

Business Manager - C# .NET Core and Angular Project

Francesco Bigi
EPHEC - Informatique de Gestion

CONTENTS

I	Back-End	3
A	Business Manager API	3
	A.1 Endpoints	3
	A.2 Logic	4
B	Authentication API	7
	B.1 Endpoints	7
	B.2 Logic	7
C	Common Library	8
II	Front-End	8
A	Angular	8
	A.1 Authentication	8
	A.2 Navigation Bar	9
	A.3 Business Pages	9
	A.4 User Pages	12
III	References	14

LIST OF FIGURES

1	Business Manager API	3
2	Authentication API	7
3	Login Page	9
4	Navigation Bar (using the Admin)	9
5	Business Create	10
6	Business Overview (using the Admin)	11
7	Business Details (using the Admin)	12
8	User Overview for the Admin	13
9	User Details	13

I. BACK-END

A. Business Manager API

As you can see in the below image, swagger generate the methodes that I have created in the class **BusinessDataController.cs** which is situated inside the project **business_manager_api**

A.1 Endpoints

BusinessData	
GET	/business/ping
GET	/business/types
GET	/business/days
GET	/business List of the businesses created
POST	/business Create a Business
GET	/business/search Search the specific Business Based by different parameters such as type, country, city, open hours
GET	/business/{id} Search the business by ID
PUT	/business/{id} Update the business by ID
DELETE	/business/{id} Delete a business by ID
GET	/business/page
GET	/business/{id}/enable
GET	/business/{id}/disable
POST	/business/force
POST	/business/{id}/logo
PUT	/business/{id}/logo
POST	/business/{id}/photo/{imageId}
PUT	/business/{id}/photo/{imageId}

Figure 1: Business Manager API

Figure 1 shows the endpoints:

- GET **/business/types/** is to retrieve the list of business types stored in an Enum Set.
- GET **/business/day** is to retrieve the days of the week, stored in Date type.

- GET **/business/** is to retrieve the list of the business created.
- POST **/business** allows to create a new business.
- GET **/business/search** is to retrieve the specific business categorized by different parameters, such as type, country, city, open hours.
- GET **/business/id** is to retrieve the business by ID.
- PUT **/business/id** is to update the business by ID.
- DELETE **/business/id** is to delete the business by ID.
- GET **/business/page** is to retrieve the page of a specific business
- GET **/business/id/enable** is to enable a business in case is blocked
- GET **/business/id/disable** is to disable a business.
- POST **/business/force** allows to save without validating
- POST **/business/logo** allows to save the logo path
- PUT **/business/id/logo** allows to update the logo path
- POST **/business/id/photo/imageId** allows to save the image path
- PUT **/business/id/photo/imageId** allows to update the image path

A.2 Logic

Queries: Queries are used to fetch a business based on for example, it's city, country or owner. It is also shown below, an example of a more complex query to fetch only businesses that are open at the time of the request. These code snippets can be found in **BusinessDataController**

Listing 1: Simple Query

```
1 set = set.Where(b => string.Equals(b.BusinessInfo.Address.Country, country, ...
   StringComparison.OrdinalIgnoreCase));
2 }
```

Listing 2: Simple Query

```
1 var today = DateTime.Now;
2 var day = today.DayOfWeek.ToString().ToUpper();
3 var hour = today.Hour;
4 var minute = today.Minute;
5 set = set.Where(b => b.WorkHours
6     .Any(a => (
7         !a.Closed
8         && a.Day == day
9         && (a.HourFrom + (float)a.MinuteFrom / 60) ≤ (hour + (float)minute / 60)
10        && (a.HourTo + (float)a.MinuteTo / 60) ≥ (hour + (float)minute / 60)
```

```

11     ));
12 }

```

User Role/Email Check: The **GetClaim** method is used mainly to extract the user **role** or **email** from the **access token**. The extracted role is used check if the user is an **ADMIN**, who can perform any CRUD operation on any business. The extracted email is used to check whether the business belongs to the user who is trying to for example, to **edit** or **disable** is the owner of that business. The method returns **NONE** if the access token doesn't contain the requested claim.

Listing 3: Getting a claim from the **access token**

```

1 private string GetClaim(string name)
2 {
3     var accessTokenString = Request.Headers[HeaderNames.Authorization].ToString();
4
5     if (accessTokenString == null || !accessTokenString.Contains("Bearer "))
6     {
7         return "NONE";
8     }
9     try
10    {
11        var accessToken = ...
12            .tokenHandler.ReadToken(accessTokenString.Replace("Bearer ", "")) as ...
13            JwtSecurityToken;
14        return accessToken.Claims.Single(claim => claim.Type == name).Value;
15    }
16    catch (ArgumentException)
17    {
18        return "NONE";
19    }
20 }

```

Class Validation: Fluent Validation is used to validate User and Business data in a request. The **BusinessDataModel** and it's sub-classes (**BusinessInfoDataModel** and **IdentificationDataModel** must be validated seperately.

Listing 4: Validating a Class

```

1 private static List<string> ValidateBusiness(BusinessDataModel businessDataModel)
2 {
3     var errors = new List<ValidationFailure>();
4
5     var businessDataValidator = new BusinessDataValidator();
6     var businessDataValidatorResult = ...
7     businessDataValidator.Validate(businessDataModel);
8     errors.AddRange(businessDataValidatorResult.Errors);
9
10    var businessInfoValidator = new BusinessInfoValidator();
11    var businessInfoValidatorResult = ...
12    businessInfoValidator.Validate(businessDataModel.BusinessInfo);
13    errors.AddRange(businessInfoValidatorResult.Errors);
14
15    var identificationValidator = new IdentificationDataValidator();

```

```
14     var identificationValidatorResult = ...
15         identificationValidator.Validate(businessDataModel.Identification);
16     errors.AddRange(identificationValidatorResult.Errors);
17     return ErrorsToStrings(errors);
18 }
```

Listing 5: IdentificationData Fluent Validation

```
1 public class IdentificationDataValidator : AbstractValidator<IdentificationData>
2 {
3     private readonly Regex _regex = new Regex(@"^[A-Za-z0-9@-_]");
4     public IdentificationDataValidator()
5     {
6         RuleFor(x => x.Name).NotNull();
7
8         RuleFor(x => x.Type).NotNull();
9
10        RuleFor(x => x.TVA).NotNull();
11        RuleFor(x => TVAClientService.ValidateVAT("BE", x.TVA))
12            .Equal(true)
13            .WithMessage("The TVA number is invalid");
14        RuleFor(e => e.EmailPro)
15            .EmailAddress(EmailValidationMode.Net4xRegex)
16            .WithMessage("invalid email address");
17        RuleFor(x => x.Description)
18            .MaxLength(1000)
19            .WithMessage("Maximum length is 1000 Characters");
20    }
21 }
```

B. Authentication API

B.1 Endpoints

Auth		▼
POST	/register	
PUT	/update/{email}	
GET	/roles	
GET	/user	
GET	/user/username	
GET	/user/validate	
GET	/user/block	
GET	/user/make-admin	

Figure 2: Authentication API

Figure 1 shows the endpoints:

- POST **/register** allows to create a new user.
- PUT **/update/email** allows to update the email address of the user.
- GET **/roles** it retrieves the list of roles.
- GET **/user?role={role}** it retrieves the list of users based on the role parameter.
- GET **/user/username** it retrieves the user based the username.
- GET **/user/validate** give a role type to the user
- GET **/user/validate** retrieve a role type to the user, block
- GET **/user/make-admin** give a role type as admin role to the user

B.2 Logic

User Roles: On user account registration, the user **role** is stored as a **claim** set to **REVIEWING**, and will be changed to **USER** when the user is validated by the admin. All the possible roles are stored using the **Role Manager** in the **Program.cs** class when the application is launched for the first time. The **admin** user is just a normal user, with **role** set to **ADMIN**, and he is stored using the **User Manager**, also in the **Program.cs** class. Blocking a user will set his role to **BLOCKED**.

Listing 6: Validating Users

```

1 [Route("user/validate")]
2 [HttpGet]
3 [Authorize(AuthenticationSchemes = "Bearer", Roles = "ADMIN")]
4 public async Task<ActionResult> ValidateUserAccount(string username)
5 {
6     var user = await _userManager.FindByNameAsync(username);
7     if (user == null)
8     {
9         return BadRequest(new
10             {
11                 data = new List<string> { "User " + username + " doesn't exist" }
12             });
13     }
14     await _userManager.RemoveFromRoleAsync(user, "BLOCKED");
15     await _userManager.RemoveFromRoleAsync(user, "REVIEWING");
16     var result = await _userManager.AddToRoleAsync(user, "USER");
17     if (!result.Succeeded)
18     {
19         return BadRequest(new
20             {
21                 data = new List<string> { result.Errors.ToList()[0].Description }
22             });
23     }
24     return Ok(new
25     {
26         data = result.ToString()
27     });
28 }

```

C. Common Library

It contains set of classes that different applications need.

II. FRONT-END

A. Angular

A.1 Authentication

Login Page: The login page shown in Figure 3 takes in the username and password, and sends it to **<https://localhost:44321/connect/token>**, with **grant_type** set to **password**, **scope** set to **bm** along with the application's client credentials. Once we receive the token, we can use a **JWT decoder** to extract all the claims such as the user email, name, surname and role, or we can call **<https://localhost:44321/connect/userinfo>** and receive all the claims directly.

The screenshot shows the 'Login' page of the 'Business Manager' application. The navigation bar at the top includes 'Home', 'Register', 'Business Overview', and 'Login'. The main content area is titled 'Login' and contains two input fields: 'Username' with the value 'admin' and 'Password' with masked characters '.....'. Below the password field are two buttons: a green 'Login' button and a blue 'Register' button.

Figure 3: Login Page

A.2 Navigation Bar

The buttons on the navigation bar are dynamically **shown** and **hidden** based on whether a user is logged in or not, and based on his role (ADMIN, USER, VALIDATED or BLOCKED) as shown in Figure 4 when logged in as an **ADMIN**.

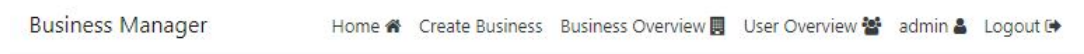


Figure 4: Navigation Bar (using the Admin)

A.3 Business Pages

Business Create/Edit Page: This is where a business can be created or edited, only by a **user** or an **admin**, so **blocked** or users who are still **not validated** cannot use this page or its corresponding API endpoint.

Figure 5 shows how errors are neatly displayed using ***ngFor** in html on the returned errors from **Fluent Validation**.

Figure 5 also shows a user friendly **work hours** configuration section, constructed using ***ngFor** on each work hour of the business.

Create Business

Please fill in all the required fields.

- Phone number is invalid
- Phone number must be a mobile.
- Url not formatted correctly
- Url Instagram not formatted correctly
- Url Facebook not formatted correctly
- Url LinkedIn not formatted correctly
- 'Type' must not be empty.
- 'TVA' must not be empty.

Name:

Business Email:

Phone:

Website Url:

Instagram Url:

MONDAY:
 Closed: ☐
 From: To:

TUESDAY:
 Closed: ☐
 From: To:

WEDNESDAY:
 Closed: ☒
 From: To:

THURSDAY:
 Closed: ☐
 From: To:

FRIDAY:
 Closed: ☐
 From: To:

SATURDAY:
 Closed: ☒

Figure 5: Business Create

Business Overview Page: This is where all the validated businesses are shown, and can be viewed in detail by clicking on their logo. A default logo is shown if no logo is uploaded. Apparently, the admin is able also see disabled businesses and Enable/Disable/Delete a business as shown in Figure 6.

A dot next to the business ID shows if a business is currently **open** or **closed** based on calculation done using the **current time** and the **work hours** of the business. It is also possible to hover the cursor over the dot to display text in case the dot color is unclear.

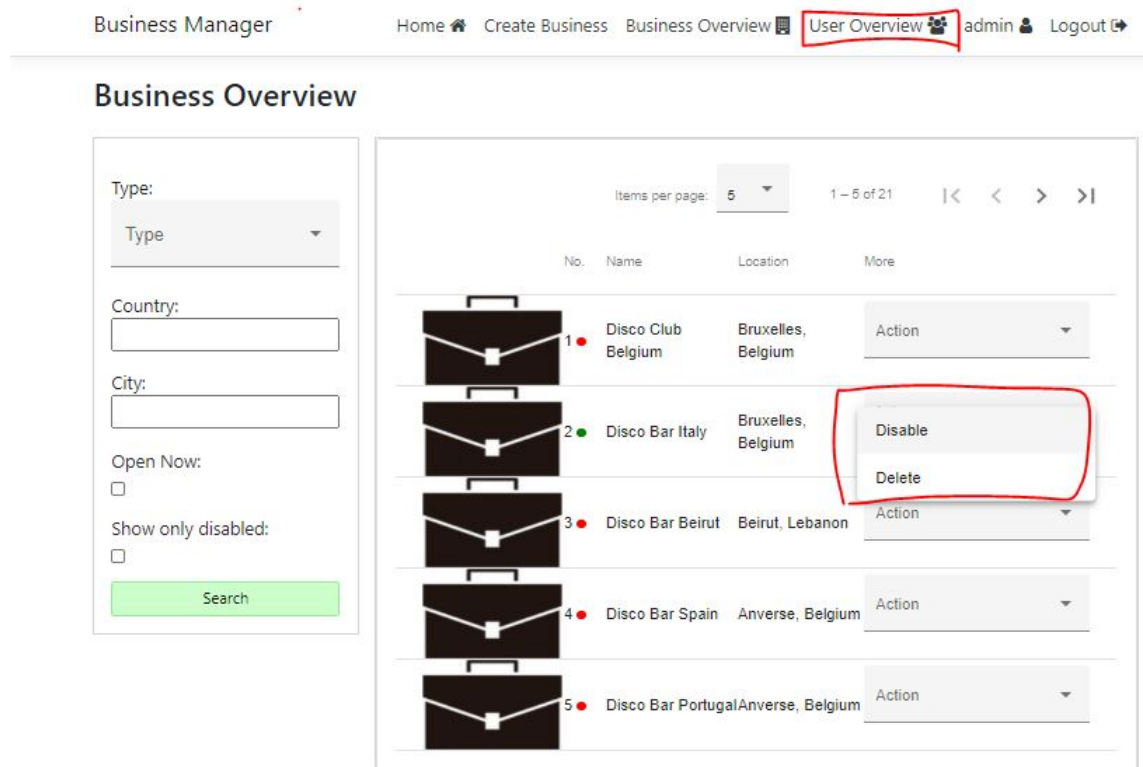


Figure 6: Business Overview (using the Admin)

Business Detail Page: This page shows all the available information about a business.

A google maps component has been added to more easily visualize the location of the business as shown in Figure 7. The map coordinates have been fetched from the Google **geocode** API, and the map component is from the **AGM** library from Google.

The **opening hours** are displayed dynamically using ***ngFor** in html on the available business **work hours**.

A **USER** can Edit/Disable his own business, and an **ADMIN** can Edit/Enable/Disable any business using the buttons on the top left.

Disco Club Belgium

Disabled: false

Edit

Disable

Delete

Details:
 Business ID: 1
 Name: Disco Club Belgium
 Type of Business: Club
 Description: Disco Club since 1990.

Address:
 Street: Rue de Belgique
 Postal code: 1100
 City: Bruxelles
 Country: Belgium
 Box Number: 1

Contact Info:
 Professional Email: mohamadjarmak@gmail.com
 Business Email: francesco.bigi.87@gmail.com
 Phone: +32493741427

Opening Hours:
 Monday: 8:00 - 17:00
 Tuesday: 8:00 - 17:00
 Wednesday: 8:00 - 17:00

Figure 7: Business Details (using the Admin)

A.4 User Pages

User Overview Page: Only an **ADMIN** has access to the **user overview** page, where he can **turn a user into an admin**, **validate/unblock a user** and **block a user** as shown in Figure 8. It is also possible to group the user's according to their roles using the drop-down on the left.

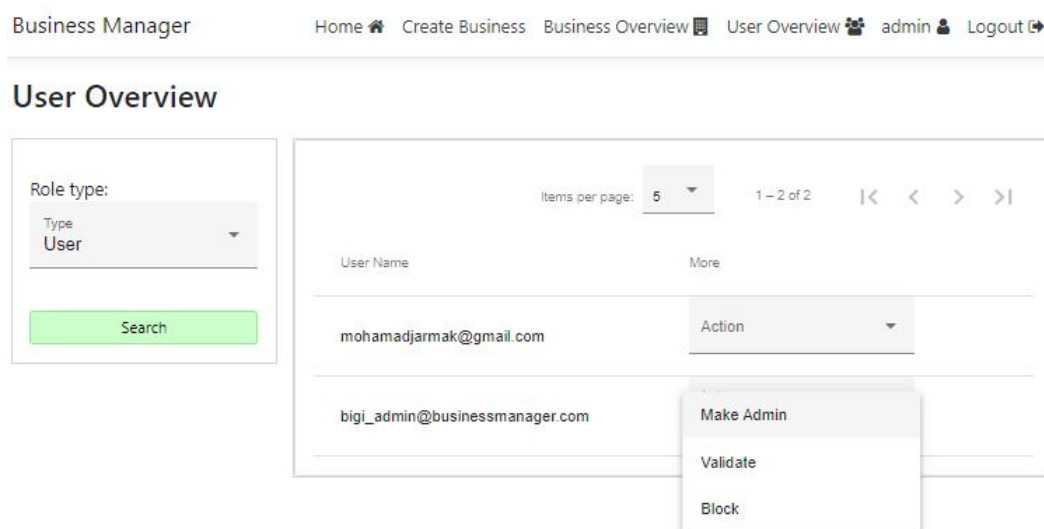


Figure 8: User Overview for the Admin

User Detail Page: A user can only **view** and **edit** his own details. Only the **name**, **surname** and **phone number** are allowed to be updated as shown in Figure 9.

The screenshot shows the 'User Details' page for 'Francesco Bigi'. The page has a title 'Francesco Bigi' and a form with the following fields: 'Email Address: bigi_admin@businessmanager.com', 'Birthday: 10/07/1995', 'Gender: Male', 'Name: Francesco', 'Surname: Bigi', 'Phone Number: +32466550935', and 'Professional: ☐'. At the bottom of the form are two buttons: 'Save' (green) and 'Cancel' (blue).

Figure 9: User Details

III. REFERENCES

- <https://github.com/>
- <https://console.developers.google.com/>
- <https://www.codewithmukesh.com/blog/repository-pattern-in-aspnet-core/>