# Dokumentation

Für SharpTodo - Projekt von Benjamin Brandtner und Lukas Dürer  
Am Beruflichen Schulzentrum für Wirtschaft und Datenverarbeitung für das Fach C-Sharp  
Lehrkraft: Katrin Stremme  
Abgabe: 27.06.2019

## Zweck des Programmes

[Habitica.com](https://habitica.com/) ist eine Online-Zeitmanagement-Anwendung, auf der man unter anderem eine Todoliste anlegen kann, tägliche Verpflichtungen (*Dailies*) und gute und schlechte Gewohnheiten (*Habits*) tracken kann. Habitica nutzt verschiedene Mechaniken aus Rollenspielen, wie z.B. einem Charakter der aufleveln oder Schaden nehmen kann, um den Nutzer spielerisch zur Erledigung seiner Todos anzutreiben und für gute Gewohnheiten zu belohnen.  
Habitica stellt eine [ausführliche API](https://habitica.com/apidoc/) zur Verfügung, mit der man 3rd Party Anwendungen entwickeln kann, die beinahe alles können was man auch auf der Webseite machen kann.  
Wir haben uns nur einen kleinen Ausschnitt der Funktionalität von Habitica hergenommen: die Todoliste. Mit *SharpTodo* kann man alle auf Habitica angelegten Todos abfragen, Änderungen machen, abhaken und neue anlegen. Das Ziel war es, eine Oberfläche zu erschaffen, die schneller und kompakter ist als die auf der Webseite.

## Frontend

### View

### Viewmodel

## Kommunikation Frontend → Backend

## Die Klasse VMMainWindow nutzt aus dem Backend den HabiticaClient, um HabiticaTodos zu laden, erstellen, usw. Sie reagiert auch auf Exceptions die im Backend geworfen werden können.

## Backend

### HabiticaClient

Im Backend ist der HabiticaClient, der von HttpClient erbt, verantwortlich, verschiedene HttpRequests an die Habitica API Server zu senden.  
Da ein Request eine Weile dauert, aber in dieser Zeit die Gui und andere Programmteile nicht stillstehen sollen, werden die Requests asynchron abgeschickt.

Für jeden Request an die API müssen bestimmte Header mitgeschickt werden, die einen auf Habitica registrierten Benutzer identifizieren.  
Die userId und das apiToken werden in den Properties.settings abgespeichert. Sind diese Einträge leer, zum Beispiel beim ersten Starten des Programmes, wird eine NoCredentialsException geworfen.

#### Kurzer Einblick in Asynchrone Programmierung mit Tasks [[1]](#footnote-1)

Durch Task Objekte und die Schlüsselwörter async und await wird die asynchrone Programmierung vereinfacht. Sie ersetzen das komplizierte manuelle Arbeiten mit Threads.

Ermöglicht wird das durch die Task Parallel Library (TPL). Sie ist der bevorzugte Weg, in .NET asynchron zu programmieren.

Innerhalb einer Methode, die selbst asynchron ist, kann auf eine weitere asynchrone Methode gewartet werden, ohne dass das Programm blockiert. Mit dem Schlüsselwort await wird die Kontrolle an den Aufrufer der Methode zurückgegeben.

async void DownloadButton\_Click()

{

//Durch das await wird nach dem Aufruf von GetStringAsync die Kontrolle an z.B. die GUI zurückgegeben.

//Ist GetStringAsync fertig ausgeführt läuft diese Methode weiter.

String s = await myClient.GetStringAsync(url);

doSomething(s);

}

### HabiticaSerializer

Da die Habitica API Daten im JSON Format erwartet und zurückschickt, nutzt der Client den HabiticaSerializer um HabitcaTodos zu serialisieren und die Server-Antworten zu deserialisieren.  
Der HabiticaSerializer nutzt das Framework *Json.NET*.  
Die meisten Methoden kriegen einen JSON-String übergeben, den parsen sie, zum Teil mit der Methode *ParseResponseData()*, und geben ein oder mehrere *HabiticaTodos* zurück. Die Methode *ParseResponseData()* reagiert folgendermaßen auf Fehlermeldungen vom Server:

Sind der eingegebene userId oder apiToken falsch, antwortet der Server, dass ein solcher Nutzer nicht existiert. Daraufhin wird eine *WrongCredentialsException* geworfen.  
Wenn etwas mit dem Request nicht stimmt, weil zum Beispiel die JSON-Daten die falsche Form haben, steht das in der Serverantwort. Daraufhin wird eine *UnsuccessfulException* geworfen, deren Message genauere Informationen enthält. Dies sollte theoretisch nur während der Entwicklung, nicht im fertigen Programm passieren.

### HabiticaTodo

Die Klasse HabiticaTodo definiert selbst nur ein Attribut, die Checklist. Es wird noch nicht benutzt, da ChecklistItems in der API gesondert behandelt werden müssen, und dies im HabitcaClient noch nicht implementiert ist. Sie erbt den Rest von der abstrakten Klasse Todo.  
Sie hat auch eine interne Klasse, mit der Prioritäten eines Todos dargestellt werden können. Diese wird noch nicht genutzt.

### Todo

Um vom JSON der API in ein Objekt der Klasse HabiticaTodo zu deserialisieren, müssen die Properties dieser Klasse um Attribute ergänzt werden, die dem JsonSerializer sagen, welche Namen die Properties im JSON haben sollen.

[JsonProperty("text")]

public string Title { get => title; set => title = value; }

Ansonsten ist diese abstrakte Klasse eine reine Datenklasse mit Feldern, Properties und einer *ToString()* Methode.

## Erweiterbarkeit des Backends

Zurzeit braucht man, um SharpTodo zu nutzen einen Account auf Habitica. Alle Todos die man anlegt werden auf diesem Account gespeichert.  
Es wäre recht simpel, stattdessen lokale Speicherung zu implementieren. Ich würde dafür eine Klasse LocalClient anlegen, die dieselben Methoden hat wie der HabiticaClient (hier bietet sich auch ein Interface TodoClient an). Der LocalClient speichert dann Todos z.B. in einer lokalen Datenbank oder JSON-Datei ab.

Auf der Oberfläche würde ich eine Einstellung hinzufügen, mit der man zwischen dem HabitcaClient und dem LocalClient wechseln kann.

1. Siehe: <https://docs.microsoft.com/en-us/dotnet/csharp/async> ,

   <https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming> [↑](#footnote-ref-1)