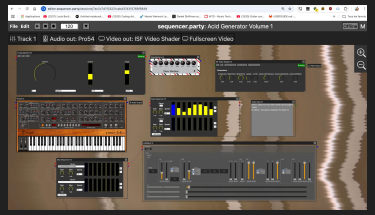
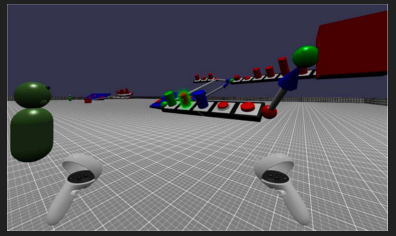




Technologies Web

Master Informatique

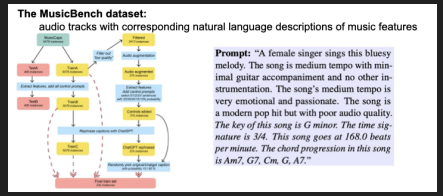
2024/2025

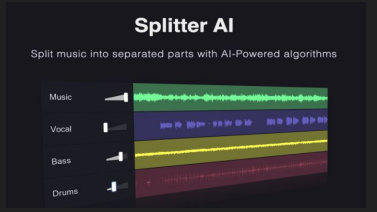
Michel Buffa

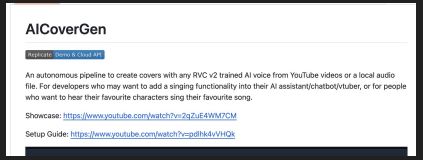
michel.buffa@univ-cotedazur.fr

LINK OF THIS COURSE : https://tinyurl.com/939xcxet

Université Côte d'Azur / Laboratoire I3S / INRIA, WIMMICS group







UNIVERSITÉ CÔTE D'AZUR | ÉCOLE UNIVERSITAIRE DE RECHERCHE SYSTÈMES NUMÉRIQUES POUR L'HUMAIN | FRANCE 2030 Initiative d'Excellence | i3S sophia antipolis | Inria | cnrs | wimmics

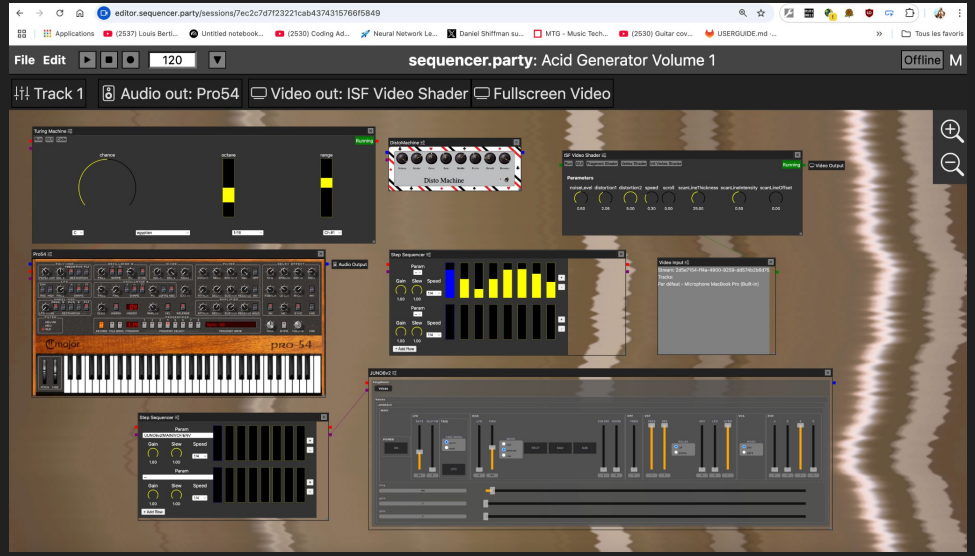
Who am I?

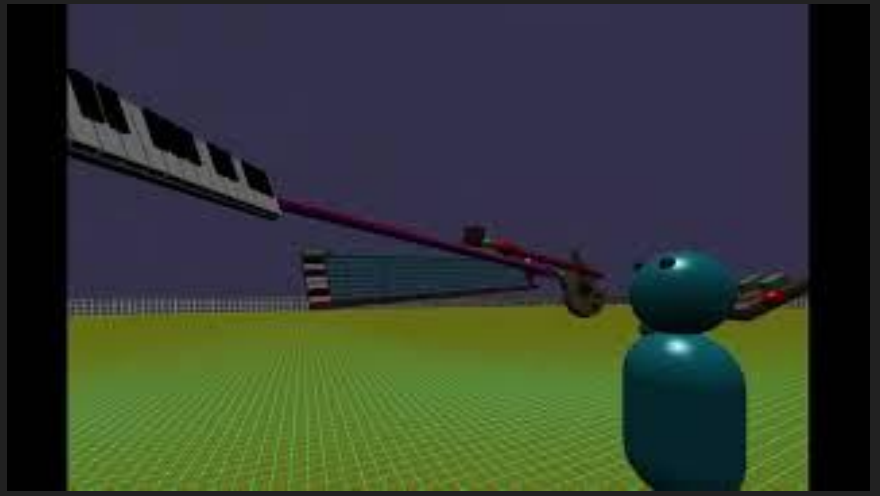
- Professor / researcher at Université Côte d'Azur (UCA), France
  - Member of the WIMMICS research group common to INRIA and I3S lab from CNRS
  - W3C Advisory Committee Representative for my university
  - Member of the W3C WebAudio Working Group since 2014
  - Main Research topic: Web Audio
  - michel.buffa@univ-cotedazur.fr

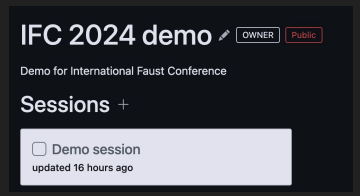
- Involved in the Web Audio Modules (WAM) plug-in standard, aka "VSTs for the Web"
- Teach "Using AI for Music Creation" at MSc Music Scoring for Visual Media and Sound Design (Cannes) and at Master Création contemporaine et nouvelles technologies option Informatique Musicale (Université Jean Monnet, Saint-Etienne)

Buffa, M., Ren, S., Campbell, O., Burns, T., Yi, S., Kleimola, J. and Larkin, O., 2022, April. Web Audio Modules 2.0: An Open Web Audio Plugin Standard. In *Companion Proceedings of the Web Conference 2022* (pp. 364-369).

Real-time collaborative improvisation, 2D and 3D/Immersive !







Buffa, M., Girard, D., Demont, S., Escobar, Q. and Hofr, A., 2024, November. Faust Plugins in (Sometimes Unexpected) Web-Based Hosts. In *International Faust Conference 2024*.

Buffa, M., Girard, D. and Hofr, A., 2024, September. Using Web Audio Modules for Immersive Audio Collaboration in the Musical Metaverse. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)* (pp. 1-10). IEEE.

Tools we’re going to use

Some don’t need installation:

- Online IDEs : jsbin.com, codepen.io, codesandbox.io etc.

Some need to be installed

- Visual Studio code, NodeJS, Git and eventuallyWebStorm
- Github copilot (free with GitHub Education)
- Gemini CLI

Also: not so up to date list of tools for web development and some recommendations about how to configure your Mac for web dev.

Web Technologies? What is it exactly?

Front-End

- **Real web standards**:
  - Langages: HTML, CSS, JavaScript, WebAssembly
    - Cross compilation…
  - Protocols : http/s, websockets, webRTC…
  - W3C web browsers' APIs: more than 50 APIs!
    - QuerySelector, DOM, Pointer Events, Drag and Drop, Forms, Tables, Multimedia (audio/video/webcam/micro streamés), Canvas, Geolocation, Motion, IndexedDB, Fetch, WebNN, Web Codecs, MediaDevices, WebGL, WebGPU, etc.
  - Web Components: Templates, Shadow Dom, Custom Elements
- Libraries and Frameworks (open source, industry…) :
  - Stencil, Lit, Angular, React, VueJS, Tailwind CSS, etc.
- Build tools:
  - Vite, WebPack, Parcel, Rollup, etc.

Web Technologies? What is it exactly?

Back-End

- **HTTP Servers**
  - Can serve HTML pages (with CSS/JavaScript and assets:images, icons, sounds, files etc.)
  - Often provide **Web Services / API / REST endpoints**
  - Asynchronous communication
- **WebRTC / WebSockets servers**
  - **For real-time, synchronous communication**
- These server use web standards, but are NOT standardized: made by the open source Community or private companies
- Large variety of servers
  - Java EE (Tomcat, Spark Java, Spring Boot, etc.)
  - Python (Flask, Django, etc.)
  - JavaScript/TypeScript: **Node JS / Express** et frameworks associés (très nombreux)

Framework	Points forts	Cas d'utilisation
Express.js	Minimaliste, large communauté	API REST simples.
Sails.js	MVC, WebSockets intégrés	Applications temps réel ou API complexes.
Koa.js	Middleware léger, modernité	Serveurs flexibles et personnalisables.
NestJS	Architecture modulaire	Applications d'entreprise robustes.
Hapi.js	Validation intégrée, riche en fonctionnalités	API structurées pour applications complexes.
AdonisJS	Inspiré de Laravel, prêt à l'emploi	Applications CRUD et backend performants.
Fastify	Ultra performant, support JSON Schema	Serveurs à haute performance.
Meteor.js	Framework full-stack	Applications en temps réel avec gestion client/serveur.
LoopBack	Génération automatique d'API	Backend basé sur des bases de données existantes.

Web Technologies? What is it exactly?

Databases: traditional SQL and NoSQL ones:

- PostgresSQL, MySQL, Oracle, SQL Server, etc.
- MongoDB, Cassandra, CouchDB, Neo4J, etc.

Real-time multi-user collaborative databases:

- FireBase/FireCloud (real-time, websocket-based)
- Redis (clusters)

Related tools (indexer, cache, etc.):

- Elastic Search, etc.

données	Type	Points forts	Cas d'utilisation
MongoDB	Document	JSON flexible, évolutif	Applications dynamiques.
Cassandra	Colonnes	Haute disponibilité, haute tolérance	Big Data, temps réel.
Redis	Clé-valeur	Ultra rapide, stockage en mémoire	Caching, messages en temps réel.
Elasticsearch	Recherche & analyse	Recherche rapide, analytique puissante	Moteurs de recherche.
Couchbase	Document/Clé-valeur	Requêtes SQL-like	Applications nécessitant des performances élevées.
DynamoDB	Clé-valeur	Gestion automatique (AWS)	Cloud, microservices.
Firebase	Clé-valeur	Synchronisation temps réel	Applications mobiles, jeux multi-utilisateurs.
Neo4j	Graphes	Analyse des relations complexes	Réseaux sociaux, chaînes logistiques.
HBase	Colonnes	Big Data sur Hadoop	Journaux massifs.

Web Technologies? What is it exactly?

Hosting / Cloud / Virtualization

- Many possibilities
  - Simple project, without a back-end : GitHub pages, CloudFlare, etc.
  - Project with back-end:
    - **What to deploy?**
      - Files, Docker image, etc.?
      - How to organize yourself for local development testing and easy deployment (CI/CD: Continuous Integration / Continuous Deployment, part of DEVOPS)
    - **Hosting**
      - render.com, Google Cloud Platform, Microsoft Azure, Amazon, Heroku, OVH
      - Dedicated Server (proxy, security)

What are we going to see in this course?

This is a brand new course

Front-End

- Basics
  - MOOCs at w3cx.org : HTML Coding Essentials and best practices, HTML5 Apps and Games, JavaScript Introduction,

- Asynchronous programming with JavaScript/TypeScript and use related Web browser APIs : Fetch API, JavaScript promises, Async / Await, Table API, Forms API, etc.
- Other APIs: WebAudio, WebMidi, MediaDevices, DOM Api, persistence APIs etc.
- Web Components: principles, reusing WC, creating pure WC

- WebComponents through **the Angular Framework**

This is a brand new course

Back-End

- We will use Node JS / Express / MongoDB (with Mongoose, sort of ORM for MongoDB)
  - Web Services, several npm modules for NodeJS
  - Authentication through JSON Web Tokens (JWT)
  - Maybe Single Sign On techniques (authenticate with Google/GitHub for example)

- Hosting on GitHub Pages and Cloud (render.com, mongodb.com)

This is a brand new course

Use of AI assistants

- ChatGpt, Gemini CLI, Copilot, GitHub services
- I plan to teach you how to integrate a LLM into a web application…

First exercices: install nodeJS, generate a back end server, test it

We will create a small web-service based back-end

Use of AI assistants

- ChatGpt, Gemini CLI, Copilot, GitHub services
- I plan to teach you how to integrate a LLM into a web application…

What are Web Services?
How can we consume them?
How can we create them?

Web Service: definition

“A Web Service is a software application designed to enable communication between different applications via the web. These services enable systems, often written in different languages or platforms, to exchange data in a standardized way.”

- *Reusability, clients and server can use different technologies, can be developed by different people!*
- *Scalability: REST Web Services do not keep session states in server memory!*
- *Durability: bases on Web Standards (HTTP/S), technology will last decades.*

Examples:

- Most available APIs! Google, OpenStreetMaps, Weather, and more generally all APIs from the Open Data initiative, such as https://www.data.gouv.fr/fr/ or more locally, https://opendata.nicecotedazur.org/site
- You can make your own REST endpoints/APIs!

Web Services?

- Use the HTTP protocol for manipulating remote data GET/POST/PUT/DELETE
- Also called API or “endpoint” or “REST endpoint”
- First example with https://jsonplaceholder.typicode.com/ (a web site for learning how to work with REST Web Services).
    - Let’s code the example from scratch in Visual Studio Code, and let’s add it to GitHub

- Another example with the OpenWeatherMap API
- It is super important to know how to use the “network” tab of your browser devtools (ctrl-shift-i and command-option-i with Macs)

Seance 2: lab exercises

REPOSITORY FOR THE COURSE

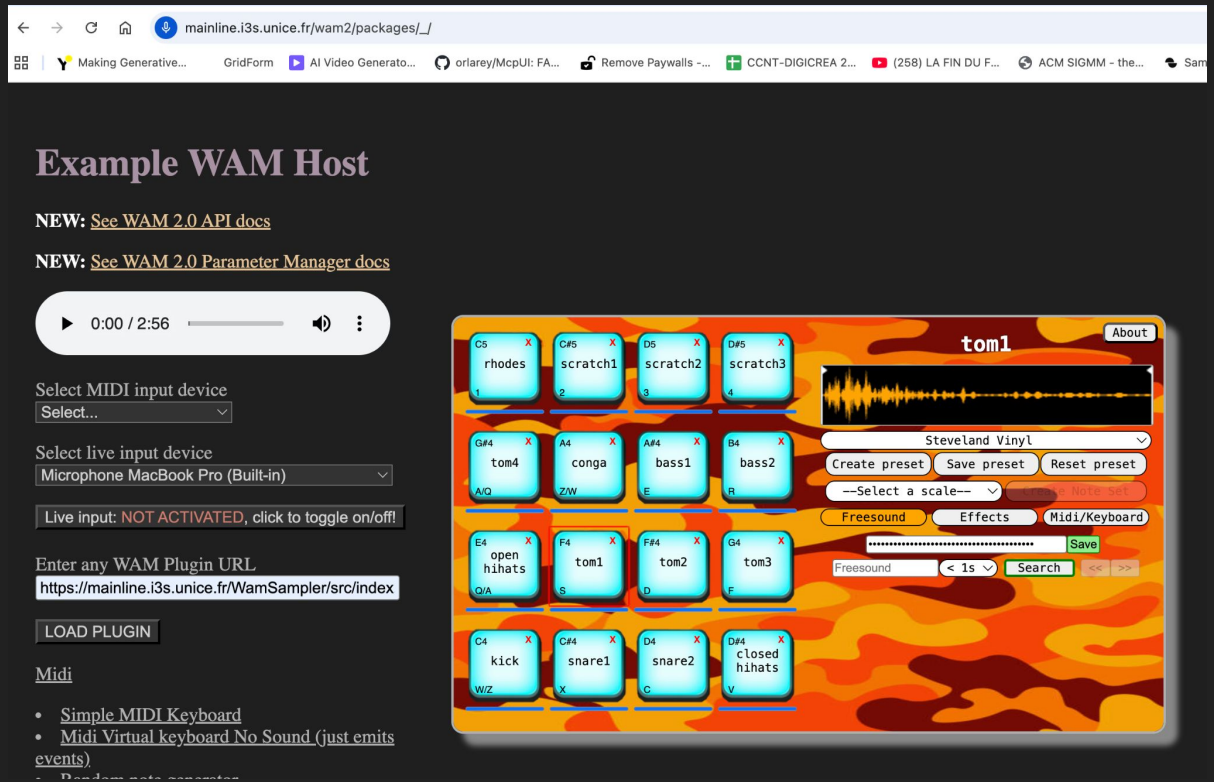
- https://github.com/micbuffa/M1InfoWebTechnos2025_2026.git

- You need to install NodeJS
  - NodeJS from nodeJS.org, get the last stable version
  - Check with node –version and npm –version that you have a recent one

- You need to install the IA assistant copilot (through GitHub Education)
- Also recommended : Gemini CLI

- You need to have a GIT client and know how to use Git

This is an audio sampler!



- https://mainline.i3s.unice.fr/wam2/packages/_/
- Enter this URL: https://mainline.i3s.unice.fr/WamSampler/src/index.js
- Click the LOAD PLUGIN button.

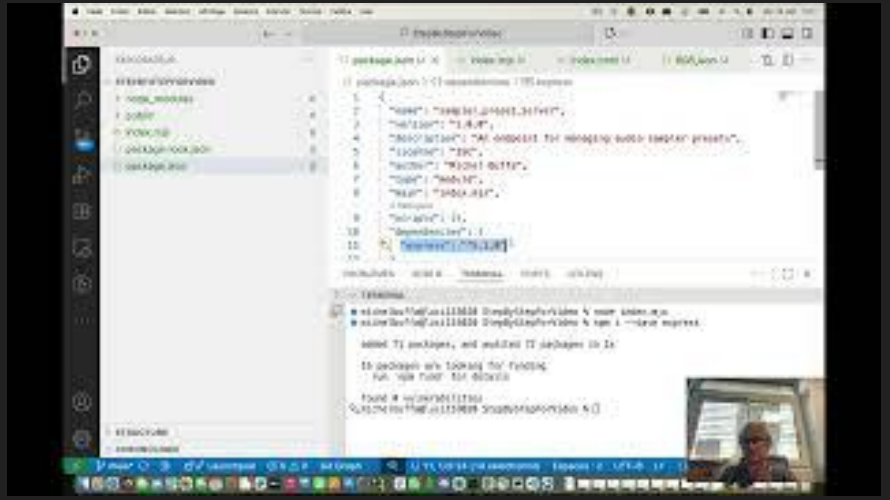
We will try to write a REST endpoint for managing the presets of such an audio sampler, later we will write an audio sampler :-)

- **Seance2 LAB EXERCICE !**
- Clone or pull https://github.com/micbuffa/M1InfoWebTechnos2025_2026.git It contains a folder named “Seance2” with two subfolders :
  - One is the skeleton of the endpoint, the other is more advanced and contains also the solution
- Run the skeleton app:
  - `cd` in the folder
  - `npm i` : installs the necessary packages
  - `npm run dev` to run the app, open http://localhost:3000/
    It will show the content of the public/index.html file and will describe YOUR WORK for today ! You will have to complete the back-end code + the front-end code
  - `npm run test` to run the unit tests
- You will “learn by doing” and use AI assistants. But here are two courses I wrote about NodeJS and about the essential npm module Express (a bit outdated), you can find many tutorials on the web.

Seance 3: more about JavaScript / async programming…

Here is a step by step video explaining how to code a simple solution to the previous exercise

- **Seance2: step by step video (you can have english subtitles with youtube options). Checj Seance2/Step by Step Video folder on the GitHub repo**



For the ones who need to improve JavaScript
And would like to get a W3C/MIT/Harvard Certification for free…

MOOC JavaScript Introduction at w3cx.org

- A MOOC I wrote for a JavaScript Introduction: https://www.edx.org/learn/javascript/the-world-wide-web-consortium-w3c-javascript-introduction

- Register to the MOOC, check “audit this course” (free)
- If you want to get access to the quizzed + get the certification, ask your teacher.

- DEMO : rapid visit of the MOOC!

- Some reminders about JavaScript follow…

Variables

Variables, constants, let, const, var, (nothing)...

Good practices

```
const  x = 4;

const  MAXIMULM_VALUE  = 100000000;

let  firstName = “michel”;

let  lastName = ‘buffa’;

let  fullName = firstName + “ “ + lastName; // string concatenation

let  fullName2 = `Your teacher is ${firstName} ${lastName}`
```

Bad practices (obsolete but still works because of backward compatibility)

```
var  x = 4;    // not the same as let, as when used as a local variable the scope = the function, not the { } block !

y = 4;         // similar to window.y = 4, in other word GLOBAL VARIABLE scope = all the application!
```

fonctions and callbacks

Standard use…

Typical examples:

```
function sum(a, b) {
  // this function returns a result
  return (a + b);
}

function afficheDansPage(message, value) {
  // this function returns nothing
  document.body.innerHTML += message + value + "<br>";
}

let result = sum(3, 4);
afficheDansPage("Resultat: ", result);


// we could have written this
afficheDansPage("Resultat: ", sum(10, 15));
```

Fonctions in JavaScript : fonction in an expression

FUnction declared as a variable:

```
let sum = function(a, b) { // <- ICI,  the function(...) block is an anonymous function
  return (a + b);
};

let afficheDansPage = function(message, val) {
  document.body.innerHTML += message + val + "<br>";
};

let resultat = sum(3, 4);
afficheDansPage("Resultat: ", result);
```

Fonctions in JavaScript : callbacks

```
// Add a click listener to the page
// the function processClick is a callback:
// a function called by the browser when
// the click event is raised
window.addEventListener('click', processClick);


window.onclick = processClick; // nearly equivalent to the previous line


function processClick(event) {
  document.body.innerHTML += "Button clicked<br>";
}

// Other possibility
window.addEventListener('click',function (evt) {
  document.body.innerHTML += "Button clicked version 2<br>";
});

// Or better : use “arrow functions” (see next slides)
```

Arrow functions

Recommended :

```
window.addEventListener('click', (evt) => {
  document.body.innerHTML += "Button clicked version 2<br>";
});
```

Rule : where we would have used the keyword “function” we omit it, and we add => after the parameter definition.

```
img.onclick = (evt) => {
    console.log(“image clicked”);
}
```

The blue part is an anonymous function

Arrow functions)

The big advantage of arrow functions is that they preserve this!

This is very useful when defining event listeners inside classes, since we often need access to `this`.

```
class Game {
    x = 3;

    method() {
        ...
        // This works : this = current instance of the class
        window.addEventListener('click', (evt) => {
          document.body.innerHTML += "X = " + this.x; // OK !
        });

        // does not work : this = the window object
        window.addEventListener('click', function(evt) {
          document.body.innerHTML += "X = " + this.x;     // this is not correct, its window, not the
                                                          // current instance of the class
        });


    }

}
```

Arrays

Arrays

```
> let myarr = ['red', 'blue', 'yellow', 'purple'];
undefined

> myarr;
["red", "blue", "yellow", "purple"]

> myarr[0];
"red"

> myarr[3];
"purple"
```

Tableaux : they are objects with methods and properties

```
> let a = [];
> typeof a;
"object"

> let a = [1, 3, 2, 5, 7];
undefined

> a.length; // number of elements
5

> a.sort(); // sort elements of a
[1, 2, 3, 5, 7]

> a.splice(2, 1); // remove 1 element starting at index 2 (3rd element)
[3]

> a; // The '3' has been removed
[1, 2, 5, 7]
```

Arrays : custom sort

```
let persons = [
    {givenName: 'Michel', familyName: 'Buffa',    age:51},
    {givenName: 'Pig',    familyName: 'Bodine',   age:20},
    {givenName: 'Pirate', familyName: 'Prentice', age:32}
];

function compareByAge(a,b) { // comparison function, a and b are persons
  if (a.age < b.age)         // compare by age
    return -1;

  if (a.age > b.age)
    return 1;

  return 0;
}

persons.sort(compareByAge); // this will call automatically compareByAge
                            // passing all persons from the array, compare
                            // them by age and sort the array.
```

Arrays : elements of different types supported

```
> let a = [1,2,3];

> a[2] = 'three';
"three"

> a
[1, 2, "three"]
```

Arrays : adding elements

```
> let a = [1,2,"three"];
undefined

> a[3] = 'four';
"four"

> a;
[1, 2, "three", "four"]

> a[a.length] = "five"; // adding at the end
[1, 2, "three", "four", "five"]

> a.push("six"); // but usually we prefer using the push method for adding
[1, 2, 3, "four", "five", "six"]  // a new element at the end
a.unshift("zero"); // insert at beginning
["zero", 1, 2, 3, "four", "five", "six"]
```

Arrays : beware to “holes” if you use indexes

```
> a[7] = 'height';
"height"

> a;
[1, 2, 3, "four", "five", "six", undefined × 1, "height"]
```

Arrays : removing one or more elements

```
> a;
[1, 2, 3, "four", "five", "six", undefined × 1, "height"]

> a.splice(6, 1); // remove element at the sixth index, the undefined one!
[undefined × 1]

> a;
[1, 2, 3, "four", "five", "six", "height"] // it's no more here :-)

> a.splice(0, 3); // remove the three first elements
[1, 2, 3]

> a;
["four", "five", "six", "height"]

> a.splice(a.length-1); // remove the last element
"height"

> a;
["four", "five", "six"]
```

Tableaux : remove last element

```
> a
["four", "five", "six"]

> a.pop(); // remember push/pop = add / remove element at last position!
"six"

> a
["four", "five"]
```

Arrays of arrays

```
> let a = [[1,2,3], [4,5,6]]; // a is a matrix: 2 rows, 3 columns.
undefined

> a[0]; // first row
[1, 2, 3]

> a[1]; // second row
[4, 5, 6]

> a[0][0]; // top left element
1

> a[0][1]; // second element, first line
2

> a[0][2]; // third element, first line
3

> a[1][0]; // first element, second line
4

> a[1][1]; // second element, second line
5

> a[1][2]; // third element, second line
6
```

Arrays : other methods

Arrays have lots of other methods, like :

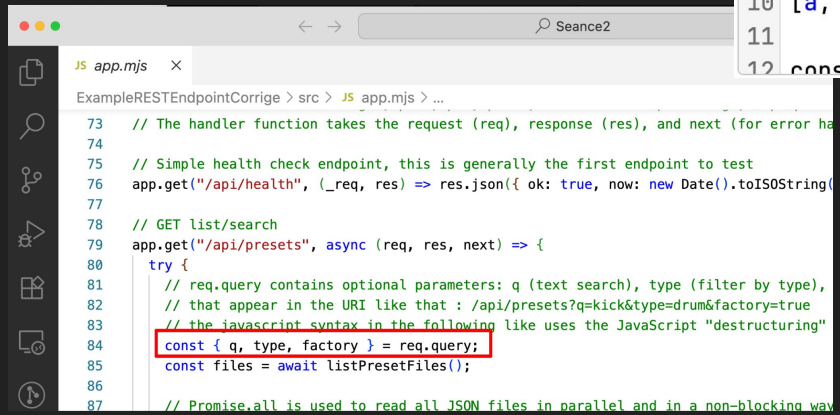
- `filter(...)` To create an array from another one, taking only the elements that meet a certain condition.
- `find(...)` for finding elements
- `join(...)` for turning an array of strings into a single array
- `map(), reduce(), flat() etc.`
- etc.

Destructuring assignment

[See this tutorial on MDN](#) : MUST SEE !

JavaScript Demo: Expressions - Destructuring assignment

```
1  let a, b, rest;
2  [a, b] = [10, 20];
3
4  console.log(a);
5  // Expected output: 10
6
7  console.log(b);
8  // Expected output: 20
9
10 [a, b, ...rest] = [10, 20, 30, 40, 50];
11
12 console.log(rest);
```



JavaScript Classes

Classic case

See JavaScript introduction MOOC or this tutorial on MDN MUST SEE!

LIVE coding DEMO!

Asynchronous JavaScript

Classic case

- See my course on JavaScript Promises
- See this complementary short set of slides about async/await

Ok, let’s go back to our web services
1 - Use Promise.all in GET endpoint
2 - Write a client that display the Sampler presets
3 - Try to build a drop down menu

Work for seance 4:

1. Register to my Mooc : HTML5 Apps and Games
2. **Follow the Module 1.5 about the Web Audio API** (estimated time 1.5 hour)

Seance 4: Lab exercises with Web Audio / Web MIDI
First Steps towards an Audio Sampler

Ressources

1. MOOC HTML5 Apps and Games, module 1.5 about Web Audio (the work you should have done before this course)
2. Web Audio tutorial I gave for the W3C
3. You can search “MDN web audio AnalyzerNode” for example… MDN is a good resource for Web Audio / Web Midi / JavaScript in general…

First exercise: look at Example1 in the GitHub repository

1. Git pull the repository of the course, go to Seance4 folder

2. The README contains the exercises, the folder contains also some examples related to the exercises…

3. There are also some assignments for the next course (Exercise 4 to finish before October 20th), I will do some oral examinations (graded) with some student teams.

Seance 5
Just commenting assignments
Live coding etc.
No slide content
Check GitHub Repo updates
(Seance5 folder)

Seance 6
Media Recorder,
Intro to Web Components

MediaStreams API and MediaRecorder

Record nearly anything :-)

- MediaRecorder examples
- Record almost everything in the browser with MediaRecorder
- Documentation on google.developers.com

MediaRecording in a few steps 1/3

1. Instantiate a MediaRecorder with a MediaStream

```
var options = {mimeType: 'video/webm; codecs=vp9'};
mediaRecorder = new MediaRecorder(stream, options);
```

2. Next, add a data handler and call the start() method to begin recording:

```
var recordedChunks = []; // will hold the recorded stream
mediaRecorder.ondataavailable = handleDataAvailable;
mediaRecorder.start();

function handleDataAvailable(event) {
 if (event.data.size > 0) {
   recordedChunks.push(event.data);
 } else {
   // ...
 }
}
```

MediaRecording in a few steps 2/3

The last piece of code adds a Blob to the `recordedChunks` array whenever data becomes available.

When you've finished recording, tell the MediaRecorder to stop recording:

```
mediaRecorder.stop();
```

Now you can preview what has been recorded:

```
function play() {
 var superBuffer = new Blob(recordedChunks);
 videoElement.src =
   window.URL.createObjectURL(superBuffer);
}
```

MediaRecording in a few steps 3/3

You can also download (or send to a REST endpoint) the recorded chunks:

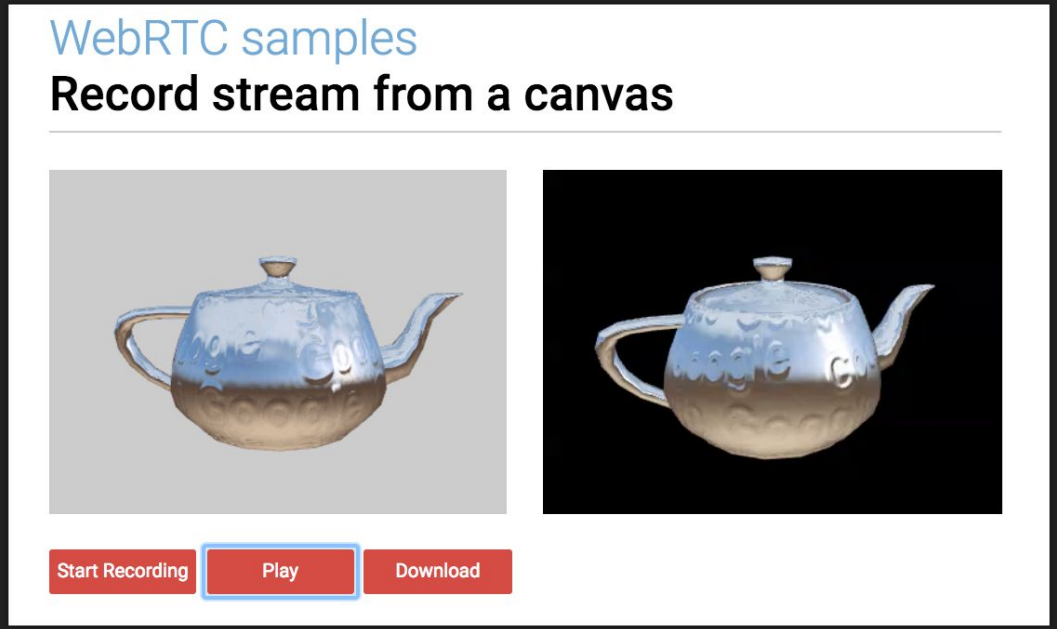
```
function download() {
 var blob = new Blob(recordedChunks, {
   type: 'video/webm'
 });
 var url = URL.createObjectURL(blob);
 var a = document.createElement('a');
 document.body.appendChild(a);
 a.style = 'display: none';
 a.href = url;
 a.download = 'test.webm';
 a.click();
 window.URL.revokeObjectURL(url);
}
```

Complete example at JsBin

- This is a JsBin version of the WebRTC sample
- We removed all unnecessary code (check if supported etc.)

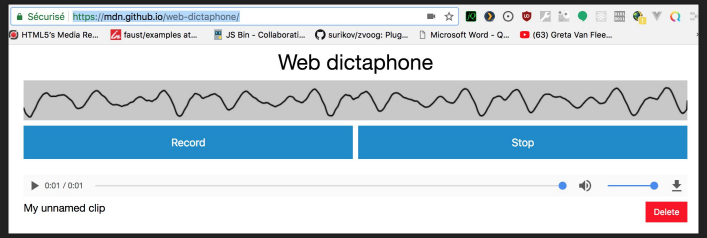

```
navigator.mediaDevices.getUserMedia({audio: true,video: true})
  .then((stream) => {
    window.stream = stream;
    // ...
  });
```

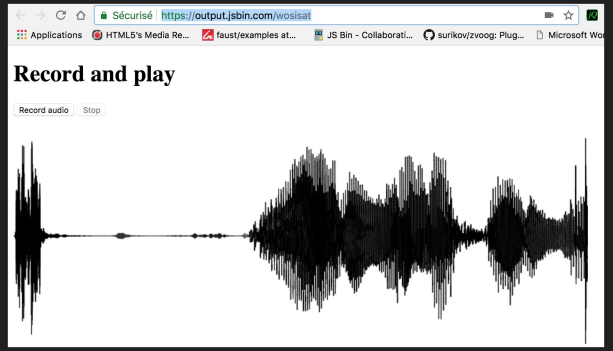
Record Stream from a canvas




```
var stream = canvas.captureStream();
```

Recording sounds...






```
navigator.mediaDevices.getUserMedia({audio: true})
  .then((stream) => {
    window.stream = stream;
    // ...
  });
```

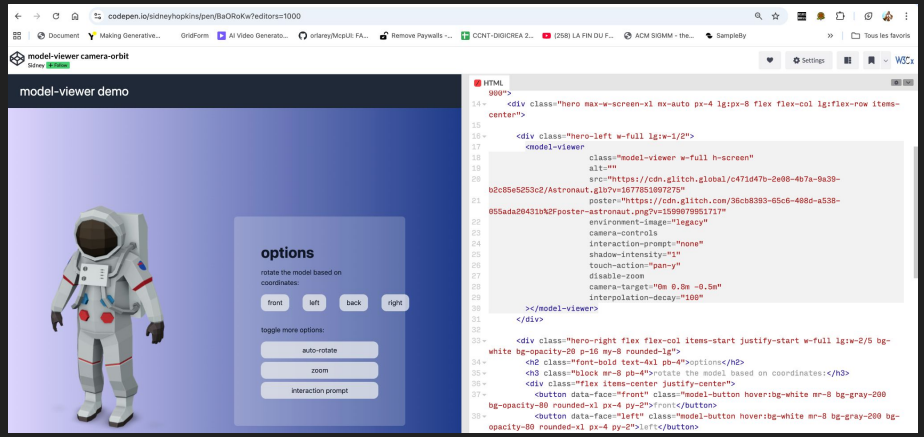
Seance 6

Before “the big three” (Angular, React, Vue), it is important to understand the concept of Web Components

Web Components?

React, Angular, VueJS, Stencil, etc. all are based on the concept of “Web Components” :

- Reusable objects made of HTML, CSS and JavaScript,
- Can be instantiated multiple times without conflicts (HTML ids, JS variables etc.)
- Beware, there are different “flavors” but only one low-level standard!
  - “Real Web Components are based on W3C and IETF standards”
  - Angular, Vue, Stencil, etc. 99% of these libraries/frameworks built on that…
  - React has his own flavor, kind of compatible, (but PReact is 100% built on the standards)
- Let’s Jump to MY COURSE ABOUT WEB COMPONENTS!



Seance 7

Debriefing Assignment “Sampler” draft

Introduction to Angular (slide deck)

Seance 8

For the sampler web app: example of use of Freesound API (GitHub repo)

We keep learning Angular (slide deck from slide 24 to XXX)

Next slides will be used later on…

SOAP (old!) and REST (yep!) Web Services

Today, 99% of the Web Services are called “REST Web Services” or “REST APIs”’ or “REST endpoint”.

REST = Representational State Transfer (i.e content negotiation between client and server, data can be plain text, HTML, JSON, XML, binary…)

- I tend to add also (unofficial) ST = Stateless!
- Most exchange are JSON based.
- The de facto standard for cloud-based applications

SOAP = old (1998), heavy (stateful, based on Remote Process Calls)

- Avoid if you can! Edt/Hyperplanning uses SOAP :-(

REST features

- Separation of responsibilities:
  - the client (e.g. mobile application, web browser) handles the user interface, while the server manages the business logic and data management.
  -
- Stateless:
  - The server retains no state between requests. For example, authentication must be sent with each request, often via tokens such as JWTs (JSON Web Tokens).
  - Gmail re-authenticate you with each request, well, nearly…
- Server responses must indicate whether or not they are cacheable.
  - Use HTTP headers such as Cache-Control or ETag.
- Uniform interface:
  - Data is a Resource! Each resource has a unique URI (Uniform Resource Identifier).
  - Resource in standard formats (often JSON or XML).
  - Use of standardized HTTP methods (GET, POST, PUT, DELETE) to do CRUD (already said)
- Layered System:
  - The architecture can be composed of several layers (authentication servers, cache servers, etc.) without the client being aware of them.

Good practices about URIs

A resource can be one single data, or a collection of data (i.e one person, or a list of persons).

There are good practices about naming URIs

You already saw some examples at https://jsonplaceholder.typicode.com/

Examples of RESTful URIs

Imagine we are building an API for a **library system**. Here's how the URIs might look:

Resource	HTTP Method	URI	Description
List all books	`GET`	`/api/books`	Retrieves a list of all books.
Get details of a specific book	`GET`	`/api/books/123`	Retrieves details of the book with ID 123.
Create a new book	`POST`	`/api/books`	Adds a new book to the library.
Update an existing book	`PUT`	`/api/books/123`	Updates the book with ID 123.
Delete a book	`DELETE`	`/api/books/123`	Deletes the book with ID 123.
List all authors	`GET`	`/api/authors`	Retrieves a list of all authors.
Get books by a specific author	`GET`	`/api/authors/45/books`	Retrieves all books written by author ID 45.