

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 3](#_Toc1353450407)

[Supplementary Specification 4](#_Toc501087280)

[Non-functional Requirements 4](#_Toc1600632703)

[Design Constraints 5](#_Toc729746755)

[Glossary 5](#_Toc178374120)

[Deliverable 2 5](#_Toc1123232848)

[Domain Model 5](#_Toc605147785)

[Architectural Design 6](#_Toc1115440048)

[Conceptual Architecture 6](#_Toc2005513277)

[Package Design 6](#_Toc596515525)

[Component and Deployment Diagram 7](#_Toc1771682286)

[Deliverable 3 7](#_Toc174956409)

[Design Model 7](#_Toc2136237097)

[Dynamic Behavior 7](#_Toc853475638)

[Class Diagram 9](#_Toc2045288274)

[Data Model 10](#_Toc1322833382)

[System Testing 10](#_Toc975007147)

[Future Improvements 11](#_Toc902595900)

[Conclusion 11](#_Toc1374315000)

[Bibliography 11](#_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*

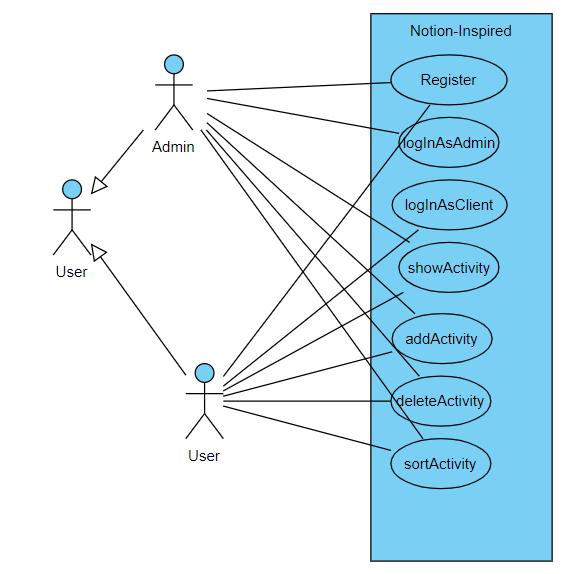
*Level:  Client-goal*

*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

În urma definirii aplicației, modelul acesteia este unul simplu. Acesta este conturat de următoarele:

* User: acesta va reține datele aplicației
* UserRole: rolul destinat utilizatorului, fie de client, fie de admin
* Activity: reprezintă obiectul generic reținut de utilizator
* FinancialActivity, ReadBookActivity, ToDoActivity, ProjectActivity: reprezintă activitățile introduse de utilzator, ce mai apoi vor fi manipulate.

Partea de baze de date și de manipulare a datelor prin servicii nu am considerat-o ca fiind o specificație necesară în acest stadiu al aplicației.

O imagine care conține diagramă

Descriere generată automat

## Architectural Design

### Conceptual Architecture

Arhitectura conceptuală duce mai departe relațiile dintre componentele cheie ale aplicației, punându-le sub un stil/mode architectural. Bazat pe relațiile definite mai sus și pe structura proiectului, acesta se poate încadra în modelul architectural Layered. “Straturile” identificate în cadrul proiectului sunt următoarele:

* Layer-ul Prezentare: Acest strat este responsabil de a oferi o interfață utilizator cu aplicația. Prezentarea este dezvoltată prin implementarea ei în front-end, utilizând limbajul React.
* Layer-ul Aplicație: Acest strat conține logica aplicației. Este responsabilă de a prelua date din aplicație, responsabilă cu procesarea acestora și prezentarea lor într-un format descifrabil. În cadrul aplicației, acest lucru este realizat de Repo-uri, Servicii, Controller-e și Mapper-e.
* Layer-ul Infrastructură: Acest strat se ocupă cu oferirea infrastructurii necesare pentru a suporta funcționalitatea aplicației. În cadrul aplicației, acest lucru este oferit de autorizația și configurarea aplicației web, în funcție de autorizația emisă.

### Package Design

O imagine care conține diagramă

Descriere generată automat

### Component and Deployment Diagram

Bazat pe arhitectura aplicației, diagrama Component arată în felul următor:

O imagine care conține diagramă

Descriere generată automat

În cadrul diagramei, regăsim următoarele componente:

* API – API-ul aplicației ce este utilizat pentru a prelua datele din aplicație
* Database – necesară pentru stocarea datelor
* App Service – serviciu ce este responsabil cu autentificarea și cererile pe partea de API
* Web Page – componenta web a aplicației
* Show Activities – componenta principală din pagina web, o simplă afișare a activităților pentru utilizatorul logat.

În ceea ce privește diagrama Deployment, aceasta arată astfel:

O imagine care conține diagramă

Descriere generată automat

# Deliverable 3

## Design Model

### Dynamic Behavior

**Scenariul 1: Clientul încearcă să se logheze cu un cont inexistent**

* Clientul trimite un request din front-end spre back-end pentru a se putea loga în aplicație. Acest request este trimis spre DemoController, unde este match-uit cu mapping-ul destinat login-ului (/login). Request-ul este trimis spre authenticationManager, ce apelează metoda authenticate, unde se va face verificarea și matching-ul din baza de date. După acest apel, rezultatul returnat poate fi fie un token specific fiecărui utilizator, în caz de succes, fie un UNAUTHORIZED status în caz de failure. Acesta poate fi vizualizat în front-end printr-un pop-ul cu “invalid login’, iar succesul direcționează utilizatorul spre meniul de login.

O imagine care conține text, captură de ecran, diagramă, linie

Descriere generată automat

**Scenariul 2: Clientul încearcă să se înregistreze în aplicație**

* Clientul trimite un request din front-end spre backend pentru a se putea înregistra în aplicație. Acest request este trimis spre DemoController, unde este match-uit cu mapping-ul destinat înregistrării (/request). Request-ul este trimis spre authenticationService, ce apelează metoda register, unde construi user-ul în funcție de datele transmise prin request. User-ul mai apoi este salvat în baza de date a aplicației. Pe lângă acestea, pentru utilizator se va genera și un token de acces în aplicație, precum și criptarea parolei. După acest apel, rezultatul returnat este un token de acces în aplicație, token valabil pentru toată sesiunea de utilizare.

O imagine care conține text, diagramă, captură de ecran, linie

Descriere generată automat

### Class Diagram

O imagine care conține text, captură de ecran, Software multimedia

Descriere generată automat

În cadrul proiectului meu am implementat pattern-ul arhitectural MVC (Model-View-Controller). Partea de model conține logica și datele aplicației. Clasele User, ReadActivity, ProjectActivity, FinancialActivity,și ToDoActivity fac parte din model, fiind responsabile de înmagazinarea și manipularea datelor primite.

Componenta View reprezintă partea vizuală a acestor date, precum și interacțiunea user-ului cu aplicația. Aceasta componentă a fost realizată pe partea de front-end, utilizând HTML, CSS și cod JS scris cu ajutorul librăriei React.

Componenta Controller a acestei aplicații face legătura între componentele Model și View. Aceasta primește datele de la utilizator prin View, le procesează conform Model și apoi le retrimite View-ului pentru afișare. Clase precum DemoControlller, FinancialActivityController, ReadBookActivityController și celelalte controller-e alcătuiesc componenta Controller din cadrul pattern-ului arhitectural MVC.

## Data Model

*O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat*

# System Testing

Metode de testare utilizate în cadrul acestui proiect:

1. Unit Testing : utilizat pentru a testa componente/ părți din aplicație. Prin această testare m-am asigurat că metodele din cadrul back-end-ului funcționează corect; astfel elimin depanarea pe back-end în cazul unor erori din cadrul proiectului

Exemple de testare: testarea pentru fiecare clasă în parte (FinancialActivity,ReadActivity,ToDoActivity,ProjectActivity) a operațiilor CRUD pe elemente (findAll, findByElement, Delete).

1. System Testing : utilizat pentru testarea aplicației în ansamblu pentru a mă asigura de funcționarea bună a acesteia.

Exemple de testare: testarea înregistrării unui utilizator, cu validările de pe frontend, testarea logării unui utilizator existent, testarea afișării activităților pentru un utilizator înregistrat, testare share button, sortare, findBy element introdus și încărcare activități din fișier .CSV

# Future Improvements

Îmbunătățiri ulterioare pe care le-am identificat pentru această aplicație sunt:

1. Refacerea Front-end-ului: până în stadiul actual, front-end-ul a fost realizat doar pentru vizualizarea funcționalităților implementate, fără a avea o ordine când acestea sunt așezate în pagină. De asemenea, accesul la pagini este făcut mecanic, nu prin selectarea unui meniu cum ar fi normal. Această îmbunătățire este prima identificată pentru proiect.
2. Adăugare funcționalități specifice Notion: aplicația în momentul de față implementează funcționalități de baza, precum vizualizare date, sortare etc. Comparativ cu Notion, există multe alte funcționalități ce pot fi integrate în proiect: vizualizarea datelor drept o bază de date, drag&drop, crearea de ferestre etc.
3. Actualizarea cerințelor non-funcționale: aplicația curentă se concentrează mai mult pe cerințele funcționale și pe securitate ca cerință non-funcțională. O îmbunătățire ulterioară este legată de implementarea celorlalte cerințe non-funcționale.
4. Adăugarea de subscripții: adăugarea de funcționalități premium (private celor care nu plătesc un abonament lunar) constituie o îmbunătățire ulterioară. Poate duce la scalarea aplicației și ridicării interesului pentru aceasta.

# Conclusion

Privind în ansamblu, acest proiect s-a dovedit unul interesant pentru mine și am putut învăța multe lucruri din acesta. De la cum se implementează o idee de început în funcție de programele alese, până la gestiunea timpului pentru a prezenta proiectul, această aplicație s-a dovedit o ocazie bună de învățare și acumulare de experiență.

# Bibliography

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

<https://legacy.reactjs.org/docs/getting-started.html> - documentație React

<https://www.baeldung.com/spring-boot> - documentație + site learn Spring Boot