Examinerande inlämningsuppgift 1

Testar kunskaper och färdigheter inom:

- Enhetstester
- Testbar design
- Underhållsvänlig, lättläst, kommenterad och skalbar kod
- Feltolerans
- Design patterns

[1-4] = Övergripande krav

[1-4] = Absoluta krav

Uppgiften går ut på att utifrån befintligt ASP .NET Core MVC-projekt implementera design patterns, skriva enhetstester, anpassa designen efter testning och utföra förbättringar kopplade till underhållsvänlighet och testbarhet.

Projektet är det fiktiva företaget Ylvas Kaffelagers webbshop.

Uppdrag:
Ylva vill skala upp verksamheten. Utöver kaffe vill hon kunna sälja kaffebryggare och andra tillbehör. En konsult har tidigare sagt att det vore bra med enhetstester för att säkerställa att beräkningarna utförs korrekt. Ibland uppstår det dessutom tomma värden i databasen vilket komplicerar affärsverksamheten. En annan konsult meddelade Ylva nyligen att projektet var ostrukturerat och i behov av refaktorering.

På önskelistan finns följande:

- Refaktorering med design patterns. Dependency injection för databaskoppling, decorator pattern för den klass som utför beräkningar och ”loosely coupled” MVC d.v.s controllers ska inte utföra beräkningar. Detta ska göras i en annan klass (alla beräkningar ska utföras på ett och samma ställe). Kommentera! Kommentar: Den dekorerande klassen ska innehålla minst en förändring. [1]
- Enhetstester för beräkning av ordersumma. Värdet ”Total” ska inte under några som helst omständigheter visa fel värde. Lägg till ett xUnit-projekt och skriv minst ett test (Should_Return_Correct_Sum). [2]
- Enhetstester för hämtning av data. Vad händer om en produkt inte existerar i databasen? Skapa ett mock object med hjälp av Moq och skriv minst ett test som säkerställer att eventuellt null exception hanteras (t. ex. om null skickas till klass som utför beräkningar). Om det blir för komplicerat, kontrollera så att ett Coffee-objekt existerar om Id matchar mot något av objektens Id i mock-objektet. Kommentar: Syftet med denna del är att använda Moq och **skapa test mot mock object**. Minimum: Skriv ett test som säkerställer att ett Coffee-objekt returneras om Id matchar något av objektens Id i mock-objektet (Should_Return_Coffee) [3]
- Fler interfaces. DbContext ska definitivt implementera ett interface. [4]
- Felhantering. Vad händer om något värde saknas vid orderläggning? Hantera felen med hjälp av try-catch, validering eller liknande. [1]

- Validering av e-post. Orderläggning ska avbrytas om e-postfältet har lämnats tomt. Användaren ska meddelas om detta så att det går att försöka igen. [2]
- Lättläst kod! Vissa metoders namn går att göra ännu tydligare. Kommentar: Se MyPage [3]
- Order-klassen behöver kommenteras eftersom det ofta är den som behöver uppdateras. [4]

Deadline: Fredag 18 november 2022

Inlämning:

- Komprimerad mapp (.zip-fil) via inlämningsmapp på Moodle.

Alternativt...

- Textfil med länk till Git-repository via inlämningsmapp på Moodle.

Betyg:

Icke godkänt (IG) Den studerande har genomfört färre än två övergripande krav och färre än fyra absoluta krav.

Godkänt (G) Den studerande har självständigt genomfört minst två övergripande krav och samtliga absoluta krav.