Lecture 3: REST APIs

Olivier Liechti
TWEB

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

14h00 - 15h00	60'	**JavaScript 101 reminders** Scopes, asynchronous functions, closures **Lecture: introduction to RESTful APIs** Core concepts: resources and operations
15h00 - 15h10	*10'*	*Break*
15h10 - 16h25	75'	**JavaScript 101: asynchronous programming** Controlling the sequence of async operations **Lecture: REST API design issues & doc** URLs, actions, pagination, security, versioning Exercise: REST with Express.js and API Copilot

JavaScript 101
Reminder

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

How do I use asynchronous functions and callbacks in JavaScript?

JavaScript 101
Reminder

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

#1 setTimeout is an example of async function

```
var myCallback = function () {
  console.log("done");
};

setTimeout( myCallback, 1000);
// callback will be called in 1000 ms
```

#2 Many developers like concise syntax

```
setTimeout( function() {
  console.log("done")
}, 1000);
```

JavaScript 101
Reminder

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

#3 This is an asynchronous function

```
var myAsyncFunc = function (callbackWhenDone) {
    console.log("  Starting asynchronous process...")

    /*
     * do my job... might take a while; this function is asynchronous, because it
     * calls an asynchronous function before returning the result.
     */
    setTimeout( function() {
        var result = "done";
        console.log("  Asynchronous process is done.")
        callbackWhenDone(result);
    }, 1000);
};
```

#4 This is how a client can use it

```
var myCallback = function (output) {
    console.log("Cool, I know have the result: " + output);
};

console.log("before call");
myAsyncFunc(myCallback);
console.log("after call");
```

JavaScript 101 Reminder

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

#3 This is an asynchronous function

```
var myAsyncFunc = function (callbackWhenDone) {
    console.log("  Starting asynchronous process...")

    /*
     * do my job... might take a while; this function is asynchronous, because it
     * calls an asynchronous function before returning the result.
     */
    setTimeout( function() {
        var result = "done";
        console.log("  Asynchronous process is done.")
        callbackWhenDone(result);
    }, 1000);
};
```

This is called a little bit more than 1000 ms after myAsyncFunc is called

This is called 1000 ms after setTimeout is called

#4 This is how a client can use it

```
var myCallback = function (output) {
    console.log("Cool, I know have the result: " + output);
};

console.log("before call");
myAsyncFunc(myCallback);
console.log("after call");
```

This is called a little bit more than 1000 ms after "before call"

This is called immediately after "before call"

heig-vd

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

RESTful APIs

REST API

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

- A REST API is **a type of Web Service interface** (and as such, is an alternative to a SOAP/WSDL API).
- A REST API is based on the core components of the Web architecture:
  - It exposes **resources** (a user, a photo, a sensor, a measure, a lecture)
  - It uses **URLs** to identify and locate resources
    - `/api/users`, `/api/users/8282`, `/api/users/8282/photos/`, etc.
  - It uses **HTTP methods** to expose **operations** applicable to resources
    - GET, POST, PUT, DELETE, PATCH
  - It uses **content negotiation** and **resource representation formats** to transfer **machine-understandable payloads** (json, xml, etc.)

REST API

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

R → 9.3
C → 9.5
U → 9.6
D → 9.7

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

```
GET /api/lectures/ HTTP/1.1
Accept: application/json
```

```
GET /api/lectures/238 HTTP/1.1
Accept: application/json
```

C **READ** UD

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

```
GET /api/lectures/ HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-type: application/json

[
  {‘id’: ‘123’},
  {‘id’: ‘238’},
  {‘id’: ‘997’}
]
```

```
GET /api/lectures/238 HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-type: application/json

{
  ‘id’ : ‘238’,
  ‘title’ : ‘intro to mongodb’,
  ‘level’ : ‘beginner’
}
```

```
POST /api/lectures/ HTTP/1.1
Content-type: application/json

{
   ‘title’ : ‘intro to mongodb’,
   ‘level’ : ‘beginner’
}
```

```
PUT /api/lectures/238 HTTP/1.1
Content-type: application/json

  {
    'title' : 'intro to MongoDB',
    'level' : 'intermediate'
  }
```

Some APIs support the PATCH method for partial updates

CRU **DELETE**

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

```
DELETE /api/lectures/238 HTTP/1.1
```

Normal | Basic Auth | Digest Auth | OAuth 1.0 | No environment

http://localhost:9000/api/things/ GET URL params Headers (1)

Accept application/json Manage presets

Header Value

Send Preview Add to collection Reset

request

Body | Cookies (1) | Headers (8) | STATUS 200 OK | TIME 112 ms

response

Pretty | Raw | Preview | JSON | XML


```
[
    {
        "_id": "545630336aae96b6be2ab0b3",
        "name": "Smart Build System",
        "info": "Build system ignores `spec` files, allowing you to keep tests alongside code. Automatic injection of scripts and styles into your index.html",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b1",
        "name": "Development Tools",
        "info": "Integration with popular tools such as Bower, Grunt, Karma, Mocha, JSHint, Node Inspector, Livereload, Protractor, Jade, Stylus, Sass, CoffeeScript, and Less.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b4",
        "name": "Modular Structure",
        "info": "Best practice client and server structures allow for more code reusability and maximum scalability",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b2",
        "name": "Server and Client integration",
        "info": "Built with a powerful and fun stack: MongoDB, Express, AngularJS, and Node.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b5",
        "name": "Optimized Build",
        "info": "Build process packs up your templates as a single JavaScript payload, minifies your scripts/css/images, and rewrites asset names for caching.",
        "__v": 0
    },
    {
        "_id": "545630336aae96b6be2ab0b6",
        "name": "Deployment Ready",
        "info": "Easily deploy your app to Heroku or Openshift with the heroku and openshift subgenerators",
        "__v": 0
    }
]
```

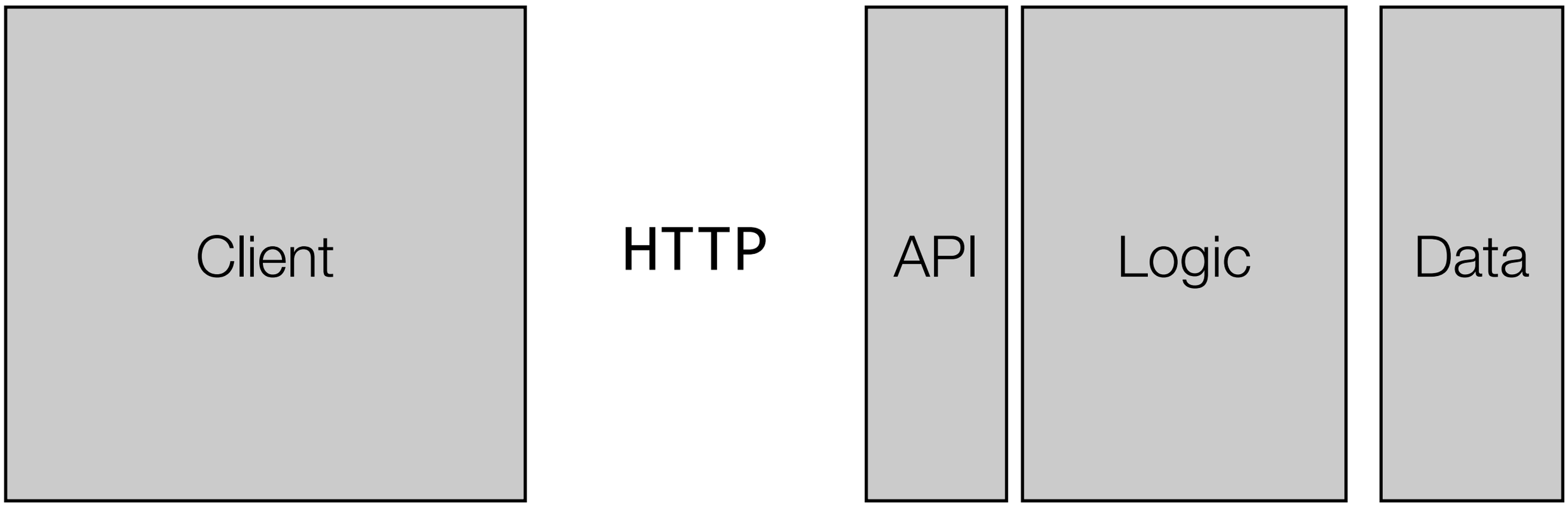

heig-vd

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

REST API

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

GET/POST/PUT/DELETE



JavaScript 101
Reminder

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

How do functions, scope and closures work in JavaScript?

JavaScript 101
Reminder

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

#1 There are 2 scopes in JavaScript (before ES6)

```
var globalVariable = "pollution...";
var f = function() {
  var localVariable = "good";
  globalVariable2 = "bad"; // "use strict" prevents this type of error
};
```

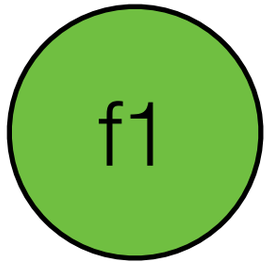
#2 Functions can be nested. This creates a scope chain.

```
var vInGlobalScope;
var f1 = function() {
  var vInScope1;
  var f2 = function() {
    var vInScope2;
    var f3 = function() {
      var vInScope3;
    };
  };
};
```

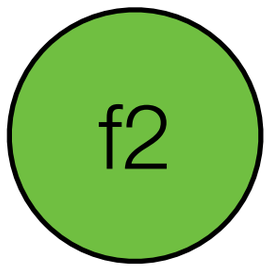
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#3 Every function is an object

```
var f1 = function() {
  console.log("hello");
};
```




```
function f2() {
  console.log("hello");
};
```




```
var g = function() {
  return function() {};
}
var f3 = g();
```

JavaScript 101
Reminder

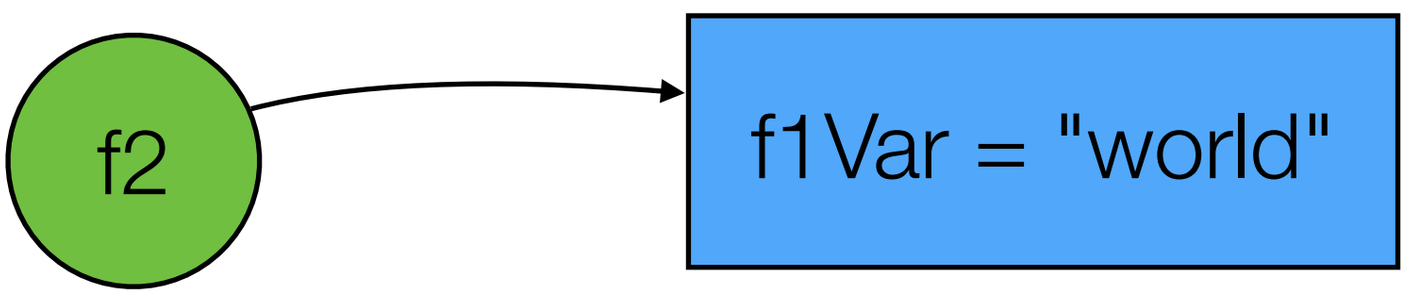
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

#3 A function which access a variable from an outer scope creates a **closure**. It keeps a reference to the outer variables forever.

```
var gVar = "world";

var f1 = function() {
    var f1Var = "happy";
    return function() {
        var f2Var = "hello"
        console.log(f1Var + " " + f2Var + " " + gVar);
    }
};

var f2 = f1()
f2();
```



JavaScript 101
Reminder

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

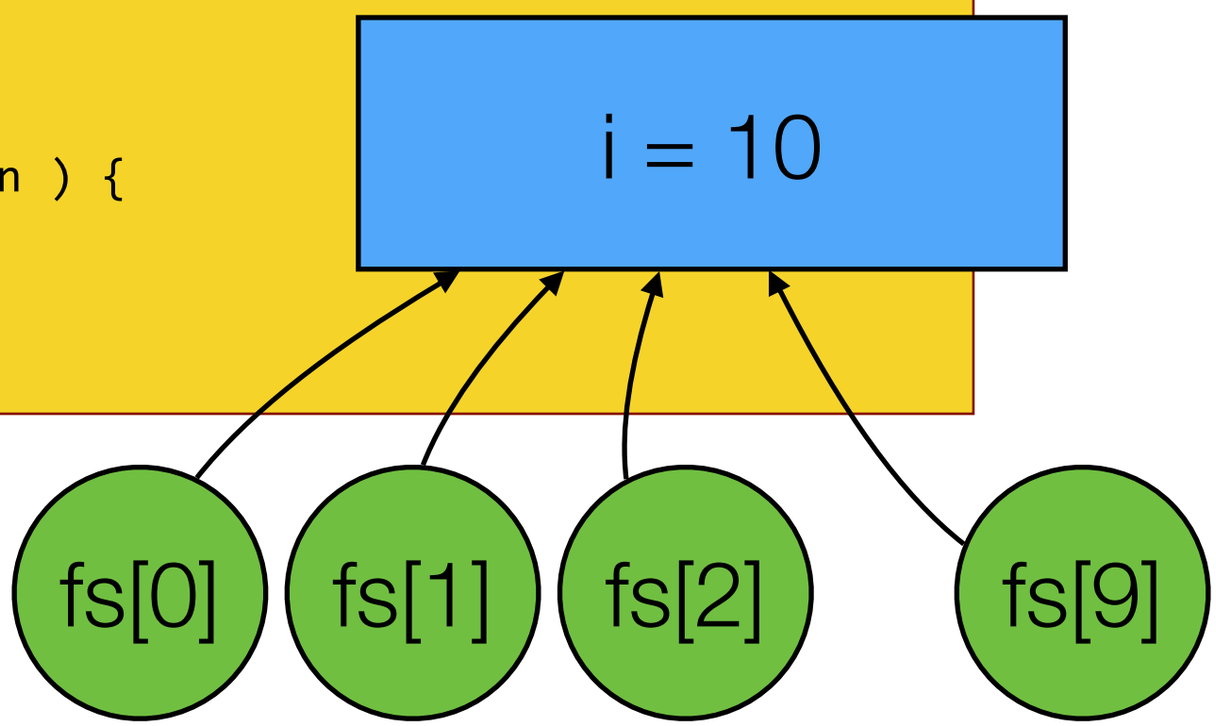
#4 Beware of functions created in loops!

```
var generateFunctions = function() {
    var generatedFunctions = [];
    for (var i=0; i<10; i++) {
        var g = function() {
            console.log(i);
        };
        generatedFunctions.push(g);
    }
    return generatedFunctions;

};

var fs = generateFunctions();

fs.forEach(function( currentFunction ) {
    currentFunction();
});
```



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

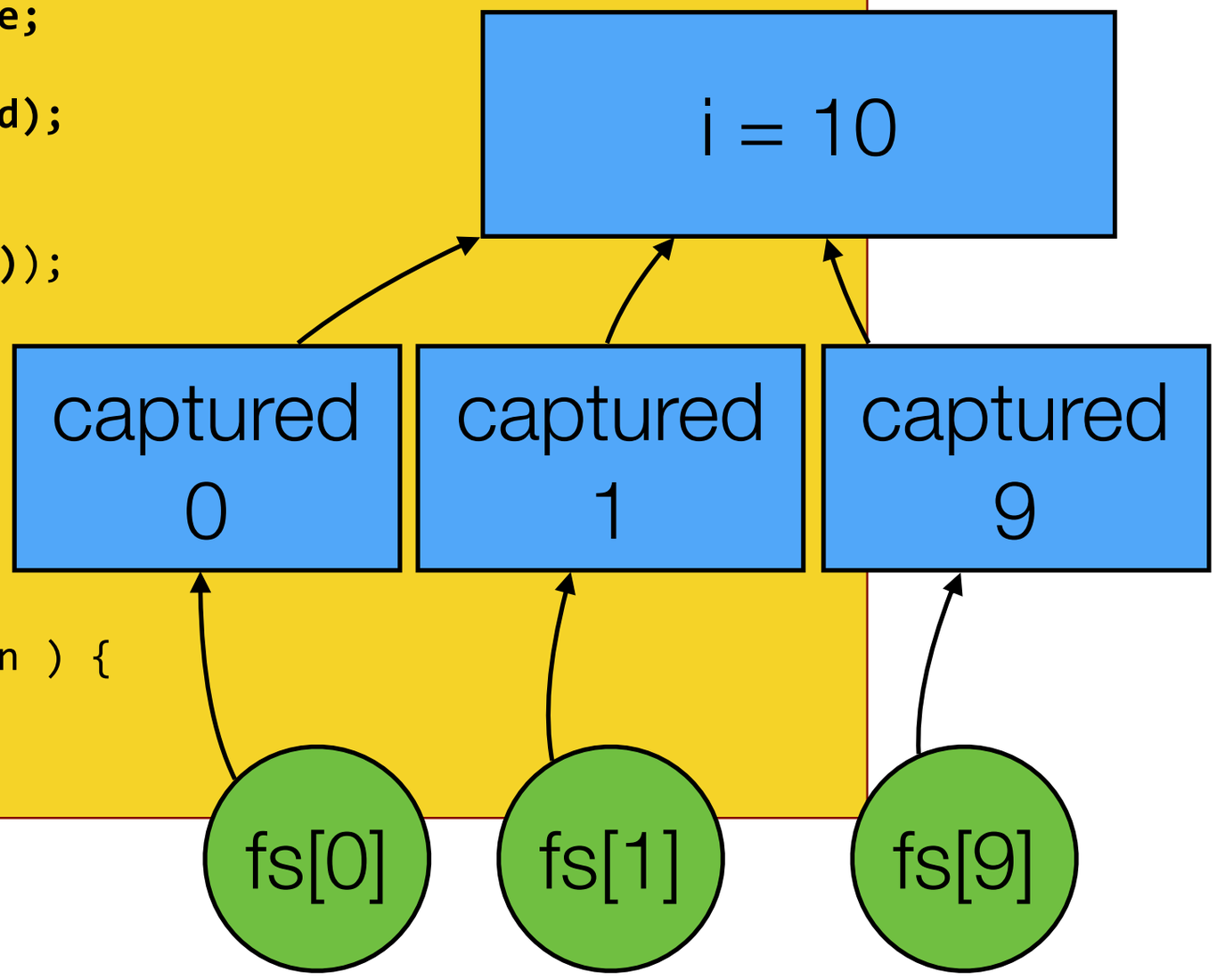
#5 Functions created in loops: capture variables

```
var generateFunctions = function() {
    var generatedFunctions = [];
    for (var i=0; i<10; i++) {
        var g = function( captureMe ) {
            var captured = captureMe;
            return function() {
                console.log(captured);
            };
        };
        generatedFunctions.push(g(i));
    }
    return generatedFunctions;

};

var fs = generateFunctions();

fs.forEach(function( currentFunction ) {
    currentFunction();
});
```



heig-vd

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

REST API design issues

URL Structure (1)

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 (2)

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

- How do you choose between these two approaches (flat vs deep structure)? There is no “right or wrong” answer and you find both styles in popular REST APIs.
- One rule that you can use to make your decision is whether you have an **aggregation** or **composition** relationship between resources. In other words, does the existence of one resource depend on the existence of another one. If that is the case, then it probably makes sense to use a hierarchical URL pattern.
- For instance, the existence of a **comment** depends on the existence of a **blog entry**. For this reason, I would probably go for:
  - `/blogs/amtBlog/posts/892/comments/`
- On the other hand, the existence of a **course** is not dependent on the existence of a **professor** (if one of you murders me, someone else will takeover the AMT course). Therefore, I would likely go for:
  - `/courses?professorName=liechti`

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)

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.).

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`

Asynchronous Programming

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

How can I execute multiple asynchronous operations in sequence?

Asynchronous Programming Techniques

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

```
setTimeout( function() {
   console.log(“the callback has been invoked”);
}, 2000);
```

An event will be added to the queue in 2000 ms. In other words, the function passed as the first argument will be invoked in 2 seconds or more (the thread might be busy when the event is posted...).

```
fs.readFile('/etc/passwd', function (err, data)
{
  if (err) throw err;
  console.log(data);
});
```

An event will be added when the file has been fully read (in a non-blocking way). When the event is taken out of the queue, the callback function has access to the file content (data).

Asynchronous Programming Techniques

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


```
$(document).mousemove(function(event){
  $("span").text(event.pageX + ", " +
event.pageY);
});
```

An event will be added to the queue whenever the mouse moves. In each case, the callback function has access to the event attributes (coordinates, key states, etc.).


```
$.get( "ajax/test.html", function( data ) {
$( ".result" ).html( data );
alert( "Load was performed." );
});
```

An event will be added when the AJAX request has been processed, i.e. when a response has been received. The callback function has access to the payload.

Beyond simple callbacks...

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

- The principle of passing a callback function when invoking an asynchronous operation is pretty straightforward.
- Things get more tricky as soon as you want to **coordinate multiple tasks**. Consider this simple example...

Do this first...

... when done, do this.

Beyond simple callbacks...

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

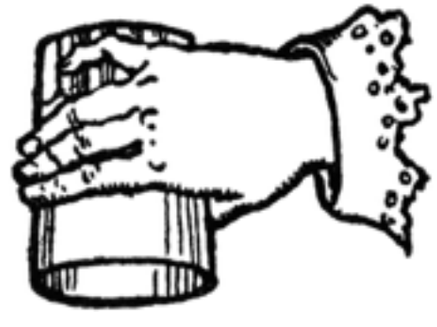
- A first attempt...

```
var milkAvailable = false;

function milkCow() {
  console.log("Starting to milk cow...");
  setTimeout(function() {
    console.log("Milk is available.");
    milkAvailable = true;
  }, 2000);
}

milkCow();
console.log("Can I drink my milk? (" + milkAvailable + ")");
```

FAIL



Beyond simple callbacks...

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

- Fixing the issue with a callback...

```
var milkAvailable = false;

function milkCow(done) {
  console.log("Starting to milk cow...");
  setTimeout(function() {
    console.log("Milk is available.");
    milkAvailable = true;
    done();
  }, 2000);
}

milkCow(function() {
  console.log("Can I drink my milk? (" + milkAvailable + ")");
});
```

SUCCESS

Beyond simple callbacks...

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

- Ok... but what happens when I have **more than 2 tasks** that I want to execute in sequence?
- Let's say we want to have the sequence B, C, D, X, Y, Z, E, F, where X, Y and Z are asynchronous tasks.

```
function f() {
   syncB();
   syncC();
   syncD();
   asyncX();
   asyncY();
   asyncZ();
   syncE();
   syncF();
}
```

```
B result available
C result available
D result available
E result available
Z result available
Y result available
F result available
X result available
```

Beyond simple callbacks...

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

- Ok... but what happens when I have **more than 2 tasks** that I want to execute in sequence?
- Let's say we want to have the sequence B, C, D, X, Y, Z, E, F, where X, Y and Z are asynchronous tasks.

```
function f() {
  syncB();
  syncC();
  syncD();
  asyncX(function() {
    asyncY(function() {
      asyncZ(function() {
        syncE();
        syncF();
      });
    });
  });
}
```

```
B result available
C result available
D result available
X result available
Y result available
Z result available
E result available
F result available
```

But welcome to the "callback hell" aka "callback pyramid"

Beyond simple callbacks...

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

- Now, let's imagine that we have 3 asynchronous tasks. We want to invoke them in parallel and **wait until all of them complete**.
- Typical use case: you want to send several AJAX requests (to get different data models) and update your DOM once you have received all responses.

```
function f(done) {

 async1(function(r1) {
    reportResult(r1);
  });
  async2(function(r2) {
    reportResult(r2);
  });
  async3(function(r3) {
    reportResult(r3);
  });

  done();

}
```

Double fail: not only do I invoke done() to early, but also I don't have any result to send back...

Beyond simple callbacks...

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

- Now, let's imagine that we have 3 asynchronous tasks. We want to invoke them in parallel and **wait until all of them complete**.
- Typical use case: you want to send several AJAX requests (to get different data models) and update your DOM once you have received all responses.

```
function f(done) {
  var numberOfPendingTasks = 3;
  var results = [];

  function reportResult(result) {
    result.push(result);
    numberOfPendingTasks -= 1;
    if (numberOfPendingTasks === 0) {
      done(null, results);
    }
  }
  async1(function(r1) {
    reportResult(r1);
  });
  async2(function(r2) {
    reportResult(r2);
  });
  async3(function(r3) {
    reportResult(r3);
  });
}
```

When this reaches 0, I know that all the tasks have completed. I can invoke the "done" callback function that I received from the client. I can pass the array of results to the function.

When a task completes, it invokes this function and passes its result. The result is added to the array and the number of pending tasks is decremented.

The three tasks are asynchronous, so they pass their own callback functions and receive a result when the operation completes.

Beyond simple callbacks...

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

To be continued...

heig-vd

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

REST API Documentation

API Documentation

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

- When you are designing and implementing a REST API, you are most often doing it for **third-party developers**:
  - Think about Twitter, Instagram or Amazon exposing services to external developers.
  - Think about an enterprise (e.g. car manufacturer) exposing services to business partners (e.g. suppliers, subcontractors, distributors).
- The documentation of your API is the first thing that third-party developers (your **customers**) will see. You want to **seduce** them.
- The documentation of your API will have a big impact on its **learnability** and **ease of use**.
- **Best practices** and **tools** have emerged. **Evaluate and apply them!**

RAML

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

- **R**ESTful **A**PI **M**odeling **L**anguage
- RAML is a language that has been developed to facilitate the **design** and **documentation** of REST APIs.
- It allows you to describe **resources**, **methods**, **parameters**, **headers** and **payloads** in a succinct manner (support for abstraction and reuse).
- From a RAML file, it is possible to **generate a user-friendly documentation** (e.g. in HTML) with various **tools**.
- Other tools support import/export exchange with REST frameworks (e.g. JAX-RS).

```
#%RAML 0.8

title: World Music API
baseUri: http://example.api.com/{version}
version: v1
traits:
  - paged:
      queryParameters:
        pages:
          description: The number of pages to return
          type: number
  - secured: !include http://raml-example.com/secured.yml
/songs:
  is: [ paged, secured ]
  get:
    queryParameters:
      genre:
        description: filter the songs by genre
  post:
  /{songId}:
    get:
      responses:
        200:
          body:
            application/json:
              schema: |
                { "$schema": "http://json-schema.org/schema",
                  "type": "object",
                  "description": "A canonical song",
                  "properties": {
                    "title":  { "type": "string" },
                    "artist": { "type": "string" }
                  },
                  "required": [ "title", "artist" ]
                }
            application/xml:
    delete:
      description: |
        This method will *delete* an **individual song**
```

RAML

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

- RAML is pretty straightforward to use:
  - You describe the list of resources managed by your application, document the support HTTP verbs, enlist the query parameters, etc.
  - If you use **Sublime Text**, you can take advantage of an extension that provides **syntax highlighting**.
    - See **RAML 100 tutorial** (http://raml.org/docs.html)
- RAML has advanced features that can make your specifications less verbose (by abstracting and reusing common elements):
  - includes
  - resource types and schemas
  - traits
  - See **RAML 200 tutorial** (http://raml.org/docs-200.html)

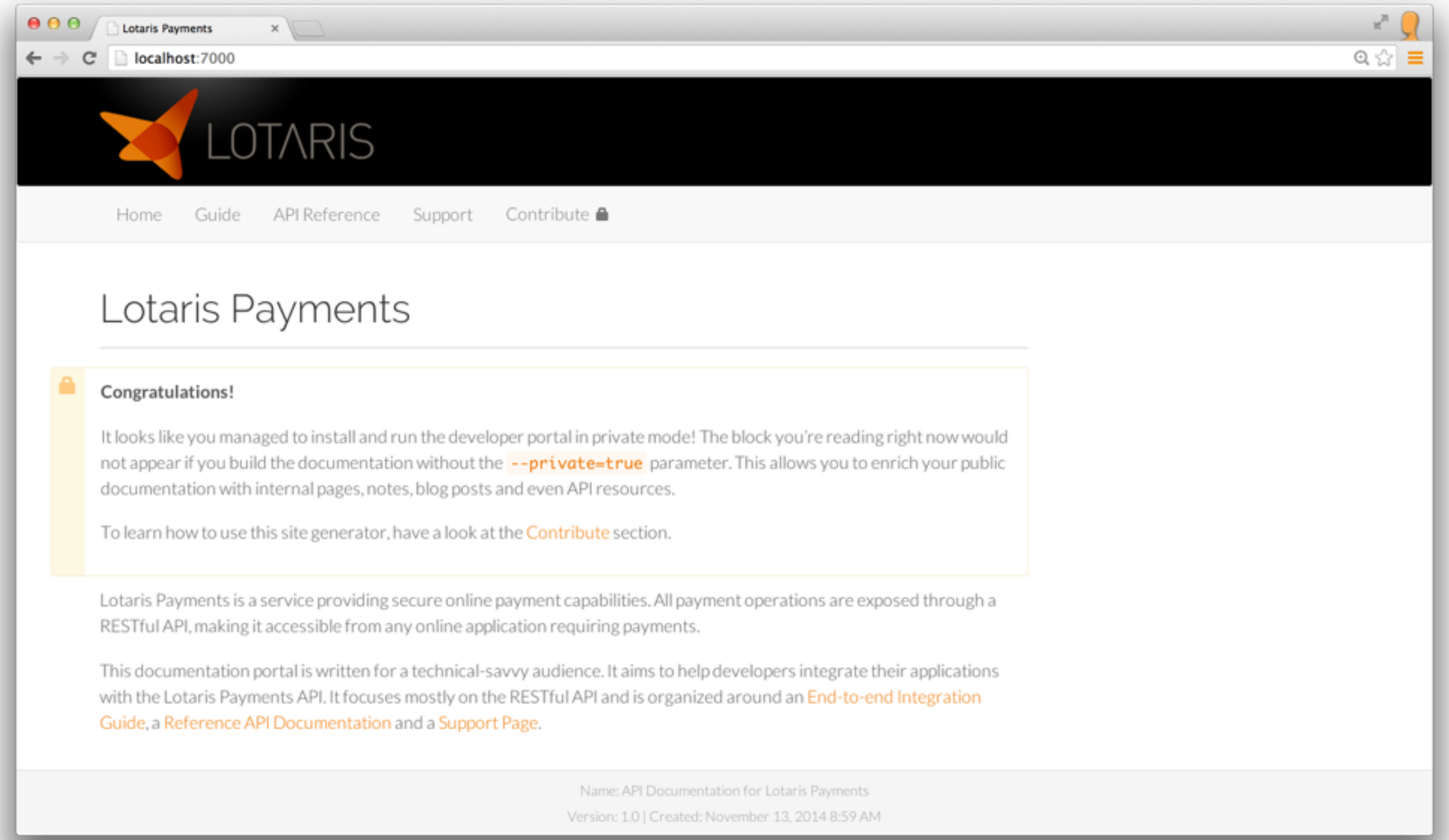
apidoc-seed

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

- **apidoc-seed** is an open source tool that is provided by Lotaris, which makes it easy to generate a complete HTML site for documenting your REST API.
- To use the tool, **clone the GitHub repo** (https://github.com/lotaris/apidoc-seed):
  - install **node.js** and **grunt** (`npm install -g grunt-cli`)
  - modify the **directory structure** to add/remove items in the main menu
  - **edit/create jade templates and markdown documents** to provide documentation for your service (general service information, usage guides, support information, etc.).
  - **edit/create RAML files to document your REST APIs**. Depending on the complexity of your APIs, you can **split** the documentation into several files.
  - follow instructions in the README.md file and generate the documentation site.
- The tool supports a notion of "**private**" API elements. This is used if your API has resources, methods or parameters that you don't want to publicly expose yet (note that this is documentation level only, nothing will prevent a user to send a request!).

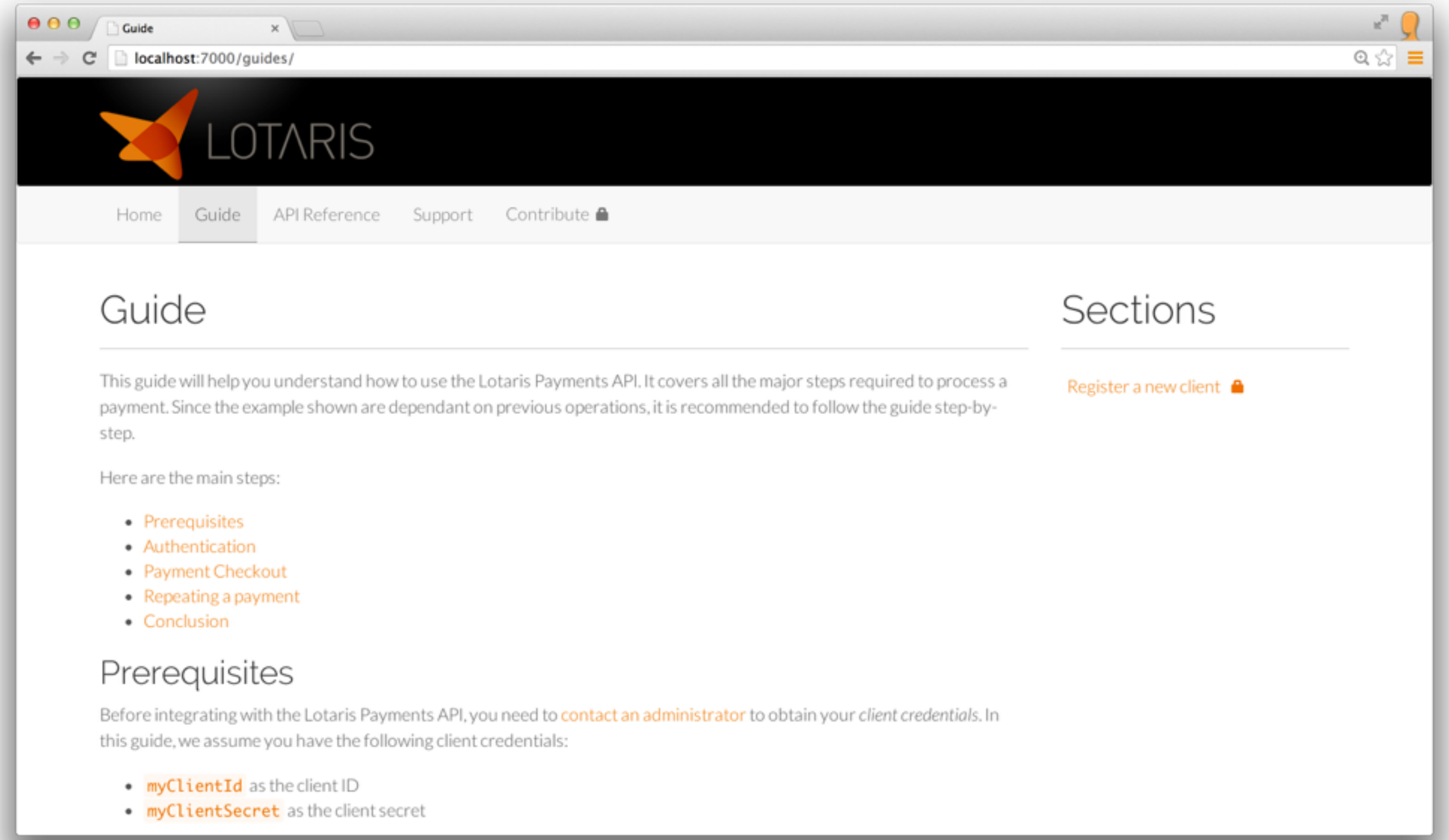
apidoc-seed

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



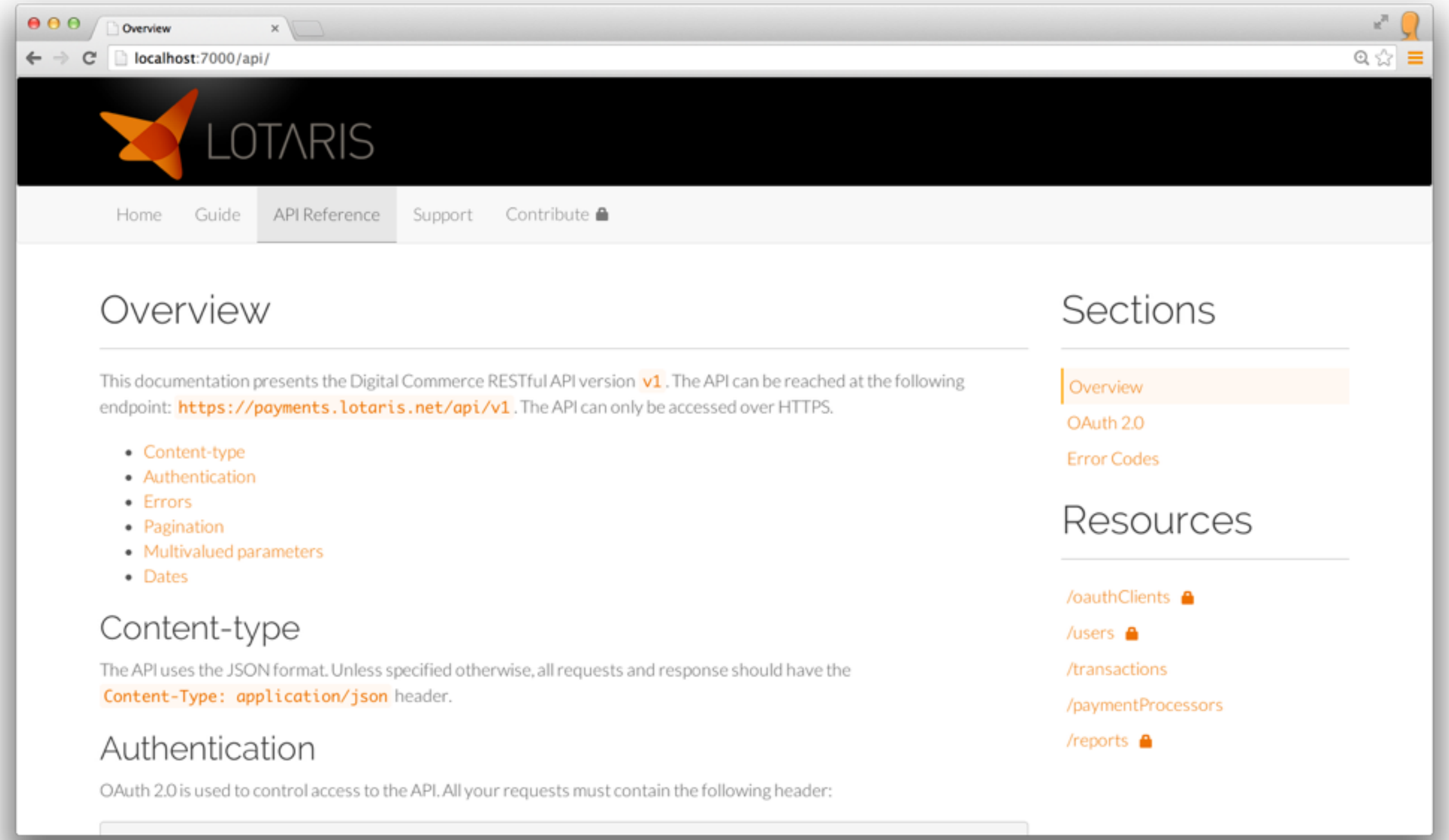
apidoc-seed

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



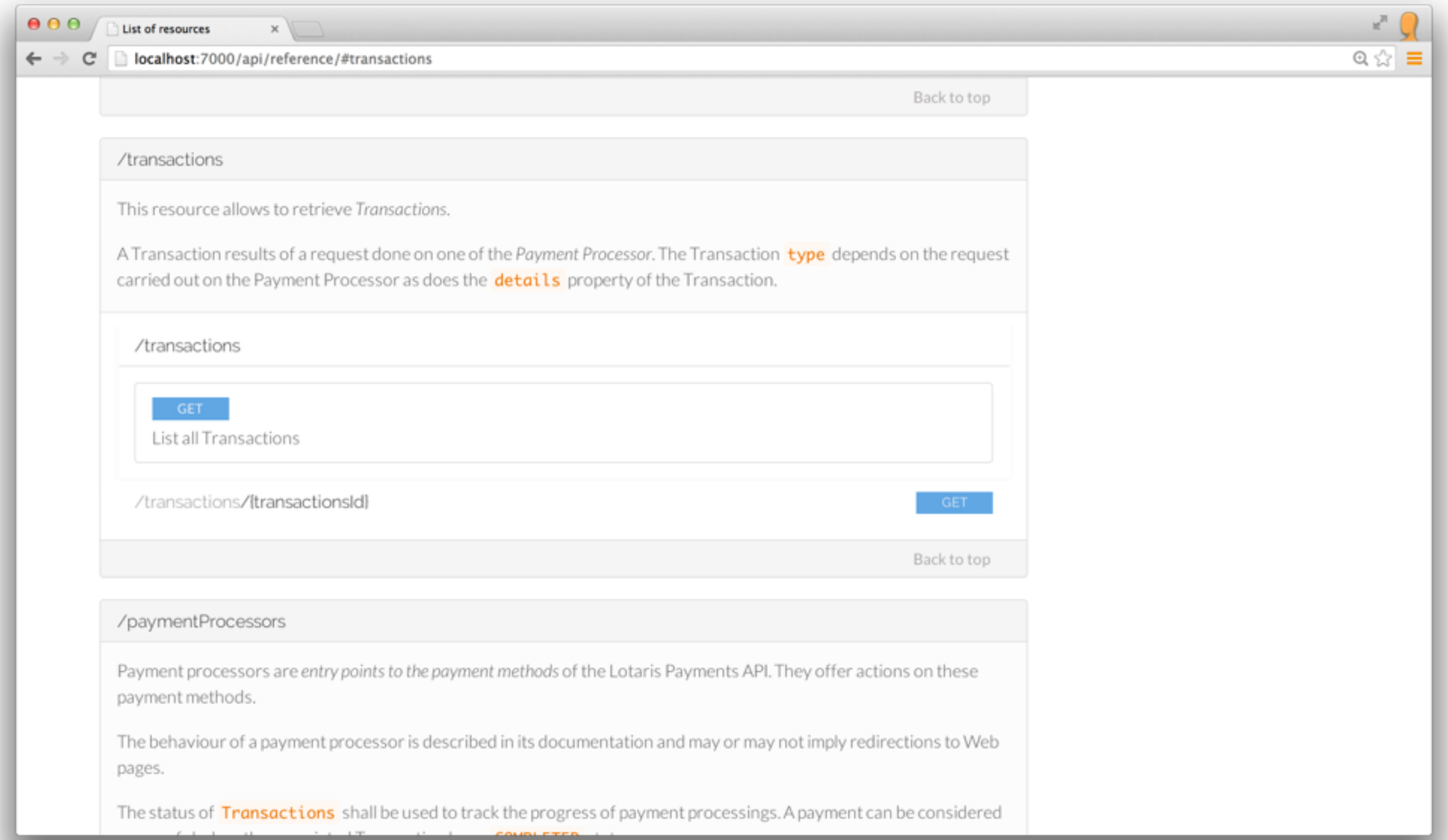
apidoc-seed

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



apidoc-seed

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



apidoc-seed

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