b.i.b. International College

Dokumentation REST-API

Benedikt Ahle | Patrick Meyer | André Münstermann

10.09.2014

1 Ziele

1.1 Ziele des Sprints

Unser Ziel war es eine funktionsfähige Rest-API zu entwickeln, welche aus C# (App) und PHP (Website) benutzt werden können. Dies teilte sich in die folgenden Aufgaben auf:

1.1.1 Funktionsfähiger Rest-API HTTP Server
1.1.2 Funktionsfähige c# Wrapper Klasse
1.1.3 Funktionsfähige php Wrapper Klasse

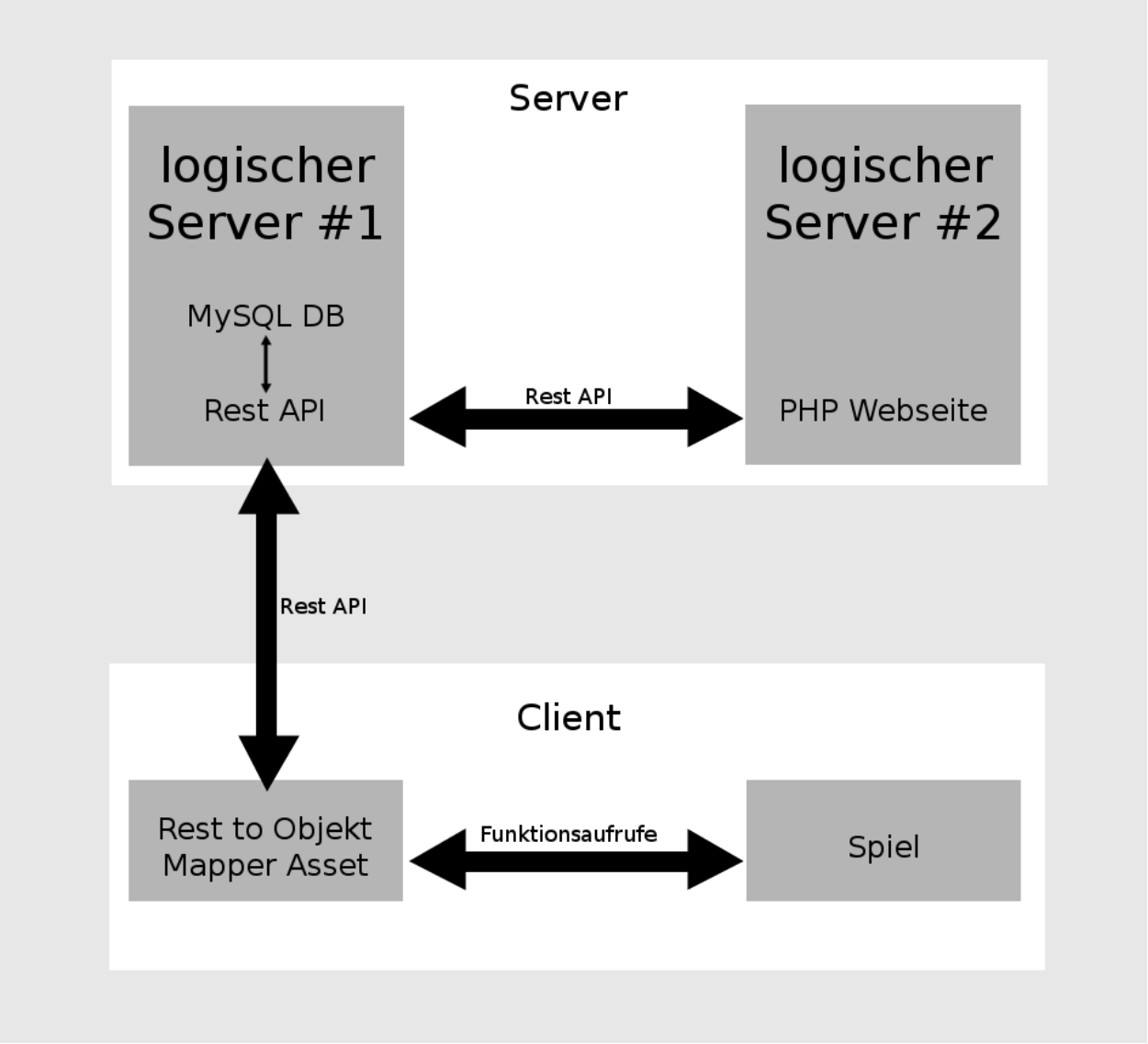

Abbildung 1: Übersicht Kommunikation der Komponenten

2 Umsetzung

2.1.1 Funktionsfähiger Rest-API Webserver

Im Laufe des Projektes wurde die RestAPI fertig gestellt. Das Datenbank Modell ist stabil und sollte sich nichtmehr großartig ändern. Es werden nur noch kleinere Änderungen vorgenommen, oder komplett neue Funktionen ergänzt.

Aus technischer Sicht ist die Rest-API ein HTTP Server, der auf POST Requests über eine URL der folgenden Form hört: http://unsereurl.de:8080/api/funktion

“funktion” wird beim Benutzen der API durch die in dem Moment benötigte Funktionalität (z.B. login) ersetzt. Als Post Parameter können dann benötigte Angaben wie username und password übergeben werden.

Die API ist hier implementiert:

https://github.com/pbat3h12b/Schnitzeljagd/blob/master/Datenbank/Cherrypy-Rest/main.py

https://github.com/pbat3h12b/Schnitzeljagd/blob/master/Datenbank/Cherrypy-Rest/apps/api/apiPageHandler.py

Unit Tests befinden sich hier:

https://github.com/pbat3h12b/Schnitzeljagd/blob/master/Datenbank/Cherrypy-Rest/unittests/apiTests.py

Session Secret und Token:

Auf der Suche nach einer möglichst trivialen und trotzdem sicheren Art, eingeloggte Nutzer zu authentifizieren, sind wir auf folgendes Schema gekommen:

- Beim Login generiert der Server einen zufälligen String und teilt diesem dem Client mit. Dieser String wird nachfolgend Session Secret genannt.

Der Server und der Client merken sich diesen String und einen “Command Counter”, welcher nach dem Login immer 0 ist.

- Einige Funktionen, wie z.B. das Updaten der eigenen Position setzen eine Authentifizierung voraus. Diese Funktionen bekommen ein “Token” zur Authentifizierung mitgeschickt.

- Ein Token wird so gebildet:

1) input = session_secret + command_counter
2) Das Token ist die md5 checksum von input in Hex-Notation.

Nach jedem neuen verwendeten Token muss der command_counter um eins erhöht werden.

Generell gilt für die API:
- Erfolgreiche Antworten sollten immer ein JSON String sein. Sollte das nicht der Fall sein, handelt es sich um einen Server-Fehler. Für den Zeitraum der Entwicklung gibt der Server Fehler in Form eines vollständigen tracebacks zurück.

- Antworten sind, falls sie keine Fehlermeldungen sind immer ein Dictionary, in denen es mindestens das Feld "success" gibt. "success" ist bool.

- Ist success False, gibt es immer auch ein Feld "error" in dem eine Fehlermeldung steht.

- Ist success True, kann die Nachricht andere, nachfolgend genauer spezifizierte Felder haben.

Ein erfolgreicher Login könnte z.B. folgende Antwort Produzieren:

```
{     'success': True,     'session_secret': 'AB345FDE3242DEF ' }
```

Eine fehlgeschlagene Registrierung wiederum eine solche:

```
{     'success': False,     'error ': 'Username already taken.' }
```

Implementierte Funktionen:

register:
Bekommt username, password
Antwort bei "success" == True:
hat keine zusätzlichen Felder.
Beispiel Antwort:

```
{
    "success": true
}
```

login:
Bekommt username, password
Antwort bei "success" == True:
"session_secret": Dieses Feld stellt zusammen mit einem Befehlzähler eine Session dar.
Beispiel Antwort:

```
{
    "session_secret": "-9\"?\"PZl?C>QB:remCIh",
    "success": true
}
```

getUsers:
Bekommt keine Parameter
Antwort bei "success" == True:
"users": Eine Liste aller User.
Beispiel Antwort:

```
{
    "users": ["Falk", "AxellStoll", "Kevin"],
    "success": true
}
```

getPositionsMap:
Bekommt keine Parameter
Antwort bei "success" == True:
"user_map": Dictionary mit mapping von Usern zu ihren letzten Laengen- und Breitengraden
Beispiel Antwort:

```
{
    "success": true,
    "user_map": {
        "Falk": [61.227301, 82.21948],
        "Kevin": [12.345678, 98.765432]
    }
}
```

getUserPath:
Bekommt username
Antwort bei "success" == True:
"positions": Liste mit Positionen der letzten 20 Minuten von username.
Beispiel Antwort:

```
{
    "success": true,
    "positions": [
        [2.3453456, 43.234523],
        [2.345347, 43.23458],
        [2.3553456, 43.234623]
    ]
}
```

getTopTenScoresForAllMinigames:
Optionales Attribut username
Antwort bei "success" == True:
"game": Ist ein Dictionary mit jedem Cache und deren top zehn Scores. Wenn ein username uebergeben wurde, werden die top zehn Spiele des Users zurueck gegeben,
Beispiel Antwort:

```
{
    "game": {
        "Zukunftsmeile": [],
        "HNF": [],
        "Fluss": [],
        "Serverraum": [],
        "Wohnheim": [{
            "username": "Falk",
            "date": 1409238709,
            "points": 851882
        }],
        "bib-Eingang": [{
                "username": "Kevin",
                "date": 1409238609,
                "points": 644884
            }, {
                "username": "Falk",
                "date": 1409238609,
                "points": 504163
            }
        ]
    },
    "success": true
}
```

getAllLogbookEntriesByUser:
Bekommt username
Antwort bei "success" == True:
"entries": Ist eine Liste mit Dictionaries. Jedes Dictionary stellt ein Logbucheintrag dar.
Beispiel Antwort:

```
{
    "success": true,
    "entries": [{
        "user": "Kevin",
        "message": "Physik",
        "cache": "bib-Eingang",
        "puzzle_solved": true,
        "found_date": 1409238607
    }, {
        "user": "Kevin",
        "message": "Mathematik",
        "cache": "Zukunftsmeile",
        "puzzle_solved": true,
        "found_date": 1409238607
    }, {
        "user": "Kevin",
        "message": "Philosophy",
        "cache": "HNF",
        "puzzle_solved": false,
        "found_date": 1409238608
    }]
}
```

secretValidForNextCache:
Bekommt username, cache_secret
Antwort bei "success" == True:
Beispiel Antwort:

```
{
    "success": true
}
```

makeGuestbookEntry:
Bekommt author, message_str
Antwort bei "success" == True:
Beispiel Antwort:

```
{
    "success": true
}
```

getGuestbookIndex:
Bekommt keine Parameter
Antwort bei "success" == True:
"index": Liste mit allen abrufbaren Guestbook Eintraegen.
Beispiel Antwort:

```
{
    "index": [1, 3, 4, 5],
    "success": true
}
```

getGuestbookEntryById:
Bekommt id
Antwort bei "success" == True:
Die Antwort haellt den Inhalt einer Nachricht.
Beispiel Antwort:

```
{
    "date": 1409238609,
    "message": "Magie = Physik / Wollen",
    "id": 1,
    "success": true,
    "author": "Axel Stoll"
}
```

nop:
Bekommt: username und token
Antwort bei "success" == True: hat keine zusätzlichen Felder.
(Diese Funktion macht nichts, außer das Token zu überprüfen und den Befehlszähler um eins zu erhöhen, wie es später nützliche Kommandos machen würden. Nop ist eine Test funktion)
Beispiel Antwort:

```
{
    "success": true
}
```

updatePosition:
Bekommt: username, token, longitude, latitude.
Antwort bei "success" == True: hat keine zusätzlichen Felder.
Beispiel Antwort:

```
{
    "success": true
}
```

makeLogbookEntry:
Bekommt username, token, secret, message_str
Antwort bei "success" == True: hat keine zusätzlichen Felder.
Beispiel Antwort:

```
{
    "success": true
}
```

markPuzzleSolved:
Bekommt username, token
Antwort bei "success" == True: hat keine zusätzlichen Felder.
Beispiel Antwort:

```
{
    "success": true
}
```

submitGameScore:
Bekommt username, token, points, cache
Antwort bei "success" == True: hat keine zusätzlichen Felder.
Beispiel Antwort:

```
{
    "success": true
}
```

2.1.2 Funktionsfähige C# Wrapper Klasse

Die C# Wrapper Klasse stellt ein Verbindungsstück zwischen den C# Programmen und der Rest-Api da. Wir konnten im Zeitraum des ersten Sprints bereits die ersten Erfolge sehen, da die Klasse auf den Mobilen Geräten ebenfalls funktionierte.

Der Vorteil dieser Klasse ist dass die Entwickler der anderen Programme, durch einen einfachen Objektaufruf ihre Informationen bekommen bzw. übermitteln können, ohne die Kommunikation über die REST-Api direkt programmieren zu müssen.

z.B : “restapiconnection.Register(“Username”,”Password”);”

In den meisten Fällen gibt die Klasse einen bool wert zurück, der wiedergibt, ob der Funktionsaufruf erfolgreich war.

Im Falle einer erfolgreichen Abfrage, gibt er selbstverständlich die gewünschten Daten zurück.

Funktionen

-RegisterNewUser(String username, String password):Boolean

Diese Funktion bekommt den Benutzernamen und dass Passwort übergeben, um einen neuen Benutzer zu registrieren.

Die Parameter werden anschließend so bearbeitet das sie als “POST”-Parameter dem Http-Request mit gegeben werden können.

Anschließend wird Der Request gemacht und der Response ausgewertet. Wenn das Feld “success” ein true wert hat dann wird auch true zurück und bei einem false wert, false zurück gegeben.

-LoginUser(String username, String password) :Boolean

Wie schon die vorangehende Funktion bekommt diese Funktion ebenfalls die Parameter “username” und “password”, die anschließend als HTTP Parameter angefügt werden. Danach wird der Request gemacht.

Wenn der Request erfolgreich war, enthält der response ebenfalls dass “session_secret” welches in der Klasse gespeichert wird, genauso wie “username” und ”password”. Ist der Request fehlerhaft, wird ein false als Boolwert zurückgegeben.

-UpdatePosition(float longitude,float latitude) :Boolean

UpdatePosition stellt die erste Funktion da, die nur nach dem Login benutzt werden kann, daher braucht sie nicht mehr “username” und “password” als Parameter. Dafür braucht die Funktion die Längen- und Breitengrade. Diese werden dann mit den anderen Parametern angefügt und anschließend per HTTP Request ausgewertet. Wenn die Kommunikation erfolgreich war wird ein Boolwert mit true zurückgegeben.

-Communication(String url, byte[] postdata)

Diese Funktion verwaltet die Verbindung zum Server Hierbei werden die übergebenen Posdata einem WebRequest angefügt und anschließend der erhaltene Response als String zurückgegeben.

-GetPostDatafromString(List<Parameter>parameter)

Diese Funktion macht aus einer Liste von Parametern einen bytearray, der dem Request angefügt werden kann.

-GenerateToken()

Diese Funktion dient dazu ein Token zu generieren. Dies wird erreicht, indem das session_secret mit einem counter verbunden wird und anschließend ein “Hash” aus diesem String erstellt wird, der anschließend zu einem String umgeformt wird und als Token weiter verwendet werden kann.

-MakeLogbookEntry(string cache, string messsage)

Die Funktion MakeLogbookentries wird dazu benutzt einen neuen Logbucheintrag zu verfassen. Damit man der Eintrag auch zum richtigen Cache gemacht wird, wird dieser mit übergeben. Als zweiter Parameter ist die Message die gesetzt werden kann.

-GetPositionsMap()

Mit der Funktion GetPositionsMap hat der Benutzer der Api die Möglichkeit eine Liste an Positionen die in den Letzten 5min aktualisiert wurden, abzufragen und dann zu benutzen der Rückgabe zyp ist eine Liste des Objekts Position die in der Klasse mit enthalten ist.

-GetTopTenScoreForAllMinigames()

Mit der Funktion GetTopTenScoreForAllMinigames bekommt der Benutzer eine Liste die Top-Ten Scores für alle Minispiele zurück. Der Rückgabetyp GamescoreListResponse ist eine eigene Klasse die neben den Spielen auch das Success-Feld eines normalen Response hat.

-SubmitScore(int score, int gameID)

Die Funktion SubmitGameScore gekommt den Score und die SpieleID übergeben. Diese werden dann benutzt um einen neuen Score zu übermitteln.

-MarkPuzzleSolved()

Die Funktion MarkPuzzleSolved wird dazu benutzt ein Spiel als gelöst zu markieren. Der Server erkennt dabei selber welches Puzzle er dafür markieren muss.

-GetAllLogbookEntries()

Die Funktion GetAllLogbookEntries gibt alle Logbuch Einträge eines Benutzers zurück damit man beim erneuten einloggen die richtigen Caches angezeigt werden. Der Rückgabetyp ist eine Listen an LogbookEntries welches ein Objekt ist das ebenfalls in der RESTAPI-Klasse enthalten

-checkCacheSecret(string userinput)

Die Funktion checkCacheSecret überprüft ob dasübergebene Secret auch valide ist. Anschließend wird dann ein Standard Response zurückgegeben

-getTopScoreByUser(string gameID)

Die Funktion getTopScoreByUser gibt einen Score zurück der von dem jeweiligen Nutzer bei dem Spiel mit der id, gameID. Das Objekt Score ist wie viele andere in der Klasse enthalten.

2.1.3 Funktionsfähige PHP Wrapper Klasse

Die PHP-Wrapper Klasse dient ausschließlich der Webseite. Sie stellt das Bindeglied zwischen RestAPI
und Webseite. Wie auch bei der C# Wrapper Klasse ist ein Objekt zu erstellen. Dieses bietet dann alle
nötigen Funktionen.

```
$wrapper = new apiWrapper;
$wrapper -> register("username","password");
```

Bisher implementierte Methoden

-register("username","password")
Registriert einen neuen User. Die Methode bekommt einen Username und das Passwort.
Als Antwort kommt "true" oder eine Fehlermeldung.

-login("username","password")
Meldet einen User an. Diese Methode liefert das session-secret.

-nop("username")
Eine Testmethode um das Authentifications Verfahren zu prüfen.

-tokenGen()
Generiert aus dem Command Counter und dem Session-Secret einen Token.

-postData(data[],"url")
Diese Funktion verwaltet die Verbindung zum Server hierbei werden die übergebenen Posdata einem WebRequest angefügt und anschließend der erhaltene Response als String zurück gegeben.

2.1.4 Unit Tests

Unit-tests der API befinden sich hier:
https://github.com/pbat3h12b/Schnitzeljagd/blob/master/Datenbank/Cherrypy-Rest/unittests/apiTests.py

Die Unittest decken eine große Anzahl von möglichen Fehlern ab und können als Nutzungs Beispiel für die API benutzt werden.

Die Unit Tests der C#-/php- Klassen sind leider aus Zeitgründen wie auch aus fehlenden Wissen im Bereich der Unit Test in Unity noch nicht angefertigt.

3 Verzeichnisse

3.1 Glossar

- JSON →**J**ava **S**cript **O**bject **N**otation
- Traceback →Ein Traceback ist die Auflistung eines Befehlsstacks. Man gibt den Stack mit dem Ziel aus, die Stelle an dem ein Programm einen Fehler auslöst zu finden.
- Wrapper

3.2 Abbildungsverzeichnis