**«Клиент — сервер»** (англ. *client–server*) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине.

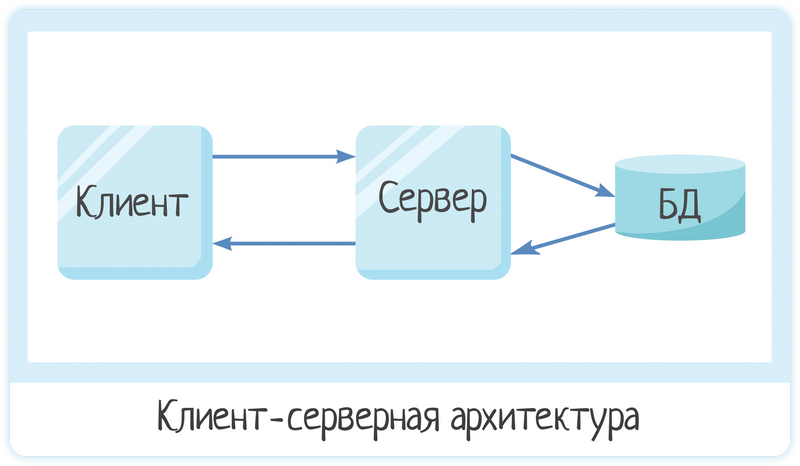


Рис 1. Клиент- серверная архитектура

Знакомая картинка? А вы ведь постоянно сталкиваетесь с этой архитектурой — когда покупаете билет в кино онлайн, бронируете путевку на море или записываетесь к врачу.

На клиент-серверной архитектуре построены все сайты и интернет-сервисы. Также ее используют десктоп-программы, которые передают данные по интернету. Поэтому ИТ-специалисту нужно понимать, что это такое и как работает.

Приложениями и сайтами одновременно могут пользоваться сотни и даже миллионы человек. Все они обращаются к одному компьютеру, который должен уметь обрабатывать запросы и присылать ответы. Такой подход называется **клиент-серверной архитектурой**. Она описывает, как происходит работа с пользователями, где хранятся данные и как обеспечивается их защита.

В клиент-серверной архитектуре используется три компонента:

* **Клиент** — программа, которую мы используем в интернете. Чаще всего это браузер, но может быть и другая отдельная программа
* **Сервер** — компьютер, на котором хранится сайт или приложение. Когда мы заходим на сайт магазина, мы обращаемся к серверу, на котором находится сайт
* **База данных** — программа, в которой хранятся все данные приложения. Для магазина это будет база клиентов, товаров и заказов

**Особенности клиента**

Клиент — это всегда программа. Ее назначение — дать пользователю удобный способ взаимодействия с сервером.

С точки зрения сервера, выбрать товар — это послать запрос на специальном языке запросов, например SQL. Но для простого пользователя это сложно. Поэтому клиент дает удобный способ взаимодействия, чтобы не писать код своими руками.

В следующих уроках мы познакомимся с работой сайтов и узнаем, что зайти на сайт — это не просто ввести его адрес в браузер. За этим действием скрываются специальные запросы, ответы и их расшифровка. Для пользователя — это лишнее, и клиент в виде браузера делает эту работу за нас.

**Особенности сервера**

Сервер — компьютер, такой же как у нас, только намного мощнее. Основная задача сервера — бесперебойная работа и возможность обрабатывать миллионы запросов от пользователей.

Сервер позволяет не дублировать приложения. Без него для заказа продуктов пришлось бы скачать весь сайт к себе на компьютер, выбрать товары, записать их и отправить на компьютер магазина. Так как сайт находится на сервере, то тысячи человек могут обращаться к одному серверу и получать от него нужную информацию.

Если сервер выполняет функции приложения и базы данных, то такая архитектура называется двухуровневой. Такой подход используют для небольших приложений, где нет большого количества клиентов. Хоть такой способ и проще, но его надежность небольшая. Если сервер взломают, то злоумышленники получат все данные.

Чтобы решить проблему безопасности в клиент-серверной архитектуре, используют базу данных. Она хранится отдельно от сервера. Сервер в этом случае выполняет роль логической машины, которая обрабатывает данные, но не хранит их.

**Особенности базы данных**

В клиент-серверной архитектуре сервер — это не только компьютер, на котором находится приложение или сайт. Еще это база, где хранятся все данные приложения. У клиентов нет прямого доступа к базе данных, так как это нарушило бы их приватность. Например, частность личной информации других пользователей в социальных сетях.

Клиенты запрашивают информацию у сервера. Если сервер считает, что у клиента есть права на получение информации, то он ее предоставляет. Благодаря этому мы не можем пользоваться учетными записями своих друзей в социальных сетях или получать информацию о банковских переводах незнакомых нам людей.

**Плюсы и минусы клиент-серверной архитектуры**

В большинстве случаев минусы клиент-серверной архитектуры связаны с работоспособностью сервера или базы данных. Разработчики умеют решать большинство минусов, но не все так просто. Решение одних минусов приводит к другим, чаще всего к росту стоимости разработки и поддержки.

Рассмотрим плюсы и минусы клиент-серверной архитектуры и начнем с хороших сторон.

**Плюсы**

* Отсутствие дублирования. Весь сайт или приложение хранится на одном компьютере-сервере. Это позволяет использовать его с разных устройств, будь то компьютер или мобильный телефон
* Минимальные требования к пользователю. От него требуется только наличие программы-клиента. Для работы с сайтами достаточно иметь браузер
* Безопасность. Данные хранятся в базе данных и пользователи не могут их просматривать. Это обеспечивает безопасность для персональных данных
* Производительность. Серверы обычно производительнее, чем компьютеры пользователей. Это позволяет обрабатывать тысячи запросов от сотни разных пользователей одновременно.

**Минусы**

* Перегрузка сервера. Популярные порталы могут получать большое количество запросов одновременно. Например, при десятке миллионов запросов в секунду сервер может не выдержать и отключиться. Этим пользуются хакеры при использовании DDoS-атаки.
* Выход из строя сервера или базы данных. Это сделает сервис недоступным для всех пользователей
* Высокая стоимость оборудования. Сервер похож на простой компьютер, но его комплектующие должны быть рассчитаны на бесперебойную работу 24/7. Такая надежность обеспечивается компонентами со специальными функциями, из-за чего стоимость оборудования повышается
* Затраты на поддержку. Обычно недостаточно просто получить сервер и забыть про его существование. Должен быть специалист, который будет обслуживать сервер и быстро реагировать в случае поломок
* Чтобы избавиться от большинства перечисленных минусов, разработчики используют кластеры серверов.

В С# одним из наиболее распространенных протоколов взаимодействия в сети является протокол TCP (Transmission Control Protocol). Этот протокол гарантирует доставку сообщений и широко используется в различных существующих на сегодняшний день программах. Для работы с протоколом TCP в .NET предназначены классы TcpClient и TcpListener. Эти классы строятся поверх класса System.Net.Sockets.Socket. TcpClient и TcpListener упрощают создание клиента и сервера, которые реализуют протокол TCP.

Во время прохождения практики необходимо было создать клиент серверное приложение.

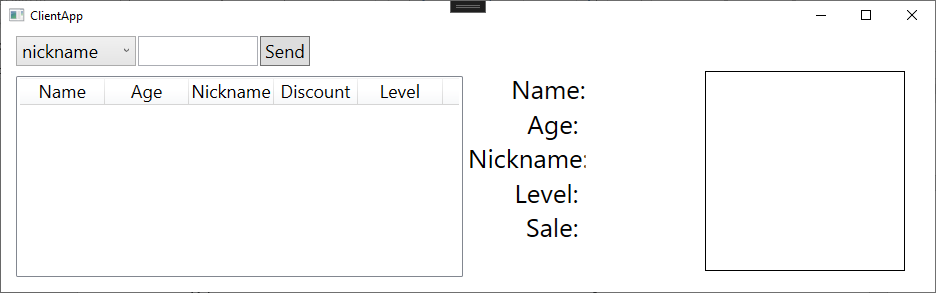


Рис. 2 Главная страница «Клиента»

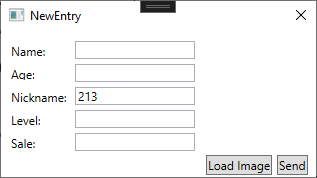


Рис. 3 Форма вставки новой записи

Так как «Сервер» не подразумевает наличие граф интерфейса все необходимые сообщения для отладки выводятся в консоль

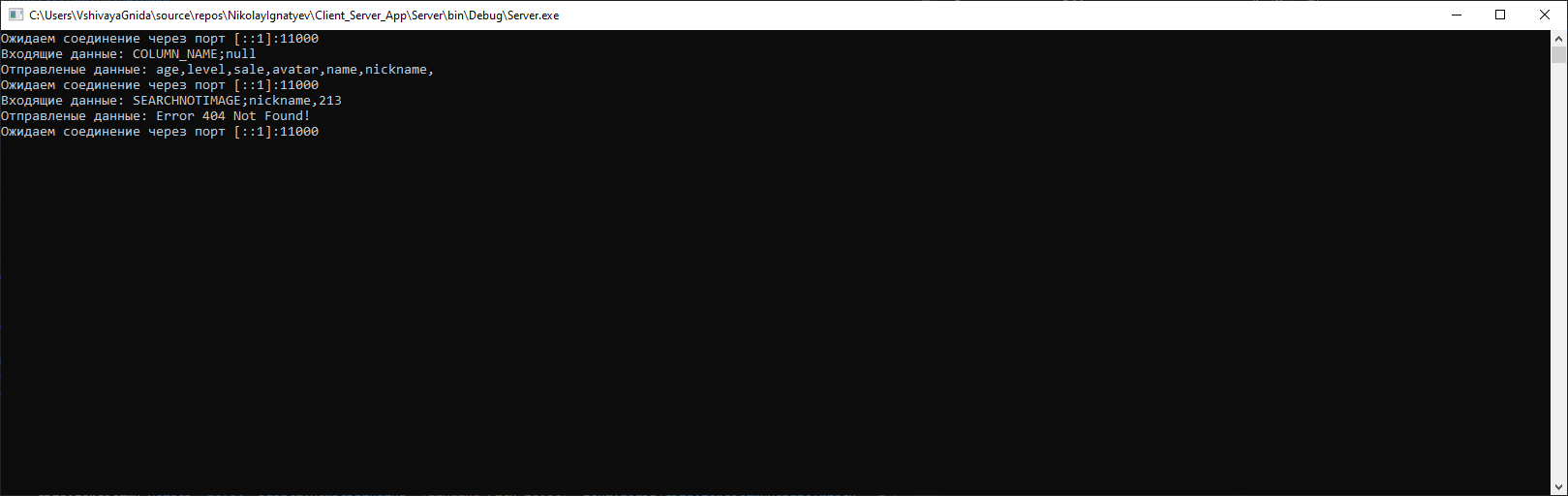


Рис. 4 «Сервер»

На основе пространства имен System.Net; был написан класс отвечающей за отправку и принятие данных на клиенте.

Листинг 1 – Класс Sender.cs

/// <summary>

/// Класс предоставляющий методы для работы с сокетом.

/// </summary>

internal static class Sender

{

// Буфер для входящих данных

static byte[] bytes = new byte[2097152];

static Socket sender;

public static void OpenSocketConnection(int port)

{

// Соединяемся с удаленным устройством

// Устанавливаем удаленную точку для сокета

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);

sender = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// Соединяем сокет с удаленной точкой

sender.Connect(ipEndPoint);

}

public static void CloseSocketConnection()

{

// Освобождаем сокет

sender.Shutdown(SocketShutdown.Both);

sender.Close();

}

public static (byte[] bytes, int bytesRec) SendMessageFromSocket(string message)

{

OpenSocketConnection(11000);

// Кодирование входящих данных в byte[]

byte[] msg = Encoding.UTF8.GetBytes(message);

// Отправляем данные через сокет

int bytesSent = sender.Send(msg);

// Получаем ответ от сервера

int bytesRec = sender.Receive(bytes);

CloseSocketConnection();

return (bytes, bytesRec);

}

public static (byte[] bytes, int bytesRec) SendMessageFromOneSocket(string message)

{

// Кодирование входящих данных в byte[]

byte[] msg = Encoding.UTF8.GetBytes(message);

// Отправляем данные через сокет

int bytesSent = sender.Send(msg);

// Получаем ответ от сервера

int bytesRec = sender.Receive(bytes);

return (bytes, bytesRec);

}

public static (byte[] bytes, int bytesRec) SendMessageFromSocket(byte[] message)

{

OpenSocketConnection(11000);

// Отправляем данные через сокет

int bytesSent = sender.Send(message);

// Получаем ответ от сервера

int bytesRec = sender.Receive(bytes);

CloseSocketConnection();

return (bytes, bytesRec);

}

public static (byte[] bytes, int bytesRec) SendMessageFromOneSocket(byte[] message)

{

// Отправляем данные через сокет

int bytesSent = sender.Send(message);

// Получаем ответ от сервера

int bytesRec = sender.Receive(bytes);

return (bytes, bytesRec);

}

В данном классе был объявлен метод с в 2 перегрузками для того чтобы получить возможность отправлять данные виде строки, которая перед самой отправкой будет закодирована в byte[] и вторя перегрузка в которой данные можно сразу передавать виде массива байтов, такой метод пригодится при отправки фото от клиента.

На форме клиента при ее загрузке происходит запрос на имеющиеся столбцы по которым нужно вести поиск, с исключением столбца содержащего фото.

Листинг 2 – MainWindow.xaml.cs

public partial class MainWindow : Window

{

string reply;

byte[] bytes = new byte[262144];

byte[] avatar = new byte[2097152];

int bytesRec;

public MainWindow()

{

InitializeComponent();

}

private void ButtonData\_Click(object sender, RoutedEventArgs e)

{

if (tbsender.Text != "")

{

try

{

if (cbColumnName.SelectedItem.ToString() == "age" |

cbColumnName.SelectedItem.ToString() == "sale" |

cbColumnName.SelectedItem.ToString() == "level")

{

int.Parse(tbsender.Text);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

try

{

(bytes, bytesRec) = Sender.SendMessageFromSocket("SEARCHNOTIMAGE;" + cbColumnName.SelectedItem.ToString() + "," + tbsender.Text);

reply = Encoding.UTF8.GetString(bytes, 0, bytesRec);

if (reply.Contains("Error 404"))

{

if (MessageBox.Show("Таких данных нет в таблице.\nХотите добавить?", "Нет данных", MessageBoxButton.OKCancel, MessageBoxImage.Question) == MessageBoxResult.OK)

{

WindowInsert windowInsert = new WindowInsert(tbsender.Text, cbColumnName.SelectedItem.ToString());

windowInsert.ShowDialog();

}

}

else

{

lstVw.Items.Clear();

string[] dataReply = reply.Split(';');

for (int i = 0; i < dataReply.Length; i++)

{

string[] user = dataReply[i].Split(',');

lstVw.Items.Add(new User { Name = user[0], Age = user[1], Nickname = user[2], Level = user[3], Discount = user[4] });

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

else

{

MessageBox.Show("Заполните поле!!!");

}

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

try

{

(bytes, bytesRec) = Sender.SendMessageFromSocket("COLUMN\_NAME;null");

reply = Encoding.UTF8.GetString(bytes, 0, bytesRec);

string[] dataReply = reply.Split(',');

Array.Reverse(dataReply);

for (int i = 1; i < dataReply.Length; i++)

{

if (!dataReply[i].Contains("avatar"))

{

cbColumnName.Items.Add(dataReply[i]);

}

}

cbColumnName.SelectedIndex = 0;

}

catch (Exception ex)

{

if (MessageBox.Show(ex.Message + "\nПовторить попытку подлючения?", "Error", MessageBoxButton.YesNo, MessageBoxImage.Error)

== MessageBoxResult.Yes)

{

Window\_Loaded(this, null);

}

else

{

this.Close();

}

}

}

private void lstVw\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (lstVw.Items.Count > 0)

{

User selectItem = (User)lstVw.SelectedItem;

tblName.Text = selectItem.Name;

tblAge.Text = selectItem.Age;

tblNick.Text = selectItem.Nickname;

tblLevel.Text = selectItem.Level;

tblSale.Text = selectItem.Discount;

(avatar, bytesRec) = Sender.SendMessageFromSocket("SEARCHIMAGE;" + "nickname" + "," + selectItem.Nickname);

if (!Encoding.UTF8.GetString(avatar, 0, bytesRec).Contains("NOFOTO"))

{

BitmapImage avatarImage = new BitmapImage();

avatarImage.BeginInit();

avatarImage.StreamSource = new MemoryStream(avatar);

avatarImage.EndInit();

imgAvatar.Source = avatarImage;

}

else

{

imgAvatar.Source = Converter.ConvertToBitmapImage(Properties.Resources.ImageNotFound);

}

}

}

}

После полной загрузки пользователь может совершить поиск по нескольким атрибутам и получить все совпадения виде таблицы, затем выбрав одну из записей увидеть ее в более презентабельном виде с закреплённым изображением, если такое есть.

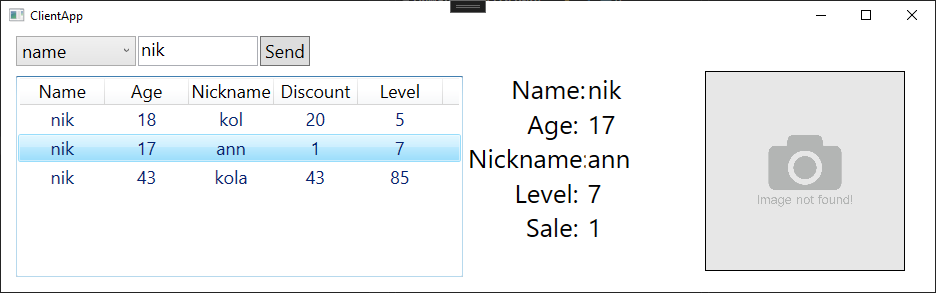


Рис. 5 Форма клиента в работе

Также при отсутствии совпадений пользователю предлагается добавить новую запись, после чего открывается форма вставки (рис. 3).

У приложения «Клиент» при запросе данных выстраивается конструкция из тегов:

тег\_задача:рабочие\_данные1,рабочие\_данные2,…

Сервер принимает данную строку и отделяет тег от данных.

Листинг 3 – Program.cs

internal class Program

{

static void Main()

{

// Устанавливаем для сокета локальную конечную точку

IPHostEntry ipHost = Dns.GetHostEntry("localhost");

IPAddress ipAddr = ipHost.AddressList[0];

int port = 11000;

IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, port);

// Создаем сокет Tcp/Ip

Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

// Назначаем сокет локальной конечной точке и слушаем входящие сокеты

try

{

int maxValueConnection = 10;

sListener.Bind(ipEndPoint);

sListener.Listen(maxValueConnection);

// Начинаем слушать соединения

while (true)

{

Console.WriteLine("Ожидаем соединение через порт {0}", ipEndPoint);

// Программа приостанавливается, ожидая входящее соединение

Socket handler = sListener.Accept();

string data = null;

// Мы дождались клиента, пытающегося с нами соединиться

byte[] bytes = new byte[1024];

byte[] avatar = new byte[2097152];

int bytesRec = handler.Receive(bytes);

data = Encoding.UTF8.GetString(bytes, 0, bytesRec);

Console.WriteLine($"Входящие данные: {data}");

string[] dataedit = data.Split(new char[] { ';', });

string work = dataedit[0];

string dataDB = dataedit[1];

ConnectionDB.OpenConnection();

string reply = "";

byte[] msg;

switch (work)

{

case "SEARCH":

reply = ConnectionDB.Search(dataDB);

msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

Console.WriteLine($"Отправленые данные: {reply}");

break;

case "SEARCHIMAGE":

avatar = ConnectionDB.SearchImage(dataDB);

handler.Send(avatar);

Console.WriteLine($"Отправленые данные: image");

break;

case "SEARCHNOTIMAGE":

reply = ConnectionDB.SearchNotImage(dataDB);

msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

Console.WriteLine($"Отправленые данные: {reply}");

break;

case "INSERT":

if (dataDB.Contains("INSIMAGE"))

{

byte[] msgImage = Encoding.UTF8.GetBytes("NEED\_IMAGE");

handler.Send(msgImage);

handler.Receive(avatar);

reply = ConnectionDB.Insert(dataDB, avatar);

}

else

{

reply = ConnectionDB.InsertNotImage(dataDB);

}

msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

Console.WriteLine($"Отправленые данные: {reply}");

break;

case "COLUMN\_NAME":

reply = ConnectionDB.GetColumnName();

msg = Encoding.UTF8.GetBytes(reply);

handler.Send(msg);

Console.WriteLine($"Отправленые данные: {reply}");

break;

}

ConnectionDB.CloseConncetion();

handler.Shutdown(SocketShutdown.Both);

handler.Close();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

finally

{

Console.ReadLine();

}

}

}

После разделения строки сервер сравнивает тег с имеющимся списком ветвлений, при нахождении совпадения запускает ветвь, на которой выполняется метод работы с БД и обратная отправка данных.

Все методы для работы с БД, в том числе и подключение к ней расположены в отдельном классе.

Листинг 4 – ConnectionDB.cs

public static class ConnectionDB

{

readonly static string cs = "Host=localhost;Username=postgres;Password=123;Database=postgres";

readonly static NpgsqlConnection con = new NpgsqlConnection(cs);

public static void OpenConnection()

{

con.Open();

}

public static void CloseConncetion()

{

con.Close();

}

public static byte[] SearchImage(string data)

{

string[] strArr = data.Split(',');

string columnName = strArr[0];

data = strArr[1];

byte[] avatar = new byte[2097152];

using (NpgsqlCommand command = new NpgsqlCommand($"SELECT avatar FROM people WHERE {columnName}='{data}';", con))

{

var reader = command.ExecuteReader();

while (reader.Read())

{

if(reader.GetValue(0) != DBNull.Value)

{

avatar = (byte[])reader.GetValue(0);

}

else

{

string reply = "NOFOTO";

avatar = Encoding.UTF8.GetBytes(reply);

}

}

reader.Close();

}

return avatar;

}

public static string Search(string data)

{

string[] strArr = data.Split(',');

string columnName = strArr[0];

data = strArr[1];

string reply = "";

using (NpgsqlCommand command = new NpgsqlCommand($"SELECT \* FROM people WHERE {columnName}='{data}';", con))

{

byte[] avatar = new byte[1024];

var reader = command.ExecuteReader();

while (reader.Read())

{

reply = ($"{reader.GetString(0)},{reader.GetInt32(1)},{reader.GetString(2)},{reader.GetInt32(3)},{reader.GetInt32(4)}");

if(reader.GetValue(5) != DBNull.Value)

{

avatar = (byte[])reader.GetValue(5);

}

}

reader.Close();

if (avatar[0] != 0)

{

reply += ",IMAGE";

}

if (reply == "")

{

reply = "Error 404 Not Found!";

}

}

return reply;

}

public static string SearchNotImage(string data)

{

string[] strArr = data.Split(',');

string columnName = strArr[0];

data = strArr[1];

string reply = "";

using (NpgsqlCommand command = new NpgsqlCommand($"SELECT \* FROM people WHERE {columnName}='{data}';", con))

{

byte[] avatar = new byte[1024];

var reader = command.ExecuteReader();

while (reader.Read())

{

reply += ($"{reader.GetString(0)},{reader.GetInt32(1)},{reader.GetString(2)},{reader.GetInt32(3)},{reader.GetInt32(4)};");

}

reader.Close();

if (reply == "")

{

reply = "Error 404 Not Found!";

}

}

return reply;

}

public static string Insert(string data, byte[] image)

{

string reply = "";

try

{

string[] dataIns = data.Split(',');

try

{

int.Parse(dataIns[1]);

int.Parse(dataIns[3]);

int.Parse(dataIns[4]);

}

catch (Exception ex)

{

reply = ex.ToString();

}

using (NpgsqlCommand command = new NpgsqlCommand($"INSERT INTO people (name, age, nickname, level, sale, avatar) VALUES (@n, @a, @nick, @lvl, @sale, @avatar)", con))

{

command.Parameters.AddWithValue("n", dataIns[0]);

command.Parameters.AddWithValue("a", int.Parse(dataIns[1]));

command.Parameters.AddWithValue("nick", dataIns[2]);

command.Parameters.AddWithValue("lvl", int.Parse(dataIns[3]));

command.Parameters.AddWithValue("sale", int.Parse(dataIns[4]));

command.Parameters.AddWithValue("avatar", image);

int nRows = command.ExecuteNonQuery();

Console.WriteLine(String.Format("Number of rows inserted={0}", nRows));

reply = String.Format("INSERTED");

}

}

catch (Exception ex)

{

reply = ex.ToString();

}

return reply;

}

public static string InsertNotImage(string data)

{

string reply = "";

try

{

string[] dataIns = data.Split(',');

try

{

int.Parse(dataIns[1]);

int.Parse(dataIns[3]);

int.Parse(dataIns[4]);

}

catch (Exception ex)

{

reply = ex.ToString();

}

using (NpgsqlCommand command = new NpgsqlCommand($"INSERT INTO people (name, age, nickname, level, sale, avatar) VALUES (@n, @a, @nick, @lvl, @sale, @avatar)", con))

{

command.Parameters.AddWithValue("n", dataIns[0]);

command.Parameters.AddWithValue("a", int.Parse(dataIns[1]));

command.Parameters.AddWithValue("nick", dataIns[2]);

command.Parameters.AddWithValue("lvl", int.Parse(dataIns[3]));

command.Parameters.AddWithValue("sale", int.Parse(dataIns[4]));

command.Parameters.AddWithValue("avatar", DBNull.Value);

int nRows = command.ExecuteNonQuery();

Console.WriteLine(String.Format("Number of rows inserted={0}", nRows));

reply = String.Format("INSERTED");

}

}

catch (Exception ex)

{

reply = ex.ToString();

}

return reply;

}

internal static string GetColumnName()

{

string reply = "";

try

{

using (NpgsqlCommand command = new NpgsqlCommand($"SELECT column\_name FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME='people'", con))

{

var reader = command.ExecuteReader();

while (reader.Read())

{

reply += $"{reader.GetString(0)},";

}

reader.Close();

}

}

catch (Exception ex)

{

reply = ex.ToString();

}

return reply;

}

}

Здесь происходит работы с БД: чтение, изменение, запись данных, а также обработка исключений, связанных с некорректными данными.

СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ