EPHEC

Business Manager - C# .NET Core and Angular Project

Francesco Bigi
EPHEC - Informatique de Gestion

Contents

List of Figures

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

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 cathegorized 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 = ...
               _tokenHandler.ReadToken(accessTokenString.Replace("Bearer ", "")) as ...
               JwtSecurityToken;
12         return accessToken.Claims.Single(claim => claim.Type == name).Value;
13     }
14     catch (ArgumentException)
15     {
16         return "NONE";
17     }
18 }
```

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 = ...
           businessDataValidator.Validate(businessDataModel);
7      errors.AddRange(businessDataValidatorResult.Errors);
8
9      var businessInfoValidator = new BusinessInfoValidator();
10     var businessInfoValidatorResult = ...
           businessInfoValidator.Validate(businessDataModel.BusinessInfo);
11     errors.AddRange(businessInfoValidatorResult.Errors);
12
13     var identificationValidator = new IdentificationDataValidator();
```

```
14      var identificationValidatorResult = ...
            identificationValidator.Validate(businessDataModel.Identification);
15      errors.AddRange(identificationValidatorResult.Errors);
16
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             .MaximumLength(1000)
19             .WithMessage("Maximum length is 1000 Characters");
20     }
21 }
```

B. Authentication API

B.1 Endpoints

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

```
[Route("user/validate")]
[HttpGet]
[Authorize(AuthenticationSchemes = "Bearer", Roles = "ADMIN")]
public async Task<ActionResult> ValidateUserAccount(string username)
{
    var user = await _userManager.FindByNameAsync(username);
    if (user == null)
    {
        return BadRequest(new
        {
            data = new List<string> { "User " + username + " doesn't exist" }
        });
    }
    await _userManager.RemoveFromRoleAsync(user, "BLOCKED");
    await _userManager.RemoveFromRoleAsync(user, "REVIEWING");
    var result = await _userManager.AddToRoleAsync(user, "USER");
    if (!result.Succeeded)
    {
        return BadRequest(new
        {
            data = new List<string> { result.Errors.ToList()[0].Description }
        });
    }
    return Ok(new
    {
        data = result.ToString()
    });
}
```

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.

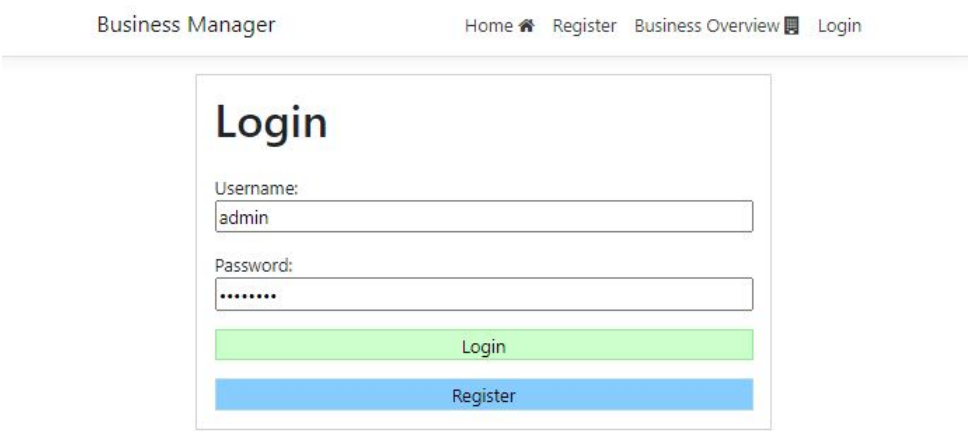

Figure 3: Login Page

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**.

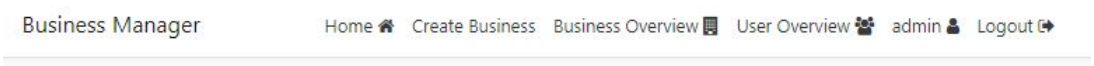

Figure 4: Navigation Bar (using the Admin)

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 it's 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.

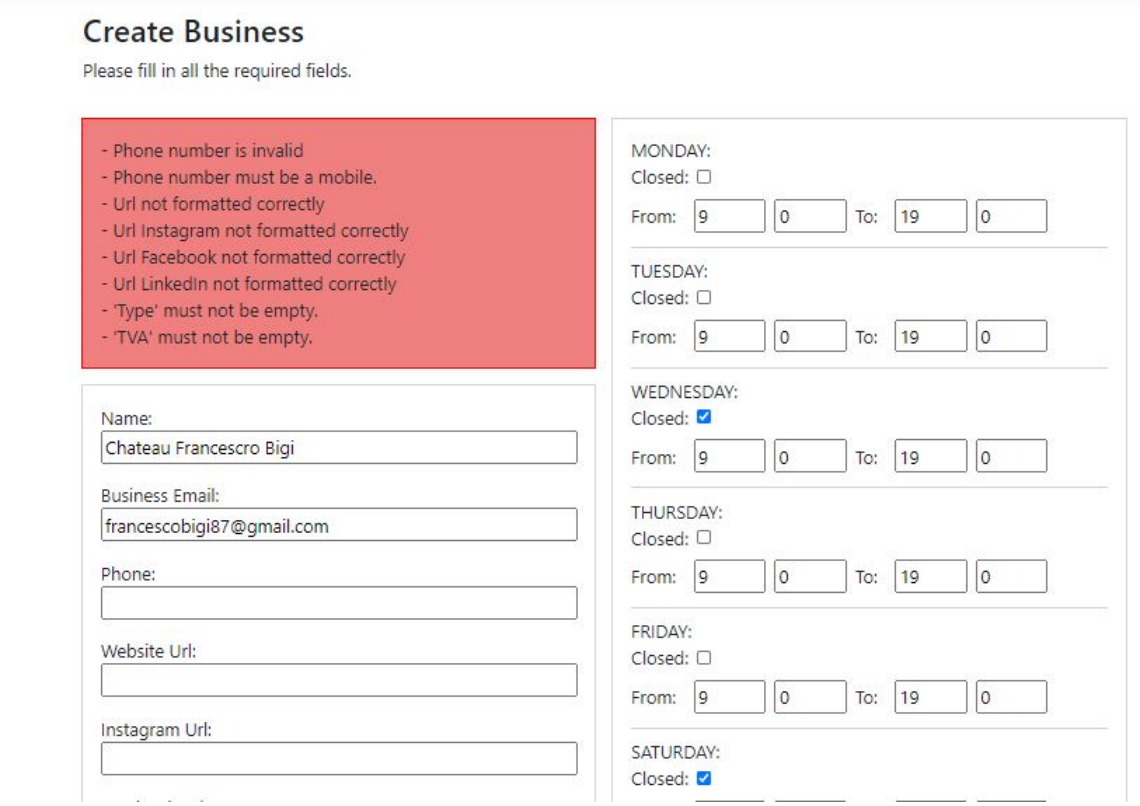

Figure 5: Business Create

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.

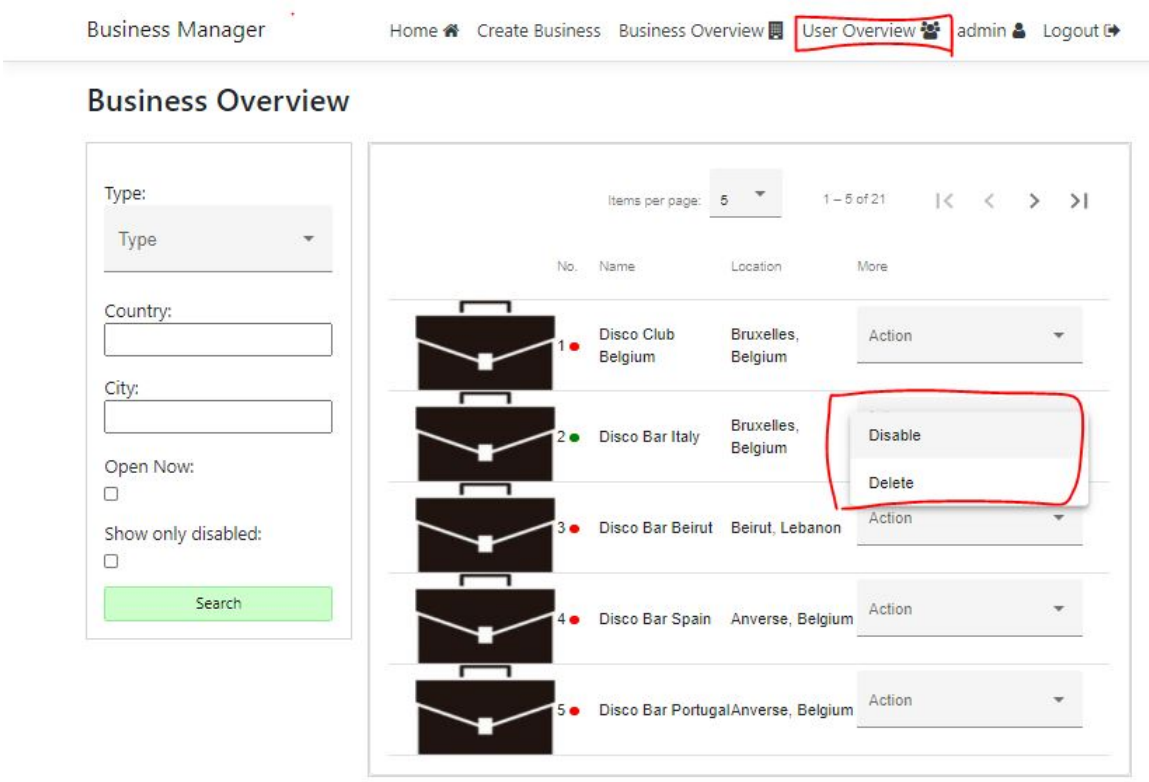

Figure 6: Business Overview (using the Admin)

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.

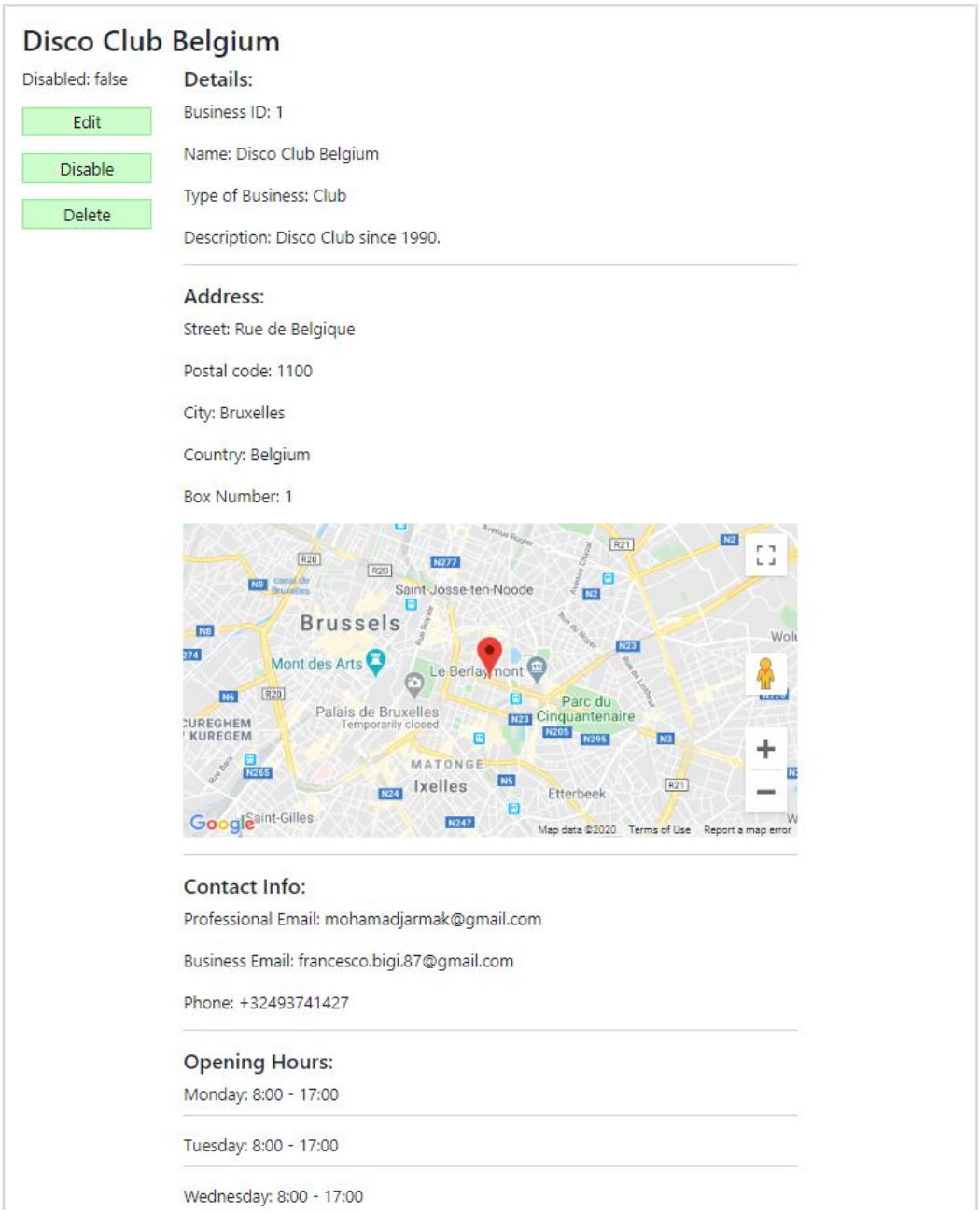

Figure 7: Business Details (using the Admin)

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.

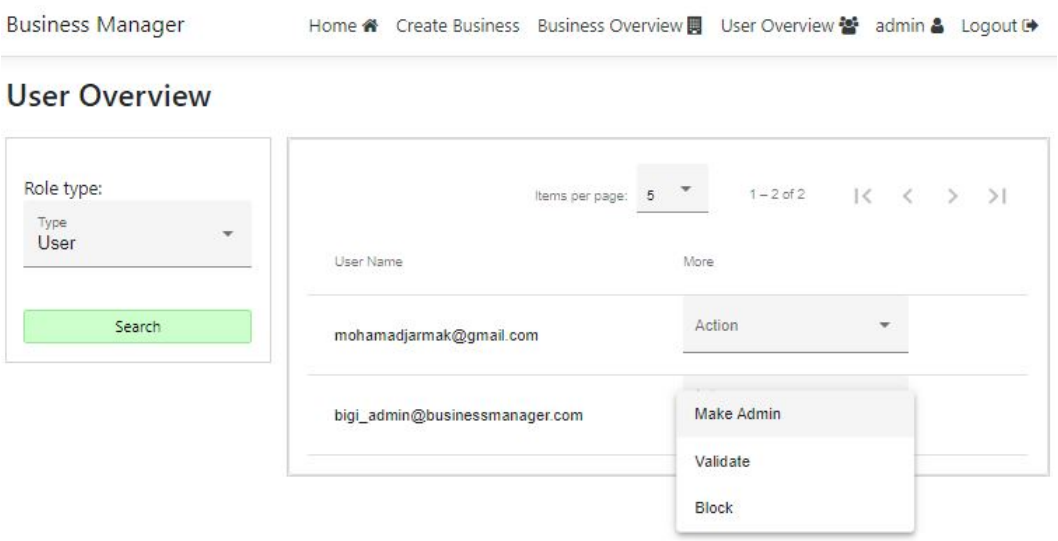

Figure 8: User Overview for the Admin

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.

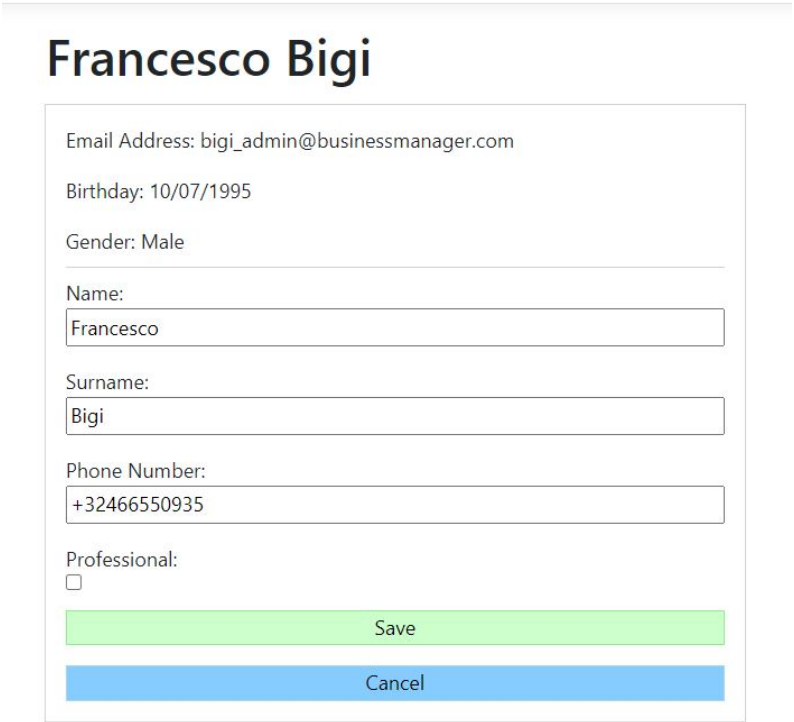

Figure 9: User Details

III. References

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