# Загальний опис серверної частини застосунку



Мал №. Інфографічне представлення розподіленої обчислювальної системи

Взяв тут https://www.confluent.io/learn/distributed-systems/

Серверна частина застосунку «» розроблена як розподілена обчислювальна система, набір незалежних комп'ютерів, що представляється їх користувачам єдиною об'єднаною системою. Кожен елемент розподіленої системи (мережі) – автономна обчислювальна машина єдиними вимогами від якої є можливість її виявлення (Service discovery) і створення каналу обміну даними по мережі з нею. Таким чином дана обчислювальна машина може бути як фізичною машиною так і віртуальною; бути розміщеною як на приватних (корпоративних) обчислювальна машинах (мережах) так і на хмарних сервісах. Розподілені обчислювальні системи надають змогу розробляти окремі її елементи незалежно, використовуючи діаметрально різні технологічні стеки (за необхідності), а також різними командами розробників. Розподілені обчислювальні системи відкривають можливість будувати відмовостійкі, надійні та доступності системи: у разі відмови обчислювальної машини решта елементів розподіленої системи буде активною, елемент системи що відмовив можна відновити, розгонувши його на новій обчислювальній машині на постійній або тимчасовій основі (поки обчислювальна машина що відмовила не буде відновлена), даний процес у певних випадках може бути автоматизований, що підвищує доступність системи, адже не накладаються часові затримки на відновлення системи, пов’язані з застосуванням людських ресурсів.



Мал №. Логотип Microsoft WCF. Взяв тут https://blog.tonysneed.com/2016/01/06/wcf-is-dead-long-live-mvc-6/

Кожен елемент розподіленої системи (надалі сервіс) розроблено з використанням технології Windows Communication Foundation (WCF) (<https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf> посилання якщо треба). WCF – фреймворк для побудови розподілених сервісно-оріентованих застосунків. WCF значно прискорює розробку мережевих додатків. Фремворк дозволяє цілковито абстрагувати код застосунку від протоколів спілкування по комп’ютерній мережі, встановлення з’єднання підключення з клієнтом сервіса, форматів повідомлень, сереалізації та десереалізації вхідних/вихідних повідомленнь, криптографічного захисту даних, перевірці даних на справжність та коректність. Завдяки наданому функціоналу наведеному вище, взаємодія з клієнта з сервісом, з точки зору написання коду, зводиться до простого виклику методу. Також варто відзначити, що великою перевагою WCF є всеосяжна детальна і зрозуміла документація, як від постачальника технології (Microsoft) так і від спільноти спеціалістів що застосовують дану технологію.

Протоколом спілкування по комп’ютерній мережі був обраний протокол Simple Object Access Protocol (SOAP), використаний разом з протоколом прикладного рівня Hypertext Transfer Protocol (HTTP), як транспортним протоколом. Сервіси обмінюються між собою повідомленнями за специфікацією SOAP (https://www.w3.org/TR/soap/ посилання якщо треба) версії 1.2 (мал. №).

POST **/InStock** **HTTP**/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: 299

SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>

**<soap:Envelope** xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org"**>**

**<soap:Header>**

**</soap:Header>**

**<soap:Body>**

**<m:GetStockPrice>**

**<m:StockName>**T**</m:StockName>**

**</m:GetStockPrice>**

**</soap:Body>**

**</soap:Envelope>**

мал. № . Приклад SOAP повідомлення-запиту надісланого по HTTP протоколу (взяв тут https://en.wikipedia.org/wiki/SOAP)

Дані повідомлення є ієрархічно структурованими за стандартом World Wide Web Consortium - розширювана мова розмітки (англ. Extensible Markup Language, скорочено XML) (<https://www.w3.org/TR/xml/> посилання якщо треба). Завдяки використанню XML, повідомлення набувають наступних якостей:

* Коректність – будь-яке XML повідомлення може бути перевірене на предмет вірності його побудови, відповідності його стандарту XML. Обробка некоректних повідомлення має бути автоматично відхилена.
* Практичність – у платформі .NET (а також у багатьох інших) доступні синтаксичні аналізатори (парсери, від англ. parser), програмні компоненти, що аналізують і валідують формат формат повідомлення XML і трансформують такі повідомлення у об’єктну модель.
* Функціональна сумісність – XML повністю відкритий стандарт, а отже споживати повідомлення такого формату може будь-яка комп’ютерна система, тобто досягається багатоплатформність кліентів.

# Опис структури застосунку

розподілена обчислювальна система «» утворюється з наступних компонентів (див мал №): клієнтського застосунку (детальний опис у розділі «»), сервіс «IronMacbeth.UserManagement», сервіс «IronMacbeth.FileStore», сервіс «IronMacbeth.BFF».



Мал №. Структура та основні компоненти застосунку

## Сервіс «IronMacbeth.UserManagement»

Відповідальність сервісу «IronMacbeth.UserManagement» (надалі у цьому підрозділі – сервіс) полягає у автентифікації користувачів, реестрації нових користувачів у системі, а також у збереженні особистих даних користувача. Сервіс надає функціонал валідації відповідності користувацького логіна паролю.

### Механізм валідації імені та паролю користувача

Зберігати користувацькі паролі у чистому вигляді (plain text) не безпечно, а також не відповідає загальному регламенту про захист даних Європейського союзу (англ. General Data Protection Regulation, GDPR; Regulation (EU) 2016/679) (взяв тут <https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D1%80%D0%B5%D0%B3%D0%BB%D0%B0%D0%BC%D0%B5%D0%BD%D1%82_%D0%BF%D1%80%D0%BE_%D0%B7%D0%B0%D1%85%D0%B8%D1%81%D1%82_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85>). У разі маловірогідної хакерської атаки або потрапляння ключів доступу до бази даних сервісу, зловмисники зможуть отримати прямий доступ до користувацьких паролів. Для вирішення даної проблеми у сервісі реалізований наступний фунціонал: отриманий при реєстрації користувача, пароль хешується з додаванням так званої солі (англ. Salt), отриманий хеш разом із сіллю зберігається у базі даних; при повторних авторизаціях користувача, наданий пароль хешується з сіллю, що була попередньо збережена у базі даних, отриманий хеш перевіряється з хешем у базі даних; якщо встановлена побітна відповідність двох хешів – користувач надав правильних пароль, якщо ні – надані ім’я та пароль користувача не відповідають один одному (див додаток «»).

Сіль – сгенерована випадковим чином послідовність даних, основною ціллю якої є приховування однакових паролів. Наприклад, два користувачі обрали один і той самий пароль, в такому разі в базі даних для них будуть збережені однакові хеші. Сіль нівелює дану вразливість системи, так як, для користувачів будуть сгенеровані різні послідовності, що унеможливить виявити однакові паролі у користувачів.

Хеш-функція – функція що перетворює вхідні дані будь-якого (як правило великого) розміру в дані фіксованого розміру, що не має зворотної (функція яка б перетворювала хеш у оригінальне значення). У сервісі застосована хеш функція SHA256. На момент написання роботи, дана хеш-функція вважається стійкою, та такою що не має вразливостей.

## Сервіс «IronMacbeth.FileStorage»

Сервіс «IronMacbeth.FileStorage» (надалі у цьому підрозділі – сервіс) відповідальний за збереження неструктурованих бінарних файлів. Даний сервіс абстрагує спосіб збереження файлів від решти розподіленої системи. Наприклад файли можуть зберігатись на дисковому носії інформацій або у хмарному сховищі; файли можуть зберігатись як заархівовані чи у оригінальному стані, чи комбінований варіант, коли часто використовувані файл архівуються і зберігаються у холодному сховищі (такому що має великий обсяг, проте невисоку швидкодію) або у гарячому (швидкодіючий носій інформації проте більш дорогий), тощо; зашифровані чи ні (наприклад файли з вразливою інформацією шифруються, а ті що були отримати з загальнодоступних джерел - ні).

Сервіс надає наступний прикладний програмний інтерфейс (англ. Аpplication Programming Interface, API) виражений інтерфейсом C# (див мал №.).

using System;

using System.IO;

using System.ServiceModel;

namespace IronMacbeth.FileStorage.Contract

{

[ServiceContract]

public interface IFileStorageService

{

[OperationContract]

Guid AddFile(Stream fileStream);

[OperationContract]

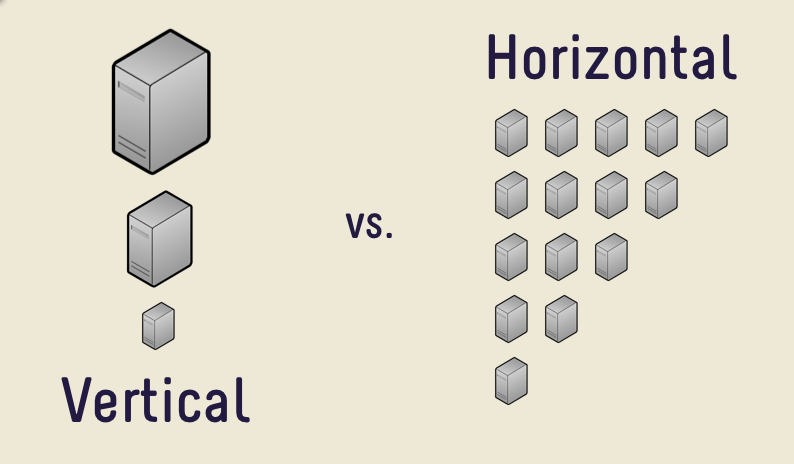
Stream GetFile(Guid fileId);

}

}

Мал № . API Сервісу «IronMacbeth.FileStorage»

Унікальними ідентифікаторами файлів будо обрано GUID (англ. Globally Unique Identifier). GUID – ідентифікатор розмірністю 16 байт з загальною кількістю унікальних ключів рівною 2128. Ймовірність того, що у світі будуть незалежно сгенеровані два однакові ідентифікатори вкрай мала. Таким чином сервіс, в подальшому, за необхідності, можна піддати як вертикальному так і горизонтальному масштабуванню (мал №.).



Мал № . Інфографічне представлення вертикального і горизонтального масштабування.

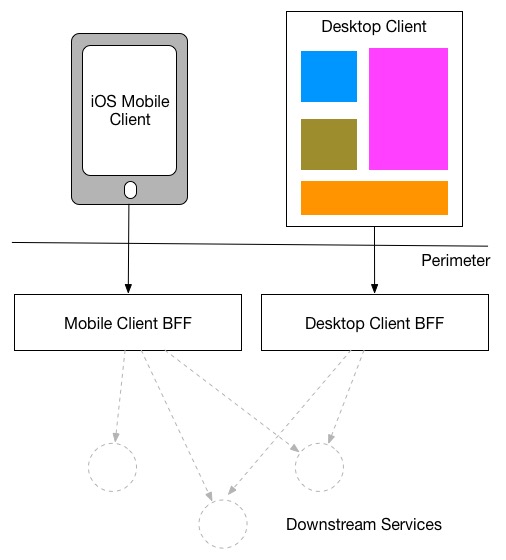
Вертикальне масштабування – нарощування комп’ютерних ресурсів (процесора, оперативної пам’яті, носіїв інформації) певної обчислювальної машини.

Горизонтальне масштабування – нарощування комп’ютерних ресурсів розподіленої системи шляхом введення додаткових обчислювальних машин, з розподіленням навантаження між існуючими і доданими обчислювальними машинами. Даний тип масштабування дозволяє досягти більшої гнучкості у порівнянні з вертикальним масштабуванням, адже не тільки загальна обчислювальна здатність горизонтально масштабованих систем значно перевищує вертикально масштабовані, а ще й збільшує стійкість системи (якщо одна обчислювальна машина з низки горизонтально масштабованих вийде з ладу, користувачі лише відчують зниження швидкості виконання операцій, але не втратять можливість користуватись функціоналом, як це відбувається з вертикально масштабованими обчислювальними машинами); також можливо досягти ефекту підвищеної ефективності системи завдяки розміщенню додаткових горизонтально масштабованих обчислювальних машин безпосередньо у фізичному регіоні де швидкодія системи найбільш важлива.

## Сервіс «IronMacbeth.BFF».

Сервіс «IronMacbeth.BFF» (BBF - скорочено від Backend For Frontend) (надалі у цьому підрозділі – сервіс) виконує роль абстракції розподіленої системи від кліента, обслуговує запити кліента, переспрямовуючи їх до інших сервісів за необхідності.

Було прийнято рішення використовувати архітектурний підхід Backends For Frontends для можливості в подальшому розширяти додаток для підтримки веб та мобільної версії застосунку які також будуть потребувати абстрагування від розподіленої системи проте іншого інтерфейсу взаємодії (див мал №.).



Мал № . Інфографічне представлення архітектури розподіленого застосунку з використанням підходу Backends For Frontends.

Підхід Backends For Frontends дозволяє значно спростити розробку клієнтського додатку, адже клієнтська частина застосунку буде розвантажена від взаємодії з розподіленою системи, а також споживати прикладний програмний інтерфейс спеціально націлений на потреби клієнтського застосунку (BFF сервіс агрегує та трансформує дані, отримані від певного переліку інших обчислюваних машин розподіленої системи, у вигляд, який найбільш підходить під потреби кліента).

## Методи забезпечення цілісності та шифрування даних, якими обмінюються компоненти

У системах, що базуються на архітектурі розподілених застосунків, невід’ємною частиною штатного функціонування системи є спілкування по мережі. Повідомлення надіслане від одного мережевого пристрою, що проходить по мережі може бути зчитане третьою стороною, таким чином конфіденційні дані можуть бути скомпрометовані. Також третя сторона може виконати атаку «людина посередині» (англ. Man-in-the-middle attack), а саме перехопити повідомлення відправника, видозмінити його і відправити повідомлення отримувачу начебто від імені оригінального відправника.

Для запобігання негативних наслідків розподілених систем, наведених вище, у застосунку використовується шифрування даних та їх цифрове підписання, що засвідчує цілісність даних та їх відправника. Даний функціонал досягається завдяки використанню серверного сертифікату інфраструктури відкритого ключа (англ. Public Key Infrastructure Certificate) стандарту X.509 (посилання <https://tools.ietf.org/html/rfc5280>).

Фунціоналу захисту даних серверним сертифікатом було досягнуто шляхом використання стандартних механізмів, наданих фреймворком WCF.

## модель авторизації між компонентами

У застосунку встановлена наступна модель авторизації між компонентами:

* IronMacbeth.Client – IronMacbeth.BFF. IronMacbeth.Client (надалі, у даному підрозділі, «кліент») має два типи з’єднання з IronMacbeth.BFF (надалі, даному підрозділі, «сервіс»): анонімне і авторизоване. Анонімне з’єднання надає користувачу значно обмежений функціонал і застосовується для того щоб користувач міг зареєструватись у системі. Авторизоване з’єднання вимагає від користувача коректну пару імені та паролю. Обидва з’єднання захищені серверним сертифікатом.
* Взаємодія між сервісами «IronMacbeth.UserManagement», «IronMacbeth.FileStore» та «IronMacbeth.BFF» (надалі, у даному підрозділі, «сервіси»). Взаємодія між сервісами захищена шляхом використання сертифікатів. Сервіс, що ініціює підключення до іншого сервісу, має надати свій сертифікат сервісу, до якого він ініціював підключення, наданий сертифікат має бути у списку сертифікатів яким довіряє сервіс до якого встановлюється підключення; сервіс, до якого встановлюється підключення, надає свій сертифікат сервісу, що ініціює підключення, для його перевірки. Якщо обидва сертифікати успішно проходять перевірку, встановлюється з’єднання, захищене сертифікатом сервісу який виступає у ролі сервера (сервіс до якого ініціюється підключення).

# Додаток . Сирцевий код (ахахха) створення і верифікації хешу користувацьких паролів (~\IronMacbeth.UserManagement\IronMacbeth.UserManagement\SecurePasswordHasher.cs)

using System;

using System.Linq;

using System.Security.Cryptography;

namespace IronMacbeth.UserManagement

{

public static class SecurePasswordHasher

{

private const int SaltSize = 16;

private const int HashSize = 20;

private const int Iterations = 10000;

public static string Hash(string password)

{

// Create salt

byte[] salt = new byte[SaltSize];

new RNGCryptoServiceProvider().GetBytes(salt);

// Create hash

var pbkdf2 = new Rfc2898DeriveBytes(password, salt, Iterations, HashAlgorithmName.SHA256);

var hash = pbkdf2.GetBytes(HashSize);

// Combine salt and hash

var hashBytes = new byte[SaltSize + HashSize];

Array.Copy(salt, 0, hashBytes, 0, SaltSize);

Array.Copy(hash, 0, hashBytes, SaltSize, HashSize);

// Convert to base64

var base64Hash = Convert.ToBase64String(hashBytes);

// Format hash with extra information

return base64Hash;

}

public static bool Verify(string password, string hashedPassword)

{

// Get hash bytes

var hashBytes = Convert.FromBase64String(hashedPassword);

// Get salt

var salt = new byte[SaltSize];

Array.Copy(hashBytes, 0, salt, 0, SaltSize);

// Create hash with given salt

var pbkdf2 = new Rfc2898DeriveBytes(password, salt, Iterations, HashAlgorithmName.SHA256);

byte[] hash = pbkdf2.GetBytes(HashSize);

// Get result

var isMatch = hashBytes.Skip(SaltSize).SequenceEqual(hash);

return isMatch;

}

}

}