1. **Назначение и условия применения программы**

Программа предназначена для упрощения ведения сведений об оказании услуг различного рода на предприятии.

Требования к техническим средствам:

|  |  |
| --- | --- |
| Название | Характеристика |
| Процессор Intel Core i3-2130 Sandy Bridge LGA1155, 2 x 3400 МГц | Сокет - LGA1155  Количество ядер - 2 x 3400 МГц  Объем кэша - L2/L3 0.50 МБ/3 МБ  Тепловыделение - 65 Вт |
| Модуль памяти DDR3 4GB Patriot Memory PSD34G13332 | Стандарт – ddr3  Объем памяти – 4гб |
| 19.5" Монитор AOC e2070Swn, 1600x900, 76 Гц, TN, черный | Тип монитора - ЖК  Диагональ - 19.5 "  Макс. Разрешение -1600x900  Соотношение сторон - 16:9 |
| Клавиатура Canyon CNS-HKB5RU | Тип клавиатуры - мембранная  Интерфейс - USB |
| Мышь компьютерная Logitech B100 белая (910-003360) | Тип мыши – оптическая  Интерфейс - USB  Разрешение dpi - 800  Тип питания - USB |
| Жесткий диск Western Digital WD1000VE 100Gb 5400 IDE 2,5" HDD | Емкость – 100 Гб  Форм-фактор – 2.5" |

Требования к программным средствам:

Для корректной работы программы необходимо: ОС Windows 10,

Microsoft excel 2010 (и выше), PostgreSQL 15.0.

1. **Характеристика программы**

Программа должна обеспечивать следующие функции:

* Добавление, удаление и редактирование списка городов
* Хранение данных о городах и погоде
* Вывод минимальной, максимальной, средней погоды за выбранный период.
* Отображения графика изменения погоды за выбранный период.

1. **Обращение к программе**

C# - современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

Программный код приложения разбит на 2 части:

Веб-сервер - получает и хранит информацию;

Приложение – интерфейс, обращается к веб серверу и выводит информацию пользователю.

Функции Веб-сервера:

В файле Program.cs содержатся методы API, вызывающие функции для получения информации:

app.MapGet("/weatherforecast", async () =>

{

return await utility.Mine();

});

app.MapGet("/get\_current/{city\_id}", async (string city\_id) =>

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return utility.get\_current(city\_id);

});

app.MapGet("/get\_geo/{geo}", async (string geo) =>

{

return await utility.get\_geo(geo);

});

app.MapGet("/login/{username}/{password}", async (string username, string password) =>

{

return utility.Login(username, password);

});

app.MapGet("/get\_cities/{user\_id}", async (string user\_id) =>

{

return utility.get\_cities(user\_id);

});

app.MapGet("/get\_user\_id/{username}/{password}", async (string username, string password) =>

{

return utility.get\_user\_id(username, password);

});

app.MapGet("/add\_city/{city}/{longtitude}/{latitude}/{user\_id}", async (string city, string longtitude, string latitude, string user\_id) =>

{

return await utility.add\_city(city, longtitude, latitude, user\_id);

});

app.MapGet("/register/{username}/{password}", async (string username, string password) =>

{

return utility.register(username, password);

});

app.MapGet("/delete/{user\_id}/{city\_id}", async (string user\_id, string city\_id) =>

{

return utility.delete(user\_id, city\_id);

});

app.MapGet("/get\_maximum/{city\_id}", async (string city\_id) =>

{

return utility.get\_maximum(city\_id);

});

app.MapGet("/get\_minimum/{city\_id}", async (string city\_id) =>

{

return utility.get\_minimum(city\_id);

});

app.MapGet("/get\_graph/{city\_id}", async (string city\_id) =>

{

return utility.get\_graph(city\_id);

});

app.MapGet("/get\_info/", async () =>

{

return utility.get\_info();

});

app.MapGet("/get\_city\_info/", async () =>

{

return utility.get\_city\_info();

});

app.MapGet("/get\_city/{city\_id}", async (string city\_id) =>

{

return utility.get\_city(city\_id);

});

app.MapGet("/update\_city\_info/{city\_id}", async (string city\_id) =>

{

return await utility.update\_city\_info(city\_id);

});

В файде utilities.cs хранятся методы для получения информации:

Метод Mine() проверяет наличие базы данных, и если она не создана создает ее и запускаем метод Main():

public async Task<string> Mine()

{

const string connStr = "Server=localhost;Port=5432;User Id=postgres;Password=1234;";

if (chkDBExists(connStr, "weather") != true)

{

create\_db();

}

return await Main();

}

Метод create\_db() содержит в себе логику создания бд:

public void create\_db()

{

const string connStr = "Server=localhost;Port=5432;User Id=postgres;Password=1234;";

const string connStr2 = "Server=localhost;Port=5432;User Id=postgres;Password=1234;Database=weather;";

var m\_conn = new NpgsqlConnection(connStr); // db connction

var m\_conn2 = new NpgsqlConnection(connStr2); // table connection

// creating a database in Postgresql

var m\_createdb\_cmd = new NpgsqlCommand("CREATE DATABASE weather;", m\_conn);

string script = "здесь содержится скрипт запроса для создания бд"

var m\_createtbl\_cmd = new NpgsqlCommand(script);

m\_createtbl\_cmd.Connection = m\_conn2;

// 3.. Make connection and create

// open connection to create DB

m\_conn.Open();

m\_createdb\_cmd.ExecuteNonQuery();

m\_conn.Close();

// open connection to create table

m\_conn2.Open();

m\_createtbl\_cmd.ExecuteNonQuery();

m\_conn2.Close();

}

Метод get\_data(string latitude, string longtitude) Получает температурные измерения в городе за указанный промежуток времени:

static async Task<string> get\_data(string latitude, string longtitude)

{

HttpClient httpClient = new HttpClient();

var dt = DateTime.Now;

DateTime month = dt.AddMonths(-1);

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://api.open-meteo.com/v1/forecast?latitude=" + latitude + "&longitude=" + longtitude + "&hourly=temperature\_2m,weathercode&daily=temperature\_2m\_max,temperature\_2m\_min&current\_weather=true&timezone=auto&start\_date=" + month.ToString("yyyy-MM-dd") + "&end\_date=" + DateTime.Now.ToString("yyyy-MM-dd"));

// получаем ответ

return await response.Content.ReadAsStringAsync();

}

Метод GetGEO(string city) получает координаты города, который добавляет пользователь:

static async Task<string> GetGEO(string city)

{

HttpClient httpClient = new HttpClient();

// получаем ответ

using HttpResponseMessage response = await httpClient.GetAsync("https://geocoding-api.open-meteo.com/v1/search?name=" + city + "&language=ru");

// получаем ответ

return await response.Content.ReadAsStringAsync();

}

Метод get\_current(string city\_id) возвращает данные о выбранном городе:

public double get\_current(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].CurrentWeather;

}

Метод Login(string username, string password) проверяет введеные пользователем логин и пароль, и в случае соответствия дает пользователю доступ к программе:

public string Login(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Password == password && x.Username == username).ToList();

if (res.Count == 1) return "Success";

else return "Failed";

}

Метод get\_cities(string user\_id) получает список городов пользователя:

public List<City> get\_cities(string user\_id)

{

var ss = WeatherContext.GetContext().UserInfos.Where(x => x.UserId == int.Parse(user\_id)).ToList();

long[] cc = new long[ss.Count()];

for (int i = 0; i < ss.Count(); i++)

{

cc[i] = ss[i].CityId;

}

var stupid = WeatherContext.GetContext().Cities.Where(x => cc.Contains(x.Id)).ToList();

List<City> cits = new List<City>();

for (int i = 0; i < stupid.Count(); i++)

{

City cur = new City();

cur.City1 = stupid[i].City1;

cur.Longitude = stupid[i].Longitude;

cur.Latitude = stupid[i].Latitude;

cur.Id = stupid[i].Id;

cits.Add(cur);

} return cits;

}

Метод get\_user\_id(string username, string password) возвращает айди пользователя по логину и паролю:

public string get\_user\_id(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Password == password && x.Username == username).ToList();

if (res.Count == 1) return res[0].Id.ToString();

else return "Failed";

}

Метод add\_city(string city, string longtitude, string latitude, string user\_id) добавляет город в список городов пользователя:

public async Task<string> add\_city(string city, string longtitude, string latitude, string user\_id)

{

var res = WeatherContext.GetContext().Cities.Where(x => x.City1 == city && x.Longitude == double.Parse(longtitude.Replace(".", ",")) && x.Latitude == double.Parse(latitude.Replace(".", ","))).ToList();

if (res.Count == 1)

{

UserInfo info = new UserInfo();

info.UserId = int.Parse(user\_id);

info.CityId = res[0].Id;

WeatherContext.GetContext().UserInfos.Add(info);

WeatherContext.GetContext().SaveChanges();

return "City already exist";

}

else {

City area = new City();

area.City1 = city;

area.Longitude = double.Parse(longtitude.Replace(".", ","));

area.Latitude = double.Parse(latitude.Replace(".", ","));

WeatherContext.GetContext().Cities.Add(area);

WeatherContext.GetContext().SaveChanges();

var current\_area = WeatherContext.GetContext().Cities.Where(x => x.City1 == city && x.Longitude == double.Parse(longtitude.Replace(".", ",")) && x.Latitude == double.Parse(latitude.Replace(".", ","))).ToList();

CityInfo city\_infos = new CityInfo();

var data = await get\_data(current\_area[0].Latitude.ToString().Replace(",", "."), current\_area[0].Longitude.ToString().Replace(",", "."));

UserInfo info = new UserInfo();

Root datas = JsonConvert.DeserializeObject<Root>(data);

List<double> max = datas.daily.temperature\_2m\_max;

List<double> min = datas.daily.temperature\_2m\_min;

double current\_weath = datas.current\_weather.temperature;

List<double> period = datas.hourly.temperature\_2m;

city\_infos.CurrentWeather = current\_weath;

city\_infos.MaxWeather = max.ToArray();

city\_infos.MinWeather = min.ToArray();

city\_infos.PeriodWeather = period.ToArray();

city\_infos.CityId = current\_area[0].Id;

info.UserId = int.Parse(user\_id);

info.CityId = current\_area[0].Id;

WeatherContext.GetContext().CityInfos.Add(city\_infos);

WeatherContext.GetContext().SaveChanges();

WeatherContext.GetContext().UserInfos.Add(info);

WeatherContext.GetContext().SaveChanges();

return "Success"; }

Метод register(string username, string password) осуществляет сохранение логина и пароля пользователя в бд, а также присвоение ему айди:

public string register(string username, string password)

{

var res = WeatherContext.GetContext().Users.Where(x => x.Username == username).ToList();

try

{

if (res.Count == 0)

{ User user = new User();

user.Username = username;

user.Password = password;

WeatherContext.GetContext().Users.Add(user);

WeatherContext.GetContext().SaveChanges();

return "Success";

}

else return "Username is already exist";

}

catch

{

return "Error";

}

}

Метод delete(string user\_id, string city\_id) удаляет город из списка пользователя:

public string delete(string user\_id, string city\_id)

{

var res = WeatherContext.GetContext().UserInfos.Where(x => x.UserId == int.Parse(user\_id) && x.CityId == int.Parse(city\_id)).ToList();

WeatherContext.GetContext().UserInfos.Remove(res[0]);

WeatherContext.GetContext().SaveChanges();

return "Success";

}

Метод get\_maximum(string city\_id) возвращает максимальную температуру по дням:

public double[] get\_maximum(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].MaxWeather;

}

Метод get\_minimum(string city\_id) возвращает минимальную температуру по дням:

public double[] get\_minimum(string city\_id)

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].MinWeather;

}

Метод get\_graph(string city\_id) возвращает температуру по часам:

{

var res = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

return res[0].PeriodWeather;

//WeatherContext.GetContext().User.Add(\_currentUser);

}

Метод get\_info() возвращает информацию о пользователях:

public List<UserInfo> get\_info()

{

var res = WeatherContext.GetContext().UserInfos.ToList();

return res;

}

Метод get\_city\_info() возвращает информацию о городах:

public List<int> get\_city\_info()

{

var ss = WeatherContext.GetContext().CityInfos.ToList();

List<int> cits = new List<int>();

for (int i = 0; i<ss.Count(); i++)

{

cits.Add(Convert.ToInt32(ss[i].CityId));

}

return cits;

}

Метод get\_city(string city\_id) возвращает информацию о конкретном городе:

public List<City> get\_city(string city\_id)

{

var res = WeatherContext.GetContext().Cities.Where(x => x.Id == int.Parse(city\_id)).ToList();

return res;

}

Метод update\_city\_info(string city\_id) позволяет обновлять данные о городе:

public async Task<string> update\_city\_info(string city\_id)

{

List<CityInfo> cc = WeatherContext.GetContext().CityInfos.Where(x => x.CityId == int.Parse(city\_id)).ToList();

List<City> current\_area = WeatherContext.GetContext().Cities.Where(x => x.Id == int.Parse(city\_id)).ToList();

var data = await get\_data(current\_area[0].Latitude.ToString().Replace(",", "."), current\_area[0].Longitude.ToString().Replace(",", "."));

Root datas = JsonConvert.DeserializeObject<Root>(data);

List<double> max = datas.daily.temperature\_2m\_max;

List<double> min = datas.daily.temperature\_2m\_min;

double current\_weath = datas.current\_weather.temperature;

List<double> period = datas.hourly.temperature\_2m;

cc[0].CurrentWeather = current\_weath;

cc[0].MaxWeather = max.ToArray();

cc[0].MinWeather = min.ToArray();

cc[0].PeriodWeather = period.ToArray();

WeatherContext.GetContext().SaveChanges();

return "Success";

}

Функции Приложения:

Данная функция вычисляет розничную цену:

Функция РозничнаяЦена(АктуальнаяДата, ЭлементНоменклатуры) Экспорт

// Создать вспомогательный объект "Отбор".

Отбор = Новый Структура("Номенклатура", ЭлементНоменклатуры);

// Получить актуальные значения ресурсов регистра.

ЗначенияРесурсов = РегистрыСведений.Цены.ПолучитьПоследнее(АктуальнаяДата, Отбор);

Возврат ЗначенияРесурсов.Цена;

КонецФункции

Данная функция применяется для получения актуальной цены.



Данная процедура фиксирует проведения:

Процедура ОбработкаПроведения(Отказ, Режим)

//{{\_\_КОНСТРУКТОР\_ДВИЖЕНИЙ\_РЕГИСТРОВ

// Данный фрагмент построен конструктором.

// При повторном использовании конструктора, внесенные вручную изменения будут утеряны!!!

// регистр ОстаткиТовара Расход

Режим = РежимПроведенияДокумента.Оперативный;

Движения.ОстаткиТовара.Записывать = Истина;

Движения.ДоходотУслуг.Записывать = Истина;

Для Каждого ТекСтрокаУслугииМатериалы Из УслугииМатериалы Цикл

Если ТекСтрокаУслугииМатериалы.Номенклатура.Родитель.Наименование = "Комплектующие" тогда

Движение = Движения.ОстаткиТовара.Добавить();

Движение.ВидДвижения = ВидДвиженияНакопления.Расход;

Движение.Период = Дата;

Движение.Номенклатура = ТекСтрокаУслугииМатериалы.Номенклатура;

Движение.Количество = ТекСтрокаУслугииМатериалы.Количество;

Иначе

Движение = Движения.ДоходотУслуг.Добавить();

Движение.Период = Дата;

Движение.Мастер = Мастер;

Движение.Клиент = Клиент;

Движение.Услуга = ТекСтрокаУслугииМатериалы.Номенклатура;

Движение.Сумма = ТекСтрокаУслугииМатериалы.Сумма;

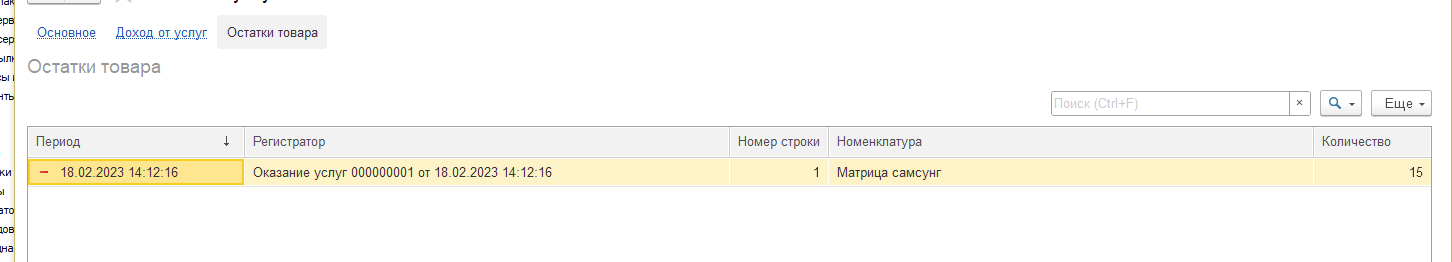
КонецЕсли;

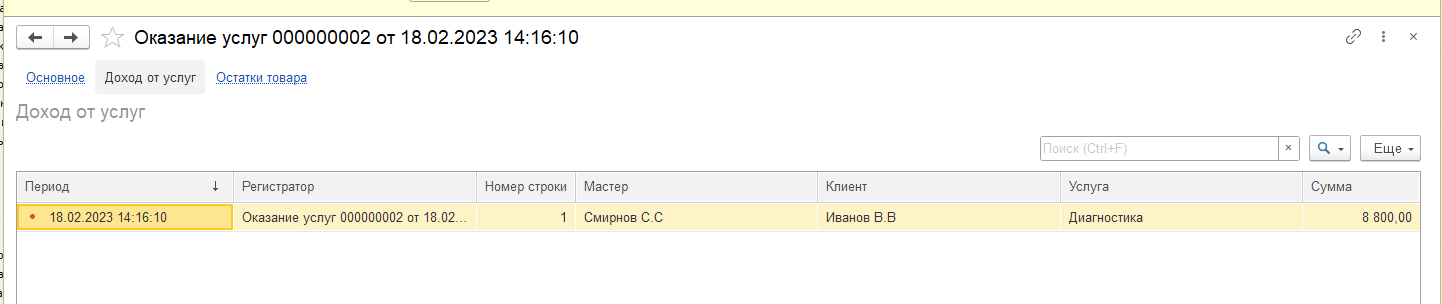
КонецЦикла;

//}}\_\_КОНСТРУКТОР\_ДВИЖЕНИЙ\_РЕГИСТРОВ

КонецПроцедуры

Результатом работы данной процедуры являются записи в разделах Доход от услуг и Остатки товара.





# Входные и выходные данные

Входные данные:

|  |  |
| --- | --- |
| **Поля** | **Тип данных** |
| ФИО | Текстовый |
| Наименование | Текстовый |
| Цена | Числовой |
| Дата | Дата |
| Материал | Текстовый |
| Услуга | Текстовый |
| Номер | Идентификатор |

**Выходная информация:**

Выходной информацией данные о погоде для пользователя.