**خلاصه مسیر پیشنهادی پروژه:**

1. ساخت پروژه Spring Boot
2. راه‌اندازی Keycloak و اتصال به پروژه
3. اضافه کردن JWT با Keycloak
4. تعریف state machine (مثلاً lifecycle سفارش)
5. مدل‌سازی همین فرآیند در Camunda
6. پیاده‌سازی Saga Orchestration با Camunda یا State Machine

اگه موافقی، می‌تونم مرحله‌به‌مرحله همراهیت کنم. دوست داری از کدوم بخش شروع کنیم؟ ساخت پروژه و تنظیم Keycloak؟

عالیه! حالا که پروژه Spring Boot آماده‌ست، بریم سراغ مرحله بعد:

# ✅ مرحله 2: راه‌اندازی Keycloak با Docker

**🎯 هدف:**

* اجرای Keycloak روی سیستم محلی با Docker
* ساخت Realm، Client و کاربر تست
* آماده‌سازی برای اتصال پروژه Spring Boot به Keycloak

**🔧 1. اجرای Keycloak در Docker**

در پوشه‌ای دلخواه، فایل زیر به اسم docker-compose.yml بساز:

version: '3.8'

services:

keycloak:

image: quay.io/keycloak/keycloak:24.0.2

container\_name: keycloak

command: start-dev

ports:

- "8081:8080"

environment:

KEYCLOAK\_ADMIN: admin

KEYCLOAK\_ADMIN\_PASSWORD: admin

و سپس اجرا کن:

docker-compose up -d

این کار Keycloak رو در آدرس زیر بالا میاره:  
[http://localhost:8081](http://localhost:8081/)

با یوزر و پسورد:

* **Username:** admin
* **Password:** admin

**🏗️ 2. پیکربندی اولیه در Keycloak**

حالا داخل پنل ادمین Keycloak (<http://localhost:8081/admin>) وارد شو و مراحل زیر رو انجام بده:

**✅ ساخت Realm جدید:**

1. از منوی بالا سمت چپ، روی "Master" کلیک کن و "Create Realm" رو بزن.
2. اسم مثلاً بذار: demo-realm

**✅ ساخت Client:**

1. داخل Realm، برو به "Clients"
2. روی "Create Client" کلیک کن
3. تنظیمات:
   * Client ID: springboot-client
   * Client Type: **OpenID Connect**
   * Root URL: http://localhost:8080
   * بقیه موارد پیش‌فرض

بعد از ایجاد:

* داخل تنظیمات Client، به تب **"Settings"** برو:
  + Access Type: confidential
  + Standard Flow Enabled: ✅
  + Direct Access Grants Enabled: ✅
  + Valid Redirect URIs: http://localhost:8080/\*
  + Save کن

بعد برو تب **Credentials** و **Client Secret** رو کپی کن. بعداً لازم می‌شه.

**✅ ساخت User تست:**

1. برو به "Users"
2. "Add User" بزن
   * Username: testuser
   * Enabled: ✅
3. بعد از ایجاد، برو به تب **Credentials**
   * Password: testpass
   * Temporary: ❌

**🎯 مرحله بعد:**

وصل کردن پروژه Spring Boot به Keycloak و گرفتن JWT بعد از لاگین.

آیا Keycloak بالا اومد و این موارد (Realm, Client, User) رو تونستی بسازی؟  
اگر بله، بریم سراغ مرحله اتصال Spring Boot به Keycloak.

سؤال بسیار خوبی پرسیدی، چون Keycloak از نسخه 17 به بعد (مخصوصاً نسخه‌های 22 به بالا) رابط کاربری جدیدی داره و بعضی گزینه‌ها **تغییر مکان دادن** یا **نام‌هاشون عوض شده**. در نسخه 24 که الان داری، مواردی مثل Access Type و Credentials دیگه اون شکلی که توی آموزش‌های قدیمی هست دیده نمی‌شن.

اجازه بده سریع راهنمایت کنم:

**🧭 آموزش تنظیم Client در Keycloak 24**

**🧩 قدم 1: ساخت Client**

1. وارد Realm دلخواهت شو (مثل demo-realm)
2. از منوی سمت چپ روی **Clients** کلیک کن
3. روی **Create client** کلیک کن

تنظیمات اولیه:

| **گزینه** | **مقدار** |
| --- | --- |
| Client ID | springboot-client |
| Client type | OpenID Connect |
| Root URL (اختیاری) | http://localhost:8080 |

روی **Next** کلیک کن.

**🧩 قدم 2: تنظیم دسترسی‌ها**

در صفحه بعدی (یا بعد از ساخت)، این تنظیمات رو انجام بده:

1. در تب **Capability config**:
   * تیک **Client authentication** رو فعال کن ✅ (این همون "Access Type: confidential" قدیمیه)
   * باقی موارد رو پیش‌فرض بذار
2. در تب **Credentials**:
   * این تب الان جزو منوی سمت چپ داخل صفحه client هست، نه تب جدا
   * برو به منوی سمت چپ همون صفحه و روی **Credentials** کلیک کن
   * اونجا **Client secret** رو می‌بینی و می‌تونی کپی کنی

lGx26eb1HeX3LYWubUyR2qLrWVK6ucOv

**🧩 قدم 3: تنظیم Redirect URI**

1. در تب **Settings** یا همون صفحه اصلی client:
   * برو پایین تا بخش **Valid redirect URIs**
   * مقدار زیر رو وارد کن:
   * http://localhost:8080/\*
   * Save رو بزن

اگه دوست داشتی، می‌تونم یه اسکرین‌شات از مسیر دقیق توی رابط جدید Keycloak برات طراحی کنم.

⬇️ وقتی این مرحله‌ها رو انجام دادی و client secret رو گرفتی، بگو تا بریم سراغ اتصال Spring Boot به Keycloak و گرفتن JWT.

خیلی خوبه 👌 حالا می‌ریم سراغ:

# ✅ مرحله 3: اتصال Spring Boot به Keycloak

**🎯 هدف:**

* پیکربندی Spring Boot برای احراز هویت با Keycloak
* محافظت از endpointها با توکن JWT

**🧩 1. افزودن وابستگی‌های Keycloak و امنیت**

در pom.xml این dependencyها رو اضافه کن:

<dependencies>

<!-- سایر وابستگی‌ها -->

<!-- Spring Security OIDC -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>

</dependency>

<!-- استفاده از JWT -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- برای خواندن مقادیر در تنظیمات -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

**🧩 2. پیکربندی در application.yml (یا application.properties)**

ترجیحاً فایل application.yml بساز و تنظیمات زیر رو بذار:

spring.application.name=demo

spring.datasource.url=jdbc:mysql://localhost:3306/demo\_db?useSSL=false&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=Bahman2462

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

server.port= 8082

spring.security.oauth2.resourceserver.jwt.issuer-uri=http://localhost:8081/realms/demo-realm

spring.security.oauth2.resourceserver.jwt.jwk-set-uri=http://localhost:8081/realms/demo-realm/protocol/openid-connect/certs

اگر فقط issuer-uri رو بدی، Spring Boot خودش کلید عمومی Realm رو از Keycloak می‌گیره.

**🧩 3. ساخت کنترلر تست**

مثلاً یک HelloController برای تست احراز هویت:

package com.example.demo;

import java.security.Principal;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/api")

public class HelloController {

    @GetMapping("/public")

    public String publicHello() {

        return "Hello Public!";

    }

    @GetMapping("/private")

    public String privateHello(Principal principal) {

        return "Hello Private, " + principal.getName();

    }

}

**🧩 4. تنظیم امنیت HTTP**

کلاسی مثل زیر بساز (مثلاً SecurityConfig):

package com.example.demo;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Bean

    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http

            .authorizeHttpRequests(authz -> authz

                .requestMatchers("/api/public").permitAll()   // آزاد

                .requestMatchers("/api/private").authenticated() // نیازمند لاگین

                .anyRequest().permitAll()

            )

            .oauth2ResourceServer(oauth2 -> oauth2

                .jwt()

            );

        return http.build();

    }

}

بایستی کلاس امنیتی داخل پکیج controller باشه و کنار هم باشن

**🧪 5. تست با توکن Keycloak**

**بعدش پروژه رو اجرا میکنیم:**

* + 1. **نصب وابستگی های جدید Po**
       1. **mvnw.cmd clean install**
    2. **اجرای پروژه بعد از اصلاح و بیلد**
       1. **mvnw.cmd spring-boot:run**

برای گرفتن توکن، از Postman یا curl استفاده کن:

POST http://localhost:8081/realms/demo-realm/protocol/openid-connect/token

Content-Type: application/x-www-form-urlencoded

client\_id=springboot-client

client\_secret=<<< client secret >>>

username=testuser

password=testpass

grant\_type=password

خروجی:

{

"access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",

...

}

توکن رو در درخواست به /api/private بفرست:

GET http://localhost:8082/api/private

Authorization: Bearer eyJhbGciOi...

باید جواب بده:

Hello Private, testuser

✅ مرحله بعدی؟  
می‌تونیم بریم سراغ:

* تعریف Roleها در Keycloak و دسترسی مبتنی بر نقش
* یا
* شروع Camunda و اتصال به پروژه

# مرحله 4

منطقیه که اول Keycloak رو کامل و اصولی راه بندازی، بعد بریم سراغ Camunda، State Machine، Saga و بقیه موارد. چون هویت و سطوح دسترسی پایه بسیاری از چیزهای بعدی هست.

**🔜 مرحله بعدی: تعریف Roleها و اعمال دسترسی بر اساس نقش در Keycloak + Spring Boot**

**✅ فازهای بعدی Keycloak که با هم انجام می‌دیم:**

1. **تعریف Roleها در Realm و تخصیص به کاربران**
2. اعمال **دسترسی‌ها در Spring Security** بر اساس نقش
3. استخراج نقش‌ها از توکن JWT
4. تعریف endpointهای مختلف با نقش‌های مختلف (مثلاً فقط Admin یا فقط User)
5. ساختن یک client با role-based access (برای frontend یا microservice)
6. تست با Postman با نقش‌های مختلف

**📌 گام بعد: تعریف Role در Keycloak**

اگر موافقی، از همین الان شروع کنیم:

**✅ گام اول – تعریف Role:**

1. وارد ادمین کنسول Keycloak شو (<http://localhost:8081/>)
2. وارد Realm خودت شو (demo-realm)
3. از منوی سمت چپ، برو به **Roles**
4. روی **Create Role** کلیک کن
   * **Role name**: مثلا ROLE\_ADMIN
   * ذخیره کن
   * دوباره یکی دیگه با نام مثلاً ROLE\_USER بساز

**✅ گام دوم – تخصیص Role به کاربر:**

1. از منوی سمت چپ برو به **Users**
2. روی testuser کلیک کن
3. برو به تب **Role Mappings**
4. در قسمت **Available Roles**:
   * ROLE\_USER رو انتخاب کن و به سمت راست (Assigned roles) ببر

اگر این کارها رو انجام دادی، بگو تا بریم سراغ پیکربندی Spring Security که نقش‌ها رو بخونه و ازش استفاده کنه برای کنترل دسترسی.

# مرحله 5

**✅ مرحله دوم: استفاده از Roleهای Keycloak در Spring Boot (Role-Based Access Control)**

**🎯 هدف:**

* بخونیم نقش‌ها (Roles) رو از JWT
* و از اون‌ها برای محدود کردن دسترسی به endpointها استفاده کنیم (مثلاً فقط ROLE\_ADMIN بتونه یک مسیر خاص رو ببینه)

**🧩 1. نقش‌ها در Keycloak کجای توکن JWT میان؟**

Keycloak توکن JWT رو به شکل زیر تولید می‌کنه:

{

...

"realm\_access": {

"roles": [

"ROLE\_USER",

"ROLE\_ADMIN"

]

},

...

}

ما باید Spring رو طوری پیکربندی کنیم که این قسمت رو بخونه.

**🛠️ 2. پیکربندی خواندن نقش‌ها از realm\_access.roles**

**✅ ایجاد کلاس KeycloakRoleConverter**

**یک کلاس ساده برای استخراج نقش‌ها**

**🧩 3. اتصال KeycloakRoleConverter به Spring Security**

در SecurityConfig.java به این شکل اضافه‌اش کن:

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Bean

    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http

            .authorizeHttpRequests(authz -> authz

                .requestMatchers("/api/public").permitAll()

                .requestMatchers("/api/private").authenticated()

                .requestMatchers("/api/admin").hasRole("ADMIN")

                .anyRequest().permitAll()

            )

            .oauth2ResourceServer(oauth2 -> oauth2

                .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthenticationConverter()))

            );

        return http.build();

    }

    @Bean

    public JwtAuthenticationConverter jwtAuthenticationConverter() {

        JwtAuthenticationConverter converter = new JwtAuthenticationConverter();

        converter.setJwtGrantedAuthoritiesConverter(new KeycloakRoleConverter());

        return converter;

    }

}

**🧪 4. تست endpoint با Role**

فرض کن این endpoint فقط برای ROLE\_ADMIN باز باشه:

@GetMapping("/api/admin")

public ResponseEntity<String> admin() {

return ResponseEntity.ok("Admin content");

}

**🧪 تست با Postman**

1. با کاربری که فقط ROLE\_USER داره، توکن بگیر → 401 Unauthorized
2. با کاربری که ROLE\_ADMIN داره، توکن بگیر → موفق

اگر تا اینجا پیاده‌سازی کردی، بهم بگو تا برایت:

* راهنمایی ساخت کاربر admin در Keycloak
* و تست درستش رو بفرستم.

بریم جلو؟ ✅

**📦 ساختار پوشه‌هات:**

src/

└── main/

└── java/

└── com/

└── example/

├── demo/

│ └── SecurityConfig.java

└── KeycloakRoleConverter.java ✅ این فایله

# مرحله 6

عالیه! 👏 یعنی تا اینجا با موفقیت انجام دادی:

✅ اتصال Spring Boot به Keycloak  
✅ تعریف Endpointهای عمومی و خصوصی  
✅ تعریف نقش‌ها در Keycloak  
✅ گرفتن Token از Keycloak و ارسال با Postman  
✅ محافظت از APIها بر اساس نقش (hasRole("ADMIN"))

**🎯 مرحله بعد: انتخاب با تو**

حالا می‌تونیم به یکی از مراحل زیر وارد بشیم:

**🔄 ۱. اتصال Keycloak به Camunda**

* استفاده از همون Keycloak برای احراز هویت کاربران در Camunda
* بررسی اینکه چطور Roleهای Camunda رو از طریق Keycloak مدیریت کنیم

**🧩 ۲. راه‌اندازی Camunda در پروژه**

* نصب و اجرای Camunda (با Docker یا embedded)
* اجرای اولین Process Model ساده
* REST APIهای Camunda و تست با Postman

**🎛 ۳. پیاده‌سازی یک State Machine یا Saga (مرحله‌های بعدی‌تر)**

بیایم مرحله اول یعنی **راه‌اندازی مستقل Camunda** رو شروع کنیم. در این مرحله:

**🔹 سوال: شما کدوم نسخه Camunda رو مدنظر دارید؟**

* **Camunda 7** (جاوا مبتنی بر Servlet/Embedded و UI ساده‌تر)
* **Camunda 8** (معماری مبتنی بر Zeebe، microservices-friendly و نیازمند بیشتر منابع)

اگر برای شروع با Camunda آشنا نیستید یا هدف یادگیری مفاهیم اصلیه، پیشنهاد من: **Camunda 7**

کدوم رو ترجیح می‌دی؟

* ✅ docker-compose با Camunda 7
* ⬜ اجرای دستی فایل WAR در Tomcat

# مرحله 7 – راه اندازی Camunda

عالیه، این تقسیم‌بندی کاملاً منطقیه. بیایم مرحله اول یعنی **راه‌اندازی مستقل Camunda** رو شروع کنیم. در این مرحله:

**✅ مرحله ۱: راه‌اندازی Camunda به‌صورت مستقل**

**مقایسه سریع دو گزینه‌ی اجرایی:**

| **روش** | **مزایا** | **معایب** |
| --- | --- | --- |
| ✅ **Docker Compose** (پیشنهاد من) | سریع‌ترین راه، بدون نیاز به نصب Tomcat یا تنظیم محیط دستی | نیاز به Docker نصب‌شده (که شما داری 👍) |
| ⬜ اجرای فایل WAR در Tomcat | اگر بخوای مستقیماً روی یک سرور Java اجرا کنی مفیده | نیاز به تنظیم دستی Tomcat، deploy فایل WAR، و پیکربندی JVM |

**✅ نتیجه: Docker Compose بهترین گزینه برای یادگیری و راه‌اندازی سریع هست.**

بریم سراغ راه‌اندازی Camunda 7 با Docker Compose:

**⚙️ مرحله راه‌اندازی Camunda 7 با Docker Compose**

۱. یک فایل docker-compose.yml بساز با محتوای زیر:

version: '3.7'

services:

camunda:

image: camunda/camunda-bpm-platform:run-latest

ports:

- "8083:8083"

۲. در پوشه‌ای که این فایل هست، اجرا کن:

docker compose up -d

۳. سپس در مرورگر باز کن:

http://localhost:8083

* یوزرنیم پیش‌فرض: demo
* پسورد پیش‌فرض: demo

✅ وقتی تونستی بالا بیاری، بهم بگو تا بریم سراغ اتصال Camunda به پروژه جاوای خودت (مرحله ۲).

اگه خواستی به جای run-latest از نسخه خاصی استفاده کنی (مثلاً 7.20.0)، می‌تونم نسخه دقیق‌تر رو هم بهت بگم.

نکته‌ی دقیقی رو اشاره کردی، و مشکلت هم واضح مشخصه:

**🔴 مشکل:**

در فایل docker-compose.yml نوشتی:

ports:

- "8083:8083"

اما تصویر Docker camunda/camunda-bpm-platform:run-latest، به‌صورت پیش‌فرض **سرویس رو روی پورت 8080 داخل کانتینر اجرا می‌کنه**.

**✅ راه‌حل:**

باید پورت **داخلی کانتینر (8080)** رو به **پورت دلخواه روی سیستم (مثلاً 8083)** نگاشت کنی. یعنی باید بنویسی:

ports:

- "8083:8080"

فرمت نگاشت پورت در Docker Compose:

"پورت\_لوکال:پورت\_کانتینر"

**🔁 پس فایل درست:**

version: '3.7'

services:

camunda:

image: camunda/camunda-bpm-platform:run-latest

ports:

- "8083:8080"

**⏭ مراحل بعد از اصلاح:**

1. فایل رو ذخیره کن
2. کانتینر قبلی رو stop و remove کن:

docker compose down

1. مجدد بالا بیار:

docker compose up -d

1. در مرورگر باز کن:

http://localhost:8083

باید صفحه مدیریت Camunda با یوزر demo و پسورد demo بیاد بالا.  
وقتی آماده شدی، بریم مرحله بعد: **اتصال پروژه جاوا به Camunda**.

# مرحله 8 –

**🚀 مرحله بعد: اتصال پروژه Spring Boot به Camunda**

برای اینکه پروژه‌ت بتونه با Camunda تعامل کنه (مثل deploy کردن فرایند BPMN یا شروع کردن process instance)، دو مسیر وجود داره:

**🔹 مسیر ۱: استفاده از REST API Camunda**

اگر فقط می‌خوای از بیرون با Camunda تعامل داشته باشی (مثلاً از پروژه‌ات فایل BPMN بفرستی، یا یک process instance رو با POST راه‌اندازی کنی)، نیازی به اضافه کردن dependency داخل پروژه نیست، فقط:

* از آدرس‌های REST استفاده می‌کنی مثل:
  + POST http://localhost:8083/engine-rest/process-definition/key/{key}/start
  + GET http://localhost:8083/engine-rest/task
* این روش سبک‌تر و decoupled هست.

**🔹 مسیر ۲: ادغام داخلی Camunda در پروژه Spring Boot**

اگر می‌خوای:

* داخل پروژه‌ی Spring Boot:
  + BPMN تعریف و deploy کنی
  + Delegate و Service Task بنویسی (کدهای Java برای Taskها)
  + مستقیماً فرآیندها رو اجرا کنی یا با Spring Bean ارتباط داشته باشن

باید Camunda رو به پروژه اضافه کنیم.

**✅ پیشنهاد من:**

فعلاً بریم سراغ مسیر ۲ (ادغام در Spring Boot) تا بتونی فرآیند بنویسی، تست کنی و به keycloak هم وصلش کنیم.

**🧱 مرحله بعد: اضافه کردن Camunda به پروژه**

بیا ابتدا Camunda رو به پروژه اضافه کنیم. فرض می‌کنم پروژه‌ی Spring Boot با Maven هست.

**1. اضافه کردن dependency به pom.xml:**

<dependency>

<groupId>org.camunda.bpm.springboot</groupId>

<artifactId>camunda-bpm-spring-boot-starter</artifactId>

<version>7.21.0</version>

</dependency>

**2. فایل application.properties یا application.yml رو ساده نگه دار:**

spring.datasource.url=jdbc:h2:mem:camunda;DB\_CLOSE\_DELAY=1000

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

camunda.bpm.admin-user.id=demo

camunda.bpm.admin-user.password=demo

camunda.bpm.admin-user.first-name=Demo

(بعداً دیتابیس رو می‌تونیم بذاریم روی MySQL یا PostgreSQL)

**3. تست ساده با فایل BPMN**

در مسیر src/main/resources یک فایل ساده process.bpmn بذار که یه Start Event، یک User Task، و یک End Event داشته باشه.

<?xml version="1.0" encoding="UTF-8"?>

<bpmn:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"

xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"

xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"

id="Definitions\_1"

targetNamespace="http://camunda.org/examples">

<bpmn:process id="simple\_process" name="Simple Process" isExecutable="true">

<bpmn:startEvent id="startEvent" name="Start">

<bpmn:outgoing>flow1</bpmn:outgoing>

</bpmn:startEvent>

<bpmn:userTask id="userTask" name="Do something">

<bpmn:incoming>flow1</bpmn:incoming>

<bpmn:outgoing>flow2</bpmn:outgoing>

</bpmn:userTask>

<bpmn:endEvent id="endEvent" name="End">

<bpmn:incoming>flow2</bpmn:incoming>

</bpmn:endEvent>

<bpmn:sequenceFlow id="flow1" sourceRef="startEvent" targetRef="userTask"/>

<bpmn:sequenceFlow id="flow2" sourceRef="userTask" targetRef="endEvent"/>

</bpmn:process>

<bpmndi:BPMNDiagram id="BPMNDiagram\_1">

<bpmndi:BPMNPlane id="BPMNPlane\_1" bpmnElement="simple\_process">

<!-- Optional visual layout: حذف شده برای سادگی -->

</bpmndi:BPMNPlane>

</bpmndi:BPMNDiagram>

</bpmn:definitions>

بعد از گذاشتن این فایل:

1. پروژه رو اجرا کن.
2. اگر همه چیز درست باشه، این فرآیند به صورت خودکار deploy میشه.
3. می‌تونی از طریق Camunda Cockpit (در آدرس: <http://localhost:8083/camunda>) بری و بررسیش کنی.