# Business Requirements

The project is about building crossing deals system:

The deal can be an offer / request.

An Offer: someone post an offer of services

Example: I’m selling a car with price XXX

A Request: some post a request of a services

Example: I need a babysitter with price XXX

Each deal offer of request can have a proposition. So, users of the applications see the deal and make a proposition, the owner of the deal will accept or reject the proposition.

The proposition is based on payment style;

* The payment should be done at the end of the services
* The payment should be done before the services is done
* The payment should be scheduled by and % of the services (for example 30% 30% 30% 10%)

The system on each payment -transaction- the system will block the amount of the transaction and then pay the beneficiary.

The applications will be deployed on a lot of country and region. the globalization should be mandatory on the design time.

All communication should be sent by notification to the users (client) of the system by email.

The invoice and the quotation also.

# Billing Model

Each post should be done with some amount of money defined on each country. With option to be free in some case (specially on the first time).

The crossing system will block in internal the money of each transaction to pay the beneficiary and take a % of the transaction.

The % should de digressive not a fixed one. For example, 100$ 3%, for 1000$ 2%.

This system also helps user to protect the quality of the services in case of a dispute.

The property of the system is a company so there were some taxes to be in consideration on each country

## Features

* Map to see the deal based on coordinate
* Search engine (each deal will be post with a description and tags)
* Image storage engine
* Email sending engine
* Authentication (with Knowing providers)
* Paiement (VISA)
* Monitoring of the system
* Administration Dashboard
* Chat engine (eventually calls-video 2nd version)
* Categorize engine (- the admin can categorize deals) for the list views
* Offline mode with Web assembly and serverless

# Components

4 end user application

For the end users

* Web application
* Desktop Applications (Targeting Windows Tablet to be Deployed in the Windows Store)
* Mobile Applications (all devises Android iOS .. )

For the Admin users

* Monitoring dashboard of the services
* Admin dashboard for statists

# Technical Architecture

The development should be done with .Net al full .Net Microsoft stack and Blazor technologies.

The main Approach is DDD (domain driven developpement ) the busness entity with be shared on the stack.

* Database: SQL Server database with multiple schemas with Entity framework and repository pattern and unit of work.
* The API: is about Blazor Server Application, for the moment the API is hosted in a single application.
* Mobile Applications is About a Native Xamarin hosting a Web Assembly and Electronized

(we need Xamarin to bring the default map of the device) Web assembly to work as stateless and serverless)

* Web , blazor application
* WebAdmin : blazor application

Sample :

Some sample:

<https://github.com/Daddoon/BlazorMobile/>

for the map native to the devices

<https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/user-interface/map/map>

Open street map for the web Part

<https://www.youtube.com/watch?v=qTPHAlHom20>

<https://www.youtube.com/watch?v=cilV8vnEjJE>

* The reuse of the component is mandatory this is why we targeting Blazor

Web Assembly as a server less technologies

The Architecture is micro-services

There is some development has been done

Sample web application:

<https://paris.craigslist.org/>

<https://play.google.com/store/apps/details?id=org.craigslist.CraigslistMobile&hl=fr&gl=US>

There is the API (70% is done) :

<http://pamdeals.com/swagger/index.html>

the payement we use Stipe

using for billing and pushing monyè to the benificiary

Webassembly : servlerless

<https://github.com/stefffdev/NubeSync>

# Design and user interface

At minimum as possible, we don’t need to create a lot of image, the business is most important the end user want a robust application with (not a lot of COLOR).

# Development Approach

The quality o the code is mandatory

The application is designed to use a Domain Driven Design, business classes are mandatory

Unit Test, specially of the payment.

Loging and services monitoring, is mandatory to save to a working database / schema the working services.

Single database to multiple database (for the moment we have multiple schema)

DDD, domain should be the same on all application. we should reuse it from to back to the front.