Sprint 3 – Business Layer, AutoMapper, Validation

1. Bauen Sie leere **Businesslayer-Komponenten** für die definierten User Stories aus dem Swagger Contact.

   Entities, Implementierung & Interfaces sollen dabei wieder **in separaten Projekten** liegen (parallel zum Service Layer).

   - *Abc.Qwe*.BusinessLogic.Entities
   - *Abc.Qwe*.BusinessLogic.Interfaces
   - *Abc*.*Qwe*.BusinessLogic
   - *Abc*.*Qwe*.BusinessLogic.Tests

   Wichtig: Ihnen liegt frei, welche Granularität Sie wählen, d.h. wieviele Logik Klassen Sie bauen. Z.B. eine Logik pro User Story, oder pro Aggregate.

   z.B.

   ```
   public class TrackingLogic : ITrackingLogic
   {
       public TrackingLogic() { … }

       public void TrackPackage(string trackindId, …) { … }
   }
   ```

   Die Komponenten sollen lose an das **REST-Service** gekoppelt sein und von dort aus aufgerufen werden.
   D.h. erstellen Sie für jede Komponente ein **Interface** und eine **Implementierung**
   (zB. ITrackLogic + TrackLogic).

   Benutzen Sie für Parameter, Variablen, etc. **immer** die **Interfaces**! Lediglich für die Erstellung (Constructor) brauchen Sie die eigentliche Klasse.

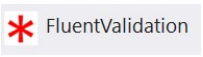
2. Erstellen Sie seperate Entitäten für den Business Layer (Namespace / Projekt *Abc*.*Qwe*.BusinessLogic.Entities). Also wiederum Warehouse, Parcel, ...

   Wichtig: Diese Entities sollen auf den Business Layer optimiert sein und nicht einfach nur eine 1:1 Kopie der DTOs aus dem REST Service sein. Sie haben jedoch die Freiheit, diese wo nötig zu verändern.

   Zum Beispiel sollen die DTOs ***NewParcelInfo, TrackingInformation und Parcel*** vom Services Layer, am Business Layer als **eine Entity *Parcel*** abgebildet werden!

3. **Rufen Sie die Business Layer Komponenten** aus den entsprechenden REST SERVICE CONTROLLERN auf. **Mappen** Sie vorher die REST-Service DTOs auf die neuen Business Entities mit Hilfe des **AutoMapper Frameworks (von Nuget)** und nachher wieder retour. Sodass der Service Layer die Übersetzung macht.

4. **Validieren Sie in den BL Komponenten die Entities** – unter Verwendung des Frameworks **FluentValidation** (von Nuget).

   a. *PostalCode* = z.B. „A-1120“ ("A-", 4 Zahlen, 0000-9999)
      (wenn Country “Austria” oder “Österreich“ ist, sonst keine Validierung)
   b. *Street* = z.B. „Hauptstraße 12/12/12“ oder „Landstraße 27a“
      (Straße, Leerzeichen, Hausnummer (Zahlen/Text/Slashes))
      (wenn Country “Austria” oder “Österreich“ ist, sonst keine Validierung)
   c. *City, Name* = Nur Groß-/Kleinbuchstaben, "-" Zeichen, Leerzeichen, beginnend mit Großbuchstaben
      (wenn Country “Austria” oder “Österreich“ ist, sonst keine Validierung)
   d. *HopArrival – Code* = entsprechend dem Pattern in Swagger
   e. *Warehouse– Description*= Nur Groß-/Kleinbuchstaben, "-" Zeichen, Leerzeichen, Zahlen (z.B. Hauptlager 27-12)
   f. *Parcel TrackingCode* = entsprechend dem Pattern in Swagger
   g. *Parcel Weight* = > 0.0
   h. Alle Referenzen (wie Recipient, ..) und Listen (wie List<HopArrival>) sollen NotNull sein.

5. Passen Sie die **existierenden Unit Tests des Service Layers an**, damit diese mit MOCK Objekten den Business Layer mocken. Dazu muss auch der Service Layer / Controller Code angepasst werden, um Constructor Injection zu erlauben.
   Verwenden Sie ab jetzt dazu MOQ für Mock Objekte in Unit Tests.
   Testdaten erstellen Sie über NBuilder.

6. Schreiben Sie **UnitTests**, die Ihre Business Komponenten testen. Halten Sie die **Code Coverage 70%** weiterhin ein und stellen Sie sicher, dass diese **im Azure DevOps BUILD aufscheint**.
   (also auch die Unit Tests im BUILD ausgeführt werden)

7. Abgabe:
   Der MAIN Branch ihres Azure DevOps Git Repositories sollte immer den Abgabestand des Source Codes enthalten. Arbeiten Sie auf einem anderen Branch und mergen Sie auf MAIN, sobald Sie abgeben / releasen.
   Nicht vergessen: Abgabe auf Moodle!