Министерство науки и высшего образования Российской Федерации

Казанский национальный исследовательский технический университет – КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ «Колледж информационных технологий»

**ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ**

Методические указания к лабораторной работе №11

Тема: «Разработка подсистемы авторизации и регистрации информационной системы»

Казань 2022

Составитель преподаватель СПО ИКТЗИ Мингалиев Заид Зульфатович

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

КНИТУ-КАИ, 2022

# ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

**ЛАБОРАТОРНАЯ РАБОТА №11**

**ТЕМА: «РАЗРАБОТКА ПОДСИСТЕМЫ АВТОРИЗАЦИИ И РЕГИСТРАЦИИ ИНФОРМАЦИОННОЙ СИСТЕМЫ»**

# ЦЕЛЬ РАБОТЫ

Изучение способов реализации подсистемы авторизации и регистрации пользователей информационной системы (ИС) с использованием базы данных.

# ХОД РАБОТЫ

# 1. Понятие регистрации и авторизации

Регистрация является неотъемлемой частью большинства цифровых продуктов. Мы постоянно заводим новые аккаунты, придумываем сложные пароли.

**Регистрация** пользователя — процедура, в результате которой пользователь становится пользователем конкретной ИС с определёнными правами доступа к определённому ограниченному объёму функций ИС.

**Авторизация** — предоставление определённому лицу или группе лиц прав на выполнение определённых действий; а также процесс проверки и подтверждения данных прав при попытке выполнения этих действий. Часто можно услышать выражение, что какой-то человек «авторизован» для выполнения данной операции — это значит, что он имеет на неё право. В информационных системах посредством авторизации устанавливаются права доступа к информационным ресурсам и системам обработки данных.

Авторизацию не следует путать с **аутентификацией** — процедурой проверки легальности пользователя или данных, например, проверки соответствия введённого пользователем пароля к учётной записи паролю в базе данных.

# 2. Установка и настройка Entity Framework

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

* **Database first**: Entity Framework создает набор классов, которые отражают модель конкретной базы данных;
* **Model first**: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере;
* **Code first**: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

Чтобы непосредственно начать работать с Entity Framework, добавим новый класс, который будет описывать данные. Пусть наше приложение будет посвящено работе с пользователями. Поэтому добавим в проект новый класс User:

**Листинг 11.1**

|  |  |
| --- | --- |
| 1 | public class User |
| 2 | { |
| 3 | public int Id { get; set; } |
| 4 | public string Login { get; set; } |
| 5 | public string Password { get; set; } |
| 6 | public string Email { get; set; } |
| 7 | public string Role { get; set; } |
| 8 | public User() |
| 9 | { } |
| 10 | public User(string Login, string Password, string Email, string Role) |
| 11 | { |
| 12 | this.Login = Login; |
| 13 | this.Password = Password; |
| 14 | this.Role = Role; |
| 15 | this.Email = Email; |
| 16 | } |
| 17 | } |

Это обычный класс, который содержит некоторое количество автосвойств. Каждое свойство будет сопоставляться с отдельным столбцом в таблице из БД.

Надо отметить, что Entity Framework (EF) при работе с Code First требует определения ключа элемента для создания первичного ключа в таблице в БД. По умолчанию при генерации БД EF в качестве первичных ключей будет рассматривать свойства с именами Id или [Имя\_класса]Id (то есть UserId).

Теперь для взаимодействия с БД нам нужен контекст данных. Это своего рода посредник между БД и классами, описывающими данные. Но, у нас по умолчанию еще не добавлена библиотека для EF. Чтобы ее добавить, нажмем на проект правой кнопкой мыши и выберем в контекстном меню «Управление пакетами NuGet...»:

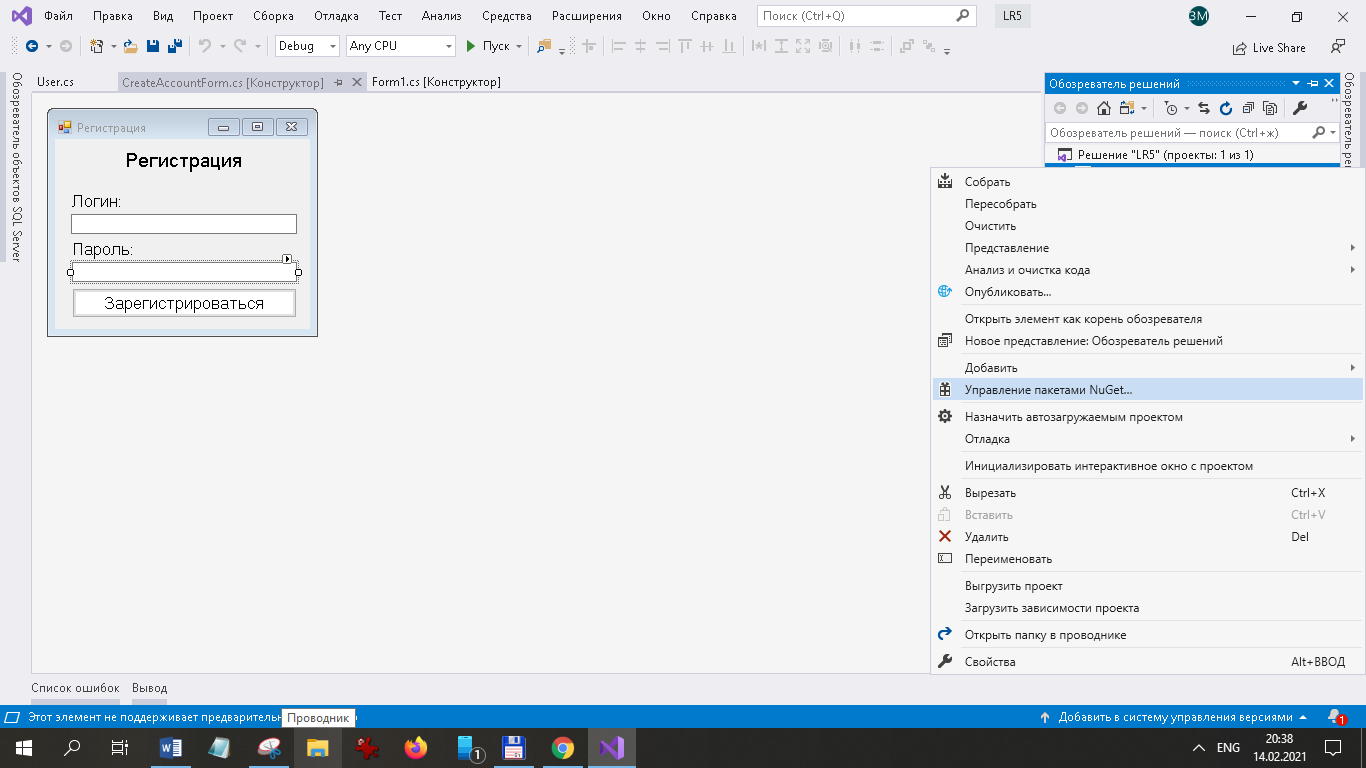


Рисунок 11.1

Затем в появившемся окне управления NuGet-пакетами в окне поиска введем слово «Entity» и выберем пакет Entity Framework для дальнейшей установки:

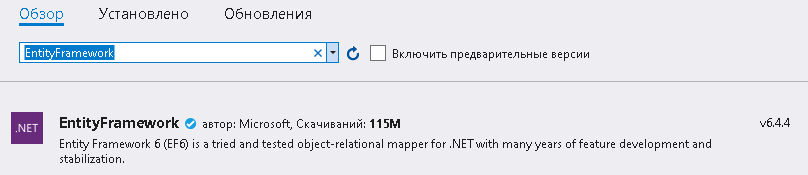


Рисунок 11.2

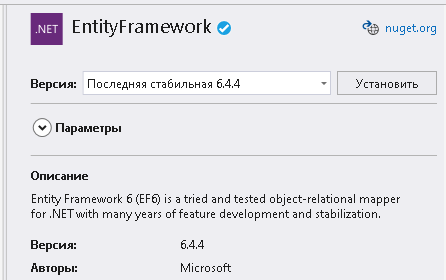


Рисунок 11.3

После успешной установки пакета добавим в проект новый класс UserContext:

**Листинг 11.2**

|  |  |
| --- | --- |
| 1 | public class UserContext : DbContext |
| 2 | { |
| 3 | public UserContext() : base("DbConnection") { } |
| 4 | public DbSet<User> Users { get; set; } |
| 5 | } |

Основу функциональности Entity Framework составляют классы, находящиеся в пространстве имен System.Data.Entity. Среди всего набора классов этого пространства имен следует выделить следующие:

1. DbContext: определяет контекст данных, используемый для взаимодействия с базой данных.
2. DbModelBuilder: сопоставляет классы на языке C# с сущностями в базе данных.
3. DbSet/DbSet<TEntity>: представляет набор сущностей, хранящихся в базе данных.

В любом приложении, работающим с БД через Entity Framework, нам нужен будет контекст (класс производный от DbContext) и набор данных DbSet, через который мы сможем взаимодействовать с таблицами из БД. В данном случае таким контекстом является класс UserContext.

В конструкторе этого класса вызывается конструктор базового класса, в который передается строка "DbConnection" - это имя будущей строки подключения к базе данных. В принципе мы можем не использовать конструктор, тогда в этом случае строка подключения носила бы имя самого класса контекста данных.

И также в классе определено одно свойство Users, которое будет хранить набор объектов User. В классе контекста данных набор объектов представляет класс DbSet<T>. Через это свойство будет осуществляться связь с таблицей объектов User в БД.

И теперь нам надо установить подключение к базе данных. Для установки подключения обычно используется файл конфигурации приложения, который называется App.config. После добавления Entity Framework он выглядит примерно следующим образом:

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<configSections>

<!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->

<section name="entityFramework"

type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"

requirePermission="false"/>

</configSections>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8"/>

</startup>

<entityFramework>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"/>

</providers>

</entityFramework>

</configuration>

Содержимое файла в каждом конкретном случае может отличаться. Но в любом случае после добавления EntityFramework в проект в нем будет содержаться элемент configSections. И после закрывающего тега </configSections> добавим следующий элемент:

<connectionStrings>

<add name="DBConnection" connectionString="data source=(localdb)\MSSQLLocalDB;Initial Catalog=userstore;Integrated Security=True;"

providerName="System.Data.SqlClient"/>

</connectionStrings>

Все подключения к источникам данных устанавливаются в секции connectionStrings, а каждое отдельное подключение представляет элемент add. В конструкторе класса контекста UserContext мы передаем в качестве названия подключения строку "DbConnection", поэтому данное название указывается в атрибуте name="DBConnection".

Настройку строки подключения задает атрибут connectionString. В данном случае мы устанавливаем название базы данных, с которой будем взаимодействовать - userstore.

# 3. Добавление и извлечение данных

Опишем разработку метода для регистрации пользователей. Предварительно создадим форму регистрации:

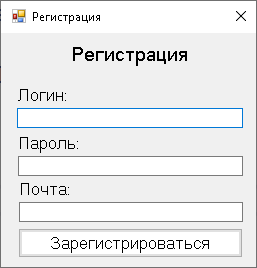


Рисунок 11.4 – Окно регистрации

При регистрации пользователем указываются следующие данные: логин, пароль и электронная почта, которая будет привязана к учетной записи.

Реализация метода обработчика события на нажатие кнопки «Зарегистрироваться» представлен в листинге 11.3.

**Листинг 11.3 – Метод регистрации**

|  |  |
| --- | --- |
| 1 | private void buttonReg\_Click(object sender, EventArgs e) |
| 2 | { |
| 3 | using (UserContext db = new UserContext()) |
| 4 | { |
| 5 | User user = new User(textBoxLog.Text, this.GetHashString(textBoxPass.Text), textBoxEmail.Text, "User"); |
| 6 | db.Users.Add(user); |
| 7 | db.SaveChanges(); |
| 8 | } |
| 9 | } |

Так как класс UserContext через родительский класс DbContext реализует интерфейс IDisposable, то для работы с UserContext с автоматическим закрытием данного объекта мы можем использовать конструкцию using.

В конструкции using создается объект User и добавляется в базу данных. Для их сохранения нам достаточно использовать метод Add: db.Users.Add(user). В строке 7 с помощью метода SaveChanges сохраняются изменения в БД.

Для авторизации пользователей создадим форму следующего вида:

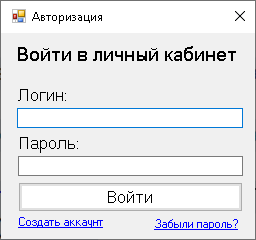


Рисунок 11.5 – Окно авторизации

Реализация метода обработчика события на нажатие кнопки «Войти» представлен в листинге 11.4.

**Листинг 11.4 – Метод авторизации**

|  |  |
| --- | --- |
| 1 | private void ButtonLogIn\_Click(object sender, EventArgs e) |
| 2 | { |
| 3 | using (UserContext db = new UserContext()) |
| 4 | { |
| 5 | foreach (User user in db.Users) |
| 6 | { |
| 7 | if (textBoxLog.Text == user.Login && this.GetHashString(textBoxPass.Text) == user.Password) |
| 8 | { |
| 9 | MessageBox.Show("Вход успешен!"); |
| 10 | UserForm userForm = new UserForm(); |
| 11 | userForm.label1.Text = user.Login; |
| 12 | userForm.Show(); |
| 13 | userForm.form1 = this; |
| 14 | this.Visible = false; |
| 15 | return; |
| 16 | } |
| 17 | } |
| 18 | MessageBox.Show("Логин или пароль указан неверно!"); |
| 19 | } |
| 20 | } |

При авторизации пользователя нам необходимо выполнить аутентификацию. Для этого мы перебираем все элементы из контекста данных: db.Users. У каждого пользователя из контекста данных мы сравниваем на равенство логин и пароль с теми данными, которые были введены в соответствующие поля окна авторизации. Если условие равенства для логина и пароля выполняются одновременно, то происходит открытие учетной записи (формы userForm) пользователя.

# 4. Хеширование паролей

В любой информационной системе необходимо организовать качественное обеспечение безопасности хранения личных данных пользователей. Для этого необходимо предотвратить проникновение злоумышленников в учетные записи пользователей. Проблема заключается в том, что если к БД получат доступ сторонние лица, то все логины/пароли пользователей будут «как на ладони».

В этом случае целесообразно шифровать пароли. То есть, при регистрации нового пользователя, в БД заносится предварительно зашифрованный пароль пользователя, так сказать в закрытом виде.

Шифровать можно по своему собственному алгоритму, например, после каждого символа в пароле дописывать некоторый предварительно сформированный набор символов, или последовательность символов пароля, записанных наоборот, или еще что-нибудь, что вы придумаете сами.

Но такой подход чреват тем, что все-таки имеется возможность обнаружить ключ к расшифровыванию. Это способны достаточно быстро сделать специальные программы. Поэтому, в таких случаях следует отдавать предпочтение хеш-представлению пароля.

Для такой задачи существует алгоритм **MD5**. По данному алгоритму, возвращается значение в виде 32-разрядной шестнадцатеричной строки. Одним словом, имея любой длины набор символов (строку) и применив к ней алгоритм шифрования MD5, мы получим строку в 32 символа, причем в 16-ричном представлении.

В листинге 11.5 представлено шифрование входной строки s алгоритмом MD5.

**Листинг 11.5 – Метод шифрования**

|  |  |
| --- | --- |
| 1 | private string GetHashString(string s) |
| 2 | { |
| 3 | byte[] bytes = Encoding.Unicode.GetBytes(s); |
| 4 | MD5CryptoServiceProvider CSP = new MD5CryptoServiceProvider(); |
| 5 | byte[] byteHash = CSP.ComputeHash(bytes); |
| 6 | string hash = ""; |
| 7 | foreach (byte b in byteHash) |
| 8 | { |
| 9 | hash += string.Format("{0:x2}", b); |
| 10 | } |
| 11 | return hash; |
| 12 | } |

В строке 3 мы переводим строку s, хранящую пароль, в массив байтов. Далее, в строке 4, сейчас объект класса MD5CryptoServiceProvider, реализующий средства шифрования. Средства шифрования реализованы в пространстве имен System.Security.Cryptography. В строке 5 вычисляем хеш-представление пароля и сохраняем его в массив байтов byteHash. Для вычисления хеша используется метод ComputeHash. После этого формируем одну цельную строку из массива (строка 7-10), сохраняя в строку hash преобразованные в строковый тип байты из массива byteHash.

При создании учетной записи пользователя мы будем передавать вместо пароля значение, которое возвращает метод GetHashString, выполняющий шифрование:

**Листинг 11.6**

|  |  |
| --- | --- |
| 1 | User user = new User(textBoxLog.Text, **this.GetHashString(textBoxPass.Text)**, textBoxEmail.Text, "User"); |

В БД будет сохраняться зашифрованный пароль вместо пароля, который будет указан в соответствующем поле ввода.

# 5. Работа с электронной почтой

Для отправки почты в среде интернет используется протокол SMTP (Simple Mail Transfer Protocol). Данный протокол указывает, как почтовые сервера взаимодействуют при передаче электронной почты.

Для работы с протоколом SMTP и отправки электронной почты в .NET предназначен класс SmtpClient из пространства имен System.Net.Mail.

Этот класс определяет ряд свойств, которые позволяют настроить отправку:

* Host: smtp-сервер, с которого производится отправление почты. Например, smtp.yandex.ru
* Port: порт, используемый smp-сервером. Если не указан, то по умолчанию используется 25 порт.
* Credentials: аутентификационные данные отправителя
* EnableSsl: указывает, будет ли использоваться протокол SSL при отправке

Еще одним ключевым классом, который используется при отправке, является MailMessage. Данный класс представляет собой отправляемое сообщение. Среди его свойств можно выделить следующие:

* Attachments: содержит все прикрепления к письму
* Body: непосредственно текст письма
* From: адрес отправителя. Представляет объект MailAddress
* To: адрес получателя. Также представляет объект MailAddress
* Subject: определяет тему письма
* IsBodyHtml: указывает, представляет ли письмо содержимое с кодом html

Используем эти классы и выполним отправку письма:

**Листинг 11.7 – Метод отправки восстановленного пароля**

|  |  |
| --- | --- |
| 1 | private void buttonSendPassword\_Click(object sender, EventArgs e) |
| 2 | { |
| 3 | MailAddress from = new MailAddress("zaid-mingaliev@mail.ru", "Zaid"); |
| 4 | MailAddress to = new MailAddress(textBoxEmail.Text); |
| 5 | MailMessage m = new MailMessage(from, to); |
| 6 | m.Subject = "Тест"; |
| 7 | using (UserContext db = new UserContext()) |
| 8 | { |
| 9 | foreach (User user in db.Users) |
| 10 | { |
| 11 | if (textBoxEmail.Text == user.Email) |
| 12 | { |
| 13 | m.Body = "<h1>Пароль: " + user.Password + "</h1>"; |
| 14 | } |
| 15 | } |
| 16 | } |
| 17 | m.IsBodyHtml = true; |
| 18 | SmtpClient smtp = new SmtpClient("smtp.mail.ru", 587); |
| 19 | smtp.Credentials = new NetworkCredential("zaid-mingaliev@mail.ru", "123"); |
| 20 | smtp.EnableSsl = true; |
| 21 | smtp.Send(m); |
| 22 | } |

Объект from – это отправитель – в конструкторе данного объекта устанавливаем адрес и отображаемое в письме имя.

Объект to – это получатель письма – в конструктор объекта-получателя передаем его адрес электронной почты.

Объект m – это объект сообщения, в конструктор которого передается объект отправитель и получатель. Subject – это свойство темы письма. В свойство body

В цикле foreach (строка 9-15) производится поиск в БД через контекст данных пользователя с адресом электронной почты, соответствующим адресу, введенному в форме.

Объект smtp хранит адрес (имя или IP-адрес хоста) smtp-сервера и порт, используемый на хосте для отправки письма.

В свойстве Credentials хранится логин и пароль электронной почты, являющейся отправителем электронного письма.

Для отправки применяется метод Send(), в который передается объект MailMessage.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать приложение, имеющее следующий функционал:

1. Регистрация пользователя согласно атрибутам сущности «Пользователь», определенным по вариантам. Сохранение данных о пользователе производить в базе данных.
2. Авторизация пользователя с загрузкой формы, где указывается информация о пользователе. Загрузку данных, необходимых для аутентификации, производить из базы данных.
3. Восстановление доступа к учетной записи пользователя путем отправки в электронную почту кода доступа.

Пароли, сохраняемые в базе данных, должны хешироваться.