UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering,
Built Environment and
Information Technology

Fakulteit Ingenieurswese, Bou-omgewing en
Inligtingtegnologie / Lefapha la Boetšenere,
Tikologo ya Kago le Theknolotši ya Tshedimošo

DEPARTMENT OF INFORMATICS

INF 354

EXAMINATION (1) **DATE: 2022-06-24**

Examiners : Mr Ridewaan Hanslo
: Mr Zola Mahlaza
: Ms Demi Murray

Time : 180 min

Moderator / External Examiner : Prof Abejide Ade-Ibijola
University of the Johannesburg

Marks : 80

Student Number								Surname	Initials
-	-	-	-	-	-	-	-	MEMO	--

Question Section	Module outcomes (as in Study Guide)							Marks allocated	Maximum mark
	MO1	MO2	MO3	MO4	MO5	MO6	MO7		
Section A			X	X	X			30	30
Section B		X				X		25	25
Section C	X						X	25	25
Total								80	80

Instructions

1. This paper consists of 3 sections with one or two questions per section (sub-sets of instructions) each.
2. Each section relates to small semi-complete programs that need to be updated or finalised.
3. Each question relates to file(s) in one of the semi-complete programs that you need to update or finalise.
4. Each sub-sets of instructions relate to activities and tasks in one of the files.
5. Answer all the questions – there are no optional questions.
6. Please read all questions, instructions and sub-sets of tasks very carefully.
7. After completing work on a relevant question, please upload the file(s) required to be uploaded to the correct upload area. In other words, each section will mention what files to upload and how to upload them.

The University of Pretoria commits itself to producing academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

SECTION A – API & APPLICATION SECURITY (30)

For Section A, you need to complete **two questions**; an **API question** and an **Application Security question**. *Both questions do not require access to an existing database file to be completed. In other words, the databases per question are generated through the programs (as demonstrated in lectures and done during assignments 2 and 3).*

However, to create the databases and test your completed programs **per question in Section A** you need to have MS SQL Server running. Further, to generate the databases through the programs you need to perform the following steps per question in **Section A** using **Visual Studio 2019**:

- Once the question's program loads, open the "**appsettings.json**" file in the Solution Explorer.
- Change and save the **Server** location, pointing to **your** *SQL Server Server Name*. See the example below.
- Example:`"DefaultConnection":`
  `"Server=YourOwnSqlServerName;Database=LeaveTheValueHereAsYouFoundIT;Trusted_Connection=True;MultipleActiveResultSets=true"`
- Next, open the Package Manager Console (**View** > **Other Windows** > **Package Manager Console**) and run each of the following 2 commands individually to create the database tables.
  - **add-migration initial**
  - **update-database**
- **Thereafter, you proceed to complete the individual questions for Section A.**

All the source files are in the following zipped files:

- **Examination1_SectionA_Question1.zip**
- **Examination1_SectionA_Question2.zip**

Once you have completed the 2 questions in this section, upload the ***HeroController.cs*** file and the ***AuthenticationController.cs*** file to the Section A upload slot.

Your task is to complete the codebase to get the applications to function as described below. For each question, please do the following:

- Read the instructions carefully.
- Apply the necessary code changes to the specified file(s) in the required application.
- Only upload the modified file(s) associated with the question to ClickUP.

(**IMPORTANT: DO NOT UPLOAD ZIP, TAR, RAR or SLN files)**

SECTION A - QUESTION 1 (API) (20 MARKS)

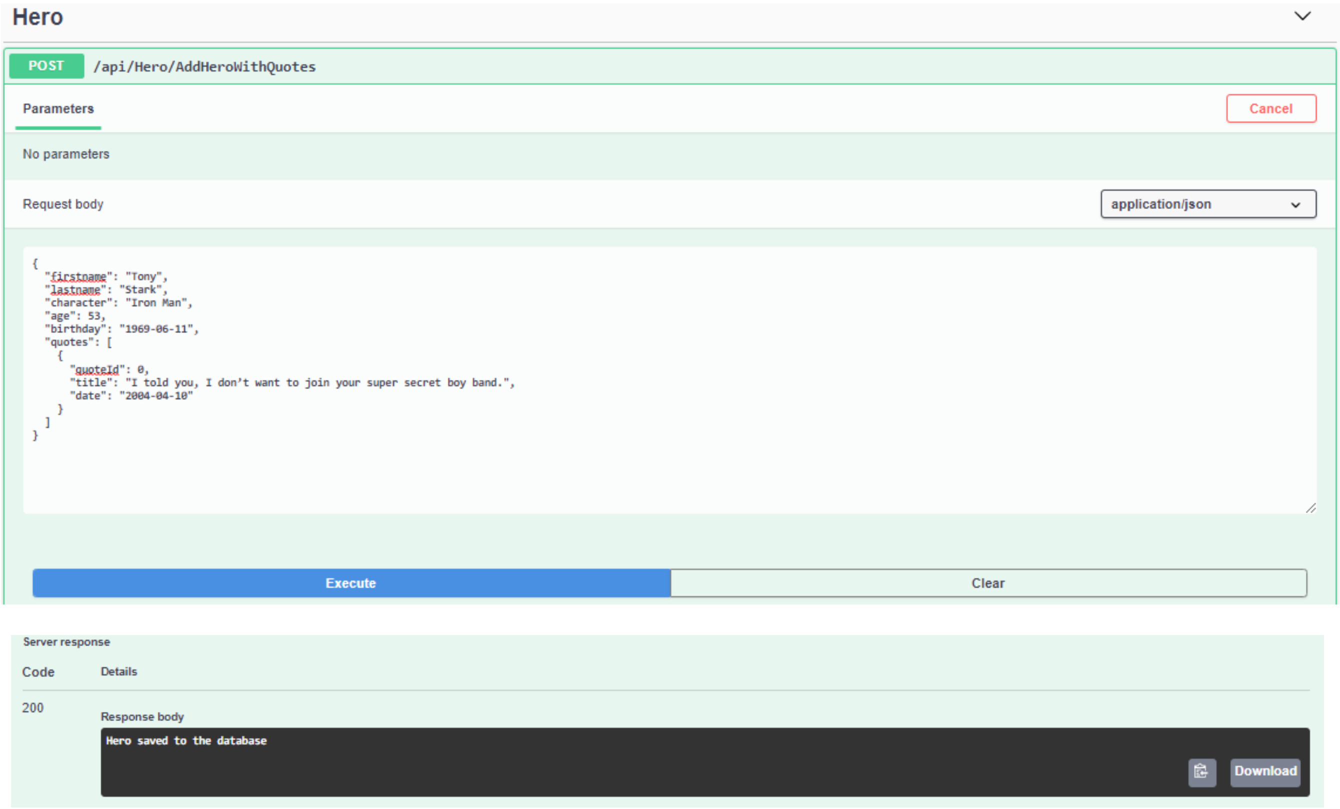
Only upload the ***HeroController.cs*** file after completing this question. **One file in total to upload**.

This question requires you to create an **API Controller** and name it ***HeroController*** in the "***Controllers***" folder. Thereafter, complete the functionality within the ***HeroController.cs*** file only to allow a **Hero** with the **Quote(s)** to be saved to the database.

- In the "***HeroController.cs***" **API Controller** you need to do the following **[20 Marks]**
  - The controller is a valid API controller with the "**api**" route prefix (3 marks).
  - Configure the **Controller** to inject (with valid dependency injection) the *Repository Model* through the constructor (3 marks).
  - Create the "***AddHeroWithQuotes***" Post endpoint/method to allow a **Hero** with the **Quotes** to be saved to the database. *Note: The endpoint/method* ***route*** *name should be the same as the endpoint/method name.* Furthermore, the endpoint/method is an asynchronous endpoint/method taking the "**HeroViewModel**" ViewModel as a parameter (5 marks).
  - Next, you need to save the data from the "**HeroViewModel**" ViewModel to the database asynchronously via the repository within a **try/catch** block (7 marks).
  - With a successful save to the database, the endpoint/method must return a **200 (Ok)** *IActionResult* response with the message "*Hero saved to the database*" (1 mark).
  - If any **Exceptions** happen during the save it should be *caught* and a **400 (BadRequest)** *IActionResult* response with the message "*Invalid Transaction*" should be returned (1 mark).
- Once you are done and have successfully implemented the required code, you can run and test the application and see the **Q1 Expected Output** example displayed below.

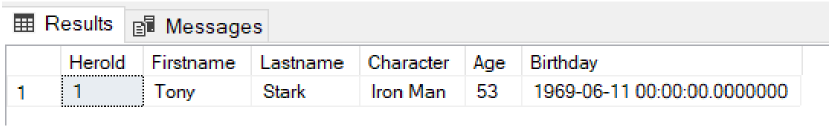
- **NB: The application was created using Visual Studio 2019. You have already been given a document for any environment pre-setup installations required and how to do it, shared on the Course Module ClickUP site. Therefore, for this question, no additional environment pre-setup installations are needed.**

Q1 Expected Output

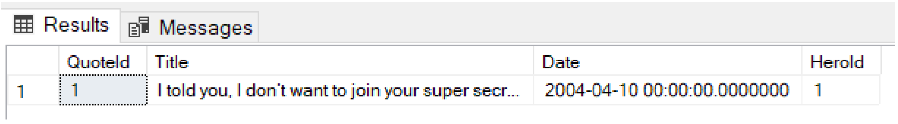


```
{
  "firstname": "Tony",
  "lastname": "Stark",
  "character": "Iron Man",
  "age": 53,
  "birthday": "1969-06-11",
  "quotes": [
    {
      "quoteId": 0,
      "title": "I told you, I don’t want to join your super secret boy band.",
      "date": "2004-04-10"
    }
  ]
}
```

```
Hero saved to the database
```



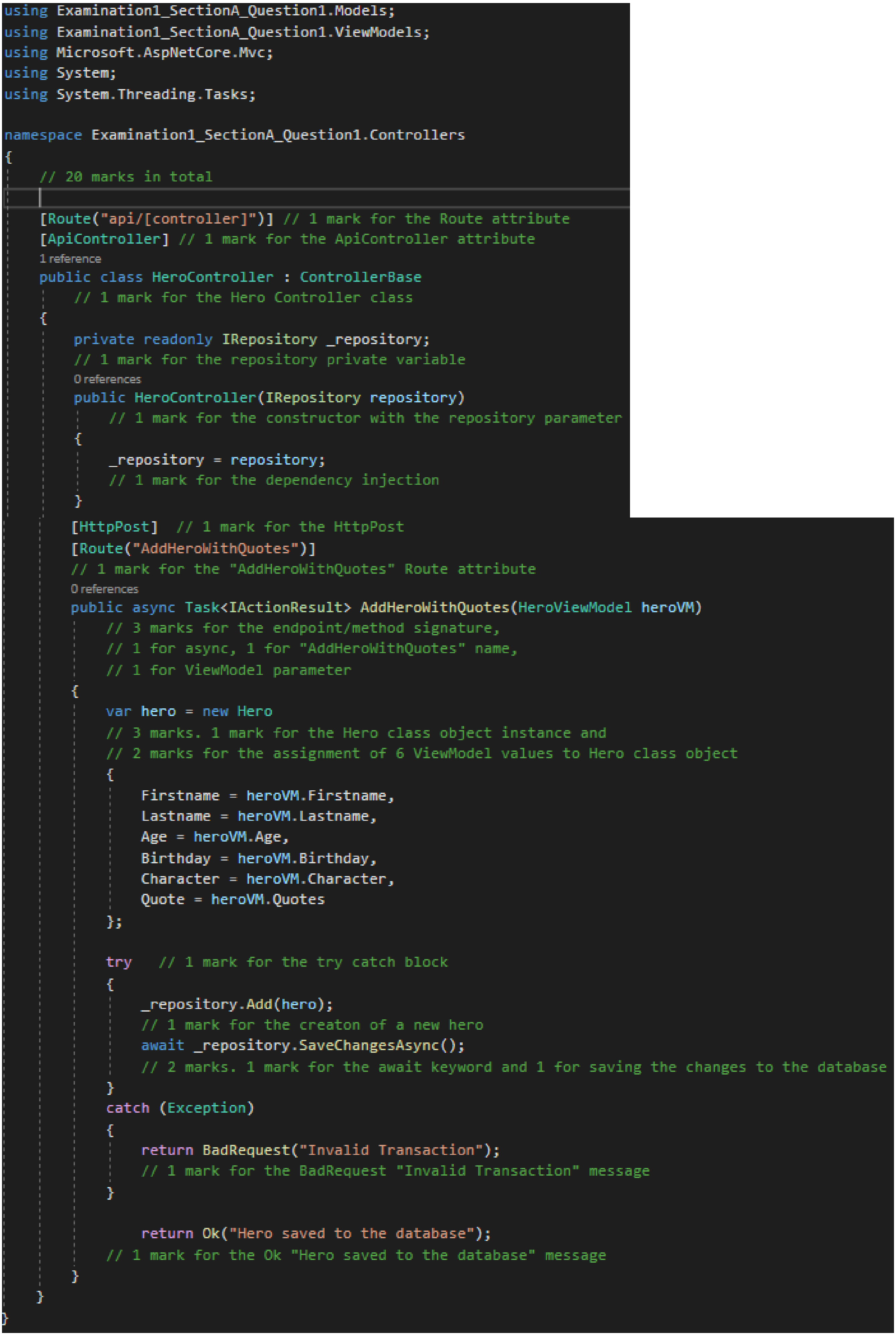
	HeroId	Firstname	Lastname	Character	Age	Birthday
1	1	Tony	Stark	Iron Man	53	1969-06-11 00:00:00.0000000



	QuoteId	Title	Date	HeroId
1	1	I told you, I don’t want to join your super secr...	2004-04-10 00:00:00.0000000	1

SOLUTION SECTION A – QUESTION 1

File: HeroController.cs



```
using Examination1_SectionA_Question1.Models;
using Examination1_SectionA_Question1.ViewModels;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Threading.Tasks;

namespace Examination1_SectionA_Question1.Controllers
{
    // 20 marks in total

    [Route("api/[controller]")] // 1 mark for the Route attribute
    [ApiController] // 1 mark for the ApiController attribute
    1 reference
    public class HeroController : ControllerBase
        // 1 mark for the Hero Controller class
    {
        private readonly IRepository _repository;
        // 1 mark for the repository private variable
        0 references
        public HeroController(IRepository repository)
            // 1 mark for the constructor with the repository parameter
        {
            _repository = repository;
            // 1 mark for the dependency injection
        }
        [HttpPost]  // 1 mark for the HttpPost
        [Route("AddHeroWithQuotes")]
        // 1 mark for the "AddHeroWithQuotes" Route attribute
        0 references
        public async Task<IActionResult> AddHeroWithQuotes(HeroViewModel heroVM)
            // 3 marks for the endpoint/method signature,
            // 1 for async, 1 for "AddHeroWithQuotes" name,
            // 1 for ViewModel parameter
        {
            var hero = new Hero
            // 3 marks. 1 mark for the Hero class object instance and
            // 2 marks for the assignment of 6 ViewModel values to Hero class object
            {
                Firstname = heroVM.Firstname,
                Lastname = heroVM.Lastname,
                Age = heroVM.Age,
                Birthday = heroVM.Birthday,
                Character = heroVM.Character,
                Quote = heroVM.Quotes
            };

            try   // 1 mark for the try catch block
            {
                _repository.Add(hero);
                // 1 mark for the creaton of a new hero
                await _repository.SaveChangesAsync();
                // 2 marks. 1 mark for the await keyword and 1 for saving the changes to the database
            }
            catch (Exception)
            {
                return BadRequest("Invalid Transaction");
                // 1 mark for the BadRequest "Invalid Transaction" message
            }

                return Ok("Hero saved to the database");
            // 1 mark for the Ok "Hero saved to the database" message
        }
    }
}
```

SECTION A - QUESTION 2 (APPLICATION SECURITY) (10 MARKS)

Only upload the ***AuthenticationController.cs*** file after completing this question. **One file in total to upload**.

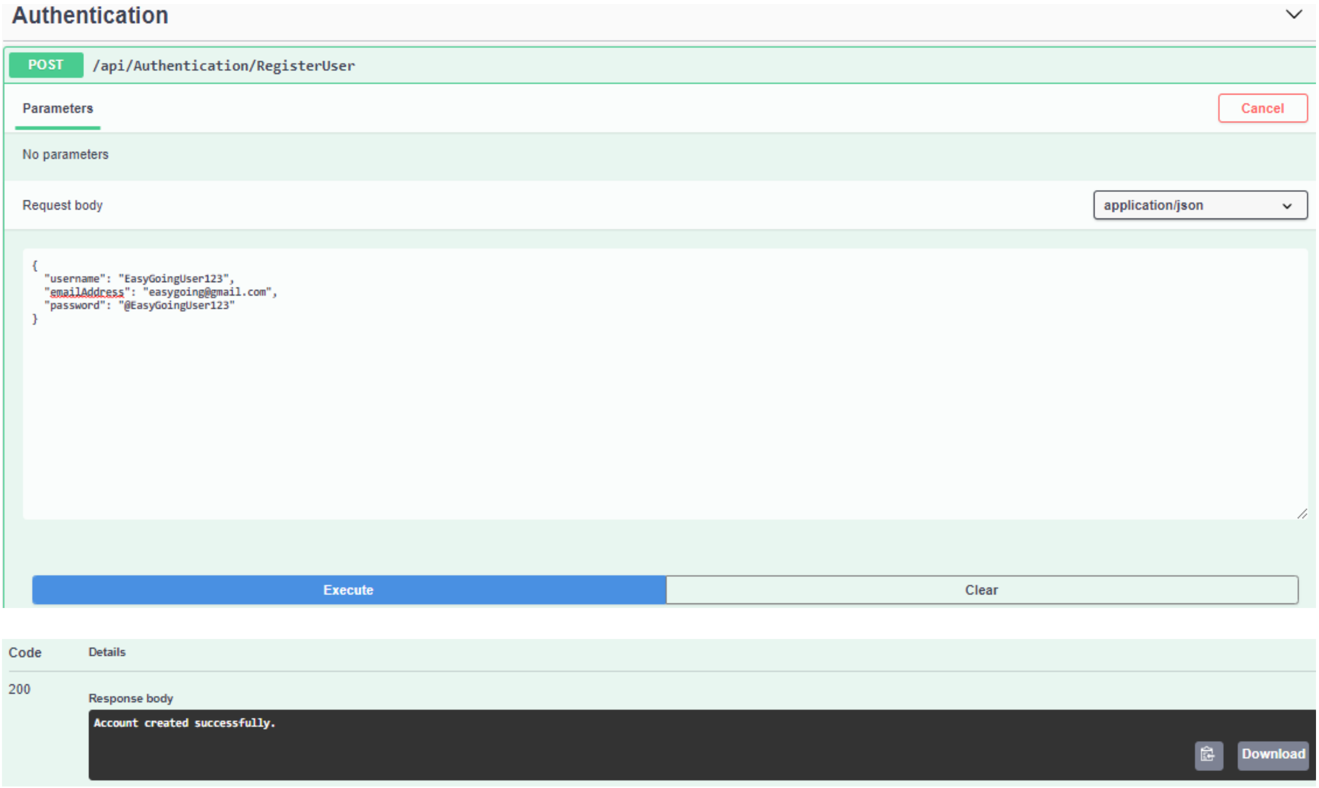
This question requires you to add functionality to **Register a User** using *ASPNetCore Identity* in the already existing "**AuthenticationController.cs**" file in the "***Controllers***" folder.

- In the "**AuthenticationController.cs**" **API Controller** you need to do the following **[10 Marks]**
  - Create the "***RegisterUser***" Post endpoint/method to allow a **New User** to be saved to the database. *Note: The endpoint/method **route** name should be the same as the endpoint/method name.* Furthermore, the endpoint/method is an asynchronous endpoint/method taking the "**UserViewModel**" ViewModel as a parameter (3 marks).
  - Next, you need to **Find** in the database the **Username** passed from the "**UserViewModel**" ViewModel. *Note: you need to use the already instantiated ASPNetCore Identity **userManager*** (1 mark)***.***
  - If the user does not already exist assign the user to a new **AppUser** and **Create** the **New User** asynchronously in the database returning a **200 (Ok)** *IActionResult* response with the message "*Account created successfully.*" (4 marks).
  - If any **errors** are generated during the **New User** creation a **500 (InternalServerError)** *IActionResult* response with the message "*Internal Server Error. Please contact support.*" should be returned (1 mark).
  - If the user already exists return a **403 (Forbidden)** *IActionResult* response with the message "*Account already exists.*" (1 mark).

- Once you are done and have successfully implemented the required code, you can run and test the application and see the **Q2 Expected Output** example displayed below.

- **NB: The application was created using Visual Studio 2019. You have already been given a document for any environment pre-setup installations required and how to do it, shared on the Course Module ClickUP site. Therefore, for this question, no additional environment pre-setup installations are needed.**

Q2 Expected Output



Authentication

POST /api/Authentication/RegisterUser

Parameters

No parameters

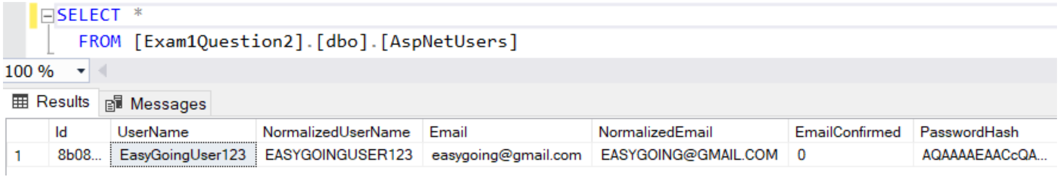
Request body: application/json

```
{
  "username": "EasyGoingUser123",
  "emailAddress": "easygoing@gmail.com",
  "password": "@EasyGoingUser123"
}
```

Code: 200

Response body:

```
Account created successfully.
```

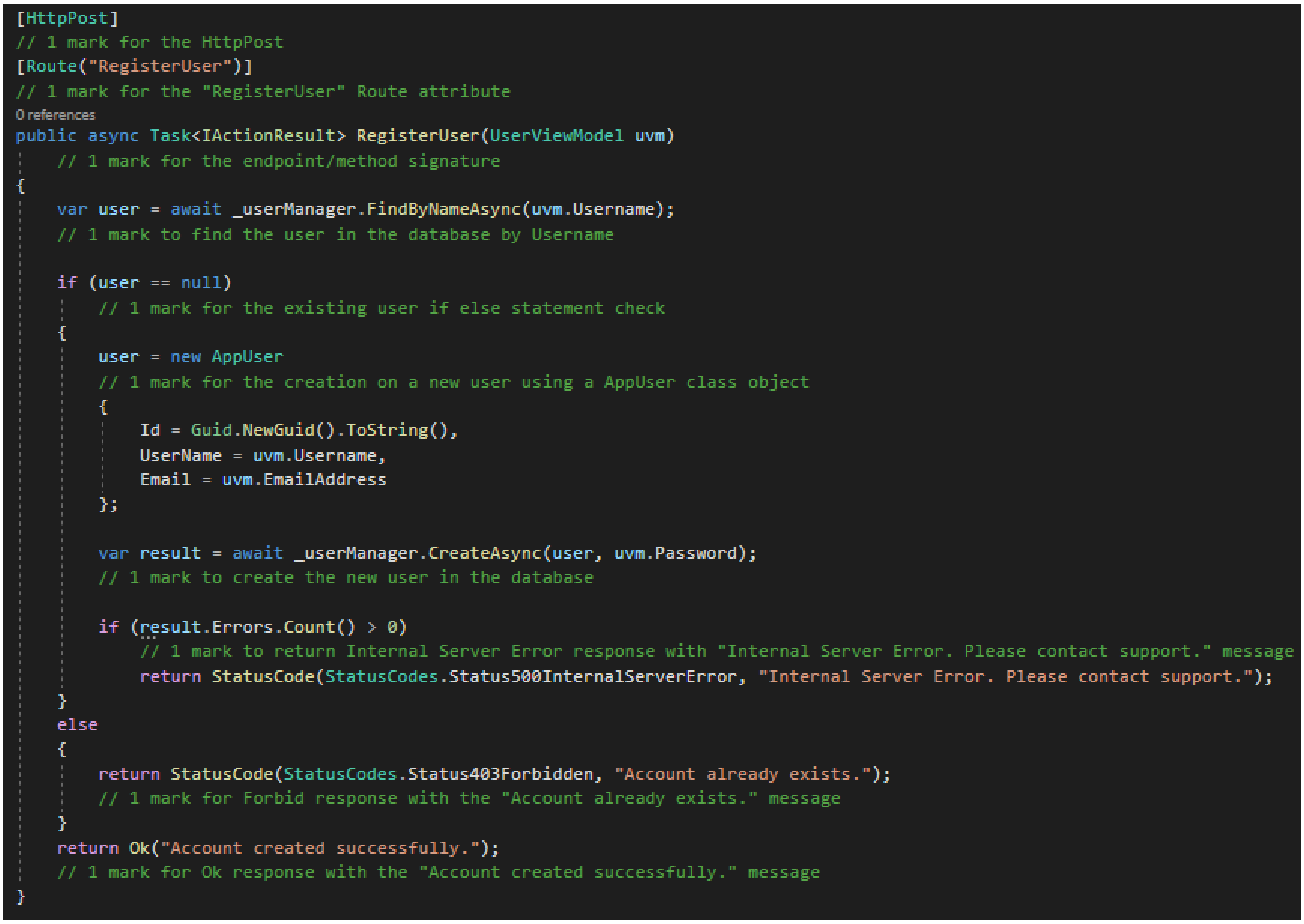


```
SELECT *
  FROM [Exam1Question2].[dbo].[AspNetUsers]
```

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
1	8b08...	EasyGoingUser123	EASYGOINGUSER123	easygoing@gmail.com	EASYGOING@GMAIL.COM	0	AQAAAAEAACcQA...

SOLUTION SECTION A – QUESTION 2

File: AuthenticationController.cs



```
[HttpPost]
// 1 mark for the HttpPost
[Route("RegisterUser")]
// 1 mark for the "RegisterUser" Route attribute
0 references
public async Task<IActionResult> RegisterUser(UserViewModel uvm)
    // 1 mark for the endpoint/method signature
{
    var user = await _userManager.FindByNameAsync(uvm.Username);
    // 1 mark to find the user in the database by Username

    if (user == null)
        // 1 mark for the existing user if else statement check
    {
        user = new AppUser
        // 1 mark for the creation on a new user using a AppUser class object
        {
            Id = Guid.NewGuid().ToString(),
            UserName = uvm.Username,
            Email = uvm.EmailAddress
        };

        var result = await _userManager.CreateAsync(user, uvm.Password);
        // 1 mark to create the new user in the database

        if (result.Errors.Count() > 0)
            // 1 mark to return Internal Server Error response with "Internal Server Error. Please contact support." message
            return StatusCode(StatusCodes.Status500InternalServerError, "Internal Server Error. Please contact support.");
    }
    else
    {
        return StatusCode(StatusCodes.Status403Forbidden, "Account already exists.");
        // 1 mark for Forbid response with the "Account already exists." message
    }
    return Ok("Account created successfully.");
    // 1 mark for Ok response with the "Account created successfully." message
}
```

SECTION A TOTAL **30**

SECTION B – REPORTING (25)

For Section B, you need to complete **two questions**; both are reporting (using Ionic) **questions**. *Both questions do not require access to an existing database file to be completed.* It also does not require you to integrate any external libraries. All the source files are in the following zipped file:

Examination1_SectionB_Question.zip

Once you have completed the 2 questions in this section, **upload 2 files**, the **home.page.ts** file and the **home.page.html** file to the **Section B upload slot**.

Your task is to complete the codebase to get the applications to function as described below. For each question, please do the following:

- Read the instructions carefully.
- After downloading and extracting the zip file with the source, run it and see what it looks like before adding new functionality.
- Apply the necessary code changes to the specified file(s) in the required application.
- Only upload the modified file(s) associated with the question to ClickUP.

(IMPORTANT: DO NOT UPLOAD ZIP, TAR, RAR or SLN files)

SECTION B - QUESTION 1 (20 MARKS)

Only upload the ***home.page.ts*** file after completing this question. **One file in total to upload**.

A fast-food company has created a mobile application that allows for a dashboard view of the company stats. To allow the CEO to have quick and convenient real-time data, they have tasked you to build a mobile application that allows the CEO to have an overview of the company's current sales. Specifically, to have a chart and table view of the sales data. Your job is to build an application that allows the CEO to click on a button that will open an action sheet that will allow them to view the sales data, after which a bar chart and table view of the data should be presented on the screen and lastly, the CEO should be able to download a PDF version of the chart data.

You are given an incomplete project and you will be required to complete the code in the **home.page.ts** file under the provided function declarations. You will need to present an action sheet to the CEO where they will be able to select the option to show the sales data, upon selecting that option the chart data should be displayed. You will then be required to generate the chart data from the data provided in the **productData.ts** file as well as populate the table with the same data. Lastly, you are required to complete the function that will allow the CEO to download a pdf of the chart that is displayed on the screen.

You must complete the code that is in the **home.page.ts** file with the specific instructions given for the following functions:

- **1.1 presentActionSheet [5 Marks]**
  - In this function (**presentActionSheet**) you are required to create an action sheet controller, making use of the relevant required imports.
  - Your action sheet's heading should read '**Data Action Sheet**' and should have 2 buttons (1 mark).
  - The first button should read '**Show Data**' and include an icon with the name '**analytics-outline**'. The handler for this button should make a call to the **generateReport** function.
  - The second button should be a cancel button with text that reads '**Cancel**', include an icon with the name '**close**' and have a role of '**cancel**' (3 marks in total for both button requirements).
  - You will then be required to **present** the *action sheet controller* that you created (1 mark).

- **1.2 generateReport [10 Marks]**
  - In this function (**generateReport**) you will be required to format the chart data and labels by making use of the data in the **productData.ts** file. Bootstrap has already been linked to the project via the **index.html** page.
  - You are required to separate the **productData** object into *2 arrays*, one of which will contain all the ***names*** and the other containing all the **values**.
  - You are required to make use of the products array variable that has been declared at the top of the **home.page.ts** file to store the **names** (2 marks).
  - You are required to make use of the values array variable that has been declared at the top of the **home.page.ts** file to store the **values** (2 marks).

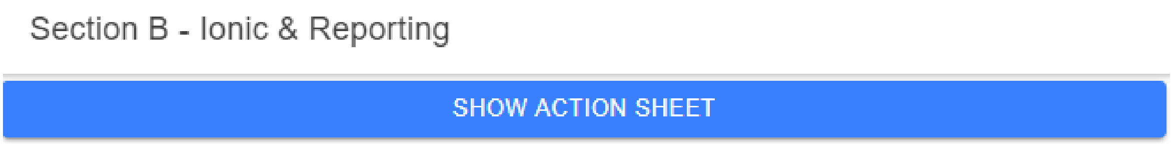
  - You will then need to **push** the values array variable to the **barChartData** variable, with the *data* property being set using the **values** *array variable*, the *label* property being set as the word '**Sales**' and including a property for the *backgroundColor* and setting it to '**rgb(0,183,255)**' (4 marks).
  - The **barChartLabels** variable should be set to the **products** array variable that you previously populated (1 mark).
  - Lastly, you are required to make a call to the **generateTable** function (1 mark).

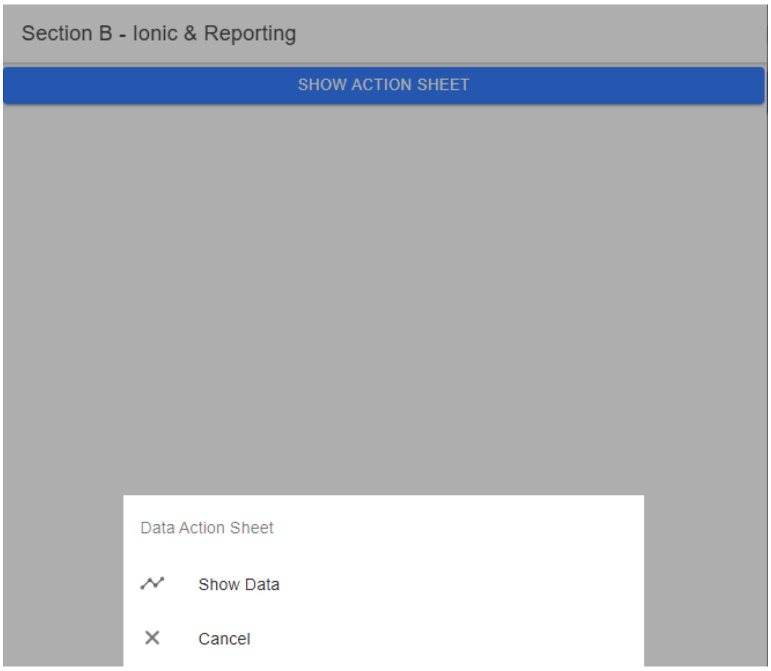
- **1.3 openPDF [5 Marks]**
  - In this function (**openPDF**) you are required to complete the code provided that will allow for the user to download a pdf of the chart with an id of '**htmlData**'.
  - You are required to create a PDF variable that will be a **jsPDF** object with the following specifications: **portrait** orientation, **mm** as the unit as well as the page size being **a4** (1 mark).
  - You will need to create a **topPosition** variable and set the value to **10** (1 mark).
  - You will need to create a **leftPosition** variable and set the value to **0** (1 mark).
  - You are then required to populate the **addImage** property of the *PDF variable* that you had previously created. You will need to populate this with the provided **contentDataUrl** variable, '**PNG**', **leftPosition** variable, **topPosition** as well as the provided **fileWidth** and **fileHeight** variables (1 mark).
  - Lastly, you will be required to *save the PDF* using the name '**Graph.pdf**' (1 mark).

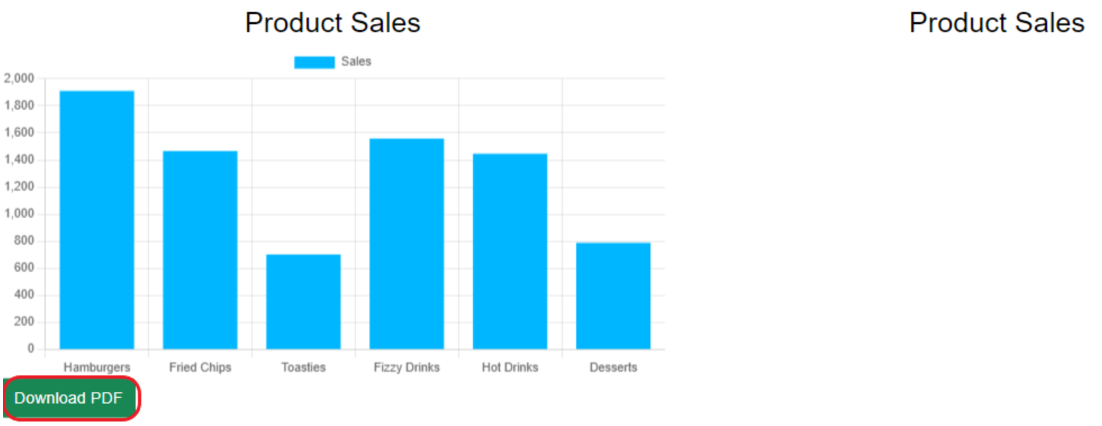
- Once you are done and have successfully implemented the required code, you can run the application and see the **Q1 Expected Output** displayed below.

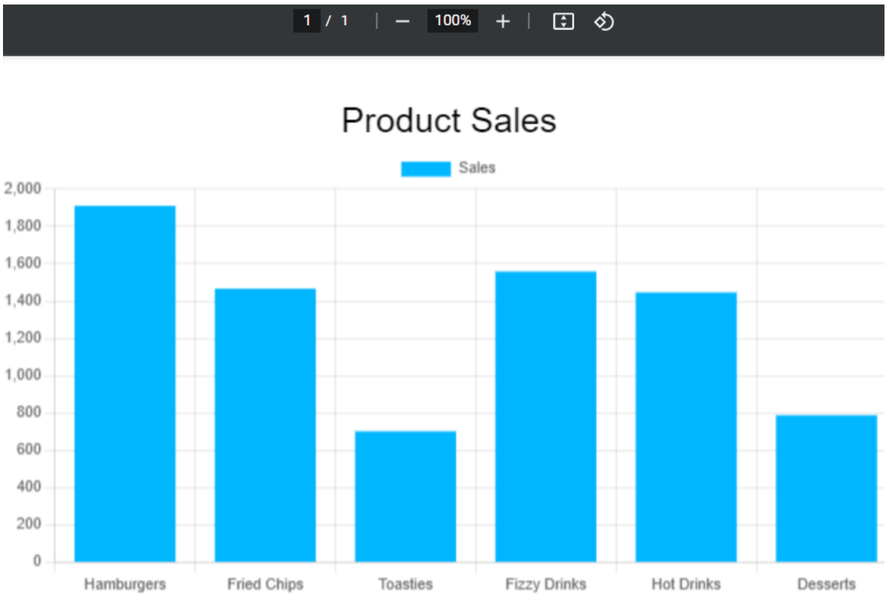
- **NB: Do not forget to run "npm install" to allow the application packages to be re-installed.**
- **NB: The application was created using Visual Studio Code and Ionic. You have already been given a document for any environment pre-setup installations required and how to do it, shared on the Course Module ClickUP site. Therefore, for this question, no additional environment pre-setup installations are needed**.

Q1 Expected Output



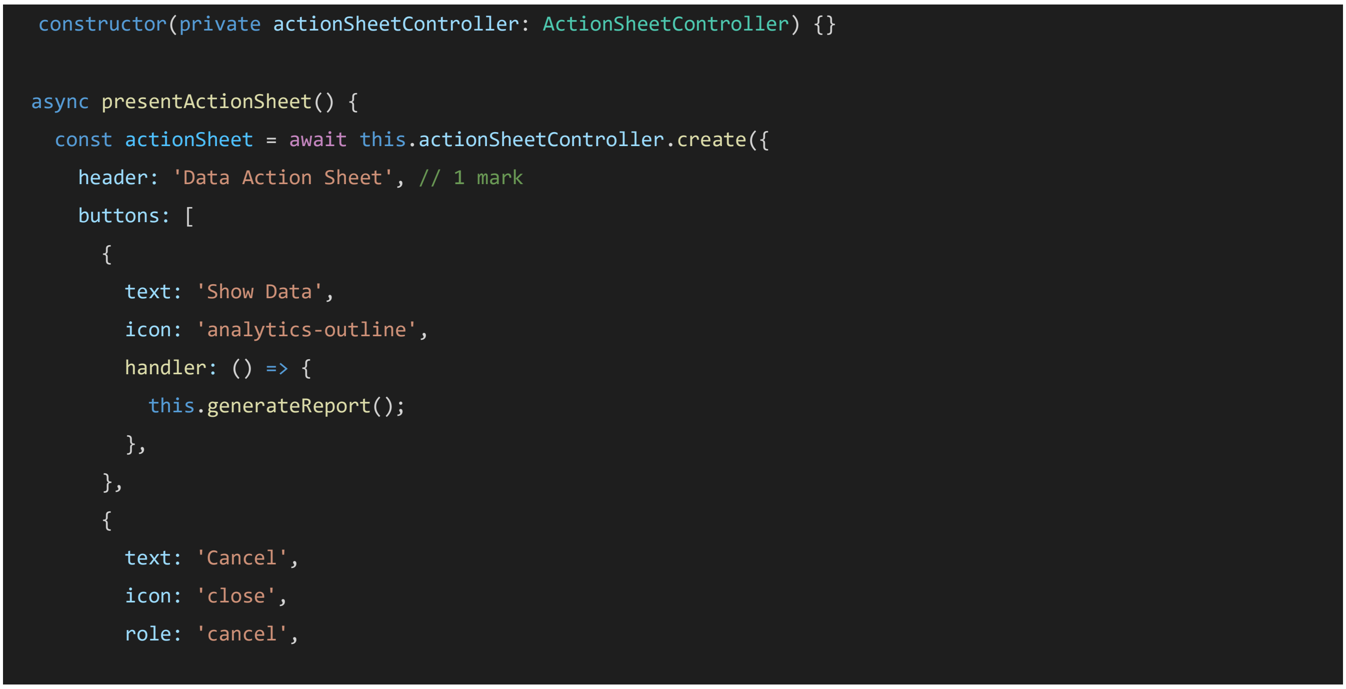






SOLUTION SECTION B – QUESTION 1

File: home.page.ts



```
 constructor(private actionSheetController: ActionSheetController) {}

async presentActionSheet() {
  const actionSheet = await this.actionSheetController.create({
    header: 'Data Action Sheet', // 1 mark
    buttons: [
      {
        text: 'Show Data',
        icon: 'analytics-outline',
        handler: () => {
          this.generateReport();
        },
      },
      {
        text: 'Cancel',
        icon: 'close',
        role: 'cancel',
      },
    ],
  }); // 3 marks - generateReport (1 mark), button text (1 mark), button icon (1 mark)
  await actionSheet.present(); // 1 mark
}

generateReport() {
  this.showData = true;

  // Restructure data for chart
  this.products = productData.map((x) => x.name); // 2 marks
  this.values = productData.map((x) => x.value); // 2 marks

  // Generate Chart
  this.barChartData.push({
    data: this.values,
    label: 'Sales',
    backgroundColor: 'rgb(0,183,255)',
  }); // 4 marks - 1 mark for each line

  this.barChartLabels = this.products; // 1 mark

  // Generate table function
  this.generateTable(); // 1 mark
}

openPDF(){
  let Data = document.getElementById('htmlData')!;
  // Canvas Options
  html2canvas(Data).then((canvas) => {

    let fileWidth = 210;
    let fileHeight = (canvas.height * fileWidth) / canvas.width;

    const contentDataURL = canvas.toDataURL('image/png');

    let PDF = new jsPDF({ orientation: 'p', unit: 'mm', format: 'a4' }); // 1 mark
    let topPosition = 10; // 1 mark
    let leftPosition = 0; // 1 mark
    PDF.addImage(
      contentDataURL,
      'PNG',
      leftPosition,
      topPosition,
      fileWidth,
      fileHeight
    ); // 1 mark
    PDF.save('Graph.pdf'); // 1 mark
  });
}
```

SECTION B - QUESTION 2 (5 MARKS)

Only upload the ***home.page.html*** file after completing this question. **One file in total to upload**.

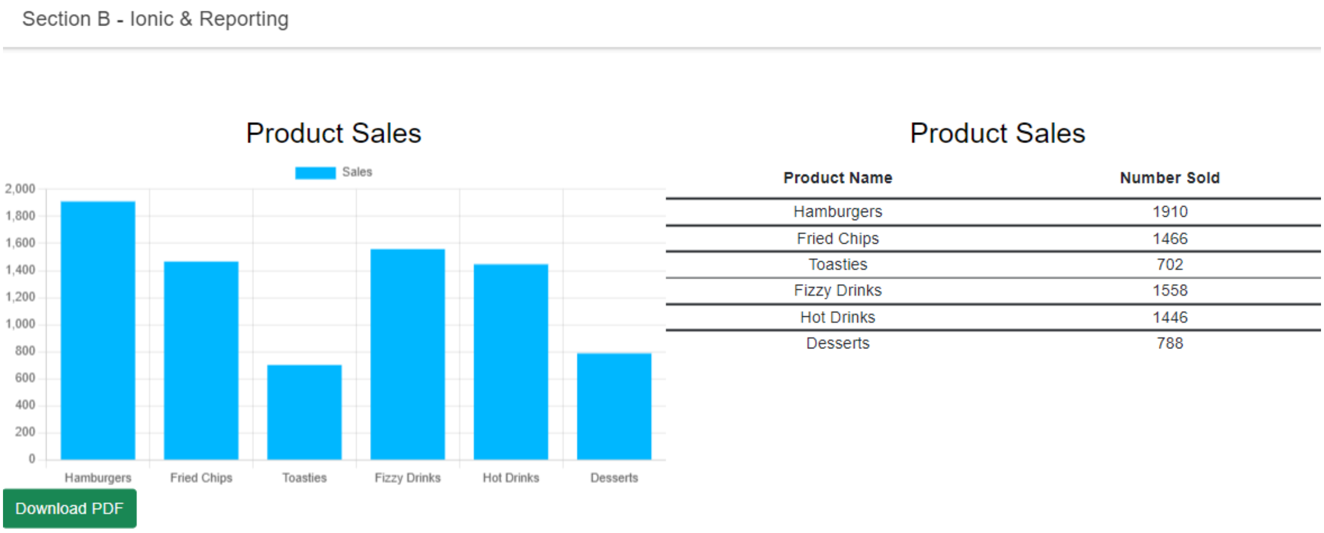
You are required to complete the **home.page.html** file for the fast-food company mentioned above. You will be making use of the same project as in Question 1 and are required to only edit the **home.page.html** file. You will need to complete the code so that the table data can be displayed.

- **home.page.html [5 Marks]**
  - In this file, you are required to complete the code provided that will allow for the user to have a table view of the data that is in the **productData.ts** file.
  - You are required to create an HTML table with the headings '***Product Name***' and '***Number Sold***' (2 marks).
  - You are then required to make use of the **tableData** variable (*created in Question 1*) to populate the rows of the table (1 mark).
  - The **name** associated with the product should be displayed in one cell and the **value** associated with the product should be displayed in one cell (2 marks).

- Once you are done and have successfully implemented the required code, you can run the application and see the **Q2 Expected Output** displayed below.

- **NB: Do not forget to run "npm install" to allow the application packages to be re-installed.**
- **NB: The application was created using Visual Studio Code and Ionic. You have already been given a document for any environment pre-setup installations required and how to do it, shared on the Course Module ClickUP site. Therefore, for this question, no additional environment pre-setup installations are needed**.

Q2 Expected Output



Section B - Ionic & Reporting

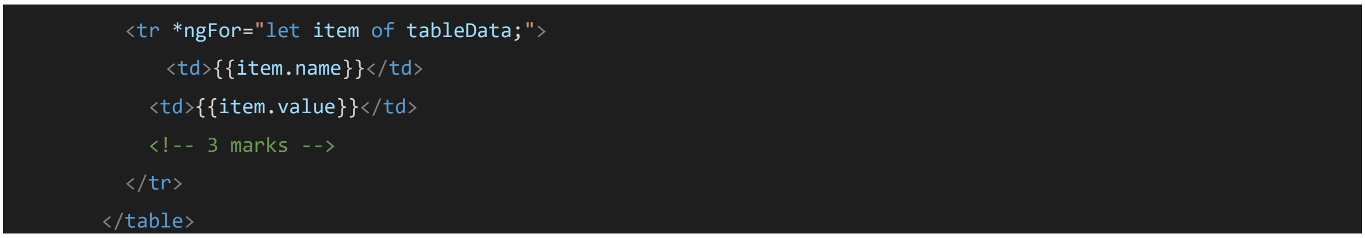
Product Sales

Product Name	Number Sold
Hamburgers	1910
Fried Chips	1466
Toasties	702
Fizzy Drinks	1558
Hot Drinks	1446
Desserts	788

SOLUTION SECTION B – QUESTION 2

File: home.page.html

```
          <table class="table">
          <thead class="thead">
          <tr>
            <th>Product Name</th>
            <th>Number Sold</th>
            <!-- 2 marks -->
          </tr>
        </thead>
```



```
  <tr *ngFor="let item of tableData;">
     <td>{{item.name}}</td>
    <td>{{item.value}}</td>
    <!-- 3 marks -->
  </tr>
</table>
```

SECTION B TOTAL **25**

SECTION C – ADVANCED CONCEPTS (25 MARKS)

For Section C, you need to complete one question. The question does not require access to an existing database file to be completed. It also does not require you to integrate any additional libraries, besides all the ones that were included as part of the document for any environment pre-setup installations required. All the source files are in the following zipped file:

Examination1_SectionC_Question.zip

Once you have completed the 1 question in this section, **upload 4 files,** the **nav-bar.component.ts**, **nav-bar.component.html**, **app.component.ts**, and **app.component.html** files to the **Section C upload slot**.

Your task is to complete the codebase to get the applications to function as described below. For this question, please do the following:

- Read the instructions carefully.
- After downloading and extracting the zip file with the source, run it and see what it looks like before adding new functionality.
- Apply the necessary code changes to the specified file(s) in the required application.
- Only upload the modified file(s) associated with the question to ClickUP.

(**IMPORTANT: DO NOT UPLOAD ZIP, TAR, RAR or SLN files)**

SECTION C - QUESTION 1 (25 MARKS)

Only upload the **nav-bar.component.ts**, **nav-bar.component.html**, **app.component.ts**, and **app.component.html** files after completing this question. **Four files in total to upload**.

The Pomodoro technique is a time management technique popularized in the 1980s. When solving a task using the technique, one uses a timer to solve/execute a task using the following steps:

1. Set the timer to 25 minutes and work on the task.
2. If the 25 minutes is over, set the timer to 10 and take a break.
3. If the 10 minutes is over and the task is not complete, go to step 1; otherwise, you're finished.

Your job is to complete a Pomodoro timer application that can be used to alternate between the "**working**" and "**on break**" states. The application will be shared with your fellow university students to help them prepare for their exams. Since you may appreciate a reward, you also have to implement a way in which they can *donate R100 to your account*, should they wish.

You are given a project that has the necessary angular components and some of them are not complete. You must edit the **app.component.ts** file to keep track of the user's status, the **navbar** component to reset the status, and **app.component.html** file to implement the donate functionality. When you run the downloaded code, before and after your changes, you will see Figures 1 and 2 respectively.

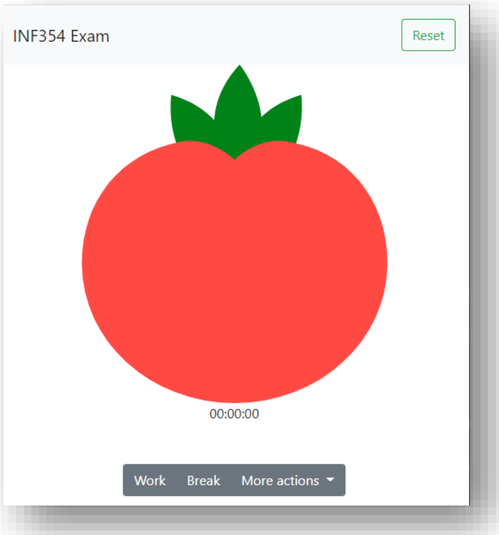


Figure 1: Screenshot of what the Pomodoro timer will look like before answering the following questions.

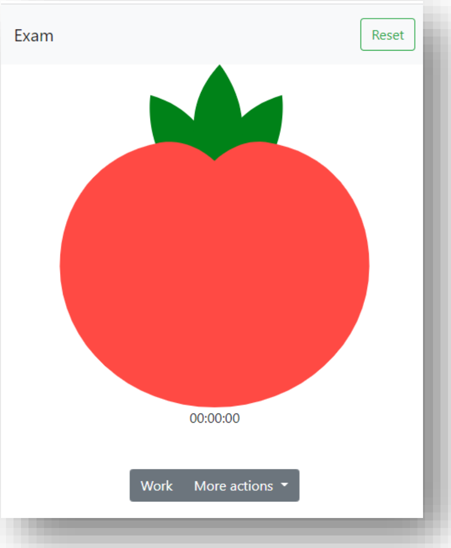


Figure 2: Screenshot of Pomodoro timer after you have answered the following questions.

- Timer status tracking in **app.component.ts**: **[4 Marks]**
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status and it is initialized to **OnBreak**. When **StartWorkSession()** is called, change **currentStatus** to **Working** (1 mark).
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status and it is initialized to **OnBreak**. When **StartBreak()** is called, set **currentStatus** to **OnBreak** (1 mark).
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status and it is initialized to **OnBreak**. When **resetPomodoroSessions()** is called, set **currentStatus** to **OnBreak** (1 mark).
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status and it is initialized to **OnBreak**. When **isWorking()** is called, check if **currentStatus** is equal to **Working** (1 mark).

- Timer status reset in **navbar** component: **[7 Marks]**
  - The **nav-bar.component.html** file defines a button labelled "*Reset*", change the HTML such that when the button is clicked the method **onResetButtonClicked** of **nav-bar.component.ts** will be called (1 mark).
  - The **nav-bar.component.ts** file defines a method called **onResetButtonClicked**, change its body and use an **EventEmitter** to notify the **navbar**'s parent component (i.e., **AppComponent**) to call **resetPomodoroSessions** (lines 38-47 of **app.component.ts**) (6 marks).

- Actions buttons in **app.component.html**: **[14 Marks]**
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status. If the status is **Working** then hide the button labelled '*Work*' (line 14 in **app.component.html**) (2 marks).
  - The **AppComponent** has a variable called **currentStatus** to monitor the current timer's status. If the status is not **Working** then hide the button labelled '*Break*' (line 16 in **app.component.html**) (2 marks).
  - When the button labelled '*Donate R100!*' is clicked (in **app.component.html**), use the HTML form post method and PayFast's sandbox (**https://sandbox.payfast.co.za/eng/process**) to donate/pay 100 South African Rands. Use the merchant id **10000100**, merchant key **46f0cd694581a**, and item description "**Pomodoro subscription**" (10 marks).

- **NB: Do not forget to run "npm install" to allow the application packages to be re-installed.**
- **You will not be penalized if PayFast is down and/or refusing your form post, as long as you use the details given above.**
- **NB: The application was created using Visual Studio Code and Angular. You have already been given a document for any environment pre-setup installations required and how to do it, shared on the Course Module ClickUP site.**

SOLUTION SECTION C – QUESTION 1

File: app.component.ts

```
   isWorking () : boolean {
    // 1 mark for checking if current status is equal to working
    return this.currentStatus == PomodoroStatus.Working;
  }

  StartWorkSession() {
    this.timer.stop();
    this.timer.start({countdown: true, startValues: {minutes: 25}});
    // 1 mark for setting current status to working
    this.currentStatus = PomodoroStatus.Working;
  }

  StartBreak() {
    this.timer.stop();
    this.timer.start({countdown: true, startValues: {minutes: 10}});
    // 1 mark for setting current status to onBreak
    this.currentStatus = PomodoroStatus.OnBreak;
  }

  resetPomodoroSessions() {
    // 1 mark for setting current status to onBreak
    this.currentStatus = PomodoroStatus.OnBreak;

    this.timer = new Timer();
    this.timer.addEventListener('secondsUpdated',  (e) => {
      this.currentTimeSlot = this.timer.getTimeValues().toString();
    })
    this.timer.addEventListener('targetAchieved', e=> { this.playNotification();})
  }
```

File: nav-bar.component.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand">Exam</a>
    <!-- 2 marks for catching the click event-->
    <button class="btn btn-outline-success my-2 my-sm-0" (click)="onResetButtonClicked()">Reset</button>
</nav>
```

File: nav-bar.component.ts

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
// Note: if a student uses a service, an eventbus lib, or any other implementation
// for communicating the 'reset' event then do not penalize them. The marking distrubution should be allocated as follows:
// 2 marks for emitting the event, 2 marks for notifying the parent and calling resetPomodoroSessions,
// and 2 marks for the event transport mechanism
@Component({
  selector: 'app-nav-bar',
  templateUrl: './nav-bar.component.html'
})
```

```
export class NavBarComponent implements OnInit {

  @Output() notifyResetClicked: EventEmitter<any> = new EventEmitter(); // 2 marks

  constructor() { }

  ngOnInit(): void {
  }

  onResetButtonClicked() {
    this.notifyResetClicked.emit(); // 2 marks
  }
}
```

File: app.component.html

```
<!--2 marks for catching notifyResetClicked event-->
<app-nav-bar (notifyResetClicked)="resetPomodoroSessions()"></app-nav-bar>
<div class="container my-auto">
  <div class="col text-center">

    <img src="assets/tomato.svg" width="400em">

    <div id="clock">
      <div class="values">{{currentTimeSlot}}</div>
    </div>

    <div class="btn-group mt-5" role="group" aria-label="SessionButtons">
      <!--2 marks for hiding work button -->
      <button type="button" class="btn btn-secondary" *ngIf="!isWorking()"
(click)="StartWorkSession()">Work</button>
      <!--2 marks for hiding break button -->
      <button type="button" class="btn btn-secondary" *ngIf="isWorking()" (click)="StartBreak()">Break</button>

      <button id="btnGroupDrop1" type="button" class="btn btn-secondary dropdown-toggle" data-toggle="dropdown"
aria-expanded="false">
        More actions
      </button>
      <div class="dropdown-menu" aria-labelledby="btnGroupDrop1">
        <!--2 marks for creating form with action and post-->
        <form action="https://sandbox.payfast.co.za/eng/process" method="post">
          <!--2 marks for creating hidden merchant_id input field-->
          <input type="hidden" name="merchant_id" value="10000100">
          <!--2 marks for creating hidden merchant_key = input field-->
          <input type="hidden" name="merchant_key" value="46f0cd694581a">
          <!--2 marks for creating hidden amount input field-->
          <input type="hidden" name="amount" value="100.00">
          <!--2 marks for creating hidden item_name input field-->
          <input type="hidden" name="item_name" value="Pomodoro subscription">
          <button class="btn btn-light" type="submit">Donate R100!</button>
       </form>
       </div>
```

```
    </div>
  </div>
</div>
```

SECTION C TOTAL **25**