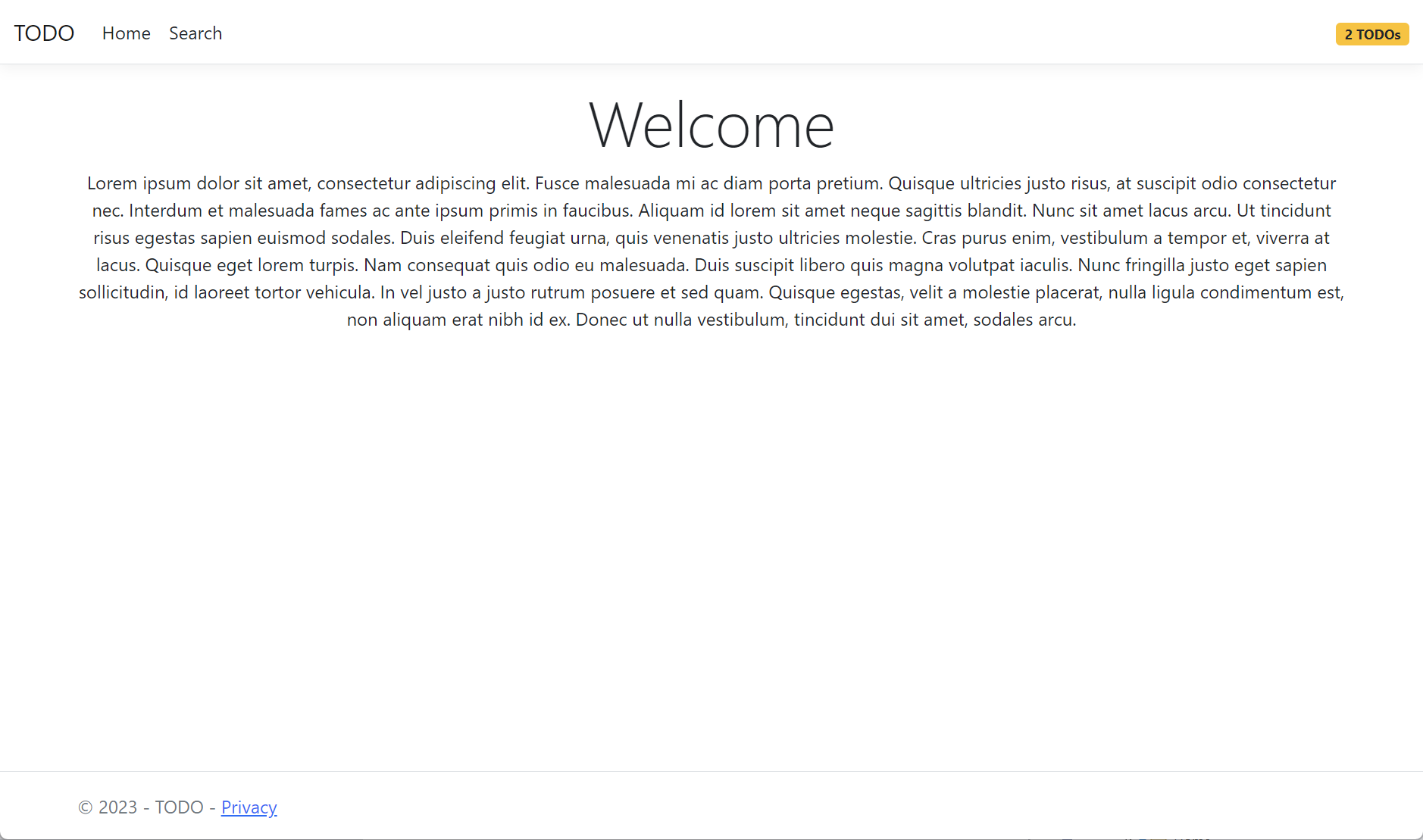
**TODO list app – a little bit of everything**

## Application

Build a web application that can be used to manage TODO lists.

The home page shows a welcome text.

In the top right corner the number of items that are not done (over all the TODO lists) is always shown.



It is possible to navigate to a search page on which you can find the TODO lists in the system

By default no lists are displayed. When the button is clicked, the matching lists are displayed. When no title filter is filled in, all the lists in the system are displayed.

Afbeelding met tekst, schermopname, Lettertype, software

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software, scherm

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, software, Lettertype

Automatisch gegenereerde beschrijving

Clicking on the “show details” button brings you to the detail page of a list.

On the detail page the items in the list are displayed and it is possible to add a new item to the list. The description of an item should be at least 4 characters long.

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, Lettertype, nummer

Automatisch gegenereerde beschrijving

There is also a REST api which can be used to:

* Create a new TODO list
* Get the details of a TODO list
* Set an item of a TODO list to done

**Use the start code, the snippets in snippets.txt and the checklist below to implement this application.**

## Checklist

Your implementation must be compliant with all the statements in the list below:

* MS SQL Server database is used. Connenctionstring must be in appSettings.Json
* Files in the wwwroot folder can be served
* Domain
  + The following tests are implemented and green:
    - ToDoListTests
      * CreateNew\_ValidTitle\_ShouldReturnNewListWithIdAndTitle
      * CreateNew\_TitleIsNullOrEmpty\_ShouldThrowArgumentException
        + This test should cover 2 cases: null and an empty string
    - ToDoItemTests
      * CreateNew\_ValidDescription\_ShouldReturnNewItem\_ThatIsNotDone\_AndWithIdAndDescription
      * CreateNew\_DescriptionIsNullOrEmpty\_ShouldThrowArgumentException
        + This test should cover 2 cases: null and an empty string
  + The *ToDoList* constructor is private (so that you must use the *CreateNew* factory method)
  + The *ToDoItem* constructor is private (so that you must use the *CreateNew* factory method)
* Infrastructure
  + The *ToDoListApp.Infrastructure* project contains implementations of the *IToDoListRepository* and *IToDoItemRepository* interfaces (the interfaces should remain unchanged)
  + Entity Framework is used for the implementation of the repositories
  + The *DbContext* contains only one set, being a set of *ToDoLists*
  + The *DbContext* has only one constructor. The constructor accepts an instance of *DbContextOptions*
  + Tip: the *DbContext* class has a *Set<T>* method that can be used to query the *ToDoItems*
  + Migrations are used to create the database
* Api
  + Should have RESTfull endpoints
    - GET to [https://localhost:5001/api/todolists/{some list id}](https://localhost:5001/api/todolists/%7bsome%20list%20id%7d) returns a 200 response with the TODO list and its items in the body if the list exists. A 404 response it returned when the list does not exist.
    - POST to <https://localhost:5001/api/todolists> adds a new list to the system and returns a 201 response with the created list in the body and a link to the created list in the Location header
    - PUT to [https://localhost:5001/api/todolists/{some list id}/items/{some item Id}](https://localhost:5001/api/todolists/%7bsome%20list%20id%7d/items/%7bsome%20item%20Id%7d%20) changes the *IsDone* status of an item (our customer says that descriptions can never be changed). It returns a 200 response with the updated item in the body
  + *ToDoListsController* should be in an *Api* subfolder of the *Controllers* folder
  + *ToDoListsController* should make use of an *IToDoListRepository* and a *IToDoItemRepository*
  + *ToDoListsController* should automatically return a 400 response when the *ModelState* is not valid (without explicitly returning a *BadRequestResult* in code)
  + The *AddNewToDoListModel* class is used for model binding an a POST request
  + The *UpdateToDoItemModel* class is used for model binding on a PUT request
  + New list must always have a title that contains at least 4 characters
  + <https://localhost:5001/api/todolists/d477ebc8-cc5d-4c3b-bcf5-7e94c73c16b6> matches the GET endpoint. <https://localhost:5001/api/todolists/123> should not match the GET endpoint and should thus not be handled by the *MapControllers* middleware.
* Search Razor Page
  + A GET request to <https://localhost:5001/ToDo/Search> should be handled by a Razor Page.
  + The *\_Layout* in *Views/Shared* is used as layout
  + The content of the <title> tag in the resulting html should be “Search – TODO”
  + The “Search html” snippet is used as a base to create the Razor code
  + The link to the detail page is constructed using tag helpers
  + A search is only performed after the form is submitted. When the page is loaded (GET) no search is performed
  + The *SearchModel* uses a *TitleFilter* property, a *ToDoLists* property and an *IToDoListRepository*
  + Tag helpers are used for labels. The value of the labels should be the same as in the screen shots
* Layout
  + Views are rendered inside the <main> tag
  + Links are constructed using tag helpers
  + The TODO link in the left of the navigation bar points to the home page
  + The Home link in the navigation bar points to the home page
  + The Search link in the navigation bar points to the search page
    - Tip: <https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/anchor-tag-helper?view=aspnetcore-6.0#usage-in-razor-pages>
  + The privacy link in the footer points to the privacy page (see *HomeController*)
* Number of TODO’s badge
  + Is show in the right of the navigation bar for all Views and Razor Pages that use de *\_Layout* layout
  + The *IToDoItemRepository* is used to retrieve the number of items that are not done
  + The “Badge html” snippet is used
* TODO list detail form
  + Is constructed using the MVC application framework
  + Convention based routing is used. The url to the form should be [https://localhost:5001/ToDoList/Detail/{id of the list}](https://localhost:5001/ToDoList/Detail/%7bid%20of%20the%20list%7d)
  + Should make use of on *IToDoListRepository*
  + When a list does not exist, the user is redirected to the home page
  + Uses *ToDoListDetailViewModel*. The constructors should not be changed.
  + When an item is added the details page is showed again, but now with the newly added item in the list of TODO item
  + Tip: use a hidden input to get the *ListId* back on a POST
  + Tag helpers are used for labels. The value of the labels should be the same as in the screen shots
  + Tip: clear the *ModelState* after adding a new item so that the new item description input field is cleared
  + The “Detail html” snippet is used
  + The content of the <title> tag in the resulting html should be “[list title] – TODO”. [list title] is the actual title of the list
  + If the new item description contains less than 4 characters, a validation error is displayed right near the input field
  + The *Back to search* link uses tag helpers to link to the search page
  + Reuse of code is promoted by putting the code to show one item in a partial view named *\_ToDoItemCard*