|  |
| --- |
|  |

|  |  |
| --- | --- |
| Departement  Finanzen und Ressourcen  Informatik Aargau |  |
|  | |
| TECHNISCHE DOKUMENTATION | |
| Einsatz von tsNodeTemplate | |

|  |  |  |
| --- | --- | --- |
| **Zweck des Dokuments**  Das Dokument beschreibt den Einsatz und die weitere Konfiguration des Vorlage Programms «tsNodeTemplate». | Dokumenttyp: | Dokumentation |
| Autor: | Marc Berchtold |
| Version: | 1 |
| Gültig ab: | 12.07.17 |

Änderungskontrolle

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Erstellt von | Bemerkungen |
| 1 | 12.07.17 | Marc Berchtold |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[1. Software Vorrausetzungen 3](#_Toc487611650)

[2. Starten der Vorlage 3](#_Toc487611651)

[3. Kompilierung der Vorlage 3](#_Toc487611652)

[4. Erklärung des Gulp Automatisierungssystem 3](#_Toc487611653)

[5. Erklärung der Quelldateien 4](#_Toc487611654)

[5.1 index.ts 4](#_Toc487611655)

[5.2 App.ts 4](#_Toc487611656)

[5.3 controllers/BaseController.ts 5](#_Toc487611657)

[5.4 controllers/UserController.ts 5](#_Toc487611658)

# Software Vorrausetzungen

Zur Ausführung der Vorlage werden die folgenden Programme benötigt:

* Node.js (v.6.11.0 LTS)[[1]](#footnote-1)

# Starten der Vorlage

Zum Starten der Vorlage muss diese und ihre Abhängigkeiten zuerst auf den lokalen Arbeitsplatz heruntergeladen werden.  
Hierzu empfiehlt es sich die neuste Version der Vorlage mittels Git in einen lokalen Ordner zu von der offiziellen Repo[[2]](#footnote-2) zu *Clonen*.

git clone http://<Benutzername>@git.bb-itag.ch/mb6t/tsNodeTemplate.git

Anschliessend kann mit einer Kommandozeile in das geclonte Verzeichnis gewechselt werden und die benötigten Bibliotheken mittels NPM installiert werden. Dieser Vorgang kann je nach Internetgeschwindigkeit einige Minute dauern.

cd tsNodeTemplate

npm install

Nach erfolgreicher Installation der Abhängigkeiten kann die zuletzt kompilierte Version mittels

npm start

gestartet werden.

# Kompilierung der Vorlage

Falls die Vorlage verändert wurde, muss sie neu kompiliert (*transpiled*) werden, damit die Änderungen lauffähig sind. Dabei überprüft der transpiler den Code auf allfällige Fehler und gibt diese in der Konsole aus. Auch die visuelle Qualität des Codes wird mithilfe von *tsLint* überprüft. Der Kompilierprozess kann mit folgenden Befehlen in einer Kommandozeile im Verzeichnis der Vorlage ausgeführt werden:

npm run build

Es besteht auch die Möglichkeit eventuelle Änderungen am Code fortlaufend zu kompilieren und die Node-Instanz neu zu starten, um so einen schnelleren Entwicklungs-Test Zyklus zu ermöglichen. Um die Quelldateien zu überwachen und sie bei Änderungen neu zu transpilen sollte der folgende Befehl in einer Kommandozeile im Verzeichnis der Vorlage ausgeführt werden:

npm run serve

# Erklärung des Gulp Automatisierungssystem

*Gulp[[3]](#footnote-3)* ermöglicht es Arbeitsschritte, welche für den Entwicklungsprozess benötigt werden zu automatisieren. Dabei wird Gulp über die Datei gulpfile.js im Projektverzeichnis konfiguriert. Die in der Konfigurationsdatei definierten *tasks* können entweder über die *gulp-cli*, welche über npm global installiert sein sollte, ausgeführt werden, oder, wenn sie in der *package.json* als *script* definiert sind, mittels *npm run* Kommandos.

Im Folgenden werden die in dieser Vorlage definierten tasks genannt und beschrieben:

* *default*
  + Überprüft alle im *src* Verzeichnis vorhandenen *.ts* Dateien auf die visuelle Code Qualität anhand der *tsLint* Regeln welche in der *tslint.json* Datei im Projektverzeichnis festgelegt werden.
  + Transpiled alle im *src* Verzeichnis vorhandenen *.ts* Dateien. Dabei werden die entstehenden *.js* Dateien im *dist* Verzeichnis abgelegt.
  + Startet eine Node Instanz, welche die Datei *dist/index.js* ausführt.
  + Überprüft alle Dateien im *src* Verzeichnis auf allfällige Änderungen bis der Gulp-Task vom Benutzer beendet wird. Falls eine Änderung erkannt wird, wird der Build und Code Überprüfungsprozess erneut durchgeführt und die Node Instanz neu gestartet.
* *serve*
  + Überprüft alle im *src* Verzeichnis vorhandenen *.ts* Dateien auf die visuelle Code Qualität anhand der *tsLint* Regeln welche in der *tslint.json* Datei im Projektverzeichnis festgelegt werden.
  + Transpiled alle im *src* Verzeichnis vorhandenen *.ts* Dateien. Dabei werden die entstehenden *.js* Dateien im *dist* Verzeichnis abgelegt.
  + Startet eine Node Instanz, welche die Datei *dist/index.js* ausführt.
* *build*
  + Überprüft alle im *src* Verzeichnis vorhandenen *.ts* Dateien auf die visuelle Code Qualität anhand der *tsLint* Regeln welche in der *tslint.json* Datei im Projektverzeichnis festgelegt werden.
  + Transpiled alle im *src* Verzeichnis vorhandenen *.ts* Dateien. Dabei werden die entstehenden *.js* Dateien im *dist* Verzeichnis abgelegt.
* *checkRepoKey*
  + Überprüft ob im Verzeichnis die Dateien *repo-key* und *repo-key.pub* vorhanden sind. Diese beiden Dateien stellen das SSH-Schlüsselpaar dar, welches benötigt wird um im Docker *build* Prozess allfällige private Abhängigkeiten aus Git-Repos in der *package.json* erfolgreich herunterladen zu können.

# Erklärung der Quelldateien

## index.ts

Die Datei *index.ts* stellt den Einstiegspunkt in die Applikation dar und ist deshalb auch die Datei welche mit Node gestartet wird (*index.js*). Hier wird auch die *express* Server-Konstante definiert, indem eine Instanz der Klasse *App* aus *App.ts* initialisiert und die Funktion *listen* aufgerufen wird.

Es sollte nicht nötig sein diese Datei zu verändern, da jegliche neuen Funktionen und/oder Modifikationen sich auf die Klasse *App* und ihre Unterklassen beschränken sollten. Eine Ausnahme hierfür stellt eventuelle Änderungen am Start des Webservers dar.

## App.ts

Die Datei *App.ts* beinhaltet die Klasse App, welche für die Initialisierung der *express* Library zuständig ist. Hier werden auch die Middlewares und Routen für den Webserver definiert.

Die Vorlage ist so strukturiert, dass jede Route erster Ordnung (Bsp. */login*) durch ihre eigene Klasse, welche einen Controller darstellt, definiert wird und in der Klasse *App* nur ihren Routen zugewiesen nicht jedoch werden.

## controllers/BaseController.ts

Die Datei *BaseController.ts* beinhaltet die Klasse *BaseController*, welche für die Definition der Route */* verantwortlich ist. Jegliche Endpunkte, welche bei */* enden (GET, POST, PUT, etc.) werden hier ihren Funktionen zugewiesen.

In der Vorlage ist hierfür als Beispiel der Endpunkt GET einer Funktion zugewiesen worden, welche ein JSON-Objekt mit dem Parameter *message* und dem Wert *Hello World* zurückgeben wird.

## controllers/UserController.ts

Die Datei *UserController.ts* beinhaltet die Klasse *UserController*, welche ein weiteres Beispiel für den Aufbau eines Controllers bieten soll. Der Aufbau ist hier nahezu identisch zu *BaseController*.

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

1. <https://nodejs.org> [↑](#footnote-ref-1)
2. <https://git.bb-itag.ch/mb6t/tsNodeTemplate> [↑](#footnote-ref-2)
3. <http://gulpjs.com/> [↑](#footnote-ref-3)