GIT-Agreements

Im folgenden Dokument geben wir die Richtlinien zum Nutzen von GIT an.

Nutzung von Fork

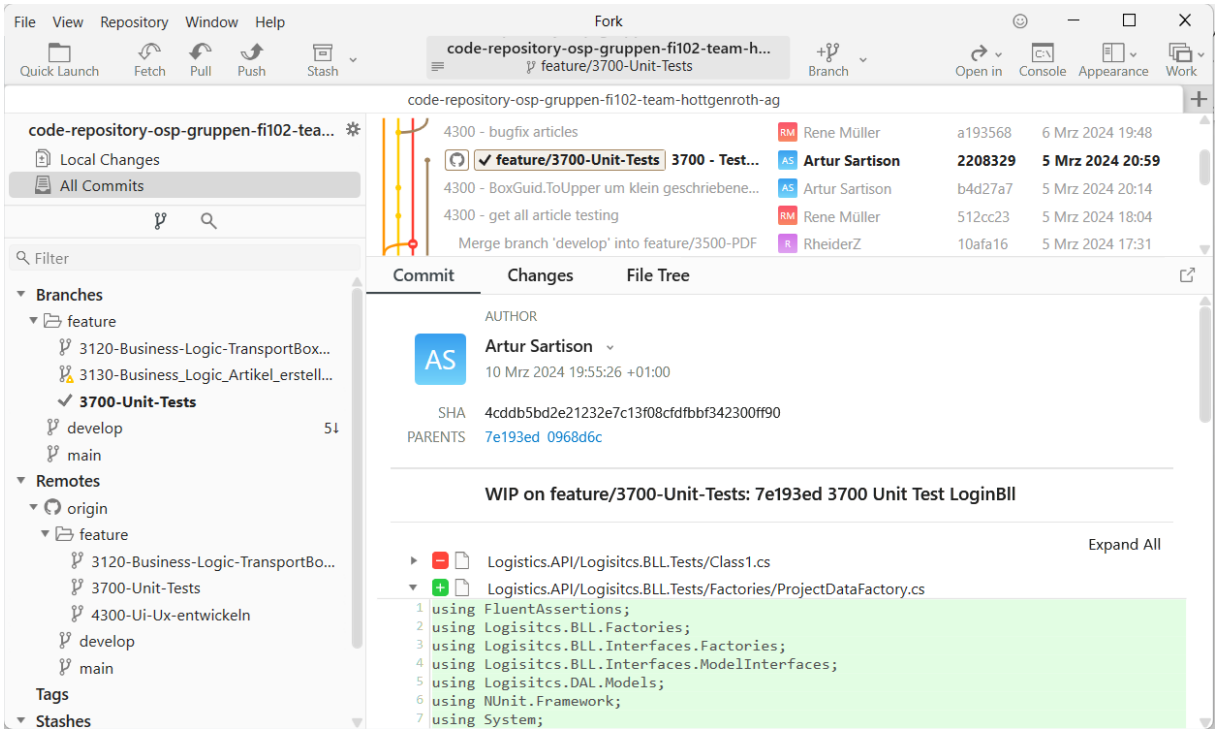
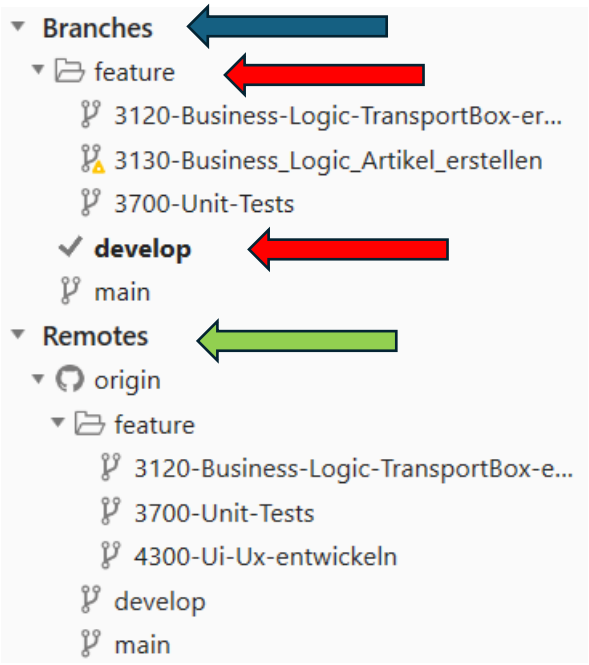
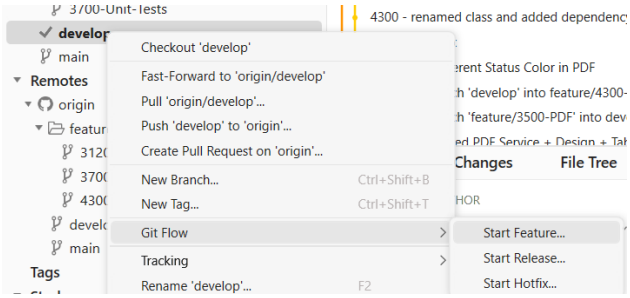


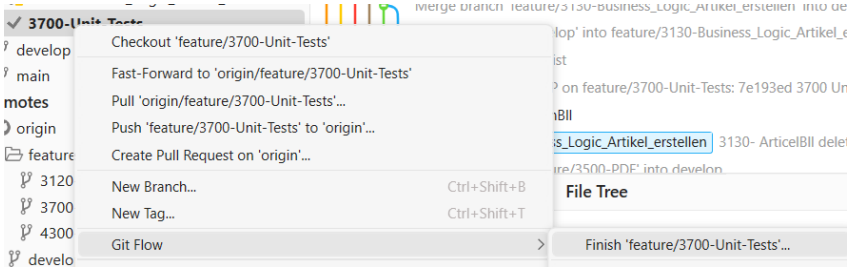
Abbildung 1 Fork Oberfläche

Als Programm zu arbeiten im GIT verwenden wir die Software Fork. Dieses bietet GIT Flow an, womit sich Features erstellen und schließen lassen. Unteranderem bietet GIT Flow eine Bearbeitungsstruktur an, dieses beinhaltet einen develop und einen feature Ordner (Rote Pfeile). Im Programm Fork gibt es immer einen Lokalen (blau) und einen Remote Anteil (grüner Pfeil). Der Lokale Anteil ist, bis man hochspielt nur auf dem Computer verfügbar. Nach einem „Push“ werden Datei Änderungen auf den Remote Spiegel gebracht und andere im Team können diese abholen, um damit arbeiten zu können.



Wenn man ein Feature starten möchte, macht man dies mit Git Flow. Mit einem Rechtsklick auf den gewünschten stand z.B. dem develop, geht man unter den Reiter Git Flow Start Feature und gibt dem Feature einen Namen.





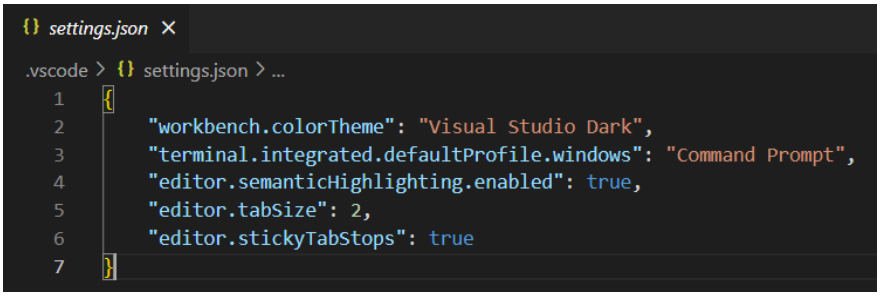
Wenn man ein Feature abgeschlossen hat und dieses Zurückführen möchte, geht man mit Rechtsklick auf Git Flow und auf „Finish Feature“. Dann wird das Feature auf den Ursprünglichen „Branch“ zurückgeführt. Sollten in der Zeit Änderungen auf dem Ursprünglichen „Branch“ vollzogen worden sein und gleiche Dateien bearbeitet sein, entsteht ein „merge“ Konflikt, der gelöst werden muss, bevor man das Feature abschließen kann.

Review Prozess innerhalb von GIT

Wir haben uns dazu entschieden Arbeitspakete als Namen von Features zu Nutzen. Dazu wird folgende Formatierung genommen „Arbeitspaket Nummer“ – „Arbeitspaket Name“ also z.B. 4300-Ui-Ux-entwickeln. Dabei ist zu beachten das man statt Leertaste ein Minus nimmt und keine Slashes einbaut da sonst ein Unterordner entsteht. Sollte man ein Feature seines Erachtens fertig gestellt haben meldet man bei einem Team Mitglied ein Review an. Dieses Beinhaltet vor dem zurückführen ein vier Augen Prinzip, d.h. es gibt einen Developer und einen Reviewer der über den Code drüber guckt. Ein Review sollte Code Verbesserung Anregungen beinhalten (best-practice) und auf Richtigkeit prüfen. Zum Abschluss führt der Reviewer nach dem er den Programm Code gesichtet hat, einen Funktionstest aus, indem er das Programm startet und die implementierte Funktion testet. Sollten diese Schritte abgeschlossen sein, wird das Feature auf den develop zurückgeführt.

Code Guidelines

Um zu gewährleisten das der Code für alle Einheitlich Formatiert wird, haben wir uns im Back-End (C#) und im Front-End (Angular) für eine vorgefertigte Konfiguration entschieden.

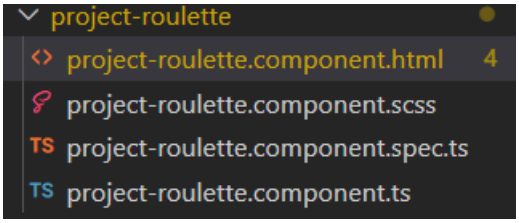


Im Front-End Teil wird das mit einer settings.json Datei angeben diese zieht die tabSize, stickyTabStops und das die default Konsole ein Command Prompt ist und nicht eine Powershell. Die tabSize gibt hierbei an, dass, wenn man eine neue Zeile angibt, diese Korrekt eingerückt wird. Die stickyTabStop Funktion bewirkt, dass VS Code Cursorbewegungen in führenden Leerzeichen ähnlich wie Tabulatoren behandelt.

Darüber hinaus, um gleichen Code zu produzieren haben wir uns auf folgende Strukturen geeinigt:

Vier Dateien

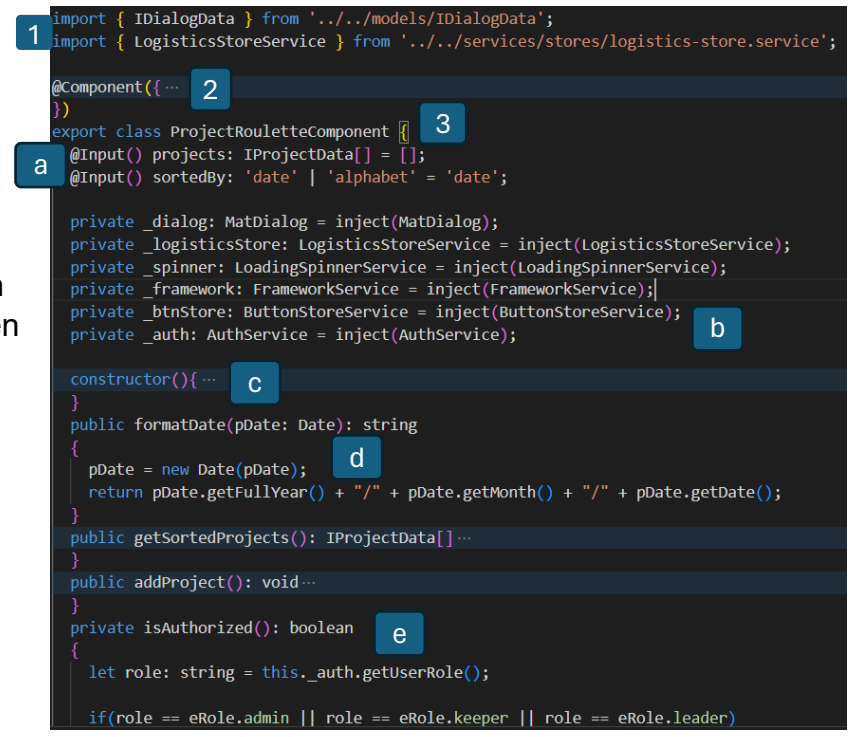
1. *.html für html Teil
2. *.scss für den Style Teil
3. *.ts für den TypeScript Teil
4. *.spec.ts für den Test Teil



Diese Dateien werden automatisch mit dem Befehl „*ng generate component [name-der-Komponente]*“ erstellt

*.ts Dateien:

1. Zuerst alle Imports
2. Decorator der Klasse
3. Klassen Definition mit:
    a. Parent Komponente (Input): Zur Objekte Initialisierung von außen
    b. Private Variablen: Namen beginnen mit „_“
    c. Konstruktor
    d. Public Methoden
    e. Private Methoden



*.html Datei:

1. Section
2. Content

```html
<section class="w-full">
  <mat-card appearance="outlined" class="w-full">
    <mat-card-header>
      <mat-icon mat-card-avatar *ngif="sortedBy === 'alphabet'" class="text-4xl">
        sort_by_alpha
      </mat-icon>
      <mat-icon mat-card-avatar *ngif="sortedBy === 'date'" class="text-4xl">
        date_range
      </mat-icon>
      <mat-card-title>
        Projects
      </mat-card-title>
      <mat-card-subtitle>
        sorted by {{sortedBy}}
      </mat-card-subtitle>
    </mat-card-header>
    <mat-card-content class="!flex flex-row overflow-x-auto gap-1">···
    </mat-card-content>
  </mat-card>
</section>
```

*.scss Datei:

beinhaltet die einzelnen Styles zwischen den einzelnen Styles wird eine Zeile frei gelassen

```scss
.selected{
  background-color: #f3f3f7;
  border: 0.5px solid black;
  border-radius: 5px;
}

.actions{
  background-color: rgba(211, 211, 211, 0.596);
  border: 2px solid black;
  border-radius: 10px;
}
```

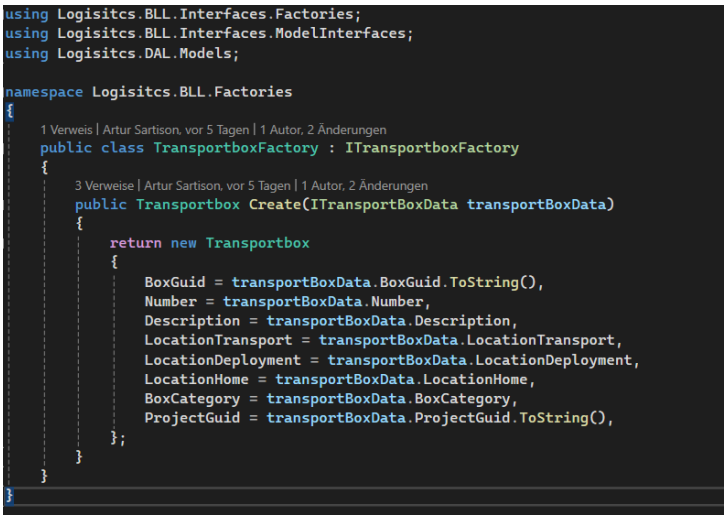
*spec.ts Datei:

Beinhaltet die Tests der einzelnen component

1. Zuerst alle imports
2. Erstellung der Testumgebung bzw. der component
3. Das Setup was vor jedem Test ausgeführt wird
4. Inhalt des Tests
5. Die Prüfung

```spec.ts
import { ComponentFixture, TestBed } from '@angular/core/testing';
import {HttpClientTestingModule } from '@angular/common/http/testing';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

import { DatePickerComponent } from './date-picker.component';
import { DatePipe } from '@angular/common';

describe('DatePickerComponent', () => {
  let component: DatePickerComponent;
  let fixture: ComponentFixture<DatePickerComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        DatePickerComponent,
        HttpClientTestingModule,
        BrowserAnimationsModule
      ],
      providers:
      [
        DatePipe
      ]
    })
    .compileComponents();

    fixture = TestBed.createComponent(DatePickerComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

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

  it('set date', () => {
    component.setDate("2012-02-20")
    expect(component.date).toEqual("2012-02-20");
  });
});
```

Im Back-End nutzen wir eine Visual Studio Extension, die für uns die Code Guidelines übernimmt. Diese Extension nennt sich CodeMaid. Dieses Überprüft bei jedem Speichern die Guidelines und setzt diese durch. Unteranderem überprüft diese auch die Namensgebung, ob diese Pascal Case sind und gibt Warnungen aus wenn dies nicht der Fall ist.



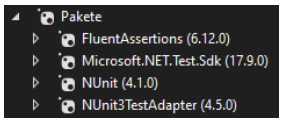
Ein paar Funktionen die CodeMaid übernimmt:

- Einrückung von Code
- Entfernen von nicht benutzten usings
- Sortieren von Klassen nach privaten Variablen, Konstruktor, Public Methoden und Private Methoden

Teststrategien

In diesem Abschnitt gehe ich kurz auf die Teststrategien ein, die wir in C# und Angular angewendet haben.

Im Backend (C#) haben wir uns dazu entschieden die Logik Schicht zu testen. Diese Schicht wird bei uns im Programm als **BLL** (**B**usiness **L**ogik **L**ayer) bezeichnet. Dort ist der Kern unserer API, diese Kommuniziert zwischen den Controllern und dem **DAL** (**D**ata **A**cces **L**ayer). Aus diesem Grund haben wir uns entschieden nur die BL Schicht zu testen. Diese Tests haben eine eigene Testdatenbank, die uns hilft, ohne die Tatsächliche Datenbank, Einträge zu verändern und die Datenbank Zugriffe zu Simulieren. Bei den Tests haben wir uns dafür entschieden einen Workflow Test durzuführen, das heißt wir bilden einen kompletten Workflow ab.



Als Framework für die Tests haben wir uns für FluentAssertions und NUnit entschieden. NUnit ist hierbei das Framework, womit man Testet, damit z.B. man Klassen bzw. Methoden als Test Markieren kann und diese von Visual Studios Test Explorer auch als Test erkannt wird. FluentAssertions wird verwendet, um die Ergebnisse am Tatsächlichen Objekt abzufragen und es mit einem erwarteten Wert zu vergleichen.

NUnit

```
[Test]
0 Verweise | Artur Sartison, Vor 14 Stunden | 1 Autor, 1 Änderung
public void Test_Update_And_Delete_Transportbox()
{
    Guid projectGuid = Guid.Parse("18580286-ad03-4240-8ec8-ffe1d3a4e77d");
    Guid boxGuid = Guid.NewGuid();
    ITransportBoxData addedTransportBoxData = new TransportBoxData
    {
        ProjectGuid = projectGuid,
        BoxGuid = boxGuid,
        Number = 201,
        Description = "Test",
        LocationDeployment = null,
        LocationHome = null,
        LocationTransport = null,
        BoxCategory = "Compound"
    };

    ITransportboxBll transportboxBll = new TransportboxBll(transportBoxDataFactory, transportboxFactory);
    //Transportbox hinzufügen
    ITransportBoxData transportBoxData = transportboxBll.AddTransportbox(addedTransportBoxData).Result;
    transportBoxData.Should().BeEquivalentTo(addedTransportBoxData);
    //TransportBox nachschauen ob die Transportbox in der DB ist
    ITransportBoxData dbTransportbox = transportboxBll.GetTransportbox(boxGuid).Result;
    dbTransportbox.Should().BeEquivalentTo(addedTransportBoxData);

    ITransportBoxData updatedTransportBoxData = new TransportBoxData
    {
        ProjectGuid = projectGuid,
        BoxGuid = boxGuid,
        Number = 201,
        Description = "TestUpdate",
        LocationDeployment = null,
        LocationHome = null,
        LocationTransport = null,
        BoxCategory = "Compound"
    };
    //TransportBox updaten
    bool updateResult = transportboxBll.UpdateTransportbox(updatedTransportBoxData).Result;
    //Prüfen ob updaten erfolgreich war
    updateResult.Should().BeTrue();
    //Nachschauen ob die Transportbox in der DB upgedatet wurde
    dbTransportbox = transportboxBll.GetTransportbox(boxGuid).Result;
    dbTransportbox.Should().BeEquivalentTo(updatedTransportBoxData);
    //Löschen der angelegten Transportbox
    bool deleteResult = transportboxBll.DeleteTransportbox(boxGuid).Result;
    //Prüfen ob löschen erfolgreich war
    deleteResult.Should().BeTrue();
}
```

FulentAssertions

Als Beispiel wird hier eine Transportbox hinzugefügt und geprüft, ob es hinzugefügt wurde. Diese Transportbox wird dann verändert und geupdatet. Nach einer Prüfung wird die Box wieder aus der Datenbank gelöscht. Dieser Test deckt nun den DAL und die BLL ab und somit unsere Logik und den Datenbank Zugriff.

Im Frontend (Angular) haben wir uns stattdessen nur für Tests entschieden die Logik von components testen. Dazu haben wir das Jasmine Framework verwendet. Dieses Framework bietet so wie FluentAssertions eine Vergleichs Option an und erlaubt es *.ts Dateien zu testen. Beispielsweise werden Sortierungen, Erstellung der Components, Validierungen und Formatierungen getestet. Der Befehl, um das Jasmine Framework zu starten ist „*ng t*“. Daraufhin bekommt man die im rechten Bild zusehende Ausgabe im Browser dargestellt und man sieht ob irgendwelche Probleme bei den Tests aufgetaucht sind.