|  |
| --- |
|  |
| SimService |
| Node.JS Service zum Speichern der Daten des FlightSimulator |

|  |
| --- |
| Patrick Müller  11.7.2016 |

Inhaltsverzeichnis

[Zielsetzung 2](#_Toc455974712)

[Sensoren 2](#_Toc455974713)

[Accelerometer 2](#_Toc455974714)

[Gyrometer 2](#_Toc455974715)

[Inclinometer 2](#_Toc455974716)

[OrientationSensor 2](#_Toc455974717)

[Fazit 2](#_Toc455974718)

[Sensoren Tests 3](#_Toc455974719)

[Accelerometer 3](#_Toc455974720)

[Inclinometer 3](#_Toc455974721)

[OrientationSensor 3](#_Toc455974722)

[Fazit 3](#_Toc455974723)

[Applikation 4](#_Toc455974724)

[Allgemeines 4](#_Toc455974725)

[Hauptmenü 4](#_Toc455974726)

[Im Flug 5](#_Toc455974727)

[Flughafen 6](#_Toc455974728)

[Einstellungen 7](#_Toc455974729)

[Sound 7](#_Toc455974730)

[DebugInfo 7](#_Toc455974731)

# Zielsetzung

Ziel dieses Projekts ist es, einen Webservice zu entwerfen und zu entwickeln, der in der Lage ist, die Spielstände der FlightSimulator-App online zu speichern und von dort jederzeit abrufbar zu machen.

Des Weiteren muss die FlightSimulator-App um die nötigen Funktionen erweitert werden, sodass eine Verbindung mit dem Webservice hergestellt werden kann, der Nutzer bzw. das Gerät sich identifiziert, und so individuelle Spielstände speichern und laden kann.

# Art des Webservice – SOAP oder REST

## REST

Die wichtigsten Merkmale von **REST** (**Re***presentational*-**S***tate*-**T***ransfer*) sind

* Performance
* Skalierbarkeit
* Einfachheit der Schnittstellen
* Einfache Modifizierbarkeit
* Verlässlichkeit

In Anbetracht des relativ kleinen Umfangs der geplanten Applikation sind Performance und Skalierbarkeit nicht die oberste Priorität. Einfache Modifizierbarkeit hingegen ist ein sehr starker Pluspunkt, da sich die Features der FlightSimulator-App ständig ändern können oder gar neue hinzukommen. Somit ist es auch sehr wichtig, dass eine simple Schnittstelle vorhanden ist, die bei Änderung der Anforderungen keine große Anpassung erfordert. Dass der Service verlässlich sein muss steht außer Frage, da ohne die gespeicherten Spielstände ein Nutzer immer von vorne beginnen müsste, was nicht zumutbar ist.

**REST** ist über eine Vielzahl von Protokollen nutzbar, z.B. HTTP/HTTPS, FTP, OData. Das Format der Daten ist den Anforderungen anpassbar, so lassen sich die Daten in XML, JSON oder sogar Plaintext-Format übermitteln. Meist werden bei einer **REST-**API die vier CRUD-Methoden Create, Refresh, Update und Delete implementiert, z.B. beim HTTP-Protokoll in Form von POST, GET, PUT, DELETE.

## SOAP

**SOAP** (**S***imple*-**O***bject*-**A***ccess*-**P***rotocol*) zeichnet sich durch folgende Merkmale aus:

* Sehr hohe Sicherheit (WS-Security, WS-ReliableMessaging)
* Sprach-, Plattform- und Transportprotokollunabhängig
* Standardisiert
* Schnittstelle liefert Beschreibung der Daten

Das hohe Maß an Sicherheit ist bei sensiblen Daten wie z.B. Bankdaten ein sehr wichtiger Faktor. In der FlightSimulator-App werden jedoch keine sensiblen Daten von Nutzern verarbeitet oder gespeichert. Die Unabhängigkeit des Protokolls von der Programmiersprache, Plattform und dem Transportprotokoll ist sehr nützlich, wenn eine Vielzahl verschiedener Clients auf die Schnittstellen des WebService zugreift. Da das **SOAP-**Protokoll standardisiert ist und die Schnittstellen auch Informationen über die von ihnen gelieferten Daten zur Verfügung stellen, wie z.B. Datentypen, Namen der Objekte und Eigenschaften, ist es sehr einfach den Großteil der Arbeit nur einmal im Webservice zu erledigen, statt es in jedem einzelnen Client implementieren zu müssen.

Auch **SOAP** ist über eine Vielzahl von Protokollen nutzbar, das Format der Daten ist allerdings auf XML festgelegt. Zu jeder Schnittstelle muss auch eine standardisierte Beschreibung vorliegen, in der die Daten und verfügbare Operationen aufgeführt sind

## Fazit

Nach abwägen der Vor- und Nachteile der zwei Möglichkeiten, habe ich mich dazu entschieden, den Webservice als eine **REST-**API zu implementieren.

Durch ständige Veränderung in der FlightSimulator-App, ist es erheblich einfacher mit den Schnittstellen einer **REST-**API zu arbeiten, da diese leicht zu modifizieren und erweitern sind. Bei **SOAP** hingegen müsste bei jeder Änderung der Schnittstelle auch die dazugehörigen Beschreibungen geändert werden.

Bezüglich der Sicherheit des Webservice ist die SSL-Verschlüsselung einer HTTPS-Verbindung ausreichend, die erweiterten Features von SOAP sind überflüssig.

Für den Zweck ein Speichersystem für Spielstände zu implementieren sind die HTTP-Schlüsselmethoden

* POST (Speichern eines neuen Spielstandes)
* PUT (Vorhandenen Spielstand überschreiben)
* GET (Laden eines vorhandenen Spielstandes)
* DELETE (Löschen eines vorhandenen Spielstandes)

hervorragend geeignet, und diese lassen sich mit einer **REST-**API sehr leicht abbilden.

# Technologien

Als Technologie wird für diesen Webservice Node.js, express.js und SQLite genutzt.

Node.js ist die zugrundeliegende JavaScript-Engine auf welcher der Code ausgeführt werden kann.

Express.js ist ein Web-Applikation Framework mit dessen Hilfe die Routen der Webservice-API erstellt werden.

Zum Speichern der Spielstände wird SQLite als Datenbank genutzt.