WEB DEVELOPER PRACTICAL TEST

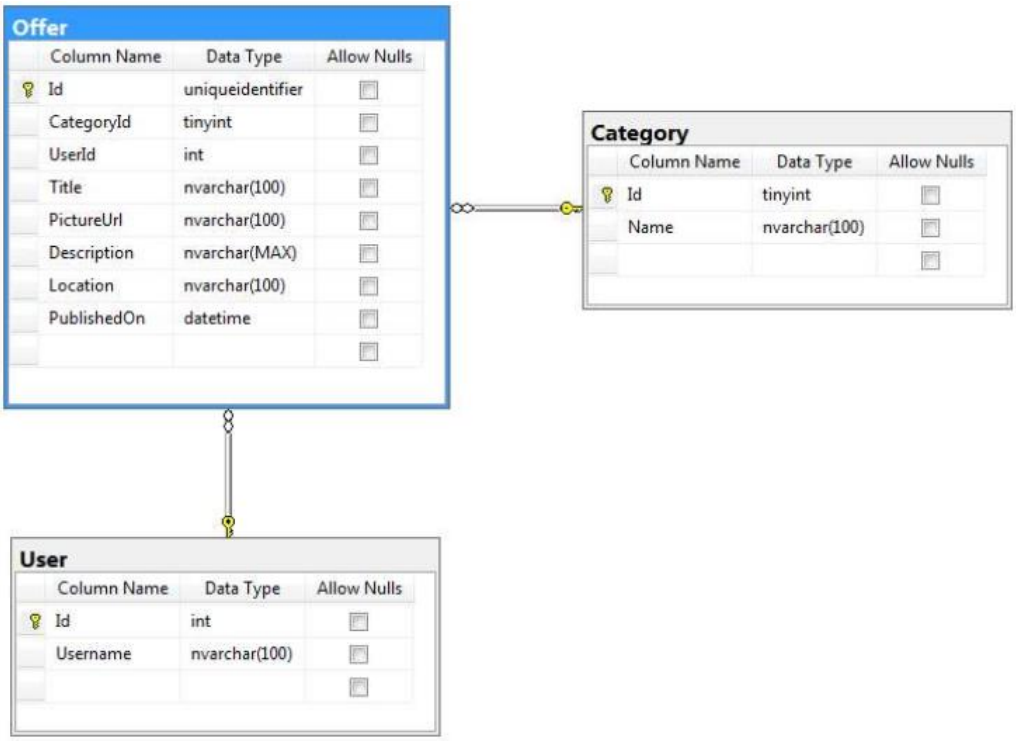
Read first: Pre-requirements

- GitHub account
- Git
- .NET CORE 6 SDK
  https://dotnet.microsoft.com/en-us/download/dotnet/6.0
- VS Code (or any other IDE you can work on)
  https://code.visualstudio.com/download
- Angular v16, Node.js, npm
  https://nodejs.org/es/download

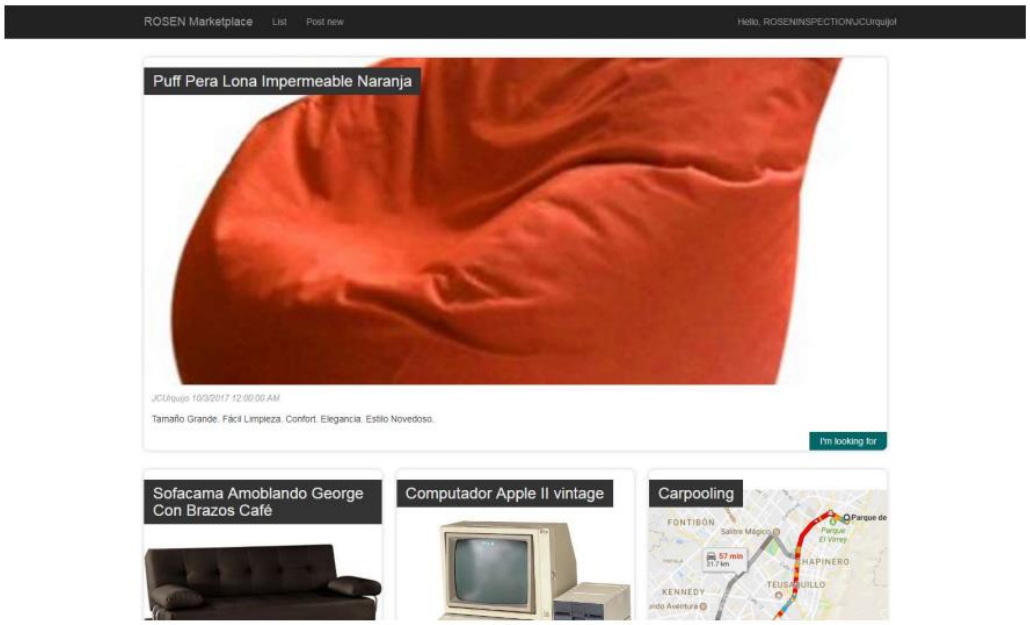
ROSEN Technology & Research Center (RTRC) requires a simple marketplace for its employees in order to publish items for resale, offers of services or inquiries for specific items. It was decided that a small responsive website should be developed using Angular, .NET 6, and SQL Lite.

You are presented with a solution started by another developer and the following design:

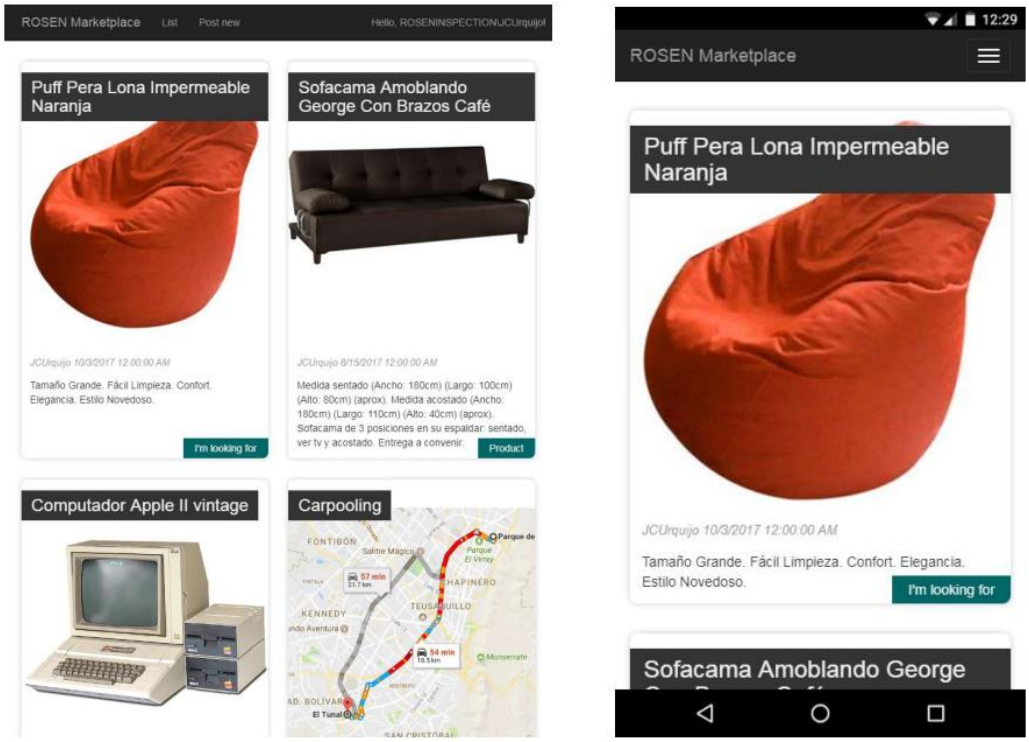
- A SQLite database attached into the project with the following model ready to be queried:



- The UX team designed the following mockup:



Desktop and larger Tablets in landscape mode. (width >= 992px)



Tablets in Portrait mode (screen width >= 768px) Smartphones (screen width <768px)

- Pagination should look like so and non-existing links (i.e.: when there are too few pages) should be hidden:

- The architect has decided that the pagination should be implemented in the frontend as the following class:

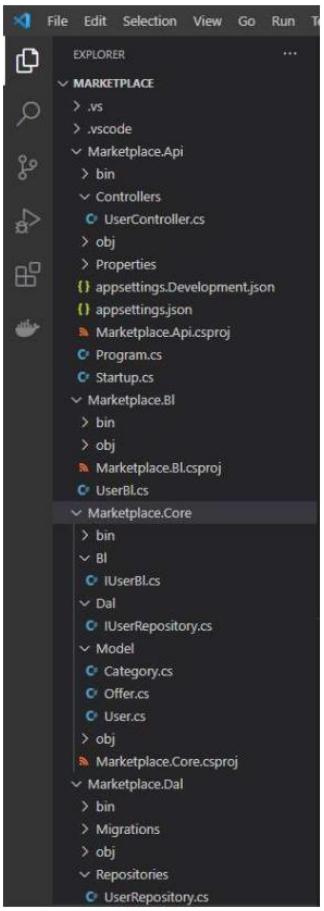
Page<T> <<class>>		
Items	Array<T>	Gets the items.
NextPageIndex	number	Gets the index of the next page or null if at the last page.
PageCount	number	Gets the page count.
PageIndex	number	Gets the index of the current page.
PreviousPageIndex	number	Gets the index of the previous page or null if at the first page.
Ctor(items, pageIndex, pageCount)		For parameter types, see the types of the homonymous properties.
GetNextPageIndexes()	Array<number>	Gets at most three next page indexes (only the ones that exists).
GetPreviousPageIndexes()	Array<number>	Gets at most three previous page indexes (only the ones that exists)

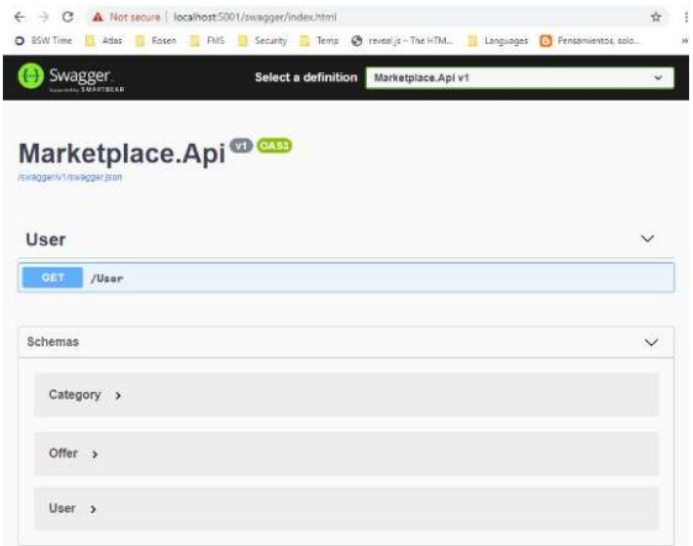
- Any user should be able to post a new offer using the form at the bottom of the page.

- Another developer on your team already started the project and did some work on the initial solution inside the given repository, which includes the API project and an Angular Application:

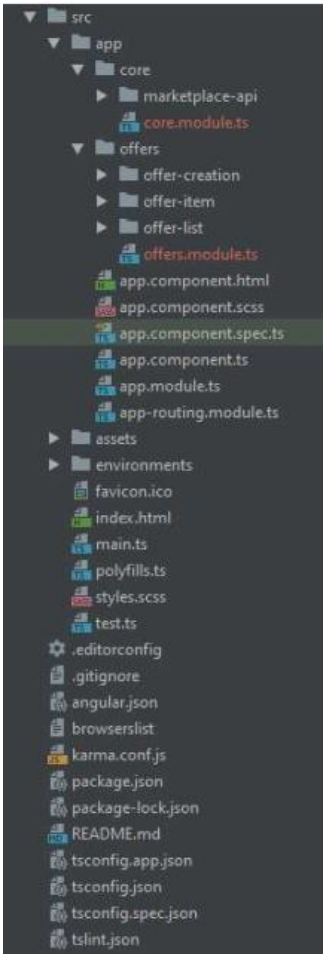
  The API Solution can be found in `./Marketplace/Api` and contains the following projects:

  1. Marketplace.Api (REST API Endpoints Controllers)
  2. Marketplace.Core (Solution Contracts)
  3. Marketplace.Bl (Bussiness Logic Contract Implementation)
  4. Marketplace.Dal (Data Access Layer)





- The Angular application can be found in `./Marketplace/WebApp` and has the following structure:



  1. core module (contains services used in all the application)
  2. offers module (contains all the components fort he offers)
  3. assets (contains the assets that need to be served as individual elements like images).

  The other developer already did the HTML for the list, the pagination and form which you'll find in offers module and an initial implementation of the `Page<T>` class, found in the core module.

  Your task:

  1. Implement the design proposed by the UX team.

  2. Implement the `Page<T>` class by first creating a set of unit tests. Make sure you test the behavior of the `Page<T>` class when in the following states:

     - Only one page is available
     - There are many pages (10 at least) and you are at the first page.
     - There are many pages (10 at least) and you are at the middle page.
     - There are many pages (10 at least) and you are at the last page.
     - Any other state you find interesting

  3. Implement the `offer` components so that all the expected behaviors work correctly. Keep in mind that you have 200 thousand Offers' rows and could be many more and it is ideal to display the information in the most optimal way possible, please consider the best practices for both front-end and back-end side.

  4. Take into account that the `Offer` table has a reference to the `User`. If the current user doesn't exists, the user shall be created before saving the offer.

  5. Implement the API Controllers and the contracts you consider for it.

  6. Add Unit tests when you consider them necessary, this will be taken into account for your score as additional points!