DISCODE

Platformă de chat cu execuție de cod integrată

Arhitectura sistemului

Nume proiect	DISCODE
Nume document	Arhitectura sistemului / Architectural Design Document
Autori	Beldiman Vladislav, Rusu Iulian, Nistor Paula-Alina
Reviewed by	-
Status	În lucru

Versiuni

Dată	Versiune	Modificări
21.10.2021	0.1	Structura documentului. Adăugarea introducerii, prezentării sistemului, arhitecturii top-level
24.10.2021	0.2	Diagramele pentru baza de date și componentele serverului backend
15.12.2021	0.3	Actualizarea arhitecturii cu serverul de web sockets
19.01.2021	0.4	Actualizarea diagramelor de componente

1 Introducere

1.1 Scopul documentului

Scopul documentului “Arhitectura sistemului” este de a acoperi arhitectura şi proiectarea sistemului la nivel înalt. Identifică componentele principale ale sistemului, şi oferă ghidare privind funcționalitățile acestora şi modul în care acestea sunt legate între ele la nivel arhitectural.

Acest document este necesar în vederea dezvoltării arhitecturii sistemului.

1.2 Definiții şi abrevieri

- UAC - User Access Control
- SGBD - Sistem de Gestiune a Bazei de Date
- ER Diagram - Entity-Relationship Diagram
- SQL - Structured Query Language
- MVC - Model-View-Controller
- REST - Representational State Transfer
- JPA - Java Persistence API

2 Prezentarea sistemului

Proiectul Discode este o soluție desktop/web prin care utilizatorii pot comunica prin mesaje scrise şi executa cod în diverse limbaje prin intermediul acestor mesaje. Această aplicație oferă posibilitatea de a crea chat-uri de grup, de a personaliza profilurile utilizatorilor şi de a raporta mesajele nepotrivite. De asemenea, aplicația vine cu un client desktop pentru administrare şi analiza mesajelor raportate, funcționalitate ce va fi disponibilă doar utilizatorilor cu rol de administrator.

3 Arhitectura top-level

3.1 Identificarea componentelor

Componentele principale ale sistemului sunt:

1. Frontend-ul implementat în Typescript utilizând Angular;
2. Serverul web backend, implementat utilizând Kotlin şi framework-ul Spring Boot;
3. Serverul de web sockets, implementat în Node.js cu socket.io
4. Serverul de baze de date, care va folosi baza de date relațională MySQL.

Componente-client:

1. Aplicația desktop (dezvoltată intern) implementată în C++ utilizând framework-ul Qt;
2. Browserul web (third-party), utilizat pentru a comunica cu serverul web frontend.

3.2 Relațiile de comunicare dintre componentele top-level

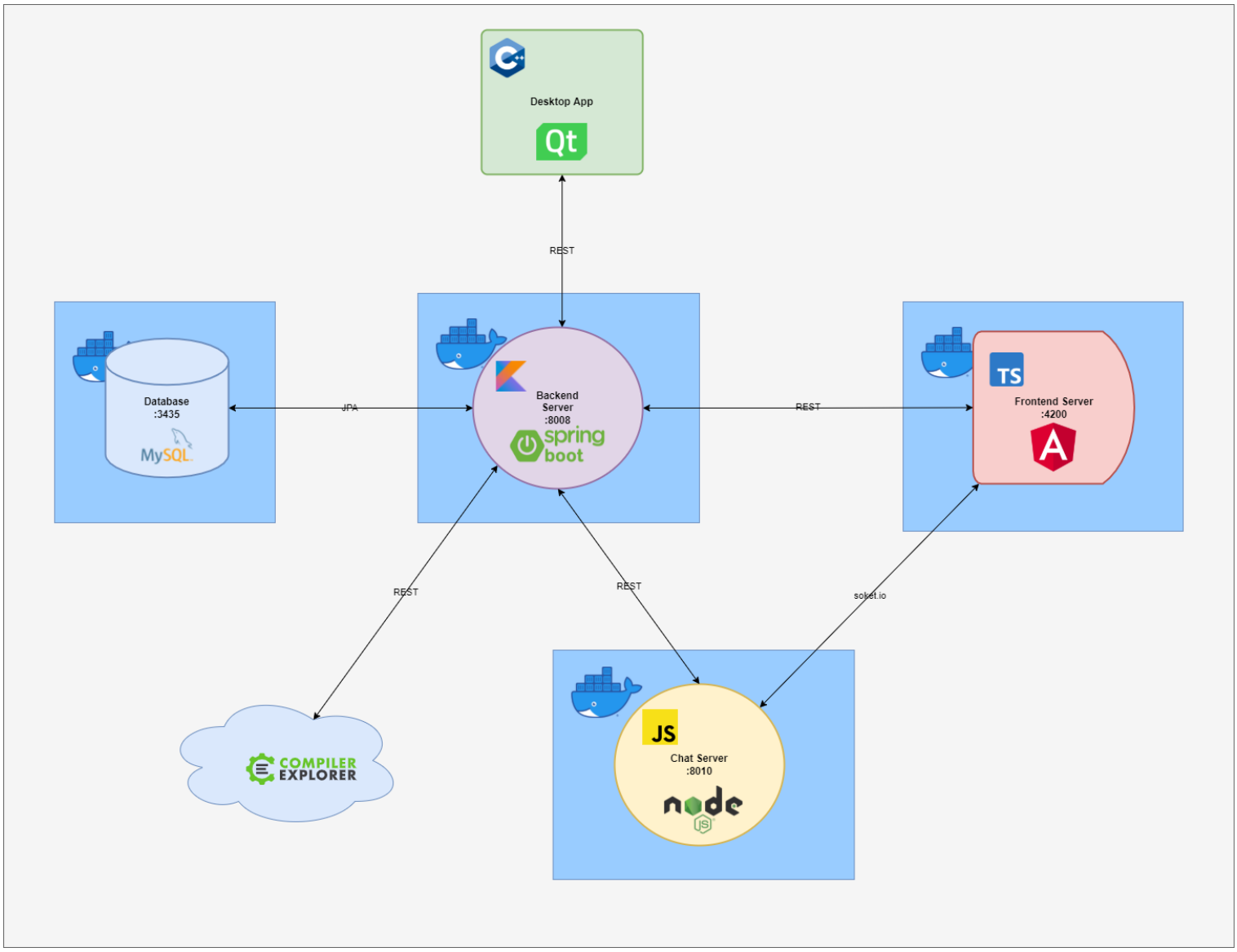

Figura 1 - Diagrama arhitecturii aplicației

4 Descrierea componentelor

4.1 Serverul de web sockets

Acest server va fi utilizat pentru a implementa funcționalitatea de notificări la primirea unui mesaj în chat. Comunicarea cu clientul se va face prin web socket-uri, utilizate de biblioteca socket.io din Javascript. Serverul propriu-zis va fi creat cu framework-ul Express și executat utilizând Node.js.

4.2 Serverul de baze de date

Această componentă este reprezentată de un SGBD relațional (MySQL). S-a ales utilizarea bazelor de date SQL pentru definirea ușoară a relațiilor dintre entitățile modelului. Astfel, în urma proiectării s-a stabilit următoarea diagramă ER a bazei de date:

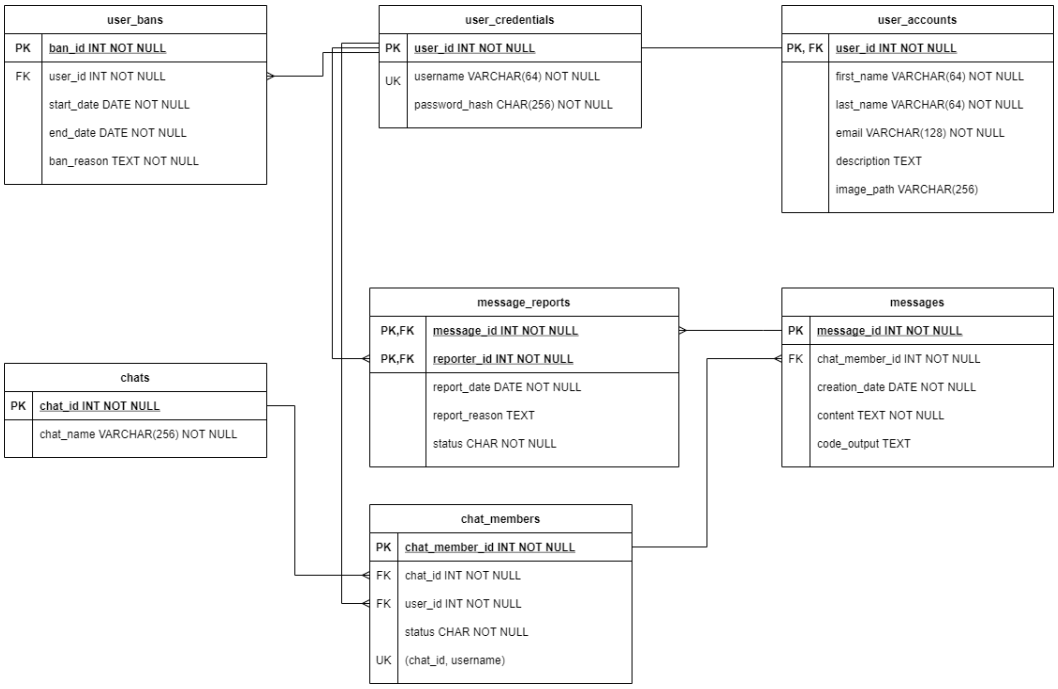

Figura 2 - Diagrama ER

4.3 Frontend-ul Web

Pentru implementarea părții web de frontend se va folosi framework-ul Angular împreună cu limbajul Typescript.

Aplicația web va fi compusă din mai multe componente, principalele componente fiind: AuthenticationComponent, HomeComponent, AdminComponent, MessageComponent, ChatComponent, ReportComponent, ProfileComponent și CodeSnippetComponent.

Comunicarea cu serverul de backend se va realiza prin intermediul serviciilor care vor face cereri către API. Serviciile vor fi incorporate în componente folosind Dependency Injection.

Atât componentele, cât și serviciile se vor utiliza de modele pentru a serializa și deserializa datele ce vor fi trimise sau primite de la serverul de back end.

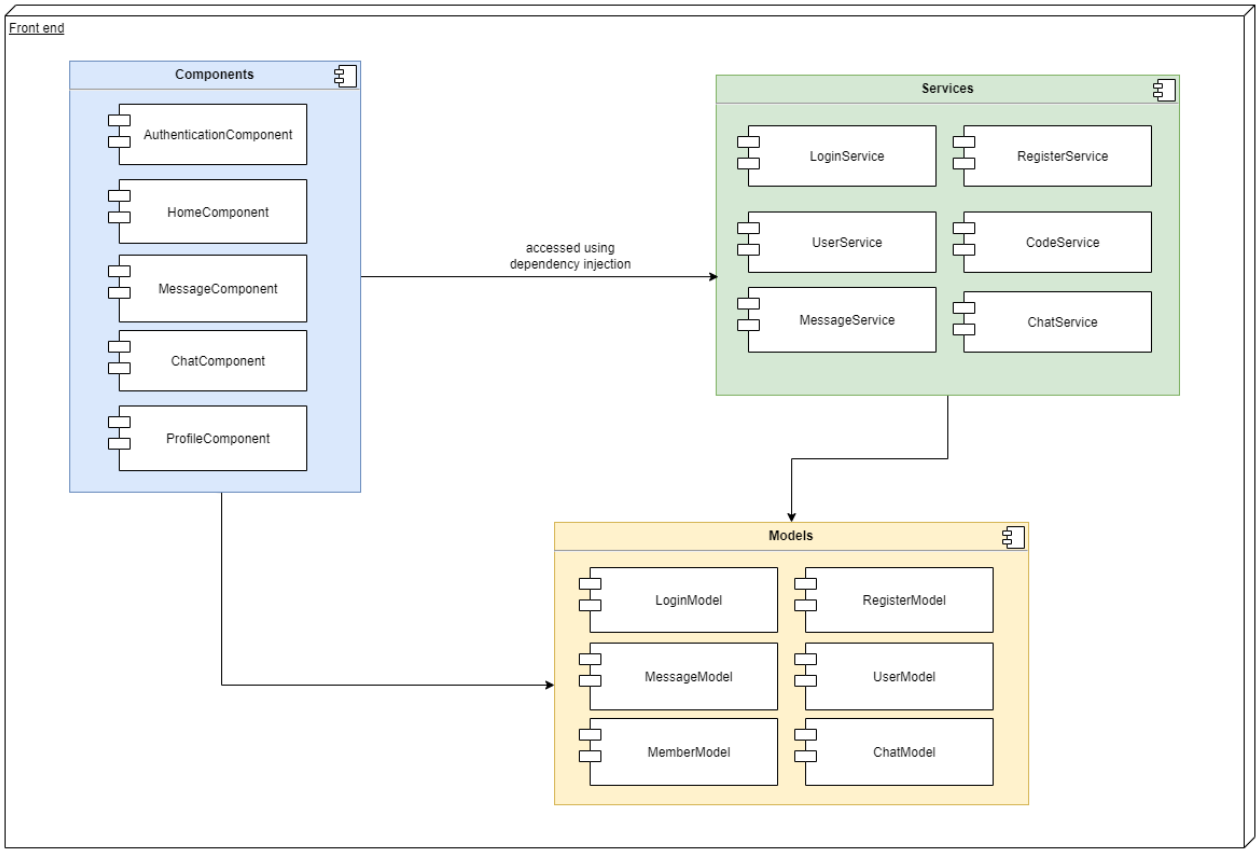

Figura 3 - Diagrama arhitecturii front end

4.4 Serverul web backend

Pentru implementarea acestei componente se va utiliza framework-ul Spring Boot în Kotlin. Arhitectura generală este una de tip MVC, fapt ce semnifică separarea modulelor în 3 niveluri:

1. Modelul, care este o abstractizare a datelor relevante;
2. View-ul, care conține diverse reprezentări pentru model. La acest nivel vor apărea componente ce gestionează modul în care serverul va transmite datele în răspuns clienților;
3. Controller-ul, care realizează legătura dintre model și view. Pentru a implementa logica de business mai complicată din controller, vor fi create servicii specializate.

Considerând aceste principii de arhitectură, s-au identificat următoarele componente principale:

1. Componente ce gestionează autentificarea și autorizarea clienților;
2. Componente ce gestionează mesajele și chat-ul;
3. Componente ce gestionează execuția codului;
4. Componente ce gestionează raportarea și blocarea utilizatorilor.

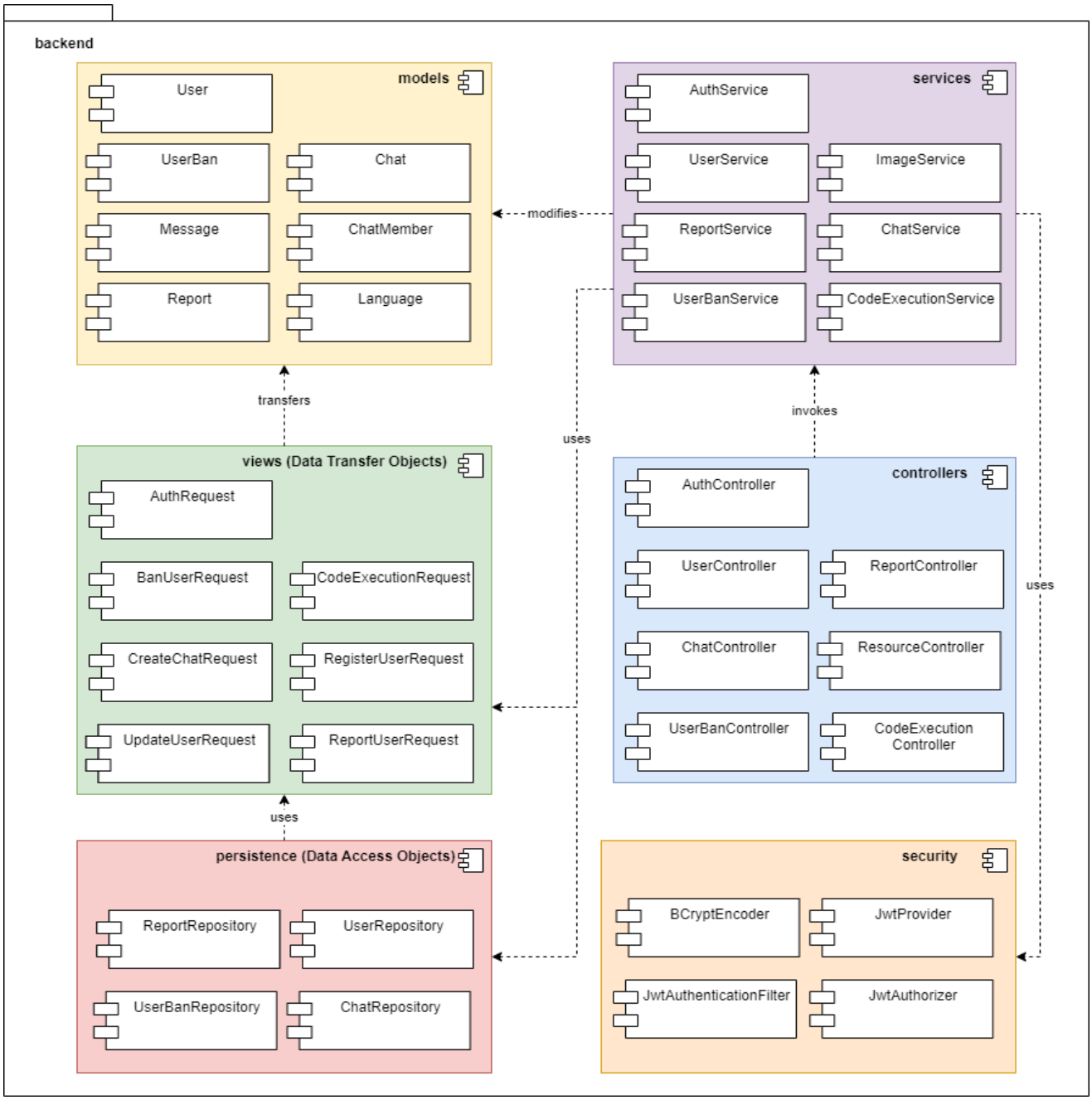

Figura 4 - Diagrama arhitecturii backend

4.5 Aplicația desktop

Implementarea acestei componente va fi realizată cu ajutorul framework-ului Qt în limbajul de programare C++.

Ea va fi compusă din 4, mari, componente:

1. Model - Componenta dată menține starea curentă a aplicației, precum datele legate de sesiune, de profil, și lista de contacte a utilizatorului curent. Aceasta e manipulată de Controller, iar în urma acestor schimbări - actualizează View-ul, astfel încât utilizatorul să poată vizualiza schimbările interne, de interes, ale aplicației;
2. View - Componenta vizibilă utilizatorului. Ea va consta din mai multe pagini între care utilizatorul va putea naviga, și cu care va putea interacționa pentru a folosi aplicația. Componenta

dată va fi actualizată de către Model, iar navigarea va fi facilitată, în anumite cazuri, de către Controller;

3. Controller - Această componentă răspunde de dirijarea aplicației. Scopul principal este de a manipula Modelul, ca răspuns la acțiunile utilizatorului sau a mesajelor de la server. În plus, va facilita navigarea între paginile din View, acolo unde e cazul. E singura componentă care are acces la Servicii, pe care le apelează, așteptând un răspuns, în mod asincron, de la server;
4. Services - Componentă care încapsulează interacțiunea cu serverul web și comunicarea între aplicații necesară pentru funcționalitățile de chat. Ea va fi utilizată de către Controller, căruia îi va furniza un răspuns în mod asincron.

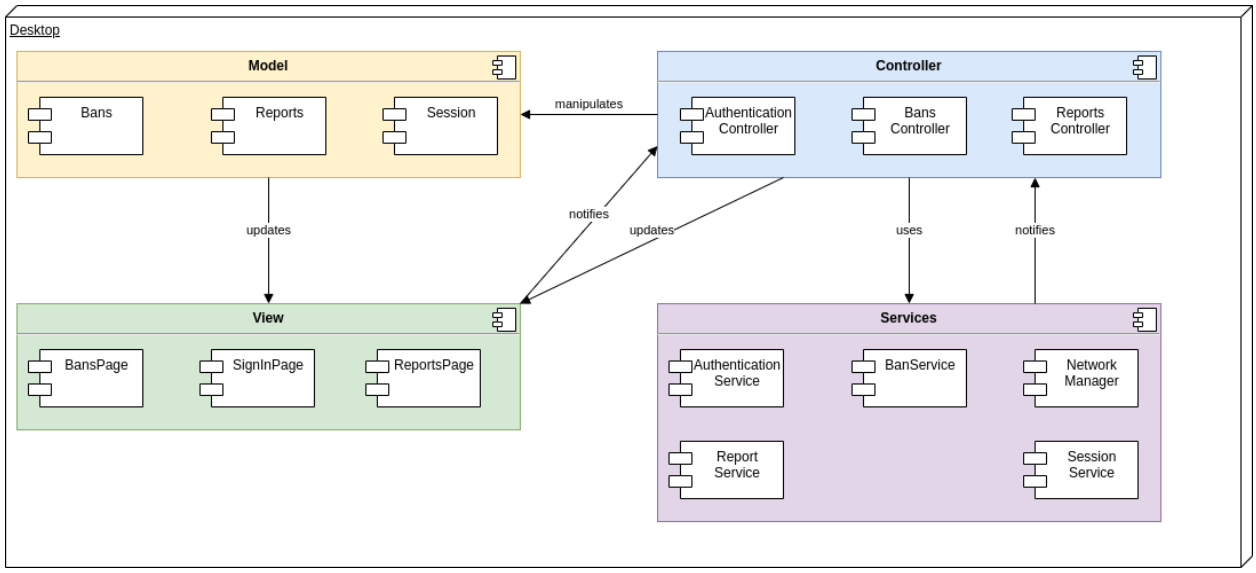

Figura 5 - Diagrama arhitecturii Desktop