

# Some tools for building web-based groupware

Michel Beaudouin-Lafon

[mbl@lisn.fr](mailto:mbl@lisn.fr)

# JavaScript

- More powerful than you think
- Functional: functions are first-class objects
  - **function** addmap(array, **f**) {  
    **var** sum = 0  
    **for** (var item of array)  
        sum += **f**(item)  
    **return** sum  
}
  - addmap([1, 2, 3], **function(x) { return x\*x }**) // 14

# Modern JavaScript

- Anonymous functions can be noted
  - param => expression
  - or*
  - (p1, p2) => { ... body ... **return** expression }
- Previous example:
  - addmap([1, 2, 3], **function(x) { return x\*x }**) // 14
  - becomes*
  - addmap([1, 2, 3], **x => x\*x**) // 14

# Modern JavaScript

- Classes
  - **class** **Point** {  
    **constructor**() { **this.x** = 0; **this.y** = 0 }  
    **moveBy**(dx, dy) { **this.x** += dx; **this.y** += dy }  
}
  - **var** p = new **Point**(); p.**moveBy**(10, 20)
- Other useful features, including
  - Iterators and generators
  - Arrow functions, Promises

# Typescript

<https://www.typescriptlang.org>

- Javascript with types
  - **function** fib(n: **number**): **number** { ... }
- Powerful type system
  - Generic types:  
**function** swap<**T**>(a: **T**, b:**T**): **T** { **var** x = a; a = b; b = x }
- Other powerful features
- Translates to plain Javascript
- Works with plain Javascript libraries
  - Through definition files

# Node.js

<https://nodejs.org/en/>

- Use JavaScript to program web app servers
- Module system
  - Modules are installed with **npm**, the Node Package Manager: `npm install <package>`
  - Packages can be local or global (`npm -g`)
  - A package is imported with `require`:  
`var fs = require('fs'); // access to file system`  
`fs.existsSync(...)`
  - Node has built-in modules to access the OS: file system, processes, HTTP protocol, ...

# Node.js

- Node implements a reactive programming model based on events and event handlers
- Asynchronous event handling
  - `var Emitter = require('events').EventEmitter`  
`var chatRoom = new Emitter(), participants = []`
  - `chatRoom.on('hello', name => {  
 console.log(name, 'says Hi!')`  
`});`
  - `chatRoom.on('hello', name => {  
 participants.push(name)`  
`});`
  - `chatRoom.emit('hello', 'Alice')`

# Node.js

- Using continuations (or callbacks) to program in an asynchronous world
  - Many node modules use callbacks, which are called when the task is actually performed:
    - `var fs = require('fs')`  
`var fd = fs.open('myFile', 'w')`  
`if (fd) fs.write(fd, 'Hello')` // wrong: file not opened!
    - `fs.open('myFile', 'w', (err, fd) => {`  
    `if (! err) fs.write(fd, 'Hello', (err, fd) => { ...});`  
`});`  
// green function called when **open** is finished  
// blue function called when **write** is finished

# Promises

- Facilitate asynchronous programming
- An asynchronous call returns a **Promise** object instead of taking a callback
- The **then** function of the promise is used to specify what to do when the call succeeds:
  - ```
var fs = require('fs'), util = require('util')
var readdir = util.promisify(fs.readdir)
readdir(path).then(files => {
    console.log(files.join('\n'))
});
```

# Promises

- Instead of being nested like callbacks, Promises are chained
- `open(path, 'w').then(fd => {  
    write(fd, 'Hello')  
    return fd  
}).then(fd => close(fd))`

# async/await

- Syntactic sugar around promises
- **async function** **f()** {  
    **return** new Promise...  
}
- ...  
    **await** **f()**   // wait until promise is resolved
- Previous example:  
    **fd** = **await** open(path, 'w')  
    **await** write(**fd**, 'Hello')  
    **await** close(**fd**)

# Express HTTP server

<https://expressjs.com>

- Basis of many web applications
  - `% npm install express`
  - `var express = require('express'),`  
`app = express()`  
`app.get('/', (request, response) => {`  
 `response.send('Hello World')`  
`})`  
`app.listen(8080)`
  - `// serve static files`  
`app.use(express.static(__dirname + '/public'))`

# socket.io

<https://socket.io>

- Communication between web page and server
- Server :
  - **% npm install socket.io**
  - **var io** = require('socket.io').listen(http)  
**io.on('connection', client => {**  
    **client.on('msg', data => {**  
        // send to all other clients  
        **client.broadcast.emit('msg', data)**  
    **})**  
**})**

# socket.io

- Client (web page)
  - `<script src="/socket.io/socket.io.js"></script>`  
`<script>`  
    **var client** = io()  
    **client.on**('connect', () => client.emit('msg', 'Hello'))  
    **client.on**('msg', data => \$('#chat').append(data))  
`</script>`

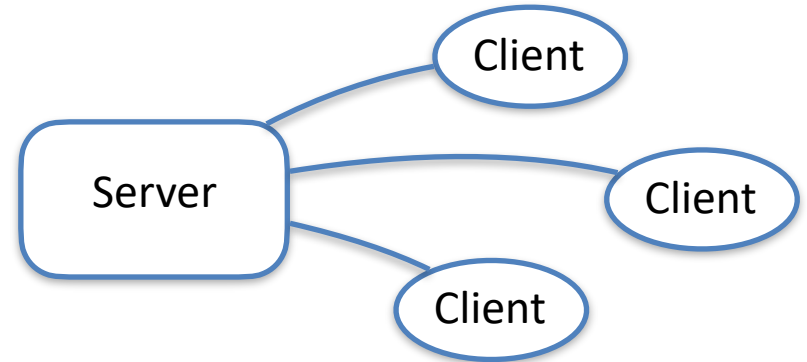
# socket.io

- Server

```

var express = require('express'),
    app = express()
app.get('/', (request, response) => {
  response.send('Hello World')
})
app.listen(8080)
app.use(express.static(__dirname + '/public'))
// manage live connections
var io = require('socket.io').listen(http)
io.on('connection', client => {
  client.on('msg', data => {
    // send to all other clients
    client.broadcast.emit('msg', data)
  })
})

```

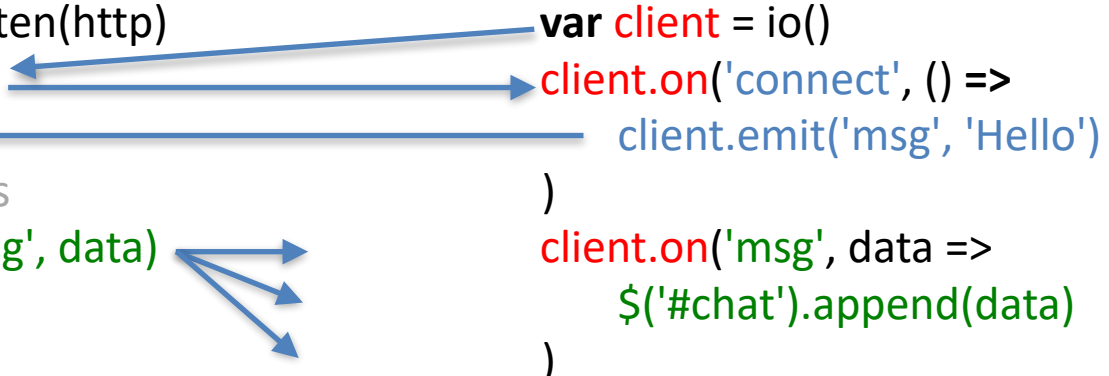


- Client (web page)

```

<script src="/socket.io/socket.io.js"></script>
<script>
var client = io()
client.on('connect', () =>
  client.emit('msg', 'Hello')
)
client.on('msg', data =>
  $('#chat').append(data)
)
</script>

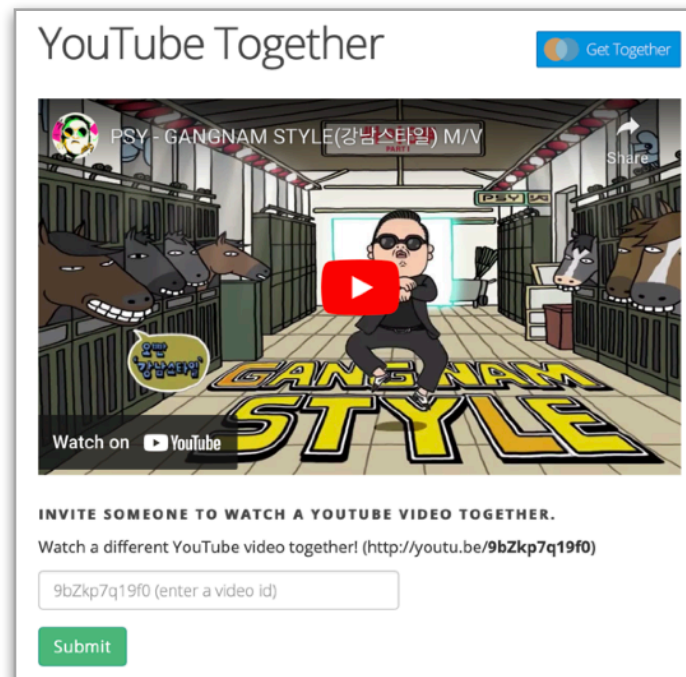
```



# TogetherJS

<https://togetherjs.com>

- Collaboration features for web applications:
  - Text/audio chat, User focus/presence, Co-browsing, Real-time content sync



# ShareDB

<https://share.github.io/sharedb/>

- Shared text strings and JSON objects synchronized through operational transformation
- Server
  - `var ShareDB = require('sharedb')`  
`var share = new ShareDB(options)`  
...
  - `// when a client connects to the server:`  
`websocketServer.on('connection', ws => {`  
 `share.listen(new WebSocketJSONStream(ws))`  
`})`

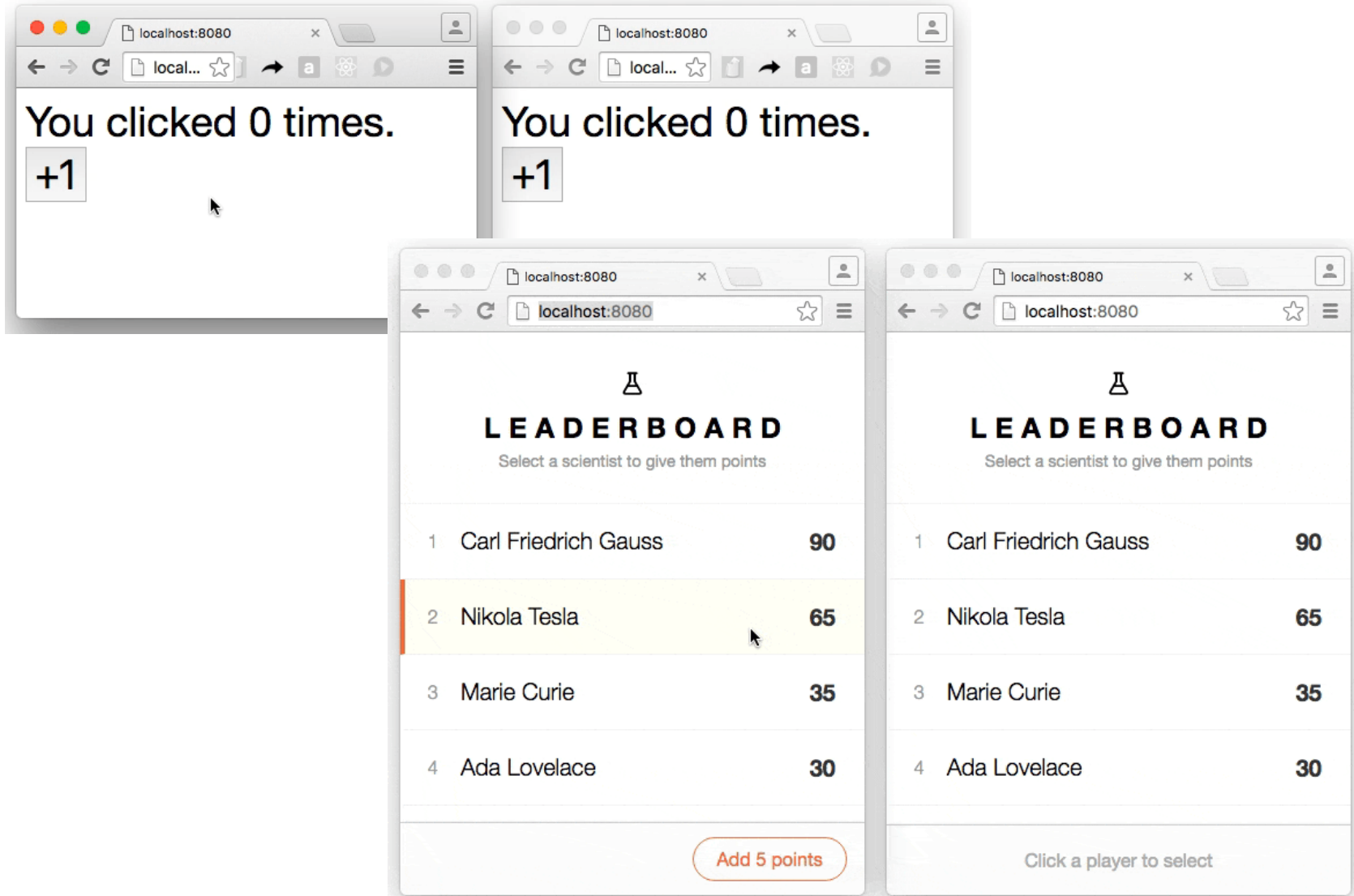
# ShareDB

<https://share.github.io/sharedb/>

- Client that synchronizes text in a textarea
  - `var ShareDB = require('shared/lib/client')`  
`var sock = new WebSocket('<server address>')`  
`var connection = new ShareDB.Connection(sock)`  
...
  - `var doc = connection.get('mydoc', 'textarea')`  
`doc.subscribe(err => {`  
    `if (err) throw err`  
    `if (! doc.type) doc.create({text: 'hello'})`  
}
  - `doc.on('op', () => console.log(doc.data.text))`

# ShareDB

<https://share.github.io/sharedb/>





<https://yjs.dev>

- Shared types that are synced peer-to-peer using Convergent Replicated Data Types (CRDTs)
  - **import \* as Y from 'yjs'**  
**import { WebRTCProvider } from 'y-webrtc'**
  - **var ydoc = new Y.Doc()**  
**var provider = new WebRTCProvider('demo', ydoc)**  
**var yarray = ydoc.getArray('data')**  
**yarray.observe(ev => console.log(yarray.toArray()))**
  - **yarray.push([1])**



<https://yjs.dev>

ProseMirror      Quill      Drawing

---


Create Version

> Changes since last version

**B** *I* <> 🔗 | Insert ▾ Type... ▾ | ↶ ↷ | “ ” □

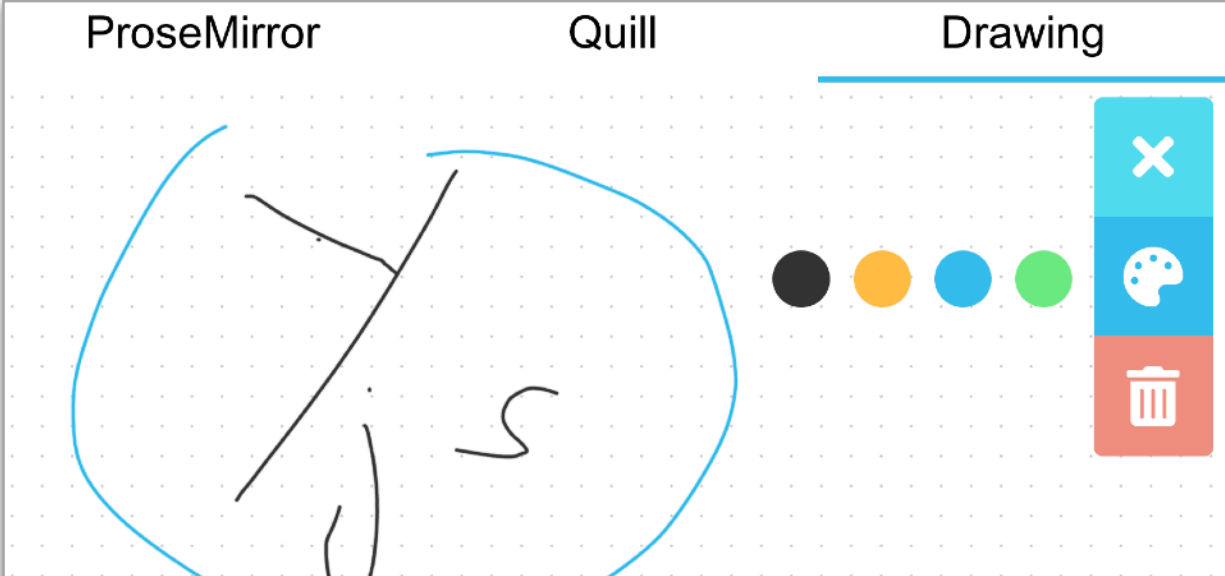
Bob

Yjs is a shared editing framework. It exposes **Shared Types** that can be manipulated like any other data type. But they are synced automatically!



ProseMirror      Quill      Drawing

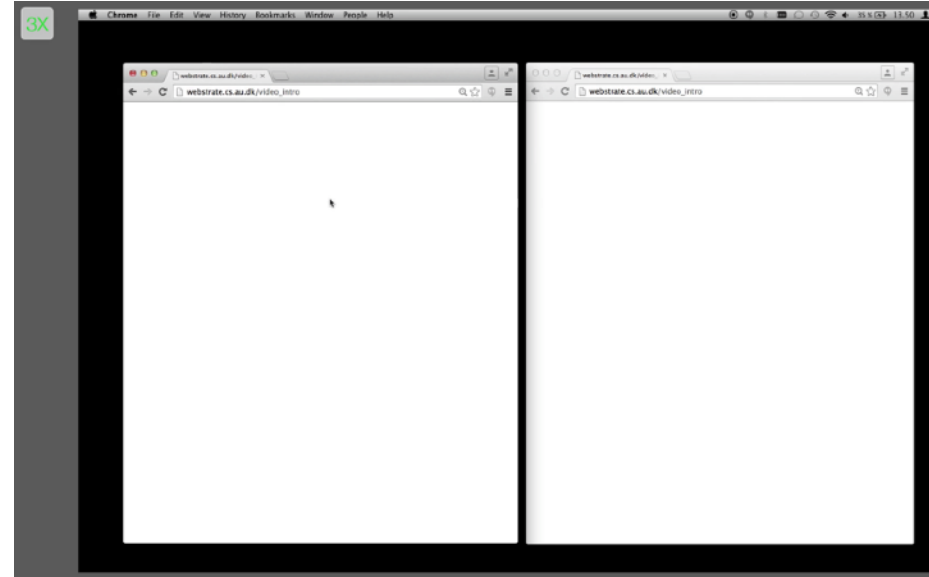
---

A drawing canvas with a dotted grid background. A hand-drawn diagram is shown, consisting of a large blue circle on the left, a vertical line, a diagonal line, and a horizontal line. To the right of the diagram are four colored circles: black, orange, blue, and green. On the far right is a vertical toolbar with three buttons: a blue button with a white 'X', a blue button with a white palette icon, and a red button with a white trash can icon.

# Webstrates

<https://webstrates.github.io>

- Web server that synchronizes the pages it serves as they are edited
- Real-time shared editing “out of the box”
- Transclusion: embed a webstrate in another webstrate
- API for advanced features: permissions, assets, cookies, messaging, transient content, ...

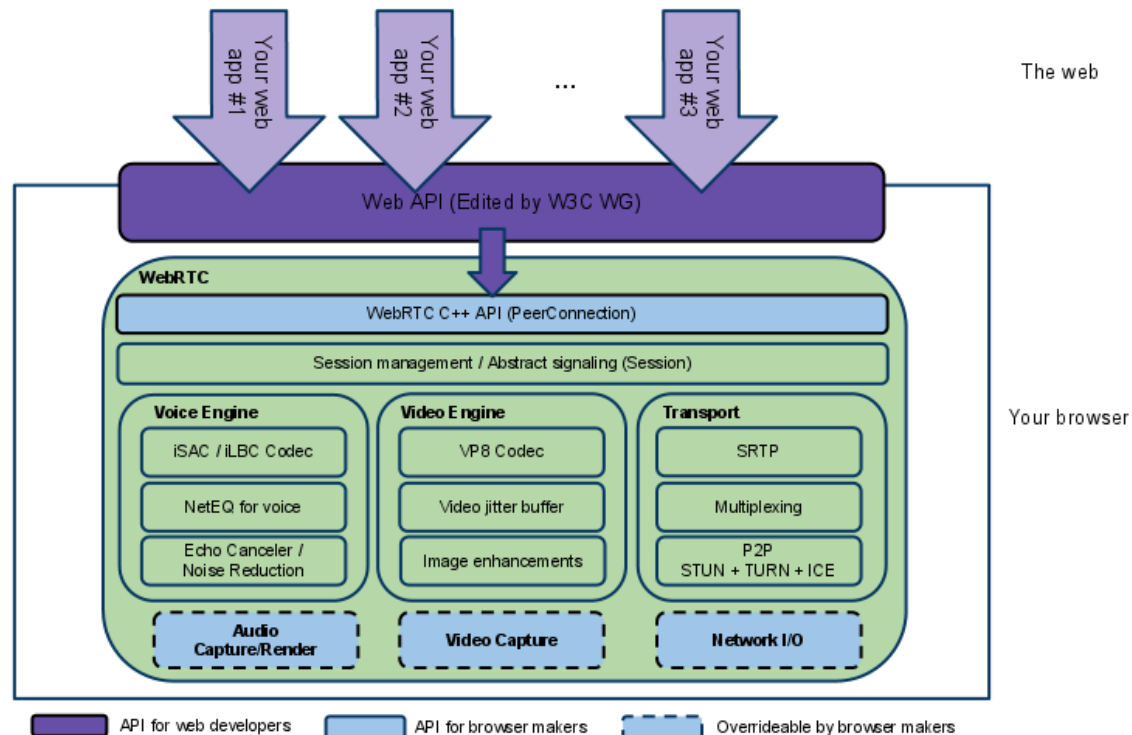


# WebRTC

<https://webrtc.org>

- Set of APIs for real-time communication
- Supports audio, video, data
- Web APIs as well as native APIs

- But:
- Complex
- Inconsistent support across browsers



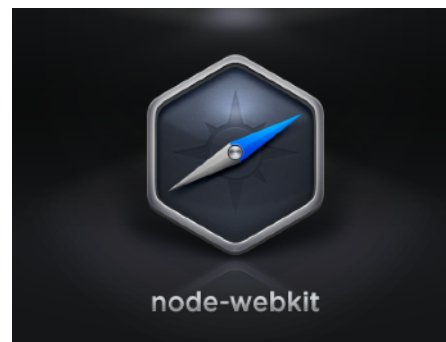
# WebRTC

- // Get access to local camera / microphone  
`stream = await getUserMedia({'video': true, 'audio': true})`  
`videoElement.srcObject = stream // local playback`
- // Connect to peer  
`conn = new RTCPeerConnection(config)`  
`stream.getTracks().forEach(t => conn.addTrack(t, stream))`  
... // negotiate offer through ICE candidates
- // Connection established: show remote video/audio  
`conn.addEventListener('connectionstatechange', ev => {`  
    `if (conn.connectionState === 'connected')`  
        `conn.addEventListener('track', async (ev) => {`  
            `remoteVideo.srcObject = ev.streams[0] })`  
    `})`

# Node-webkit (nwjs)

<https://nwjs.io>

- Combines a node.js server and a web browser
- Access to desktop: files, menus, ...
- Desktop-based web applications
- Avoids web protocol issues (same-origin, ...)
- Global namespace shared between node server and every web window



# Node-webkit

- A web page that lists the files in the current directory

- `<script>`

```
var fs = require('fs') // node module in HTML file!
```

```
fs.readdir('.', function(err, files) {
```

```
  files.map(function(filename) {
```

```
    $('#list').append('<li>'+filename+'</li>');
```

```
  });
```

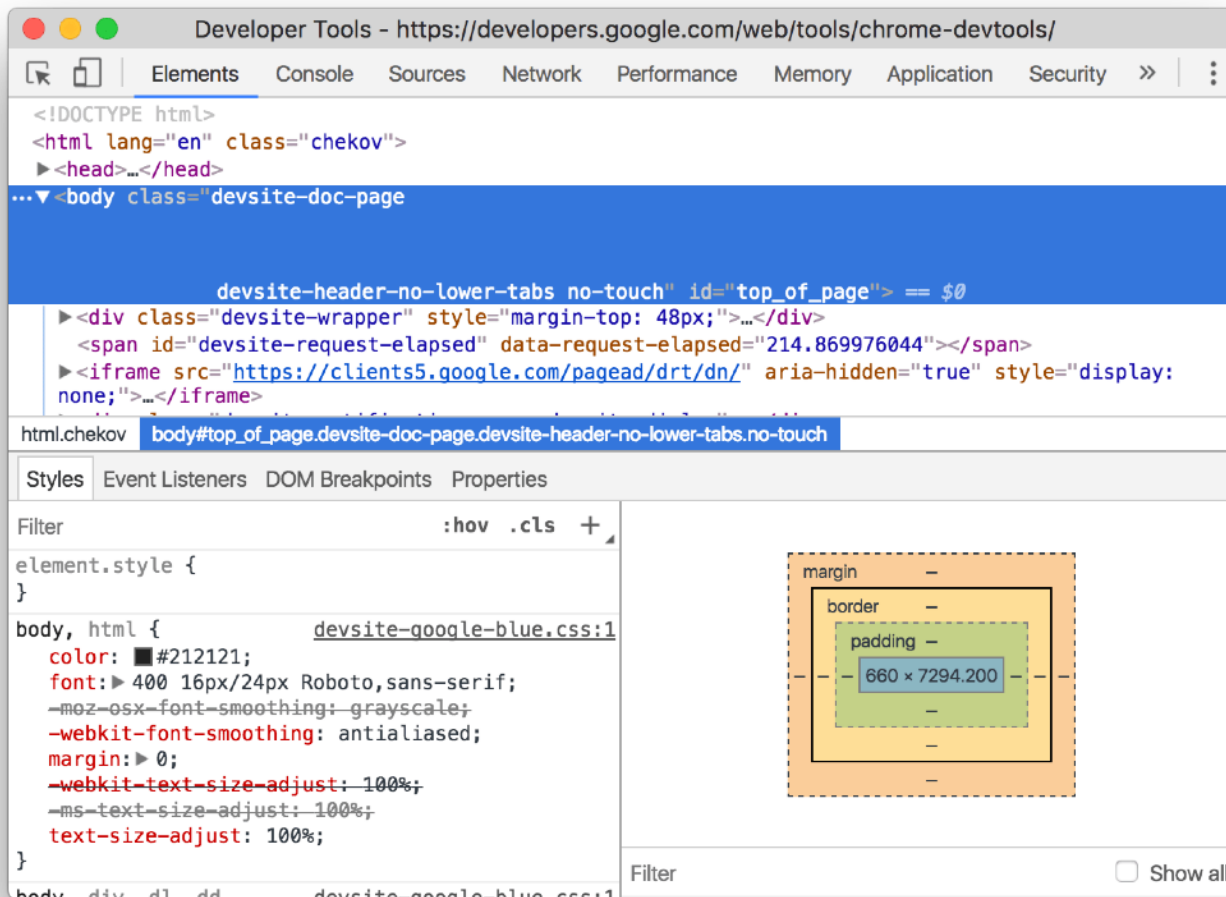
```
});
```

```
</script>
```

```
<ul id='list'></ul>
```

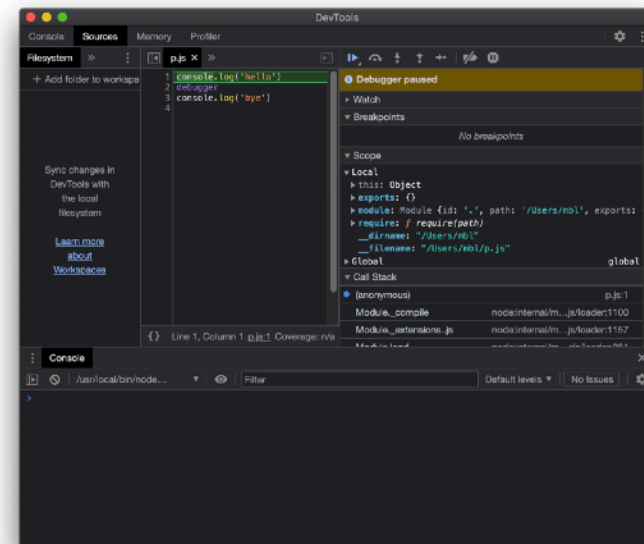
# Debugging

- In the browser: developer tools



# Debugging

- Node.js
  - Built-in debugger:
    - `% node inspect myscript.js`
    - `% node debug myscript.js`
    - (you need to add calls to **debugger** in the script)
  - Remote debugging with web inspector
    - Run the script in debug mode
      - `% node inspect myscript.js`
    - or
      - `% node --inspect myscript.js`
    - On Chrome, goto <chrome://inspect> and configure network targets



# WebStorm (IntelliJ), VSCode (Microsoft)

<https://www.jetbrains.com/webstorm/>

<https://code.visualstudio.com>

- Integrated Development Environment (IDE)
- Project manager
- Editor
  - Multiple cursors
  - Code completion
  - Refactoring
  - Linting
- Debugger
- Support Typescript

Master Informatique Paris-Saclay - (c) Michel Beaudouin-Laton 2022-2023

bin/www

Project test2 (~/Desktop/test2)

- bin
- node\_modules (library home)
- public
- routes
  - index.js
  - users.js
- views
  - app.js
  - package.json
- External Libraries
  - Node.js v0.11.13 Core Modules

```
1 var express = require('express');
2 var path = require('path');
3 var favicon = require('serve-favicon');
4 var logger = require('morgan');
5 var cookieParser = require('cookie-parser');
6 var bodyParser = require('body-parser');
7
8 var routes = require('./routes/index');
9 var users = require('./routes/users');
10
11 var app = express();
12
13 app.use(express.static(path.join(__dirname, 'views')));
14 app.set('view engine', 'jade');
15
16
17 // uncomment after placing your favicon in /public
18 //app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
19 app.use(logger('dev'));
20 app.use(bodyParser.json());
21 app.use(bodyParser.urlencoded({ extended: false }));
22 app.use(cookieParser());
23 app.use(express.static(path.join(__dirname, 'public')));
24
25 app.use('/', routes);
26 app.use('/users', users);
27
28 // catch 404 and forward to error handler
29 app.use(function(req, res, next) {
30   var err = new Error('Not Found');
```

Debugger bin/www

Frames

- app.js:25
- Module.\_compile(), module.js:449
- Module.\_extensions.js(), module.js:467
- Module.load(), module.js:349
- Module.\_load(), module.js:305
- Module.require(), module.js:357
- require(), module.js:373
- www:7
- Module.\_compile(), module.js:449
- Module.\_extensions.js(), module.js:467
- Module.load(), module.js:349
- Module.\_load(), module.js:305
- Module.runMain(), module.js:490

Variables

- Local
  - app = function (req, res, next) {
  - bodyParser = function (arg0) {
  - exports = Object
  - module = Module {id: "/Users/mbi/Desktop/test2/app.js"}
  - path = Object
    - \_\_dirname = "/Users/mbi/Desktop/test2"
    - \_\_filename = "/Users/mbi/Desktop/test2/app.js"
  - this = Object
  - Functions
- Global

Watches

No watches

5: Debug 6: TODO Terminal

8:8 LF UTF-8

The screenshot shows the VS Code editor interface. The Explorer sidebar on the left displays a project structure with folders like 'server', 'web', and 'src'. The main editor area shows the file 'api.js' with the following code:

```

1 // @ts-check
2
3 import * as config from './config';
4 import * as moment from 'moment';
5
6 /**
7  * @param {boolean} [testMode] Enable demo mode.
8  *
9  * @return {Promise<number>} Number of users.
10 */
11 export async function getUserCount(testMode = false) {
12   const response = await fetch(`${config.apiEndpoint}/v0/numberServed`);
13   const data = await response.jsonX();
14   if (testMode) {
15     return data.numberServed * moment.now();
16   }
17   return data.numberServed;
18 }
19
20
21
22

```

A dropdown menu is open over the `moment.now()` call on line 15, listing various moment.js methods such as `normalizeUnits`, `now`, `months`, `monthsShort`, `version`, `duration`, `parseZone`, `defineLocale`, `isDuration`, `calendarFormat`, `isMoment`, and `toString`. A tooltip for the `moment.now()` function is also visible, stating: "Returns unix time in milliseconds. Overwrite for profit."

The bottom status bar shows the current cursor position: "Ln 15, Col 45". The Problems panel at the bottom indicates a TypeScript error: "Property 'jsonX' does not exist on type 'Response'. Did you mean 'json'? ts(2551) [13, 33]".