Lab01 –Project Setup

Hinweis:

Das Aufgabenblatt enthält neben den Aufgaben zum Modulpraktikum auch Zusatzfragen zum Verständnis und zur Repetition des Stoffes. Diese Fragen müssen nicht für die Fallstudie beantwortet werden. Sie können jedoch zur Prüfungsvorbereitung verwendet werden.

Aufgabe 1. Routeplanner Konsolenapplikation

a) Erstellen Sie mit Visual Studio ein neues Projekt vom Typ Console App (.NET Core) mit dem Namen `RoutePlannerConsole` und benennen Sie die Solution (Projektmappe) `RoutePlanner`.

b) Stellen Sie Ihre Solution direkt nach der Erzeugung unter Versionskontrolle in einem lokalen Git-Repository, in dem Sie die Menu-Option "File – Add to source control" ausführen.

c) Ändern Sie in den Projekt-Properties den Default-Namespace des Programms auf `Fhnw.Ecnf.RoutePlanner.RoutePlannerConsole`

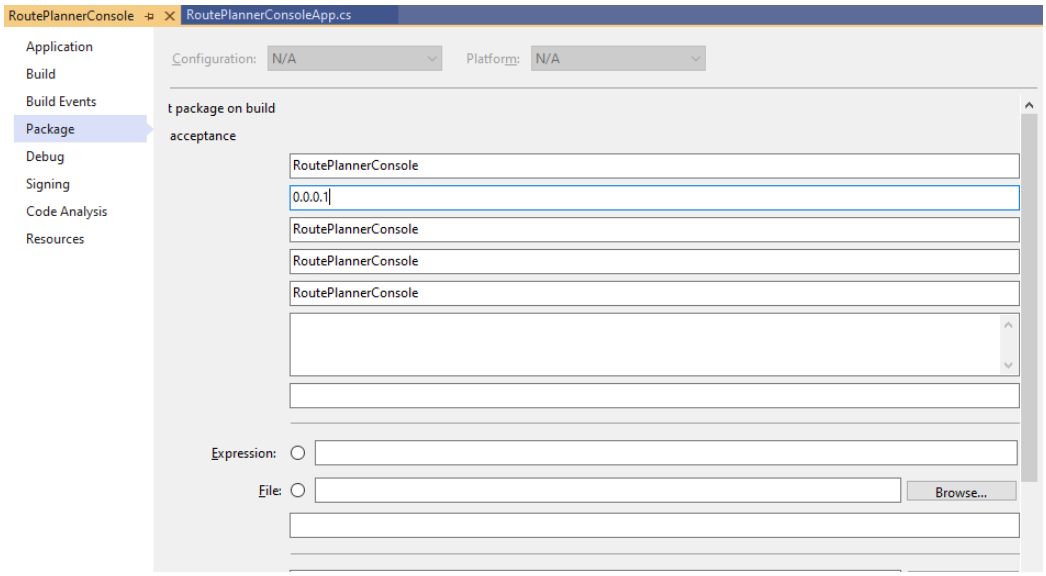
   Stellen Sie sicher, dass Ihr Projekt als «Target framework».NET Core 3.1 verwendet.

d) Legen Sie im Projekt die Klasse `RoutePlannerConsoleApp` an (oder benennen Sie die bestehende Applikationsklasse um). Benennen Sie auch das File entsprechend um. Tipp: Verwenden Sie die eingebaute Refactoring Funktion für das Renaming. Stellen Sie sicher, dass Sie auch den Namespace der umbenannten Klasse nachführen.

e) Ergänzen Sie Ihre Main-Methode so, dass Sie folgende Ausgabe auf die Konsole ausgibt:

   ```
   Welcome to RoutePlanner (Version 0.0.1.0)
   ```

   Dabei soll die Versionsnummer aus der bei den Projekt-Properties unter Package gesetzten Versionnummer gelesen werden und muss dort entsprechend angepasst werden. (Wie das geht haben Sie im Unterricht am Beispiel des PersonAdminSystem kennengelernt)



Kompilieren Sie das Programm und lassen Sie es zu Testzwecken laufen.

Aufgabe 2. Routeplanner Library

a) Fügen Sie Ihrer Solution ein neues Projekt vom Typ *Class Library (.NET Core)* mit Namen `RoutePlannerLib` hinzu. In dieser Library werden Sie sämtliche weitere Applikationslogik implementieren. Verwenden Sie die Console-Applikation nur noch zu Testzwecken.

Löschen Sie die automatisch generierte Klasse `Class1` aus dem Projekt.

Definieren Sie als Default-Namespace für das Library-Projekt

`Fhnw.Ecnf.RoutePlanner.RoutePlannerLib`

b) Fügen Sie die neue Assembly Ihrer bestehenden Konsolen-Applikation unter *Dependencies* hinzu.

c) Fügen Sie dem neuen Projekt die public Klasse `WayPoint` hinzu; fügen Sie der Klasse drei Properties `Name`, `Latitude`, `Longitude` hinzu. Dabei sind die Properties `Latitude` und `Longitude` vom Typ `double` und das Feld `Name` vom Typ `string`.

Hinweis: Die Felder können Sie als sogenannte automatischen Properties implementieren (siehe Code Snippet unten); dazu nächste Woche mehr.

```
public class WayPoint
{
    public string Name { get; set; }
    public double Longitude { get; set; }
    public double Latitude { get; set; }

    public WayPoint(string name, double latitude, double longitude)
    {
        Name = name;
        Latitude = latitude;
        Longitude = longitude;
    }
}
```

Die Klasse `WayPoint` soll im Namespace `Fhnw.Ecnf.RoutePlanner.RoutePlannerLib` abgelegt werden.

d) Legen Sie zu Testzwecken in der Main()-Methode der Konsolen-Applikation eine WayPoint-Instanz an und geben Sie deren Inhalt auf die Konsole aus. Sie können auf die WayPoint-Properties wie auf Felder zugreifen.

Sie müssen hierzu die Library wieder dem Main-Projekt als Referenz hinzuweisen.

```
var wayPoint = new WayPoint("Windisch", 47.479319847061966, 8.212966918945312);
Console.WriteLine($"{wayPoint.Name}: {wayPoint.Latitude}/{wayPoint.Longitude}");
```

Kompilieren Sie das Programm und lassen Sie es zu Testzwecken laufen.

Aufgabe 3. Verbinden des Projektes mit dem Remote Repository

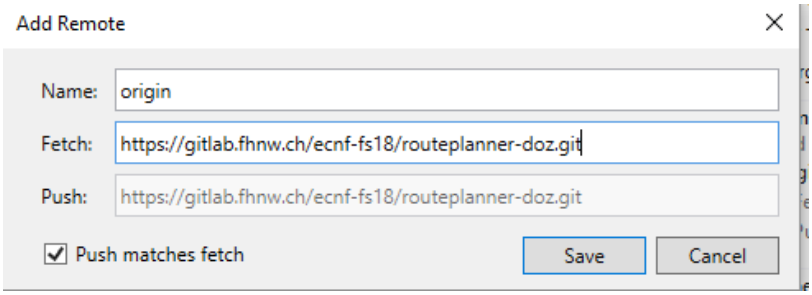
Sobald Sie sich als Team eingeschrieben und den Dozierenden benachrichtigt haben (**beachte MS0**), wird Ihnen ein remote Git Repository auf gitlab der FHNW zugewiesen.

Die folgenden Aufgaben darf nur **ein Teammitglied** ausführen:

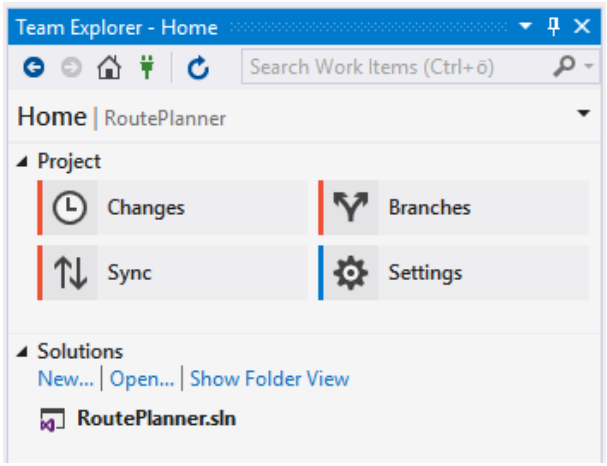
a) Verbinden Sie Ihr lokales Git Repo mit dem Ihnen zugewiesenen Remote Repo auf dem Gitlab Server. Sie machen das mit der Team Explorer View – Home – Settings.

Wählen Sie «Repository Settings» aus und unter «Remotes» die «Add…» Funktion.

Geben Sie im angezeigten Dialog unter Namen «origin» ein und unter «Fetch» die URL **Ihres** remote Repositories.



b) Pushen Sie nun ihr lokales Repo auf das Remote Repository mittels «Team Explorer – Home – Sync»



Führen Sie den Push Befehl aus; geben Sie einen sinnvollen Kommentar.

Siehe auch unter https://docs.microsoft.com/en-us/vsts/git/tutorial/creatingrepo?tabs=visual-studio#connect-a-local-repo-to-a-remote

Die folgende Aufgabe muss von den **anderen Teammitgliedern** durchgeführt werden, um sich den Code vom zentralen Repository zu holen und sich selbst mit diesem zu verbinden.

c) Sie müssen sich nun ihrerseits mit dem remote Repository verbinden.

Folgen Sie dazu den Anweisungen wie unter https://docs.microsoft.com/en-us/vsts/git/tutorial/clone?tabs=visual-studio#clone-from-another-git-provider beschrieben.

Nachdem Sie die Solution geöffnet haben, sollten Sie den Code ihrer Kollegin in ihrer Working Copy haben.

Testfragen

1. Was versteht man unter „Managed Code“, was unter „Native Code“? Nennen Sie jeweils einige Vor- und Nachteile der beiden Ansätze.
2. Was versteht man unter der CLR? Was sind deren Aufgaben?
3. Was ist ein Assembly?

4. Ein Ziel von .NET war unter anderem *Language Interoperability*. Was versteht man darunter? Wird das auch von Java unterstützt?
5. Welche Bedingungen müssen .NET Programme erfüllen, damit diese „Language Interoperabel“ sind (dass also z.B. ein C#-Programm eine VB.NET-Assembly aufrufen kann).