

Webanwendung in Processing

Projektdokumentation

Lilian Hung

February 3, 2017

Inhalt

1	Einleitung	3
1.1	Beschreibung der Software als Produkt	3
2	Technische Dokumentation	3
2.1	Processing und Processing.js	3
2.1.1	Skizzenaufbau	6
2.1.2	Würfel in Processing und Processing.js	7
2.2	Dokumentation der REST-Schnittstelle	8
2.3	Serverseitige Implementierung	11
2.4	Datenbankmodell	12
3	Entwicklungs- und Umsetzungsphase	13
3.1	Definition der Zielgruppe	14
3.2	Projektziele	14
3.3	Nutzerinteraktionen	15
4	Abschlussphase	15
5	Fazit	16
5.1	Ausblick	16

Liste der Beispiele

1	Einbinden in Canvas-Tag	5
2	Processing-Code in Script-Tag	5
3	Processing.js als Library in JavaScript	6
4	setup()-Funktion	7
5	box()-Funktion	7
6	Modellierungsfunktionen	7
7	Response-Body der GET-Operation auf "/cubes"	8
8	Request-Body der POST-Operation auf "/cubes"	8
9	Response-Body der POST-Operation auf "/cubeSketches"	9
10	Request-Body der POST-Operation auf "/cubeSketches"	10
11	CRUD-Operationen aus storage.js	12
12	Request-Routing und Implementierung der REST-Methoden aus app.js	12

1 Einleitung

Die Processing-Webapp soll einige der Möglichkeiten, die sich mit Processing und Processing.js ergeben, demonstrieren und Nutzern ohne Vorkenntnisse in Programmierung oder visuellen Bearbeitung von 2D- oder 3D-Objekten näherbringen. In diesem Projekt wurde eine interaktive Processing-Skizze erstellt und eine REST-Schnittstelle geschaffen, um die erstellten Skizzen direkt für andere Nutzer der Webseite verfügbar zu machen.

1.1 Beschreibung der Software als Produkt

Die Processing Webapp ist eine Applikation zum Experimentieren an verschiedenen Parametern eines Objekts, das mit Hilfe von Processing beschrieben und gerendert wird. Welches Objekt konkret angeboten wird hängt von dem Anbieter ab. In diesem Beispiel ist es anfangs nur ein Würfel. Dieser Würfel lässt sich in einem dreidimensionalen Raum in seiner Größe skalieren, in alle Richtungen drehen und in seiner Position auf allen der x- und y-Achse verschieben. Als zusätzliches Feature können, nach dem Festlegen des aktuellen Würfels, weitere Würfel nach und nach hinzugefügt werden und dadurch eine Skizze erstellt werden. Diese Informationen können mit einem Klick auf den "Speichern"-Button in eine öffentliche Liste hinterlegt werden. Diese Liste ist so aufgebaut, dass sie auf der selben Seite angezeigt und beim Speichern eines neuen Objekts aktualisiert wird. Zusätzlich lässt sich jedes Objekt der Liste mit einem Klick auf den "Anzeigen"-Button in der Processing-Canvas anzeigen.

2 Technische Dokumentation

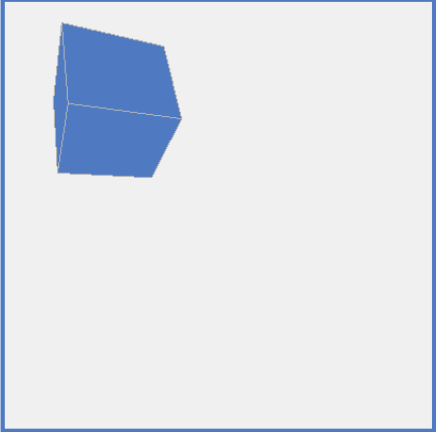
2.1 Processing und Processing.js

In der Applikation ist Processing.js ein zentrales Werkzeug. Processing.js ist eine Bibliothek und ein Übersetzer für die visuelle Programmiersprache Processing in JavaScript, um eine direkte Einbindung in Webseiten zu ermöglichen.

Processing wurde ursprünglich für (bildende) Künstler und Designer entwickelt, um einen schnelleren und einfacheren Zugang zum Programmieren zu ermöglichen. Es ermöglicht hauptsächlich die Erstellung von Illustrationen und Animationen in einem digitalen Kontext. Processing baut auf

Processing Web App

Cube



x-coordinate:

y-coordinate:

Cube size:

Rotation x:

Rotation y:

Cube Alias: [Enter alias](#)

Current cube alias:

[Save sketch](#)

[Save cube values](#)

[Add cube](#)

[Reset](#)

Saved sketches

- ID: 588f01f9bb2430292720f665
 - First cube's alias: genericCubeAlias[Show](#)
- ID: 588f03e0bb2430292720f666
 - First cube's alias: genericCubeAlias[Show](#)
- ID: 588f0470bb2430292720f667
 - First cube's alias: genericCubeAlias[Show](#)

Saved cubes

- 58774fa2bb8bdd1faaf2f352
 - Alias: cubey
 - x-coordinate: 267
 - y-coordinate: 95
 - cube size: 69
 - rotation x: 0.89
 - rotation y: 4.76[Show](#)
- 587b5bf2dad06947f5b9e0b7
 - Alias: cube2
 - x-coordinate: 100
 - y-coordinate: 100
 - cube size: 90
 - rotation x: 0.5[Show](#)

Figure 1: Aufbau

der Programmiersprache Java auf. Es ist in einigen Punkten vereinfacht – unter anderem in seiner Syntax – und beinhaltet eine vereinfachte Java-Schnittstelle zum Erstellen von Grafiken. Die Verwendung von Processing findet in einer eigenen integrierten Entwicklungsumgebung statt, die bereits die programmierte Skizze rendern kann.

Wie erwähnt ermöglicht Processing.js den Einsatz von Processing direkt integriert in Webseiten mittels HTML5 und dem Canvas-Tag. Es gibt grob gesehen drei verschiedene Möglichkeiten, Processing-Skizzen in eine Webseite einzufügen. Diese Methoden sind nicht ausschließlich auf diese Art zu nutzen, sondern können miteinander verwendet beziehungsweise stückweise ausgetauscht werden.

Die erste wäre ein Canvas zu erstellen und darin eine Processing-Datei (.pde) zu referenzieren (Siehe 1). Diese Methode ist sehr simpel. Es ist aber auch möglich ein Script zu schreiben, das Inhalte der eingebundenen Processing-Datei abrufen oder verändert.

Die zweite Möglichkeit wäre, den Processing-Code, wie man ihn auch nativ in Processing selbst schreiben würde, mittels eines Script-Tags einzubinden (Siehe 2).

Letztere Möglichkeit ist es, keinen Processing-Code zu schreiben, sondern nur die Bibliothek von Processing.js zu nutzen und die Processing-Anweisungen in JavaScript in einem Sketch-Objekt zu schreiben und dieses dann in einem neuen Processing-Object einzubinden (Siehe 3). Hier ist gut zu erkennen, dass die Verwendungsweisen von Processing.js viel Spielraum haben. Letztere Möglichkeit gibt weiterhin einen Einblick darin, wie Processing.js den Processing code kompilieren würde.

```
1 <canvas data-processing-sources="hello-web.pde"></canvas>
```

Beispiel 1: Einbinden in Canvas-Tag

```
1 <script type="text/processing" data-processing-target="
  canvasID">
2 void setup(){
3   ...
4 }
5
6 void draw(){
```

```

7      ...
8  }
9  </script>
10 <canvas id="canvasID"></canvas>

```

Beispiel 2: Processing-Code in Script-Tag

```

1 <canvas id="canvasID"></canvas>
2 <script type="application/javascript">
3     var sketch = new Processing.Sketch();
4
5     sketch.attachFunction = function(processing) {
6
7         processing.setup = function() {
8             ...
9         };
10        processing.draw = function() {
11            ...
12        };
13    };
14
15    var canvas = document.getElementById("canvasID");
16    // attaching the sketch to the canvas
17    var p = new Processing(canvas, sketch);
18 </script>

```

Beispiel 3: Processing.js als Library in JavaScript

2.1.1 Skizzenaufbau

Wie in Beispiel 2 und 3 sichtbar, werden in Processing wiederkehrend die Funktionen `setup()` und `draw()` verwendet. Diese beiden Funktionen werden benötigt, um eine Skizze aufzubauen. In `setup()` werden in der Regel die Größe, bei einer Animation die Framerate, Hintergrundfarbe oder weitere Eigenschaften definiert, die einmal festgelegt sein sollten, damit die Skizze von Anfang an korrekt gezeichnet wird.

Wie erwähnt wird in `setup()` die Größe der Skizze festgelegt. Dafür wird die Funktion `size()` verwendet. `size()` hat Parameter für die Höhe und Breite und zusätzlich noch die Option den *Kontext* der Skizze. Dieser Kontext legt fest, ob die Skizze zwei- oder dreidimensional ist. Für den zweidimensionalen Kontext ist es nicht zwingend notwendig, den Parameter anzugeben. Sollte man allerdings für den dreidimensionalen Kontext den Parameter vergessen,

so werden dreidimensionale Objekte nicht gezeichnet. Als Parameter für den dreidimensionalen Kontext sind beispielsweise P3D und OPENGGL möglich. (Siehe auch 4)

In `draw()` werden jene Definitionen gemacht, die ein regelmäßiges Update benötigen, um wie gewünscht angezeigt zu werden.

```
1 setup(){
2     size(400, 400, P3D);
3     ...
4 }
```

Beispiel 4: `setup()`-Funktion

2.1.2 Würfel in Processing und Processing.js

Das zentrale Objekt, welches in der Applikation zur Verwendung kommt, ist der dreidimensionale Würfel (5).

```
1 box(10);
2 box(20, 10, 30);
```

Beispiel 5: `box()`-Funktion

Die `box()`-Funktion lässt einen Würfel oder auch einen Quader zeichnen, je nachdem, ob ein oder drei Parameter übergeben werden. Für weitere Änderungen am Erscheinungsbild des Würfels werden die Funktionen `translate()`, `rotateX()`, `rotateY()` und `rotateZ()` benötigt.

Die `translate()`-Funktion nimmt im dreidimensionalen Raum die Transformation in x-, y- und z-Richtung entgegen. Die Rotationsfunktionen (`rotateX()`, `rotateY()`, `rotateZ()`) lassen das Objekt auf den jeweiligen Achsen rotieren. Diese Modellierungen gelten ohne weitere Definitionen für alle Objekte. Um Objekte unabhängig von den Modellierungen anderer Objekte zu machen müssen die Funktionen `pushMatrix()` und `popMatrix()` verwendet werden. Diese Funktionen spalten die Modellierungen von einander ab (siehe 6).

```
1 pushMatrix();
2 fill(200, 120, 190);
3 translate(200, 120, 0);
4 rotateY(0.5);
5 rotateX(0.5);
6 box(60);
7 popMatrix();
```

Beispiel 6: Modellierungsfunktionen

2.2 Dokumentation der REST-Schnittstelle

Die REST-Schnittstelle stellt folgende Methoden und URLs bereit, die jeweils mit der Struktur des Request-Pfades und der Antwort angegeben werden.

GET /cubes

```
1 [{ "_id" : "58774fa2bb8bdd1faaf2f352",  
2   "alias" : "cubey",  
3   "boxX" : "267",  
4   "boxY" : "95",  
5   "cubeSize" : "69",  
6   "rotX" : "0.89",  
7   "rotY" : "4.76"  
8 }, {  
9   "_id" : "587b5bf2dad06947f5b9e0b7",  
10  "alias" : "cube2",  
11  "boxX" : 100,  
12  "boxY" : 100,  
13  "cubeSize" : 90,  
14  "rotX" : 0.5,  
15  "rotY" : 0.5  
16 },  
17 { ... }, ... ]
```

Beispiel 7: Response-Body der GET-Operation auf "/cubes"

Diese Methode gibt eine Liste von allen gespeicherten Würfel-Objekten zurück. Diese bestehen jeweils aus einer eindeutigen Id und den Attributen des Würfels. Die Id wird von der Datenbank zum Zeitpunkt der Erstellung vergeben. Die Id wird im diesem Projekt nicht verwendet, aber die Option das Projekt um eine Methode zum Abruf eines bestimmten Würfels zu erweitern ist somit gegeben.

POST /cubes

```
1 { "alias" : "testcube",  
2   "boxX" : 100,  
3   "boxY" : 100,
```



```

4   "cubeSize" : 90,
5   "rotX"    : 0.5,
6   "rotY"    : 0.5
7   }

```

Beispiel 8: Request-Body der POST-Operation auf ”/cubes”

In der POST-Methode wird ein Würfel-Objekt übergeben, in welchem noch keine Id festgelegt ist. Diese wird wie oben beschrieben von der Datenbank festgelegt. Die Antwort ist leer und durch den Statuscode 201 wird signalisiert, dass die Erstellung erfolgreich war.

GET /cubeSketches

```

1  [
2    {
3      "_id": "588f01f9bb2430292720f665",
4      "cubes": [
5        {
6          "alias": "genericCubeAlias",
7          "boxX": "99",
8          "boxY": "204",
9          "cubeSize": "55",
10         "rotX": "3.38",
11         "rotY": 0.5
12       },
13       {
14         "alias": "genericCubeAlias",
15         "boxX": "81",
16         "boxY": "85",
17         "cubeSize": "55",
18         "rotX": "3.38",
19         "rotY": 0.5
20       },
21       {
22         "alias": "genericCubeAlias",
23         "boxX": "225",
24         "boxY": "85",
25         "cubeSize": "55",
26         "rotX": "3.38",
27         "rotY": 0.5
28       }
29     ]

```

```

30 },
31 {
32   "_id": "5892422d89fc95049f562c69",
33   "cubes": [
34     {
35       "alias": "genericCubeAlias",
36       "boxX": "99",
37       "boxY": "204",
38       "cubeSize": "55",
39       "rotX": "3.38",
40       "rotY": 0.5
41     }
42   ]
43 }
44 ]

```

Beispiel 9: Response-Body der POST-Operation auf `"/cubeSketches"`

Wie in im Beispiel 9 zusehen ist, besteht die Antwort aus einer in sich verschachtelten Struktur. Die äußerste Ebene ist eine Liste von Sketch-Objekten. Diese haben wiederum jeweils eine eindeutige Id und eine Liste von Würfeln als Attribute. Die Liste von Würfeln enthält Würfel-Objekte, ähnlich wie bei der Antwort der GET-Methode von `"/cubes"`, wobei die Würfel hier keine Id besitzen.

POST `/cubeSketches`

```

1 {"cubes" : [
2   {
3     "alias" : "testcube",
4     "boxX" : "257",
5     "boxY" : "102",
6     "cubeSize" : 90,
7     "rotX" : "3.6",
8     "rotY" : "1.0"
9   },
10  {
11    "alias" : "testcube",
12    "boxX" : 100,
13    "boxY" : 100,
14    "cubeSize" : 90,
15    "rotX" : 0.5,
16    "rotY" : 0.5

```

```

17 | },
18 | {
19 |   "alias" : "testcube",
20 |   "boxX" : "99",
21 |   "boxY" : "102",
22 |   "cubeSize" : 90,
23 |   "rotX" : "3.6",
24 |   "rotY" : "4.54"
25 | }
26 | ]
27 | }

```

Beispiel 10: Request-Body der POST-Operation auf `"/cubeSketches"`

Die Methode `POST /cubeSketches` funktioniert analog zu `post cubes`. Das Objekt hat ein einziges Attribut `"cubes"`, welches eine Liste von einzelnen Würfel-Objekten zugewiesen hat. Die Id für ein solches Objekt wird wieder von der Datenbank hinzugefügt. Die zusätzliche Verschachtelung in einem weiteren Objekt ist notwendig, da die Datenbank eine Speicherung einer Liste von Objekten nicht ermöglicht.

2.3 Serverseitige Implementierung

Die REST-Schnittstelle ist unter Verwendung von *Node.js* und *Express.js* implementiert worden. Sie besteht aus zwei Komponenten. Die eine Komponente ist der Controller (`app.js`), in dem die REST-Methoden implementiert sind und gleichzeitig das Routing definiert ist. Die andere Komponente ist das Datenbankmodul, das unter Verwendung des MongoDB-Clients für *Node.js* die benötigten CRUD-Operationen bereitstellt. In diesem Projekt sind es konkret nur das Lesen aller Objekte und das Erstellen eines neuen Objekts in der zugehörigen Collection. Die Namen der jeweiligen Collections sind statisch und global über das gesamte Projekt definiert.

Es kann ein einzelner Würfel gespeichert werden oder eine Skizze, die eine Liste von Würfeln enthält. In `storage.js` werden diese Such- und Schreibabfragen implementiert (??). Aufgerufen werden die Datenbank-Operationen in `app.js` (siehe Beispiel 12). Sowohl die `findAll`-Methode als auch die `create`-Methode erhalten den Parameter `collectionName`. Diese Bestimmen, in welche Collection die Werte geschrieben werden, da es wie erwähnt in der Applikation möglich ist einzelne Würfel-Einstellungen oder ganze Skizzen bestehend aus einem oder mehreren Würfeln zu speichern.

```

1 [...]
2 //find list
3 Database.prototype.findAll = function(collectionName){
4     return mydb.collection(collectionName).find({});
5 }
6 //create
7 Database.prototype.create = function(cubeObj, collectionName)
8     {
9     mydb.collection(collectionName).insertOne(cubeObj);
10 };
11 [...]

```

Beispiel 11: CRUD-Operationen aus storage.js

```

1 [...]
2 app.get('/cubeSketches', function(req, res) {
3     console.log("Selecting all sketches");
4     cubeStorage.findAll('sketches').toArray().then(function(
5     arr){
6         res.send(arr);
7     });
8 });
9 app.post('/cubeSketches', function(req, res){
10     console.log("Creating new sketch entry");
11     cubeStorage.create(req.body, 'sketches');
12     res.status(201).end();
13 });
14 [...]

```

Beispiel 12: Request-Routing und Implementierung der REST-Methoden aus app.js

2.4 Datenbankmodell

Die verwendete Datenbank heißt MongoDB und ist eine NoSQL-Datenbank. Die Modelle, die bei diesem Projekt zum Einsatz kommen sind das "Document Model" und das "Multivalue model".

Die Datenbank enthält zwei Collections, eine für einzelne Würfel-Objekte und eine weitere für Listen von Würfelobjekten. Diese Aufteilung hat sich

aus der Projektentwicklung ergeben, in welcher anfangs nur das einzelne Würfel-Objekt existierte.

_id	alias	boxX	boxY	cubeSize	rotX	rotY
111	someCube	10	200	54	0.5	0.4
135	...					
...	...					

Table 1: Aufbau der Würfel-Collection

_id	cubes
123	[{ cube1 }, { cube2 }, { cube3 }]
124	[{ cube1 }]
349	[...]
...	...

Table 2: Aufbau der Skizzen-Collection

3 Entwicklungs- und Umsetzungsphase

Das Projekt habe ich aufbauend auf einer im HTML eingebundenen Processing-Skizze strukturiert. Zuerst habe ich ein Objekt gesucht, das sich mit nicht all zu komplexen Mitteln manipulieren lässt. Die Koordinaten und Drehung eines dreidimensionalen Würfels haben sich dafür als gut geeignet erwiesen. Da die Client-Seite bereits durch JavaScript beziehungsweise auch Processing dominiert wurde, wollte ich auch auf der Server-Seite ein Framework in JavaScript verwenden, um beidseitig mit der selben Sprache zu arbeiten.

Das Framework habe ich schließlich nach seiner Flexibilität und Schlankheit ausgewählt, da ich für die Umsetzung kein großes, umfangreiches Framework brauchte. Es werden keine object-relationale Mapper implementiert, sondern die Daten werden direkt als JSON-Dokumente abgelegt.

Processing kannte ich zuvor schon und habe es hauptsächlich in seiner eigenen Desktopanwendung verwendet. Für dieses Projekt wollte ich Processing.js ausprobieren und es in einem für andere Nutzer interaktiven Rahmen zur Anwendung bringen.

Client-seitig habe ich nur *jQuery* und das *Document-Object-Model* verwendet, da alle außerhalb des Processing-Codes implementierten Funktionen kein Framework erforderten. Der komplexere Teil der Implementierung befindet sich im Processing-Code. Sollte die Applikation jedoch größer werden und mehr Features benötigen, kann sie beispielsweise auch mittels eines Frameworks erweitert werden.

3.1 Definition der Zielgruppe

Da die Processing-Webapp ohne Login verfügbar ist, soll sie generell allen interessierten Personen einen Zugang zu Processing bieten. Zum einen ist die Applikation von Personen ohne Programmierkenntnisse gut zu verstehen und zu bedienen, zum anderen können auch Personen mit Vorkenntnissen in der Programmierung sich Beispiele zu den Möglichkeiten mit Processing ansehen.

Sollte die Processing-Webapp erweitert werden, kann sie außerdem auch kreativen Personen dazu dienen, Bilder mit Processing zu erstellen, ohne Code schreiben zu müssen. Die Speicherung der Skizzen ermöglicht zusätzlich eine Erreichbarkeit von Personen, die nach Inspiration suchen.

3.2 Projektziele

Zu Beginn des Projekts wurden einige Ziele festgelegt. Unter anderem sollte die Benutzung der Applikation ohne Erstellung eines Kontos, also ohne Login, möglich sein. Eine Erweiterung um zusätzliche Inhalte, die nur mit Login zur Verfügung stehen, ist nicht ausgeschlossen.

Außerdem sollte es die Option zum Editieren, Speichern und Veröffentlichen von Processing-Objekten geben. Es hat sich während der Umsetzung auch noch das Ziel ergeben, dass nicht nur einzelne Objekte, sondern auch Skizzen bestehend aus mehreren Objekten erstellt und gespeichert werden können.

Zu Beginn des Projekts waren die User-Stories folgendermaßen definiert:

- Der Nutzer sieht eine Processing-Canvas und kann mit dieser inter-

agieren. (Zum Beispiel indem er das Aussehen eines angezeigten Objekts ändern kann.)

- Der Nutzer kann individuelle Objekte beziehungsweise deren Einstellungen in einer serverseitigen Datenbank ablegen.
- Der Nutzer kann individuelle Objekt-Einstellungen aus einer Liste abrufen, welche auch von anderen Nutzern erstellt wurde.

Im folgenden Abschnitt werden die umgesetzten Nutzerinteraktionen im Detail beschrieben.

3.3 Nutzerinteraktionen

Der Nutzer erreicht die Processing-Webapp direkt durch das Aufrufen der HTML-Seite. Das Hauptinteraktionselement, das Canvas-Element (siehe Abbildung 2), ist direkt oben links auf der Seite zu sehen. Es enthält anfangs ein Würfel-Objekt und kann mittels der Regler, welche rechts von dem Canvas-Element positioniert sind, verändert werden. Der Nutzer kann die Attribute der Würfels variieren und wenn er damit zufrieden ist, entweder direkt in der Würfel-Objekte-Liste ablegen oder zuerst dessen Einstellungen speichern und einen weiteren Würfel positionieren. Die gesamte Skizze kann der Nutzer ebenfalls speichern.

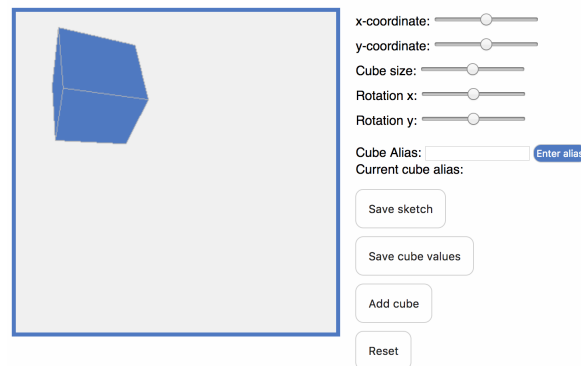


Figure 2: Aufbau

In der anderen Richtung kann der Nutzer sich auch aus den Listen, die sich unter dem Canvas-Element befinden, gespeicherte Würfel oder Skizzen

anzeigen lassen. Diese werden mit einem Klick auf den jeweiligen Button in dem Canvas-Element angezeigt.

Sollte ein Nutzer mehrere Würfel-Objekte erstellt haben, aber nicht zufrieden sein, kann er diese mit dem "Reset"-Button löschen. Ein flexibler Würfel bleibt dabei erhalten.

4 Abschlussphase

Am Ende des ersten Teils des Projekts konnte gleichzeitig ein einzelner Würfel modelliert, gespeichert und wieder aufgerufen werden. Diese Interaktion sollte ausgebaut werden und um das Feature, mehrere Würfel gleichzeitig darzustellen und modellieren zu können, erweitert werden. An diesem Punkt stellte sich das Framework als sehr flexibel heraus. Die wichtigste strukturelle Anpassung war es, dass Objekte von mehreren Würfeln nochmals in einem JSON-Objekt verpackt werden mussten, um sie in der Datenbank zu speichern.

5 Fazit

erstes serverseitiges projekt mehr erfahrung clientseitig mongodb hat mir gefallen, weil ich direkt objekte speichern konnte ohne vorher ein schema anlegen konnte selbes objekt "quasi" in datenbank, serverseitig und clientseitig Dank der Objektdatenbank wird kein Objekt-relationales Mapping benötigt.

5.1 Ausblick

Da dieses Projekt noch sehr grundlegend ist, bieten sich an verschiedenen Stellen noch Erweiterungsmöglichkeiten an. Es könnten noch andere unterschiedliche Objekte für die Komposition der Skizze zur Verfügung gestellt werden. Dies geht natürlich auch mit einer komplizierteren Struktur der Objekte einher. Für die Nutzerseite würden sich auch das Generieren und Anzeigen von Thumbnails der gespeicherten Skizzen anbieten, um die Bedienung zu erleichtern. Interessant wäre auch die Erweiterung um einen privaten Bereich, der per Login erreicht werden kann, sodass Nutzer ihre eigenen Skizzen gesammelt betrachten können oder einzelne davon löschen können.

References

- [1] Processing.js, <http://processingjs.org/>, 02.02.2017.
- [2] Processing, <https://processing.org/>, 02.02.2017.
- [3] MongoDB, <https://www.mongodb.com/>, 02.02.2017.
- [4] MongoDB, <https://docs.mongodb.com/manual/reference/method/>, 02.02.2017.
- [5] Express.js, <https://expressjs.com/>, 02.02.2017.