1 – Topic and objectives of the web application

The objective of this application is to create a copy of Daft.ie, since (…)

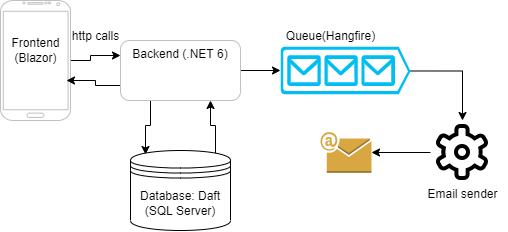
2 – Target audiences

Anyone interested in renting, sharing or buying a property within Ireland or in another country later on.

3 – Competitors web apps

Some competitors of Daft.ie are: Rent.ie, Qwertee, Citylets, Propertyfinder, Homesales, Hemnet service, Footer, HouseWeb and even facebook or Instagram in some sense.

4 – Structural Diagram



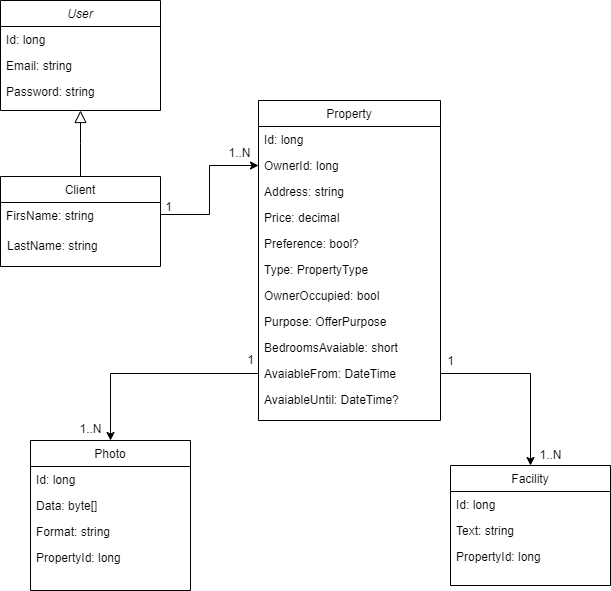
5 – Visual Design

Bootstrap 5, open-iconic, Helvetica Neue, Helvetica, Arial, sans-serif.

Since the bootstrap framework was used, all screen resolutions are supported since bootstrap has responsive layouts which adapts to the screen resolution of the client allowing more comfort and ergonomic for users while using the application.

Colours used: mostly blue and white to keep adherence with the original Daft website

6 – Database ERD



7 – Technologies / Techniques used

For this Project I used DDD (domain design driven principles) to pave the structure of the overall project, using C# as the main language for front-end and back-end. I have used many OOP design principles such as inheritance for controllers and entities, polymorphism, generics and abstractions altogether with SOLID principles to create a simple, easy-to-read, easy-to-understand code base.

For database connection I used EntityFrameworkCore which is the framework created, maintained and recommended by Microsoft to connect, read and update tables to the SQL Server database. I’ve added repository interfaces to abstract behaviour against the database, which means – whatever the database is, I just need to implement the IAsyncRepository<T> interface in another class to keep all the code up and running, which is very handful if I see the need to change the infrastructure database management system.

For the structure / scalability point of view, I’ve also implemented a queue using Hangfire to handle any number of emails sent by users at any time, which makes the system more robust, fault tolerant and scalable since you can add more workers to handle an increasing volume of requests. Hangfire also has a nice dashboard and a web application made for management of the processes running on it and the status of past events, triggers and so on, which also makes the experience really nice.

All the methods to retrieve data from the API are asynchronous so that the API can benefit from the multicore CPUs and users can have a more fluid experience while using the system. Its notable the benefits of asynchronous methods while searching for addresses at the main page of the project. This means that all methods accept a CancellationToken argument which can be used to stop any long-running process, usually an IO action, either reading records from database or writing files to the disk.

8- Maintenance

For future work I’ll be adding unit and integration tests to the codebase so that any further changes will be backed by the tests itself, but for now, as the infrastructure of the project was created with DDD and SOLID principles in mind, it’s easy to maintain and extend.

9 – Search Engine Optimization

I have chosen Blazor for the frontend stack because it supports pre-rendering which means that any static content will be compiled and it’ll generate static files/templates so that it’s possible to publish them and profit from the speed which these pre-compiled templates are loaded, and also because of the static content itself. Blazor also has web assembly support, which removes one layer of abstraction and allows the application to “communicate” directly with the browser which can immensely increase the speed and responsiveness of the application.

10- Strengths & Weakness

Because of the design of the API and Blazor features, I would say that the application is very extensible and dynamic with good degree of freedom to build anything on top of existing code, such as creating new controllers or new endpoints, but also considerably fast and robust, since I used the latest technologies from .NET ecosystem which is being evolving for decades and has a pretty stable environment and structure.

In addition, all queries made to the database are asynchronous and very simple, which means they can be cancelled at anytime and does not consume much resources from the database itself.

One of the weaknesses I came across while developing this application was the necessity of adding a captcha service before adding a user to the system and also before sending emails to clients, since malicious users can use these inputs entry points as a vulnerability to flood the application with useless data, waste resources and also shutdown the application with a DOS (Denial of Service) attack. Deriving from those same principles, for future work, its also necessary to add a policy for making requests to the API such as being able to do only 100 calls per minute to each endpoint, since any authenticated user could start calling the API indefinitely (either by a wrong implementation or by maliciously trying to undermine the application) and also create the same results previously mentioned.

Since my application don’t have unit tests yet, this can be also considered as a weakness for the long run of the system/company, since any future change can insert another bug in the application not previously thought of. This is fundamental if the company is already fully functional and cannot tolerate faults.

11 – List of web browsers used to view the application

Chrome, Microsoft Edge, Firefox, Opera, Apple Safari, Brave, DuckDuckGo

12 – References / Bibliographies

<https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor>

<https://docs.microsoft.com/en-us/dotnet/csharp/?WT.mc_id=dotnet-35129-website>

https://docs.microsoft.com/en-us/aspnet/core/?WT.mc\_id=dotnet-35129-website&view=aspnetcore-6.0

<https://docs.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0>

<https://docs.microsoft.com/en-us/aspnet/core/web-api/?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0>

13 – Addition references

<https://en.wikipedia.org/wiki/Domain-driven_design>

<https://en.wikipedia.org/wiki/SOLID>

<https://en.wikipedia.org/wiki/KISS_principle>

https://getbootstrap.com/docs/5.0/getting-started/introduction/

<https://useiconic.com/>

https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15