Student:Buretia Alex-Dian

**Group:30238**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

[Application description]

Trebuie sa facem o aplicatie de monitorizare a viselor. Utilizatorul va introduce numele, durata, stresul si nivelul de energie dobandita in urma somnului.

# Functional Requirements

*[Present the functional requirements]*

Baza de date utilizata de mine este PostresQL si clientul va introduce acele date intr-un form conceput cu ajutorul framework-ului React. In Java si Spring se vor prelua acele date si se vor executa operatiile si se vor transmite la baza de date si la fisierul de creere rapoarte.

# Non-functional Requirements

*[Discuss the non-functional requirements for the system]*

Arhitectura layerd si design patternul factory sunt utilizate in cadrul acestui proiect. Factory pattern este menit sa creeze 3 rapoarte(de durata, stress si nievlul de energie). Implementarea mea a fost sa scriu simplu intr-un fisier aceste informatii.

2. Use-Case Model

*[Create the use-case diagrams and provide one use-case description (according to the format below).*

*Use-Case description format:*

*Use case: <use case goal>*

*Level: <one of: summary level, user-goal level, sub-function>*

*Primary actor: <a role name for the actor who initiates the use case>*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*

*]*

USE-CASE: introducerea de vise de catre utilizator

Level:

Primary actor: utilizatorul

Main success scenario : cee ace introduce utilizatorul este prelucrat de Java si Spring BOOT si trimis la baza de date pentru a fi stocat.

USE-CASE: creerea de rapoarte

Level:

Primary actor: utilizatorul

Main success scenario : ceea ce introduce utilizatorul este prelucrat de Java si Spring BOOT si trimis la baza de date pentru a fi stocat.

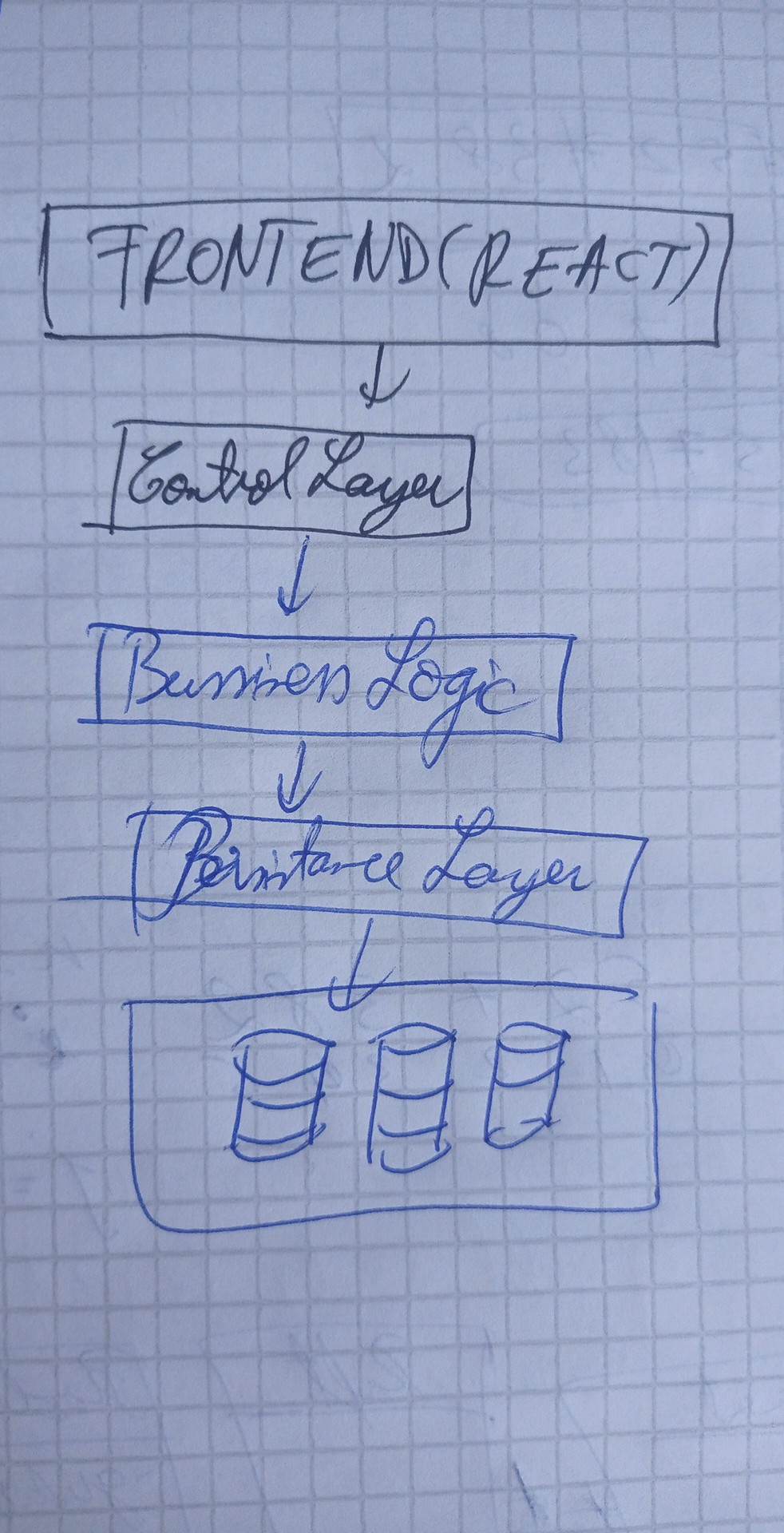
3. System Architectural Design

**3.1 Architectural Pattern Description**

*[Describe briefly the used architectural patterns.]*

Layerd architecture l-am utilizat in cadrul proiectului. Exista 4 layere(Presentation- care contine frontEnd-ul, BusinessLogic in cadrul careia se fac principalele operatii, PersistanceLayer care realizeaza conexiunea la baza de date si Baza de date in sine).Astfel, acest layerd architecture m-a ajutat sa am o coeziune a pachetelor mai mica, sa nu depinde unul de altul asa de mult.

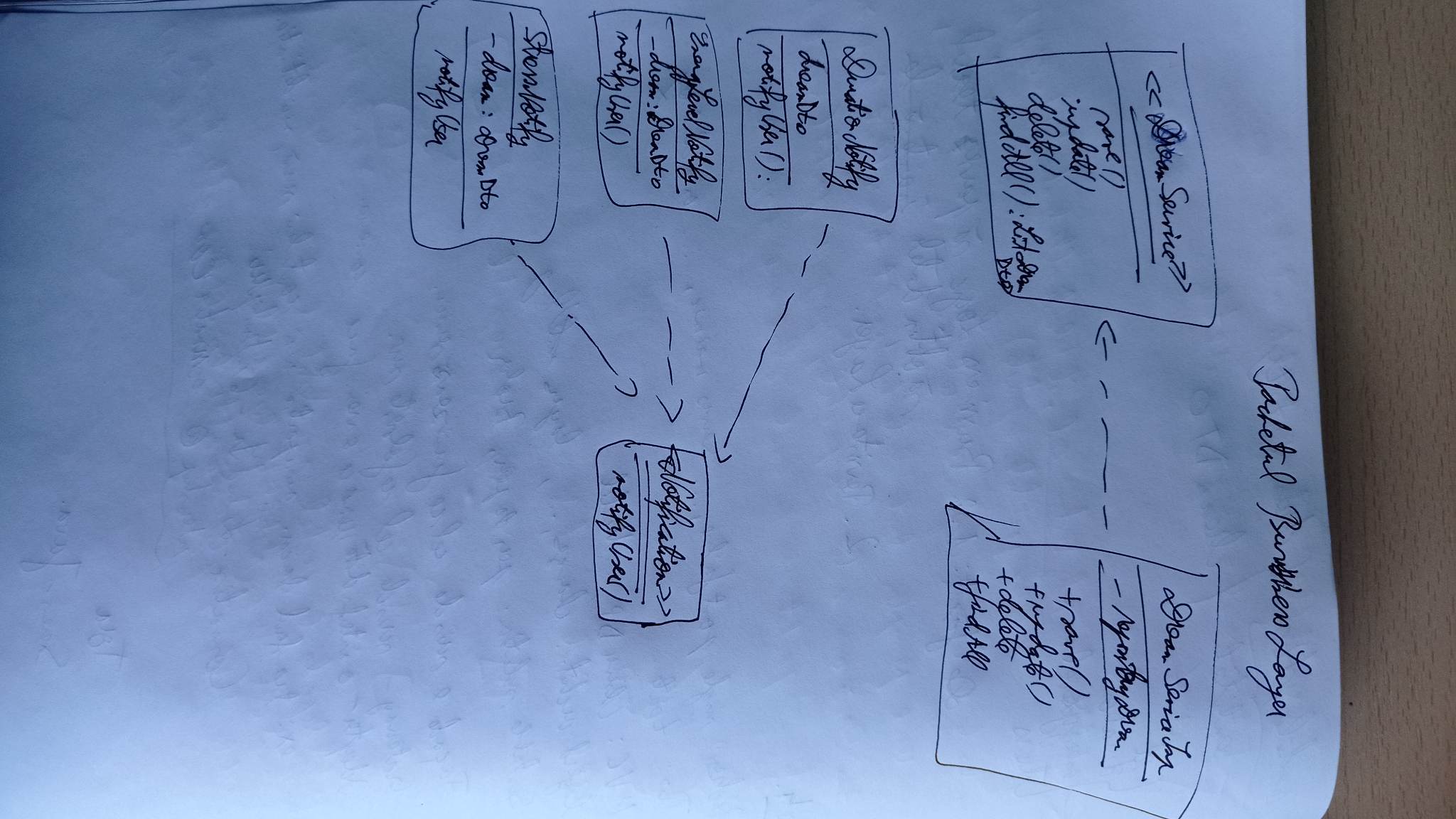
**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams] *

4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*

In diagram de mai jos, putem observa cu usurinta care este diagram UML corespunzatoare layer-ului BusinessLogic, cel care stocheaza operatiile de baza facute de catre programul nostru.

**

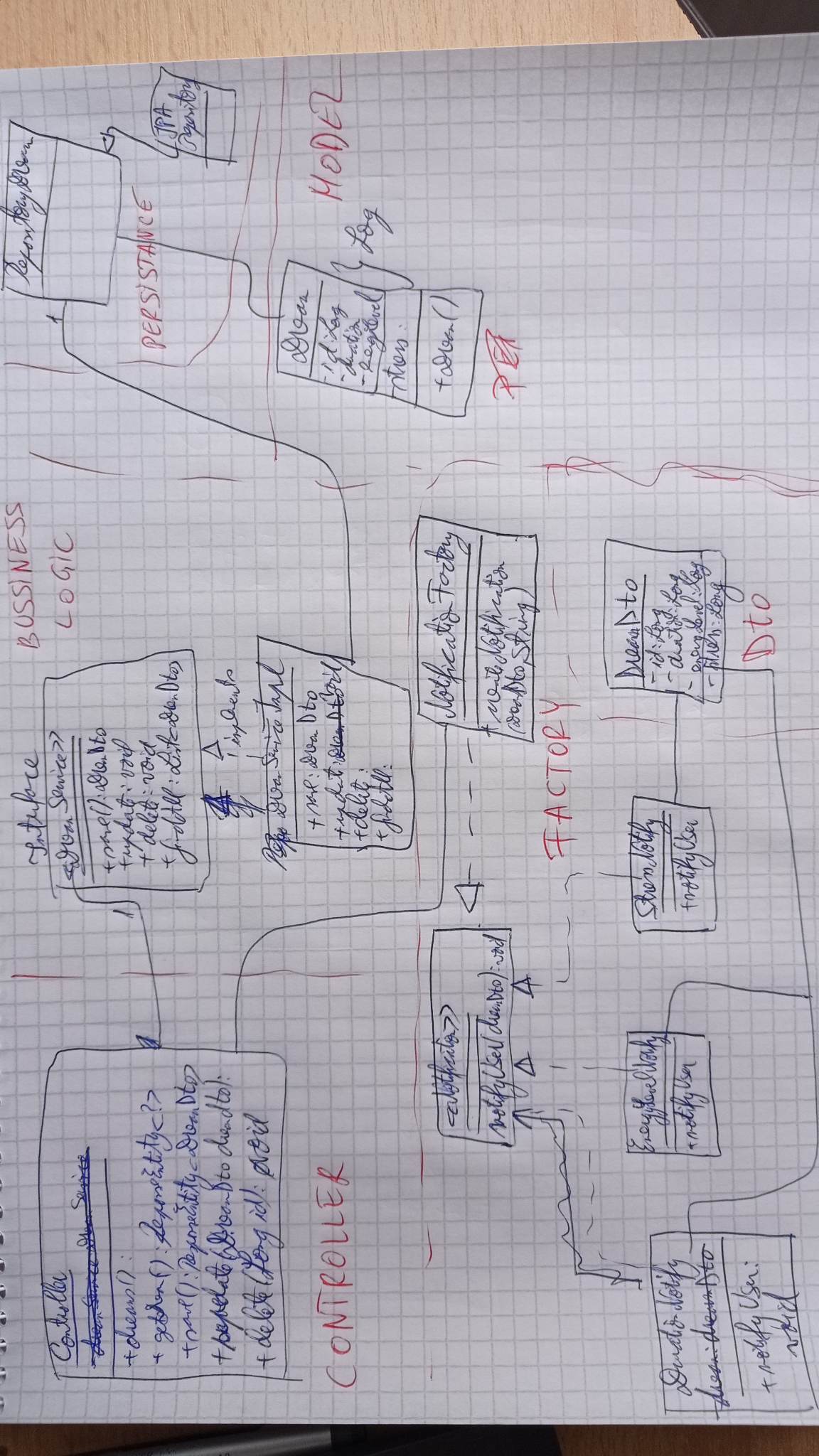
5. Class Design

**5.1 Design Patterns Description**

Design pattern-ul folosit este Factory. Cu ajuotul acestuia am creat niste report-uri legat de acitivitatea din timpul somnului. Astfel, am utilizat o interfata care are o metoda care va fi implementata de 3 clase(DurationNotify, EnergyLevelNotify si StressNotify). Decizia este luata o clasa pe baza unui string trimis ca parametru din controller, din metoda de POST, care ia din baza de date.

**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*



6. Data Model

*[Present the data models used in the system’s implementation.]*

Modelul folosit este Dream. Acesta contine 3 field-uri( energyLevel, stress si duration) de tipul Long. ID-ul este autogenerat. Pe baza acestor field-uri se creaza baza de date.

7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

8. Bibliography