Fachbereich
Mathematik/Informatik

Bachelorarbeit

Entwicklung einer Webschnittstelle zur Durchführung von Kurswahlen an Gymnasien

Autor: Tilo Wiedera
tilo@wiedera.de

Prüfer: Prof. Dr. Oliver Vornberger

Abgabedatum: 11.10.2013

I Kurzfassung

Jedes Jahr werden an deutschen Gymnasien verschiedene Wahlpflichtfächer angeboten. Innerhalb dieser Arbeit wird eine Webapplikation auf Basis von Backbone.js und Ruby On Rails entwickelt. Diese Anwendung Vereinfacht die Durchführung von Kurswahlen an Gymnasien. Dazu wurde die Applikation in Kooperation mit dem Graf-Stauffenberg-Gymnasium aus Osnabrück evaluiert. Die Arbeit legt einen Schwerpunkt auf die Client-seitige Programmierung mit CoffeeScript und nutzt Rails als erweiterten Datenbankserver mit REST-Schnittstelle. Es werden Methoden und Konzepte zur Entwicklung einer modernen und responsiven Web-Anwendung aufgezeigt.

Abstract

Every year there are several elective subjects for pupils to choose from at german secondary schools. This thesis will illustrate the development of a web application based on Backbone.js and Ruby On Rails. The application simplifies the selection of those subjects. It has been tested at the Graf-Stauffenberg-Gynmnasium of Osnabrück for this purpose. The thesis emphasizes a rich client, heavily utilizing CoffeeScript and Ruby On Rails as an extended database serving JSON via a REST interface. Methods and concepts for developing a modern and responsive web application are demonstrated.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

V Listing-Verzeichnis

VI Abkürzungsverzeichnis

DRY Don't Repeat Yourself
ROA Ressource Oriented Architecture
REST Representational State Transfer
COC Convention over Configuration
MVC Model-View-Controller
ERB Embedded Ruby Code
ROR Ruby on Rails
POLS Principle of Least Surprise
ORM Object Relational Mapper
CRUD Create, Read, Update and Delete
HTTP Hypertext Transfer Protocol
URL Unified Resource Locator
JSON Javascript Object Notation
TDD Test Driven Development
AJAX Asynchronous Javascript and XML
XML Extensible Markup Language
API Application Programming Interface
APT Advanced Packaging Tool
AMD Asynchronous Module Definition
DOM Document Object Model
CSS Cascading Stylesheets
CDN Content Delivery Network
HTML Hypertext Markup Language
EJS Embedded Javascript
SOC Separartion of Concerns
MV* Model-View-Star
SPA Single Page Application
JS Javascript
SASS Syntactically Awesome Stylesheets
UI User Interface
SME Subject Matter Expert
GSG Graf-Stauffeberg-Gymnasium Osnabrück
XP Extreme Programming
VCS Version Control System
DVCS Distributed Version Control System
DSL Domain-Specific Language
ERD Entity-Relationship Diagram
XHR Extensible Markup Language (XML) Hypertext Transfer Protocol (HTTP) Request
HATEOAS Hypermedia as the Engine of Application State
YAML YAML Ain't Markup Language (ehemals: Yet Another Markup Language)
CSV Comma-Seperated Values
PDF Portable Document Format
LOC Lines of Code

SSL Secure Socket Layer

1 Einführung

Vorab wird die Motivation der Bachelorarbeit erläutert und anschließend eine grobe Übersicht der inhaltlichen Struktur der Ausarbeitung gegeben.

1.1 Motivation

Jedes Jahr werden an deutschen weiterführenden Schulen, wie zum Beispiel Gymnasien, verschiedene Wahlpflichtkurse angeboten. Diese Kurse können von den Schülern beispielsweise vom Wechsel der achten zur neunten Klasse gewählt werden. Die Wahl eines Kurses will gut überlegt sein und bietet im Hinblick auf das Abitur bereits die Möglichkeit bestimmte Schwerpunkte zu setzen. So werden in Niedersachsen zwei Schwerpunktfächer im Rahmen der Abiturprüfung belegt. Hier entstehen gewisse Randbedingungen, die bei der Wahl der Kurse einzuhalten sind. Es ist im Allgemeinen nicht möglich, ein Abitur ohne Fremdsprachenkenntnisse zu erlangen. Ziel dieser Arbeit ist es nun, eine moderne Webapplikation zu entwerfen und zu implementieren, die Lehrern an deutschen Schulen die Kurswahl ihrer Schüler erleichtert.

Dabei sollen insbesondere zwei Punkte abgedeckt werden. Zum einen muss es für die Schüler leicht ersichtlich sein, welche Kurse sie wählen dürfen und müssen. Die Bedingungen und Regeln sollen also intuitiv verständlich gemacht werden. Zum anderen ermöglicht eine digitale Abgabe der Wahl ein wesentlich effizienteres Auswerten und Prüfen der Wahlergebnisse.

Beispielhaft soll das System nun für das Graf-Stauffenberg-Gymnasium in Osnabrück entwickelt und auch dort erprobt werden.

1.2 Methodik

Die Arbeit gliedert sich in zwei größere Teilbereiche. Zuerst werden Konzepte und verschiedene Frameworks vorgestellt, die im Rahmen der Umsetzung umfangreich genutzt wurden. Dazu gehören insbesondere Ruby on Rails (ROR) sowie Backbone.js.

Es schließt sich eine detaillierte Betrachtung der Umsetzung der Entwicklung an. Dabei werden die im ersten Teil vorgestellten Konzepte und Frameworks aufgegriffen und eingeordnet.

Abschließend wird kurz die erbrachte Leistung reflektiert und neben einem Rückblick auch die zukünftige Entwicklung und Anwendung thematisiert.

2 Technologien und Frameworks

In diesem Kapitel werden die verwendeten Frameworks und Technologien erklärt, insbesondere Rails auf Seite des Servers und Backbone für den Client. Dabei wird auch ein Blick auf die von den Frameworks verwendeten Sprachen, CoffeeScript und Ruby geworfen.

2.1 Ruby On Rails

ROR wurde erstmal im Dezember 2005 veröffentlicht und bietet ein vollständiges Framework zur Entwicklung umfangreicher Webapplikationen. ROR folgt dabei drei wesentlichen Patterns: Don't Repeat Yourself (DRY), Convention over Configuration (COC) und Ressource Oriented Architecture (ROA). [Wikg] ROR wird als open-source Projekt auf Github entwickelt und gehört mit fast 2000 Koautoren und über 10.000 commits zu den umfangreichsten Projekten auf Github. [Gita]

ROR-Anwendungen werden typischerweise gemäß dem Model-View-Controller (MVC) pattern entworfen. Dabei dient das Model zur Kapselung der Datenbankzugriffe. Soll ein Model gerendert werden, so werden die entsprechenden Daten vom Controller an die View übergeben. Views sind in ROR templates mit Embedded Ruby Code (ERB). Weil ROR in dieser Bachelorarbeit nicht direkt für das rendern der Daten zuständig ist, sondern dies mit Javascript erfolgt, werden wir die Views nicht weiter betrachten.

2.1.1 Ruby

Ruby ist eine interpretierte, strikt objektorientierte Programmiersprache und bildet die Basis jeder Rails Applikation. Ruby wurde erstmals 1995 durch Yukihiro Matsumoto veröffentlicht. [Wike] Ruby folgt in seiner gesamten Gestaltung dem Principle of Least Surprise (POLS). Die Sprache wird also so gestaltet, dass dem mit Ruby vertrauten Programmierer möglichst wenig Überraschungen und Widersprüche begegnen.

Ruby realisiert viele spannende Aspekte von denen hier nur ein für ROR relevanter Auszug vorgestellt wird. Für weitere Informationen sei an dieser Stelle auf die Literatur verwiesen [Tho04] [Bla09].

Arrays und Hashes Die wichtigsten Datenstrukturen in Ruby sind Arrays und Hashes, diese sind wie alle anderen Variablen auch, Objekte. Hashes speichern beliebige Werte unter beliebigen Schlüsseln während Arrays einen Integer als Schlüssel voraussetzen. Es handelt sich also um zwei Typen indizierter Collections. Es müssen weder Anzahl noch Typ der zu indizierenden Elemente festgelegt werden, insbesondere ist es möglich, Objekte

vollkommen unterschiedlichen Typs abzulegen. Im Vergleich zu Hashmaps erfolgt der Zugriff bei Arrays etwas schneller.

```
inst_section = {
  :cello => 'string',
  :clarinet => 'woodwind',
  :drum => 'percussion',
  :oboe => 'woodwind',
  :trumpet => 'brass',
  :violin => 'string'
}

inst_section[:oboe]     #=> 'woodwind'
inst_section[:cello]    #=> 'string'
inst_section[:bassoon] #=> nil
```

Listing 1: Hashmaps in Ruby

In Listing 1 wird das Anlegen und Zugreifen auf eine Hashmap beispielhaft dargestellt. Der Zugriff auf die Elemente eines Arrays erfolgt analog dazu. Ein weiterer im Listing eingesetzter Datentyp ist das Symbol. Symbole werden in Ruby verwendet um String-Konstanten in-line zu definieren und werden durch einen Doppelpunkt eingeleitet.

Hashmaps werden in Ruby (wie auch in Javascript) unter anderem als keyword arguments eingesetzt. Dabei wird als einziger Parameter einer Funktion ein Hash übergeben. Die eigentlichen Parameter werden dann als Schlüssel-Wert-Paare in diesen Hash eingefügt. Weil die Sprache sowieso mit duck typing arbeitet, entsteht dabei keine zusätzliche Unsicherheit für den Programmierer. [Wikd]

Blöcke und Iteratoren Ruby erlaubt die native Übergabe von einem Codeblock an eine Funktion. Dieses Konzept ist vergleichbar mit anonymen Funktionsdeklarationen. Die aufgerufene Funktion kann den Block nun beliebig oft aufrufen und ihm dabei jeweils verschiedene Parameter übergeben. So können Blöcke beispielsweise eingesetzt werden, um Arrays mittels eines Vergleichs-Blocks zu sortieren.

```
[:a,:c,:b].sort_by { |v| v } # => [:a, :b, :c]

# executes the supplied block
def callBlock
  yield
end
```

Listing 2: Blöcke in Ruby

In Listing 2 wird deutlich wie elegant Blöcke in Ruby eingesetzt werden könnnen. In Zeile 1 wird ein Array durch direkten Vergleich seiner Werte sortiert. Würde man hier eine andere Sortierung vornehmen wollen, müsste man lediglich den übergebenen Block anpassen. Die in Zeile 4 bis 6 definierte Methode ruft einen beliebigen, übergebenen Block mittels des Schlüsselworts *yield* auf. Hier wird auch eine Einschränkung deutlich: Ruby unterstützt die Übergabe von maximal einem Block pro Funktionsaufruf.

Module und Klassenmakros Wie eingangs erwähnt, handelt es sich bei Ruby um eine interpretierte Sprache. Insbesondere existiert kein starres Klassenkonstrukt welches bei Programmstart fixiert ist. So ist es in Ruby möglich, Klassen einschließlich all ihrer Methoden zur Laufzeit komplett zu manipulieren. Dieses Konzept bietet im Vergleich zu klassischen Vorgehensweisen eine Reihe von Unsicherheiten (besonders wenn man auf ausgiebiges Testing verzichtet), erlaubt es aber, viele Programmabschnitte sehr viel knapper und präziser zu formulieren und unterstützt folglich das DRY pattern.

```
class Domain < VoteData
  has_many :courses, :dependent => :destroy

  include StrippedName

  attr_accessible :name

  validates :name,
      :uniqueness => { :case_sensitive => false, :scope => :vote_id },
      :length => { :minimum => 3 }

  def as_json options = {}
    {
      :id => id,
      :name => name,
      :vote_id => vote_id
    }
  end
end
```

Listing 3: Dynamische Klassen in Ruby

Listing 3 beschreibt ein Model, der im Rahmen der Bachelorarbeit entwickelten Anwendung. Auffällig ist, dass lediglich eine Methode *as_json* deklariert wird. Alle weiteren Methoden der Klasse *Domain* sind entweder vererbt oder mittels Klassenmakro oder Modul erstellt. Module erlauben dabei das Hinzufügen bestimmter Funktionalitäten zu beliebigen Klassen. Im Gegensatz zu C++ unterstützt Ruby keine Mehrfachvererbung. Das hier eingebundene

Modul hat den Namen *StrippedName*. Weiterhin werden verschiedene Klassenmakros eingesetzt, um bestimmte Werte beim Abspeichern des Models zu validieren. Diese Makros werden beim Interpretieren der Klasse im Kontext der Klasse aufgerufen und modifizieren die Klasse selbst.

2.1.2 Model und Datenbank

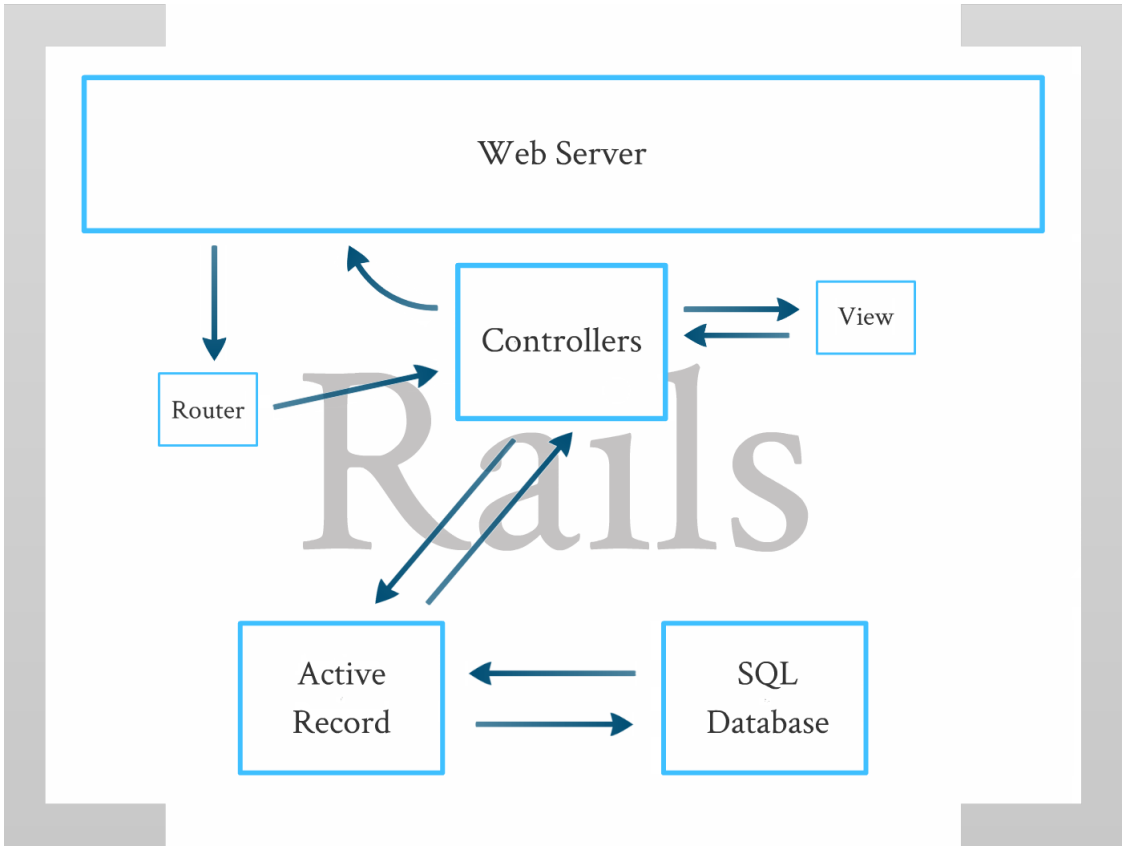
Rails verwendet ein eigenes Object Relational Mapper (ORM) namens Active Record, welches eine abstrakte Sicht auf die Datenbank erlaubt. Dabei ist Active Record nicht auf einen bestimmten Datenbanktyp beschränkt, sondern kann dank diverser Adapter in nahezu jedem Datenbanksystem eingesetzt werden. [Val11] In Listing 3 wurde bereits ein von Active Record abgeleitete Klasse *Domain* präsentiert. Diese Klasse verwaltet alle Daten der Tabelle *domains*. Hier werden abermals die Prinzipien COC und DRY deutlich: Der Tabellenname wird zur Laufzeit aus dem Klassennamen extrahiert, ebenso werden Informationen über das Tabellenschema erst zur Laufzeit aus der Datenbank ausgelesen. Eine von Active Record abgeleitete Klasse verwaltet typischerweise genau eine Tabelle. Dabei ist jede Instanz dieser Klasse für genau eine Zeile zuständig. Dies spiegelt den Ansatz der ROA wieder.

Die Verantwortung über das Datenbankschema liegt damit nicht bei Active Record. In ROR wird das Datenbankschema sukzessive mittels Migrations erzeugt. Jede Migration manipuliert das Schema auf eine bestimmte Art und Weise zu einem genau definierten Zeitpunkt. So kann eine Migration beispielsweise eine neue Tabelle und neue Relationen einfügen. Auch sind diese Migrationen nicht an bestimmte Datenbankentypen gebunden. Somit ist es ein Leichtes auf veränderte Anforderungen im Datenbankdesign zu reagieren oder voreilige Veränderungen des Schema zurückzunehmen. Migrations können wie fast jede ROR-Komponente mittels Scaffolding erzeugt werden. Dazu wird im Falle einer Migration der Name ebendieser an einen Generator übergeben, der dann alle benötigten Dateien inklusive Timestamp erstellt.

Active Record unterstützt Create, Read, Update and Delete (CRUD) durch Ruby-Methoden. Es ist keine direkte Datenbankinteraktion von Nöten. Obwohl direkte SQL Abfragen prinzipiell möglich und in Ausnahmefällen unvermeidbar sind, stellt ein konsequenter Verzicht auf nativen SQL Code den effektivsten Schutz gegen SQL Injections dar.

Außerdem unterstützt Active Record Transaktionen und die Validierung von Datenbankfeldern durch Klassenmakros. Beispielsweise wird in Listing 3 die Eindeutigkeit des Attributs *name* innerhalb eines Wahlvorgangs (spezifiziert mittels *vote_id*) garantiert.

Abbildung 1: Rails Architektur



2.1.3 Controller und Routing

Controller stellen in Rails die Schnittstellen für HTTP Requests dar. Pro Anfrage wird durch den Router genau eine bestimmte Controller-Methode (genannt *Controller-Action*) aufgerufen. Diese erhält alle vom HTTP Request übermittelten Informationen, wie zum Beispiel GET- und POST-Parameter. Der Controller konstruiert dann typischerweise nach Interaktionen mit dem Model eine HTTP Response, der Response-Body kann dabei durch ein View-Template erzeugt werden.

Abbildung 1 visualisiert die für Rails Anwendungen typische Architektur.

Alle Routes werden in einer zentralen Datei verwaltet und binden Actions an das Matching von regulären Ausdrücken auf die Unified Resource Locator (URL). Rails unterstützt eine an Resourcen orientierte Architektur. Wichtigster Bestandteil von ROA sind Representational State Transfer (REST) Schnittstellen. Dabei wird jede Aktion zu einer Resource, also zum Beispiel einem Rails Model, an eine bestimmte URL gebunden und mit der entsprechenden HTTP Methode versehen.

Tabelle 1 zeigt beispielhaft eine REST Schnittstelle für die Ressource Äpfel. Es können aber auch nur eine Teilmenge oder noch weitere URLs enthalten sein, zum Beispiel zum

Tabelle 1: REST Schnittelle

GET	/apples	Liste von Äpfeln anzeigen
POST	/apples	neuen Apfel mit übergebenen Daten erzeugen
GET	/apples/:id	einzelnen Apfel anzeigen
PUT	/apples/:id	bestimmten Apfel modifizieren
DELETE	/apples/:id	einen Apfel löschen

Anzeigen eines entsprechenden Formulars. Weil wir uns hier aber auf den Gebrauch von REST mit Javascript Object Notation (JSON) beschränken, ist es in diesem Rahmen nicht nötig, darauf weiter einzugehen.

2.1.4 Asset Pipeline

Während die Asset Pipeline in klassischen Rails-Anwendungen eher eine untergeordnete Rolle spielt, ist sie für Single Page Application (SPA) ein entscheidendes Feature. Hier werden alle statischen Dateien, die der HTTP-Server ausliefert, organisiert. Dazu gehören insbesondere JS- und CSS-Dateien.

Diese Dateien können auch in anderen Sprachen beschrieben werden und erst vor dem Ausliefern übersetzt werden. So unterstützt Rails hier den Einsatz von CoffeeScript und Syntactically Awesome Stylesheets (SASS), welche wiederum in einer ERB-Datei gekapselt werden können. Die Kapselung in ERB ist allerdings nur bedingt sinnvoll, weil es sich dann nicht mehr um statische Dateien handelt - der Inhalt kann prinzipiell von Request zu Request variieren, damit fällt die Option des Caching weg. [Hana]

Im Entwicklungsmodus (Development Environment) werden die Dateien einzeln übersetzt und an den Client ausgeteilt. Dabei ist ein Browser-Caching dieser Dateien möglich, folglich muss immer nur die gerade manipulierte Datei neu geladen werden. Das Caching erfolgt seit Rails 3.1 anhand von Fingerprints, die laufend für jede Datei vergeben werden und die Eindeutigkeit des Dateinamens garantieren.

Im Produktionsmodus (Production Environment) hingegen werden die Dateien entsprechend ihrer Abhängigkeiten kompiliert und minimiert. Die resultierenden Dateien müssen dann nur einmal vom Client geladen werden und unterstützen so eine Anwendung mit geringer Latenz.

2.1.5 Qualitätssicherung durch Testing

ROR motiviert den Programmierer, seine Anwendung regelmäßig zu testen. Werden Models oder Controller durch Scaffolding angelegt, so erstellt Rails automatisch die zugehörigen Testfälle. Testing ist schon wegen der Dynamik der Programmiersprache Ruby unerlässlich,

um qualitativen Anforderungen gerecht zu werden. Dabei kann es sinnvoll sein, Test Driven Development (TDD) einzusetzen. Diese Arbeit setzt teilweise auf TDD, nicht jedoch für die Programmierung der Client-seitigen Backbone.js Anwendung. Tests ermöglichen in jedem Fall ein aggressives Refactoring und eine schnelle Reaktion auf geänderte Anforderungen.

Tests benötigen oftmals Beispieldaten, um durchgeführt werden zu können. Diese Beispieldaten werden in Rails durch Fixtures erzeugt. Fixtures versetzen die Test-Datenbank in einen vordefinierten Zustand, der über alle Tests gleich ist. Weil dies aber nicht immer ausreichend ist, werden Fixtures in vielen Rails Anwendungen durch dynamischere Konzepte ersetzt. Für diese Anwendung wird FactoryGirl eingesetzt. Dabei wird für jedes zu erstellende Model mindestens eine Factory erstellt. Diese Factories können dann innerhalb der Tests dynamisch angefordert werden und erzeugen die benötigten Datenbankeinträge. [Hanb] [RTH11]

```
FactoryGirl.define do
  factory :user do
    name 'Tilo Wiedera'
    date_of_birth { 24.years.ago }
  end
end

# building valid user object
# overriding name attribute
FactoryGirl.build(:user, :name => 'John Doe')
```

Listing 4: FactoryGirl

Es werden drei Typen von automatischen Tests unterschieden.

Unit Tests Models werden in ROR durch Unit Tests abgedeckt, wobei die Geschäftslogik der Anwendung getestet wird. Komplexe Validierungen und alle in den Models definierten Methoden werden hier getestet.

Functional Tests Bei Functional Tests wird pro Testfall genau ein HTTP Request generiert und ohne die Routing Tabelle zu konsultieren, direkt an die entsprechende Controller Action gesendet. Im Rahmen dieser Bachelorarbeit wurden Functional Tests eingesetzt, um das REST-Application Programming Interface (API) zu spezifizieren und zu testen.

Integration Tests Integration Tests decken komplette User Stories ab. Dabei werden typischerweise mehrere Requests pro Testfall generiert und sukzessive ausgeführt. Es wird nicht jedes Details der HTTP Response überprüft, insbesondere können dank Asynchronous Javascript and XML (AJAX) mehrere Requests/Responses auftreten. Hier kann das Framework Capybara eingesetzt werden, welches eine Simulation des Benutzers durch automatisierte Browsertests inklusive Javascript Interpreter unterstützt.

2.1.6 Rails Plugins

Die Sprache Ruby wird seit Version 1.9 mit einem eigenen Package Manager namens RubyGems ausgeliefert. [Wikf] Dieser Manager erlaubt das unkomplizierte Installieren von Ruby-Bibliotheken und ganzen Anwendungen, wie z.B. dem Ruby Build Werkzeug Rake und ähnelt damit Systemen wie dem Advanced Packaging Tool (APT).

Weiterhin existieren Gems, die speziell auf Rails zugeschnitten sind. Die benötigten Gems werden zentral in einer Datei mit dem Namen *Gemfile* für jede ROR Applikation abgelegt.

```
gem 'thin'                      # webserver
gem 'sqlite3'                   # database
gem 'email_validator'           # handles email validations
gem 'jquery-rails'              # jquery adapter
gem 'devise'                    # used for authentication
gem 'yard'                      # doc generator for ruby code
gem 'acts-as-taggable-on'       # tags describing the attributes of pupils
gem 'twitter-bootstrap-rails'   # bootstrap adapter
gem 'prawn_rails'               # used to render pdf files
gem 'colormath'                 # validates colors associated with rules
gem 'axlsx_rails'               # renders xlsx spreadsheets
gem 'roo'                       # parses xls, xlsx and csv spreadsheets
```

Listing 5: Auszug aus einem Gemfile

2.2 CoffeeScript

CoffeeScript wurde erstmal 2009 durch Jeremy Ashkenas veröffentlicht. Es handelt sich um eine Skriptsprache die zu Javascript übersetzt werden kann. CoffeeScript versucht die typischen Probleme im Umgang mit Javascript zu umgehen und bessere Les- und Schreibbarkeit zu erreichen. Seit Rails 3.1 wird CoffeeScript nativ unterstützt. [Wika] [Era12] Syntaktisch ähnelt CoffeeScript Sprachen wie Ruby und Python und übernimmt auch einige semantische Konzepte dieser Sprachen.

2.2.1 Grundlegende Syntax

CoffeeScript und auch Javascript sind dynamische Programmiersprachen, die Sprachen verzichten genau wie Ruby auf statische Typisierung. CoffeeScript erzwingt im Gegensatz zu Javascript ein korrektes Einrücken, Blöcke müssen also genauso wie in Python nicht explizit eingeleitet und terminiert werden. Außerdem sind Semikola und Klammern bei Funktionsaufrufen und Kontrollstrukturen optional.

Variablen Lokale Variablen werden in Javascript mit dem Schlüsselwort *var* eingeleitet. Entfällt dieses Schlüsselwort, so wird die Variable global angelegt. Dies geschieht in Javascript-Anwendungen oftmals ungewollt und ist als *global namespace pollution* bekannt. [Cro08] CoffeeScript legt nun alle Variablen lokal an,. Soll eine Variable global verfügbar sein, so muss sie explizit an ein globales Objekt gebunden werden.

Kontrollstrukturen Schleifen werden in CoffeeScript meist direkt auf eine Collection angewandt. Soll eine einfache Zählschleife realisiert werden, so wird zunächst ein entsprechendes Array als Range-Literal deklariert. Im Allgemeinen dürfen die Kontrollstrukturen in CoffeeScript sowohl vor als auch hinter der zu kontrollierenden Anweisen stehen, wobei auf die korrekte Einrückung zu achten ist.

```
if a isnt b
  console.log 'not equal'
else
  console.log 'equal'

console.log(i) for i in [1..10]

for str in ['Hello', 'World']
  console.log str
```

Listing 6: Kontrollstrukturen in CoffeeScript

Listing 6 zeigt den Einsatz verschiedener Strukturen, es wird sowohl über eine Range als auch über ein Array iteriert.

Funktionen Ein mächtige Eigenschaft von Javascript ist es, Funktionen als Variablen zu betrachten. Ruby setzt diese Eigenschaft nur bedingt um, hier ist jeweils eine explizite Konvertierung zu einem Proc Objekt nötig. CoffeeScript übernimmt nun diese Funktionsvariablen von Javascript und ermöglicht so die einfache Übergabe von Funktionen als Parameter. Damit sind Funktionen in CoffeeScript genau wie in Javascript *first class citizens*.

```
fib = (n) -> # function declaration, parameter: n
  if n > 1
    fib(n-1) + fib(n-2) # recursive function call
  else
    n # no need for return statement

test = fib # just an assignment of functions

test 3 # actual function call
```

Listing 7: Funktionen und Funktionsvariablen in CoffeeScript

In Listing 7 wird deutlich, wie kompakt sich eine Funktion in CoffeeScript definieren lässt: Es sind weder *function* noch *return* Statements nötig.

Weitere Konstrukte CoffeeScript zeichnet sich durch eine Reihe weiterer syntaktischer Konzepte aus, auf die hier kurz eingegangen werden soll. Oftmals sind die Änderungen im Vergleich zu Javascript nur syntaktischer Zucker, schaffen in ihrer Summe aber eine wesentlich verbesserte Les- und Schreibbarkeit.

Vergleichsoperatoren prüfen in CoffeeScript auf echte Gleichheit während in Javascript ein typübergreifender Vergleich stattfindet, der teilweise widersprüchliche Ergebnisse liefert. [Croa]

```
'' == '0'          // false
0  == ''           // true
0  == '0'          // true
false == 'false'   // false
false == '0'       // true
false == undefined // false
false == null      // false
null == undefined  // true
" \t\r\n " == 0    // true
```

Listing 8: Vergleichschaos in Javascript

Objekt Literale sind in Javascript äquivalent zu Hashmaps und können dank JSON schon in Javascript sehr kompakt geschrieben werden. CoffeeScript erreicht durch das Entfernen von Klammern und Kommata eine noch einfachere Syntax.

String Interpolierung ist analog zu Ruby mit dem #{} Operator möglich.

Arrays und Hashes könnnen mit dem *in* Operator nach der Existenz eines bestimmten Wertes gefragt werden. Ranges können nicht nur in Schleifen, sondern auch als Alias für Javascript's *splice* beim Zugriff auf Array-Elemente eingesetzt werden.

Funktionen können mit variabler Parameterzahl (Splat Arguments) und mit Standard-Argumenten definiert werden. Um eine Funktion an den ihre Deklaration umgebenden Kontext zu binden kann der => Operator verwendet werden.

Existenz-Operatoren prüfen in CoffeeScript auf die Werte *null* und *undefined*.

```
# true if bar exists
bar?

# execute func if func exists
func?()

# execute bar.func if bar exists
bar?.func()

# assign a value to foo if it doesnt already exist
foo ?= 'value'

# evaluates to foo if foo exists, bar otherwise
foo ? bar
```

Listing 9: Existenzoperatoren in CoffeeScript

2.2.2 Klassen

Javascript ist eine objektorientierte aber klassenlose Programmiersprache. Objekte werden im Gegensatz zu klassischen objektorientierten Sprachen durch Prototypen konstruiert. Dabei kann ein Objekt insbesondere immer nur von einem anderen konkreten Objekt erben.

```
class MyClass           # class definition
  y: 3                  # instance variable

  constructor: ->       # beeing called upon instanciation
    @name = 'MyObject'  # creates a new instance variable

  bar: (x) ->           # defines a method with param x
    @y * x              # returns this.y * x

myObj = new MyClass()   # instanciates a new object
myObj.bar 2             # returns 3 * 2 = 6
```

Listing 10: Klassen in CoffeeScript

CoffeeScript führt das Schlüsselwort *class*, mit dem Klassen als Abstraktion der Javascript Prototypes zur Verfügung stehen, ein. Klassen können explizite Konstruktoren erhalten,

einfach voneinander erben, vererbte Methoden überschreiben, super-Methoden aufrufen und Klassenvariablen (u.a. Klassenmethoden) definieren. Die Klassendefinition wird dann in einen Javascript Konstruktor übersetzt, insbesondere sind die Klassen und Objekte weiterhin zur Laufzeit vollständig manipulierbar. [Bur11]

```
class MyOtherClass extends MyClass # creates a new, derived class

  constructor: (@y) ->             # constructor overrides y attribute
    super()                        # super calls are allowed in all methods

  @myCLassMethod: ->               # "this" refers to the class,
                                   # thus defining a class method
    console.log 'Kontext: #{@}'    # prints "this" to the console

MyOtherClass.myCLassMethod()       # calls the class method

myOther = new MyOtherClass(10)     # creates an instance of
                                   # MyOtherClass with y = 10
myOther.bar 2                      # returns 10 * 2 = 20

MyClass::foo = true                # adds a new attribute to the super class
                                   # using the prototype operator

myOther.foo                        # returns true
```

Listing 11: Vererbung in CoffeeScript

Listing 10 und Listing 11 zeigen die wesentlichen Konzepte im Bezug auf Klassen in CoffeeScript. Der @ Operators ist im Prinzip nur syntaktischer Zucker, um *this* abzukürzen und kann auch im Kontext der Klasse zur Definition von Klassenvariablen verwendet werden.

2.2.3 Best Practices

Um CoffeeScript sind mittlerweile eine ganze Reihe verschiedener Konzepte entstanden, die sich nicht direkt in der veränderten Syntax widerspiegeln. Teilweise wurden Paradigmen aus Javascript übernommen, teilweise neue Prinzipien erarbeitet. Einige der wichtigsten Konventionen werden im folgenden näher betrachtet. [Mac13] [Bra]

Private Variablen CoffeeScript unterstützt genau wie Javascript keine privaten Variablen oder Attribute, durch Closures kann aber eine Kapselung auf Basis des Kontext erreicht werden. Dies bedeutet im Hinblick auf Klassen, dass private Attribute und Funktionen innerhalb des Konstruktors deklariert werden müssen. Insbesondere sind diese Funktionen

nach dem Übersetzen nicht Teil der Prototyp Definition, was Probleme bezüglich der Vererbung verursacht. Weiterhin sind alle öffentlichen Methoden als *first class functions* manipulierbar, damit auch Getter und Setter Methoden. Einzige Einschränkung ist, dass der Kontext des Konstruktors hier wegfällt und die privaten Variablen nicht mehr referenziert werden können. Es ist aber durchaus möglich, den Rückgabewert einer Getter Methode von außen zu überschreiben. [Crob]

Nachdem also keine privaten Attribute mit sinnvollem Zugriff in CoffeeScript möglich sind, wird im Weiteren die Konvention verwendet, private Member mit einem Unterstrich-Präfix zu versehen. Es liegt dann in der Verantwortung des Programmierers, diese Member nur nach einem @ (this) zu verwenden.

Method Call Chaining Oftmals liefern Funktionen einen Verweis auf den Funktionskontext selbst als Rückgabewert. Dies dient dazu, das sukzessive Aufrufen von Funktionen eines bestimmtes Objekts ausdrucksstark und knapp darzustellen.

```
bar.foo()
bar.boo()
bar.func()

# aquivalent if each procedure returns its context
bar.foo().boo().func()
```

Listing 12: Verkettung von Funktionsaufrufen in CoffeeScript

Module Module dienen der Vermeidung von globalen Namenskonflikten und basieren ebenfalls auf Javascript Closures. Dabei werden die Inhalte einer Datei jeweils innerhalb einer anonymen Funktion, die sofort nach ihrer Definition aufgerufen wird, gekapselt. Über den Rückgabewert der Funktion wird die Schnittstelle des Modul nach außen definiert. Ttypischerweise werden hier Klassen oder Funktionen zurückgegeben. Projekte wie CommonJS und Asynchronous Module Definition (AMD) versuchen dieses Prinzip zu standardisieren und werden mittlerweile in vielen Javascript-Umgebungen eingesetzt.

Mixins CoffeeScript erlaubt keine Mehrfachvererbung. Manchmal teilen aber auch nicht verwandte Klassen ähnliche Funktionalitäten. Abhilfe schaffen dabei Mixins, die dynamisch Methoden und Attribute zu einem Objekt oder dessen Prototyp hinzufügen.

Wie aus Listing 13 ersichtlich wird, kommen Mixins ohne jegliche syntaktische Neuerung aus. Hier zeigt sich die Stärke von Javascripts Prototypes und CoffeeScripts Klassen, die selbst als Objekte behandeln werden.

```
# apply mixin to object
extend = (obj, mixin) ->
  obj[name] = method for name, method of mixin
  obj

# apply mixin to class / prototype
include = (classDef, mixin) ->
  extend classDef.prototype, mixin

# usage, anonymous mixin
include Parrot,
  isDeceased: true

(new Parrot).isDeceased # => true
```

Listing 13: Mixins in CoffeeScript [Mac13]

2.3 Javascript Bibliotheken

Nachdem die Grundlagen von CoffeeScript vorgestellt wurden, sollen zwei Javascript Bibliotheken knapp vorgestellt werden, welche dem Programmierer viele Routinen vereinfachen. Weil Javascript einen so kleinen Funktionsumfang besitzt, existieren kaum native Funktionen, die Programmierern die Arbeit mit dem Browser erleichtern. Dies ist auch gar nicht immer nötig, zum Beispiel kann Javascript auch auf Seite des Servers mit node.js eingesetzt werden.

Für diese Bachelorarbeit aber wird das gesamte Frontend mit Javascript umgesetzt, folglich ist hier ein Einsatz dieser Bibliotheken nicht nur sinnvoll sondern fast schon unabdingbar. Alle Javascript Bibliotheken können genauso mit CoffeeScript eingesetzt werden, da die Sprache erst übersetzt und dann interpretiert wird.

2.3.1 jQuery

jQuery wurde erstmals 2006 von John Resig veröffentlicht und zielt darauf ab, erweiterte Funktionalität für die Client-seitige Programmierung zu schaffen. Dazu zählen vereinfachte Dokumententraversierung und -Manipulation, Eventhandling, Animationen und eine einheitliche AJAX Schnittstelle. jQuery wird mittlerweile auf etwa 65% der 10.000 meistbesuchten Websites eingesetzt. jQuery wird wie andere, beliebte Bibliotheken auch, häufig über ein Content Delivery Network (CDN) ausgeliefert. [Wikb]

Die jQuery Familie umfasst etliche Plugins und einige erweiternde Bibliotheken wie jQuery Mobile oder jQuery UI, auf die hier nicht näher eingegangen werden soll. Wichtigster Bestandteil von jQuery im Rahmen dieser Arbeit ist die selector enginge namens Sizzle, die auch als Standalone zur Verfügung steht. Für weitere Informationen sei hier auf die

Tabelle 2: wichtigste Parameter der globalen jQuery Funktion

Selektor	Globales Matching gegen den Selektor
Funktion	Ausführen der Funktion bei entsprechendem *readystatechange* Event
HTML-Element	Bestehendes Hypertext Markup Language (HTML)-Element in jQuery Objekt umwandeln
HTML-Element-String	Erstellen eines neuen Elements, wird noch nicht ins Document Object Model (DOM) eingefügt
jQuery-Element	jQuery Element Klonen

Literatur verwiesen. [Fre12] [BKR08] Das im Rahmen dieser Arbeit verwendete Framework Backbone.js benötigt entweder jQuery oder Zepto, eine etwas leichtgewichtigere Bibliothek, die etwa den gleichen Anforderungen wie jQuery genügt.

Die jQuery Bibliothek exportiert ein einziges globales Objekt: Die jQuery Funktion. Auf diese Funktion kann auch über das globale *$*-Objekt zugegriffen werden. Ddies ist möglich, weil auch *$* ein gültiger Variablenname in Javascript ist. Zur Wahrung der Kompatibilität mit anderen Bibliotheken lässt sich aber eine explizite Verwendung ohne *$* erzwingen. Die jQuery Funktion hat je nach übergebenem Parameter sehr unterschiedliche Bedeutung.

Tabelle 2 verdeutlicht die verschiedenen Möglichkeiten die jQuery Funktion aufzurufen. Die jQuery Funktion deckt nur die wichtigsten Funktionalitäten direkt ab. jQuery Selektoren liefern jQuery Objekte zurück, auf die eine Vielzahl weiterer Operationen angewandt werden kann.

Sizzle Ziel von Sizzle ist es, DOM Selektoren ähnlich den Cascading Stylesheets (CSS) Selektoren anzubieten. Javascript unterstützt in der Browserumgebung nur sehr primitiven Zugang zu den DOM Elementen, z.B. durch die Methode *getElementById* des *document* Objekts. Sizzle wurde erstmal 2009 eingeführt und 2012 komplett überarbeitet. [Wikb]

Listing 14 zeigt einen kleinen Ausschnitt der möglichen jQuery Selektoren. Für eine umfangreiche Dokumentation lohnt sich ein Blick auf [Gitb] und [eEG+]. Alle gelisteten Ausdrücke liefern ein gültiges jQuery Objekt zurück, selbst wenn kein entsprechendes HTML Element selektiert werden konnte.

Events Javascript unterstützt im Kontext des Browser eine große Menge vordefinierter Ereignisse. [Net] jQuery bietet eine einheitliche Schnittstelle um auf die entsprechenden Events zu reagieren. Dazu wird die Methode *on* eines jQuery Objekts aufgerufen. Soll ein Event dokumentenweit registriert werden, so wird der Callback an das globale *document*

```
# select div with id="wrapper"
$('div#wrapper')

# select form with css classes "edit" and "bookmark"
$('form.edit.bookmark')

# select url input of said form
$('form.edit.bookmark input[type="url"]')

# select each cell in every second table row
$('table.bookmarks tr:odd td')
```

Listing 14: jQuery Selektoren in CoffeeScript

Objekt gebunden. Callbacks können registriert und wieder deregistriert werden.

Wird im Browser ein Event beispielsweise durch einen Klick auf einen Button ausgelöst, so unterteilt sich die Bearbeitung des Events in zwei Phasen. Es werden sukzessive alle betroffenen Elemente, beginnend bei dem Top Level Element des DOM über das Ereignis informiert (Capturing). Ist das eigentliche Zielelement erreicht, wird das Event erneut in aufsteigender Reihenfolge ausgelöst (Bubbling). Dies hängt mit der abweichenden Behandlung von Events in verschiedenen Browsern zusammen. Während ein Großteil der Browser mit Bubbling arbeitet, unterstützen ältere Version des Internet Explorer Bubbling nur für bestimmte Events. An diesen Stellen simuliert jQuery das Event Bubbling. jQuery hat den Anspruch, eine über alle Browser eine einheitliche Schnittstelle zu bieten. [Fou]

AJAX AJAX erlaubt das asynchrone Laden von Daten im Hintergrund. Insbesondere kann der Nutzer dabei wie gewohnt auf der Seite weiter surfen. In Javascript sind XML-HTTP-Request Objekte für die Durchführung dieser Vorgänge verantwortlich. jQuery bietet Funktionen, denen im Prinzip nur URL und Callback übergeben werden müssen um einen gültigen AJAX-Request zu erzeugen. Dabei ist auch ein automatisches Parsen von empfangenem JSON möglich. Außerdem wird globale Fehlerbehandlung durch die entsprechenden Ereignisse unterstützt.

Animation Viele jQuery Effekte lassen sich seit dem neuen CSS3 Standard auch ohne direkte Manipulation der CSS-Eigenschaften erreichen. Wann immer möglich wurde dies innerhalb dieser Arbeit, gemäß dem Prinzip der Trennung von Anwendungslogik und Darstellung, auch umgesetzt.

Listing 15 verdeutlicht neben der Animationsmechanik in jQuery auch das typische method-chaining durch Rückgabe des eigenen Kontextes.

```
# fade dialog after 3 seconds
$('div#dialog').delay(3000).fadeOut()

# same effect with css animations
# requires respective css code for class fadeOut
applyFadeOut = ->
  $('div#dialog').addClass 'fadeOut'

setTimeout applyFadeOut, 3000
```

Listing 15: jQuery Animationen in CoffeeScript

Tabelle 3: Javascript Einbettung in EJS-Templates

$<\%\ ...\ \%>$	keine Ausgabe
$<\%=\ ...\ \%>$	direkte Ausgabe in den HTML Code
$<\%-\ ...\ \%>$	Ausgabe in den HTML Code nach Escaping

2.3.2 Underscore.js

Underscore wird durch Jeremy Ashkenas entwickelt und als *utility-belt library for JavaScript* bezeichnet. Es ist neben jQuery eine Voraussetzung für den Einsatz von Backbone. [Ashb] Wichtigster Einsatzbereich im Rahmen von Backbone ist das Rendering von EJS Templates und die Traversierung sowie Manipulation von Backbone Collections.

Templates Templates stellen sich in Underscore analog zu den ERB Templates von Rails dar. Es Handelt sich um HTML Code mit eingebetteten Javascript Statements.

Tabelle 3 listet die verschiedenen Möglichkeiten, Javascript Code in Underscore Templates einzubinden. Es handelt sich also im Gegensatz zu Bibliotheken wie Mustache um Logik-behaftete Templates. EJS-Templates werden zur Laufzeit in eine Funktion übersetzt, die dann mit beliebigen Schlüssel-Wert-Paaren aufgerufen werden kann. Bei Backbone entsprechen diese Paare in der Regel den Attributen eines bestimmten Models.

Collections Collections sind ein zentraler Bestandteil von Backbone und werden später noch detaillierter vorgestellt. Underscore vereinfacht das Iterieren und Manipulieren der in einer Collection abgelegten Models. Gerade bezüglich dieses Aspekts bedient sich Underscore der Nomenklatur von Ruby. Es existieren die Methoden *each*, *map*, *reduce*, *filter*, *where*, *sortBy* und ähnliche.

2.4 Backbone.js

Backbone, ebenfalls durch CoffeeScript Entwickler Jeremy Ashkenas im Jahr 2010 veröffentlicht, verleiht großen Javascript Applikation eine MVC Architektur. Technisch betrachtet handelt es sich bei Backbone auch nur um eine Javascript-Bibliothek. Richtig eingesetzt kann dieses Framework aber einen starken, positiven Einfluss auf die Wart- und Erweiterbarkeit des resultierenden Programms haben. [Asha]

Backbone macht keine starren Vorgaben bezüglich der unterstützten Strukturen. Es steht dem Programmierer frei beispielsweise nur die angebotenen Model Klassen zu verwenden und alles weitere von Grund auf selbst zu entwickeln. Damit ist es möglich und oftmals sinnvoll, das Framework in den verschiedensten Bereichen, von sehr kleinen Applikationen wie einer Todo-Liste bis hin zu umfangreichen Projekten wie der Website von *USA Today* oder *DocumentCloud*, einzusetzen.

Backbone unterteilt eine typische Anwendung in vier große Bereiche: Routers, Views, Models und Collections. Es handelt sich um eine Form des Pattern Model 2. [Wikc] Dabei nehmen Router aufgerufene URLs entgegen und rendern dann die entsprechenden Views, die als Schnittstelle zwischen Models und DOM fungieren. Damit integriert Backbone die typische Funktionalität eines Controllers in die View. Man spricht in diesem Zusammenhang auch von Model-View-Star (MV*).

2.4.1 Typische Probleme ohne SOC

Während klassische Websites einen Großteil der Arbeit auf dem Server erledigen, geht der Trend neuerdings in Richtung umfangreicher Webclients. Bei diesen Anwendungen handelt es sich oft um SPAs. Vorteil ist, dass nur die eigentlichen Daten mit dem Server kommuniziert werden müssen und damit nicht die gesamte Seite jedes mal neu geladen wird. Das bedeutet zum einen weniger Arbeit für den Server und zum anderen eine effizientere Nutzung der Verbindung. Dabei wird natürlich ein Mindestmaß an Leistung auf dem Client vorausgesetzt. Dieses Maß können heute aber schon einfache Smartphones erfüllen.

jQuery bietet zwar viele Funktionen, trägt aber wenig zur verbesserten Strukturierung des Programms bei. Nun würde man ohne Backbone entweder ein eigenes, Backbone ähnliches, System entwickeln und warten müssen um eine hinreichend strukturierte Anwendung zu erreichen oder man würde größtenteils auf eine Separartion of Concerns (SOC) verzichten. In diesem Fall wäre der Programmfluss nur sehr schwer nachvollziehbar und die Javascript Ereignisse, die prinzipiell an beliebiger Stelle im Code versandt und empfangen werden können, würden zu einer Art asynchronem Spaghetti Code führen.

Somit bietet sich Backbone als sinnvoller Architekturgeber von Fat Webclients an. [Osm12]

2.4.2 Routing

Router funktionieren ähnlich wie in Rails, dabei werden reguläre Ausdrücke an bestimmte Callbacks gebunden. Diese Callbacks sind im Gegensatz zu Rails nicht Controller Actions, sondern beliebige Methoden des Routers. Die Methoden instanziieren dann die entsprechenden Views. Während eine SPA theoretisch auch ganz ohne Router auskommen könnte, bieten diese einige signifikante Vorteile. Um Bookmarks, das Teilen von Hyperlinks oder eine Browserhistorie zu erzeugen, sind persistente URLs zwingend erforderlich.

Eine Backbone Applikation kann beliebig viele Router definieren. Oft ist es aber der Übersicht halber sinnvoll, nur einen Router pro Anwendung oder Modul umzusetzen.

2.4.3 Models und Collections

Models stellen die Grundlage unserer Geschäftslogik dar. Sie erlauben das Speichern von beliebigen Schlüssel-Wert-Paaren. Oftmals benötigt man eine Menge von Models gleichen oder ähnlichen Typs, zum Beispiel für eine Auflistung von Produkten. Dieses Problem wird in Backbone durch Collections gelöst. Collections sind eine Menge von Models, die bestimmte Funktionalitäten für diese Models bündeln. Dazu gehören insbesondere Ereignisbehandlung und Server-Synchronisation. Auf Collections können außerdem die weiter oben behandelten Underscore utility methods angewandt werden. Während Collections prinzipiell beliebige Models enthalten können, sollten sie immer nur Models eines bestimmten Typs enthalten.

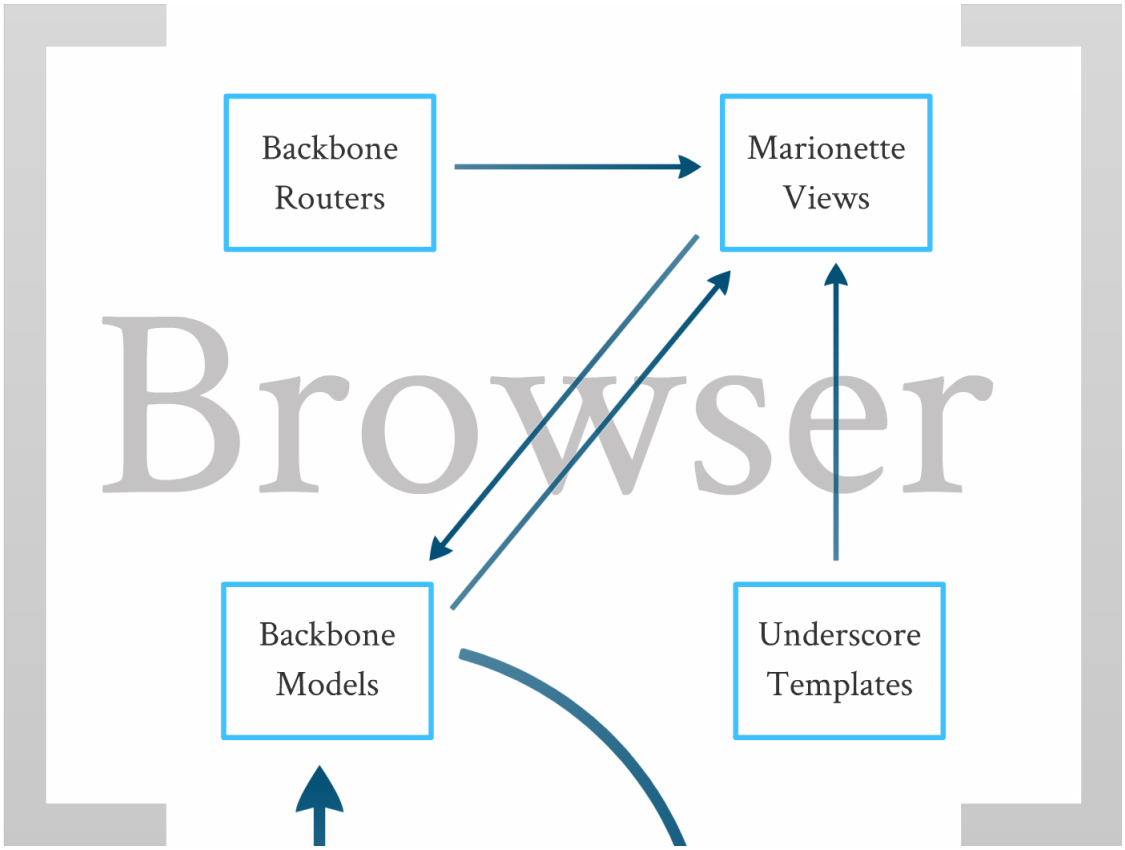
Models triggern automatisch bestimmte Events, wenn ihre Hashes manipuliert werden. Dabei können auch Callbacks an die Manipulation einzelner Schlüssel gebunden werden. Backbone Collections unterstützen REST-Interfaces für die Synchronisation mit dem Server. Durch Überschreiben von *Backbone.sync* können andere Schnittstellen genutzt werden.

Models unterstützen außerdem Standard-Werte für auf der Client-Seite neu erstellte Models und können genau wie in Rails Validationen durchführen. Weil Validations oftmals mehr Daten benötigen als auf dem Client geladen sind (insbesondere um Konflikte mit anderen Datensätzen zu erkennen) wird in der hier entwickelten Anwendung auf Client-Side-Validations verzichtet. Stattdessen wird eine entsprechende Anfrage an den Server gesendet. Falls die Daten nicht gespeichert werden können wird eine genaue Fehlermeldung vom Server geliefert.

2.4.4 Views

Views stellen in Backbone die Verbindung zwischen Models und dem DOM her. Views selbst referenzieren typischerweise HTML-Templates, enthalten in der Regel aber auch

Abbildung 2: Backbone-Marionette Architektur



einfache Logik. Die Bezeichnung View führt gerade bei Einsteigern oft zu einer gewissen Verwirrung. In einer klassischen MVC-Architektur wird die durch die Backbone View abgedeckte Funktionalität durch den Controller abgedeckt. Views realisieren das Observer Pattern, dabei werden Änderungen des Models beobachtet und gegebenenfalls das DOM aktualisiert. Jede Backbone View enthält ein `el` Attribut, dieses Element wird beim Rendern der View mit Inhalt gefüllt, typischerweise mit einem ausgewertetem Template.

Weiterhin unterstützen Views mit einem Event Hash das automatische Binden von View spezifischen Ereignissen an Methoden der View. Über diese Events wird beispielsweise das asynchrone Versenden von HTML Formularen realisiert.

2.4.5 Marionette.js

Marionette ist ein Framework, das auf Backbone aufbaut und dabei verschiedene Patterns realisiert, die sich in bestehenden Backbone Applikationen als sinnvoll erwiesen haben. Genau wie Backbone erzwingt Marionette nicht, dass alle dieser Konzepte in einer bestimmten Anwendung genutzt werden. Zu den für diese Anwendung wichtigsten Patterns gehören die spezialisierten View Typen, welche den Boilerplate Code für Views reduzieren und damit dem DRY Prinzip folgen. So existieren *ItemView* und *CollectionView*, um Collections

beziehungsweise einzelne Models zu visualisieren. Dabei werden auch die entsprechenden Eventhandler automatisch gesetzt. Wird zum Beispiel die Collection vom Server neu geladen, so wird die gesamte CollectionView erneut gerendert. Marionette verfügt über weitaus mehr Konzepte, auf die hier nicht im Details eingegangen werden kann. Stattdessen sei hier auf die Literatur verwiesen. [Sul13a] [Sul13b]

3 Organisation und Entwicklungsmodell

Dieses Kapitel widmet sich der organisatorischen Strukturierung der Arbeit. Dabei werden die Aufgabenplanung und das Vorgehensmodell kurz erläutert.

3.1 Agile Softwareentwicklung

Es wurde für diese Arbeit nicht strikt ein bestimmtes Entwicklungsmodel verfolgt sondern sich insgesamt an den Methoden der agilen Softwareentwicklung orientiert. Im Allgemeinen steht für eine Bachelorarbeit nur ein sehr begrenzter Zeitrahmen zur Verfügung. Weil außerdem nur ein einziger Entwickler intensiv an dem Programm arbeitet, ist ein Planungsoverhead, der bei vielen klassischen Entwicklungsmodellen zu schnell eintreten würde, zu vermeiden. Der Begriff *Agile Software Development* wurde 2001 durch das *Agile Manifesto* eingeführt. [Wikh] Vorteile der agilen Entwicklung werden insbesondere bei kurzen Projektlaufzeiten und unklaren Anforderungen deutlich. Im folgenden werden die im *Agile Manifesto* genannten Grundsätze näher erläutert und in Bezug zur Arbeit gesetzt.

3.1.1 Agile Manifesto

Das Agile Manifesto wurde erstmal 2001 vorgestellt, es legt den Grundstein der Agilen Softwareentwicklung durch Priorisierung verschiedener im Hinblick auf Softwareentwicklung in Konflikt zueinander stehender Aspekte. [agi]

Individuals and interactions over processes and tools Dieser Grundsatz ist primär im Hinblick auf Teamwork interessant. Es sollten kooperierende Entwickler grundsätzlich eine intensive Kommunikation untereinander aufrecht erhalten. Es ist von zentraler Bedeutung, dass Software-Werkzeuge diese direkte Kommunikation niemals ersetzen können.

Working software over comprehensive documentation Funktionierende Software wird vom Kunden in der Regel höher geschätzt als ein Dokument, dass die geplanten Funktionalitäten theoretisch erläutert. Das gilt vor allem für Projekte mit kurzer Laufzeit. Durch ein frühes Bereitstellen, selbst von nur teilweise funktionsfähiger Software, können schnell Fehler sowohl seitens der Entwickler als auch seitens der Kunden entdeckt und behoben werden.

Das Graf-Stauffenberg-Gymnasium hat im Rahmen dieser Arbeit die Anforderung gesetzt, die Software umfassend zu dokumentieren. Um ein agiles Vorgehen besser zu unterstützen, wurde die Dokumentation des Quellcodes jeweils erst nach entsprechenden Akzeptanztests vorgenommen. Dabei wurde die Dokumentation nicht zur Definition von Anforderungen und Funktionalitäten genutzt.

Customer collaboration over contract negotiation Wenn die Softwareanforderungen nicht im Detail vorliegen, ist es umso wichtiger, diese durch Rücksprache mit dem Kunden zu erörtern. Oftmals ist es gar nicht möglich, zu Projektstart alle Anforderungen zu erfassen, was auch an einem unvollständigen Problemverständnis des Kunden liegen kann. So ist es nicht nur die Aufgabe des Entwicklers, Software nach Vorgabe des Kunden zu entwickeln sondern ihn auch in Bezug auf die zu lösenden Probleme zu schulen.

Ein Vertrag über die Entwicklung der Software ist deshalb zwar wichtig, aber er kann nicht alle Anforderungen im Detail enthalten. Eine kontinuierliche Kommunikation zwischen Entwicklern und Kunden ist unerlässlich, um Anforderungen und Lösungen laufend zu überprüfen und anzupassen.

So wurden auch im Rahmen dieser Arbeit mehrere Akzeptanztests durchgeführt. bei Projektstart wurden die Anforderungen in einem Kickoff Meeting grob erläutert und ein UI-Prototyp präsentiert.

Responding to change over following a plan Es ergeben sich verändernde Anforderungen aus der nicht abschließenden Anforderungensanalyse zu Projektbeginn, auf diese Änderungen muss nun möglichst zeitnah reagiert werden können. Auch nutzen moderne Anwendungen oftmals viele Bibliotheken und Frameworks die ebenfalls zeitlichen Änderungen unterliegen. Wichtigstes Prinzip zur Einhaltung dieses Grundsatzes ist DRY.

Aus den oben erörterten Prinzipien lassen sich nun konkrete Handlungsaufträge für die Durchführung dieser Bachelorarbeit ableiten. Die wichtigsten werden im folgenden vorgestellt.

3.1.2 Automatisiertes Testen

Wie in Unterunterabschnitt 2.1.5 bereits dargestellt, unterstützt Rails den Entwickler beim Erstellen automatischer Tests, die eine wichtige Ausprägung agiler Softwareentwicklung sind. Sie vereinfachen das Refactoring und erlauben neuen Anforderungen schnell zu entsprechen. Dies spiegelt den vierten Grundsatz des *Agile Manifesto* wieder. Werden Tests eingesetzt, um Module zu spezifizieren, so spricht man von TDD, einer konkreten Entwicklungsmethode der agilen Softwareentwicklung. Dabei werden erst die Tests geschrieben und anschließend alle in den Tests gestellten Anforderungen erfüllt.

3.1.3 Prototypen

Prototypen vermitteln einen Eindruck vom späteren Programm. Grundsätzlich zu unterscheiden sind horizontale von vertikalen Prototypen. Während vertikale Prototypen eine

bestimmte Funktionalität oder ein Modul der Anwendung umsetzen, definieren horizontale Prototypen zum Beispiel nur das User Interface (UI), nicht aber die zugrunde liegende Geschäftslogik.

Durch die dem Kunden vermittelten Eindrücke kann dieser die Anwendung früh evaluieren und oftmals ein besseres Problemverständnis entwickeln. Prototypen haben sich insbesondere bei Anwendungen mit hoher Benutzerinteraktion bewährt. [Wiki]

Innerhalb dieser Arbeit wurde primär auf UI- und somit horizontales Prototyping gesetzt. Dabei wurde die Anwendung immer weiter verfeinert, was einem evolutionären Vorgehen entspricht. Prototyping hilft sowohl den zweiten als auch dritten Grundsatz des *Agile Manifesto* zu gewähren.

3.1.4 Akzeptanz-Tests

In nicht agilen Entwicklungsmodellen werden Akzeptanz-Tests erst kurz vor dem Projektabschluss, typischerweise vor der Deployment Phase, durchgeführt. Extreme Programming (XP), ein agiles Entwicklungsmodell, sieht vor, dass nach jeder sukzessiven Erweiterung des Programmumfangs (Iteration) eine entsprechende Validierung mit dem Kunden durchzuführen ist. Dabei werden sogenannte *User Stories* umgesetzt und getestet. Sie beschreiben klar abgegrenzte Funktionalitäten, die vom Kunden gefordert werden. Zur Evaluation der Funktionalität wird eine Person, die sich mit der durch die Anwendung berührten Thematik gut auskennt, benötigt. Dies ist in der Regel der spätere Anwender der Software. Er wird als Subject Matter Expert (SME) bezeichnet. Die Tests werden als Black-Box Tests umgesetzt. Es werden also ausschließlich Benutzeranforderungen, nicht aber die Modullogik im einzelnen getestet. Durch Akzeptanztest kann die Kommunikation mit dem Kunden verbessert und so Punkt vier des *Agile Manifesto* weiter unterstützt werden.

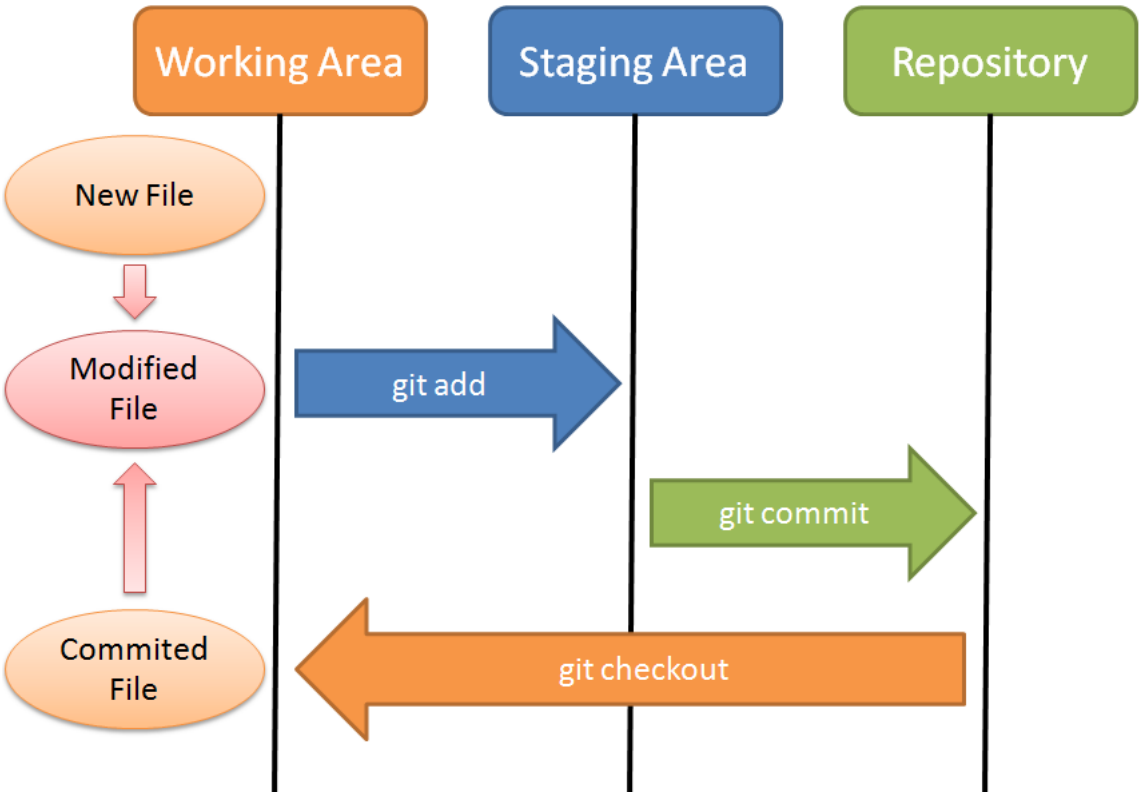
Für diese Arbeit wurde nach Einführung größerer neuer Features jeweils Rücksprache mit Herrn Grove, dem primären Anwender der Software am GSG, gehalten.

3.2 Bitbucket

Bitbucket ist ein, GitHub ähnlicher, Hosting Service für durch Git oder Mercurial verwaltete Projekte. Im Gegensatz zu GitHub stellt Bitbucket private Repositories kostenlos für kleine Teams zur Verfügung.

Das Hosting von Projekten in der Cloud bringt generell einige Vorteile mit sich, so ist das Projekt von überall an zentraler Stelle erreichbar und kann mit minimalem Aufwand

Abbildung 3: Git Arbeitsablauf



repliziert werden. Dies ist insbesondere im Hinblick auf die dezentrale Struktur der möglichen Version Control System (VCS)s wichtig.

Bitbucket ermöglicht weiterhin das Management von Entwicklerteams, was im Rahmen dieser Arbeit aber nicht benötigt wurde. Außerdem kann ein Issue Tracker sowie ein Wiki zur fortlaufenden Dokumentation erstellt werden.

3.2.1 Versionsverwaltung mit Git

Git ist ein Distributed Version Control System (DVCS) und durch seinen vollständig dezentralen Charakter sehr schnell und robust. Das System arbeitet im Gegensatz zu vielen anderen VCSs mit Snapshots anstelle von Deltas. Dies ermöglicht ein schnelles und leichtgewichtiges Branching. Snapshots werden in Git als Commit bezeichnet. Jeder Commit wird mit dem 40-stelligen SHA-1 Hash des Hauptverzeichnis versehen. Wird eine neue Datei eingefügt so durchläuft sie drei Phasen wie in Abbildung 3 illustriert. Git ermöglicht außerdem ein Tagging bestimmter Commits, beispielsweise um Versionsnummern zu vergeben. [Cha09]

3.2.2 Issue Tracking

Das auf Bitbucket frei verfügbare Issue Tracking kann, speziell in einem agilen Umfeld, die Arbeitsorganisation sehr erleichtern. Hier können neben Fehlern auch Anforderungen und zentrale Elemente der User Stories dokumentiert werden. Dies ist vergleichbar mit einem Scrum Backlog. Eine kontinuierliche Übersicht über bisher geleistete und noch zu erledigende Aufgaben helfen den Projektstatus zu definieren und können sich positiv auf die Motivation der Entwickler auswirken. Weiterhin ermöglicht der Issue Tracker den Anwendern selbst Fehler auf einfache Weise zu melden.

4 Umsetzung

Nachdem ein Überblick über die zugrunde liegenden Prinzipien und Werkzeuge gegeben wurde, beginnt hier die Dokumentation der eigentlichen Umsetzung. Dabei werden die verschiedenen Projektphasen sukzessive erläutert und jeweils ausgesuchte Details der Arbeit vorgestellt.

4.1 Anforderungsanalyse

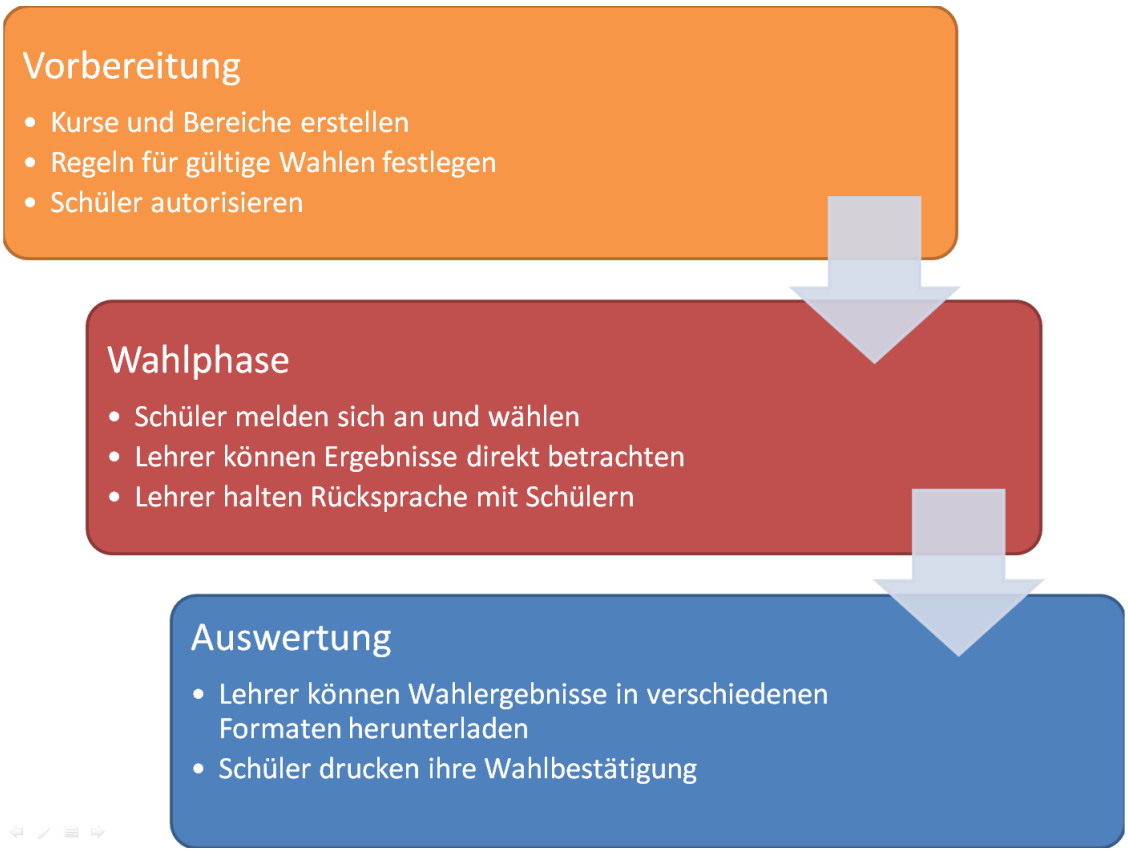
Zu Beginn des Entwicklungsprozesses stand eine knappe Anforderungsanalyse. Diese wurde entsprechend dem agilen Anspruch nicht zu umfangreich betrieben. Dabei wurden nur einige Schlüsselaspekte der zu erstellenden Anwendung festgehalten. Diese Analyse fand noch vor dem eigentlichen Kickoff Meeting statt und basierte vorerst nur auf der durch Herrn Vornberger ausgeschriebenen Aufgabenstellung. Nach dem Kickoff Meeting wurde die Analyse entsprechend verfeinert. Es sollte also eine Anwendung erstellt werden, die es den Schülern erlaubt, ihre Kurswahl am Computer durchzuführen. Von zentraler Bedeutung ist dabei der Anspruch, den Schülern in leicht verständlicher Weise darzulegen, welche Kurse sie wählen dürfen. Außerdem soll eine vereinfachte Verarbeitung der Wahlergebnisse ermöglicht werden. Insbesondere müssen sich die Lehrer die Schüler nach Klassen und Kursbelegungen sortiert und gefiltert anzeigen lassen können. Das Kickoff Meeting hat außerdem ergeben, dass die Schüler in der Lage seien müssen, eine Bestätigung ihrer Wahl herunterzuladen, um somit ein unterschriebenes Formular an die Schule aushändigen zu können. Weiterhin sollen die Schüler ihre Wahl nicht nur einmal abgeben können, sondern innerhalb der Wahlphase ändern können, wenn sie es wünschen. Dies ist wichtig, um eine kontinuierliche Kommunikation zwischen Lehrern und Schülern über die Kursvorstellungen zu ermöglichen. Wählt beispielsweise ein Schüler einen sprachlichen Schwerpunkt, so können ihn Lehrer, die der Meinung sind, ein sprachlicher Schwerpunkt sie nicht optimal, ansprechen. Trotzdem hat der Schüler die Sicherheit, dass seine Wahl gültig ist, falls er sich nicht anders entscheiden will.

4.2 Kickoff Meeting

Nach dem Kickoff Meeting mit Herrn Groove und weiteren Lehrern konnte bereits recht genau ein Ablauf der Wahl festgelegt werden. Dabei wurde das bisherige Vorgehen der Schule analysiert und auf mögliche Verbesserungen hin überprüft. Insbesondere wurde geprüft, welche Abläufe sich automatisieren lassen. Die Durchführung einer Kurswahl konnte in Teilprobleme zerlegt werden, die in Abbildung 4 illustriert werden.

Im weiteren Projektverlauf wurden hier natürlich noch Verfeinerungen und Ergänzungen vorgenommen. So hat sich zum Beispiel erst nach einiger Rücksprache ergeben, dass

Abbildung 4: Teilprozesse der Kurswahl

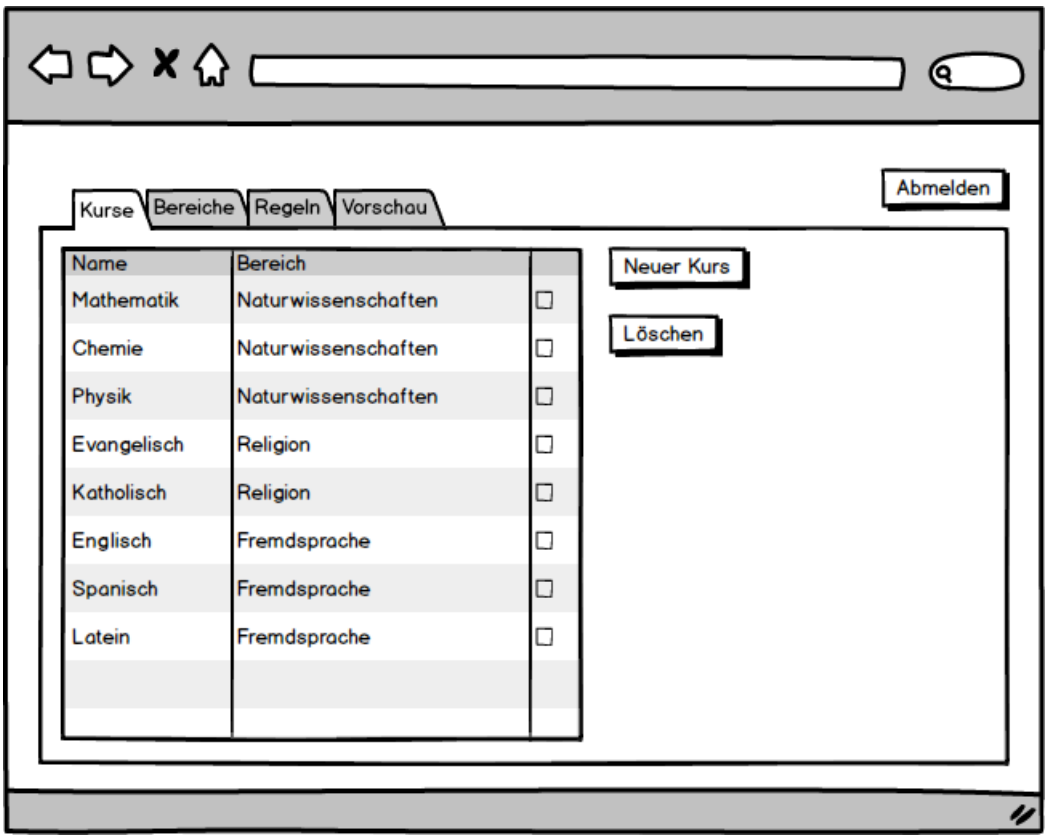


der Lehrer in der Lage seien sollte, die Wahl für einen Schüler durchzuführen oder das Kursangebot vorangegangener Wahlen zu importieren.

4.3 Randbedingungen

Grundsätzlich muss zwischen zwei Arten von Bedingungen unterschieden werden, die für eine gültige Wahl eines einzelnen Schülers einzuhalten sind. Auf der einen Seite stehen Bedingungen, die bestimmte Kurskonstellationen für ungültig erklären. So kann es beispielsweise verboten sein, mehr als 3 naturwissenschaftliche Fächer zu wählen. Auf der anderen Seite stehen Bedingungen, die sich nur auf bestimmte Schüler auswirken. So kann es sein, dass Schüler, die bereits in vorherigen Klassenstufen die Pflicht zur zweiten Fremdsprache erfüllt haben, in der aktuellen Wahl andere Möglichkeiten haben, als solche die noch eine zweite Fremdsprache wählen müssen. Dies wird durch eine eigene Domain-Specific Language (DSL) zur Formulierung logischer Bedingungen erreicht. Diese Bedingungen können sowohl an Kurse als auch an die weiter oben beschriebenen Regeln gebunden werden. Trifft die Bedingung eines Kurses für einen bestimmten Schüler nicht zu, so darf er den entsprechenden Kurs nicht wählen, zum Beispiel *Latein, weiterführend*. Wird die Bedingung hingegen auf eine Regel angewandt, so hat die durch die Regel beschriebene

Abbildung 5: Erster Entwurf



Einschränkung nur Gültigkeit, falls die Bedingung eintritt, zum Beispiel *Mindestens eine weitere Fremdsprache, falls kein Latein ab 5.* Mit den genannten Bedingungen ist es nun möglich, alle geforderten Regeln abzudecken.

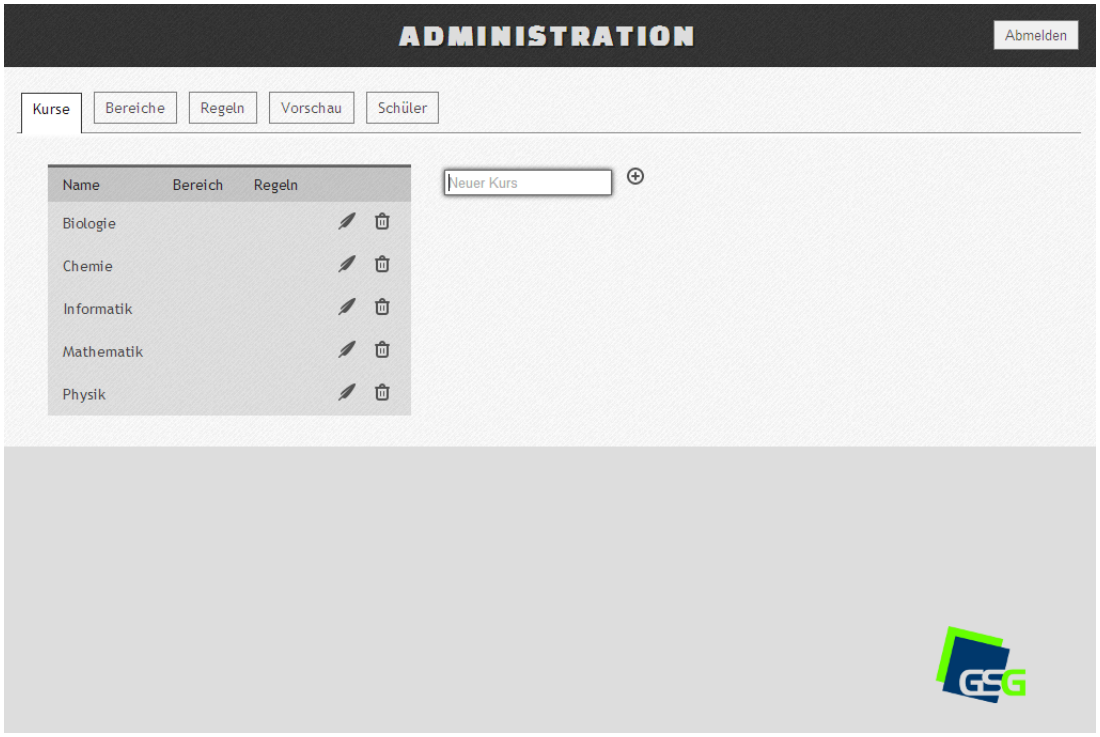
4.4 GUI Prototyping

Dem eigentlichen Prototyping gingen erste Entwürfe vorraus, um lediglich das visuelle Design einzugrenzen. Dabei wurden einfache Bilder des angestrebten UI erzeugt, wie beispielsweise in Abbildung 5 zu erkennen ist.

Der erste erstellte Prototyp verdeutlicht die Benutzeroberflächce zur Manipulation eines einzelnen Wahlprozess, dazu wurden bereits einige Models festgelegt. So wird dem Benutzer eine Ansicht geboten, die die Verwaltung von Kursen, Bereichen und Regeln ermöglicht, siehe dazu Abbildung 6.

Natürlich spiegelt die Entwicklung der Protoypen auch die Entwicklung der gesamten Anwendung wieder. Wie bereits erwähnt handelt es sich um evolutionäres Prototyping. Die optisch auffälligste Änderung des UI zeigte sich, nachdem das Framework Bootstrap in die Anwendung integriert wurde. Obwohl keine neue Funktionalität im eigentlichen

Abbildung 6: Erster Prototyp



Sinne hinzugefügt wurde, kommt der Anwender in einer ihm eher vertrauten Umgebung besser zurecht. Twitter Bootstrap wird mittlerweile von vielen bekannten Web-Applikation verwendet und ist mit seinen standardisierten Elementen vielen Benutzern geläufig. [boo]

Im weiteren Verlauf wurde die Möglichkeit, mehrere Wahlprozesse gleichzeitig durch das Programm zu verwalten, geschaffen. Außerdem wurden die Regeln um Schüler spezifische Eigenschaften erweitert, weil sich erst im Verlauf der Entwicklung geklärt hat, dass verschiedene Schüler grundsätzlich verschiedene Wahlmöglichkeiten haben können. Dabei hat sich wiederum die Stärke des agilen Entwicklungsmodell, und der verwendeten Frameworks Rails und Backbone bewiesen. Abbildung 7 zeigt das UI, nachdem neue Funktionalität hinzugefügt wurde.

4.5 Datenhaltung

Aus den oben beschriebenen Prototypen lassen sich bereits einige Entitäten der zugrundeliegenden Datenbank erkennen. Die Daten werden sowohl Server- als auch Client-seitig vorgehalten, weshalb hier nur eine abstrahierte Perspektive auf die in beiden Fällen zu speichernden Daten geboten wird. Im Rahmen der Betrachtung der einzelnen Programmarchitekturen werden die Models noch detaillierter erklärt. Abbildung 8 zeigt die abstrakte Datenbankstruktur. Wie aus dem Entity-Relationship Diagram (ERD) ersichtlich wird, wurden an dieser Stelle noch keine Annahmen über die Umsetzung Schüler spezifischer

Abbildung 7: Prototyp mit Prozessverwaltung und Bootstrap

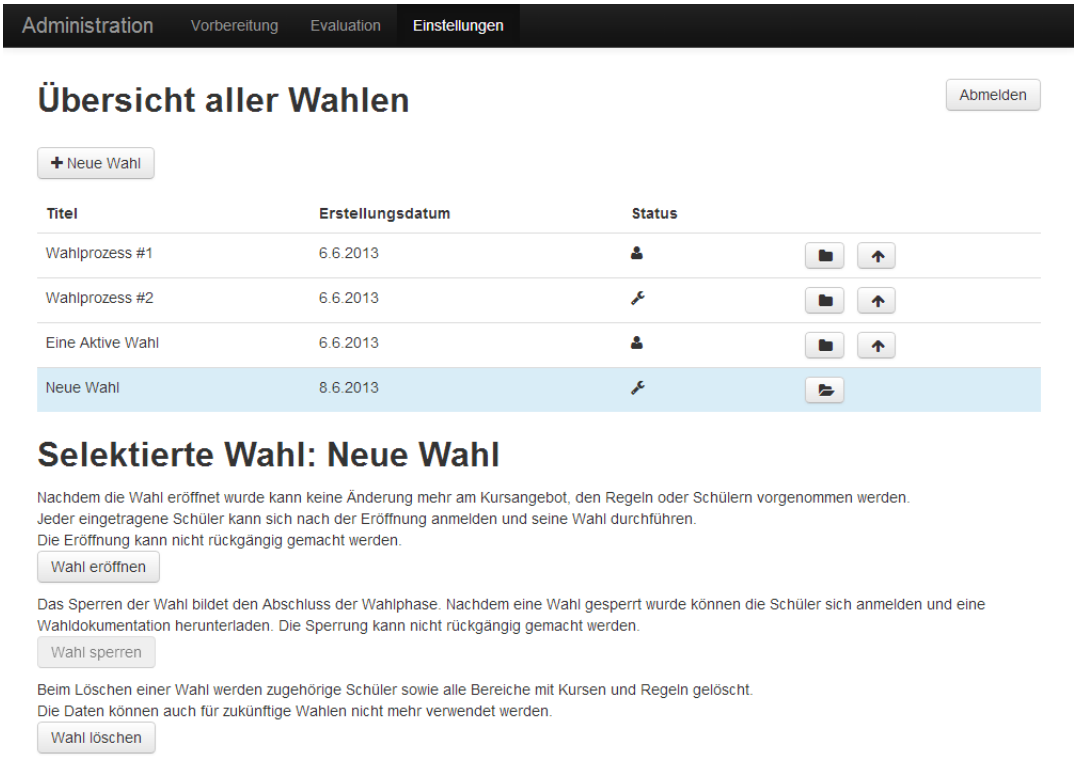
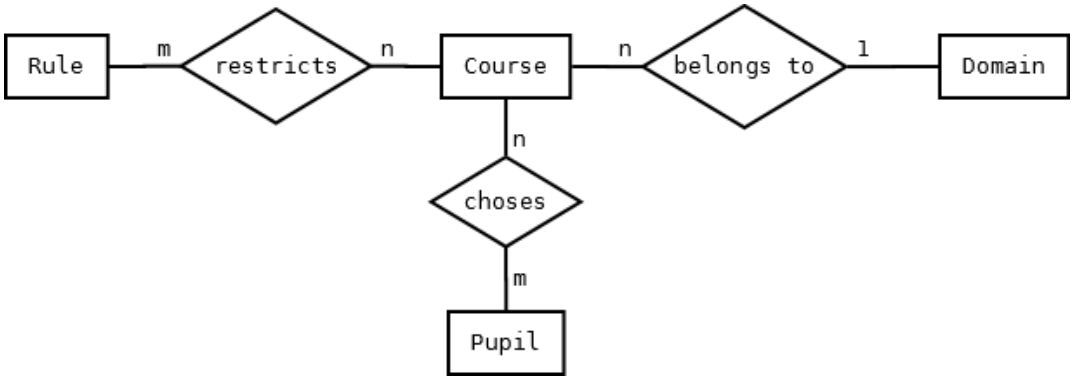


Abbildung 8: Abstraktes ERD



Eigenschaften gemacht. Auch fehlen Informationen über den Administrator.

4.6 Client-Server-Modell

Eines der grundlegenden Netzwerkmodelle der Informatik ist das Client-Server-Modell. Dabei stellt eine zentrale Instanz, nämlich der Server, Ressourcen zur Verfügung, die von verschiedenen Programmen, sogenannten Clients über eine Schnittstelle angefordert werden können. In der Regel findet dabei die Kommunikation über ein Netzwerk statt. Es gibt aber auch Fälle, in denen sowohl Server als auch Client auf der gleichen Maschine laufen. Dies ist zum Beispiel bei mit Datenbankservern kommunizierenden Webservern oftmals der Fall. Es wird in einem solchen Szenario deutlich, dass ein Programm sowohl Server als auch Client eines weiteren Servers sein kann.

Da es sich bei der entwickelten Webschnitstelle um eine Anwendung handelt, die von einer Website aus an einen Browser ausgeliefert wird, wurde hier zwingend das Client-Server-Modell verwendet. Anwendungen dieser Art bezeichnet man auch als Client-Server-System.

Der konkrete Aufbau einer Verbindung geht immer vom Client aus. Der Client kann nur eine Verbindung zum Server anfordern, nicht aber in direkten Kontakt zu anderen Clients treten.

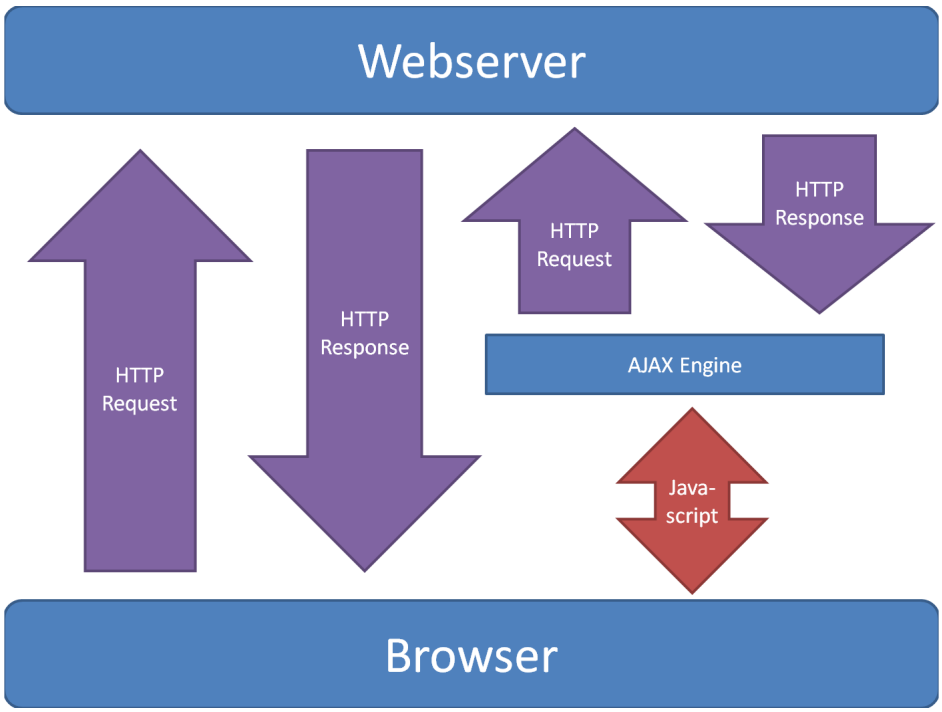
4.6.1 HTTP und AJAX

Im Bezug auf HTTP spricht man von Webserver und Webclient. Der Client ist dabei in der Regel ein Browser, kann aber auch ein anderes Programm, wie beispielsweise ein Webcrawler sein.

HTTP arbeitet mit Request-Response-Paaren. Dabei kann der Client eine Anfrage gegen den Server absetzen und erhält dann eine Antwort, die alle angeforderten Informationen enthält. Insbesondere ist es nicht nicht möglich, dass der Server sich ungefragt beim Client meldet (beispielsweise um eine Änderung der Daten zu propagieren). Diese sehr einfache Strukturierung von HTTP Verbindungen hat im Hinblick auf SPA einen Nachteil: Es können nicht alle Daten beim initialen Laden der Anwendung geladen werden, dies wäre ein zu großer Overhead. Weiterhin müsste bei jeder Aktion, zum Beispiel zum Löschen einer Entität, die gesamte Seite neu geladen werden.

Dieses Problem kann durch AJAX gelöst werden. AJAX erlaubt das dynamische Senden von Anfragen im Hintergrund, dabei muss die Website nicht neu geladen werden. Jegliche Interaktion von Client und Server, ausgenommen das initiale Laden der Website, wird somit durch AJAX abgedeckt. Das Prinzip ist in Abbildung 9 illustriert. Dazu wird ein XML HTTP Request (XHR) mit Javascript instanziiert und an den Server gesendet. Dies kann

Abbildung 9: Architektur einer Webapplikation mit AJAX



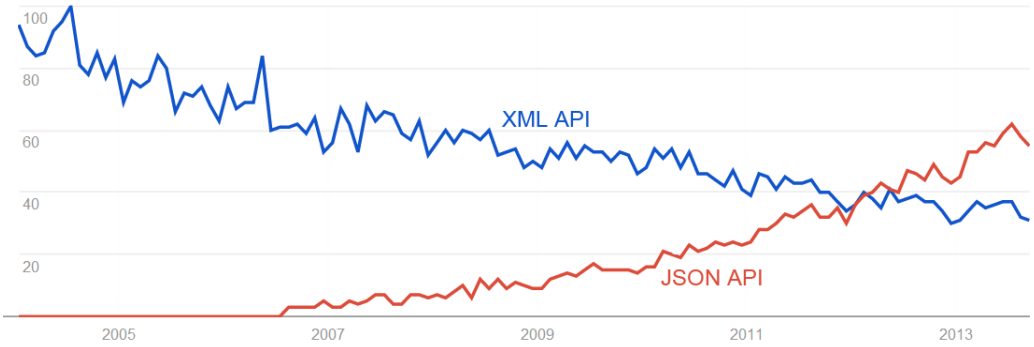
wie in Absatz 2.3.1 beschrieben durch jQuery vereinfacht werden. Obowhl der Name den Gebrauch von XML als Sprache zur Beschreibung der Datensätze suggeriert, ist diese nicht vorgegeben. Prinzipiell kann hier jedes Textformat verwendet werden. Im Rahmen dieser Anwendung kommt JSON zum Einsatz. Es zeichnet sich im Gegensatz zu XML durch einen geringeren Overhead und eine sehr einfache Syntax aus. jQuery unterstützt nativ das Parsen von JSON. Der Einsatz von JSON hat in den letzten Jahren den Gebrauch von XML als Sprache zur Definition von APIs überflügelt, wie in einer Statistik von Google Trends (Abbildung 10) zu erkennen ist. Eine weitere gebräuchliche Alternative zu JSON ist YAML Ain't Markup Language (ehemals: Yet Another Markup Language) (YAML). Diese Sprache wird unter anderem von ROR verwendet, um Fixtures zu definieren.

Allerdings ist auch mit AJAX eine direkte Reaktion auf Veränderungen auf Seite des Servers nicht ohne weiteres möglich, sondern wird indirekt durch AJAX Polling erreicht. Mit HTML5 haben WebSockets ihren Eingang in den HTML Standard gefunden. Diese Technologie ermöglicht neueren Browsern eine direkte Reaktion auf serverseitige Ereignisse. [For]

4.7 JSON-API

Aufbauend auf den oben genannten Einschränkungen von HTTP kann nun die konkrete JSON-API betrachtet werden. Listing 16 zeigt den Response-Body nachdem die Route

Abbildung 10: JSON vs XML



`/api/rules` mit `GET` angefragt wurde.

```
[
  {
    "id" : 2,
    "name" : "Regel #2",
    "min" : 2,
    "max" : null,
    "color" : "#760083",
    "condition" : "",
    "vote_id" : 1
  },
  {
    "id" : 3,
    "name" : "Regel #3",
    "min" : 0,
    "max" : 5,
    "color" : "#0e1400",
    "condition" : "",
    "vote_id" : 1
  }
]
```

Listing 16: JSON API: `GET` für Ressource Regel

Es werden hier alle benötigten CRUD Operationen auf die HTTP-Verben `GET`, `POST`, `PUT` und `DELETE` für jede Ressource abgebildet. Entsprechend der ROA handelt es sich um eine REST konforme Schnittstelle. Damit wird gleichzeitig der Grundstein für die Clientseitigen Models gelegt, so werden jedem Schüler nur genau die Kurse angezeigt, die er auch wählen darf. Es fällt auf Seite des Browser beispielsweise eine Entscheidung über die Gültigkeit der Wahl gar nicht an. Diese Entscheidung muss, um Daten-Manipulation durch Veränderungen am Client-Code auszuschließen, sowieso Server-seitig erfolgen.

Die API unterstützt aktuell folgende Ressourcen: `Courses`, `Domains`, `Rules`, `Pupils` und `Votes`. Dabei ist der Zugriff auf diese Ressourcen für Schüler teilweise eingeschränkt, sie haben weder Einblick in das gesamte Kursangebot, noch können sie einsehen, welche anderen Schüler an der Wahl teilnehmen oder welche Kurse ebendiese wählen.

Aus Pragmatismus wurden einige REST Konzepte bewusst nicht strikt umgesetzt. So wurde auf Hypermedia as the Engine of Application State (HATEOAS) verzichtet und die Ressourcen in Abhängigkeit vom aktuell angemeldeten User ausgegeben (unterschiedliches Kursangebot für verschiedene Schüler). Dies ist möglich, weil die API nur durch die Backbone Anwendung verwendet werden soll und keinen Zugriff durch externe Programme unterstützen muss. Sollte irgendwann der Wunsch bestehen, die Schnittstelle in andere Programme einzubinden, so wäre eine Versionierung der API neben HATEOAS eine sinnvolle Maßnahme.

4.8 Server Architektur

Dieser Abschnitt wird sich mit der grundlegenden Architektur der Serveranwendung auseinandersetzen. Diese entspricht den in Unterabschnitt 2.1 beschriebenen Konventionen des Rails Frameworks. Natürlich kann hier nicht im Detail auf jedes Konstrukt eingegangen werden. Wichtige Konzepte werden aber anhand bestimmter Klassen oder Module exemplarisch aufgezeigt.

4.8.1 Model und Datenbank

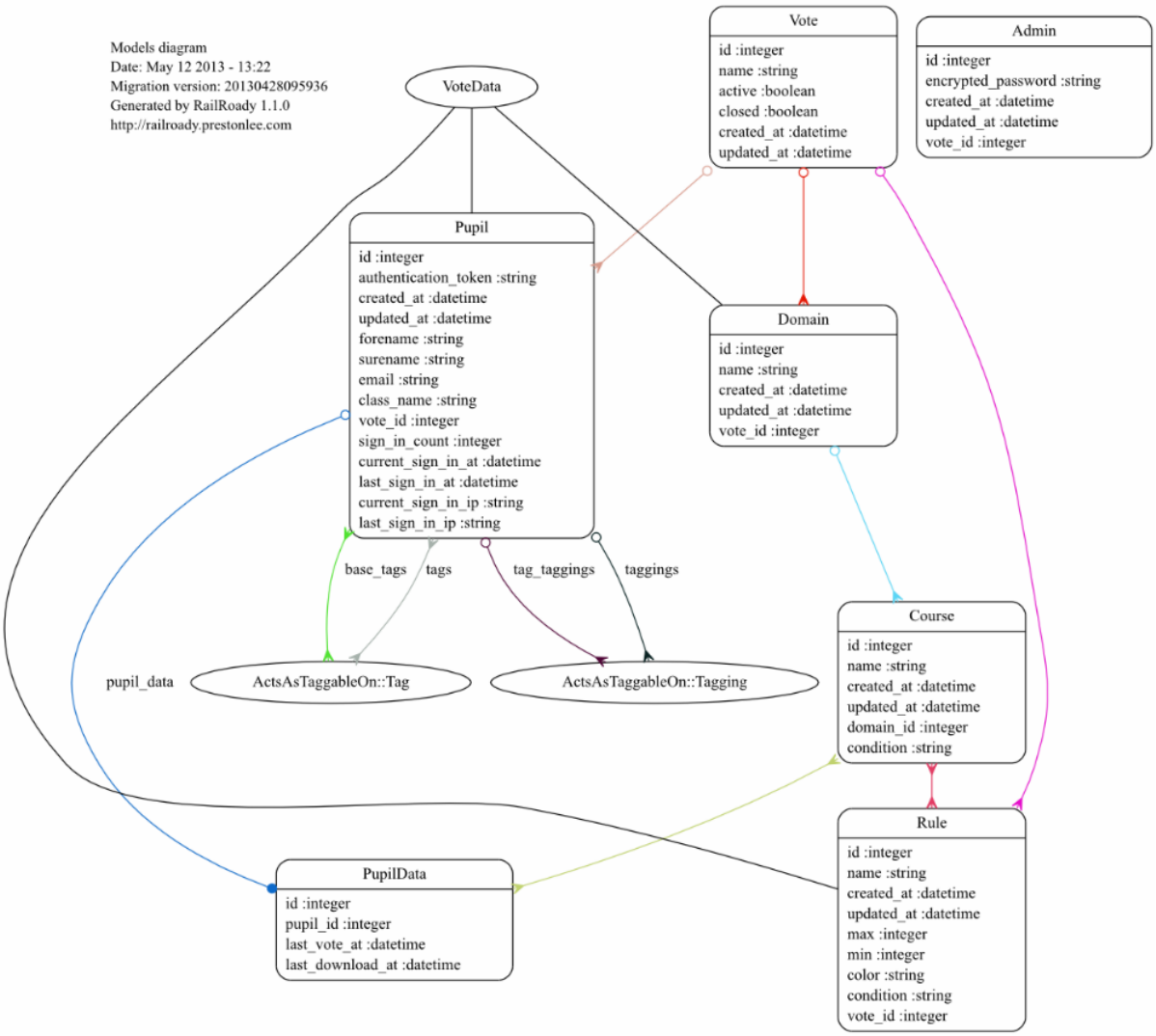
Um ein genaueres Verständnis der Server Applikation zu erlangen, ist es hilfreich, das Datenbankschema näher zu betrachten, siehe dazu Abbildung 11.

Das Diagram wurde mit dem Ruby Gem `RailRoady` erstellt und visualisiert das gesamte Datenbankschema. Die Anwendung verwendet eine SQLite3 Datenbank. Im Gegensatz zu Abbildung 8 werden hier viele Details deutlich.

Während aktuell nur ein Admin unterstützt wird und die Tabelle dementsprechend theoretisch wegfallen könnte, erlaubt das Speichern der Administratoren in der Datenbank einige Vorteile sowohl in Bezug auf die verwendeten Authentifizierungsmechanismen als auch bezüglich möglicher zukünftiger Anforderungen.

Alle weiteren Entitäten sind jeweils einem bestimmten Wahlprozess, in der Datenbank als `Vote` bezeichnet, untergeordnet. Dies kann entweder direkt, wie bei `Domain` oder indirekt wie es bei `Course` über `Domain` der Fall ist, geschehen. Jene Records, die direkt an einen Wahlprozess gebunden sind, erben von der Klasse `VoteData`, die ihrerseits von `ActiveRecord` abgeleitet ist. `VoteData` stellt dabei sowohl sicher, dass die Entität an

Abbildung 11: Datenbank Model



einen existenten Wahlprozess gebunden wird also auch, dass nachträgliche Änderungen an den Daten verhindert werden. Eine Änderung ist nachträglich, genau dann wenn sie nach Eröffnung der Wahl stattfindet. Es darf zum Beispiel nicht möglich sein, Kurse umzubenennen, für die sich Schüler bereits eintragen konnten.

Die Entwicklung der Datenhaltung stellt sich in Rails als Sequenz von Migrationen dar. Wie bereits in Unterunterabschnitt 2.1.2 dargelegt, handelt es sich dabei um Dateien, die jeweils bestimmte Änderungen an der Datenbank beschreiben.

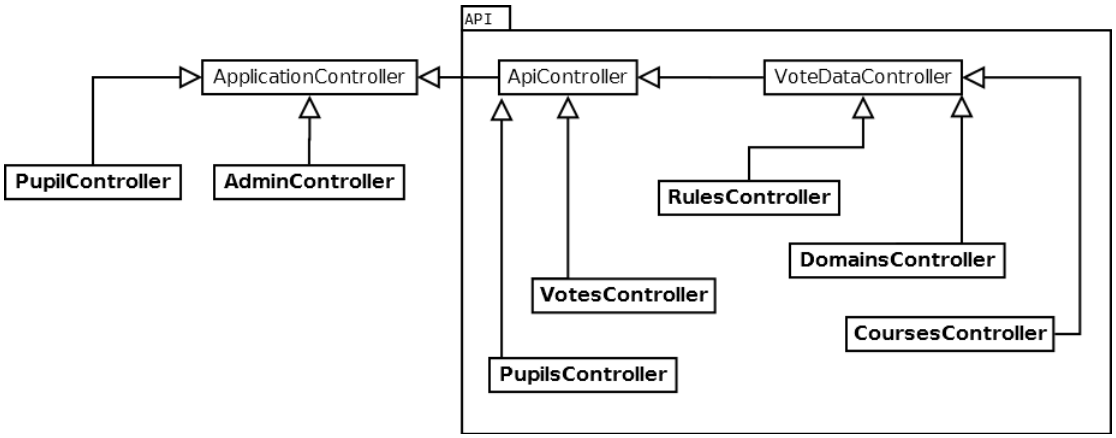
ActsAsTaggableOn Wie bereits erwähnt ist es notwendig, Schüler spezifische Eigenschaften zu speichern. Dies wird durch das Ruby Gem `acts-as-taggable-on` realisiert, das es erlaubt, Entitäten mit Tag-Listen zu versehen. Jeder Tag repräsentiert dabei eine den Schüler betreffende Eigenschaft. So könnte beispielsweise ein Schüler den Tag `latein_ab_7` erhalten. Die Tags können dabei fast beliebig gewählt werden und dürfen nur nicht im Konflikt zur Sprache für die Bedingungen stehen. Wird eine Eigenschaft in die Liste eines Schülers aufgenommen, so erfüllt der Schüler diese Eigenschaft, andernfalls nicht. Es handelt sich also einfach um Wahrheitswerte, die falls nicht gesetzt als falsch angenommen werden. `acts-as-taggable-on` wurde 2009 durch Michael Bleigh veröffentlicht.

Mini-DSL für Bedingungen Analog zu den Tags können Bedingungen sowohl an Kurse als auch an Regeln gebunden werden. Ist eine Bedingung für einen Kurs erfüllt, so wird dieser für den entsprechenden Schüler als Wahloption angezeigt. Leere Bedingungen werden immer als erfüllt angesehen. Ist eine Bedingung für eine Regel erfüllt, so muss diese Regel eingehalten werden, damit die Wahl gültig ist. Sonst wird die Regel für den aktuellen Schüler ignoriert.

Bedingung sind dabei einfache Zeichenketten, sie bestehen aus Tags und den Operatoren `UND`, `ODER` sowie `NICHT`. Es handelt sich bei `ODER` um den inklusiven oder-Operator. So wäre beispielsweise der folgende String eine gültige Bedingung: `"NICHT (latein_ab_7 ODER franzoesisch_ab_7)"`. Bedingungen werden durch das Hilfsmodul `ConditionHelper` validiert und evaluiert. Dazu implementiert das Modul den Shunting-yard Algorithmus nach Dijkstra.

Devise `devise` wurde 2009 von der Firma Plataformatec veröffentlicht und ist eine auf `Warden` basierende Lösung zur Authentifiziertung von Nutzern innerhalb Rails. Im Rahmen dieser Arbeit wurde sowohl die Authentifierung mittels Password für den Administrator als auch die Authentifizierung durch Tokens für die Schüler eingesetzt. Dies spiegelt sich

Abbildung 12: Übersicht der Controller



auch in Abbildung 11 wieder. Die Tabelle `Admins` enthält die Spalte `encrypted_password`, die Tabelle `Pupils` die Spalte `authentication_token`.

4.8.2 Controller

Es werden alle Anfragen an den Server durch einen Controller entgegen genommen. Ausgenommen davon sind statische Dateien im `public` Verzeichnis. Hier muss grundlegend zwischen zwei Arten von Controllern unterschieden werden. Zum einen gibt es die Controller, die den Nutzer authentifizieren und den initialen HTTP-Request beantworten, namentlich `AdminController` und `PupilController`. Auf der anderen Seiten stehen jene Controller, die nur über AJAX angesprochen werden sollten und keine HTML- sondern JSON-Antworten liefern. Diese Controller erben alle von `Api::ApiController`. Alle in der Anwendung verwendeten Controller erben wiederum, den Rails Konventionen entsprechend, von der Klasse `ApplicationController`. Diese definiert einige global verwendete Methoden sowie die Routen für die Weiterleitung durch `devise`.

Listing 17 zeigt das Abbilden aller Controller Actions auf die entsprechenden Routen.

Authentifizierung Innerhalb jeder Controller-Action kann eine Beschränkung der Nutzung durch eine `devise`-Direktive erfolgen. Dabei kann durch Rails `before_filter` eine Authorisierung für eine Teilmenge der Actions direkt festgelegt werden. Erfolgt ein nicht authorisieter Zugriff, so leitet `devise` den Nutzer auf eine Login-Seite um. Diese Seite ist nicht Teil der SPA, ein erfolgreicher Login leitet dann auf die entsprechende Anwendung um. Nach erfolgter Umleitung wird entweder das ERB-Template für Schüler oder Administratoren gerendert. die Templates unterscheiden sich im wesentlichen durch einen abweichenden `javascript_include_tag`. Streng genommen existieren also zwei SPAs:

Eine Anwendung für Schüler und eine Weitere für Administratoren. Diese haben aber Überschneidungen hinsichtlich des Quellcodes. Mehr dazu in Unterabschnitt 4.9.

API-Controller Jeder API-Controller übernimmt die Verwaltung einer bestimmten Ressource, so ist zum Beispiel die Klasse `Api::CoursesController` verantwortlich für das Löschen, Erstellen, Ausgeben und Editieren (CRUD) aller Kurse. Die Ausgabe erfolgt dabei im Gegensatz zu den klassischen Controllern nicht durch das Rendern einer View sondern durch direkte Ausgabe von JSON im Response-Body. Dabei werden die `as_json` Methoden der angeforderten Models ausgewertet. Die Definitionen dieser Methoden legen fest, welche Attribute über die API ausgegeben werden können.

```
Votan::Application.routes.draw do
  devise_for :admins, :path => 'admin'
  devise_for :pupils, :path => 'pupil'

  root :to => redirect("/pupil/sign_in")
  match 'admin' => "admin#index"
  match 'pupil' => "pupil#index"
  match 'pupil/confirmation.pdf' => "pupil#confirmation"
  match 'admin/evaluation.pdf' => 'admin#evaluationPrinted'
  match 'admin/evaluation.xlsx' => 'admin#evaluationExcel'
  match 'admin/import/pupils' => 'admin#importPupils'

  namespace :api do
    get 'votes/selected' => 'votes#getSelected'
    post 'votes/import/:id' => 'votes#importFrom'
    post 'votes/:id/select' => 'votes#select'
    post 'do_vote' => 'pupils#doVote'
    resources :rules, :pupils, :courses, :domains, :votes
    get 'pupil' => 'pupils#showCurrent'
  end
end
```

Listing 17: Routen Konfiguration

4.8.3 Views und Mailers

Es existieren Views für alle herunterladbaren, nicht statischen Dateien. Dazu gehören sowohl die Partials für die Administrator- und Client-SPA mit dem zugehörigen Layout als auch PDF und XLS/CSV Dateien. Weiterhin werden die E-Mail Templates im `View`-Verzeichnis als ERB-Dateien abgelegt.

Views können nur die von den Controllern an sie übergebenen Daten nutzen, um das Template zu rendern. Die Übergabe erfolgt dabei als Attribut des aufrufenden Controllers.

Es müssen Benachrichtigungen sowohl über die Zulassung zu einer Kurswahl als auch Erinnerungen an den Download der Wahlbestätigung automatisch versandt werden.

Prawn und Axlsx-Rails `prawn` ist ein Ruby Gem der das Generieren von Portable Document Format (PDF) Dateien mit Ruby vereinfacht. `prawn` rendert im Rahmen dieser Anwendung die Wahlergebnisse in Form einer Übersicht für das Lehrpersonal und außerdem die Wahlbestätigung, die jeder Schüler herunterzuladen und zu unterschreiben hat.

Bei `axlsx-rails` handelt es sich ebenso um einen Ruby Gem, dieser dient der Erzeugung von Excel-Dateien, die als Alternative zu den PDF-Dateien als Download der Wahlergebnisse zur Verfügung stehen.

Weiterhin existiert eine Helper Klasse namens `SpreadsheetParser`, die das Einlesen von verschiedenen Dateien mit tabellarischen Daten realisiert. Diese Daten können genutzt werden, um die Schülerlisten zu initialisieren. Dies ist sinnvoll, weil oftmals bereits Datenbanken existieren, in denen ein Großteil der wahlberechtigten Schüler gespeichert ist.

4.9 Client Architektur

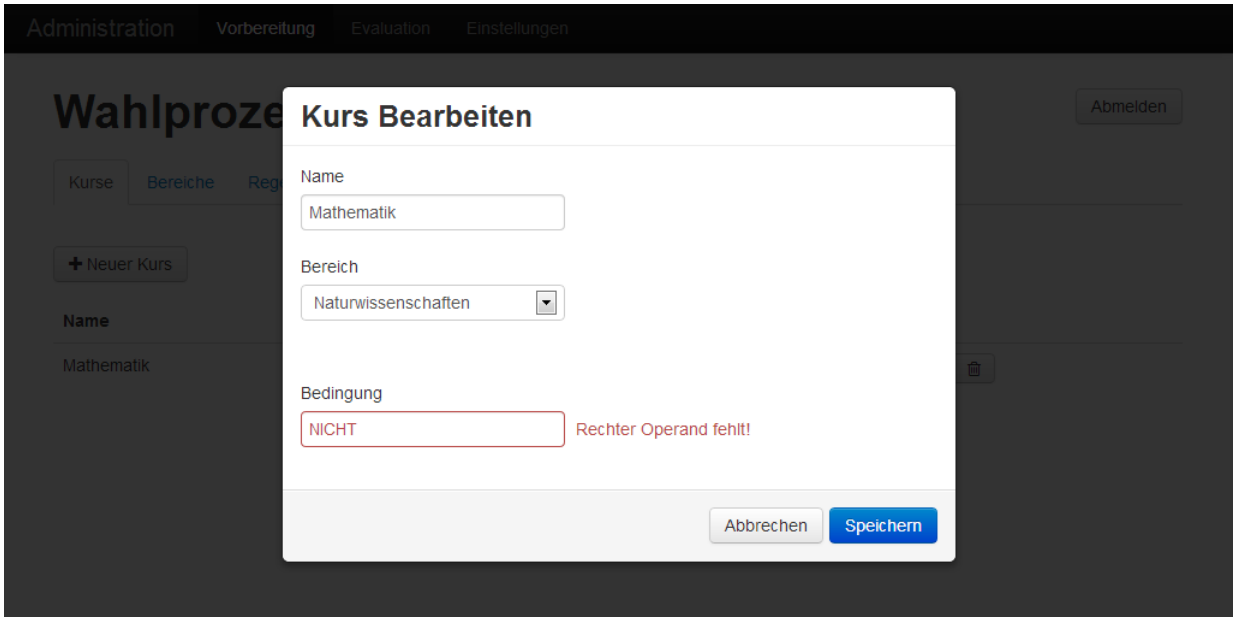
Eigentlich muss hier von Client-Anwendungen gesprochen werden, da sowohl eine Javascript Datei für die Schüler als auch für den Administrator generiert wird. Die Anwendungen haben aber Überschneidungen insbesondere hinsichtlich Models und Collections sowie einiger Views und Templates. Durch die `require` Anweisungen innerhalb der Javascript und CoffeeScript Dateien werden die Abhängigkeiten der zwei Anwendungen aufgelöst.

4.9.1 Dialog System

Um den Einsatz von Popups zu vermeiden, muss eine Alternative Lösung zur Kommunikation mit dem Nutzer gefunden werden. Es müssen Fehlermeldungen, Bestätigungsdialoge und ähnliches angezeigt werden können. Eine sinnvolle Lösung bietet hier das in Bootstrap integrierte `modal`-Plugin. Dieses Plugin erlaubt das Erstellen einfacher Dialoge unter Zuhilfenahme eines entsprechend gefüllten DOM-Elements und des Aufrufs einer von Bootstrap gestellten Javascript Funktion. Nun ist es aber nicht möglich, jeden Dialog im Voraus als DOM-Element in die Anwendung zu integrieren. Vielmehr müssen diese Dialoge zur Laufzeit dynamisch generiert werden. Dieses Generieren und Anzeigen dynamischer Dialoge übernimmt die Klasse `Dialog.Base`.

Jeder Dialog enthält einen Titel, beliebigen HTML-konformen Inhalt und eine Menge von Buttons. Soll der Dialog angezeigt werden, so wird das Dialog-EJS-Template mit den

Abbildung 13: Dialog zum Editieren eines Kurs



entsprechenden Werten gerendert. Dabei wird neben dem eigentlichen `modal` Element für Bootstrap auch ein Overlay erzeugt, das die restliche Seite halbtransparent verdeckt.

Bestimmte Typen von Dialogen werden nun öfter benötigt. Dazu zählen insbesondere Dialoge zum Erstellen und Editieren von Models sowie Dialoge zur Bestätigung von Aktionen. Es muss zum Beispiel das Löschen eines Kurses oder das Veröffentlichen der Wahl bestätigt werden. Diese Fälle werden durch die Klassen `Dialog.Confirm` und `Dialog.Edit` abgedeckt.

`Dialog.Confirm` nimmt lediglich einen Titel, eine Nachricht sowie optional Callbacks für Ja/Nein Eingaben des Nutzers entgegen. `Dialog.Edit` wird verwendet, um die Werte eines Models zu manipulieren und Fehlermeldungen des Servers bezüglich der Validierung dieses Models auszugeben. Dazu können dem Dialog die anzuzeigenden Felder mit ihren Typen (Beispielsweise `text-input` oder `select` neben dem zu bearbeitenden Model übergeben werden. Für Details sei hier auf die Dokumentation der Client-Anwendungen verwiesen.

4.9.2 Model

Das Model der Clients stellt sich nun im Verhältnis zu Serveranwendung wesentlich simpler dar. Es existieren Kurse, Bereiche, Regeln, Schüler und Wahlprozesse, die per se nicht miteinander in Bezug stehen, sondern jeweils eine abgeschlossene Menge von Schlüssel-Wert Paaren sind.

Eine Besonderheit stellt die Basisklasse `Collections.TagConditionedCollection` der Klassen `Collections.Rules` und `Collections.Courses` dar. Sowohl wählbare Kurse als

auch einzuhaltende Regeln hängen von den Eigenschaften des entsprechenden Schülers ab und müssen dementsprechend auch auf Basis dieser Eigenschaften bezogen werden. Dazu werden die zutreffenden Eigenschaften des Schülers als Query-Parameter an die URL zum Abrufen der Kurse oder Regeln angehängt. Es ist technisch möglich, hier falsche Angaben zu den Eigenschaften eines Schülers auf Client-Seite zu machen. Deshalb muss die Gültigkeit der Wahl letztlich auf dem Server festgestellt werden muss. Wichtig ist nur, dass für die nicht manipulierte Client-Anwendung das gültige Wahlangebot mit den entsprechenden Einschränkungen angezeigt wird.

Nun stehen bestimmte Models indirekt aber doch in Relation. Beispielsweise sind die Kurse den Bereichen untergeordnet und alle Daten einem Wahlprozess zugeordnet. Wird der aktuell selektierte Wahlprozess geändert, so müssen alle Collections neu geladen werden, näheres dazu im nächsten Abschnitt. Es werden die vom Server vergebenen `ids` verwendet, um den Bereich eines Kurses zu referenzieren und gegebenenfalls aus der Collection der Kurse zu holen.

4.9.3 App Object

Das Anwendungsobjekt stellt den Einstiegspunkt für die Client-Anwendungen dar. Basisklasse beider Anwendungen ist die Klasse `BaseApp`. Diese erfüllt das Singleton-Pattern, initialisiert die AJAX-Engine und instanziiert die in beiden Anwendungen verwendeten Collections.

Es wird zuerst der aktuelle Wahlprozess vom Server geholt. Im Fall eines Schülers kann dies nur ein bestimmter Wahlprozess sein. Der Administrator kann aus allen existierenden Prozessen auswählen und erhält somit den aktuell selektierten Wahlprozess.

Für die Schüleranwendung `PupilApp` stellt sich das weitere Vorgehen sehr einfach dar. Es wird den zwischengespeicherten Daten noch der aktuelle Schüler hinzugefügt, um Daten wie den Namen ausgeben zu können. Anschließend wird der Schüler-Router instanziiert und die Anwendung gestartet. Der Router wiederum bindet alle benötigten Dateien ein, um die unter seien Routen vorhandenen Funktionen anzubieten, also insbesondere die benötigten View Klassen.

Die Klasse `AdminApp` ist für die Administrationsanwendung verantwortlich und stellt sich etwas komplexer dar. Allem voran ist es hier möglich, den Wahlprozess im Betrieb der SPA zu wechseln, also ohne sich neu anzumelden. Dazu wird ein `CurrentVoteWrapper` definiert, der es ermöglicht, mit jQuery Ereignissen auf ein Client-seitiges Wechseln oder Aktualisieren des selektieren Wahlprozesses zu reagieren. Die Methode `fetchData` der Basisklasse wird überschrieben und definiert Ereignisse, die nach dem Laden des aktuellen Wahlprozess vom

Server ausgelöst werden. Alternativ hätte man hier mit `Backbone.Relational` arbeiten können. Weiterhin müssen mehrere Router instanziiert werden, die jeweils einen anderen Bereich der Anwendung abdecken. Dabei werden auch die benötigten Collections und Models an die Router weitergegeben, wodurch ein qualifizierter Zugriff ermöglicht wird.

4.9.4 Views

Die Views bilden nun das Bindeglied zwischen DOM und Models. Dabei wird typischerweise eine View auf eine Route abgebildet. Damit erfüllen die Views in Backbone die analoge Aufgabe der Controller in Rails. Das DRY-Prinzip wird auch bei den Client-Views durch Vererbung und Partials bzw. Layout-Views realisiert. Diese Layouts können beliebige weitere Views enthalten. Im folgenden werden einige Views näher erläutert.

Views.Error Bei `Error` handelt es sich um eine relativ simple View. Hier wird lediglich ein String im Template gerendert und nicht auf Veränderungen am Model reagiert. Die View dient der Anzeige einer Fehlermeldung, typischerweise bei gescheiterten AJAX-Requests, zum Beispiel wegen unterbrochener Internetverbindung.

Views.MultiItemView `MultiItemView` ist eine Klasse, die nicht direkt verwendet werden solltem, sondern als abstrakte Basisklasse vieler Views dienen kann. Diese View bindet Callbacks an die Backbone `sync`-Ereignisse aller übergebener Models. Damit ist es möglich, eine beliege Anzahl unterschiedlichster Models in einem View-Template-Paar darzustellen und auf Veränderungen an diesen Models zu reagieren. Backbone.Marionette stellt eine solche Funktionalität lediglich für Views zur Verfügung, die nur ein einziges Model oder eine einzige Collection verwalten.

Views.Admin.VoteSpecificLayout Dies ist das abstrakte Layout für die Vorbereitung und Auswertung eines Wahlprozess durch den Administrator. Wie der Name suggeriert, wird es nur von der Administrations-Applikation verwendet und findet sich somit nicht im compilierten Javascript der `PupilApp` wieder. Die Klasse erbt von `Marionette.Layout` und definiert eine Region, um Sub-Views einzubinden. Dabei kann über beliebig betitelte Tabs zwischen den Sub-Views gewechselt werden. Außerdem wird wie bei `Marionette.ItemView` die Darstellung eines einzelnen Models unterstützt. Hier sollte ein Wahlprozess als Model zum Einsatz kommen. Somit kann der Administrator einsehen, welchen Wahlprozess er gerade vorbereitet oder auswertet.

Von dieser Klasse erben nun `Admin.Eval.Main` und `Admin.Prep.Main`. Diese Klassen können jetzt sehr kompakt definiert werden, siehe Listing 18.

```
#= require ../vote_specific_layout

class Votan.Views.Admin.Eval.Main extends Votan.Views.Admin.VoteSpecificLayout
  buttons:
    overview:
      title: 'Übersicht'
      desc:  'Anzahl der Schüler pro Kurs und Klasse'
      href:  '#eval'
    details:
      title: 'Details'
      desc:  'Genaue Kursbelegung für jeden Schüler'
      href:  '#eval/details'
```

Listing 18: Klasse für Evaluationsansicht des Administrators

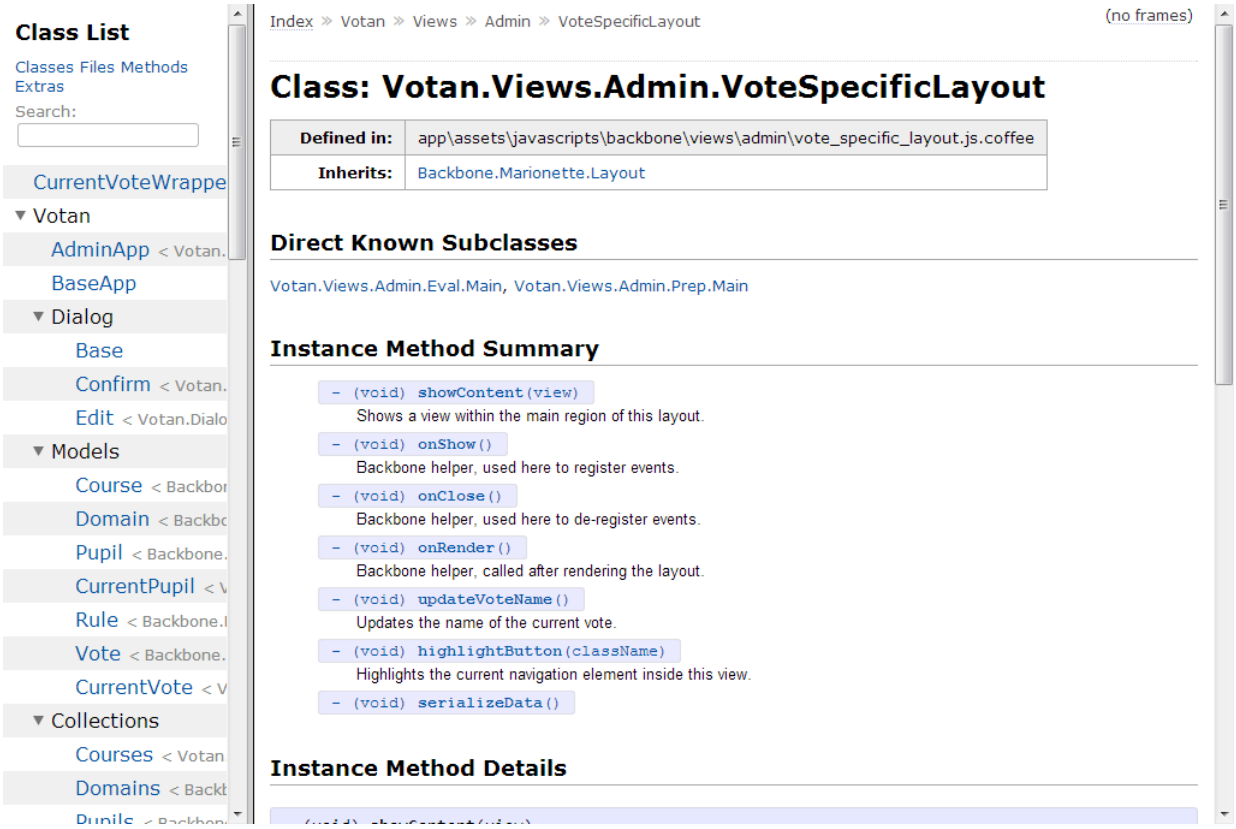
Views.VoteForm `VoteForm` realisiert das Formular zur Durchführung einer Wahl. Diese Ansicht wird sowohl für die Vorschau im Administrationsbereich als auch für die Durchführung der tatsächlichen Wahl eingesetzt. Die Klasse erbt von `Marionette.CompositeView`, einer Kombination aus `Marionette.CollectionView` und `Marionette.ItemView`. Dabei wird über die verschiedenen Bereiche der Wahl als Collection iteriert und jeweils alle Kurse dieses Bereiches ausgegeben. Bei Änderungen am Formular wird in Echtzeit die Gültigkeit der Wahl festgestellt und sofort an den Nutzer gemeldet. Hat der Nutzer Manipulationen an seiner Anwendung vorgenommen, so kann es hier natürlich vorkommen, dass ungültige Wahlen als gültig angezeigt werden - diese würden aber vom Server innerhalb einer separaten Prüfung abgewiesen werden.

4.10 Dokumentation

Einer der Forderungen seitens der Schule war eine umfassende Dokumentation der Arbeit. Dazu gehört auch die Dokumentation des Quellcodes der Anwendung. Obwohl gerade im Umfeld dynamischer Sprachen und agiler Entwicklung gerne auf ausführliche Dokumentation verzichtet wird, wurden sowohl Client- als auch Server-seitige Anwendungen ausgiebig mit inline Dokumentation versehen. Auf Seite des Servers wurde die Dokumentation mittels `yard` vorgenommen. `yard` erstellt eine Sammlung von HTML Dateien, die alle Klassen/Module der Anwendung detailliert dokumentieren. Die Beschreibungen von Methoden/Klassen und weiteren Konstrukten werden dabei ähnlich zu `Javadoc` aus den Kommentaren im Quelltext extrahiert.

Mit `Codo` existiert ein nahezu gleichwertiges Werkzeug für CoffeeScript. Wie auch in `yard` ist es möglich, andere Klassen und Methoden in den Kommentaren zu referenzieren und Statistiken über die Kommentierung zu generieren. `Codo` erstellt ebenfalls eine Website zur Betrachtung der Dokumentation, wie aus Abbildung 14 ersichtlich wird.

Abbildung 14: Dokumentation des CoffeeScript Code



4.11 Deployment

Die Einrichtung einer Instanz dieser Anwendung erfordert einige Einstellungen. Dabei muss zuerst die Produktions-Datenbank erstellt und der Administrator durch ein entsprechendes Passwort geschützt werden. Der Administrator kann entweder über die Rails Konsole oder durch ein Editieren der `seed`-Datei erstellt werden. Ebenfalls sollten die E-Mail Templates und gegebenenfalls die Seiten für den Login und das Wählen angepasst werden. Anschließend können die CSS und CoffeeScript Quelldateien übersetzt, minimiert und verpackt werden. Dies erfolgt automatisch durch die Rails Asset Pipeline. Falls ein gültiges Zertifikat für die Servermaschine vorhanden ist, kann außerdem Secure Socket Layer (SSL) aktiviert werden.

Der gesamte Prozess könnte in Zukunft auf einen bestimmten Anbieter, wie zum Beispiel *Heroku* zugeschnitten und automatisiert werden.

5 Zusammenfassung

Nachdem die Planung und Umsetzung der Arbeit dargestellt wurde, widmen wir uns kurz einer abschließenden Betrachtung. Dabei sollen Stärken und Schwächen der Arbeit aufgezeigt werden, sowie auf die weitere Entwicklung eingegangen werden.

5.1 Evaluation

Es sollte eine wohldokumentierte Webschnittstelle zur Durchführung von Kurswahlen am Graf-Stauffeberg-Gymnasium Osnabrück (GSG) entwickelt werden. Diese Aufgabe kann als vollständig erfüllt angesehen werden. Sowohl Herr Grove, seinerseits zuständig für die Koordination des Zehnten Jahrgangs, als auch Herr Wieneke, der als Informatiklehrer meine Arbeit verfolgt hat, haben sich mit der erreichten Funktionalität sehr zufrieden gezeigt.

Allerdings weist die Arbeit auch einige Aspekte auf, die verbesserungswürdig erscheinen. So ist die Eingabe des für die Kurswahlen nötigen Regelwerks relativ kompliziert und setzt ein gewisses technisches Verständnis voraus. Dies kann teilweise durch einen bereits implementierten Mechanismus kompensiert werden, der es erlaubt, Regeln und Kurse vorheriger Wahlprozesse in einen neuen Prozess zu importieren. Somit kann die Problematik auf die Einrichtung der Schnittstelle beschränkt werden. Kleinere Änderungen am Regelwerk sind später natürlich immer noch leicht möglich. Mit dem beschriebenen Model zur Serialisierung der Regeln ist es nicht möglich, Kurse zu erzwingen falls bestimmte andere Kurse im selben Wahlprozess gewählt wurden. So sind Regeln der Form *(Kurs A und Kurs B) oder nicht (Kurs A oder Kurs B)* nicht formulierbar. Dies kann aber umgangen werden indem *Kurs A* und *Kurs B* in einen einzigen Kurs, *Kurs AB*, überführt werden.

Als besonders positiv an der Umsetzung kann die Nutzung verschiedener, bekannter Frameworks angesehen werden. Diese Frameworks erleichterten die Arbeit ungemein und helfen typische Fehler zu vermeiden und den Quellcode aufgeräumt und übersichtlich zu gestalten. Das UI orientiert sich an Bootstrap und erscheint insgesamt intuitiv bedienbar, was durch mehrere Usability Tests bestätigt werden konnte. Weiterhin sichert Bootstrap die Darstellbarkeit der Anwendung durch eine breite Basis von Browsern.

Werden die Anwendungen mit der Rails Asset Pipeline für die Produktionsumgebung kompiliert, so können sie auf Größen von weniger als 200 Kilobyte beschränkt werden. Erfolgt zusätzlich eine Komprimierung, wie sie von vielen aktuellen Browsern unterstützt wird, so findet eine weitere Reduzierung auf weniger als 50 Kilobyte statt. Damit ist die komprimierte Anwendung selbst kleiner als beispielsweise das aktuelle `jQuery 1.10.2` in

der minimierten Version. Die Anwendung wurde im Hinblick auf die Performanz ebenfalls mit Werkzeugen wie *Fiddler* getestet. Es konnten zufriedenstellende Ergebnisse auch bei Verbindungen mit geringer Bandbreite gemessen werden. Die Anwendungen umfassen etwas mehr als 7000 Lines of Code (LOC). Davon entfallen etwa 4500 auf die Serveranwendung, wobei die Tests ungefähr 2000 Zeilen einnehmen.

Alternativen Gerade im Nachhinein wird deutlich, welche anderen Frameworks ebenfalls hätten eingesetzt werden können. Dazu gehören für die Gestaltung der Serverapplikation *node.js*, was eine Programmierung der gesamten Anwendung mit CoffeeScript ermöglicht hätte. Frameworks wie *Restify* unterstützen hier die saubere Erstellung einer JSON-API inklusive `semver` Versionierung.

Mit *Angular.js* steht ein ausgezeichnete Alternative für die Programmierung der Client-Anwendung bereit. Zur Kompilierung kann hier auf *Grunt* zurückgegriffen werden. Viele Komponenten, für die in Backbone keine Lösungen vorgegeben werden, sind integraler Bestandteil des Angular Frameworks. Für klassische Szenarien, wie das Rendern der Daten eines einzelnen Model, braucht man in Angular oft weitaus weniger LOC.

Letztlich sind aber sowohl Planung als auch Durchführung durch die Vorgaben der Bachelorarbeit zeitlich sehr beschränkt. Somit ist der Einsatz eines gut dokumentierten und hinreichend vertrauten Werkzeugs wie Rails sicherlich eine richtige Wahl gewesen.

Die agile Herangehensweise hat sich als voller Erfolg dargestellt. Prototyping und kurze Iterationen mit anschließenden Akzeptanztests konnten die auf Wünsche des Kunden zielgerichtete Entwicklung sicherstellen.

5.2 Ausblick

Da sich im Rahmen des agilen Vorgehens einige Anforderungen erst im Verlauf der Entwicklung ergeben, kann nicht jede zum Abschluss der Arbeit als sinnvoll erachtete Funktionalität bereits implementiert sein. So ergaben sich auch für diese Arbeit einige nicht vitale aber dennoch wünschenswerte Verbesserungen.

Um die Administration durch mehrere Benutzer zur gleichen Zeit zuzulassen, ist es nötig, dass der Server den Client über eine Änderungen der Daten informieren kann. Dies ist mit der aktuellen Version noch nicht möglich, kann aber mit HTML5 Websockets umgesetzt werden. E-Mail und weitere Templates müssen aktuell beim Einrichten des Programms angepasst werden. Dies könnte in Zukunft dynamischer durch einen Administrator geschehen. Die in Niedersachsens Schulen verwendeten Datenbanksysteme *DaNiS* und *Appollon13* konnten im Rahmen dieser Arbeit noch keine Anbindung an die Anwendung erfahren. Dies

ist primär den Datenschutz-rechtlichen Bestimmungen des GSG geschuldet. In Zukunft könnte sich eine solche direkte Im- und Exportfunktionalität aber als sehr praktischer erweisen. Eine kleinere möglicherweise ebenfalls sinnvolle Ergänzung wäre eine automatische Vervollständigung von Schüler-spezifischen Eigenschaften anhand vorheriger Eingaben im Bearbeitungsdialog der Schüler.

Außerdem wäre ein Refactoring bestimmter Komponenten der Anwendung gegebenenfalls angebracht. So kann durch den Einsatz von Backbone.Relational die Verwaltung der Beziehung zwischen den Models durch eine externe Bibliothek übernommen werden. Weiterhin könnte das Dialog System in die von Backbone zur Verfügung gestellten Komponenten eingebunden und mit einem eigenen Template versehen werden, was eine striktere Trennung von Funktionalität und Darstellung ermöglichen würde. Abgesehen davon kann das Auffinden von bisher nicht entdeckten Fehlern natürlich nur minimiert aber keineswegs ausgeschlossen werden. Ziel ist also, das Projekt längerfristig zu begleiten und bei Bedarf weitere Nachbesserungen vorzunehmen. Insgesamtg aber hat die Anwendung einen Status erreicht, mit dem eine zuverlässige Arbeit inklusive aller eingangs geforderten Funktionalitäten gut möglich ist.

6 Quellenverzeichnis

[agi] *Agile Manifesto*. http://agilemanifesto.org/. – Zugriff: 19.08.2001

[Asha] Ashkenas, Jeremy: *Backbone.js*. http://backbonejs.org/. – Zugriff: 01.08.2013

[Ashb] Ashkenas, Jeremy: *Underscore.js*. http://underscorejs.org/. – Zugriff: 01.08.2013

[BKR08] Bibeault, Bear ; Katz, Yehuda ; Resig, John: *jQuery in Action*. Manning, 2008

[Bla09] Black, David A.: *The Well-Grounded Rubyist*. Manning, 2009

[boo] *Bootstrap Exposition*. http://expo.getbootstrap.com. – Zugriff: 11.09.2013

[Bra] Brady, David: *CoffeeScript Cookbook*. http://coffeescriptcookbook.com/. – Zugriff: 30.07.2013

[Bur11] Burnham, Trevor: *CoffeeScript Accelerated JavaScript Development*. Pragmatic Bookshelf, 2011

[Cha09] Chacon, Scott: *Pro Git*. Apress, 2009

[Croa] Crockford, Doug: *Javascript: The Good Parts*. http://www.youtube.com/watch?v=hQVTIJBZook. – Zugriff: 27.07.2013

[Crob] Crockford, Doug: *Private Members in JavaScript*. http://javascript.crockford.com/private.html. – Zugriff: 30.07.2013

[Cro08] Crockford, Douglas: *Javascript: The Good Parts*. O'Reilly, 2008

[eEG+] Çelik, Tantek ; Etemad, Elika J. ; Glazman, Daniel ; Hickson, Ian ; Linss, Peter ; Williams, John: *Selectors Level 3*. http://www.w3.org/TR/css3-selectors/. – Zugriff: 31.07.2013

[Era12] Erasmus, Michael: *CoffeeScript Programming with jQuery, Rails, and Node.js*. Packt Publishing, 2012

[For] Force, Internet Engineering T.: *The WebSocket Protocol*. http://tools.ietf.org/html/rfc6455. – Zugriff: 11.09.2013

[Fou] Foundation jQuery: *jQuery API*. http://api.jquery.com/. – Zugriff: 31.07.2013

[Fre12] Freeman, Adam: *Pro jQuery*. Apress, 2012

[Gita] GITHUB: *Ruby On Rails*. https://github.com/rails/rails. – Zugriff: 24.07.2013

[Gitb] GITHUB: *Sizzle Documentation*. https://github.com/jquery/sizzle/wiki/Sizzle-Documentation. – Zugriff: 31.07.2013

[Hana] HANSSON, David H.: *The Asset Pipeline*. http://guides.rubyonrails.org/asset_pipeline.html. – Zugriff: 04.08.2013

[Hanb] HANSSON, David H.: *A Guide to Testing Rails Applications*. http://guides.rubyonrails.org/testing.html. – Zugriff: 26.07.2013

[Mac13] MACCAW, Alex: *The Little Book on CoffeeScript*. O'Reilly, 2013

[Net] NETWORK, Mozilla D.: *Event Reference*. https://developer.mozilla.org/en-US/docs/Web/Reference/Events. – Zugriff: 31.07.2013

[Osm12] OSMANI, Addy: *Developing Backbone.js Applications*. O'Reilly, 2012

[RTH11] RUBY, Sam ; THOMAS, Dave ; HANSSON, David H.: *Agile Web Development with Rails*. The Pragmatic Bookshelf, 2011

[Sul13a] SULC, David: *Backbone.Marionette.js: A Gentle Introduction*. Leanpub, 2013

[Sul13b] SULC, David: *Structuring Backbone Code with RequireJS and Marionette Modules*. Leanpub, 2013

[Tho04] THOMAS, Dave: *Programming Ruby, The Pragmatic Programmers' Guide*. The Pragmatic Bookshelf, 2004

[Val11] VALIM, José: *Crafting Rails Applications*. The Pragmatic Bookshelf, 2011

[Wika] WIKIPEDIA: *CoffeeScript*. http://en.wikipedia.org/wiki/CoffeeScript. – Zugriff: 27.07.2013

[Wikb] WIKIPEDIA: *jQuery*. http://en.wikipedia.org/wiki/JQuery. – Zugriff: 31.07.2013

[Wikc] WIKIPEDIA: *Model 2*. http://en.wikipedia.org/wiki/Model2. – Zugriff: 02.08.2013

[Wikd] WIKIPEDIA: *Named Parameter*. http://en.wikipedia.org/wiki/Named_parameter. – Zugriff: 24.07.2013

[Wike] WIKIPEDIA: *Ruby*. http://en.wikipedia.org/wiki/Ruby_(programming_language). – Zugriff: 24.07.2013

[Wikf] Wikipedia: *Ruby Gems*. http://en.wikipedia.org/wiki/RubyGems. – Zugriff: 27.07.2013

[Wikg] Wikipedia: *Ruby On Rails*. http://en.wikipedia.org/wiki/Ruby_on_Rails. – Zugriff: 24.07.2013

[Wikh] Wikipedia: *Software Prototyping*. http://en.wikipedia.org/wiki/Software_prototyping. – Zugriff: 21.08.2013

[Wiki] Wikipedia: *Software Prototyping*. http://en.wikipedia.org/wiki/Software_prototyping. – Zugriff: 11.09.2013

Erklärung

Hiermit versichere ich, dass ich meine Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum: .....

(Unterschrift)