Технически Университет – София

Факултет по Компютърни Системи и Технологии

Специалност КСИ

**Дисциплина**: Програмни среди

Ася Асенова Узунска

Поток 9 Гр.42 фак. № 121221018

**Курсов Проект 3 курс**

# Съдържание

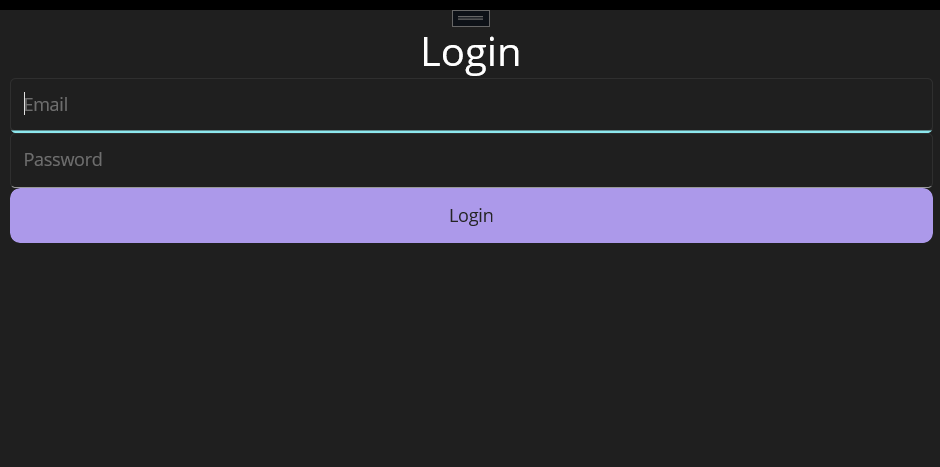
1. Обобщено представяне на проекта
2. Информация за работа с .NET MAUI
3. Трудности и решения
4. Описание на класове и структури
5. Важни части от кода

# Обобщено представяне на задачата

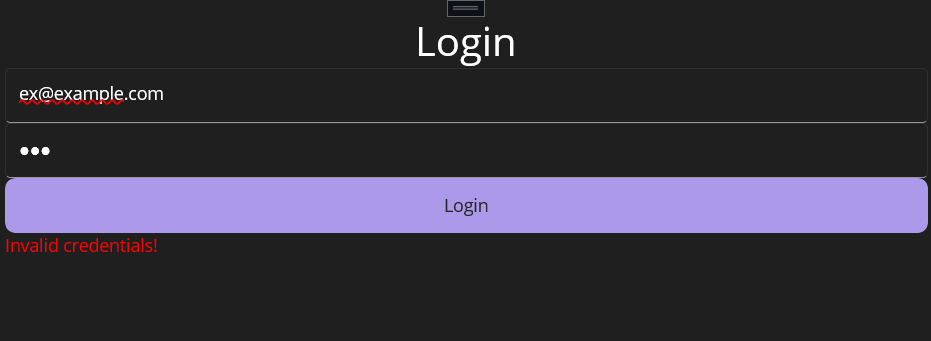
В допълнение на създадената по време на лабораторните упражнения по Програмни среди система за представяне на информацията за потребители от дадена база данни, в проекта са добавени различни възможности за потребители с администраторска роля и такива за потребители без администраторска роля. При логин в системата ролята на потребителя се проверява и спрямо нея се предприемат съответните действия.

Идентифициране и влизане в системата

При стартиране на приложението, пред потребителя се визуализира логин форма, която изисква попълване на имейл и парола за достъп до системата.



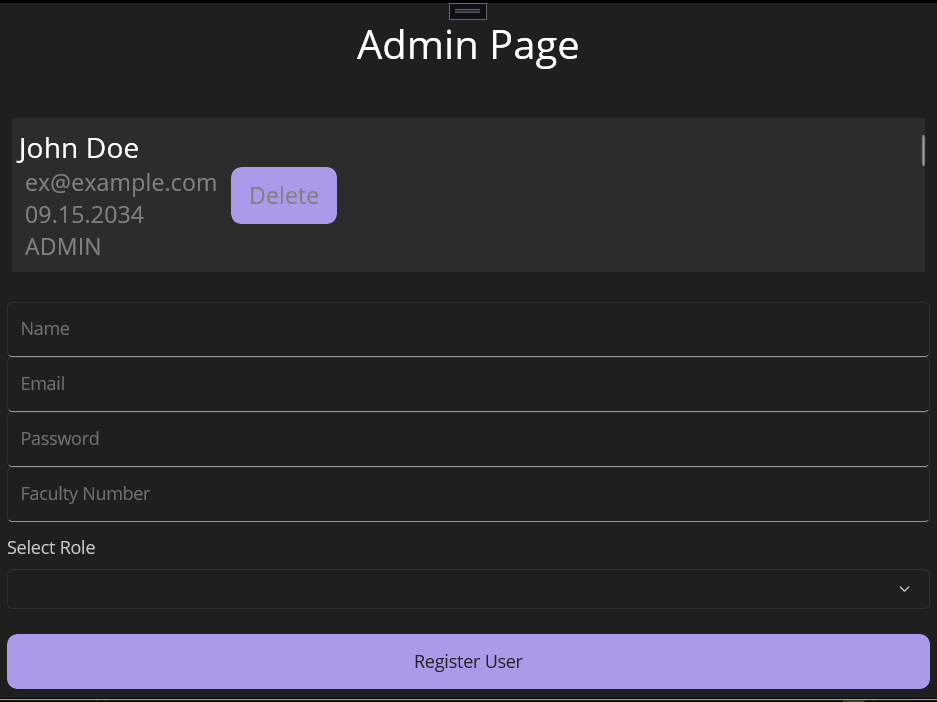
Данните се сравняват със съхранените такива в базата данни. При несъответствие се изобразява съобщение за грешка и не се осъществява достъп до системата.



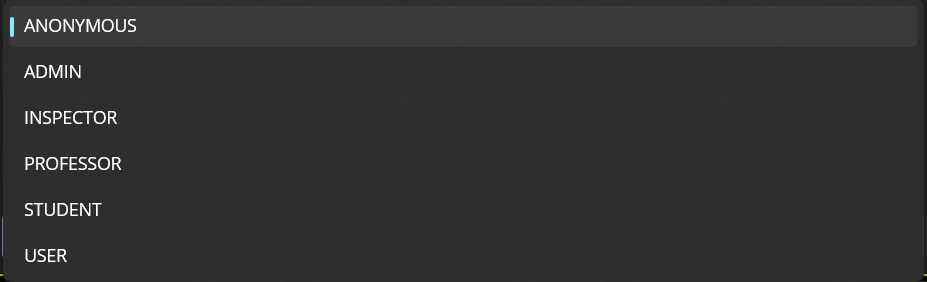
При съвпадение с данните в базата данни потребителят получава съответния достъп до системата, който му осигурява ролята, която притежава.

Потребител с администраторски права

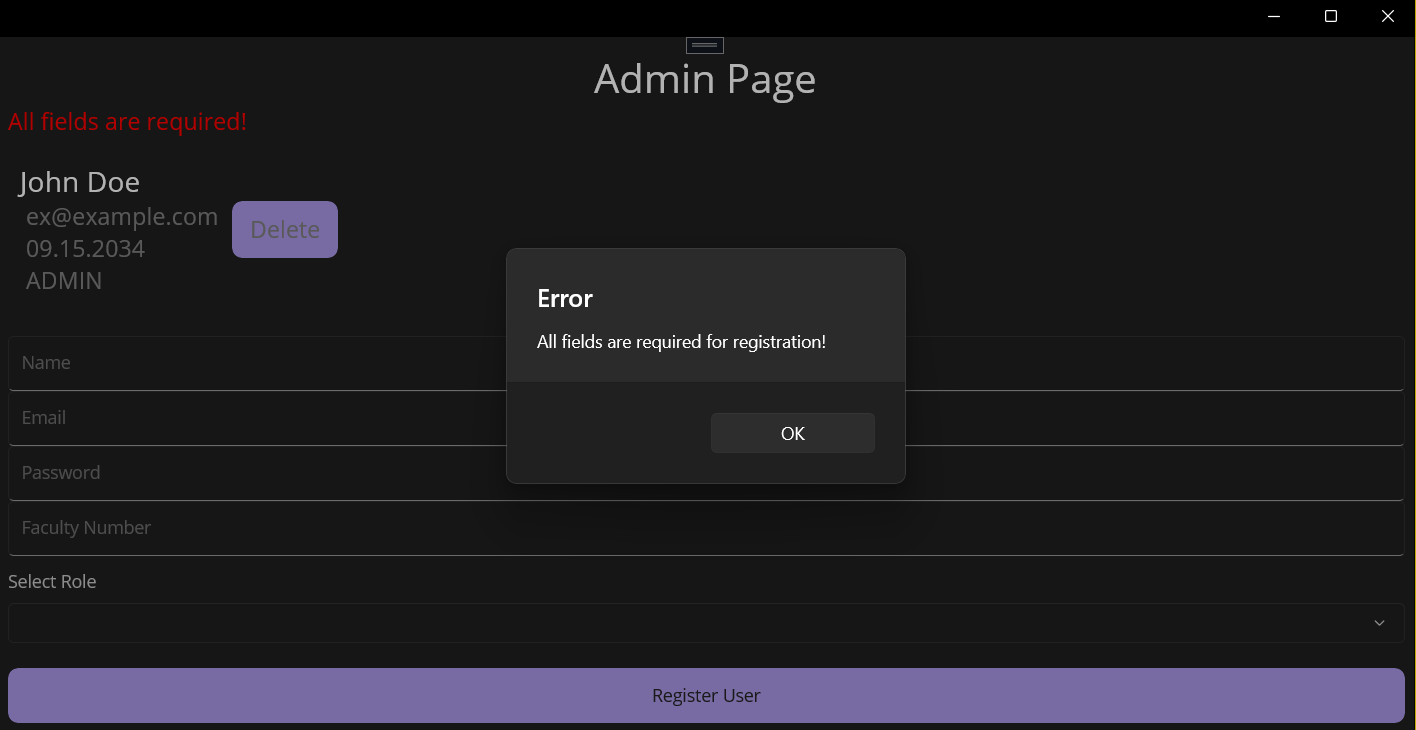
Ако ролята на идентифицирания потребител е ADMIN, то той автоматично бива препращан към нова страница. В страницата се визуализира списък с всички потребители, налични в базата данни. Освен имената, се визуализира и допълнителна информация – имейл и дата на изтичане на достъпа до системата. Страницата дава възможност на администратора да изтрие даден потребител от базата данни. До данните на всеки потребител в списъка се изобразява бутон Delete, който извършва операцията.



Освен изтриване на потребители, администраторът може да добавя и нови такива в базата данни посредством регистрационна форма в долната част на страницата. За регистрацията се попълват имена, имейл, факултетен номер и начална парола. Също може да бъде избрана роля за новия потребител от падащо меню.

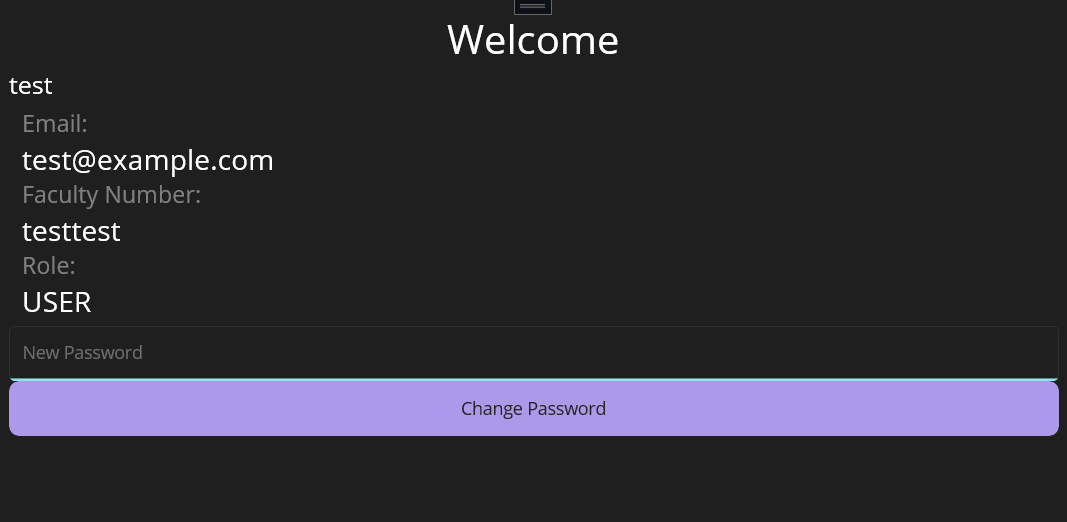


При успешна регистрация на потребител, той веднага става видим в списъкът отгоре и съответно може да бъде изтрит. При празно поле в регистрационната форма или опит за предаване на изцяло празна регистрационна форма се извеждат съобщение за грешка и допълнителен изскачащ прозорец, които да информират потребителя, че регистрацията не може да бъде извършена при липсващи данни.

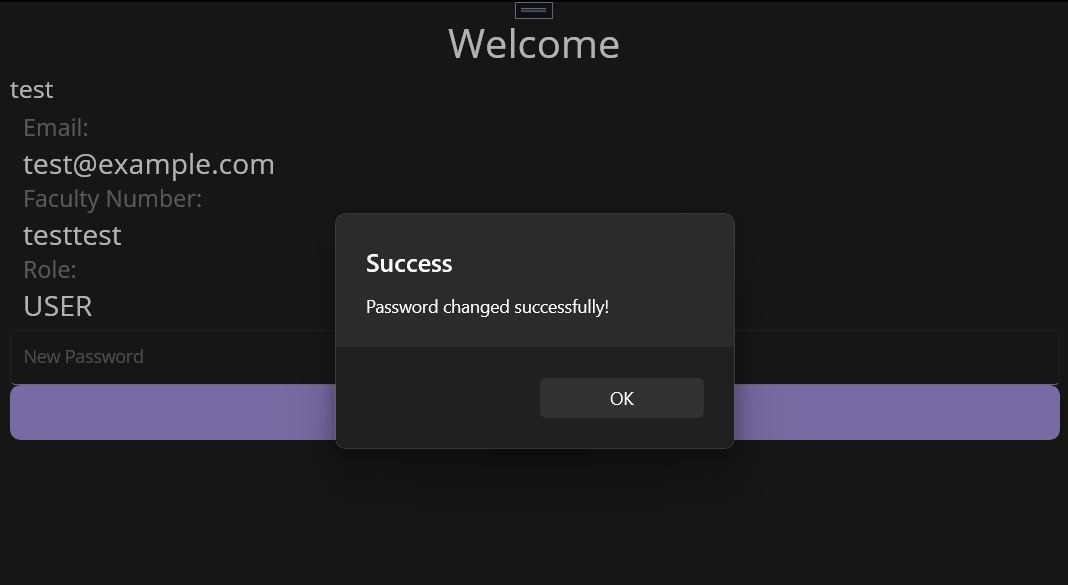


Потребител без администраторски права

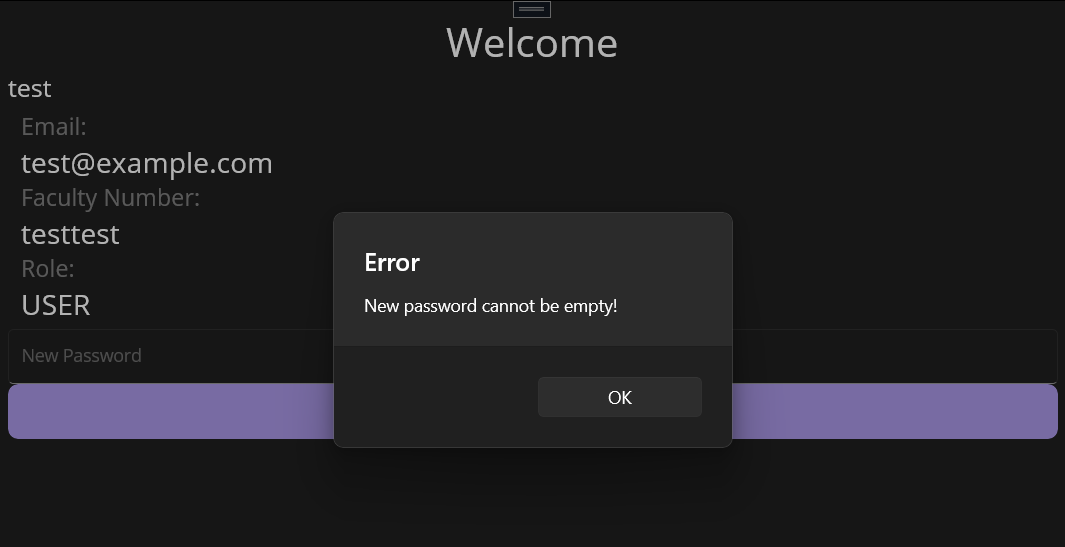
При успешно влизане в системата на потребител без администраторски права, той бива пренасочен към страница с поздравително съобщение, на която освен това се визуализират данните му – имена, имейл, роля и факултетен номер. Потребителят има възможност да промени паролата си посредством полето за смяна на парола под данните.



При успешна промяна на паролата се визуализира съобщение, което потвърждава промените и потребителят вече може да достъпва системата с новата парола.



При опит за подаване на формата за смяна на парола без въведена стойност в полето за нова парола, се извежда предупредително съобщение и операцията не се осъществява.

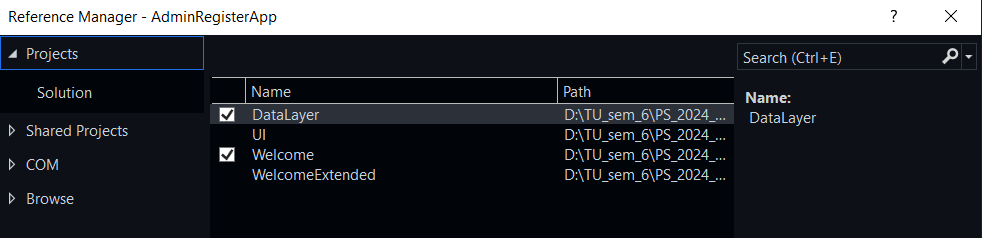


# Информация за работа с .NET MAUI

# Трудности и решения

# Описание на кода

За създаването на проекта са използвани 2 от проектите, създадени по време на лабораторните упражнения по Програмни среди – Welcome и DataLayer, които са добавени като референции към проекта AdminRegisterApp.



Welcome проект

Welcome проекта представлява базовата структура на цялата апликация и съдържа в себе си списък с ролите, които могат да бъдат дадени на потребител на системата във файла UserRolesEnum – към основните роли е добавена и допълнителна такава (USER) с цел да бъде роля по подразбиране за създаване на нови потребители извън създадения интерфейс.

namespace Welcome.Others

{

public enum UserRolesEnum

{

ANONYMOUS,

ADMIN,

INSPECTOR,

PROFESSOR,

STUDENT,

USER

}

}

Основна роля в този проект играе файла User.cs, който представлява обобщен модел на потребителя на дадената система. В него се дефинират частни полета, които да съхраняват необходимата за всеки един акаунт информация – индекс, имена, имейл, роля, парола, дата на изтичане на достъпа. За достъп до частните полета се използват публични getter и setter.

Индексът се използва за идентифициране на потребителя в базата данни, но това ще бъде разгледано на по-късен етап.

namespace Welcome.Model

{

public class User

{

private int \_id;

private string \_names;

private string \_email;

private string \_password;

private UserRolesEnum \_role;

private string \_facNum;

private DateTime \_expires;

public virtual int ID

{

get { return \_id; }

set { \_id = value; }

}

public string Names {

get { return \_names; }

set { \_names = value; }

}

public string Email {

get { return \_email; }

set { \_email = value; }

}

public string Password

{

get { return \_password; }

set{\_password = value;}

}

public UserRolesEnum Role

{

get { return \_role; }

set { \_role = value; }

}

public string FacNum

{

get { return \_facNum; }

set { \_facNum = value; }

}

public DateTime Expires

{

get { return \_expires; }

set { \_expires = value; }

}

}

}

Друг основен файл е UserViewModel.cs класът, който е отговорен за връзката между интерфейсът и модела на данните. Класът имплементира интерфейсът INotifyPropertyChanged, чиято роля е да позволява на потребителския интерфейс да реагира на всяка промяна на данните в модела. В класа е декларирано частно поле от тип User, което реферира към гореспоменатия клас User. Всяко пропърти на UserViewModel класа отговаря на пропърти на основния клас User, като публичните getter и setter имплементират OnPropertyChanged метода, който отговаря за актуализирането на данните в интерфейса при промяна.

namespace Welcome.ViewModel

{

public class UserViewModel : INotifyPropertyChanged

{

private User \_user;

public UserViewModel(User user)

{

\_user = user;

}

public string Names

{

get { return \_user.Names; }

set {

\_user.Names = value;

OnPropertyChanged();

}

}

public string Email

{

get { return \_user.Email; }

set {

\_user.Email = value;

OnPropertyChanged();

}

}

public string Password

{

get { return \_user.Password; }

set {

\_user.Password = value;

OnPropertyChanged();

}

}

public UserRolesEnum Role

{

get { return \_user.Role; }

set {

\_user.Role = value;

OnPropertyChanged();

}

}

public string FacNum

{

get { return \_user.FacNum;}

set {

\_user.FacNum = value;

OnPropertyChanged();

}

}

За правилното имплементиране на интерфейса INotifyPropertyChanged се изисква имплементация на събитието PropertyChanged, което се извиква при всяка промяна на пропърти и индикира за актуализиране стойността на използваните контроли в интерфейса.

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}