Hochschule der Medien Stuttgart

# GrEdit – Grafikeditor Web-App

## Dokumentation


| **Autoren** | **Datum** | **Betreuer** |
| --- | --- | --- |
| Dominik Herbst<br>Niclas Steigelmann | 30.07.2013 | Prof. Dr. Fridtjof Toenniessen |


# Inhalt


1. Einführung ..... 2
   1.1. Idee ..... 2
   1.2. Einsatzzweck ..... 2
2. Grafikformat ..... 3
3. Rendering-Prozess ..... 4
4. Editoraufbau ..... 5
   4.1. Komponenten ..... 5
   4.2. Arbeitsfläche ..... 5
   4.3. Objektbaum ..... 6
   4.4. Eigenschaften ..... 6
   4.5. Color Picker ..... 6
   4.6. Ressourcen Manager ..... 6
   4.7. Werkzeuge ..... 7
   4.8. Shapes ..... 7
   4.9. Save / Save as ..... 8
   4.10. Print ..... 8
   4.11. Export ..... 8
5. Bedienung ..... 8
   5.1. Neue Grafik erstellen ..... 8
   5.2. Objekte einfügen und anpassen ..... 8
   5.3. Objekte verwalten ..... 9
   5.4. Grafiken speichern, laden und löschen ..... 10
6. Von der Idee zur tatsächlichen Anwendung ..... 10
   6.1. Definition des Produkts ..... 10
   6.2. Die Wahl der Mittel ..... 11
   6.3. Meilensteine ..... 12
   6.4. Arbeiten mit JSWDF ..... 13
   6.5. Layout und Logo ..... 13
   6.6. Stolpersteine ..... 14
   6.7. Entwicklungszeit ..... 14
   6.8. Aktueller Stand ..... 15
   6.9. Ausblick ..... 15
   6.10. Geschäftsmodell ..... 16

# 1. Einführung

Diese Dokumentation beschreibt den GrEdit Grafikeditor von der Idee über die Bedienung bis zu den technischen Hintergründen und gibt einen Einblick in die Entstehungsgeschichte.

GrEdit wurde von uns im Rahmen des Studiengangs Medieninformatik an der Hochschule der Medien entwickelt.

Dominik Herbst ist ausgebildeter Fachinformatiker und entwickelt seit 9 Jahren Internetseiten und Backends für Websysteme.

Niclas Steigelmann ist ausgebildeter Mediengestalter für Printprodukte, hat die Internetabteilung einer Zeitung mit aufgebaut und entwickelt und wartet seit 15 Jahren im Auftrag von verschiedenen Kunden Internetseiten und dazugehörige Systeme.

Somit konnten wir beide bereits vorher Erfahrungen im Umgang mit Webtechnologien wie HTML, CSS, JavaScript und PHP sammeln. Durch diese Vorkenntnisse befähigt, haben wir uns an dieses komplexe Projekt gewagt.

## 1.1. Idee

Mit einem webbasierten grafischen Editor sollen Grafiken bzw. Grafikkompositionen erstellt werden können, die durch dynamische Daten beeinflusst werden. Diese Daten stammen aus unterschiedlichsten Quellen, die für die Verwendung in den Grafiken aufbereitet werden.

In den Grafiken beeinflussen die Daten z.B. die Form, den Text oder die Farbe einzelner Grafikobjekte. Eine Grafik ist somit dynamisch und repräsentiert immer den aktuellen Stand der verknüpften Datenquellen.

Zur weiteren Verwendung, zum Beispiel auf Webseiten im Internet, sollen die Grafiken in diversen Formaten wie PNG veröffentlicht  und exportiert werden können.

## 1.2. Einsatzzweck

Besonders bei Spiele-Communities im Internet ist das Phänomen weit verbreitet, dass Spieler ihre erreichten Leistungen aus einem Spiel in Form von Grafiken in Foren und Webseiten einbinden. Die Anbieter dieser Grafiken lassen zu, dass die Grafiken nach dem eigenen Geschmack angepasst werden. Dies ist jedoch mit einem hohen Aufwand verbunden. Hier soll GrEdit mit seiner grafischen Bearbeitungsoberfläche ansetzen. Angedacht ist, dass GrEdit eine Anbindung an die Daten des Spielers erhält und diese in Form einer Grafik darstellen kann. Wenn sich die Daten des Spielers ändern, wird dies beim nächsten Rendering der Grafik erkannt und die aktuellen Daten werden verwendet.

Bei diesen Daten handelt es sich meist um Zahlen – unter anderem die Anzahl erreichter Punkte oder die Spielzeit. Diese sollen in GrEdit als Text in die Grafik eingebunden werden können, oder das Aussehen eines Grafikobjektes (Maße, Farbe, o.ä.) definieren. Es kann sich jedoch z.B.  auch um das Lieblingsauto des Spielers handeln, das als Bild eingebunden werden soll.

## 2. Grafikformat

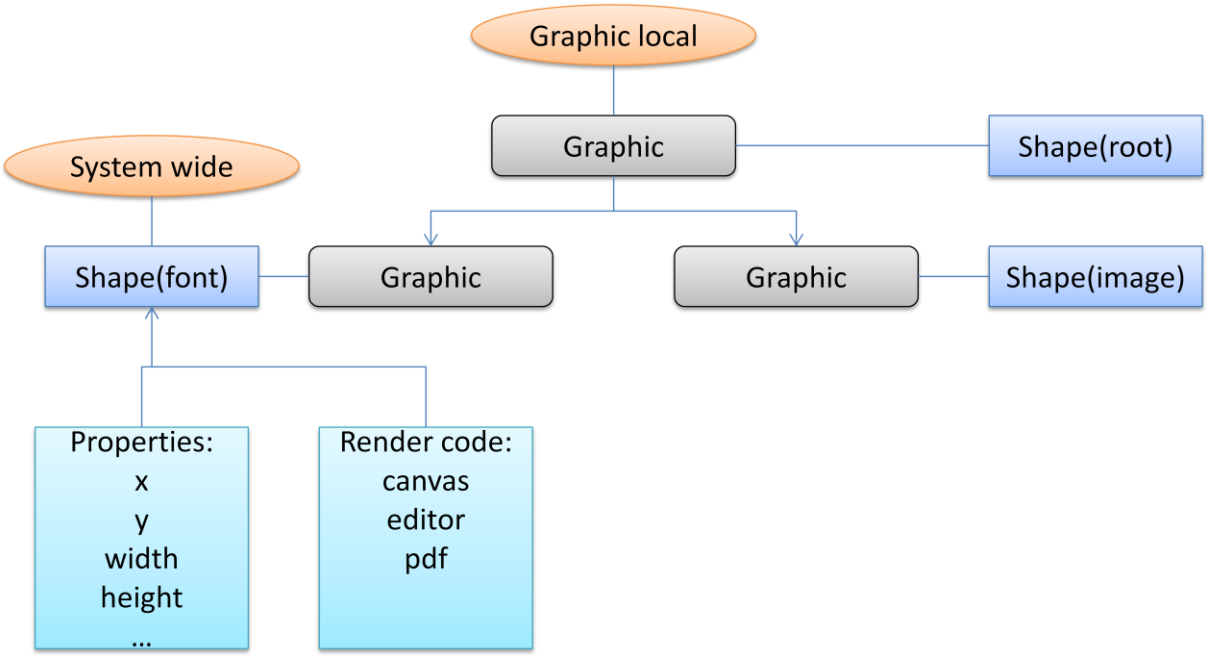



Abbildung 1: Aufbau der Grafikarchitektur

Eine Grafik besteht aus einzelnen Objekten, die als Knoten in einem Baum repräsentiert sind. Jedes Objekt ist mit X,Y-Koordinaten auf der Ausgabefläche positioniert. Dabei sind Kind-Objekte von den Koordinaten der Elternobjekte abhängig. Wird z.B. ein Eltern-Objekt verschoben, verschieben sich alle Kinder auf der Sichtfläche um denselben Abstand.

Das Shape definiert, um was für ein Grafikobjekt es sich handelt. Als Beispiele seien hier Text, Rechteck, Kreis  oder schlicht ein Bild genannt. Da jedes Grafikobjekt andere Parameter benötigt, werden diese vom Shape geliefert. So benötigt ein Rechteck Eigenschaften wie X- und Y-Position, Breite, Höhe und Winkel. Ein Bild benötigt zusätzlich noch die Bildquelle und den Bildausschnitt der verwendet werden soll.

Jedes Grafikobjekt gehört zu einem Shape und speichert die für sich relevanten Eigenschaften, die vom Shape vorgegeben werden.

Die Baumstruktur der Grafikobjekte wird zu JSON serialisiert, um die Grafik zu speichern. Der JSON-String kann dazu verwendet werden, den Grafikobjektbaum in einer JavaScript-Runtime aufzubauen.

## 3. Rendering-Prozess

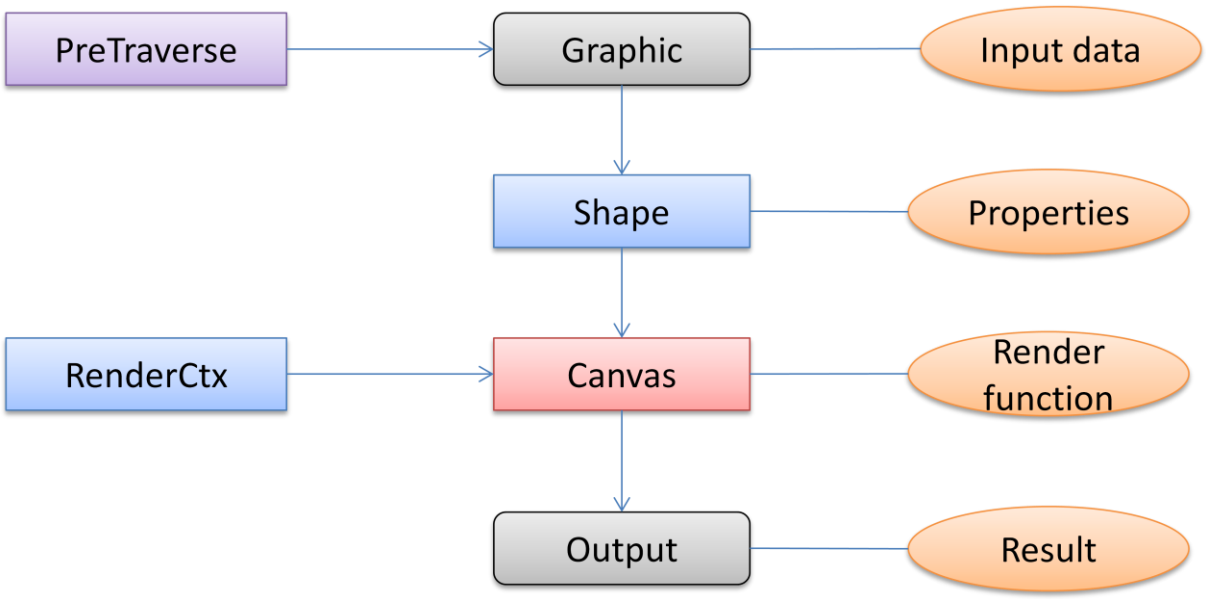



Abbildung 2: Render-Pipeline

Bevor der Render-Prozess startet, wird ein Render-Kontext erzeugt. Dieser stellt verschiedene Einstellungen und APIs bereit, auf die beim Rendering zugegriffen wird. Dies beinhaltet zum Beispiel welche Ausgabe verwendet wird, oder mit welchem Skalierungsfaktor gerendert werden soll.

Für das Rendering der Grafik wird der Grafikbaum in Pre-Order traversiert. Hierbei wird der Wurzelknoten des Baumes betrachtet und anschließend die Kinder von links nach rechts durchlaufen. Jedes Grafikobjekt ist ein Knoten zu dem ein Shape gehört. Das Shape besitzt für jede Ausgabe eine Renderfunktion, z.B. eine Ausgabe über das HTML-Canvas-Element. Beim Aufruf der Renderfunktion werden ihr der Render-Kontext, das Grafikobjekt und das Elternobjekt übergeben. Die Renderfunktion zeichnet das Objekt dann auf die Ausgabe. Später gerenderte Objekte überzeichnen hierbei früher gerenderte Objekte.

Der Wurzelknoten legt mit seiner Breite und Höhe die Größe der Grafik fest. Er wird im Objektbaum des Grafikeditors jedoch nicht als Knoten angezeigt, da er immer existiert und nicht gelöscht oder verschoben werden darf.

Wenn eine neue Ausgabe hinzugefügt werden soll, müssen alle Shapes, die in einer Grafik für die neue Ausgabe verwendet werden, eine weitere Renderfunktion bekommen. So kann es vorkommen, dass manche Shapes nicht für jede Ausgabe geeignet sind, solange keine entsprechende Renderfunktion geschrieben wurde.

# 4. Editoraufbau

Der Editor besteht aus verschiedenen Komponenten. Diese sind in JavaScript implementiert. Zur Ausgabe im Web-Browser wird HTML5 und CSS verwendet.

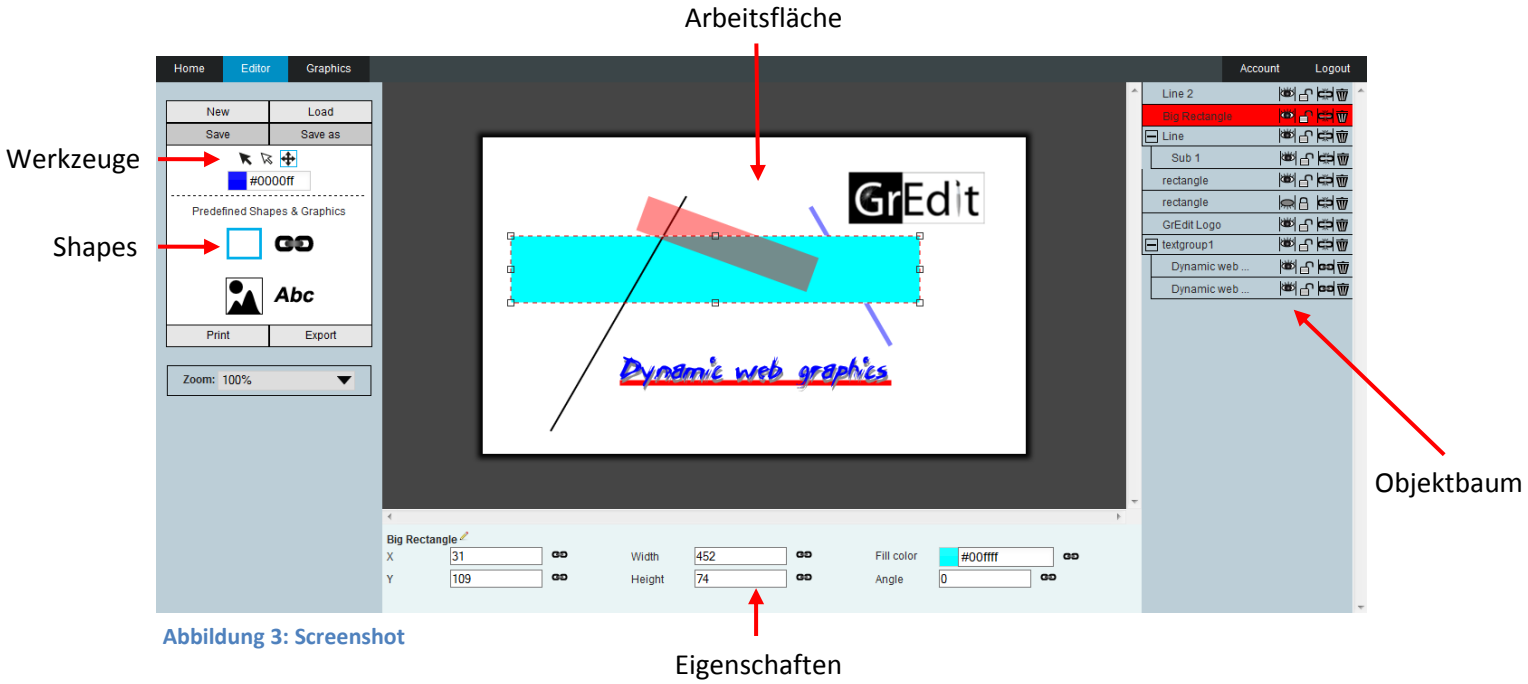



Abbildung 3: Screenshot

## 4.1. Komponenten

Jede einzelne Komponente hat einen globalen Namen und befindet sich zur besseren Aufteilung und Verwaltung in einer eigenen JavaScript-Datei. Es existiert immer eine zentrale Editor-Komponente, welche die anderen Komponenten aufruft und verwaltet. So werden durch die Erstellung einer Editor-Instanz alle Komponenten eingebunden und im Web-Browser dargestellt. Um auf die Benutzereingaben komponentenübergreifend reagieren zu können, müssen die Komponenten miteinander kommunizieren und sich gegenseitig aktualisieren können. Dies geschieht über die Editor-Komponente.

## 4.2. Arbeitsfläche

Die Arbeitsfläche ist der mittlere Bereich der Anwendung, in dem die Grafik an sich dargestellt wird und direkt bearbeitet werden kann. Diese Komponente überwacht Tastatur- und Mauseingaben um Objekte zu markieren oder zu bearbeiten.

Für Elemente wie Hilfslinien und Bearbeitungspunkte liegt über der eigentlichen Grafik eine weitere Zeichenfläche. Diese kann besonders schnell gerendert werden und sorgt so für ein flüssiges Arbeitsgefühl.

Mit der Zoom-Einstellung auf der linken Seite des Editors kann die Arbeitsfläche skaliert werden. Bei der Skalierung der Arbeitsfläche werden alle Objekte dem Skalierungsfaktor entsprechend vergrößert oder verkleinert. Desweiteren wird die Position entsprechend angepasst.

## 4.3. Objektbaum

Auf der rechten Seite des Editors befindet sich die Objektbaumdarstellung. Diese zeigt wie die Objekte im Baum angeordnet sind. Hier können Objekte markiert und verwaltet werden.

Für jedes Objekt gibt es folgende Optionen:

 *Ein/Ausblenden:* legt fest, ob das sichtbar ist und somit gerendert wird, oder nicht.

 *Sperren/Entsperren:* legt fest, ob das Objekt markiert und verändert werden kann, oder nicht.

 *Gruppieren:* kann verwendet werden, um mehrere markierte Elemente zu gruppieren.

 *Löschen:* entfernt das Objekt und seine Kinder.

Abhängig davon welches Auswahlwerkzeug gewählt wurde, werden einzelne Objekte oder ganze Objektgruppen ausgewählt. Durch das halten der Strg- oder Umschalt-Taste können mehrere Objekte markiert werden.

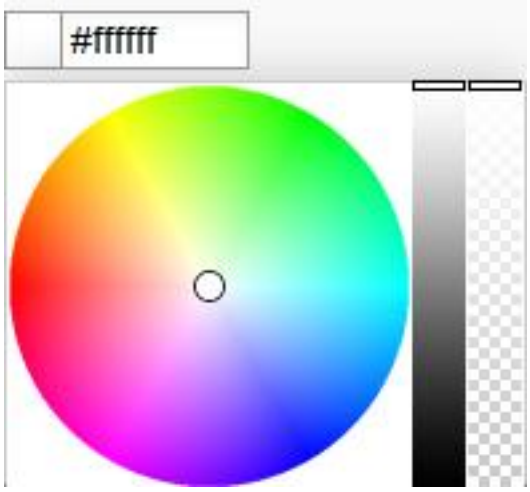
Per Drag & Drop kann die Reihenfolge der Objekte verändert werden. Positioniert man Objekte genau übereinander, können diese gruppiert werden.

## 4.4. Eigenschaften

Im unteren Bereich befindet sich die Eigenschaften-Leiste. Hier werden kontextbezogen Eigenschaften angezeigt, die der Benutzer für das angewählte Objekt ändern kann. Welche Eigenschaften angezeigt und wie diese gruppiert werden, wird von dem jeweiligen Shape festgelegt.

Verschiedene Typen von Eigenschaften besitzen unterschiedliche Eingabemöglichkeiten. Zahlen und Zeichenketten können einfach eingetippt werden. Farben können mit Hilfe eines Color Pickers gewählt werden. Ressourcen wie Bilder und Fonts können mit dem Ressourcen Manager ausgewählt werden.

## 4.5. Color Picker




Der Color Picker bietet vier Ansatzpunkte.

Wie in HTML und CSS lässt sich im Eingabefeld oben ein Hexwert eingeben, der die Farbe definiert.

Die eigentliche Stärke des Color Pickers stellt der Farbkreis dar, in dem man die gewünschte Farbe wählen kann. Anschließend lässt sich mit dem Slider in der Mitte des Auswahlmenüs die Farbintensität einstellen. Der rechte Slider bietet schließlich noch die Möglichkeit, den Transparenzgrad zu verändern.

## 4.6. Ressourcen Manager

Der Ressourcen Manager kann über die Objekt-Eigenschaften aufgerufen werden, sofern diese die Einbindung externer Ressourcen unterstützen. Er dient dazu, Dateien wie Bilder und Schriftarten zu verwalten. Für jeden Ressourcentyp gibt es hierfür einen separaten Tab.

Wenn der Benutzer eingeloggt ist, kann er durch einen Klick auf Upload neue Dateien hochladen. Um eine größtmögliche Browserkompatibilität zu gewährleisten, verwendet der Uploader die HTML5 File API. Dieses bietet unter anderem auch die Möglichkeit Fortschrittsbalken anzuzeigen und mehrere Dateien gleichzeitig hochzuladen.

## 4.7. Werkzeuge

Auf der linken Seite der Anwendung finden sich verschiedene Werkzeuge. Diese legen fest, wie mit Objekten auf der Arbeitsfläche, aber auch im Objektbaum interagiert werden kann.

###  Auswahl

Mit dem Auswahl-Werkzeug können nicht gesperrte Objekte im Arbeitsbereich ausgewählt und verschoben werden. Bei der Auswahl eines Objekts, das Teil einer Gruppe ist, wird immer die gesamte Gruppe ausgewählt.

###  Einzelauswahl

Das Einzelauswahl-Werkzeug kann einzelne Objekte im Arbeitsbereich und im Objektbaum auswählen und verschieben, unabhängig davon, wie diese gruppiert sind.

###  Bearbeitung

Das Bearbeitungs- oder Transformationswerkzeug verhält sich wie das Auswahlwerkzeug, bietet jedoch zusätzlich Skalierungspunkte auf der Arbeitsfläche. Mit diesen kann die Größe des Objekts skaliert werden.

### Standardfarbe


Hier kann die Farbe festgelegt werden, die beim Erzeugen von neuen Form- und Textshapes verwendet wird. Über die Eigenschaften eines Objektes kann diese auch nachträglich noch geändert werden.

## 4.8. Shapes

Unterhalb der Werkzeuge findet sich eine Übersicht der Shapes, die in die Grafik eingefügt werden können. Durch Klick auf ein Shape wird davon ein Objekt mit Standardwerten erstellt und kann auf der Arbeitsfläche frei positioniert werden.

Zur Verfügung stehen Form-Shapes, Image-Shape, Text-Shape und Group-Shape.

### Form-Shapes

Form-Shapes erzeugen geometrische Grundformen. In dieser Version wurde nur das Rechteck-Shape umgesetzt.

### Image-Shape

Das Image-Shape bietet dem User die Möglichkeit, eigene Bilder über den Ressourcen Manager hochzuladen.

#### Text-Shape


Mit dem Text-Shape können eigene Texte eingegeben werden. Über den Ressourcen Manager können hierfür eigene Schriftarten zur weiteren Personalisierung hochgeladen werden.

#### Group-Shape

Das Group-Shape bietet die Möglichkeit, ein leeres Gruppenobjekt zu erzeugen. In dieses können dann andere Objekte als Kindobjekte eingefügt werden.

### 4.9. Save / Save as

Um seine Grafik speichern zu können, muss man eingeloggt sein. Andernfalls wäre keine spätere Zuordnung mehr möglich und man müsste die Grafik für jeden frei veränderbar online stellen.

Allgemein orientiert sich der Speichervorgang zur besseren Usability an bekannten Standards. So wird beim ersten Speichervorgang nach dem zu verwendenden Namen gefragt. Bei weiteren Malen erscheint nur noch eine kurze Einblendung, ob der Speichervorgang erfolgreich war. Um eine bestehende Grafik unter einem neuen Namen abzulegen, verwendet man den Save as-Dialog.

### 4.10. Print

Die Druckfunktion erstellt ein PNG der sichtbaren Grafik und ruft den Druckdialog des Browsers auf.

### 4.11. Export

Bei einem Klick auf die Export-Schaltfläche öffnet sich die aktuell im Editor sichtbare Grafik als PNG in einem neuen Browser-Tab. Dort kann diese zum Beispiel auf der eigenen Festplatte gespeichert werden.

## 5. Bedienung

Die Bedienung des Editors wurde möglichst einfach und intuitiv gestaltet. Mit wenigen Schritten lassen sich bereits erste Ergebnisse erzielen.  Um alle Features ohne Einschränkungen nutzen zu können, empfiehlt es sich von Beginn an eingeloggt zu sein, da sonst unter anderem ein Speichervorgang nicht möglich ist.

### 5.1. Neue Grafik erstellen

Eine neue Grafik wird bereits mit dem Aufruf des Editors erstellt. Damit steht von Anfang an eine Standardgrafik bereit die dann verändert werden kann.

Die Größe und die Hintergrundfarbe kann in den Eigenschaften angepasst werden. Die Eigenschaften der Grafik können jederzeit durch einen Klick auf den Hintergrund oder Demarkierung aller markierten Objekte aufgerufen werden. Falls keine Hintergrundfarbe gewünscht ist, kann der Hintergrund auch vollständig transparent eingestellt werden.

### 5.2. Objekte einfügen und anpassen

Durch Klick auf ein Shape wird im Speicher ein neues Objekt erzeugt. Anschließend kann das neue Objekt durch einen weiteren Klick an die gewünschte Position frei auf der Arbeitsfläche positioniert werden. Hierbei wird das Objekt in den Grafikbaum eingefügt und ist nun Teil der Grafik.

## 5.3. Objekte verwalten

Objekte können markiert, verschachtelt, gelöscht, gesperrt und versteckt werden. Diese Aufgaben können mit Hilfe des Objektbaums mühelos erledigt werden.

### Markieren

Objekte können direkt auf der Arbeitsfläche oder im Objektbaum markiert werden. Mit dem normalen Auswahlwerkzeug wird bei einem Klick auf ein Objekt immer die gesamte Gruppe markiert. Mit dem Einzelauswahlwerkzeug wird die Gruppierung ignoriert.

Durch halten der Strg- oder Umschalt-Taste können mehrere Objekte markiert werden. Alle markierten Objekte können dann zusammen per Drag & Drop auf der Arbeitsfläche verschoben werden.

### Verschieben

Per Drag & Drop können die Objekte im Objektbaum verschoben werden. Die Reihenfolge der Objekte legt fest, wann diese gerendert werden. Je weiter unten ein Objekt ist, desto später wird es gerendert. Später gerenderte Objekte verdecken dabei jene, die früher gerendert wurden.

Desweiteren ist eine pixelgenaue Verschiebung mit den Pfeiltasten möglich. Wird währenddessen die Strg-Taste gedrückt, erfolgt die Verschiebung immer um 10 Pixel.

### Verschachteln

Der Objektbaum kann dazu verwendet werden Objekte in andere Objekte zu schieben. Dazu wird ein Objekt markiert und per Drag & Drop auf das gewünschte Elternobjekt geschoben. Danach sind die Objekte Kinder des Elternobjekts. Bei diesem Vorgang werden die X, Y Koordinaten der Kindobjekte angepasst, damit diese ihre globale Position erhalten, aber relativ zu ihrem neuen Elternobjekt positioniert sind.

Verschiebt man nun das Elternobjekt, verschieben sich alle Kindobjekte ebenfalls um dasselbe Maß.

### Löschen

Im Objektbaum befindet sich hinter jedem Objekt ein Papierkorb, der als Löschen-Schaltfläche dient. Nach einem Klick auf diese Schaltfläche, wird man aufgefordert, den Löschvorgang zu bestätigen. Durch einen Klick auf „OK“ werden das Objekt und all seine Kinder aus dem Objektbaum gelöscht.

Welche Objekte dabei sonst noch markiert sind, ist irrelevant.

### Sperren

Damit Objekte nicht versehentlich verändert werden, können diese gesperrt werden. Auf der Zeichenfläche sind sie dadurch auch nicht mehr anwählbar.

Im Objektbaum befindet sich hierfür hinter jedem Objekt eine Schloss-Schaltfläche. Diese zeigt an, ob das Objekt gesperrt ist. Durch einen Klick auf die Schaltfläche lässt sich das Objekt sperren bzw. entsperren.

### Ausblenden

Soll ein Objekt nicht mehr in der Grafik gerendert werden, kann es ausgeblendet werden. Dies gilt auch immer für all seine Kind-Objekte.

Hinter jedem Objekt im Objektbaum befindet sich eine Auge-Schaltfläche. Ist das Auge offen, wird das Objekt gerendert. Ist das Auge geschlossen, wird das Objekt nicht gerendert. Durch einen Klick auf die Schaltfläche lässt sich das ändern.

## 5.4. Grafiken speichern, laden und löschen

Sobald man Grafiken dauerhaft erhalten und auch später wieder darauf zurückgreifen will, ist es zwingend notwendig, dass ein Benutzerkonto erstellt wurde. Hat man sich damit eingeloggt, kann eine Grafik in diesem gespeichert und später wieder geladen werden. In seinem Account findet man auch eine Übersicht aller gespeicherten Grafiken und kann einzelne bei Bedarf löschen.

Beim Speichervorgang wird der Grafikbaum zu JSON serialisiert und per AJAX an die Serverkomponente von GrEdit übertragen. Dort wird die Grafik in der Datenbank abgelegt.

# 6. Von der Idee zur tatsächlichen Anwendung

## 6.1. Definition des Produkts

Zu Beginn des Projekts haben wir die Idee und mehrere Konzepte ausformuliert und aufgeschrieben. Dadurch waren wir uns einig in welche Richtung das Projekt gehen sollte.

Damit man sich auch optisch eine Vorstellung machen konnte, wurden erste Scribbles und Mockups mit Adobe Illustrator erstellt.

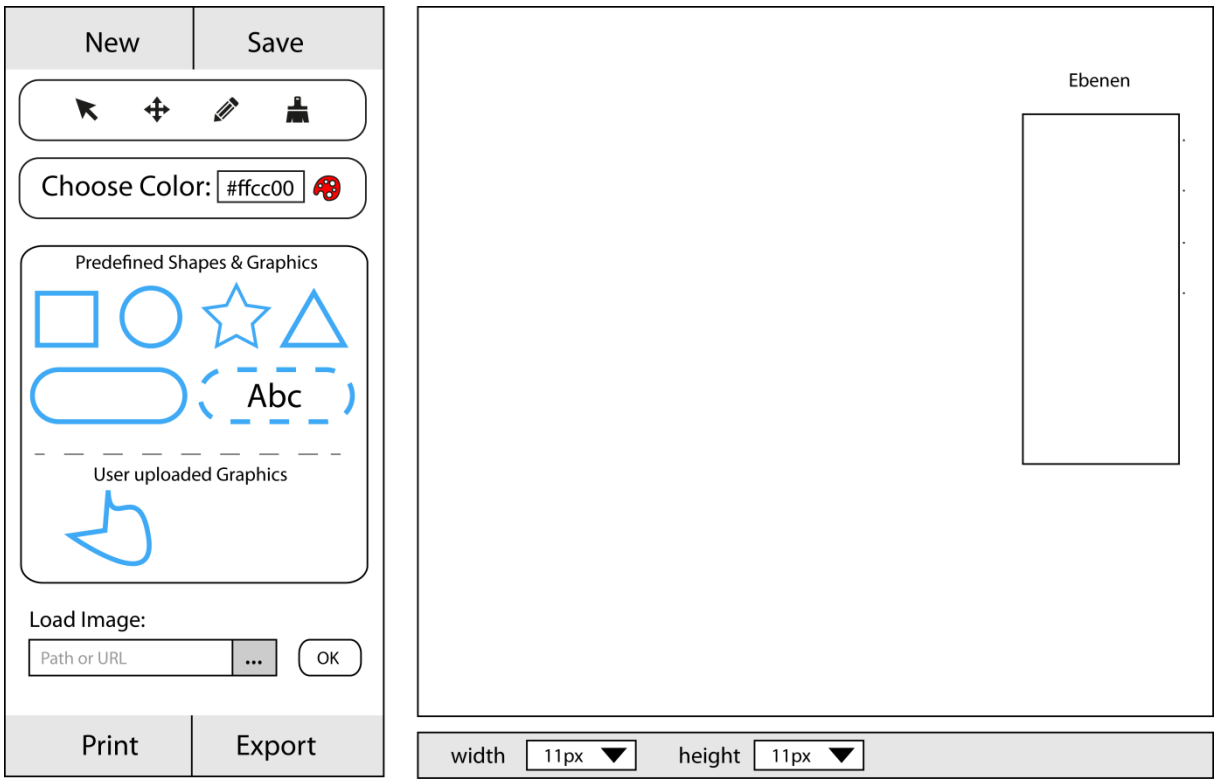



Abbildung 4: Ein frühes Mockup

## 6.2. Die Wahl der Mittel

Trotz der Fülle an Skripten, Frameworks und Tools, die sich im Internet finden lassen, haben wir uns dazu entschieden, die Anwendung grundlegend neu zu implementieren. Dies sollte uns einen tieferen Einblick in die Arbeitsweise der Anwendung ermöglichen. Des Weiteren wollten wir so verhindern, viel Entwicklungszeit für die Behebung von Kompatibilitätsproblemen zwischen den Frameworks zu verlieren.

Letztlich entschieden wir uns für folgende Komponenten:

### JSWDF als Basis

Ein von Dominik Herbst entwickeltes JavaScript Framework für die Definition der Softwarestruktur und der Zusammenarbeit zwischen Client und Server.

JSWDF (JavaScript Web Development Framework) basiert auf Node.js und bindet verschiedene Node.js Module und Frameworks für die Annahme, Verarbeitung und Beantwortung von HTTP Anfragen ein. Zudem bringt es ein Templatesystem und eine Sammlung nützlicher Komponenten mit sich.

### Node.js als Server

Mit seinem async-IO Konzept ist Node.js im Vergleich zu etablierten serverseitigen Webtechnologien wie PHP bei vielen Anfragen schneller und performanter. Das Konzept bringt zudem einen ganz anderen Programmierstil mit sich, bei dem Code nicht sequentiell abgearbeitet wird, sondern der Zeitpunkt der Ausführung bestimmter Funktionen von einer Anzahl verschiedener Faktoren abhängt.

Dieser Stil kann auf Server und Client gleichermaßen eingesetzt werden. So kommt man zu einem schnellen Ergebnis, das auch bei vielen Anfragen und komplizierten Aufgaben mit sehr geringer Hardwarelast auskommt.

### jQuery auf dem Client

Im Vergleich zu der Arbeit mit der nativen JavaScript DOM API kann man mit jQuery viel schneller und effizienter entwickeln. Zudem ist es über alle Browser hinweg kompatibel und nimmt einem die Arbeit ab, für jeden Browser Sonderlösungen entwickeln zu müssen.

### MongoDB als Datenbank

MongoDB ist eine dokumentorientierte NoSQL Datenbank, die beinahe nahtlos mit Node.js zusammenarbeitet. Die Daten werden in einem binären JSON Format abgelegt und übertragen, die Abfragesprache (Query-Sprache) basiert ebenfalls auf JSON. Zudem ist die Datenbank sehr hardwareschonend und unterstützt die wichtigsten Features von SQL Datenbanken, wie Indexierung, Replikation und Sharding (Verteilung auf mehrere Server). Sie bringt so gut wie keinen Konfigurations- oder Einrichtungsaufwand mit. Schemata oder ähnliches müssen nicht angelegt werden.

### jQuery MiniColors

Ein jQuery Plugin zur Einbindung eines Color Pickers für den Editor.

### Weitere Technologien

Für das Rendering der Grafiken wurde das HTML 5 Element Canvas verwendet. Serverseitig ist dies als Node.js Modul verfügbar.

Alles andere wurde eigenhändig programmiert bzw. im Falle der Grafiken und Icons selbst erstellt.

### Tools

Als Software kam zum Einsatz:

- JetBrains IntelliJ Idea / WebStorm (Programmierung)
- Adobe Photoshop und Illustrator (Icons und Grafiken)
- TortoiseSVN (Versionsverwaltung)

## 6.3. Meilensteine

Die wichtigsten Meilensteine im Entstehungsprozess von GrEdit

| | |
|---|---|
| **Schriftliche Fixierung der Idee**<br>(Siehe Punkt 6.1) | |
| *Technisch* | *Grafisch* |
| **Einrichtung des Frameworks**<br>Ordner erstellen, Konfiguration anlegen | **Scribbles und Mockups**<br>Skizzen auf Papier und in digitaler Form |
| **Aufbau Grafikobjekt**<br>Implementierung der Klassen | **Erster HTML-Prototyp**<br>noch quasi funktionslos, erste Tests mit JQuery |
| **Aufbau Renderstruktur**<br>Jedes Shape und jede Komponente bringen ihre Renderanweisungen mit | **Erstellung der Icons und Grafiken**<br>Die Icons wurden in Illustrator als Vektorgrafik erstellt, um später verlustfrei verschiedene Größen erzeugen zu können |
| **Zusammenführung**<br>Technische Umsetzung der grafischen Konzepte,<br>Einbinden von CSS und Icons | |
| **Erste lauffähige Version** | |
| **Erste gerenderte Grafik**<br>Der Objektbaum wurde korrekt durchlaufen und die Grafik wurde erfolgreich gerendert | |
| **Speichern und Laden**<br>Zum Speichern der Grafiken wurde eine Datenbankanbindung notwendig.<br>Für die Verknüpfung von Usern und Grafiken war ein Login System nötig. | |
| **Ressourcenmanagement**<br>Erstellung eines Ressourcen Managers als Uploadschnittstelle<br>sowie zur Verwaltung hochgeladener Dateien | |
| **Font Rendering**<br>Um einen Text rendern zu können, muss zuerst eine Schriftart hinterlegt werden.<br>Mit dem Ressourcen Manager wird diese hierzu hochgeladen.<br>Anschließend kann man aus allen hochgeladenen Schriften wählen.<br>Default-Schriftarten sind derzeit noch nicht hinterlegt. | |

## 6.4. Arbeiten mit JSWDF

Während der Großteil der Anwendung clientseitig arbeitet, galt es trotzdem auch serverseitig einiges zu beachten. Hier wurden vor allem die Voraussetzungen geschaffen, um später auf Clientseite alle nötigen Dateien und Schnittstellen zur Verfügung zu haben.

Ein Vorteil, wenn man JavaScript auf Server- als auch auf Clientseite hat, ist der Shared Code. Die Klassen für Shapes, Grafik-Knoten, Eigenschaften, deren Verhalten, die Serialisierung und Deserialisierung – all das fällt unter Shared Code und kann gleichermaßen auf dem Client als auch auf dem Server eingesetzt werden. Der Server liefert diesen Shared Code als JavaScript Modul an den Browser aus. Dort kann dann mit dem Modulnamen darauf zugegriffen werden.

Die Hauptaufgabe der Serverseite liegt jedoch darin, AJAX Anfragen des Editors zu beantworten. Dazu gehört auch das Laden der Shapes. Dabei werden die Shapes aus der Datenbank geladen und an den Browser ausgeliefert. Aus diesen Rohdaten werden dann JavaScript Objekte mit entsprechendem Verhalten.

JSWDF ermutigt dazu, an den Client auszuliefernden JavaScript Code durch einen Uglyfier laufen zu lassen, dann zusammenzufassen und zu cachen.

Der Vorteil schneller Ladezeiten, der dadurch erreicht wird, ist jedoch nicht das einzige Ziel. Während der Entwicklung müssen manche JavaScript Daten ständig verändert werden und es gilt durch Debugging Fehlerursachen zu erkennen. Das Caching verhindert jedoch, dass Änderungen überhaupt am Client ankommen und der Uglyfier führt dazu, dass Fehlermeldungen nicht mehr richtig interpretiert werden können.

Aus dieser Überlegung heraus, wurde entschieden, alle direkt für den Editor notwendigen JavaScript Dateien während der Entwicklung einzeln zu laden. Caching und Uglyfier kommen somit nur beim Shared Code und jQuery zum Einsatz.

## 6.5. Layout und Logo

Von vornherein stand fest, dass wir uns an bekannten Grafik- und Bildbearbeitungsprogrammen orientieren wollen, um die User Experience nicht unnötig zu verkomplizieren. Zu Beginn stand ein dreigeteiltes Layout mit Werkzeugleiste links, Zeichenfläche rechts und einer Eigenschaftenleiste unten im Raum (siehe Abbildung 4).

Zusätzlich sollte bei Bedarf ein Panel über dem rechten Teil der Zeichenfläche floaten, welches die Erstellung und Bearbeitung von Ebenen erlaubt. Letztlich haben wir uns für eine Vierteilung entschieden, da der Objektbaum im Verlauf des Projektes an Bedeutung gewann und statt Ebenen nun die einzelnen Elemente gemäß der Baumstruktur dauerhaft rechts angezeigt werden.

Grafisch orientierten wir uns am Trend aktueller „Modern UIs“ wenige Farben und schlichte Symbole zu verwenden. Alle Icons wurden selbst erstellt, wobei auch hier darauf geachtet wurde, sich an bekannten Darstellungen zu orientieren, um beim User den Wiedererkennungswert zu erhöhen und somit die Einarbeitung zu erleichtern.

Das Logo entstand erst sehr spät. Die beiden Farben Schwarz und Weiß im Hintergrund stehen hier für die beiden gegensätzlichen Seiten Client und Server. Die Verbindung zwischen beiden stellt der Schriftzug dar. Das Spinnennetz hebt hervor, dass es sich um eine Web-Anwendung handelt. Während die allgemeine Farbgebung in Schwarz und Weiß auch als Hommage an den monotonen Strom fließender Einsen und Nullen verstanden werden kann, stellt der Pinsel mit seinem Verlauf und den runden Formen den künstlerischen Aspekt dar.

## 6.6. Stolpersteine

Die optimale Strukturierung großer Mengen an komplexem JavaScript Code ist von vornherein nicht immer ersichtlich. Der Code wächst mit fortschreitender Entwicklung und wird immer unübersichtlicher. Um den Code wieder in eine übersichtliche Form zu bringen, haben wir in späteren Entwicklungsstadien mehrere Umstrukturierungen durchgeführt.

Um die Überschaubarkeit des Codes zu erhöhen, haben wir Redundanzen vermieden und den Code in kleinen Funktionen mit aussagekräftigen Namen strukturiert. Es sollten keine schwer zu verstehenden Funktionen mit mehreren hundert Zeilen Code.

Ein anderes Problem gab es beim Zugriff auf ein Attribut einer Komponente von einer anderen Komponente aus. Da der Objektpfad relativ lang ist, wird er einer kurzen Variablen zugewiesen. Jetzt muss man jedoch darauf achten nur Methoden auf der Variable aufzurufen, die das Objekt in Place bearbeiten. Eine Zuweisung zerstört jedoch die Referenz auf das Objekt. Die Lösung sieht so aus, dass man Methoden für die Bearbeitung des Attributs anbietet, so dass kein direkter Zugriff auf das Attribut nötig wird.

Folgendes Beispiel leert das Array der markierten Objekte und legt danach ein Objekt hinein.

```
var markedObjects=this.editor.views.canvasView.markedObjects;
markedObjects=[];
markedObjects.push(myObject);
```

Bei der 2. Zuweisung wird die Referenz auf das originale Array in der canvasView überschrieben. Obwohl wir es leeren wollten, wurde das originale Array nicht angetastet. Das Objekt wird dann in unser neues Array eingefügt, aber nicht in das der canvasView.

```
var canvasView=this.editor.views.canvasView;
canvasView.emptyMarkedObjects();
canvasView.addToMarkedObjects(myObject);
```

Die Methoden übernehmen für uns die Änderungen an dem Array der CanvasView, wodurch wir uns keine Sorgen mehr über überschriebene Referenzen machen müssen.

## 6.7. Entwicklungszeit

Angefangen im April 2013 hatten wir Zeit bis zum Präsentationstag am 26. Juni 2013 eine vorzeigbare Anwendung zu entwickeln. Erste Konzepte und Ideen hatten wir schon vorher niedergeschrieben. So ging es ab April darum, ein Layout zu finden und die Softwareseite des Projekts umzusetzen.

Jede Woche haben wir viel Zeit in das Projekt investiert. Hierbei wurde die Arbeit grob aufgeteilt und selbstständig durchgeführt. Ein bis zweimal pro Woche trafen wir uns, um gemeinsam Fragen zu klären und das weitere Vorgehen zu besprechen.

Der aktuelle Stand wurde stets in einem Subversion Repository gepflegt. Dadurch konnten wir effizient zusammenarbeiten und auf eine (meist) funktionierende Codebasis aufbauen.

## 6.8. Aktueller Stand

Im aktuellen Stadium ist der Grafikeditor nur begrenzt produktiv einsetzbar. Es ist möglich, sich einzuloggen und Grafiken zu erstellen. Das Anlegen und Bearbeiten von Objekten funktioniert nahezu reibungslos. Ebenso ist das Speichern und Laden sowie der Export als statische PNG-Datei möglich. Auch Ressourcen wie Bilder und Schriften können mit dem Ressourcen Manager hochgeladen und ausgewählt werden.

Die Anbindung an externe Daten fehlt jedoch noch. Wir haben bisher nur Prototypen getestet, bei denen die Daten statisch direkt im Code eingebunden wurden. Dabei werden bestimmte Eigenschaften auf einen Pfad gesetzt, unter dem der Datenwert für diese Eigenschaft erreichbar ist. Beim Rendervorgang wird der Pfad aufgelöst und der dahinterstehende Wert verwendet um das Objekt zu rendern.

## 6.9. Ausblick

Obwohl schon viel Arbeit in den Editor geflossen ist, wurden manche der vorgesehenen Funktionen noch nicht oder nicht vollständig implementiert. Desweiteren bieten sich vielfältige Erweiterungsmöglichkeiten an.

### Datenanbindung und Import

Für das Einbinden von dynamischen Daten ist es nötig, entsprechende Datentreiber zu schreiben, die Daten aus einer externen Quelle in ein passendes Format bringen.

Der Editor soll einen Datenexplorer erhalten, der immer dann aufgerufen wird, wenn man eine Eigenschaft wie z.B. den Text von einem Textobjekt mit entsprechenden Daten verknüpfen möchte.

### Grafikveröffentlichung

Eine Grafik soll z.B. auf einer Webseite eingebunden werden. Hierfür wird eine URL benötigt, unter der die Grafik erreichbar ist. Die Erzeugung dieses Links nennen wir Veröffentlichung. Bei jedem Aufruf des Links wird die Grafik mit den aktuellen Daten aus der Datenquelle gerendert und ausgeliefert.

Der Link muss dabei Angaben über die zu benutzende Grafik enthalten, sowie eventuell den Datensatz, der aus der Datenquelle verwendet werden soll.

### Weitere Exportformate

Beim statischen Export sind weitere Formate wie PDF denkbar.

### Weitere Shapes

Bisher ist die Shape-Auswahl recht klein. Diese soll um weitere Basis-Shapes aber auch um komplexere Shapes erweitert werden. Vorstellbar ist z.B. ein Fraktal-Shape, das Fraktale auf die Grafik zeichnet, oder Diagramm-Shapes, die aus den Eingabedaten ein Diagramm zeichnen. Allein die Renderfunktion eines Shapes gibt vor, was auf die Ausgabe gezeichnet wird. Insofern sind hier keine Grenzen gesetzt, wenn es darum geht neue Shapes zu erstellen.

### Standard-Schriften

Es ist angedacht, bereits einige OpenSource-Schriften im System zu hinterlegen. So können User Texte erstellen ohne vorher eine Schriftart hochladen zu müssen.

### Vorschaubilder

In der Grafikübersicht, von der aus die Grafiken geladen werden können, sollte für jede Grafik eine Vorschau angezeigt werden.

Im Ressourcen Manager sollten für die verfügbaren Ressourcen geeignete Thumbnails als Vorschau angezeigt werden. Auch im Objektbaum sind Thumbnails zur schnelleren Identifizierung der Grafikobjekte angedacht.

### Editor Rückgängig und Wiederherstellen

Eine wichtige Komfortfunktion in Editoren aller Art ist das Rückgängig machen und Wiederherstellen von Änderungen. Dies soll bei GrEdit noch hinzugefügt werden.

### Editor Optimierung

Der Editor bietet noch viele Möglichkeiten zur Optimierung von Funktionen und Erscheinungsbild. Angedacht ist, dass der Editor über die Session hinweg speichert, welches Werkzeug und welches Grafikobjekt aktuell gewählt sind, oder welche Zoomeinstellung gesetzt ist.

### Benutzerkonten

Bis jetzt ist es möglich, sich zu registrieren und einzuloggen. Für den Benutzer ist das jedoch noch nicht ausreichend. Man soll auch seine Emailadresse, Passwort und weitere Angaben ändern können. Zudem soll es möglich sein, das eigene Konto zu löschen, wobei zur Auswahl stehen soll, ob die Grafiken dabei mit entfernt werden, oder nicht.

## 6.10. Geschäftsmodell

Der Editor ist Teil eines Webservices. Kunden erstellen Grafiken, diese werden von uns gehostet und ausgeliefert. Es bietet sich an, die Anzahl der Grafiken pro Kunde zu beschränken und Kunden die einen hohen Bedarf haben, dafür bezahlen zu lassen.

Zudem werden Daten aus unterschiedlichsten Quellen benötigt. Für jede Quelle müssen Datentreiber programmiert werden. Dies stellt eine Dienstleistung dar, die wir gegen Bezahlung anbieten könnten.