**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**PROIECTARE SOFTWARE**

**DOCUMENTATIE**

**TEMA 1**

**An academic:**

**2022-2023**

**NUME STUDENT: VASILACHE ANDREI-FLORIN**

**GRUPA: 30223**

**CUPRINS**

1. Obiectivul temei ......................................................................................................................... 3

2. Instrumente utilizare .................................................................................................................. 3

3. Justificarea limbajului ales ......................................................................................................... 3

4. Descrierea diagramelor UML .................................................................................................... 4

5. Descrierea aplciatiei ................................................................................................................... 7

6. Concluzii .................................................................................................................................. 10

7. Bibliografie .............................................................................................................................. 10

1. **Obiectivul temei**

Obiectivul principal al acestui proiect este acela de a realiza o aplicatie web pentru o agentie turistica. Aplicatia client va avea 3 tipuri de utilizatori: client, angajat si administrator.

Utilizatorii de tip clientpot efectua următoarele operații fără autentificare:

❖Vizualizarea listei pachetelor oferite de agenție sortată după destinație și perioadă;

❖Filtrarea listei pachetelor după destinație, perioadă, preț.

Utilizatorii de tip angajatpot efectua următoarele operații după autentificare:

❖Toate operațiile permise utilizatorilor de tip client;

❖Operații CRUD în ceea ce privește persistența pachetelor oferite de agenție;

❖Operații CRUD pentru informațiile legate de utilizatorii de tip client.

Utilizatorii de tip administratorpot efectua următoarele operații după autentificare:

❖Toate operațiile permise utilizatorilor de tip client;

❖Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;

❖Vizualizarea listei utilizatorilor care necesită autentificare.

1. **Instrumente utilizate**

Pentru proiect am codat in limbajul C# cu framework-ul .NET, iar ca IDE am folosit Visual Studio, pentru o mai buna manageriere a bazei de date (SQL Manager Studio), iar pentru API-uri am folosit variantele REST.

1. **Justificarea limbajului ales**

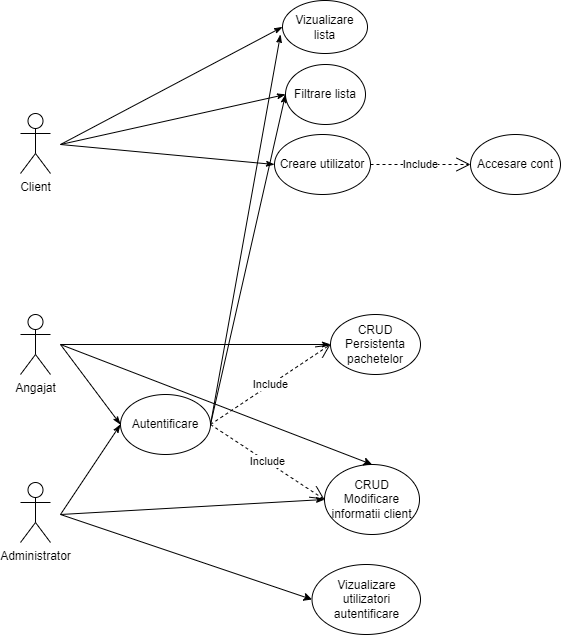
Am ales sa fac proiectul in .NET, deoarece este fiabil si scalabil, codul este interpretabil si reutilizabil, are un sistem de caching simplu. De asemenea, lansarea proiectului este flexibil din mai multe puncte de vedere: IIS Express, WSL, proiectul sursa, etc.

Un avantaj considerabil este acela de a fi “open source”, adica orice utilizator poate contribui la crearea unor noi tool-uri si framework-uri.

Din nefericire, .NET vine si cu anumite dezavantaje:

* Entity Framework nu are o asa buna flexibilitate si nu suporta toate tipurile de date de baza, iar suportul obiect-relatie este limitat;
* Chiar daca este “open source”, tot Microsoft are ultimul cavand in luarea deciziilor;
* Scursuri de memorie.

1. **Descrierea diagramelor UML:**
2. Use Case Diagram

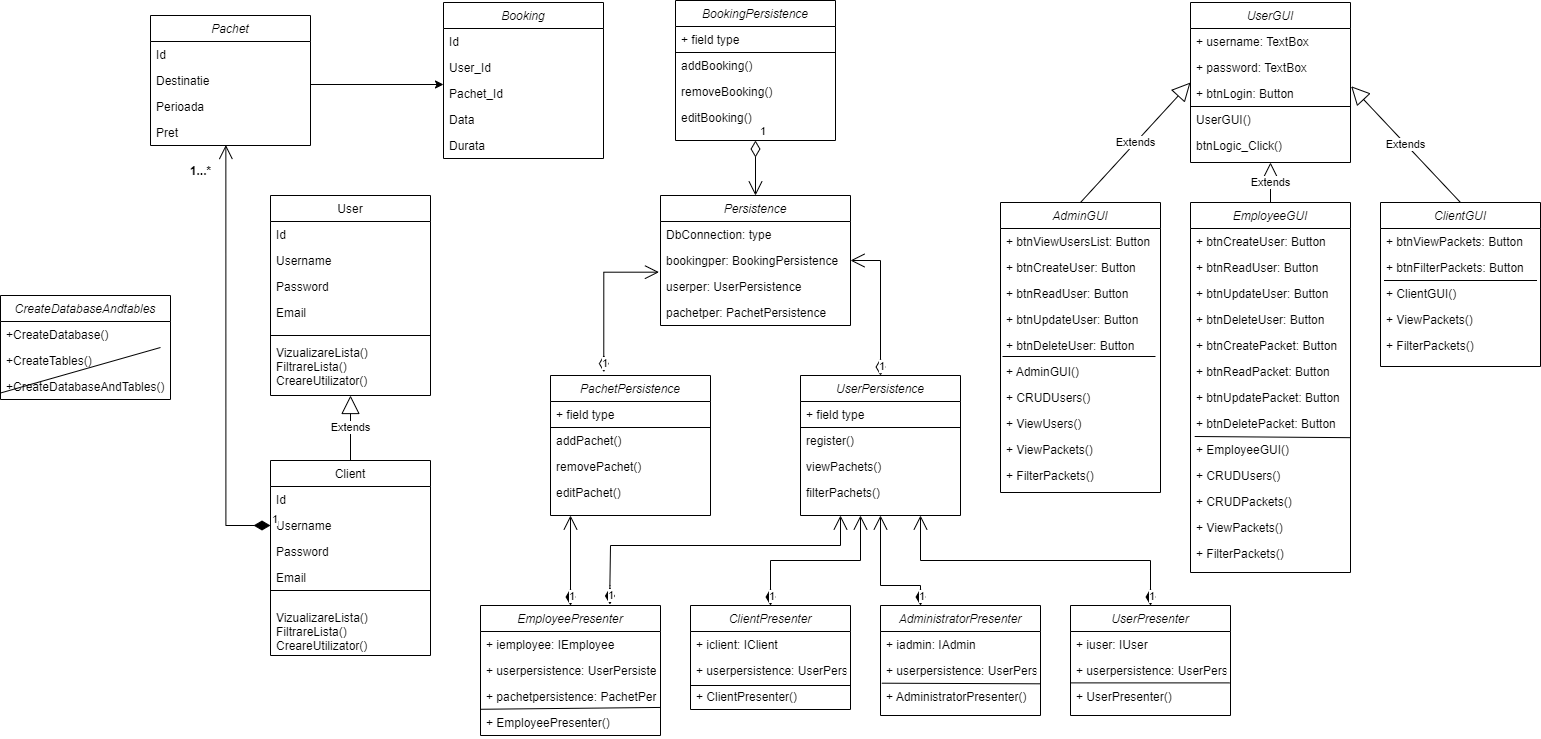
****

Aceasta diagrama UML este alcatuita din:

1. Actori: interactioneaza cu sitemul. Acestia sunt de 3 tipuri:

* Client: poate sa vizualizeze si sa filtreze lista de pachete turistice, si de asemenea poate crea un nou cont de utilizator, pe care il poate accesa;
* Angajat: poate face toate functionalitatile clientului, dar si operatiile CRUD (Create, Read, Update, Delete) pentru acele pachete;
* Administrator: poate, de asemenea, face toate functionalitatile clientului, dar spre deosebire de angajat, acesta poate vizualiza lista de utilizatori care necesita autentificare.

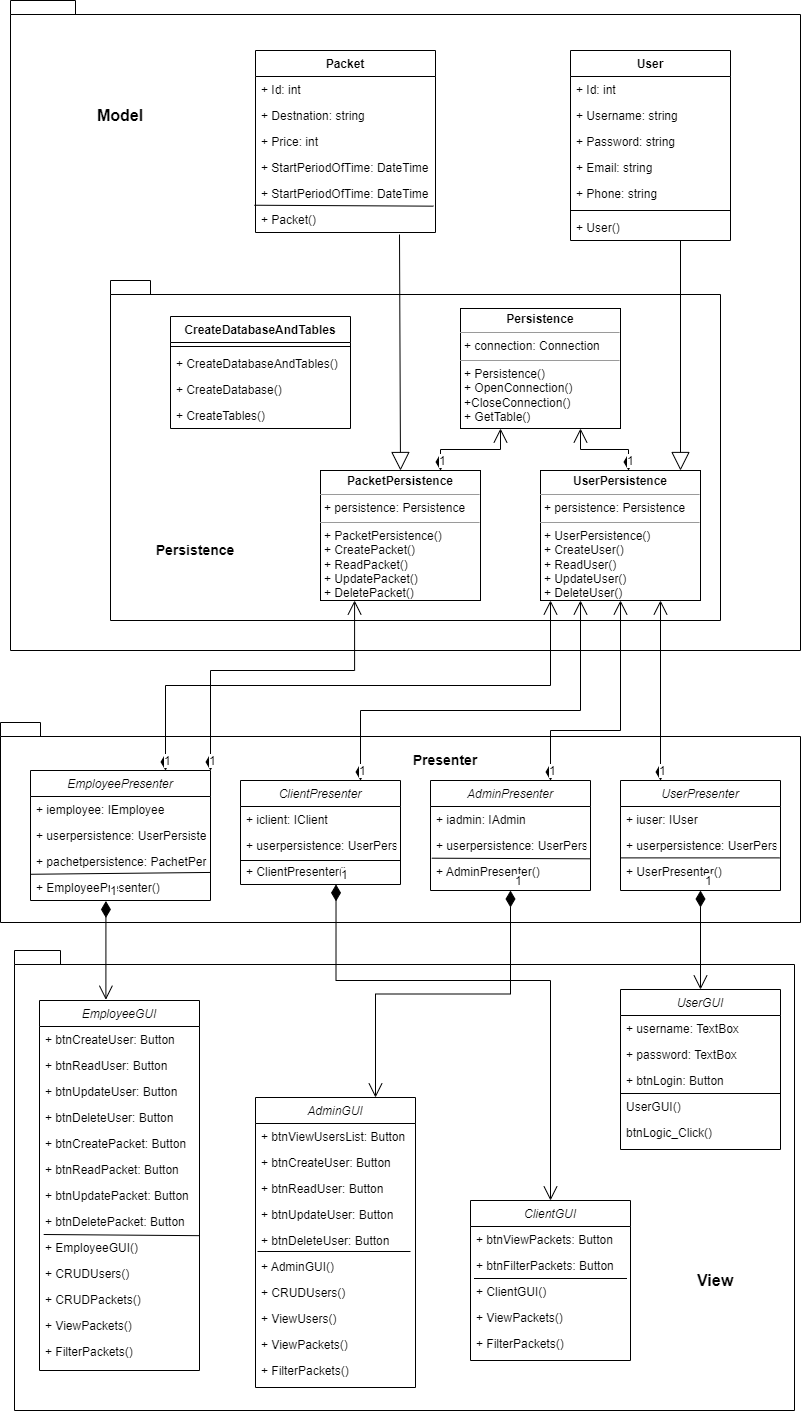
1. Sistem: o serie consecutiva de actiuni si interactiuni dintre actori si sistem. Acesta poate sa fie cunoscut drept scenariu. Aceste actiuni constituie in operatiile CRUD, cat si filtrarea si vizualizarea pachetelor de date.
2. Scop: rezultatul tuturor use case-urilor. Un scenariu corect ar trebui sa descrie activitatile si variantele folosite pentru a atige acel scop. De exemplu: vizualizarea clientilor si pachetelor turistice.
3. ERD Diagram



O diagramă de relație între entități (ER) este un tip de diagramă care ilustrează modul în care „entitățile” precum oamenii, obiectele sau conceptele se relaționează între ele în cadrul unui sistem. Diagramele ER sunt cel mai adesea folosite pentru a proiecta sau a depana baze de date relaționale în domeniile ingineriei software, sistemelor de informații de afaceri, educație și cercetare. Cunoscuți și ca ERD-uri sau modele ER, ele folosesc un set definit de simboluri, cum ar fi dreptunghiuri, romburi, ovale și linii de legătură pentru a descrie interconexiunea entităților, relațiilor și atributele acestora. Ele oglindesc structura gramaticală, cu entități ca substantive și relații ca verbe.

Diagramele ER sunt legate de diagramele de structură a datelor (DSD), care se concentrează pe relațiile elementelor din cadrul entităților în loc de relațiile dintre entitățile înseși. Diagramele ER sunt adesea folosite împreună cu diagramele fluxului de date (DFD), care trasează fluxul de informații pentru procese sau sisteme.

1. MVP Diagram

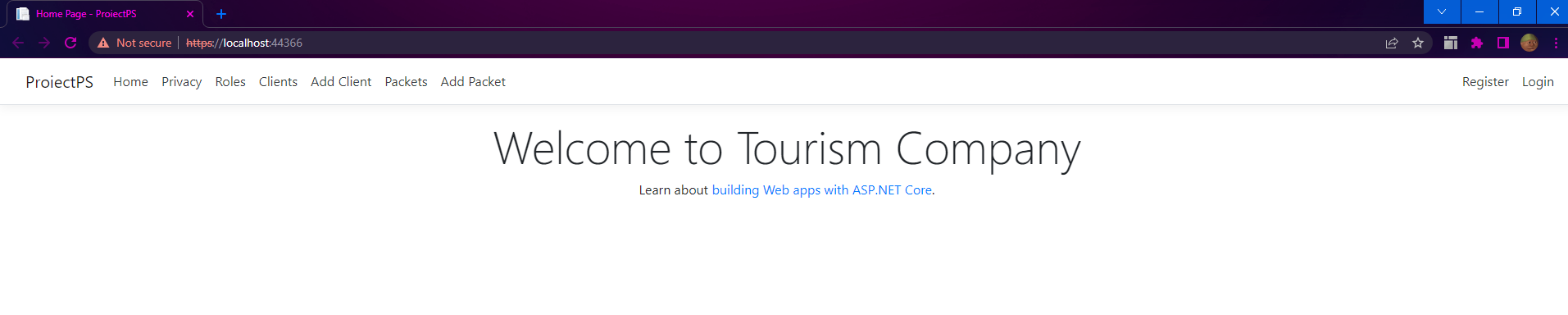


După cum am văzut, relația Model-View este una indirectă bazată pe modelul de proiectare Observer. Modelul poate notifica vizualizarea că au sosit date noi, iar vizualizarea își poate actualiza datele din modelul la care este abonat. Bower și McGlashan au pus la îndoială natura acestei legături indirecte și au sugerat că Modelul poate obține acces direct la interfața cu utilizatorul.

Ideea de bază a fost „răsucirea MVC” într-un mod în care vizualizarea absoarbe funcționalitatea Controllerului și se adaugă noua clasă (Prezentatorul). Prezentatorul poate accesa Vizualizarea și Modelul direct, iar relația Model-Vizualizare poate exista în continuare acolo unde este relevant. În general, vizualizarea afișează date, iar prezentatorul poate actualiza direct modelul și vizualizarea.

In proiectul nostru avem cele 3 parti:

* + Model: unde avem modelele pentru User si Packet, si de asemenea persistenta acestora cu metodele aferente, cat si pentru crearea bazei de date;
  + Presenter: este intermediarul dintre View si Model, unde avem metodele in care se face trecerea de date dintre acestea. Avem Presenter pentru toate tipurile de utilizator: User, Client, Employee, si Administrator.
  + View: este de obicei implementata de catre o Activitate, care face referire catre un Presenter. Singurul lucru pe care il face este sa apeleze metodele din Presenter de fiecare data cand se apeleaza o actiune din interfata. Ca si pentru Presenter, avem 4 tipuri de GUI: User, Client, Employee, si Administrator.



1. **Descrierea aplicatiei:**

In acest proiect am realizat o aplicatie web a unei agentii turistice. Am folosit ca tip de arhitectura MVP (Model-View-Presenter).  
In pachetul Models avem 2 tipuri de modele: user si packet. Pentru User avem atributele: Id, Username, Password, Email, Phone si Role, acesta din urma pentru a putea face distinctia dintre tipurile de utilizatori: Administrator, Employee sau Client. Username-ul si Password-ul sunt folosite pentru a se putea autentifica pe site, iar celelalte campuri sunt folosite pentru a crea contul. Pentru modelul de Packet am folosit atributele: Id, Destination, Price, StartPeriodOfTime si EndPeriodOfTime. De asemenea, am creat si clase de persistenta pentru User in care am utilizat metode pentru crearea unui utilizator si salvarea lui in baza de date, de actualizare a datelor ale acestuia, de stergerea unui cont din baza de date, de vizualizarea acestora, cat si informatiilor despre ei, dar si daca sunt aprobati sau nu de catre administrator. Cat pentru persistenta pachetelor, am realizat operatii CRUD pentru acestea (Create, Read, Update si Delete), dar si operatii de tipul vizualizare si filtrare a acestora.

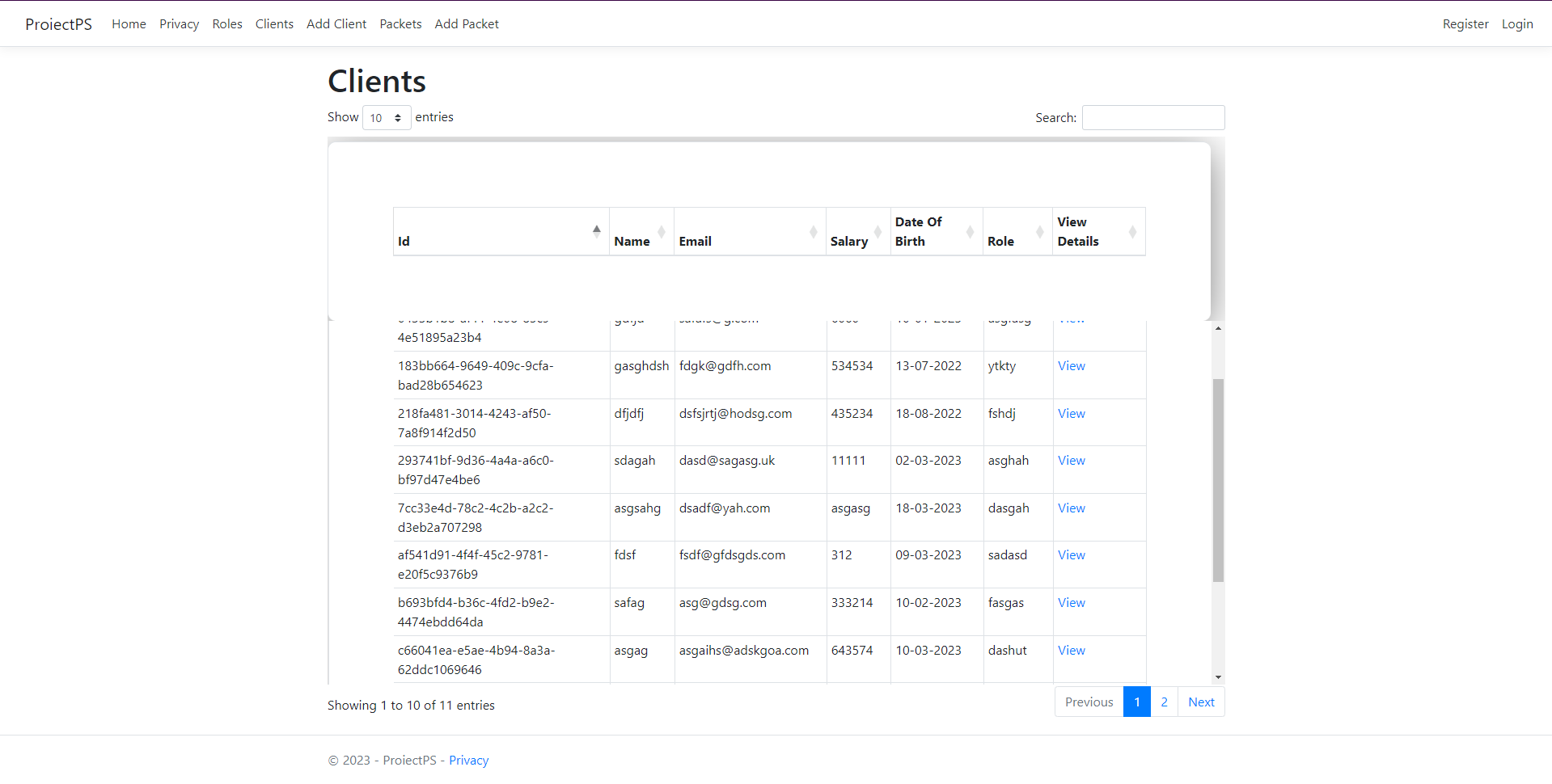
In pachetul Presenter, am realizat toate functiile care au rolul de a transmite datele din Model catre View. Acesta a fost impartit in 3 bucati: AdminPresenter, ClientPresenter si EmployePresenter.

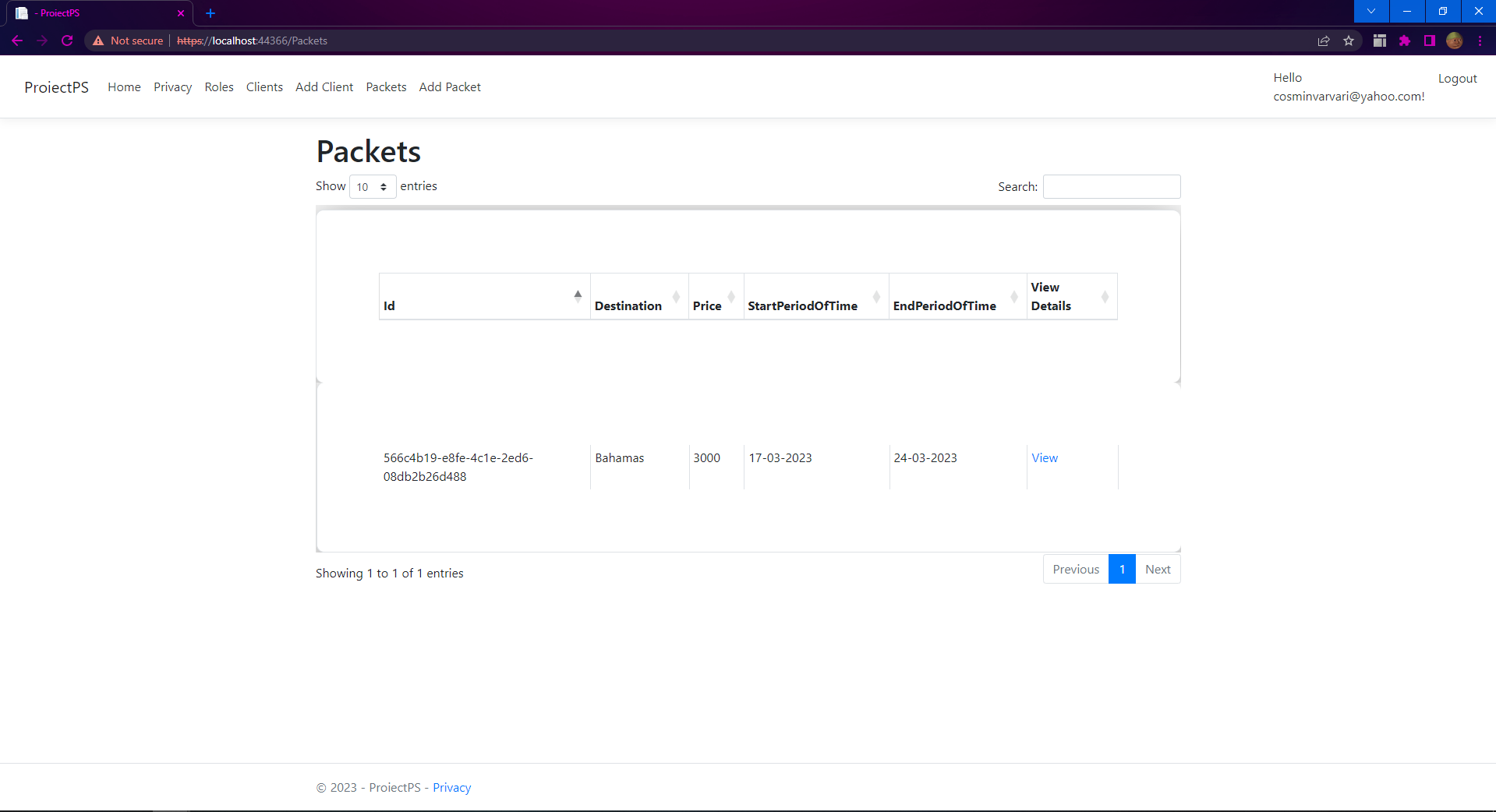
* ClientPresenter: am realizat operatii pentru accesarea tuturor pachetelor si de filtrare a acestora;
* EmployeePresenter: sunt posibile operatiile CRUD pentru pachete, cat si cele pentru utilizatori(clienti);
* AdminPresenter: am folosit operatiile pentru a vizualiza lista de utilizatori care necesita autentificare, cat si operatiile disponibile pentru client.

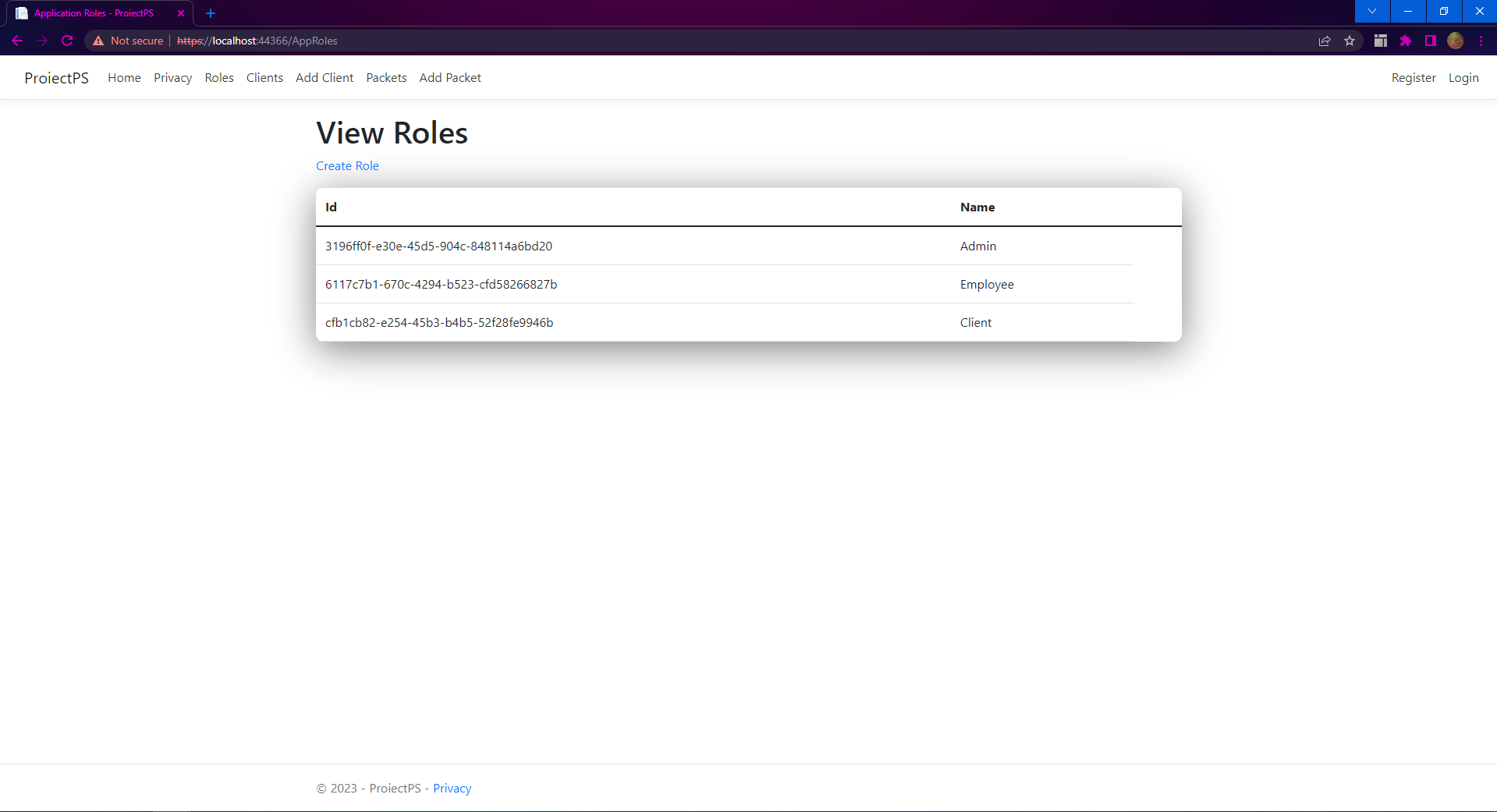
In pachetul View avem functiile care redirectioneaza datele primite catre interfata utilizatorului. Fiecare View pentru Packet si User sunt alcatuite din Add, Index si View. Add genereaza codul html pentru GUI, Index creaza tabelul si populeaza cu datele primite din baza de date in GUI, iar in Index sunt afisate functionalitatile si afisarea butoanelor interactive in Navbar.

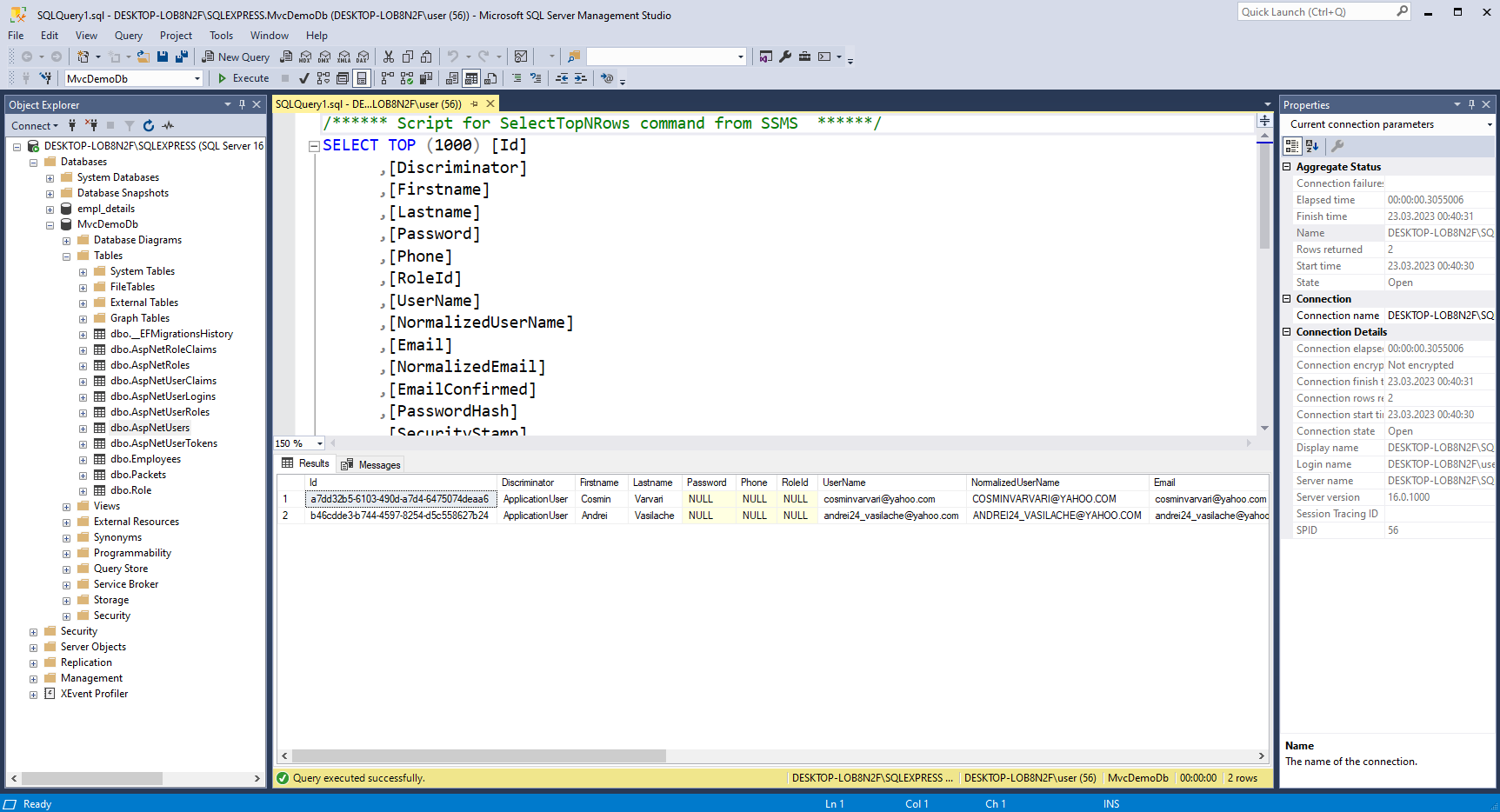
De asemenea, am creat un pachet de testari, denumit “Tests” in care am testat toate functionalitatile claselor construite in Presenter si Model. Totodata, am creat si testari pentru a verifica daca crearea si popularea bazei de date este in parametri optimi.

Pentru partea de front-end, am folosit CSS, HTML si script-uri de Javascript in care am modelat interfata utilizatorilor, pentru inregistrare, crearea contului, afisarea tabelelor cu utilizatori si pachete turistitce, cat si pentru stilizarea butoanelor din bara de navigatie.









1. **Concluzii**

Concluzia principala pe care am dedus-o in urma realizarii acestei teme este faptul ca pentru o mai buna gestionare a unui proiect este nevoie ca prima data sa ne gandim cum l-am putea implementa, ce arhitectura ar fi cea mai potrivita si sa ii cream design-ul care modifica substantial eficienta si din timpul de lucru, deoarece are acces la adresa de spatiu si la resursele procesului. De asemenea, pentru o mai buna gestionare a memoriei calculatorului este indicat sa se foloseasca structuri de date de memorare in liste si cozi in loc de vectori.

In concluzie, acest proiect m-a ajutat sa aprofundez paradigmele Object Oriented Programming, sa inteleg cum functioneaza Http Requests si operatiile CRUD, cat si sa imi consolidez cunostintele in limbajul de programare .NET si, totodata, sa imi folosesc abilitatile gandirii logice aplicate in viata reala.

1. **Bibliografie**

* <https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/>
* <https://www.baeldung.com/mvc-vs-mvp-pattern>
* <https://learn.microsoft.com/en-us/dotnet/>
* <https://stackoverflow.com/>