**СОДЕРЖАНИЕ**

1.ВВЕДЕНИЕ

1.1. Назначение системы

1.2. Область применения системы

1.3. Определения, акронимы, аббревиатуры

1.4. Обзор системы

2. ОБЩЕЕ ОПИСАНИЕ СИСТЕМЫ

2.1. Режимы и состояния системы

2.2. Основные функциональные возможности системы

2.3. Основные условия системы

2.4. Основные ограничения системы

2.5. Характеристики пользователя

2.6. Оперативные сценарии

3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ, УСЛОВИЯ И ОГРАНИЧЕНИЯ СИСТЕМЫ

3.1. Физические

3.1.1. Конструкция

3.1.2. Износостойкость

3.1.3. Адаптируемость

3.1.4. Условия окружающей среды

3.2. Рабочие характеристики системы

3.3. Безопасность системы

3.4. Работа системы

3.4.1. Эргономика системы

3.4.2. Ремонтопригодность системы

3.4.3. Надежность системы

3.5. Стратегия и регулирование

3.6. Устойчивость жизненного цикла системы

4. ИНТЕРФЕЙСЫ СИСТЕМЫ

5. КЛАССЫ КОДИРОВАНИЯ

5.1. Описание классов

5.2. Соответствие классов проектирования и классов кодирования

1. ВВЕДЕНИЕ

1.1. Назначение системы

Система доставки еды создаётся с целью оптимизации процессов доставки блюд и уменьшения расходов на обслуживание

1.2. Область применения системы

Система применяется в ресторанном бизнесе.

1.3. Определения, акронимы, аббревиатуры

Транзит данных – процесс получения и обработки данных из базы данных для предоставления сервером данных клиенту.

URL - Единый указатель ресурса. URL служит стандартизированным способом записи адреса ресурса в сети Интернет.

Авторизация - предоставление определённому лицу или группе лиц прав на выполнение определённых действий, а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

Аккаунт - Учётная запись — запись, содержащая сведения, которые пользователь сообщает о себе некоторой компьютерной системе.

База данных – это информационная модель, позволяющая упорядоченно хранить данные о группе объектов, обладающих одинаковым набором свойств. Программное обеспечение, предназначенное для работы с базами данных, называется система управления базами данных (СУБД).

1.4. Обзор системы

Система имеет вид клиент-серверного приложения. Пользователь системы работает с заказами клиентов. Клиент может работать в системе оффлайн и копить материал для загрузки на сервер. Во время соединения с сервером данные пользователя синхронизируются с данными других пользователей системы.

2. ОБЩЕЕ ОПИСАНИЕ СИСТЕМЫ

2.1. Режимы и состояния системы

Система может находиться в следующих состояниях:

* Онлайн – в этом режиме все изменения внесённые в базу данных пользователем системы синхронизируются с серверной базой данных.
* Оффлайн – в этом режиме все изменения внесённые в базу данных пользователем системы сохраняются в специально отведённой для этого таблице.

2.2. Основные функциональные возможности системы



2.3. Основные условия системы

Для работы системы необходимы сервер (обработка запросов от клиентов, синхронизация клиентской и серверной баз данных), клиентское приложение (работа с локальной базой данных) и сеть интернет (связь клиентского приложения с сервером)

Пользователю системы необходимо только клиентское приложение для доступа к системе.

2.4. Основные ограничения системы

Пользователь не сможет использовать систему, если его ресторан не зарегистрирован на сервере. Кроме этого, в случае неполадок на сервере синхронизировать работу сотрудников будет невозможно.

2.5. Характеристики пользователя

Оператор - конечный пользователь системы. Количество пользователей этого типа может быть любым. Использует систему для обработки заказов клиентов и обмена данными с другими операторами.

Администратор - имеет все права и полный доступ к системе.

2.6. Оперативные сценарии

Данная система предполагает следующий сценарий:

Все пользователи системы получают доступ к системе через клиентское приложение. Для доступа к ресурсам системы необходимо пройти процесс авторизации или регистрации, если пользователь не зарегистрирован.

Пользователь системы (оператор) обрабатывает заказы клиентов, позвонивших оператору по телефону.

Администратор редактирует базу данных сотрудников.

3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ, УСЛОВИЯ И ОГРАНИЧЕНИЯ СИСТЕМЫ

3.1. Физические

3.1.1. Конструкция

Серверная часть располагается на пространстве арендованного у третьей стороны сервера. Техническое обслуживание и поддержка сервера реализуется арендатором.

Клиентское приложение является закупаемым у автора ПО.

3.1.2. Износостойкость

Данный параметр системы зависит от арендатора, предоставляющего сервер.

3.1.3. Адаптируемость

В случае увеличения нагрузки на сервер, требуется купить сервер с характеристиками по лучше.

3.1.4. Условия окружающей среды

Состояние серверов минимально зависит от условий окружающей среды, погодных условий и других внешних факторов.

Возможны перебои в работе системы или полная или частичная потеря работоспособности системы в результате повреждения сервера или линий связи природными или техногенными катастрофами.

3.2. Рабочие характеристики системы

Критическое количество запросов, обрабатываемых сервером, равно 4000 в час или приблизительно 65 запросов в минуту. При превышении данного порога запросы начнут обрабатываться с задержкой пропорциональной превышению порога.

3.3. Безопасность системы

Система должна быть обеспечена защитой от XSS-атак, DDoS-атак.

При неправильном вводе пароля в количестве трех раз, для авторизации необходимо ввести код с изображения.

3.4. Работа системы

3.4.1. Эргономика системы

Требования по распределению функций для различных групп пользователей описано в ТЗ.

3.4.2. Ремонтопригодность системы

Ремонтопригодность системы зависит от арендодателя сервера.

3.4.3. Надежность системы

Техническая составляющая надежности системы зависит от арендодателя сервера.

3.5. Стратегия и регулирование

Администраторы отвечают за базу данных сотрудников (операторов системы).

Регистрация ресторана в системе требует разрешения арендатора сервера.

3.6. Устойчивость жизненного цикла системы

Для устойчивости жизненного цикла системы используется обратная связь с арендатором сервера.

4. ИНТЕРФЕЙСЫ



Сортировщик данных

interface IDB

{

string GetDBInformation();

}

Офлайн-контроллер

interface IOfflineController

{

bool CompareDataDateTime();

void UpdateData();

}

Транзит данных

interface IServer

{

void Start(TcpClient client);

}

Графический интерфейс

interface IGUI

{

void ShowGUI();

}

Хранилище данных

interface IDBConnection

{

string GetDBConnectionParameters();

}

Генератор запросов

interface IConnection

{

bool Ping();

void RegCompany(string companyName);

void AuthCompany(string companyKey);

void RegManager(string name);

void AuthManager(string name, string password);

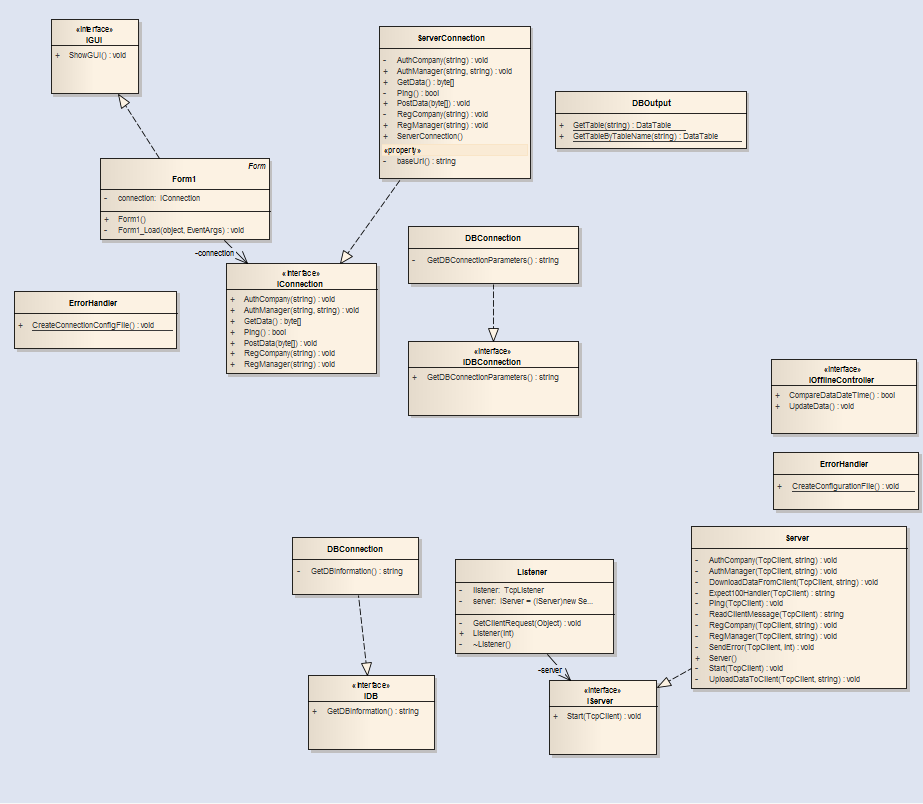
byte[] GetData();

void PostData(byte[] data);

}

5. КЛАССЫ КОДИРОВАНИЯ

5.1) UML диаграмма



5.2) Классы кодирования

Обработчик ошибок

class ErrorHandler

{

public static void CreateConnectionConfigFile()

{

using (StreamWriter sw = new StreamWriter("connection.cfg"))

{

sw.WriteLine("http://127.0.0.1:8080");

sw.Close();

}

}

}

Генератор запросов

class ServerConnection : IConnection

{

private string baseUrl { get; set; }

public ServerConnection()

{

if (!File.Exists("connection.cfg")) ErrorHandler.CreateConnectionConfigFile();

using (StreamReader sr = new StreamReader("connection.cfg"))

{

baseUrl = sr.ReadLine();

}

}

bool IConnection.Ping()

{

using (var client = new HttpClient())

{

var method = "/Ping";

client.BaseAddress = new Uri(baseUrl);

var response = client.GetAsync(method);

//if(response.Result.StatusCode==HttpStatusCode.)

return response.Result.IsSuccessStatusCode;

}

}

void IConnection.RegCompany(string companyName)

{

using (var client = new HttpClient())

{

var method = "/RegCompany";

client.BaseAddress = new Uri(baseUrl);

HttpContent content = new ByteArrayContent(Encoding.UTF8.GetBytes(companyName));

HttpContent httpContent = null;

var response = client.PostAsync(method, content)

.ContinueWith(responseTask =>

{

httpContent = responseTask.Result.Content;

});

response.Wait();

string responseFromServer = httpContent.ReadAsStringAsync().Result;

using (StreamWriter sw = new StreamWriter("companyIdentificator"))

{

sw.WriteLine(responseFromServer);

sw.Close();

}

}

}

void IConnection.AuthCompany(string companyKey)

{

using (var client = new HttpClient())

{

var method = "/AuthCompany";

client.BaseAddress = new Uri(baseUrl);

HttpContent content = new ByteArrayContent(Encoding.UTF8.GetBytes(companyKey));

HttpContent httpContent = null;

var response = client.PostAsync(method, content)

.ContinueWith(responseTask =>

{

httpContent = responseTask.Result.Content;

});

response.Wait();

string responseFromServer = httpContent.ReadAsStringAsync().Result;

}

}

public void RegManager(string name)

{

throw new **NotImplementedException**();

}

public void AuthManager(string name, string password)

{

throw new **NotImplementedException**();

}

public byte[] GetData()

{

throw new **NotImplementedException**();

}

public void PostData(byte[] data)

{

throw new **NotImplementedException**();

}

}

Подключение к базе данных

class DBConnection : IDBConnection

{

string IDBConnection.GetDBConnectionParameters()

{

using (StreamReader sr = new StreamReader("db.cfg"))

{

return sr.ReadLine();

}

}

}

Вывод данных с БД

class DBOutput

{

public static DataTable GetTable(string sql)

{

IDBConnection conn = (IDBConnection) new DBConnection();

using (var connection = new NpgsqlConnection(conn.GetDBConnectionParameters()))

{

connection.Open();

DataSet dataSet = new DataSet();

NpgsqlDataAdapter dataAdapter = new NpgsqlDataAdapter(sql, connection);

dataSet.Reset();

dataAdapter.Fill(dataSet);

return dataSet.Tables[0];

}

}

public static DataTable GetTableByTableName(string tableName)

{

string sql = String.Format("SELECT \* FROM {0}", tableName);

return GetTable(sql);

}

}

Подключение к БД сервера

class DBConnection : IDB

{

string IDB.GetDBInformation()

{

using (StreamReader sr = new StreamReader("db.cfg"))

{

return sr.ReadLine();

}

}

}

Обработчик ошибок

class ErrorHandler

{

public static void CreateConfigurationFile()

{

using (StreamWriter sw = new StreamWriter("server-port.cfg"))

{

sw.WriteLine("8080");

sw.Close();

}

}

}

Слушатель

class Listener

{

TcpListener listener;

IServer server = (IServer)new Server();

public Listener(int Port)

{

listener = new TcpListener(IPAddress.Any, Port);

listener.Start();

while (true)

{

GetClientRequest(listener.AcceptTcpClient());

}

}

void GetClientRequest(Object StateInfo)

{

server.Start((TcpClient)StateInfo);

}

~Listener()

{

if (listener != null)

{

listener.Stop();

}

}

}

Транзит данных

class Server :IServer

{

private string ReadClientMessage(TcpClient Client)

{

byte[] Buffer = new byte[Client.ReceiveBufferSize];

int count;

try

{

count = Client.GetStream().Read(Buffer, 0, Buffer.Length);

return (count > 0) ? Encoding.UTF8.GetString(Buffer, 0, count) : null;

}

catch

{

Client.Close();

return null;

}

}

void IServer.Start(TcpClient Client)

{

string Request = ReadClientMessage(Client);

if (!Client.Connected) return;

Match ReqMatch = Regex.Match(Request, @"^\w+\s+[\W]+([^\s\?]+)[^\s]\*\s+HTTP/.\*|");

if (ReqMatch == Match.Empty)

{

SendError(Client, 400);

return;

}

string RequestUri = String.Empty;

RequestUri = ReqMatch.Groups[1].Value;

RequestUri = Uri.UnescapeDataString(RequestUri);

if (RequestUri.IndexOf("..") >= 0)

{

SendError(Client, 400);

return;

}

var methods = this.GetType().GetMethods(BindingFlags.NonPublic | BindingFlags.Instance);

foreach (var method in methods)

{

if (method.Name == RequestUri)

{

method.Invoke(this, (method.GetParameters().Length == 1) ? new object[] { Client }

: new object[] { Client, Request });

Console.WriteLine("Invoked {0}",method.Name);

return;

}

}

SendError(Client, 404);

}

private void SendError(TcpClient Client, int Code)

{

Console.ForegroundColor=ConsoleColor.Red;

Console.WriteLine("Error");

Console.ForegroundColor = ConsoleColor.Gray;

string MessageText = String.Format("HTTP/1.1 {0} {1}\n\n", Code, (HttpStatusCode)Code);

byte[] buf = Encoding.UTF8.GetBytes(MessageText);

Client.GetStream().Write(buf, 0, buf.Length);

Client.Close();

}

public Server()

{

}

private void Ping(TcpClient Client)

{

string MessageText = "HTTP/1.1 200 OK\n\n";

byte[] buf = Encoding.UTF8.GetBytes(MessageText);

Client.GetStream().Write(buf, 0, buf.Length);

Client.Close();

}

private void RegCompany(TcpClient Client,string Request)

{

try

{

if (Request.IndexOf("Expect: 100-continue") > 0)

{

Request = Expect100Handler(Client);

if (!Client.Connected) return;

}

string resource = Guid.NewGuid().ToString();

string MessageText = string.Format(

"HTTP/1.1 200 OK\nContent-Length:{0}\n\n{1}",

resource.Length, resource);

byte[] buf = Encoding.UTF8.GetBytes(MessageText);

Client.GetStream().Write(buf, 0, buf.Length);

Client.Close();

}

catch

{

SendError(Client,400);

}

}

private void AuthCompany(TcpClient Client, string Request)

{

try

{

if (Request.IndexOf("Expect: 100-continue") > 0)

{

Request = Expect100Handler(Client);

if (!Client.Connected) return;

}

string MessageText = "HTTP/1.1 200 OK\n\n";

byte[] buf = Encoding.UTF8.GetBytes(MessageText);

Client.GetStream().Write(buf, 0, buf.Length);

Client.Close();

}

catch

{

SendError(Client, 400);

}

}

private void RegManager(TcpClient Client, string Request)

{

throw new **NotImplementedException**();

}

private void AuthManager(TcpClient Client, string Request)

{

throw new **NotImplementedException**();

}

private void DownloadDataFromClient(TcpClient Client, string Request)

{

throw new **NotImplementedException**();

}

private void UploadDataToClient(TcpClient Client, string Request)

{

throw new **NotImplementedException**();

}

private string Expect100Handler(TcpClient Client)

{

string MessageText = "HTTP/1.1 100 Continue\n\n";

byte[] buf = Encoding.UTF8.GetBytes(MessageText);

Client.GetStream().Write(buf, 0, buf.Length);

return ReadClientMessage(Client);

}

}