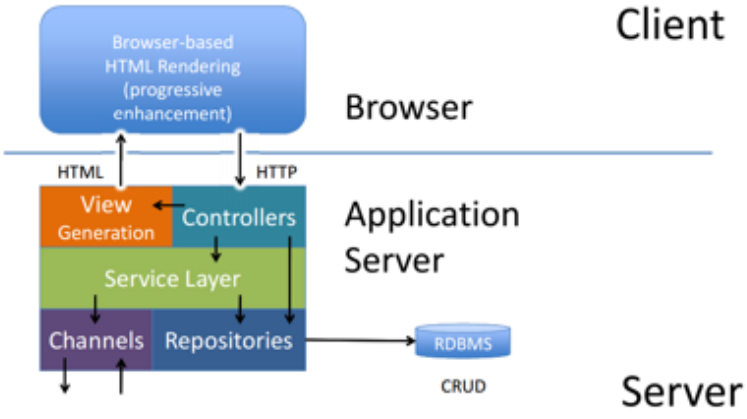
Arhitektura veb aplikacija

Nama je cilj da na Modulu 2 kreiramo veb alikaciju koja prati sledeću arhitekturu, to je klasična veb aplikacija, MVC aplikacija u kojoj se HTML kod generiše na serverskoj stani i isporučuje klijentu. U prethodnoj tehnologiji koja se radili to su bili Servleti i JSP stranice. Servleti su bili kontroleri, a pogled su bili JSP stranice.

Kod komunikacije na vebu, sa jedne strane je Klijent, a sa druge je Server, koji komuniciraju po HTTP protokolu (vidi sliku). Klijent je brauzerska aplikacija, i sve sve što brauzer može da uradi je da pošalje HTTP zahtev ka Serveru tražeći od njega neki resurs (HTML stanica), i da po dobijanju HTTP odgovora prikaže dobijenu HTML stanicu. Server prihvata HTTP zahteve, obrađuje ih i vraća HTTP odgovor. Server na oslovu URL zaključuje koji to resurs korisnik traži, kontroleri po potrebi odrađuju neku poslovnu logiku, koriste razne servise da bi dobavili neke podatke iz baze ili nešto ažurirali u bazi, popunjavaju podatke u model, model zatim proseđuju pogledu koji će dinamički generisti HTML stranicu, a Server će tako tako generisanu HTML stanicu vratiti klijentu. Generisana HTML stanica ne sadrži Java kod već samo HTML elemente.

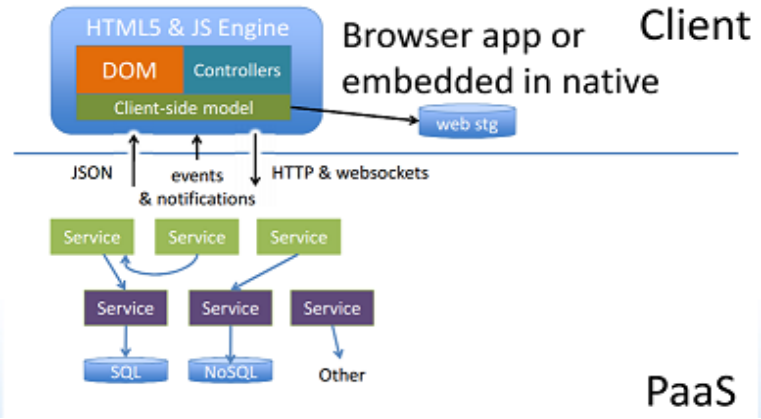
Classical web - server side generated HTML:



Ovakav način arhitekture ima par potencijalnih problema. Sa Klijentske strane na kojoj se nalazi veb brauzer, sve što radi klijentski računar je da renderuje i prikazuje HTML. Na drugoj stani Server radi sav neophodan posao, obrada podataka, generisanje HTML stanica, kompletna poslovna logika se izvršava na stani servera. Server je najverovatnije postavljen negde u oblaku, za njega vlasnici aplikacije plaćaju resurse, broj procesora, količina memorije,… Kod MVC aplikacije Korisnik klikne na dugme ili link i čeka nekoliko sekundi za odgovorom aplikacije, ekran je beo, neizvesnost da li aplikacija radi, osećaj gubitka kontrole, ako se šalje ceo HTML to dodatno opterećuje mrežu. Da rezimiramo: bilo kakva, pa makar i minimalna interakcija Klijenta sa Serverom, Server generiše celokupan HTML stranice i vraća je Klijentu.

Poslednjih godina na Klijektskoj strani se koristi računaru sa visokim hardverskim performansama, te bi bilo zgodno da se deo poslovne logike i obrade sa Servera prebaci na stranu Klijenta u našem slučaju brauzer. Ideja je da Server Klijentu vraćaja delove HTML stranica ili samo podatke, na osnovu kojih Klijent samostalno može da obavi izmene delova postojeće HTML stanice na svojoj straini. To ažuriranje delova HTML stranica na Klijenstoj strani je moguće jer u brauzeru može izvršavati neki JavaScript. Server će Klijentu slati samo podatke, to je Data centric approach (vidi sliku). Na kraju drugog modula će se samo uvesti ta priča sa AJAX tehnologijom, ona će se do detalja obraditi na trećem modulu. Klijentski deo aplikacije se naziva FrontEnd i radi se na 2 delu 3 modula, odabrana tehnologija je React, dok je bekend deo aplikacije je u Springu.

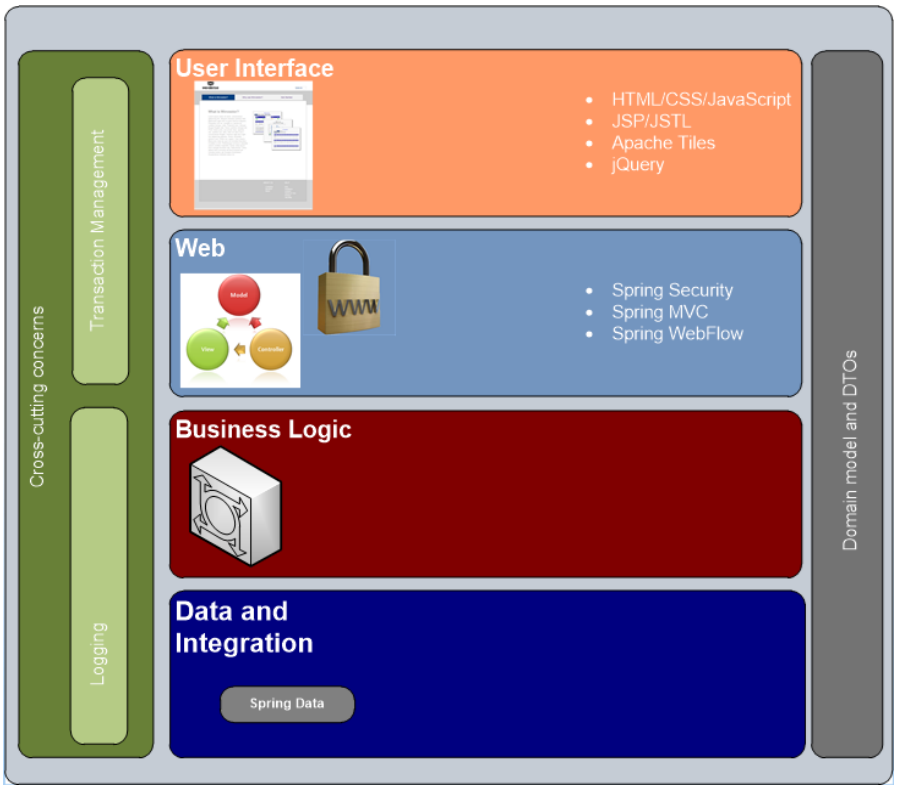
Data centric approach - server side generated data:



Naravno MVC arhitektura veb aplikacija ima i svoje prednosti. Ovakav način razvoja aplikacije je dosta *straightforward* jer je to objektno orijentisana programska arhitektura. Razvoj aplikacije sa MVC arhitetkurom postaje brz, separacija taskova i uloga u timu koji razvija aplikaciju doprinosi paralelizmu i omogućuje sa lakoćom da više developera sarađuju i rade zajedno, olakšano je održavanje i ažuriranje aplikacije, i olakšano je debagovanje aplikacije. Više o benefitima možete viditi ovde (https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/, http://siyainfo.com/2017/01/16/top-6-important-benefits-mvc-architecture-web-application-development-process/)

Uvod u veb aplikacije u Spring

Primer delova veb aplikacije napisane u Springu možete videti na sledećoj slici.



U pethodnim iteracijama JWD kurseva < 43, se MVC veb aplikacija razvijala u Servlet tehnologiji. Korišćenjem te tehnologije se standardna Java aplikacija proširivala podrškom za Servlete tako da se dobije veb aplikcija. Neophodno je bilo ubaciti dodatne Java biblioteke i implementirati odgovarjuće Servlet Java klase.

Sada se prešlo na Spring radni okvir za kreiranje MVC veb aplikacija. Šta je to spring radni okvir? To je nešto što je već neko uradio za nas, nešto gotovo, nešto što bi nam olakšalo razvoj veb aplikacija.

Koja je razlika korišćenja bibloteka iz prvog slučaja i korišćenja radnog okvira? Kod korišćenja biblioteke ona se ugrađuje u **naš** projekat, i na odgovarjućem mestu se poziva funkcionalnost te biblioteke, a ona vraća neki rezultat. Sa druge strane kada se koristi radni okvir, on implementira neke učestale mehanizme, nizove koraka, koji se uvek izvršavaju u aplikaciji. Kada se dođe do konkretnog koraka koji treba da se izvrši, tada je neophodno za radni okvir isprogramirati i ugraditi u njega odgovarajuću komponentu, koja će izvršiti odgovarjuću funkcionalnost.

Rezimiramo: Prvi slučaj biblioteka se ubacuje u projekat, u drugom slučaju se komponenta ugrađuje u već predefinisanu aplikaciju (aplikacija kao da već postoji samo je treba doraditi). U prvom slučaju kontrola je na našoj strani, mi pozivamo bibloteku, dok u drugom slučaju radni okvir poziva našu komponentu, to je suštinska razlika. Princip koji se koristi kod Spring radnog okvira zove se inverzija kontrole (Inversion of Control) ili može se čak i zvati Hollywood Principle.

U knjizi *Head First Design pattern* se princip objašnjava:

> With the Hollywood Principle, we allow low-level components to hook themselves into a system, but the high-level components determine when they are needed, and how. In other words, the high-level components give the low-level components a “don’t call us, we’ll call you” treatment.

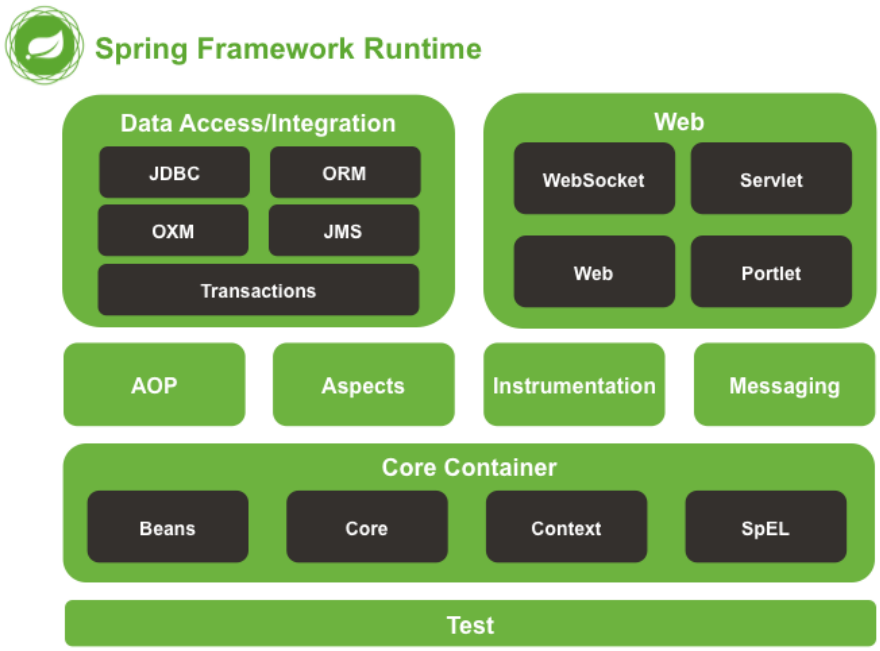
According to the paper written by Martin Fowler(https://martinfowler.com/bliki/InversionOfControl.html), inversion of control is the principle where the control flow of a program is inverted: instead of the programmer

controlling the flow of a program, the external sources (framework, services, other components) take control of it. It's like we plug something into something else.

Spring ima široko polje primene pa se može reće da je to radni okvir za izradu aplikacija. To mogu biti veb aplikacije (klasična MVC arhitektura ili nova arhitektura), desktop aplikacije, mobilne aplikacije, itd.

Kako je Spring radni okvir širok, ne bi bilo praktično ugraditi ceo radni okvir u projekat. Spring je podeljen na module, svaki modul zadužen za neku funkcionalnost. Postoji Spring Core (vidi sliku) koji implementira osnovne mehanizme rada Springa, te se razvoj svih Spring aplikacija svodi na korišćenje Spring Core modula i onih modula koji su potrebni za konkretan tip aplikacije čija je funkcionalnost neophodno implementirati. Za kreiranje veb aplikacija uključuje se Web modul. Ako je potreba perzistencija podataka uključuje se Data Access/Integration modul. Više o Springu na https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html

Spring framework



Na današnjem terminu neće se razvijati Sring aplikacija, već će se razvija componenta koje će se ugraditi u Spring projekat.

Kreiranje Maven Veb projekta u Eclipse alatu

Za kreiranje Spring aplikacije koristiće se Maven alat. Maven alat (https://maven.apache.org/users/index.html) ima za cilj olakšavanje razvoja kompleksnih projekata. To je *build tool*, alat za izgradnju tj. konstrukciju projekta kompleksnih aplikacija. Maven se može koristi kao nazavisan *stand alone* alat preko komandne linije ili se može koristiti i pozvati iz Eclipse softvera oslanjajući se na Maven Eclipse plug-in. Za korišćenje Maven kao nazavisan *stand alone* alat potrebno je alat preuzeti sa https://maven.apache.org/ . U svim novijim verzijama Eclipse softvera već je integrisan Maven plugin sa Maven instlacijom, dok starije verzije imaju samo one označene kao *Eclipse IDE for Enterprise Java Developers*.

Na prethodnim kursevima, neophodno je bilo iz Eclipse pozvati kompajliranje Java koda veb aplikacije koje je razvijana u Servlet thenologiji tj. dobiti class fajlove od Java fajlova. Zatim je bilo potrebno zapakovati class fajlove i ostale veb resurse (HTML, CSS, slike i druge fajlove) u arhivu, čija je ekstenzija war. Zatim je neophodno bilo ubaciti war arhivu u veb server koji je bio Tomcat tj. izvršiti *deployment* te veb aplikacije. A Tomcat veb server će izvršavati kompajlirani Java kod deplojovane veb aplikacije. U suštini je sada opisan životni ciklus izgradnje programskog paketa za veb aplikaciju.

Glavni zadatka Maven alata je da izvršava *Build life-cycle* (otvorite https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html). *Build life-cycle* predstavlja niz koraka tj. faza koje se izvršavaju svaki put kada se builduje Maven aplikacija. Mogu se izvršiti svi koraci ili se može izvršiti do određenog koraka. Posle svako uspešnog izvršenog koraka ide se na sledeći.

1. validate – proverava tj. validira da li je projekat korektan, da li su tu svi neohodni delovi projekta. Ako je sve u redu ide
2. compale – pretvara izvorni u izvršni kod.
3. test – testiranje aplikacije, pokreću se Unit testovi. Ako testovi prođu ide
4. package – od kompajliranog koda, u zavisnosti od tipa aplikacije, kreira se paket tipa war ili jar.
5. verify – pokreću se integracioni tetovi
6. install – instalcija paketa u lokalni repozitorijum
7. deploy – instalacija paketa u udaljeni repozitorijum

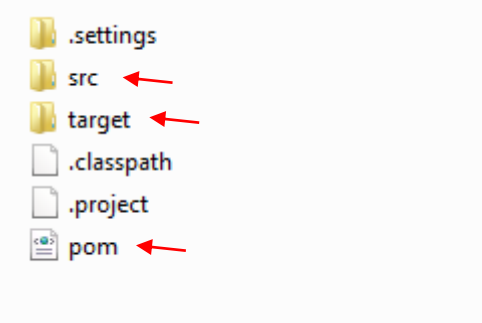
Zašto su neophodni svi ovi koraci?

Ako se uzme u razmatranje da je tester u projektnom timu prijavio određeni bug, najlogičnije bi bilo da se samo ispravi greška u kodu, pokrene se aplikacija i proveri se da li taj specifični deo koda radi. Na prethodno opisani način niko ne garantuje da izmena jednog dela koda neće loše uticati na rad ostatka aplikacije, popravila je 1 stvar ali je to nehotice uticalo na rad nekoliko drugih delova aplikacije. Zato postoje i izvršavaju se svi pomenuti koraci u ciklusu, među kojima da se testira rad pojedini delova koda, a zatim da se testira rad aplikacije kao celine.

U Maven projektu poštuje se princip *Convention over Configuration*. Kod kompleksnih projekat dosta vremena se potroši na konfiguraciju datog projekta, što implicira da kada se napiše deo koda tj. Java fajl to je neophodno adekvatno ispratiti u određenom konfiguracionom fajlu, opisom funkcionalnosti i lokcije u projektu za kreirani deo koda. Ideja je da se konfiguracija projekta ne radi eksplictno, već da postoji dogovor/konvencija po kojoj se od programera očekuje samo da se klase za određenu funkcionalnost nalaze u tačno predefinisanim folderima u projektu, da se one nazivaju po nekoj konvenciji i da je na taj način projekat iskonfigursan adekvatno. Ukoliko je neophodno izneniti nešto od podrazumevanih dogovora, dostupno je dodatno konfigurisanje projekta.

Maven koristi specifičnu strukturu direktorijuma unutar projekta i ova struktura MORA biti ispoštovana - Standard Directory Layout.

Predefinisani prostorni raspored foldera u Maven projektu je sledeći
https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

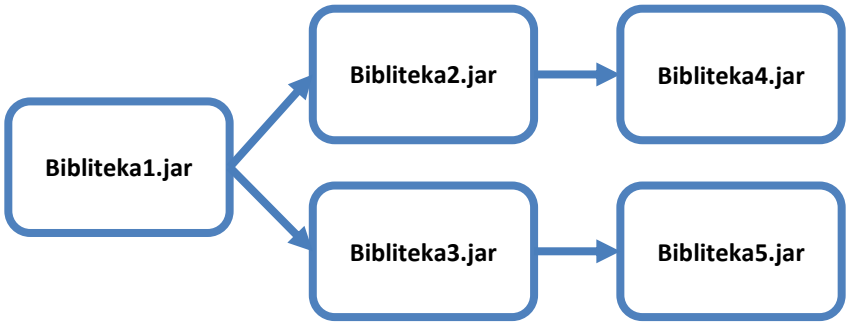


U root projekta nalaze se samo folder *src* i *target*, i *pom.xml* fajl. Ostale fajlove i foldere je kreirao Eclipse.

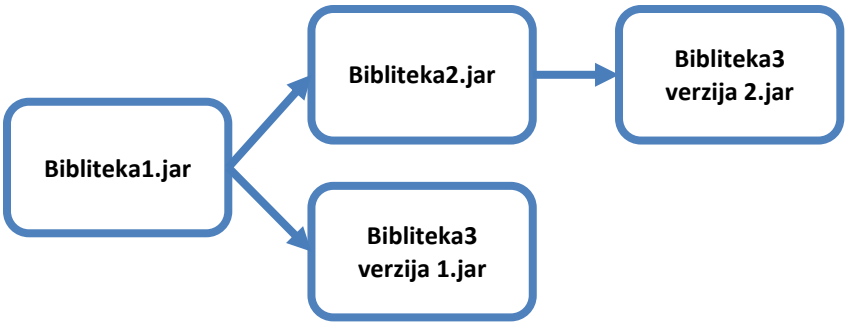
Fajl pom.xml sadrži konfiguraciju Maven projekta. U folderu *src* se nalazi sve što je isprogramirano, dok se u folderu *target* nalazi sve što je rezultat Maven *Build life-cycle*. Folder src sadrži predefinisane podfoldere. U folderu

- src/main/java – se nalazi izvorni kod aplikacije
- src/main/resources – se resursi koji nisu Java fajlovi, npr. property fajlovi, konfiguracioni fajlovi
- src/main/webapp – se nalazi web.xml fajl (deployment descriptor), folder WEB-INF, statički resursi, slike, CSS fajlovi, JavaScript fajlovi, biblioteke, itd. Takođe, tamo se nalazi i izvorni kod za kreiranje prezentacionog sloja aplikacije (fajlovi: .jsp, .jspf, .jspx, .ftl, .ftlh, .ftlx, .html). Spring može da koristi različite Template Engine za kreiranje prezentacionog sloja MVC aplikacije. Za kreiranje dinamičkih HTML stranica može se koristiti JavaServer Pages (.jsp), FreeMarker (.ftl), Thymeleaf (.html),… Sadržaj webapp foldera bi odgovarao folderu WebContent kod Servleta
- src/test/java – se nalaze Unit testovi izvorni kod aplikacije

Maven omogućava *Dependency management* tj. automatsko upravljanje dependency-ima za korišćene biblioteke. Kada bi kod klasičnog projekta koristili biblioteku neophodno bi bilo fizički ubaciti biblioteku u projekat i dodati je u *build path* projekta. U slučaju da ubačena biblioteka zavisi od nekih drugih biblioteka neophodno bi bilo pribaviti i ubaciti i ostale tražene biblioteke. Proces ponavljati sve dok više nema zavisnih biblioteka.



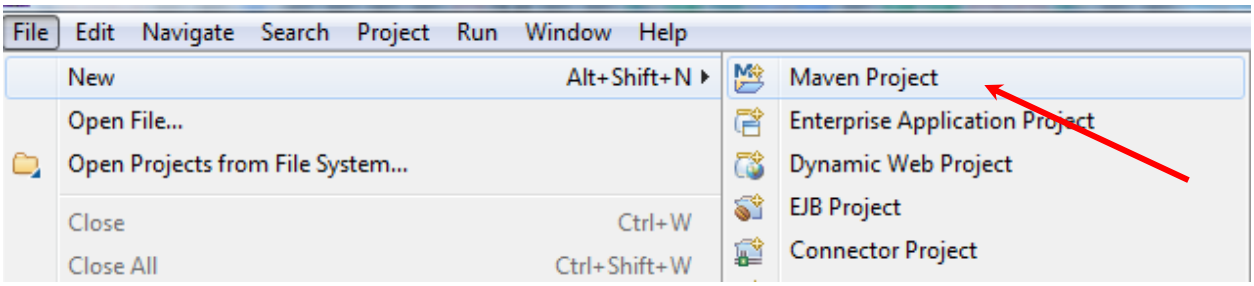
Maven alatu je neophodno samo navesti da projekat zavisi od neke biblioteke. Alat će sam naći tu biblioteku, pruzeti je, ubaciti u projekat, dodati je u *build path* projekta. A zatim će ponoviti isti postupak za sve biblioteke od kojih preuzeta bibliteka zavisi, i ponoviti isti process za novopreuzete bibliteke, sve dok više ne bude zavisnosti. Takođe Maven će detektovati i obavestiti nas o situacije da u projektu treba da se preuzmu različite verzije iste biblioteke, što nije moguće.



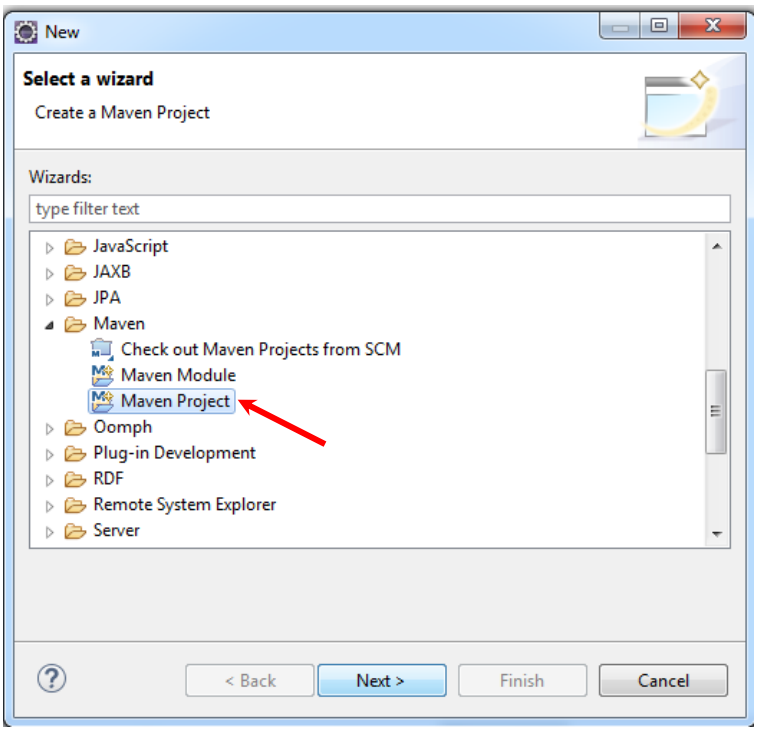
Na kursu će se koristiti Maven Eclipse plug-in.

Kreiranje novog Maven veb projekta sa programom Eclipse

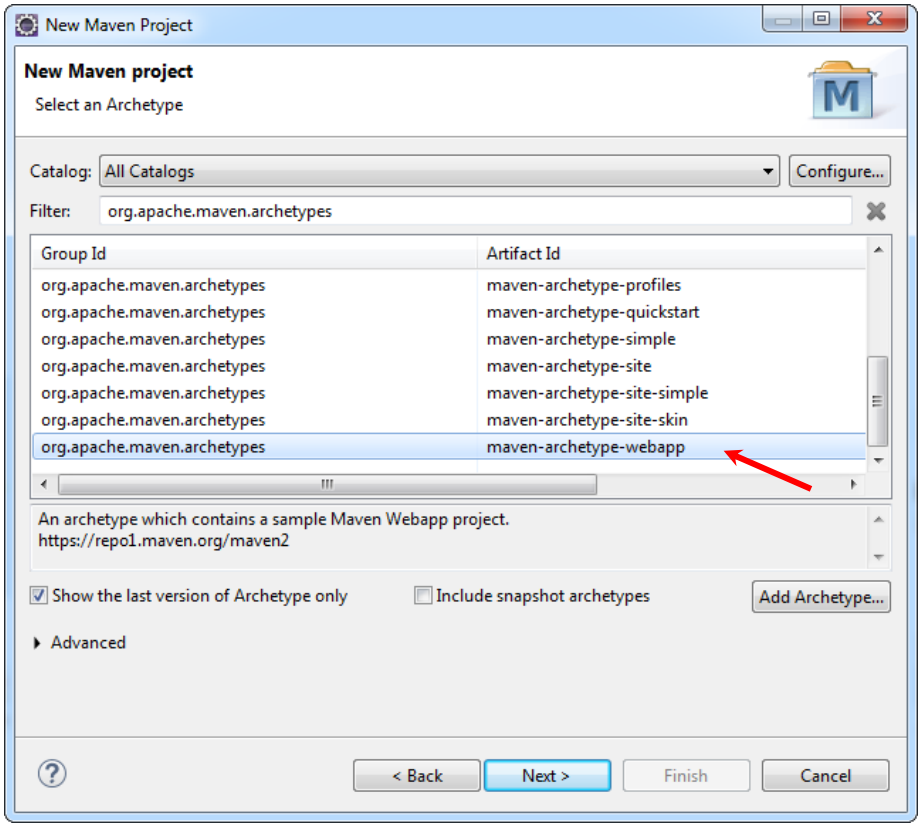
Iz *Java EE* perspective, kliknite *File->New-> Maven Project*



Iz *Java* perspective, kliknite *File->New->Other*, a kada se otvori novi prozor u njemu odaberite *Maven->Maven Project*, pa *Next*



Otvoriće se novi prozor u kome samo klik *Next*.
U prikazanom prozoru treba da se odabere tip Maven arhitekture koji odgovara veb projektu. Odabrati za *Group Id* vrednost *org.apache.maven.archetypes* a za *Artifact Id* vrednost *maven-archetype-webapp*, pa *Next*. ***Group Id* označava grupu projekta, najčešće nešto vezano za organizaciju. Grupa *maven.archetypes* predstavlja konfuguracije Maven projekta koja se koristi za kreiranje određenog tipa aplikacije. *Artifact Id* prestavlja osnovnu gradivnu jedinicu u Maven alatu.**

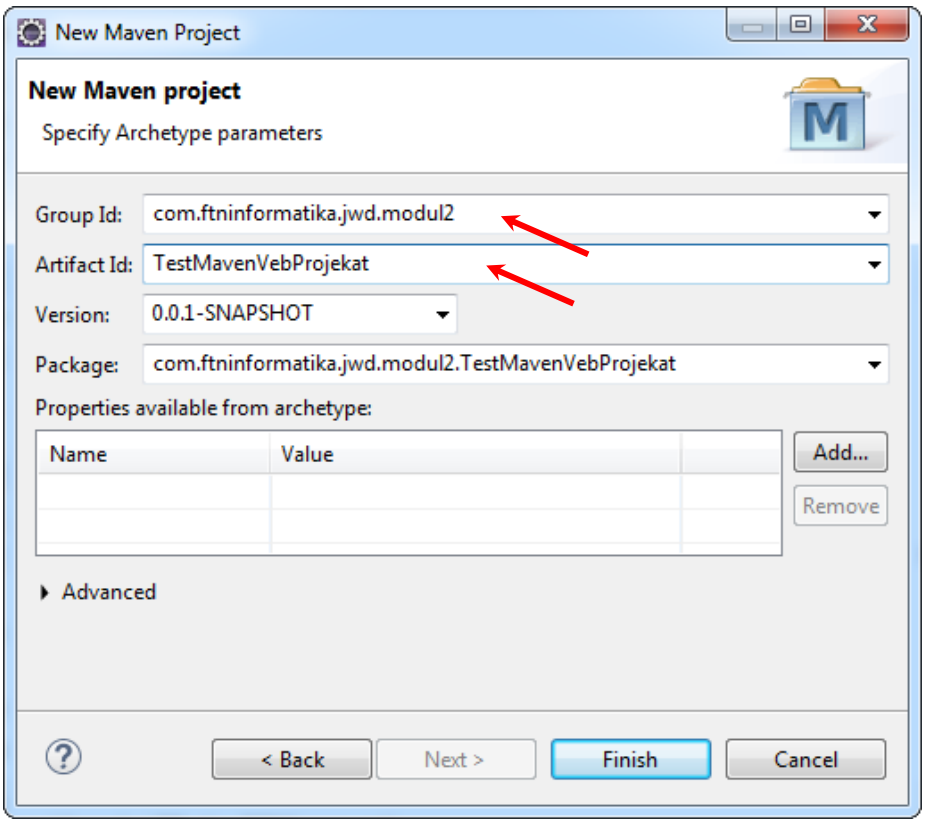


U novom prozoru treba da se unesu vrednosti za *Group Id*, *Artifact Id* i *version* za projekat koji se kreira.

Pomenute vrednosti imaju za cilj da se za projektnu deliverablu - *project deliverable* (jar/war/ear …) definiše jedinstveni identitet u repozitorijumu. **Projekat koji se kreira predstavlja jedan Maven artifakt, a drugi projekti od kojih kreirani projekat će zavisiti predstavlju druge Maven artifakte.**

Prateći pomenutu logiku, neko treće lice može korisiti naš projekat u svom projektu, samo treba da u njegovom projektu navede da on zavisi od našeg Maven artifakta i ceo naš projekat će se pevući kod tog trećeg lica.

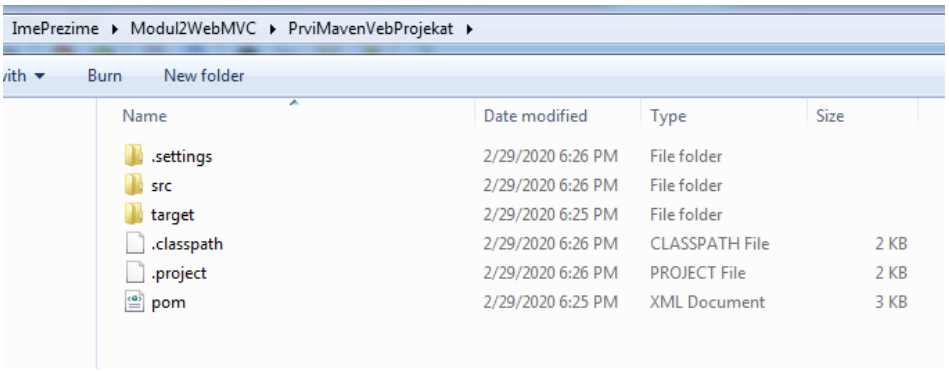
Za *Group Id* unesite *com.ftninformatika.jwd.modul2*, *Artifact Id* unesite *TestMavenVebProjekat*, za verziju neka ostane predložena vrednost ili sami možete definisati verziju za projekat. Maven će predložiti naziv korenskog paketa koji se koristi u projektu, nastaje spajanjem vrednosti *Group Id* i *Artifact Id*. Klik na *Finish*.



Maven će preuzeti neophodne stvari sa internet, pogledati desni donju ugao. Sačekajte dok se proces preuzimanja ne završi.

– Note: projekat ima ikonicu M

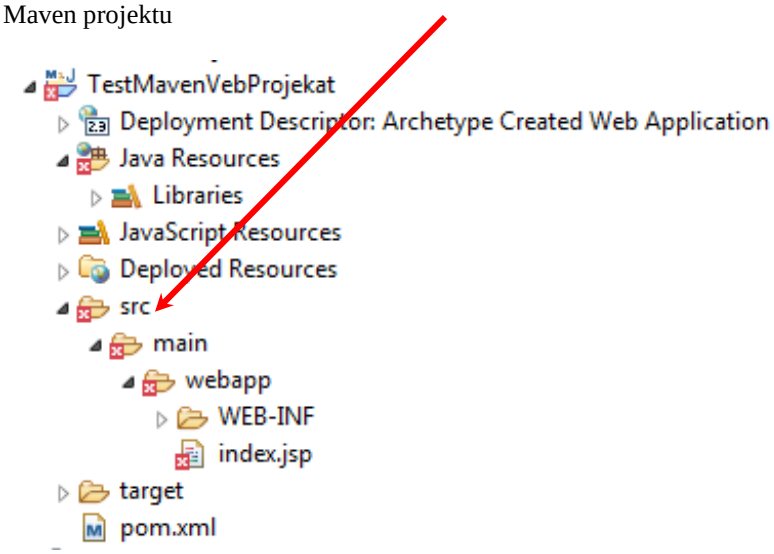
Direktorijumi *src* i *target* i datoteka *pom.xml*, postoje fizički na disku. Sve stavke projekta što se vide u eklipsi predstavlju delove projekta koje je prepoznao *Eclipse Maven Plug-in*.



Kao što je već ranije rečeno fajl pom.xml sadrži konfiguraciju Maven projekta. U folderu *src* se nalzi sve što je isprogramirano, dok se u folderu *target* nalzi sve što je rezultat Maven *Build life-cycle*.

Obrišite index.jsp fajl

Potrebno je kreirati ostale foldere koji nedostaju a u skladu su sa predefinisanim prostornim rasporedom foldera u Maven projektu



Kreirajte

- src/main/java - u folderu src/main folder java (nalaze se Java klase za naš projekat)
- src/test/java – u folderu src kreirajte folder test i u njemu folder java (nalaze Unit testovi za naš projekat)
- src/main/resources – u folderu src/main kreirajte folder resources (nalaze resursi koji nisu Java fajlovi, npr. property fajlovi, konfiguracioni fajlovi za naš projekat)

Primetite da folder *src* postoji u gornjem delu i u donjem delu projekta.

U gornjem delu se nalazi sve što je ubačeno u Build Path projekta. Src u folderu Java Resources kao folder za Java pakete

U donjem delu projekta se nalazi kao fizička reprezentacija foldera na disku.

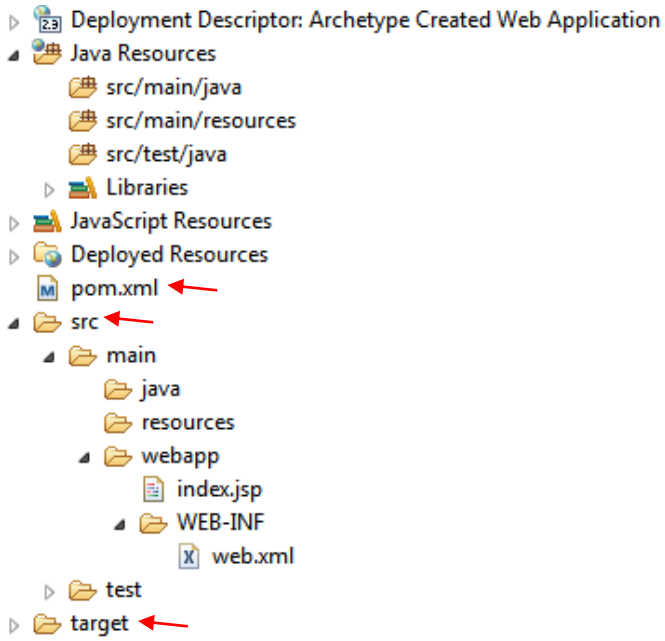
U gornjem delu se kreiraju fajlovi koji će se uvesti kao Java resursi za projekat (svi Java fajlovi se kreiraju ovde), dok se u donjem delu kreiraju obični fajlovi koji se samo nalaze na toj lokaciji na disku.

Može se folder iz donjeg dela dodati u gornji deo sa Desni klik->Build Path->Use as SourceFolder

Stavke projekta ne moraju predstavljati fizičke direktorijume ili datoteke na disku

- Doployment deskriptor – je u stvarnosti fajl web.xml, sadrži opis naše veb aplikacije za potrebe servera
- Deployed Resources – ne postoji folder na disku, prikazuje sve resurse koji će biti postavljeni na veb server
- Java resuorces – ne postoji folder na disku

Detaljnije prikaz structure direktorijuma unutar maven projekta



Objašnjavanje pom.xml (sadrži konfiguraciju Maven projekta). Izgled pom.xml fajl

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.ftninformatika.jwd.modul2</groupId>
  <artifactId>TestMavenVebProjekat</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>TestMavenVebProjekat Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```
  <build>
    <finalName>TestMavenVebProjekat</finalName>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults
(may be moved to parent pom) -->
      <plugins>
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-war-plugin</artifactId>
          <version>3.2.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

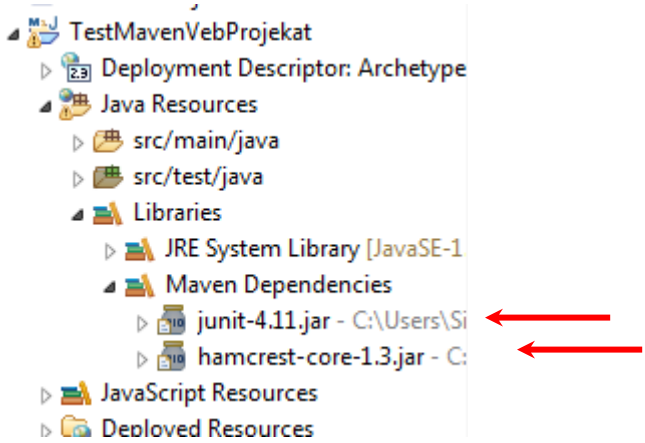
Otvoriti pom.xml sekcija pom.xml. Postoje razni tagovi.

- `<modelVersion>4.0.0</modelVersion>` - predstavlja verziju pom modela koji se koristi
- Group Id, Artifact Id, version – uneti kroz process kreiranja projekta
- `<packaging>war</packaging>` - označava da će Maven *Build life-cycle* u fazi *package* kreirati projektnu deliverablu kao war arhiva (jar/war/ear…)
- `<name> TestMavenVebProjekat Maven Webapp</name>` - opisno ime projekta
- `<url>http://www.example.com</url>` - link do dokumentacije projekta
- `<properties>` - imenovane varijable koje se mogu koristiti u ostatku fajla sa ${imePropetija}
- `<dependencies>` - navode se biblioteke od kojih projekat zavisi. Za svaku zavisnost se definiše jedinstveni identitet (`<groupId>`, `<artifactId>`, `<version>`) i opseg (`<scope>`) kojim se navodi za koji deo *Build life-cycle* se koristi ta zavisnost. Scope može da se izostavi, tada zavisnost važi za sve faze

Build life-cycle. Trenutno je navedena zavisnost od JUnit biblioteke. Za konkretnu JUnit biblioteku se navodi opseg *test*, što znači da se JUnut biblioteka neće koristiti za narednu fazu koja dolazi posle *test* faze tj. za fazu *package*.

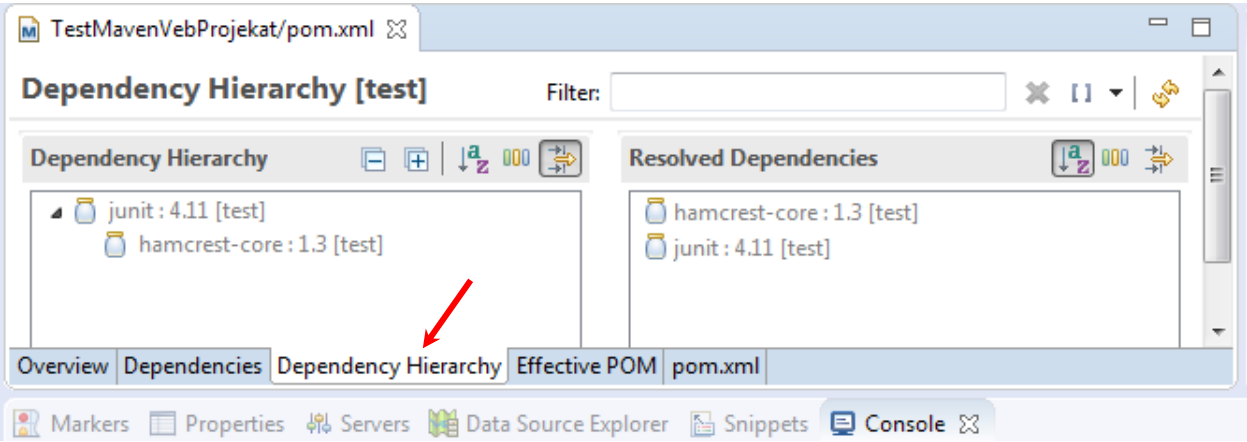
- `<build>` - opisuju se podešavanja vezana za bildovanje deliverable
- `<finalName>` - ime deliverable
- `<pluginManagement> <plugins>` - sekcija za priključcima, koji se sve priključci koriste za bildovanje projekta u projektnu deliverablu
- `<plugin>` - priključak se identifikuje sa `<groupId>`, `<artifactId>` i `<version>`. Za njega se može postaviti dodatna konfiguracija sa `<configuration>`. Trenutno su navedene različite verzije maven priključaka za različite delove Build life-cycle.

Zavisnosti se mogu videti u okviru dela *Libraries->Maven Dependencies*. Tu se može videti junit jar arhiva jer je ona navedena u pom.xml u tagu <dependencies>. Pored nje postoji i hamcrest-core jar arhiva koja je neophodna za rad JUnit biblioteke. Postojanje hamcrest-core jar arhive je dobar primer Maven Dependency management u akciji.



Pored taba *pom.xml* postoji i tab *Effective POM*. *Effective POM* se dobija kada se na definisani *pom.xml* dodaju pomovi svi projektnih zavisnosti, i od zavisnosti pomovi njihovih zavisnosti,… to spoji u jedan fajl.

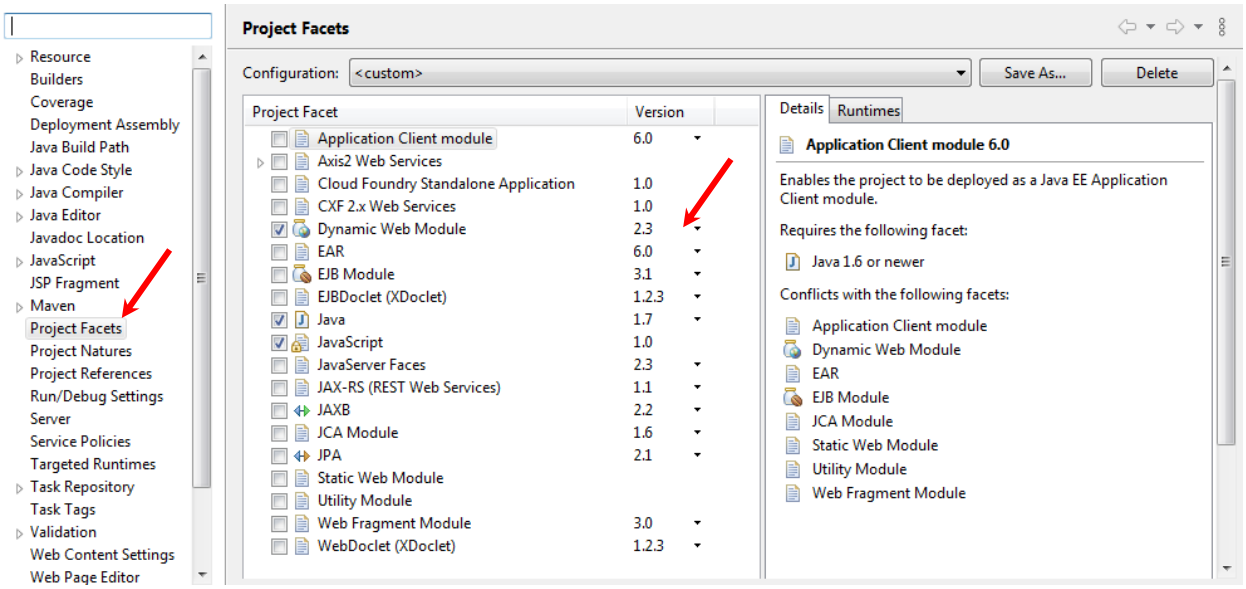
Tab *Dependacy Hierarchy* na jednostavan način vizuelno prikazuje zavisnosti u projektu.



U projektu je prethodno postojala greška jsp fala u *index.jsp*

The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path

Da bi se greška uspešno rešila a da se **ne obriše** fajl *index.jsp* neophodno je u pom.xml fajlu navesti zavisnosti koje bi omogućile rad sa JSP tehnologijom. Prvo proveriti u Project Facet podešavanje projekta (Desni klik na projekat *Properties->Project Facets->Dynamic Web Module*). Zatim u tagu `<dependencies>` neophodno bi bilo dodati zavisnost od Servleta i JSP tehnologije, tako da ta zavisnost odgovara serverskoj specifikaciji navedenoj u Project Facet.
Desni klik na projekat pa Maven->Update Project ...




```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
       <groupId>javax.servlet</groupId>
       <artifactId>servlet-api</artifactId>
       <version>2.3</version>
       <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.1</version>
      <scope>provided</scope>
  </dependency>
</dependencies>
```