**Technical Report – Fragrance store**

**Blaga Bogdan-Csaba**

Table of Contents

[1 Introduction 2](#_Toc152535123)

[2 Presentation of the applications 3](#_Toc152535124)

[2.1 Web App 3](#_Toc152535125)

[2.2 Mobile App 7](#_Toc152535126)

[3. Conclusion 13](#_Toc152535127)

# 1 Introduction

The primary objective of this technical report is to explain the functionality of the "Fragrance Store Web" and "Fragrance Store Mobile" applications. At the same time, it will also present how the communication between them is carried out.

This study is intended for technical people with programming knowledge. Each aspect will be presented in depth so that both applications are fully understood.

# 2 Presentation of the applications

## 2.1 Web App

The web application was developed in the AL programming language. AL is used to manipulate data, such as inserting, modifying and deleting records from a Dynamics 365 Business Central database. It controls the execution of various application objects such as pages, reports or code units.

Dynamics 365 Business Central is a resource planning system for small and medium-sized businesses that automates and streamlines business processes and helps manage them. It's highly adaptable and you can easily add functionality relevant to your region of operation. Business Central is quick to deploy, easy to configure, and simplicity guides innovation in product design, development, deployment and use.

The web application is the server side of the project.

In developing a Business Central project, AL code is written in objects. The existing object types are: table, page, report, codeunit, query, XMLPort, profile, add-in control, table extension, page extension, page customization, enum, enum extension and interface.

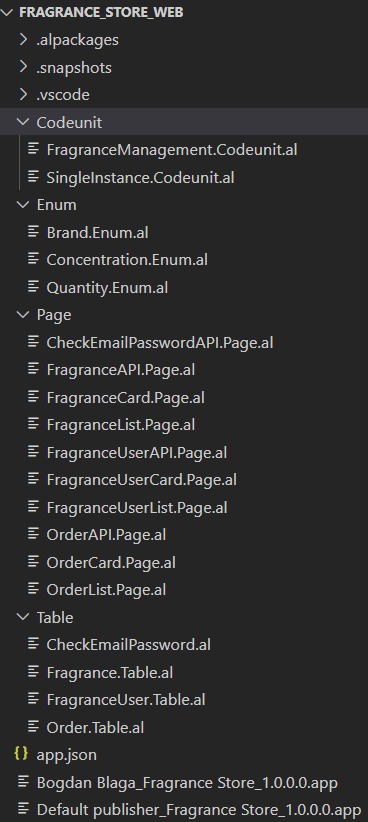
At the same time, due to its structure, AL allows the creation of API pages. Thus, data can be easily transmitted from/to external applications. To use API pages, you need to log in to Business Central with a username and password. In most cases, Business Central will be installed with Windows authentication. To access the application with a username and password you need to follow a few steps:

* Go to the "User Card" page in Business Central and add a user with „SUPER[[1]](#footnote-1)” rights, and set a user name and password.
* A self-signed SSL[[2]](#footnote-2) certificate will be created. An SSL certificate is a digital certificate that authenticates the identity of a website and enables an encrypted connection. SSL is a security protocol that creates an encrypted connection between a web server and a web browser. This ensures the security of Internet connections and prevents criminals from reading or modifying information transferred between two systems. To do this, we used the script "GenerateSSCertificate.ps1" which can be downloaded from: <https://www.navuser.com/create-self-signed-certificate-using-windows-powershell-ise/>.
* The "Permission to Service Account" certificate will be updated.
* For the instance used, update the "Certificate Thumbprint" field with the fingerprint of the certificate resulting from running the above script, the "Credential Type" field with the value "NavUserPassword" and the "Enable Certificate Validation" field with "true".
* Update the web server configuration file from "Windows" to "NavUserPassword".
* Access via IIS the website for the modern client and map the SSL certificate.

A Business Central project needs several components to work:

* **A database:** must be SQL Server or Azure SQL
* **IIS:** with the following features enabled: HTTP enable, .NET Extensibility 4.5, .NET Extensibility 4.5, .NET Extensibility 4.6, ASP.NET 4.5, ASP.NET 4.6 or ASP.NET 4.7 (depending on Windows version), ISAPI extensions, ISAPI filters, request filtering, Windows authentication, default document, directory navigation, HTTP errors, static content
* **Cmd, powershell:** or a GUI tool for instance management, such as "ServiceTierAdministration"
* **Visual Studio:** with "AL Language" extension installed (for developers only)

The project has been structured as follows:



Tables are objects used to store data. Regardless of how the data is recorded, from manually entering it into the application to using a web service, the results of that transaction will be recorded in a table. Tables cannot be accessed from the GUI[[3]](#footnote-3) and the data in them cannot be modified by the user. A table created using the AL programming language will also be automatically created in the database.

Pages will be used to access the information in the database. Thus, the user can read, create, update and delete data.

The following fragrance information will be saved in the Fragrance table: ID, name, brand, image, price, concentration, quantity, price per quantity, base notes, middle notes, top notes. The "Price per quantity" field is uneditable and automatically calculates the perfume price per 100 ml when one of the following fields is updated: "Price", "Quantity". For the calculation, the simple rule of three was used, based on quantity and price.

Primary keys from all existing tables in the web application are clustered. Thus, they determine the order in which the data is stored in the table and at the same time improve the access time to the data.

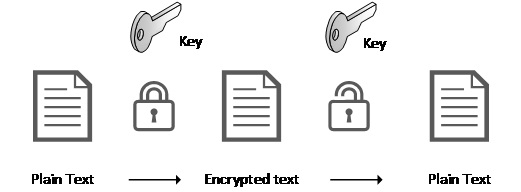
The "Fragrance User" table will store the data of the mobile app users: email, first name, last name, address, password. In addition to these, there is another field that cannot be accessed by users: "Dummy String". This field will always be empty, but it has an important role: when a user is registered, the web service will use this field to transmit the unencrypted password. When the data arrives in the web application, before inserting it in the database, the value in the "Dummy String" will be encrypted and saved in the "Password" field as BLOB[[4]](#footnote-4). The value in the Dummy String will then be automatically deleted. Note that in AL, a Text variable can hold a maximum of 2048 characters, and encrypting a password often exceeds this value. Furthermore, even encrypted, the password must not be visible to users. This is why the BLOB was chosen.

Password encryption is done as follows: First, check whether encryption is enabled on the server running the application for the database being used and whether an encryption key exists.

Encryption keys are stored in a secure location and are accessed at runtime when needed. Additional functionality is provided to export and import keys, which is important when moving solutions from one location to another.

The data encryption and decryption process requires a key. An encryption key is usually a random string of bits generated specifically to encrypt and decrypt data. Encryption keys are created using algorithms designed to ensure that each key is unique and unpredictable. The keys that are used by Dynamics 365 Business Central are generated by the .NET Framework Data Protection API.

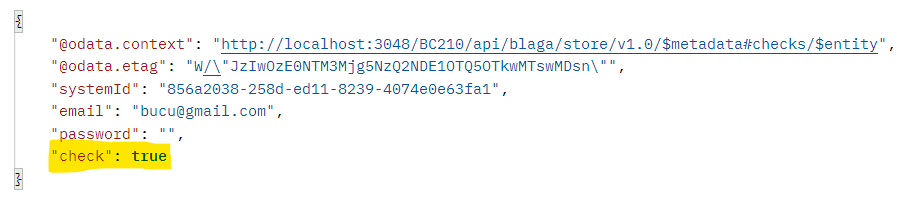
The encryption key is stored in a file in a directory that Dynamics 365 Business Central has access to. When a key is created or imported, the data is recorded in a system table that records that encryption has been enabled.



The resulting text after encryption will be written to a stream created from the "Password" field.

In order to verify that a user trying to connect to the mobile app is using valid credentials, a web service will be sent to the web app with the following information: email address and password. The web service will then determine if the combination is valid and return "true" or "false" otherwise via the "check" field.

To determine if the combination is valid, it will search the database for the email address, and decrypt the password found as follows: a stream will be created from the "Password" field, which will be saved in a dotnet variable, to escape the limitations imposed by Business Central. That stream will then be decrypted and compared with the password received via the web service.



## 2.2 Mobile App

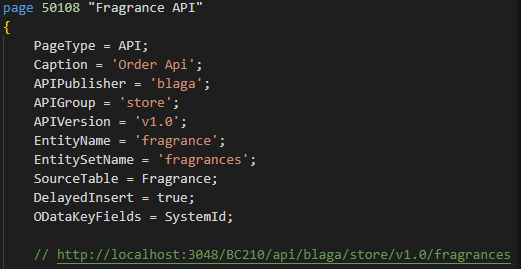
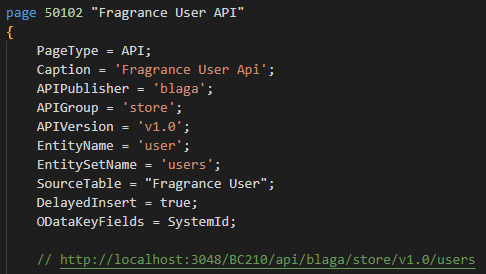
The mobile app was developed using the Java programming language in Android Studio. This is the client side of the project.

Initially, the user is greeted with the opening page, from where they can choose to log into the app or register if they do not already have an account.

For the frontend, the "Constraint Layout" and "Linear Layout" from Android Studio were used. All classes developed in the app were added to the "AndroidManifest.xml" page to make them accessible. At the same time, in order to be able to use the web services, the application was given access to the internet.

All requests to the web application are formatted as follows: „http:// AddressIPv4/InstanceNameForBusinessCentral/api/APIPublisher/APIGroup/APIVersion/EntitySetName”:

* **Address IPv4:** can be found by opening cmd and running the command: ipconfig
* **Instance Name for Business Central:** is the value of "serverInstance" set in the "launch.json" file in the web application
* **api:** will always appear in applications, unmodified
* **API Publisher:** is set in the API page to which the request is to be made
* **API Group:** is set in the API page to which the request is to be made
* **API Version:** is set in the API page to which the request is to be made
* **Entity Set Name:** is set in the API page to which the request is to be made

****

On the registration page the user will fill in some information about him/herself, such as: name, surname, address, email and password. On the login page, you will only need your email and password. For both pages, the behaviour is the same: When the "Sign up" or "Sign in" button is clicked, it is checked that all fields are filled in and that the password is at least 6 characters long. A JSON object is then created with the data in it. The JSON is then sent to the web application via a POST request. Based on the data sent, the web app sends a response which is checked in the mobile app as follows:

* **Registration Page:** check that the web service call returns code 201 "Created"
* **Login Page:** check that the web service call returns code 201 "Created" and that the value of the "check" field is **true**

If the credentials entered are correct, the application's main page will open. There are three buttons on this page:

**Search:**

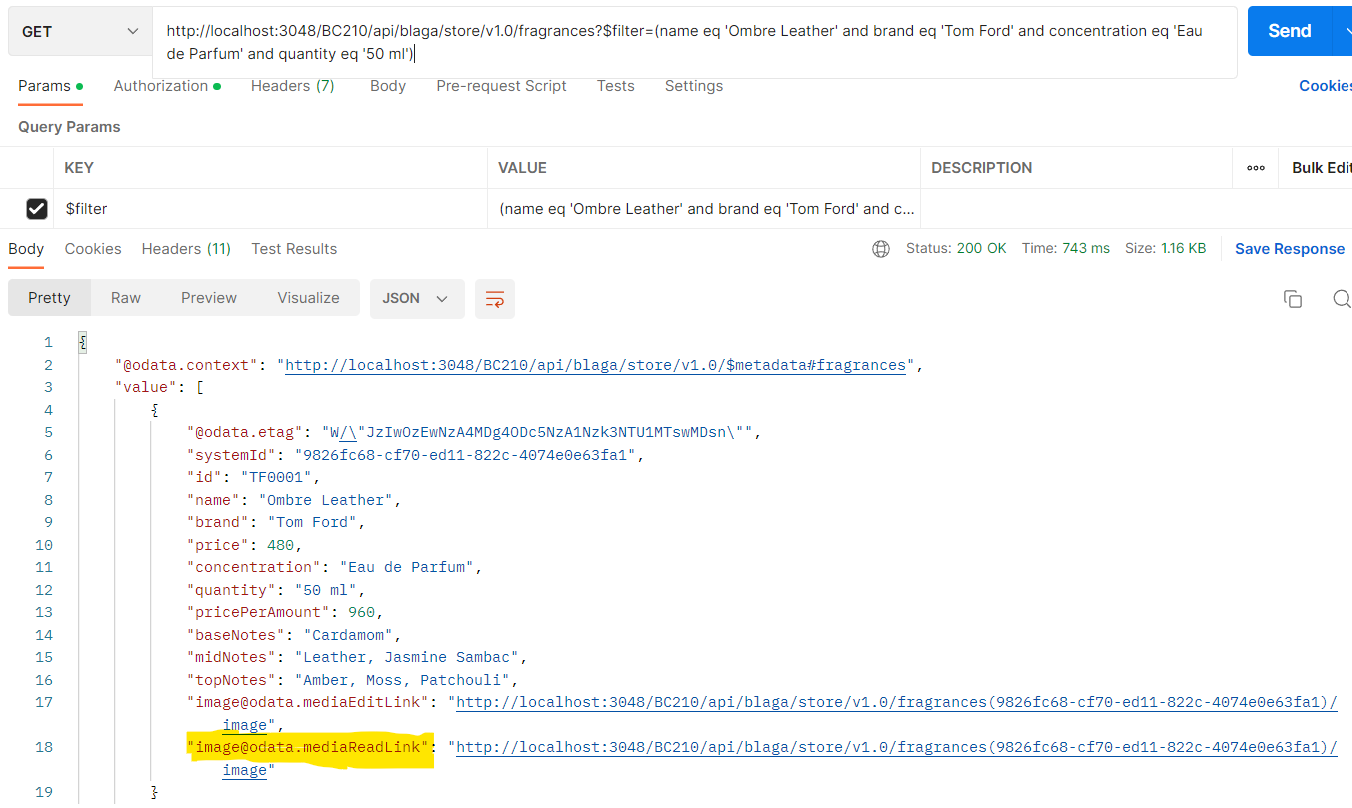
It will send a GET request filtered according to the brand, name, concentration and quantity of the fragrance desired by the user, as follows:

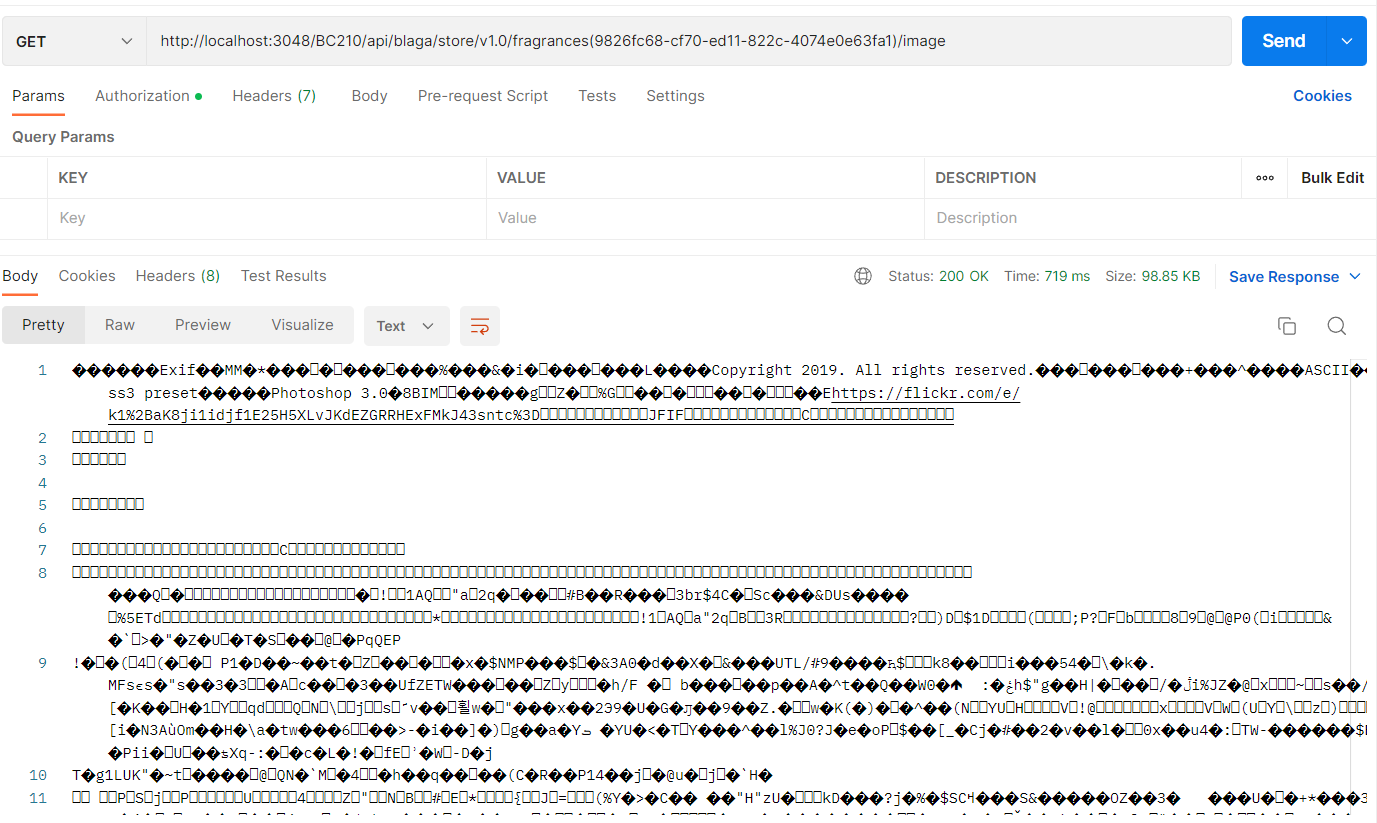
* the data entered by the user in the TextBoxes on the page will be retrieved
* will be added to the request as "?$filter=(PropertyName eq 'UserEnteredValue')". An example of a filter that can be added to a request is: "?$filter=(name eq 'Ombre Leather' and brand eq 'Tom Ford' and concentration eq 'Eau de Parfum' and quantity eq '50 ml')

If the fragrance is not found in the web application database, a message describing this will be returned. If the scent exists in the database, a new page will be opened, where the details resulting from the GET request will be extracted from the JSON and displayed to the user.

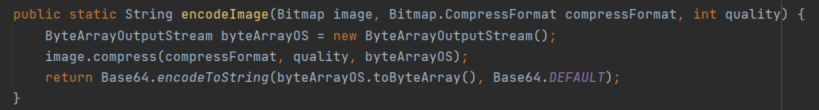
Passing them from one page to another is done by creating a bundle. Throughout the application, when moving from one page to another, at least one property will be added to the bundle, the logged in user's email. Thus, regardless of the page, the results displayed will only be those for the logged in user. In this case more details will be sent in the bundle, such as: perfume name, brand, price, etc.

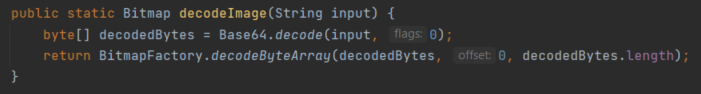
To get the perfume image as a bitmap, a new GET request was sent to the address received in the first GET request sent:





Because a bitmap takes up a lot of memory, in order to send the image from one page to another, it is converted as a String to base 64, and then converted again as a bitmap on the displayed page.

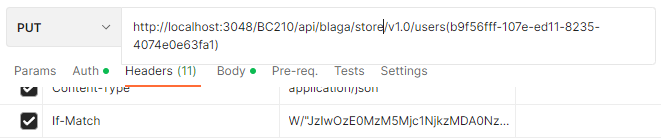




**Update user details:**

It will open a new page, where the user can change their name, surname, and address. This is done by sending a GET request, followed by a PUT request to the web application as follows:

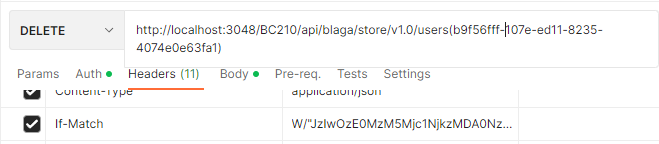
* Initially, when the user clicks on the "Update user details" button, a GET request will be sent to the web application to find out the values of two properties: " @odata.etag" and " systemId".
* If the request response is 200, a bundle with the user's email address and the values of the two properties in the previous step is created and sent to the user data update page.
* When you press the "Save" button, a JSON consisting of the completed TextBoxes from the new page is created, which is then sent via a PUT request to the web application. However, the request is different this time. An "If-Match" header will be added, and the request will contain the value from systemId: "http:// AddressIPv4/InstanceNameForBusinessCentral/api/APIPublisher/APIGroup/APIVersion/EntitySetName(systemId)".
* If the answer is 200, the user is notified by a message that the update was successful and is redirected to the main page.



**Delete account:**

It will delete the user's account. This is done by sending a GET request followed by a DELETE request to the web application as follows:

* Initially, when the user clicks on the "Update user details" button, a GET request will be sent to the web application to find out the values of two properties: " @odata.etag" and " systemId".
* If the request response is 200, a DELETE request is sent to the web application. The way the request is constructed is similar to the one shown for PUT.
* If the answer is 204, the user is notified by a message that the deletion was successful and is redirected to the home page of the application.



# 3. Conclusion

The Fragrance Store Web and Fragrance Store Mobile apps are designed to work together. However, the web app can also be used as a stand-alone app, independent of the mobile app. Instead, for the mobile app to work properly, the web app must be running.

1. The SUPER right in Business Central gives the user the ability to do anything in the application. This right contains all other existing rights. It can be compared to the ADMIN right in other applications. [↑](#footnote-ref-1)
2. SSL - short for "Secure Sockets Layer" [↑](#footnote-ref-2)
3. GUI – abreviere pentru: „Graphical User Interface” [↑](#footnote-ref-3)
4. BLOB – abreviere pentru: „Binary Large Object” [↑](#footnote-ref-4)