Seri Manajemen Proyek Software

Template Aplikasi Standar ArtiVisi

Arsitektur, Framework, dan Tools Pengembangan Aplikasi

Version:	*Last Updated:*
1.0	3 Desember 2012

Template Aplikasi Standar ArtiVisi

Endy Muhardin

3 Desember 2012

Contents

1 Pendahuluan

1.1 Tentang Dokumen Teknis

1.2 Tentang Template Aplikasi ArtiVisi

1.3 Profil Pembaca

2 Arsitektur Aplikasi

Aplikasi ini dirancang dengan beberapa tujuan:

- modular: aplikasi dipecah menjadi beberapa modul agar bisa digunakan (reuse) di aplikasi lain
- layered: aplikasi dibagi menjadi layer tampilan, proses bisnis, dan akses data. Dengan pembagian ini, perubahan di satu layer dapat dilakukan secara independen tanpa mengganggu layer lainnya.

Untuk mencapai tujuan di atas, kita bisa membagi aplikasi menjadi dua dimensi yaitu vertical slice dan horizontal slice.

2.1 Vertical Slice

Vertical slice adalah membagi aplikasi sesuai fungsi dan fitur. Sebagai contoh, kita dapat membagi aplikasi menjadi modul user management, security, konfigurasi, dan sebagainya.

Dengan pembagian ini, bagian-bagian yang memiliki fungsi umum seperti security atau user-management dapat digunakan lagi di aplikasi lain.

2.2 Horizontal Slice

Horizontal slice artinya membagi aplikasi sesuai layer, misalnya tampilan, proses bisnis, dan akses data atau integrasi.

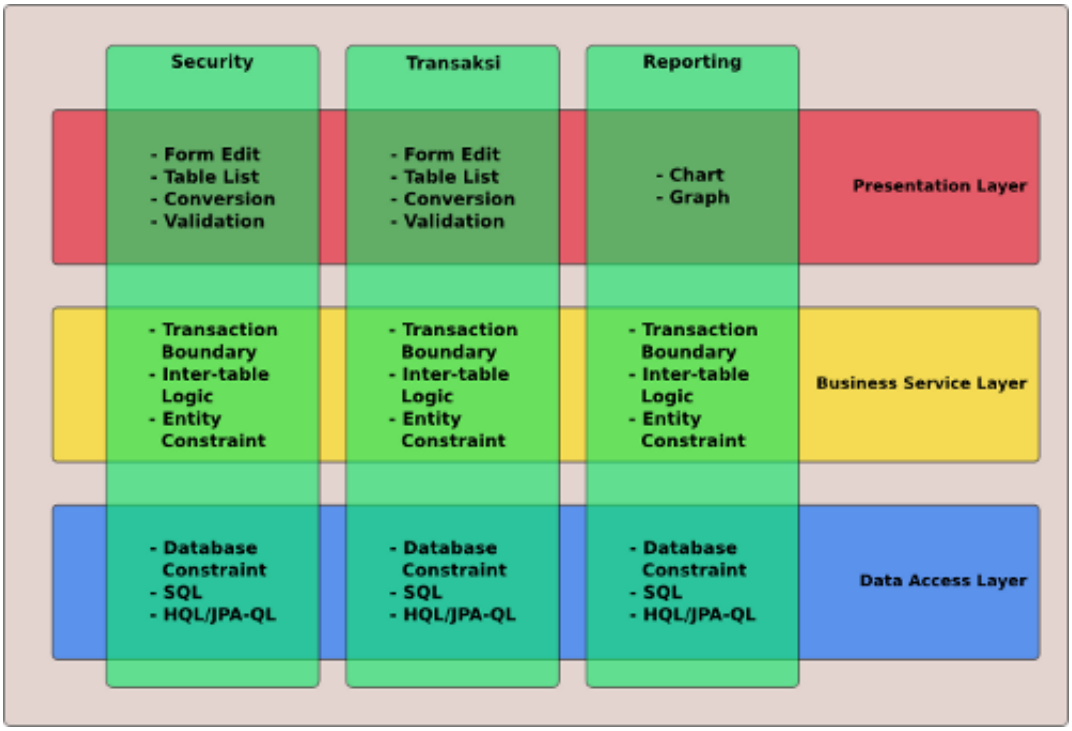

Figure 1: Pembagian Aplikasi secara Fungsional

Berikut adalah penjelasan dari pembagian modul dalam bentuk diagram. Kita bisa membagi aplikasi secara fungsional sebagai berikut.

Untuk masing-masing bagian, berikut adalah pemetaan penggunaan framework dan library.

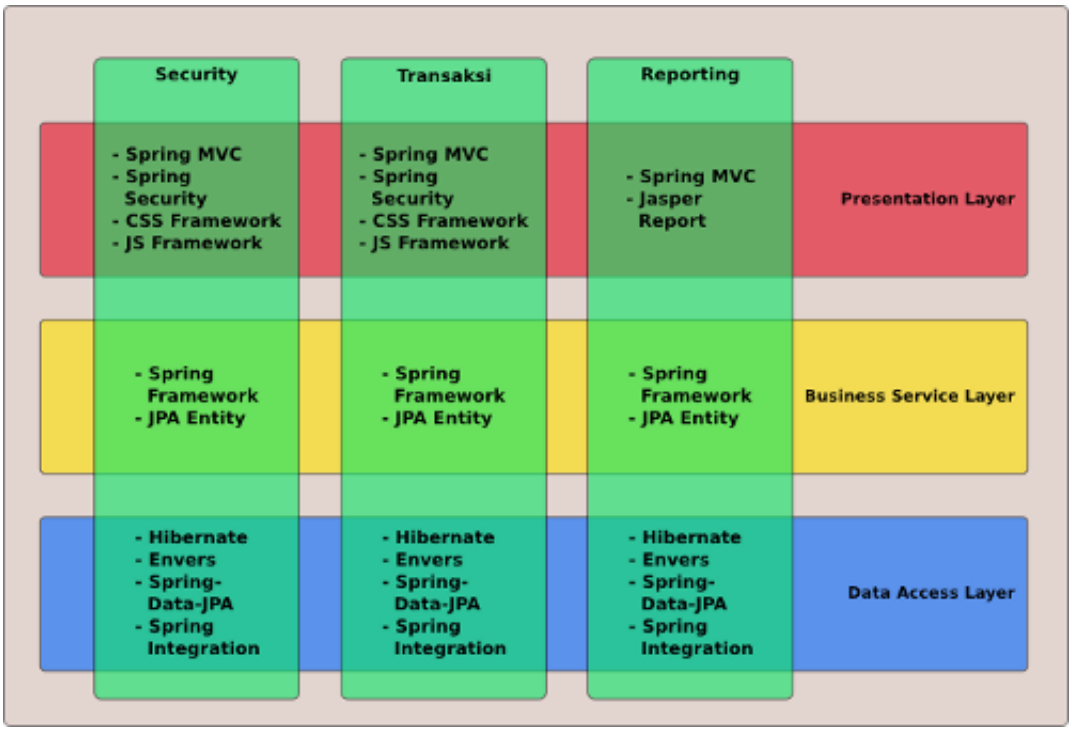

Figure 2: Penggunaan Framework dalam Modul

2.3 Component Reuse

Untuk aplikasi Java, satu unit modul dipaketkan dalam bentuk file `jar`. File ini bisa disertakan dalam aplikasi lain bila kita ingin menggunakannya.

2.4 Development Workflow

Maven adalah tools untuk melakukan kegiatan build di Java. Maven memiliki fitur dependency management, yaitu dia mampu mencari dan memasang semua dependensi

yang dibutuhkan.

Dengan menggunakan Maven, kita dapat menggunakan modul yang sudah kita buat di aplikasi lain. Demikian juga kita dapat membagi modul yang kita baru saja buat untuk digunakan di aplikasi lain.

Dalam mencari dependensi, Maven akan mengakses file server yang disebut repository. Di internet ada repository besar yang memuat banyak library opensource seperti Spring Framework, Hibernate, dan sebagainya. Berbagai library ini disusun dalam struktur folder yang seragam sehingga mudah dicari dan digunakan.

Selain repository yang tersedia di internet, kita juga bisa membuat repository untuk kita gunakan secara internal dalam perusahaan. Aplikasi yang digunakan adalah Nexus. Nexus berbentuk aplikasi web yang diinstal di server. Untuk library atau modul yang hanya digunakan di internal, kita bisa menggunakan akses kontrol berupa username dan password agar modul internal kita tidak bisa diakses sembarang orang.

Untuk mengisi Nexus, aplikasi perlu dibuild dulu agar menghasilkan jar atau war. Jar atau war ini disebut dengan istilah *artifact*. Kegiatan build ini bisa diotomasi menggunakan aplikasi Continuous Integration seperti misalnya Jenkins. Jenkins akan mengambil source-code terbaru dari version control, melakukan build, dan mengunggah hasilnya ke Nexus.

Skema workflow ini dapat dilihat pada diagram berikut:

Kita bisa lihat pada diagram di atas bahwa artifact yang kita buat akan diunggah ke Nexus. Misalnya, kita membuat library untuk security dengan nama sebagai berikut:

- Group ID : com.artivisi.security
- Artifact ID : user-management
- Version : 1.2.3

Setelah diunggah ke Nexus, project lain bisa menggunakan artifact tersebut dengan memasang dependensi berikut di konfigurasi Maven (`pom.xml`):

```
<dependency>
    <groupId>com.artivisi.security</groupId>
    <artifactId>user-management</artifactId>
    <version>1.2.3</version>
</dependency>
```

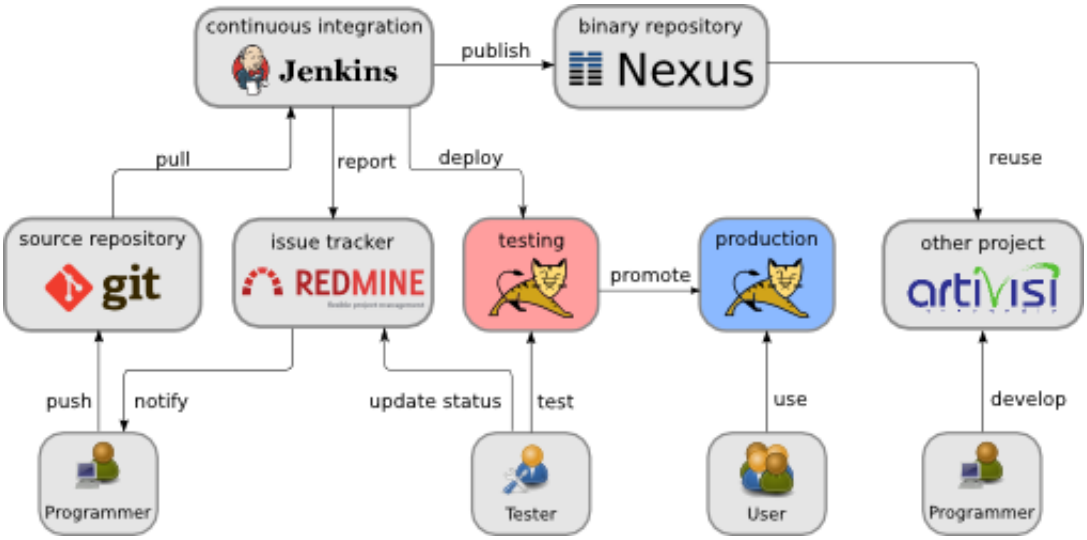

Figure 3: Workflow Development Aplikasi

Maven otomatis akan mencari artifact tersebut, mengunduh ke folder lokal, dan mendaftarkannya di `CLASSPATH` sehingga bisa diimport dalam source code.

2.5 REST dan Single Page Application

Aplikasi jaman sekarang biasanya dibuat menggunakan antar muka berbasis web. Tampilan diperindah menggunakan CSS, perilaku aplikasi dibuat menggunakan JavaScript, dan pertukaran data dilakukan dengan format JSON.

Aplikasi seperti ini biasanya hanya terdiri dari satu halaman/file HTML saja, sehingga disebut Single Page Application (SPA). Salah satu aplikasi populer yang dibuat dengan arsitektur SPA adalah Gmail.

Berikut adalah diagram aplikasi yang dibuat menggunakan arsitektur ini.

Ada banyak sekali pilihan library yang bisa digunakan untuk membuat aplikasi SPA. Di template ini kita menggunakan kombinasi berikut:

- CSS : Twitter Bootstrap
- JavaScript : AngularJS

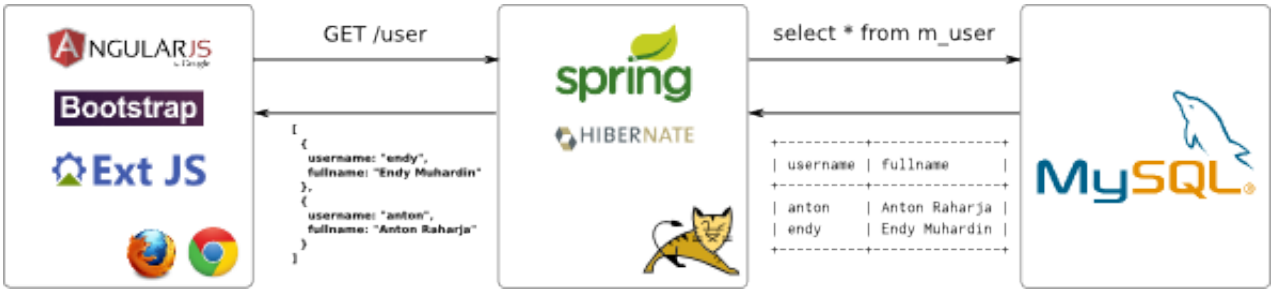

Figure 4: Arsitektur RESTful

- Serverside : SpringMVC

Informasi lebih detail mengenai library dan framework di atas akan disampaikan pada bagian selanjutnya di bawah.

2.6 Build Tools

Agar dapat mengelola aplikasi modular dengan mudah, kita membutuhkan tools untuk mengelola hubungan antar modul. Ada banyak alternatif tools yang tersedia, diantaranya:

- Maven
- Ant + Ivy
- Gradle

Dari ketiga alternatif tersebut, kita akan menggunakan Maven karena paling populer. Itu artinya dokumentasi banyak tersedia, demikian juga masalah-masalah yang umum terjadi telah ditemukan pemecahannya.

Fitur utama yang akan sangat bermanfaat bagi kita adalah dependency management. Misalnya aplikasi kita menggunakan library Hibernate. Hibernate ini membutuhkan library lain yang bernama cglib dan entah berapa banyak library lainnya. Dependensi lapis kedua dan seterusnya ini disebut dengan istilah *transitive dependency*. Dengan menggunakan dependency management, kita cukup mengetahui bahwa aplikasi kita membutuhkan Hibernate. *Transitive dependency* tidak perlu kita pusingkan.

2.6.1 Dependency Management

Dependensi dalam aplikasi ditulis dalam file konfigurasi `pom.xml`. Berikut adalah contoh konfigurasi dependensi terhadap Spring Framework dan Hibernate.

```
<dependencies>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.1.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.2</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>1.7.2</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>log4j-over-slf4j</artifactId>
            <version>1.7.2</version>
            <scope>runtime</scope>
        </dependency>
</dependencies>
```

Untuk menyatakan dependensi di Maven, kita harus menyebutkan 3 variabel wajib dan 1 variabel tambahan, yaitu:

- groupId : biasanya organisasi yang membuat library
- artifactId : nama modul atau library

- version : versi library
- scope : kapan dependensi ini digunakan

Variabel scope tidak wajib. Bila kita tidak sebutkan akan diisi dengan nilai `compile`. Berikut adalah nilai yang bisa kita isikan untuk scope:

- compile : digunakan untuk kompilasi dan juga disertakan dalam produk akhir
- runtime : tidak dibutuhkan untuk kompilasi, tapi disertakan dalam produk akhir
- test : hanya dibutuhkan untuk test saja, tidak disertakan dalam produk akhir
- provided : dibutuhkan untuk kompilasi, tapi tidak disertakan dalam produk akhir karena sudah tersedia. Ini biasanya dipakai bila aplikasi akan dideploy di application server, sehingga library yang kita pakai sudah disediakan.

2.6.2 Artifact Publishing

Bila kita bisa menggunakan library yang dibuat orang lain, tentu kita juga bisa mempublikasikan library yang kita buat agar bisa digunakan orang lain. Agar bisa melakukan hal tersebut, kita butuh dua hal:

- tools untuk mengunggah library
- tools untuk menerima unggahan dan menampilkan isinya agar bisa diunduh orang lain

Untuk mengunggah library, kita bisa menggunakan Maven. Sedangkan untuk menyimpan dan menampilkan hasilnya, kita bisa menggunakan Nexus.

Berikut adalah konfigurasi di Maven yang menyebutkan lokasi Nexus.

```
<distributionManagement>
    <!-- releases repository -->
    <repository>
        <id>artivisi-nexus</id>
```

```
        <name>ArtiVisi Release Repository</name>
        <url>
http://forge.artivisi.com:8080/nexus/content/repositories/releases
        </url>
    </repository>

    <!-- snapshot repository -->
    <snapshotRepository>
        <id>artivisi-nexus</id>
        <name>ArtiVisi Snapshot Repository</name>
        <url>
http://forge.artivisi.com:8080/nexus/content/repositories/snapshots
        </url>
    </snapshotRepository>
</distributionManagement>
```

Agar bisa mengunggah, tentu kita harus menyediakan username dan password untuk mengakses Nexus. Username dan password ini kita tulis di file `settings.xml` yang berada di folder `.m2` dalam `HOME FOLDER` user komputer. Di sistem Linux, bila usernamenya adalah endy, maka `HOME FOLDER` adalah `/home/endy`.

Berikut isi dari `settings.xml`

```
<settings>
    <servers>
        <server>
            <id>artivisi-nexus</id>
            <username>admin</username>
            <password>admin123</password>
        </server>
    </servers>
</settings>
```

Bila menyimpan *plain text* password seperti di atas terlalu mengerikan, kita bisa menyimpan password secara terenkripsi dengan mengikuti panduan di websitenya Maven.

3 Inisialisasi Project

Untuk mulai membuat aplikasi, terlebih dulu kita harus membuat struktur folder sesuai dengan standar yang sudah kita tetapkan di atas. Maven memiliki fitur untuk menyiapkan struktur project sesuai kebiasaan kita, yaitu yang dinamakan dengan `archetype`.

3.1 Maven Archetype

Maven archetype adalah seperangkat template untuk membuat project. ArtiVisi telah memiliki archetype untuk struktur project seperti yang telah dijelaskan di atas. Archetype ini dapat diambil di Github.

Pertama, kita harus menginstal dulu archetype tersebut di repository lokal. Jalankan perintah berikut di dalam folder archetype:

```
mvn clean install
```

Selanjutnya, kita dapat mulai membuat project. Posisikan command prompt kita di dalam folder dimana aplikasi akan disimpan. Kemudian jalankan perintah berikut:

```
mvn archetype:generate \
-DarchetypeGroupId=com.artivisi.template \
-DarchetypeArtifactId=standard-webapp-archetype
```

Kita an ditanyai beberapa pertanyaan sebagai berikut:

- groupId : nama modul kita, biasanya nama package induk dari aplikasi. Misalnya `com.artivisi.akunting`
- artifactId : nama aplikasi kita tanpa spasi. Misalnya `akunting`
- project-name : nama aplikasi dalam bentuk manusiawi, misalnya `Aplikasi Akunting`

Setelah semua dijawab, Maven akan mulai membuat struktur folder untuk aplikasi.

3.2 Struktur Folder

Struktur folder yang dihasilkan archetype adalah sebagai berikut:

- config : berisi konfigurasi untuk keseluruhan aplikasi, misalnya koneksi database, logging, dan sebagainya
- domain : berisi Java class yang memuat mapping table, constraint dan relasi, juga interface berisi daftar proses bisnis yang dilayani modul ini
- service : berisi implementasi proses bisnis berupa kode program Java dan query database (HQL, JPAQL, QueryDSL, dan sebagainya).
- web : mengexpose proses bisnis dengan protokol REST. Kita juga bisa menambahkan user interface menggunakan HTML dan framework JavaScript seperti AngularJS, ExtJS, ataupun Dojo Toolkit.

3.3 Inisialisasi Database

Sebelum memulai proses build, terlebih dulu kita siapkan database. Konfigurasi awal (default) dari archetype ini adalah sebagai berikut:

- Tipe database : MySQL 5
- Nama database : sesuai format `artifactId_development`, misalnya `akunting_development`.
- Username database : root
- Password database : admin

Bila kita ingin menggunakan konfigurasi selain di atas, kita harus mengedit file `pom.xml` di project induk dan mengganti di baris berikut:

```
<!-- konfigurasi database development -->
<hibernate.dialect>isi hibernate dialect di sini</hibernate.dialect>
<db.driver>JDBC Driver Class</db.driver>
<db.url><![CDATA[URL Database]]></db.url>
<db.username>username database</db.username>
<db.password>password database</db.password>
```

3.4 Proses Build

Setelah database disiapkan, kita bisa menjalankan proses build dengan perintah

```
mvn clean install
```

Maven akan mengunduh semua library yang dibutuhkan, kemudian melakukan kompilasi, menjalankan semua tes, dan melakukan packaging untuk membentuk hasil akhir berupa `jar` dan `war` sesuai modulnya.

Bila proses build dijalankan dengan sukses, di akhir log akan muncul tulisan `BUILD SUCCESSFUL`.

3.5 Test Report

Dalam modul `service` dan `web` ada serangkaian tes otomatis yang dijalankan oleh Maven. Hasil tesnya disimpan dalam folder `target/failsafe-reports` di masing-masing modul tersebut. Bila hasil akhir proses build adalah `BUILD FAILURE`, kita harus memeriksa isi folder tersebut dan membaca isinya untuk mengetahui penyebab terjadinya error.

3.6 Menjalankan Aplikasi

Setelah aplikasi sukses dibuild, kita dapat menjalankan aplikasi dengan menggunakan *application server* Jetty yang dipasang sebagai plugin Maven. Berikut cara menjalankannya:

1. Pindah ke folder web

   ```
   cd akunting-web
   ```

2. Jalankan perintah berikut:

   ```
   mvn jetty:run
   ```

3. Browse ke `http://localhost:10000`

4 Tools, Library, dan Framework

4.1 Liquibase

Liquibase adalah tools untuk memelihara versioning pada database. Dengan Liquibase, kita dapat:

1. Menandai (tagging) rilis aplikasi versi tertentu
2. Membuat rilis selanjutnya yang mengandung perubahan skema database
3. Bila terjadi error, kita bisa mengembalikan (rollback) skema database ke versi sebelumnya

Perubahan skema database ditulis dalam file changelog. Berikut adalah contoh file changelog untuk membuat tabel di database.

```
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-2.0.xsd
">
    <changeSet id="1" author="endy">
        <createTable tableName="c_application_config">
            <column name="id" type="varchar(255)">
                <constraints primaryKey="true" nullable="false" />
            </column>
            <column name="name" type="varchar(255)">
                <constraints unique="true" nullable="false" />
            </column>
            <column name="label" type="varchar(255)">
                <constraints nullable="false" />
            </column>
            <column name="value" type="varchar(255)">
                <constraints nullable="false" />
            </column>
```

```
        </createTable>
    </changeSet>
</databaseChangeLog>
```

4.2 Spring Framework

- Spring Framework Digunakan untuk membangun dan mengkonfigurasi sebuah aplikasi yang kompleks dari komponen-komponen yang sederhana.
- Berikut adalah contoh dependency maven untuk spring framework :

```
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>3.1.3.RELEASE</version>
  </dependency>
```

- Berikut adalah contoh file konfigurasi untuk spring framework :
- Perintah di bawah ini digunakan untuk mendeklarasikan namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.1.xsd
        http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
">
```

- Perintah dibawah ini untuk memanggil properties file.

```
<context:property-placeholder
    location="classpath*:jdbc.properties" />
```

- Perintah dibawah ini digunakan untuk mencari class-class mana aja yang mengandung atau yang di dalamnya terdapat annotation @controller, @service.

```
<context:component-scan base-package="com.artivisi.training.service" />
<tx:annotation-driven/>
```

4.3 Hibernate

- uuid : untuk menggenerate id. Berikut contohnya :

```
@Id @GeneratedValue(generator="system-uuid")
@GenericGenerator(name="system-uuid", strategy = "uuid2")
```

- untuk menggunakan hibernite ada 2 pilihan :
  - menggunakan langsung hibernate
  - menggunakan spring JPA
- DatabasePlatform : digunakan untuk melihat database apa yang digunakan.
- hbm2ddl.auto : gunanya untuk mengotomatis generate table.
- dataSource digunakan untuk konfigurasi database.
- EntityManagerFactory digunakan untuk mapping class entity.
- packagesToScan : untuk memanggil class entity yang berada didalam package.
- databasePlatform : untuk menentukan type tabel yang digunakan.
- showSql : untuk menampilkan query yang dilakukan kedalam log.
- generateDdl : untuk mengaktifkan atau menonaktifkan pembuatan tabel dari JPA.
- format_sql : untuk merapikan format query yang akan tampil di log.

- hbm2ddl.auto : untuk mengotomatis generate table.

```
 <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
      <property name="dataSource" ref="dataSource" />
   <property name="packagesToScan" value="com.artivisi.training.domain" />
      <property name="jpaVendorAdapter">
      <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"
         <property name="databasePlatform" value="org.hibernate.dialect.MySQL5In
              <property name="showSql" value="true" />
              <property name="generateDdl" value="true" />
          </bean>
      </property>
      <property name="jpaProperties">
          <props>
              <prop key="hibernate.format_sql">true</prop>
             <prop key="hibernate.hbm2ddl.auto">update</prop>
          </props>
      </property>
  </bean>
</beans>
```

4.4 Spring Data JPA

- EntityManagerFactory adalah bawaan Spring JPA
- JpaTransactionManager digunakan untuk memanagement transaksi database.

4.5 Spring MVC

- Pertama kita buka link berikut : http://www.springsource.org/spring-framework
- Kemudian klik Documentation, untuk melihat dokumentasi spring MVC
- Tambahkan servlet spring-webmvc ke dalam web.xml :

  example org.springframework.web.servlet.DispatcherServlet 1

```
<servlet-mapping>
    <servlet-name>secman</servlet-name>
    <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

- Tambahkan file baru di dalam webapp dengan nama secman-servlet.xml, lalu masukkan Enabling MVC Java Config or the MVC XML Namespace didalam file tersebut :
- annotation-driven : gunanya untuk mengaktifkan controller, requestMapping, ResponseBody, dan masih banyak lagi.
- component-scan : gunanya untuk mencari class-class mana aja yang mengandung atau yang di dalamnya terdapat annotation-driven.

4.6 Spring Security

4.7 Logging Framework

ConsoleAppender : akan di tampilkan langsung out.println ke terminal.

- logback : gunanya untuk menulis atau menampilkan log (langsung out.println ke terminal).
- spring.profiles.active:-development : digunakan untuk mengaktifkan profiles. Jenisnya banyak, tergantung kita mau menggunakan yang mana. Berikut beberapa jenis profiles :
  - Development
  - Testing
  - Production

Berikut contoh profiles development :

```
<configuration>
```

```
   <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d %-5level %logger{35} - %msg %n</pattern>
        </encoder>
    </appender>

   <appender name="FILE" class="ch.qos.logback.core.FileAppender">
      <file>${catalina.home:-.}/logs/secman2-${spring.profiles.active:-development
        <encoder>
            <pattern>%d %-5level %logger{35} - %msg %n</pattern>
        </encoder>
    </appender>

   <include resource="logback-${spring.profiles.active:-development}.xml"/>

</configuration>
```

4.8 JUnit

4.9 DBUnit

4.10 Rest Assured

4.11 Surefire dan Failsafe

4.12 Java Melody

4.13 Jenkins