Release 02

Release Summary

In this release everything related to the new entities, the Owner and the Flat was implemented.

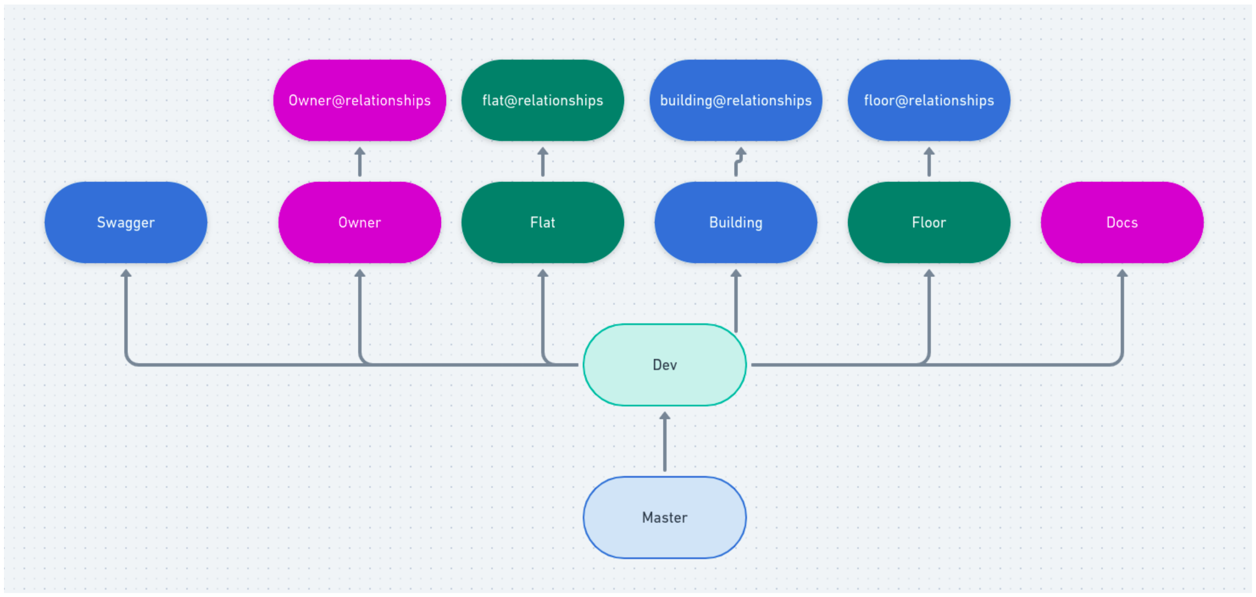
With the introduction of Swagger, all application endpoint documentation was created.

Release Phases

Phase 1 - Planning

- In this release we start by interpreting the new statement and collecting all the information we might need
- It was also at this stage that we owed tasks and their respective branches. Looking like this:
    - **Dayana Moreira** → Ownwe branch to work with everything related to Owner entity
    - **João Brito** → Flat branch to work with everything related to Flat entity
    - **João Gomes** → Swagger branch to work with swagger documentation

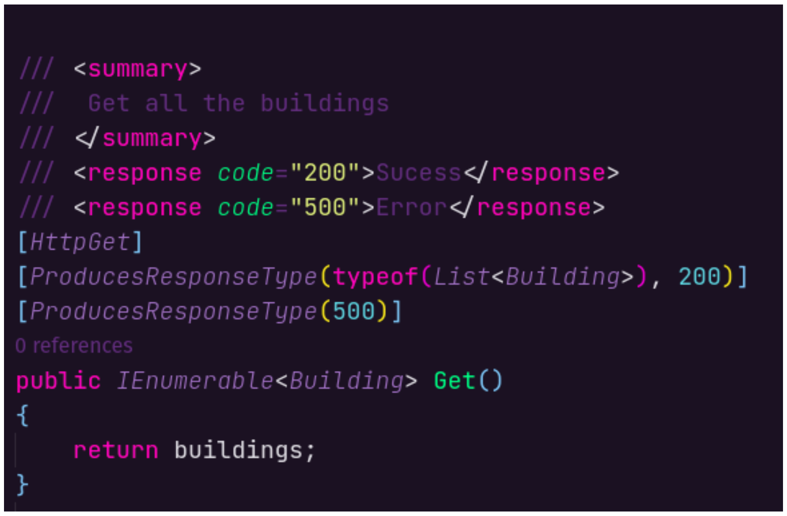
Branch tree update



Phase 2 - Execution

Swagger inplementation

- **How we did**



```
///  <summary>
///   Get all the buildings
///  </summary>
///  <response code="200">Sucess</response>
///  <response code="500">Error</response>
[HttpGet]
[ProducesResponseType(typeof(List<Building>), 200)]
[ProducesResponseType(500)]
0 references
public IEnumerable<Building> Get()
{
    return buildings;
}
```

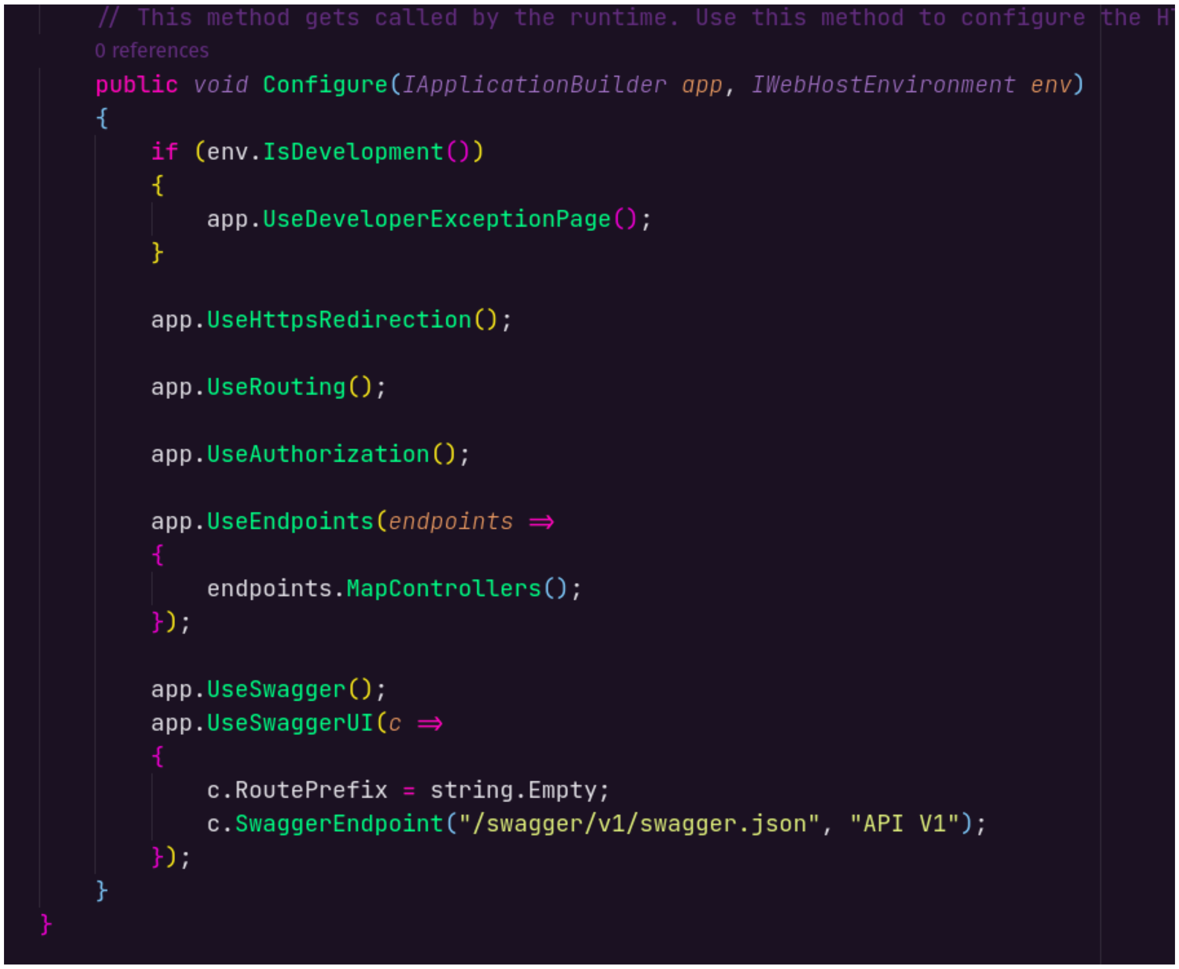
For the swagger implementation we put the comments above the endpoint.

we could also implement it through a json file but we came to the conclusion that this way was better.

- **Where it was necessary to change the code**

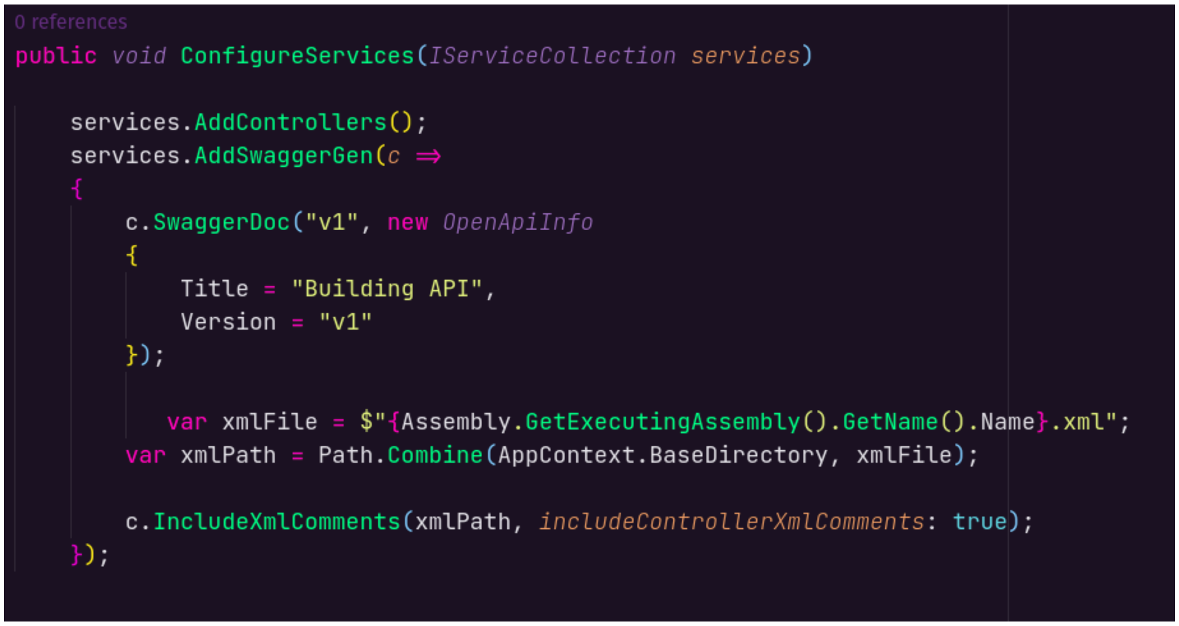
  We modified the code in the following files

  - **Startup.cs**



```
    // This method gets called by the runtime. Use this method to configure the H
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });

        app.UseSwagger();
        app.UseSwaggerUI(c =>
        {
            c.RoutePrefix = string.Empty;
            c.SwaggerEndpoint("/swagger/v1/swagger.json", "API V1");
        });
    }
}
```

- **Program.cs**



```
0 references
public void ConfigureServices(IServiceCollection services)

    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo
        {
            Title = "Building API",
            Version = "v1"
        });

          var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);

        c.IncludeXmlComments(xmlPath, includeControllerXmlComments: true);
    });
```

- **The advantages of using Swagger are:**
    - Better API documentation
    - Endpoint documentation
    - Best for teamwork
    - Better code read
    - Synchronizes the API documentation with the server and client at the same pace.
    - Allows us to generate REST API documentation and interact with the REST API. The interaction with the REST API using the Swagger UI Framework gives clear insight into how the API responds to parameters.
    - Provides responses in the format of JSON and XML.
    - Implementations are available for various technologies, such as Scala, Java, and HTML5.

Phase 3 - Documentation

- For the construction of the project documentation, we used the Notion tool, a very practical application of notes in Markdown, with support for PDF export.
- Endpoint documentation with Swagger.
    - We decided to implement the swagger as it is an excellent way to test and document the built endpoints.
    - To access it you should:
        - Clone the repository
        - Build and Run the project
        - Access the http://localhost:5000 and you be redirected to Swagger page

---

Tools used

- The tools that were used for the development of the project were:
    - **VS Code with pack extensions for .Net CORE**
        - We chose VS Code because the operating system that most members of the group use is Linux, and the main IDE for .NET (Visiual Studio) is not available for it.
    - **Dotnet CLI**
        - The use of the dotnet CLI was fundamental for the project's productivity, and with simple commands we can give the project **build** and **run**

    This development environment pleased us immensely, as we were able to implement the code and execute it very easily. Without a doubt a very good long-term environment.

Remote Repositories

Github

GitHub - JoaoGomes5/DevOps2022G-JJD: 🚧 Project carried out for the Software Development and Operations subject to improve the practice of versio
🚧 Project carried out for the Software Development and Operations subject to improve the practice of version control and development in .NET Core 🐙 - GitHub - JoaoGom
for the Software Development and Operations subject to improve the practice of version control and development in .NET Core 🐙

https://github.com/JoaoGomes5/DevOps2022G-JJD

Bit Bucket

Bitbucket

https://bitbucket.org/joaogomes13/devops2022gi/src/master/