/documentation/latest/user-guide.html#json
https://jersey.java.net/documentation/latest/user-guide.html#json.jackson

JAX-RS 101

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

- JAX-RS provides you with annotations to pass request attributes to your callback methods

```
@Stateless
@Path("/students")
public class StudentsResource {

   @GET
   @Path("/{studentId}/grades")
   @Produces("application/json")
   public List<Grade> getGrades(
      @PathParam(value="studentId") Long studentId,
      @QueryParam("ifLessThan") Double threshold
     ) {
       ...
     }
   }
}
```

```
GET /students/73/grades?ifLessThan=4 HTTP/1.1
Host: localhost:8080
Accept: application/json
```

See https://docs.oracle.com/javaee/7/tutorial/jaxrs002.htm#GIPYW

The Data Transfer Object (DTO) pattern

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

- **Requirement**: we need Java classes that capture the state of our business domain entities (Students, Companies, etc.), so that we can **exchange the related information with HTTP clients**.
- **Situation**: we have already created business entities in the `model` package. These are our JPA entities, which we use to exchange information between the services and the database.
- **Question**: can we simply r**euse these JPA** classes for the communication between the REST API and the clients?
- **Answer: not everybody agrees!**
- **Answer: I personally strongly recommend not to do that.**

The Data Transfer Object (DTO) pattern

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

- **Pattern description**: when you apply the DTO design pattern:
  - You introduce a package, where you define **simple Java Beans** (POJOs) with properties, getters and setters.
  - You create instances of these POJOs in the REST API layer and send them to clients (for GET requests). Typically, your business / DAO services give your JPA entities, **which you transform into DTOs**.
  - In the other direction, JAX-RS deserializes JSON into DTO instances. You use properties of these DTOs when you invoke business / DAO services (**you create the JPA entities and initialize them** with the content of the DTOs).

The Data Transfer Object (DTO) pattern

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

- **Pattern benefits**: there are **at least 4 reasons** for applying the pattern:
  - Control on the **visibility** of your data (**security**, **confidentiality**). In the MVCDemo project, employees have a **salary**. While it is necessary to have this property in the JPA entity (because it has to be stored in the database), it is clearly not something that you want to leak out via your REST API.
  - Have **full control on the data structure** presented to your clients. **Your API will be cleaner and easier to use**.
  - **Reduce the chattiness** and **improve the performance** of your clients applications. If you use JPA entities, then it is likely that REST clients will need to send a lot of HTTP requests to get all components of a page (1 call for the company, n calls for the sectors, etc.). With a DTO layer, you can aggregate multiple small business entities into a coarse-grained object that you send over the network.
  - **Avoid tricky technical issues**. If you try to use JPA entities, you will have to deal with circular references (@XmlTransient) and other issues. Trust me, you will spend a lot of time fighting with the underlying frameworks.

The Data Transfer Object (DTO) pattern

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

```
public class CompanySummaryDTO {

  private URI href;
  private List<String> sectors = new ArrayList<>();
  private String name;
  private long numberOfEmployees;
  private String ceo;

  public URI getHref() {
    return href;
  }

  public void setHref(URI href) {
    this.href = href;
  }

  public List<String> getSectors() {
    return sectors;
  }
...
}
```

```
@Stateless
@Path("/companies")
public class CompaniesResource {

  @GET
  @Produces("application/json")
  public List<CompanySummaryDTO> getCompanies() {
    List<CompanySummaryDTO> result = new ArrayList<>();
    List<Company> companies = companiesDAO.findAll();
    for (Company company : companies) {
      long companyId = company.getId();
      CompanySummaryDTO dto = new CompanySummaryDTO();
      populateSummaryDTOFromEntity(company, dto);
      result.add(dto);
    }
    return result;
  }

}
```

The Data Transfer Object (DTO) pattern

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

```
private CompanySummaryDTO populateSummaryDTOFromEntity(Company company, CompanySummaryDTO dto) {
  long companyId = company.getId();
  URI companyHref = uriInfo
    .getAbsolutePathBuilder()
    .path(CompaniesResource.class, "getCompany")
    .build(companyId);
  dto.setHref(companyHref);
  dto.setName(company.getName());
  dto.setNumberOfEmployees(companiesDAO.countEmployees(company.getId()));
  List<String> sectorsDTO = new ArrayList<>();
  for (Sector sector : company.getSectors()) {
    sectorsDTO.add(sector.getName());
  }
  dto.setSectors(sectorsDTO);

  List<Employee> employees = companiesDAO.findEmployeesByTitle(companyId, "CEO");
  if (employees.size() == 1) {
    Employee ceo = employees.get(0);
    dto.setCeo(ceo.getFirstName() + " " + ceo.getLastName());
  } else if (employees.isEmpty()) {
    dto.setCeo("There is no CEO");
  } else {
    dto.setCeo("There are " + employees.size() + " co-CEOs");
  }

  return dto;

}
```

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Designing RESTful APIs

URL Structure

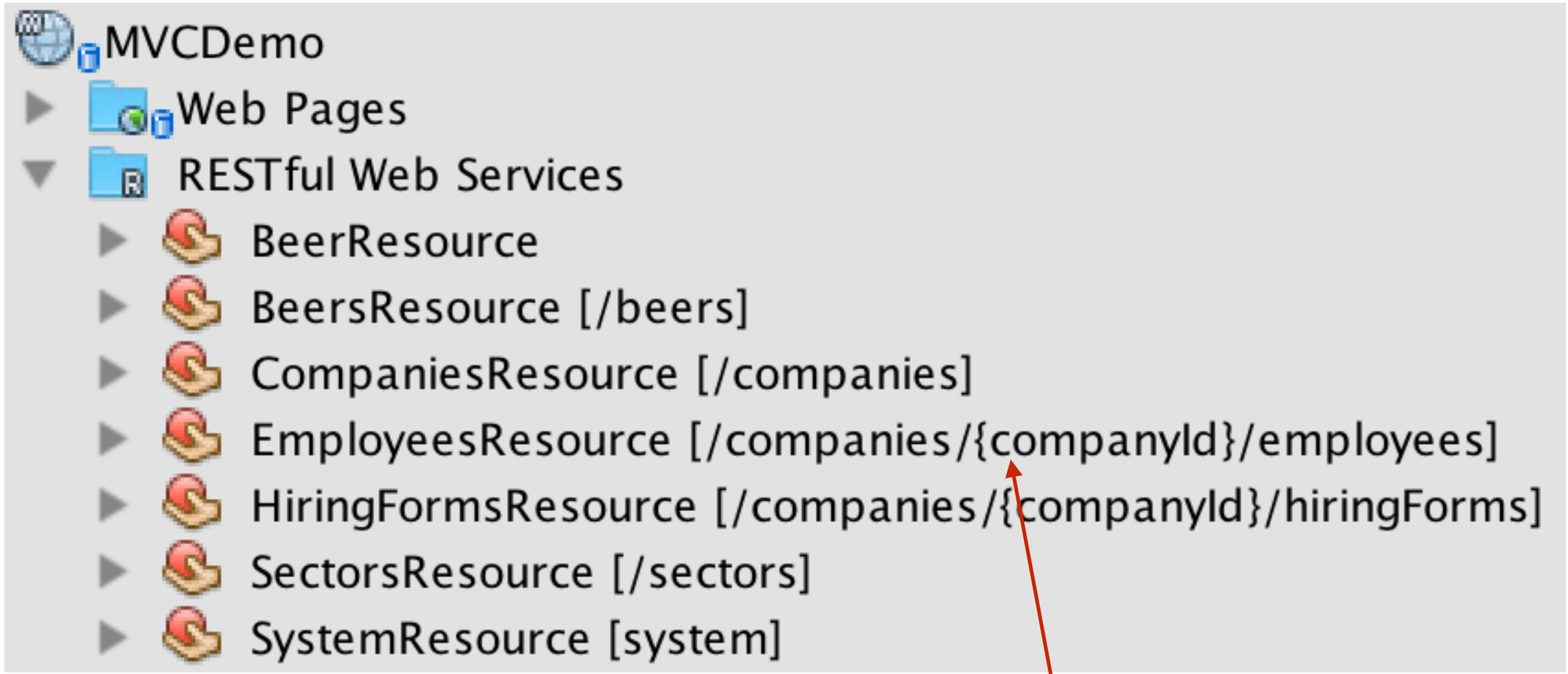
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- In most applications, you have **several types of resources** and there are relationships between them:
  - In a blog management platform, **Blog** authors create **BlogPosts** that can have associated **Comments**.
  - In a school management system, **Courses** are taught by **Professors** in **Rooms**.

- When you design your REST API, you have to define URL patterns to give access to the resources. There are often different ways to define these:
  - `/blogs/amtBlog/posts/892/comments/`
  - `/comments?blogId=amtBlog&postId=892`
  - `/professors/liechti/courses/`
  - `/courses?professorName=liechti`

URL Structure

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



Motivations

All employees belong to one company

Accessing all employees via `/employees` could make sense for the platform administrator, but we don't have a use case yet

URL Structure

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

```
@Stateless
@Path("/companies/{companyId}/employees")
public class EmployeesResource {

  @Context UriInfo uriInfo;
  @EJB private EmployeesDAOLocal employeesDAO;
  @EJB private CompaniesDAOLocal companiesDAO;

  @GET
  @Produces("application/json")
  public List<EmployeeSummaryDTO> getEmployees(@PathParam(value="companyId") long companyId) {
    List<EmployeeSummaryDTO> result = new ArrayList<>();
    List<Employee> employees = companiesDAO.findAllEmployeesForCompanyId(companyId);
    for (Employee employee : employees) {
      long employeeId = employee.getId();

      URI href = uriInfo.getAbsolutePathBuilder().path(EmployeesResource.class, "getEmployee").build(employeeId);

      EmployeeSummaryDTO dto = new EmployeeSummaryDTO();
      dto.setHref(href);
      dto.setFirstName(employee.getFirstName()); dto.setLastName(employee.getLastName()); dto.setTitle(employee.getTitle());
      result.add(dto);
    }
    return result;
  }

  @GET
  @Path("/{id}")
  @Produces("application/json")
  public EmployeeSummaryDTO getEmployee(@PathParam(value="id") long id) throws BusinessDomainEntityNotFoundException {
      Employee employee = employeesDAO.findById(id);
      return new EmployeeSummaryDTO(employee.getFirstName(), employee.getLastName(), employee.getTitle());
  }

}
```

Linked resources

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

- In most domain models, you have relationships between domain entities:
  - Example: one-to-many relationship between "Company" and "Employee"
- Imagine that you have the following REST endpoints:
  - `GET /companies/{id}` to retrieve one company by id
- Question: what payload do you expect when invoking this URL?

Linked resources

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

```
{
  "name": "Apple",
  "address" : {},
  "employees" : [
    {
      "firstName" : "Tim",
      "lastName" : "Cook",
      "title" : "CEO"
    },
    {
      "firstName" : "Jony",
      "lastName" : "Ive",
      "title" : "CDO"
    }
  ]
}
```

Embedding
(reduces "chattiness", often good if there are "few" linked resources; company-employee is not a good example)

```
{
  "name": "Apple",
  "address" : {},
  "employeeIds" : [134, 892, 918, 9928]
```

References via IDs
(not recommended: the client must know the URL structure to retrieve an an employee)

```
{
  "name": "Apple",
  "address" : {},
  "employeeURLs" : [
    "/companies/89/employees/134",
    "/companies/89/employees/892",
    "/companies/89/employees/918",
    "/contractors/255/employees/9928",
 ]
}
```

References via URLs
(better: decouples client and server implementation)

Linked resources

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

```
[
    {
        "href": "http://localhost:8080/MVCDemo/api/companies/1",
        "sectors": [
            "IT",
            "Telecommunications",
            "Entertainment"
        ],
        "name": "Apple",
        "numberOfEmployees": 85,
        "ceo": "Tim Cook"
    },
    {
        "href": "http://localhost:8080/MVCDemo/api/companies/9",
        "sectors": [
            "Sector-1444102164062-4",
            "Sector-1444102164062-5"
        ],
        "name": "Company-1444102164062-2",
        "numberOfEmployees": 0,
        "ceo": "There is no CEO"
    },
    {
        "href": "http://localhost:8080/MVCDemo/api/companies/4",
        "sectors": [
            "Financials"
        ],
        "name": "UBS",
        "numberOfEmployees": 0,
        "ceo": "There is no CEO"
    }
]
```

```
@Stateless
@Path("/companies")
public class CompaniesResource {

  @GET
  @Produces("application/json")
  public List<CompanySummaryDTO> getCompanies() {
    List<CompanySummaryDTO> result = new ArrayList<>();
    List<Company> companies = companiesDAO.findAll();
    for (Company company : companies) {
      long companyId = company.getId();
      CompanySummaryDTO dto = new CompanySummaryDTO();
      populateSummaryDTOFromEntity(company, dto);
      result.add(dto);
    }
    return result;
  }

}
```

Resources & Actions (1)

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

- In some situations, it is fairly easy to **identify resource** and to map related **actions** to HTTP request patterns.
- For instance, in an **academic management system**, one would probably come up with a Student resource and the associated HTTP request patterns:
  - `GET /students` to retrieve a list of students
  - `GET /students/{id}` to retrieve a student by id
  - `POST /students` to create a student
  - `PUT /students/{id}` to update a student
  - `DELETE /students/{id}` to delete a student

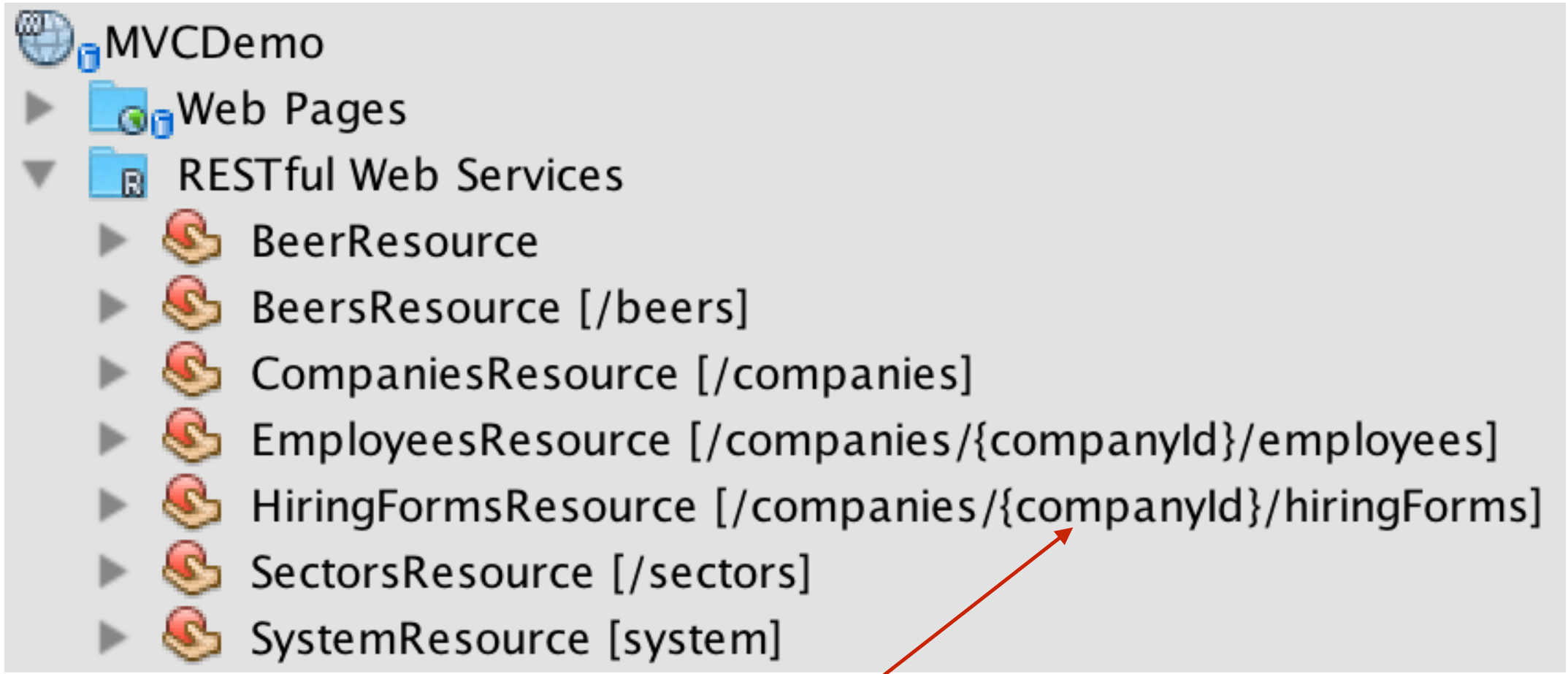
Resources & Actions (2)

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

- Some situations are not as clear and subject to debate. For instance, let us imagine that with your system, you can **exclude students** if they have cheated at an exam. How do you implement that with a REST API?
- Some people would propose something like this:
  - `POST /students/{id}/exclude`
    - Notice that “exclude” is a verb. In that case, there is no request body and we do not introduce a new resource (we only have student).
- Other people (like me) would prefer something like this:
  - `POST /students/{id}/exclusions/`
    - In that case, we have introduced a new resource: an exclusion request (think about a form that the Dean has to fill out and file). In that case, we would have a request body (with the reasons for the exclusion, etc.).

Resources & Actions (3)

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



When we hire an employee, we don't do a POST on /companies/23/employees. We don't do a POST on /employees/.

Instead, we create a HiringForm resource, which contains the employee details and the title. A business service processes this form

Resources & Actions (4)

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

```
@POST
@Consumes("application/json")
public Response submitHiringForm(HiringFormDTO hiringForm, @PathParam("companyId") long companyId)
  throws BusinessDomainEntityNotFoundException {

  Company company = companiesDAO.findById(companyId);
  Employee employee = humanResourcesService.hireEmployee(company, hiringForm);

  URI newHireURI = uriInfo
    .getBaseUriBuilder()
    .path(EmployeesResource.class)
    .path(EmployeesResource.class, "getEmployee")
    .build(company.getId(), employee.getId());

  return Response
    .created(newHireURI)
    .build();

}
```

Resources & Actions (4)

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

```
  @Override
  public Employee hireEmployee(Company company, HiringFormDTO hiringForm) throws
BusinessDomainEntityNotFoundException {

    company = companiesDAO.findById(company.getId());
    Employee newHire = new Employee();
    newHire.setFirstName(hiringForm.getFirstName());
    newHire.setLastName(hiringForm.getLastName());
    newHire.setTitle(hiringForm.getTitle());
    assignStartingSalary(newHire);

    newHire = employeesDAO.createAndReturnManagedEntity(newHire);
    companiesDAO.hire(company, newHire);
    return newHire;
  }

  private void assignStartingSalary(Employee employee) {
    String title = employee.getTitle();
    switch (title) {
      case "CEO":
        employee.setBasicSalary(Chance.randomDouble(1, 200000));
        employee.setBonus(Chance.randomDouble(50000, 5000000));
        break;
      case "software engineer":
        employee.setBasicSalary(80000);
        employee.setBonus(1000);
        break;
      default:
        employee.setBasicSalary(Chance.randomDouble(50000, 150000));
        employee.setBonus(Chance.randomDouble(0, 50000));
    }
```

Pagination (1)

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

- In most cases, you need to deal with **collections of resources that can grow** and where it is not possible to get the list of resources in a single HTTP request (for performance and efficiency reasons).
  - `GET /phonebook/entries?zip=1700`
- Instead, you want to be able to **successively retrieve chunks of the collection**. The typical use case is that you have a UI that presents a "page" of *n* resources, with controls to move to the previous, the next or an arbitrary page.
- In terms of API, it means that you want to be able to request a page, by providing an offset and a page size. In the response, you expect to find the number of results and a way to display navigation links.

Pagination (2)

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

- **At a minimum, what you need to do:**
  - When you **process an HTTP request**, you need a **page number** and a **page size**. You can use these to query a page from the database (do not transfer the whole table from the DB to the business tier!). You need to decide how the client is sending these values (query params, headers, defaults values).
  - When you generate the **HTTP response**, you need to **send the total number of results** (so that the client can compute the number of pages and generate the pagination UI), **and/or** send **ready-to-use links** that point to the first, last, prev and next pages. You use HTTP headers to send these informations.

```
Link: <https://api.github.com/user/repos?page=3&per_page=100>; rel="next",
  <https://api.github.com/user/repos?page=50&per_page=100>; rel="last"
```

Pagination (3)

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

- **Examples:**
  - http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#pagination
  - https://developer.github.com/v3/#pagination
  - https://dev.evrythng.com/documentation/api

http://tools.ietf.org/html/rfc5988#page-6

```
Link: <https://api.github.com/user/repos?page=3&per_page=100>; rel="next",
  <https://api.github.com/user/repos?page=50&per_page=100>; rel="last"
```

Pagination (4)

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

- **Example:**

Pagination

When retrieving a collection the API will return a paginated response. The pagination information is made available in the `X-Pagination` header containing four values separated by semicolons. These four values respectively correspond to the number of items per page, the current page number (starting at 1), the number of pages and the total number of elements in the collection.

For instance, the header `X-Pagination: 30;1;3;84` has the following meaning:

- `30` : There are 30 items per page
- `1` : The current page is the first one
- `3` : There are 3 pages in total
- `84` : There is a total of 84 items in the collection

To iterate through the list, you need to use the `page` and `pageSize` query parameters when doing a `GET` request on a collection. If you do not specify those parameters, the default values of 1 (for `page`) and 30 (for `pageSize`) will be assumed.

Example: The request `GET /myResources?page=2&pageSize=5 HTTP/1.1` would produce a response comparable to the following:

```
HTTP/1.1 200 OK
X-Pagination: 5;2;7;35
...

{
  [
    { "id": 6 },
    { "id": 7 },
    { "id": 8 },
    { "id": 9 },
    { "id": 10 }
  ]
}
```

Sorting and Filtering

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

- Most REST APIs provide a mechanism to sort and filter collections.
- Think about GETting the list of all students who have a last name starting with a ‘B’, or all students who have an average grade above a certain threshold.
- Think about GETting the list of all students, sorted by rank or by age.
- The standard way to specify the sorting and filtering criteria is to use query string parameters.
- IMPORTANT: be consistent across your resources. The developer of client applications should be able to use the same mechanism (same parameter names and conventions) for all resources in your API!

Authentication

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

- In most cases, REST APIs are invoked over a secure channel (**HTTPS**).
- For that reason, the **basic authentication scheme** is often considered acceptable.
- Every request contains an “Authorization” header that contains either **user credentials** (user id + password) or some kind of **access token** previously obtained by the user.
- When the server receives an HTTP request, it extracts the credentials from the HTTP header, validates them against what is stored in the database and either grants/rejects the access.

OAuth

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

- In many REST APIs, OAuth 2.0 is used for **authorization** and **access delegation**:
  - when you use a **Facebook Application** (e.g. a game), you are asked whether you agree to **authorize** this third-party Application to access some of **your Facebook data** (and actions, such as posting to your wall).
  - If you agree, the Facebook Application receives a **bearer token**. When it sends HTTP requests to the **Facebook API**, it sends this token in a HTTP header (typically in the Authorization header). Because the Application has a valid token, Facebook grants access to your data.
  - In other words, using OAuth is similar to handing your car keys to a concierge.

API versioning

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

- If you think about the **medium and long term evolution of your service** (think about Twitter), your API is very likely to evolve over time:
  - You may add new types of resources
  - You may add/remove query string parameters
  - You may change the structure of the payloads
  - You may introduce new mechanisms (authentication, pagination, etc.)

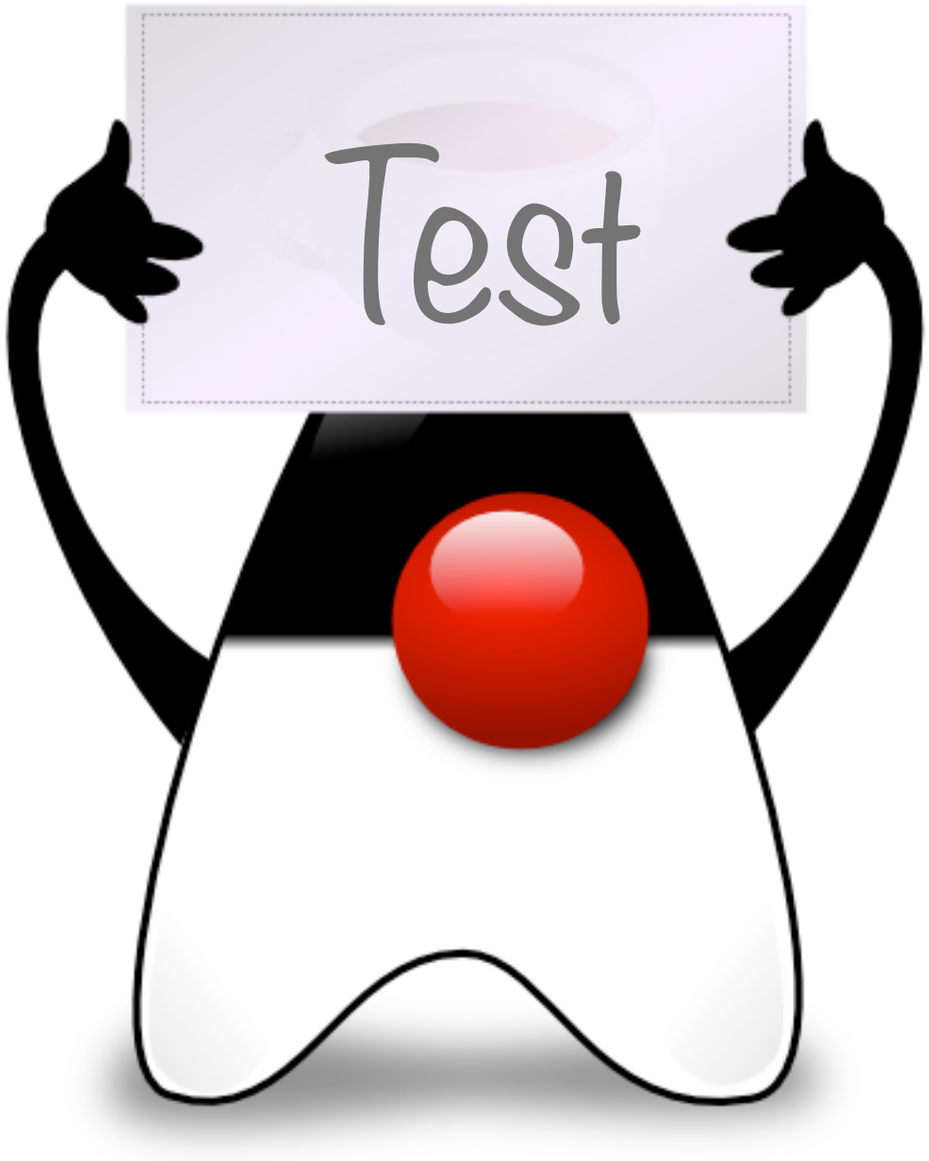
- When you introduce a change in your API (and in the corresponding documentation), you will have a **compatibility issue**. Namely, you will have to support **some clients that still use the old version** of the API and **others that use the new version of the API**.

- For this reason, when you receive an HTTP request, you need to know which version is used by the client talking to you. As usual, there are different ways to pass this information (path element, query string parameter, header).

- A lot of REST APIs include the version number in the path, e.g. `http://www.myservice.com/api/v2/students/7883`

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud



Testing REST API

Testing the REST API

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

- So far, we have already seen **different types tests**:
  - **Non functional tests** (performance, scalability, etc.) with **JMeter**
  - User acceptance tests (controlled browser) with **Selenium** & **WebDriver**
  - You already knew about JUnit tests
- There are different strategies, tools and frameworks to test the REST API. We have used one approach in the **MVCDemoUserAcceptanceTests** project:
  - We use the **Jersey Client** framework
  - This provides us with a **fluent API** that makes it easier to prepare HTTP requests and to inspect HTTP responses

Testing the REST API

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

Send an HTTP request

```
@Test
public void itShouldBePossibleToListCompanies() throws IOException {
  WebTarget target = client.target("http://localhost:8080/MVCDemo/api").path("companies");
  Response response = target.request().get();

  assertThat(response.getStatus()).isEqualTo(Response.Status.OK.getStatusCode());

  String jsonPayload = response.readEntity(String.class);
  assertThat(jsonPayload).isNotNull();
  assertThat(jsonPayload).isNotEmpty();

  JsonNode[] asArray = mapper.readValue(jsonPayload, JsonNode[].class);
  assertThat(asArray).isNotNull();
  assertThat(asArray.length).isNotEqualTo(0);

  for (JsonNode company : asArray) {
    assertThat(company.get("ceo")).isNotNull();
    assertThat(company.get("sectors")).isNotNull();
    assertThat(company.get("numberOfEmployees")).isNotNull();
  }
}
```

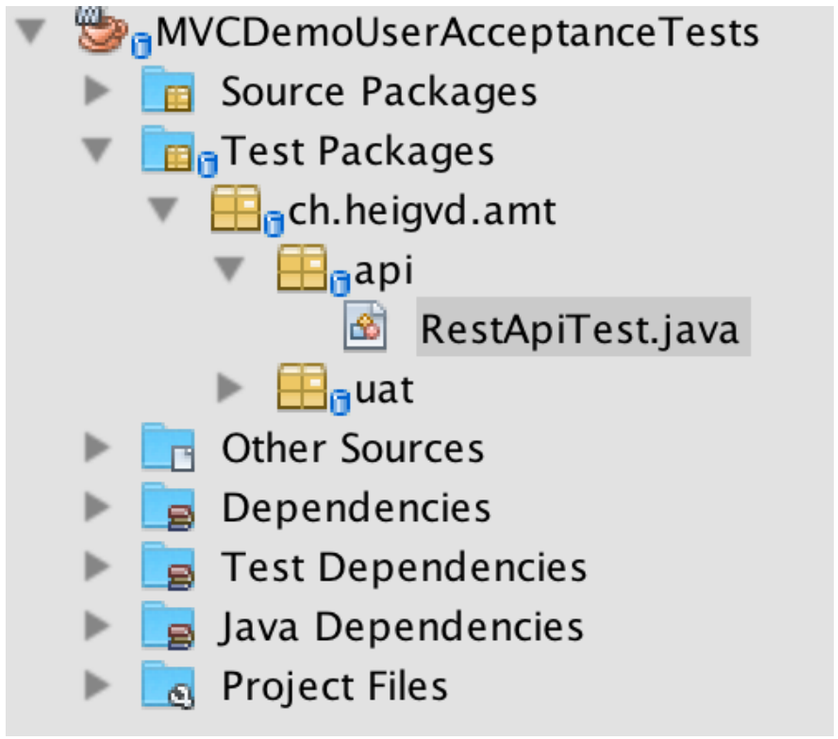
Check that we get a 200

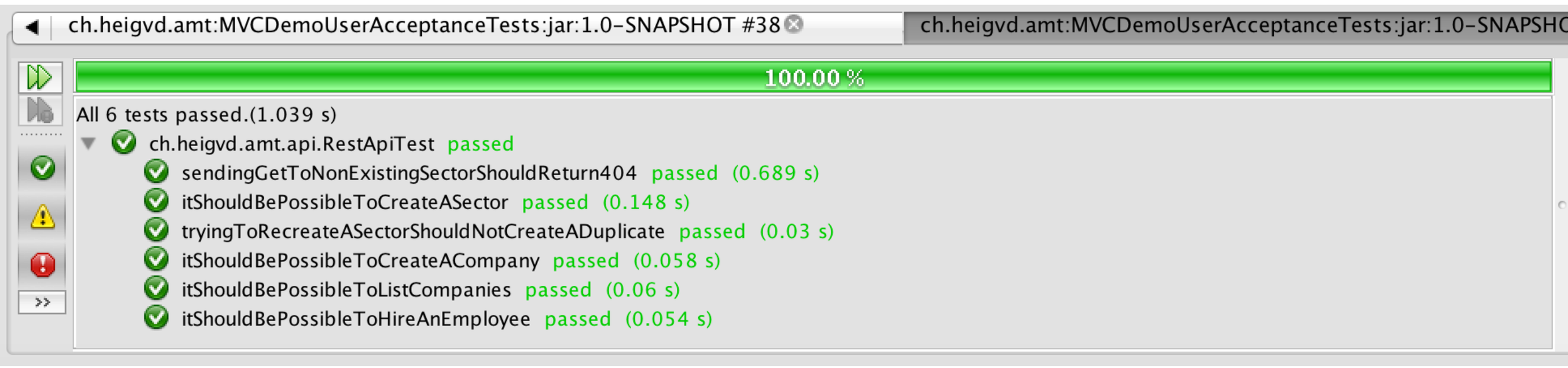
Get the JSON payload as string and parse it

Iterate over the array of companies and validate that JSON properties are there

Testing the REST API

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





Probe Dock

Olivier

https://trial.probedock.io/wasabi-technologies/reports

Probe Dock | Dashboard | Reports | Projects | Help | oliechti

Wasabi Technologies

Latest Reports | 09:18 by oliechti

Results	Runner(s)	Duration	Date	Details
6	oliechti	1s 90ms	Tue, Oct 6, 2015 9:18 AM	MVCDemoJavaEE 1.0.0 unit
6	oliechti	788ms	Tue, Oct 6, 2015 5:32 AM	MVCDemoJavaEE 1.0.0 unit
6	oliechti	785ms	Tue, Oct 6, 2015 5:30 AM	MVCDemoJavaEE 1.0.0 unit
5 1	oliechti	856ms	Tue, Oct 6, 2015 5:29 AM	MVCDemoJavaEE 1.0.0 unit
5 1	oliechti	915ms	Tue, Oct 6, 2015 5:18 AM	MVCDemoJavaEE 1.0.0 unit
5	oliechti	903ms	Tue, Oct 6, 2015 5:02 AM	MVCDemoJavaEE 1.0.0 unit
13	oliechti	37s 378ms	Mon, Oct 5, 2015 9:59 PM	MVCDemoJavaEE 1.0.0 unit
8	oliechti	34s 797ms	Mon, Oct 5, 2015 9:49 PM	MVCDemoJavaEE 1.0.0 unit

Probe Dock

Olivier

https://trial.probedock.io/wasabi-technologies/reports/mwqd2x4ay9ze

Probe Dock | Dashboard | Reports | Projects | Help | oliechti

Wasabi Technologies

Latest Reports | 09:18 by oliechti

Test Run Report - Tue, Oct 6, 2015 9:18 AM

MVCDemoJavaEE 1.0.0 | unit

Details

- 6 test results
- Run in 1s 90ms

Summary

Health

Rest api test: it should be possible to create a company

Duration: 59ms

Filter by result

Filter by category

All categories

Filter by ticket

All tickets

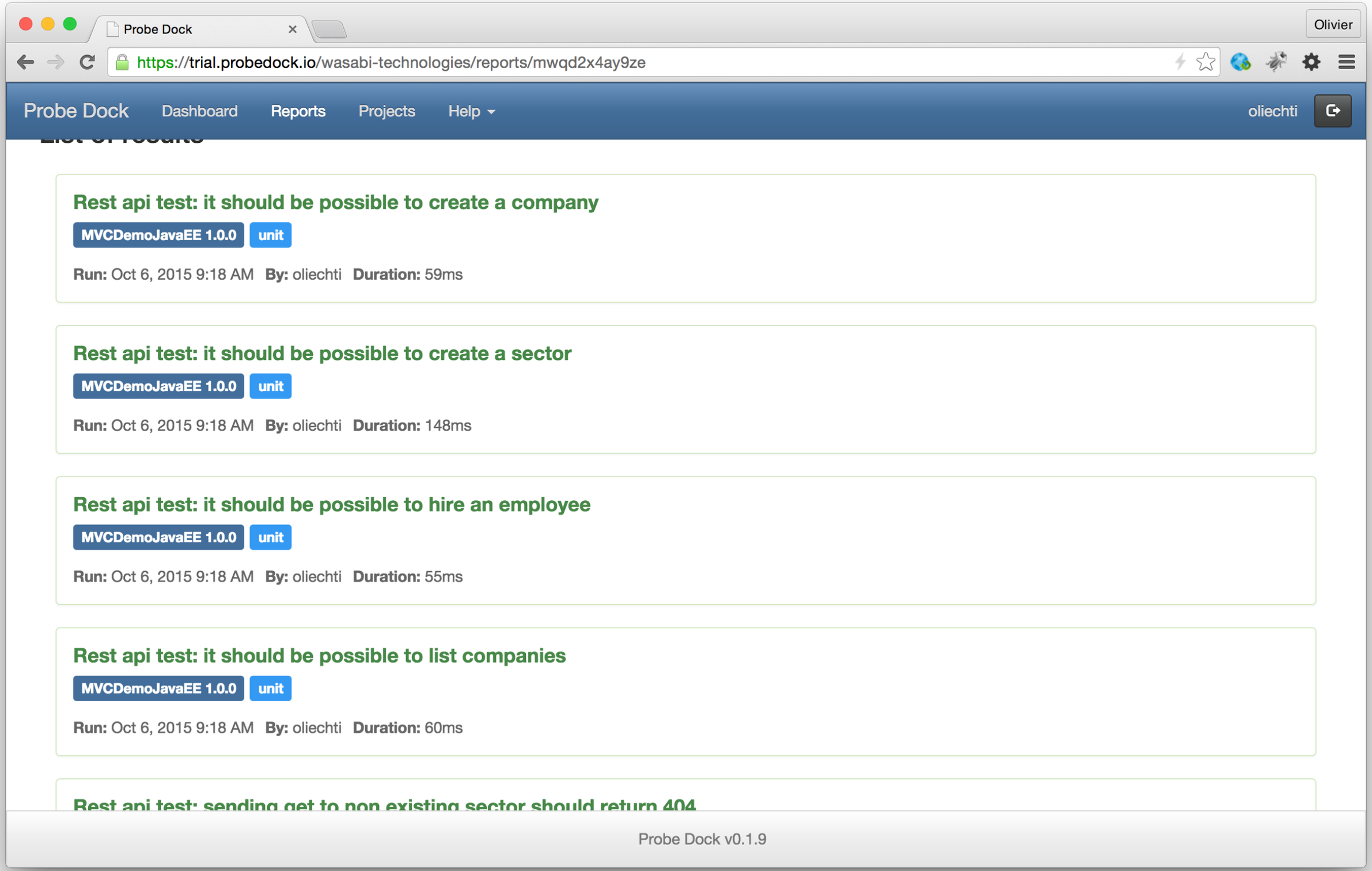
Filter by name

Any name

Filter by tag

All tags

Probe Dock v0.1.9


Probe Dock
Olivier
https://trial.probedock.io/wasabi-technologies/reports/mwqd2x4ay9ze
Probe Dock
Dashboard
Reports
Projects
Help
oliechti
Rest api test: it should be possible to create a company
MVCDemoJavaEE 1.0.0
unit
Run: Oct 6, 2015 9:18 AM By: oliechti Duration: 59ms
Rest api test: it should be possible to create a sector
MVCDemoJavaEE 1.0.0
unit
Run: Oct 6, 2015 9:18 AM By: oliechti Duration: 148ms
Rest api test: it should be possible to hire an employee
MVCDemoJavaEE 1.0.0
unit
Run: Oct 6, 2015 9:18 AM By: oliechti Duration: 55ms
Rest api test: it should be possible to list companies
MVCDemoJavaEE 1.0.0
unit
Run: Oct 6, 2015 9:18 AM By: oliechti Duration: 60ms
Probe Dock v0.1.9