

JHipster Microservice

- JHipster Registry
- Create Microservice
 - Microservice: Job App
 - Microservice: Gateway App
- Security
 - JWT (JSON Web Token)
- Access Control
 - Microservice - Job App API
 - Microservice - Gateway App API

JHipster Registry

The JHipster Registry is a runtime application, provided by the JHipster team. Like the JHipster generator, it is an Open Source, Apache 2-licensed application, and its source code is available on GitHub under the JHipster organization at [jhipster/jhipster-registry](https://github.com/jhipster/jhipster-registry).

JHipster registry is a management tool for monitoring microservice.

```
git clone https://github.com/jhipster/jhipster-registry.git
```

The JHipster Registry can be cloned/forked/downloaded directly from [jhipster/jhipster-registry](https://github.com/jhipster/jhipster-registry), and we recommend you use the same version tag as the one you use for your JHipster generator. As the JHipster Registry is also a JHipster-generated application, you can run it like any other JHipster application:

- run it with `./mvnw` in development, it will use by default the dev profile and the Eureka Registry will be available at <http://127.0.0.1:8761/>.
- use `./mvnw -Pprod` package to package it in production, and generate the usual JHipster executable WAR file.

- JHipster registry administrative page. Note that we have GATEWAYAPP and JOBAPP running.

When the gateways and the microservices are launched, they will register themselves in the registry (using the `eureka.client.serviceUrl.defaultZone` key in the `src/main/resources/config/application.yml` file).

JHipster Registry v2.5.5

You are logged in as user "admin".

System Status

Environment	test
Data Center	default
Current Time	2016-12-10T09:32:18 -0800
System Uptime	00:04
Below Renew Threshold	false

General Info

Total Available Memory	693mb
Current Memory Usage	397mb (57%)
Number of CPU	8
Instance Ip Address	192.168.1.69
Instance Status	UP

Instances Registered

App	Instance ID	Status
GATEWAYAPP	gatewayapp:3abe3b65f9ed7233303fcccfe4b477cf	UP
JOBAPP	jobapp:90ab344a23d733f308a7b2a53b103680	UP

Health

ConfigServer	UP
DiscoveryComposite	UP
DiscoveryComposite - discoveryClient	UP
DiscoveryComposite - eureka	UNKNOWN
DiskSpace	UP
RefreshScope	UP
Hystrix	UP

Create Microservice

Please refer [jHipster Instructions](#) for jHipster basics and installation.

Microservice: Job App

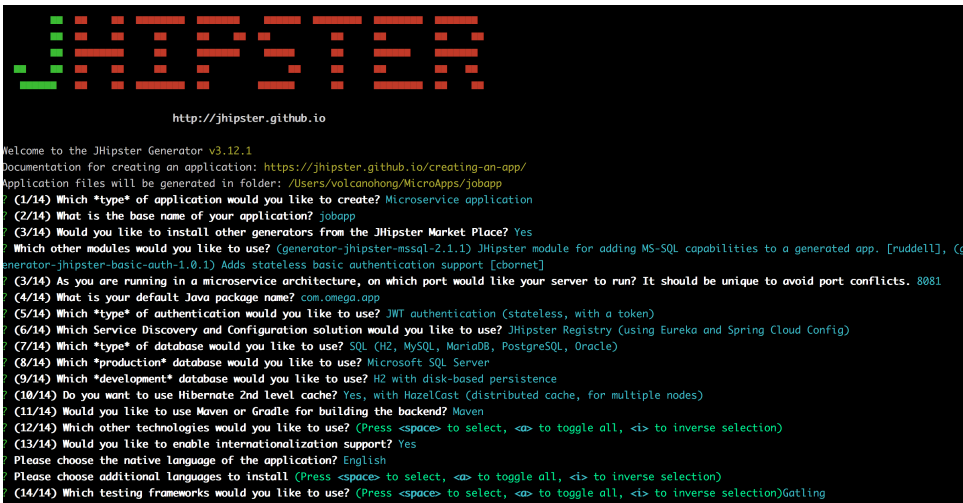
At least, we need install 'yeoman'.

```
npm install -g yo
```

Then, install JHipster generator, note version is '**generator-jhipster@3.12.1**' currently.

```
npm install -g generator-jhipster
```

Then, run `yo jhipster` to create an application. The app configuration is listed below. Note that we must choose **microservice application**.



```
JHIPSTER
http://jhipster.github.io

Welcome to the JHipster Generator v3.12.1
Documentation for creating an application: https://jhipster.github.io/creating-an-app/
Application files will be generated in folder: /Users/yolcanohong/MicroApps/jobapp
(1/14) Which *type* of application would you like to create? Microservice application
(2/14) What is the base name of your application? jobapp
(3/14) Would you like to install other generators from the JHipster Market Place? Yes
Which other modules would you like to use? (generator-jhipster-mysql-2.1.1) JHipster module for adding MS-SQL capabilities to a generated app. [ruddell], (g
enerator-jhipster-basic-auth-1.0.1) Adds stateless basic authentication support [chornet]
(3/14) As you are running in a microservice architecture, on which port would like your server to run? It should be unique to avoid port conflicts. 8081
(4/14) What is your default Java package name? com.omega.app
(5/14) Which *type* of authentication would you like to use? JWT authentication (Stateless, with a token)
(6/14) Which Service Discovery and Configuration solution would you like to use? JHipster Registry (using Eureka and Spring Cloud Config)
(7/14) Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle)
(8/14) Which *production* database would you like to use? Microsoft SQL Server
(9/14) Which *development* database would you like to use? H2 with disk-based persistence
(10/14) Do you want to use Hibernate 2nd level cache? Yes, with HazelCast (distributed cache, for multiple nodes)
(11/14) Would you like to use Maven or Gradle for building the backend? Maven
(12/14) Which other technologies would you like to use? (Press <space> to select, <=> to toggle all, <i> to inverse selection)
(13/14) Would you like to enable internationalization support? Yes
Please choose the native language of the application? English
Please choose additional languages to install (Press <space> to select, <=> to toggle all, <i> to inverse selection)
(14/14) Which testing frameworks would you like to use? (Press <space> to select, <=> to toggle all, <i> to inverse selection)Gatling
```

A hidden file in the root directory called **`.yo-rc.json`** contains the detail app configuration.

Create Entity

Using json file in **`.jhipster`** directory.

For example, create a **`.dwPaymentSummary`** entity. Note that we must have the **microservice configuration**: "microserviceName": "jobapp".

```

{
  "fluentMethods": true,
  "relationships": [],
  "fields": [
    {
      "fieldName": "userId",
      "fieldType": "Long",
      "fieldValidateRules": [
        "required"
      ]
    },
    {
      "fieldName": "partyId",
      "fieldType": "Long"
    }
  ],
  "changelogDate": "20161209225809",
  "dto": "no",
  "service": "no",
  "entityTableName": "dw_payment_summary",
  "pagination": "pagination",
  "microserviceName": "jobapp",
  "searchEngine": false
}

```

Microservice: Gateway App

JHipster allows use to generate an API gateway. This gateway is a normal JHipster application, so you can use the usual JHipster options and development workflows on that project, but it also acts as the entrance to your microservices. More specifically, it provides HTTP routing and load balancing, quality of service, security and API documentation for all microservices.

Install `yeoman` and `jhipster`.

Then, run `yo jhipster`. Note that we must choose **microservice gateway**.

The additional modules that I selected :

1. A Jhipster based generator to create Angular material + spring boot application [deepu8224],
2. (generator-jhipster-mssql-2.1.1) JHipster module for adding MS-SQL capabilities to a generated app. [ruddell],
3. (generator-jhipster-basic-auth-1.0.1) Adds stateless basic authentication support [cbornet],
4. (generator-jhipster-angular-datatables-1.2.0) JHipster module to change default grid with Angular DataTables [hermeslm]

All detailed configuration shows below.

```
JHIPSTER

http://jhipster.github.io

Welcome to the JHipster Generator v3.12.1
Documentation for creating an application: https://jhipster.github.io/creating-an-app/
Application files will be generated in folder: /Users/volcanohong/MicroApps/gatewayapp
? (1/14) Which *type* of application would you like to create? Microservice gateway
? (2/14) What is the base name of your application? gatewayapp
? (3/14) Would you like to install other generators from the JHipster Market Place? Yes
? Which other modules would you like to use? (generator-jhipster-material-0.0.0) A Jhipster based generator to create Angular material + spring boot applicati
on [dangb224], (generator-jhipster-mysql-2.1.1) Jhipster module for adding MS-SQL capabilities to a generated app. [ruddell], (generator-jhipster-basic-auth-
1.0.13) Adds stateless basic authentication support [cbornet], (generator-jhipster-angular-datatables-1.2.0) Jhipster module to change default grid with Angula
r DataTables [hermes1n]
? (3/14) As you are running in a microservice architecture, on which port would like your server to run? It should be unique to avoid port conflicts. 8080
? (4/14) What is your default Java package name? com.omega.app
? (5/14) Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? (6/14) Which Service Discovery and Configuration solution would you like to use? JHipster Registry (using Eureka and Spring Cloud Config)
? (7/14) Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? (8/14) Which *production* database would you like to use? Microsoft SQL Server
? (9/14) Which *development* database would you like to use? H2 with disk-based persistence
? (10/14) Do you want to use Hibernate 2nd level cache? Yes, with HazelCast (distributed cache, for multiple nodes)
? (11/14) Would you like to use Maven or Grate for building the backend? Maven
? (12/14) Which other technologies would you like to use? Search engine using Elasticsearch
? (13/14) Would you like to use the libSass stylesheet preprocessor for your CSS? No
? (14/14) Would you like to enable internationalization support? Yes
Please choose the native language of the application? English
Please choose additional languages to install (Chinese (Simplified), Dutch, French, German, Turkish)
? (15/14) Which testing frameworks would you like to use? (Press <space> to select, <a> to toggle all, <i> to inverse selection)Gatling
```

The gateway is a front end UI application. It takes longer time to create than the previous app.

Create Entity:

Run command

```
yo jhipster:entity dwPaymentSummary
```

We need to generate the entity from an existing microservice and specify the root directory. Then, choose **`re-generate the entity`**.

```
Hongs-MacBook-Pro:gatewayapp volcanohong$ yo jhipster:entity dwPaymentSummary

The entity dwPaymentSummary is being created.

? Do you want to generate this entity from an existing microservice? Yes
? Enter the path to the microservice root directory: ../jobapp

Found the .jhipster/DwPaymentSummary.json configuration file, entity can be automatically generated!

? Do you want to update the entity? This will replace the existing files for this entity, all your custom code will be overwritten Yes, re generate the entity

create .jhipster/DwPaymentSummary.json
create src/main/webapp/i18n/en/transactionType.json
create src/main/webapp/i18n/zh-cn/transactionType.json
create src/main/webapp/i18n/fr/transactionType.json
create src/main/webapp/i18n/de/transactionType.json
create src/main/webapp/i18n/tr/transactionType.json
create src/main/webapp/app/entities/dw-payment-summary/dw-payment-summary.html
create src/main/webapp/app/entities/dw-payment-summary/dw-payment-summary-detail.html
create src/main/webapp/app/entities/dw-payment-summary/dw-payment-summary-dialog.html
create src/main/webapp/app/entities/dw-payment-summary/dw-payment-summary-delete-dialog.html
conflict src/main/webapp/app/layouts/navbar/navbar.html
? Overwrite src/main/webapp/app/layouts/navbar/navbar.html? overwrite
force src/main/webapp/app/layouts/navbar/navbar.html
```

Security

make sure both apps have same security token in the ``application-dev.yml`` and ``application-prod.yml``

```
security:
  authentication:
    jwt:
      secret: 7bd82039d5735345abfbb4fff39555f785b1460b
      # Token is valid 24 hours
      tokenValidityInSeconds: 86400
```

JWT (JSON Web Token)

JWT (JSON Web Token) is an industry standard, easy-to-use method for securing applications in a microservices architecture.

JHipster uses the [JJWT library](#), provided by Stormpath, for implementing JWT.

Tokens are generated by the gateway, and sent to the underlying microservices: as they share a common secret key, microservices are able to validate the token, and authenticate users using that token.

Those tokens are self-sufficient: they have both authentication and authorization information, so microservices do not need to query a database or an external system. This is important in order to ensure a scalable architecture.

For security to work, a JWT secret token must be shared between all applications.

- For each application the default token is unique, and generated by JHipster. It is stored in the `.yo-rc.json` file.
- Tokens are configured with the `jhipster.security.authentication.jwt.secret` key in the `src/main/resources/config/application.yml` file.
- To share this key between all your applications, copy the key from your gateway to all the microservices, or share it using the JHipster Registry's Spring Config Server.
- A good practice is to have a different key in development and production.

Access Control

In this case, we will skip the authentication and make api open to public. In summary, we may have two ways to access an api either by microservice itself or by the gateway.

Microservice - Job App API

The gateway will automatically proxy all requests to the microservices, using their application name: for example, when microservices `app1` is registered, it is available on the gateway on the `/app1` URL.

For example, if your gateway is running on `localhost:8080`, you could point to <http://localhost:8080/app1/rest/foos> to get the `foos` resource served by microservice `app1`. If you're trying to do this with your Web browser, don't forget REST resources are secured by default in JHipster, so you need to send the correct JWT header (see the point on security below), or remove the security on those URLs in the microservice's `MicroserviceSecurityConfiguration` class.

Based on the above description, we can manually make any api open to public usage without JWT authentication. We can achieve this by changing the **MicroserviceSecurityConfiguration.java** file.

```

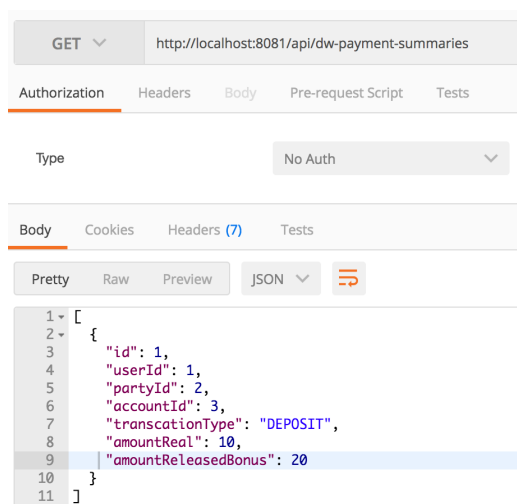
public class MicroserviceSecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Inject
    private TokenProvider tokenProvider;

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring()
            .antMatchers(HttpMethod.OPTIONS, "/*")
            .antMatchers("/app/**/*.js,html")
            .antMatchers("/bower_components/**")
            .antMatchers("/i18n/**")
            .antMatchers("/content/**")
            .antMatchers("/swagger-ui/index.html")
            .antMatchers("/test/**")
            .antMatchers("/h2-console/**")
            .antMatchers("/api/**");
    }
}

```

Test with Postman:



Microservice - Gateway App API

By default all registered microservices are available through the gateway. If you want to exclude a specific API from being exposed through the gateway, you can use the gateway's specific access control policy filter. It is configurable using the `jhipster.gateway.authorized-microservices-endpoints` key in the `application-*.yaml` files:

```

jhipster:
  gateway:
    authorized-microservices-endpoints: # Access Control Policy, if left empty for a route, all
    endpoints will be accessible
    appl: /api,/v2/api-docs # recommended dev configuration

```

For example, if we only want the `/api/dw-payment-summaries` endpoint of microservice `jobapp` to be available:

```

jhipster:
  gateway:
    authorized-microservices-endpoints:
      jobapp: /api/dw-payment-summaries

```

Test with Postman:

GET

http://localhost:8080/jobapp/api/dw-payment-summaries

Authorization

Headers

Body

Pre-request Script

Tests

Type

No Auth

Body

Cookies

Headers (12)

Tests

Pretty

Raw

Preview

JSON

1

2

3

4

5

6

7

8

9

10

11

[

{

"id": 1,

"userId": 1,

"partyId": 2,

"accountId": 3,

"transcationType": "DEPOSIT",

"amountReal": 10,

"amountReleasedBonus": 20

}

]