

Verteilte Systeme: Unsere Software

Testat

6. Theoriephase

des Studiengangs Informationstechnik (TIT18)

an der DHBW Ravensburg Campus Friedrichshafen

17.06.2021

| | |
|----------------------|----------------|
| Bearbeitungszeitraum | Theoriephase 6 |
|----------------------|----------------|

| | |
|----------------------|----------------|
| Matrikelnummer, Kurs | 5644223, TIT18 |
|----------------------|----------------|

| | |
|--|----------------|
| | 2041428, TIT18 |
|--|----------------|

| | |
|--|----------------|
| | 9424827, TIT18 |
|--|----------------|

| | |
|--|----------------|
| | 6017787, TIT18 |
|--|----------------|

| | |
|---------------------------------|-------------|
| Gutachter der Dualen Hochschule | Thomas Vogt |
|---------------------------------|-------------|

Inhaltsverzeichnis

| | |
|-----------------------------------|-----------|
| 1. Anforderungen | 3 |
| 2. Architektur | 6 |
| 3. Technologieentscheidung | 8 |
| 4. Umsetzung | 9 |
| 4.1 Datenbankbindung | 9 |
| 4.2 REST API Definition | 10 |
| 4.3 Serverkomponente | 11 |
| 4.4 Clientkomponente | 13 |
| 4.4.1 Benutzung | 13 |
| 4.4.2 Aufbau | 14 |
| 5. Fazit | 15 |

1. Anforderungen

In dieser Arbeit sollen viele umfangreiche Funktionen umgesetzt werden. Es sollen unter Anderem Blackboards erstellt, verändert und viele weitere Funktionen umgesetzt werden. Die Anforderungen werden in den folgenden Punkten beschrieben.

- **Das Erstellen eines Blackboards**

- Erstellt auf dem Server ein neues leeres Blackboard
- Es gibt zwei Übergabeparameter
 - Name: Enthält den Namen des Blackboards
 - Gültigkeit: Enthält die Gültigkeit einer Nachricht in Sekunden
- Es gibt einen Rückgabewert
 - Rückgabe, ob Blackboard erfolgreich oder nicht erfolgreich erstellt wurde
- Folgende Werte existieren für den Rückgabewert
 - Blackboard erfolgreich erstellt
 - Blackboard existiert schon
 - Ungültige Parameter
 - Interne Fehler

- **Inhalt eines Blackboards aktualisieren**

- Aktualisiert den Inhalt eines Blackboards und dessen Zeitstempel für die Daten
- Es gibt zwei Übergabeparameter
 - Name: Enthält den Namen des Blackboards
 - Daten: Enthält die neue Information, die auf das Blackboard gespeichert werden soll
- Es gibt einen Rückgabewert
 - Rückgabe, ob Blackboard erfolgreich aktualisiert oder nicht erfolgreich aktualisiert wurde
- Folgende Werte existieren für den Rückgabewert
 - Blackboard erfolgreich aktualisiert
 - Blackboard existiert nicht
 - Ungültige Parameter
 - Interne Fehler

- **Das Löschen des Inhalts eines Blackboards**

- Löscht den Inhalt eines Blackboards
- Es gibt einen Übergabeparameter
 - Name: Enthält den Namen des Blackboards
- Es gibt einen Rückgabewert
 - Rückgabe, ob der Inhalt des Blackboards erfolgreich oder nicht erfolgreich gelöscht wurde
- Folgende Werte existieren für den Rückgabewert
 - Blackboard erfolgreich aktualisiert
 - Blackboard existiert nicht
 - Ungültige Parameter
 - Interne Fehler

- **Das Auslesen des Inhalts eines Blackboards**

- Liest den Inhalt eines Blackboards aus und die Gültigkeit der Daten werden signalisiert. Ist eine Nachricht veraltet, so wird diese Information ebenfalls zurückgegeben
- Es gibt einen Übergabeparameter
 - Name: Enthält den Namen des Blackboards
- Es gibt bis zu drei Rückgabewerte
 - Der Inhalt des Blackboards
 - Die Gültigkeitsinformation des Blackboards
 - Rückgabe, ob Blackboard erfolgreich oder nicht erfolgreich ausgelesen wurde
- Es existieren folgende Werte zu dem Rückgabewert des Erfolgs/Misserfolgs:
 - Blackboard erfolgreich gelesen
 - Blackboard existiert nicht
 - Blackboard ist leer
 - Ungültige Parameter
 - Interne Fehler

- **Rückgabe des Statuses eines Blackboards**

- Gibt den aktuellen Status eines Blackboards zurück
- Es gibt einen Übergabeparameter
 - Name: Enthält den Namen des Blackboards
- Es gibt vier Rückgabewerte
 - Blackboard gefüllt oder leer
 - Die Gültigkeitsinformation des Blackboards als Zeitstempel
 - Die Gültigkeit des Blackboards selbst, also gültig oder ungültig
 - Rückgabe, ob der Status eines Blackboards erfolgreich oder nicht erfolgreich zurückgegeben wurde
- Es existieren folgende Werte zu dem Rückgabewert des Erfolgs/Misserfolgs:
 - Blackboard Status erfolgreich gelesen
 - Blackboard existiert nicht
 - Ungültige Parameter
 - Interne Fehler

- **Auflistung der Blackboards**

- Listet alle vorhandenen Blackboards auf
- Es gibt keine Übergabeparameter
- Es gibt zwei Rückgabewerte
 - Liste der Namen der existenten Blackboards
- Es existieren folgende Werte zu dem Rückgabewert des Erfolgs/Misserfolgs:
 - Blackboard Liste wurde erfolgreich zurückgegeben
 - Interne Fehler

- **Löschen eines Blackboards**

- Löscht ein Blackboard
- Es gibt einen Übergabeparameter
 - Name: Enthält den Namen des Blackboards
- Es gibt einen Rückgabewert
 - Rückgabe, ob Blackboard erfolgreich oder nicht erfolgreich gelöscht wurde
- Folgende Werte existieren für den Rückgabewert
 - Blackboard wurde erfolgreich gelöscht
 - Blackboard existiert nicht
 - Ungültige Parameter
 - Interne Fehler

- **Löschen aller Blackboards**
 - Löscht alle Blackboards
 - Es gibt keine Übergabeparameter
 - Es gibt einen Rückgabewert
 - Rückgabe, ob die Blackboards erfolgreich oder nicht erfolgreich gelöscht wurden
 - Folgende Werte existieren für den Rückgabewert
 - Alle Blackboards erfolgreich gelöscht
 - Interne Fehler

2. Architektur

Das Gesamtsystem wird in drei verschiedene Teilsysteme unterteilt:

- Client-Komponente
- Server-Komponente mit Logik
- Datenspeicher

Die Teilsysteme kommunizieren untereinander über fest definierte Schnittstellen. Damit ist es möglich ein einzelnes Teilsystem auszutauschen, solange die Kompatibilität mit der Schnittstelle gewährleistet wird.

Über die Server-Anwendung ist es möglich die verschiedenen Arten von Verteilungstransparenz zu erfüllen. Indem die Client-Anwendung nicht direkt auf den Datenspeicher zugreift können folgende Kriterien erreicht werden:

- Zugriffstransparenz
Der Client sieht nicht, wie die Daten gespeichert werden. Der Server hat die Aufgabe die Daten zwischen der Darstellung im Datenspeicher und der Darstellung in der Client-Server-Schnittstelle umzuwandeln.
- Ortstransparenz
Über die Client-Server-Schnittstelle bleibt verborgen, wo sich die Daten befinden. Außerdem kann über die Schnittstelle der verarbeitende Server verborgen werden
- Migrationstransparenz.
Äquivalent zur Ortstransparenz spielt der Standort der Daten keine Rolle für den Client. So bekommt dieser auch nicht mit, sollte sich der Standort ändern.
- Replikationstransparenz

Über die Client-Server-Schnittstelle kann die Position und Anzahl der Server verborgen werden. So können sich zum Beispiel die gleichen Server und Daten an verschiedenen Orten befinden. Über die Server-Datenspeicher Schnittstelle kann auch zusätzlich aus Server-Sicht verborgen werden, dass es mehrere Datenspeicher an verschiedenen Orten gibt

- Nebenläufigkeitstransparenz

Diese muss entweder vom Datenspeicher selbst bereitgestellt werden, kann ansonsten aber über den Server implementiert werden. Der Client bekommt davon nicht mit und kann über die Client-Server-Schnittstelle unabhängig von anderen Systembenutzern das Gesamtsystem nutzen

Als Diagramm dargestellt sieht die Architektur folgendermaßen aus:

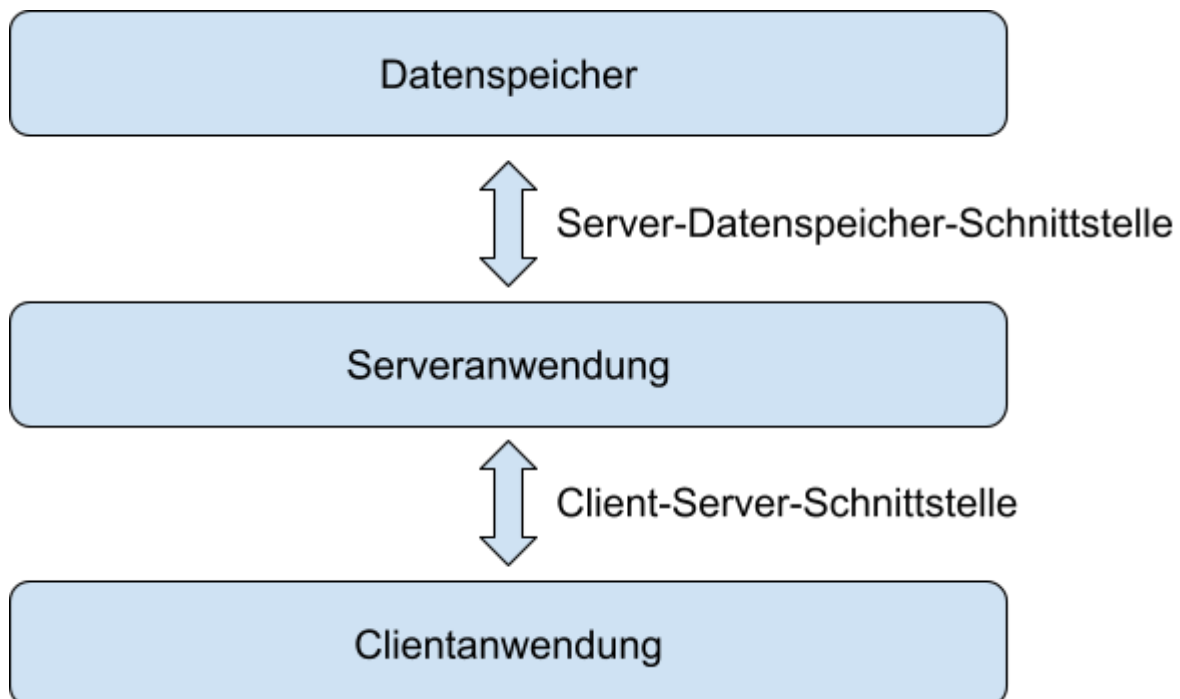


Abbildung 1: Visuelle Darstellung der Teilkomponenten und Schnittstellen

3. Technologieentscheidung

Datenspeicher:

Für den Datenspeicher wurde als Technologie MongoDB gewählt. Diese bringt Kriterien wie Nebenläufigkeits- oder Replikationstransparenz von sich aus mit und verringert dadurch den Aufwand, der in Serverkomponente betrieben werden muss um diese sicherzustellen. Des Weiteren arbeitet MongoDB als Dokumentenbasierte Datenbank und ist dementsprechend optimiert. Die zu speichernden Daten entsprechend diesem Ansatz, dabei ist ein Blackboard ein Dokument und die Eigenschaften entsprechen den Daten des Dokumentes.

Server-Datenspeicher-Schnittstelle:

Aus MongoDB resultierend wird dessen Abfragesprache als Server-Datenspeicher-Schnittstelle gewählt. Für diese gibt es zahlreiche Bibliotheken wodurch die Nutzung erleichtert wird.

Client-Server-Schnittstelle:

Für die Schnittstelle zwischen Client und Server wurde sich für REST entschieden. Auf dem HTTP-Protokoll basierend ermöglicht diese bereits eine Ortstransparenz durch das Nutzen von URI-Adressen. Ein weiterer Grund ist der weitreichende Support durch diverse Bibliotheken, welche die Umsetzung erleichtern.

Client- / Serverkomponente:

Für die Umsetzung der Client- bzw. Serverkomponente ist die Wahl auf Python als Programmiersprache gefallen. Die Hauptgründe dafür sind

- Plattformunabhängige Programmiersprache
- Definierte Schnittstellen sind als Bibliotheken verfügbar
- Schnelle Entwicklungszeit

4. Umsetzung

In diesem Kapitel wird die Umsetzung des Projektes im Detail erläutert. Das Kapitel ist in drei Unterkapitel aufgegliedert, die die einzelnen Komponenten beschreiben.

4.1 Datenbankanbindung

Um die Daten eines Blackboards abspeichern zu können, bedarf es einer Datenbank. Dieses Projekt soll mittels einer dokumentenbasierten Datenbank aufgebaut werden. Als Datenbank läuft eine MongoDB, welche als Clouddienst bei AWS gehostet wird.

Um MongoDB in diesem Projekt verwenden zu können, müssen zwei packages installiert werden:

- pymongo (z.B. pip install pymongo)
- dnspython (z.B. pip install dnspython)

Für die Datenbankanbindung ist das Dokument `server\database.py` erstellt worden. In diesem Dokument existiert die Klasse Database, welche alle Anfragen und Antworten die der Client sendet und empfängt verarbeitet. In der Methode `__init__` wird die Datenbank im Code deklariert und eine Verbindung zur MongoDB aufgebaut. Die Klasse Database wird über die Datei `server\server.py` aufgerufen und übermittelt die angeforderten Daten, beziehungsweise schreibt neue Informationen in die Datenbank.

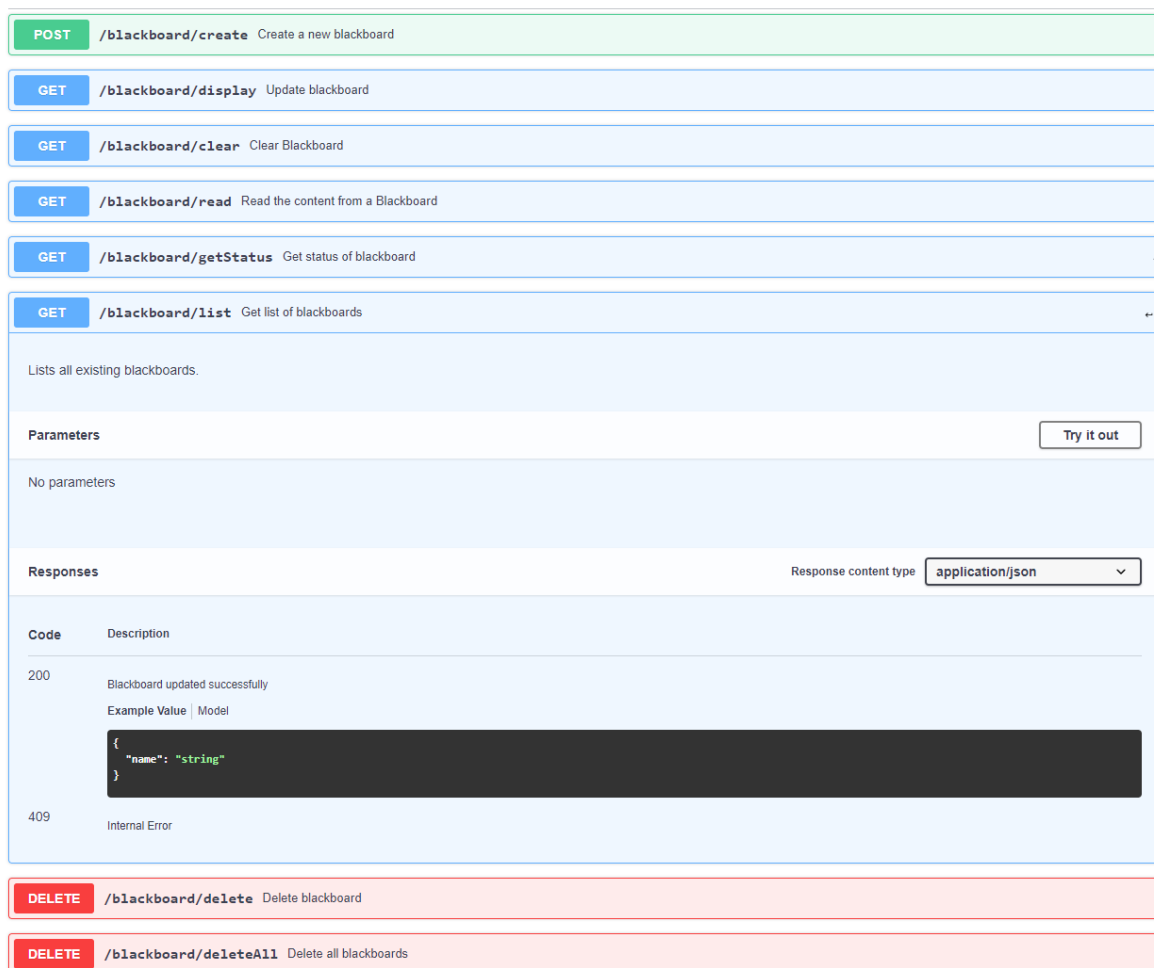
Die Datenbank ist so aufgebaut, dass es eine collection von Blackboards gibt, in der alle Blackboards gespeichert werden. Beim Anlegen eines Blackboards muss der Name (name) und eine Gültigkeitsdauer (validityTime) angegeben werden. Jedes Blackboard erhält beim Anlegen eine automatisch generierte ID (`_id`) und den aktuellen Zeitstempel (timestamp), sowie die vorerst leere Variable content.

Über den Client ist es möglich die in der Datenbank gespeicherten Blackboards so zu bearbeiten und aktualisieren, wie es in den Anforderungen beschrieben ist. Dazu kann der Content (content) eines Blackboards gesetzt, geändert oder gelöscht werden. Bei allen Änderungen wird der Zeitstempel des Blackboards aktualisiert, um die Gültigkeit (validity) des Blackboards berechnen zu können. Diese Änderungen der Daten und die Berechnung der Gültigkeit geschehen in den Methoden der Klasse Database.

Desweiteren können Information über die Blackboards mit dem Client abgerufen werden. Dazu werden Anfragen mit Parametern, wie beispielsweise dem Namen eines Blackboards, an die Datenbank gesendet. Die Antworten der Datenbank werden über den Client in der Konsole ausgegeben.

4.2 REST API Definition

Das Aussehen der REST API wurde mittels des “OpenAPI Specification 2.0” definiert. Hierfür wird eine YAML-Datei mit dem entsprechendem Format der Spezifikation bereitgestellt, welche die Methoden und Datenstrukturen der REST API beschreibt. Diese Datei kann von vorhandenen Tools weiterverwendet werden, um zum Beispiel eine übersichtliche und leichter lesbare Dokumentation für den Menschen zu erzeugen (siehe Abbildung 2).



The screenshot displays a REST API documentation interface with the following endpoints:

- POST** `/blackboard/create` Create a new blackboard
- GET** `/blackboard/display` Update blackboard
- GET** `/blackboard/clear` Clear Blackboard
- GET** `/blackboard/read` Read the content from a Blackboard
- GET** `/blackboard/getStatus` Get status of blackboard
- GET** `/blackboard/list` Get list of blackboards
 - Lists all existing blackboards.
 - Parameters**: No parameters
 - Responses**: Response content type is `application/json`
 - Code 200**: Blackboard updated successfully
 - Example Value | Model
 - ```
{
 "name": "string"
}
```
  - Code 409**: Internal Error
- DELETE** `/blackboard/delete` Delete blackboard
- DELETE** `/blackboard/deleteAll` Delete all blackboards

Abbildung 2: REST-API Definition

Es gibt drei verschiedene HTTP Anfragen GET, POST und DELETE, die in diesem Projekt verwendet wurden. Die Datei befindet sich im Quellordner unter "server/API\_definitions.yaml". Eine genauere Erläuterung des Formats und der Interpretation des Inhaltes findet sich auf der Spezifikationsseite unter <https://swagger.io/specification/v2/>

## 4.3 Serverkomponente

Nachdem die Anforderungen an das Projekt klar definiert wurden, kann mit der Implementierung begonnen werden. Um eine Kommunikationsschnittstelle zwischen dem Benutzer und der Datenbank zu erstellen, wird eine REST-API verwendet. Die REST-API nimmt Anfragen entgegen, liest die Daten aus der Datenbank aus, verarbeitet die Daten entsprechend der Anforderungen und stellt sie dem Benutzer zur Verfügung.

Hierzu wurde für jede Anfrage verschiedene Routen erstellt.

. Für jede Route (Endpoint) kann es Übergabe und Rückgabeparameter geben..

Anhand der API-Definition in Abbildung 1 kann mit der Implementierung des Webservers begonnen werden. Wie bereits in der Technologie Entscheidung beschrieben, wird für die Umsetzung Python mit der Bibliothek Flask verwendet. Flask ermöglicht dem Entwickler, die Routen auf trivialer Vorgehensweise zu erstellen. Hierzu werden Standardmethoden von Flask verwendet. Mit `@app.route()` wird eine neue Route erstellt. Mithilfe eines String wird der Name der Route übergeben. Zusätzlich kann die HTTP Anfragen Methoden festgelegt werden. Nachdem die Route erstellt ist, wird für jede Route eine separate Methode erstellt. Die Routen-Methode wertet die Daten aus.

Übergabeparameter werden mit einem Requests abgerufen.

```
NAME = request.args.get('<NAME>')
```

Mit einem Rückgabewert werden die Daten dem Benutzer zur Verfügung gestellt werden. in allgemeiner Aufbau einer Route ist in der Abbildung 3 zu sehen.

```
@app.route('/blackboard' , methods=['GET'])
def blackboard():
 return 'blackboard'
```

**Abbildung 3: Routenerstellung**

Damit der Benutzer auf Daten aus der Datenbank zugreifen kann, wird eine Verbindung zu der Datenbank benötigt. Hierzu wird die Datenbank, die im Kapitel 4.1 beschrieben ist, verwendet. In der Datenbank gibt es Abfragemethoden, die aufgerufen werden. Je nach Methode ist definiert, welche Daten zurückgegeben werden. Zusätzlich zu jedem Rückgabewert wird ein Statuscode mitgegeben. Der Statuscode dient dazu, um fehlerhafte Abfragen zu identifizieren.

| Code | Message            | Description               |
|------|--------------------|---------------------------|
| 200  | OK                 | Operation is successfully |
| 400  | Invalid parameters | Request-Message Error     |
| 404  | Not found          | Element not found         |
| 409  | Conflict           | Internal Error            |
| 444  | No Response        | Don't get a response.     |

**Tabelle 1: HTTP-Statuscodes**

Wenn die Abfrage erfolgreich ist, werden die Daten verarbeitet. Hierzu wird je nach Anforderung bestimmte Daten ausgegeben. Damit sowohl der Client als auch ein Mensch die Daten auslesen kann, sollen die Daten im JSON-Format übermittelt werden. Hierzu wird die Funktion "jsonify" von Flask verwendet, die die Daten aus dem Klartext in das JSON-Format konvertiert. Als Rückgabewert werden die konvertierten Daten und der Statuscode übermittelt.

## 4.4 Clientkomponente

Im Folgenden wird die Clientkomponente des Projektes beschrieben. Diese dient dem Nutzer zur Nutzung der Software und kommuniziert mit dem Server.

### 4.4.1 Benutzung

Die entwickelte Clientkomponente ist als reine Konsolenanwendung konzipiert. Die Anwendung bietet verschiedene Optionen um mit den Blackboards zu interagieren. Die Option und eventuelle Daten werden als Kommandozeilen-Parameter beim Aufrufen der Anwendung übergeben. Die Datei befindet sich im Quellordner unter "client/client.py". Mit dem optionalen Parameter "-h" wird die genau Syntax für einen Befehl angezeigt. Mit dem optionalen Parameter "--debug" wird die Debug-Anzeige aktiviert und die Kommunikation mit der Server auf der Konsole ausgegeben.

Für die Interaktion mit den Blackboard sind folgende Optionen möglich

- `./client.py create <Blackboard Name> <Validitätszeit>`  
Erstellt ein neues Blackboard
- `./client.py display <Blackboard Name> <Nachricht>`  
Schreibt eine neue Nachricht auf das Blackboard
- `./client.py clear <Blackboard Name>`  
Löscht ein Blackboard
- `./client.py read <Blackboard Name>`  
Gibt die Nachricht und die Gültigkeit auf dem gewählten Blackboard aus
- `./client.py status <Blackboard Name>`  
Zeigt Status Informationen zum gewählten Blackboard an
- `./client.py list`  
Listet alle existierende Blackboards auf
- `./client.py delete <Name> [--all]`  
Löscht das gewählte Blackboard. Wird der Parameter "--all" verwendet, wird der Name ignoriert und es werden alle Blackboards gelöscht

#### 4.4.2 Aufbau

Die Anwendung ist in zwei Schichten untergliedert. Die eine Schicht ist für die Kommunikation mit dem Benutzer zuständig, die andere Schicht für die Kommunikation mit dem Server.

Die Server-Kommunikations-Schicht als Klasse "API" implementiert. Mittels der "requests"-Bibliothek werden REST Anfragen an die Server-Komponente gesendet und empfangen. Die Daten richten sich dabei nach der API-Definition. Zu Debugging Zwecken gibt es die Möglichkeit über ein Flag die Ausgabe der Server Antworten ein- oder auszuschalten. Ausgegeben wird hierbei der Status-Code und der zurückgesendete Seiteninhalt. Antwortet der Server mit einem in der API-Definition definierten Fehler, so wird im Code ein API-Error geworfen mit dem Grund als Fehlernachricht. Ist der Fehler nicht in der API-Definition definiert, so ist die Fehlernachricht "Unknown Error".

Die Benutzer-Kommunikationsschicht benutzt die "argparse"-Bibliothek um die Kommandozeile auszuwerten. Die übernimmt auch die Ausgabe der Hilfe wenn die Parameter nicht wie vorgeschrieben gegeben werden. Abhängig von den Parametern wird die entsprechende Funktion der API-Klasse aufgerufen und die nötigen Parameter übergeben. War der API-Aufruf erfolgreich, werden die empfangen Daten dem Benutzer präsentiert. Bei einem Fehler wird dieses dem Benutzer signalisiert und der Fehlercode sowie die Fehlernachricht in der Konsole ausgegeben.

## 5. Fazit

Das Ergebnis der Arbeit umfasst die Erstellung eines Servers, der Dienste zum Erstellen, Löschen von Blackboards, sowie zum Schreiben und Lesen von Blackboards anbietet.

Der Server wird vom Client über ein Kommandozeilenprogramm angesprochen.

Anhand der klar definierten Anforderungen an Server, Datenbank und Client, konnte das Testat bearbeitet werden.

Das Projekt wird mit der Programmiersprache Python umgesetzt und die Daten sollen in einer dokumentenbasierten MongoDB gespeichert werden. Für die Kommunikation zwischen Datenbank und Client wird Flask als Python Bibliothek verwendet.

Damit die Kommunikationsschnittstelle für alle Entwickler eindeutig ist, wurde zu Beginn die API genau spezifiziert. Hierzu wurde der Swagger Editor verwendet, der die Definition in einer YAML-Datei speichert.

Die Routen (Endpoints) konnten mit Flask einfach realisiert werden. Innerhalb der Routen werden die Daten aus der Datenbank abgefragt bzw. es werden Daten in die Datenbank geschrieben. Die Datenbankklasse wurde logisch einfach gehalten und dient nur zur Datenabfrage. Die logische Vorgehensweise, welche Daten angezeigt bzw. in die Datenbank geschrieben werden, wird in der server.py Datei realisiert. Mithilfe der Realisierung des Clients über ein Kommandozeilenprogramm können die Blackboards mit wenigen Befehlen verwaltet werden.

Zusätzlich für die Erstellung der REST-API wurde für jede Route ein Testcase implementiert, dass Fehler entdeckt und behoben werden konnten.

Zusammenfassend lässt sich sagen, dass das Testat mit Python mithilfe von Flask und der MongoDB umgesetzt werden konnte. Sehr wichtig für die Bearbeitung war die Definition der REST-API mithilfe von Swagger. Damit konnte jedes Gruppenmitglied sein Arbeitspaket ohne großen Kommunikationsaufwand umsetzen. Jedes Arbeitspaket konnte mithilfe der Tests auf Fehler überprüft werden. Damit konnte jedes Gruppenmitglied die Übersicht über das Projekt behalten.