<E-commerce>

Analysis and Design Document

Student:Savan Catalin

**Group:30235**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <10/04/18> | <1.0> | <details> | <Savan Catalin> |
| <25/05/18> | <2.0> | <iteratia finala> | <Savan Catalin> |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 5

2.1 Conceptual Architecture 5

2.2 Package Design 5

2.3 Component and Deployment Diagrams 5

III. Elaboration – Iteration 1.2 7

1. Design Model 7

1.1 Dynamic Behavior 7

1.2 Class Design 9

2. Data Model 11

3. Unit Testing 12

IV. Elaboration – Iteration 2 12

1. Architectural Design Refinement 12

2. Design Model Refinement 13

V. Construction and Transition 15

1. System Testing 15

2. Future improvements 15

VI. Bibliography 15

# Project Specification

*[Present the project specification]*

1. *Inregistrare*

* *Se realizeaza prin intrefata de logare in cazul in care nu aveti un cont in baza de date*

1. *Logare*

* *Prin email si parola , acest lucru permite navigarea prin site.ul de cumparaturi online cu rolul de client autentificat acest lucru permite mai multe activitati posibile.*

1. *Interfata principal*

* *Prin aceasta interfata se navigheaza prin produsele existente in stoc , acestea fiind afisate pe site cu pretul si o mica descriere a acestora.*

1. *Bara de cautare*

* *Cautarea unui produs cu ajutorul tag.urilor asigante fiecarui produs.*

1. *Pagina produs*

* *In aceasta pagina se prezinta produsul cu o imagine descriere si pretul acestuia.*

1. *Logare Administrator*

* *Precum logare unui client doar ca aceasta logare se face prin interfata specifica doar a administratorului .*

1. *Interfata administrator*

* *In aceasta interfata se pot alege diferite actiuni precum verificare stoc ,adaugare produs verificare comenzi ,etc.*

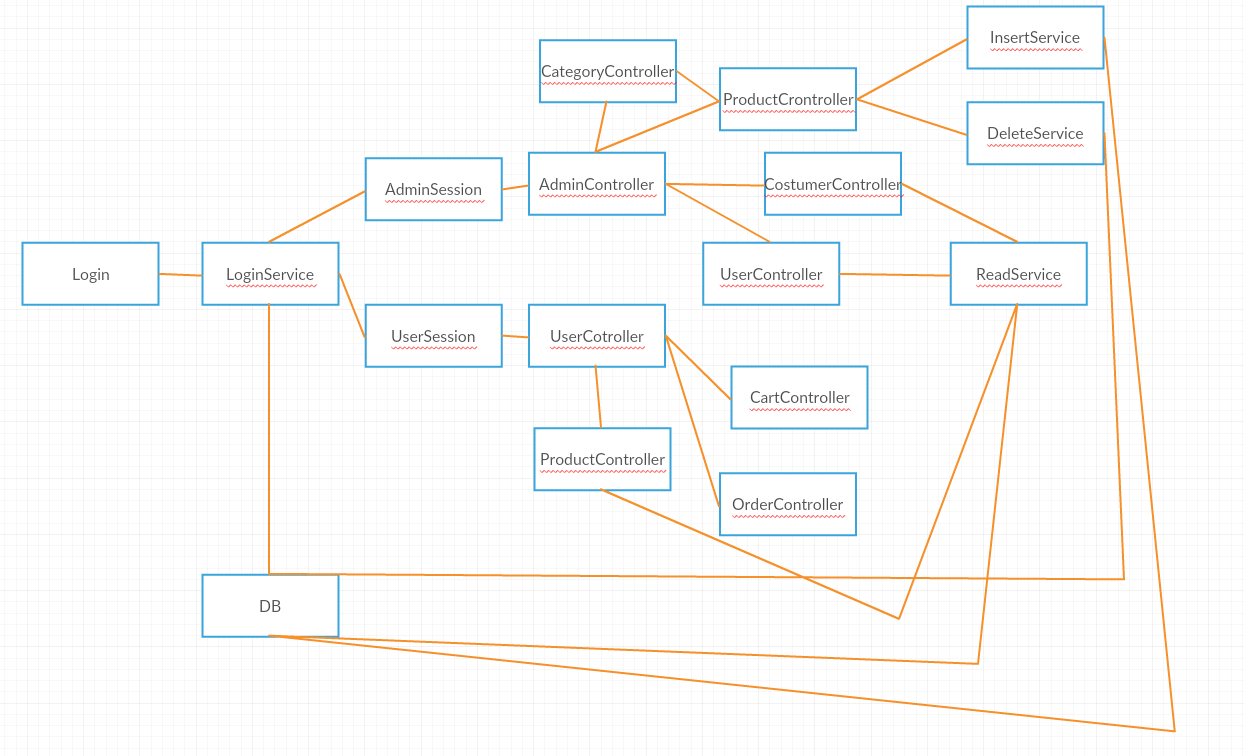
1. *Interfata cosului de cumparaturi*

* *Aici client vor vedea produsele selectate care urmeaza sa fie transmise unei comenzi.*

# Elaboration – Iteration 1.1

# Domain Model

*[Define the domain model and create the conceptual class diagrams]*

**

# Architectural Design

## Conceptual Architecture

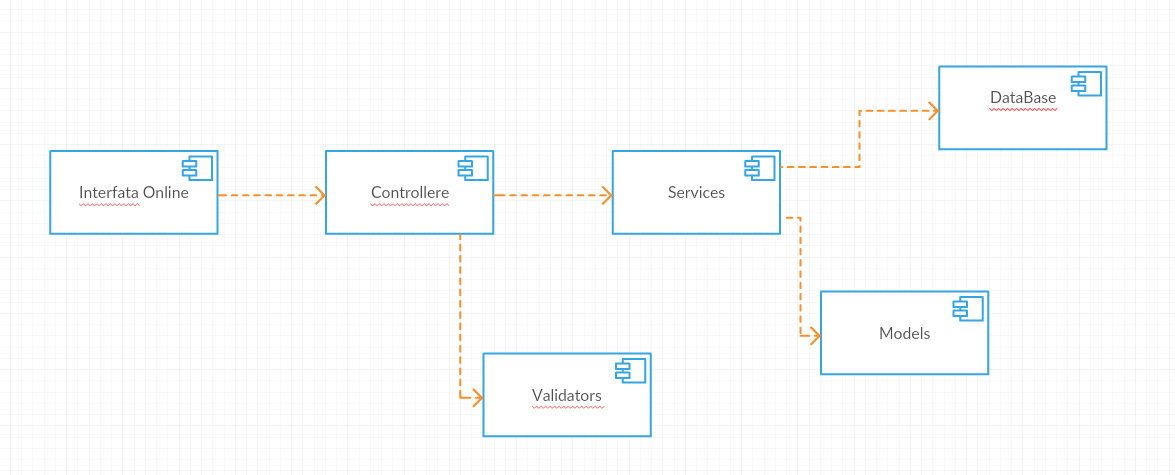
*[Define the system’s conceptual architecture; use an architectural style and pattern - highlight its use and motivate your choice.]*

*Arhitectura pentru acest proiect este una pe layere fiind usor de implementat si mai usor de gestionat , aceasta arhitectura se face prezenta prin impartirea functionalitatilor programului in diferite pachete.*

*In proiect avem mai multe pachete fiecare cu actiunile specific unu.i nivel din arhitectura pe layere: pachetul de model unde sunt definite obiectele prezente in proiect , pachetul de controller unde se realizeaza actiunile fara a fi direct influientata baza de date , deoarece se leaga la nivelul de service unde sunt prezente actiunile cu baza de date precum concetare , adaugarea in baza de date , stergerea , logarea etc. Toate aceste layere sunt apelate din primul nivel al arhitecturi si anume site.ul efectiv adica interfata programului.*

## Package Design

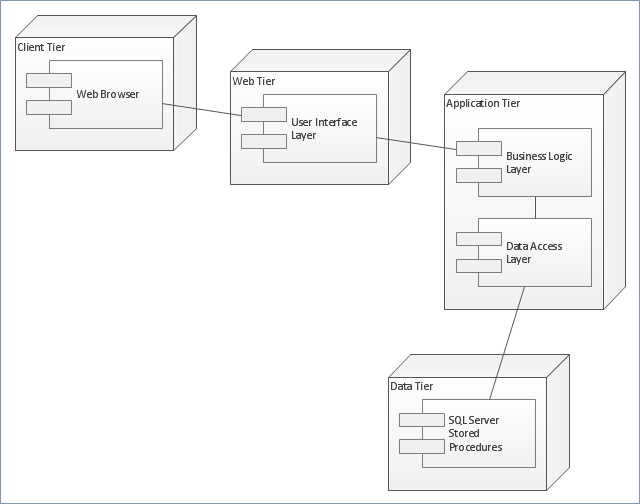
*[Create a package diagram]*

**

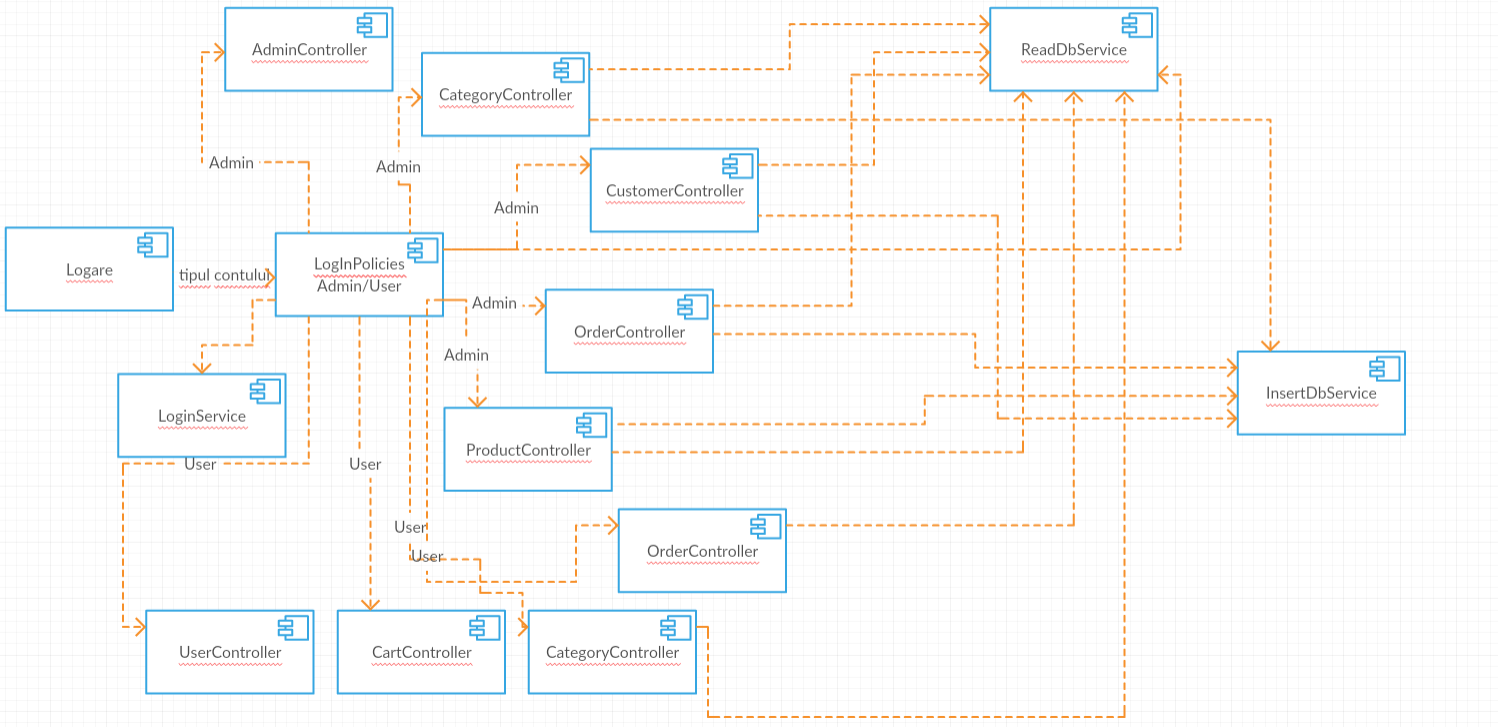
## Component and Deployment Diagrams

*[Create the component and deployment diagrams.]*

Diagrama de deployment



**Component Diagram**



# Elaboration – Iteration 1.2

# Design Model

**Design Patterns Description:**

**In acest proiect se folosesc design pattern-urile:**

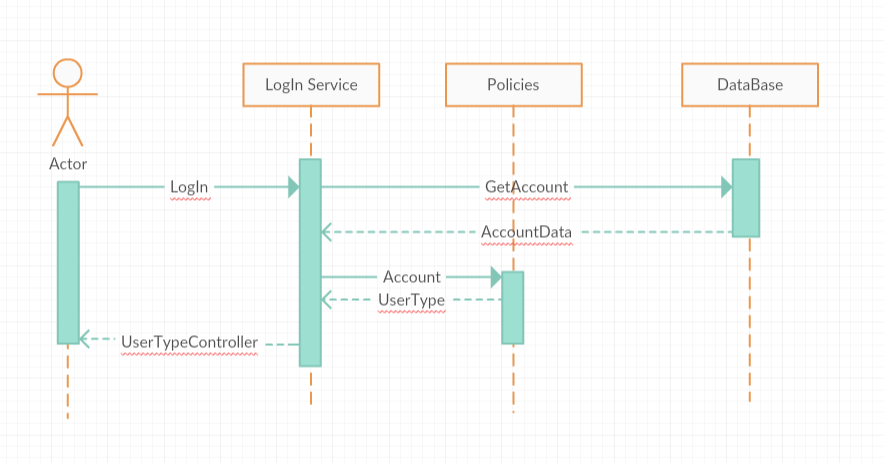
**-Singleton pentru conexiunea la baza de date , aceasta permite o scadere considerabila a cantitatii de cod necesara conexiuni baze de date deoarece se realizeaza un obiect care face direct conexiunea . Motivul pentru care am utilizat acest design patter este ca node.jb se bazeaza in mare pe acesta.**

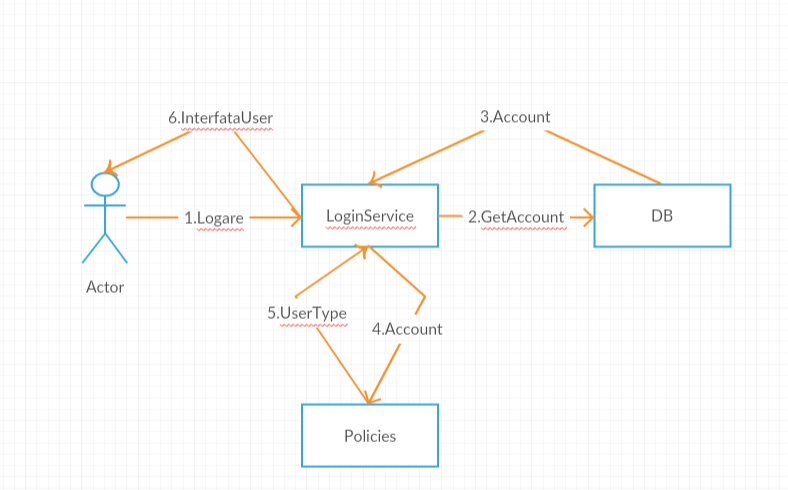
**-MVC – acest design pattern architectural permite prelucrarea datelor in controller fara a fi vizibila logica de procesare in partea de interfata aceasta preluand doar datele gata prelucrate de controller. Motivul alegeri acestui design patter este o mai usoara implementare si intelegere a transmiterii de date pt interfata , aceasta apeland doar metodele fara a fi necesara logica suplimentara.**

## Dynamic Behavior

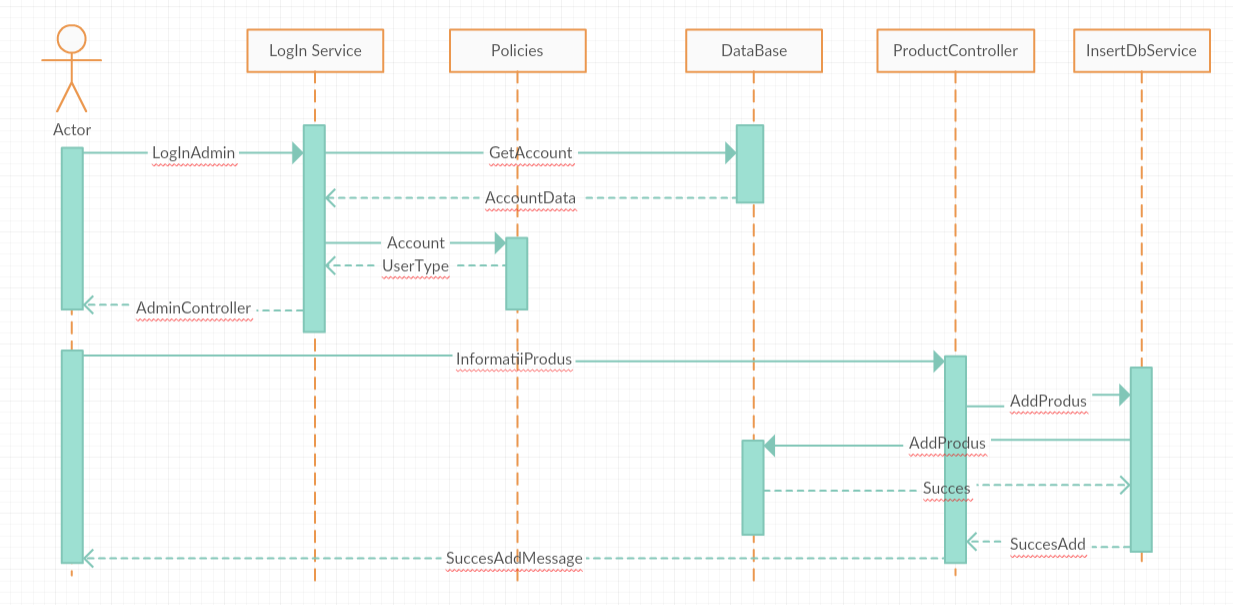
*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

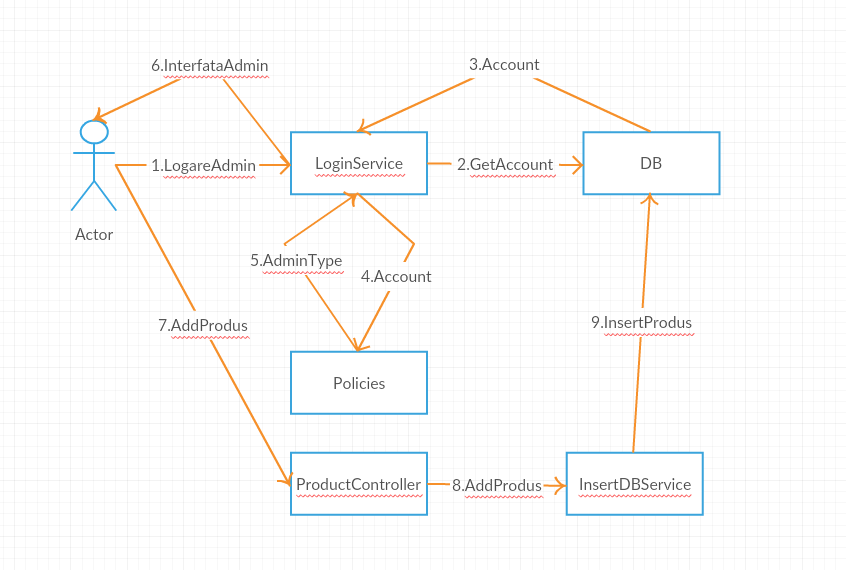
*LogInScenario:*

**

**

*Adaugare Produs:*

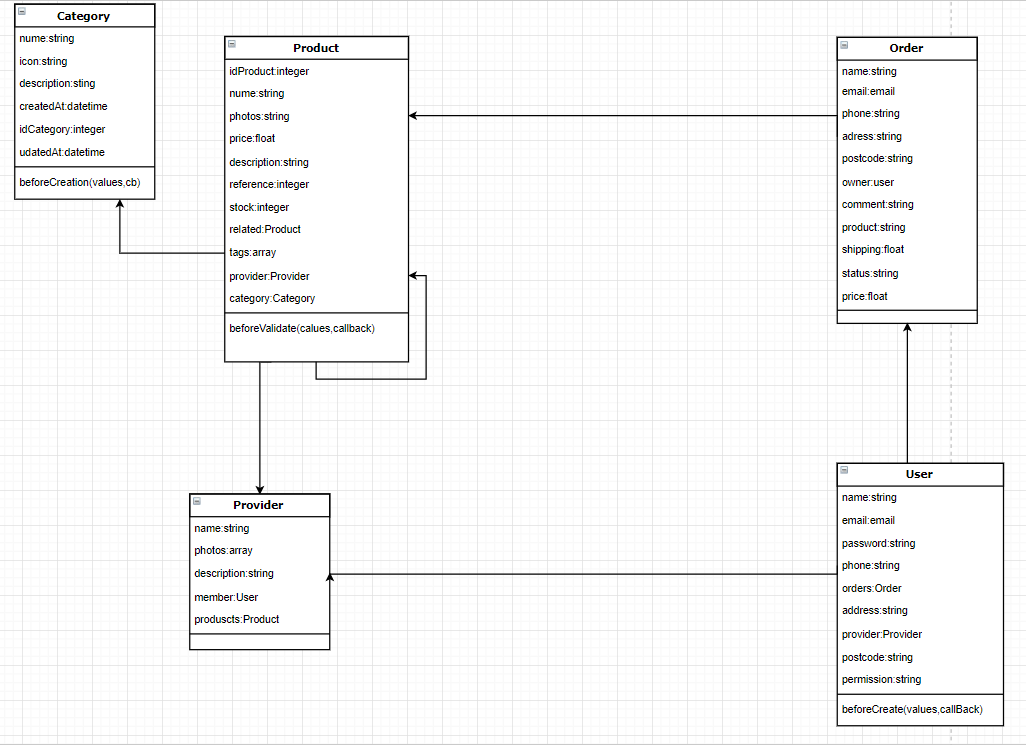
**

**

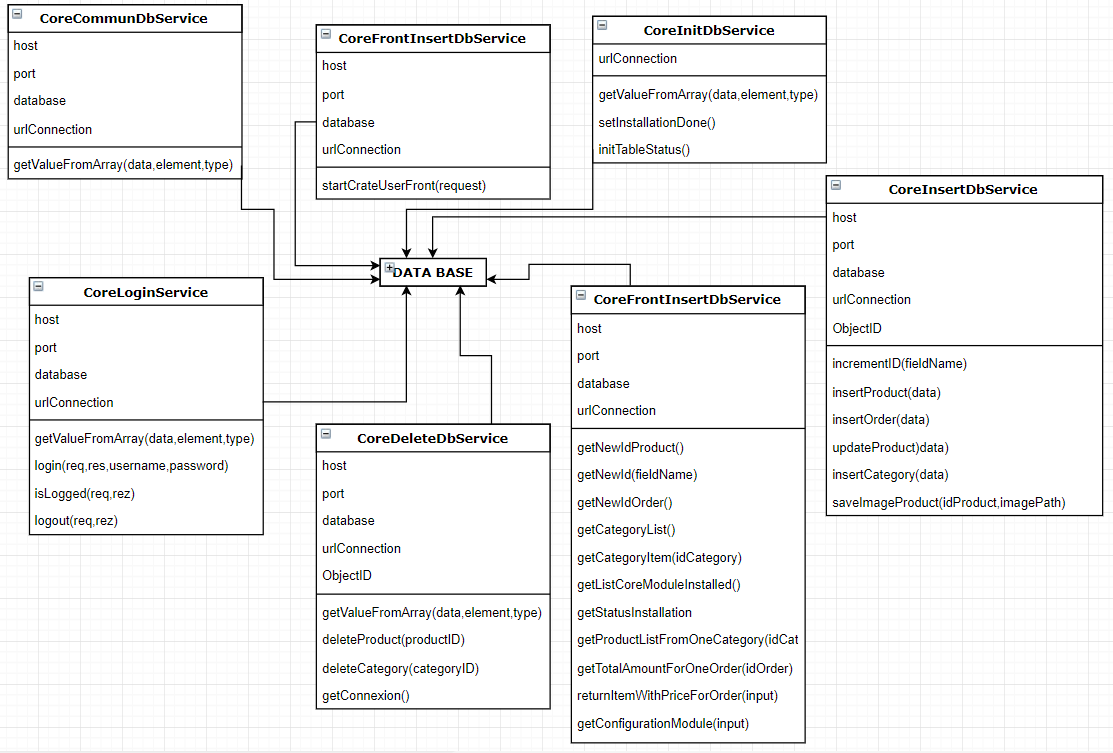
## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

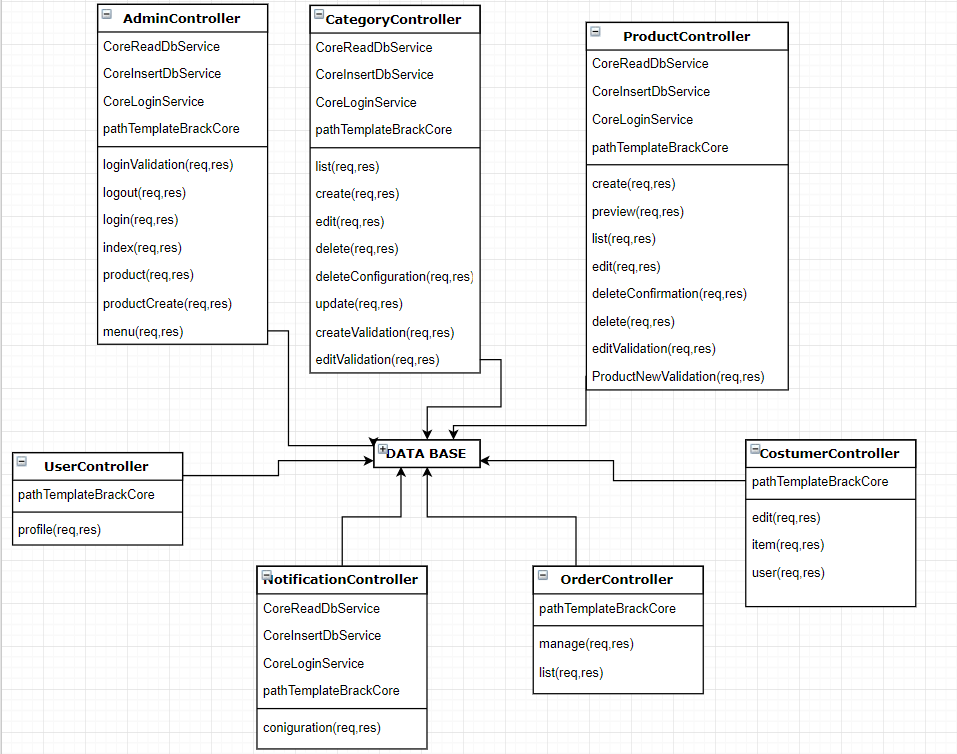
*Model*



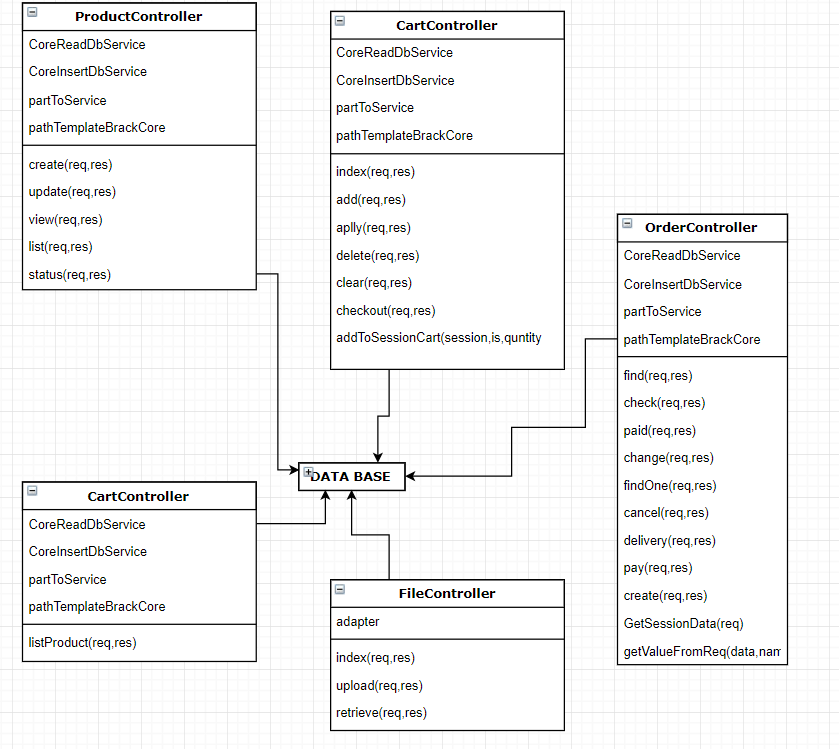
*Servici Baza De Date*



*Controller Admin*

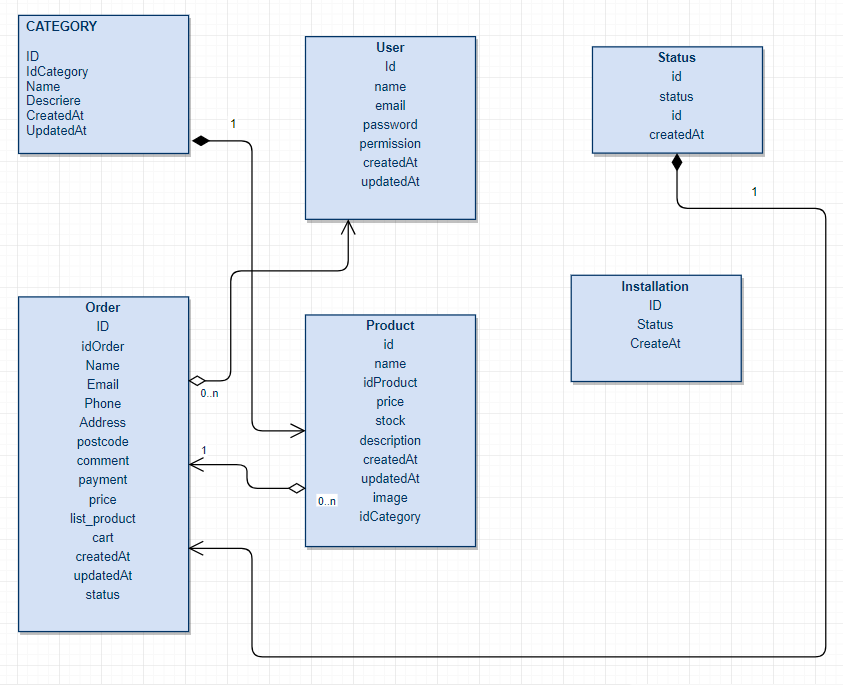
****

*Controller User*

****

# Data Model

*[Create the data model for the system.]*



# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

In perioada de implementeare am observant aparitia mai multor Bug-uri , in urma unor cazuri de testare a site-ului.

1) In urma logari ca si user , nu se putea accesa fereasta de informatii a userului .

Linkul catre controllerul de profil a trebui updatat pentru a se poate face conexiunea cu succes:

2) In urma achizitionarii unu.i produs acesta nu era sesizat de baza de date Stoc-ul

ramanand neschimbat.

Aceasta problema era cauzata de lipsa updatarii stockului produsului respectiv.

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

**In acest proiect se folosesc design pattern-urile:**

**-Singleton pentru conexiunea la baza de date , aceasta permite o scadere considerabila a cantitatii de cod necesara conexiuni baze de date deoarece se realizeaza un obiect care face direct conexiunea . Motivul pentru care am utilizat acest design patter este ca node.jb se bazeaza in mare pe acesta.**

**-MVC – acest design pattern architectural permite prelucrarea datelor in controller fara a fi vizibila logica de procesare in partea de interfata aceasta preluand doar datele gata prelucrate de controller. Motivul alegeri acestui design patter este o mai usoara implementare si intelegere a transmiterii de date pt interfata , aceasta apeland doar metodele fara a fi necesara logica suplimentara.**

**-Middlewares/pipelines- este un desing pattern behavioral utilizand waterfall-u care executa in ordine un set de functi transmitand mai departe valoarea returnata de functia precedenta la functa urmatoare .**

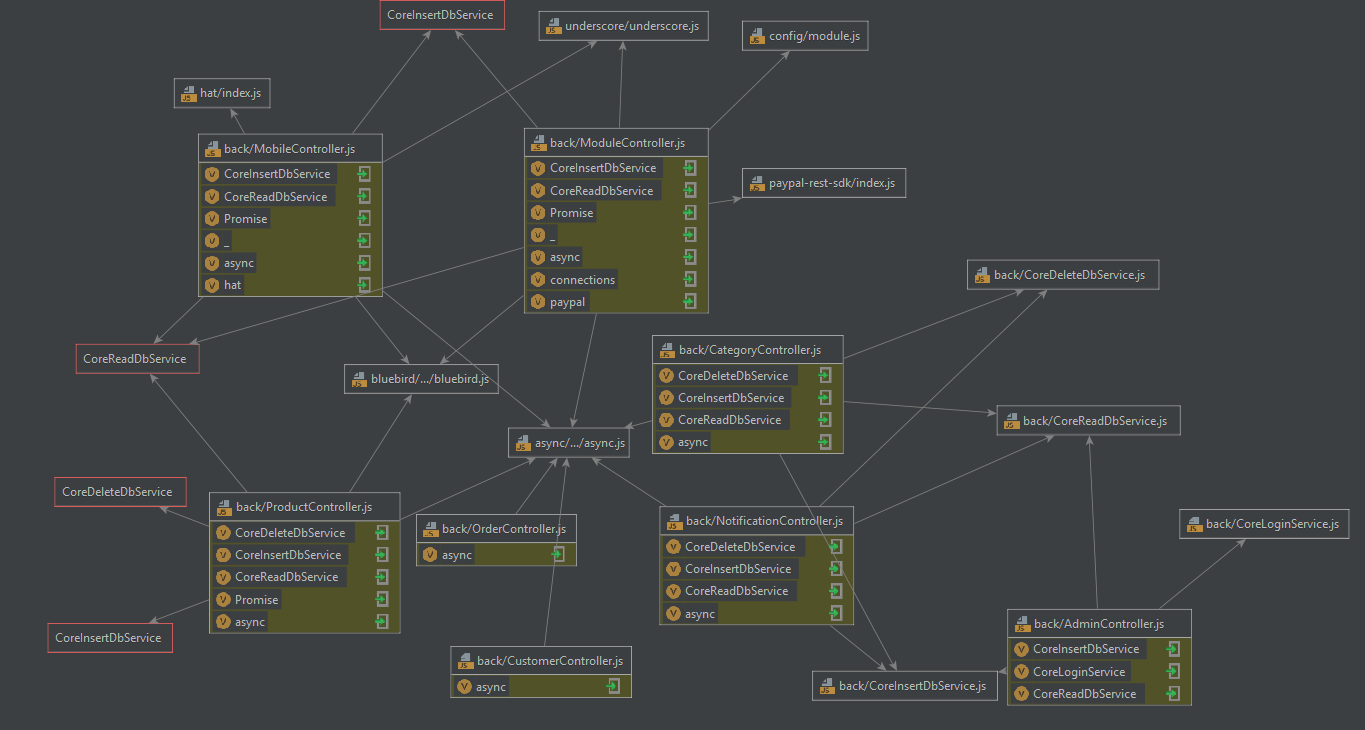
**Toti utilizatori logati pot vizualiza produsele adaugate recent de admin printr.un simplu refresh la pagina , insa si adminul primeste notificari daca un client goleste stocul printr.o comanda a sa , arhitectura si conctiunea la baza de date fiind prin aceleasi service si la admin si la useri permite preluarea datelor si cand unul dintre acestia modifica date in DB.**

**Framework:**

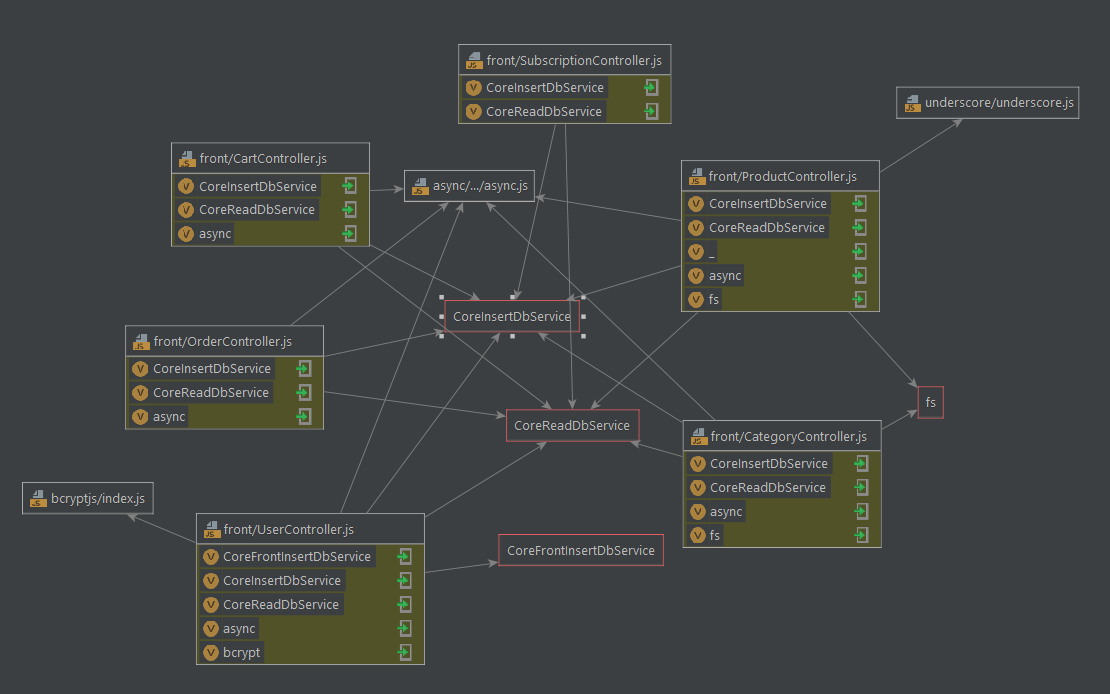
- **Sails este cel mai popular cadru MVC pentru Node.js, conceput pentru a emula modelul familiar MVC al cadrelor ca Ruby in Rails, dar cu suport pentru cerințele aplicațiilor moderne: API bazate pe date cu o arhitectură scalabilă și orientată spre servicii.**

# Design Model Refinement

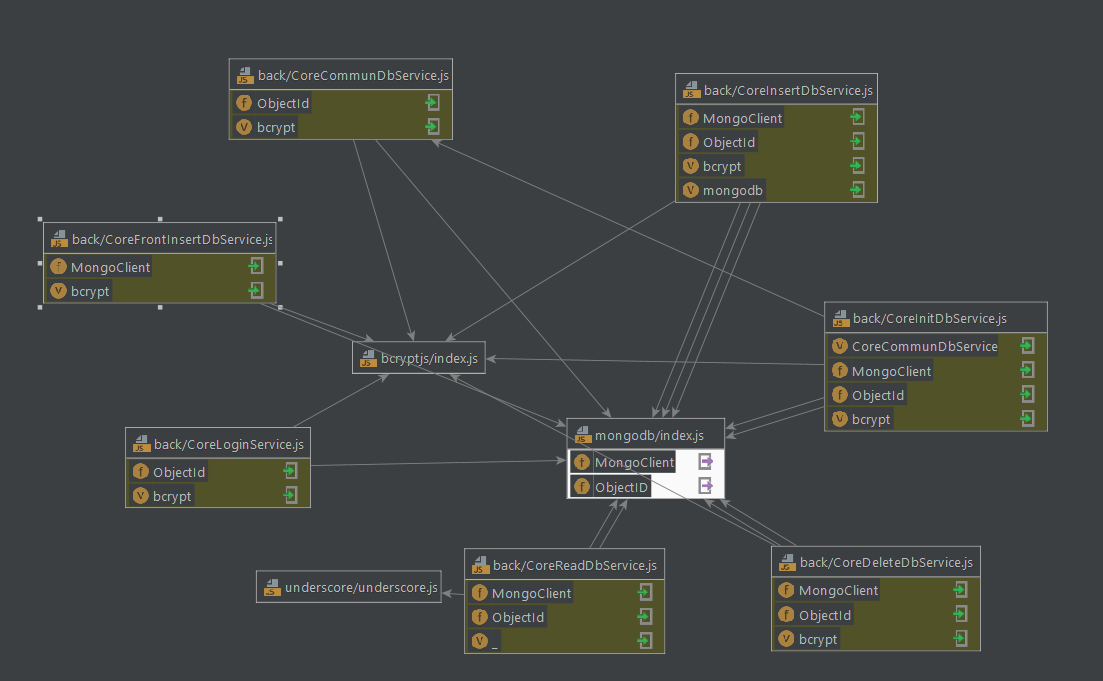
*Model Controller backend:*

**

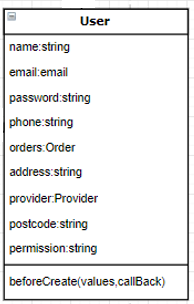
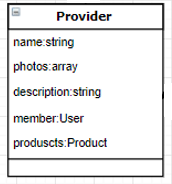
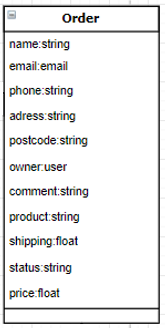
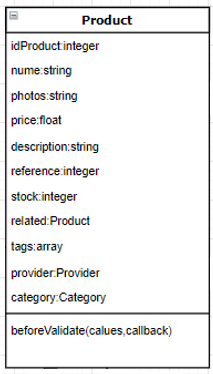
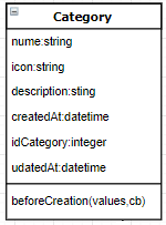
*Model Controller FrontEnd:*

**

*Model Servici:*

**

*Modele:*

**

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

*Testarea proiectului a fost realizata pe parcurs si anuma verificarea pas cu pas a actiunilor unui utilizitor si verificarea datelor sau a corectitudini modificarii acestora .*

*De exmeplu modificarea datelor din cont si verificarea daca acestea au fost schimbate in baza de date sau adaugarea unui produs si verificarea in baza de date si a interfetei utilizator , pana si realizarea unei comenzi sa fie vizibila de catre admin ,dar si modificarea stocului actual .*

*In urma acestor testes au gasit diferite bug.uri care au fost rezolvate si mai apoi rafinate .*

# Future improvements

*[Present future improvements for the system]*

*Dezvoltari ulterioare pentru adaugare modurilor de plata, trimiterea de mail cu comanda si crearea unui document.*

# Bibliography

<https://en.wikipedia.org/wiki/Node.js>

<https://en.wikipedia.org/wiki/MongoDB>

<https://nodejs.org/dist/latest-v8.x/docs/api>

<https://en.wikipedia.org/wiki/Sails.js>

<https://www.youtube.com/watch?v=Do_Hsb_Hs3c>

<https://www.youtube.com/watch?v=h5lJYSE7xD4>

https://www.youtube.com/watch?v=TlB\_eWDSMt4