

---

## Spot

**Spot** (*"Single Point Of Truth"*) is a concise, developer-friendly way to describe your API contract.

Leveraging the TypeScript syntax, it lets you describe your API and generate other API contract formats you need (OpenAPI, Swagger, JSON Schema).

You don't need to use TypeScript in your codebase to benefit from using Spot.

Example of an API definition file `api.ts` which defines a single **POST** endpoint to create a user:

```
1 import { api, endpoint, request, response, body } from "@airtasker/spot";
2
3 @api({
4   name: "My API"
5 })
6 class Api {}
7
8 @endpoint({
9   method: "POST",
10  path: "/users"
11 })
12 class CreateUser {
13   @request
14   request(@body body: CreateUserRequest) {}
15
16   @response({ status: 201 })
17   response(@body body: CreateUserResponse) {}
18 }
19
20 interface CreateUserRequest {
21   firstName: string;
22   lastName: string;
23 }
24
25 interface CreateUserResponse {
26   firstName: string;
27   lastName: string;
28   role: string;
29 }
```

## Getting Started

Get started with writing Spot contracts - Spot Guide

For all available syntax, see Spot Syntax

---

## Installation

With yarn installed and initialized add `@airtasker/spot` to your project:

```
1 yarn add @airtasker/spot
```

You can pass the definition above to a generator by simply running:

```
1 npx @airtasker/spot generate --contract api.ts
```

## Why we built Spot

At first glance, you may wonder why we bothered building Spot. Why not use OpenAPI (formerly known as Swagger) to describe your API?

At the core, we built Spot because we wanted a better developer experience.

## Writing contracts

OpenAPI documents are stored as YAML files, following a very specific schema. You won't know that you used the wrong field name or forgot to wrap a type definition into a schema object unless you run a good OpenAPI linter. Most developers who aren't intimately familiar with the OpenAPI specification end up using a visual editor such as Swagger Editor or Stoplight.

Since Spot leverages the TypeScript syntax, all you need is to write valid TypeScript code. Your editor will immediately tell you when your code is invalid. It will tell you what's missing, and you even get autocomplete for free. We could have picked any other typed language—TypeScript just happened to be one of the most concise and ubiquitous for us.

## Reviewing contracts

We believe that API contracts should be checked into Git, or whichever code versioning system you use. In addition, API contracts should be systematically peer reviewed. It's far too easy for a backend engineer to incorrectly assume what client engineers expect from an endpoint.

Because of their complex nested structure and the richness of the OpenAPI specification, OpenAPI documents can be difficult to review in a pull request. They're great for machines, but not always for humans.

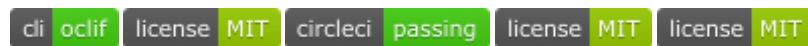
Spot aims to be as human-readable as possible. We've seen developers become a lot more engaged in discussions on pull requests for Spot contracts, compared to our previous OpenAPI documents.

---

## Interoperability with various formats

Depending on what you're trying to achieve (testing, documentation, client code generation...), you'll find tools that only work with OpenAPI 2 (Swagger), and newer tools that only support OpenAPI 3. You may also find tools for a different API ecosystem such as JSON Schema or API Blueprint.

We built Spot with this in mind. Instead of having to juggle various API format converters, Spot can generate every major API document format. This is why we called it "Single Point Of Truth".



- Spot
- Why we built Spot
- Usage
- Commands

## Usage

To get started and set up an API declaration in the current directory, run:

```
1 npx @airtasker/spot init
```

You can then run a generator with:

```
1 npx @airtasker/spot generate --contract api.ts
```

## In Memory Usage

```
1 import { Spot } from "@airtasker/spot";
2
3 const contract = Spot.parseContract("./api.ts")
4 const openApi = Spot.OpenApi3.generateOpenAPI3(contract);
5
6 console.log(openApi);
7
8 /*
9 {
10   openapi: '3.0.2',
11   info: { title: 'my-api', description: undefined, version: '0.0.0' },
12   paths: { '/users': { post: [Object] } },
13   components: {
14     schemas: { CreateUserRequest: [Object], CreateUserResponse: [Object] },
15     securitySchemes: undefined
```

---

```
16  },
17  security: undefined
18  }
19  */
```

## Commands

- `spot checksum SPOT_CONTRACT`
- `spot docs SPOT_CONTRACT`
- `spot generate`
- `spot help [COMMAND]`
- `spot init`
- `spot lint SPOT_CONTRACT`
- `spot mock SPOT_CONTRACT`
- `spot validate SPOT_CONTRACT`
- `spot validation-server SPOT_CONTRACT`

### **spot checksum SPOT\_CONTRACT**

Generate a checksum for a Spot contract

```
1  USAGE
2    $ spot checksum SPOT_CONTRACT
3
4  ARGUMENTS
5    SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8    -h, --help  show CLI help
9
10 EXAMPLE
11    $ spot checksum api.ts
```

See code: *build/cli/src/commands/checksum.js*

### **spot docs SPOT\_CONTRACT**

Preview Spot contract as OpenAPI3 documentation. The documentation server will start on `http://localhost:8080`.

```
1  USAGE
```

---

```
2  $ spot docs SPOT_CONTRACT
3
4  ARGUMENTS
5    SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8    -h, --help      show CLI help
9    -p, --port=port  [default: 8080] Documentation server port
10
11  EXAMPLE
12    $ spot docs api.ts
```

See code: *build/cli/src/commands/docs.js*

## spot generate

Runs a generator on an API. Used to produce client libraries, server boilerplates and well-known API contract formats such as OpenAPI.

```
1  USAGE
2    $ spot generate
3
4  OPTIONS
5    -c, --contract=contract  (required) Path to a TypeScript Contract
                             definition
6    -g, --generator=generator  Generator to run
7    -h, --help                show CLI help
8    -l, --language=language   Language to generate
9    -o, --out=out             Directory in which to output generated
                             files
10
11  EXAMPLE
12    $ spot generate --contract api.ts --language yaml --generator
    openapi3 --out output/
```

See code: *build/cli/src/commands/generate.js*

## spot help [COMMAND]

display help for spot

```
1  USAGE
2    $ spot help [COMMAND]
3
4  ARGUMENTS
5    COMMAND  command to show help for
6
```

---

```
7 OPTIONS
8   --all  see all commands in CLI
```

See code: [@oclif/plugin-help](#)

## spot init

Generates the boilerplate for an API.

```
1  USAGE
2    $ spot init
3
4  OPTIONS
5    -h, --help  show CLI help
6
7  EXAMPLE
8    $ spot init
9    Generated the following files:
10   - api.ts
11   - tsconfig.json
12   - package.json
```

See code: [build/cli/src/commands/init.js](#)

## spot lint SPOT\_CONTRACT

Lint a Spot contract

```
1  USAGE
2    $ spot lint SPOT_CONTRACT
3
4  ARGUMENTS
5    SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8    -h, --help  show
9                  CLI help
10   --has-discriminator=(error|warn|off)
11                  Setting for has-discriminator
12   --has-request-payload=(error|warn|off)
13                  Setting for has-request-payload
14   --has-response=(error|warn|off)
15                  Setting for has-response
16   --has-response-payload=(error|warn|off)
17                  Setting for has-response-payload
18   --no-inline-objects-within-unions=(error|warn|off)
19                  Setting for no-inline-objects-within-unions
```

---

```

14  --no-nullable-arrays=(error|warn|off)
    Setting for no-nullable-arrays
15  --no-nullable-fields-within-request-bodies=(error|warn|off)
    Setting for no-nullable-fields-within-request-bodies
16  --no-omittable-fields-within-response-bodies=(error|warn|off)
    Setting for no-omittable-fields-within-response-bodies
17  --no-trailing-forward-slash=(error|warn|off)
    Setting for no-trailing-forward-slash
18
19  EXAMPLES
20  $ spot lint api.ts
21  $ spot lint --has-discriminator=error
22  $ spot lint --no-nullable-arrays=off

```

See code: *build/cli/src/commands/lint.js*

## spot mock SPOT\_CONTRACT

Run a mock server based on a Spot contract

```

1  USAGE
2  $ spot mock SPOT_CONTRACT
3
4  ARGUMENTS
5  SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8  -h, --help                show CLI help
9  -p, --port=port          (required) [default:
    3010] Port on which to run the mock server
10 --pathPrefix=pathPrefix  Prefix to prepend to
    each endpoint path
11
12 --proxyBaseUrl=proxyBaseUrl  If set, the server will
    act as a proxy and fetch data from the given
13                             remote server instead of
    mocking it
14
15 --proxyFallbackBaseUrl=proxyFallbackBaseUrl  Like proxyBaseUrl,
    except used when the requested API does not match
16                                             defined SPOT contract.
    If unset, 404 will
    always be returned.
17
18 --proxyMockBaseUrl=proxyMockBaseUrl  Like proxyBaseUrl,
    except used to proxy draft endpoints instead of
19                                     returning mocked
    responses.
20

```

---

```
21 EXAMPLE
22 $ spot mock api.ts
```

See code: *build/cli/src/commands/mock.js*

## **spot validate SPOT\_CONTRACT**

Validate a Spot contract

```
1  USAGE
2    $ spot validate SPOT_CONTRACT
3
4  ARGUMENTS
5    SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8    -h, --help      show CLI help
9
10 EXAMPLE
11 $ spot validate api.ts
```

See code: *build/cli/src/commands/validate.js*

## **spot validation-server SPOT\_CONTRACT**

Start the spot contract validation server

```
1  USAGE
2    $ spot validation-server SPOT_CONTRACT
3
4  ARGUMENTS
5    SPOT_CONTRACT  path to Spot contract
6
7  OPTIONS
8    -h, --help          show CLI help
9    -p, --port=port     [default: 5907] The port where application will be
                           available
10
11 EXAMPLE
12 $ spot validation-server api.ts
```

See code: *build/cli/src/commands/validation-server.js*