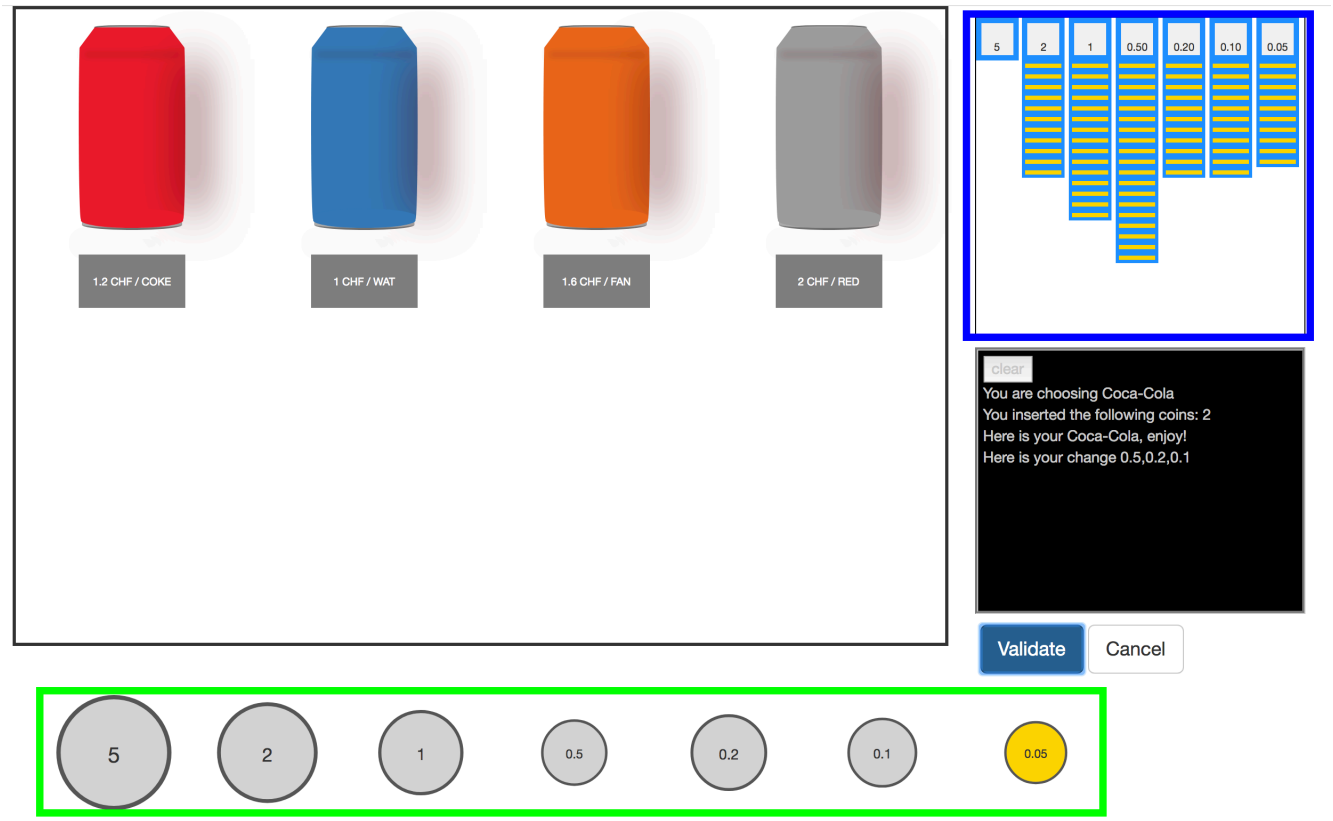
DRINK VENDING MACHINE

This application is a practical exercise, representing a vending machine on a web application. The purpose of this exercise is to use Angular and the .NET Core framework.

TABLE OF CONTENTS

FUNCTIONING AND REPRESENTATION

The vending machine is represented in a simple way:



The black shape is representing the window or the screen you will have on a real vending machine. The cans represent the different drink available in the machine. They are piled up by kind, and there is no just 1 x Coca Cola, 1 x Water, 1 x Fanta and 1 x RedBull, but you have to imagine that there are m x Coca Cola, n x Water, p x Fanta and q x RedBull. m, n, p, q being the stock you will have in the machine. Here, it is also used to select the drink you want to buy, by clicking on it.

The blue shape represents the coins stored in the machine, the yellow line being the different coins. It is a side view representation of an object that would store coins.

Just below the coin storage, there is a screen control, where you can read the instructions of the vending machine, with two buttons, one two validate the order, and the other to cancel it.

Finally, the green shape represents the money the client can insert in the vending machine. You can think about it as a wallet, but it is an unlimited wallet, because the user can use the coins indefinitely. The purpose here was to handle a vending machine, not a wallet. To insert a coin, just click on it.

When ordering a drink, the machine is going to tell you when it delivers it, and “gives” you your change if you have any, by telling you on the screen, what it gives you back. (It is the same thing for the drink.) The coin storage will be updated, and the Windows part as well.

APPLICATION DESIGN

Technical choices

This application was developed with a front end in Angular 4, a back end in .NET CORE, with Entity framework ORM, using a SQLite database.

I chose .NET CORE because I am a MAC user in the private life, and .NET CORE is a cross-platform framework, so the app works on Windows without any additional development. I also like open-source technology, so when I can use it, I try to.

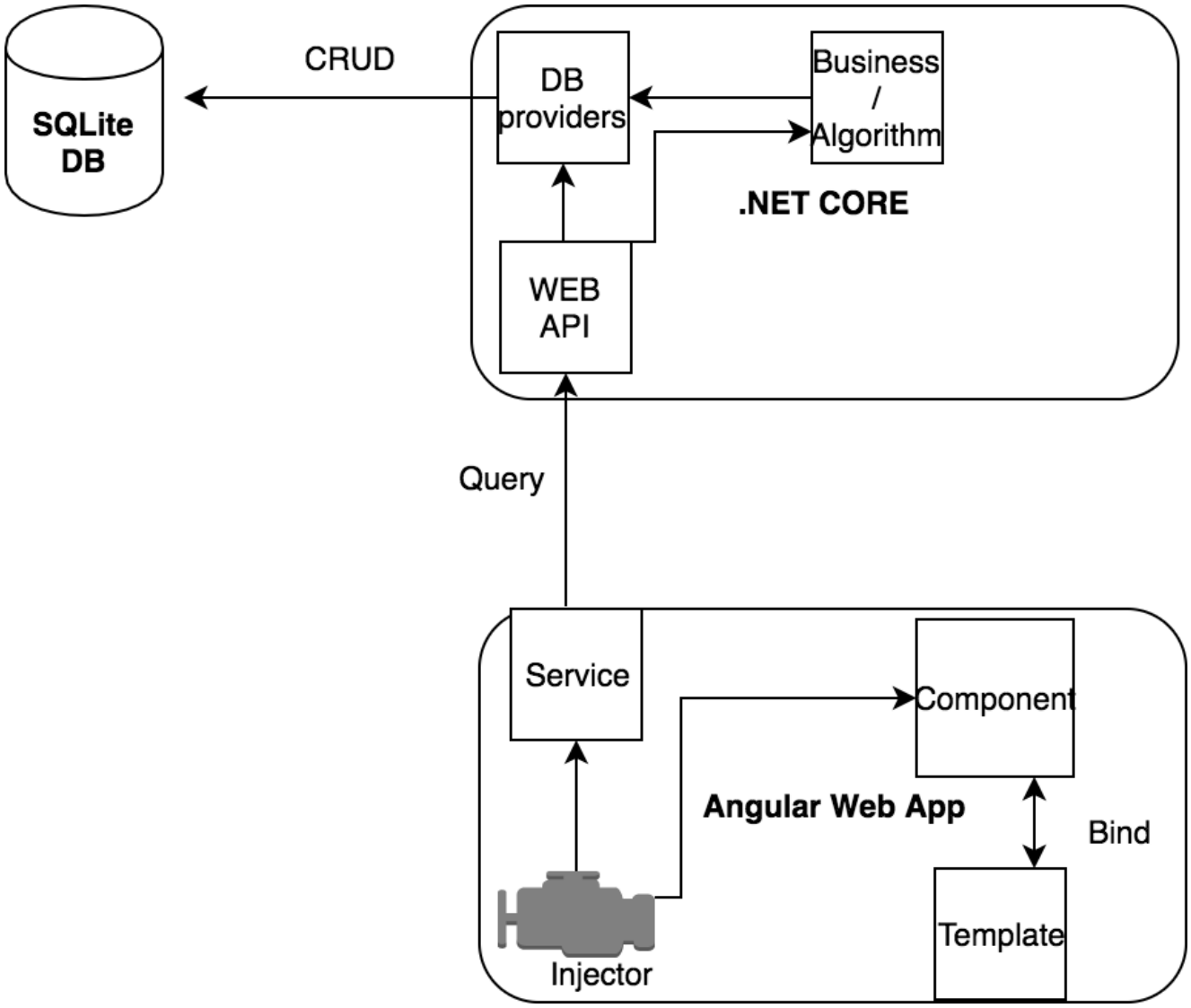
For the front, Angular was suggested for the exercise, because it is the framework becoming the most popular for Web apps. And I have to admit it is very convenient: how things are separated (Template, Style, Class) in components, how data is bind between the View and the controller is also very efficient for the developer.

Finally, I chose to use SQLite, because for this application, there is no much data to persist, the data model is pretty simple, and because it is very convenient for deployment: the file being created by the ORM, with some default data.

The template for the project, is a Visual Studio template. I did change the Angular version to use a more recent version, because some component I used are more convenient in their new version.

Application Architecture

This diagram represents the architecture of the application.



The Angular Application works with components. Components could be seen as entity with a controller, a template and eventually a style. For example, in this app the dispenser (vending machine) is a component, which includes other components, such as the coin store, the screen control, drink cans and a coin drawer. A menu could be also a component, it is not necessary something visible. Templates are component's graphical representation. Data is

exchange between controllers and templates thanks to the binding. From the template to the controller (a user input), from the controller to the template (data displayed).

In the app, we also use services. For instance, a service is responsible to print message in the screen control and others are requesting data from the Web API. Thanks to the platform, there is dependency injection which instantiates the services, seen as “providers” and allows us to use them in the components.

As we have seen, some of the angular services are making calls to the back end. The .NET app exposes things through a WEB API / Web Service REST. It has also a layer responsible for the business: here the **recursive algorithm** handles the change, which is given back to the client is in the business part. The data is stored in a SQLite database and to create, read, update, delete data, a layer composed of providers is used. Dependency injection is also used in the back end, it is provided by the framework.

One more thing that is not represented on the diagram is the fact that the Angular App is being delivered by the .NET, to the web client. It could have been Node JS also, but for simplicity I kept what was in the project template.

The API could also be separated from the .NET application, to improve flexibility, but here, the project doesn’t have enough content to be worth it.

Testing

Unit tests have been added on the business layer. There is a Visual project in the solution for this purpose. The Web API has been tested with Postman, which is a tool for sending many kinds of request. Functional tests have been made during the development.

Data model

I did not mention the data model in the documentation, because there are only 3 simple tables.

INSTALLATION

It is possible to **download** the project on GitHub, on the following repository: https://github.com/sebreceveur/DrinkVendingMachine

After having it downloaded / cloned, it has to be open with Visual studio. A recent version would be preferable. To run it, it must be **built**: it might take some time at first, because the IDE will look for the dependencies and there are many of them.

The project is set to run with IIS Express, which is a standalone version of the web server IIS. So, when hitting **Run,** it will be launched directly without having to care about its deployment.

Database

The database is automatically generated when the project is built. It is a file named “VendingMachine.db” at the root of the project. A procedure adds data at the first start.

Thanks to DB Browser for SQLite we can explore the data of VendingMachine.db.

IMPROVEMENTS

- Obviously, the style could be improved. It was not the main subject of this exercise, so I kept it simple.
- Administrations functionality could be added, to change the drinks, add or remove coins in the machine.