

Votre ami le débugger

C'est quoi ?

Un outil génial permettant de :

- stopper l'exécution
- inspecter le contexte

Pourquoi faire ?

- comprendre ce qui se passe à l'exécution
...plutôt que d'ajouter des logs

Je trouve ça où ?

- dans le navigateur
- dans l'IDE

Comment je m'en sers ?

- activer le debugger dans l'application
- placer un point d'arrêt dans le code (breakpoint)
- exécuter le code
 - via un test unitaire
 - manuellement en navigant dans l'application

Qu'est ce que je peux faire avec ?

Mettre le programme en pause

- à la volée
- à une ligne spécifique du programme
- lorsqu'une erreur est déclenchée
- en fonction d'une valeur d'une variable
- en fonction d'un autre breakpoint

Inspecter l'état du programme

- inspecter les valeurs
- inspecter la pile d'appels
- appeler une méthode pour voir son résultat

Comprendre le déroulement d'un algorithme

- dérouler pas à pas
- entrer/sortir d'une fonction
- revenir en arrière

Influer sur le programme

- modifier une valeur
- déclencher des erreurs
- changer le flot d'exécution

Avec Node.js

Lancer le serveur avec l'option `--inspect` :

```
{
  "scripts": {
    "start:debug": "node --require ts-node/register --inspect src/index.ts"
  }
}
```

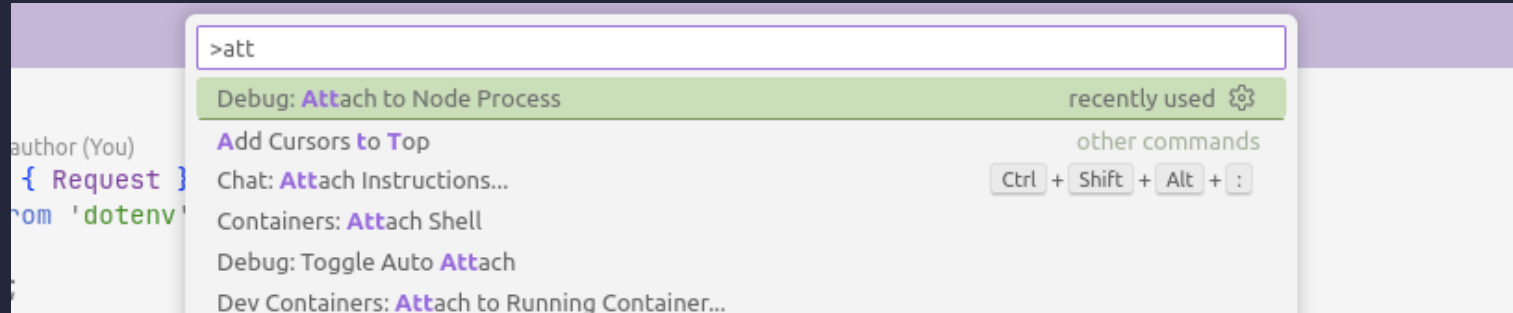
Définir un point d'arrêt

```
20
21 app.get('/fizzbuzz/:i', (req: Request<{i: number}>, res) => {
22   const i = req.params.i
23
24   let fb;
25
26   if(i % 15 == 0) {
27     fb = 'FizzBuzz'
28   }
29   else if(i % 3 == 0) {
30     fb = 'Fizz'
31   }
32   else if(i % 5 == 0) {
33     fb = 'Buzz'
34   }
35   else {
36     fb = `${i}`
37   }
38
39   res.send(fb)
40 }
```

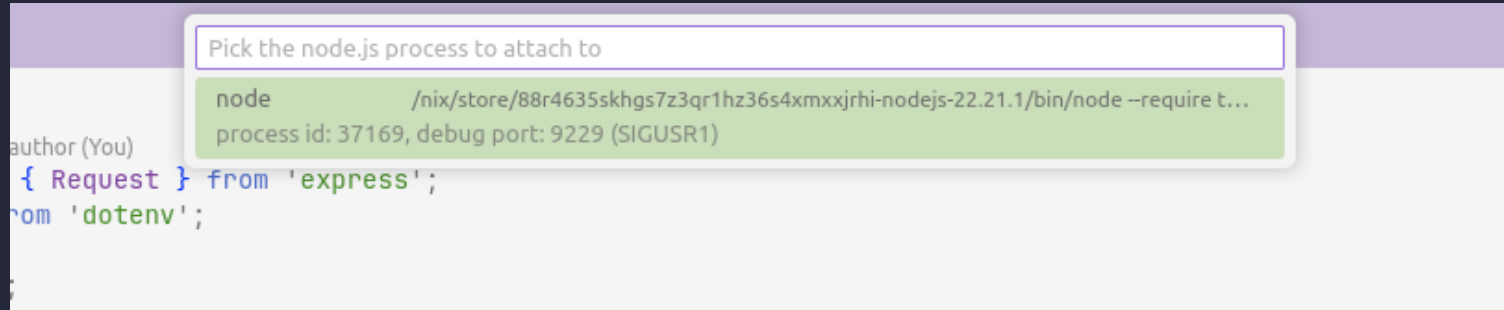
Click to add a breakpoint

You, last week · chore(2026): init

Démarrer le debugger



Sélectionner le serveur



Déclencher le point d'arrêt

The screenshot shows the Visual Studio Code interface with a debugger. The top bar indicates 'RUN AND DEBUG' and 'No Config...'. The main editor displays a TypeScript file named 'index.ts' with the following code:

```
3
4  dotenv.config();
5
6  const app = express();
7  const port = process.env.P
8
9  app.listen(port, () => {
10 |   console.log(`server: S
11 | });
12
13 app.get('/', (_, res) => {
14 |   res.send('Welcome');
15 | });
16
17 app.get('/hello/:name', (r
18 |   res.send('Hello ${req.pa
19 | });
20
21 app.get('/fizzbuzz/:i', (r
22 |   const i = req.params.i
23
24   let fb;
25
26   if(i % 15 == 0) {
27 |     fb = 'FizzBuzz'
28 |   }
29   else if(i % 3 == 0) {
30 |     fb = 'Fizz'
31 |   }
32   else if(i % 5 == 0) {
33 |     fb = 'Buzz'
34 |   }
35   else {
36 |     fb = `${i}`
37   }
38 }
```

The debugger is paused on a breakpoint at line 33, where the variable `fb` is assigned the value `'Buzz'`. The left sidebar shows the 'VARIABLES' panel with the following state:

- Local**
 - `fb` = undefined
 - `i` = '10'
 - `req` = IncomingMessage { _events: {...}, _r...
 - `res` = ServerResponse { _events: {...}, _ev...
 - `this` = undefined
- Closure**
 - `port` = '8090'
- Global**

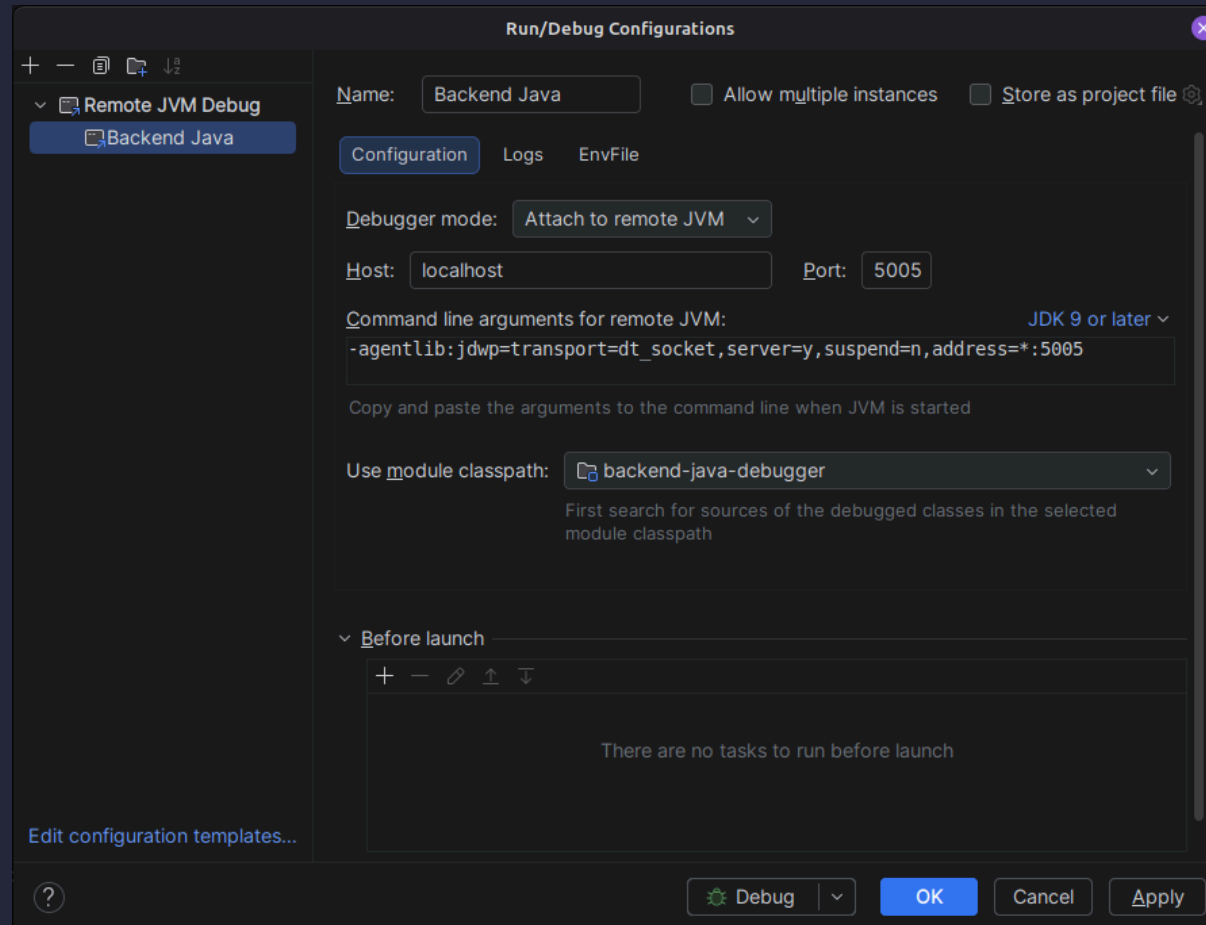
The 'WATCH' panel is empty. The 'CALL STACK' panel shows the current call stack, with the top frame being the current function call at line 33:7.

Avec Java

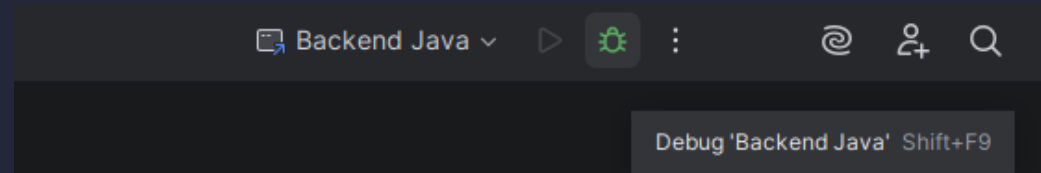
Ajouter l'agent au démarrage de l'application via l'option suivante :

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005
```


Configurer le debugger



Lancer le debugger



Déclencher le point d'arrêt

```
1 | .get( path: "/fizzbuzz/{i}", Context ctx → { ctx: io.javalin.http.servlet.JavalinServletContext@45594348
9 |     var value = Integer.parseInt(ctx.pathParam(s: "i")); value: 15
1 |
2 |     if (value % 15 == 0) { value: 15
3 |         ctx.result(resultString: "FizzBuzz"); ctx: io.javalin.http.servlet.JavalinServletContext@45594348
4 |     } else if (value % 3 == 0) {
5 |         ctx.result(resultString: "Buzz");
6 |     } else if (value % 5 == 0) {
7 |         ctx.result(resultString: "Fizz");
8 |     } else {
9 |         ctx.result(Integer.toString(value));
10 |     }
11 | })
12 | .start( port: 8080);
13 | }
14 | }
```

Debug Backend Java x

Threads & Variables Console

✓ "JettyServerThreadPool-32"@2,812 in group "main": RUNNING

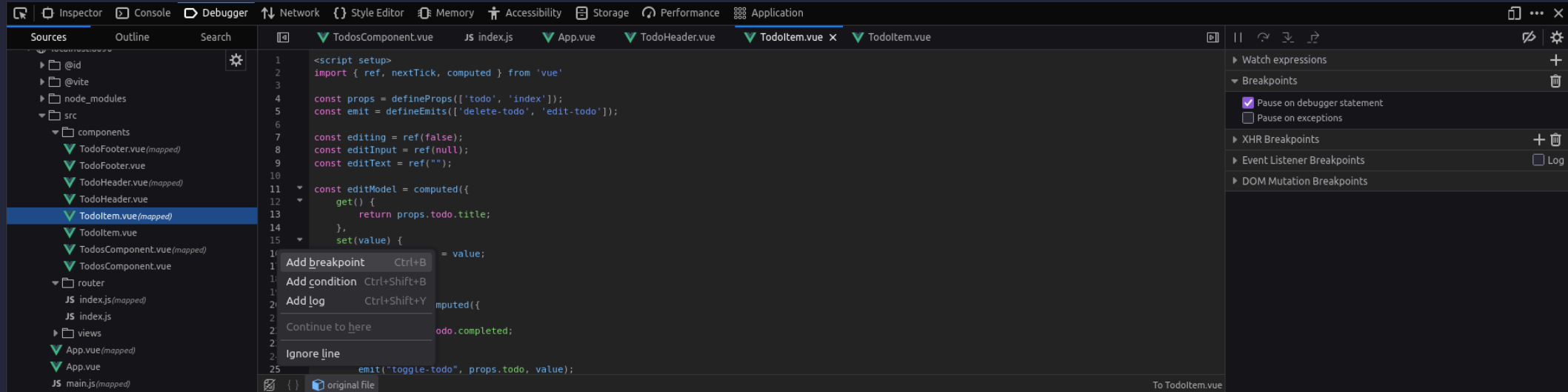
↳ lambda\$main\$2:12, App
40 hidden frames

> @ ctx = (JavalinServletContext@2814) io.javalin.http.servlet.JavalinServletContext@45594348
value = 15

Avec Javascript

Ouvrir les outils développeur : F12

Définir un point d'arrêt



Déclencher le point d'arrêt

The screenshot displays the Chrome DevTools interface with the 'Debugger' tab active. The 'Sources' panel on the left shows a file tree where 'TodoItem.vue' is selected. The main editor shows the code for 'TodoItem.vue', with a breakpoint (yellow bar) set on line 16, column 9, at the line `editText.value = value;`. The right-hand 'Debugger' panel shows the state of the application at the breakpoint:

- Paused on breakpoint:** set - TodoItem.vue:16:8
- Breakpoints:** A list of breakpoints is shown, with the current one for 'TodoItem.vue' at line 16:9 checked.
- Scopes:** The current scope is 'set', with the variable `value` having the value `"tes"`.

```
1 <script setup>
2 import { ref, nextTick, computed } from 'vue'
3
4 const props = defineProps(['todo', 'index']);
5 const emit = defineEmits(['delete-todo', 'edit-todo']);
6
7 const editing = ref(false);
8 const editInput = ref(null);
9 const editText = ref("");
10
11 const editModel = computed({
12   get() {
13     return props.todo.title;
14   },
15   set(value) { value: "tes"
16     editText.value = value;
17   },
18 });
19
20 const toggleModel = computed({
21   get() {
22     return props.todo.completed;
23   },
24   set(value) {
25     emit("toggle-todo", props.todo, value);
26   }
27 });
```

Merci de votre attention