- Smart Living -
Shopping & Cost Planner APP

.NET APPLICATION - MADE FOR SOFTBINATOR LABS 2023 -

ION APELIA

Why would you need an app to plan your shopping?

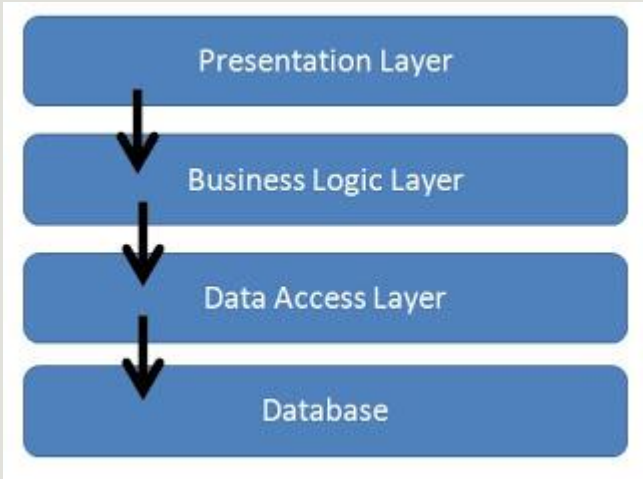
- Helps you stay on budget: When you plan your shopping, you can create a budget for yourself and stick to it. This will help you avoid overspending and ensure that you have enough money to cover all of your necessary expenses.

- Saves time: Planning your shopping in advance can help you save time by allowing you to create a list of items you need to buy, and prioritize your shopping according to your needs.

- Reduces stress: When you have a plan for your shopping, you are less likely to feel overwhelmed and stressed about the process. You will know exactly what you need to buy, where to buy it, and how much you need to spend.

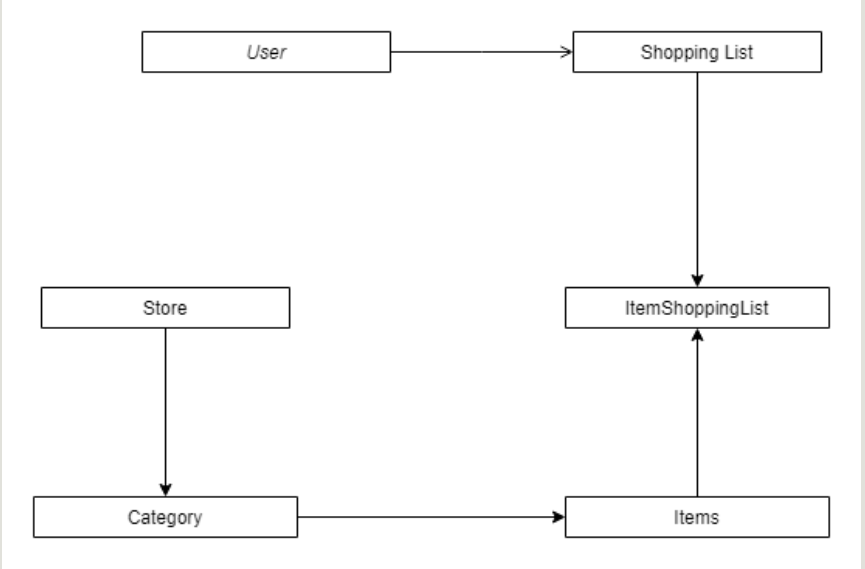
App architecture:

- Three-tier architecture: The app is designed using a three-tier, which includes a presentation layer (API), a business logic layer (Application), a core layer (Domain) and a data access layer (Infrastructure).
- The app architecture is designed to be scalable, meaning that it can handle a large number of users and data without compromising performance or stability. architecture allows for easy maintenance and upgrades, as each layer can be modified independently of the others.
- Separation of concerns: The app architecture follows the principle of separation of concerns, meaning that each layer has a distinct responsibility and does not interfere with the others.



Entities

- User: This entity represents a user of the app. Users can create accounts, save shopping lists.
- Shopping List: This entity represents a list of items that a user wants to purchase. Each shopping list would be associated with a specific user.
- Item: This entity represents a single item that a user wants to purchase. Each item would be associated with a shopping list and would have a name, quantity, price.
- Category: This entity represents a category of items, such as groceries, clothing, or electronics. Each category would be associated with one or more items.
- Store: This entity represents a store where a user can purchase items. Each store would have a id, name, and a list of categories.



Functionality

The application allows users to create an account and log in to manage their shopping lists.

Once logged in, users can view their shopping lists, add new items to an existing list, and create new shopping lists.

The application also provides the ability to edit and delete existing shopping lists and items.

The user can add multiple items to a shopping list, specify the quantity, and set the unit price of each item.

The application calculates the total cost of each shopping list based on the quantity and unit price of each item.

Overall, the application helps users plan their shopping and keep track of their expenses.

Security: Authentication and Authorisation

By implementing JWT authentication in the shopping and cost planner app, we are able to provide a secure and scalable way for users to authenticate and access their data, while also ensuring that user data is protected from unauthorized access and malicious attacks.

By implementing authentication in our shopping and cost planner app, we can provide granular access control to our users, ensure that sensitive data is protected, and comply with relevant security and privacy regulations.

Exemple:
Here, only an authenticated person can use the endpoint to send an email

```
[AllowAnonymous]
[HttpGet("Get all users")]
0 references
public async Task<ActionResult<IEnumerable<User>>> GetAllUsers()
{
    var users = await _userService.GetAllUsers();
    _logger.LogInformation("Get All Users performed");
    return Ok(users);
}

[Authorize]
[HttpPost("Send-email")]
0 references
public async Task<IActionResult> SendEmail(EmailSendModel email)
{
    await _emailService.SendAsync(email);

    return Ok();
}
```

Good Engineering Practices Implemented

Dependency Injection: Dependency injection allows for loose coupling between components and promotes modularity, testability, and maintainability in code.

```
// Add services
//builder.Services.AddTransient<ITokenService, TokenService>();
//builder.Services.AddTransient<IUserService, UserService>();
builder.Services.AddServices();
builder.Services.AddTransient<IUserRepository, UserRepository>();
```

Loggers

```
public async Task<User> GetUserById(int id)
{
    _logger.LogInformation("Getting user by ID: {Id}", id);
    return await _userRepository.GetUserById(id);
}
```

BaseClasses

```
5 references
public class BaseEntity : IBaseEntity
{
    [Key]
    15 references
    public int Id { get; set; }

    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public DateTime CreatedAt { get; set; }

    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    0 references
    public DateTime LastUpdated { get; set; }

}
```

END

Questions?