

Festlegung der verwendeten Werkzeuge

Inhalt

1 Einleitung	2
2 Auflistung und Argumentation der verwendeten Technologien	3
3 Vergleich von Parserbibliotheken für CAD-Dateien	5

1 Einleitung

Nun, da die Anforderungen festgelegt sind, sind wir in der Lage Entscheidungen über die verwendeten Werkzeuge und Technologien zu treffen, welche bei dem neuen Werkzeug zur Gebäudeplanvisualisierung zum Einsatz kommen sollen. Im ersten Schritt wollen wir uns um eine Auflistung und Argumentation der verwendeten Technologien kümmern.

Nachdem zum Beispiel die verwendete Programmiersprache festgelegt ist, wird unter anderem eine Bibliothek für das Lesen von CAD-Dateien gesucht. Die verschiedenen Möglichkeiten werden gesammelt und anschließend verglichen, um die bestmöglichen Voraussetzungen für die weitere Entwicklung der Anwendung zu schaffen.

2 Auflistung und Argumentation der verwendeten Technologien

Im vorangegangenen Vergleich von CAD-Viewer Programmen haben wir zwei grobe Kategorien von Programmen betrachtet: *web-basierte* und *klassische Desktopanwendungen*. Beide Varianten können mit denselben Features aufwarten und unterscheiden sich somit lediglich in der Art der Bereitstellung: Während die web-basierten Anwendungen durch die Eingabe einer URL in der Adresszeile eines Browsers gestartet werden können, müssen die Desktopprogramme lokal auf dem System des Benutzers installiert und gestartet werden.

Anhand der nichtfunktionale Anforderung **NA1** (*Einfache und schnelle Bereitstellung/Installierbarkeit*) tendieren wir daher eher zu einer Webanwendung. Diese sind tendenziell schneller zu verwenden und leichter zu teilen. Außerdem stehen geeignete Browser praktisch jedem Endanwender zur Verfügung - selbst auf dem Smartphone.

Im Gegensatz dazu verleitet die nichtfunktionale Anforderung **NA4** (*Ruckelfreie Navigation im Gebäudeplan*) eher dazu, die Gegenseite der klassischen Desktopanwendungen zu befürworten. Webanwendungen liefern klassischerweise weniger Performance verglichen mit Desktopanwendungen. Grund dafür ist die Programmiersprache *JavaScript*, welche von den verschiedenen Browserimplementierungen mehr oder weniger schnell interpretiert wird. Allerdings gab es in den letzten Jahren deutliche Performanceverbesserungen (Rose & Hildebrand, 2015, S. 576). Beispielsweise gibt es speziell für das Rendering von Visualisierungen mittlerweile Unterstützung durch die vorhandene Grafikhardware des Benutzers, auf welche mit Hilfe der *WebGL* API zugegriffen werden kann (Rose & Hildebrand, 2015, S. 576).

Da mit der zu entwickelnden Anwendung möglichst auch äußerst komplexe und große Gebäudepläne dargestellt werden sollen, ist es sinnvoll ebenso auf der WebGL API aufzusetzen. Vorhandene JavaScript Bibliotheken wie *three.js* („three.js - JavaScript 3D library“, 2020) kapseln die WebGL API auf einer höheren Ebene, um diese schnell und einfach zu verwenden. Das ist für unser Projekt ebenfalls wichtig, da die nichtfunktionale Anforderung **NA2** eine einfache Wartbarkeit fordert. Die WebGL API ist an OpenGL ES 2.0 angelehnt und daher nur mit dem nötigen Hintergrundwissen zu verwenden („WebGL“, 2020). Die Bibliothek *three.js* scheint daher geeignet für das Projekt, um eine spätere gute Wartbarkeit sicherzustellen.

JavaScript ist eine dynamisch typisierte Programmiersprache. Alternativen

wie *TypeScript* von Microsoft hingegen versuchen eine statische Typisierung um JavaScript zu ermöglichen (Gao, Bird, & Barr, 2017, S. 758). Es wird oft argumentiert, dass eine statische Typisierung zu einem besseres Verständnis des Programmes, neben anderen Vorteilen, führt (Gao, Bird, & Barr, 2017, S. 758). Um die Wartbarkeit unseres Projekts weiter zu erhöhen, soll daher TypeScript als Programmiersprache verwendet werden.

Die Entwicklung einer Webanwendung kann durch die Verwendung eines geeigneten Frameworks vereinfacht werden. Wir haben uns für das Framework *Angular* („Angular“, 2020) entschieden, da dieses bereits alle benötigten Komponenten für eine moderne UI mitliefert („Angular Material“, 2020) und standardmäßig mit TypeScript zu verwenden ist.

Zusammenfassend soll das Werkzeug zur Gebäudeplanvisualisierung also als **Webanwendung** entstehen. Diese soll in der Programmiersprache **TypeScript** unter Verwendung des Frameworks **Angular** verfasst werden. Zusätzlich wird **three.js** zur einfacheren Verwendung der WebGL API verwendet, womit schlussendlich die Gebäudepläne visualisiert werden.

3 Vergleich von Parserbibliotheken für CAD-Dateien

Literaturverzeichnis

WebGL. (2020, Oktober). Abgerufen von MDN-Web-Dokumentation website:

https://developer.mozilla.org/de/docs/Web/API/WebGL_API

Rose, A. S., & Hildebrand, P. W. (2015). NGL Viewer: a web application for molecular visualization. *Nucleic Acids Research*, 43(W1), W576–W579.

<https://doi.org/10.1093/nar/gkv402>

Angular Material. (2020, Oktober). Abgerufen von Angular Material website: <https://material.angular.io>

Gao, Z., Bird, C., & Barr, E. T. (2017). To Type or Not to Type: Quantifying Detectable Bugs in JavaScript. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 758–769. <https://doi.org/10.1109/ICSE.2017.75>

three.js - JavaScript 3D library. (2020, Oktober). Abgerufen von <https://three.js.org>

Angular. (2020, Oktober). Abgerufen von <https://angular.io>