Tema 1

## -Proiectarea sistemelor software complexe-



Vlad-Iosif Orbulescu

Anul IV, IS

Politehnica Timisoara

Facultatea de Automatica si Calculatoare

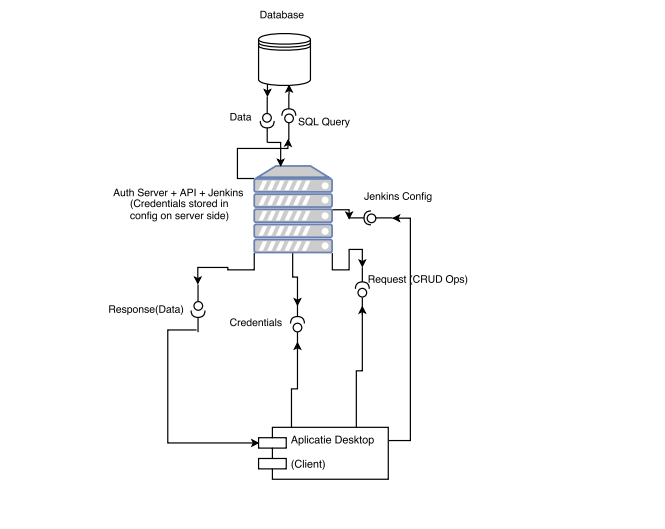
## 1.Prezentarea cerintelor funcționale si non-functionale si identificarea acelor cerinte care influenteaza arhitectura:

Sistemul pe care l-am ales este unul care se ocupa cu managementul inventarului unui magazin de piese auto. Utilizatorul acestui software este patronul magazinului a carui datorie este sa modifice inventarul in functie de stocul actual al magazinului.

1. Cerintele functionale ale sistemului:
   1. Programul trebuie sa permita adaugarea unui produs nou
   2. Programul trebuia sa permita editarea detaliilor si a stocului unui produs existent
   3. Programul trebuie sa permita stergerea unui produs, ca rezultat al indisponibilitatii permanente in stoc a produsului
   4. Programul trebuie sa permita cautarea unor produse dupa anumite criterii
   5. Programul trebuie inregistreze vanzari si achizitii pentru produse
   6. Programul trebuie sa genereze rapoarte lunare cu date despre vanzari, cheltuilei si stoc disponibil al magazinului
2. Cerintele non-functionale ale sistemului:
   1. Programul trebuie sa permita back-up-ul tuturor datelor legate de inventar
   2. Sistemul necesita autentificare cu un singur set de credentiale, date de catre patronul firmei
   3. Server-ul unde se afla baza de date si API se afla in incinta magazinului
3. Cerintele care influenteaza arhitectura:
   1. Se va folosi modelul Client-Server pentru ca aplicatia sa acceseze date
   2. Se va folosi un framework numit Jenkins pentru partea de server pentru a face back-up automat si periodic, configurat din aplicatia client
   3. Pentru comunicarea cu serverul si baza de date se va creea un API

## 

## 2.Descompunere in componente, definirea responsabilitatilor componentelor si a relatiilor dintre ele:

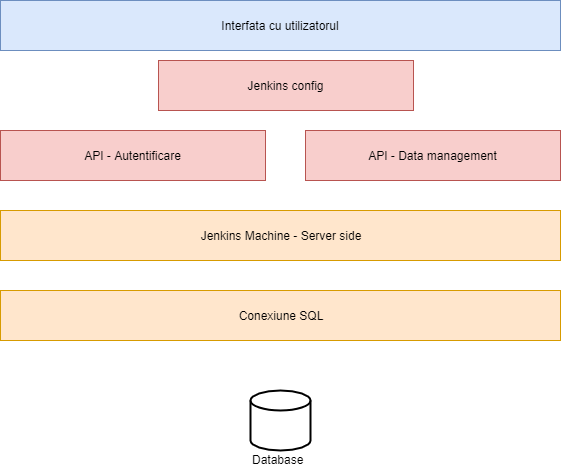


**fig.1**

Datorita folosirii modelului de design ‘Client-Server’ , primele 2 componente vor fi, evident, aplicatia client si aplicatia server. Partea de server include un API pentru a accesa datele din baza de date, un serviciu care va autentifica utilizatorul in aplicatie si o parte de Jenkins care se va ocupa cu back-up automat. Toate aceste parti terte vor fi accesate din aplicatia client, utilizatorul putand sa apeleze API-ul pentru a primi diferite date si totodata va putea sa configureze masina Jenkins pentru a alege periodicitatea back-up-ului.

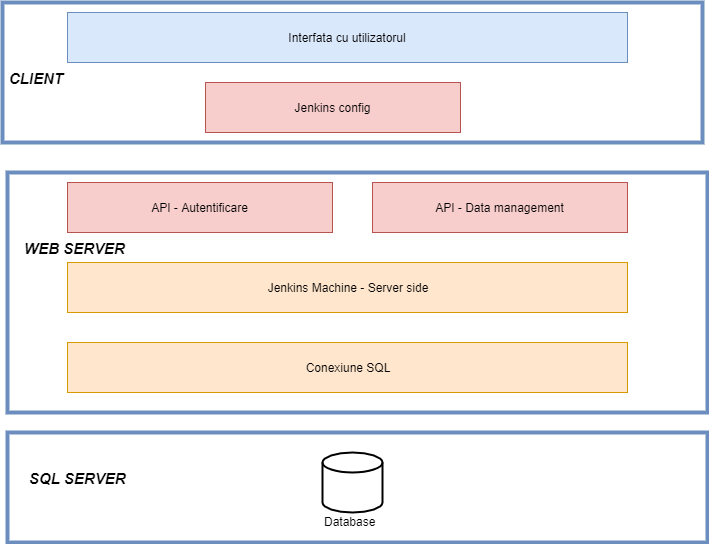
## 3.Prezentarea sistemului software din doua perspective:

* Perspectiva logica: Cele mai semnificative componente privite din perspectiva logica a arhitecturii sistemului, sunt dupa cum urmeaza : interfata cu utilizatorul (aplicatia desktop), API-ul (care contine toate metodele necesare pentru generarea rapoartelor si redarea tuturor datelor necesare, cat si modificarea acestora), conexiunea server-ului cu baza de date si intr-un final, baza de date.



**fig.2**

* Perspectiva proces: Privind aplicatia din perspectiva de proces a arhitecturii, flow-ul de informatii decurge in urmatorul fel : autentificare -> comenzi de la aplicatia desktop -> interpretarea comenzii la API -> interogarea bazei de date -> intoarcerea raspunsului catre utilizator. Avand in vedere ca sistemul trebuie sa fie capabil sa execute operatii CRUD orientate pe un obiect (sau o multime de obiecte) o data per comanda, nu avem nevoie de concurenta in sistem, astfel toate request-urile vor fi tratate secvential.



**fig.3**

## 4.Identitifcarea celor mai importanți 3 indicatori de calitate, specificarea masurii alese pentru fiecare indicator de calitate si argumetarea alegerii:

Cei 3 cei mai importanti indicatori de calitate prezenti in sistemul prezentat sunt :

1. Scalabilitatea - datorita impartirii celor doua mari task-uri ale sistemului intr-un component care ‘cere’ si unul care ‘da’, adaugarea de noi functionalitati este foarte usoara daca va fii cazul.
2. Toleranta la modificari - Din punctul meu de vedere, scalabilitatea implica toleranta la modificari. In cazul in care problema pentru care problema pentru care sistemul a fost proiectat creste si acesta poate suporta noi implementari in mod usor, atunci acesta este tolerant la modificari.
3. Integrarea - avand in vedere ca deja avem un sistem scalabil si tolerant la modificari, si totodata acesta este construit pe o arhitectura client-software si este bine ‘compus’, dezvoltarea acestuia pentru o gama mai larga de task-uri si pentru diferite domenii nu este greu de realizat.

## 5.Identificarea tehnologiilor middleware folosite pentru a comunica intre componente, argumentarea alegerilor:

Pentru sustinerea API-ului folosim tehnologia .NET Web API, acesta fiind hostat pe server-ul intern si manageriat prin SSMS. Pentru comunicarea cu baza de date, de asemenea vom folosi metode integrate in .NET Web API, si anume diagrame structurate ADO Framework si Linq.Pentru configurarea back-up-ului automat vom folosi Jenkins, configurat cu platforma sa de pe PC si hostat pe server. Autentificarea se va face printr-un algoritm Diffie-Hellman pentru arhitectura client-server. Aceasta combinatie de tehnologii este usor de folosit impreuna si ne pune la dispozitie tot ce avem nevoie pentru a realiza o comunicatie/conexiune securizata si stabila intre componente necesare sistemului.

## 6.Identificarea pincipalelor modele și stiluri arhitecturale folosite, argumentarea alegerilor:

Cum am mentionat si in cerintele anterioare, principalul model arhitectural folosit este client-server, deoarece avem nevoie de comunicare sincrona, posibilitati de securizare a accesului si error-handling eficient in momentul ‘manevrarii’ datelor. Avand in vedere faptul ca toate componentele acestui sistem, fiind separate, cel mai bine li se preteaza sa fie unite prin acest model, rezulta intr-un final un sistem stabil si bine sa se adapteze cerintelor date.

## 7.Prezentarea scenariilor de validare a arhitecturii:

Arhitectura prezentata este valida, deoarece luand in considerare urmatoarele scenarii, fiecare cerinta este corect indeplinita:

* Sistemul trebuie sa fie capabil sa stocheze si sa aiba acces la date usor si oricand -> interogarea bazei de date de catre API (server)
* Sistemul trebuie sa aiba o interfata usor de folosit -> aplicatie desktop care acceseaza API (client)
* Sistemul trebuie sa fie capabil sa treaca cu bine de orice eroare, cu afisaj user friendly si sa isi poata continua sesiunea de rulare -> API error handling (server side)
* Sistemul trebuie sa poata salva toate datele periodic -> Jenkins
* Sistemul trebuie sa fie securizat si sa ofere rapoarte -> Autentificare , acces restrictionat la baza de date (API handling)