



ulm university universität  
**uulm**

**Fakultät für  
Ingenieurwissenschaften,  
Informatik und  
Psychologie**

Institut für Software-  
technik und Program-  
miersprachen

# Endabnahmedokument

Softwaregrundprojekt an der Universität Ulm

**Vorgelegt von:**

Gruppe 10

**Dozent:**

Florian Ege

**Betreuer:**

Stefanos Mytilineos

2018/2019

---

# Inhaltsverzeichnis

<b>1</b>	<b>Verwendete Technologien</b>	<b>3</b>
1.1	Server und KI . . . . .	3
1.1.1	Buildtools und Frameworks . . . . .	3
1.1.2	Statische Analyse und Tests . . . . .	3
1.1.3	Dokumenation und Styleguide . . . . .	3
1.1.4	Externe Librarys . . . . .	3
1.1.5	Tools . . . . .	4
1.2	Client . . . . .	4
1.2.1	Framework und Dependencies . . . . .	4
1.2.2	Dev-Dependencies . . . . .	5
1.2.3	Tools . . . . .	6
<b>2</b>	<b>Projekttagbuch</b>	<b>6</b>
<b>3</b>	<b>UML - Komponentendiagramm</b>	<b>9</b>
3.1	Komponentendiagramm zum Server . . . . .	9
3.2	Komponentendiagramm zur KI . . . . .	11
3.3	Komponentendiagramm zum Benutzer-Client . . . . .	12
3.4	Komponentendiagramm zum Level-Editor . . . . .	13

---

# 1 Verwendete Technologien

## 1.1 Server und KI

### 1.1.1 Buildtools und Frameworks

Der Server und die KI wurden in C++17 geschrieben. Als Compiler wurde dafür der GNU-C++-Compiler (GCC) (Version 8.3, Lizenziert unter GPLv3) verwendet, als Buildsystem wurde CMake (Version 3.10, Lizenziert unter BSD-3) in Kombination mit GNU Make (Version 4.1, Lizenziert unter GPLv3) genutzt.

### 1.1.2 Statische Analyse und Tests

Zur statischen Codeanalyse wurde Clang-Tidy (Version 6.0, Lizenziert unter Apache 2) verwendet, als weiteres Tool zur Codeanalyse wurde der Sonar-Scanner (Version 3.3.0.1492, proprietär) in Verbindung mit der Community-Edition des Webinterfaces SonarCloud (<https://sonarcloud.io>) genutzt.

Unit- und Mock-Tests wurden mit Google-Test bzw. Google-Mock (Version 2.56, Lizenziert unter BSD-3) implementiert. Um undefiniertes Verhalten zu vermeiden, wurde das Programm während der Ausführung mit Adress Sanitizer (Version 5.0, Lizenziert unter Apache 2) überprüft.

### 1.1.3 Dokumentation und Styleguide

Alle Teile des Quellcodes wurden mithilfe von Doxygen-Kommentaren (Version 1.8, Lizenziert unter GPLv2) dokumentiert.

Es wurde sich während des Entwicklungsprozess an den Linux-Kernel-Styleguide gehalten. Zwei wesentliche Abweichungen von diesem Styleguide sind eine Einrückungstiefe von 4 Spaces, sowie die Platzierung der öffnenden Klammer bei Funktionsaufrufen in der selben Zeile (die Unterscheidung bei der Platzierung von öffnenden geschweiften Klammern ist wohl ursprünglich K&R zuzuschreiben, die diese Inkonsistenz damit begründen, dass Funktionen sich nicht verschachteln lassen, das ist mit C++11 durch Lambdas nicht mehr der Fall).

### 1.1.4 Externe Librarys

Als externe Librarys wurde libwebsockets (Version 3.1.0, LGPLv2, <https://github.com/warmcat/libwebsockets>), nlohmann::json (Version 3.6.0 bzw. 3.6.1, MIT, <https://github.com>).

---

com/nlohmann/json) sowie MLP (GPLv3, <https://github.com/aul12/MLP>) verwendet.

### 1.1.5 Tools

Als Entwicklungsumgebung wurden die IDE CLion (Version 2019.1, proprietär) sowie der Editor vim (Version 8.0, Lizenziert als Careware).

## 1.2 Client

### 1.2.1 Framework und Dependencies

**Vue.js** Als Frontend-Framework wurde Vue.js verwendet, das es u. a. erlaubt, umfangreiche SPAs zu entwickeln (Version 2.6.10, <https://vuejs.org/>, mit vue-template-compiler: 2.5.21).

- The MIT License (MIT)

**Fontawesome** Fontawesome bietet eine umfangreiche Sammlung kostenloser SVG-Grafiken zur Gestaltung von UIs, die sich einfach einbinden und konfigurieren lassen. Weitere Informationen unter: <https://fontawesome.com/>

Verwendete Versionen:

- fontawesome/fontawesome-svg-core: 1.2.19
- fontawesome/free-solid-svg-icons: 5.9.0
- fontawesome/vue-fontawesome: 0.1.6
- CC BY 4.0 License

**Animate.css** Animated.css bietet eine Sammlung vorgefertigter CSS-Animationen, die sich einfach in die Anwendung einbinden lassen (Version 3.7.2, <https://daneden.github.io/animate.css/>).

- The MIT License (MIT)

**Umfassendes JavaScript** Mit diesem Package lassen sich auch die neuesten Features von JavaScript bis hin zu ES2018 in die Anwendung integrieren. Babel (siehe unten), erlaubt es dann, diese Features in von allen Browsern unterstütztes JavaScript zu übersetzen (Version 2.6.5, <https://www.npmjs.com/package/core-js>).

- The MIT License (MIT)

---

**vuex** Vuex ist eine Erweiterung für Vue, die Features für application state management nach dem Vorbild von Redux zur Verfügung stellt (Version 3.1.1, <https://vuex.vuejs.org/>).

- The MIT License (MIT)

### 1.2.2 Dev-Dependencies

Um den Entwicklungsprozess möglichst produktiv zu gestalten, wurden einige Dev-Dependencies eingesetzt:

**Vue CLI** Die Vue CLI erlaubt ein schnelles Setup von neuen Vue-Projekten über die Kommandozeile. Dabei können vorab oder nachträglich über einfache Befehle Konfigurationen vorgenommen werden (Version 3.6.0, <https://cli.vuejs.org/>).

- The MIT License (MIT)

**ESLint** Da JavaScript eine dynamische Programmiersprache ist, kann die Fehlersuche von Zeit zu Zeit recht umständlich sein. Mit ESLint, einem JavaScript Linter, können viele Fehler (und spezifizierte Stil-Regeln) vorab erkannt bzw. durchgesetzt werden. Hier wird es in Verbindung mit Babel (neuste ECMA Script Syntax-Unterstützung) und Prettier verwendet. Weitere Informationen unter: <https://eslint.org/>

Verwendete Versionen:

- eslint: 5.16.0
- eslint-loader: 2.1.2
- eslint-plugin-vue: 5.0.0
- eslint-config-prettier: 4.3.0
- eslint-plugin-prettier: 3.1.0
- babel-eslint: 10.0.1
- vue/cli-plugin-eslint: 3.6.0
- The MIT License (MIT)

**Babel** Babel erlaubt es, die neuesten Features von JavaScript (ECMA Script) zu verwenden, ohne auf eine vollständige Browser-Unterstützung angewiesen zu sein. Babel übersetzt die neuesten Features in Code mit einer von allen Browsern unterstützten Version von ECMA Script (Version 7.4.4, <https://babeljs.io/>).

---

**Prettier** Prettier ist ein Style Formatter, der hier im Zusammenspiel mit ESLint zum Einsatz kommt. Bei jedem Speichervorgang werden die Style-Guides des Vue-Core-Teams automatisch umgesetzt. Es entsteht eine saubere und einheitliche code base (Version 1.18.2 mit vue/cli-plugin-babel: 3.6.0, <https://prettier.io/>).

- The MIT License (MIT)

Das genaue Setup für Linting und Formatierung ist von diesem (<https://medium.com/@gogl.alex/how-to-properly-set-up-eslint-with-prettier-for-vue-or-nuxt-in-vscode-e42>) Medium-Artikel inspiriert.

### 1.2.3 Tools

**VisualStudio** Als Texteditor kam für den Client ausschließlich VisualStudio Code zum Einsatz (<https://code.visualstudio.com/>).

**Affinity Designer** Zum Erstellen von Spielgrafiken wurde die kommerzielle Software Affinity Designer v1.7.1 von Serif verwendet (<https://affinity.serif.com/de/designer/>).

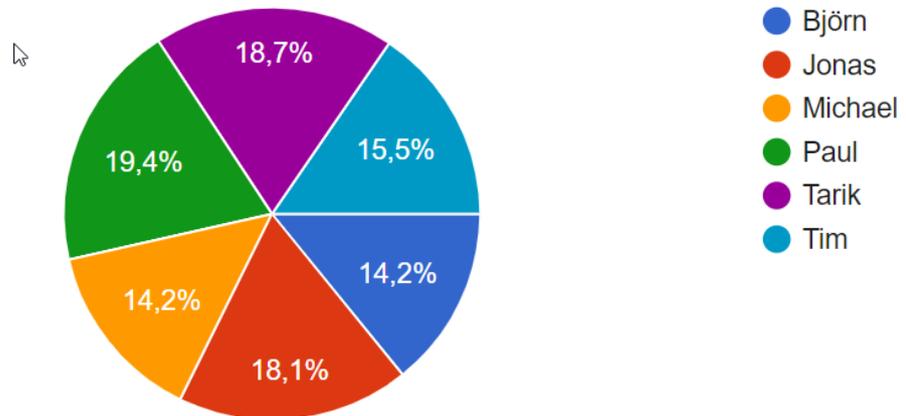
## 2 Projekttagbuch

Das Projekttagbuch wurde mit der Hilfe eines Google-Formular verwaltet. Die dazugehörige Eingabemaske ist unter folgendem Link zu finden: [https://docs.google.com/forms/d/e/1FAIpQLSdlc-WKKGjKdk1e3qPVxXkzdcqY6LYrXj0sgBuA4\\_gtZcW2Hw/viewform](https://docs.google.com/forms/d/e/1FAIpQLSdlc-WKKGjKdk1e3qPVxXkzdcqY6LYrXj0sgBuA4_gtZcW2Hw/viewform)  
Die einzelnen Einträge sind unter dem nachfolgenden Link einsehbar: <https://drive.google.com/open?id=1AJPGgD-9MZZR8e6xCiLpxgbUAKqJjAv-QcIBXes6I5M>  
Aus diesen Einträgen wurden die folgenden Diagramme automatisch generiert:

---

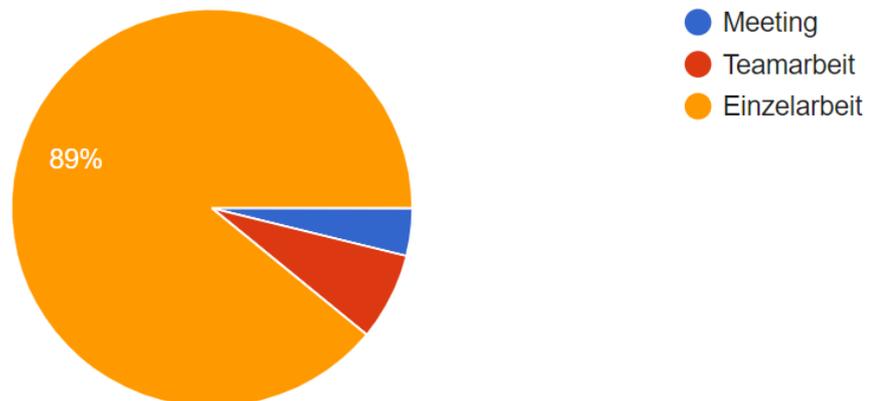
## Verantwortlicher

155 Antworten



## Typ

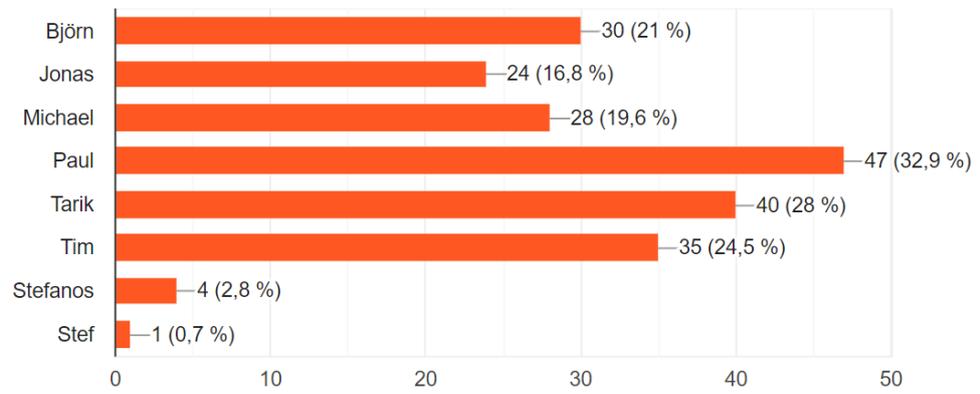
155 Antworten



---

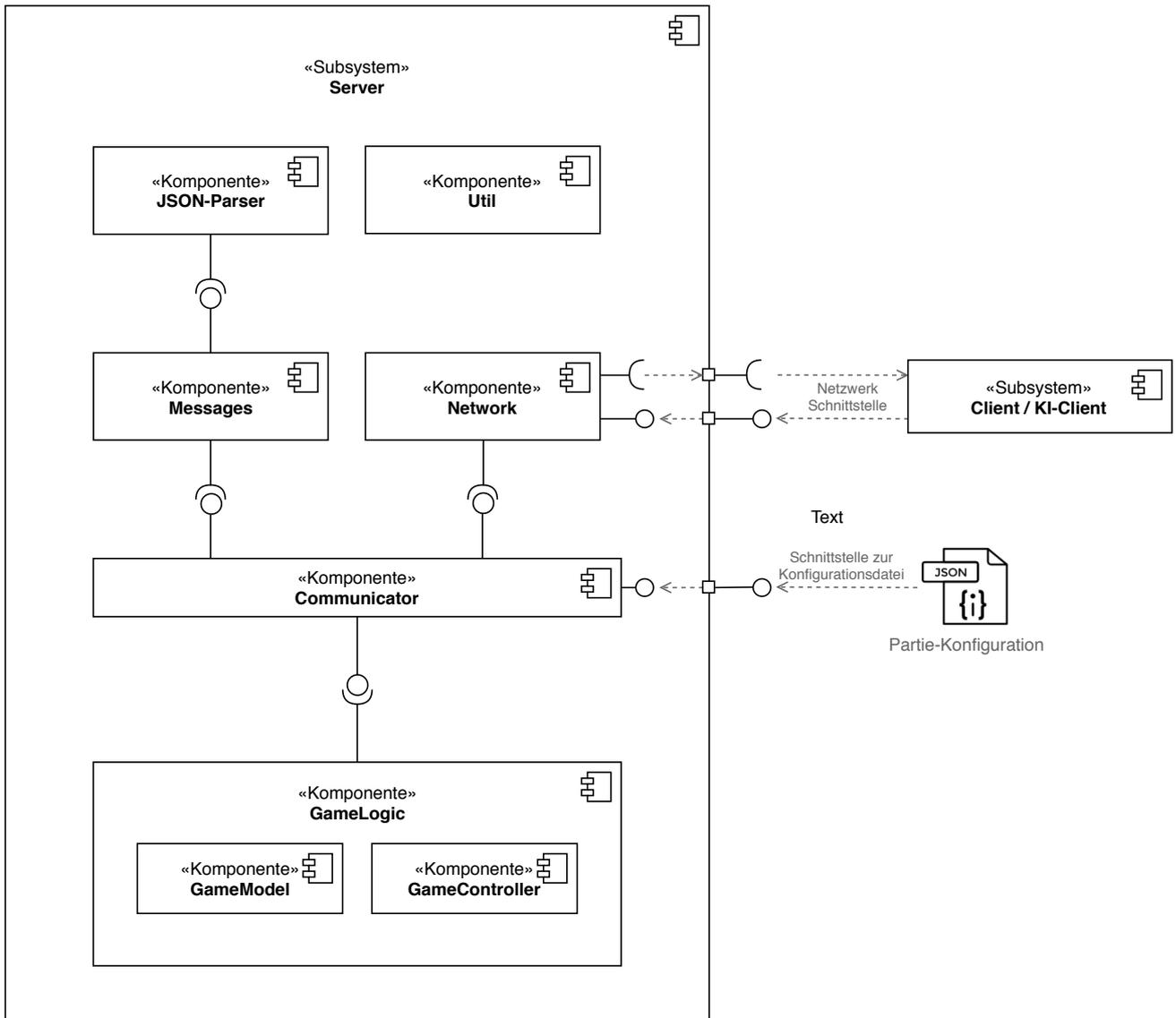
## Anwesend

143 Antworten



## 3 UML - Komponentendiagramm

### 3.1 Komponentendiagramm zum Server



Die Serveranwendung lässt sich grob in zwei Teile aufteilen: Die Spiellogik und den Kommunikator. Der Kommunikator ist für das Senden und Empfangen von Nachrichten, sowie für das Verwalten von Verbindungen und mehreren Lobbys zuständig. Empfangene Nachrichten werden an die Spiellogik weitergeleitet, die so den internen Spielzustand aktualisieren kann. Diese Updates werden dann wieder über den Kommunikator an die Clients

---

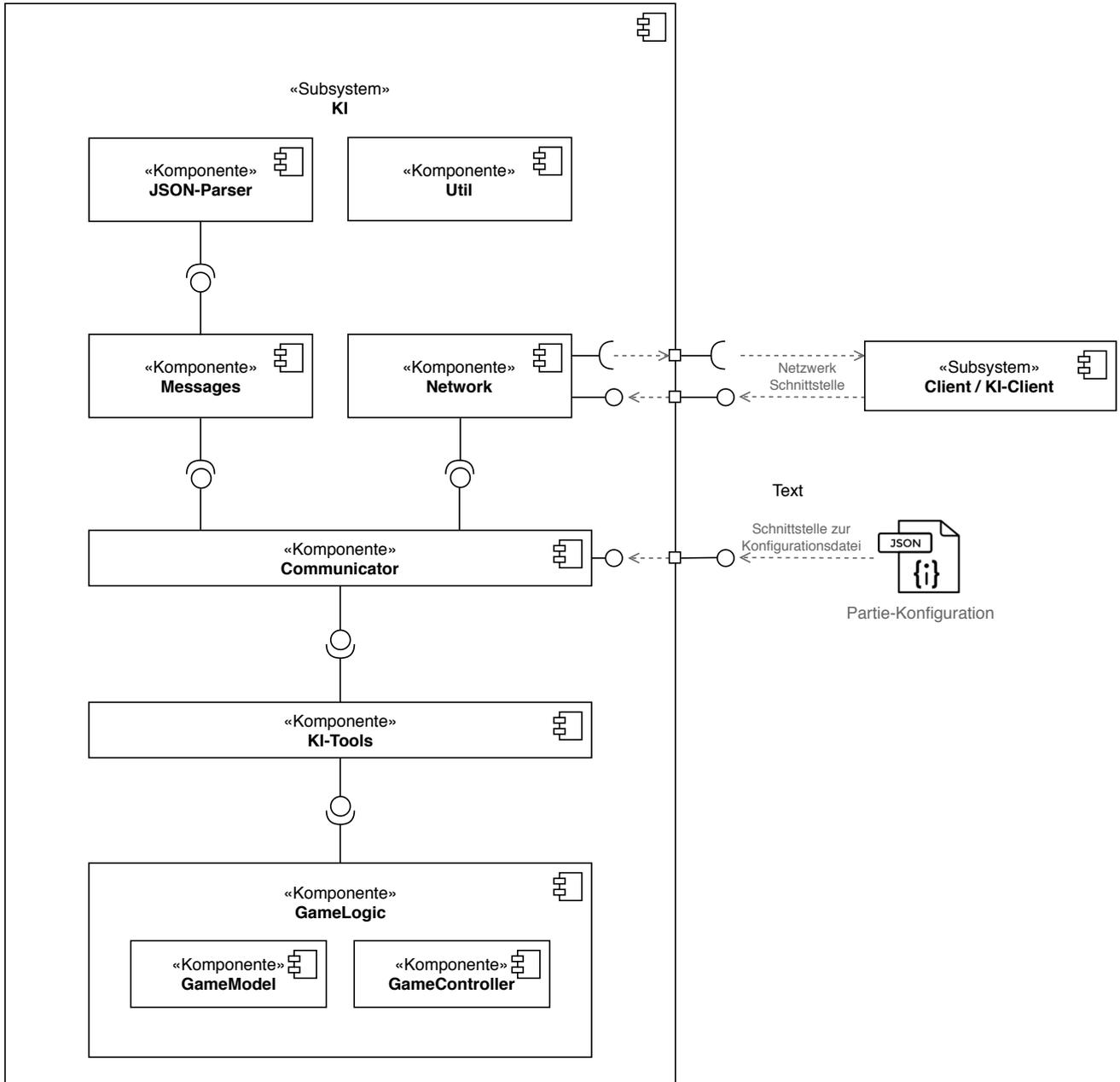
verteilt. Diese Aufteilung ermöglicht es uns, die einzelnen Komponenten bei der KI wiederzuverwenden. Die Komponenten sind voneinander komplett unabhängig und können deswegen einfacher getestet und verändert werden.

Genauer betrachtet, sind die zwei Hauptkomponenten weiter unterteilt. Die Spiellogik besteht aus GameModel und GameController. Erstere repräsentiert ausschließlich einen kompletten Spielzustand, ohne Informationen über die aktuelle Phase oder Ähnliches. Ein so repräsentierter Spielzustand lässt sich durch Methoden im GameController in einen anderen konsistenten Spielzustand überführen. Auch hier dient die Modularität der besseren Test- und Wartbarkeit.

Der Kommunikator nutzt intern die Messages- und Network-Bibliotheken. Erstere implementiert alle verschiedenen Nachrichtentypen, zweite ist ein moderner C++-Wrapper für die in C geschriebene libwebsockets-Bibliothek.

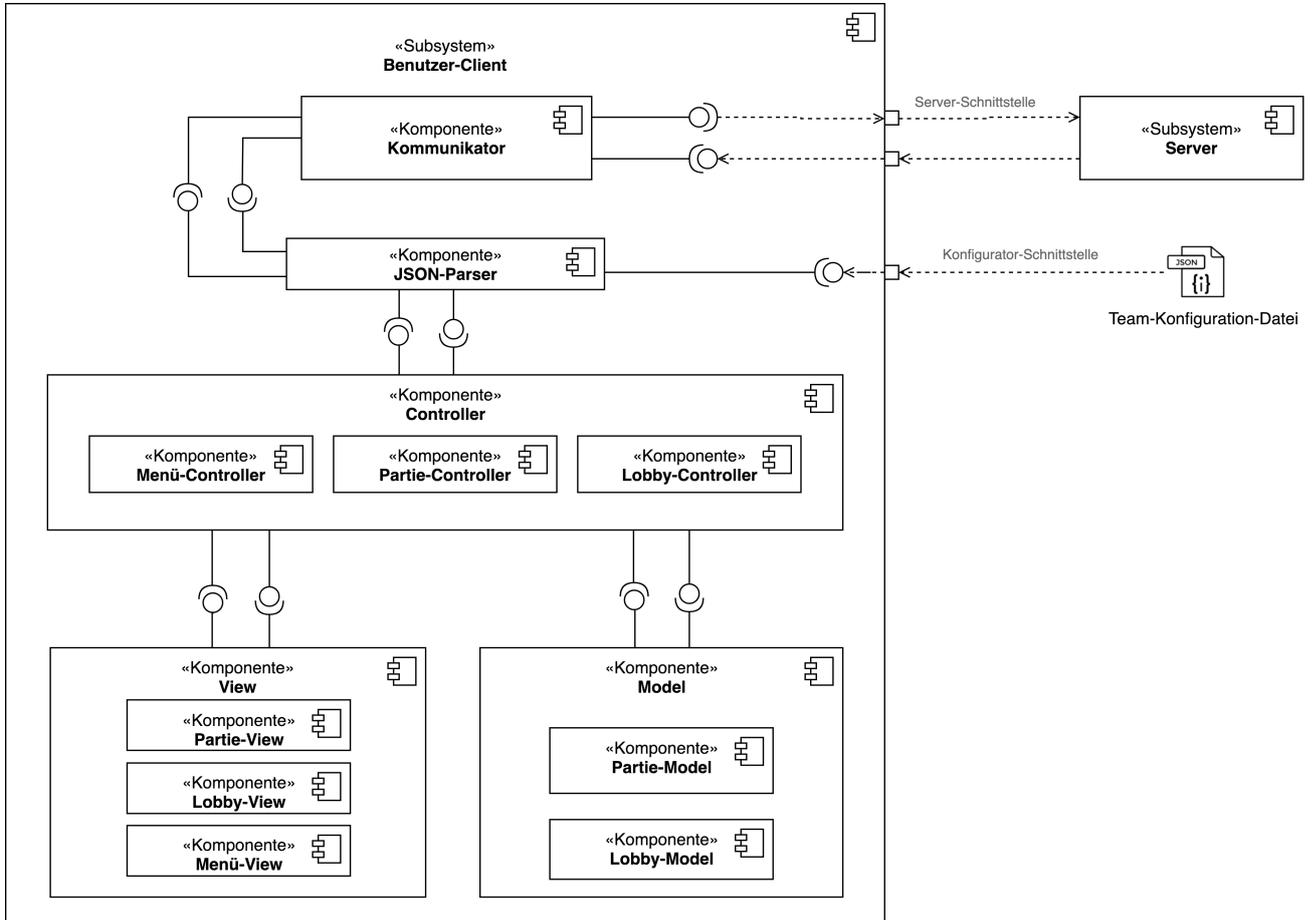
Zu guter Letzt werden allgemeine Tools wie z.B. ein Logging-Mechanismus oder Timer von unserer Utils-Bibliothek zur Verfügung gestellt.

### 3.2 Komponentendiagramm zur KI



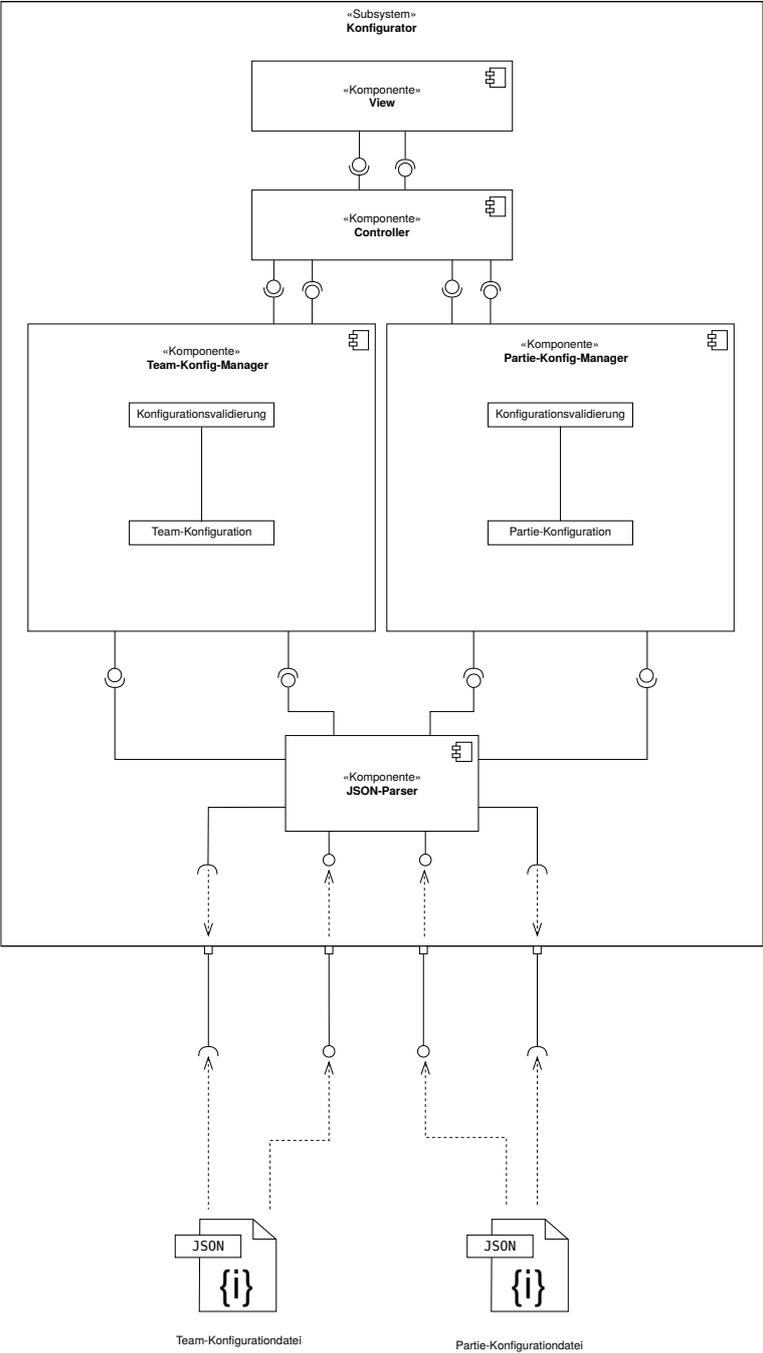
Durch die bereits bei der Serverkomponente angesprochenen Vorteile ist die KI fast identische zum Server aufgebaut. Der einzige Unterschied ist, dass die KI selbstständig Züge ausführen muss. Dazu verwendet sie Methoden aus der AI-Toolbox wie beispielsweise Such- und Optimierungsalgorithmen.

### 3.3 Komponentendiagramm zum Benutzer-Client



Der Client ist nach dem MVC-Prinzip aufgebaut. Für die verschiedenen Ansichten gibt jeweils eine View. Im Model werden die aktuelle Spielzustände bzw. Daten für den Anmeldevorgang gespeichert. Der Controller kommuniziert sowohl mit Model und View, als auch mit der Netzwerkschnittstelle.

### 3.4 Komponentendiagramm zum Level-Editor



Wie auch der eigentliche Client ist der Konfigurator nach dem MVC-Prinzip aufgebaut.