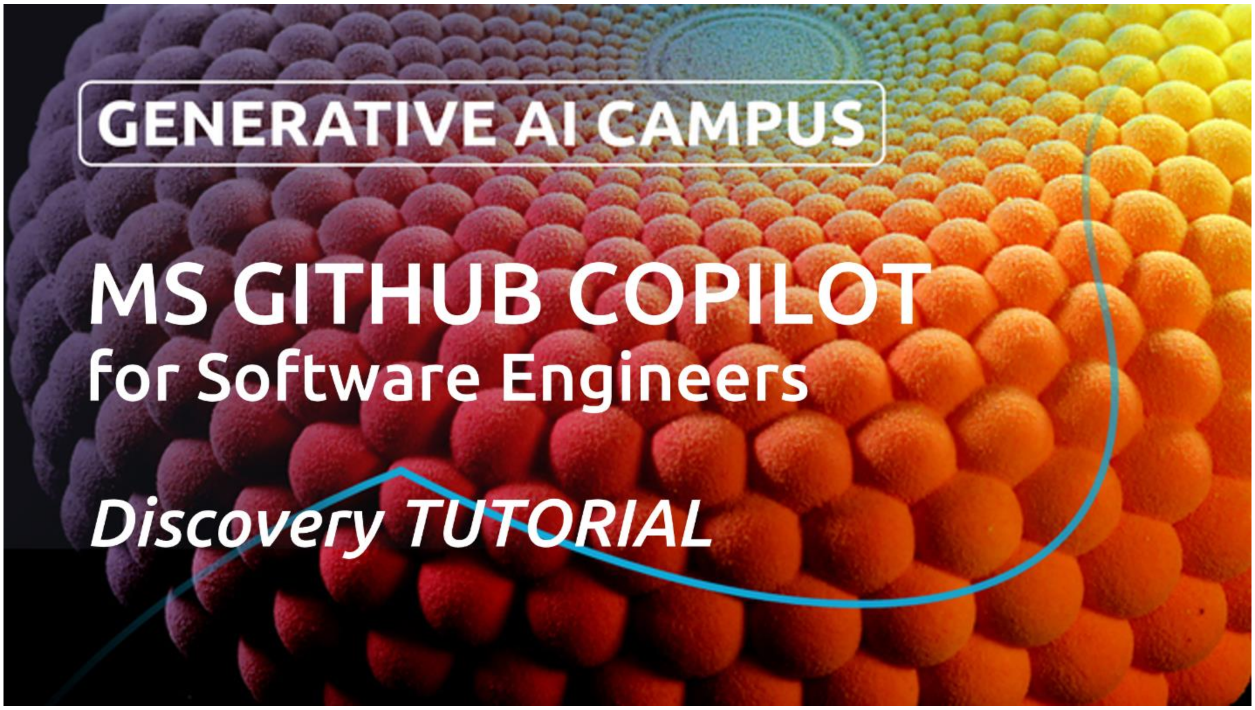


Table of Contents

Assignment 1 – Promotion Finder 2

- **Setup code in VsCode** 2
- **CLI(Command specific)** 2-4
- **Inline(Comment to code)** 4-16
- **Chat** 16-25
- **Feature Enhancement** 25-28

Assignment 1: Promotion Finder

I. Setup Code in VSCode

Please find code at the following link: Hands-on

The following versions of .NET and Angular will be required with access to MongoDb Atlas.

- .NET Version: .NET 7,
- Angular Version: v16,
- DB: Mongo Atlas.

II. Using Copilot in different versions as CLI, Inline Suggestions, and Chat Based.

a) CLI based Copilot:

1) Purpose: “Seeking methods or commands to terminate processes that are actively using files that have been deleted from the filesystem in interactive mode”

Prompt: gh copilot suggest -t shell 'kill processes with open files that have been deleted'

Response:

```
 # Explanation:

# To kill processes with open files that have been deleted, you can use the following command:

fuser -k -u <file_path>

# Explanation:

#  - fuser command is used to identify processes using files or sockets.

#  - -k option kills processes using the specified files.

#  - -u option specifies that processes accessing the file should be killed.

#  - <file_path> is the path of the file that has been deleted but is still open by some process.
```

2) Purpose: “Seeking clarification on the command "rm *" and its potential implications in file deletion”

Prompt: gh copilot explain "rm *"

Response:

```
# Explanation:
```

```
#   rm is used to remove files and directories.
#   * is a wildcard that matches any file or directory in the current directory.
```

3) Purpose: “Requesting practical examples or methods for terminating a process using its name in various operating systems or environments.”

Prompt: gh copilot explain "examples of how to kill a process by name"

Response:

```
# Explanation:

# • To kill a process by name, you can use the pkill command.

#  • pkill <process_name> kills all processes with the specified name.

# • Another option is to use the killall command.

#   • killall <process_name> kills all processes with the specified name.

# • If you want to manually find the process ID (PID) of a process by name and
then kill it, you can use the

#  following commands:

#   • pgrep <process_name> retrieves the PID of the process with the specified
name.

#   • kill <PID> kills the process with the specified PID.
```

4) Purpose: “Seeking guidance or commands to terminate the Vim text editor process on a computer system.”

Prompt: "examples of how to kill the process for vim"
Response:

```
# Explanation:

#  • To kill the process for Vim, you can use the following commands:

#   • pkill vim - This command sends a signal to terminate all Vim processes.

#   • killall vim - This command terminates all Vim processes by their names.

#   • kill <PID> - This command kills a specific Vim process identified by its
process ID (PID).

#     • -9 or --signal=9 can be added to forcefully terminate the process.
```

5) Purpose: “Inquiring about methods or commands to terminate processes that are still holding onto files that have been deleted, potentially causing issues or preventing disk space reclamation.”

Prompt: gh copilot suggest "Kill processes holding onto deleted files"
Response:

```
# Explanation:

#  To kill processes holding onto deleted files, you can use the following command:

#  • lsof is used to list open files and the processes that have them open.

#    • | grep deleted filters the output to only show files that have been marked as deleted.

#    • | awk '{print $2}' extracts the second column from the output, which contains the process IDs.

#    • | sort | uniq sorts the process IDs and removes duplicates.

#    • | xargs kill -9 sends a SIGKILL signal to each process ID to forcefully terminate it.
```

6) Purpose: “Seeking guidance or steps to create and set up a new Git repository for version control in interactive mode”

Prompt: gh copilot suggest -t shell "initialize a git repository"
Response:

```
#  Suggestion:

#  lsof +L1

#  ? Select an option

# > Copy command to clipboard

#  Explain command

#  Execute command

#  Revise command

# Command copied to clipboard!
```

b) <u>Inline based Copilot:</u>

1)

Purpose: To create a basic layout for the `Dashboard Component.`

Prompts: **make one paragraph element with class head displaying Promotion finder page and make two class called as space with two button and linking them respectively and also include a router-outlet.(In Dashboard.component.html)**

Response:

```

<div class="space">
  <button routerLink="/resp">Button 1</button>
  <button routerLink="/resp">Button 2</button>
</div>

<router-outlet></router-outlet>

```

2)

Purpose: Create heading with Styling.

Prompts: **create a welcome heading with good styling align at the centre of screen in html**

Response:

```
<div class="welcome-heading">Welcome</div>*

 <style>
    .welcome-heading {
        text-align: center;
        font-size: 36px;
        font-weight: bold;
        margin-top: 50vh;
        transform: translateY(-50%);
    }
</style>

<div class="welcome-heading">Welcome</div>



```

3)

Purpose: These methods manage pagination logic for a component.

Prompts: **write 4 methods one is called as ngOnChanges that will update the total number of pages based on changes in page size, and second is called as nextPage() it will increment the currentPage if it is less then total pages and emit the new page number and third is onPageChanges it will update current page and last 4th one is prevPage() it is use to decrement the current page of it is greater than 1**

Response:

```
@Component({
  selector: 'app-paginator',
  templateUrl: './paginator.component.html',
  styleUrls: ['./paginator.component.css']
})
export class PaginatorComponent implements OnChanges {
  @Input() pageSize: number = 10;
  @Input() totalItems: number = 0;
  @Output() pageChange = new EventEmitter<number>();
  currentPage: number = 1;
  totalPages: number = 10;

  ngOnChanges(): void {
    this.totalPages = Math.ceil(this.totalItems / this.pageSize);
  }

  nextPage(): void {
    if (this.currentPage < this.totalPages) {
      this.currentPage++;
      this.pageChange.emit(this.currentPage);
    }
  }

  onPageChange(pageNumber: number): void {
    this.currentPage = pageNumber;
    this.pageChange.emit(this.currentPage);
  }

  prevPage(): void {
    if (this.currentPage > 1) {
      this.currentPage--;
      this.pageChange.emit(this.currentPage);
    }
  }
}


}


```

4)

Purpose: The code implements functionality to navigate through pages and display current page information for pagination.

Prompts: write a code that implements the prevPage and nextPage functions to handle pagination. It displays the current page number, total number of pages, and buttons for navigation.

Response:

```
  <p>Page {{ currentPage }} of {{ totalPages }}</p>
  <button (click)="prevPage()" [disabled]="currentPage === 1">Previous</button>
  <button (click)="nextPage()" [disabled]="currentPage === totalPages">Next</button>
</div>


```

5)

Purpose: Create TypeScript methods for component initialization, value retrieval and promotion fetching, page navigation, modal opening and closing with data assignment.

Prompts: write a ts code to create six methods 1st one is ngOnInit, 2nd one is getValue it will retrieve and log the selected value, then fetch promotions based on the value from the API. and 3rd one is onPageChange that will update the currentPage with new page number, 4th is openModal thal will open the modal and assign the passed data to terms and 5th one is openLocationModal it will use to Open the location modal and assign the passed data to locations and 6th and last one is closeModal it will close both modals

Response:

```

@Component({
  selector: 'app-promotion',
  templateUrl: './promotion.component.html',
  styleUrls: ['./promotion.component.css']
})
export class PromotionComponent implements OnInit {
  currentPage: number;
  terms: any;
  locations: any;

  ngOnInit(): void {
    // Initialize component
  }

  getValue(selectedValue: any): void {
    console.log('Selected value:', selectedValue);
    // Fetch promotions based on the selected value from the API
  }

  onPageChange(newPage: number): void {
    this.currentPage = newPage;
    // Update the currentPage with the new page number
  }

  openModal(data: any): void {
    // Open the modal and assign the passed data to terms
    this.terms = data;
  }

  openLocationModal(data: any): void {
    // Open the location modal and assign the passed data to locations
    this.locations = data;
  }

  closeModal(): void {
    // Close both modals
    this.terms = null;
    this.locations = null;
  }
}


```

6)

Purpose: Configure testing module with RouterTestingModule for AppComponent and declare it for testing.

Prompts: Set up the testing module for the AppComponent with RouterTestingModule and declare AppComponent.
Response:

```
describe('AppComponent', () => {
  beforeEach(() => TestBed.configureTestingModule({
    imports: [RouterTestingModule],
    declarations: [AppComponent]
  }));

```

7)

Purpose: Creating AppComponent through Testing.

Prompts: Test that the AppComponent is created successfully.

Response:

```
     it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });
```

8)

Purpose: **Ensure the AppComponent displays the correct title PromotionFinderPage through testing.**

Prompts: Test that the AppComponent has the correct title 'PromotionFinderPage'.

Response:

```
  it(`should have as title 'PromotionFinderPage'`, () => {
  const fixture = TestBed.createComponent(AppComponent);
  const app = fixture.componentInstance;
  expect(app.title).toEqual('PromotionFinderPage');
});
```

9)

Purpose: Create a LocationService class for CRUD operations on LocationDTO objects using MongoDB, with database connection initialized in the constructor.

Prompts: Write a C# class, LocationService, which interacts with a MongoDB database to perform CRUD operations on LocationDTO objects, initializing the database connection in the constructor and providing methods to fetch, add, and retrieve data.

Response:

```
public class LocationService
{
    private readonly IMongoCollection<LocationDTO> _locations;
    public LocationService(IConfiguration configuration)
    {
        var client = new MongoClient(configuration.GetConnectionString("MongoDB"));

        var database = client.GetDatabase("PromotionFinder");
        _locations = database.GetCollection<LocationDTO>("Locations");
    }

    public async Task<List<LocationDTO>> GetByLocationIds(IEnumerable<string> locationIds)
    {

        var locationFilter = Builders<LocationDTO>.Filter.In(x => x.Id, locationIds);
        var locations = await _locations.Find(locationFilter).ToListAsync();
        return locations.ToList();
    }

    public async Task<List<string>> GetAllCountries()
    {
        return await _locations.Distinct(x => x.Country, x => true).ToListAsync();
    }

    public async Task<IEnumerable<LocationDTO>> GetAll()
    {
        return await _locations.Find(x => true).ToListAsync();
    }
    public async Task Add(LocationDTO location)
    {
        if (location.Id == null)
        {
            location.Id = ObjectId.GenerateNewId().ToString();
        }
```

```
            await _locations.InsertOneAsync(location);
        }
    }
```

10)

Purpose: Initialize MongoDB client, connect to "PromotionFinder" database, and retrieve "Locations" collection in LocationService constructor.

Prompts: Create LocationService constructor which initializes a MongoDB client using a connection string from the configuration, and connect to the "PromotionFinder" database, and retrieve the "Locations" collection to interact with.

Response:

```
public LocationService(IConfiguration configuration)
        {
            var client = new MongoClient(configuration.GetConnectionString("MongoDB"));

            var database = client.GetDatabase("PromotionFinder");
            _locations = database.GetCollection<LocationDTO>("Locations");
        }

```

11)

Purpose: Implement GetByLocationIds method to fetch LocationDTO objects from MongoDB using provided location IDs in LocationService.

Prompts: Create `GetByLocationIds` method which fetches a list of `LocationDTO` objects from the MongoDB collection based on a provided list of location IDs.

Response:

```
public async Task<List<LocationDTO>> GetByLocationIds(IEnumerable<string> locationIds)
        {

            var locationFilter = Builders<LocationDTO>.Filter.In(x => x.Id, locationIds);
            var locations = await _locations.Find(locationFilter).ToListAsync();
            return locations.ToList();
        }

```

12)

Purpose: To create method that will retrieve list of country name and store it in mongoDB collection.

Prompts: Create a GetAllCountries method which retrieves a distinct list of country names from the LocationDTO objects and store it in the MongoDB collection.

Response:

```
    public async Task<List<string>> GetAllCountries()
       {
           return await _locations.Distinct(x => x.Country, x => true).ToListAsync();
       }
```

13)

Purpose: To create a method that will insert location objects into collections.

Prompts: Create Add method and insert a LocationDTO object into the MongoDB collection. If the location doesn't have an ID, a new ObjectId will generate and assign before insertion.

Response:

```
public async Task Add(LocationDTO location)
       {
           if (location.Id == null)
           {
               location.Id = ObjectId.GenerateNewId().ToString();
           }
           await _locations.InsertOneAsync(location);
       }

```

14)

Purpose: To create one class that will do all the get, post, put and delete operations.

Prompts: Create PromotionService class which manages promotions stored in a MongoDB database. It offers methods to retrieve all promotions, get a promotion by its ID, filter promotions by country and validity dates, as well as add, update, and delete promotions.

Response: The PromotionService constructor sets up a connection to a MongoDB database using the provided configuration, specifically targeting the "PromotionFinder" database and the "Promotions" collection for subsequent operations.

```
public class PromotionService
  {
      private readonly IMongoCollection<PromotionDTO> _promotions;
      public PromotionService(IConfiguration configuration)
      {
          var client = new MongoClient(configuration.GetConnectionString("MongoDB"));

          var database = client.GetDatabase("PromotionFinder");
          _promotions = database.GetCollection<PromotionDTO>("Promotions");
      }

      public async Task<IEnumerable<PromotionDTO>> GetAll()
      {
          return await _promotions.Find(x => true).ToListAsync();
      }

      public async Task<PromotionDTO> Get(string id)
      {
          return await _promotions.Find(p => p.Id == id).FirstOrDefaultAsync();
      }

      public async Task<IEnumerable<PromotionDTO>> GetByCountry(string country)
      {

          var dateNow = DateTime.Now;
          var promotionFilter = Builders<PromotionDTO>.Filter.And(
              Builders<PromotionDTO>.Filter.Eq(x => x.Country, country),
              Builders<PromotionDTO>.Filter.Lte(x => x.ValidityStart, dateNow),
              Builders<PromotionDTO>.Filter.Gt(x => x.ValidityEnd, dateNow)
          );
          return await _promotions.Find(promotionFilter).ToListAsync();
      }

      public async Task Add(PromotionDTO promotion)
      {
          if (promotion.Id == null)
          {
              promotion.Id = ObjectId.GenerateNewId().ToString();
          }
          await _promotions.InsertOneAsync(promotion);
      }

      public async Task Update(PromotionDTO promotion)
      {

          var existingPromotion = await _promotions.Find(p => p.Id == promotion.Id).FirstOrDefaultAsync();
```

```
            if (existingPromotion != null)
            {
                await _promotions.ReplaceOneAsync(p => p.Id == promotion.Id, promotion);
            }
        }

        public async Task Delete(string id)
        {

            await _promotions.DeleteOneAsync(p => p.Id == id);
        }
    }
```

15)

Purpose: To create a method that will retrieve all the data that are present in “Promotions” collection.

Prompts: Create a GetAll method with retrieves all PromotionDTO objects from the MongoDB "Promotions" collection and returns them as an enumerable list.

Response:

```
public async Task<IEnumerable<PromotionDTO>> GetAll()
        {
            return await _promotions.Find(x => true).ToListAsync();
        }

```

16)

Purpose: To create a method that will fetch the details by passing the country details.

Prompts: Create a GetByCountry method and retrieve PromotionDTO objects from the MongoDB "Promotions" collection that match the specified country and are currently valid based on the current date.

Response:

```
public async Task<IEnumerable<PromotionDTO>> GetByCountry(string country)
        {

            var dateNow = DateTime.Now;
            var promotionFilter = Builders<PromotionDTO>.Filter.And(
                Builders<PromotionDTO>.Filter.Eq(x => x.Country, country),
                Builders<PromotionDTO>.Filter.Lte(x => x.ValidityStart, dateNow),
                Builders<PromotionDTO>.Filter.Gt(x => x.ValidityEnd, dateNow)
            );
            return await _promotions.Find(promotionFilter).ToListAsync();
        }

```

17)

Purpose: To create one method that will add details in database.

Prompts: Create an Add method which adds a PromotionDTO object to the MongoDB "Promotions" collection. If the promotion doesn't already have an ID, it generates a new unique ID before inserting the promotion.

Response:

```
public async Task Add(PromotionDTO promotion)
        {
            if (promotion.Id == null)
            {
                promotion.Id = ObjectId.GenerateNewId().ToString();
            }
            await _promotions.InsertOneAsync(promotion);
        }

```

18)

Purpose: To create a method that will update details in database.

Prompts: Create an Update method to modify a PromotionDTO object in the MongoDB "Promotions" collection using its ID. If a matching promotion is found, it will update the record with the new promotion details.

Response:

```
public async Task Update(PromotionDTO promotion)
        {

            var existingPromotion = await _promotions.Find(p => p.Id == promotion.Id).FirstOrDefaultAsync();
            if (existingPromotion != null)
            {
                await _promotions.ReplaceOneAsync(p => p.Id == promotion.Id, promotion);
            }
        }
```

19)

Purpose: To create method for connecting the database, storing in database and to retrieve data from database.

Prompts: Create a TermsService class which manages terms stored in a MongoDB database. It initializes the database connection in the constructor and provides methods to add new terms, retrieve all terms, and retrieve terms by their IDs.

Response:

```
public class TermsService
  {
      private readonly IMongoCollection<TermDTO> _terms;
      public TermsService(IConfiguration configuration)
      {
          var client = new MongoClient(configuration.GetConnectionString("MongoDB"));

          var database = client.GetDatabase("PromotionFinder");
          _terms = database.GetCollection<TermDTO>("Terms");
      }

      //Add a new term
      public async Task Add(TermDTO term)
      {
          if (term.Id == null)
          {
              term.Id = ObjectId.GenerateNewId().ToString();
          }
          await _terms.InsertOneAsync(term);
      }
      //Get all terms
      public async Task<IEnumerable<TermDTO>> GetAll()
      {
          return await _terms.Find(x => true).ToListAsync();
      }
      public async Task<IEnumerable<TermDTO>> GetAllByIds(IEnumerable<string> termIds)
      {
          var termsFilter = Builders<TermDTO>.Filter.In(x => x.Id, termIds);
          var terms = await _terms.Find(termsFilter).ToListAsync();
          return terms;
      }

  }

```

20)

Purpose: To create a method that will connect to mongoDB database.

Prompts: Create a TermsService constructor to connect to a MongoDB database and initializes the _terms collection for TermDTO objects.

Response:

```
public TermsService(IConfiguration configuration)
      {
          var client = new MongoClient(configuration.GetConnectionString("MongoDB"));
```

```

            var database = client.GetDatabase("PromotionFinder");
            _terms = database.GetCollection<TermDTO>("Terms");
        }

```

21)

Purpose: To create one method that will store data in mongoDB collection.

Prompts: Create an Add method to insert a TermDTO object into the MongoDB _terms collection, generating a new ID if none is provided.

Response:

```
public async Task Add(TermDTO term)
        {
            if (term.Id == null)
            {
                term.Id = ObjectId.GenerateNewId().ToString();
            }
            await _terms.InsertOneAsync(term);
        }

```

22)

Purpose: To create method to get the data by ids.

Prompts: Create GetAllByIds method to fetche TermDTO objects from the MongoDB _terms collection based on provided IDs and returns them as an enumerable list.

Response:

```
public async Task<IEnumerable<TermDTO>> GetAllByIds(IEnumerable<string> termIds)
        {
            var termsFilter = Builders<TermDTO>.Filter.In(x => x.Id, termIds);
            var terms = await _terms.Find(termsFilter).ToListAsync();
            return terms;
        }

```

c) Chat based Copilot:

1)
Purpose: To generate unit test cases

Prompts: @workspace /tests generate unit test case for selected code
Response:

```
  let component: PaginatorComponent;
  let fixture: ComponentFixture<PaginatorComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [PaginatorComponent]
    });
    fixture = TestBed.createComponent(PaginatorComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should calculate total pages on changes', () => {
    component.totalItems = 100;
    component.pageSize = 10;
    component.ngOnChanges();
    expect(component.totalPages).toEqual(10);
  });

  it('should increment current page by 1 when next page is called', () => {
    component.currentPage = 1;
    component.totalPages = 2;
    component.nextPage();
    expect(component.currentPage).toEqual(2);
  });

  it('should not increment current page when next page is called and current page is last page', () => {
    component.currentPage = 2;
    component.totalPages = 2;
    component.nextPage();
    expect(component.currentPage).toEqual(2);
  });

  it('should decrement current page by 1 when prev page is called', () => {
    component.currentPage = 2;
    component.prevPage();
    expect(component.currentPage).toEqual(1);
  });

  it('should not decrement current page when prev page is called and current page is first page', () => {
    component.currentPage = 1;
    component.prevPage();
    expect(component.currentPage).toEqual(1);
  });
});
```

2)

Purpose: To generate pagination html code

Prompts: Generate HTML code for a pagination component that displays the current page number, the total number of pages, and buttons for navigating to the previous and next pages.

Response:

```
    <button class="prev-btn" onclick="prevPage()">Previous</button>
    <span id="page-num">1</span> / <span id="total-pages">10</span>
    <button class="next-btn" onclick="nextPage()">Next</button>
</div>

```

3)

Purpose: To generate functioning of pagination

Prompts: (Provide the code snippet generated by above prompt) in the above code The previous button should be disabled when the current page is the first page, and the next button should be disabled when the current page is the last page.

Response:

```
    <button class="prev-btn" id="prev-btn" onclick="prevPage()">Previous</button>
    <span id="page-num">1</span> / <span id="total-pages">10</span>
    <button class="next-btn" id="next-btn" onclick="nextPage()">Next</button>
</div>

<script>
    var currentPage = 1;
    var totalPages = 10;

    document.getElementById('page-num').innerText = currentPage;
    document.getElementById('total-pages').innerText = totalPages;

    function updateButtons() {
        document.getElementById('prev-btn').disabled = currentPage === 1;
        document.getElementById('next-btn').disabled = currentPage === totalPages;
    }

    function prevPage() {
        if (currentPage > 1) {
            currentPage--;
            document.getElementById('page-num').innerText = currentPage;
            updateButtons();
        }
    }

    function nextPage() {
        if (currentPage < totalPages) {
            currentPage++;
            document.getElementById('page-num').innerText = currentPage;
            updateButtons();
        }
    }

    updateButtons();
</script>

```

4)

Purpose: To use multi cursor in VS code

Prompts: @vscode how do I use multi cursor

Response:
In Visual Studio Code, you can use multi-cursor for various tasks such as editing multiple lines simultaneously, renaming variables, and more. Here's how you can use it:

1. Add additional cursors to all occurrences of the current selection: Use the Add Cursors To Top (editor.action.addCursorsToTop) or Add Cursors To Bottom (editor.action.addCursorsToBottom) commands.

2. Add a new cursor at a specific location: Hold the Alt key and click at the location where you want to add a new cursor.

3. Add a new cursor to all occurrences of a specific word: Select the word and use the Add Cursors To Top or Add Cursors To Bottom commands.

5)

Purpose: To center align pagination

Prompts: @workspace /fix my page is not center aligned.

Response:
To center align your pagination component, you can use CSS Flexbox. Add the following CSS properties to your .pagination class:

```
pagination {
    display: flex;
   justify-content: center;
align-items: center;
 }

```

6)

Purpose: To understand the working of selected code

Prompts: @workspace /explain please explain the selected code

Response:
The selected code is a method in C# that fetches promotions for a specific country from a MongoDB database. The method is part of a service class, likely used in a web API or similar application. Here's a breakdown:

```
public async Task<IEnumerable<PromotionDTO>> GetByCountry(string country)
```

This line defines the method signature. The method is public, meaning it can be accessed from outside the class. It's async, which means it's asynchronous and returns a Task. The Task wraps a collection (IEnumerable) of PromotionDTO objects. The method takes one parameter, a string representing the country.

```
var dateNow = DateTime.Now;
```

This line gets the current date and time.

```
);
var promotionFilter = Builders<PromotionDTO>.Filter.And(
    Builders<PromotionDTO>.Filter.Eq(x => x.Country, country),
    Builders<PromotionDTO>.Filter.Lte(x => x.ValidityStart, dateNow),
    Builders<PromotionDTO>.Filter.Gt(x => x.ValidityEnd, dateNow)
```

```
);
```

This block creates a filter for the MongoDB query. It uses the Builders class from the MongoDB driver to create a compound filter with three conditions:

- The Country property of the PromotionDTO must equal the country parameter.
- The ValidityStart property must be less than or equal to the current date and time.
- The ValidityEnd property must be greater than the current date and time.

This filter will match promotions that are valid for the specified country and whose validity period includes the current date and time.

```
return await _promotions.Find(promotionFilter).ToListAsync();
```

7)

Purpose: To generate a service method

Prompts: @workspace generate a service method to fetch all the promotion by country

Response:
Based on the structure of your workspace and the existing code in PromotionService.cs, you can add a method to fetch all promotions by country. Here's an example of how you can do it:

```
}
public async Task<IEnumerable<PromotionDTO>> GetAllByCountry(string country)
{
    var promotionsByCountryFilter = Builders<PromotionDTO>.Filter.Eq(x => x.Country, country);
    return await _promotions.Find(promotionsByCountryFilter).ToListAsync();
}

```

This method, GetAllByCountry, is asynchronous and returns a Task that wraps a collection (IEnumerable) of PromotionDTO objects. It takes one parameter, a string representing the country.

The method creates a filter that matches PromotionDTO objects where the Country property equals the country parameter. It then uses this filter to find matching documents in the _promotions collection, converts the result to a list, and returns it.

Please add this method to your PromotionService.cs file.

8)

Purpose: To install angular and run project

Prompts: @terminal /explain Commands need to be executed in terminal to install and run angular code.

Response:
To install Angular CLI, you can use the following command in your terminal:

```
npm install -g @angular/cli
```

This command installs Angular CLI globally on your system. The -g flag is used to specify that the package should be installed globally.

After the installation is complete, you should be able to use the ng command. You can verify the installation by checking the version of Angular CLI:

```
ng --version
```

Once Angular CLI is installed, you can serve your Angular application using the ng serve command:

```
ng serve
```

This command starts the development server. By default, the server runs on http://localhost:4200/.

Please ensure that you're in the correct directory (the root directory of your Angular project) before running the ng serve command.

9)

Purpose: To understand the use of ngOnInit()

Prompts: @workspace /explain how this code works and what is need of using ngOnInit()

Response:
This code defines an Angular component named PromotionComponent. The PromotionComponent class is decorated with @Component, which tells Angular that this class is an Angular component.

The PromotionComponent class has several properties and methods:

promotions: An array of Promotion objects.

valuevalue: A variable to store the value of a selected option from a dropdown.

showModal and showModal2: Booleans to control the visibility of modals.

itemsPerPage and currentPage: Variables for pagination.

countries, terms, and locations: Arrays to store data fetched from APIs.

constructor: The constructor method is where the HttpClient is injected into this component, and where the component fetches the list of countries from an API.

ngOnInit: This is a lifecycle hook that is called after Angular has initialized all data-bound properties of a directive. In this code, it's empty and can be removed if not used.

pagedEnergyData: A getter method that returns a slice of the promotions array based on the current page and items per page.

getValue: A method that fetches promotions based on the selected value from a dropdown.

onPageChange: A method that updates the current page when the user navigates to a different page.

openModal, openLocationModal, and closeModal: Methods to open and close modals.

The ngOnInit() method is a lifecycle hook in Angular, and it's a good place to put initialization logic. This method is called just after the constructor call and after the first ngOnChanges() call (which is another lifecycle hook that is called when an input or output binding value changes). If you need to fetch data when the component initializes, ngOnInit() is a good place to do it. In this code, ngOnInit() is not used and can be removed.

10)

Purpose: To overview the project files

Prompts: @workspace list the file names which are in my workspace.

Response:
Here are some of the file names in your workspace:

```
1. app.component.ts
 2. app.module.ts
 3. Promotion.ts
 4. module-routing.module.ts
 5. module.module.ts
 6. .gitkeep
 7. custom-theme.scss
 8. environment.ts
 9. index.html
10. main.ts
11. styles.css
12. tsconfig.app.json
13. tsconfig.json
14. tsconfig.spec.json
```

```

```

11)

Purpose: To create a jupyter notebook

Prompts: @workspace /newNotebook create a notebook which should contain all angular commands

Response:
Here's an outline for a Jupyter notebook that contains all Angular commands:

Setting Up the Angular Environment

Creating a New Angular Application

Generating Angular Components

Generating Angular Services

Generating Angular Modules

Building and Serving Angular Applications

Running Angular Unit Tests

Create Notebook

d) New Feature Enhancement(UI & Backend):

DB:

Purpose:

The purpose of the prompt is to design a MongoDB schema for storing states, including a foreign key reference to countries, state IDs, and state names.

Prompts: create State table with Country_id as a foreign key, State_Id, State_Name in MongoDB.

Response:
json

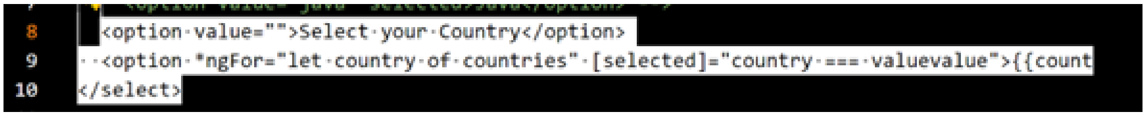
```
{

 "_id": ObjectId("..."),

 "State_Id": 1,

 "State_Name": "California",

 "Country_id": ObjectId("...") // Foreign key referencing the Country document

}
```

- **_id** is the unique identifier for each document in the State collection. MongoDB automatically assigns a unique **_id** to each document if you don't provide one explicitly.
- **State_Id** is an identifier for the state.
- **State_Name** is the name of the state.
- **Country_id** is a reference to the Country document where this state belongs. It's typically stored as the ObjectId of the related country document.

To create this structure in MongoDB, you would insert documents into a State collection. Before creating the State collection, make sure you have a Country collection where each document represents a country. The **Country_id** in the State documents would reference the **_id** field of the corresponding country document
UI:

Purpose:

The purpose is to enhance the existing schema by enabling filtering of states based on both country and product type, thereby improving query flexibility and data retrieval capabilities.
Prompts: Add one more filter for showing State based on Country and Product type mentioned below:
Response:



```
  <option value="">Select your Country</option>
  <option *ngFor="let country of countries" [selected]="country === valuevalue">{{count
</select>
```

API:
Purpose: The purpose is to develop a .NET 7 API endpoint called "getState" that retrieves states based on a specified country, facilitating state information retrieval for a given country in a .NET environment.
Prompts: create a getState api based on country in dotnet 7

Response:

```
// Country model
public class Country
{
    public int Id { get; set; }
    public string Name { get; set; }
}

// State model
public class State
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int CountryId { get; set; }
}

// Fake repository for demonstration purposes
public interface IStateRepository
{
    IEnumerable<State> GetStatesByCountry(int countryId);
}

public class StateRepository : IStateRepository
{
    private readonly List<State> _states = new List<State>
    {
        new State { Id = 1, Name = "California", CountryId = 1 },
        new State { Id = 2, Name = "Texas", CountryId = 1 },
        // Add more states...
    };

    public IEnumerable<State> GetStatesByCountry(int countryId)
    {
        return _states.Where(s => s.CountryId == countryId);
    }
}



[ApiController]
[Route("api/[controller]")]
public class StateController : ControllerBase
{
    private readonly IStateRepository _stateRepository;

    public StateController(IStateRepository stateRepository)
    {
        _stateRepository = stateRepository;
    }

   [HttpGet("getState/{countryId}")]
    public IActionResult GetStatesByCountry(int countryId)
    {
        var states = _stateRepository.GetStatesByCountry(countryId);
        if (states == null)
        {
            return NotFound();
        }
        return Ok(states);
```