**Backend Tasks Project Requirements:**

The application is designed in four layers:

* Backend: the console application only to show data.
* Backend.BussinessLogic: here are all the entities with the business logic functions related to each process
* Backend.DataAccessLayer (DAL): here are all the entities with the functions to select or manipulate physical data (in this case, the json file) related to each process. It’s the only layer prepared to log errors, the idea is not to log business errors (just for this particular scenario)
* Backend.Models: the plain objects to exchange data through the application

Task 1

For this task it uses a Person model in the different layers

* Backend.BussinessLogic: **GetBornAfterDateByNationality** is the main function to retrieve the data, it uses two parameters for the filter, to not expose the dynamic search at the Access Layer
* Backend.DataAccessLayer (DAL):
  + **GetByFilter** and **GetAll** are the only exposed methods.
  + The class is prepared to read the data from a rest api or a file. In the settings the **JsonSource** value is the one that determinates the data source
  + **GetPersons** is the method that reads the data, it uses **GetJsonFromFile** or **GetJsonFromUri** depends on the source type
  + To add new Get methods with filters, just add them on the Business Layer, setup the filter, and call the filter method on DataLayer. There is no need, to add new functions on DAL to retrieve filtered data
* Backend.Models: just a plain object with the person attributes, it’s cross to the whole application

Task 2

For this task it uses a Person model in the different layers

* Backend.BussinessLogic:
  + **Salaries** is the main exposed class to calculate the final salary
  + To make salary discounts more dynamic, there is a base class called BaseDiscount. It has only one method, that calculates the final salary. The parameters are the base salary and the discount model class.
  + To add new discounts, modify GetImport on the Salaries class. If new rules are needed, just create a new instance of the existent discount models
  + If new rules are needed, build a new class that inherits from BaseDiscount, and override the GetImport method.
* Backend.DataAccessLayer (DAL):
  + Since this task doesn’t manipulate physical data, it doesn’t have a DAL layer
* Backend.Models: just a plain object with the person attributes, it’s cross to the whole application

Loggin

For the log it uses an extension pattern and a class with plain attributes (except for Message, that it’s just a readonly attribute)

**Frontend Tasks Project Requirements:**

All the base logic to build the table with the XML data, it’s on Script.js file.

1. getXmlData: it retrieves the data from the service. Is the first function called once the document is fully loaded
2. createTable: with the data from getXmlData start to build the table. It uses other two functions
   1. createTableHeader to build the table header with the column titles
   2. createTableBody to build the table header with the column data
3. There are other two auxiliar functions:
   1. getXmlElementByXpath that returns the specific value from the XML using XPath.
   2. countXmlElementByXpath that returns how many nodes has the XPath result query.