Heaven & HILLS

HANDOUT

TypeScript

Sven Erdem, s0553945@htw-berlin.de
Matthes Ortel, s0554616@htw-berlin.de
Patricia Steves, s0548381@htw-berlin.de
Rico Stucke, s0556273@htw-berlin.de
Hafidh Zuhdi, s0556960@htw-berlin.de

Entstehung

- ist ein Superset von JavaScript
- wurde 2012 von Microsoft veröffentlicht und wird immer noch weiterentwickelt
- inzwischen weitgehend in der Frontend-Entwicklung etabliert, da es JS einfacher macht

- Leicht von CoffeeScript inspiriert, daher findet man einige Elemente aus CoffeeScript auch in TypeScript wieder
- TypeScript gilt als hinsichtlich der Verständlichkeit als Verbesserung gegenüber JavaScript
- basierte auf dem ursprünglichen Entwurf für den damals neuen ES6-Standard, der zur Zeit der Entwicklung von TypeScript noch nicht (vollständig) implementiert war
- die Entwickler wollten objekt-orientierte Konzepte (Klassen, Module, Vererbung, anonyme Funktionen, Generics) schon früher einsetzen können, um vom prototypenbasierten JavaScript abweichen zu können

Grundkonzepte

- JavaScript ist immer gültiges TypeScript, also kann jede Bibliothek und jedes Snippet einfach in eine TypeScript-Datei eingefügt werden, ohne auf Probleme zu stoßen.
- TypeScript wird am Ende zu reinem JavaScript kompiliert, wodurch es in jedem Browser läuft. Je nach Compiler-Einstellungen auch im IE6
- Kompatibilität wird durch den Compiler so weit maximiert, dass ein guter Browsersupport ohne Polyfills ermöglicht wird

Verwendung von TypeScript

- Installation über NPM (package typescript)
- Entwicklung direkt in Visual Studio
- Alternativ gibt es Plugins für andere Texteditoren
- Zudem verfügbar für Grunt, Gradle und damit vielseitig einsetzbar

Generics

- in JS durch „any“ Typ realisiert
- führt zu Verlust von Informationen über Input- und Rückgabetyp
- in TS sind Generics vorhanden ähnlich wie C# oder Java
- Funktionsparameter müssen so benutzt werden als ob sie sämtliche Typen hätten
- generische Klassen mit Syntax „class foo<T>{ }“
- generische Klassen sind nur auf Instanzebene generisch
- Generics können eingeschränkt werden mit Interfaces

Namespaces

- werden genutzt zum Organisieren von Code
- ähnlich zu Paketen in Java
- Syntax: namespace Foo{ }
- Funktionen und Klassen werden mit „export“ Keyword sichtbar
- ein Namespace kann über mehrere Dateien genutzt werden

Warum TypeScript?

- ist ein JavaScript und Java überall verwendet.
- “static typing”, classes, enum, generics, interfaces,etc gibt Vorteil für IDE und IDE kann mit Fehlererkennung, Auto-Complete unterstützen
  => Arbeitsleistungsverbeserrung
- JavaScript wird durch ECMAScript-Standards standardisiert. TypeScript kann nach beliebig ECMAScript kompilieren => Bereit für die Zukunft
- Open Source, groß Community (seit 2012; 2017-10-31 stable releases)

Vorteil	Nachteil
• TypeScript ist JavaScript: Wer schon • JavaScript auswendig kennen, soll keine Angst davor haben. Was in JavaScript funktioniert auch in TypeScript • static typed: dadurch ist die Code einfacher und lesbar zu analysieren. • IDE-Unterstützung: Mit “types” kann die IDE die Arbeit erleichtern. Beispielweise, Auto-complete, früherer Fehlererkennung wie typos, Refactoring, usw. Wenn Zweifel hätte, einfach in JS kompilieren, und in JS weiterarbeiten.	• Arbeiten ist nicht so schnell wie in JavaScript. • muss man auf “types” achten, im Gegensatz dazu, JavaScript ist dynamisch und flexibel. • Transpiling kosten Zeit (ms-Bereich)

Wann TypeScript?

- Große Projekt (bis zu/mehr als 1. Mio Line Codes) hat große Risiken. TypeScript hilft, Zeit und Ressource zu sparen
- Libraries oder Frameworks die mit Typescript zusammen “spielen”. (Angular 2, JSX, etc)
- Team mit “strong typed” Programmiersprache (Java, C#)