EFitness

Humeiuc Dan-Stefan

Cotei Ruxandra-Maria

Profesor indrumator: Mitrea Delia-Alexandrina

1. Cuprins

2. Specificații şi analiza sistemului

Aplicația de fitness va oferi utilizatorilor o soluție simplă și eficientă pentru gestionarea obiectivelor de sănătate și fitness.

Funcționalitățile principale vor include urmărirea progresului fizic, planificarea antrenamentelor și monitorizarea alimentației, toate disponibile într-un format ușor de folosit.

Utilizatorii vor putea să își creeze planuri personalizate de antrenamente și să își urmărească evoluția în timp real, cu ajutorul unor grafice clare și statistici care le vor permite o evaluare constantă a progresului.

Mai mult decât atât, aplicația va include opțiuni suplimentare care vor face experiența mai plăcută și mai motivațională, precum oferirea de sugestii pentru antrenamente sau mese, în funcție de obiectivele individuale.

Aceste funcționalități vor ajuta utilizatorii să își mențină un ritm constant de îmbunătățire a sănătății și performanțelor fizice, fără a fi necesare integrarea cu dispozitive externe sau notificări constante.

Aplicația se va concentra pe simplitate și eficiență, oferind toate instrumentele necesare pentru un trai mai sănătos, într-un format accesibil și ușor de utilizat.

Cerinte functionale:

Must have:

- goal tracking
- food tracking
- workout tracking

Should have:

- workout planner

Could have:

- Animation
- Connect with my trainer
- Map with closest gyms

Cerinte functionale:

Performanta

Scalabilitate

Securitate

3. Precizare limbaj şi sistem de operare

FrontEnd:

Partea de interactiune cu utilizatorul si interfata grafica se va reliza in framework-ul React petru simplitatea si fiabilitatea acestuia .

BackEnd:

Partea de server se va realiza in C++ cu ajutorul farmework-ului CrowCPP si alte pachete ajuatoare pentru performanta si versatilitatea acestuia (performanta permite utiilizarea algoritmilor inceti pentru hashing-ul parolelor in baza de date)

Sistem de operare

Aplicatia v-a fi una distribuita, pe microservicii astfel s-a ales sa se containerizeze pe folosind Docker; deci sistemul de operare pe care vor rula executabile este Linux.

4. Proiectarea aplicației

4.1.Descrierea arhitecturii sistemului

Microserviciile permit scalarea independentă a fiecărui modul. Dacă un serviciu specific are nevoie de mai multe resurse (de exemplu, procesarea plăților), acesta poate fi scalat separat, fără a afecta restul sistemului.

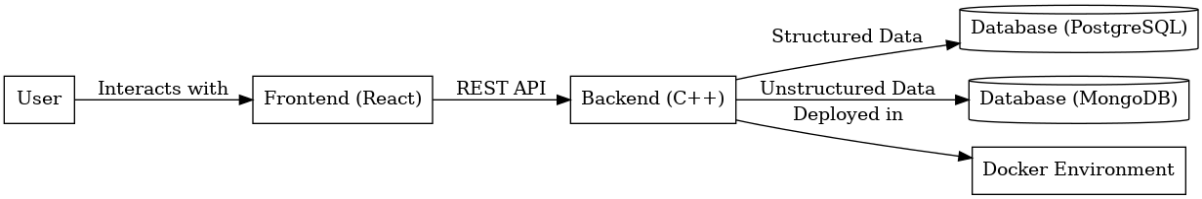
Fiecare microserviciu poate fi dezvoltat și implementat separat, reducând dependențele dintre echipe și accelerând livrarea. Echipele pot folosi tehnologii diferite, adaptate fiecărei funcționalități.

Eșecul unui serviciu nu afectează întreaga aplicație. Doar funcționalitatea respectivă este indisponibilă, restul sistemului continuând să funcționeze.

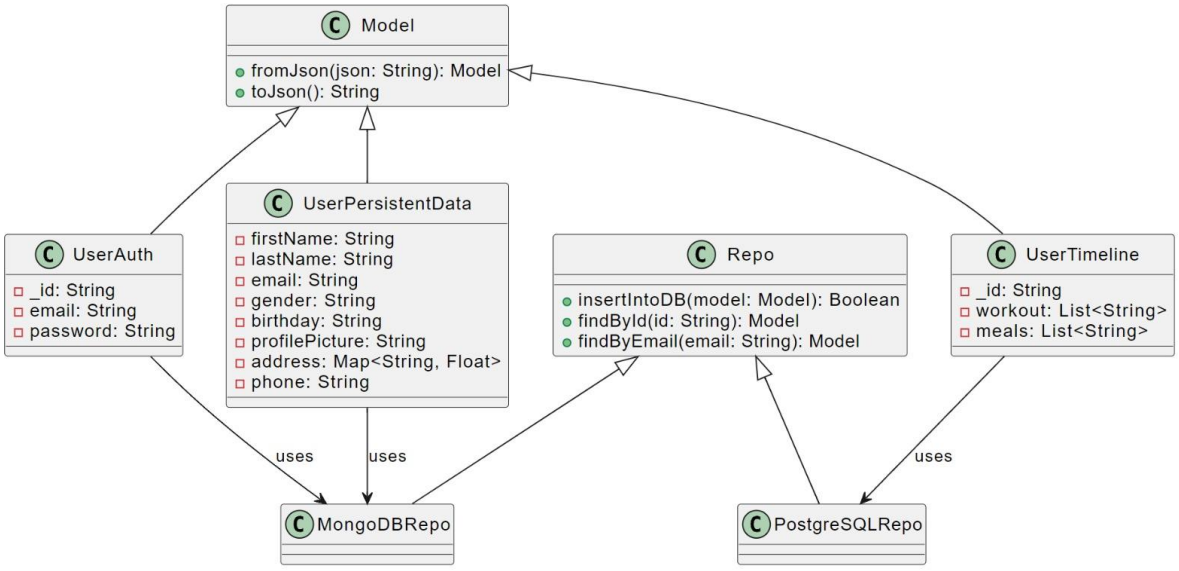
Serviciile sunt autonome și pot fi reutilizate în alte aplicații, economisind timp și resurse.

Microserviciile se integrează bine cu practicile DevOps și livrarea continuă (CI/CD), simplificând implementările și actualizările frecvente.

Deși extrem de eficiente, aceste arhitecturi pot aduce provocări, precum complexitatea gestionării și necesitatea unor instrumente avansate pentru orchestrare și monitorizare.

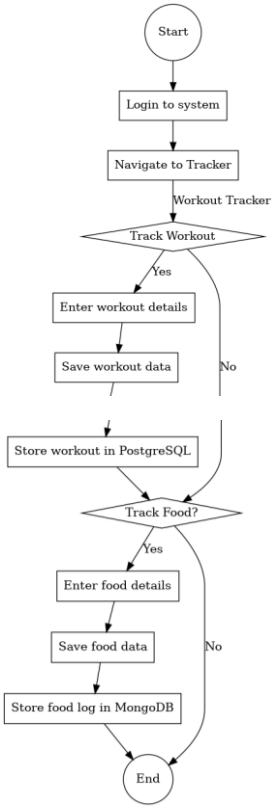


4.3.Diagrama de clase

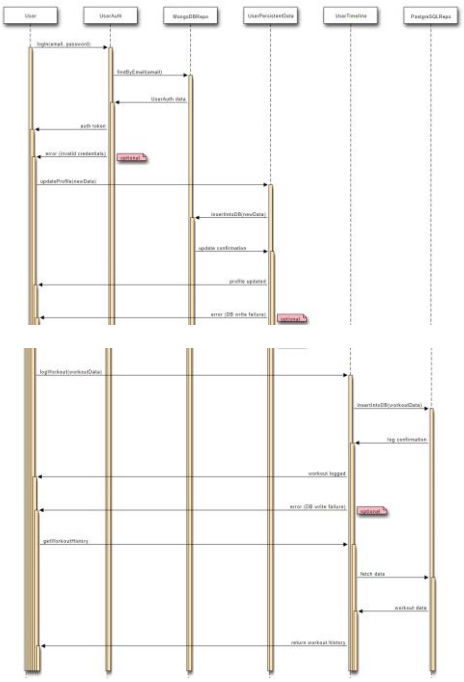


4.4.Diagrame de interacțiune: activitate, secvență

Activitate:



Secventa:



4.5. Detalii de implementare: diagrama bazei de date, diagrama de clase cu attribute şi metode, diagrama de module şi componente

Diagrama bazei de date:

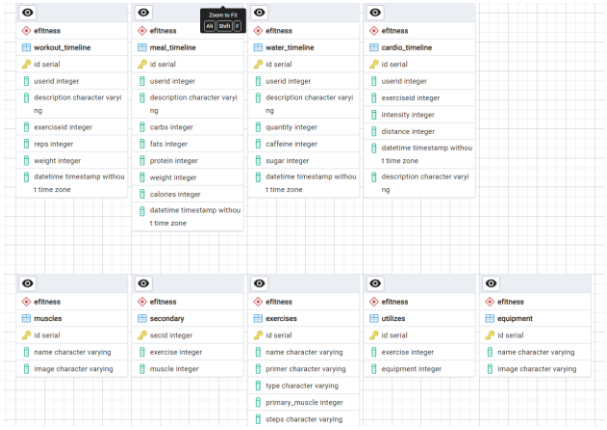


Diagrama de clase:

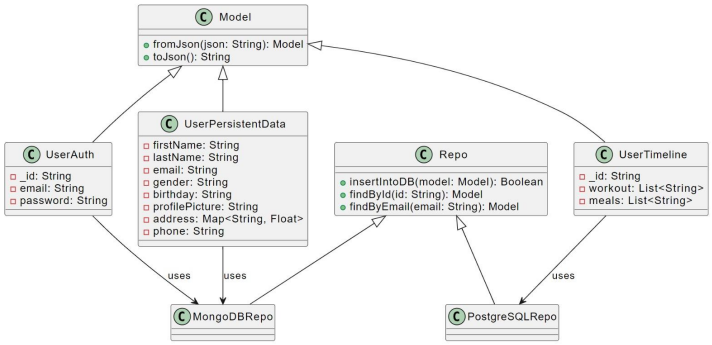
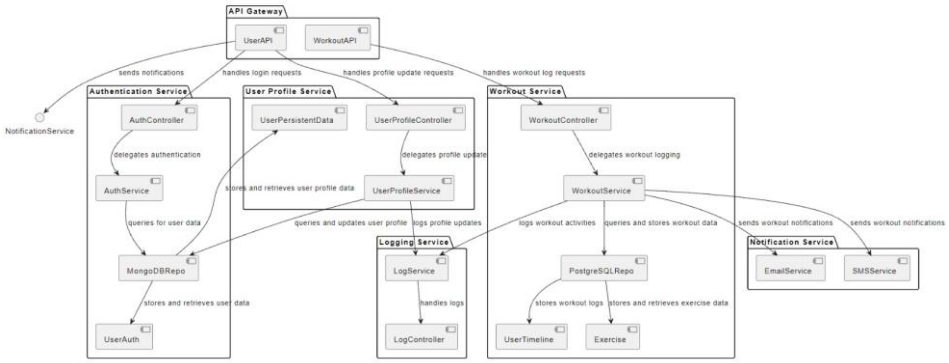


Diagrama de module si componente:



Design Patterns

In aceasta aplicatie este utilizat pattern-ul de SINGLETON pentru a permite driver-ului de MongoDB sa existe pe tot parcursul aplicatiei fara sa se reinstantieze conform documentatiei acestuia.

Exista un system de PUB-SUB pentru a realiza comunicarea intre containere. Pentru implementarea acestuia se foloseste ZeroMQ care comunica intre instantele sale folosind TCP.

In ultimul rand se utilizeaza BUILDER pentru creerea obiectelor CPP din Json. Unele campuri pot fi Null sau inexistente astfel Builider evita potentiale probleme care ar putea aparea in cadrul parsarii si interpretarii.

5. Testarea propriu-zisă a aplicației

Pentru a testa aplicația de fitness dezvoltată în C++, am utilizat o combinație de testare manuală și testare automată cu ajutorul unui framework de unit testing.

Testarea Manuală: Am verificat manual funcționalitățile aplicației, simulând utilizarea acesteia de către utilizatori reali.

Scenarii de testare funcțională:

Autentificare și înregistrare: Am testat introducerea de date valide și invalide pentru utilizatori noi și existenți.

Urmărirea progresului: Verificarea calculului corect al caloriilor arse, duratei antrenamentului și evoluției utilizatorilor.

Setarea obiectivelor: Asigurarea că utilizatorii pot seta și modifica obiectivele de fitness.

Interfața: Am testat toate comenzile (meniuri, opțiuni) pentru a verifica dacă interacțiunea cu utilizatorul este intuitivă.

Testare negativă: Am introdus date incorecte sau incomplete (de exemplu, nume goale sau valori negative pentru calorii) pentru a verifica gestionarea excepțiilor.

Compatibilitate: Aplicația a fost testată pe diferite configurații hardware și sisteme de operare pentru a verifica funcționarea constantă.

Testarea Automată cu Framework de Unit Testing

Am utilizat un framework de testare, cum ar fi Google Test (gTest), pentru a asigura acuratețea logicii aplicației.

Structura testelor unitare:

Teste pentru module: Fiecare componentă (de exemplu, clasa User, WorkoutPlan, ProgressTracker) a fost testată separat pentru funcționalitate.

Metode cheie: Am creat teste pentru funcții importante precum calculateCaloriesBurned(), setGoal() sau trackProgress().

Tipuri de teste unitare:

Teste pozitive: Verificarea funcționării corecte pentru date valide.

Teste negative: Asigurarea că erorile sunt gestionate corect pentru date invalide sau excepții.

Teste de performanță: Am verificat eficiența algoritmilor folosiți în procesarea datelor (de exemplu, calculul progresului).

Exemple de cazuri de testare:

Pentru funcția calculateCaloriesBurned(weight, duration), am testat:

Valori normale: (70, 30) → rezultat așteptat: 210.

Valori la limită: (0, 0) → rezultat așteptat: 0.

Date incorecte: (-5, 30) → gestionare prin excepție.

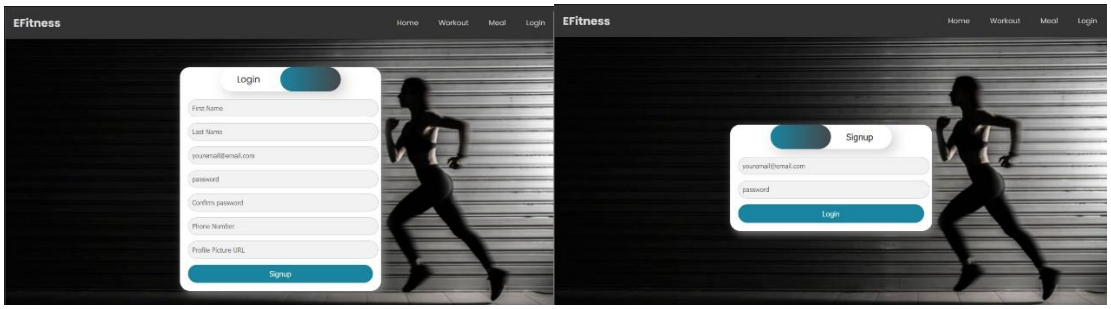
Integrarea testelor:

Am folosit pipeline-uri automate pentru a rula testele la fiecare modificare semnificativă a codului.

6. Manual de instalare şi utilizare

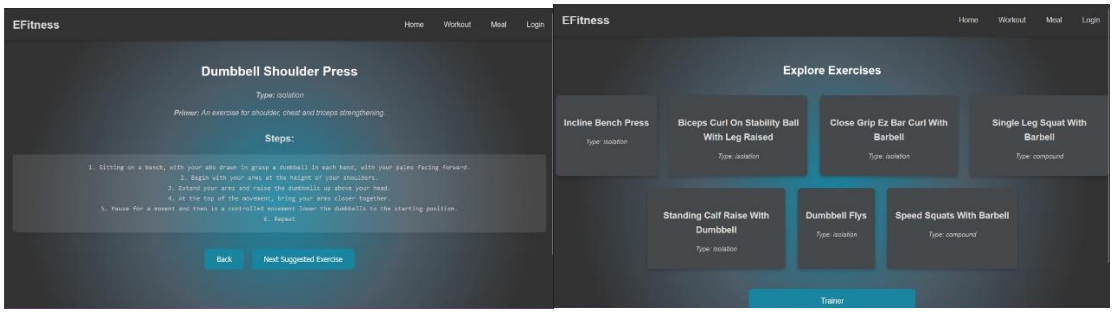
Proiectul permite crearea unui cont prin functia de Signup, pe baza caruia se vor retine informatiile introduce.

In pagina de Login, utilizatorul se conecteaza si datele inregistrate anterior vor fi disponibile pe celelalte pagini.

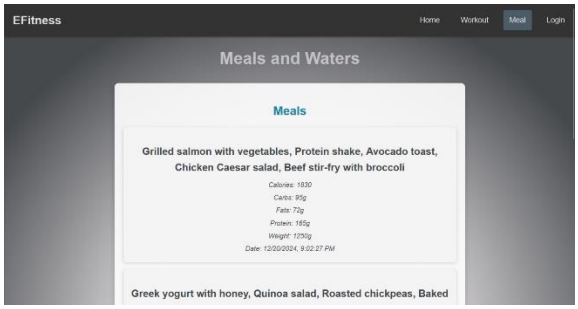


Din pagina Workouts utilizatorul poate vedea exercitii generate pentru antrenamentul sau, iar, prin apasarea pe ele, acestea se extind si apar toate detaliile referitoare la ele.

Optiunea Trainer a paginii de Workouts va incarca exercitii personalizate de Trainer pentru fiecare utilizator.



Pagina Meals permite vizualizarea meselor introduse cu toate detaliile necesare, precum si a cantitatilor de apa baute.



7. Concluzii şi dezvoltări ulterioare

Pe lângă plannerul bazat pe un algoritm tradițional sau cel personalizat creat de un trainer uman, se poate implementa și un planner bazat pe rețele neuronale, care poate aduce beneficii semnificative utilizatorilor prin personalizare avansată și adaptabilitate.

Planner bazat pe rețele neuronale:

Acest tip de planner folosește tehnici de Machine Learning (ML) și Deep Learning pentru a genera planuri de antrenament și nutriție personalizate, adaptate nevoilor și obiectivelor fiecărui utilizator.

8. Bibliografie

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides.
Design Patterns: Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1994.
This seminal book introduces 23 classic design patterns and provides examples applicable across multiple programming languages, including C++.

Freeman, Eric, and Elisabeth Robson.
Head First Design Patterns: A Brain-Friendly Guide.
O'Reilly Media, 2004.
A practical and engaging book that introduces design patterns with real-world examples and applications.

Meyers, Scott.
Effective C++: 55 Specific Ways to Improve Your Programs and Designs.
Addison-Wesley, 2005.
A must-read book that includes design pattern best practices tailored to C++.

Meyers, Scott.
More Effective C++: 35 New Ways to Improve Your Programs and Designs.
Addison-Wesley, 1996.
Explores advanced design pattern strategies in C++ with practical insights.

Lakos, John.
Large-Scale C++ Software Design.
Addison-Wesley, 1996.
Focuses on design strategies and patterns for developing large-scale C++ applications.

Alexandrescu, Andrei.
Modern C++ Design: Generic Programming and Design Patterns Applied.
Addison-Wesley, 2001.
A deep dive into advanced C++ design patterns and generic programming techniques.

Stroustrup, Bjarne.
The C++ Programming Language.
Addison-Wesley, 2013 (4th Edition).
Comprehensive coverage of the C++ language, including sections that touch on design patterns.

Goos, Gerhard, and Juris Hartmanis (Eds.).
Object-Oriented Design with Applications.
Addison-Wesley, 1991.
Explores foundational concepts of object-oriented design, including early discussions of patterns.

Coplien, James O.
Advanced C++: Programming Styles and Idioms.
Addison-Wesley, 1992.
Discusses idiomatic programming styles in C++ that form the foundation for many modern design patterns.