* Spring Boot nu suporta in mod default JSP.Asta e din cauza ca embeded TomCat adaugat automat in proiect nu suporta JSP, De aceea, trebuie importate niste dependente:

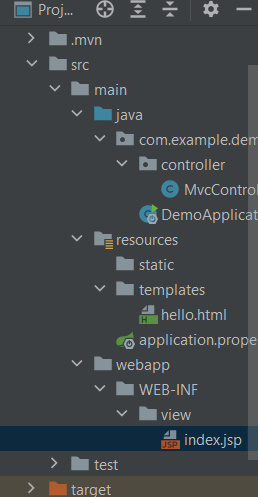
<dependency>  
 <groupId>org.apache.tomcat.embed</groupId>  
 <artifactId>tomcat-embed-jasper</artifactId>  
</dependency>

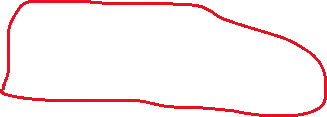
<dependency>  
 <groupId>jakarta.servlet.jsp.jstl</groupId>  
 <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>  
</dependency>

<dependency>  
 <groupId>org.glassfish.web</groupId>  
 <artifactId>jakarta.servlet.jsp.jstl</artifactId>  
 <version>3.0.0</version>  
</dependency>

Asa, se vor citi paginile .jsp si chiar se va putea crea un folder webapp care va fi luat in seama de Spring Boot si va fi pus in classpath

* Conventional, paginile JSP sunt puse in src/main/webapp/WEB-INF si daca vrem mai cream in WEB-INF un view page. Exact asa facem si aici:





E necesar sa facem asa, deoarece JSP se conduc dupa vechiul principiu, ca ele se afla in /src/main/webapp/WEB-INF/ si orice file folosit in ele va fi cautat in webapp by default si spring MVC face la fel. **Daca punem .jsp altundeva decat webapp, nu vor putea fi citite!!!**

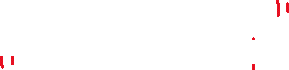
* **Deoarece avem embeded TOMCAT, Spring Boot se ocupa de gasirea,citirea si trimiterea paginilor catre Tomcat. Problema e ca Spring Boot nu suporta pagini JSP si nici nu le poate citi, dar dependentele de mai sus vor face ca Spring Boot sa il lase pe TomCat sa se ocupe de paginile web, dar tomcat le cauta mereu in webapp.**

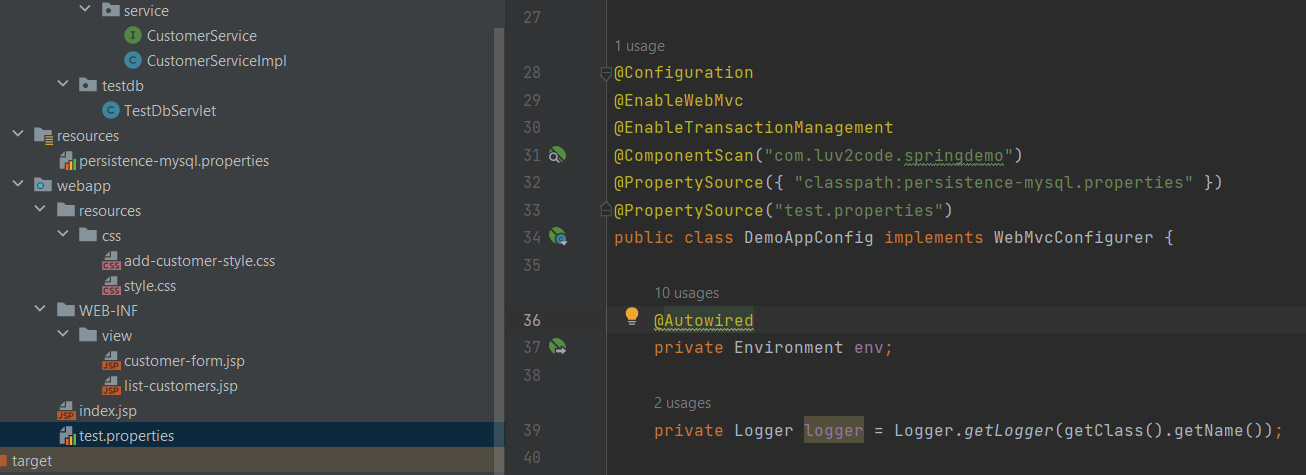
Apoi:

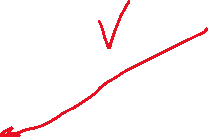
@SpringBootApplication  
public class JspApplication implements WebMvcConfigurer {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(JspApplication.class, args);  
 }  
  
 @Bean  
 public ViewResolver viewResolver(){  
 InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();  
 viewResolver.setPrefix("/WEB-INF/view/");  
 viewResolver.setSuffix(".jsp");  
  
 return viewResolver;  
 }  
   
 @Override  
 public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler("/resources/\*\*").addResourceLocations("/WEB-INF/resources/");  
 }  
}

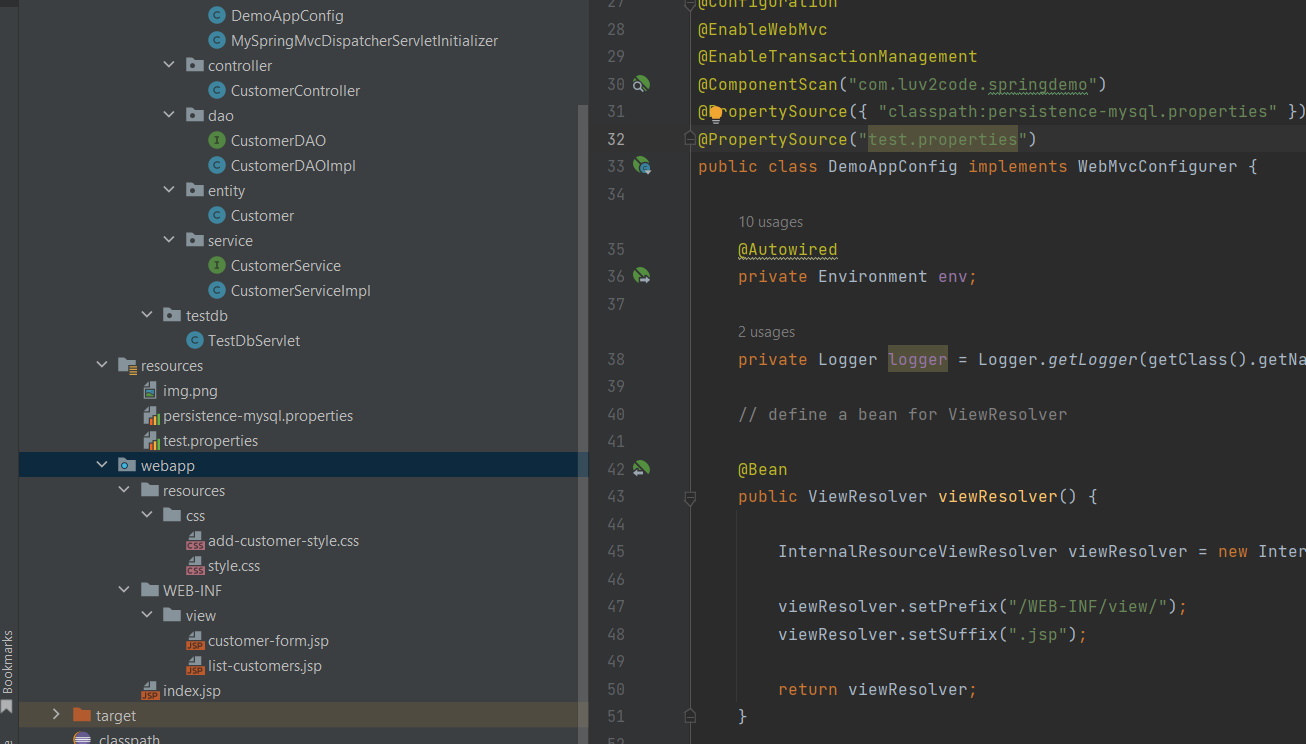
**Maven in Spring simplu**

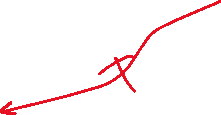
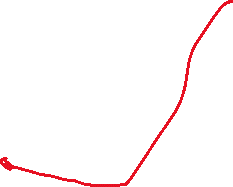
Cand folosim Spring simplu, fara Spring Boot, Maven va crea o asa structura:



* folderul java are acces doar la classpath, si de aceea codul din el mereu se duce si cauta o resursa in folderul java, si apoi in folderul resources. Are referinta la ambele automat, din claspath, dar nu poate nicidecum sa acceseze webapp folder! Acolo se afla configuratii importante ale webaplicatie ce nu trebuie accesate.Atentie! **Anotatiile ca @PropertySource pot accesa liber ce e si in webapp!Deci, intai cauta in resources si apoi in webapp**
* ****



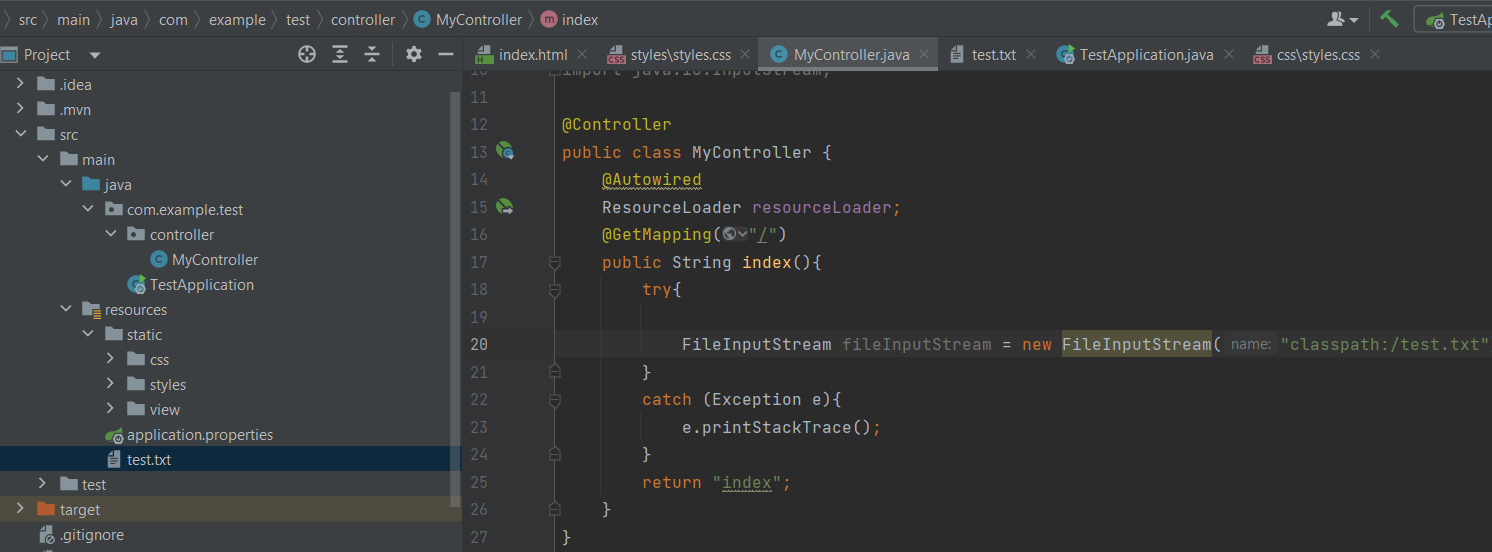
* **Atentie, daca nu folosim bara /, facem mereu trimitere doar la webapp!**
* ****

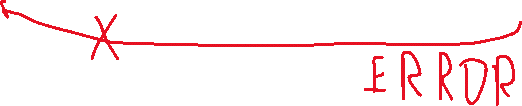


* resources in schimb, stie ca in el se pot gasi configuratii xml ale aplicatiei ce s-ar putea sa aiba nevoie sa accese chestii din webapp, de aceea el automat are referinta la webapp folder. Deci, daca la resursa adaugata nu vom scrie classpath: ea va fi cautata in webapp, daca scriem classpath: se va cauta automat in java si tot in el, in resources
* webapp poate accesa classpath doar prin classpath:

**In spring Boot**

* In Spring boot e cam la fel. La fel exista classpath si il putem folosi la fel prin “classpath:”
* **Dar exista o deosebire, si anume ca java code nu citeste automat fisierele din resources!**





* Cand adaugam si webapp folder, cu dependentele necesare, el nu va fi adaugat in classpath:, si codul java din java folder nu il pot accesa, dar iarasi cu exceptia anotatiilor ca in Sprin simplu! Atentie ca bara se refera si la resources si webapp, dar fara bara / ne referim doar la webapp!Ca in Sprin simplu

**Cum rulam aplicatia acum**

1. In pom.xml trebuie neaparat sa specificam ca aplicatia va fi un war:

**.jar nu suporta JSP!!!**

<packaging>war</packaging>

1. In Package de baza al aplicatiei, mai cream o clasa cu @SpringBootApplication si extindem SpringBootServletInitializer

@SpringBootApplication  
public class SpringBootTomcatApplication extends SpringBootServletInitializer {  
}

e necesara pentru a rula .war apps

1. mvn clean package – asa stergem toate package existente
2. mvn spring-bot:run

**Atentie! Nu folosim java -jar .....war**