

**DEBUGGING**

---

**NODEJS**



DEBUGGING

---

NODEJS

# AGENDA

- ▶ Introduction
- ▶ Using the NodeJS build-in debugger
  - ▶ Watchers
  - ▶ Commands
- ▶ Using node-inspector
- ▶ Using VSCode
- ▶ Labs

# INTRODUCTION

- ▶ V8 comes with an extensive debugger, accessible out-of-process via a simple **TCP protocol**.
  - ▶ <https://github.com/v8/v8/wiki/Debugging%20Protocol>
- ▶ Node.js has a built-in client for this debugger.
- ▶ Simple step and inspection is possible.

# INTRODUCTION



```
% node debug myscript.js  
< debugger listening on port 5858  
connecting... ok  
break in /home/indutny/Code/git/indutny/myscript.js:1  
  1 x = 5;  
  2 setTimeout(function () {  
  3   debugger;  
debug>
```



# INTRODUCTION

- ▶ Set '**debugger;**' statements as code-breakpoints in your script file to stop execution of the code.



```
// myscript.js
x = 5;
setTimeout(function () {
  debugger;
  console.log("world");
}, 1000);
console.log("hello");
```

- ▶ Then once the debugger is run, it will break on line 4.

# WATCHERS

- ▶ Watch expression and variable values while debugging
- ▶ On breakpoint each watcher will be evaluated and displayed before the breakpoint's source code listing.
- ▶ Start => `watch("my_expr")`
- ▶ Remove => `unwatch("my_expr")`
- ▶ List => `watchers`

```
debug> watch("requestcounter")
debug> cont
< server listening
break in app.js:6
Watchers:
  0: requestcounter = 0

4
5 http.createServer((req, res) => {
> 6   debugger;
7   requestcounter++;
8   res.end("hello world");
debug> cont
break in app.js:6
Watchers:
  0: requestcounter = 1

4
5 http.createServer((req, res) => {
> 6   debugger;
7   requestcounter++;
8   res.end("hello world");
debug> _
```

**DEMO WATCHERS**



# COMMANDS

- ▶ Command categories
  - ▶ Stepping commands
  - ▶ Breakpoint commands
  - ▶ Info commands
  - ▶ Execution control commands
  - ▶ Various commands

# STEPPING COMMANDS

command	alias	explanation
cont	c	continue execution
next	n	step next
step	s	step into
out	o	step out
pause		pause execution

# BREAKPOINT COMMANDS

command	alias	explanation
setBreakpoint()	sb()	Set breakpoint on current line
setBreakpoint(line)	sb(line)	Set breakpoint on specific line
setBreakpoint('fn()')	sb(...)	Set breakpoint on a first statement in functions body
setBreakpoint('script.js', 1)	sb(...)	Set breakpoint on first line of script.js
clearBreakpoint('script.js', 1)	cb(...)	Clear breakpoint in script.js on line 1

# INFO COMMANDS

command	alias	explanation
backtrace	bt	Print backtrace of current execution frame
list(5)		List scripts source code with 5 line context (5 lines before and after)
watch(expr)		Add expression to watch list
unwatch(expr)		Remove expression from watch list
watchers		List all watchers and their values
repl		Open debugger's repl for evaluation in debugging script's context

# EXECUTION CONTROL AND VARIOUS OTHER COMMANDS

command	alias	explanation
run		Run script (automatically runs on debugger's start)
restart		restart script
kill		kill script
scripts		List all loaded scripts
version		Display v8's version

# INFO COMMANDS

command	alias	explanation
backtrace	bt	Print backtrace of current execution frame
list(5)		List scripts source code with 5 line context (5 lines before and after)
watch(expr)		Add expression to watch list
unwatch(expr)		Remove expression from watch list
watchers		List all watchers and their values
repl		Open debugger's repl for evaluation in debugging script's context



laat zien hoe je met repl een  
intermediate window achtige interface  
met de huidige sessie hebt

# DEMO COMMANDS

# USING NODE-INSPECTOR

- ▶ NPM module which enables Chrome Dev Tool (Blink) debugging of NodeJS code.
- ▶ steps
  - ▶ `npm install node-inspector -g`
  - ▶ in one terminal: start "`node-inspector &`"
  - ▶ in other terminal: start "`node -debug script.js`"
  - ▶ start Chrome and use address from first terminal

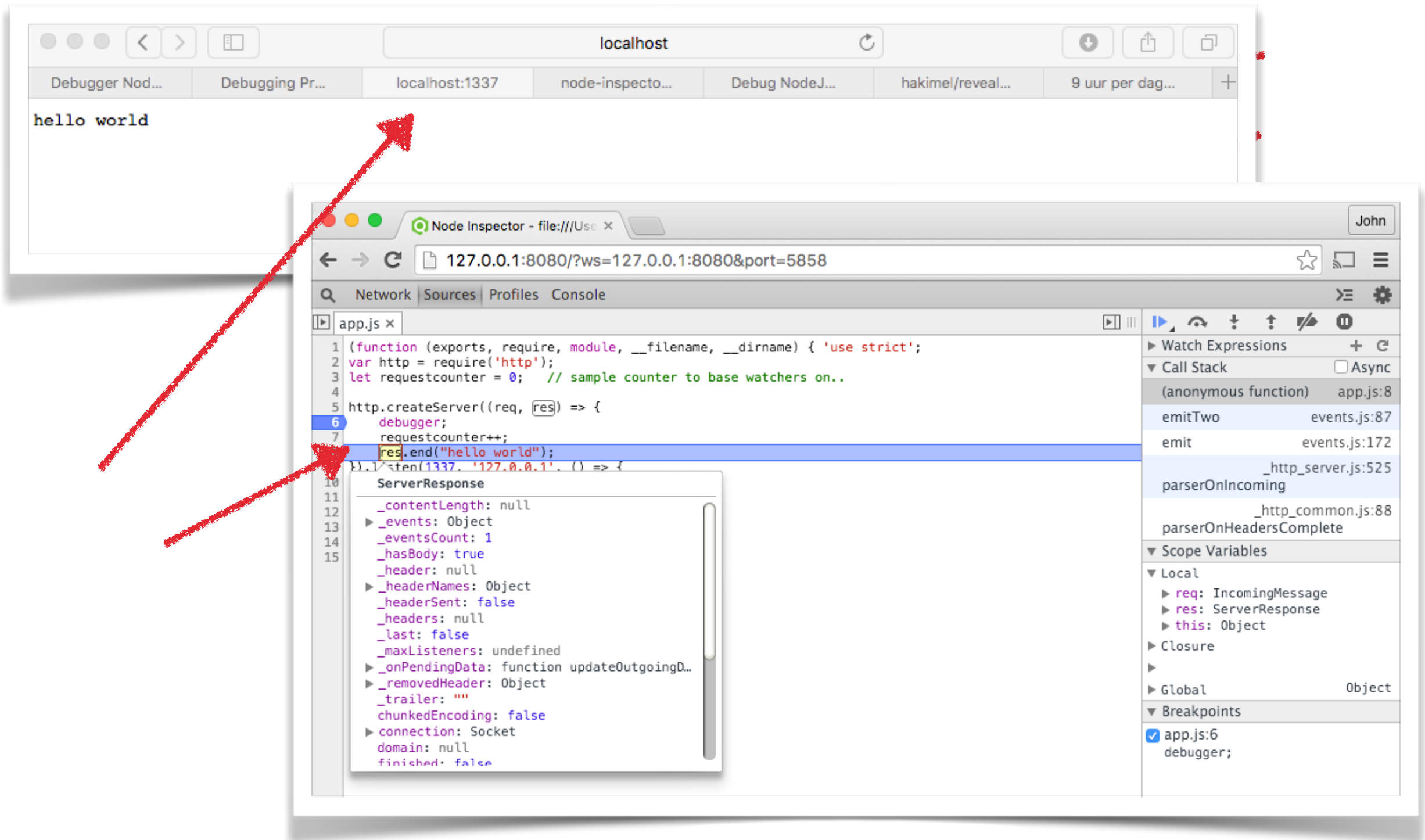
# USING NODE-INSPECTOR

```
1. node
Johns-MacBook-Pro:debugging johngorter$ node --debug app.js
Debugger listening on port 5858
server listening

2. node
Johns-MacBook-Pro:debugging johngorter$ node-inspector
Node Inspector v0.12.5
Visit http://127.0.0.1:8080/?ws=127.0.0.1:8080&port=5858 to start debugging.
```

- ▶ start the inspector before starting the script in debug mode..

## USING NODE-INSPECTOR



## USING NODE-INSPECTOR

The screenshot displays the Node-Inspector web interface. The left pane shows the source code of `yahoo-finance-with-nodeJS.js`. The right pane contains the debug console and various breakpoint categories. Annotations with red arrows highlight key features:

- Our code that node is running**: Points to the source code editor.
- watch after expressions**: Points to the **Watch Expressions** panel, which shows `financeDetails: Array[58]` and `keyStr: Array[58]`.
- Set a breakpoint in a certain location**: Points to the **Breakpoints** panel, where a breakpoint is set at `yahoo-finance-with-nodeJS.js:65`.
- Yep! We got the good old console for you...**: Points to the **Console** panel at the bottom, which shows the output of `console.log`.

```
1 (function (exports, require, module, __filename, __dirname) { /**
2  * Fetch data from the web and save it into a Google Sheet document
3  *
4  * Requirements:
5  * 1. NodeJS
6  * 2. npm install request
7  * 3. npm install cheerio
8  * 4. npm install edit-google-spreadsheet
9
10 * Author: Ido Green | plus.google.com/+Greenido
11 * Date: 15 Aug 2013
12 */
13
14 var request = require('request');
15 var cheerio = require('cheerio');
16 var Spreadsheet = require('edit-google-spreadsheet');
17
18 // Some parameters
19 // TODO: move from the global scope
20 var ticker = "LVS";
21 var yUrl = "http://finance.yahoo.com/q/ks?s=" + ticker;
22 var financeDetails = new Array();
23 var keyStr = new Array();
24
25
26 // Upload our data to G-Sheet
27 function sheetReady(err, spreadsheet) {
28   if (err) throw err;
29   spreadsheet.add({ 1: { 1: "Attribute" } });
30   spreadsheet.add({ 1: { 2: "Value" } });
31
32   spreadsheet.add({
33     2: {
34       1: keyStr
35     }
36   });
37   spreadsheet.add({
38     2: {
39       2: financeDetails
40     }
41   });
42
43   spreadsheet.send(function(err) {
44     if(err) throw err;
45     console.log("Updated " + financeDetails.length + " Items with data");
46   });
47 }
48
49 //
50 // The main call to fetch the data, parse it and work on it.
51 //
52 request(yUrl, function (error, response, body) {
53   if (!error && response.statusCode == 200) {
54     var $ = cheerio.load(body);
55   }
56 })
```

Line 70, Column 1

Watch Expressions

- financeDetails: Array[58]
- keyStr: Array[58]

Call Stack

Not Paused

Scope Variables

Not Paused

Breakpoints

- ☒ yahoo-finance-with-nodeJS.js:65  
\$(tData).each(function(j, val) {

DOM Breakpoints

No Breakpoints

XHR Breakpoints

No Breakpoints

Event Listener Breakpoints

- ☐ Animation
- ☐ Control
- ☐ Clipboard
- ☐ DOM Mutation
- ☐ Device
- ☐ Keyboard
- ☐ Load
- ☐ Mouse
- ☐ Timer
- ☐ Touch

# DEMO NODE- INSPECTOR

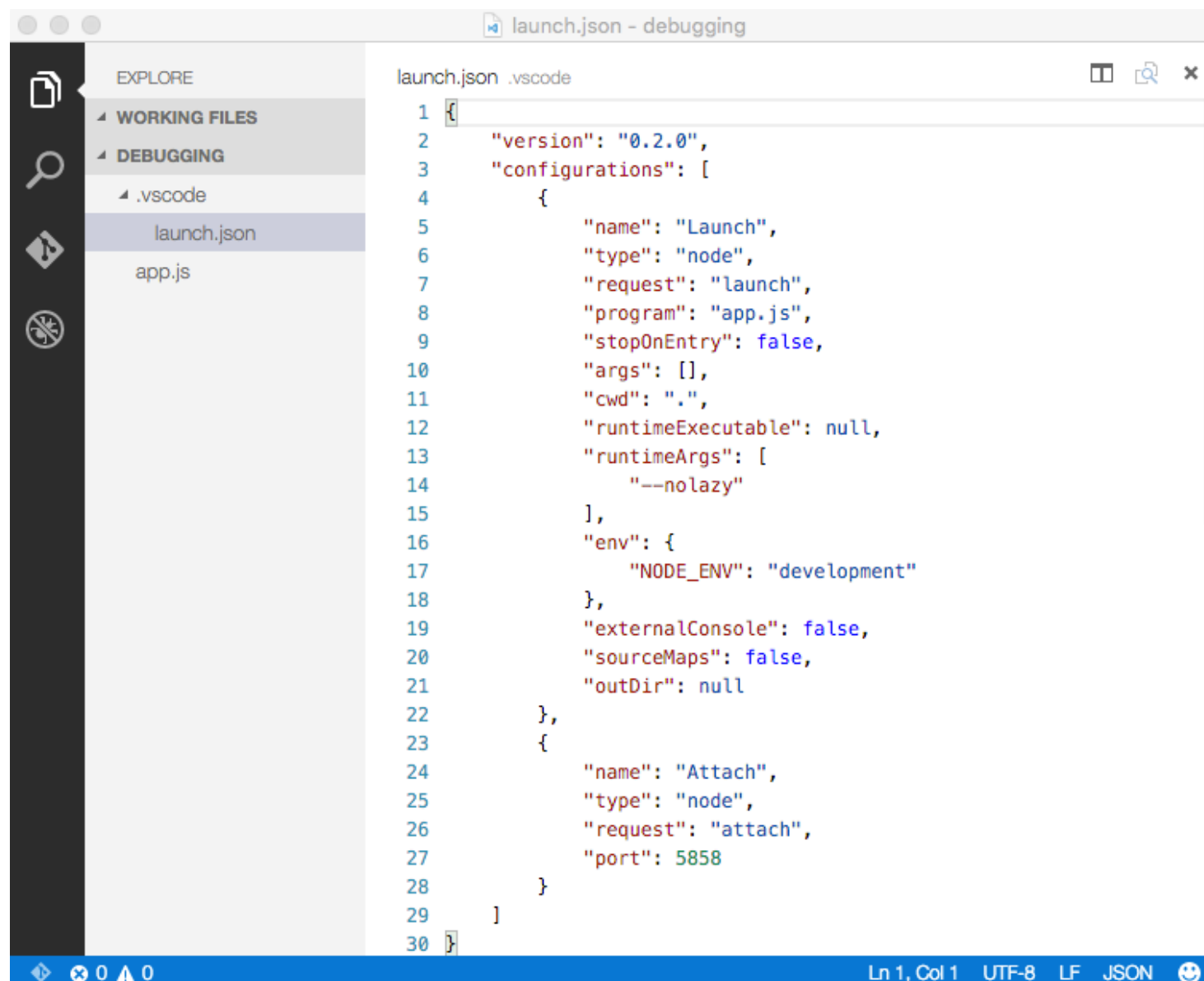


# USING VSCODE

- ▶ Lightweight IDE for web development from Microsoft
- ▶ Free download
- ▶ F5 experience

# USING VSCODE

- ▶ press F5 and let VSCODE generate startup configuration

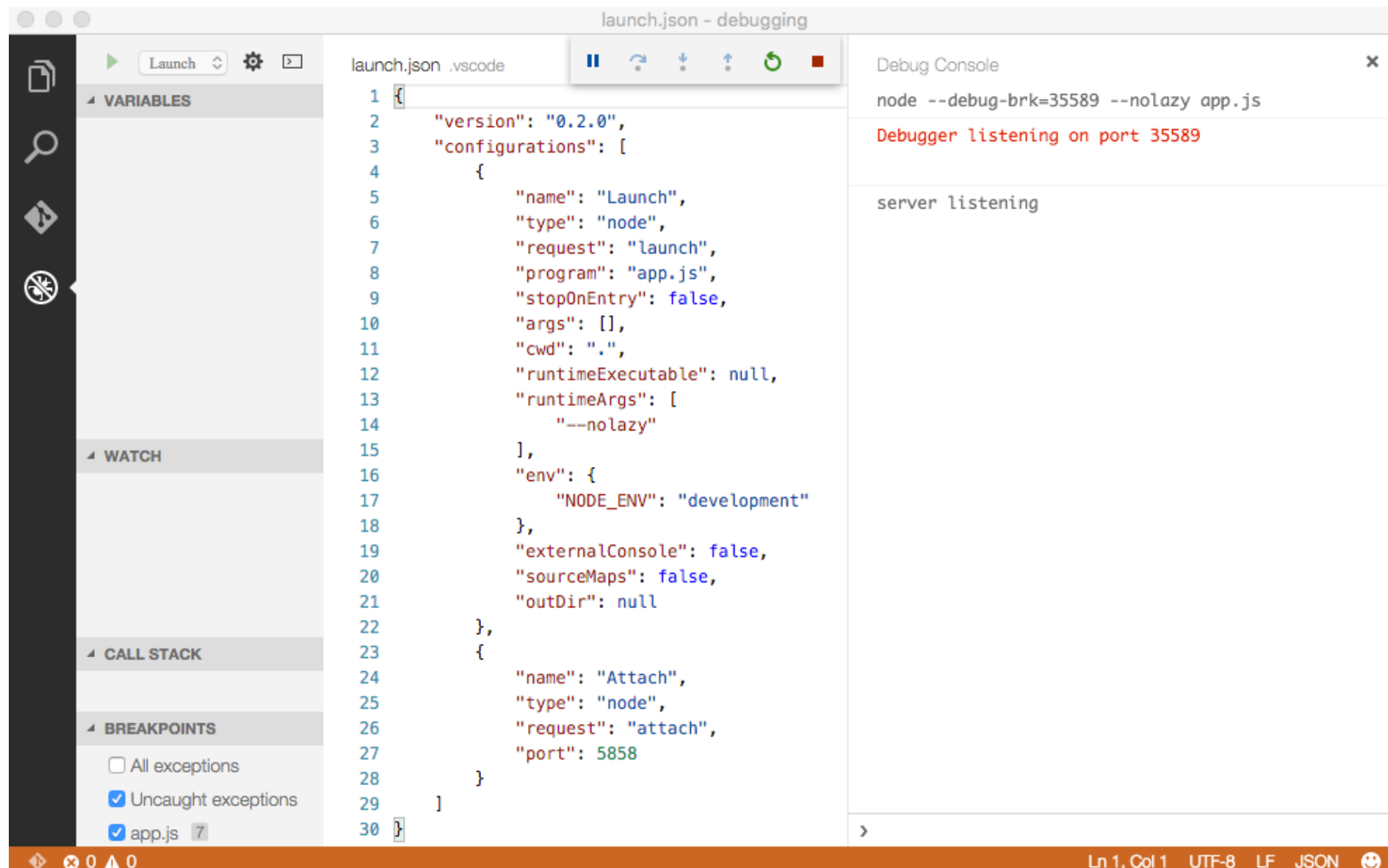


The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORER' sidebar is open, showing the file structure with 'WORKING FILES' and 'DEBUGGING' sections. Under 'DEBUGGING', the '.vscode' folder is expanded, showing 'launch.json' and 'app.js'. The main editor area displays the 'launch.json' file, which contains a JSON configuration for debugging. The configuration includes a 'Launch' configuration for running the application and an 'Attach' configuration for attaching to a running process. The status bar at the bottom indicates the current position is Line 1, Column 1, and the file is encoded in UTF-8 with LF line endings.

```
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Launch",
6       "type": "node",
7       "request": "launch",
8       "program": "app.js",
9       "stopOnEntry": false,
10      "args": [],
11      "cwd": ".",
12      "runtimeExecutable": null,
13      "runtimeArgs": [
14        "--no-lazy"
15      ],
16      "env": {
17        "NODE_ENV": "development"
18      },
19      "externalConsole": false,
20      "sourceMaps": false,
21      "outDir": null
22    },
23    {
24      "name": "Attach",
25      "type": "node",
26      "request": "attach",
27      "port": 5858
28    }
29  ]
30 }
```

## USING VSCODE

- ▶ press F5 again to start debugging



**DEMO VS CODE**