aruva - empowering ideas Follow
Jan 24 · 5 min read · · Listen

Auto-Generate REST Integrations from Swagger in Camel

REST-based services are everywhere, and rightfully so. REST brings significant advantages to the world of Integration, from Simplicity using standard HTTP methods to being stateless, allowing better scalability and reliability

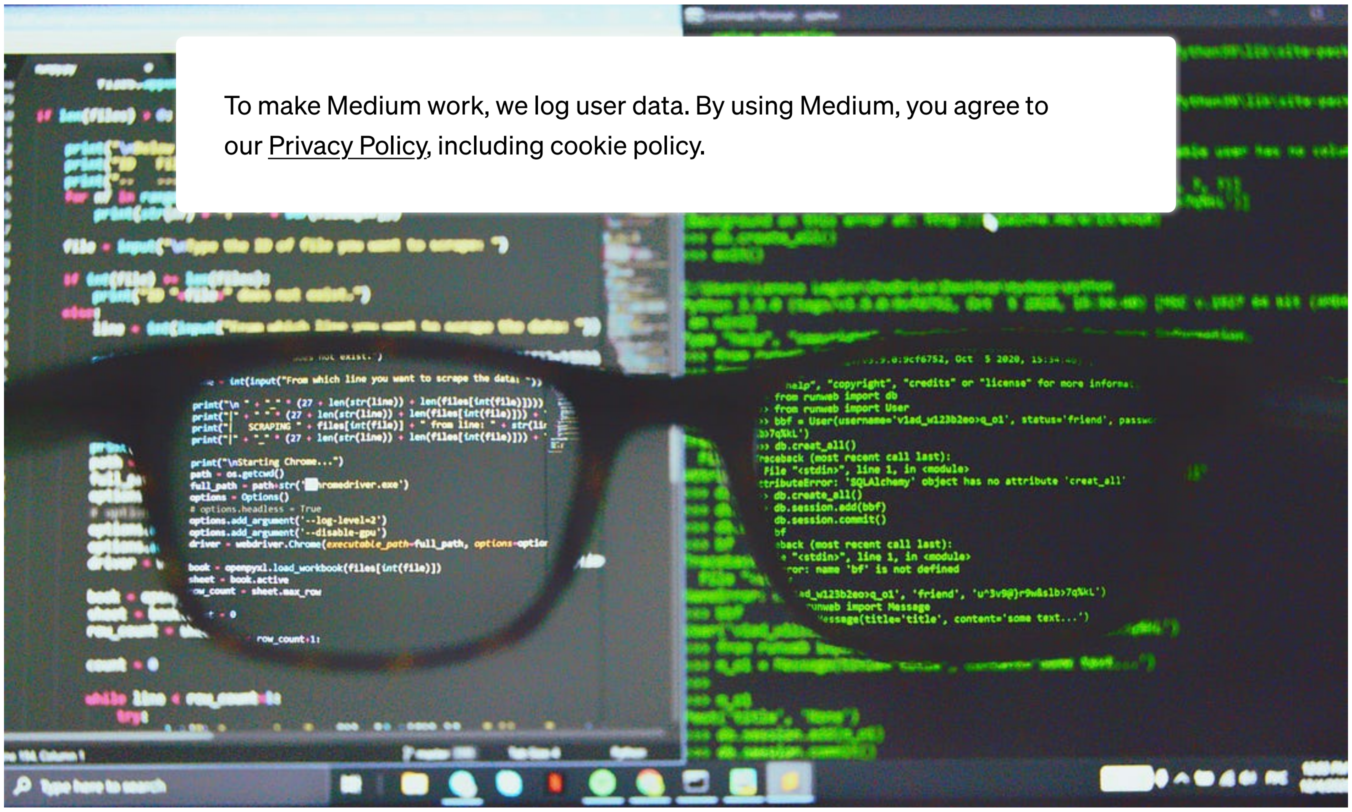


Photo by Alex Chumak on Unsplash

It makes it all the more important to properly document your REST APIs, which can be effectively communicated with developers utilizing these services. Swagger has been the go-to standard for documenting these APIs, and with its successor `Open API 3.x` specification, the documentation has become more flexible, predictable and less ambiguous overall

In this blog post, we will utilize an Open API 3.0 document (in JSON format) to generate and host Camel REST endpoint routes programmatically.

Let's get started

Pre-Requisites

Follow the Hello Camel blog post to set up the basic project. Alternatively, you can run the following maven command to get started with a project skeleton quickly.

```
mvn arc                                                              oes \ -Darc
```

Provide the required values, groupId, artifactId, and version. For example, I have used.

```
groupId: xyz.aruva
artifactId: rest-provider
version: 1.0-SNAPSHOT
```

Make sure the project is running by typing.

```
mvn spring-boot:run
```

Maven Configuration

Let's begin by adding the required `maven` dependencies to our project

```
<dependency>
  <groupId>org.openapi4j</groupId>
  <artifactId>openapi-parser</artifactId>
  <version>1.0.7</version>
</dependency>
<dependency>
  <groupId>io.swagger.core.v3</groupId>
  <artifactId>swagger-core</artifactId>
  <version>2.2.8</version>
</dependency>
<dependency>
  <groupId>com.google.code.gson</groupId>
```

```
    <artifactId>gson</artifactId>
  </deper
  <depenc
    <grou
    <arti
    <version>3.16.0</version>
  </dependency>
```

Here is a quick explanation of the above dependencies:
- ***openapi-parser***: required to parse OpenAPI3.0 specifications, also validates the file against the OpenAPI3.0 schema
- ***swagger-core***: provides the servlet that can be integrated with our spring-boot application. Also, it can generate the respective client/server code, if required
- ***gson***: Required to parse to/from JSON strings, including deserializing Java objects from JSON
- ***camel-servlet-starter***: Embeds the servlet within the application camel-context

Next, let's configure our `camel-restdsl-openapi-plugin`
Add it to the `<builds> -> <plugins>` section

```
  <plugin>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-restdsl-openapi-plugin</artifactId>
    <version>3.20.1</version>
    <executions>
      <execution>
        <id>generate-sources</id>
        <phase>generate-sources</phase>
        <goals>
          <goal>generate-with-dto</goal>
        </goals>
        <configuration>
          <!--suppress UnresolvedMavenProperty -->
          <specificationUri>${project.build.resources[0].directory}/apispec/apide
          <outputDirectory>${project.build.directory}/generated-sources/rest</out
          <packageName>xyz.aruva.routes</packageName>
          <modelOutput>${project.build.directory}/generated-sources/dto</modelOut
          <modelPackage>xyz.aruva.dto</modelPackage>
        </configuration>
      </execution>
    </executions>
  </plugi
```

This plugin is used to auto-generate routes for all endpoints defined in the OpenAPI specification, including request and response parameters, serialization, error handling and more.

The configurations for this plugin are explained below
- ***specificationURI***: Path to OpenAPI3.0 specification, which is our case, is `src/main/resources/apispec/apidefinition.json` file
- ***outputDirectory***: Path to generate the servlet controller and APIs
- ***packageName***: package name for the auto-generated java classes, which in our case is `xyz.aruva`
- ***modelOutput***: Path to generate DTOs, i.e. POJOs based on OpenAPI3.0 specification
- ***modelPackage***: package name for these generated POJO classes

Setting up OpenAPI Specification

Next, let's create the aforementioned `apispec` folder in `src/main/resources` and create an `apidefinition.json` file in the folder

Paste your OpenAPI3.0 specification in the file. I have provided a sample specification below

```
{
  "openapi": "3.0.1",
  "info": {
    "version": "1.0",
    "title": "User API"
  },
  "servers": [
    {
      "url": "/users"
    }
  ],
  "pat
    "/
      '
        "summary": "Retrieve a list of users",
        "operationId": "getUsers",
        "responses": {
          "200": {
            "description": "A list of users",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/User"
                  }
                }
              }
            }
          }
        }
      },
      "post": {
        "summary": "Create a new user",
        "operationId": "createUser",
        "requestBody": {
          "description": "The user to create",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/User"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "The created user",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/User"
                }
              }
            }
          }
```

```
            }
          }
        }
      },
      "comp
        "schemas": {
          "User": {
            "type": "object",
            "properties": {
              "id": {
                "type": "integer",
                "format": "int64"
              },
              "name": {
                "type": "string"
              },
              "email": {
                "type": "string"
              }
            }
          }
        }
      }
    }
```

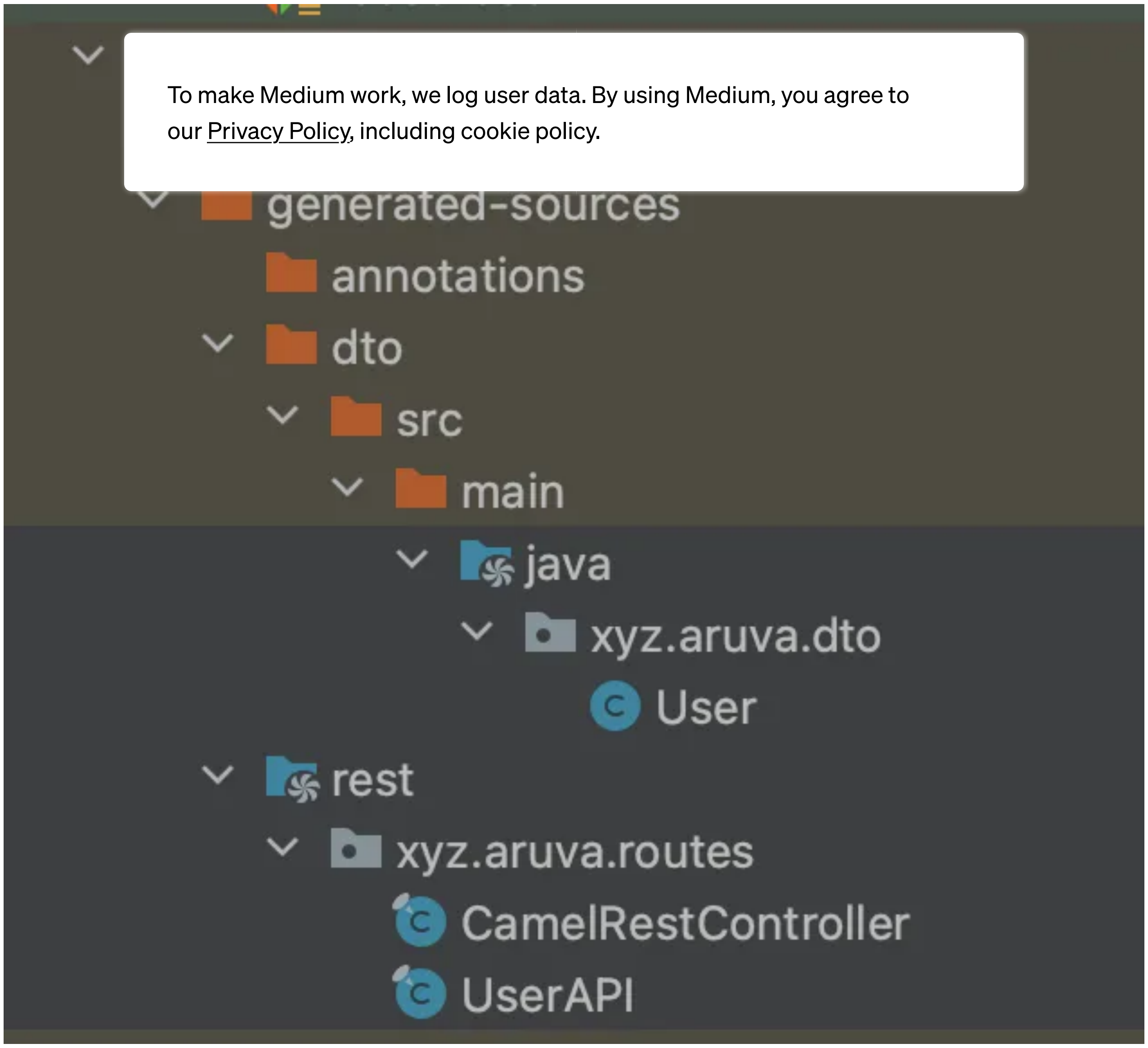
note: You can validate the correctness of the specification by opening this on editor.swagger.io

Let’s Compile

Next, go to the terminal on your machine or use an IDE to `clean install` the project

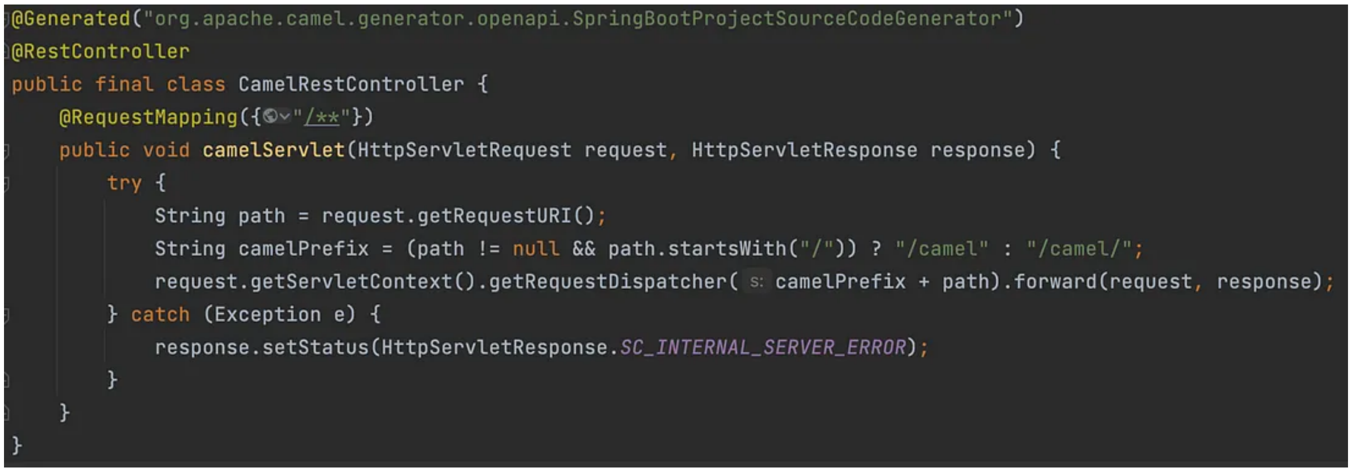
```
mvn clean install
```

Once complete, explore the `target` folder and you should see the generated sources available there


To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy.
generated-sources
annotations
dto
src
main
java
xyz.aruva.dto
User
rest
xyz.aruva.routes
CamelRestController
UserAPI

auto-generated sources

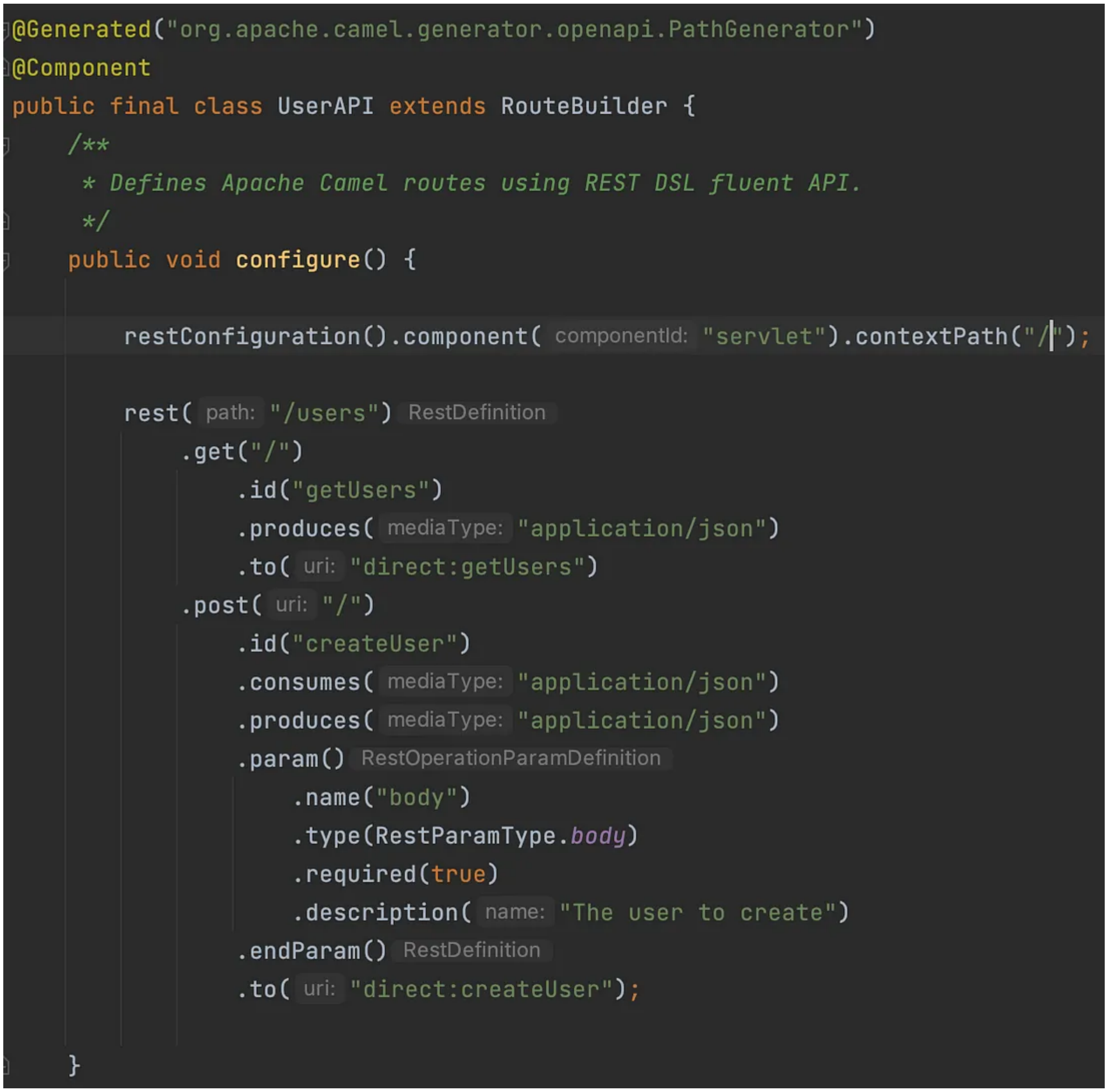
You should see a `CamelRestController` the class which provides the servlet definition.



```
@Generated("org.apache.camel.generator.openapi.SpringBootProjectSourceCodeGenerator")
@RestController
public final class CamelRestController {
    @RequestMapping({"/**"})
    public void camelServlet(HttpServletRequest request, HttpServletResponse response) {
        try {
            String path = request.getRequestURI();
            String camelPrefix = (path != null && path.startsWith("/")) ? "/camel" : "/camel/";
            request.getServletContext().getRequestDispatcher( s: camelPrefix + path).forward(request, response);
        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        }
    }
}
```

This class creates a dispatcher servlet which intercepts all incoming requests and then forwa

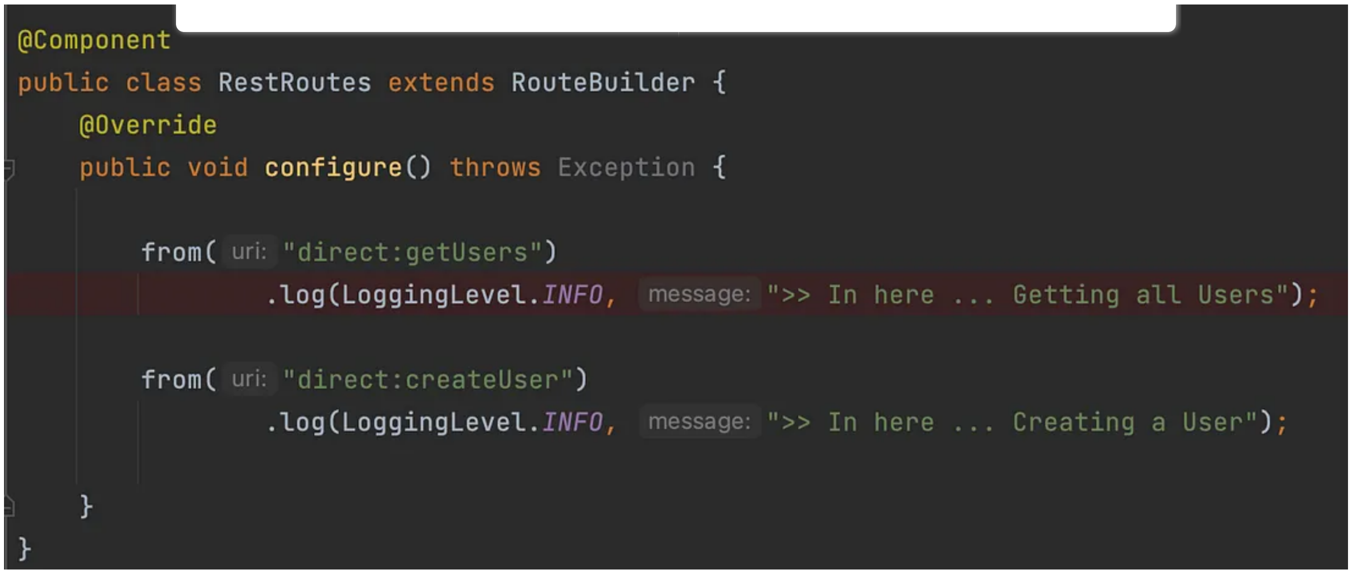
A `UserAPI`



```
@Generated("org.apache.camel.generator.openapi.PathGenerator")
@Component
public final class UserAPI extends RouteBuilder {
    /**
     * Defines Apache Camel routes using REST DSL fluent API.
     */
    public void configure() {

        restConfiguration().component( componentId: "servlet").contextPath("/");

        rest( path: "/users") RestDefinition
            .get("/")
                .id("getUsers")
                .produces( mediaType: "application/json")
                .to( uri: "direct:getUsers")
            .post( uri: "/")
                .id("createUser")
                .consumes( mediaType: "application/json")
                .produces( mediaType: "application/json")
                .param() RestOperationParamDefinition
                    .name("body")
                    .type(RestParamType.body)
                    .required(true)
                    .description( name: "The user to create")
                .endParam() RestDefinition
                .to( uri: "direct:createUser");

    }
```

This class builds the REST routes, which can be invoked by the dispatcher servlet above, and then forward them to `direct` endpoints (one for every exposed API endpoint)

That's great. Now, all we need to do is provide our implementation of these `direct` endpoints



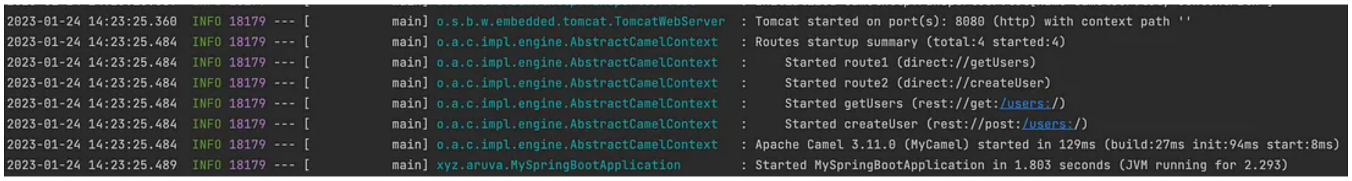
```
@Component
public class RestRoutes extends RouteBuilder {
    @Override
    public void configure() throws Exception {

        from( uri: "direct:getUsers")
                .log(LoggingLevel.INFO,  message: ">> In here ... Getting all Users");

        from( uri: "direct:createUser")
                .log(LoggingLevel.INFO,  message: ">> In here ... Creating a User");

    }
}
```

note: this class will be in `src/main/java` i.e. `soource` directory and not `target`

Let's test it out. Start the spring-boot application from the IDE or using

```
mvn springboot:run
```

and on start, you should see the logs indicating the routes loaded and started in the context

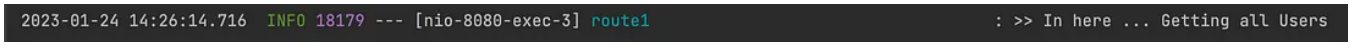


```
2023-01-24 14:23:25.360  INFO 18179 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   : Routes startup summary (total:4 started:4)
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   :     Started route1 (direct://getUsers)
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   :     Started route2 (direct://createUser)
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   :     Started getUsers (rest://get:/users:/)
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   :     Started createUser (rest://post:/users:/)
2023-01-24 14:23:25.484  INFO 18179 --- [           main] o.a.c.impl.engine.AbstractCamelContext   : Apache Camel 3.11.0 (MyCamel) started in 129ms (build:27ms init:94ms start:8ms)
2023-01-24 14:23:25.489  INFO 18179 --- [           main] xyz.aruva.MySpringBootApplication        : Started MySpringBootApplication in 1.803 seconds (JVM running for 2.293)
```

You can now use `postman` or just regular `curl` commands to validate the invocation of these `direct` routes

```
curl --
```

and in the logs

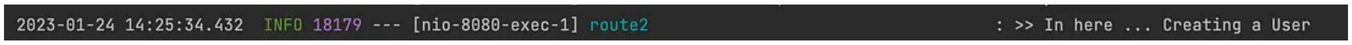


```
2023-01-24 14:26:14.716  INFO 18179 --- [nio-8080-exec-3] route1                                   : >> In here ... Getting all Users
```

and for POST call

```
curl --location --request POST 'http://localhost:8080/users' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--data-raw '{
    "id": 1,
    "name": "Aruva",
    "email": "contact@aruva.xyz"
```

API Programming Java Apache Camel Spring Boot



```
2023-01-24 14:25:34.432  INFO 18179 --- [nio-8080-exec-1] route2                                   : >> In here ... Creating a User
```

To update these endpoints, upda 15 **and regenerate the classes using**

```
mvn clean install
```

Enjoy the read? Reward the writer. Beta

Your tip will go to aruva - empowering ideas through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip s for reading

Please subscribe here, never miss our new posts, and don't forget to clap if you enjoyed reading this post.

Get an email whenever aruva - empowering ideas publishes.