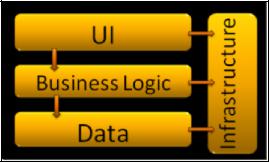
**Arhitecturi**

1. **Multilayered architecture** este o arhitectură de tip client-server in care prezentarea, aplicarea, procesul și data management-ul functiunilor sunt separate fizic, conform Bass 2003.

*Fig 4.2 Arhitectura Multitier*

<https://www.incredible-web.com/blog/the-onion-architecture/>

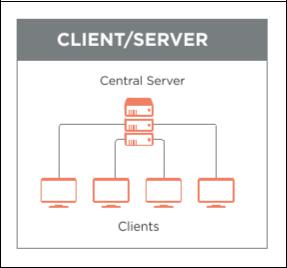
Scurtă prezentare a Layer-elor:

* **Presentation User Interface Layer** este cel mai înalt nivel al aplicațieiși are rolul de ainteracționa cu utilizatorii de aceea va transforma date de intrare și de ieșire în date interpretabile de utilizatori. Prinn intermediul acesteia utilizatorii interacționează în diferite moduri cu servicii din aplicație acționând în mod direct în fucție de nevoile fiecăruia, de exemplu: actorul Receptor poate interoga baza de date cu ajutorul butonului ***Cautare***.
* **Data Layer** este secțiunea în care sistemul interacționează cu mediile de stocare a datelor, încazul de față baza de date de tip MySQL asigurând un API (Applicațion Programming Interface) nivelului de business. În cadrul Data Layer se desfășoare evenimente de Data Logic ca de exemplu interogarile asupra baze de date.
* **Applicațion Business Logic Layer** este locul unde se **întâmplă** majoritatea proceselor sau acalculelor precise privind factoruul de evlaluare, validare și decizie a aplicației.

1. **Arhitectura client-server** este un model software format din două părți, sisteme client și sisteme de servere, ambele comunicând printr-o rețea de calculatoare sau pe același computer. O aplicație client-server este un sistem distribuit alcătuit din software client și software server. Aplicația client-server oferă o modalitate mai bună de a partaja volumul de lucru. Procesul client inițiază întotdeauna o conexiune la server, în timp ce procesul serverului întotdeauna așteaptă solicitările de la orice client. Atunci când atât procesul clientului cât și procesul serverului rulează pe același computer, aceasta se numește configurație unică a scaunului.

Relația client-server este modul în care clientul face o cerere către server și modul în care serverul poate accepta aceste solicitări, le poate procesa și le poate returna clientului. Interacțiunea dintre client și server este deseori descrisă folosind diagrame de secvențe. Tipul de bază al arhitecturii de software client-server utilizează numai două tipuri de gazde: clienți și servere. Acest tip de arhitectură este denumit uneori ca două niveluri. Arhitectura pe două niveluri înseamnă că clientul acționează ca un singur nivel, iar procesul de server acționează ca și celălalt nivel.

Arhitectura software-ului client-server a devenit unul dintre modelele de bază ale rețelelor de calcul. Multe tipuri de aplicații au fost scrise utilizând modelul client-server. Funcțiile standard de rețea, cum ar fi schimbul de e-mail, accesul la Internet și accesul la baze de date, se bazează pe modelul client-server. De exemplu, în cazul nostru, clientul este aplicația de pe orice dispozitiv care are ca sistem de operare android iar serverul este un program instalat pe orice sistem de calcul pe care ruleaza Windows 7. Clientul poate accesa serverul oriunde ar fi acesta în lume.



*Fig 4.3 Arhitectura Client-Server*

<https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/>

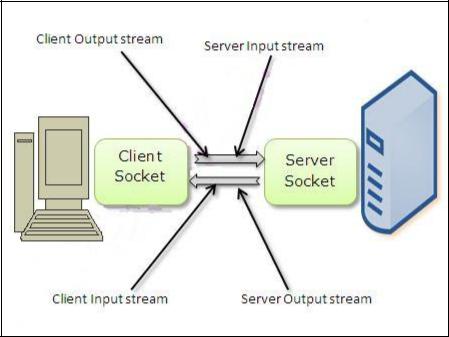
**Conectarea client-server**

Adresele URL și URLConnections oferă un mecanism relativ la nivel înalt pentru accesarea resurselor pe Internet. Uneori, programele necesită o comunicație de rețea la nivel inferior, de exemplu, atunci când doriți să dezvoltăm o aplicație client-server.

În aplicațiile client-server, serverul oferă unele servicii, cum ar fi procesarea interogărilor bazei de date sau trimiterea prețurilor curente ale acțiunilor. Comunicarea care are loc între client și server trebuie să fie fiabilă. Adică, nu se pot pierde date și trebuie să ajungă pe partea clientului în aceeași ordine în care serverul la trimis. TCP oferă un canal de comunicare punct-la-punct, pe care aplicațiile client-server de pe Internet le utilizează pentru a comunica între ele. Pentru a comunica prin TCP, un program client și un program de server stabilesc o conexiune între ele. Fiecare program leagă un soclu la capătul conexiunii. Pentru a comunica, clientul și serverul citesc și scriu în soclul legat de conexiune.

Ce este un soclu?

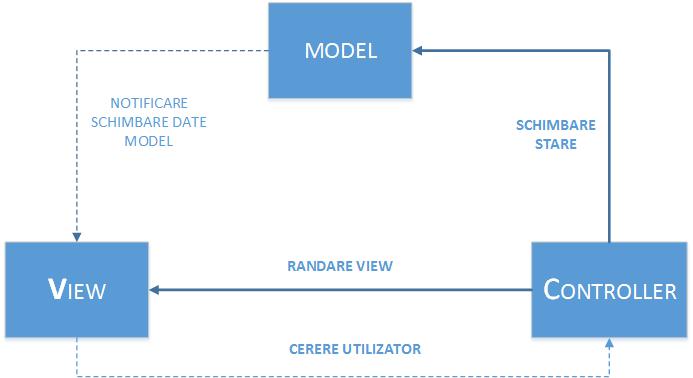
Un soclu este un punct final al unei legături de comunicare bidirecționale între două programe care rulează în rețea. Clasele de clase sunt utilizate pentru a reprezenta conexiunea dintre un program client și un program de server. În aplicația noastră am folosit pachetul java.net care oferă două clase Socket și ServerSocket implementând partea de client a conexiunii și partea de server a conexiunii, respectiv.



*Fig 4.4 Conexiune Java Socket*

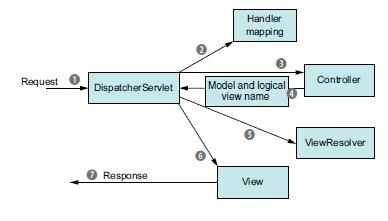
<http://www.softpost.org/java/socket-programming-in-java/>

3. **MVC provine combinația Model-View-Controller** și se bazează pe principiul **separarea** **conceptelor**( ”separation of conceirn”), astfel aplicația este compusă din cele 3 părți de Model,View și Controller, fiecare cu o responsabilitate proprie și utilă pentru mentenanța și flexibilitatea de a aduce schimbări fără a modifica codul în multe locuri, ci doar unde este nevoie. **Modelul** reprezintă locul în care sunt stocate datele, **View**-ul este locul în care datele sunt vizualizate sau randate de către Controller. **Controller**-ul poate fi definit ca o combinație între Model și View, el ia datele din model după anumite reguli( de cele mai multe ori de business), acțiuni ale utilizatorului(input) și pe baza lor furnizează un rezultat(output) sub forma unui View.



În cadrul aplicației de față am ales să folosesc framework-ul **Spring MVC**, dezvoltat de Oracle pe baza limbajului de programare Java, ideal pentru dezvoltarea aplicațiilor web și care implementează design pattern-ul MVC, separând conceptele și rolurile. Printre avantajele Spring MVC se numără și librăria **JSP**(Java Server Pages) cu formatările cu care vine, flexibilitatea și faptul că suportă o reutilizare destul de mare.

Spring MVC se bazează pe un Servlet unic numit **Dispatcher Servlet**, care gestionează toate cererile HTTP și răspunsurile, oferind toate funcționalitățile server ale unei aplicații web. Craig Walls definește în cartea *Spring in Action*, următoarea diagramă reprezentativă pentru Dispatcher Servlet:



1. În acest pas, utilizatorul aplicației web **trimite o cerere din browser** către server care ajunge la **Dispatcher Servlet**. Exemple de cereri pot fi informațiile trimis în cadrul formularului de login, adăugare produse sau accesarea altei pagini. Odată ce ajunge la Servlet, ca și în celelalte framework-uri web Java, un singur servlet deleagă responsabilitățile către alte componente.
2. Pentru a trimite cererea către Controller, DispatcherServlet consultă **Handler Mapping** pentru a vedea URL-ul de pe care a venit cererea. În cadrul acestui URL, se definește un

RequestMapping, cu o denumirea de care are nevoie controller-ul( în cazul de față avem:

*RequestMapping=”/addProduct”, RequestMapping=”/register”)* și metoda de pe care avenit cererea(GET, POST, DELETE, PUT), exemplu: *RequestMethod=* *requestMethod.GET .*

1. Odată ce se va ajunge la **Controller**, acesta va delega de cele mai multe ori informația către componenta de business și se așteaptă procesarea informației. Controller-ul va fi o clasă concretă reprezentativă pentru un obiect din model și va conține **adnotarea @Controller** pentru a se diferenția.

În cazul de față, fiecare metodă din Controller delegă informația către un model de business, abreviat cu **DAO (Data Acces Object)**, care va face legătura cu baza de date. Se poate observa că fiecare clasă Controller (UserController, ProductController) va conține un membru de tip DAO în clasa sa (UserDAO, ProductDAO – interfețe și vor realiza upcast).

1. După preluarea datelor de la componenta de business, Controller-ul are responsabilitatea de a randa informația primită ca și output către View. Se observă creare unui obiect de tip **ModelAndView**, căreia i se va seta un nume și va returna obiectul respectiv.
2. Consultarea **ViewResolver**-ul care va știi unde să mapeze la nivel logic implementarea are loc aici, este posibil ca pagina să fie una JSP sau HTML, tag-urile se pot imbrica. ViewResolver-ul este definit în momentul configurării servlet-ului DispatcherServlet.
3. 7) În cadrul acestor pași, view-ul este returnat către client, preferabil să fie o pagină cu un format JSP, pentru a suporta obiectele trimise de Controller.

Având în vedere componentele menționate anterior, structura aplicației este alcătuită din **partea de client**( browser), **partea de server** (fiind o aplicație Java Web Enterprise), **driver-ul JDBC** (Java Database Conectivity–conectorul la baza de date) și **baza de date** de tipMySQL.

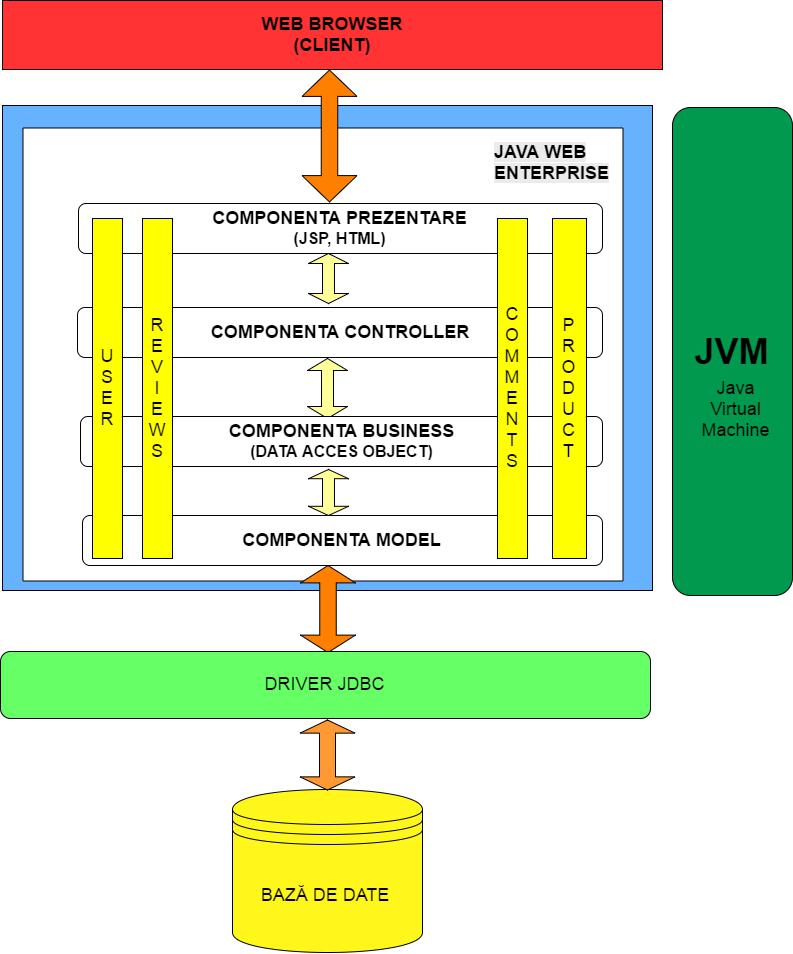
Pe partea de server se poate observ existența unei aplicații de tip **Java Enterprise**, precum și structura prezentată anterior când a fost detaliat DispatcherServlet. Aplicația se va conecta la o baza de date de tip MySQL aflată pe același calculator (în cazul extinderii la nivel comercial va fi pe un server extern) cu ajutorul JDBC. **Java Database Conectivity** facilitează conexiunea la baza de date prin furnizarea serverului unde este stocată baza de date și a credențialelor de conetare (utilizator și parolă).

**Componenta Model** va conține câte o clasă pentru fiecare tabelă din baza de date, iar încadrul ei membrii vor avea același nume și tip ca și coloanele din baza de date (*User,Product,* *ProductType,UserProduct*). Astfel căîn componenta Business, se va utiliza de tip Model oride cât ori vom face o operație.

**Componenta Business** este reprezentată de **interfețe de tip DAO** și **clase specifice DAOImpl** care descriu operațiunile care se pot face cu un model (*UserDAO, UserDAOImpl*).Într-o interfață DAO se regăsește definiția metodei, iar corpul ei va fi în DAOImpl; este importantă separarea logicii, căci componenta Business va conține informații doar despre Model și incapsularea este garantată.

**Componenta Controller** conține clase detip Controller( care au această adnotare) și pebaza logicii și regulilor de business, va randa operația și va returna un obiect de tip ModelAndView – regăsit în componenta prezentare.

**Componenta Prezentare (View)** va conține fișiere de tip JSP sau HTML( de preferat JSPîn cazul folosirii Spring MVC) care vor fi vizualizate de către client în browser. Toate operațiile din server sunt gestionate de JVM (Java Virtual Machine).



**Componenta Model** conține clase cu aceeași structură ca șitabelele din baza de date.Fiecare clasă va conține un membru ca și numele coloanei din baza de date, iar ca și metode getter și setter pentru fiecare proprietate. Pentru eficiență, **incapsularea** este garantată prin declararea membrilor privați, iar metodele vor fi publice și se vor modifica valorile membrilor. Reprezentarea claselor din model se vor face cu **diagrame UML**

**Componenta Business** conține interfețepentru fiecare entitate din Model (DAO) cufuncțiile ei specifice și implementări (DAO Impl) care conțin corpul metodelor. Metodele din DAO vor executa o operație adăugare, șterere, modificare sau vizualizare după anumite criterii.

**Componenta Controller** conține clase cu adnotarea*@Controller*, responsabile să preiadatele din Model, le va prelucra și va returna un obiect de tip ModelAndView (pagină JSP sau HTML, după caz) . Fiecare metodă din clasa Controller va avea adnotarea *@RequestMapping* , care va indica numele cererii ( *value =* *“/exemplu”* ), precum și metoda pe care vine cererea ( *method= RequestMethod.GET, method =RequestMethod.POST*). De asemenea, între membrii seva regăsi obiectul de business corespunzător modelului pe care controller-ul în prelucrează.

**Componenta View** conține fișiere de tip HTML sau JSP, returnate de Controller, fiindobiecte de tip ModelAndView. Spre deosebire de fișierele HTML, cele JSP au o flexibilitate mai mare prin adăugarea structurilor de control (ex: *foreach* - parcurgere obiecte din cadrul unei liste). De asemenea, pentru a se evita redundanța, pagini precum footer sau header pot fi scrise separat și incluse în fiecare pagină.