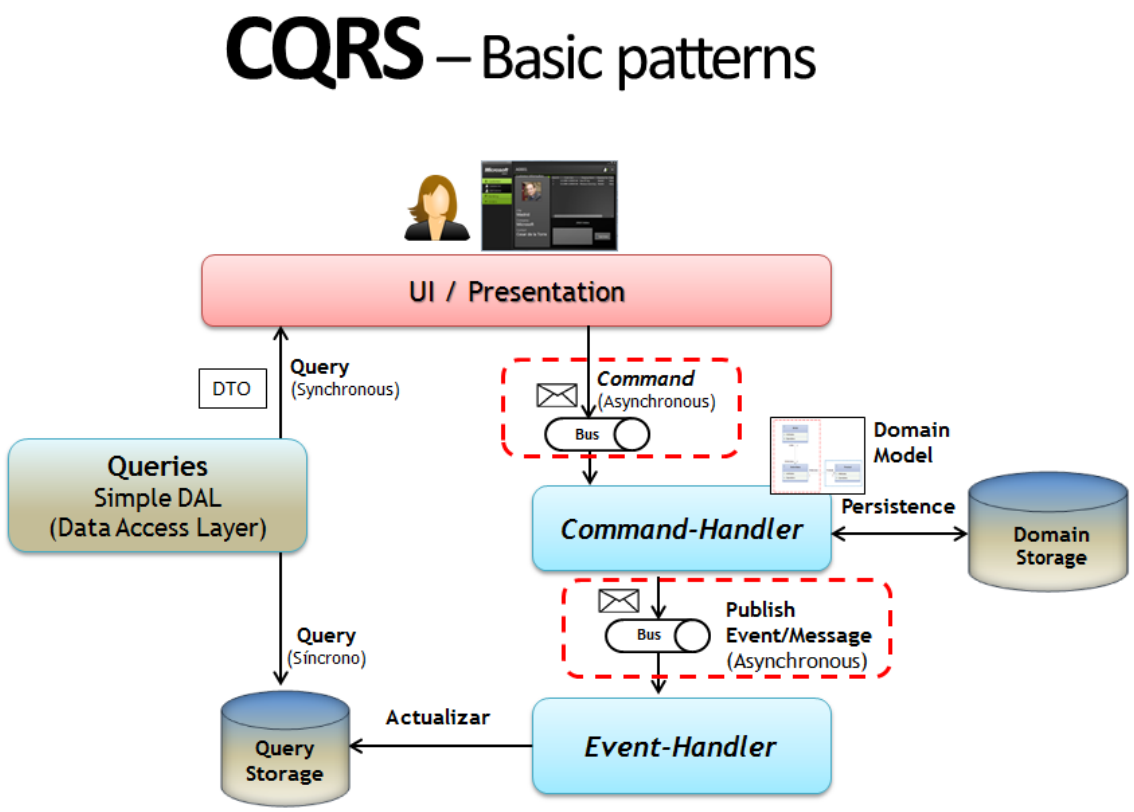
Design of the Architecture

The decided architecture was based on scalability and separation of concern. Even thought this is an architectural project to show front-end skills, the back-end took a major role in the development.

I dediced to build the backend based complety on the concepto f CQRS, since it would be such a large step for one small Project this was scalled down to a CQS Architecture, without event source.

CQS stands for Command Query Separation, where all the code is based on the premises that querys never update, insert or delete anything, it is whole purpose is to query data from the database. The Commands on the other hand, updates information, but do not return any value to the user otherwise by the ID. Even the ID could not be returned if the Keys were based on a GUID so it could be created beforehand. This options was not used but the concept is still valid anyways.

Below is an image of what is an CQRS Architecture, the difference for the CQS is the EventHandler:



To Keep everything consistente with the architecture 5 layers had to be created in the Backend, and they are:

Crossover.Api – Here all the API Controllers are hosted, the OAUTH server was also set in this Project instead of a separate one to keep the example simple. The Authorization and Global Exception handler is all in this module.

The Hub should also be here, but because of the time-limit I was not able to put it to work with CORS Enabled, so even though there is code showing how it would be, it is not working or integrated with the frontend. Hubs here are the code necessary to give real-time updates for connected users.

Crossover.Domain – All Entities are mapped here with poco classes. Here is the layer that makes the interface with the database.

Crossover.Commands – Here is where all logic to Insert, Update or Delete na Entitie is available. A Command is made of two parts, a Command (Data that is sent from the user) and a CommandHandler (a logical/procesed that). All updates, inserts and deletes are done in the commandHandler. This aprouch to use CQS makes or Controllers very thin and don´t require any tests. All unit tests in the server should be done in this layer. (No tests for backend was created, since it is a front-end exam. But if they where added this is where the logical part of the application would be and requiring tests)

Crossover.Queries – Here is where all the logic of queries are in the application. A Query is made of 3 parts: a Query (Data that is sent from the user), a QueryHandler (The controller responsible for the logic to get this information from the database to the user) and a QueryResult (Classes that shows exact what information each query allows the user to see and use.

Crossover.DependencyInjection - Here is where ao IOT is done. It is responsible to inject all components and layers in other layers as requested by the application. To keep complety separation of concern this layer had to be created to integrate all parts.

Crossover.Util – All tools that might be need for the query layer or commandlayer would stay here, such as PasswordHash. Only tool that this application needed.

The Backend were based on OWIN and WebApi on top of that an Authentication and Authorization Server was built. To keep this application compatible with front-end and mobile na OAUTH server was created to authenticated the users and a JWT token is generated with all information that is need. This way the api can be consumed by any device or application, is very secure and can work with third parties for integration.

To Dispach message across the CQS pattern I used the library MediatR as the core of the pub/sub events. AutoMapper was also a library used to map objects from Commands to Domain and Domain to Queries.

The Front-end was entirely build with AngularJS 1.4.x.

The Project is based on 3 tables (User, Topic and Comment)

The User table hold all user information with their encrypted password. Even if two people choose the same password it will have diferente hashes in the database, making it really secure.

The Topic is where all questions can be stored in the application.

Comment is where all replies to those questions are stored in the application.

To view the topics do not require an authenticated User. But the read it complety and answer do require na authenticated user.

Front-end

For the front-end the AngularJS was the chosen framework. All code is in modules inside the folder /app. There is no link with any .NET dlls in this Project **(Crossover.Frontend**). Is is build complete separate and can be run on any server, be it na IIS Server or Node.JS server. The only thing to run is the index.htlm file.

All code is in modules where are divided in Directives, Filters, Controllers, Views, Tests, Services. The Service Modules can be further organized in Factories, Service, Providers, Values.

External libs are managed by the bower application and is included in the /app/libs folder.

All code used in the application and external libraries are managed with the task runner Grunt. It is responsible to inject everything needed in the index.html file. It i also resposible to compress, minimize assets (this option was inclued in grunt, but the code is not doing it, so the code can be easily read and debugged online if needed), on a real production application the configuration to do it is all done in this grunt config file.

The code follow the Style Guide of the Framework from JohnPapa and Todd Motto. All Javascript code is encapsulated, there is no gloval variables. Eveythings is modular and any part of the application can be tested separated.

Ui-router was used to manage the states of the application.

All resources are managed with a single service, AppResourceFactory. And the resources in the service are build from this to make the calls and validate data. This resource is also responsible to emit events or integrate call-backs where desired to get notified by events (real-time should be integrated here but I failed to do so because of the time limit and the use of CORS)

An authentication module and authorization module was also built in the front-end. It is responsible to deny access to the routes that the user do not have access like post Topics or Replys and pop-up for authentication. Everytime a route or resource requires a level of permission the user do not have, he is prompt to inform new credentions to continue. Failed to do so, deny access to the resource and route. If he gives the required information the route and resources are resolved again and shown for the user.

I18 was not used in this application, since it was not part the requirements to show internationalization.

Many directives was build to integrate data and information toguether. Part of the code was also built without directive like listTopics, so i could show complete skills in diffrente part. ListComments is already a directive. Objective here was to show how the code could be organized as the cliente demands, how well it is written, easy to read and follow etc.

All code is working really well toguether and I do feel it is a very good example of a well build application. The time limit to build it was really short, because of the requirements to build the backend of the solution, database etc. I do feel the test should have been based on open Apis, like integrating with Git to do stuff etc.

The initial time-frame was 3 days and after the test started it was 7 days. Don´t know why it has been changed, but I used 5 days to complete the work. All work done here is based on 8 to 10 hours a day for 5 days.

I had a problem with the video, I got carried away and it got bigger then 30 Mb, since the time to deliver the Project do not allow me to record again I will send it later if it is still possible.

I had some difficult to set the software online on a free hub, since it is in .NET. I am working now with appHarbor and the link of the DEMO will be sent also later today when everything is deployed and working online. Eveything can be run using IIS Express.

I enjoyed building the application and any clarification can and will be done in the next step of the evaluation.

Best Regards,

Campos, Getulio