

Filmento

Name: Gligor Mihai

Group: 30243

Table of Contents

[Project Specification 2](#_Toc124517719)

[Functional Requirements 2](#_Toc124517720)

[Database Design 3](#_Toc124517721)

[Architectural Design 3](#_Toc124517722)

[Conceptual Architecture 3](#_Toc124517723)

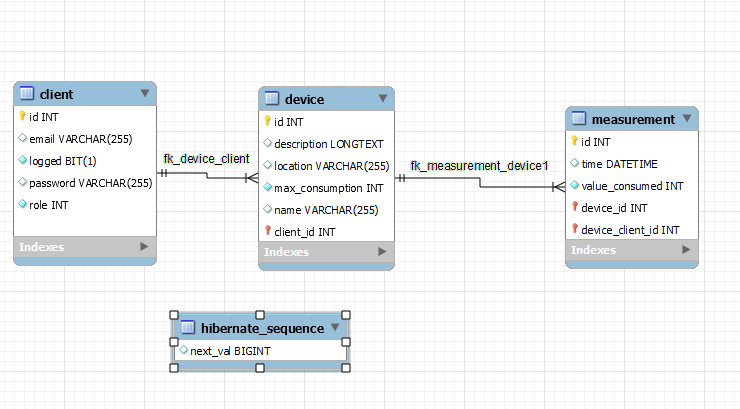
## Project Specification

Proiectul current permite utilizatorilor sa isi creeze un cont , sa se intre pe conturile lor si sa vizualizeze device-urile . Adminului poate sa adauge un device ,sa stearga un device sau un client si sa atribuie clientilor device-uri .

## Functional Requirements

* Inregistrare : pentru ca user ul sa poata folosii site ul filmento , acesta trebuie sa aiba mai intai un cont , acest lucru fiind usor realizabil prin intermediul paginii de access . Mai intai clientul trebuie sa introduca un email valid , dupa care sa introduca parola Dorita si sa o repete pentru a se asigura ca a introdus parola correct.
* Login : aceasta functionalitate permite accesul clientului la aplicatia filmento si se realizeaza prin intermediul paginii de access prin introducere email-ului si a parolei corespunzatoare contului .
* List : Fiecare utilizator , odata ajuns pe pagina de My List poate sa vizualizeze device urile pe care le are inregistrate pe cont de catre admin .

## Database Design



Baza de date are un design relativ simplu care se bazeaza pe doua relatii de one to many. Una dintre relatii se intampla intre device si client (un client are mai multe device uri) si cealalta intre device uri si masuratori (fiecarui device ii se fac mai multe masuratori ).De asemenea mai avem un tabel in care se retine automat ultimul id generat

## Architectural Design

### Conceptual Architecture

O imagine care conține text, interior, monitor

Descriere generată automat

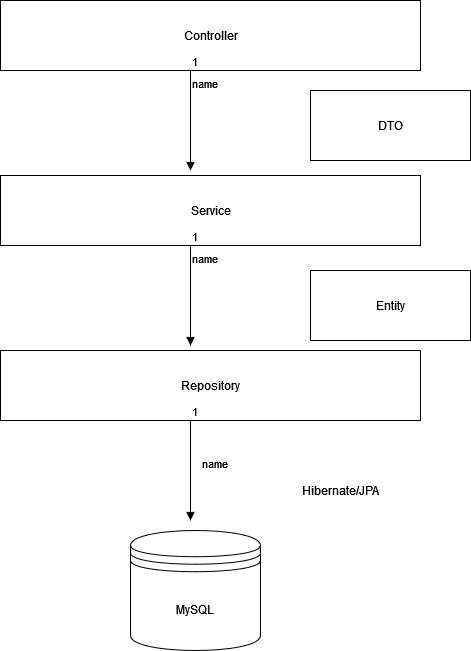
Arhitectura layered , este organizarea structurii proiectului în patru categorii principale: Prezentare, aplicație, domeniu și infrastructură. Fiecare dintre straturi conține obiecte legate de preocuparea specială pe care o reprezintă.

presentation layer conține partea responsabila pentru interfata grafica sau trimiterea răspunsului înapoi la client.

aplication layer conține toată logica cerută de aplicație pentru a-și îndeplini cerințele funcționale și, în același timp, nu face parte din regulile domeniului.

domain layer reprezintă domeniul de bază, constând în principal din entități de domeniu și, în unele cazuri, servicii. Regulile de afaceri, cum ar fi invarianții și algoritmii, ar trebui să rămână toate în acest strat.

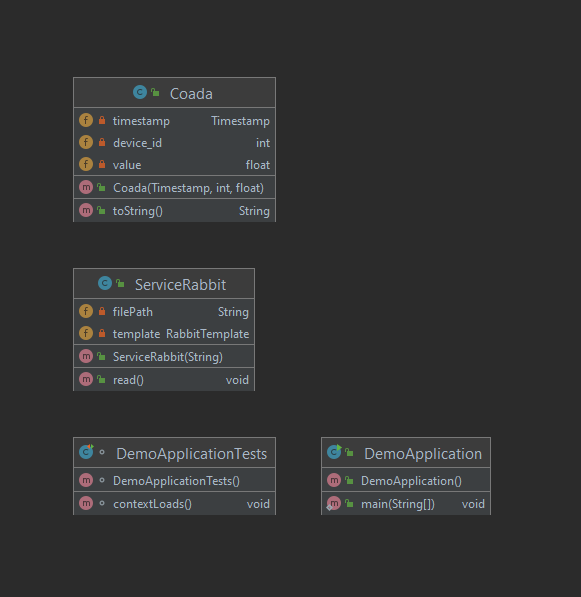
database layer (cunoscut și sub numele de stratul de persistență) conține toate clasele responsabile pentru a face lucrurile tehnice, cum ar fi persistența datelor în baza de date, cum ar fi DAO-uri, depozite sau orice altceva pe care îl utilizați.



### 

**WebSocket** API este o tehnologie avansată care face posibilă deschiderea unei sesiuni de comunicare interactivă bidirecțională între browserul utilizatorului și un server. Cu acest API, puteți trimite mesaje către un server și puteți primi răspunsuri bazate pe evenimente fără a fi nevoie să chestionați serverul pentru un răspuns.

**RabbitMQ** este un software open-source message-broker care a implementat inițial Advanced Message Queuing Protocol și a fost extins cu o arhitectură plug-in pentru a sprijini streaming Text Oriented Messaging Protocol, MQ Telemetry transport și alte protocoale.



**Readme**

Pentru a rula acest proiect, utilizatorul va trebui să aibă instalat un JDK și NodeJS și Postgres.

Pentru a rula backend-ul aplicației, este recomandat un IDE (intellij), configurați JDK și JRE, apoi construiți aplicația. Când începe backend-ul, generează automat tabelele necesare în baza de date și populează baza de date a senzorilor cu date simulate.

Când vine vorba de frontend, se recomandă un IDE (am folosit Visual Studio), apoi, într-o consolă care are calea setată la folderul rădăcină al frontend, rulăm comanda „npm install” și după aceea se termină execuția, „npm run dev”.