Implementierungsbericht: Crayons 2.0

Marc Jepsen, Natalia Krylova,
Levin Schickle, Ian Fitzgerald, Ali Akil, Julius Busch

20. März 2017

Inhaltsverzeichnis

1 Einleitung **4**

1.1 Bezug auf Planungsphase 4

1.2 Bezug auf Entwurfsphase 4

2 Umsetzung von Wunsch- und Musskriterien **5**

2.1 Frontend 5

2.1.1 Einheitseditor 5

2.1.2 Bibliotheken Views 6

2.1.3 Kurseditor 6

2.1.4 Layout/Style und Responsiveness 8

2.1.5 Generelles zur GUI 8

2.2 Backend 9

2.2.1 Datenbank und SpringFramework 9

2.2.2 Login und Authentifizierung 9

2.2.3 Rollen und Authorisierung 9

2.2.4 Editoren Backend 10

3 Verzögerungsgründe **11**

3.1 Kurseditor 11

3.2 Spring 11

3.3 Vorbereitung 12

3.4 Entwurf 12

3.5 Autorisierung 12

4 Lessons learned **14**

4.1 Architektur und Grundstruktur 14

4.2 JavaScript Connector 14

4.3 Navigation 14

4.4 Style 15

4.5 View 16

4.6 Spring 17

4.7 Team-Organisation 19

4.8 Komponenten und Tests 19

5 Arbeitsaufteilung **20**
5.1 Marc 20
5.2 Ian 20
5.3 Natalia 20
5.4 Ali 20
5.5 Levin 20
5.6 Julius 21

1 Einleitung

1.1 Bezug auf Planungsphase

Die in der Planungsphase recherchierten Tools haben sich in der Implementierungsphase bewährt. Das Einzigste, was zu einem eher kleineren Problem führte, war die Wahl der Javascript Library, welche die Aufgabe hat, den Lerngraphen automatisch zu rendern. Hierzu gibt es eine genauere Erläuterung unter der Rubrik Kurseditor. Des Weiteren konnten nicht nur Musskriterien umgesetzt werden.

1.2 Bezug auf Entwurfsphase

Alle in der Entwurfsphase festgelegten Entscheidungen zur Architektur und den verwendeten Frameworks konnten umgesetzt werden. Während des Implementierungsprozesses wurden Details erkannt, wie zum Beispiel, dass die AppConfig Klasse nicht notwendig ist oder, dass es besser ist eine zusätzliche Graphrepräsentation für Kurse zu schreiben. Wie schon im Entwurf beschrieben, sorgt Spring für Entkoppelung der Objekte per Dependency Injection. Daten werden in einer PostgrSQL Datenbank abgelegt und die vom Spring Boot übernommene Konfiguration übernimmt die Anbindung an einen Tomcatserver. Hierzu eine genauere Erläuterung unter der Rubrik Datenbank und Spring. Die aufwändigsten zu erledigenden Arbeiten waren der Kurseditor und die Datenbankanbindung in Kombination mit Spring. Bei beiden Aufgaben wurde viel Zeit investiert, um die Problematik zu verstehen und sich bei knapper offizieller Dokumentation durch das Meer an inoffiziellen Lösungstrategien zu kämpfen. Am Ende wurden jedoch auch hier die Verständnisprobleme von Spring in Kombination mit Vaadin und die Umsetzung des Graphrenderers gelöst. Mehr hierzu ist unter der Rubrik Verzögerungsgründe zu finden.

2 Umsetzung von Wunsch- und Musskriterien

2.1 Frontend

2.1.1 Einheitseditor

Zunächst wurde ein oberes Menü als HorizontalLayout mit drei Komponenten vom Typ DragAndDropWrapper erstellt: Text, Bild und Multiple-Choice-Aufgabe. Über LayoutClickListener für das Menü wurde Ziehen-Funktion implementiert. Jede von drei Komponenten kann per Ziehen dem Inhalt der Lerneinheit hinzugefügt sein.

Der Inhalt der Einheit ist als CustomComponent implementiert. CustomComponent enthält den Einheitstitel und alle Komponenten der Lerneinheit, die in DragAndDropWrapper eingepackt sind. Somit kann man die Position der Komponente wechseln. Wenn der Inhalt leer ist, wird dem Benutzer ein "Drop Area"gezeigt.

Da jede Komponente nach dem Erstellen in DragAndDropWrapper eingepackt war, gab es ein Problem mit ihrem Entfernen. Das wurde dadurch gelöst, dass ein als DragAndDropWrapper implementierten Entfernen-Button erstellt wurde. Die Komponenten können über das Button geschoben und dadurch gelöscht werden.

Text-Komponente ist als CKEditor implementiert.

2.1.2 Bibliotheken Views

2.1.3 Kurseditor

Die in der Planung und im Entwurf angedachte Umsetzung des Graphen sah vor eine Vaadin Komponente zu schreiben welche in einem bestimmten festgelegten Bereich im Layout ein mit Javascript gerenderten Graphen anzeigen kann. Dies ist zunächst auch gelungen. Als Erstes wurde eine Klasse implementiert, die von der von Vaadin bereitgestellten Klasse `AbstractJavaScriptComponent` erbt. Diese stellt die eigentliche Vaadin Komponente dar, die später dem Layout hinzugefügt werden kann. Sie benutzt eine Zustandsklasse, um Änderungen der Daten zu verwalten und wird per Annotation mit dem eigentlichen Javascript File verbunden. Im Javascript kann dann wiederum auf das DOM der Vaadinkomponte zugegriffen werden. Außerdem wird eine Methode `function onStateChange()` unterstützt welche automatisch aufgerufen wird, sobald sich Daten am Objekt ändern. Mit dieser Vorgensweise war es zunächst möglich Javascript anzuzeigen und auf UI-Elemente wie Buttons zu reagieren. Das eigentlich vorgesehene Framework Joint.js brachte leider folgende Probleme mit sich: Zunächst konnte nach langem Ausprobieren und Recherchieren das DOM der Vaadin Komponente mit einem Canvas Javascript Fenster angezeigt werden, in welchem ein Graph angezeigt wird. Hierbei wurde der Graph aber nicht automatisch gerendert, sondern nur die Kanten zwischen den Knoten. Die Koordinaten der Knoten mussten genau bestimmt werden. Danach ist aufgefallen, dass es nötig war vorerst ein layout plugin von Joint.js zu installieren, um die gewünschte Funktionalität zu ermöglichen. Hier gab es jedoch die Sackgasse. Um die Methoden dieses Plugins zum laufen zu bringen, mussten zwei weitere Bibliotheken hinzugefügt werden (Backbone und Dagre). Wobei Dagre das Rendering übernimmt und Backbone eine RestFul-JSON Schnittstelle ist, die verschiedene Backend Dinge übernehmen kann. Leider war es danach nicht mehr möglich auf das DOM Element zuzugreifen. Da Dagre für das Rendering in Joint.js benutzt wurde und eigentlich genau das ja auch gebraucht wurde konnte die Dagred3 library Abhilfe verschaffen. Hier war es unter anderem auf Grund der guten Dokumentation der D3 Library, welche die grafische Grundlage von DagreD3 ist, am Ende möglich den Graphen automatisch zu rendern. Letztendlich gab es nur noch das Problem, dass die Kommunikation zwischen der AbstractJavascript-Component von Vaadin und dem Javascript File leider nicht für die eigentlich verwendeten Objekte der Graphrepräsentation funktionierte, sondern nur für primitive Datentypen. Deshalb mussten Methoden implementiert werden, die

die Repräsentation des Graphen in String Arrays übersetzen. Diese Arrays werden dann einfach einer setState Methode übergeben und im Javascript wiederum mit getState aufgenommen.

2.1.4 Layout/Style und Responsiveness

Das Layout gliedert sich in zwei Hauptbereiche. Zum einen der LoginScreen und zum anderen der Mainscreen. Beides wurde mit einem CSSLayout umgesetzt. Im Mainscreen wird nochmal ein Menu als CSSLayout instanziert. Dieses beinhaltet die Navigationsleiste und befindet sich am linken Rand. Der Rest vom Layout stellt den Content Bereich dar. Mit Hilfe der Vaadin Methode setStyleName() wurde von vorgefertigten CSS Regeln gebrauch gemacht. Genauer wurden hier Regeln des Vaadin Valo Themes benutzt, die es ermöglichen das Mainscreen Layout und die Navigationsleiste an die Bildschirmauflösung anzupassen. Die Anbindung an das CSS erfolgt per Annotation über der Vaadin HauptUI Klasse im default Package. Mithilfe von scss Files wird 'On the Fly Compilation' umgesetzt, die einen schnelleren Seiten Zugriff ermöglicht, indem nur die gerade nötigsten Regeln benutzt werden. Sobald Änderungen am Theme vorgenommen wurden muss es jedoch neu kompiliert werden. Hierfür wird der Vaadin Widgetset Compiler verwendet. Viel CSS kommt auch im Zusammenhang mit dem Uniteditor zum Einsatz. Hier sind nach einem ähnlichen Prinzip die Palette Leiste als Navigation im oberen Bereich und als Content Area das pageLayout beide mit dem CSSLayout implementiert. CSS übernimmt verschiedene Regeln beim Hover der Elemente, wie z.B. das farbige Hervorheben, Anzeigen und Verbergen eines Bearbeiten Buttons, oder Anzeigen des cursers im move Modus. Für das Drag und Drop CSS wird wieder Gebrauch von den Vaadin Valo Regeln gemacht.

2.1.5 Generelles zur GUI

Für die Implementierung der graphischen Benutzeroberfläche wurde Vaadin Framework benutzt. Das Framework hat es ermöglicht, fast das gesamte GUI in Java zu schreiben. Die Styling von einigen Elementen, wie zum Beispiel Positionierung von Buttons, wurde mit CSS implementiert. Es wurden folgende Views erstellt: Login-Bereich, Registrierungsformular (für alle Benutzer erreichbar); Autorenbibliothek, Benutzerbibliothek, Suche, Einstellungen (über Menü für alle registrierten Benutzer erreichbar); Kurseditor, Einheitseditor, Kursansicht für Schüler, Einheitsansicht für Schüler (kurs- und benutzerspezifisch, deswegen nur aus den anderen Views erreichbar).

@TO DO: Controller -> View / autowiring

2.2 Backend

2.2.1 Datenbank und SpringFramework

Die Datenbank wurde wie schon Entwurf beschrieben, in DataAccesObjects (DAO's) und dem jeweilig dazugehörendem Service umgesetzt. Die DAO's sind für den eigentlichen Datenbankzugriff zuständig sind, während Services die DAO's um Logik erweitern und als Schnittstelle für den Controller dienen.

Mangels Zeit und Erfahrung wurde für jede Komponente (users, courses, units, ...) eine einfache Tabelle angelegt. Auf Querverweise und Fremdschlüssel wurde verzichtet, stattdessen werden die Daten über die Logik in den Services passend gefiltert.

Für das ausführen der SQL-Anweisungen wird das JDBCTemplate von Spring verwendet.

2.2.2 Login und Authentifizierung

Zunächst wurde um das Prinzip des Navigators und die Funktionsweise eines Logins in Vaadin zu verstehen eine einfache Applikation mit Dummylogin geschrieben. Dies wurde danach in die beim Entwurf festgelegte Modellierung umgeschrieben. Im Entwurfsheft sind hierzu schon genauere Sequenz und Klassendiagramme zu finden. Im Nachhinein musste der Wechsel zwischen LoginScreen und Mainscreen mit dem Navigator an die generelle Springumsetzung angepasst werden. Zuvor wurde der Navigator erst nach dem Login im Mainscreen erzeugt und die UI nach erfolgreicher Authentifizierzung neu aufgesetzt. Dies führte allerdings zu Problemen die nach der Anpassung des Navigators behoben werden konnten,sodass es mit dem Autowiring von Spring keine Probleme mehr gab. Hierzu gibt es genauere Erläuterungen unter der Rubrik Verzögerung und Spring (Punkt 3.2).

2.2.3 Rollen und Authorisierung

Durch Schwierigkeiten in der anfänglichen Authentifizierung mussten für die Rechte auf ein primitiveres Rechtelevelsystem zurückgegriffen werden, welches jedem Benutzer ein Authorisierungslevel zuweist und beim Laden der Programmelemente etwaige Funktionen blockiert.

2.2.4 Editoren Backend

Die Editoren, als auch Import und Export, arbeiten mit der Ausgabe von .bin Dateien, welche in die Datenbank geladen und wieder hergestellt werden können. Dies ermöglicht eine hohe Flexibilität der speicherbaren Objekte, schränkt die Software allerdings für spätere Änderungen ein.

3 Verzögerungsgründe

3.1 Kurseditor

Wie schon in der Rubrik Musskriterien erwähnt hat sich bei der Umsetzung des Graphrenderings mit Joint.js eine Sackgasse entwickelt. Eine genauere Erläuterung des Problems und wie dieses gelöst wurde ist unter 2.1.1 zu finden. Außer der dort beschriebenen Lösungsfindung wurden verschiedene Addons von Vaadin installiert und ausprobiert. Als Notlösung wurde eine alterntive Darstellung des Graphen mittels eines Gridlayouts angedacht.

3.2 Spring

In Vaadin Dokumentation war nicht ganz eindeutig wie man Spring mit Vaadin integriert.Wir haben uns daher an Spring Dokumentation orientiert. Im Laufe des Projekts stießen wir öfters auf Exceptions mit unbekannter Ursache. Nach langer Recherche haben wir herrausgefunden dass Vaadin eine eigene Java-Notationen[VS] für Spring hat. Im Spring kann man die graphische Oberfläsche mit hilfe 'WebMvcConfigurerAdapter' Interface anbinden. Da jedoch Vaadin ein Sigle-Page-Architektur hat war solcher Einsatz nicht möglich. Wir mussten dementsprechend viele Änderungen an dem Projekt anpassen vorallem an der Navigation. Im Grunde sollte jedes View mit @SpringView(name=<name>) annotiert werden. Man kann dann zu diesem View mit UI.getUI().getNavigator().navigateTo(<name>) navigieren. Zudem sollten die Klassen, die man dafür eine Spring-Bean erstellen möchte ,mit @SpringComponent anstatt @Component annotiert werden.Vaadin hat ein Component interface und kann daher zu zweideutigkeit kommen.

Desweiteren wurde erst bei der realisierung des Controllers, was leider sehr spät erst vorran getrieben wurde, festgestellt, dass die Initalisierung der einzelnen Componenten komplett überarbeitet werden musste. Spring bietet für das @Autowiering die Initalisierungsannotation @Postcontruct, welche über der Initialisierungsfunktion angebracht werden sollte. Problematisch dabei ist, dass diese Funktionen keine Parameter erfordern darf und void als Rückgabewert besitzen muss. Um zu verhindern, dass andere zugreifende Klassen die Bean neu initialisieren, darf diese Funktion jedoch private sein.

3.3 Vorbereitung

Die verwendeten Tools erwiesen sich häufig als tückisch, da unvorhergesehen Abhängigkeiten zu umfangreichen Fehlern geführt haben.
Zum Beispiel das Verhalten zwischen Spring und der Datenbankanbindung, welche nur unter Einhaltung einer Vielzahl von Bedingungen in allen View-, Service- und Controller-Klassen möglich ist und mit viel Aufwand behoben werden musste.
Um dieses Problem in Zukunft zu umgehen wäre es hilfreich mit sich mit einer umfangreicheren Vorbereitung der verwendeten Hilfsmittel auf die Implementierung vorzubereiten. Diese Fehler sind allerdings im Wasserfallmodell im Entwurf schwer zu erkennen, da man sich noch nicht mit allen Details der verwendeten Frameworks auseinander setzt.

3.4 Entwurf

Uns sind leider in der Implementierung Lücken beim Entwurf aufgefallen. Besonders gravierend erwiesen sie sich bezüglich grundlegendster Strukturen, wie das Verwalten des aktuellen Benutzers/Kurses, korrekte Implementierung der Views, Verwendung von Spring, etc. und führten zu projektumfassenden Kompatiblitätsfehlern, Lücken und Bugs. Die anzuwendenden Fehlerbehebungen sind entweder arbeitsaufwändig oder unschön.

[VS]http://vaadin.github.io/spring-tutorial/

3.5 Autorisierung

- Appfoundation
  - Um Autorisierung mit Vaadin zu erreichen werden das Plugin 'AppFoundation' empfohlen. Die Dokumentation des Plugins ist nicht für Anfänger geeignet und ziemlich veraltet.Das Problem mit dem Plugin lag daran dass der Plugin mehrere Dependencies benötigt und das Widgetset dafür nicht leicht zu kompilieren ist. Das Maven-Configuration-dile(pom.xml) aus Springboot projekt stammt und war nicht an Vaadin-plugin-architektur angepasst und die Versionen der Dependencies werden in der Dokumentation nicht vorgegeben. Zudem die Community ist sehr klein und der Support ziemlich schlecht. Aus diesen Gründen haben wir uns nach anderer Lösung gesucht.

Spring Security war eine andere möglichkeit. Eine kleine Einfürung über Spring Security ist unter 4.6 zu finden.

- Spring Security
  - In Vaadin Book findet man soviel wie garnicht wie man Spring security mit Vaadin-Anwendungen integriert. Wenn man nach 'Spring Security' sucht findet man lediglich einen Treffer 'Integration with authorization solutions, such as Spring Security, is provided by additional unofficial add-ons on top of Vaadin Spring.' Nach einer Recherche findet man einen gebastelten Hybird-Ansatz[HA]. Vaadin Anwendungen haben keine normale Web-Seiten-Navigation, sie laufen in der Regel in einer 'Signle-Page'. Aus diesem grund kann Spring Security die Zugriffskontrolle auf Vaadin-Views nicht kontrollieren. Man kann daher Spring Security nur für method level securityïn Backend verwenden. Zudem wurde in dem Hybrid-Einsatz Vaadin-Session anstatt Http-Session eingesetzt. Da wir jedoch die daten in http-session speichern konnte dieser Einsatz nicht berücksichtigt werden.

[HA]https://vaadin.com/blog/-/blogs/a-hybrid-approach-to-spring-security-in-vaadin-applications

4 Lessons learned

4.1 Architektur und Grundstruktur

Um einen guten Überblick über die Vaadinarchitkektur zu bekommen empfielt es sich die jeweiligen Seiten im Vaadin Book zu lesen. Vor allem die Komponentenhirarchie ist hier sehr hilfreich. Zur Architektur wird generell zu einem MVP pattern geraten, was allerdings zu wesentlich mehr Komplexität an Code führt. Nach langer Recherche haben wir uns schon im Entwurf zu einer eher simpleren Architektur entschieden. Dies hat sich während der Implementierung bewährt und ist auch so zu empfehlen.

4.2 JavaScript Connector

Leider ist dieses Gebiet im Vaadin Book nicht sehr ausführlich dokumentiert und sonst ist es auch schwierig Dinge zu finden, die nützlich sind. Nach langer Recherche war jedoch dieser Artikel von großer Hilfe.
Die direkte Verlinkung ist leider nicht möglich. Deswegen am besten im rechten Menür den Reiter 2013 auswählen und dann runter scrollen bis zu Überschrift 'Using JavaScript libraries (D3) in Vaadin webapplications'.

4.3 Navigation

Zur Navigation wird in Vaadin Applikationen die `Navigator` Klasse benutzt.
Im Groben zusammengefasst wird ein Navigator Objekt instanziert, welchem dann die Views hinzugefügt werden. View ist wiederum ein Interface, welches von den einzelnen View Klassen implementiert werden muss, sofern man sie mit dem Navigator erreichen soll.Um das Prinzip zu verstehen und eine Grundstruktur schreiben zu können, wie beispielsweise von einer Login Seite zu einem Mainscreen und dann im Mainscreen zwischen den Views gewechselt wird, empfielt es sich die Quicktickets Demo und den Beispiel Maven Archetyp von Vaadin anzuschauen. Generell kann dies alleine mit dem Navigator umgesetzt werden.

4.4 Style

In Vaadin Applikationen wird meistens ein Theme verwendet. Es ist jedoch auch möglich alle Komponenten individuell mit eigenem CSS zu stylen.
Hierfür kann jeder Komponente mit `Component.setStyleName(exampleString)` eine CSS ID gegeben werden. Für Layouts außerdem öfters verwendete Methoden sind:

- `Layout.setMargin(boolean)` Setzt die Margin(Rahmen der Komponente) entweder auf einen bestimmten Abstand, oder einfach einen deafult Abstand.
- `Layout.setSpacing(boolean)` Komponenten, die einem Layout hinzugefügt wurden haben einen Abstand zueinander.
- `Layout.setSizeFull(boolean)` Das Layout nimmt den gesamten Platz der nächst höheren Komponente ein.
- `Layout.setComponentAllignment(Component,Allignment)`
  Positioniert eine Komponente in dem jeweiligen Layout.
  Mögliche Postionen sind:

  `Alignment.TOP_LEFT`

  `Alignment.TOP_RIGHT`

  `Alignment.TOP_CENTER`

  `Alignment.MIDDLE_LEFT`

  `Alignment.MIDDLE_RIGHT`

  `Alignment.MIDDLE_CENTER`

  `Alignment.BOTTOM_LEFT`

  `Alignment.BOTTOM_RIGHT`

  `Alignment.BOTTOM_CENTER`

4.5 View

Um eine Kompenente in der View anzeigen zu lassen, muss zunächst eine View Klasse erstellt werden, welche ein von Vaadin unterstütztes Layout implementiert. (Diese sind auch im Vaadin-Book ausführlich erklärt) Die am meisten verwendeten Layouts sind:

- `VerticalLayout`
  Alle Komponenten, die dem Layout hinzugefügt worden sind (per addComponent) werden untereinander aufgereiht.

- `HorizontalLayout`
  Alle Komponenten, die dem Layout hinzugefügt worden sind, werden nebeneinander aufgereiht.

Hier ist zu Beachten, dass Anfangs des Öfteren eine kleines Detail ein wenig Zeit kostet:

- Für Klassen, die `View` implementieren ist
  `Layout.addComponent(Kompenente)` zu verwenden

- Für Klassen, die `Window` implementieren oder
  in der MainUI Klasse von Vaadinliegen ist
  `Layout.setContent(Kompenente)` zu verwenden

Hierbei ist es wichtig zu wissen, dass Windows als Popups verwendet werden, während Views im Content Bereich des Layouts angezeigt werden sollten.

4.6 Spring

Das Spring Framework bietet ein Programmier- und Konfigurationsmodell für die Erstellung von Java-basierten Enterprise -Anwendungen. Spring bietet viele nützliche Dienste für den Aufbau von Anwendungen, wie Authentifizierung, Autorisierung, Datenzugriff, etc. Die Kernkraft des Spring Frameworks ist die Dependency-Injection.

- es handelt sich um ein Entwurfsmuster und wird damit ausgedrückt, dass Module (Objekte, Klassen) ihre Abhängigkeiten von einer anderen, externen Instanz zugewiesen bekommen, was dann auch als Injektion bezeichnet wird
- fördert einen guten Code da die Funktionalität der Klasse wird generalisiert und ist von unbekannter Implementierung losgelöst was zu einer besseren Verständlichkeit führt.
- erleichtert auch das Testen da die Testobjekte injected werdenkönnen um JUnitTests zu vereinfachen.

Dadurch ist das Framework streng modular aufgebaut mit möglichst wenig Abhängigkeiten zwischen den einzelnen Modulen. Das hat zur Folge, dass der Entwickler, eine Anwendung mithilfe einiger Build-in, dritter und selbstcodierter Komponenten ganz einfach zusammenbauen kann. Neben den Kernkomponenten des Frameworks, verfügt Spring über weitere Projekte, wie Spring Security, Spring Data , Spring Boot etc.

- Spring Boot
  - Ein Spring-Projekt zusammenzubauen ist nicht so leicht da man die richtigen JARs und Komponente von Hand holen muss und sie richtig konfigurieren. Aus diesem Grund entstand das Projekt Spring Boot. In http://start.spring.io/ kann man die benötigten Dependencies aussuchen und ein fertig konfiguriertes Projekt exportieren
  - In http://start.spring.io kann man die benötigten Abhängigkeiten aussuchen und ein fertig konfiguriertes Spring Projekt exportieren. Spring Boot automatisiert die Initialisierung und Konfiguration eines Projekts und vereinfacht dementsprechend die Entwicklung. Dabei wird die Kontrolle von dem Entwickler nicht weggenommen. Man kann jede Zeit Spring-Boot-Automatisierung überschreiben.

- Spring Security
  - Spring Security kann sowohl Authentifizierungs- und Autorisierungsanfragen umsetzen. Die Autorisierung kann sowohl auf Web-Ebene als auch auf Methodenebene geschehen.

4.7 Team-Organisation

Die für das Projekt gewählte Herangehensweise war in keiner Weise den gestellten Bedingungen angemessen. Die horizontale Arbeitsverteilung (in zB. View, Controller, Service, Backend) ist für ein derartiges Projekt nicht empfehlenswert, da die Unerfahrenheit und unregelmäßigkeiten der Arbeitsrythem zu Fehlern in den zahlreichen Schnittstellen führt. Eine dynamische Vergabe der Aufgaben anhand der definierten Kriterien bereits in der Entwurfsphase, hätte vermutlich zu einem besseren Ergebnis geführt.

4.8 Komponenten und Tests

Ein Kernelement der Softwareentwicklung stellt das Fertigstellen und Testen von Teilkomponenten dar, um den Entwicklungsprozess anderer Elemente zu ermöglichen. Eine vernünftige Definition der Reihenfolge und Funktion dieser Komponenten inklusive umfangreicher Tests ist somit vorausgesetzt.

5 Arbeitsaufteilung

5.1 Marc

- Suche
- Import/Export
- Umsetzung Backend
- Speicherformat

5.2 Ian

- CK Editor Installation
- Autorisierung
- Umsetzung Backend

5.3 Natalia

- Einheitseditor
- View
- Kurseditor Frontend

5.4 Ali

- Spring Recherche
- BugFixes Backend

5.5 Levin

- Datenbank
- Grundlage für Umsetzung Backend
- Mehrsprachigkeit

5.6 Julius

- Style und Layoutgrundlage
- Navigation
- Authentifizierung und Login mit Spring
- Graphrepräsentation und Rendering
- Einheitseditor

Zu Beachten ist, dass es sich hierbei um Schätzwerte handelt, da es Formatierungs- und strukturelle Commits gab die viel selbstgenerierten Code hinzugefügten.

	LOC	Investierte Zeit
Ian	1750	170 h
Marc	1900	200 h
Julius	2700	220 h
Ali	1500	170 h
Natalia	1500	170 h
Levin	2300	220 h

Im benutzten Github Repository sind weitere Statisktiken vorhanden.