Swift On Linux/Haupteil/Programmieren anhand eines Beispiels/

Aus SwiftWiki

Navigation

Inhaltsverzeichnis

Server installieren

Der Server wird auf Github ausgecheckt und compiliert. Dazu folgenden Link Github/Projekt mit Git ausführen oder als zip herunterladen. In diesem Repository befinden sich der Ordner NodeServer in dem der Reverenz-Server in JavaScript mit Node und Express.js geschrieben ist. Im Ordner SwiftServer befindet sich der in Swifft geschriebene Server, der für diese Installation relevant ist. Damit der Server läuft ist der Node-Server irrelevant, sodass dieser gelöscht werden kann oder ausgeführt werden kann wie im Kapitel "Installation Node Server" beschrieben ist.

Installation mit Git

Folgende Befehle sind auszuführen:

```
mkdir SwiftServer
cd SwiftServer
git clone https://github.com/hofchris15/Projektarbeit.git
swift build
```

oder

```
swift build --configuration release
```

Mit dem ersten Befehl wird in einem beliebigen Verzeichnis der Ordner SwiftServer erstellt, der als Projektordner dient. Danach wechselt man in das Verzeichnis und checkt den Server von Github aus. Danach liegt der Server in Form von Source Code im Verzeichnis "SwiftServer" auf und kann mit dem letzten Befehl compiliert werden. Mit diesem Kommando wird ein debug compilierung durchgeführt, das für die Entwicklung benötigt wird, da mehr Informationen während der Laufzeit an den Entwickler ausgegeben werden. Um den Server jemanden außerhalb des Entwicklungsbereichs zu übergeben wird empfohlen den compiler auf den release-Modus zu stellen. Dazu wird der Befehl `swift build --configration release` ausgeführt. Je nach compiler Konfiguration wird das Produkt im Verzeichnis .build/debug/exe abgelegt oder .build/release/exe abgelegt. Damit ist die Installation des Servers abgeschlossen und der Server kann gestartet werden.

Installation ohne Git

Der Ablauf ohne Git ist geringfühgig anders als bei der Installation mit Git. Dazu wird der Server als .zip-Archiv von Github heruntergeladen und in das zuvor erstellt Verzeichnis "SwiftServer" entpackt.

```
mkdir SwiftServer
cd SwiftServer
sudo tar -xcf /path/to/SwiftServer.zip ./
swift build
```

oder

```
swift build --configuration release
```

Danach ist die Installation auch ohne Git abgeschlossen und der Server kann gestartet werden.

Server starten

Der Server kann über ein Terminal gestartet werden nachdem dieser compiliert wurde siehe auch: Installation von Swift#Server installieren Danach kann der Server mit folgende Kommandos gestartet werden. Einzige Vorraussetzung ist, das man sich bereits mit dem Terminal im Projektordner befindet.

```
./.build/debug/exe <code>

oder mit

<code>./.build/release/exe
```

Danach sollte folgend Ausgabe auf dem Terminal angezeigt werden:

```
[DBG] init HTTPServer()
[DBG] makeRoutes()
[INFO] set config
[INFO] Starting HTTP server on 0.0.0.0:3000
```

Mit der letzen Zeile wird darüber informiert, dass der Server gestartet wurde und auf dem Port 3000 hört. Somit kann dieser mit einem bliebigen Browser angesteuert werden. Dazu wird in die Adressleiste localhost:3000 eingegeben und es sollte die Login-Seite des Services angezeigt werden.

Weiter mit einigen Details zum Server und den Packages die Verwendet werden.

Details zum Server

Der Server entstand im zusammenhang mit einer Projektarbeit mit dem Thema Swift on Linux. Dazu wurde von uns rechachiert und zwei interessante Frameworks für die Entwicklung eines Servers ausgewählt. Diese beiden Frameworks sind Kitura.io und Perfect.org . Grundsätzlich sind beide Frameworks sehr ähnlich, wobei hierbei einige Punkte einzubringen und beachten sind. Kitura ist ein Framework das von IBM erstellt wurde und auf MAC sowie auf Linux lauffähig ist. Perfect wurde von PerfectlySoft Inc. einem kleinem Startup Unternehmen, das sich auf die Entwicklung mit Swift spezialisiert hat. IBM's Kitura hat einen Grundsätzliche Bibliothek die verwendet wird um einen HTTP Server umzusetzen und einige Packages um die Funktionalität zu erweitern. Perfect setzt auf das selbe Prinzip wie Kitura und hält wie auch Kitura ein Template als Startpunkt für den Server zur Verfügung. Einziges manko bei Kitura ist auf den ersten Blick, dass die Dokumentation nicht so ausführlich erscheint wie bei Perfect. Schlussendlich haben wir uns für swift Perfect entschieden. Grund dafür waren der eben erwähnte Punkt der Dokumentation, die von Perfect sehr ausführlich ist und auch einige Beispiele für verschiedenen Bereiche eines Servers bereithält. Die Beispiel und viele Bibliotheken und Packages werden von PerfectlySoft auf deren Github Account gehostet: PerfectlySoft on Github

Installation von Perfect

Auch Perfect setzt einige Packet auf Linux voraus, darunter openssl, libssl-dev und uuid-dev. Diese können sehr leicht mit

```
sudo apt-get install openssl libssl-dev uuid-dev
```

installiert werden.

Perfect Template

Das Perfect Template wurde von uns als Ausgangspunkt der Entwicklung des HTTP Servers im MVC-Pattern gewählt. Als Reverence wurde von uns der selbe Server zuvor mit Node.js und dem Framework Express.js implementiert. Das Perfect Template hält die Standard Baumstruktur des swift init-Prozesses bereit, die bereits in der Installation von Swift#Swift Package Manager vorgestellt wurde. Weiter befinden sich ein Datei names "main.swift" enthält, die die Initalisierung eines Serves bereithält. Interressant ist jedoch das in der Package.swift enhaltene Abhängigkeit [1]. Der Perfect-HTTPServer hat einige sehr nützliche Abhängigkeiten die auch wir im Server verwendet haben.

Die wichtigsten Packages

Wie oben beschrieben sind einige Packete grunsätlich notwendig um den Server ins laufen zu bekommen. Diese Grund-Pakete sind PerfectLib COpenSSL, HTTP, HTTPServer, PerfectLib, LinuxBridge und Net.

- Foundation ist die standard Bibliothek von Swift und bietet die grunsätlichen Funktionen und Definition wie (Strings, Numbers,...) die von der Swift.org veröffentlicht wurden. https://developer.apple.com/documentation/foundation
- COpenSSL bietet verschieden Verschlüsselungstechniken und SSL/TLS Methoden zur Verfügung die auf Linux in C installiert werden (Dieses Paket wurde nicht verwendet) https://github.com /PerfectlySoft/Perfect-COpenSSL
- HTTP bietet verschiedene Strukturen und Methoden um mit http Clients zu kommunizieren. Diese Paket muss mit "import PerfectHTTP" im Projekt eingebunden werden https://github.com /PerfectlySoft/Perfect-HTTP
- HTTPServer bietet die Hauptstrukturen für HTTP 1.1 und HTTP 2 server und ist die Hauptabhängigkeit für das Projekt und muss mit "import PerfectHTTPServer" eingebunden werden

https://github.com/PerfectlySoft/Perfect-HTTPServer
- PerfectLib ist das Herz des Perfect Frameworks und ist ohne nicht lauffähig. Muss ebenfalls mit "import PerfectLib" eingebunden werden. https://github.com/PerfectlySoft/Perfect
- LinuxBridge bildet die Brücke zwischen Perfect und den Linux Distriputionen https://github.com /PerfectlySoft/Perfect-LinuxBridge
- Net ist ein networking Paket welches TCP, SSL, UNIX Socket files und IO Event Handling zur Verfügung stellt. https://github.com/PerfectlySoft/Perfect-Net

zusätzliche Packages

- Perfect-Logger und Perfect-RequestLogger sind Pakete die zum Loggen von Informationen auf die Konsole oder in ein Log File. der RequestLogger ist eine Kindklasse vom Logger und fängt durch das einbinden von Filtern alle Request und logged diese mit. https://github.com/PerfectlySoft/Perfect-RequestLogger
- Perfect-Session wurde verwendet um einen Session zu starten, verwalten und wieder zu vernichten. https://github.com/PerfectlySoft/Perfect-Session
- Perfect-Websockets bietet eine Funktionen zum Aufbauen von WebSocket-Verbindungen
- SwiftyBeaver bietet verschiede Verschlüsselungstechniken für Passwörter oder anderen empfindlichen Daten https://medium.com/swiftybeaver-blog/logging-in-server-side-swift-85bdecb6be80

Wichtige Punkte der Implementierung

Linux

Das Projekt soll nur auf Linux laufen deshalb wurde am Anfang folgender Code eingebunden, welcher das Betriebssystem überprüft und beendet sollte es kein Linux sein:

```
#if !os(Linux)
import Glibc
print("We are sorry this is only meant to be run on Linux")
exit(1)
#endif
```

Session Verwaltung

Bei der Session sind verschieden Konfigurationen möglich. Zu beachten ist jedoch, dass die cookieDomain, die festgelegt werden kann, während der Entwicklung nicht festgelegt werden darf. Erst wenn der Server gehostet wird, kann die Domain angepasst werden, was jedoch nicht nötig ist, da mit das IPAddressLock und der userAgentLock eingeschalten sind.

```
SessionConfig.name = "mobileExtendSession" //Session name which is set as cookie
SessionConfig.idle = 86400 // idle time set to one day
// Optional
//SessionConfig.cookieDomain = "localhost"
SessionConfig.IPAddressLock = true //Session is bind to the IP address of the first request
SessionConfig.userAgentLock = true //Session is bind to the user
```

JSON

Objekte die als JSON serialisiert werden sollen, müssen von der Klasse JSONConvertibleObject erben, sowie die Methoden "setJSONValues" und "getJSONValues" müssen überschrieben bzw definiert werden.

Zusätlich zu den üblichen Attributen des Objektes kommt ein Registrierungsname zum Einsatz. Dieser dient als Schlüssel für die Objektklasse und als Registrierungsschlüssel im Register der JSON-Decodable-Object Referenz. Mit diesem Schlüssel kann die Decodierung eines JSON Strings erfolgen und wird dem Objekt, dass erstellt werden soll zugeteilt.

```
public class Profile: JSONConvertibleObject {
static let registerName = "profile"
   var username = ""
   var password = ""
....
....
override public func setJSONValues(_ values: [String: Any]) {
   self.username = getJSONValue(named: "username", from: values, defaultValue: "")
   self.password = getJSONValue(named: "password", from: values, defaultValue: "")
   self.firstname = getJSONValue(named: "firstname", from: values, defaultValue: "")
....
....
override public func getJSONValues() -> [String: Any] {
return [
   JSONDecoding.objectIdentifierKey: Profile.registerName,
   "username": username,
   "password": password,
   ...
   ]
....
....
```

Wichtig ist auch das die Objekt beim Start des Servers als JSONDecodable registriert werden müssen:

```
JSONDecoding.registerJSONDecodable(name: Profile.registerName, creator: {return
Profile()})
```

working Directory

Beim setzen des Working Directory ist zu beachten, dass solange der Server nicht neu gestartet wird, nicht mehr im Verzeichnisbaum nach oben gändert werden kann, nur nach unten. Deshalb sollte das Working Direktory vom Standardwert Sources, nur maximal in ein Unterverzeichnis, wie bei unserem Server, dem Verzeichnis "exe", geändert werden. Beim Setzen des Working Directory wird der Ausführungsprozess in ein darunterliegendes Verzeichnis verschoben.

```
func setupDir(_: Void) -> Void {
   let workingDir = Dir("./Sources/exe")
   if workingDir.exists {
      do {
         try workingDir.setAsWorkingDir()
         LogFile.debug("Working directory set to \(workingDir.name)")
      } catch {
         LogFile.debug("error in getFile() setting WorkingDir: \(error)")
         }
   } else {
      LogFile.error("Directory \(workingDir.path) does not exist. Main executable not started
from root of MVC cannot find resources?")
      exit(2)
```

```
    }
    issetup = true
}
var issetup = false
```

Zurück zur Startseite des Buches

Abgerufen von „http://192.168.42.2/mediawiki/index.php?title=Swift_On_Linux/Haupteil/Programmieren_anhand_eines_Beispiels/&oldid=70“
