

Notion-inspired

app

Name: Chiorean Bogdan-Alin

Group: 30236

Table of Contents

[Deliverable 1 3](#_Toc2055786399)

[Project Specification 3](#_Toc825512728)

[Functional Requirements 3](#_Toc362225657)

[Use Case Model 1 3](#_Toc324720183)

[Use Cases Identification 3](#_Toc159950011)

[UML Use Case Diagrams 4](#_Toc1353450407)

[Supplementary Specification 4](#_Toc501087280)

[Non-functional Requirements 4](#_Toc1600632703)

[Design Constraints 4](#_Toc729746755)

[Glossary 4](#_Toc178374120)

[Deliverable 2 5](#_Toc1123232848)

[Domain Model 5](#_Toc605147785)

[Architectural Design 5](#_Toc1115440048)

[Conceptual Architecture 5](#_Toc2005513277)

[Package Design 5](#_Toc596515525)

[Component and Deployment Diagram 5](#_Toc1771682286)

[Deliverable 3 5](#_Toc174956409)

[Design Model 5](#_Toc2136237097)

[Dynamic Behavior 5](#_Toc853475638)

[Class Diagram 5](#_Toc2045288274)

[Data Model 5](#_Toc1322833382)

[System Testing 5](#_Toc975007147)

[Future Improvements 5](#_Toc902595900)

[Conclusion 5](#_Toc1374315000)

[Bibliography 5](#_Toc2082771450)

# Deliverable 1

## Project Specification

Proiectul ales de mine reprezintă o aplicație web interactivă, inspirată de o aplicație deja existentă, Notion. Notion este considerată drept o aplicație “freemium”, deoarece conține elemente premium gratis. Aceasta include produse organizatorice precum management-ul sarcinilor, urmărirea proiectelor, liste To-Do, contorizarea cărților citite, etc.

Datorită complexității aplicației Notion descrise mai sus, chiar dacă interfața este una foarte simplă și accesibilă tuturor, am decis să imit decât 4 comportamente ale aplicației de bază. Printe acestea se regăsesc paginile de To-Do List, Financial Activity, Read Books Activity, precum și Project Activity.

## Functional Requirements

Așa cum am descris la paragraful trecut, doresc să imit 4 funcționalități ale aplicației Notion , respectiv : paginile specifice unui To-Do list, Financial Activity, Read Books Activity și Project Activity.

Pe baza comportamentelor care sunt implementate se pot specifica cerințele funcționale pentru utilizator, precum și cerințele derivate din acestea, așa cum urmează:

* Utilizatorul trebuie să se logheze drept Admin, în cazul în care acesta este admin (incomplet)
* Utilizatorul trebuie să se logheze drept User, în cazul în care acesta este user (incomplet)
* Utilizatorul poate vizualiza activitățile de tip To-Do, Financial, Read Book sau Project
* Utilizatorul poate adăuga o activitate de tip To-Do, Financial, Read Book sau Project
* Utilizatorul poate șterge o activitate de tip To-Do, Financial, Read Book sau Project
* Utilizatorul poate sorta activitățile în funcție de un câmp specific activității alese

## Use Case Model 1

### Use Cases Identification

*Use-Case1:  Register*

*Level:  User-Goal*

*Primary Actor: User*

*Main success scenario:  Successful register*

*Extensions: Format ilegal de date*

*Use-Case2:  LoginAsAdmin*

*Level:  Subfunction*

*Primary Actor: Admin*

*Main success scenario:  Successful login as admin*

*Extensions: - verificare cheie specifică adminilor*

*Use-Case3:  LogInAsClient*

*Level:  Subfunction*

*Primary Actor: Client*

*Main success scenario:  Successful login as client*

*Extensions: -*

*Use-Case4:  showActivity*

*Level:  Client-goal*

*Primary Actor: Client*

*Main success scenario:  Clientul vizualizează cu succes activitățile din fereastra cerută*

*Extensions: -*

*Use-Case5:  addActivity*

*Level:  Client-goal*

*Primary Actor: Client*

*Main success scenario:  Clientul șterge cu succes activitățile din fereastra cerută*

*Extensions: - Verificare format activitate*

*Use-Case6:  deleteActivity*

*Level:  Client-goal*

*Primary Actor: Client*

*Main success scenario:  Clientul șterge cu succes activitățile din fereastra cerută*

*Extensions: - Verificare existență*

*Use-Case7:  sortActivity*

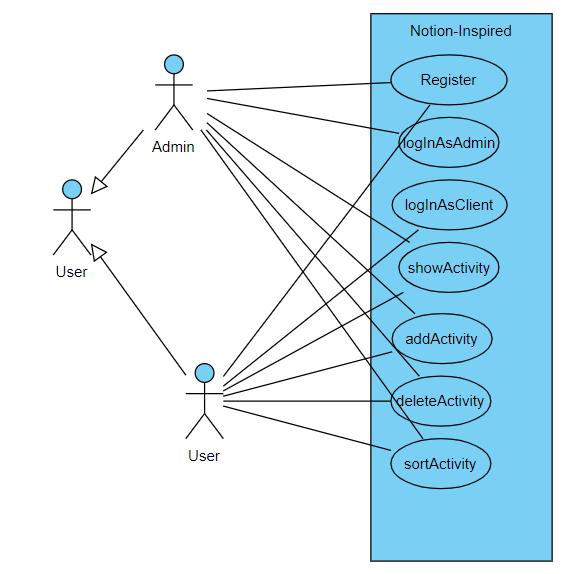
*Level:  Client-goal*

*Primary Actor: Client*

*Main success scenario:  Clientul vizualizeazăa cu succes activitățile sortate din fereastra cerută*

*Extensions: -*

### UML Use Case Diagrams



Bazată pe descrierea cerințelor funcționale din paragraful anterior, diagrama use-case este afișată mai sus.

## Supplementary Specification

### Non-functional Requirements

Datorită design-ului ales pentru aplicație, precum și tipul aplicației, consider că următoarele cerințe non-funcționale sunt esențiale pentru buna funcționare a acesteia:

* **Securitate:** reprezintă o cerință nonfuncțională ce asigură faptul că toate datele cuprinse în interiorul sistemului sunt protejate împotriva atacurilor malware sau împotriva accesurilor nepermise. Consider acest atribut nonfuncțional ca fiind esențial aplicației mele deoarece nu dorim compromiterea datelor utilizatorilor conectați la aplicație. De asemenea, nu dorim ca un admin neautorizat (un admin sub nume fals) să distrugă toate datele înmagazinate de utilizatorii aplicației.
* **Mentenabilitate:** reprezintă un atribut nonfuncțional ce definește timpul necesar pentru ca o soluție să fie fixată, schimbată pentru a crește performanța altor calități, sau adaptată la un mediu în schimbare. Consider necesar acest atribut pentru aplicația în cauză deoarece ulterior se va dori o dezvoltare spre nivelul aplicației Notion într-un timp cât mai mic. Astfel, acest proces va aduce câștiguri atât pe partea de clienți, cât și pe partea de recenzii bune.
* **Performanță:** atribut nonfuncțional ce definește cât de rapid un sistem software sau o parte particulară a sa poate răspunde unor acțiuni ale utilizatorului sub o anumită sarcină de lucru. În cazul acestei aplicații, dorim un răspuns cât mai mic, raportat la numărul de date reținute pentru fiecare dintre cele 4 pagini ale programului. Dacă acest factor temporal este mare, interesul utilizatorului față de aplicația în cauză va scădea drastic.
* **Utilizabilitate:** acest atribut nonfuncțional reprezintă răspunsul la următoarea întrebare: Cât de dificil este de utilizat aplicația? Pentru aplicația în cauză dorim un design și o utilizare cât mai simple, deoarece aplicația necesită atenția constantă a utilizatorului. Schimbările în listele To-Do pot fi de la o zi la alta, proiectele se pot finaliza sau se pot crea proiecte noi, cumpărăturile devin înregistrate zilnic și cărțile pot fi adăugate o dată la câteva zile, astfel atenția consumatorului trebuie stimulată de un design ușor accesibil.

### Design Constraints

Aplicația este implementată cu ajutorul framework-ului Java Spring Boot. Acesta este utilizat în mod deosebit pentru dezvoltarea de aplicații web în limbajul de programare Java. Alegem o abordare orientata-obiect deoarece dorim să organizăm datele și logica aplicației în obiecte reutilizabile.

Pentru simplificarea codului scris în Java se va utiliza librăria Lombok. Aceasta furnizează un set de adnotări care pot fi utilizate pentru a înlocui coduri comune, precum getter-e și setter-e. Astfel este redus codul ce trebuie scris de către un developer, plus eventualele erori ce trebuie depanate.

Datele ce urmează să le manipuleze aplicația sunt reținute în baze de date, baze create cu ajutorul sistemului de management a bazelor de date relaționale MySQL. Folosind acest sistem, datele ce dorim să fie introduse și reținute în cadrul aplicației prind o formă definită de noi, și nu stau aleatoriu în program.

Maparea obiectelor specificate în Java pe o bază de date MySQL este realizată cu ajutorul specificației JPA (Java Persistence API). Această specificație definește o listă de adnotări pentru a putea realiza mapările claselor Java în tabele din baza de date, precum și o listă de API-uri pentru interogarea și manipularea datelor din baza de date. De asemenea sunt definite și tipuri de relații între entități, precum one-to-one, many-to-one și many-to-many.

Legătura între aplicația scrisă în Java și bazele de date definite în MySQL o face API-ul JDBC. Acest API ne permite un acces ușor la baza de date. De asemenea, odată conectați putem executa interogări și extrage date din bază.

## Glossary

[Present the noteworthy terms and their definition, format and validation rules if appropriate.]

# Deliverable 2

## 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.]

### Package Design

[Create a package diagram]

### Component and Deployment Diagram

[Create the component and deployment diagrams.]

# Deliverable 3

## Design Model

### Dynamic Behavior

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

### Class Diagram

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

## Data Model

[Create the data model for the system.]

# System Testing

[Describe the testing methides and some test cases.]

# Future Improvements

[Present some features that apply to the application scope.]

# Conclusion

# Bibliography

<https://www.altexsoft.com/blog/non-functional-requirements/#:~:text=the%20development%20itself.-,Security,malware%20attacks%20or%20unauthorized%20access>. – descriere cerinte nonfunctionale