**Web Service – W3C Definition**

Software system designed to support interoperable machine-to-machine interaction over a network

**3 KEYS**

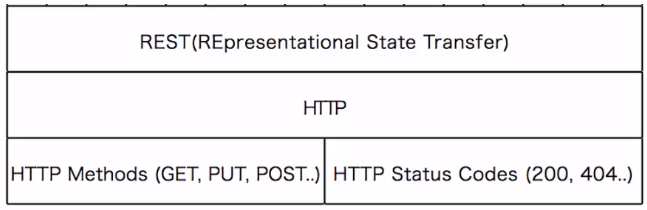
* Designed for machine-to-machine (or application-to-application) interaction
* Should be interoperable – Not platform dependent
* Should allow communication over a network

**Key Terminology**

* Request and Response
* Message Exchange Format
  + XML and JSON
* Service Provider or Server
* Service Consumer or Client
* Service Definition : The contract between Service Provider and Service Consumer
  + Request/Response Format
  + Request Structure
  + Response Structure
  + Endpoint
* Transport
  + HTTP and MQ

**WEB SERVICE GROUPS**

* **SOAP-based :** Simple Object Access Protocol
  + Format
    - SOAP XML Request
    - SOAP XML Response
  + Transport
    - SOAP over MQ
    - SOAP over HTTP
  + Service Definition
    - WSDL : Web Service Definition Language
* **REST-styled** : Representational State Transfer
  + Make Best User of HTTP



* + Key Abstraction – Resource
    - A resource has an URI (Uniform Resource Identifier)
      * /user/galih/todos/1
      * /user/galih/todos
      * /user/galih
    - A resource can have different representations
      * XML
      * HTML
      * JSON
  + Data Exchange Format
    - No Restriction. JSON is popular
  + Transport
    - Only HTTP
  + Service Definition
    - No Standard. WADL/Swagger/…

**REST VS SOAP**

* Restrictions vs Architectural Approach
* Data Exchange Format
* Service Definition
* Transport
* Ease of implementation

===============================================================================================

Generate project dari <https://start.spring.io/> dengan config sbb:

**Project :** Maven Project

**Language :** Java

**Spring Boot :** 2.1.7 ( Agustus 2019)

**Group :** gaw

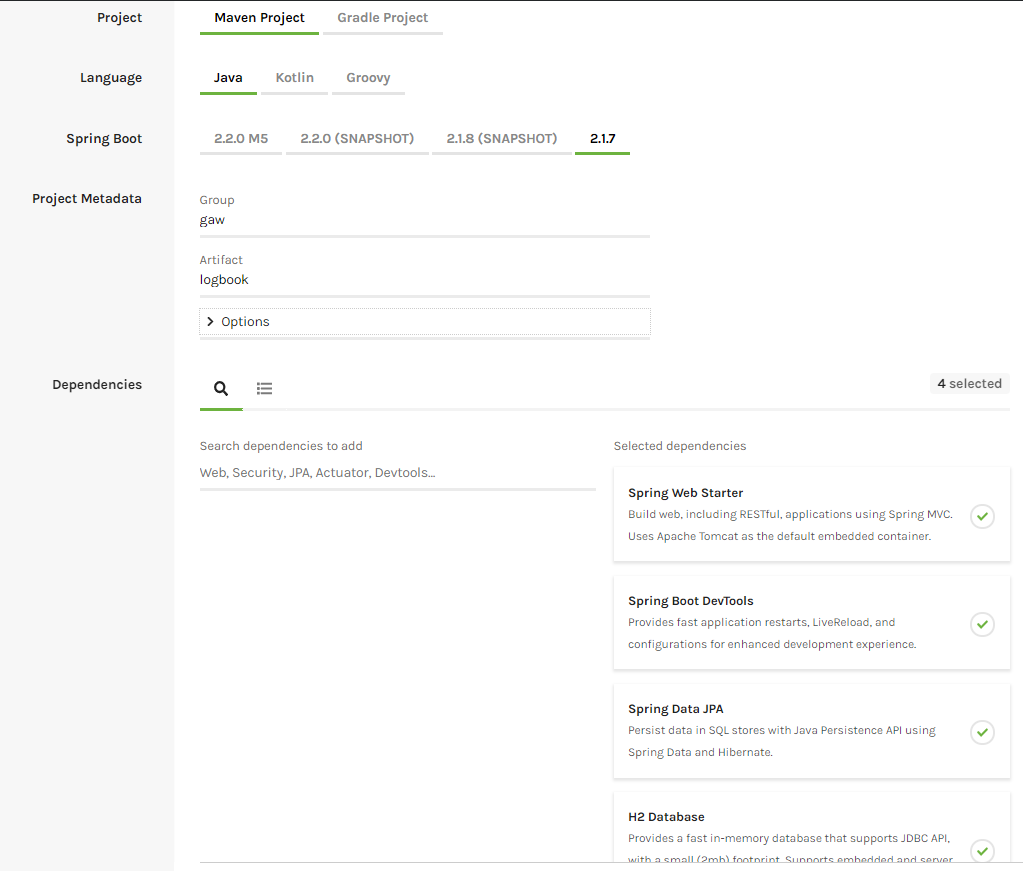
**Artifact :** logbook

**Options :** bebas

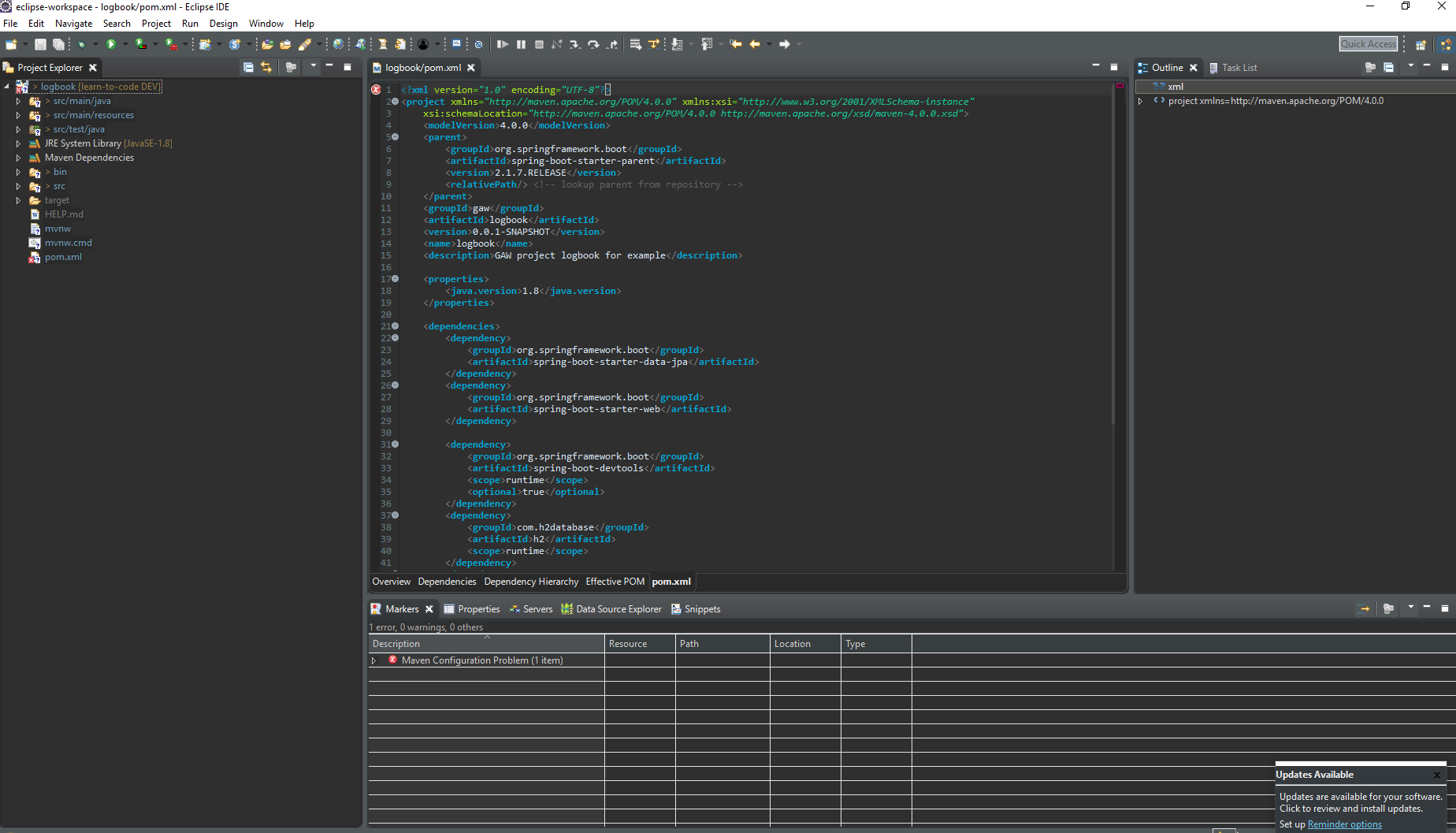
**Dependencies :**

* Web (Spring Web Starter)
* Devtools (Spring Boot Devtools)
* JPA (Spring Data JPA)
* H2 (H2 Database)

Generate the project, next import ke eclipse



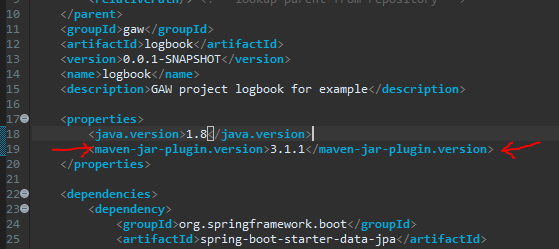
Import project ke eclipse (File > Open Projects from File Systems…)



Masih ada warning (Maven Configuration Problem), ada bug di eclipse. Fix it by downgrade maven jar plugin version dari 3.1.2 ke 3.1.1. dengan menambahkan ke properties section di pom.xml

<**maven-jar-plugin.version**>3.1.1</**maven-jar-plugin.version**>

Sehingga



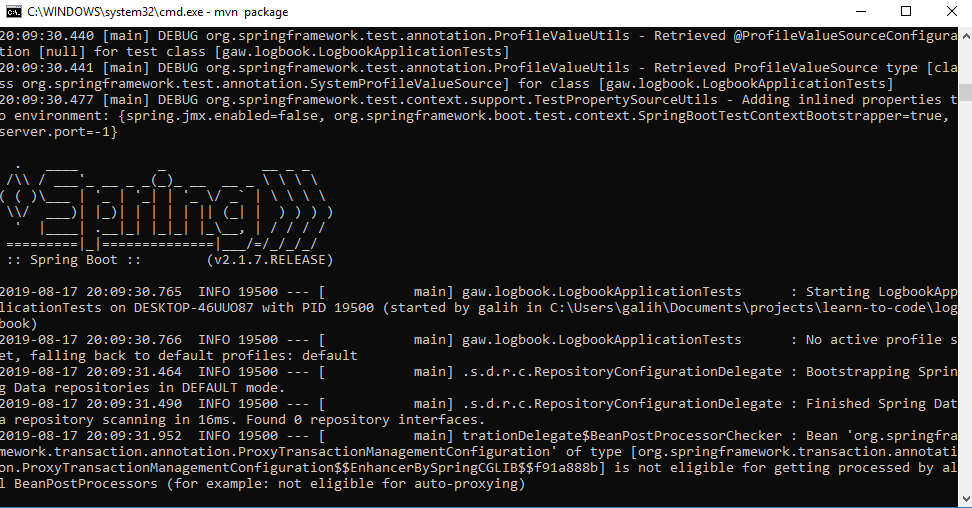
Solved!

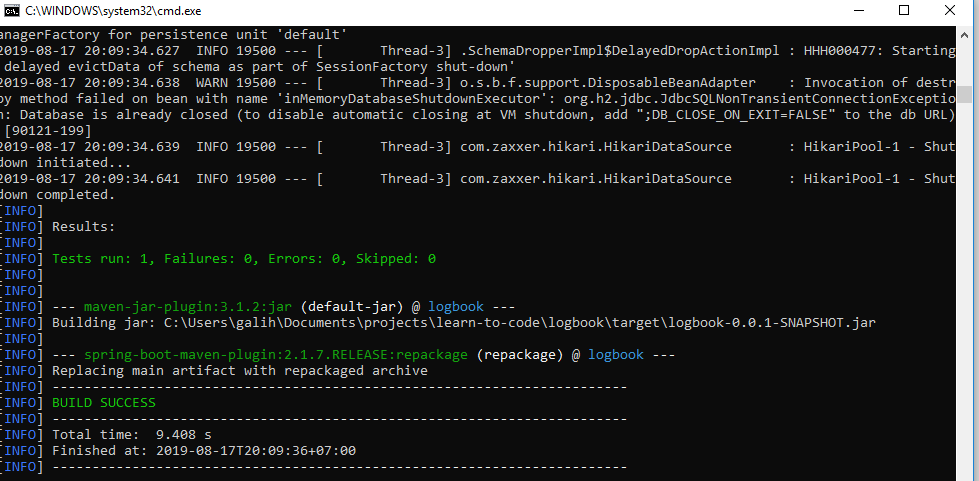
**Compile**

Bisa dari eclipse build & running (Klik kanan 🡪 Run as 🡪 Java application), tapi personal lebih suka pake cmd. Real men using cmd. cmd is badass.

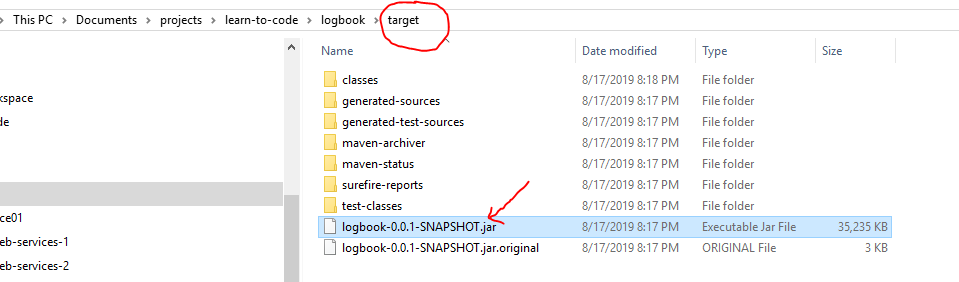
Masuk ke directory project 🡪 mvn package or mvc clean package



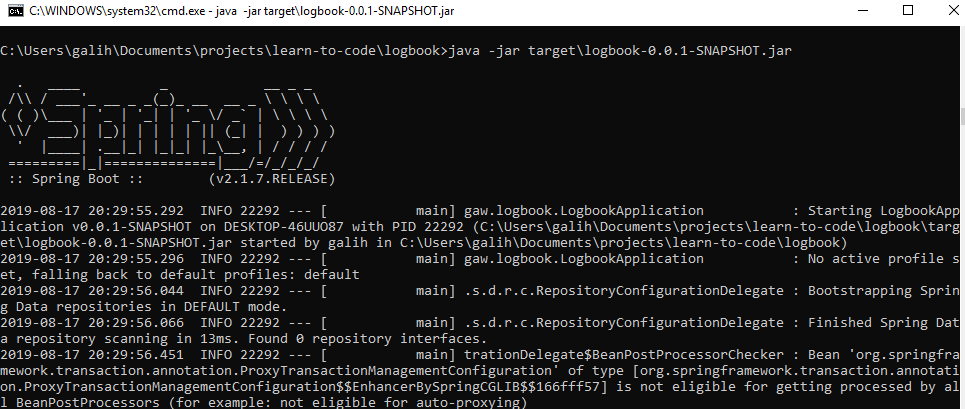




Maka terbentuklah folder ‘target’ & di dalamnya ada file .jar hasil compilannya

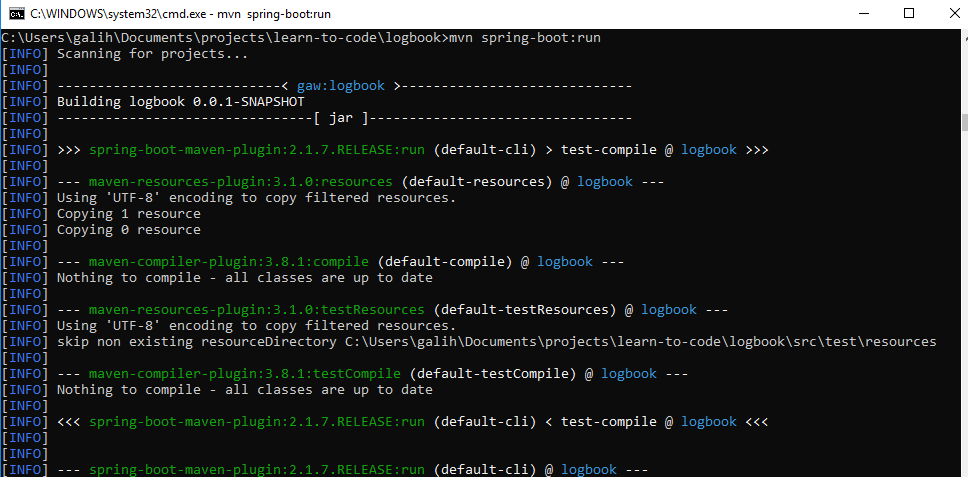


to Run 🡪 java -jar target\logbook-0.0.1-SNAPSHOT.jar



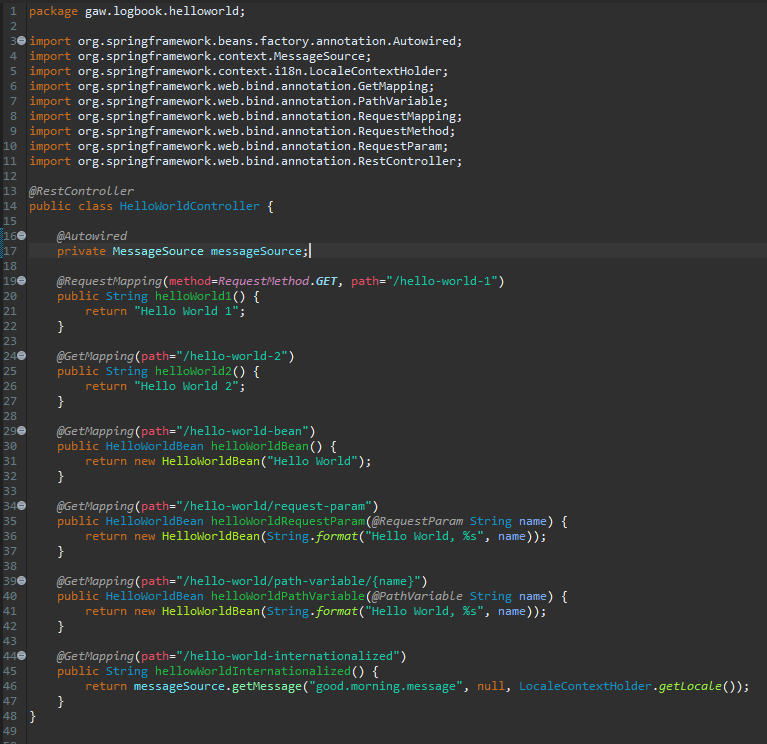
OR cara cepatnya 🡪 mvn spring-boot:run

Otomatis compile sekaligus running, tapi cara ini hanya bisa dipake pada proses development



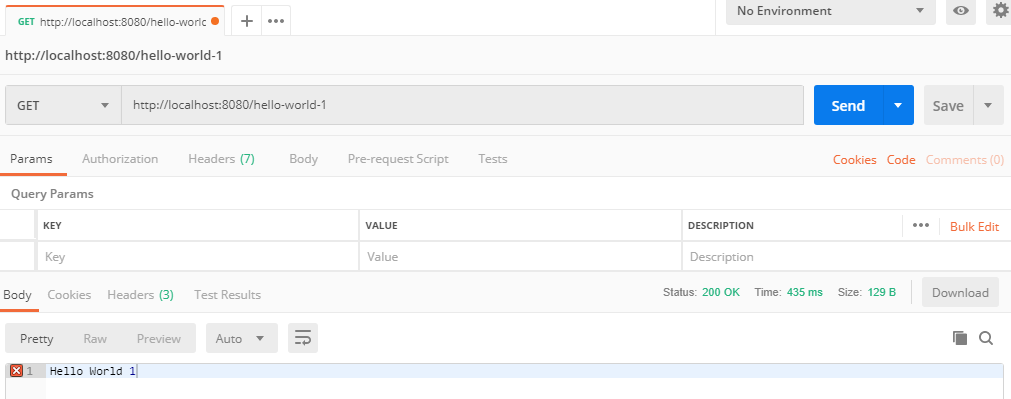
==================================================================================

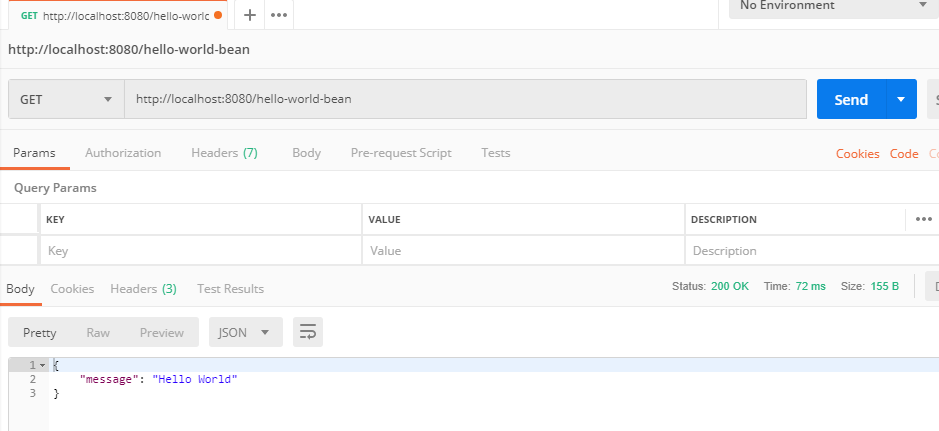
**GET**



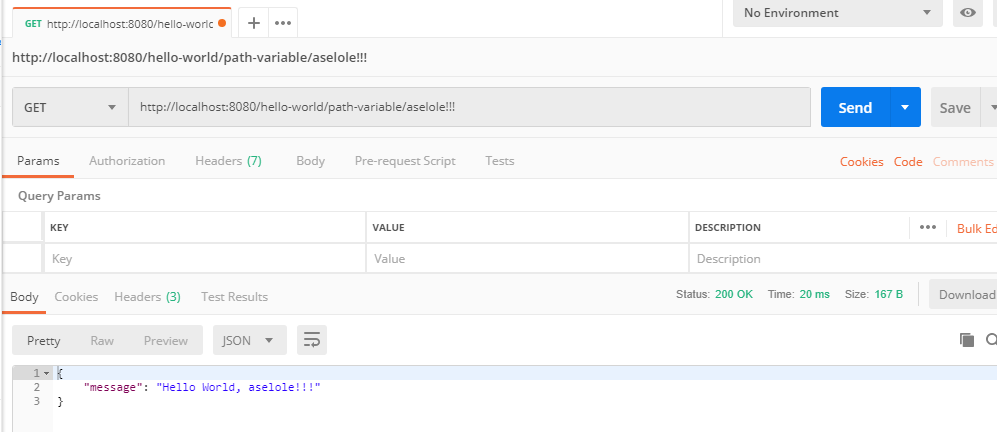
Ada 2 cara dalam request annotation: @RequestMapping atau @GetMapping

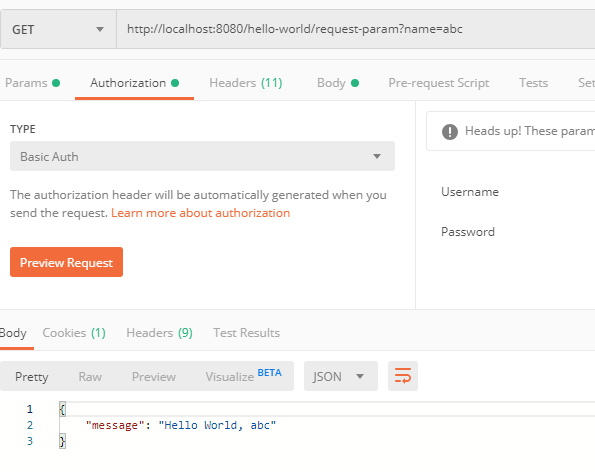
Ada 2 jenis tipe responsenya: String atau Class yang akan di-convert ke JSON





Path variable atau request param untuk melewatkan parameter dalam URI

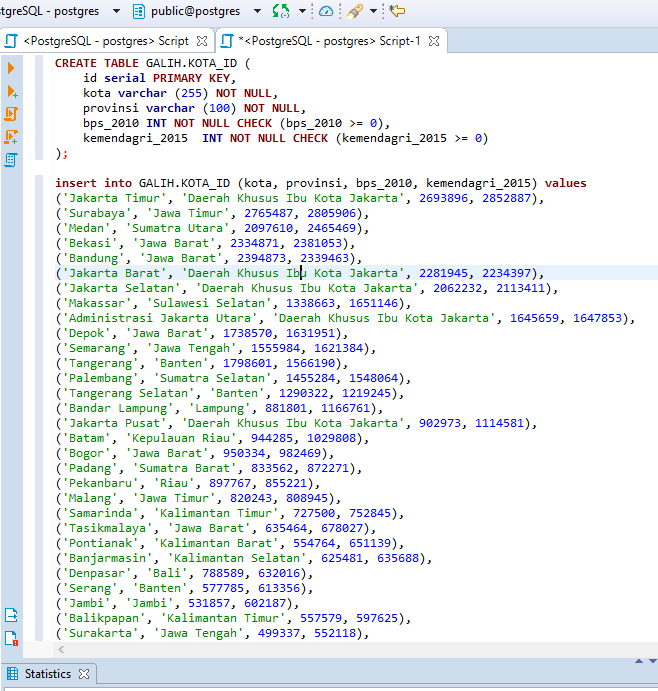




**Connect to Database (Postgre) using myBatis**

* **Create DB**

1. Create schema galih
2. Create Table KOTA\_ID
3. Insert Data

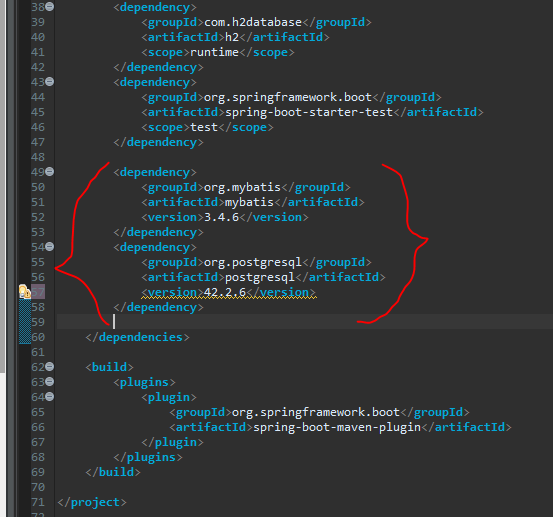




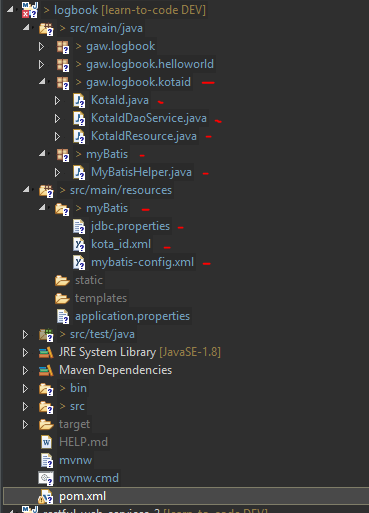
* **Configuration on app**

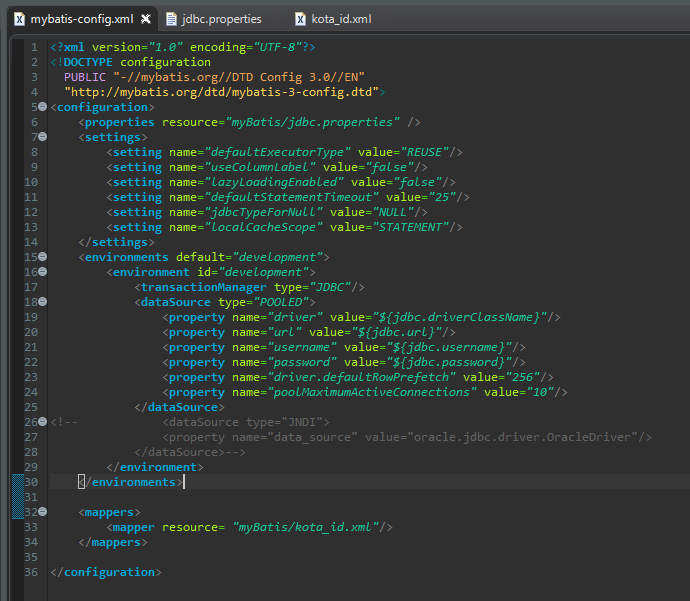
1. Tambahkan dependency myBatis & jdbc postgre di pom.xml
2. Create folder ‘myBatis’ pada bagian resource, isinya:
   1. mybatis-config.xml
   2. jdbc.properties
   3. kota\_id.xml
3. Create package ‘myBatis’ pada java source
   1. MyBatisHelper.java 🡪 Untuk koneksi antara config myBatis ke Data Access Object (DAO) Service
4. Create package ‘kotaid’ pada java source
   1. KotaId.java 🡪 POJO class (Plain Old Java Object)
   2. KotaIdDaoService.java 🡪 Manage pemanggilan query di kota\_id.xml ke controller/resource
   3. KotaIdResource.java 🡪 Controller untuk RESTful Web Service

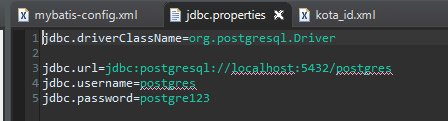
1 ----------------------------------------------------------------------------------------------------------------------------------------------------

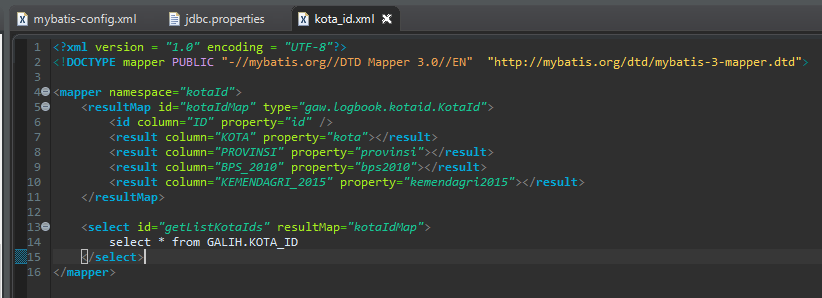


2 ----------------------------------------------------------------------------------------------------------------------------------------------------

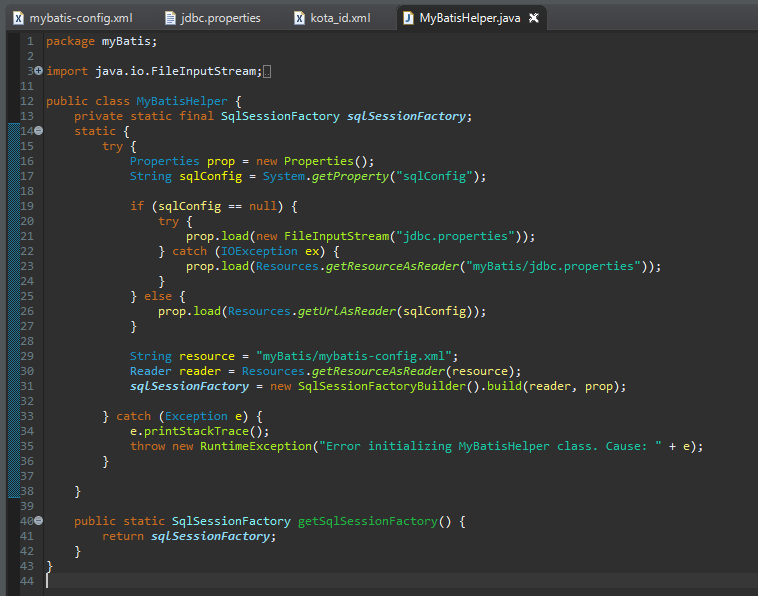








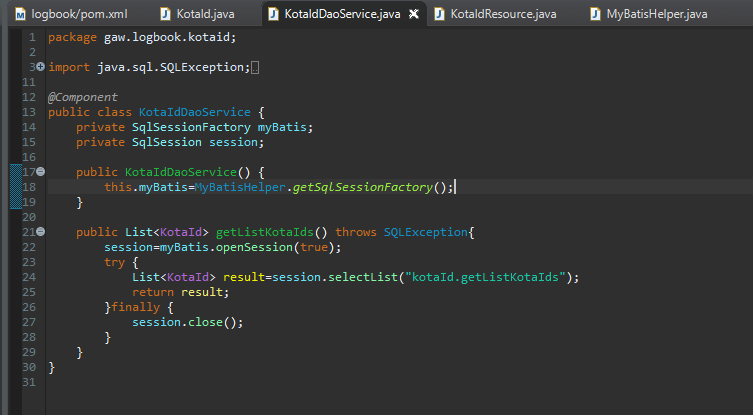
3 ----------------------------------------------------------------------------------------------------------------------------------------------------



4 ----------------------------------------------------------------------------------------------------------------------------------------------------

POJO class sesuai field database dengan private properties, generate getters/setters, constructor

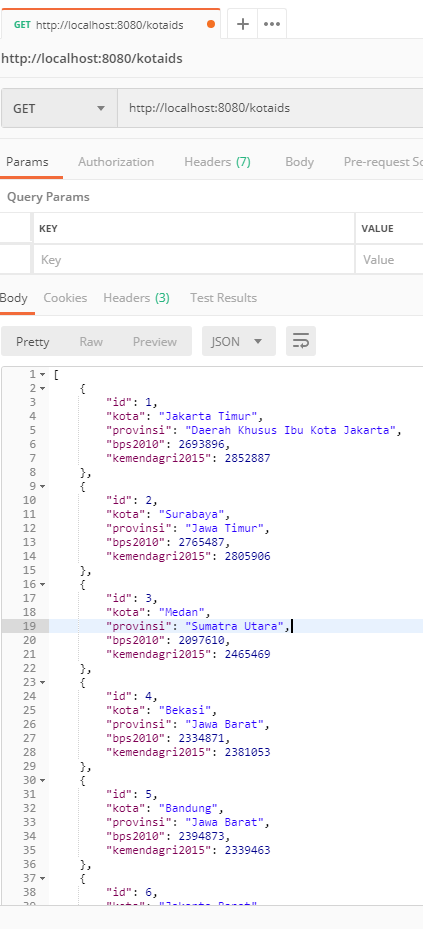






-------------------------------------------------------------------------------------------------------------------------------------------------

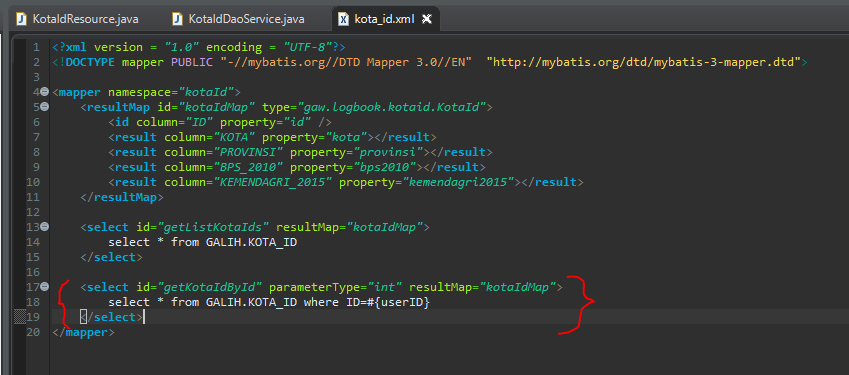
Compile, run, hit

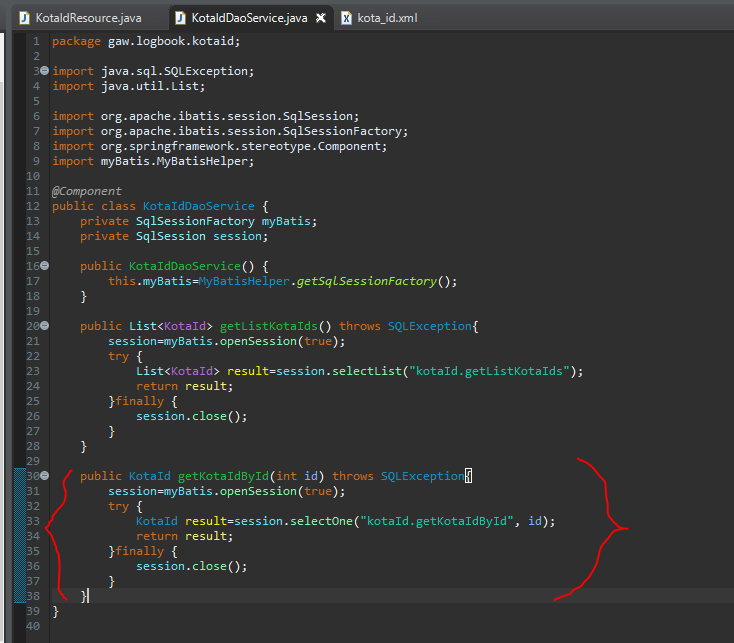


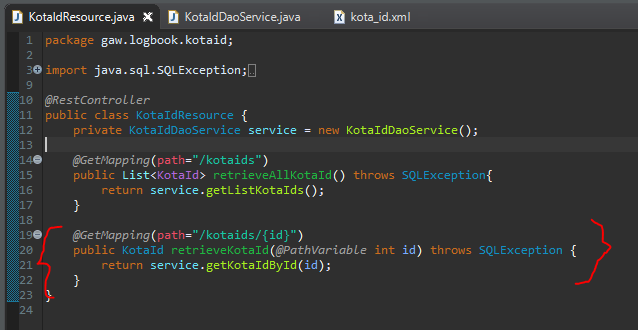
===============================================================================================

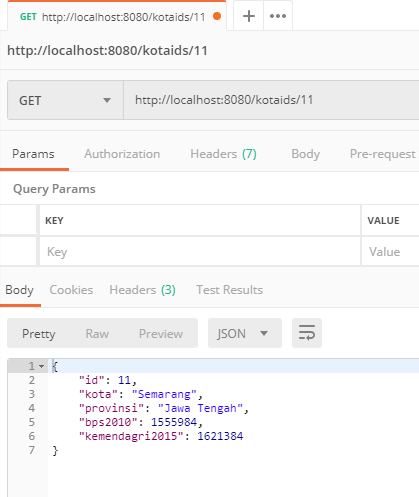
**GET with PathVariable**

1. kota\_Id.xml
2. KotaIdDaoService.java
3. KotaIdResource.java









===============================================================================================

**POST - Basic**

Ada 2 macam insert dalam postgres:

1. Insert data biasa

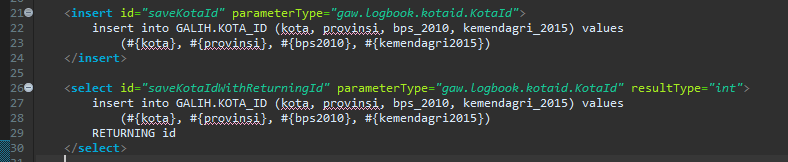
insert row baru ke table KOTA\_ID and returns jumlah row yang berhasil di-insert

1. Insert data dengan return ID

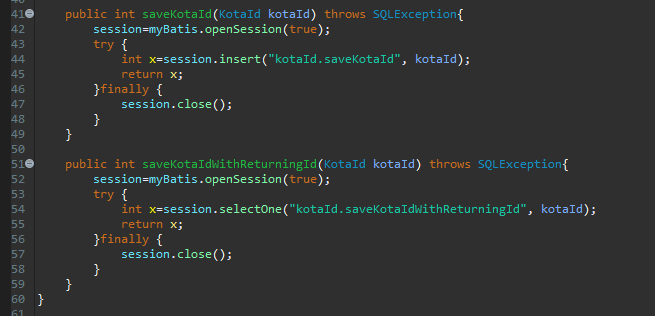
insert row baru ke table KOTA\_ID and returns value generated for the id column

note: The sequence generator is not transaction-safe. It means that if two concurrent database connections attempt to get the next value from a sequence, each client will get a different value. If one of the clients rolls back the transaction, the sequence number of that client will be unused, creating a gap in the sequence.

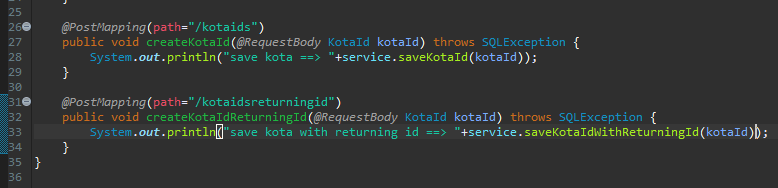
kota\_id.xml



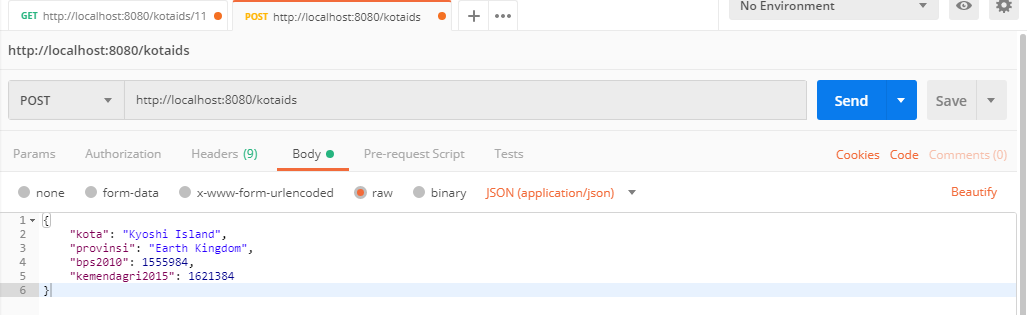
KotaIdDaoService.java



KotaIdResource.java



Test 1





Test 2

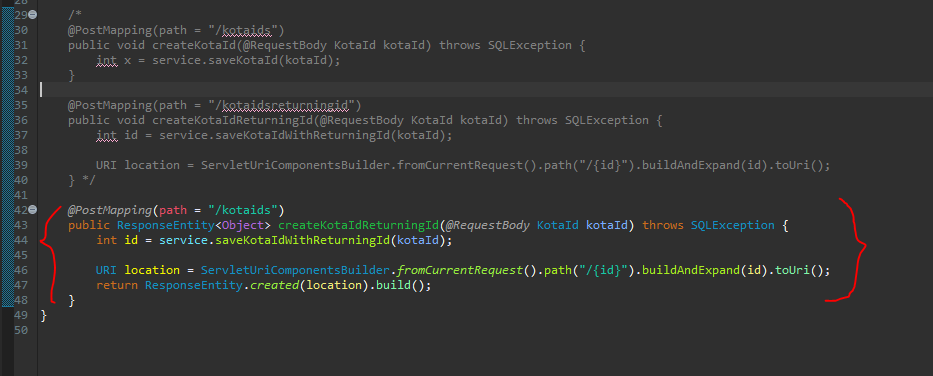




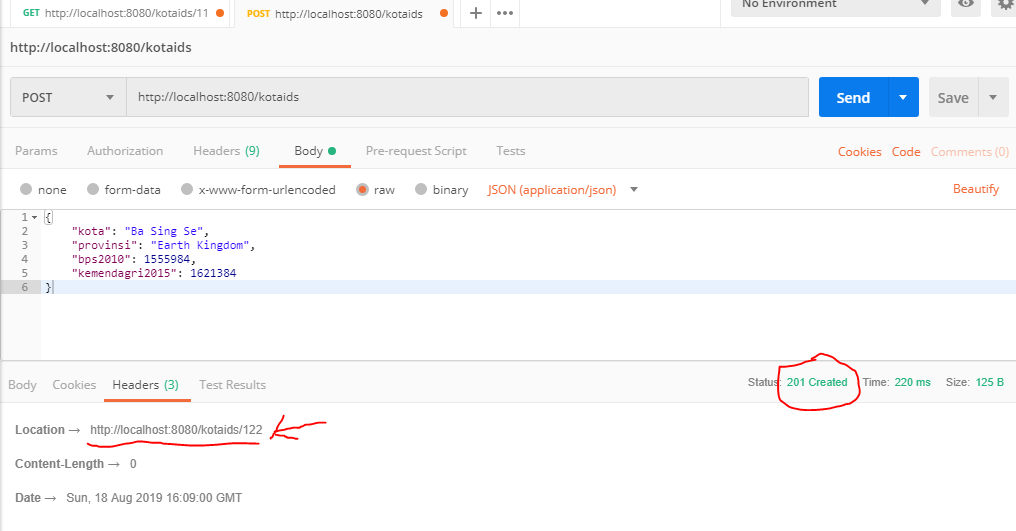
-------------------------------------------------------------------------------------------------------------------------------------------------

**POST – return correct HTTP STatuc Code and Location**

Comment 2 fungsi insert sebelumnya, kita pakai insert yang returning id



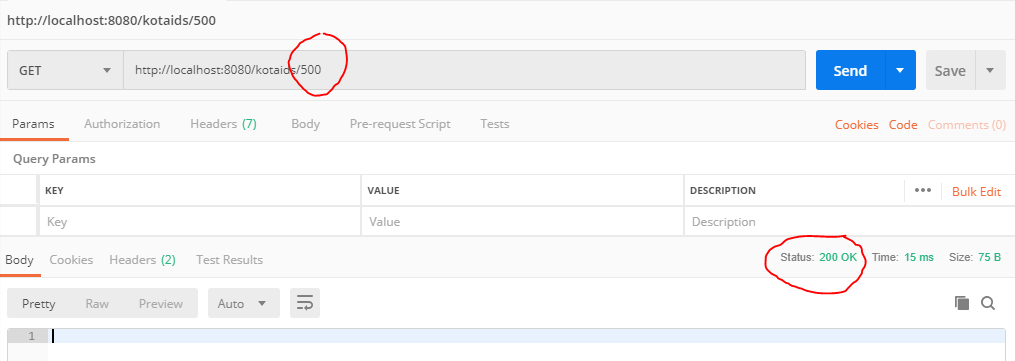
Statusnya menjadi ‘201 Created’ dan pada Headers Responsenya ada ‘Location’



===============================================================================================

**Exception Handling – 404 Resource Not Found**

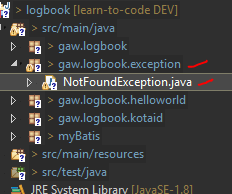
Problem: Ketika retrieve data yang tidak ada, response statusnya 200 OK

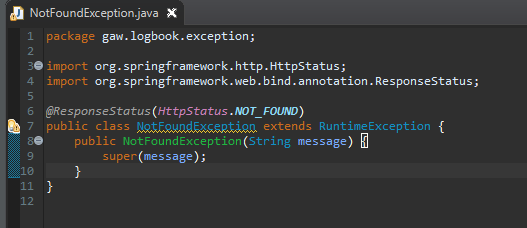


Create package ‘exception’

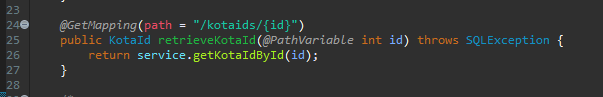
* NotFoundException.java

Pada controller, tambahkan logic if result==null, throw NotFoundException

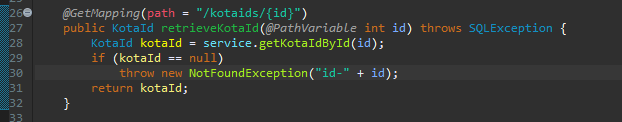




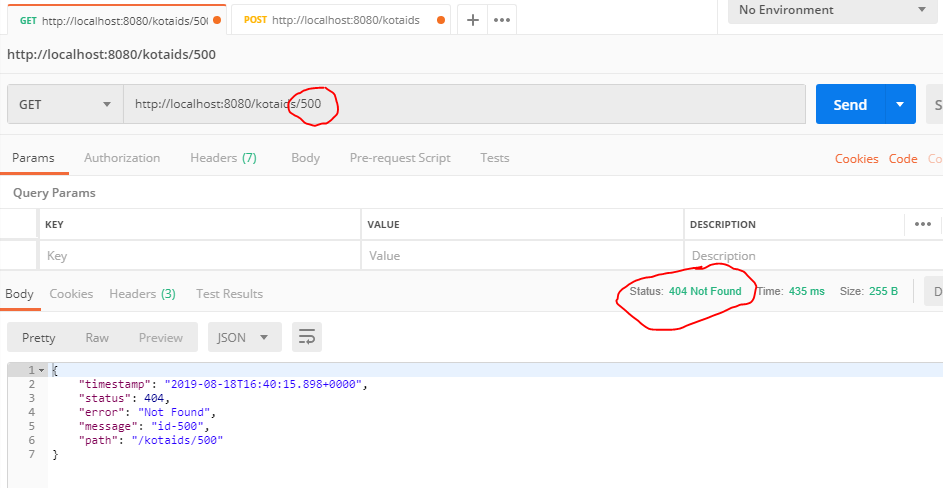
KotaIdResource.java **Before**



KotaIdResource.java **After**



Test



-----------------------------------------------------------------------------------------------------------------------------------------------------------

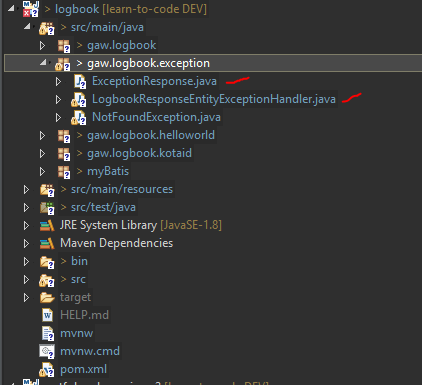
**Exception Handling – Cuztomized Response Entity Exception Handling**

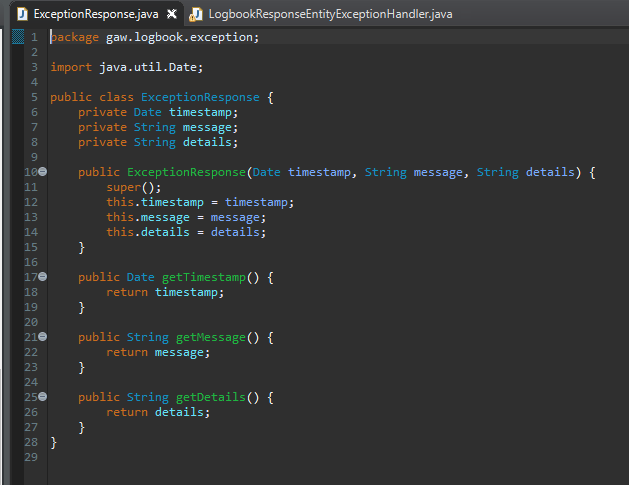
Create package ‘exception’

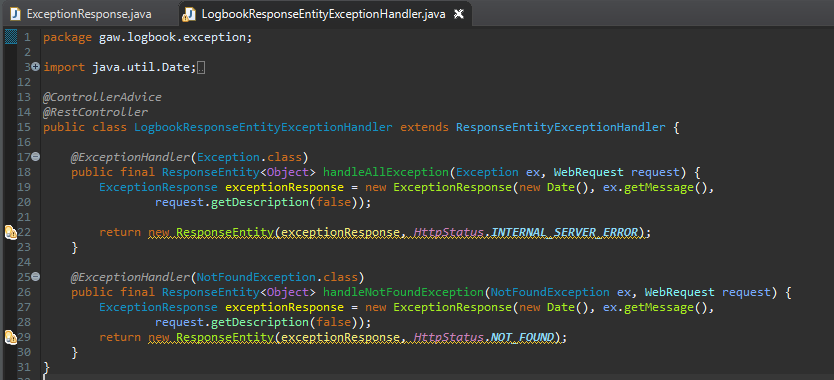
* ExceptionResponse.java

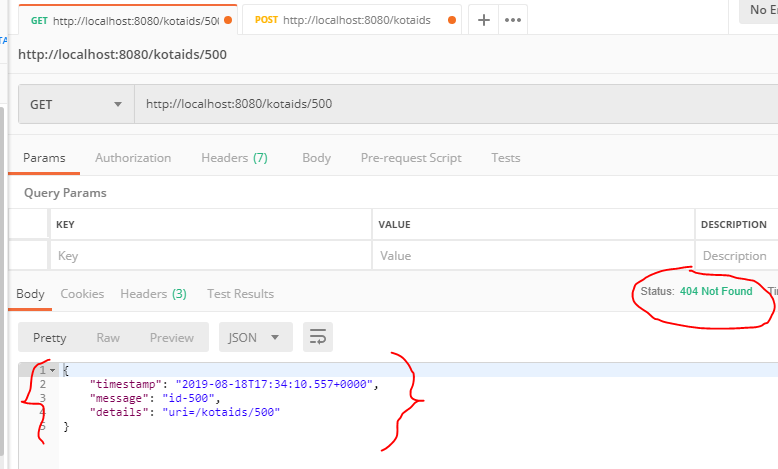
POJO class dengan construtors & getters

* LogbookReponseEntityExceptionHandler.java









===============================================================================================

**DELETE**

Ada 3 macam cara

1. Delete secara biasa

Delete data dengan return int jumlah data yang telah di-delete

1. Delete dengan return Object

Delete data dengan return Object yang telah di-delete

1. Delete dengan return List of Object

Delete data dengan return List of Object yang telah di-delete

kota\_id.xml



KotaIdDaoServie.java

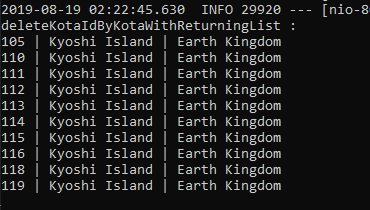


KotaIdResource.java

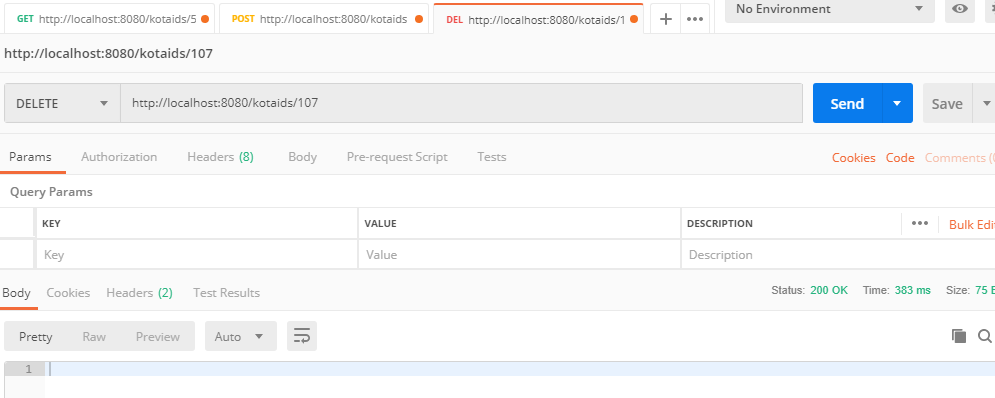


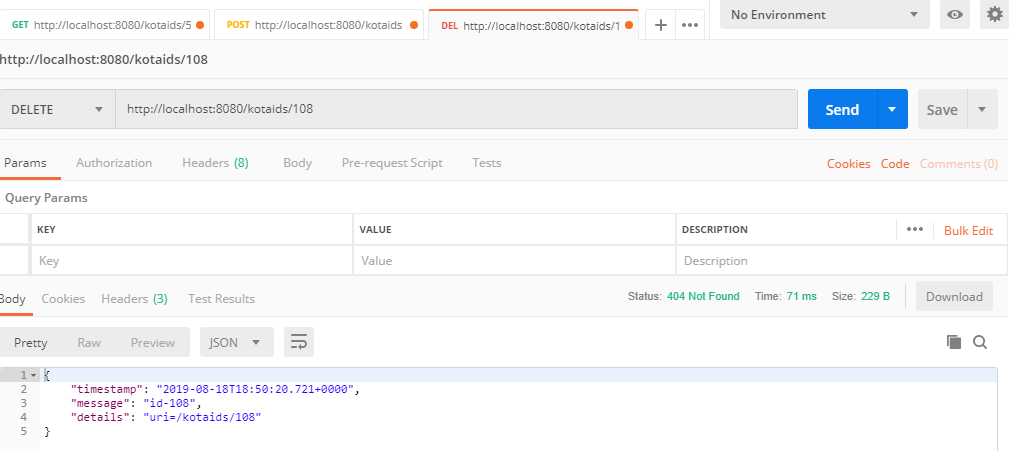
Test



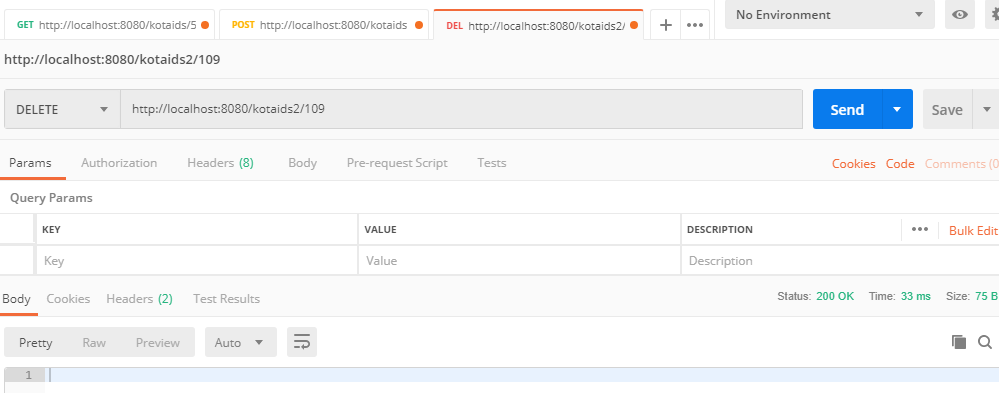


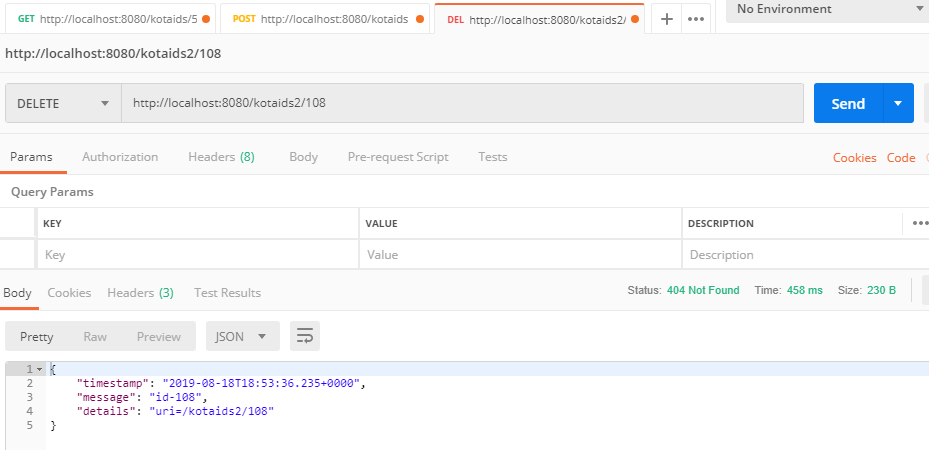
Test 1



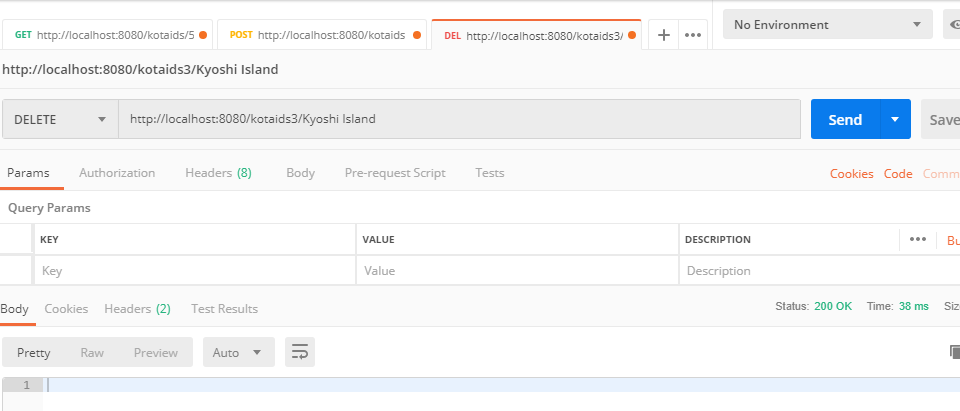


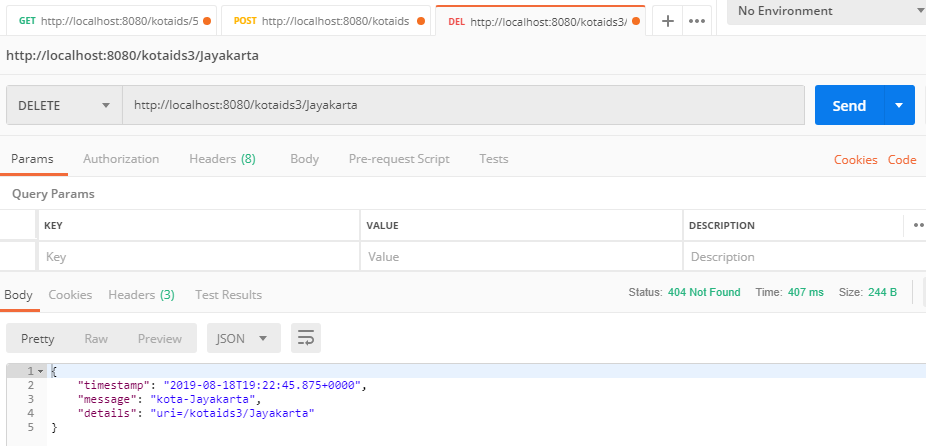
Test 2





Test 3





===============================================================================================

**VALIDATION & CUZTOMIZATION JSON PROPERTY**

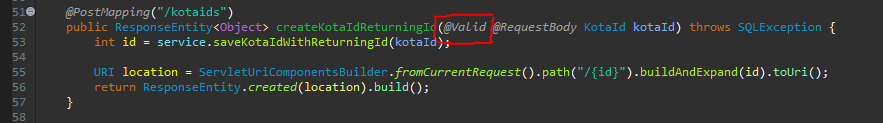
**Validation**

* Controller/resource | KotaIdResource.java 🡪 Tambahkan @Valid annotation pada RequestBody
* POJO Class | KotaId.java 🡪 javax.validation.constraints
* ExceptionHandler | LogbookResponseEntityExceptionHandler.java 🡪 Set Response Status & menampilkan response error

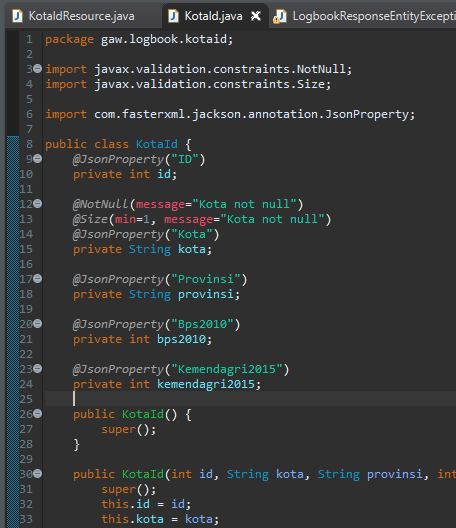
**Cuztomization JSON Property**

* POJO Class | KotaId.java 🡪 @JSonProperty

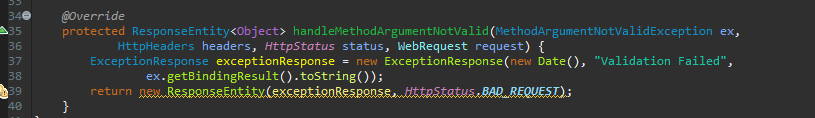
KotaIdResource.java



KotaId.java



LogbookResponseEntityExceptionHandler.java



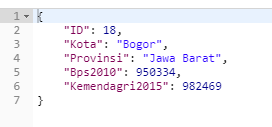
Test Validation



Test JSON Property Before



Test JSON Property After

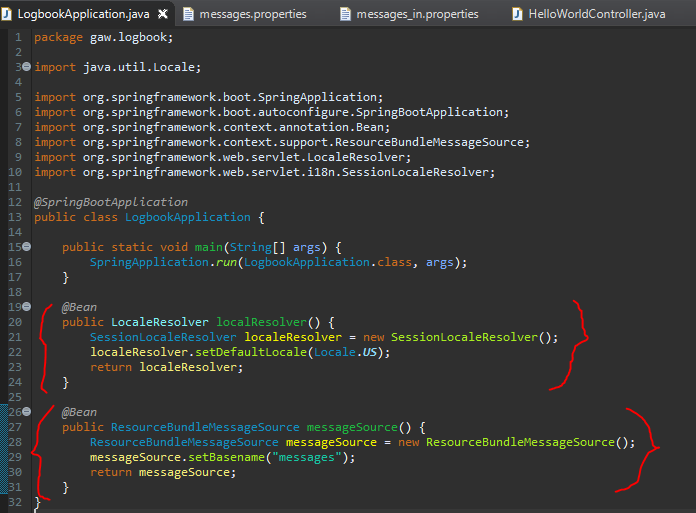


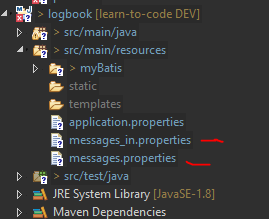
===============================================================================================

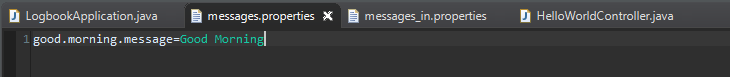
**INTERNATIONALIZATION**

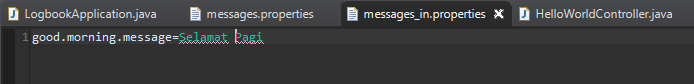
**Cara 1**

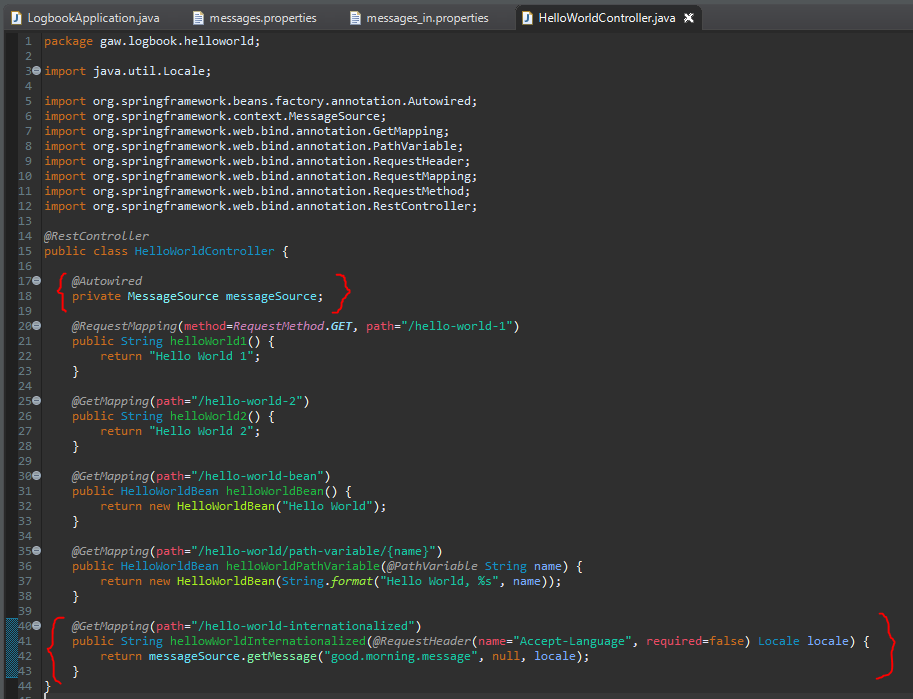
* Config di kelas utama dimana @SpringBootApplication berada 🡪 LocaleResolver + messageResource
* Properties file 🡪 messages.properties & messages\_in.properties
* HelloWorldController.java 🡪 messageSource.getMassage





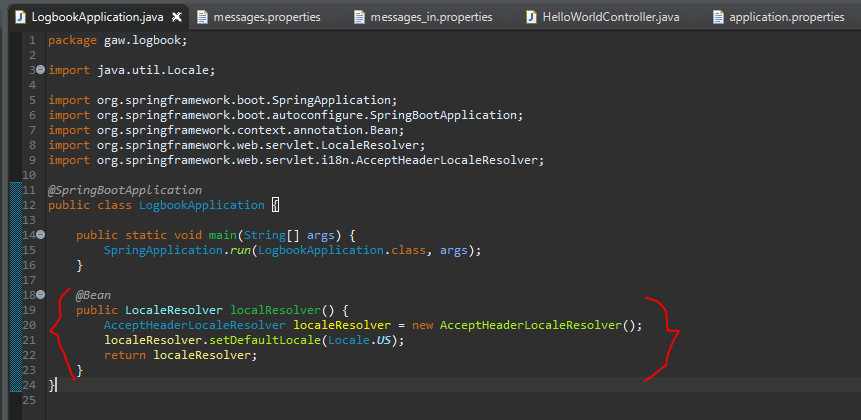


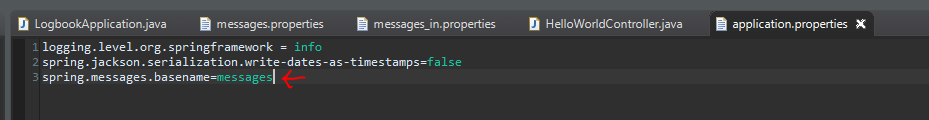


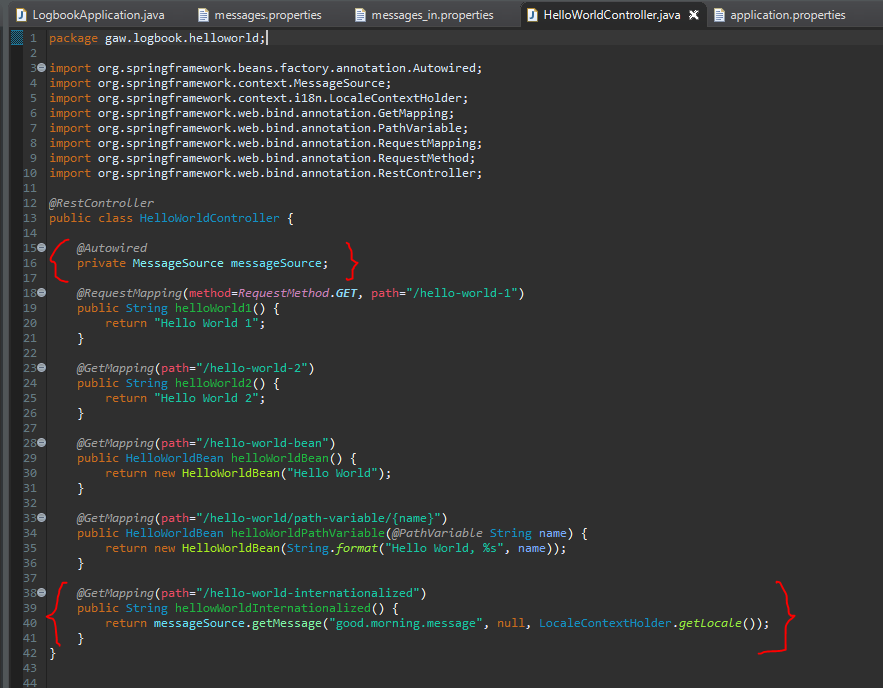


**Cara 2** – Simplifikasi Cara 1

* Config di kelas utama dimana @SpringBootApplication berada 🡪 LocaleResolver
* Properties file 🡪 messages.properties & messages\_in.properties + set Basename di application.properties
* HelloWorldController.java 🡪 messageSource.getMassage





===============================================================================================

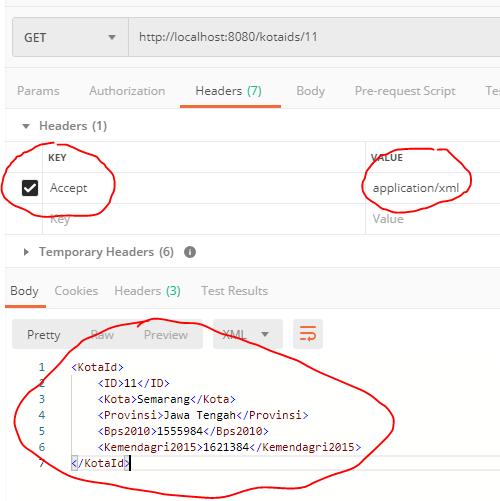
**CONTENT NEGOTIATION**

Responnya ga cuma JSON tapi bisa XML juga

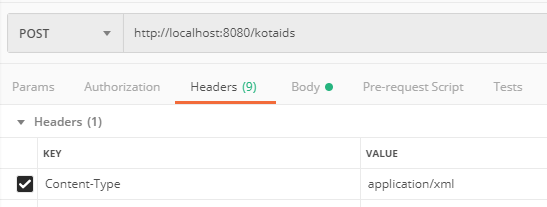
* pom.xml 🡪 Jackson-dataformat-xml

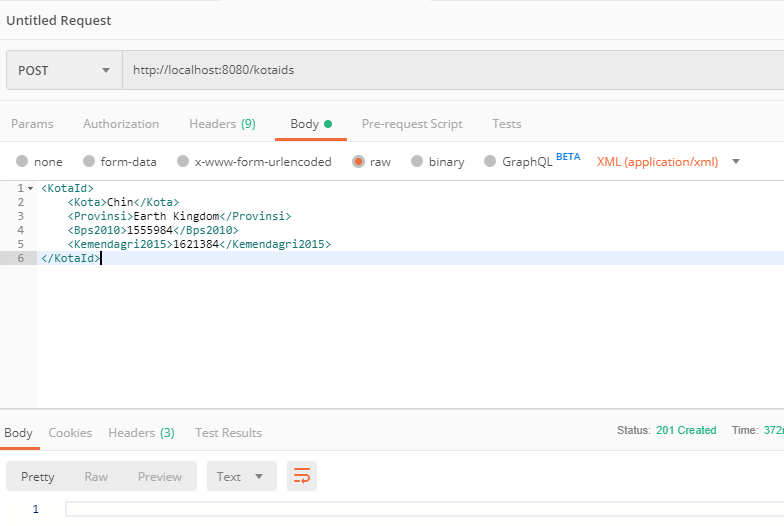


Test GET



Test POST



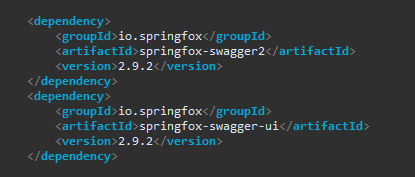


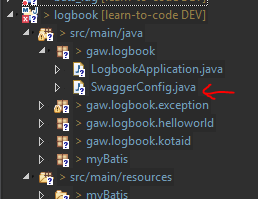
===============================================================================================

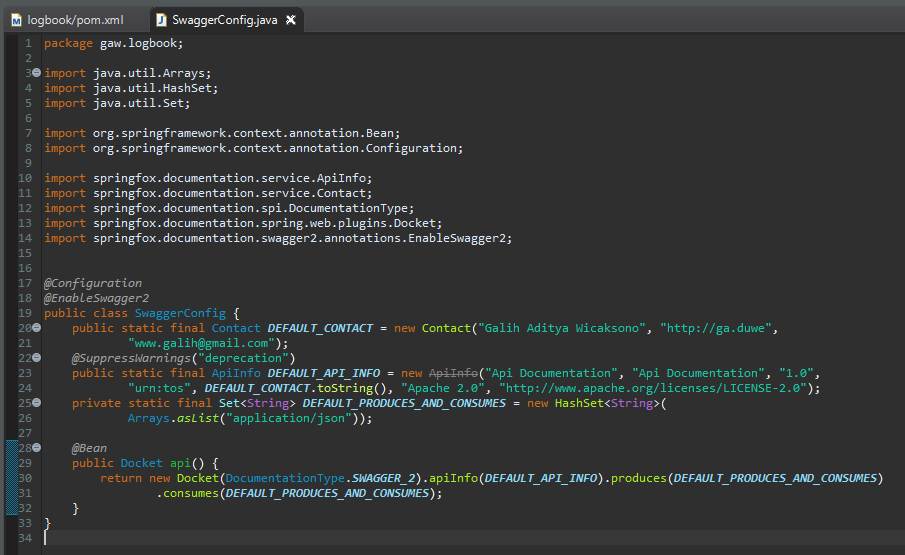
**SWAGGER**

One of the most popular documentation format for restful services

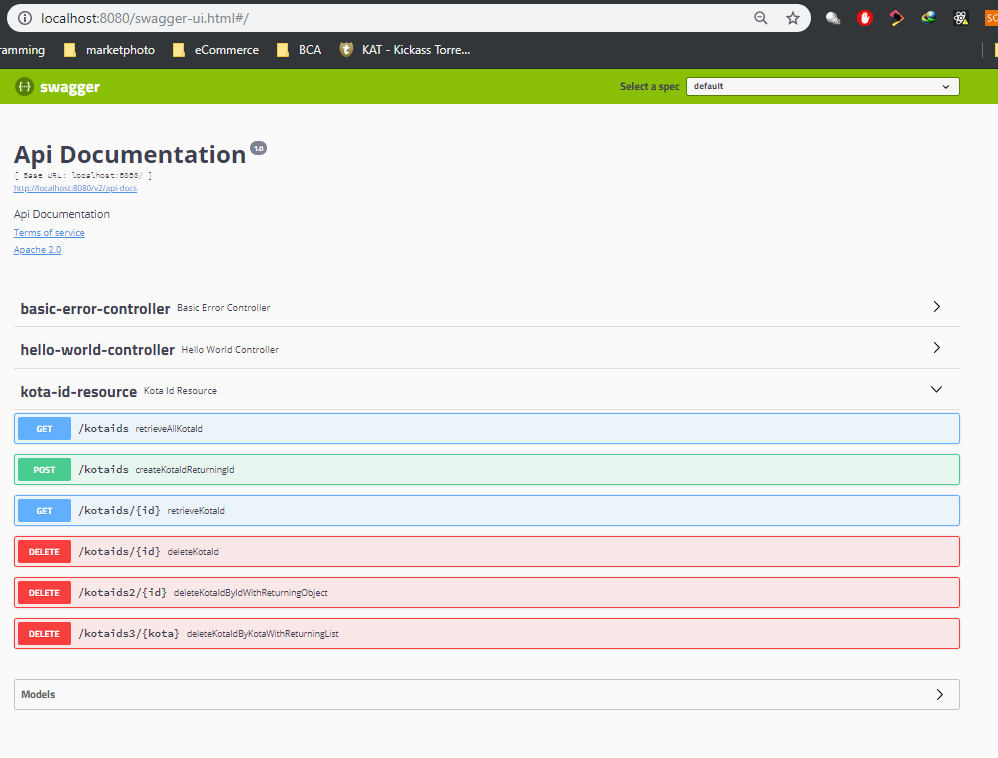
* pom.xml
  + springfox-swagger2
  + springfox-swagger-ui
* SwaggerConfig.java di package yang sama dengan *@SpringBootApplication* berada







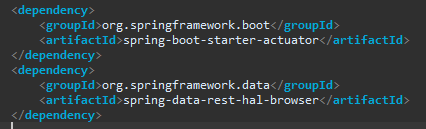
Akses <http://localhost:8080/swagger-ui.html>

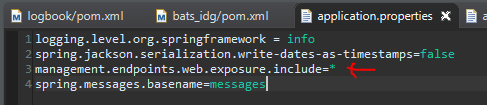


==============================================================================================

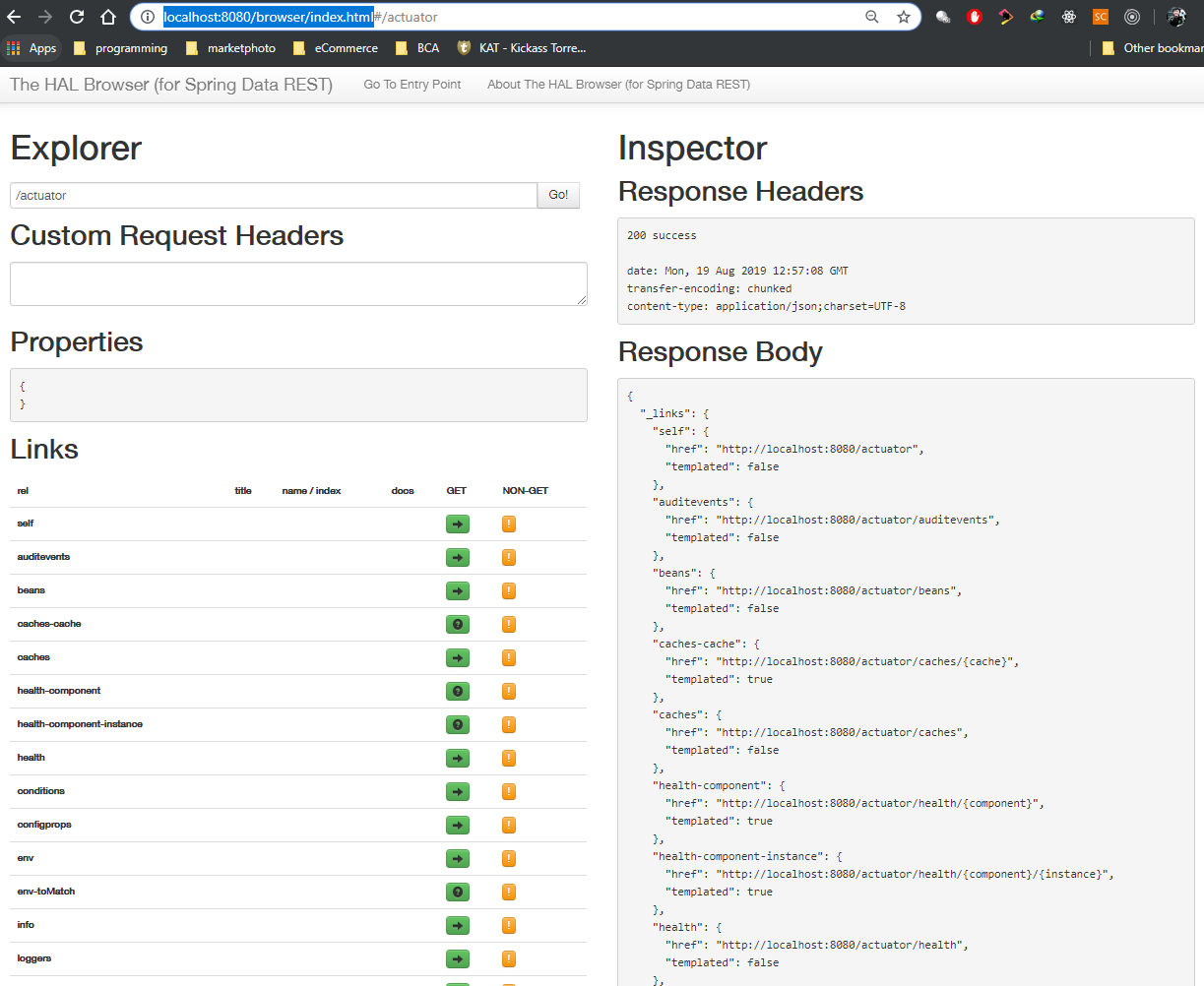
**MONITORING API** with Spring Boot Actuator

* pom.xml
  + spring-boot-starter-actuator
  + spring-data-rest-hal-browser 🡪 HAL : Hypertext Application Language
* application.properties
  + management.endpoints.web.exposure.include=\*
* nbbb





<http://localhost:8080/browser/index.html>



**MONITORING API** with Spring Boot Admin Server

Source :

* <http://codecentric.github.io/spring-boot-admin/2.1.6/#getting-started>
* <https://github.com/codecentric/spring-boot-admin>
* <https://www.vojtechruzicka.com/spring-boot-admin/>

How to:

* **Admin**

1. Generate project from <https://start.spring.io/>, mport ke eclipse
2. pom.xml
   * Spring Boot Server starter to dependencies
   * Spring boot starter security
   * Spring boot starter web, exclude tomcat (karena masih ada bug). Ganti pake jetty.
3. LogbookAdminApplication.java
   * @EnableAdminServer
4. SecurityConfig.java
5. application.properties

* **Client**

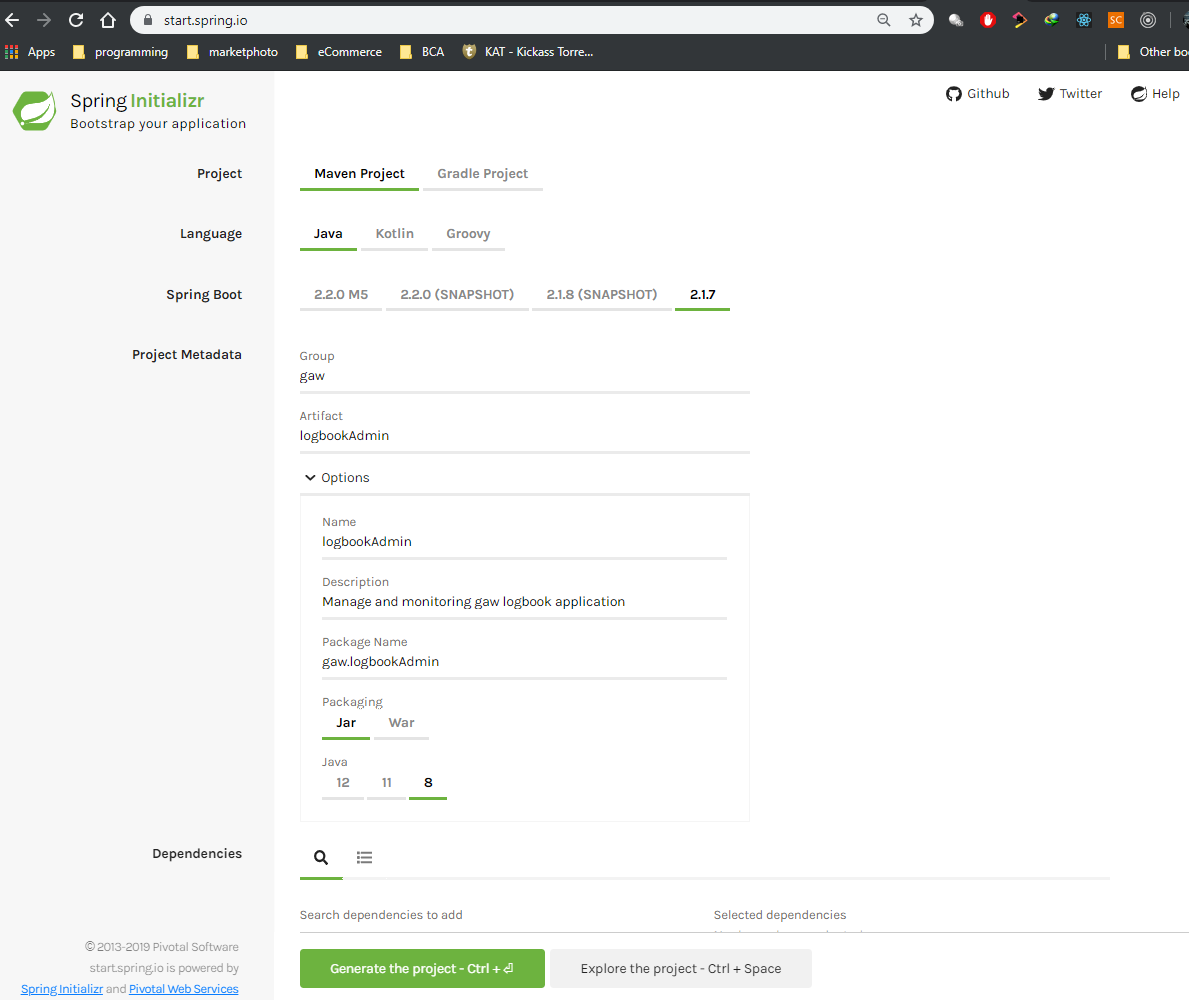
1. pom.xml
   * Spring boot admin starter client
2. LogbookApplication.java

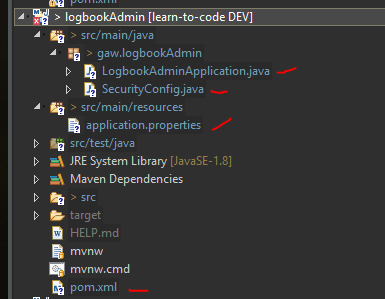
* SecurityPermitAllConfig extends WebSecurityConfigurerAdapter

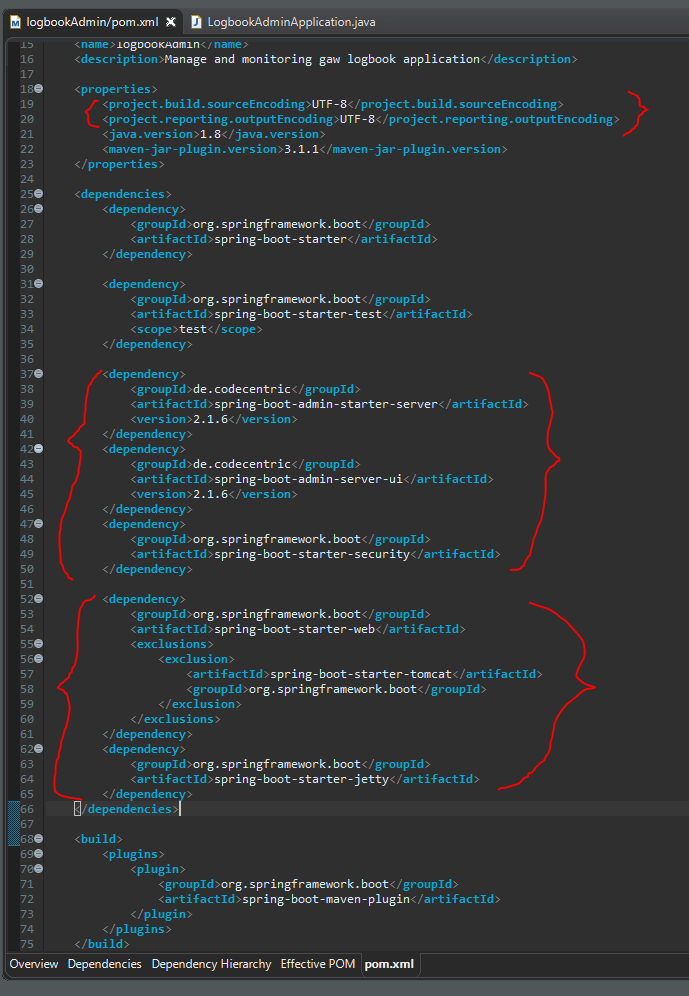
1. application.properties

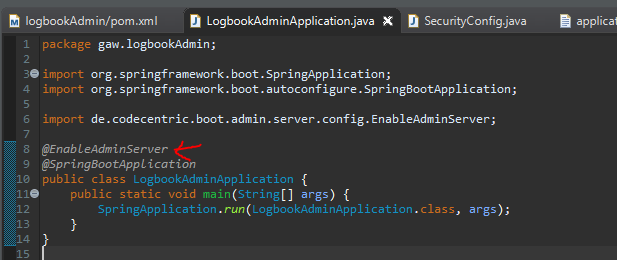
* registrasi server admin & config lainnya

**Admin** -------------------------------------------------------------------------------------------------------------------------------------------------

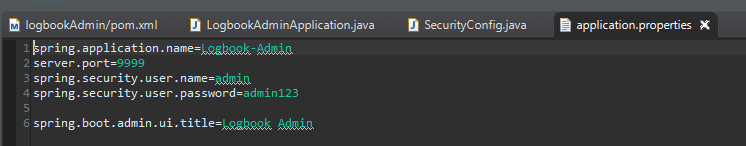




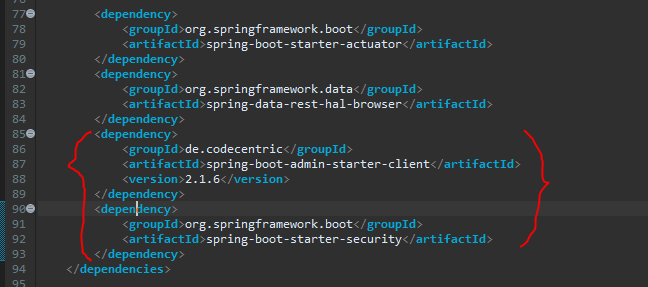


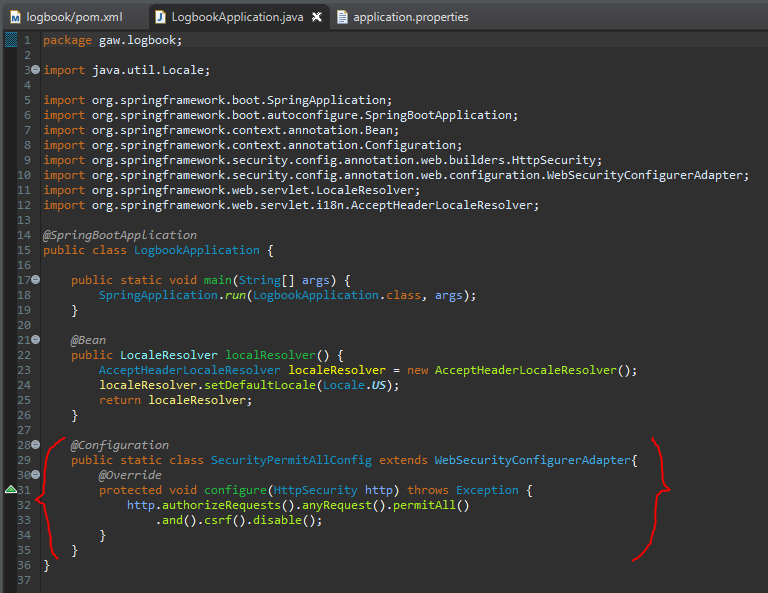


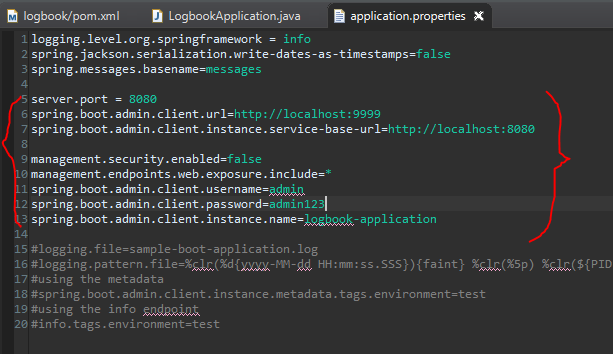




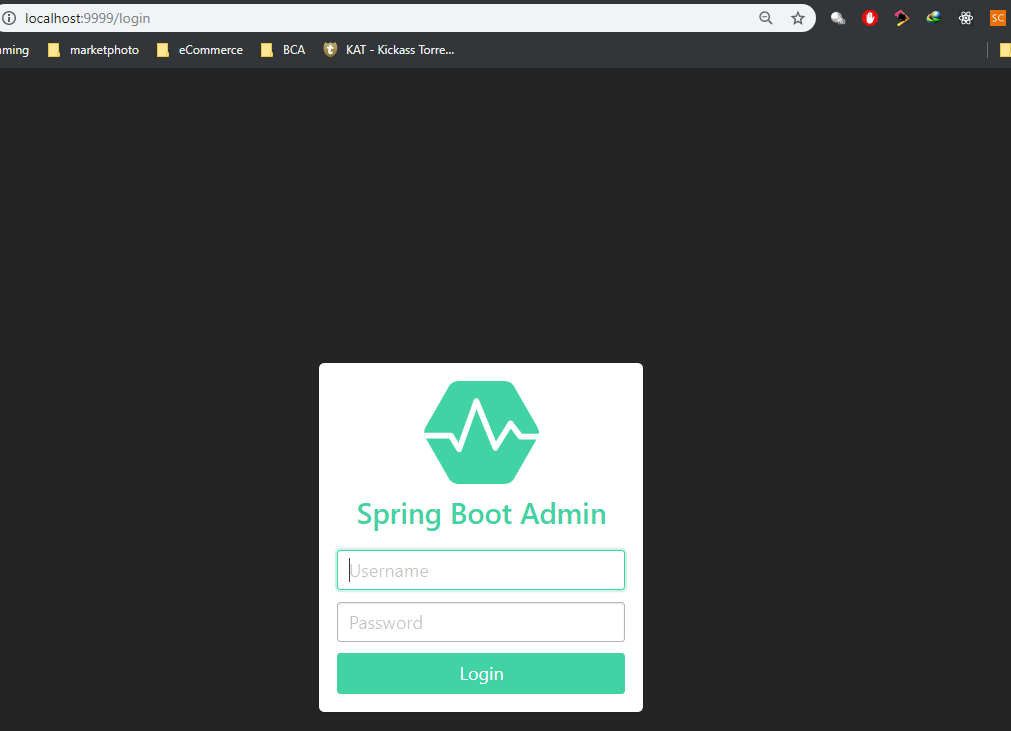
**Client** --------------------------------------------------------------------------------------------------------------------------------------------------

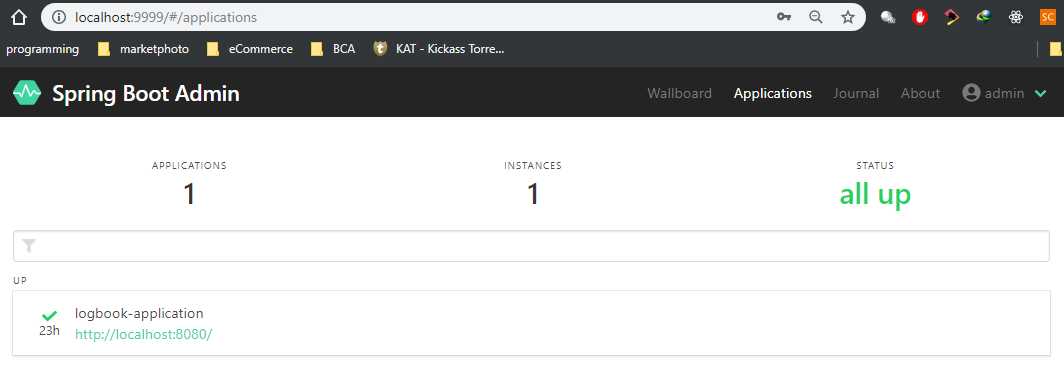


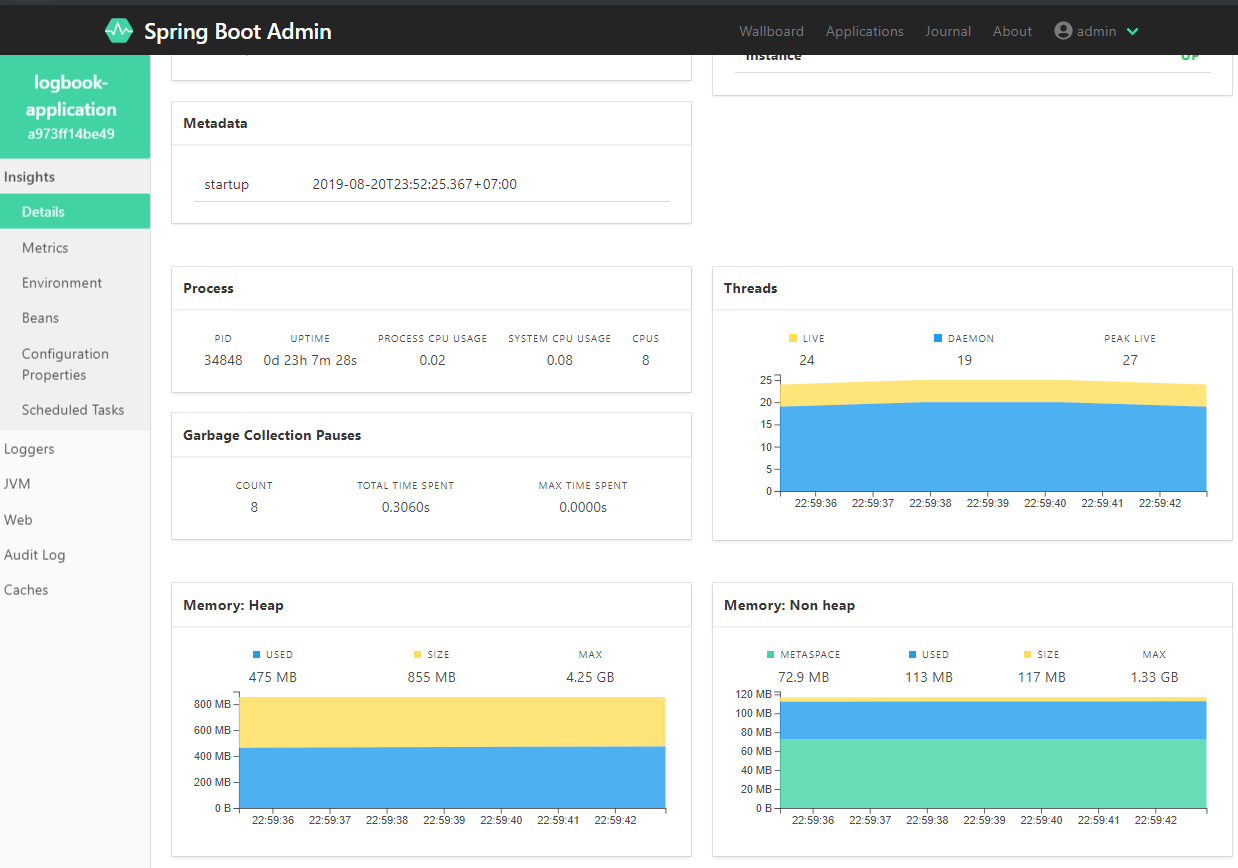




**Test** --------------------------------------------------------------------------------------------------------------------------------------------------





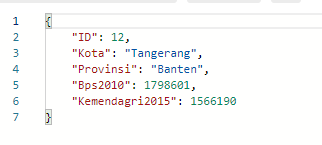


==============================================================================================

**FILTERING**

1. Static 🡪 Tambahkan @JsonIgnore pada property class POJO/Bean
2. Dynamic
   * KotaIdResource.java 🡪 return MappingJacksonValue
   * KotaId,java 🡪 Tambah @JsonFilter pada class POJO/Bean

Before



**Static** --------------------------------------------------------------------------------------------------------------------------------------------------

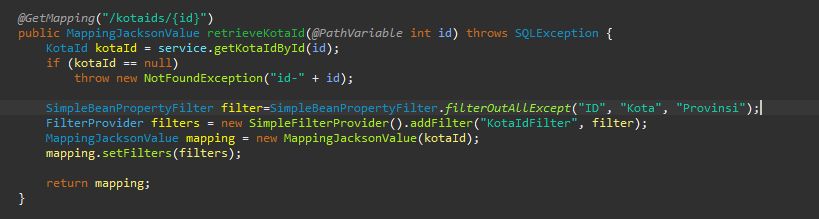
KotaId.java



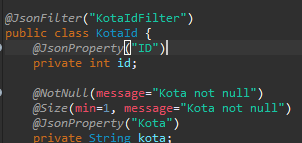


**Dynamic** ----------------------------------------------------------------------------------------------------------------------------------------------

KotaIdResource.java



KotaId.java





==============================================================================================

**VERSIONING**

1. Media type versioning (a.k.a "content negotiation" or "accept header") 🡪 Github
2. (Custom) headers versioning 🡪 Microsoft
3. Request Parameter versioning 🡪 Amazon
4. URI versioning 🡪 Twitter

- Factors

- URI Pollution

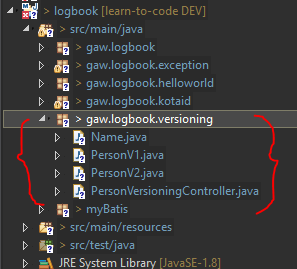
- Misuse of HTTP Headers

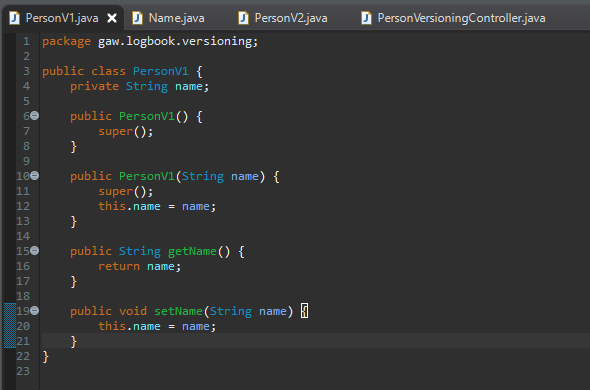
- Caching

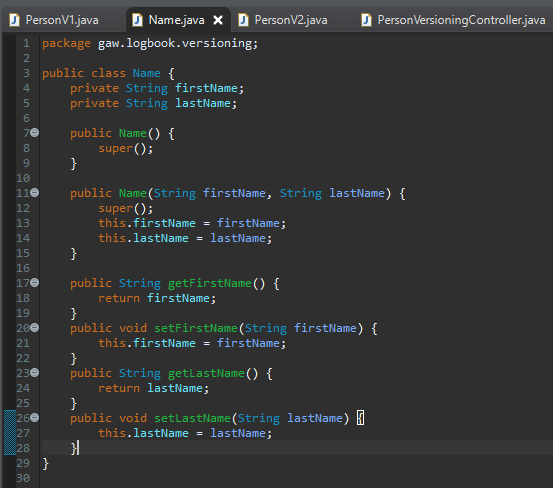
- Can we execute the request on the browser?

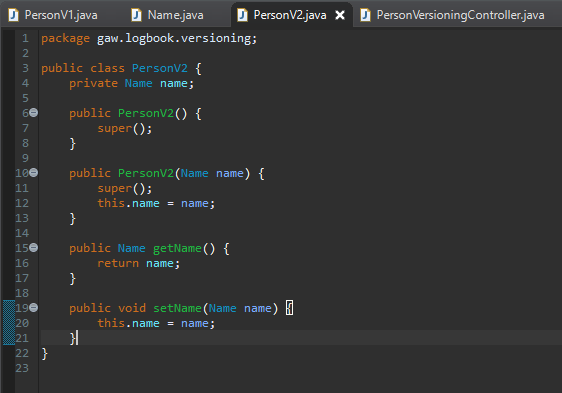
- API Documentation

- No Perfect Solution



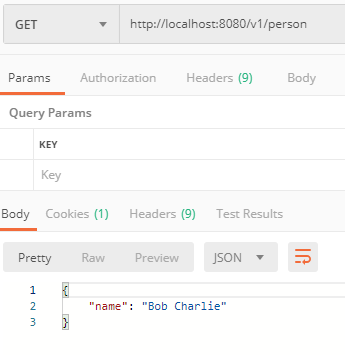


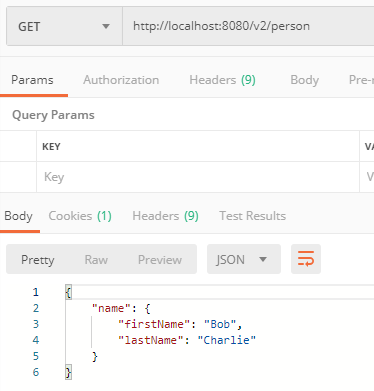




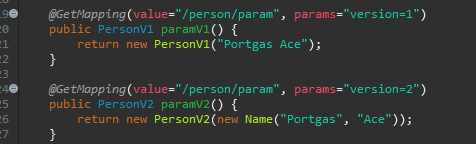
URI versioning ---------------------------------------------------------------------------------------------------------------------------------------

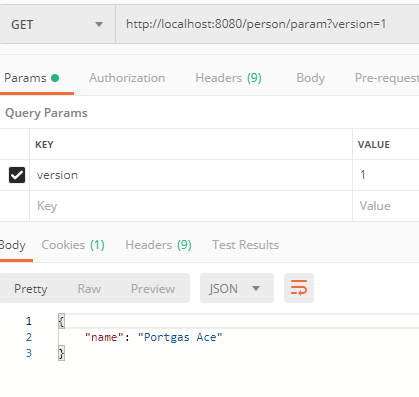


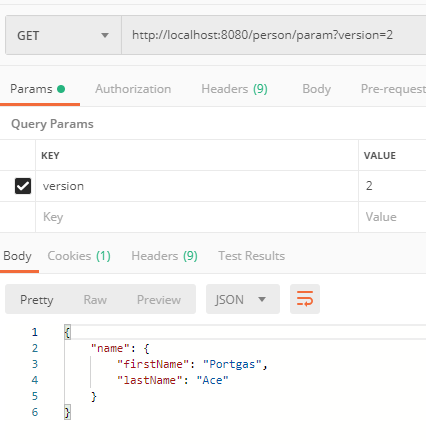




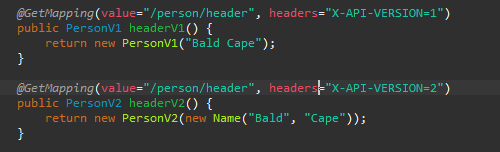
Request Parameter versioning --------------------------------------------------------------------------------------------------------------

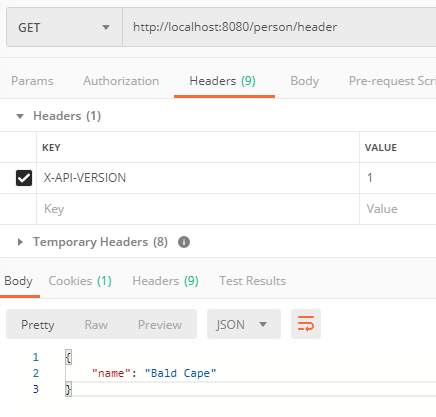


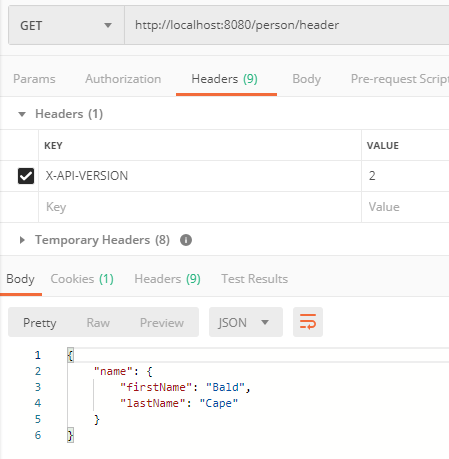


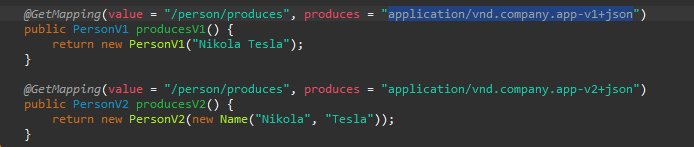


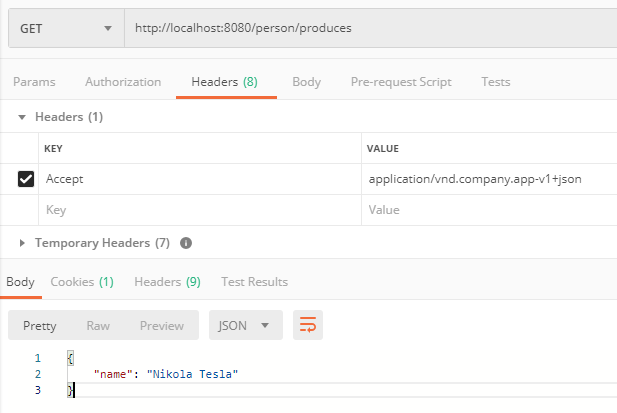
(Custom) headers versioning --------------------------------------------------------------------------------------------------------------------

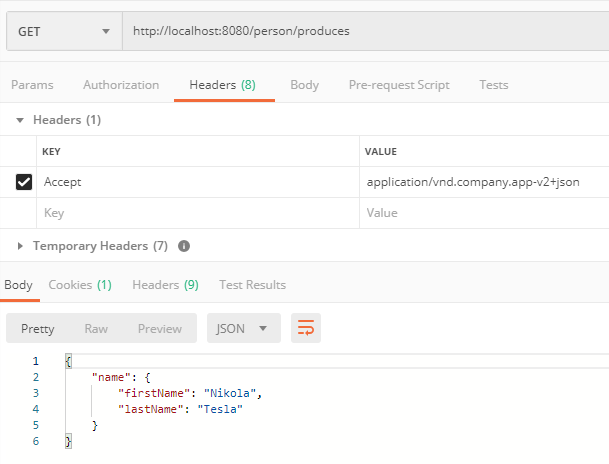






Media type versioning ----------------------------------------------------------------------------------------------------------------------------

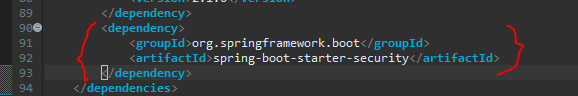


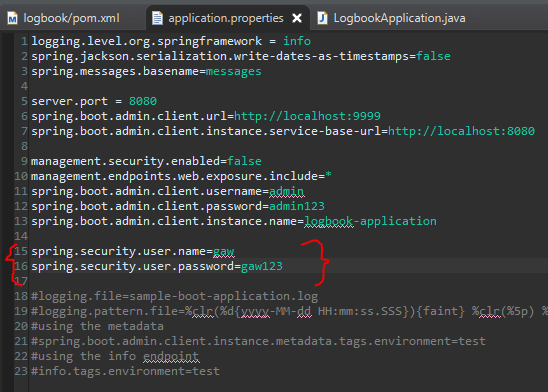


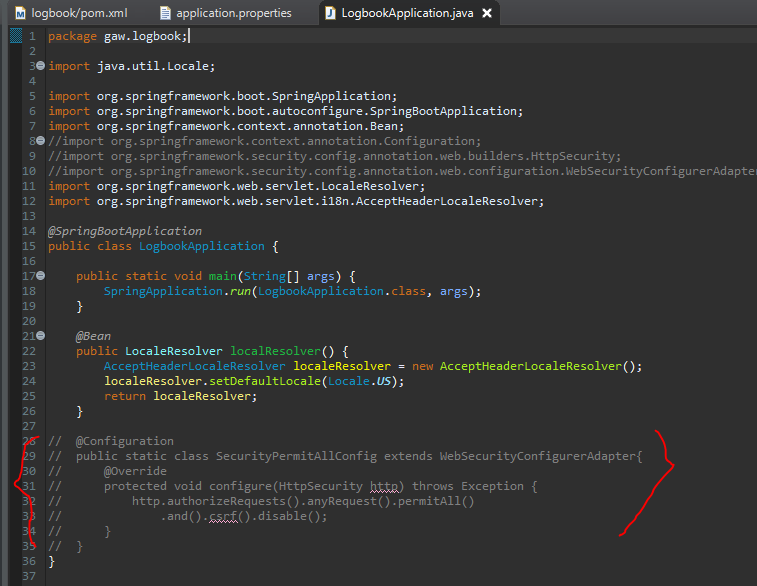
==============================================================================================

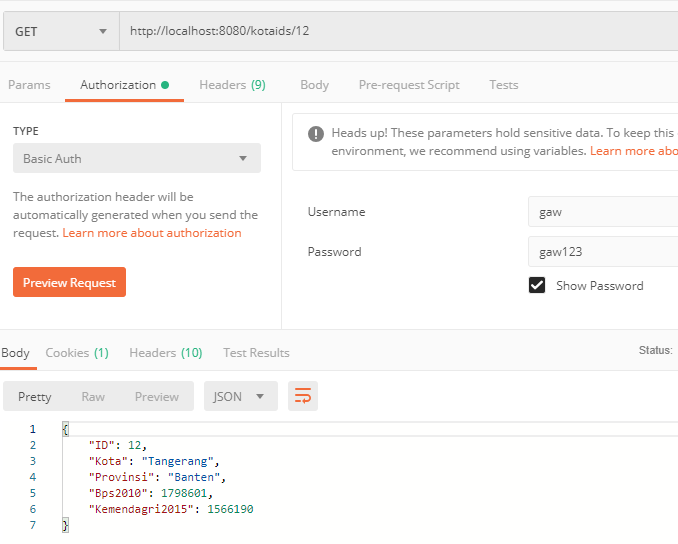
**AUTHENTICATION –** Basic

1. pom.xml 🡪 spring-boot-starter-security
2. application.properties 🡪 config username & password
3. LogbookApplication.java 🡪 Disable config securityPermitAllConfig

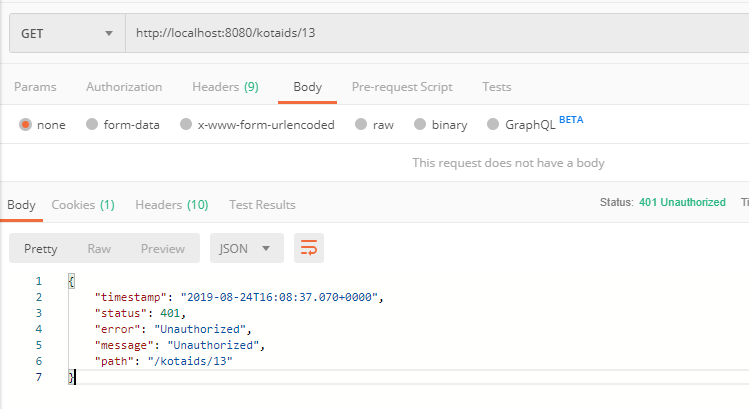








Jika tanpa Auth



===============================================================================================

**RICHARDSON MATURITY MODEL**

1. **level 0**

Expose SOAP Web Services in REST style

* http://server/getPosts
* http://server/deletePosts
* http://server/doThis

1. **Level 1**

Expose Resources with proper URI

* http://server/accounts
* http://server/accounts/10

Note: Improper use of HTTP methods

1. **Level 2**

Level 1 + HTTP Methods

1. **Level 3**

Level 2 + HATEOAS

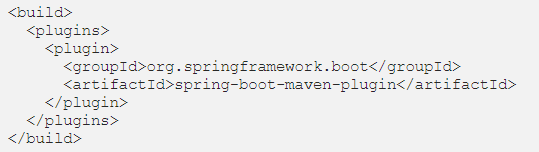
Data + Next possible actions

========================================= **DEPLOYMENT** =========================================

Source : <https://medium.com/swlh/deploying-spring-boot-applications-15e14db25ff0>

1. Deploying in Java Archive (JAR) as a standalone application
2. Deploying as Web Application Archive (WAR) into a servlet container
3. Deploying in Docker Container
4. Deploying behind NGINX web server — direct setup
5. Deploying behind NGINX web server — containerized setup
6. **Deploying in Java Archive (JAR) as a standalone application**

Spring Boot applications can easily be packaged into JAR files and deployed as standalone applications. This is done by the spring-boot-maven-plugin. The plugin is automatically added to pom.xml once the Spring project is created via Spring Initializr as a Maven project.

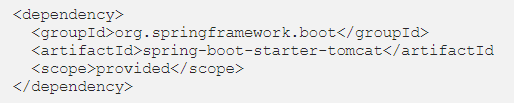


In order to package the application in a single (fat) jar file, run the maven command mvn package under project directory. This will package the application inside an executable jar file with all its dependencies (including the embedded servlet container — if its a web application). To run the jar file, use the following standard JVM command java -jar <jar-file-name>.jar.

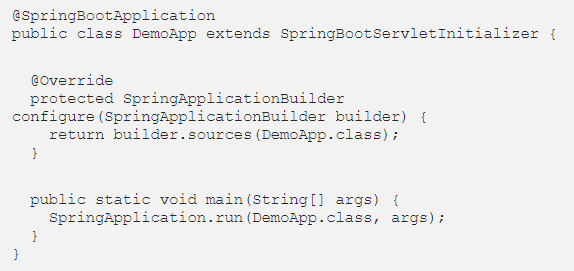
1. **Deploying as Web Application Archive (WAR) into a servlet container**

Spring Boot applications can be packaged into WAR files to be deployed into existing servlet containers (such as Tomcat, Jetty etc.). This can be done as follows:

Specify WAR packaging in pom.xml file via <packaging>war</packaging>. This will package the application into a WAR file (instead of JAR). On the second step, set the scope of Tomcat (servlet container) dependency to provided (so that it is not deployed into WAR file):



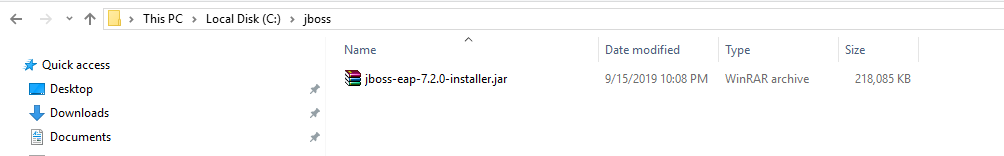
Initialise the Servlet context required by Tomcat by extending SpringBootServletInitializer and overriding configure method as follows:



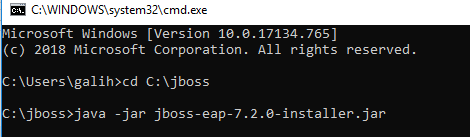
In order to package the application in a war file, run the standard maven command mvn clean package under project directory. This will generate the WAR package which can be deployed into a servlet container. To run the application inside an existing Tomcat container, copy the generated WAR file to tomcat/webapps/ directory.

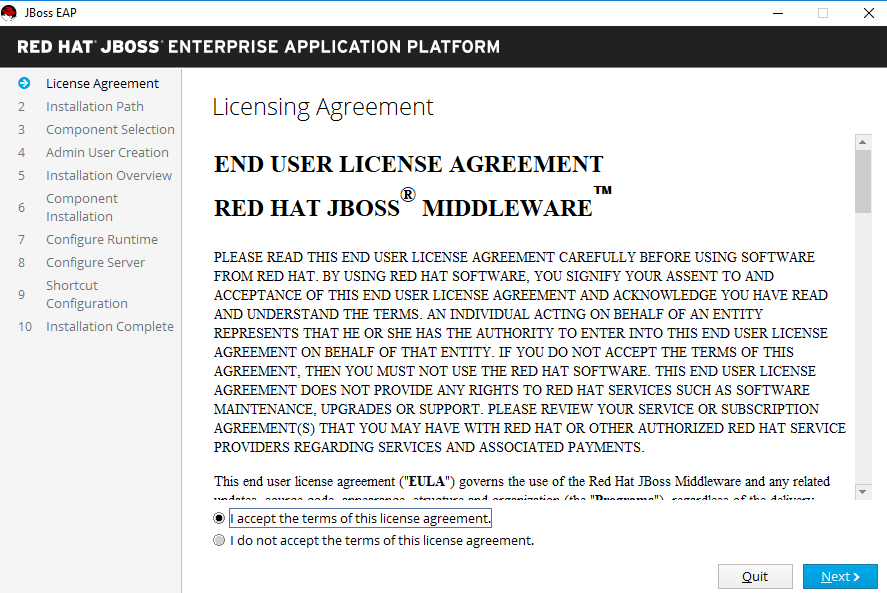
---------------------------------------------------------------------------------------------------------------------------------------------------

Download jboss 🡪 <https://developers.redhat.com/products/eap/download>



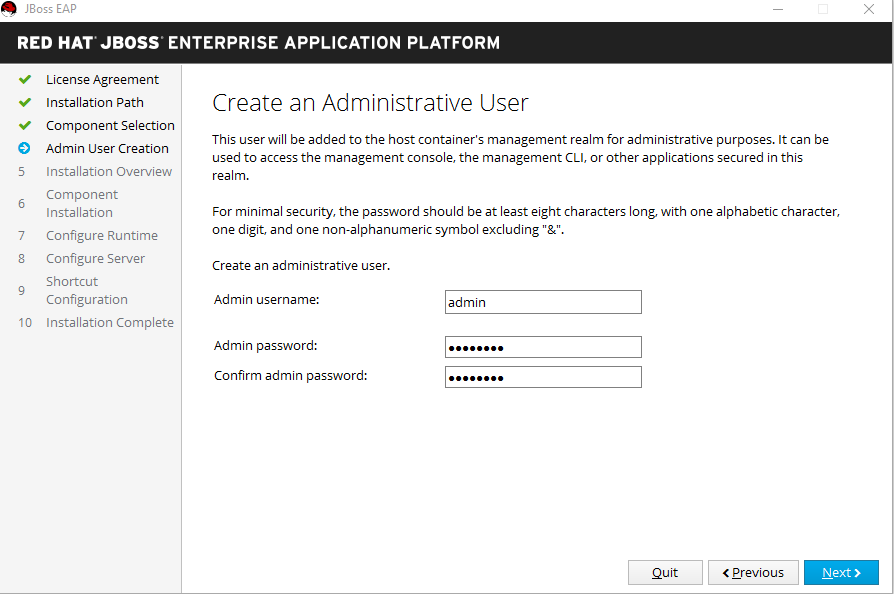
java -jar jboss-eap-7.2.0-installer.jar, lalu muncul dialog box untuk instalasi





Admin username: admin

Admin password: admin123!

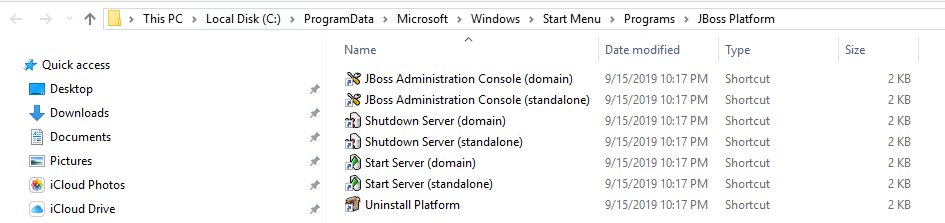


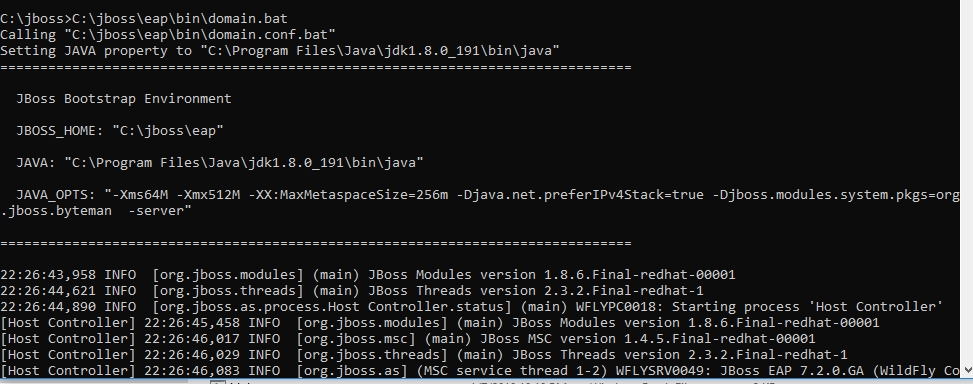
How to start:

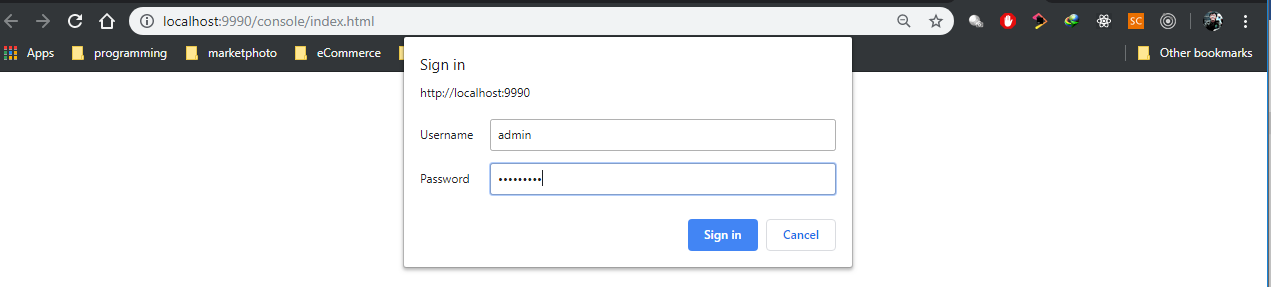
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\JBoss Platform

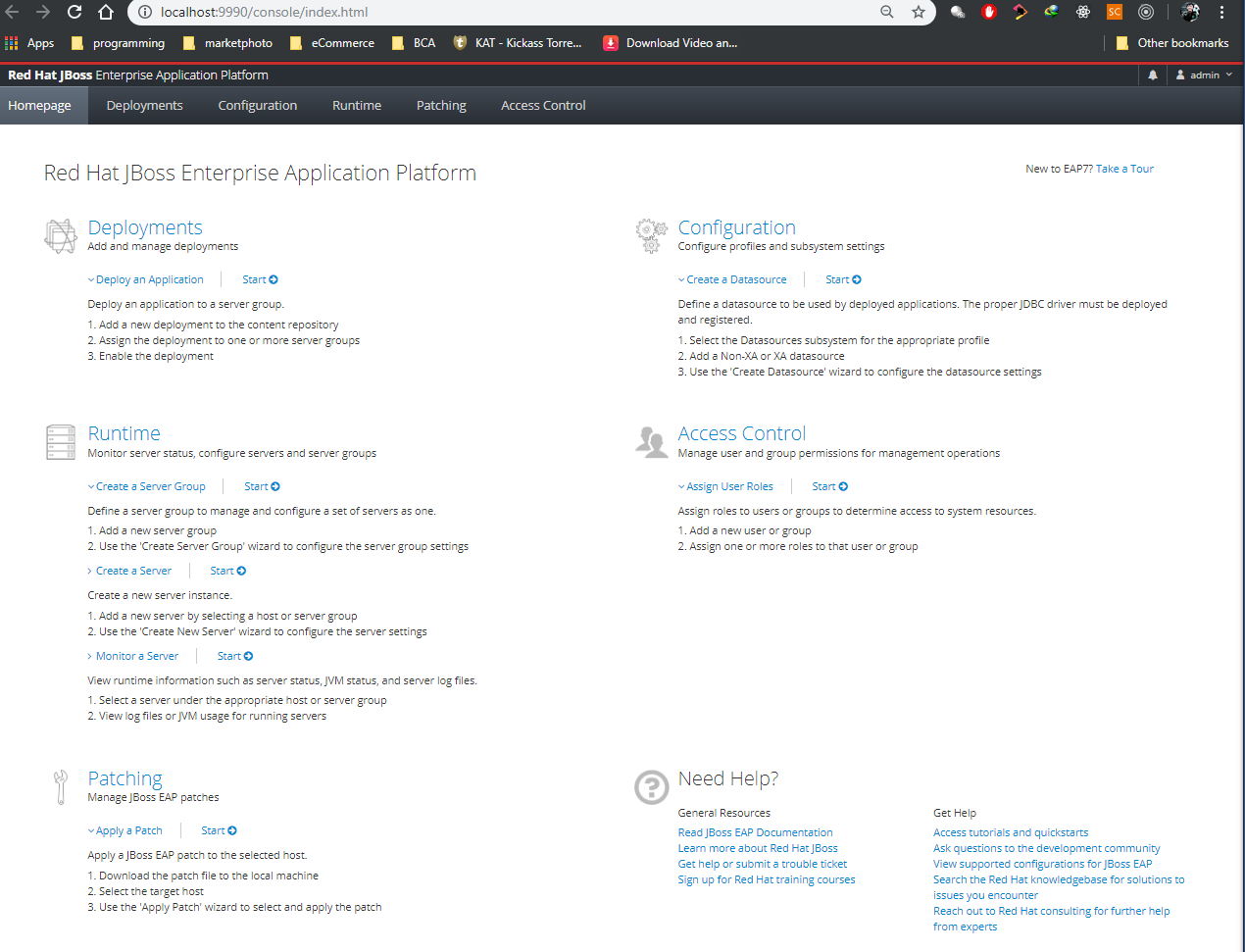
execute 🡪 Start Server (domain) 🡪 C:\jboss\eap\bin\domain.bat

execute 🡪 JBoss Administration Console (domain) 🡪 <http://localhost:9990/console/index.html>





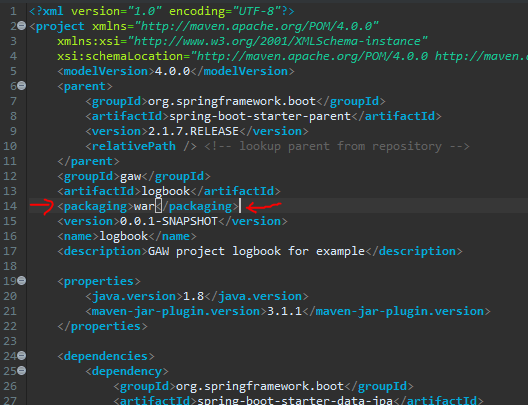


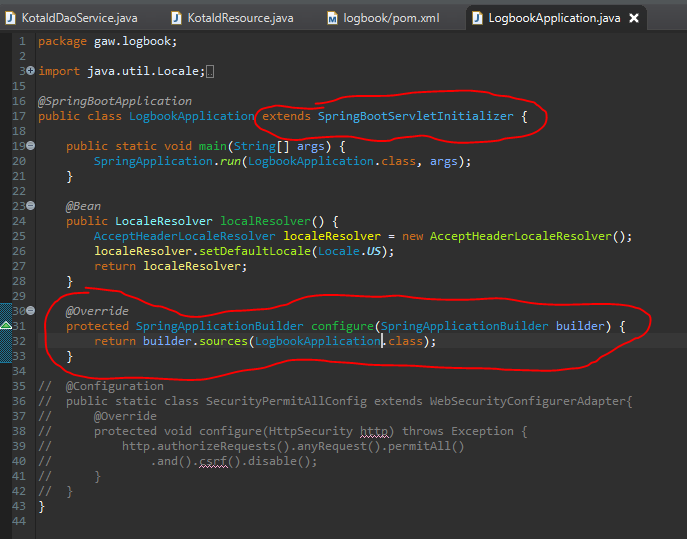


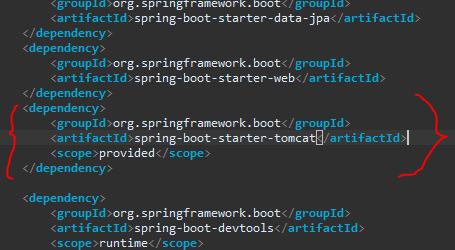
------------------------------------------------------------------------------------------------------------------------------------------------------

**Kembali ke eclipse**

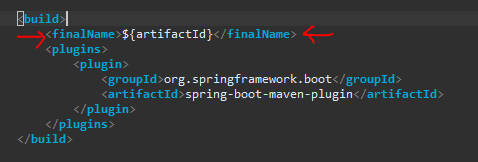
* Tambahkan <**packaging**>war</**packaging**> di pom.xml
* Extend SpringBootServletInitializer & override SpringApplicationBuilder
* Marked the embedded servlet container as provided



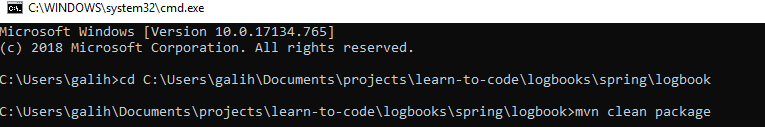




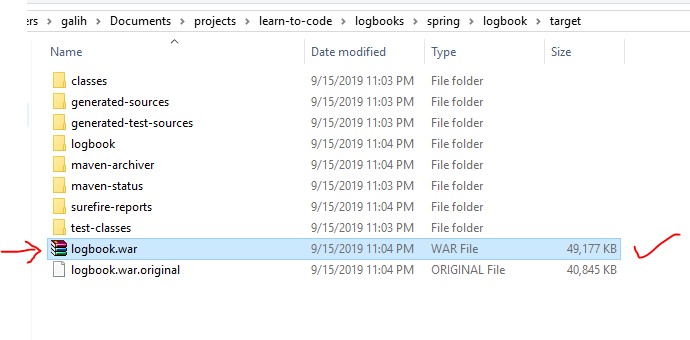
* Tambahan: mari tambahkan finalName biar tak perlu version



mvn clean package

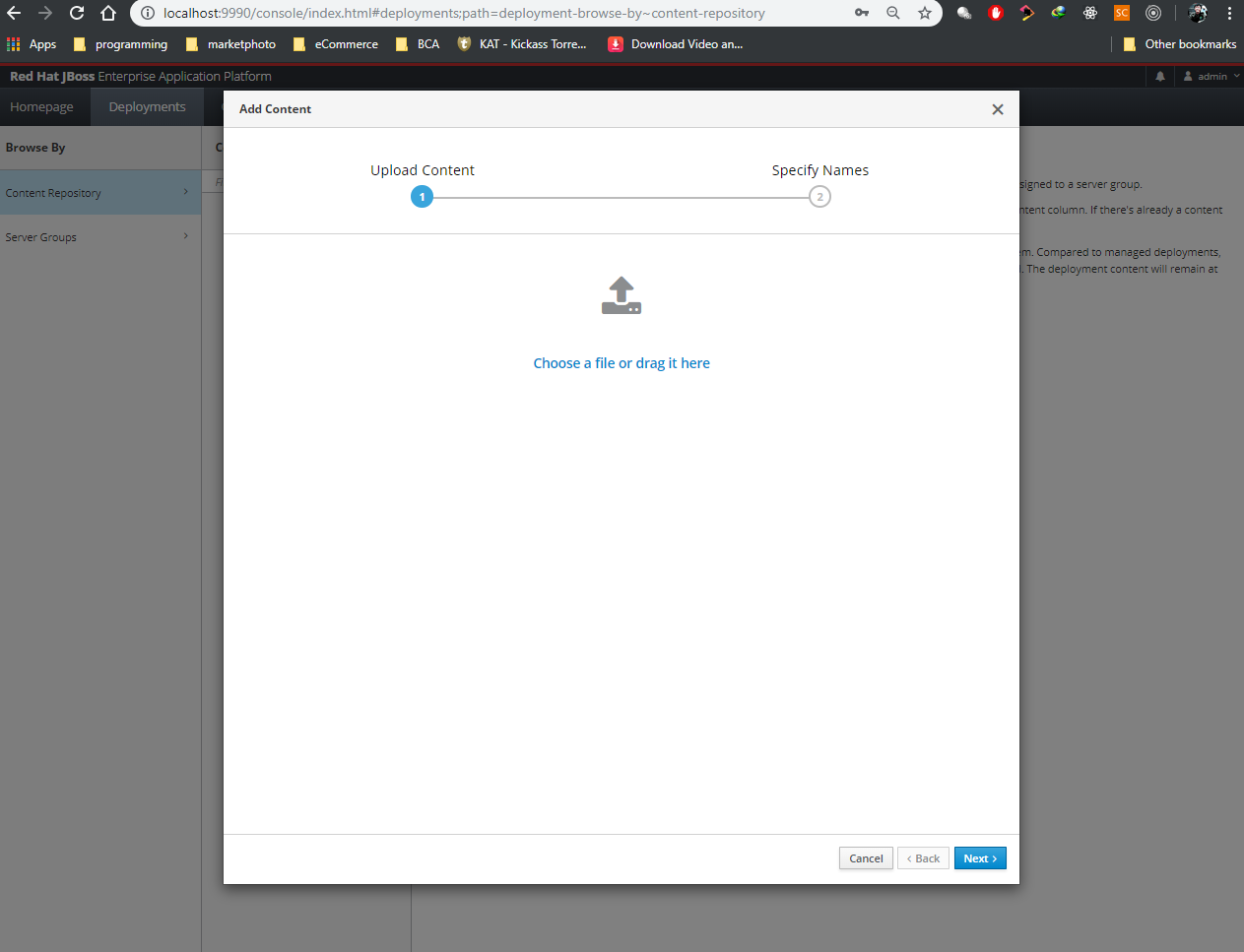


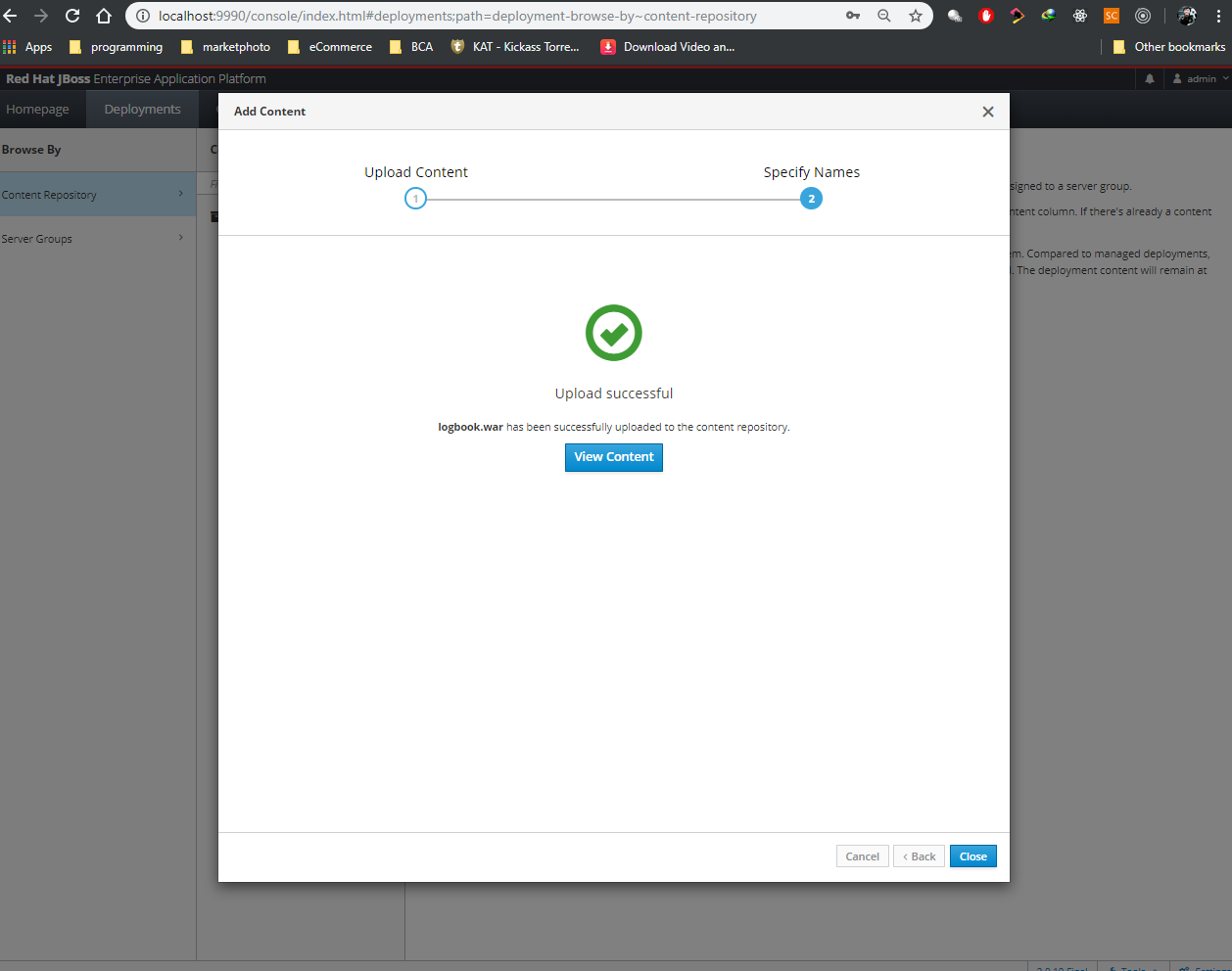
Hasil compile

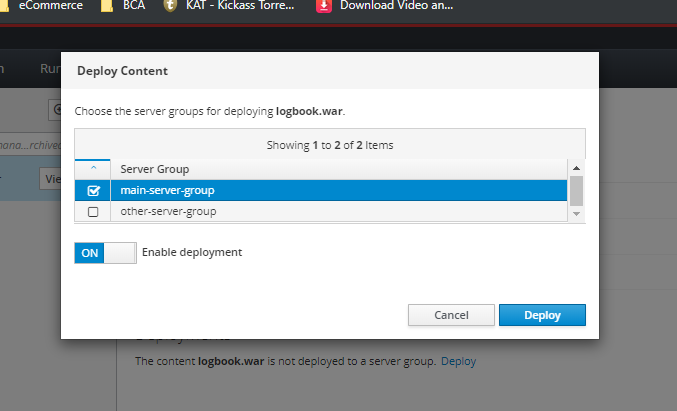


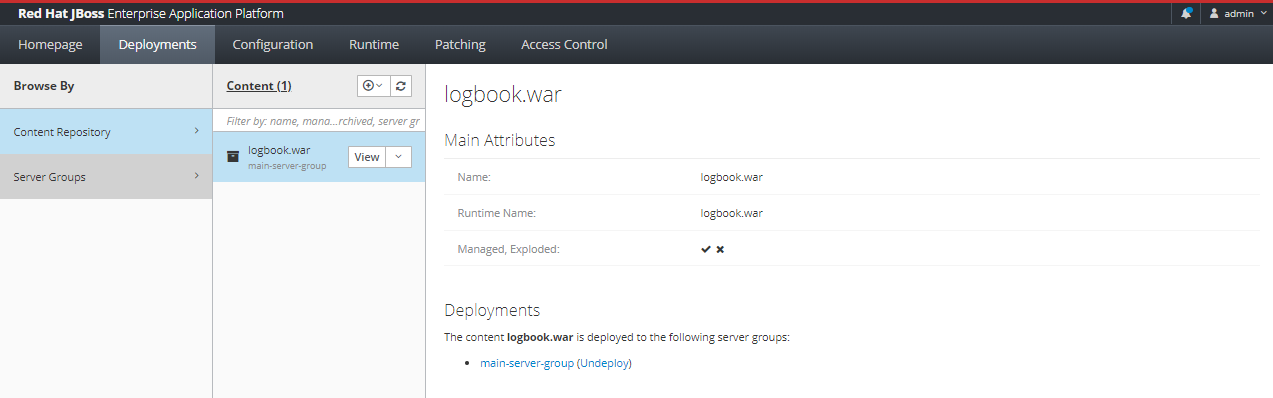
------------------------------------------------------------------------------------------------------------------------------------------------------

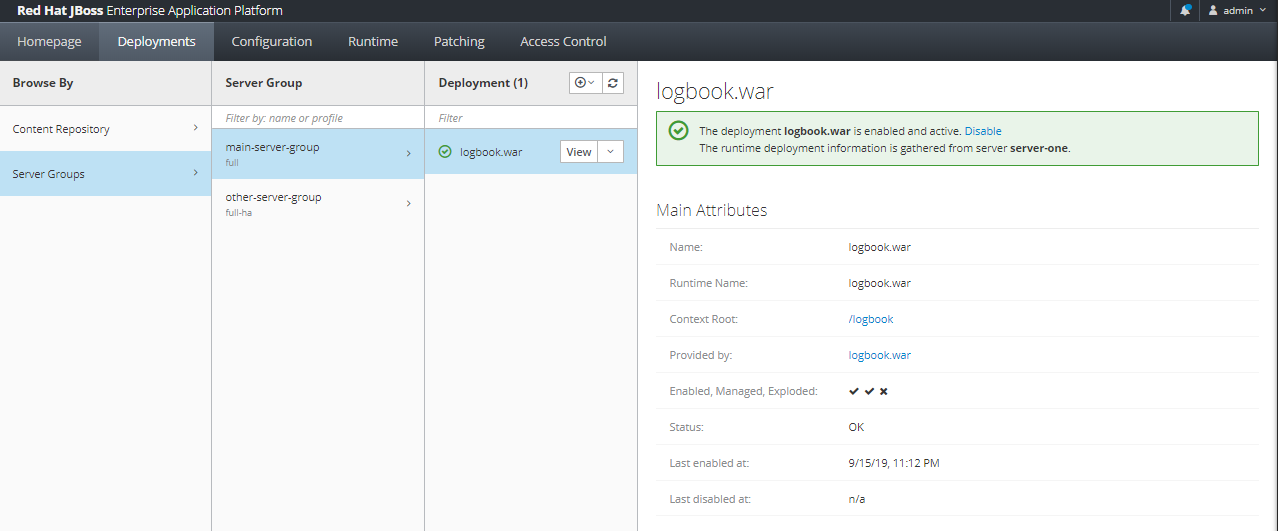
**Deploy logbook.war ke Jboss**



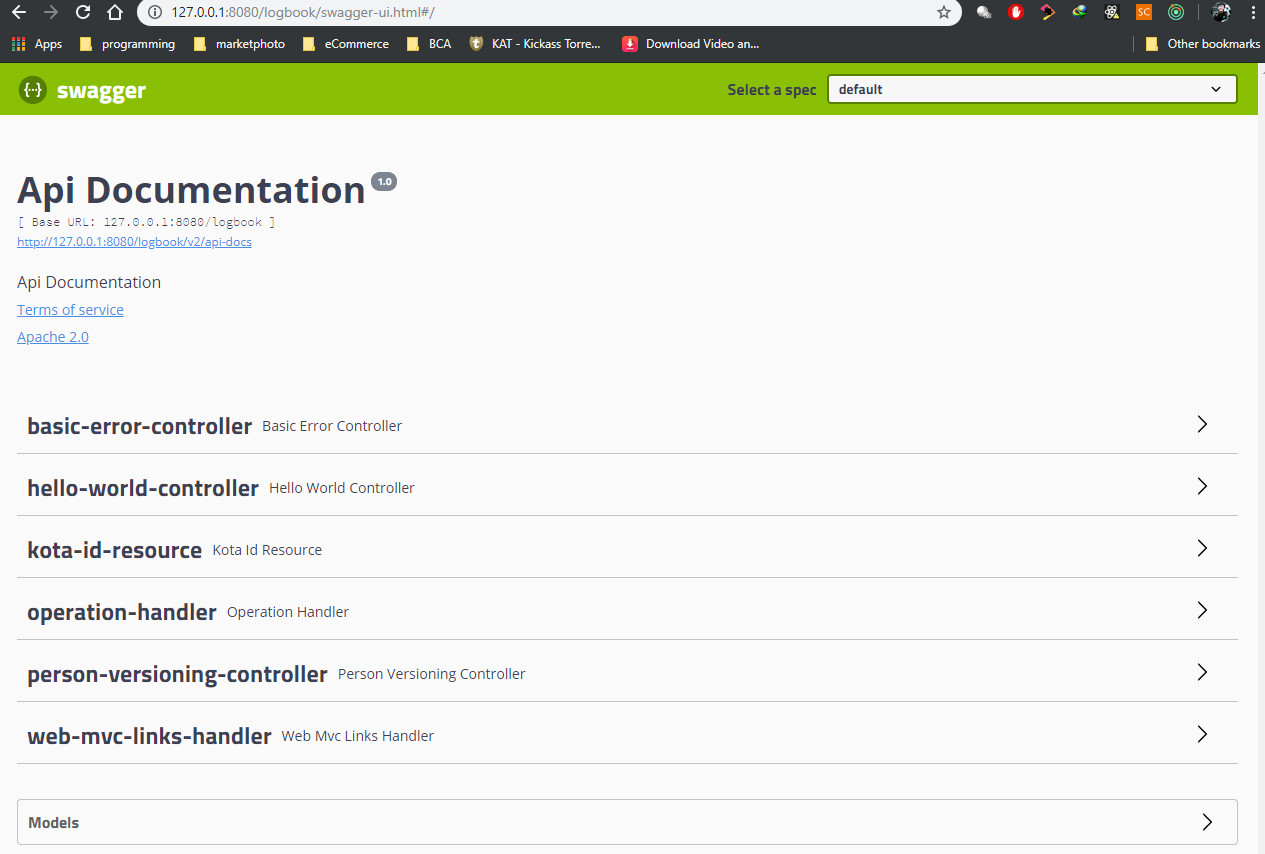




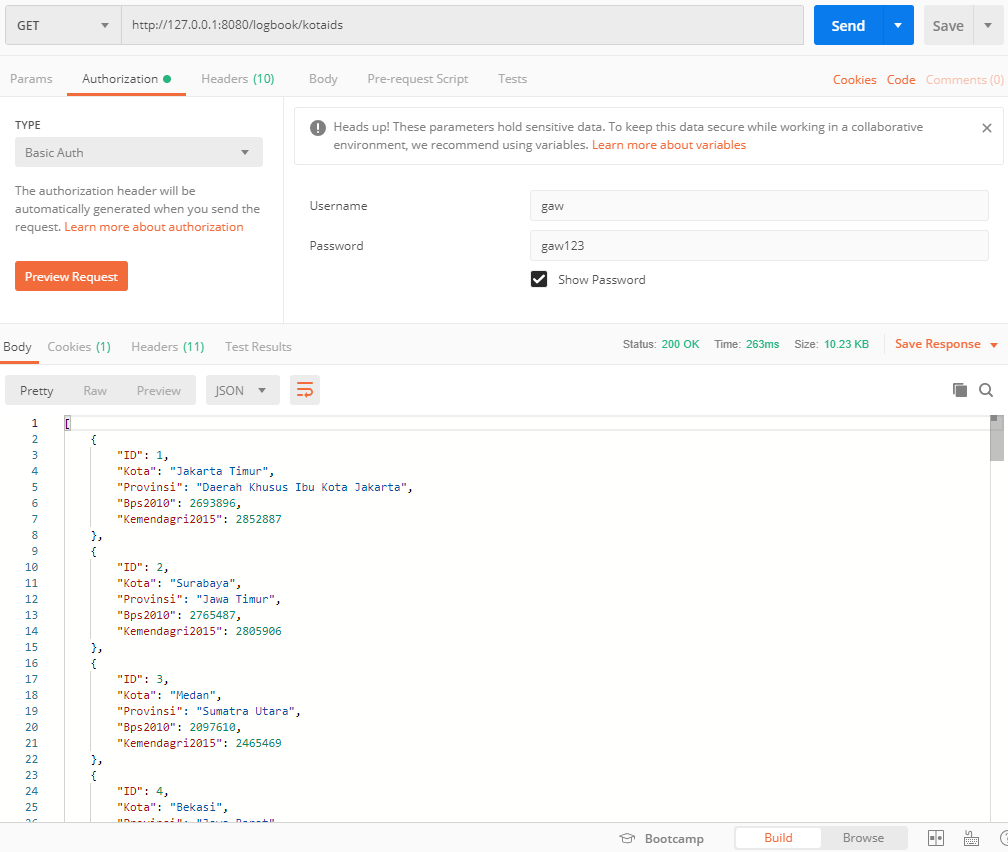




Tes akses swagger 🡪 <http://127.0.0.1:8080/logbook/swagger-ui.html#/>



Tes hit 🡪 GET http://127.0.0.1:8080/logbook/kotaids



1. **Deploying in Docker Container**
2. **Deploying behind NGINX web server — direct setup**

**Load balancing**

1. **Deploying behind NGINX web server — containerized setup**