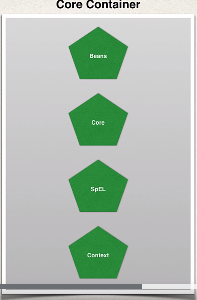
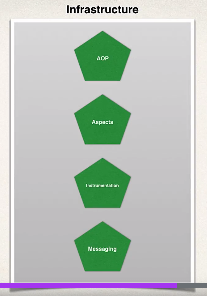
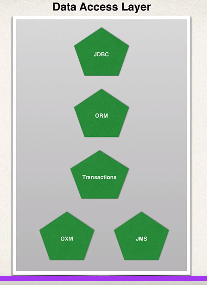
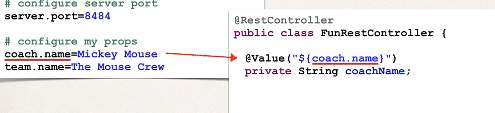
**Core Spring**  
  
Partendo dal contenitore centrale di spring,  
  
   
  
**CORE →** possiamo considerarlo un elemento principale, questo gestisce il modo in cui vengono creati i beans, in pratica può leggere i file di configurazione e impostare le proprietà e le dipendenze, gestendoli nella memoria.  
**SpEL** → è il linguaggio di espressione di spring, utilizzato nei file di configurazione per fare riferimento ad altri bean   
  
 Sezione AOP  
  
**AOP** → programmazione orientata agli aspetti, consentendo di creare servizi a livello di applicazione come logging, sicurezza, transazioni. Questo si traduce appunto nella configurazione dei file o tramite l’aggiunta di annotazioni, e il servizio verrà applicato, tuttto questo senza scrivere codice.



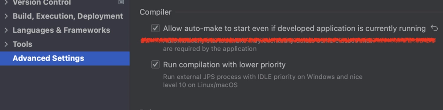
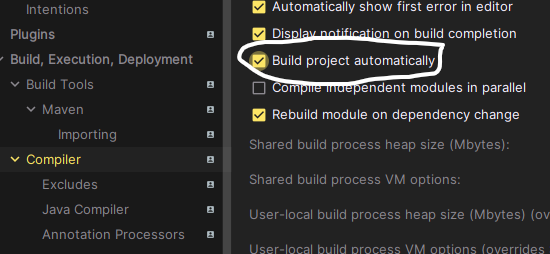
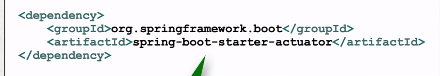
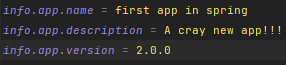
Proseguendo abbiamo la sezione di accesso ai dati quindi : Data access Layer  
  
   
  
Trattandosi di comunicare col database, sia esso un db relazione e un NoSQL  
**1)** vediamo JDBC, in quanto spring fornisce alcune classi di aiuto per rendere molto più semplice l’accesso a un database utilizzado JDBC.  
**2) ORM** → Object relational mapping (mappatura da oggetto a relazione), in pratica ci consente di agganrarsi a **hibernate o JPA**  
  
**JMS** → **Java Message Service,** consente di inviare messaggi a una coda di messaggi, in modo asincrono.  
Questa è una parte fondamentale di Java EE, consentendo l’uso del java message service   
**Transactions** → Spring inoltre supporta un gestore di transazioni o transazioni di supporto, potendo usare trasazioni sui metodi, sulle chiamate al database, praticamente   
  
In sintesi è possibile creare applicazioni web utilizzando Spring core, spring controller e spring view.  
Inoltre suppportano il remoting, per cui è possibile utilizzare il **remoting web**, in cui i client esterni possono effettuare chiamate al contenitore spring   
  
 

**File application properties**

In questo file inizialmente vuoto, possiamo impostare delle proprietà personalizzate oltre che a impostare una porta di ascolto diversa da quella di default.

Ritornando alle proprietà personalizzate possiamo poi richiamarle ovunque nel codice, con la sintassi che vediamo nella seconda immagine.  
  
  
  
 Package **RESOURCES**  
Poi abbiamo visto la struttura del package **resources** dentro il quale sono contenuti altri due package   
**static →** dentro questo package troveremo file statici come pagine html file css, script, immagini

avremo un package **templates →** dove troveremo questi templeate **FreeMarker, Thymeleaf, Mustache,**

i quali saranno caricati da spring boot per impostazione predefinita.  
  
Possiamo considereare l’aggiunta di questa dipendenza che permetterà di apportare modifiche live, senza dover essere costretti nel riavviare il server.  
  
  
  
non basta dobbiamo andare a spunatre un opzione nel compiler   
  
  
  
inoltre tramite l’aggiunta di questa dipendenza , spring ci metterà a disposizione degli endpoint raggiungibili per verificare lo stato di salute dell’applicazione  
  
  
  
  
  
  
  
  
**Spring actuator** → è un sotto-progetto di spring-boot Include una serie di funzionalità aggiuntive che ci aiutano a monitorare e gestire l'applicazione Spring Boot. Contiene gli endpoint dell'attuatore  
  
tramite questa dipendenza avremo a dispizione degli endpoint, tutti gli endpoint avranno come prefisso /actuator  
  
/health , /info per poter raggiungere quest’ultimo endpoint avremo bisogno di lavoare sul file application.properties e aggiungere queste due stringhe  
  
  
  
se proviamo ad accedere a quest’endpoint, otteniamo questa risposta {} un oggetto json vuoto questo perchè abbiamo bisogno di personalizzare le nostre info, in questo modo  
  
  
Ci sono più di 10 endpoint che sono disponibili successivamente ad aver caricato la dipendenza actuator  
  
  
  
Per esporre tutti gli endpoint dobbiamo aggiungere nel file properties questo codice   
  
  
[http://localhost:8080/actuator/**threaddump**](http://localhost:8080/actuator/threaddump) **→ si ottiene un elenco di tutti threa in esecuzione sulla nostra applicazione.**Bene, abbiamo reso disponibili alcuni endoint forniti da actuator, ma questi dati sono esposti a chiunque quindi vogliamo tutelarci e rendere questi dati non accessibili a chiunque, come facciamo?

Risolviamo aggiungendo questa dipendenza, quando proveremo ad accedere a questi dati ci sarà fornita una password dalla console di intellij, mentre il name di default è user.  
  
  
  
Volendo possiamo modificare questi dati di accesso a questi endpoint, ovviamente andando a lavorare sul file application.properties e personalizzare quei campi e quindi :

