**Assumptions:**

* The developer knows and understands common principles of design of enterprise application ([IoC task](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\IoC))
* The developer knows and understands IoC principles ([IoC task](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\IoC))
* The developer knows Dapper or EF Core
* The developer knows how to test RESTful services within 3rd-party tools like [Postman](https://www.getpostman.com/) or [Postman plugin for Chrome](https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=ru)

**General notes:**

* The main purpose of this task is deep knowledge of ASP.NET WebAPI Core. This knowledge is necessary for passing task interview;
* Task interview will be scheduled after all tasks will be completed and reviewed;
* “Need to know/understand something” phrase means that need to be ready to discuss this theme deeply with mentor;
* This version of the document should be copied to the root of your repository branch which will be used for these tasks. If during tasks implementation some mistakes will be found (or you have some recommendations or tasks improvements) then it should be reflected in the mentioned document’s copy. For this action need to enable change tracking in review tab of MS Word;
* In case if some external and useful tasks related resources will be found then it should be reflected in the mentioned document’s copy.

**Documentation notes:**

* ASP.NET WebAPI books can be found [here](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\ASP.NET%20WebAPI\Books) . Choose one good book for reading;
* ASP.NET WebAPI videos can be found [here](file:///\\docs.oxagile.com\DEPARTMENTS\HR\TA\Library\dotNet-internal-tasks\ASP.NET%20WebAPI\Videos);
* Microsoft recommended ASP.NET WebAPI resources can be found [here](https://dotnet.microsoft.com/apps/aspnet/apis).

**Theory:**

* Need to understand what is RESTful service.
* Need to know in details and understand ASP.NET WebAPI application lifecycle;
* Need to understand what is GET, POST, PUT, DELETE methods. What's the difference between them?
* Need to understand what is Routing;
* Need to understand what is OAuth;
* Need to understand what is OData;

**Implementation notes:**

* All external libraries should be referenced as nuget packages;
* MS SQL or PostgreSQL database should be used;
* All tasks should be implemented in accordance with [SOLID](http://en.wikipedia.org/wiki/SOLID_%28object-oriented_design%29) principles;
* Hardcode and “magic number” anti-patterns are not allowed;
* Dead code, empty folders, unnecessary commented code are not allowed also;
* The code should be styled in the same manner as [Code Conventions Oxagile](https://wiki.oxagile.com/pages/viewpage.action?spaceKey=DNET&title=Code+style+rules);
* The tasks should be implemented using ASP.NET Core WebAPI (пакет Microsoft.AspNetCore.App);
* ORM should be used for DB access. ORM should be defined by mentor (Dapper or EF Core);
* “Need to know/understand something” phrase means that need to be ready to discuss this theme deeply with mentor;
* Need to understand difference between ErrorFilters, AuthorizationFilters;
* Need to know all approaches of filters applying;
* Need to know how to add Swagger UI to WebAPI project.
* Visual studio project should not have references to ASP.NET MVC Libraries.

**Task 1: CRUD**

* Need to implement simple ASP.NET Core WebAPI CRUD service. The domain model that covers all requirements of this task displayed on the figure 1.0. Besides this application should meet the requirements below.

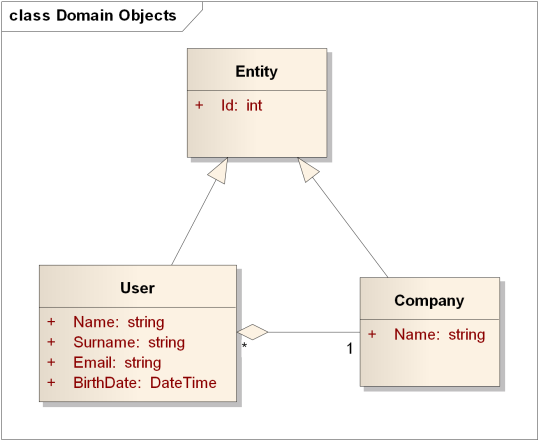


Figure 1.0 CRUD Application domain model

1. Company service should allow:

* Create company
* Edit company
* Delete company
* List all companies
* Get company by ID.

1. User service should allow:

* Create user
* Edit user
* Change user’s company
* Delete user
* List all users
* List all users of specific company
* Get users by ID.
* Each method should be available with only one appropriate HTTP Method (GET, POST, PUT, PATCH, DELETE)
* Urls of all methods should satisfy [guidelines](http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api) and be self-describing. It means than developer who will use that API should be able to understand what concrete method does. Example (Get all users of company with ID = 15):
  + Method: GET
  + Url: <http://domain.com/api/company/15/users>
* No need to implement validation functionality in context of this task.

**Task 2: Validation**

* Need to add data annotation based server validation:
* Email is correct;
* Email is required;
* Name and Surname is required;
* The maximum number of users for one company is N (Configured value).
* The validation should be replaced to fluent based validation (nuget package: [FluentValidation.WebAPI](https://www.nuget.org/packages/FluentValidation.WebAPI/)).
* *Need to implement appropriate client based validation (optional).*

**Task 3: API Documentation**

* Need to add auto-generated API documentation using Swagger UI. Use [Swashbuckle package](https://github.com/domaindrivendev/Swashbuckle)
* Email is correct;
* Email is required;
* Name and Surname is required;
* The maximum number of users for one company is N (Configured value).

**Task 4: Error handling**

* Need to add Custom Error Message Exception Handler
* All errors should be handled.
* Exception data should not be send to user with response message. Custom message (stored in resource files) should be sent instead.
* Need to add Custom Error Message Logger.
* All errors should be logged with all necessary data like datetime, thread id, request url, request parameters.
* Log exceptions via [SeriLog](https://github.com/serilog/serilog);

**Task 5: Media Formatters**

* Need to implement custom Media Formatter that allows user to request data in csv format as well as in json and xml formats by default. It can be reached using WebApiContrib.Core.Formatter.Csv media formatter that converts response data to CSV format.
* Custom media formatter should not contain hardcode to convert concrete type to csv
* 3rd-party library can be used to generate csv.

**Task 6: Asynchronous Controllers**

* Add possibility to upload photo for each user. User DTO should contain the url to uploaded photo.

**Details:**

* Photo resizing should be performed dynamical for each icon request. You could use one of the following packages: ImageSharp, SkiaSharp, Magick.NET. Useful article [.NET Core Image Processing](https://devblogs.microsoft.com/dotnet/net-core-image-processing/)
* Asynchronous controllers and core library methods should be applied in appropriate way.
* DB should contain relative links to the photos only. Blob files should be stored on file system.

**Task 7: Create an OData v4 Endpoint Using ASP.NET Core Web API**

* Add possibility to query user data with OData technology with possibility to filter data, skipping entities, limit amount of result entities.
* Use package Microsoft.AspNetCore.OData or Microsoft.AspNetCore.OData.Versioning.
* There are could be some issues with OData in .NET Core:

<https://github.com/OData/WebApi/issues/1177>

<https://github.com/Microsoft/aspnet-api-versioning/issues/361>

<https://github.com/OData/WebApi/issues/1196>