INF 354

Assignment 1
Out of 50 Marks

DUE: 4 April 2025

IMPORTANT NOTES:

- This is an individual assignment.
- Homework assignments are based on assessment objectives. If an objective has been achieved, a mark will be allocated.
- **All assignments are submitted via ClickUP. See the Assignments section**.
- **You only upload your Angular App (.zip), API (.zip), and video demo (.mp4).**
- **You may use the L&P: 01 Architecture with TypeScript Source Code shared with you previously. See the code on ClickUP under Course Content. You then build upon it**.
- **Please execute the API migration before building your angular application**. **See the Angular and API installation and configuration section.**
- **If you are caught plagiarising, we will give you zero per cent (0%), and you will be reported for plagiarism immediately.** We will audit historical assignments throughout the semester. **We trust that you understand the importance of this point.**

VIDEO INSTRUCTIONS:

- Make sure that everything is running when you start recording the video. The video should not be longer than 15 minutes showing the items in the **Standard Requirements** against the **Rubric**.
- When showing something from the **Standard Requirements**, show us as much detail as required. ***See the Rubric for the assessment criteria***. For example, when assessing the "***Program Functionality,***" you must show the CRUD functionality is working and that the data is saved and updated in the database. Similarly, for the "***Program Output,***" the correct records are displayed on the related pages per CRUD. The web pages display the right data in a valid format, demonstrating all the pages. Further, for the "***Code readability***" we expect you to show us your code and display the organization of the code and descriptive names (*i.e., all the code used to create the program,* ***not the configuration files like package.json****, etc.*). **The same applies to the rest of the Rubric. See below.**
- If something did not work in your code, in the video, explain to us what you wanted to do and what you wanted to achieve with your approach. **This is to assess you correctly according to the Rubric**.
- **See the "Video Recording and Compression and Assignment Upload Guide" in the Assignments section on ClickUP for video recording, compression, and upload assistance.**

SUBMISSION INSTRUCTIONS:

- In this assignment, you will be given the requirements to implement.
- **Source Code:** Zip your source code files together, and for the API name it **uXXXXXXXX_HW01_API.zip**, where the XXXXXXXX is your student number, e.g., u12345678_HW01_API.zip. Further, for the Angular App, name it **uXXXXXXXX_HW01_Angular.zip**, where the XXXXXXXX is your student number, e.g., u12345678_HW01_Angular.zip.
- **Video Demo:** **Do not** zip your **video demo**. In other words, submit the actual "**.mp4**" file. Name the video demo **uXXXXXXXX_HW01.mp4**, where the XXXXXXXX is your student number, e.g., u12345678_HW01.mp4.
- If files are uploaded to the wrong upload area, we will not look for the upload. Uploads should be submitted correctly.
- **Please Note**: **If you omit the code (.zip) or the video (.mp4) submission, you will lose 50% of your assignment mark**. **If no files are uploaded (neither the .zip nor .mp4), you lose 100%. Please take this seriously and plan accordingly to submit it on time**.
- ***Note: you upload the code (.zip files) and the video demo (.mp4 file) together in the same location in the Assignment 01 Submission section. See the ClickUP information in the Assignments section (when readily available).***

- Please **do not** upload the "***node_modules"*** and "***.angular"*** folders for the **Angular App**. In other words, once you have completed your program and created your video, delete the "***node_modules"*** and "***.angular"*** folders. As the Lecturing Team, we will reinstall the ***node_modules*** folder dependencies using the "***npm install***" terminal command, *where necessary*. **This is so you do not take long to upload your code with the video demo.**
- In addition, **do not** upload the "***bin"*** and "***obj"*** folders for the **.Net API application**. In other words, once you have completed your program and created your video, delete the "***bin"*** and "***obj"*** folders.

SUBMISSION DEADLINE: 4 April 2025

- Submission is due at **11:59 AM**
- There shall be no extensions to the deadline above.
- If homework submissions are uploaded too late, then upload errors **will** happen.
- Do not wait until the last minute to complete the assignment.
- Start working on the assignment as soon as possible.
- E-mail submissions **will not** be accepted.
- Late submissions **will not** be accepted.
- **No exceptions will be made for anyone.**

USE CASE:

- A client requests you to create a proof-of-concept application using Angular and .Net 8 Web API. They want to see if software development productivity improves using the Angular framework.
- You are requested to develop the back end using a **.Net 8 API** and the front end using **Angular**.
- For the application, you need to build the capability for users to Create, Read, Update, and Delete (CRUD) courses stored in the SQL Server database.
- When the application is launched, the landing page must be the "**Product Listing" page**, and navigation to all other pages must be done via angular routing, subject to the restrictions that will be detailed under "***Standard Requirements***".

STANDARD REQUIREMENTS:

- Product Listing page:
    - The product listing page should pull through the records you created when you have done the API migration using the seed data provided in the **AppDbContext.cs** file (Fig. 1).
    - In addition, the cards that list the ten records should have an **Edit** and **Delete** button per record (Fig. 1).
    - The cards should display only the **Name**, **Description**, and **Price** data from the Product table.
    - Clicking on the Delete button should delete the relevant record (e.g., Running Shoes) and display the updated listing (Fig. 2).
    - Clicking the Edit button should take you to the "Edit Product" page to edit the existing database record (Fig. 3).
    - The Navigation Bar displayed in Fig.1. should be shared across all the created pages. The "Product Listing" link takes you to the "Product Listing" page (current page), and the "Add Product" link routes you to the "Add Product" page (Fig. 6).

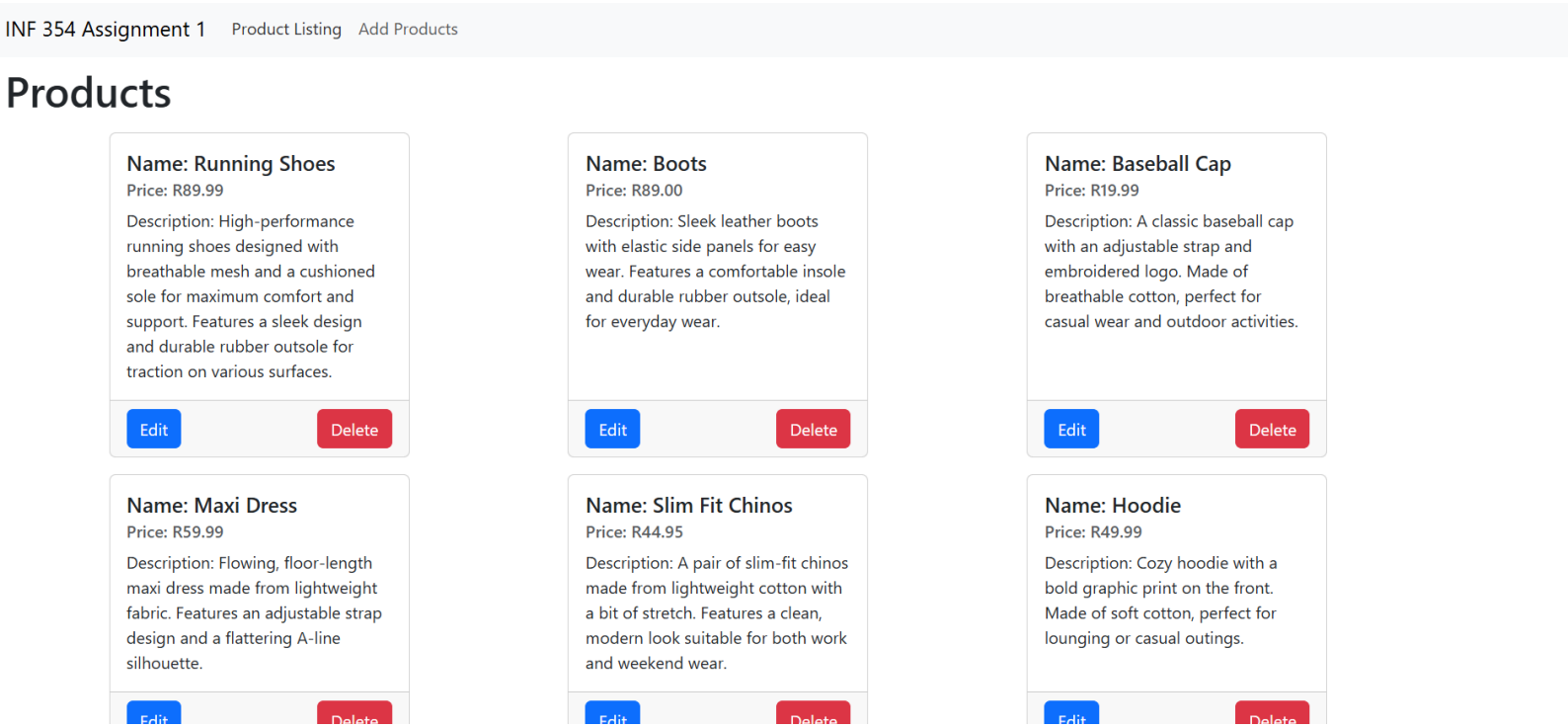

Fig. 1

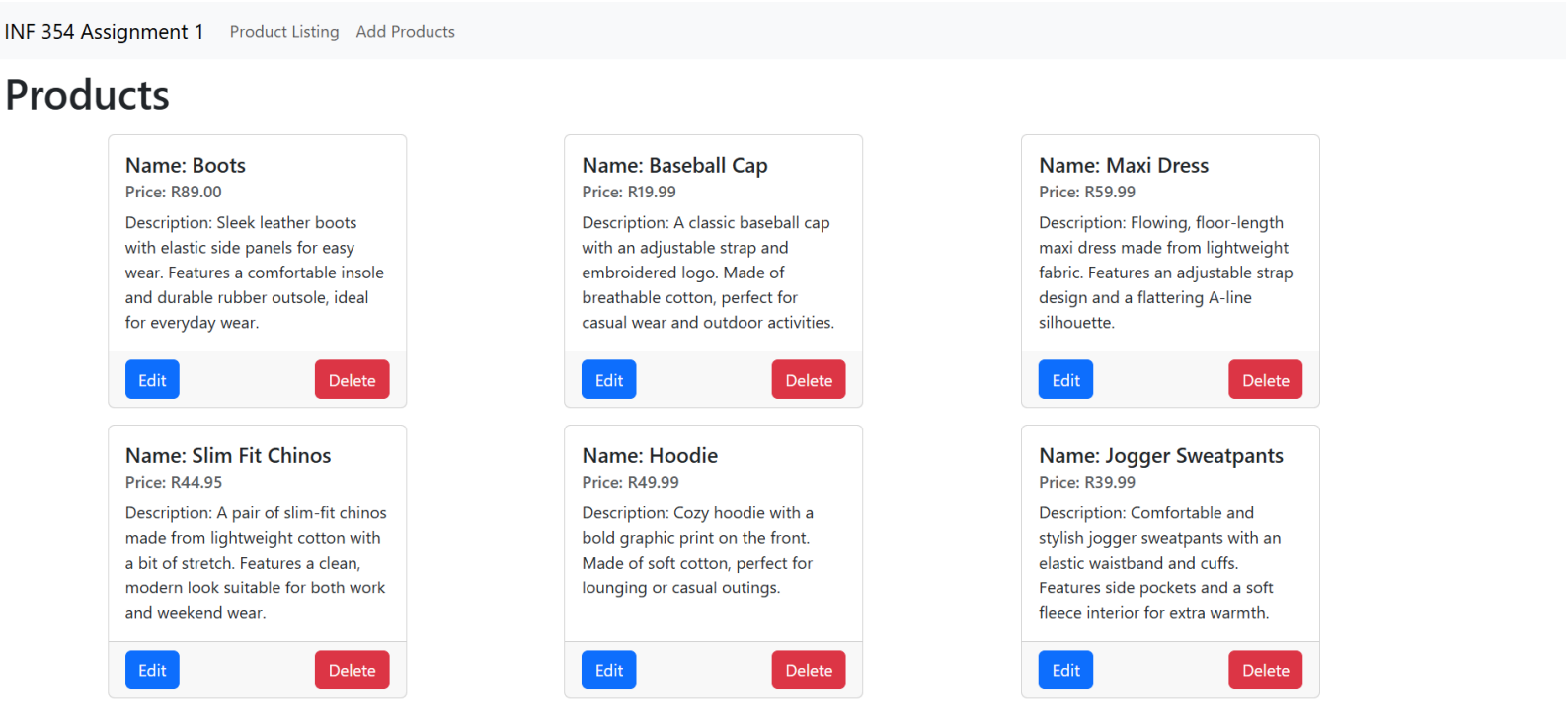

Fig. 2

- Edit Product page:
  - The “Edit Product” page should allow the user to change the current product's Name, Description, and Price. The user can update any one, two, or all of the control values (Fig. 4).
  - After clicking the submit button, the record updates in the database, and the user is routed to the “Product Listing” page (Fig. 5)
  - Clicking on the cancel button returns the user to the “Product Listing” page without making any changes to the data.

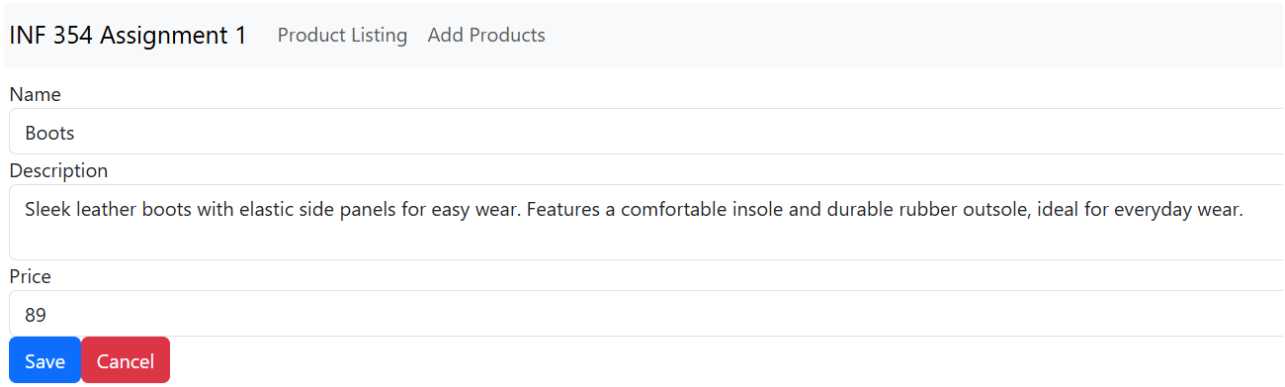

Fig. 3

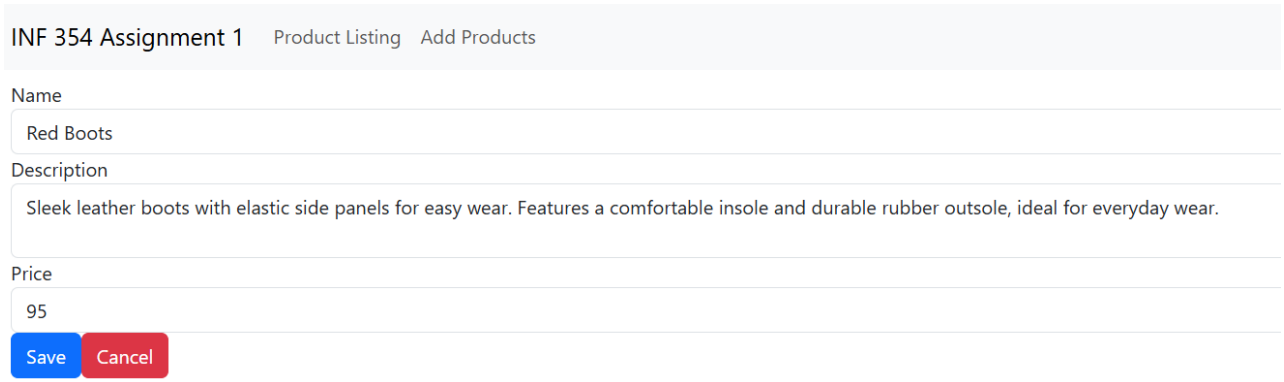

Fig. 4

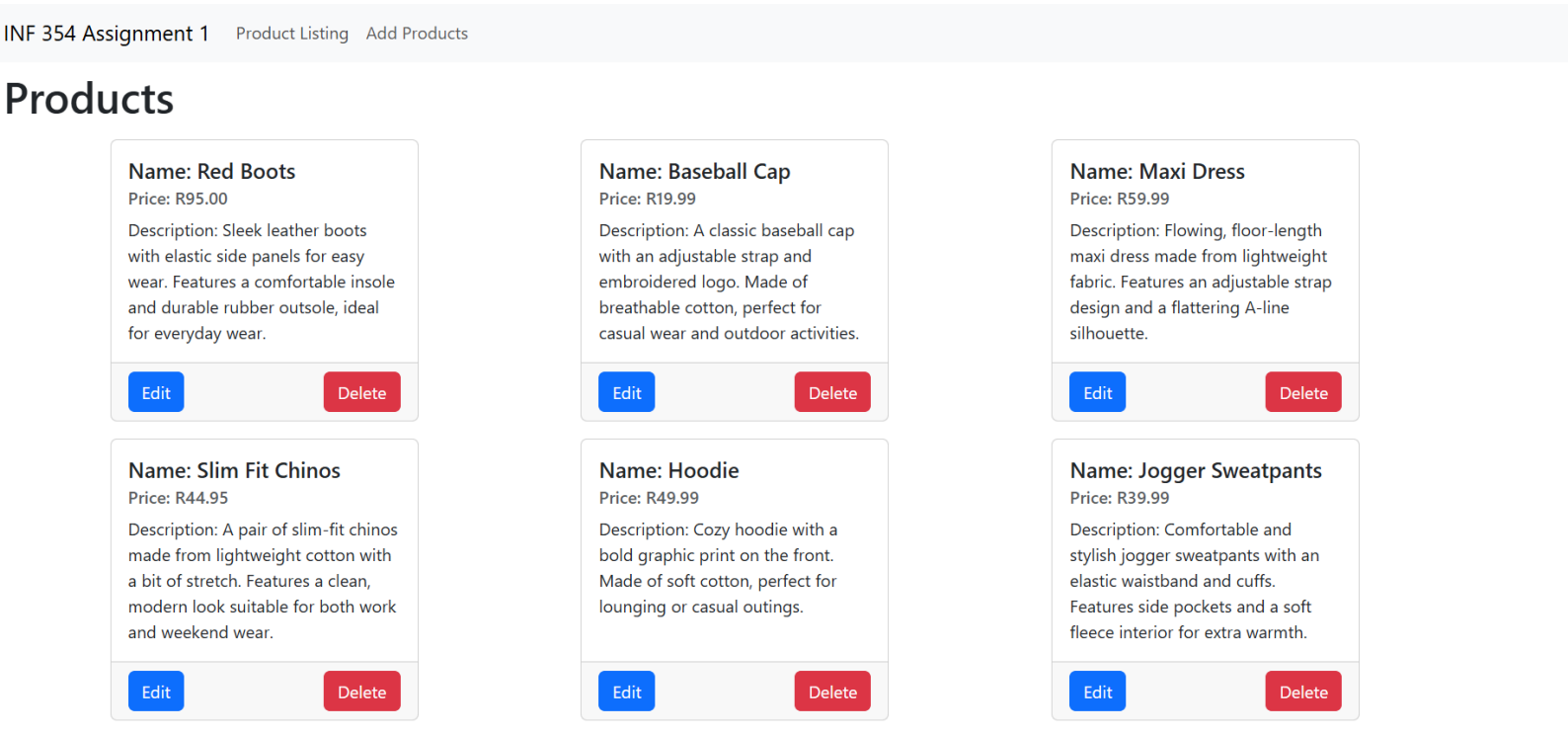

Fig. 5

- Add Product page:
  - Once the user clicks on the “Add Product” link, they should be routed to the “Add Product” page (Fig. 6)
  - The submit button is disabled and will only be enabled once values are provided for **all three Form Controls** (Fig. 7).
  - After clicking the submit button, the record is created in the database, and the user is routed to the “Product Listing” page (Fig. 8)
  - **Note =>** The newly added product should be the first item displayed in the list.
  - Clicking on the cancel button returns the user to the “Product Listing” page without making any changes to the data.

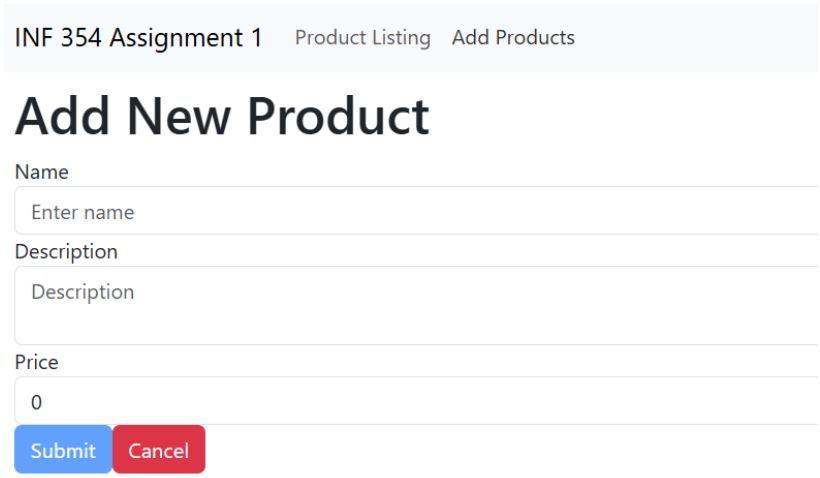

Fig. 6

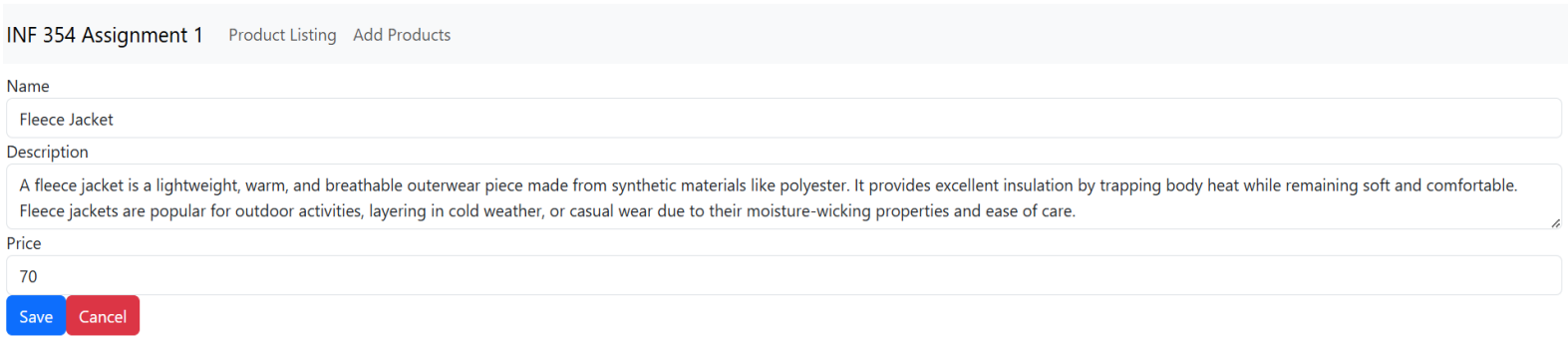

Fig. 7

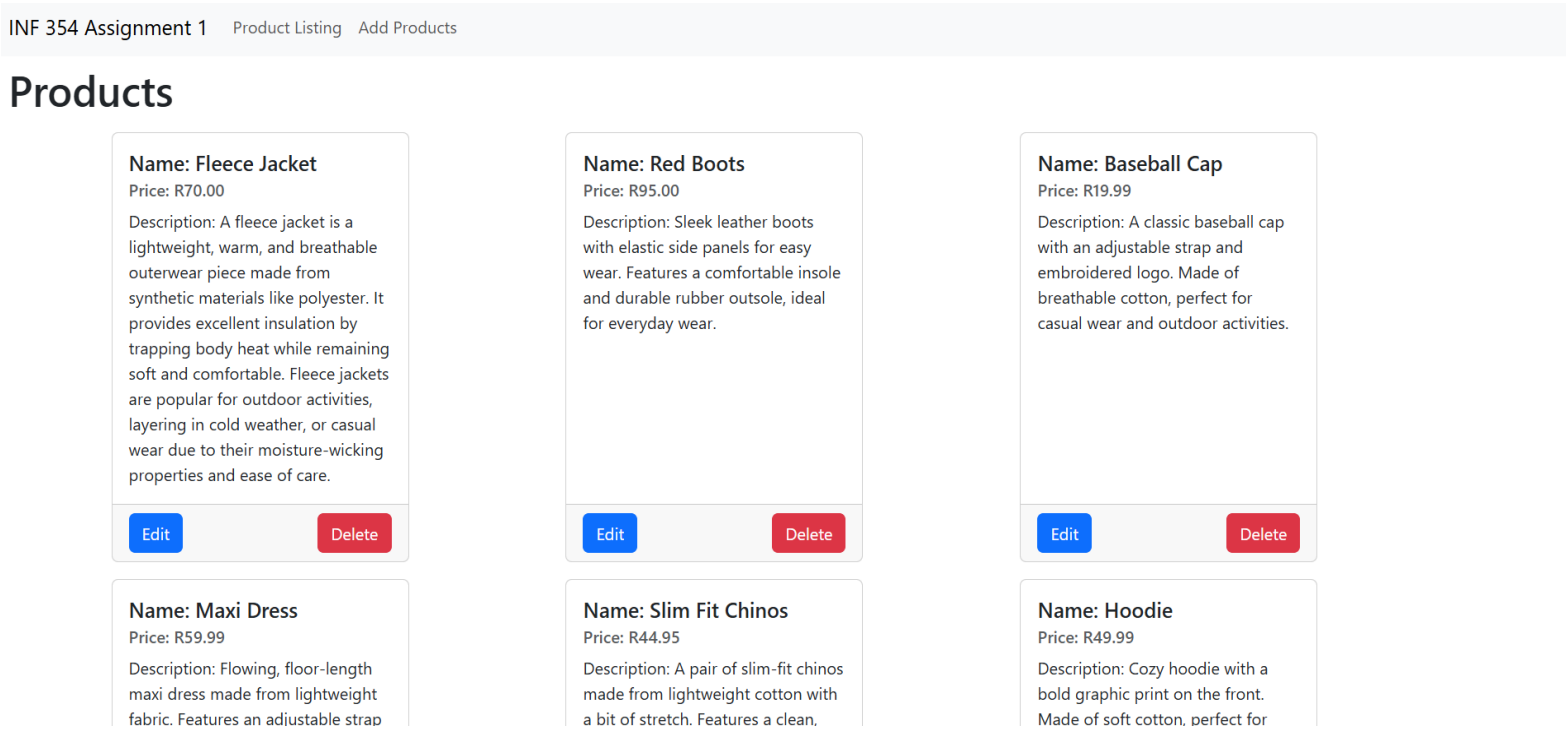

Fig. 8

Unit Testing:

In your API backend, implement Unit Testing by completing the following xUnit test methods:

- Retrieve all Products
- Retrieve a Product by ID

Each test should verify that the returned objects are NOT NULL and produce a response of type "OK" (HTTP 200 Status Code).

Note: Ensure the proper use of Assert methods to validate the response.

ANGULAR AND API INSTALLATION AND CONFIGURATION:

- API:
  - You can work off of the **Source Code** made available in the API I and API II lectures. L&P 03 and L&P 04.
  - Open the **API .Net Core application** in Visual Studio 2022.
  - Once the application loads, open the "**appsettings.json**" file in Solution Explorer.
  - Change and save the **Server** location, pointing to your *SQL Server Server Name*). Alternatively, you can just replace the server name with a period (.). See the example below.
  - Example:
    ```
    optionsBuilder.UseSqlServer("Server=.;Database=Assignment1;Trusted_Connection=True;MultipleActiveResultSets=True");
    ```
  - Next, open the Package Manager Console (**View** > **Other Windows** > **Package Manager Console**) and run the following 2 commands individually to create the database tables from the abovementioned entities.
    - add-migration initial
    - update-database
  - The **Assignment1 MS SQL Server database will create the Product table**.
  - Now, run the API and have it running when you are trying to connect your Angular App to it. In other words, the API and the Angular App must be running for the application to work correctly.

- Angular:
  - You can work off of the **Source Code** made available in the Angular I and Angular II lectures. L&P 05 and L&P 06.
  - When using the Source Code, open Visual Studio Code, and run "***npm install***" in the terminal window in the correct folder. I.e., the folder where you run "npm install" must have, for example, the "angular.json" file directly in it or you will run into problems.
  - To run the application, type "**ng serve**" in the terminal window.

SUGGESTIONS:

- For the API, you will likely have 1 controller (*the API controller with endpoints (functions) to talk to the database and Angular App*).
  - For example, a **ProductController** with at least 4 endpoints (functions) to **GetAllProducts (GET)**, **AddProduct (POST)**, **EditProduct (PUT),** and **DeleteProduct (DELETE)**.
- You can **design your UI** *any way you want*, **as long as it has all the controls and output required as specified in the Standard Requirements**.
- You can **develop your API** *any way you want*, **as long as it can perform the functionality required as specified in the Standard Requirements**.

RUBRIC:

Your assignment submission will be marked according to the following rubric:

Program (50 pts)	(Exceptional)	(Very good)	(Good)	(Satisfactory)	(Poor)	(Very poor)
Program Execution	The program executes correctly with no syntax or runtime errors. *I.e. the program has no execution issues.* (10)	The program executes with one or two syntax or runtime errors. *E.g. the program loads with no crashing but displays minor bugs in the debugger.* (8)	The program executes with a few syntax or runtime errors. *E.g. A couple of runtime errors and/or the program crashes at one screen/section.* (6)	The program executes with many syntax or runtime errors. *E.g. A few runtime errors and/or the program crashes at two screens/sections.* (5)	The program executes with major errors. *E.g. The program can execute, however, it is plagued with the runtime or syntax errors, or the program keeps crashing during use.* (3)	The program does not execute. *I.e. The application fails to run.* (0)
Program Functionality	Program functionality is in line with the requirements. *I.e. the program has all the correct functionality implemented.* (10)	Program functionality has one minor inconsistency. *E.g. One of the functional requirements is incorrect.* (8)	Program functionality has a few minor inconsistencies. *E.g. Two of the functional requirements are incorrect or one is missing.* (6)	Program functionality has many inconsistencies. *E.g. Some of the functional requirements are incorrect or half is missing.* (5)	Program functionality has major inconsistencies. *E.g. Most of the functional requirements are incorrect or missing.* (3)	Program functionality is missing. *E.g. None of the functionality works or all the functionality is missing.* (0)
Program Output	The program displays the correct output in line with the requirements. *I.e. It produces the same output as required.* (10)	The program has one or two very minor output discrepancies. *I.e. It produces output with barely noticeable inconsistencies. E.g. one or two formatting issues.* (8)	The program has a few output discrepancies. *I.e. It produces output with easily noticeable inconsistencies. E.g. The program does not return some of the data or there are a few formatting issues.* (6)	The program has many output discrepancies. *I.e. It produces output with many noticeable inconsistencies. E.g. The program does not return half of the data or there are plenty of formatting issues.* (5)	The program has major output discrepancies. *I.e. The output is plagued with inconsistencies. E.g. The program does not return most of the data or there are substantial formatting issues.* (3)	Output is incorrect. *E.g. The program does not provide any of the requested output or all the formatting is not as requested in the requirements.* (0)
Program Interface (UI)	The program interface is professionally done. I.e. *The interface is implemented correctly and looks very good.* (5)	The program interface is done well. *I.e. The interface is implemented correctly and looks good. E.g. One or two styling/layout issues.* (4)	N/A	The program interface is good enough. *I.e. The interface is implemented correctly and looks okay. E.g. A few styling/layout issues.* (3)	The program interface is poorly done. *I.e. The interface is mostly incorrect or looks poorly done. E.g. The layout is mostly incorrect or has plenty of styling issues.* (2)	The program interface is very poor. *I.e. The interface is entirely incorrect or is very poorly done. E.g. The layout is completely incorrect or the styling is missing.* (0)
Code Readability	The program code is well organized and makes good use of white space. Variables have	The program code is organized and makes use of white space. Variables have descriptive names.	N/A	Program code is mostly organized and makes use of white space. Most variables have descriptive	Program code is somewhat organized, and not easy to read and understand. *E.g. There are plenty of variable naming convention issues*	Program code is difficult to read. *E.g. Variable naming conventions are*

Program (50 pts)	(Exceptional)	(Very good)	(Good)	(Satisfactory)	(Poor)	(Very poor)
	descriptive names. I.e. *There is nothing to fault on.* (5)	*E.g. There are one or two variable naming convention issues or white space issues.* (4)		names. *E.g. There are a few variable naming convention issues or program code organization that could be improved.* (3)	*or the code is challenging to follow.* (2)	*missing or the code is hard to follow.* (0)
Video Demonstration	The program is exceptionally well presented. I.e. *The student demonstrated and displayed all the required functionality, output, interfaces, and code.* (10)	The program is well presented. *E.g. The student demonstrated and displayed all the required functionality, output, interfaces, and code. However, one of the descriptions or illustrations was lacking.* (8)	The program presentation is good. *E.g. The student demonstrated and displayed most of the required functionality, output, interfaces, and code. However, two of the functionality, output, interfaces and code descriptions or illustrations were lacking or missing.* (6)	The program presentation is adequate. *E.g. The student demonstrated and displayed most of the required functionality, output, interfaces, and code. However, a few to half of the functionality, output, interfaces and code descriptions or illustrations were lacking/missing.* (5)	The program is presented poorly. *E.g. The student demonstrated and displayed a few of the required functionality, output, interfaces, and code. However, most functionality, output, interfaces, and code descriptions or illustrations were lacking/missing.* (3)	The program has barely been presented or has been presented very poorly. *E.g. The student failed to demonstrate and display the required functionality, output, interfaces, and code or it was missing.* (0)