FEIERTAGE BERECHNEN MIT NAGER.DATE

Feiertagsrechner

Das Handling von Datumsangaben ist mühsam und fehleranfällig. Hier hilft Nager.Date.

Die Berechnung mancher Feiertage ist einfach, zum Beispiel für die festen oder nicht beweglichen. Neujahr ist immer am 1. Januar, der Tag der Arbeit am 1. Mai und der erste Weihnachtsfeiertag am 25. Dezember.

Bei den beweglichen Feiertagen wird die Sache schon komplizierter. Und erst recht, wenn freie Tage im Ausland berücksichtigt werden sollen. So ist zum Beispiel der 7. April 2021 der Women's Day in Mosambik und der 9. April 2021 der Martyrs' Day in Tunesien.

Vieles wird beispielsweise vom Ostersonntag aus berechnet. Gründonnerstag ist der Ostersonntag (OS) minus drei Tage. Karfreitag ist OS minus zwei Tage. Pfingstsonntag ist Ostersonntag plus 49 Tage und so weiter. Der Ostersonntag lässt sich mit der gaußschen Osterformel [1] nach Carl Friedrich Gauß berechnen (siehe Kasten Die Osterformel), und der Buß- und Bettag hat eine eigene Formel.

Alles in allem ist das aufwendig und kostet Zeit. Dabei sind die Anwendungsfälle zahlreich: zum Beispiel wenn es gilt, die langen Wochenenden für einen Urlaub zu ermitteln, den letzten Montag in einem Monat oder den letzten Bankarbeitstag von einem Datum aus zu berechnen.

Daher ist der Einsatz einer Bibliothek mindestens eine Überlegung wert. Es gibt mittlerweile zahlreiche Implementierungen, die alle ihre Vor- und Nachteile haben. Zudem gibt es noch verschiedene Herangehensweisen an das Problem, wenn doch keine Bibliothek zum Einsatz kommen soll oder wenn die Daten beispielsweise zwischengespeichert werden müssen. Darauf geht der Artikel gegen Ende ein.

Der Hauptfokus in diesem Artikel liegt aber auf der Bibliothek Nager.Date [2]. Sie verspricht, die zahlreichen Fallstricke und Probleme bei der Feiertagsberechnung zu lösen. Nicht nur in Form einer Bibliothek mit .NET-API, sondern auch über eine REST-Schnittstelle und als Self-hosted-Variante. Was die Bibliothek alles kann, zeigt der Artikel im weiteren Verlauf.

Was ist Nager.Date?

Nager.Date ist eine in C# geschriebene Bibliothek für das .NET-Universum. Die Kernfunktionen lassen sich am besten damit beschreiben, dass mit Nager.Date Feiertage und Wochenenden abgefragt werden können. Zu einem gegebenen Datum gibt die Bibliothek somit die Information zurück, ob es ein Feiertag, ein Wochenende oder ein ganz normaler Tag ist. Auch die Abfrage nach einem spezifischen Tag von einem Datum aus oder die Abfrage von langen Wochenenden sind mit Nager.Date realisierbar.

Nager.Date gilt als performant und unterstützt über 100 Länder. Damit ist die Bibliothek insbesondere in Projekten sehr interessant, in denen nicht nur deutsche Feiertage berechnet werden müssen.

Zudem wird Nager.Date nicht nur als Bibliothek angeboten, sondern besitzt darüber hinaus ein öffentliches API, das auch in Form eines Docker-Containers zum Selbsthosten zur Verfügung steht.

Wer einen Überblick über die unterstützten Länder bekommen möchte, findet eine gute Übersicht auf der Website von Nager.Date [3]. Dabei wird deutlich, dass Europa mit über 98 Prozent am besten abgedeckt ist, gefolgt von Amerika mit über 57 Prozent, Afrika mit über 23 Prozent und Asien mit 14 Prozent. Ozeanien und die Antarktis liegen darunter.

Installation

Die Installation von Nager.Date ist kein Problem. Es steht ein NuGet-Paket zur Verfügung, um die Bibliothek zum eigenen Projekt hinzuzufügen. Zudem existiert ein öffentliches API, implementiert als REST-Schnittstelle, und ein Docker-Container, falls zu viele API-Anfragen geplant sind und daher eine selbstgehostete Lösung zu bevorzugen ist. Beim Schreiben dieses Artikels war die Version 1.28.2 vom März 2021 aktu-

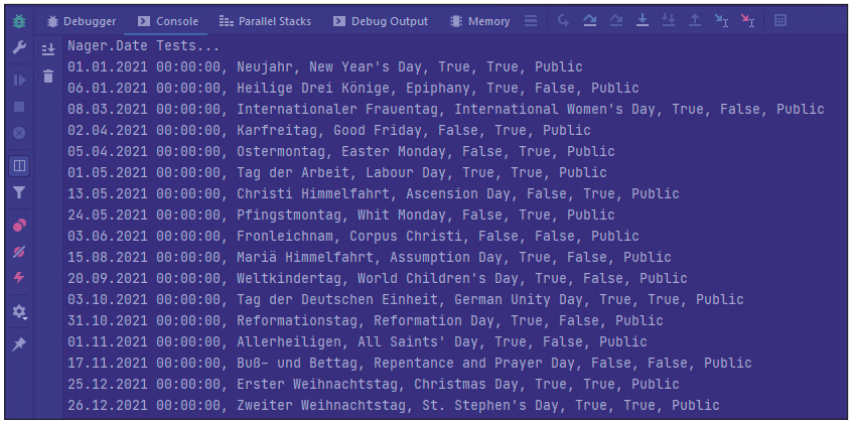
Die Osterformel

Die sogenannte Osterformel wurde von Carl Friedrich Gauß entwickelt und öfters überarbeitet. Die Formel besteht aus mehreren Teilen. Zudem sind zwei Konstanten, *M* und *N*, im Einsatz. Diese sind Hilfszahlen für den Gregorianischen Kalender und ändern sich im 100-Jahre-Rhythmus. Die Herleitung ist an dieser Stelle nicht so wichtig. Zwischen den Jahren 2000 und 2099 werden diese Konstanten auf $M = 24$ und $N = 5$ festgelegt. Die Osterformel lautet im Detail:

$a = Jahr \bmod 4$
$b = Jahr \bmod 7$
$c = Jahr \bmod 19$
$d = (19c + M) \bmod 30$
$e = (2a + 4b + 6d + N) \bmod 7$

Berechnung des Ostertags:
$f = (c+11d+22e)/451$

Der Ostersonntag ergibt sich aus der Formel $22+d+e-7f$. Wenn das Ergebnis größer 31 ist, so liegt Ostern im April. Dann muss vom Ergebnis der Ostersonntag-Formel die Zahl 31 abgezogen werden, sodass sich die Formel zu $d+e-7f-9$ ändert.



Die Ausgabe von Feiertagen für das Jahr 2021 auf der Konsole in JetBrains Rider (Bild 1)

ell. Die Website [2] zum Projekt erklärt, welche Funktionen und Länder zur Verfügung stehen. Eine eigene Unterseite [4] zeigt auf, wie das API aufgebaut ist. Darüber hinaus existiert eine mit Swagger erstellte eigene Schnittstellendokumentation [5] ebenfalls als Unterseite zur Projektwebsite.

Der Code zu Nager.Date ist auf GitHub [6] verfügbar. Die Bibliothek ist Open Source und steht unter der MIT-Lizenz. Der hauptverantwortliche Entwickler von Nager.Date ist Tino Hager [7] aus Österreich, der die Entwicklung an Nager.Date bereits im Jahr 2014 gestartet hat. Im Rahmen dieser Kolumne gab es bereits die Vorstellung des Projekts Nager.PublicSuffix [8], das ebenfalls von Tino initiiert wurde.

Die Beispielanwendung zum Artikel ist mit .NET Core 3.1 als Konsolenanwendung realisiert, was keinerlei Probleme verursacht hat.

Abfrage von Feiertagen

Wie schon erwähnt, lässt sich Nager.Date im Code als Bibliothek oder über das REST-API als Online- beziehungsweise selbst gehostete Variante in Form eines Docker-Containers einbinden. Zu diesem REST-API gibt es auch NPM-Pakete, die allerdings nur die Nutzung des API kapseln. Ob das eine gute Idee ist, bleibt dahingestellt, denn das Original-API kann auch einfach abgefragt werden. Die Schnittstellen dieser unterschiedlichen Implementierungen sind vergleichbar bis identisch. Es gibt die Möglichkeit, zu einem Jahr und einem Länderkennzeichen die Feiertage zu erhalten. Der REST-Endpunkt lautet dazu

```
/Api/v2/PublicHolidays
  /{year}/{countryCode}
```

Im .NET-Code ist dafür

```
DateSystem.GetPublicHoliday(...)
```

zuständig. Die zusätzliche Angabe eines Start- und eines End-Datums ermöglicht die Eingrenzung der zurückgelieferten Daten. Auf diese Weise kann auch für einen einzelnen Tag geprüft werden, ob es sich um einen Feiertag handelt. Ein Aufruf der Schnittstelle

```
/Api/v2/IsTodayPublicHoliday/{countryCode}
```

ermittelt, ob es sich bei dem aktuellen Tag und dem übergebenen Länderkennzeichen um einen Feiertag handelt. Das ist für Geräte des Internets der Dinge spannend. Auch Telefonanlagen können davon profitieren, um beispielsweise festzustellen, ob die Mailbox aktiv geschaltet werden soll oder nicht. Dazu muss nur der Statuscode ausgewertet werden. Liefert ein Aufruf *200* zurück, ist der aktuelle Tag ein Feiertag. Bei der Rückgabe *204* ist es kein Feiertag.

Im .NET-Code existiert dafür die Methode

```
DateSystem.IsPublicHoliday(...)
```

die es zudem erlaubt, ein Datum zu übergeben. Darüber hinaus gibt es zahlreiche Schnittstellen, um den nächsten Feiertag für ein Länderkennzeichen oder weltweit zu ermitteln, für ein Datum und ein Länderkennzeichen zu prüfen, ob es sich um ein langes Wochenende handelt, und um die verfügbaren Länder zu ermitteln, die von Nager.Date unterstützt werden.

Listing 1 zeigt ein Beispiel dafür, wie die Feiertage für ein gesamtes Jahr ermittelt werden können. Bild 1 zeigt die Konsolenausgabe von JetBrains Rider.

In diesem Beispiel werden Informationen wie das Datum, der lokale Name des Feiertags im jeweiligen Land, in diesem Fall auf Deutsch, der englische Name, sowie die Informationen, ob es sich um einen fixen Feiertag, einen globalen Feiertag und um welchen Typ es sich handelt, ausgegeben. Letzteres gibt an, ob es sich um einen öffentlichen Feiertag oder zum Beispiel einen freien Tag für Schulen handelt.

Bei dem Beispiel fällt auf, dass der Ostersonntag nicht zurückgegeben wird. Das liegt daran, dass die Beispielabfrage für ganz Deutschland durchgeführt wurde. Da der Ostersonntag immer ein Sonntag ist, wird dieser nicht als Feiertag geführt. Laut Wikipedia führt nur ein Bundesland, nämlich Brandenburg, den Ostersonntag offiziell als Feiertag. In ▶

Listing 1: Feiertage für ein Land für ein ganzes Jahr

```
var publicHolidays =
  DateSystem.GetPublicHoliday(2021, "DE");

foreach (var publicHoliday in publicHolidays)
{
  Console.WriteLine(
    $"{publicHoliday.Date},
    {publicHoliday.LocalName},
    {publicHoliday.Name}, {publicHoliday.Fixed}, " +
    $"{publicHoliday.Global},
    {publicHoliday.Type}");
}
```

Hessen ist es beispielsweise so, dass jeder Sonntag als gesetzlicher Feiertag gilt.

Abfrage von spezifischen Tagen

Wer nicht ein komplettes Jahr abfragen möchte, kann bei der gleichen Methode eine Datumsspanne angeben. Listing 2 zeigt dazu ein Beispiel. Die Methode erlaubt es zudem, die Datumsspanne auf ein einziges Datum festzulegen:

```
var startDate = new DateTime(2021, 4, 2);
var endDate = new DateTime(2021, 4, 2);
```

Wenn es an diesem Tag einen Feiertag gibt, wird dieser in diesem Beispiel, wie die vorherigen Beispiele auch, auf der Konsole ausgegeben:

```
02.04.2021 00:00:00, Karfreitag, Good Friday,
  False, True, Public
```

Alternativ lässt sich über die *IsPublicHoliday*-Methode prüfen, ob ein Tag ein Feiertag ist:

```
var date = new DateTime(2021, 4, 5);
if (DateSystem.IsPublicHoliday(date, CountryCode.DE)) {
  Console.WriteLine("Dieser Tag ist ein Feiertag!");
}
```

Das liefert aber nur zurück, ob es sich um einen Feiertag handelt oder nicht, und nicht, um welchen Feiertag es sich handelt.

Eine sehr ähnliche Vorgehensweise lässt sich nutzen, um abzufragen, ob ein Datum auf ein Wochenende fällt:

```
var weekendDate = new DateTime(2021, 4, 4);
if (DateSystem.IsWeekend(weekendDate, CountryCode.DE)) {
  Console.WriteLine("Dieser Tag ist ein Wochenende!");
}
```

Listing 2: Feiertage für ein Land in einem Bereich

```
var startDate = new DateTime(2021, 4, 1);
var endDate = new DateTime(2021, 4, 30);
var publicHolidaysRange =
  DateSystem.GetPublicHoliday(
    startDate, endDate, CountryCode.DE);

foreach (var publicHoliday in publicHolidaysRange)
{
  Console.WriteLine(
    $"{publicHoliday.Date},
    {publicHoliday.LocalName},
    {publicHoliday.Name}, {publicHoliday.Fixed}, " +
    $"{publicHoliday.Global},
    {publicHoliday.Type}");
}
```

Weitere Abfragemöglichkeiten

Nager.Date bietet über die Methoden *FindDay*, *FindDayBefore*, *FindDayBetween* und *FindLastDay* noch weitere Abfragen an. *FindDay* ermöglicht es, nach einem bestimmten Wochentag zu suchen, der als Nächstes vom angegebenen Startdatum aufzufinden ist, zum Beispiel den nächsten Freitag nach dem 7. April 2021:

```
var day = DateSystem.FindDay(2021, 4, 7,
  DayOfWeek.Friday);
```

Zurückgeliefert wird korrekterweise der 9. April 2021.

Ähnlich, aber mit einer Rückwärtssuche, agiert die Methode *FindDayBefore*. Sie sucht nach dem spezifischen Wochentag rückwärts vom angegebenen Datum aus:

```
var dayBefore = DateSystem.FindDayBefore(2021, 4, 7,
  DayOfWeek.Friday);
```

In diesem Fall wird der 2. April 2021 zurückgeliefert.

Ebenfalls sehr nützlich sind die Methoden *FindDayBetween* und *FindLastDay*. Erstgenannte sucht einen Wochentag zwischen zwei Datumsangaben. Letztgenannte sucht den letzten Wochentag in einem Monat. Im folgenden Beispiel den letzten Montag im April 2021:

```
var lastMonday = DateSystem.FindLastDay(2021, 4,
  DayOfWeek.Monday);
```

Ebenfalls in vielen Anwendungsszenarien nützlich ist die Methode *GetLongWeekend*. Sie ermittelt die langen Wochenenden, die sich durch Feiertage ergeben können. Ein Aufruf für das Jahr 2021 ermittelt davon acht Stück:

```
var longWeekends =
  DateSystem.GetLongWeekend(2021, CountryCode.DE);
```

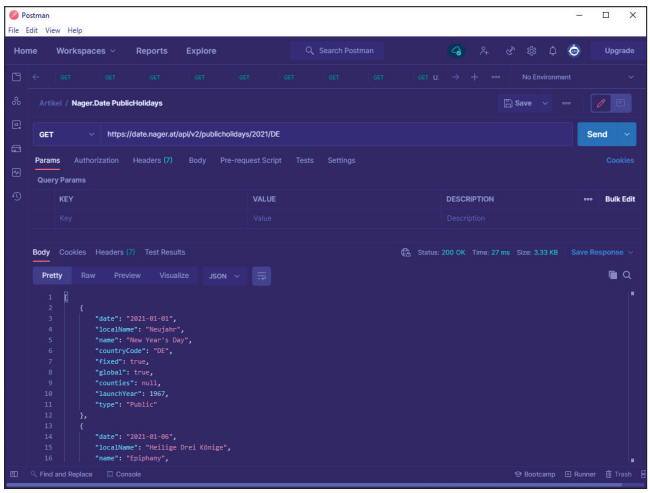
Zum Beispiel zwischen dem 13. Mai 2021 und 16. Mai 2021. Dort gibt es insgesamt vier Tage. Interessant ist die Angabe, ob bei dem langen Wochenende ein Brückentag notwendig ist. Bei dem genannten Beispiel von 13. bis zum 16. Mai 2021 ist das der Fall, da am Freitag, dem 14. Mai 2021 Urlaub genommen werden muss.

Die zuvor beschriebenen Beispiele und zurückgelieferten Daten des .NET-API stehen auch über die REST-Schnittstelle zur Verfügung.

In der mit Swagger erstellten Dokumentation [5] werden die Data Transfer Objects (DTOs) als Schemata aufgeführt, die von den Endpunkten genutzt beziehungsweise zurückgeliefert werden. Insbesondere die DTOs *LongWeekendDto* und *PublicHolidayDto* liefert die Informationen zurück, die in den vorherigen Beispielen genutzt wurden.

Nutzung des REST-API

Das REST-API [4] von Nager.Date zu nutzen ist ebenso einfach wie die vorherigen C#-Beispiele. Die Dokumentation zum API beschreibt die Möglichkeit sehr gut und detailliert



Beispiel-Request zur Abfrage der Feiertage für das Jahr 2021 in Deutschland über das öffentliche API (Bild 2)

mit Beispielen, sowie die Möglichkeit, sie direkt online auszuführen. Das folgende Beispiel ist mit Postman entstanden. Bild 2 zeigt die Abfrage des folgenden GET-Requests:

```
https://date.nager.at/api/v2/publicholidays/2021/DE
```

Zurückgeliefert werden alle Feiertage im JSON-Format mit den gewohnten Daten, die auch schon in der C#-Version zur Verfügung standen:

```
{
  "date": "2021-04-05",
  "localName": "Ostermontag",
  "name": "Easter Monday",
  "countryCode": "DE",
  "fixed": false,
  "global": true,
  "counties": null,
  "launchYear": 1642,
  "type": "Public"
}
```

Die Macher von Nager.Date bitten darum, dass dieses öffentliche API nur dann genutzt wird, wenn weniger als 50 Requests pro Tag benötigt werden.

Wer mehr nutzen oder den Service lieber selbst anbieten möchte, kann die Self-hosted-Variante von Nager.Date nutzen. Dafür steht ein Docker-Container auf der Dockerhub-Website zur Verfügung.

Über den folgenden Aufruf lässt sich der Container herunterladen und starten:

```
docker run -e "EnableCors=true"
  -e "EnableIpRateLimiting=false"
  -e "EnableSwaggerMode=true" -p 80:80 nager/nager-date
```

Anschließend steht das REST-API direkt unter der *localhost*-Adresse zur Verfügung:

```
localhost:80/api/v2/publicholidays/2020/DE
```

Alternativ lässt sich über Swagger ein Client für das API erzeugen:

```
docker run --rm -v C:\Temp:/local
  swaggerapi/swagger-codegen-cli-v3 generate
  -i https://date.nager.at/swagger/v1.0/swagger.json
  -l csharp-dotnet2 -o /local/out
```

Auch das lief im Test problemlos.

Alternative Projekte

Wer Nager.Date nicht einsetzen möchte oder kann, findet etliche alternative Bibliotheken dazu.

Für PHP gibt es beispielsweise die Bibliothek Yasumi [9], die nach dem japanischen Wort für Feiertag benannt ist. Das Projekt [10] wird regelmäßig gepflegt und unterstützt mittlerweile 37 Länder und über 100 sogenannter Unterbereiche (Subdivisions). Mit diesen Unterbereichen sind verschiedene Regierungszonen in den jeweiligen Ländern gemeint, die sich auf die Feiertage auswirken können – wie etwa die Bundesländer in Deutschland. Beim Schreiben dieses Artikels ist Version 2.3.0 aus dem Juni 2020 die neueste Version.

Eine ebenso gut gepflegte Bibliothek für JavaScript ist date-holidays [11]. Unterstützt werden beim Schreiben dieses Artikels über 140 Länder inklusive vieler Staaten und Regionen. Das Projekt [12] unterstützt zahlreiche Browser und lässt sich so konfigurieren, dass nur ausgewählte Länder zur Verfügung stehen. Auf diese Weise wird der Bedarf an Speicherplatz reduziert, was für Webprojekte mitunter enorm wichtig ist.

Für Java steht Jollyday [13] zur Verfügung, das weniger aktuell zu sein scheint als die beiden vorherigen Bibliotheken. Unterstützt werden über 60 Länder. Laut Readme des GitHub-Projekts ist ein neuer Branch in Planung, der das Jakarta XML Binding ersetzen möchte. Beim Schreiben dieses Artikels scheint das aber noch nicht umgesetzt zu sein.

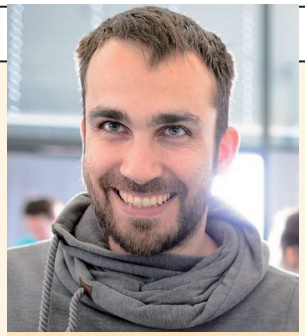
Für das .NET Framework existiert neben Nager.Date noch die Bibliothek Holiday [14]. Auch diese Bibliothek ist nicht ganz so gepflegt wie die ersten beiden, was auf den ersten Blick aber kein Problem zu sein scheint, denn die beim Schreiben des Artikels zuletzt veröffentlichte Version 2.3.0.0 ist aus dem November 2020. Unterstützt werden über 20 Länder mit einem starken Fokus auf Europa.

Das API ist vergleichbar zu Nager.Date. Etwas befremdlich sieht die Designentscheidung aus, die verschiedenen Länder in eigene Kalender auszulagern, die dann von der Bibliothek herangezogen werden, anstatt ein Länderkennzeichen oder Ähnliches zu nutzen. Das scheint den Einsatz der Bibliothek in Szenarien mit verschiedenen Ländern unnötig zu verkomplizieren.

Wer unter Python auf der Suche nach einer Bibliothek ist, darf beispielsweise zwischen python-holidays [15] und Workalendar [16] auswählen. Beide Bibliotheken sind sehr aktuell. Bei python-holidays [17] ist die zuletzt genannte Version die 0.11.1 von Anfang April 2021, und bei Workalendar [18] ist das der Release von Version 15.1.0 aus dem März 2021. Die ▶

Interview mit Tino Hager

Nager.Date hat 1,2 Millionen Downloads



Tino Hager
ist 36 Jahre alt und kommt aus Dornbirn (Österreich). Er entwickelt schon seit 20 Jahren Software, und es macht ihm noch immer sehr viel Spaß. Er ist Initiator von Nager.Date und Nager.PublicSuffix.

Wann hat deine Arbeit an Nager.Date begonnen, und kannst du uns den Hauptgrund nennen, der zum Start beigetragen hat?
Tino Hager: Nager.Date ist wahrscheinlich so wie viele andere Projekte aus Eigenbedarf entstanden. Als ich mich im Jahr 2014 mit dem Thema beschäftigt habe, wollte ich nicht wie damals üblich die Daten jährlich neu importieren und suchte daher nach einer nachhaltigen Lösung. Das war der Start von Nager.Date, und seitdem gibt es immer was zu tun.

Arbeitet ihr in einem Team oder nimmst du jede Änderung selbst vor? Wie seid ihr organisiert?
Hager: Ursprünglich habe ich das Projekt allein gestartet. Da das Projekt und die Community aber so schnell gewachsen sind, wollte ich es nicht nur von mir abhängig machen. Seit ein paar Wochen sind wir ein Team, Martin Stühmer (novaCapta) unterstützt mich nun. Wir würden uns jedoch über weitere Unterstützung und Pull Requests freuen.

Wie bist du auf den Projektnamen gekommen?
Hager: Der Projektname ist schnell erklärt: Meine Projekte tragen alle das Präfix Nager, und da sich vieles im Projekt um Datumsbausteine handelt, kam es zu Nager.Date. Heute würde ich es vielleicht Nager.Holiday nennen.

Arbeitest du noch an fremden Open-Source-Projekten mit?
Hager: Da Nager.Date nur eines meiner Open-Source-Projekte ist und ich noch einige zusätzliche Projekte habe, fehlt mir dazu die Zeit.

Nutzt du zur Entwicklung spezielle Vorgehensweise oder Tools? Wie testest du insbesondere die korrekte Berechnung von Feiertagen oder Wochenenden? Das kann doch mitunter sehr aufwendig sein, oder?
Hager: Mittlerweile kommen die meisten Anregungen/Verbesserungen aus der Community, woraufhin wir uns erst einmal mit der Recherche beschäftigen – etwa zu Berechnungsgrundlagen oder weiteren Informationen wie Wikipedia oder andere Quellen. Leider gibt es für viele Feiertage, vor allem außerhalb von Europa, keine einheitlichen Quellen, sodass wir hier nicht jede Community-Anfrage sofort bearbeiten können.

Unterscheidet sich die Komplexität in der Berechnung/Abfrage bei den verschiedenen Ländern, die Nager.Date unterstützt?
Hager: Katholische Länder basieren eigentlich fast immer auf dem Ostersonntag. Hier haben wir eine gut dokumentierte Berechnungsgrundlage. Die Komplexität beschränkt sich dann meist nur auf die nationalen und regionalen Besonderheiten. Schwieriger ist es mit Ländern, welche die Feiertage von einem Mondkalender abhängig machen, wie zum Beispiel für die Berechnung von Ramadan und Bayram.

Wie baut ihr die Unterstützung für neue Länder ein?
Hager: Dies ist von Land zu Land unterschiedlich, folgt aber meistens dem gleichen Muster. Für die Recherche nach Quellen reicht oft schon eine kurze Suche nach dem Land und den Public Holidays. Danach folgt meist eine einfache Implementierung des Providers und einem dazugehörigen Satz an Unit-Tests, damit wir die Ergebnisse der Implementierung validieren können.

Hast du einen ungefähren Überblick über die Verbreitung von Nager.Date?
Hager: Auf NuGet haben wir bereits 1,2 Millionen Downloads. Der Docker Container wurde laut Dockerhub bereits 100 000-mal heruntergeladen. Das REST-API hat täglich circa 200 000 Zugriffe. Einige bekannte GitHub-Projekte setzten mittlerweile auf unser

erstgenannte Bibliothek unterstützt über 30, die letztgenannte Bibliothek über 50 Länder.

Diese kleine Übersicht verdeutlicht, dass Nager.Date vom Funktionsumfang ziemlich weit vorne ist. Insbesondere bei der Unterstützung von Ländern, Staaten und Regionen kann keine andere Bibliothek mithalten.

Weitere Dienste

Neben den genannten Projekten, die als Bibliotheken oder Module zur Verfügung stehen, gibt es zudem zahlreiche öffentliche APIs. Viele von diesen besitzen einen noch deutlich größeren Funktionsumfang als Nager.Date, zumindest was die unterstützten Länder betrifft.

Das Working Days API [19] liefert zum Beispiel die Anzahl der Arbeitstage, Wochentage und eine Liste der Feiertage ab einem Datum zurück. Das Calendarific Global Holidays API [20] erlaubt den Zugriff auf die Datenbank von Calendarific und ermöglicht es so, auf die Feiertage von über 200 Ländern zuzugreifen. Von diesen Beispielen gibt es noch zahlreiche weitere – zu viele, um sie hier alle aufzuführen. Spezielle Übersichtsseiten [21] und Blogposts bieten einen guten ersten Eindruck, welche APIs es gibt.

Fazit

Nager.Date ist eine ausgezeichnete Bibliothek. Der Funktionsumfang ist zwar überschaubar, passt aber sehr gut zum

NuGet-Paket wie Material.Blazor [22] und Our Umbraco [23]. Die Firma Resourcify verwendet Nager.Date zum Beispiel, um zu verhindern, dass Entsorgungsaufträge versehentlich auf einen Feiertag gelegt werden.

Verdienst du mit Nager.Date oder einem deiner anderen Open-Source-Projekte Geld?
Hager: Ja, durch einen sehr freundlichen Sponsor hat das Projekt mittlerweile ein monatliches Einkommen von 5 Euro. Wir würden uns freuen, wenn uns weitere Personen oder Firmen für die Arbeit an dem Projekt unterstützen: in Form von einem GitHub-Sponsoring, damit wir auf lange Sicht unsere monatlichen Kosten decken können. GitHub Sponsor bietet eine einfache Möglichkeit für Privatpersonen wie auch Unternehmen, dieses und andere Open-Source-Projekte zu unterstützen, darüber hinaus wird dies konform mit einer Rechnung belegt.

Gibt es eine (interne) Roadmap zu Nager.Date? Planst du generell eine Weiterentwicklung, und wenn ja, welches Feature würdest du als nächstes planen?
- Die Webseite auf .NET 5 aktualisieren.
- Moment.js durch eine moderne JavaScript-Library ersetzen.
- Den Länder-Support für Amerika weiter ausbauen.
- Ich möchte das ursprüngliche Datum des Feiertags integrieren, wenn er aus verschiedenen Gründen verschoben wird.
- Eindeutige IDs für Feiertage, damit diese über Jahre oder Ländergrenzen gruppiert werden können.
- Dazu kommen noch diverse offene Anfragen aus der Community, wie zum Beispiel die Unterscheidung von Feiertagen, die auf Städten basieren.

Wenn du einen Wunsch in Bezug auf Nager.Date frei hättest, welche Änderung würdest du dir wünschen?
Hager: Weitere Unterstützer und Teammitglieder würden uns bei dem Chaos der weltweiten Feiertagsregelungen und nationalen-/regionalen Besonderheiten definitiv helfen. Dann würde uns jede Feature-Anfrage und jeder Bugfix deutlich einfacher von der Hand gehen.

Anwendungsfall. Die Dokumentation ist zudem gut und die verschiedenen Möglichkeiten, Nager.Date einzusetzen, vervielfachen die Anwendungsfälle noch einmal.

Zudem ist der Umfang der unterstützten Länder, Staaten und Regionen in Nager.Date ebenfalls sehr groß. Wer insbesondere mit Europa und Amerika zu tun hat, kann Nager.Date sicherlich gut einsetzen. Bei vielen anderen Ländern hat auch diese Bibliothek noch Nachholbedarf. Dennoch ist die Rate der unterstützten Länder deutlich höher als bei den anderen erwähnten Open-Source-Bibliotheken.

Alles in allem hat sich Nager.Date eine Empfehlung und ein „Sehr gut" verdient. Wer viel mit Feiertagen zu tun hat, darf dieser Bibliothek gerne eine Chance geben. ■

[1] Die Osterformel auf Wikipedia, www.dotnetpro.de/SL2106Frameworks1
[2] Die offizielle Projektwebsite zu Nager.Date, https://date.nager.at
[3] Die Übersicht der unterstützten Regionen von Nager.Date, www.dotnetpro.de/SL2106Frameworks2
[4] Informationen zum Nager.Date API, www.dotnetpro.de/SL2106Frameworks3
[5] Die Schnittstellendokumentation (Swagger) zum Nager.Date API, www.dotnetpro.de/SL2106Frameworks4
[6] Das Nager.Date-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks5
[7] Das Profil von Tino Hager auf GitHub, www.dotnetpro.de/SL2106Frameworks6
[8] Fabian Deitelhoff, Domains nachschlagen, dotnetpro 10/2019, Seite 72 f., www.dotnetpro.de/A1910Frameworks
[9] Das Yasumi-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks7
[10] Die offizielle Projektwebsite zu Yasumi, www.yasumi.dev
[11] Das date-holidays-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks8
[12] Die offizielle Projektwebsite zu date-holidays, www.dotnetpro.de/SL2106Frameworks9
[13] Das Jollyday-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks10
[14] Das Holiday-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks11
[15] Das python-holidays-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks12
[16] Das Workalendar-Repository auf GitHub, www.dotnetpro.de/SL2106Frameworks13
[17] Die offizielle Projektwebsite zu python-holidays, www.dotnetpro.de/SL2106Frameworks14
[18] Die offizielle Projektwebsite zu Workalendar, www.dotnetpro.de/SL2106Frameworks15
[19] Informationen zum Working Days API, www.dotnetpro.de/SL2106Frameworks16
[20] Informationen zum Calendarific Global Holidays API, www.dotnetpro.de/SL2106Frameworks17
[21] Übersichtsseite zu weiteren APIs für Ferientage, www.dotnetpro.de/SL2106Frameworks18
[22] Material.Blazor, www.dotnetpro.de/SL2106Frameworks19
[23] Our Umbraco, www.dotnetpro.de/SL2106Frameworks20

Dr. Fabian Deitelhoff
arbeitet nach seiner Promotion als Innovation- und Transfermanager am Centrum für Entrepreneurship & Transfer an der TU Dortmund. Auch ist er als Autor, Dozent und Softwareentwickler im .NET- und Web-Umfeld tätig.
@FDeitelhoff

dnpCode A2106Frameworks