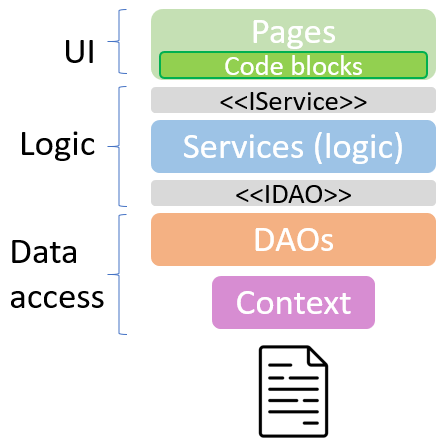
DNP Assignment, Part 2

In this second part, you will add a Web API and Web Clients to contact the former, from Blazor.

Your current architecture, if you followed the directions from the previous part, should look like this:



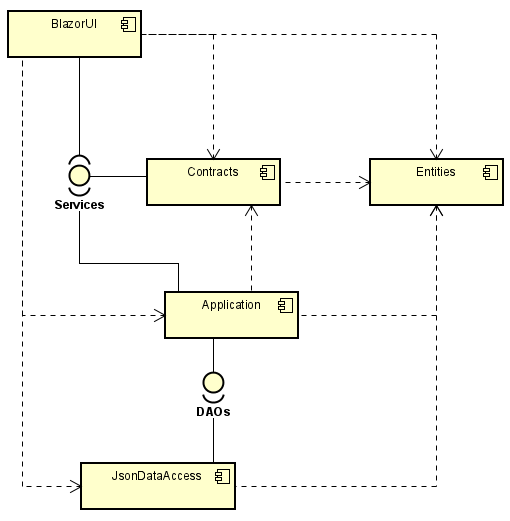
Now, we want to inject a network layer between Logic and UI, so that Logic goes into the server program.

This assignment is in two parts: WebAPI and making a Client layer for the Blazor App.

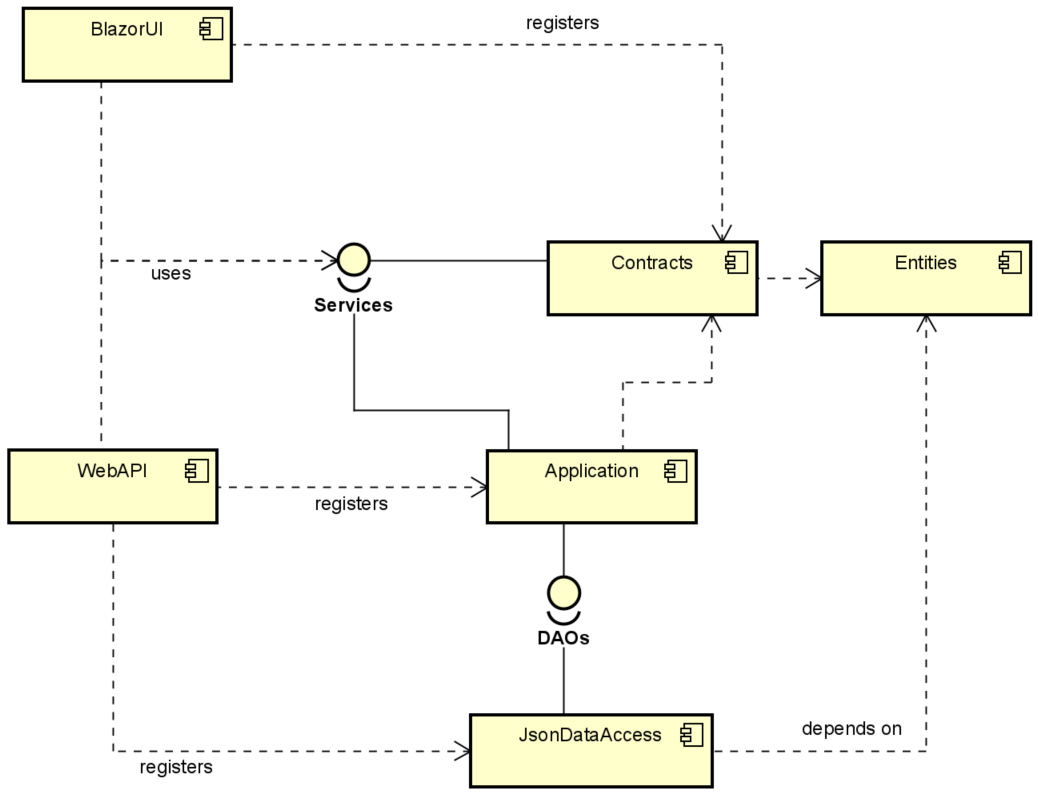
You may develop the two parts in parallel, or any order. But the Web API is described first, as that is the topic to be covered first.

# Web API

This was the component structure of part 1:



We now expand that with two new components, RestClient and WebAPI. The structure will look like this:

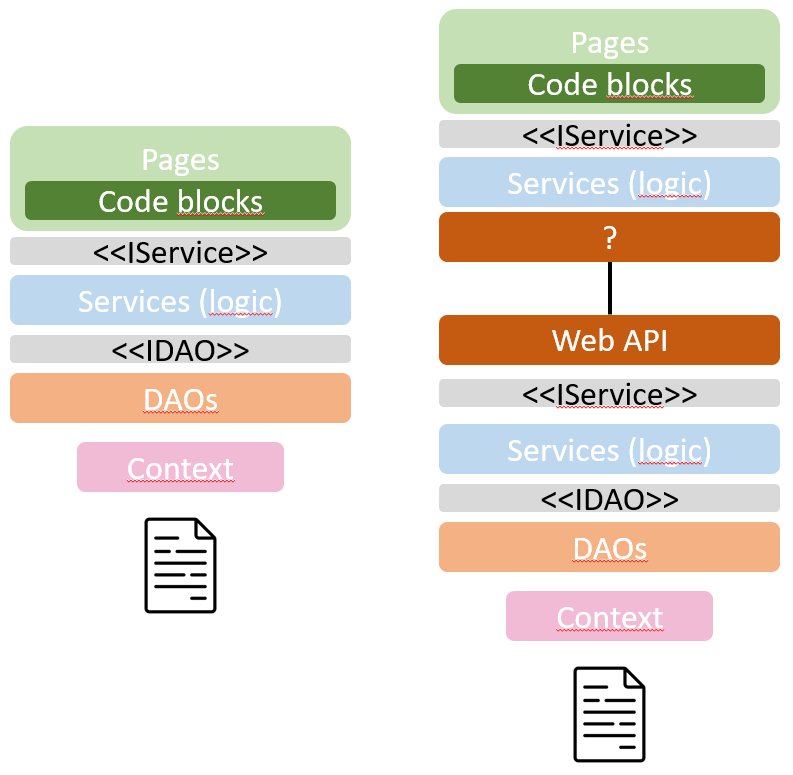


Now, we have a new component: **WebAPI**. It uses the **Service** interfaces from **Contracts**, and through these gets access to the **Application** layer.

There are dependencies to **Application**, and **JsonDataAccess**, because classes and interfaces from these two components must be registered as service in **WebAPI**.

The following layered diagram shows the setup to create. Left side is current setup, right side is next setup, when **Web API** is introduced.

**BlazorUI** and **Application** also knows about **Entities** through a transitive dependency in **Contracts.**   
I.e. because **BlazorUI 🡪 Contracts 🡪 Entities**, then **BlazorUI 🡪 Entities.** Same is true for Application.



The red layer, ”network”, is injected in between. The ”?” means that we have not yet any replacement-classes for the Blazor component to use.

The two interface layers, <<IService>>, are the same interface, from the **Contracts** component.

You must create a new component, a Web API. This component must register the services as seen to the right above, i.e. various logic classes, and DAO classes, and the FileContext.

You must create Controllers, so that the client can create and retrieve the needed data.

Remember good practices here:

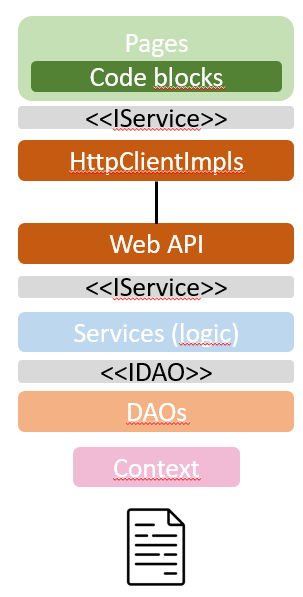
* Create a Controller for each category of object, e.g. Post, Comment, SubForum.
* Follow URI conventions.
  + ”/SubFrom/3/Posts”
    - Making a GET request to this URI will retrieve all posts of SubForum with id 3.
    - Making a POST request will add a new Post to SubForum with id 3.
  + ”Post/312/Comments”
    - Making a GET will retrieve all comments of post 312
    - Making a POST will add a new comment to post 312.

Your WebAPI now uses the Service layer to validate whatever is needed, along with other business logic your app may have.

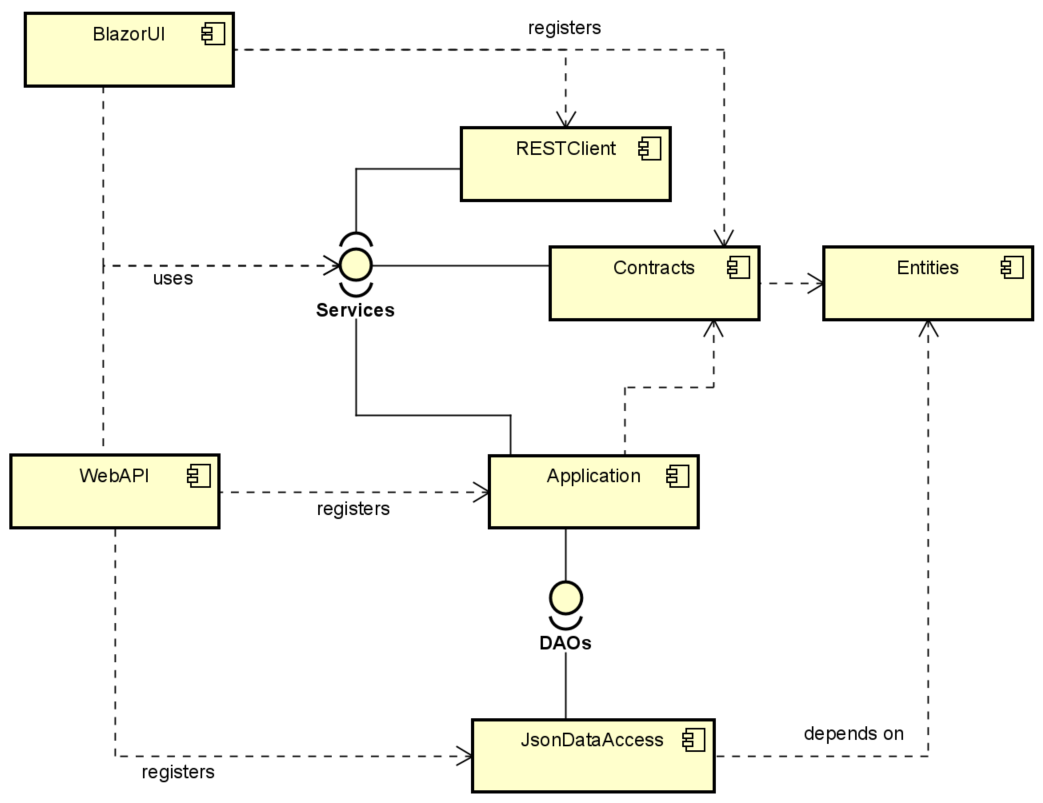
# Http Client

Your Blazor app now needs to be modified, so that it will get data from the Web API instead of from the file storage.

Your layers will look like this:



And your component structure is expanded with a new RESTClient Component, like so:



It’s getting a bit tangled.

**BlazorUI** depends on **Service** interfaces, in **Contract**. These interfaces are implemented by relevant classes in **RESTClient** (and classed in **Application**)

Remember to modify the services registered in the **BlazorUI.Program.cs** class. You are already registering interfaces of IServiceSomething. Now you just need to modify which implementation is registered along with the interface.

This change makes the Blazor app get data through your REST clients, instead of from the file storage (”locally”).

The classes in **RESTClient** must make calls to your Web API to send/retrieve whatever data your Blazor app needs.

# Formalities

You may work in groups on this assignment.

This assignment must be handed in to be allowed to attend the exam.

Deadline will be posted on itslearning.

# What to hand in

A link to github where you have your code.

An updated class diagram, in .svg file format.