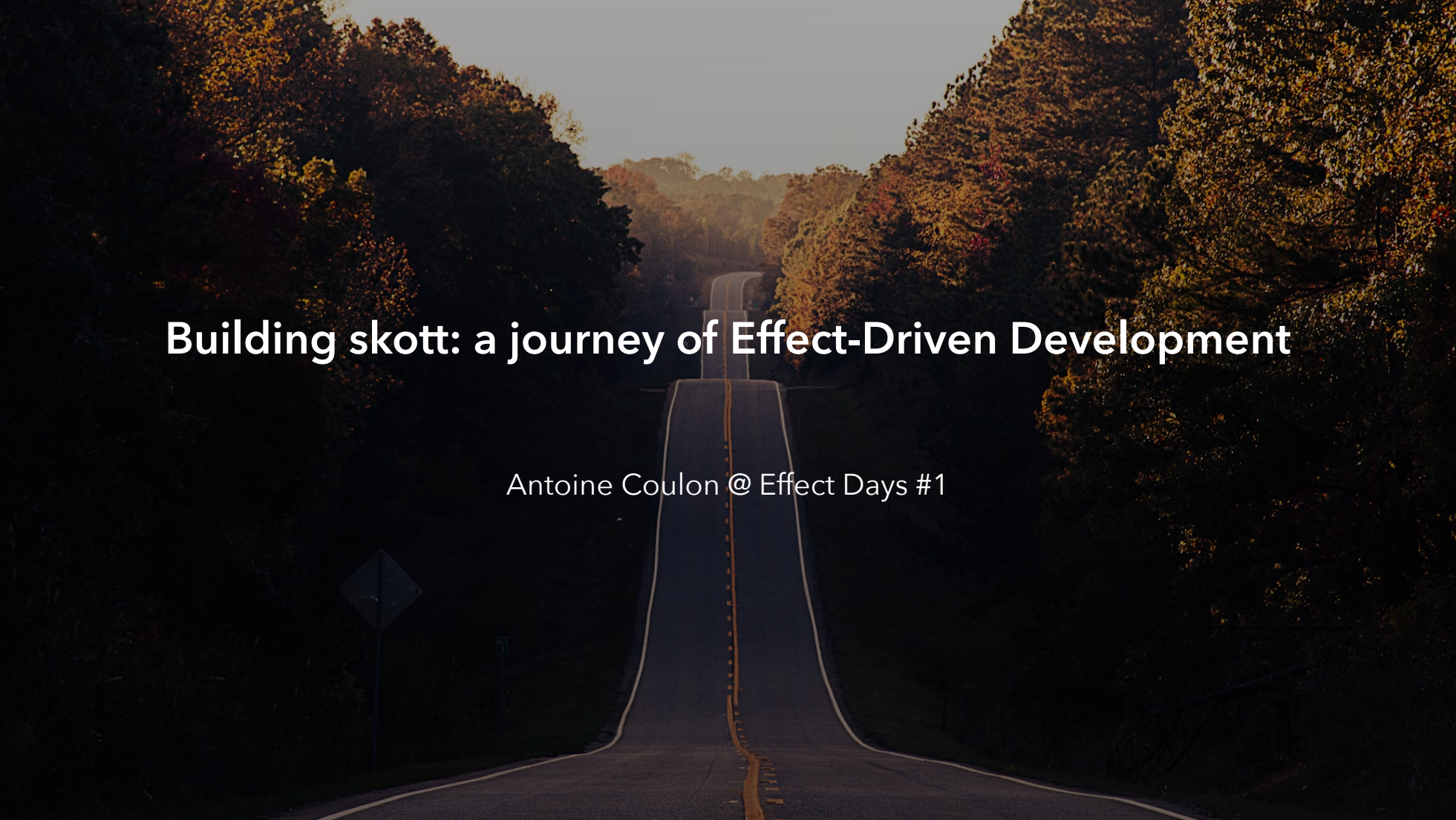


Building skott: a journey of Effect-Driven Development

Antoine Coulon @ Effect Days #1

Antoine Coulon

Lead Software Engineer @ **evryg**

Author of **skott**

Author of **effect-introduction**

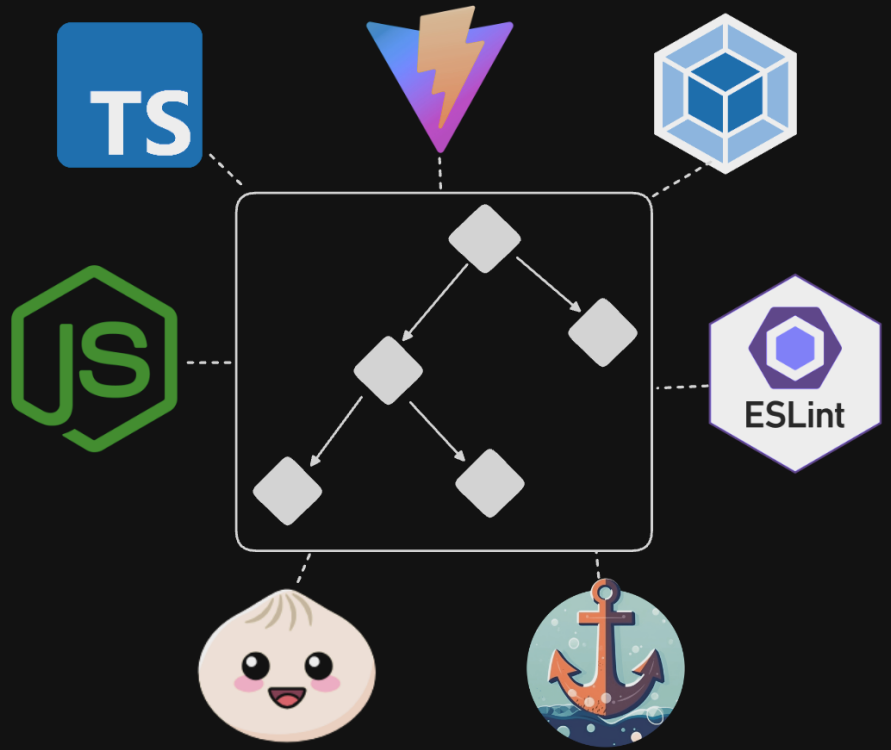
Contributing to **Rush.js, NodeSecure**

antoine-coulon

c9antoine

dev.to/antoinecoulon

The problem: graphs are everywhere, but hidden



skott is there to reveal graphs

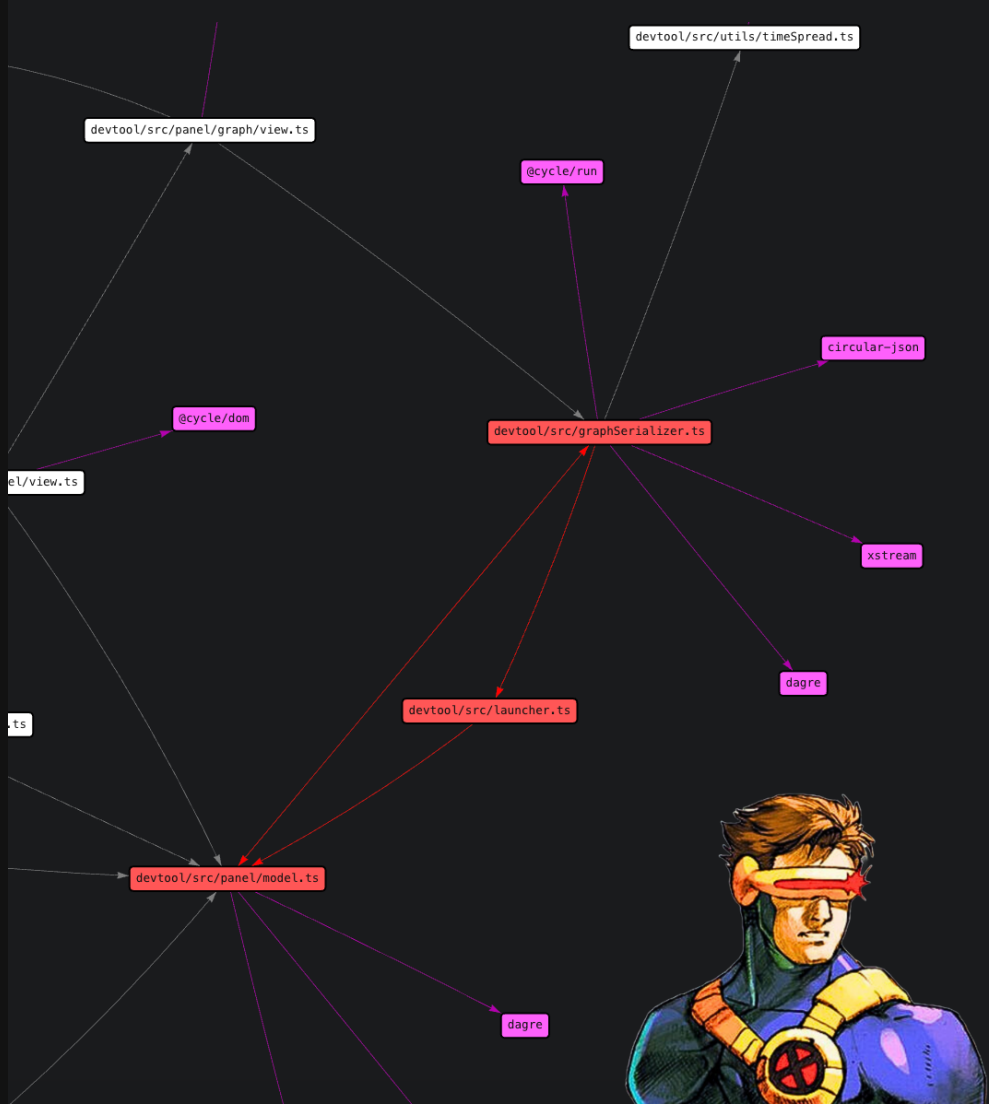
- Supports **JavaScript** + **TypeScript** projects

- Can be used via **CLI** or **JavaScript API**

- Written with **TypeScript**, running on **Node.js**

- Focused on providing great **developer experience**

- Lot of **display modes** out of the box to visualize graphs



skott needs what all softwares needs

skott's domain: mostly I/O

reading the file tree, module resolution, interactions with local cache

```
for await (const rootFile of this.fileReader.readdir(
      this.fileReader.getCurrentWorkingDir(),
      this.config.fileExtensions
    )) {
      const rootFileContent = await this.fileReader.read(rootFile);

      // [...]

      if (this.config.incremental) {
        this.#cacheHandler.addSourceFile(rootFile, rootFileContent);
      }

      await this.addNode(rootFile);
      await this.collectModuleDeclarations(rootFile, rootFileContent);
    }
```

Also common things

logging, control flows, error recovery, dependency injection, concurrency...

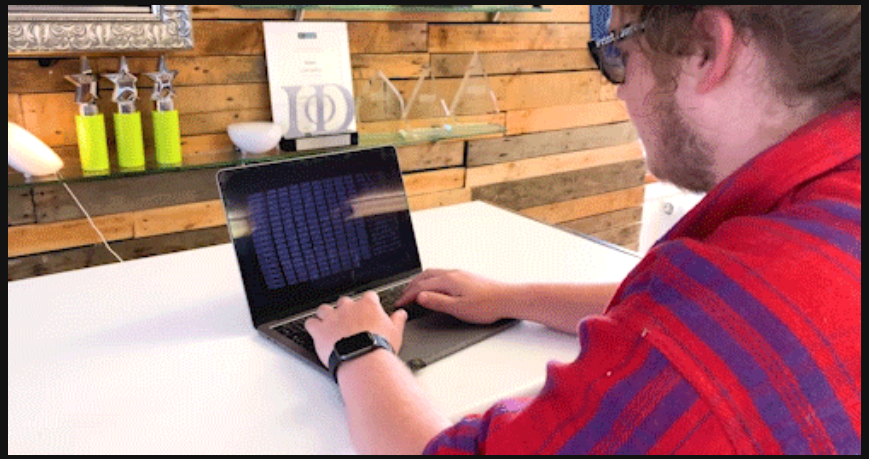
```
export function resolveImportedModulePath(
  module: string
): Effect.Effect<string, ModuleNotFoundError, FileReader | Logger> {
  return Logger.pipe(
    Effect.tap((logger) =>
      Effect.sync(() =>
        logger.info(`Start resolution for ${highlight(module)}`)
      )
    ),
    Effect.zip(isExistingModule(module)),
    Effect.flatMap(([logger, moduleExists]) => {
      if (isFileSupportedByDefault(module) && moduleExists) {
        return pipe(
          Effect.sync(() =>
            logger.success(`Succesfully resolved ${highlight(module)}`)
          ),
          Effect.map(() => module)
        );
      }
```

Developing reliable software is hard

When it comes to developing software, we can put in practice various disciplines

Test-Driven Development (TDD)

Type-Driven Development (TDD as well)



Introducing Effect-Driven Development

A streamlined and unified way of developing software

Effect enables both **Test** and **Type-Driven Development**

- **Dependency Inversion Principle** at heart

- Extreme **type-safety**

- Fast **feedback loop** overall

The trap: Third-Party-Driven Development

- 🧱 Composition between all dependencies is hard
- ⏳ Maintenance cost
- 📈 Learning curve
- 🔌 Compatibilities issues
- 🛡️ Higher risk of vulnerabilities

Select your difficulty level: Either<HardMode, EasyMode>

Option[HardMode]: npm install *universe*

😵‍💫 A fragmented ecosystem complexifying composability and maintenability

```
{
  "dependencies": {
-   "fp-ts": "*",
-   "async": "*",
-   "p-queue": "*",
-   "p-retry": "*",
-   "inversify": "*",
-   "ts-pattern": "*",
-   "ts-results": "*",
-   "neverthrow": "*",
-   "ramda": "*",
-   "winston": "*"
  }
```

Option[EasyMode]: npm install effect

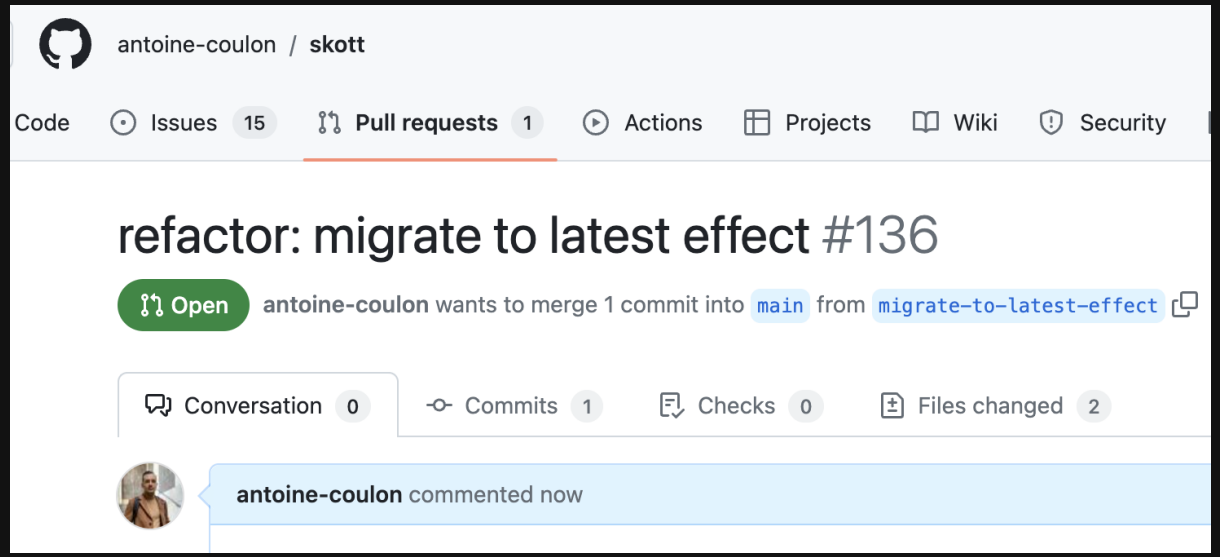
👊 A streamlined way of developing software with unified standard library and ecosystem

```
1   {
2     "dependencies": {
3   +   "effect": "latest"
4     }
```

Spoiler: I chose life on easy mode for skott

```ts
import { Effect, Either } from "effect";

declare const lifeChoice: Either.Either<HardMode, EasyMode>;
const easyModeOnly = lifeChoice.pipe(Effect.orDie, Effect.runSync);
```



Thanks for listening

- skott: https://github.com/antoine-coulon/skott
- Effect introduction: https://github.com/antoine-coulon/effect-introduction