

BookShop

 Proiect la Programare Software

Name: Alina Aurică

Group: 30236

​​

​​

Table of Contents

[**Deliverable 1 2**](#_Toc130977857)

[**Project Specification 2**](#_Toc130977858)

[**Functional Requirements 2**](#_Toc130977859)

[**Use Case Model 1 2**](#_Toc130977860)

[Use Cases Indentification 2](#_Toc130977861)

[UML Use Case Diagram 2](#_Toc130977862)

[**Supplementary Specification 2**](#_Toc130977863)

[Non-Functional Requirements 2](#_Toc130977864)

[Design Constraints 2](#_Toc130977865)

[**Glossary 2**](#_Toc130977866)

[**Deliverable 2 2**](#_Toc130977867)

[**Domain Model 2**](#_Toc130977868)

[**Architectural Design 4**](#_Toc130977869)

[Conceptual Architecture 4](#_Toc130977870)

[Package Design 4](#_Toc130977871)

[Component and Deployment Diagram 4](#_Toc130977872)

[**Deliverable 3 4**](#_Toc130977873)

[**Design Model 4**](#_Toc130977874)

[Dynamic Behavior 4](#_Toc130977875)

[Class Diagram 4](#_Toc130977876)

[Data Model 4](#_Toc130977877)

[**System Testing 4**](#_Toc130977878)

[**Future Improvements 4**](#_Toc130977879)

[**Conclusion 4**](#_Toc130977880)

[**Bibliography 4**](#_Toc130977881)

# Deliverable 1

## Project Specification

Am ales ca temă de proiect să creez un site web de vânzare-cumpărare orientat în jurul achiziției de cărți. Partea de back-end (de funcționalități) va fi realizată în Java, utilizând framework-ul Spring. Partea de front-end (user-interface) va fi realizată în Angular.

Proiectul are ca scopuri principale: familiarizarea cu framework-ul Spring, crearea unei legături cu o bază de date de tip SQL (în cazul nostru, MySQL), integrarea front-end-ului cu back-end-ul și realizarea de funcționalități specifice temei proiectului.

Funcționalitățile proiectului se impart în: operații pe CRUD și funcționalități care nu depind de CRUD. În cele ce urmează, se vor analiza mult mai în detaliu aceste funcționalități, precum și specificațiile non-funcționale și constrângerile de design pe care le-a atins proiectul.

## Functional Requirements

Autentificarea:

Partea de autentificare se împarte în 2 mari categorii: register/modifyData și logIn/logout. Prima dintre ele îi permite unui user nou să-și creeze un cont prin introducerea datelor sale cu caracter personal, iar dacă aceste date sunt într-un format valid, se poate crea un user cu respectivele credențiale (e-mail și parolă). Partea de modifyData îi permite utilizatorului să își schimbe parola sau email-ul. Cea de-a doua parte este cea de logIn/logOut. În cadrul acesteia, user-ul introduce credențialele, iar dacă acestea sunt corecte, el este redirecționat pe o pagină specifică (în funcție de rolul user-ului: client sau admin) unde poate realize anumite operații în funcție de rolul. Când user-ul își va încheia vizita pe site, acesta se va deloga. Acest login/logout se va realize prin marcarea într-o coloană din tabel cu 1 când user-ul e on și cu 0 când user-ul e off.

Funcționalități pe Client:

Pe partea de operații CRUD, clientul este capabil să interogheze baza de date, căutând după numele, autorul și/sau genul cărții. Poate să selecteze o carte și să o introducă în coșul de cumpărături sau, în cazul în care se răzgândește, poate să șteargă cartea respectivă din coș. O altă funcționalitate ar fi să poată lăsa recenzii la cărți. Pe baza acestora, adminul va putea crea un news letter cu recenzii pentru cartea/cărțile care vor fi reduse în săptămâna respectivă.

La finalul sesiunii de cumpărături, clientul are posibilitatea să finalizeze comanda sau să renunțe la aceasta. Voi dezvolta prima situație, deoarece prin intermediul acesteia vom ajunge la funcționalitatea de plată, unde se introduc datele de plată și se generează o factură, care va fi afisata în interfață, dar va fi trimisă și pe email.

De asemenea, clientul are posibilitatea să vorbească cu un Chat Bot de asistență care îi va furniza informații standard despre politica de retur, costul de livrare etc. Bot-ul va fi conectat cu clientul prin WebSocket-uri.

Funcționalități pe Admin:

La fel ca și clientul, adminul poate realize operații pe CRUD, care sunt, în mare parte, operațiile care pot modifica baza de date cu cărți. Aceste operații sunt cele de insert a unei noi cărți în baza de date, de update al prețului sau stocului unui produs, dar și de a șterge o carte din baza de date.

Adminul este cel raspunzător pentru trimiterea de email-uri, care se impart în 2 categorii: trimiterea de facturi și crearea și trimiterea de news letter-uri. De asemenea, la finalul zilei, acesta realizează rapoarte (cea mai bine vândută carte, cărțile vândute în acea zi etc.), pe care le exportă sub formă de fișiere XML.

Ultima funcționalitate a site-ului este aceea că notifică clientul de cărțile noi care au mai apărut. Pentru aceasta intenționez să folosesc tot WebSocket la fel ca la Chat Bot.

## Use Case Model 1

### Use Cases Indentification

*Use-Case1:  Register*

*Level:  User-Goal*

*Primary Actor: User*

*Main success scenario: User face register. Succes*

*Extensions: Format ilegal de date(email fără @, parola cu număr mai mic de caractere)*

*Use-Case2:  LogIn*

*Level:  Subfunction*

*Primary Actor: User*

*Main success scenario: Userul se loghează. Succes*

*Extensions: Format ilegal de date(email și/sau parolă invalide)*

*Use-Case3:  LogOut*

*Level:  Subfunction*

*Primary Actor: User*

*Main success scenario: Userul se deloghează. Succes.*

*Extensions: -*

*Use-Case4:  ModifyData*

*Level:  Subfunction*

*Primary Actor: User*

*Main success scenario: Userul își schimbă parola/email-ul. Succes.*

*Extensions: Modificările nu se salvează*

*Use-Case5: BuyABook*

*Level:  Client-Goal*

*Primary Actor: Client*

*Main success scenario: Clientul caută cartea pe care dorește să o cumpere. O introduce în coș. Alege finalizare comandă și plătește.*

*Extensions: Cartea nu se găsește pe stoc. Clientul nu poate introduce cartea în coș. Clientul nu finalizează comanda/ nu o poate finalize. Clientul nu poate efectua plata.*

*Use-Case6: ChatWithChatBot*

*Level:  Client-Goal*

*Primary Actor: Client*

*Main success scenario: Clientul solicit asistență de la Chat Bot. Chat-ul răspunde correct.*

*Extensions: Chat-ul nu răspunde sau răspunde greșit.*

*Use-Case7: WriteAReview*

*Level:  Subfunction*

*Primary Actor: Client*

*Main success scenario: Clientul scrie un review.*

*Extensions: Review-ul nu se salvează.*

*Use-Case8: CreateAGiftCard*

*Level:  Subfunction*

*Primary Actor: Client*

*Main success scenario: Clientul comandă un card cadou.*

*Extensions: Clientului nu i se permite să comande cardul. Introduce o sumă invalidă.*

*Use-Case9: Delete/Insert/ModifyDBBook*

*Level:  Admin-Goal*

*Primary Actor: Admin*

*Main success scenario: Adminul inserează/șterge/modifică o carte.*

*Extensions: Modificările nu se salvează. Ștergem/modificăm ce nu trebuie.*

*Use-Case10: GenerateABill*

*Level:  Subfunction*

*Primary Actor: Admin*

*Main success scenario: Adminul generează o factură.*

*Extensions: Factura are datele incorecte/incomplete.*

*Use-Case11: GenerateAReport*

*Level:  Subfunction*

*Primary Actor: Admin*

*Main success scenario: Adminul generează un raport.*

*Extensions: Raportul are datele incorecte/incomplete.*

*Use-Case12: GenerateANewsLetter*

*Level:  Subfunction*

*Primary Actor: Admin*

*Main success scenario: Adminul generează un news letter.*

*Extensions: News letter are datele incorecte/incomplete.*

*Use-Case12: NotifyClientNewBook*

*Level:  Subfunction*

*Primary Actor: Admin*

*Main success scenario: Adminul notifică clientul de apariția unei noi cărți.*

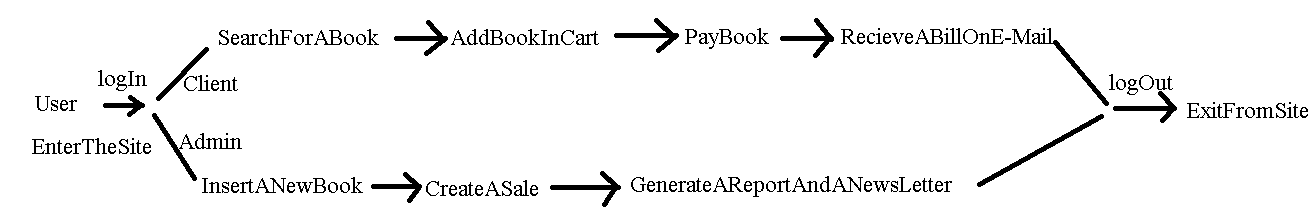
*Extensions: Notificarea nu funcționează (nu apare).*

*Use-Case13:  Book Shop*

*Level:  High-level*

*Primary Actor: User, Admin, Client*

*Main success scenario:*

**

*Extensions: Clientul poate renunta la cărțile din coșul de cumpărături și să iasă de pe site fără a achiziționa vreo carte. Clientul vrea mai multe informații despre politica de retur, așa că vorbește cu Chat Bot-ul, apoi părăsește site-ul. Adminul poate să nu creeze reduceri și, ca atare, nu va mai crea News Letter-ul. De asemenea, poate doar să șteargă o carte și după să iasă de pe site. ETC.*

### UML Use Case Diagram



## Supplementary Specification

### Non-Functional Requirements

Portabiblitate și compatibilitate:

Portabilitatea determină modul în care un sistem sau elementul său poate fi lansat într-un mediu sau altul. De obicei, include specificații hardware, software sau alte platforme de utilizare. Mai simplu, stabilește cât de bine sunt executate acțiunile efectuate de o platformă pe alta. De asemenea, stabilește cât de bine pot fi accesate elementele sistemului și pot interacționa din două medii diferite. Compatibilitatea, ca aspect suplimentar al portabilității, definește modul în care un sistem poate coexista cu un alt sistem în același mediu.

Din această perspecivă, site-ul web pe care îl dezvolt este dezvoltat să ruleze pe sisteme de operare de tip Windows (de la versiunea 10 în sus) si este compatibil cu antivirusul integrat al sistemului de operare. Ar trebui să fie compatibil cu sistemele de operare de tip Android, adică am putea accesa site-ul web și de pe un SmartPhone cu acest sistem de operare.

Securitate:

Securitatea este o cerință nefuncțională care asigură că toate datele din interiorul sistemului sau o parte a acestuia vor fi protejate împotriva atacurilor malware sau a accesului neautorizat.

Prind din două perspective, proiectul va implementa criptarea parolei pentru ca spargerea bazei de date a site-ului să îndreuneze și spargerea conturilor user-ilor stocați. De asemenea, acesta va implementa o securitate care să limiteze accesul user-ilor la paginile care nu sunt destinate rolului lor. Spre exemplu, clientul nu poate avea accesul la pagina deschisă adminului și vice-versa. De asemenea, nicio persoana din exterior nu ar putea accesa paginile secundare, fără a se loga pe pagina principală (prin schimbarea url-ului manual).

Utilizabilitatea:

Utilizabilitatea abordează o întrebare simplă: Cât de greu este să folosești produsul? Datorită simplității user-interface-ului, clientul este cababil să utilizeze foarte ușor aplicația. Simbolurile și modul de abordare al creării butoanelor și textField-urilor este sugestivă și poate fi utilizată cât mai ușor.

Mentenabilitate:

Mentenabilitatea definește timpul necesar pentru ca o soluție sau componenta sa să fie reparată, modificată pentru a crește performanța sau alte calități sau adaptată la un mediu în schimbare.

Datorită unei împărțiri eficiente a claselor în pachete după funcționalitate (model, repository, service, controller), debug-ul poate fi făcut liniar de la high-level la low-level și pot fi identificate și reparate problemele. De asemenea, se pot realiza îmbunătățiri pentru creșterea sustenabilității foarte ușor, modificându-se numai anumite componente, care sunt independente de restul proiectului.

### Design Constraints

Site-ul este construit utilizând ca limbaj de programare Java, versiunea 17, împreună cu framework-ul Spring. La acestea s-au adăugat: librăria Lombok din care am utilizat adnotarea @Data pentru generarea getter-elor și setter-elor și Mockito pentru generarea de Unit tests.

Proiectul este construit pe arhitectura MVC, unde fiecare nivel este împărțit în mai multe layere. Pe partea de Model avem pachetele: model, repository, service, pe partea de Controller este pachetul controller, iar pachetul resources este folosit pentru view-urile create cu Angular.

Accesul și trasportul datele este liniar, făcându-se de la nivelul extragerii din baza de date, până la nivelul de afișare în front-end. Legătura între tabelele din baza de date a fost realizată prin intermediul relațiilor ORM( One-To-Many, Many-To-One).

## Glossary

**MVC** (Model-View-Controller): Este un model în proiectarea software-ului utilizat în mod obișnuit pentru a implementa interfețe cu utilizatorul, date și logica de control.

**CRUD**: În programarea computerelor, crearea, citirea, actualizarea și ștergerea sunt cele patru operațiuni de bază ale stocării persistente.

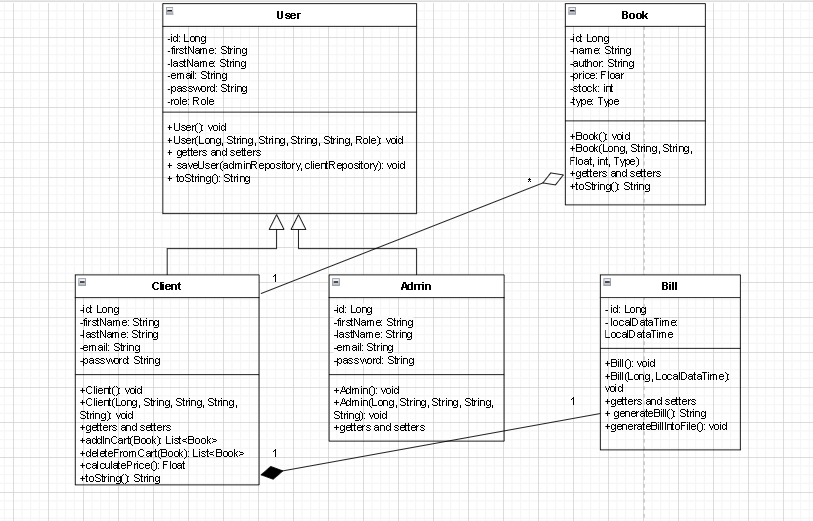
**ORM**: Hibernate ORM permite dezvoltatorilor să scrie mai ușor aplicații ale căror date supraviețuiesc procesului de aplicare. Ca cadru ORM (Object/Relational Mapping), Hibernate se preocupă de persistența datelor, deoarece se aplică bazelor de date relaționale (prin JDBC).

**SQL** (Structured Query Language): este un limbaj de programare standardizat care este utilizat pentru a gestiona bazele de date relaționale și pentru a efectua diverse operații asupra datelor din acestea.

# Deliverable 2

## Domain Model

Un domain model este o reprezentare vizuală a entităților cheie, relațiilor și atributelor care există într-un anumit domeniu sau context de afaceri. Oferă o imagine de ansamblu conceptuală a domeniului și servește drept bază pentru proiectarea sistemului software.



## Architectural Design

### Conceptual Architecture

Architectural pattern-ul de bază al proiectului este Layered pattern, deoarece programul poate fi descompus în grupuri de subtask-uri., unde fiecare grup are un nivel particular de abstractizare. Fiecare astfel de nivel oferă servicii pentru următorul nivel. Astfel, nivelurile întâlnite în proiectul meu sunt:

1. Presentation layer: conține toată partea de frontEnd care la rândul ei e împărțită în layer-e (Model, Service, Component)
2. Controller layer: nivelul API-urilor
3. Service layer: sub-ramură de la Application layer
4. Repository layer: sub-ramură de la Data access layer
5. Model layer și DTO layer: sub-ramuri de la Business logic layer

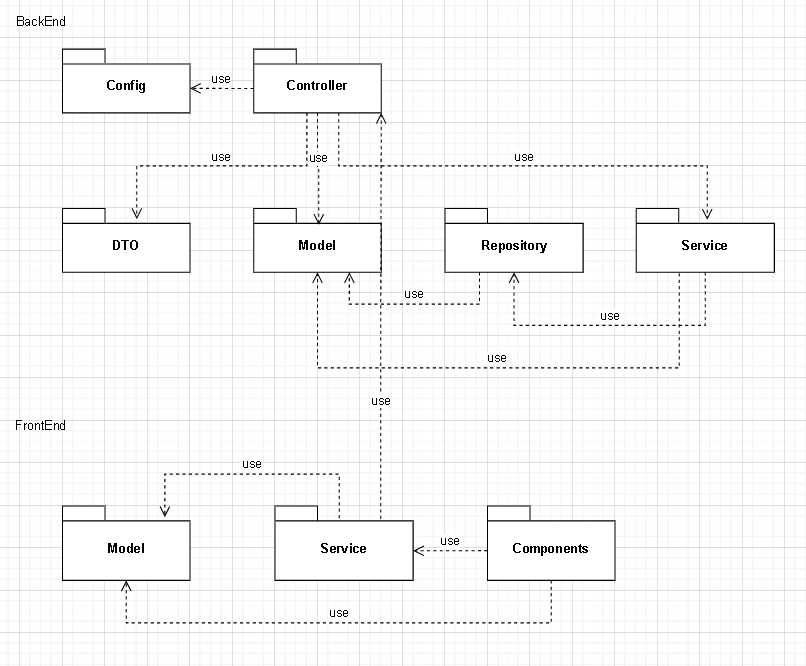
De asemenea, este utilizat modelul MVC (Model-View-Controller), care organizează tot proiectul în 3 părți:

1. Model: conține funcționalitatea de bază și datele
2. View: afișează informațiile utilizatorului (pot fi definite mai multe vizualizări)
3. Controler: gestionează intrarea de la utilizator

Acest lucru se face pentru a separa reprezentările interne ale informațiilor de modurile în care informațiile sunt prezentate și acceptate de la utilizator. Decuplează componentele și permite reutilizarea eficientă a codului.

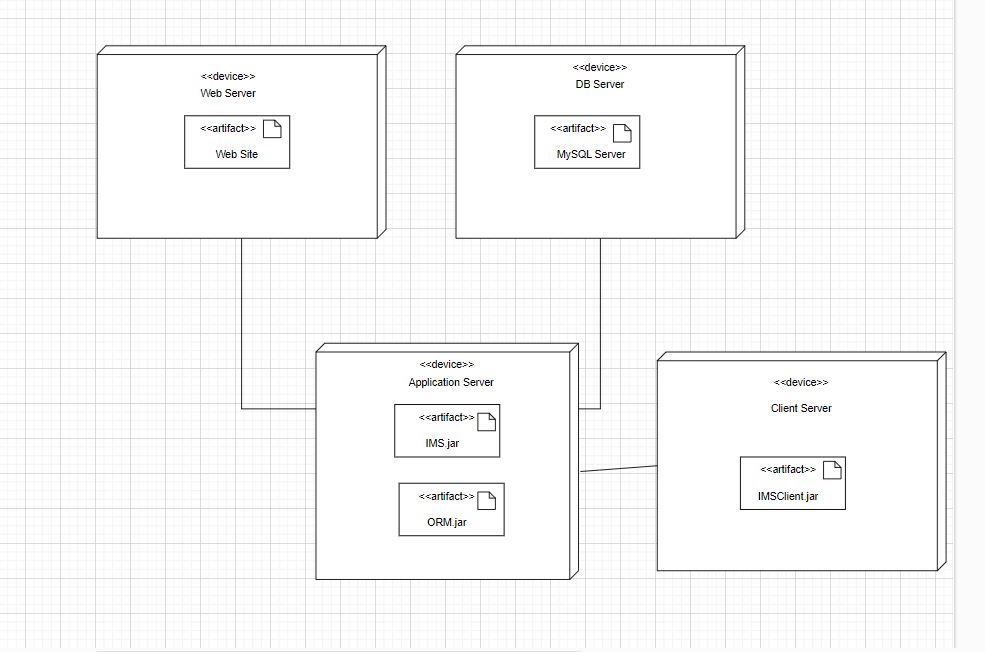
Acest architectural pattern este cel mai des recomandat perntru crearea de pagini World Wide Web.

### Package Design

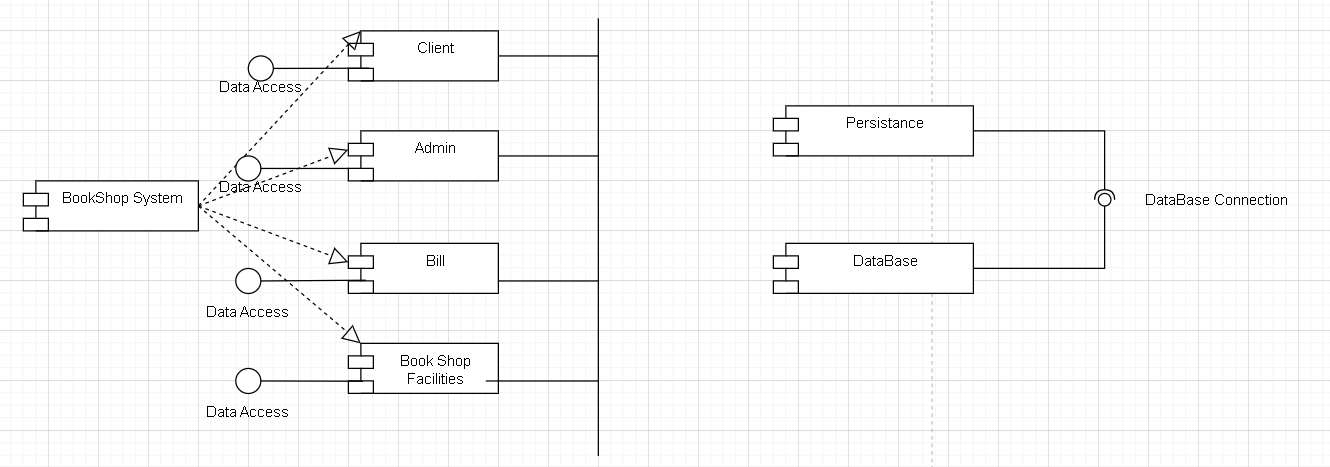
**

### Component and Deployment Diagram

Deployment Diagram:



Component Diagram:

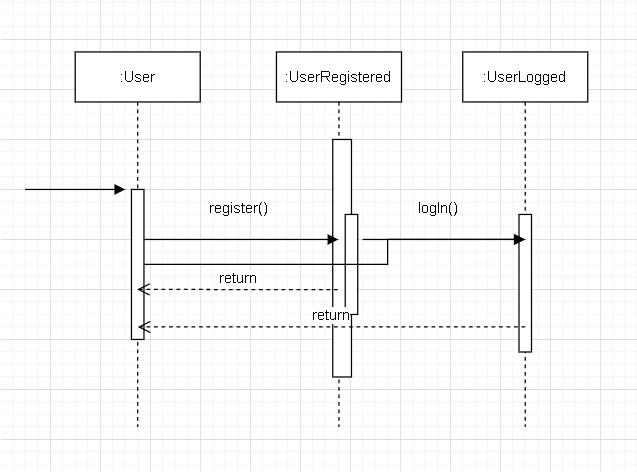


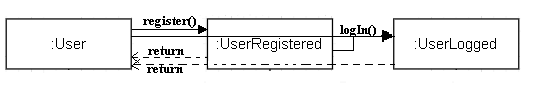
# Deliverable 3

## Design Model

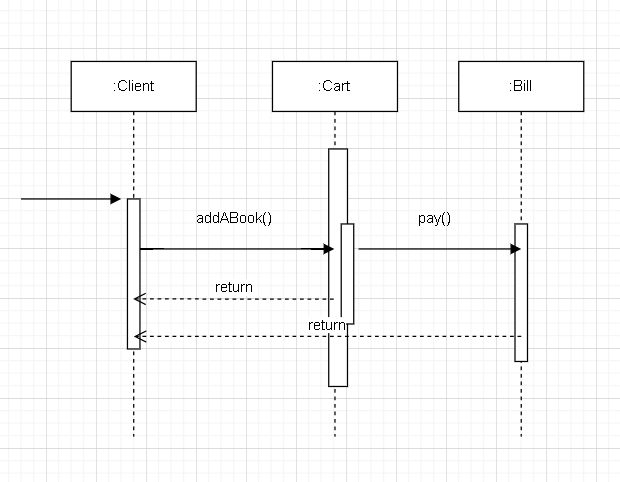
### Dynamic Behavior

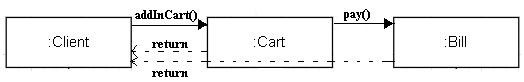
LogIn Scenario:





Buy A Book:





### Class Diagram



GoF patterns:

1. Composite Design Pattern:

Descrie un grup de obiecte care sunt tratate în același mod ca o singură instanță a aceluiași tip de obiect. Intenția acestuia este de a „compune” obiecte în structuri arborescente pentru a reprezenta ierarhii parțial-întreg.

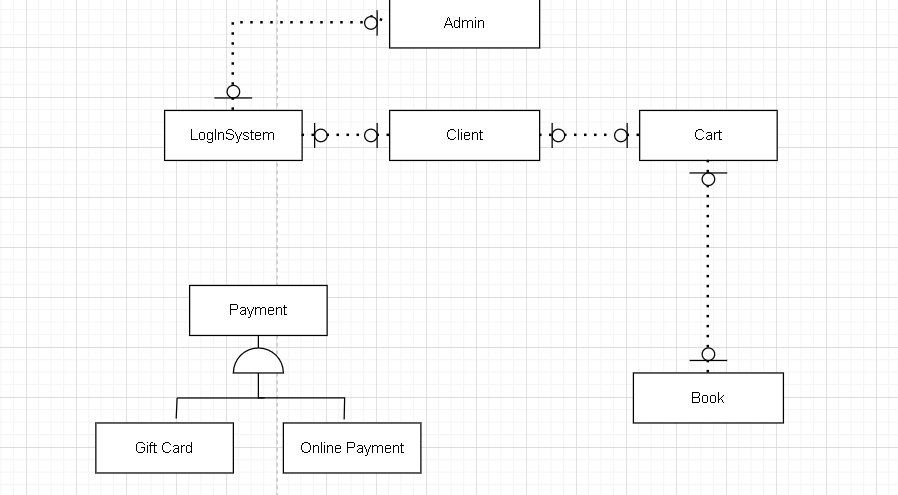
Am ales acest design pattern deoarece dorim să lucrăm într-o ierarhie de moștenire, unde noi să putem avea elemente compuse din alte elemente singure sau compuse.

1. Builder Pattern:

Builder Pattern este un model de design conceput pentru a oferi o soluție flexibilă la diferite probleme de creare a obiectelor în programarea orientată pe obiecte. Intenția este de a separa construcția unui obiect complex de reprezentarea acestuia.

Am utilizat acest design pattern prin crearea unur interfețe care separă declarea metodelor de implementarea acestora. Spre exemplu, acesta este principiul de creare al layer-ului service.

### Data Model



## System Testing

Am utilizat ca sistem de testare JUnit Tests, alături de metodele și adnotarile oferite de Spring Boot și Java. Mare parte din ceea ce s-a validat, s-a realizat prin Mock-uri. Aceastea simulează interacțiunea cu baza de date, adică instrucțiunile pe care noi le realizăm (operații CRUD), nu modifică sub nicio formă baza de date.

Test case-urile s-au bazat pe 2 situații: metoda pe care noi doream să o testăm returnează un rezultat correct sau returnează un rezultat greșit. Spre exemplu: în situația unui update, dacă elementul pe care dorim să îl modificăm există în baza de date, atunci acesta este abdatat și verifică valoarea pe care știm că ar trebui să o obținem. Dacă elementul nu există în baza de date, atunci metoda returnează null, adică nu putem realize modificarea. Pentru acestea am utilizat assert-uri, precum: assertNotNull, assertEquals etc.

Ca și adnotări am folosit-o pe cea de @Mock pentru instanțe ale repository-urilor, pe cea de @BeforeEach pentru o metodă de init a tuturor variabilelor de care au nevoie testele, iar metodele propriu-zise de testare au fost marcate cu @Test. În metoda de init() s-a utilizat sintaxa when(X).thenReturn(Y) pentru a sugera ca pentru metoda X, raspunsul este Y.

## Future Improvements

În primul rând, momentan prin intermediul site-ului nu poți să plătești propriu-zis, așa că principal îmbunătățire ar fi realizarea și conectarea cu o metodă de plată online.

În al doilea rând, cred că ar mai fi de dezvoltat partea de interfață cu utilizatorul, deoarece momentan este una destul de minimalistă. În aceeași măsură, ar mai fi de rezolvat micile bug-uri care apar pe partea aceasta (ex: prima dată când se deschide pagina de login, apar toate field-urile goale, chiar dacă ele sunt scrise). Cu alte cuvinte, majoritatea îmbunătățirilor ar fi estetice.

În al treilea și ultimul rând, partea de securitate ar mai putea fi îmbunătățită prin adăugarea unui regex pentru parolă și printr-o autentificare prin 2 pași.

## Conclusion

Proiectul realizat în cadrul laboratorului de Proiectare Software, care a constat în realizarea unui Book Shop Online, mi-a dezvoltat lucrul cu framework-ul Spring, specific limbajului de programare Java, am învâțat un limbaj de programare specific front-end-ului, precum și cum să leg între componentele de back-end și de front-end. De asemenea, am învățat să-mi structurez pachetele și să aplic diverse pattern-uri arhitecturale și de design. Am aplicat în cadrul acestui proiect concepte de securitate și de arhitectură Client-Server.

Cu alte cuvinte, am învățat foarte multe lucruri noi în cadrul proiectului și am luat pentru prima dată contact cu ceea ce se cere pe piața muncii.

## Bibliography

1. Pentru Securitate: <https://www.youtube.com/watch?v=KxqlJblhzfI&feature=youtu.be&ab_channel=Amigoscode>
2. Pentru realizarea tuturor tipurilor de diagrame:

<https://www.tutorialspoint.com/uml/uml_interaction_diagram.htm>

1. Pentru Angular:

<https://angular.io/guide/form-validation>

1. Pentru Spring:

<https://github.com/TorneaAdriana/demoLab>

1. Pentru Model Mapper:

<https://www.baeldung.com/java-modelmapper>