Система для замовлення піци

“Прийшли З’їли”.

Загальний огляд розроблених архітектур

Backend веб сервісу для мережі піцерій “Прийшли З’їли” був реалізований за допомогою трьох різних технологій:

- WebApi (.NET) - https://github.com/yholub/Pizza_ordering
- Python - https://github.com/yholub/Pizza_Ordering_Python
- Microservices - https://github.com/yholub/Pizza_Ordering_Microservices (відокремлення рівня представлення бази даних (детальніше у описі нижче))

Frontend

Оскільки ми хотіли досягнути максимальної незалежності модулів, то на фронтенді було вирішено створили SPA (Single Page Application), яка комунікує з бекендом через REST сервіси.

Було реалізовано модель Model-View-ViewModel за допомогою KnockoutJS, який дав можливість гнучкої взаємодії між моделями на представленням.

Окрім цього було використано jquery і звісно стандартний стек веб-технологій – javascript, html, css.

.NET

.NET проект складається з кількох шарів: Web, Services, DataProvider, Domain, Common. Така архітектура дала змогу будувати компоненти з меншою залежністю один від іншого.

Оскільки ми створили SPA, яка комунікує з бекендом через REST сервіси, то для реалізації REST на .NET вдало підійшла технологія Asp.Net Web ApI.

Python

На python дещо змінилась структура проекту, відповідно до вимог даної мови. Для реалізації REST було використано Flask та SQL alchemy в якості ORM.

Microservices

Для того, щоб розроблений проект був максимально гнучким ми відокремили доступ до бази окремим рівнем. За допомогою нової архітектури можна легко писати модулі різними мовами програмування і при цьому використовувати усі переваги мов, обираючи для потсавлених задач ті, які краще підійдуть.

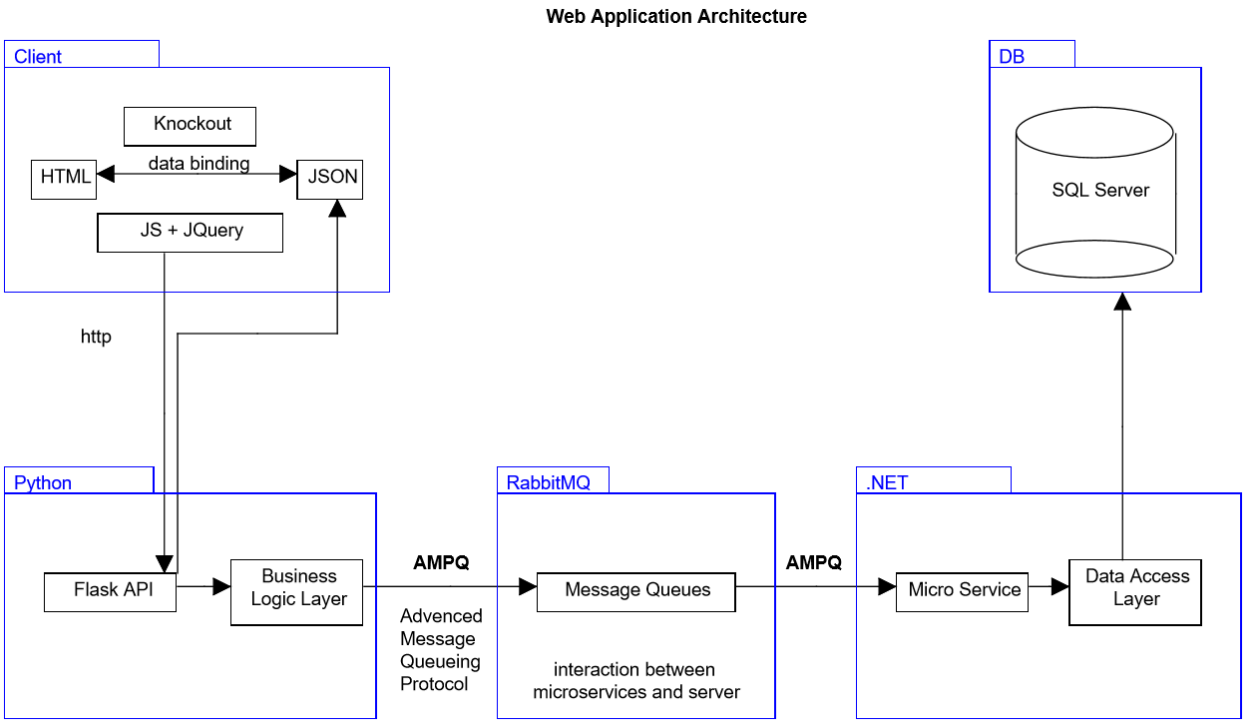
В центрі нашої архітектури стоїть RabbitMQ – система обміну повідомленнями. Детально можливості та приклади роботи з RabbitMQ можна розглянути за наступними посиланнями:

- https://www.rabbitmq.com/
- https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html

RabbitMQ має багато переваг, зокрема – стійкість до помилок - при некоректному завершенні роботи сервера дані в черзі не втрачаються, а зберігаються і після підняття сервера знову готові до використання. Ще одна перевага RabbitMQ - дистрибутивність, адже consumer та producer можуть бути розташовані на різних серверах.

Нижче наведено діаграму архітектури проекту, а також опис усіх її компонент.За допомогою цієї архітектури, написавши логіку на Python ми звертаємось до рівня доступу до даних, що реалізований на .NET.

Діаграма


Web Application Architecture
Client
Knockout
HTML
data binding
JSON
JS + JQuery
http
DB
SQL Server
Python
Flask API
Business Logic Layer
AMPQ
Advenced Message Queueing Protocol
RabbitMQ
Message Queues
interaction between microservices and server
AMPQ
.NET
Micro Service
Data Access Layer

Опис

Name	Description
Client	Frontend part of the functionality. Provides Single Page Application for user to interact with. Used frameworks include knockout, jquery.
Python	Backend part of the functionality. Consists of self-hosted Flask server, provides static content (js, css, html files), manages user session, answers client's ajax requests with necessary information (HTTP protocol), and Business Logic Layer.
Business Logic	Layer, that encapsulates requesting and processing data received from microservices.
AMPQ	Advanced Message Queuing Protocol - a binary protocol based on TCP/IP stack, used by RabbitMQ for communication.
RabbitMQ	A message broker, that is used for Business Logic Layer and Microservices communication. It allows loosely coupling of Business Logic and Data Access Layer.
Message Queues	Means of communication provided by RabbitMQ, are used to pass task requests and return responses. Main advantage - flexible load balancing.
Microservice + Data Access Layer	A .NET application that listens to incoming messages via RabbitMQ and sends requests to MSSQL server using Entity Framework.
Microservice	Layer, that provides hosting and communication with Business Logic Layer via RabbitMQ.
Data Access Layer	Provides creating, connecting and sending requests to the database.
DB	Microsoft SQL Server - a relational database management system

Підсумок – порівняння використаних технологій

.NET Web API 2.0	Python Flask	.NET + Python + RabbitMq (Microservice)
Переваги		
• Одразу ж наявний стек технологій Entity Framework, Identity та ін. • Потужні засоби LINQ для утворення запитів. Аналогічний функціонал python comprehension lists, не надає стільки ж можливостей.	• Простота вивчення. • Низькорівневість та гнучкість. • Використання таких ORM як SQL alchemy за рахунок їх більшої низькорівневості (порівняно з LINQ), дозволяє більше конкретизувати який саме вигляд мають мати запити до бази.	• Гнучкість архітектури. Незалежність шару доступу до бази від серверу. Надає можливість скомбінувати гнучкість Flask та потужні можливості LINQ. • Простота горизонтального масштабування та розподілу навантаження за рахунок мікросервісної архітектури.
Недоліки		
• Недостатня гнучкість архітектури: процедура внесення змін тривала, ти вимагає вищого рівня знань (порівняно з flask зокрема).	• Відсутність одразу ж наявного стеку технологій може бути достатньо сильною перепоною для новачків. • Серіалізація вхідних даних та десеріалізація результатів надзвичайно незручна.	• Сповільнення комунікації між сервером та шаром бази даних через використання протоколу TCP/IP замість прямих викликів процедур. • Великий обсяг коду та складність комунікації між сервером та мікросервісами, що робить архітектуру слабко придатною для невеликих систем.