Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Diplom Thesis Informatics

TODO: ? Framework for connecting Trello to other services and applications

Sebastian Engel

11th September 2012

Referees

Prof. Dr. Torsten Grust
(Informatik)
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Name Zweitgutachter
( )
Medizinische Fakultät
Universität Tübingen

Engel, Sebastian:
Framework for connecting Trello to other services and applications
Diplom Thesis Informatics
Eberhard Karls Universität Tübingen
Thesis period: 13th march to 11th September 2012

Abstract

Trello is a collaboration webservice to manage projects and assign their to-do items to co-workers. There are many collaboration tools available today, but most of them are very basic. Trello however is very extensive and it is optimal fo small businesses. But although it works fine the way it's supposed to it also has its limitations. Trello as it is now is a closed system. Nothing get in or out unless Trello itself is used. But sometimes it would be handy if is would be possible to apply data of Trello to other applications. For example a CMS which should contain alist of completed theses which are already managed in Trello.

So this thesis addresses small scripts which let Trello interact with other web-services and applications. To accomplish this task in the most dynamic way possible an API wrapper of the Trello API in the Ruby programming language emerged.

Acknowledgements

Thanks to Prof. Grust and M.Sc. Tom Schreiber for the oppurtunity to work on this interesting project. A great thank you to my lectors Alena Dausacker and Jennifer Proehl. Thanks to Moritz Uhlig for introducing me to HTML in the first place. Thanks to Sabrina Pfeffer, Martin Lahl, Philipp Wolter, Thomas Zappe and Steffen Zietkowski for the support during my studies. Thanks to my parents for the financial support and the patience.

Contents

List of Figures

List of Listings

Nomenclature

API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
DBMS	Database Management System
ERB	embedded Ruby
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
MVC	Model View Controller
REST	Representational State Transfer
RFC	Request For Comments
RSS	Really Simple Syndication
UCS	Universal Character Set
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

UTC	Universal Time Coordinated
UTF-8	8-Bit UCS Transformation Format
XHTML	Extensible Hyper Text Markup Language
XML	Extensible Markup Language

Chapter 1

Introduction

Die Arbeit gliedert sich dazu wie folgt: Die Grundlagen von BlaBlaBla werden in Kapitel 2 erarbeitet. ... Eine Diskussion und ein kurzer Ausblick im Kapitel **??** beschliesen diese Arbeit.

Bevor wir uns der Auswertung bzw. Bewertung der gewonnenen Primärdaten zuwenden, wollen wir zunächst einige grundlegende Begriffe der deskriptiven Statistik wiederholen.

TODO: !

Chapter 2

Principles

2.1 Trello

2.1.1 How Trello works

Trello is a webservice by the New York City based web corporation Fog Creek Software[1]. It is a collaboration tool to manage projects and was launched in 2011[2].

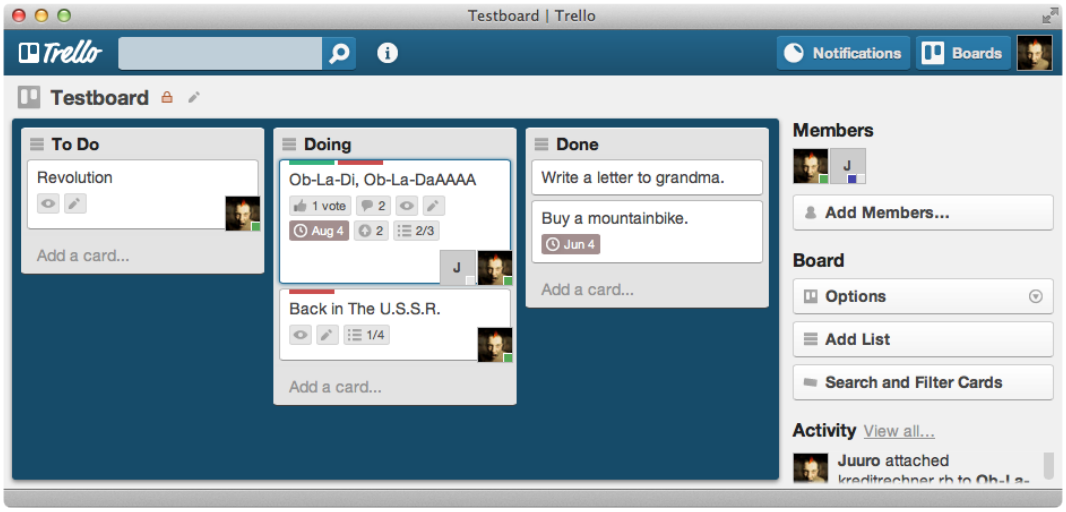

Figure 2.1: A Trello board.

[1]Official Fog Creek Software website: http://www.fogcreek.com
[2]The original launch post in the Trello blog: http://blog.trello.com/launch/

There is the concept of the so called *boards* which contain several configurable lists. Figure 2.1 shows a board with the three standard lists *To Do*, *Doing* and *Done*. In these lists the user can create to-do items. These to-do items are called *cards*. The cards can contain several additional data. Each card has a title and maybe a description, some assigned members, a due date, some labels, votes, checklists, comments and attachments. The creator of the board is the owner in the first place and the owner can add other Trello users to his boards and cards. So everone who's working on a project can see whats going on at the moment. Users who are assigned to a board can even create new to-do items by themselves. If somebody works at more than one company with many projects each there is the concept of *organizations*. This is useful in order to ensure a clear separation.

2.1.2 Why Trello

Trello is not just one of hundreds of thousands of to-do applications. It is streamlined for the purposes of small businesses. So it is perfect for the needs of a university with small groups of people working on the same things. Trello has already prooved its value for several months. The Trello website is written in HTML 5 with the use of AJAX where it makes sense. Trello provides an iOS [tre12a] and Android [tre12c] app. Both are constantly evolving. So the system is state-of-the-art. In addition the company behind Trello is not just a start-up with three employees, which is also of importance. A product of a small business, which is just based on the enthusiasm of the founders often doesn't last long. Fog Creek Software is over ten years old and has several products.

The first aim was to see the due dates of the cards someone is assigned to in Google Calendar. But thinking about that there were many other use cases for small scripts which could run as cron jobs on a server to serve several regular tasks. These scripts are described in more detail in Chapter 3.

2.1.3 Trello API

Trello has an API which is still in beta. But it is already very extensive. [tre12d]

Authentification

The access the private boards in Trello there has to be some kind of authentication. For user applications with a frontend the Trello API provides OAuth2. But because of the concept of OAuth2 the user is required to enter his Trello username and password. [oau12] My scripts are supposed to run on servers as cron jobs. There is no user who could manually enter data. For this kind of applications Trello provides a key/token-system. Every user has a private key. With this key the user can generate a token. This token will be sent along every request to the Trello API. The token tells Trello which scope the request can see. While generating a token the user can specify the scope of the token and when it will expire. The possible expriation date of a token is between one day and never. In our case we will use *never*. To generate a token one has to visit a special URL: `https://trello.com/1/authorize?key=SUBSTITUTEWITHYOURPRIVATEKEY &name=My+Application&expiration=never&response_type=token &scope=read,write` In this example the token would never expire and could read and write everything the user can access with the API. Other valid values instead of `never` for expiration would be `1day`, `30days`. `30days` is the default value. [tre12b]

REST

The Trello API is a *RESTful* web API. That means that the API is conform to the REST design model. REST is a common style of software architecture for distributed systems. It's built on four of the HTTP request methods: GET, POST, PUT and DELETE. An implementation of a REST web service follows four basic design principles:

- Use HTTP methods explicitly.

- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML , JSON, or both.

[res12]

Following these conventions a GET URL of a RESTful web service looks like this:

```
https://api.trello.com/1/cards/4fc8dd3e1b9ecf0c3571902f?
key='PRIVATEKEY&token=TOKEN
```

This is a GET request to get a specific card with the id `4fc8dd3e1b9ecf0c3571902f`. If this URL is visited in a browser (with correct *key* and *token*) the browser will show plain JSON. In order for Ruby to be able to work with it, it must somehow capture this data. To fulfill the requirements of REST in Ruby there are several gems. Here the RestClient gem is used. This GET request with the RestClient gem in Ruby looks like 2.1.

```
1 member = RestClient.get('https://api.trello.com/1/cards/4
    fc8dd3e1b9ecf0c3571902f?key='+$key+'&token='+$token)
2 pp JSON.parse(member)
```

Listing 2.1: GET request using RestClient.

In comparison to the open-uri library, which is included in the Ruby standard library, it's much more tidied up when it comes to POST requests.

```
1 response = RestClient.post(
2   'https://api.trello.com/1/boards',
3   :name => board['name'],
4   :desc => board['desc'],
5   :key => $key,
6   :token => $token
7 )
```

Listing 2.2: POST request using RestClient.

```
uri = URI('https://api.trello.com/1/boards')
req = Net::HTTP::Post.new(uri.path)

req.set_form_data(
  'name' => board['name'],
  'desc' => board['desc'],
  'key'=>$key,
  'token'=>$token
)

Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.
   scheme == 'https') do |http|
  response = http.request(req)
end
```

Listing 2.3: POST request with open-uri.

Listing 2.2 and listing 2.3 show the very same API call. But 2.2 is realised with RestClient and listing 2.3 with open-uri. Not only is the open-uri code much longer, but open-uri also doesn't detect the correct scheme from the given URI. If the call should performed in HTTPS this has to be set explicitly. This implies that for the handling of RESTful web services RestClient is the better choice.

2.2 Ruby

Ruby is a modern general-purpose object-oriented programming language. Its big difference to most other languages is that it focuses on humans rather than computers.

Yukihiro Matsumoto, the designer of Ruby, once said:

> Ruby is simple in appearance, but is very complex inside, just like our human body.[rub00]

That means that Ruby is very easy to read and is intuitive for humans even though it can perform complex tasks. This is achieved with English keywords

instead of brackets and curly brackets. The result for the programmer of this consistent philosophy is a very easy to read language which is also very plain. Because of the English words instead of abstract characters Ruby is easy to understand. Even non-programmers are mostly able to understand whats going on. So programmers produce a lot fewer errors while writing the code. A wrongly spelt word is more intuitively recognisable than a missing bracket or semicolon. [rub12a] More about Ruby can be found at http://www.ruby-lang.org.

2.2.1 RubyGems

Ruby has many methods and classes every Ruby installation provides[3]. But there are hundreds of extensions for special use cases – to communicate with RESTful Web APIs for example – made by third party developers. In Ruby such extensions are called *gems*. To manage and publish these third party libraries there is the standard *RubyGems*. It provides a standard format for third party libraries for Ruby, a tool to manage the installation of gems and a server for distributing the gems. [rub12c] Some Ruby distributions are delivered with several gems. Gems can be added to an existing Ruby installation at any time.

To install an additional gem on a Unix operating system the following command can be used:

```
gem install gemname
```

Where `gemname` is the name of the respective gem. If the installation performed without errors, the gem is ready to use. [rub12d]

To use an installed gem in a Ruby script the following code at the top of the script before the code starts is necessary:

```
1 require 'gemname'
```

Listing 2.4: Using the gem *gemname*

[3]Libraries that are included with the Ruby standard library: http://www.ruby-doc.org/stdlib-1.9.3/

Again, *gemname* stands for the name of the respective gem. If any gems that are not part of the Ruby standard library are used in this script, they are listed at the beginning of the related description. Further information at http://doc.rubygems.org and http://rubyforge.org/projects/rubygems/. Additionally http://rubygems.org is a reference book like website for Ruby gems.

2.2.2 Ruby and MySQL

In order to access databases in Ruby there are additional libraries needed. The Ruby standard library doesn't support any databases. To enable support for databases in Ruby additional libraries are needed. In this API wrapper the MySQL gem is used. This gem is an API wrapper around the MySQL C client API.**TODO: true?** MySQL is used here because it is the most popular open source database. [mys08] It is especially popular for the use in web applications. The scripts described here access primarily web applications, so it's the right choice to support MySQL.

```
1 my = Mysql.init
2 my.options(Mysql::SET_CHARSET_NAME, 'utf8')
3 my.real_connect(dbhost, dbuser, dbpassword, db)
4 my.query("SET NAMES utf8")
```

Listing 2.5: Initialising database connection.

Listing 4.26 above shows the initialisation of a new connection to a MySQL database with the Ruby MySQL gem. The variable `my` is a new database handler. If the correct charset isn't set it won't work. The host, the user, the password and the database name should be stored in seperate variables so they are easily editable. `SET NAMES utf8` in line 6 of listing 4.26 is a first execution of a MySQL query. It tells the server the charset name of future connections. In this example the server expects all future queries as UTF8 charset.

MySQL statements can executed directly or as *prepared statements*. The example in line of 4.26 is a normal statement which is executed directly. Listing 4.27 shows a longer example.

```
1 response = my.query("
2   SELECT fieldname1
3   FROM tablename
4   WHERE  fieldname2=value
5   AND fieldname3=value
6 ")
7 response.each do |row|
8   puts row
9 end
```

Listing 2.6: Example for a directly executed MySQL query.

The MySQL statement between line 2 and 5 responses with data. So the result of the query must be read somehow. The response of a `SELECT` statement is an array of rows from the given table. So to get the lines of the response table the `each` method is used to iterate through the array and print each row.

```
1 statement = my.prepare("
2   INSERT INTO tablename (
3     fieldname1,
4     fieldname2
5   )
6   VALUES (
7     ?,
8     ?
9   )
10 ")
11 statement.execute value1, value2
12 statement.execute othervalue1, othervalue2
```

Listing 2.7: `joomlaMultiple.rb` usage.

The listing above shows how the MySQL library for Ruby handles prepared statements. The big difference to directly executed statements are the wildcards in the `VALUES` part. It's more like a template for a statement. Although the statement doesn't contain all required values it's sent to the underlying DBMS of MySQL. The DBMS is able to perform query optimisation on this

statement template. To fill in the actual values the statement must be executed with the `execute` method. The lines 1 and 2 are two executions of the specified statement. In this step the wildcards are replaced by the wildcards. The DBMS performs additional query optimising if needed. Sometimes query optimisation depends on the values of the fields. Prepared statements can be executed multiple times. That saves performance, because the initial query optimisation must be performed only once. In addition performed statements are safer that directly executed statements. Because the DBMS processes the MySQL code seperately to the actual data it can distinguish between valid and invalid data. The DBMS checks every given value before writing it to the database. Because of that SQL injection isn't possible with prepared statements. So for every MySQL statement that writes data to the database it's reasonable to use prepared statements. [mys12]

2.3 JSON

All the responses to Trello API calls use JSON[4]. It is a subset of the JavaScript programming language. Despite its relation to JavaScript it's language independent. JSON is a data-interchange format like XML. But JSON is built on two structures. One is a list of key/value pairs. In most programming languages this is realised as a hash, struct, object or associative array. The other structure is an ordered list of values. This is realised as an array, list, vector or sequence in popular programming languages. In JSON itself these structures are called *object* and *array*. Objects start and end with curly brackets. Each key is followed by a colon and the key/value pairs are separated by commas. Arrays start and end with squared brackets. The values are separated by commas. Both can be arbitrary nested. Everytime one of the scripts saves content to any other place than Trello it's in the JSON format, too. That's because it guarantees easy compatibility with Trello. JSON can also be saved in files. A JSON file has the suffix `.json`. [jso12a]

```
1 {
```

[4]RFC document for JSON: The application/json Media Type for JavaScript Object Notation (JSON) http://www.ietf.org/rfc/rfc4627.txt

```
    "id": "4eea4ffc91e31d1746000046",
    "name": "Example Board",
    "desc": "This board is used in the API examples",
    "lists": [{
        "id": "4eea4ffc91e31d174600004a",
        "name": "To Do Soon"
    }, {
        "id": "4eea4ffc91e31d174600004b",
        "name": "Doing"
    }, {
        "id": "4eea4ffc91e31d174600004c",
        "name": "Done"
    }]
}
```

Listing 2.8: JSON example.

In listing 2.8 a JSON example is shown. This is the response of

`https://api.trello.com/1/boards/4eea4ffc91e31d1746000046?lists=open&list_fields=name,desc&key=PRIVATEKEY&token=TOKEN`

The JSON in listing 2.8 starts with a curly bracket. That means the uppermost structure is an object. Here are a few key/value pairs like `"id": "4eea4ffc91e31d1746000046"`. The key `"list"` has an array as value. So their value is in squared brackets. Each element of the array is an object again.

To process the received JSON Ruby needs to parse it first. For that purpose there is the a gem simply called *JSON*. It parses the JSON with the

`JSON.parse(receivedJson)`

command, where `receivedJson` is a variable that contains the plain JSON received from the Trello API with a GET request. After the parsing the JSOn objects are now represented as Ruby Hashes and the JSON arrays as Ruby arrays. To post content to the Trello API it has to be in JSON, too. So there is another method of the JSON class in Ruby which generates JSON from Ruby hashes and arrays.

```
JSON.generate(hashBoards)
```

The result is a valid JSON formatted string, ready to write to a JSON file or to send to an API.

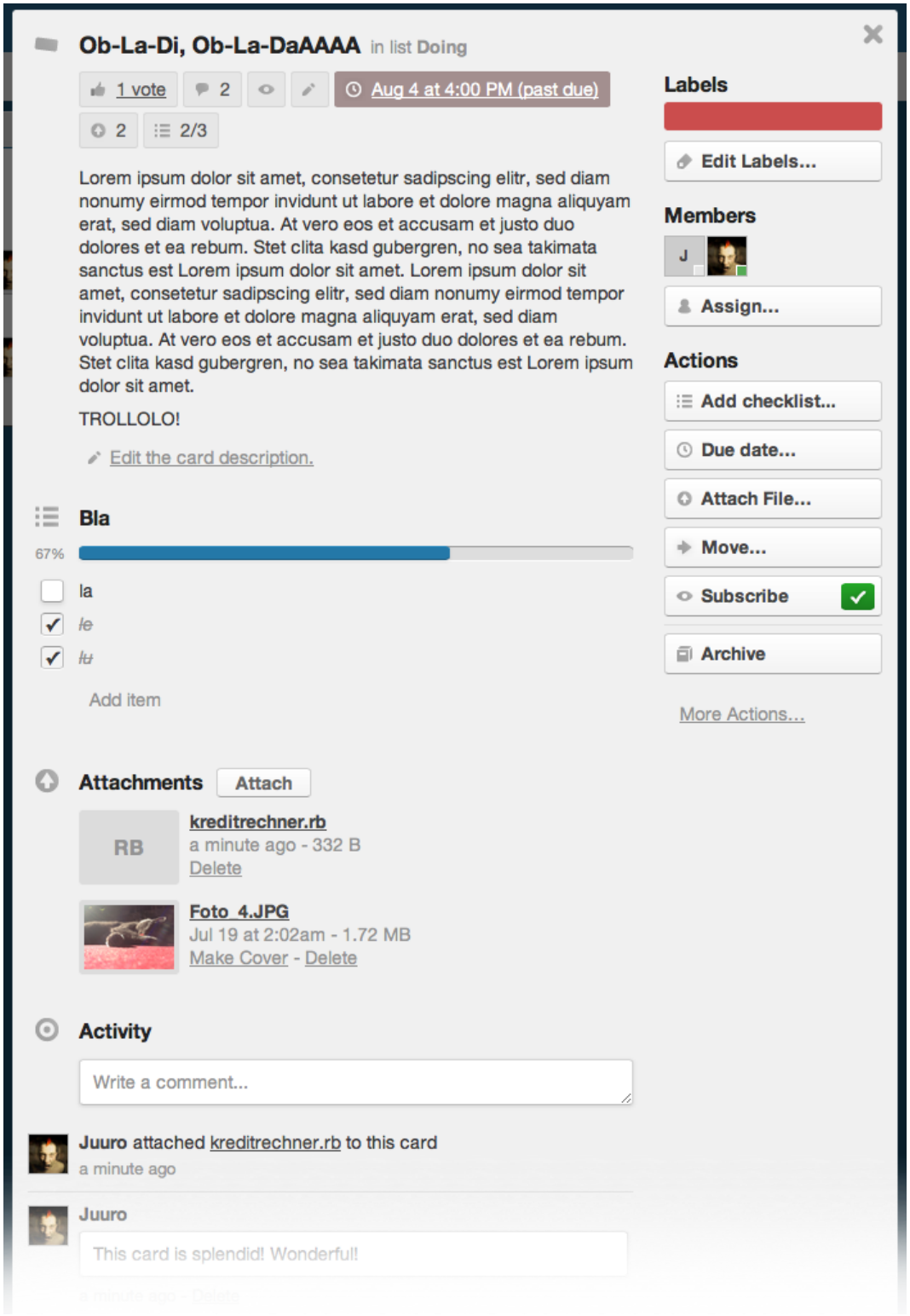

Figure 2.2: A opened card in Trello.

Chapter 3

Trello API wrapper

These scripts fulfill very different tasks, but they have also much in common. For example almost every script potentially loads single cards. This API wrapper represent Trello in Ruby. This is a kind of translation from Trello to Ruby and vice versa. Additionally, now the scripts can use the functions and in consequence they can stay very lightweight and clean. Almost everything that's possible with the Trello API is also possible with this API wrapper. But it doesn't cover all features, because the API is still in beta phase, so it changes quite quickly.

The API wrapper also has functions to pre-process data for Ruby. From a developer's point of view, Trello is all about cards. Cards are the only things in Trello with real data, not just meta data. So if the task is to get a board from the API it means to get the cards of the board. There is an API call to get all cards which are in a specific board. But with this call the developer doesn't get all the information about the cards. So the API wrapper has to execute the API call for a single card to cumulate all the information about all the cards of the board. This is the function of the API wrapper to keep the actual script clean. So the developer can work with the data and doesn't have to worry about determining it.

TODO: So stehen lassen? The API wrapper is meant to perform all API calls which are required by the scripts. None of the API calls should be performed by the scripts that use the bucket.

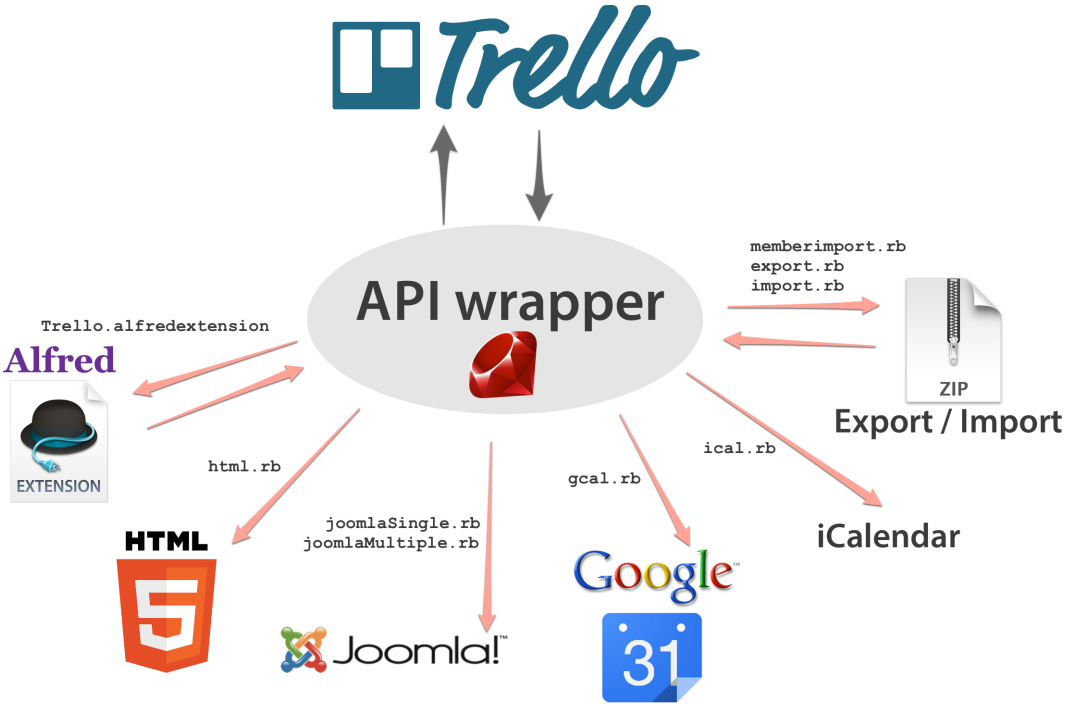

Figure 3.1: Connections between Trello, the API wrapper and the actual features. [rub][htm][joo12c][goo12a]

3.1 Error handling

TODO: Error Handling

Error handling in Ruby is realised with *begin* blocks. A *begin* block is like an *if* expression but for `exceptions`.

```
1  begin
2    my = Mysql.init
3    my.real_connect(dbhost, dbuser, dbpassword, db)
4  rescue => e
5    puts e.message
6  else
7    puts "It works!"
8  ensure
9    mysql.close if mysql
10 end
```

Listing 3.1: Error handling with *begin* blocks.

A *begin* block has several section. The first section is the *begin* section. Here the code to check is executed. The example in listing 3.1 shows a simple database connection. With the *begin* block araound it Ruby checks if there occur any exceptions while executing this code. If there are exception indeed, the *rescue* sections comes to help out. The error is stored in the variable `e`. At least the message can be printed here, to inform the user. A *begin* block may contain multiple *rescue* sections. It's possible to specify special types of exceptions so corresponding actions can be performed.

```
1 rescue Mysql::Error => e
```

Listing 3.2: `joomlaMultiple.rb` usage.

In listing 3.2 the `Mysql::Error` class is specified. In this *rescue* section only exception of this class will be handled.

An *else* section in a *begin* block like in 6 of listing 3.1 is allowed only if there is one or more *rescue* blocks. It is executed if there are no exceptions. The last section is the *ensure* block. It is executed no matter whether there is an exception or not. In the example the initialiser database connection is closed. This section is supposed to finish the execution of the whole block.

3.2 Command Line Interface

Almost every script needs some information. The information which every scripts needs is the key and token of the user whose account should be used to access Trello. The scripts have to know which cards, lists and boards they have to look at. So this information has to be passed on to the scripts, too. At first we set this information at the top of the script. But it emerged that it's very unpractical to hard code this in each script. So it would be impossible to use the same Ruby file with several Trello accounts. For every Trello account the user has to generate a dedicated file. The solution for this problem is a command-line interface (CLI). With a CLI the user can pass on information to the script in a predefined format, so the script knows exactly what to do. For every other call the user can specify different information for the same script.

The Ruby class OptionParser[rub12b] provides easy customisable command-line option analysis. The developer is able to specify its own options for each script. For this purpose a dedicated class is used. In order to let the actual script *know* about the CLI arguments the developer has to require the respective CLI class with the command-line option definitions.

```
1 ruby html.rb -c 4ffd78a2c063afeb066408b8
```

Listing 3.3: Example usage of a script with CLI.

An example usage of a script with CLI would look like Listing 3.3. The `-c` is a command-line option. If there is a string behind the option, like in this case, the string is a so called *argument*. But there are command-line options which stand by themselves. These are called *flags*. Flags are only for polar decisions.

```
1 # Trello list(s)
2 opts.on("-l", "--lists x,y,z", Array, "Ids of one or more
      Trello lists.") do |lists|
3   options.lists = lists
4 end
```

Listing 3.4: Definition of a command-line option

Listing 3.4 shows the definition of the option `-l` for passing one or more IDs of lists to a script. In Line 2 the word `Array` casts the list argument to an Array object.

OptionParse provides an automated help option. If the user types

```
ruby script.rb -h
```

he gets the explanation the developer wrote in the CLI class for this script with all possible options. This list is automatically generated by the definitions of the command-line options like in Listing 3.4. It works with `-help` and `--help` instead of `-h`, too.

```
1 Usage: ical.rb [options]
2 Select the input cards with -c, -l, -b or -a
```

```
3
4  Specific options:
5   -a, --[no-]all          Set this if all due dates of all
       cards of all boards  this user can see shall be used.
6   -l, --lists x,y,z       Ids of one or more Trello lists.
7   -b, --boards x,y,z      Ids of one or more Trello boards.
8   -c, --cards x,y,z       Ids of one or more Trello cards.
9   -k MANDATORY, --key     Your Trello key.
10  -t MANDATORY, --token   The Trello token.
```

Listing 3.5: Output of the `-h` option.

Listing 3.5 shows the Output of `ruby ical.rb -h`. These are the basic CLI commands used by every script. For some scripts there are additional commands. They are explained in their respective sections.

3.3 Actual methods

TODO: Working title!!! The API calls which are wrapped by the following methods need a private key and a token to access content in private boards. Thus, private key and token need not be sent to each method, it will be initialized in each script as global variables. The variable for the private key is alled `$key` and the one for the token is called `$token`.

3.3.1 Handling of date and time

Content in Trello may have set dates. These dates are represented by the Trello API as an ISO 8601[1] formatted string. The timezone of the date is UTC. To ensure that the correct time is always displayed the date has to be adapted to the local time.

[1]More information about ISO 8601 on the website of the International Organization for Standardization: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40874. The RFC 3339 is a profile of ISO 8601: http://www.ietf.org/rfc/rfc3339.txt

getDate(date, format='de')

```
1  def getDate(date, format='de')
2    fdate = Time.iso8601(date).getlocal
3
4    if format=='de'
5      return fdate.strftime('%d.%m.%Y %H:%M:%S')
6    elsif format=='us'
7      return fdate.strftime('%m/%d/%Y %I.%M.%S %P')
8    elsif format=='joomla'
9      return fdate.strftime('%Y-%m-%d %H:%M:%S')
10   elsif format=='ical'
11     return fdate.strftime('%Y%m%dT%H%M%S')
12   elsif format=='year'
13     return fdate.strftime('%Y')
14   elsif format=='iso8601'
15     return fdate.iso8601
16   end
17 end
```

Listing 3.6: `getDate()`

In line 2 of listing 3.6 the given date string in ISO 8601 format is parsed to a Ruby Time object. The `getlocal` is the important part here. This function of the Time class determines the server's time zone and readjusts the time accordingly. The method respects the daylight saving time in several time zones, too. For this function working as intended it's important that the correct time zone is set on the used server.

The `getDate()` method additionaly converts the date to othe formats.

3.3.2 Collect data from Trello

The several objects in Trello such as cards, lists and boards are accessed by these scripts regularly. To keep the actual scripts clean the API wrapper includes methods to determine and process the information of these onjects.

getBoardsByMember(memberId)

```
1 def getBoardsByMember(memberId)
2   boards = RestClient.get("https://api.trello.com/1/
      members/"+memberId+"/boards?key="+$key+"&token="+
      $token+"&filter=open")
3   boards = JSON.parse(boards)
4 end
```

Listing 3.7: `getBoardsByMember()`

The method `getBoardsByMember()` receives a member id and determines all boards accessible by this Trello account.

getListsByBoard(boardId)

```
1 def getListsByBoard(boardId)
2   list = RestClient.get("https://api.trello.com/1/boards/
      "+boardId+"/lists?key="+$key+"&token="+$token)
3   list = JSON.parse(list)
4 end
```

Listing 3.8: getListsByBoard()

getList(listId)

```
1 def getList(listId)
2   list = RestClient.get("https://api.trello.com/1/lists/"
      +listId+"?key="+$key+"&token="+$token)
3   list = JSON.parse(list)
4 end
```

Listing 3.9: getList()

getSingleCard(cardId)

```
1 def getSingleCard(cardId)
2   card = RestClient.get("https://api.trello.com/1/cards/"
      +cardId+"?key="+$key+"&token="+$token)
3   card = JSON.parse(card)
4 end
```

Listing 3.10: getSingleCard()

getCardsByBoard(boardId)

```
1 def getCardsByBoard(boardId)
2   board = RestClient.get("https://api.trello.com/1/boards
      /"+boardId+"/cards?key="+$key+"&token="+$token+"&
      filter=open")
3   board = JSON.parse(board)
4 end
```

Listing 3.11: getCardsByBoard() label

getCardsByList(listId)

```
1 def getCardsByList(listId)
2   list = RestClient.get("https://api.trello.com/1/lists/"
      +listId+"/cards?key="+$key+"&token="+$token+"&filter
      =open")
3   list = JSON.parse(list)
4 end
```

Listing 3.12: getCardsByList()

getCardsAsArray(arrayCardsStd, downloads = true)

```
1  def getCardsAsArray(arrayCardsStd, downloads = true)
2    arrayCardsFull = Array.new
3    directoryNameAttachments = File.join(Dir.tmpdir, "
       attachments")
4
5    arrayCardsStd.each do |card|
6      # export members
7      memberArray = Array.new
8      card['idMembers'].each do |memberId|
9        member = getMember(memberId)
10       memberArray << member
11     end
12     membersForCard = Hash.new
13     membersForCard['members'] = memberArray
14     card = card.merge(membersForCard)
15     # end export members
16
17     # export checklists
18     hasChecklist = getChecklist(card['id'])
19
20     if hasChecklist[0] != nil
21       arrayChecklists = Array.new
22       hasChecklist.each do |checklist|
23         hashChecklist = Hash.new
24         hashChecklist['id'] = checklist['id']
25         hashChecklist['name'] = checklist['name']
26         arrayItems = Array.new
27         checklist['checkItems'].each do |item|
28           hashItem = Hash.new
29           hashItem['name'] = item['name']
30           if isCompleted(card['id'], item['id'])
31             hashItem['completed'] = true
32           else
33             hashItem['completed'] = false
34           end
35           hashItem['pos'] = item['pos']
```

```
36            arrayItems.push(hashItem)
37          end
38          hashChecklist['items'] = arrayItems
39          arrayItems = nil
40          arrayChecklists.push(hashChecklist)
41          hashChecklist = nil
42        end
43
44        hashCheckListsForCard = Hash.new
45        hashCheckListsForCard['checklists'] =
             arrayChecklists
46
47        card = card.merge(hashCheckListsForCard)
48      end
49      # end export checklists
50
51      # export comments
52      if card['badges']['comments'] != 0
53        comments = getCardComments(card['id'])
54        hashCommentsForCard = Hash.new
55        hashCommentsForCard['commentsContent'] = comments
56        card = card.merge(hashCommentsForCard)
57      end
58      # end export comments
59
60      # export attachments
61      if card['badges']['attachments'] != 0
62        attachments = getAttachment(card['id'])
63        hashAttachmentsForCard = Hash.new
64        hashAttachmentsForCard['attachments'] = attachments
65        card = card.merge(hashAttachmentsForCard)
66
67        if downloads
68          # url runterladen
69          attachments.each do |attachment|
70            fileDomain = URI.parse(attachment['url']).host
```

```
71          filePath = attachment['url'].gsub(URI.parse(
               attachment['url']).scheme+"://"+URI.parse(
               attachment['url']).host, '')
72          fileExtension = File.extname(attachment['url'])
73
74          fileName = attachment['id']+File.basename(
               attachment['url'])
75          puts "Downloading \'"+fileName+"\'"
76
77          if !Dir.exists?(directoryNameAttachments)
78            Dir::mkdir(directoryNameAttachments)
79          end
80
81          Net::HTTP.start(fileDomain) do |http|
82              resp = http.get(filePath)
83              open(directoryNameAttachments+"/"+fileName,
                    "wb") do |file|
84                  file.write(resp.body)
85              end
86          end
87        end
88        # url runterladen
89      end
90    end
91    # end export attachments
92
93    # export votes
94    if card['badges']['votes'] > 0
95      reply = RestClient.get(
96          'https://api.trello.com/1/cards/'+card['id']+'/
               membersVoted?key='+$key+'&token='+$token
97      )
98      members = JSON.parse(reply)
99      membersVotedArray = Array.new
100     members.each do |member|
101        membersVotedArray.push(member['id'])
102     end
```

```
103       hashMembersVotedForCard = Hash.new
104       hashMembersVotedForCard['membersVoted'] =
              membersVotedArray
105       card = card.merge(hashMembersVotedForCard)
106     end
107     # end export votes
108
109     arrayCardsFull.push(card)
110   end
111
112   return arrayCardsFull
113 end
```

Listing 3.13: getCardsAsArray()

TODO: Describe all the methods. Consider renaming for constancy?

3.3.3 Further information on a card

getCardActions(cardId)

```
1 def getCardActions(cardId)
2   actions = RestClient.get("https://api.trello.com/1/
      cards/"+cardId+"/actions?key="+$key+"&token="+$token
      )
3   actions = JSON.parse(actions)
4 end
```

Listing 3.14: getCardActions()

getCardComments(cardId)

```
1 def getCardComments(cardId)
2   actions = RestClient.get("https://api.trello.com/1/
      cards/"+cardId+"/actions?filter=commentCard&key="+
      $key+"&token="+$token)
```

```
3   actions = JSON.parse(actions)
4 end
```

Listing 3.15: getCardComments()

cardUpdated(cardId)

```
1 def cardUpdated(cardId)
2   reply = RestClient.get('https://api.trello.com/1/cards/
        '+cardId+'/actions?filter=updateCard&key='+$key+'&
        token='+$token)
3
4   updates = JSON.parse(reply.body)
5 end
```

Listing 3.16: cardUpdated()

cardCreated(cardId)

```
1 def cardCreated(cardId)
2   reply = RestClient.get('https://api.trello.com/1/cards/
        '+cardId+'/actions?filter=createCard&key='+$key+'&
        token='+$token)
3
4   updates = JSON.parse(reply.body)
5 end
```

Listing 3.17: cardCreated()

isCompleted(cardId, itemId)

```
1 def isCompleted(cardId, itemId)
2   completedItems = RestClient.get("https://api.trello.com
        /1/cards/"+cardId+"/checkitemstates?key="+$key+"&
        token="+$token)
```

```
3   completedItems = JSON.parse(completedItems)
4
5   completedItems.each do |item|
6     if item['idCheckItem'] == itemId
7       return true
8     end
9   end
10
11  return false
12 end
```

Listing 3.18: isCompleted()

getChecklist(cardId)

```
1 def getChecklist(cardId)
2   checklists = RestClient.get("https://api.trello.com/1/
      cards/"+cardId+"/checklists?key="+$key+"&token="+
      $token)
3   data = JSON.parse(checklists)
4
5   return data
6 end
```

Listing 3.19: getChecklist()

getAttachment(cardId)

```
1 def getAttachment(cardId)
2   attachments = RestClient.get("https://api.trello.com/1/
      cards/"+cardId+"/attachments?key="+$key+"&token="+
      $token)
3   data = JSON.parse(attachments)
4
5   return data
```

```
6 end
```

Listing 3.20: getAttachment()

TODO: Describe all the methods. Consider renaming for constancy?

3.3.4 Member information

getMember(memberId)

```
1 def getMember(memberId)
2   member = RestClient.get("https://api.trello.com/1/
      members/"+memberId+"?key="+$key+"&token="+$token+"&
      filter=open")
3   member = JSON.parse(member)
4 end
```

Listing 3.21: getMember()

isThisMe(memberId)

```
1 def isThisMe(memberId)
2   if getMember('me')['id'] == memberId
3     return true
4   else
5     return false
6   end
7 end
```

Listing 3.22: isThisMe()

getMembersByBoard(boardId)

```
1 def getMembersByBoard(boardId)
```

```
2   members = RestClient.get("https://api.trello.com/1/
        boards/"+boardId+"/members?&key="+$key+"&token="+
        $token)
3   members = JSON.parse(members)
4 end
```

Listing 3.23: getMembersByBoard()

3.3.5 Accessing CMS

trelloToJoomlaSingle(joomlaArticleId, articles)

Listing 3.24: trelloToJoomlaSingle()

trelloJoomlaSync(cardId, sectionid, catid, joomlaVersion)

Listing 3.25: trelloJoomlaSync()

Chapter 4

Applications

4.1 Export to HTML

Used libraries:

- `erb`
- `json`
- `rest_client`
- `pp`
- `kramdown`

```
1 ruby html.rb --title TITLE [-c CARDID[,CARDID]] [-l
     LISTID[,LISTID]] [-b BOARDID[,BOARDID]] [-a] [-t TOKEN
     ] [-k KEY]
```

Listing 4.1: `html.rb` usage.

The `html.rb` script exports the data from one or more cards to an HTML file. The resulting HTML file lists the cards one below another. The order is determined by the order in the command-line argument. If the command-line looks like listing 4.2 the script will at first process the list with the id `4ffd78ff7f0c71780cc5aa1c`. That means that all cards in

the list are in the HTML file and below these the single card with the id `4ffd78a2c063afeb066408b8`. In addition to the command-line options described in section 3.2 the option `--title` is used. Here the user has to specify a title for the web page. The title will be displayed at the top of the page in `<h1>` HTML tags.

```
1 ruby html.rb -l 4ffd78ff7f0c71780cc5aa1c
     -c 4ffd78a2c063afeb066408b8 --title 'Madness'
```

Listing 4.2: Example of a `html.rb` call.

Each card is displayed with all its information. This includes title, description, members, due date, labels, votes, checklists, comments and attachments.

Trello itself distinguishes between photos and other attachments. Normal attachments are linked under the description. Photos are embedded in the HTML code as thumbnails. Trello detects JPEG, GIF and PNG files as pictures and displays them as thumbnails. In addition to these formats the resulting HTML file displays TIFF, PSD, BMP and JPEG2000 as thumbnails, too. All modern Browsers support these formats.

The `html.rb` script generates static HTML. Of course the goal could also reached with a dynamic solution with PHP or Ruby on Rails. But the upside is that the user of the respective website doesn't have to wait for the webserver. Dynamic websites are mostly fast in the meantime, but with static HTML files the developer is on the safe side. This approach pays off especially if the data to be displayed doesn't change every minute. The server doesn't have to generate the whole data with every visit. It just to send the static HTML files.

4.1.1 Markdown

Markdown is a small lightweight plain text formatting snytax, designed by John Gruber. It's designed for the use with blogs and CMS. In these cases HTML is often too much. Markdown represents most of the features of HTML that are needed for writing. The designer of Markdown, had the goal that a text, written in Markdown, is still easy to read. John Gruber provides a

software tool, written in the Perl programming language, that converts the Markdown formatted text to valid HTML. [mar04]

```
### iCloud:

1.   Shared Photo Streams Now you can *share* just the **
   photos** you want, with just the people you choose.
2.   Reminder

------------------------------

Here is an example of AppleScript:

    tell application "Foo"
        beep
    end tell

![Apple logo](http://upload.wikimedia.org/wikipedia/
   commons/f/fa/Apple_logo_black.svg "Apple logo")
```

Listing 4.3: Example for a text written in Markdown.

```
<h3>iCloud:</h3>

<ol>
  <li>Shared Photo Streams Now you can <em>share</em>
      just the <strong>photos</strong> you want, with
     just the people you choose.</li>
  <li>Reminder</li>
</ol>

<hr>

<p>Here is an example of AppleScript:</p>

<pre>
```

```
13   <code>tell application "Foo"
14   beep
15   end tell</code>
16 </pre>
17
18 <p><img alt="Apple logo" title="Apple logo" src="http
     ://upload.wikimedia.org/wikipedia/commons/f/fa/
     Apple_logo_black.svg"></p>
```

Listing 4.4: Listing 4.3 converted to HTML.

Listing 4.3 shows a small example of Markdown. The `###` in line 1 is a header equal to `<h3>` in HTML. The first list item of the ordered list in line 3 contains italic and bold words. In line 6 there is a horizontal line. After a normal line of text a code block starts in line 10. At the end in line 14 there is a picture with `title` and `alt` texts. After the conversion it looks like listing 4.4 in HTML. The appearance, of course, depends on the used CSS on the respective websites. The appearance in Trello is as shown in figure 4.1.

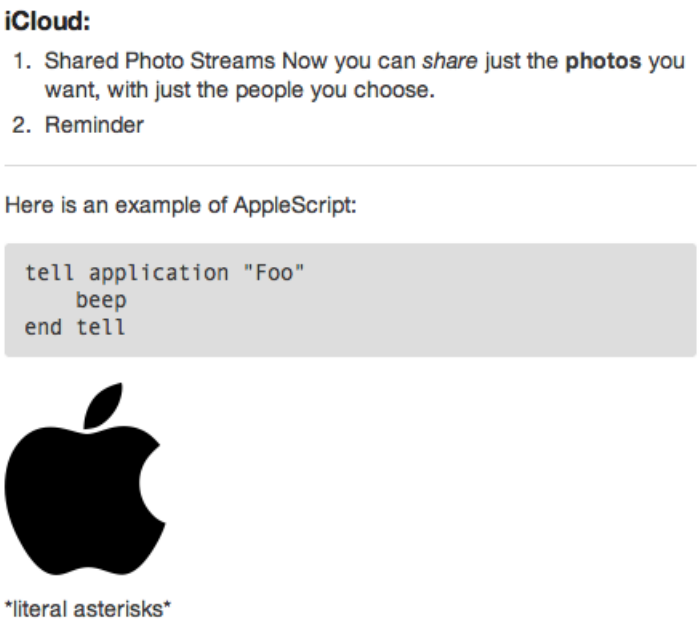

Figure 4.1: The browser view of the HTML converted from the Markdown in listing 4.3

Meanwhile Markdown has become quite popular. Many blogging platforms support it, at least there are Markdown plug-ins for most platforms. Trello

supports it in the description of cards. In the unlikely case that markdown reaches its limits inline HTML can be used. The only restriction is, that HTML block-level-elements have to be separated to the previous and following Markdwon blocks.

kramdown

In order to convert Markdown to HTML the gem kramdown is used. Figure 4.2 describes the convert options of kramdown. It converts HTML and Markdown to LaTeX, PDF and a special kramdown format, too. The kramdown format is an extended Markdown syntax. Like input formats it accepts HTML and kramdown besides standard Markdown. [kra12] These additional features of kramdown might be useful for future approaches. To generate lists bibliographies for scripts, papers or books.

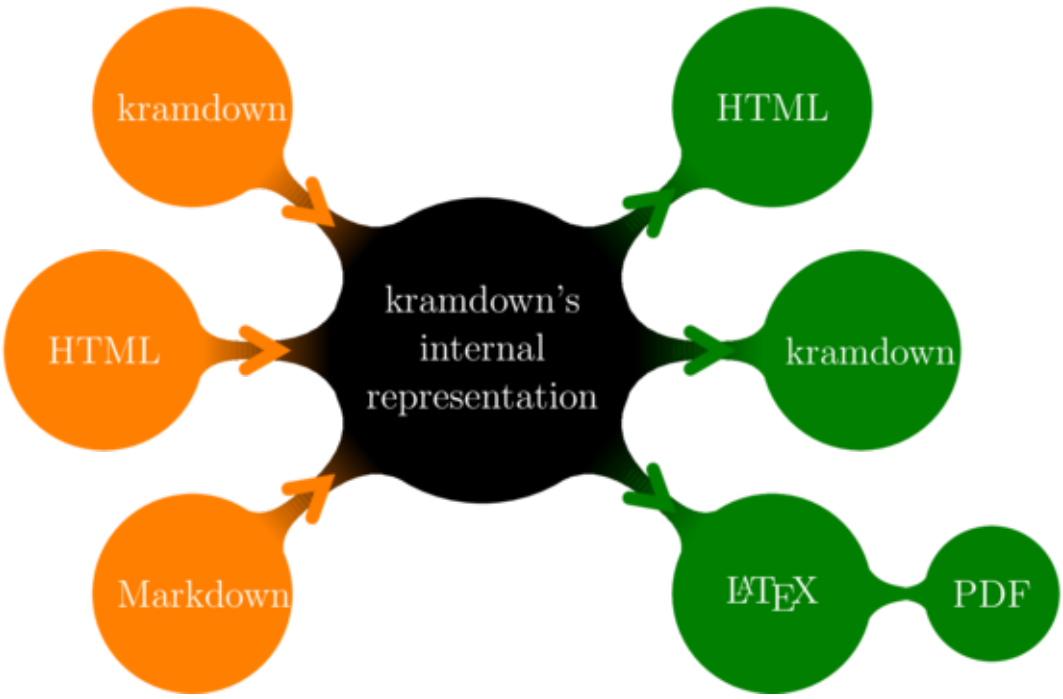

Figure 4.2: Overview about kramdowns converting options. [kra12]

4.1.2 Twitter Bootstrap Framework

In 2011 Twitter released Bootstrap[1]. Bootstrap is a collection of methods and scripts for creating front-ends of websites. It contains templates for site structuring, tables, text, buttons, menus, forms, lists and some other often used elements of websites. Additionaly some functions are supported by JavaScript. Bootstrap is written in HTML, JavaScript, CSS and LESS. LESS is a dynamic style sheet language and extends CSS. This enables the use of variables, functions, nested selectors and operators.[les12] Bootstrap is completely free of charge and under open source licence. [boo12]

Bootstrap evolved at Twitter during work on several projects with different libraries. The projects became inconsistent and a high administrative effort was needed. Some developers at Twitter led by Mark Otto worked on o tool to document and share common designs within the company. Twitter determined that this toolkit could be more than an intern helper tool. So they equipped it with all the common function which are needed for modern web development and released it on GitHub. [mar12]

4.1.3 HTML5

Twitter Bootstrap supports HTML5. HTML is the markup language for displaying content in a web browser. HTML5 is the latest revision of the HTML standard. It's an open format developed by the World Wide Web Consortium (W3C). HTML5 has been a working draft since 2008. In 2011 the HTML Working Group at W3C advanced HTML5 to Last Call. At the moment HTML communities all over the world are asked to confirm the standard. It is estimated that HTML5 will reach W3C Recommendation by 2014.[htm11] Although it is not yet finished, it is already widely used.

Before HTML5 there were two popular standards: HTML4 and XHTML1. XHTML1 defines a XML serialisation for HTML4. With HTML5 there is only one language called HTML. This language can be written in HTML syntax and XML syntax. [htm12d] The W3C realised the importance of smartphones

[1]Blog post form the lead developer about the launch of Twitter Bootstrap: https://dev.twitter.com/blog/bootstrap-twitter

in the future early on. They guided the development of HTML5 with the consideration of being able to run on low-powered devices. The market share of HTML5 enabled devices is still rising. [sma11] Smartphone sales even beat PC sales in 2012. Thus, it is becoming increasingly important that web apps work well on small devices. To ensure this HTML5 is used. The foundation created here can therefore be used as a basis for the future.

HTML5 introduces several new elements. Some of these are used in `html.rb`. At first there is the `<header>` element. A header is an area at the top of a web page. It is used for navigational aids, logos or a search bar. [htm12a] At the bottom is the `<footer>` element. The `<footer>` element can be used as a side wide footer and as a footer of sections. [htm12b] Here it is just used as side wide footer as preparation for data like licencing information, imprint and information about the author. The cards are enclosed in `<article>` elements. Every single card is represented as an `<article>` element. This makes sense because typically a card is an article, especially if the user would like the card to be displayed on a web page. The W3C requires an `<article>` element to be self-contained. [htm12c] A card is a collection of information which is ordered by the several types of data.

This foundation of generating HTML out of Trello can be used in various ways. It's predestinated for website with recurring kinds of information. Every business card style web page can definitely be managed with this script. Very small websites right through to whole blogs can build on it.

4.1.4 Templating with ERB

Every card is embedded in the same HTML structure. Templates specify this HTML without the actual data. Instead of the data there is just a wildcard and a few control structures. Templating systems are used to organise the source code in operationally-distinct layers. The design is completely handled in the template file. But the control structure is partially in the template file, too. Template engines are however unsuitable for a real separation of the data models and the logic components. For this porpose additional concepts such as Model View Controller are required (MVC).

ERB is a templating system for Ruby. It's part of the Ruby standard library.

ERB accepts every string as a template, no matter whether it is stored in a file, a database or some other kind of storage. ERB is mainly used for generating HTML files. It is also able to generate any other kind of structured text, like RSS feeds and XML. [erb11] [erb12]

While generating the HTML ERB copies the plain text parts of the template file directly to the resulting document. The parts which ERB has to process are marked with certain tags listed in listing 4.5

```
1 <% Ruby code -- inline with output %>
2 <%= Ruby expression -- replace with result %>
3 <%# comment -- ignored -- useful in testing %>
4 % a line of Ruby code -- treated as <% line %>
5 %% replaced with % if first thing on a line and %
     processing is used
6 <%% or %%> -- replace with <% or %> respectively
```

Listing 4.5: Recognised tags in ERB.

Only the first two tags are used here. The optimum would be if only `<%= Ruby expression -- replace with result %>` would be used. As this would imply that there are no control structures in the template file, just wildcards for data.

A real example of an ERB template is listing 4.6.

```
1 <small><%= getDate(card['due'], format='de') %></small>
```

Listing 4.6: Ruby method in ERB template.

A Ruby method is used as wildcard here. When processing the template file the method will be executed with the given variables and the result will be copied into the HTML file. But there are control structures, too. The block around the line in listing 4.6 is shown in listing 4.7

```
1 <% if card['due'] %>
2   <small><%= getDate(card['due'], format='de') %></small>
3 <% end %>
```

Listing 4.7: Ruby method in ERB template.

The *if* construct is embedded with the `<% ... %>` tag for ERB. This is because it's not a wildcard which is replaced with actual content. This tag is just for control structures.

The alternative to using a templating system is to write the HTML at the same time and in the same file when the data is processed. This would result in a very confusing file which produces equally confusing HTML code. If the developer made sure that the HTML code is well-structured the source code in the script would look even messier. That's because character escape codes like `\t` and `\n` have to be inserted manually in the source code. Otherwise the templating system takes over this task. An example of such a confusing mixing of HTML, character escape codes and Ruby is shown in 4.8.

```
1  htmlSite << "</strong></span></p>
2    \t\t\t\t<div style=\"text-align: left; padding-left: 5
       px;\"><span style=\"font-size: xx-small;\">"
3  htmlSite << description
4  htmlSite << "</span></div>
5    \t\t\t\t<div style=\"text-align: left;\"><span style=\"
       font-weight: normal; font-size: small;\">
6    \t\t\t\t\t\t<ul>"
7  if element.attachments != []
8    attachments.each do |attachment|
9      name = attachment.name
10     url = attachment.url
11     htmlSite << "\t\t\t\t\t\t\t\t<li><a href=\""
12     htmlSite << url
13     htmlSite << "\">"
14     htmlSite << name
15     htmlSite << "<a/></li>"
16   end
17 end
```

Listing 4.8: Generating HTML without a templating engine.

To represent the list of cards with the title in Ruby there is the Ruby class `webpage`. It is defined in listing 4.9.

```
class Webpage
  def initialize( title )
    @title = title

    @cards = [ ]
  end

  def add_card( card )
    @cards << card
  end

  def get_binding
    binding
  end
end
```

Listing 4.9: Generating HTML without a templating engine.

There are three methods. The `initialize( title )` method generates the actual instance of the class. The instances modeled according to this method contain the given title and an empty array for the cards. The `add_card( card )` method simply adds a new card to the `@cards` array. The last method is `get_binding`. It generates a `Binding` object of the current local variables.

```
templateFile = File.open("templateHtml.html.erb", "rb")
template = templateFile.read

rhtml = ERB.new(template)

webpage = Webpage.new( @htmlTitle )

cardsFull.each do |card|
  webpage.add_card(card)
end

html = rhtml.result(webpage.get_binding)

```

```
14 fileHtml = File.new("index.html", "w+")
15 fileHtml.puts html
16 fileHtml.close()
```

Listing 4.10: Generating HTML with ERB.

In listing 4.10 the template data is set up. At first in line 1 the template file is opened, in the next line read and saved in the `template` variable. So the template is saved as string in `template`. In line 4 with `rhtml` an instance of ERB is created. After that in line 6 the `Webpage` class is used. One instance with the given title is generated. In line 8 each card is added to the instance of `Webpage`. Finally in line 12 the `Binding` object of `webpage` is created. With the ERB method `result` the data in the `Binding` object and the template come together. This is the step where the wildcards in the template file are filled with the actual data of the `Binding` object. The resulting HTML code is saved in the string variable `html` and `html`, is saved to the file `index.html` in line 14.

4.2 Synchronisation Google Calendar

Used libraries:

- `erb`
- `json`
- `rest_client`
- `pp`
- `google/api_client`

Usage:

```
1 ruby gcal.rb [-c CARDID[,CARDID]] [-l LISTID[,LISTID]] [-
    b BOARDID[,BOARDID]] [-a] [-t TOKEN] [-k KEY]
```

Listing 4.11: `gcal.rb` usage.

Google Calendar is a free web service by Google for time-management. The service can be enabled in several calendar applications such as Apple Calendar (it was called iCal prior to Mac OS X 10.8) and Microsoft Outlook. Even all the important mobile operating systems support it. Google Calendar is one of the most popular calendar web services. One advantage over other suppliers is the excellent integration with all the other Google services, most of which are also very popular.

4.2.1 Google Calendar API

Google provides an API to access Calendar. There is even an API wrapper for Ruby made by Google. But either it contains many bugs or the documentation is poorly written. Some parameters the documentation says are able to send an API call at aren't actually able to do so. Google grants normal developers a courtesy limit of 10,000 requests per day. Developers who need more requests per day for their application have to negotiate with Google and to contract into a higher request rate.

```
1  client = Google::APIClient.new
2  client.authorization.client_id = ' '
3  client.authorization.client_secret = ' '
4  client.authorization.scope = 'https://www.googleapis.com/
       auth/calendar'
5  client.authorization.refresh_token = ' '
6  client.authorization.access_token = ' '
7
8  result = client.authorization.fetch_access_token!
9  client.authorization.access_token = result['access_token'
       ]
10
11 service = client.discovered_api('calendar', 'v3')
```

Listing 4.12: Initialisation of the Google Calendar API connection.

Authentication with Google is much more complicated than with Trello. Listing 4.12 shows the initialisation of the Google Calendar API connection. At first the project has to be registered in the Google APIs Console.

[goo12c] There the developer can get the `client_id` and the `client_secret`. The `scope` depends on the Google API the developer wants to use. Here it is `https://www.googleapis.com/auth/calendar`. [goo12b] To get the `access_token` this URL must be called:

```
https://accounts.google.com/o/oauth2/auth?scope=https%3A%2F%2Fwww.
googleapis.com%2Fauth%2Fcalendar&redirect_uri=https%3A%2F%2Foauth2-
login-demo.appspot.com%2Fcode&response_type=code&client_id=CLIENTID.
apps.googleusercontent.com&access_type=offline
```

If the request succeeds the response is as noted in listing 4.13.

```
{
  "access_token":"1/fFAGRNJru1FTz70BzhT3Zg",
  "expires_in":3920,
  "token_type":"Bearer",
  "refresh_token":"1/xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259
     HOF2aQbI"
}
```

Listing 4.13: Response of the token request.

The access token is the actual important value. But it expires after about an hour typically. If it is used after the expiration date the API will respond with an error. beacuse of that it's important to store the refresh token. Otherwise, the user has to enter his Google login data every time an access token expires. If the application looses the refresh token, the API calls will no longer work. No new access tokens can be generated. The user, or in this case the developer, has to obtain a refresh token manually again. [goo12d] Unfornately, how long the refresh tokens are valid is unknown.

4.2.2 Synchronisation

For the synchronisation to Google Calendar only cards with a due dated are considered. At first the script instructs the API wrapper to get all cards. Sadly the Trello API provides no filter method to get only the cards with a due date.

The next step is to check if the cards have due dates. If a card has a due date the script checks if this card is already added as an event in Google Calendar. To do so the Google library looks after this card on the basis of its id. If it's not already in Google Calendar the script has to add a new event to Google Calendar.

```
1  event = {
2    'summary' => card['name'],
3    'description' => card['desc'],
4    'location' => card['id'],
5    'start' => {
6      'dateTime' => getDate(card['due'], format='iso8601'),
7      'timeZone' => 'Europe/Berlin'
8    },
9    'end' => {
10     'dateTime' => getDate(card['due'], format='iso8601'),
11     'timeZone' => 'Europe/Berlin'
12   }
13 }
14
15 insertevent = client.execute(:api_method => service.
       events.insert,
16                 :parameters => {'calendarId' => 'primary'
                     },
17                 :body => JSON.generate(event),
18                 :headers => {'Content-Type' => '
                     application/json'})
```

Listing 4.14: Adding a new event to Google Calendar.

At first a new `event` object has to be created. That happens in listing 4.14 in lines 1 to 13. A hash is used for that. The received information about a card from Trello is stored in a hash. So to get the information `card['KEYWORD']` is used, where `KEYWORD` stands for the respective name of the cards field that is needed. In the location field of an event the script stores the card id. That's not exactly the correct kind of data to put in a location field. But to determine if a card is already represented as an event in Google Calender the script has

to use any unique identifier. Since there is no location for Trello cards anyway, the location field can be used. Sadly Google doesn't provide any *hidden* fields which developers can use for such purposes.

From line 15 is where the actual request happens. The `service.events.insert` tells the Google API which method the developer wants to adress. [goo12d] In this case it's the method to insert a new event to an existing calendar. In line 16 the developer has to specify the id of the calendar in which the new event should apper. Clearly `primary` is not an id. `primary` stands for the primary calendar. In Google Calendar the user can create several calendars for different purposes. One for work, one for private stuff and so forth. So `primary` is a valid value, too. In the following line the body is being sent. To do that the `event` object created before has to be converted to a JSON string. This is achieved by the `generate` method of the JSON class. [jso12b] In the last line of listing 4.14 the Google API is told that the body of this request is formatted as JSON.

If the event is already inserted in the calendar the script checks if the card's summary, description or due date have changed since the last synchronisation to Google Calendar. But before it checks back if this event really is the representation of the actual card. The Google API provides no method to search for events with a special location. So it isn't possible to search in the location of an event. The script looks in all fields of an event for the id of the corresponding Trello card. To ensure that the currently handled card is the same the currently handled event is the representation for, the script compares the location field of the event and the card id. If the comparison matches it runs almost the same code as in listing 4.14. But this time the Google Calendar API method which is used for updating the event is called `service.events.update`.

The third possible scenario is an orphaned event in Google Calendar. If a user in Trello removes the due date of a card or the card itself, the event in Google Calendar is no longer valid. To solve this problem the script loads all cards with due dates and all events from Google Calendar. All cardids are saved in an array and all location field entries in another. Now the array with the card ids from Trello is subtracted from the array with the location fields. The resulting array contains just events which don't have acorresponding card with a due date in Trello. After that the script checks if the remaining location

field entries are in the format of Trello card ids. But the only indicators that can be used are the length of the string – a Trelloc ard id has 24 characters – and if it contains only numbers and letters. TBut there's a problem with this approach. If there are other events which are not inserted by this script which have accidently location fields with 24 characters containing only numbers and letters, they will be deleted. It's not very likely that this happens. Names of cities or other typical used date in the location field don't have the length of 24 characters. GPS positions contain dots and are shorter, too. But the possibility of a match exists. The solution would be to use a dedicated calendar only for the use with this script. Or to be very careful with adding new events manually. Of course it's possible to enable this function completely. But in this case there will remain in Trello delted cards as orphaned events in Google Calendar. They have to be deleted manually.

4.3 Export to iCalendar

Used libraries:

- `icalendar`
- `date`
- `json`
- `rest_client`
- `pp`

Usage:

```
1 ruby ical.rb [-c CARDID[,CARDID]] [-l LISTID[,LISTID]] [-
    b BOARDID[,BOARDID]] [-a] [-t TOKEN] [-k KEY]
```

Listing 4.15: `ical.rb` usage.

iCalendar is a popular format to exchange calendar data of all sorts. It's built on the prior vCalendar standard released 1996 by the Internet Mail Consortium. [vca] Because of that some object name begin with a V. Originally it was

introduced in 1998 by the Internet Engineering Task Force Calendaring and Scheduling Working Group in RFC 2445. [rfc98] Today, the actual specification is RFC 5545. [rfc09] Meanwhile it is supported by many applications which work with events of any kind. iCalendar is the defacto standard in this field. Hence it has great compatibility with many programs. The standard defines the MIME type `text/calendar`. So iCalendar is mostly used for exchanging calendaring data, like sending invitations, and for providing public calendaring data, the timetable of the football world championship for example.

4.3.1 iCalendar format

```
1  BEGIN:VCALENDAR
2  CALSCALE:GREGORIAN
3  METHOD:PUBLISH
4  PRODID:iCalendar-Ruby
5  VERSION:2.0
6  BEGIN:VTIMEZONE
7  TZID:Europe/Berlin
8  BEGIN:STANDARD
9  DTSTART:19960811T073001
10 RRULE:FREQ=DAILY;INTERVAL=2
11 TZNAME:UTC+01:00
12 TZOFFSETFROM:+0200
13 TZOFFSETTO:+0100
14 END:STANDARD
15 END:VTIMEZONE
16 BEGIN:VEVENT
17 CATEGORIES:FAMILY
18 DESCRIPTION:4ffa4c76e75c29032a88ed19
19 DTEND;TZID=Europe/Berlin:20120804T170000
20 DTSTAMP:20120829T184941
21 DTSTART;TZID=Europe/Berlin:20120804T170000
22 SEQUENCE:1
23 SUMMARY:Ob-La-Di\, Ob-La-DaAAAA
24 TRANSP:TRANSPARENT
```

```
25 UID:2012-08-29T18:49:41+02:00_158310425@u-081-c222.eap.
     uni-tuebingen.de
26 URL:https://trello.com/card/ob-la-di-ob-la-daaaaa/4
     ffa4c5ce75c29032a88ea31/2
27 END:VEVENT
```

Listing 4.16: iCalendar example.

Listing 4.16 shows an example of a file in the iCalendar format this script can produce. The top-level element of an iCalendar file is the core object. The first line is *BEGIN*:VCALENDAR and the ending line has to be *END*:VCALENDAR. Between these two tags is the *icalbody*. Following are the iCalendar properties. These properties apply to the entire calendar. CALSCALE:GREGORIAN in line 2 of listing 4.16 defines the calendar scale of this calendar. Hence the standard calendar format is Gregorian world wide it'n not neccessary to specify it in the iCalendar file. The method specification can has other values than PUBLISH, such like REQUEST and CANCEL. REQUEST would be used if the sender wants to know if the receiver is free at the time of this event. CANCEL would be used when the provided events in this calendar should be cancelled. The PRODID simply specifies the product which generated the iCalendar file. The VERSION property specifies the version that is required in order to interpret the iCalendar file. Version 2.0 is the last version. In line 6 the definition of the timezone starts. TZID specifies the identifier of this time zone. TZOFFSETFROM and TZOFFSETTO specify both the offset to UTC. TZOFFSETFROM for daylight saving time and TZOFFSETTO for standard time. DTSTART in this context describes the first onset date-time for the observance. The RRULE property defines a repeating pattern for recurring events. TZNAME is just a name for the specified timezone. In 16 the definition of the actual event begins. Every single event is specified within an VEVENT object. The VEVENT objects are listed after each other. There is the possibility to divide the events into categories with the CATEGORIES property. The DESCRIPTION contains a description string of the event. Here the script enters the id of the respective card. DTSTART and DTEND specify the starting and ending times of the event. The DTSTAMP property contains the time of the export to iCalenar format. SUMMARY is the events title. The SEQUENCE is teh revision counter of an event object. If the user makes significant changes to the event, the SEQUENCE must be incremented. When the event object is created

the SEQUENCE is zero. TRANSP determines whether the event object consumes time on a calendar or not. Events that consume time with the calendar should have the TRANSP value OPAQUE. Events which are marked as OPAQUE can be detected as consuming time by free/busy time searches on the calendar. If events don't consume actual time on the calendar they should be marked as TRANSPARENT. Here the script gives all events an TRANSP value of TRANSPARENT because the due date of a Trello card is not an actual appointment with a duration. It's more like a deadline for a particular task. The UID countains information abaout the creator of the iCalendar data. In this case it's a time and the name of my computer in the network of Uni Tübingen. Finally the URL property contains simply the URL of the card at Trello. With this URL the user can visit the card from within his calendaring application. So he can visit and edit the card without navigation through Trello by himself. [ica03]

This is just a subset of the iCalendar geared to use with the `ical.rb` script. iCalendar supports much more. For example, to-do lists, recurring events, journals, searching for free/busy time and updating calendars. These features wouldn't be useful with the intented use with Trello.

4.3.2 Export

After the `ical.rb` script got all necessary information calculated by the API wrapper, a new VCALENDAR object must be generated. Of course is possible to write own methods to generate a valid iCalendar code. But there is a sophisticated Ruby gem that covers all needede features. This gem is simply called *icalendar*[2].

```
1 cal = Calendar.new
2
3 cal.timezone do
4   timezone_id             "Europe/Berlin"
5
6   standard do
7     timezone_offset_from  "+0200"
8     timezone_offset_to    "+0100"
```

[2]The icalendar gem's GitHub repository: https://github.com/sdague/icalendar

```
9       timezone_name           "UTC+01:00"
10      dtstart                 "19960811T073001"
11      add_recurrence_rule     "FREQ=DAILY;INTERVAL=2"
12    end
13  end
```

Listing 4.17: Generating a new `VCALENDAR`.

Listing 4.19 generates the iCalendar file in listing 4.16 until line 16. The `timezone` method creates a new `VTIMEZONE` object. In line 6 the *BEGIN*`:STANDARD` is generated and fillied in the following lines. The methods used to add the properties are not exactly self-explanatory. Therefore it is recommended to read the documentation carefully.

```
1  cardsFull.each do |card|
2    if card['due'] != nil
3
4      cal.event do
5        dtstart        getDate(card['due'], 'ical')
6        dtend          getDate(card['due'], 'ical')
7        summary        card['name']
8        description    card['id']
9        transp         "TRANSPARENT"
10       categories     ["FAMILY"]
11       sequence       0
12       url            card['url']
13     end
14   end
15 end
```

Listing 4.18: Generating the `VEVENT` object.

In listing 4.19 the iCalendar code for every event is created. The array `cardsFull` contains all cards which sould be checked if they have a due date. The source code block starting at line **??** must be executed for every card with a due date in Trello. In line 5 and 6 the `getDate` method is used. It computes the date in the correct format for iCalendar. If that's done the iCalendar object is created.

```
1 cal.publish
2
3 icalendar = File.new("icalendar.ics", "w+")
4 icalendar.puts cal.to_ical
5 icalendar.close()
```

Listing 4.19: Saving the iCalendar file.

The last step is to write the generated iCalendar objet to a file. In line 4 of listing 4.19 the `publish` method publishes the iCalendar object. This is an important step. It ensures that later in the iCalendar file the `METHOD` property is set to `PUBLISH`. In line 2 the `to_ical` method converts the Calendar object instantiated in the first line of listing 4.19 to a string in iCalendar format. This string is saved to a file which is named `icalendar.ics`. [ica12]

Now the `icalendar.ics` file can be sent in an email or uploaded to a web server. No matter on which way this file is distributed, if there's a software which is able to handle the iCalendar format, everybody can read it.

The iCalendar format has a field for attendees. If this is specified everyone can see who's invited to this event. But unfortunately it cann't be used with this script because the Trello API doesn't provide a method to get a users email adress. That's for privacy reasons, for sure, but it would be great if there were such an option. Maybe only for private boards though.

4.3.3 Comparison to the Google Calendar synchronisation

In comparison to the Google synchronisation, the script is much more simple. There is no need for synchronization because the iCalendar file is regenerated with each run of the script and there is no API to work with. There is only one file in the correct format to be created. That's much faster than working with two APIs and perform several API calls. The server has to compute less data and there's less traffic to the APIs while updating. But of course the iCalendar file must be available on a server. With the Google solution there is no need for an own web server. The another bigger problem with the

webserver is, that the URL is accesible for everybody. If the calendar includes critical data this is a disqualifier for the user or the company. Although it is possible to provide the server with password protection, this would only be more complicated. Either the calendaring software doesn't support .htaccess or any other web authentication technology orthe user has to enter his login information periodically. The only way to use it safely with critical data is to use it in an intranet. There won't be any need for additional protcetion. But for the domestic use and for most small companies the solution with Google should be sufficient. Google even provides an iCalendar feed itself.

4.4 Synchronisation to Joomla

Used libraries:

- `icalendar`
- `date`
- `json`
- `rest_client`
- `pp`
- `kramdown`

Usage:

```
1 ruby joomlaMultiple.rb --section SECTIONID --category
    CATEGORYID [-c CARDID[,CARDID]] [-l LISTID[,LISTID]]
    [-b BOARDID[,BOARDID]] [-a] [-t TOKEN] [-k KEY]
```

Listing 4.20: `joomlaMultiple.rb` usage.

Trello is perfect to depict recurring structures. Blog posts, for example, have such a structure which is repeated for each post. Each post has a title, a text, a creation date, an author, etc. The script `joomlaMultiple.rb` posts a card in Trello to the Joomla CMS.

Joomla[3] is one of the most popular CMS overall. It's open source and free of cost. Joomla is written in object oriented PHP [joo10] and uses software design patterns. [joo12b] Joomla is the result of a fork of the older CMS Mambo[4] from the year 2005. Mambo exists since 2000.

The Joomla CMS doesn't have an API for the here needed purposes. It's built on top of the Joomla *Platform* (formerly known as Joomla *Framework*). That's a framework which provides classes and methods to build a web application on top of it – such as Joomla. Since 2011 the Joomla Platformis distributed seperately from the CMS. This makes it easier for developers to use the Joomla Platform for web applications which are not a CMS. [joo12a] But because there is no API, developers have to access the underlying database to add content to Joomla.

This script has additionally to the standard set of command-line arguments two special arguments. The `--section` argument defines a section in Joomla under which the article should be listed. The `--category` means the same for Joomla categories. Sections in Joomla are the top content structure. Every category is related to a section. Articles in Joomla are related to a section and a category.

```
1 sectionid = options.section.first
2 catid = options.category.first
3
4 cardsToImport.each do |card|
5   trelloJoomlaSync(card['id'], sectionid, catid, '1.5')
6 end
```

Listing 4.21: Passing the needed information to the trelloJoomlaSync() method.

The `joomlaMultiple.rb` script gets its data from the API wrapper. The first two line of listing 4.21 read the section and category id from the command-line arguments. The API wrapper provides a method to accomplish the synchronisation of a Trello card to Joomla. Thus the `joomlaMultiple.rb` script has just to pass this function an id of the card which should be synchronised. In

[3] Joomla project website: http://www.joomla.org
[4] The Mambo foundations website: http://mambo-foundation.org

line 4 the script traverses the array with the cards to import to Joomla. The next line calls the `trelloJoomlaSync()` method for every card. It just passes the card id, the section id and the category id and the Joomla version which is used. Everything else is handled by the `trelloJoomlaSync()` method.

```
1 card = getSingleCard(cardId)
2 title = card['name']
3 description = Kramdown::Document.new(card['desc']).
    to_html
```

Listing 4.22: Getting standard card information.

`trelloJoomlaSync()` must determine a cards whole information itself. The `getSingleCard()` supplies the standard information of a card such as the title in line 2 of listing 4.22 and the description in line 3. Again, the `kramdown`gem is used to convert the Markdown formatted description string to HTML. But attachments and checklists would also be meaningful to depict in a CMS. Besides, the cards creation or update date is required to decide whether a card has changed or not.

```
1 changed = nil
2 if !cardUpdated(cardId).empty?
3   changed = getDate(cardUpdated(cardId).first['date'], '
      joomla')
4 else
5   changed = getDate(cardCreated(cardId).first['date'], '
      joomla')
6 end
```

Listing 4.23: Getting the date of a cards last change.

There is an API call for the last update of a card. But if the card has never been updated the response would be empty. So in this case the API call for the creation date must be used.

```
1 hasAttachment = getAttachment(cardId)
2
3 if hasAttachment[0] != nil
```

```
4   description += "<ul>"
5   hasAttachment.each do |att|
6     description += "<li><a href=\""+att['url']+"\">\""+
        att['name']+"\"</a></li>"
7   end
8   description += "</ul>"
9 end
```

Listing 4.24: Processing the attachments of a card.

To get the attachments of a card the `getAttachment(cardId)` method is used in line 1 of listing 4.24. If the API call contains attachment data the HTML tag `<ul>` is appended to the description. After that the attachments are appended as list items. Their names are simply linked with the URL to the file on Trellos servers.

```
1  hasChecklist = getChecklist(cardId)
2
3  if hasChecklist[0] != nil
4    hasChecklist.each do |checklist|
5      description += "<h4>"+checklist['name']+"</h4>"
6      description += "<ul>"
7      checklist['checkItems'].each do |item|
8        if isCompleted(cardId, item['id'])
9          description += "<li><del>"+item['name']+"</del></
             li>"
10       else
11         description += "<li>"+item['name']+"</li>"
12       end
13     end
14     description += "</ul>"
15   end
16 end
```

Listing 4.25: Processing the checklists of a card.

To process the checklists of a card the method `getChecklist(cardId)` is called first. If the response contains checklist data a `<h4>` HTML tag with the check-

lists name is added to the description. After that a `<ul>` is started again. For every checklist item the `isCompleted()` method must be called to resolve the determine of the checklist item. Depending on that the name of the item is displayed as crossed out or not. Rubys append method is used because the description is a HTML string already.

Now that the content is available to the script, it must be written to the database. Because the `trelloJoomlaSync()` method supports Joomla versions 1.5 and 2.5, every database query is existing twice. From Joomla 1.5 to Joomla 2.5 the underlying database structury has changed a bit.

```
1  begin
2    existingArticleQuery = my.query("
3      SELECT id, created, modified
4      FROM jos_content
5      WHERE metadata='"+cardId+"'
6    ")
7  rescue Mysql::Error => e
8    puts e
9  else
10   # if article doesn't exist insert it into the db
11   if existingArticleQuery.num_rows == 0
12     begin
13
14       # Insert new article, see listing 4.27
15
16     rescue Mysql::Error => e
17       puts e
18       return
19     ensure
20       stmt.close if stmt
21     end
22   else
23     # this should be only one because per Trello card id
         should only exist one article in Joomla
24     existingArticleQuery.each do |thisArticle|
25
```

```
26        existingId = thisArticle[0]
27        existingCreated = thisArticle[1]
28        existingModified = thisArticle[2]
29
30        # check if the modiefied timestamp im Trello is
             different to the modiefied timestamp in Joomla
31        begin
32          if existingModified != changed
33
34            # Update article, see listing 4.28
35
36            puts 'Changed: '+cardId+" : "+title
37          else
38            puts 'Nothing changed: '+cardId+" : "+title
39          end
40        rescue Mysql::Error => e
41          puts e
42          return
43        ensure
44          stmt.close if stmt
45        end
46      end
47    end
48  ensure
49    my.close if my
50  end
```

Listing 4.26: `joomlaMultiple.rb` usage.

In the first *begin* block of listing 4.26 from line 2 to line 6 the method searches the database table `jos_content` after an article with the actual handled cardId in the `metadata` field. If the response of this query is an empty array, the card isn't in the database. So the script must insert a new article into the database. Thats performed in line 14. The corresponsing MySQL code is showed in listing 4.27. If the resulting array in line 2 contains a row, the script must check if the new data is newer as the data in the database. This array can not contain more than one row, because the script inserts the article just once

if it's new, otherwise it replaces the old article with an updated version. In order to do that the script looks at the `modified` field of the existing article. This date is saved in the `existingModified` variable. The update date of the actual Trello card, that is determined in listing 4.23, is stored in the `changed` variable. If `existingModified` differs from `changed` the article is updated with the new date of the Trello card in line 34. In this behaviour the script assumes that the Trello card contains always the correct data. So it's not necessary to check back if the `changed` is actual newer. The MySQL statemnt of the update statemnt is showed in listing 4.28.

```
1  begin
2    stmt = my.prepare("
3      INSERT INTO jos_content (
4        title,
5        alias,
6        `introtext`,
7        state,
8        sectionid,
9        catid,
10       created,
11       created_by,
12       modified,
13       parentid,
14       ordering,
15       access,
16       metadata
17     )
18     VALUES (
19       ?,
20       ?,
21       ?,
22       1,
23       ?,
24       ?,
25       ?,
26       62,
```

```
27        ?,
28        0,
29        1,
30        0,
31        ?
32      )
33    ")
34
35    stmt.execute title, title.downcase, description.gsub(
         /'/, '&#39;'), sectionid, catid, changed, changed,
         cardId
36    puts 'New article: '+cardId+" : "+title
37  rescue Mysql::Error => e
38    puts e
39    return
40  ensure
41    stmt.close if stmt
42  end
```

Listing 4.27: Insert new article in the Joomla database.

```
1  stmt = my.prepare("
2    UPDATE jos_content
3    SET
4      title = '"+title+"',
5      alias = '"+title.downcase+"',
6      `introtext` = '"+description.gsub(/'/, '&#39;')+"',
7      state = 1,
8      sectionid = 5,
9      catid = 34,
10     created = '"+changed+"',
11     created_by = 62,
12     modified = '"+changed+"',
13     parentid = 0,
14     ordering = 1,
15     access = 0
16   WHERE
```

```
17      metadata = '"+cardId+"'
18  ")
19  stmt.execute
```

Listing 4.28: Updating existing Joomla article.

4.4.1 Joomla category page

After the cards from Trello are imported to Joomla as articles they are saved in Joomla, but not necessarily accessible on the web site. To view the articles on one page Joomla provides the item type *Category Blog Layout*. This is a menu object type. The user must specify a category when executing the script. All the articles which are assigned to this category will be displayed on a atomatically created web page. If the user wants to display all the imported articles from Trello on one page he have to create such an category blog layout for the category he specified. The user can customize the appearence of the site.

4.4.2 Single article

Sometimes it makes more sense if all the cards are stored as a single article in Joomla. The `joomlaSingle.rb` does that. This script works exclusively with Joomla version 1.5. This verion of Joomla uses still HTML tables for structuring the websites. So the script generates a special HTML site of all cars, similar to the `html.rb` script described in section 4.1.

4.5 Backup

Trello doesn't provide a complete backup solution. Single cards can be downloaded as *.json files, but that's not worth colling it a backup solution. It might be usefull to have the possibility to access the whole sight of a users Trello account offline. For example to use it in a dedicated Trello application or another project dedicated software. Anyhow the user is on the safe side if he backs his Trello boards up.

4.5.1 Export

Usage:

```
ruby export.rb -n filename -t TOKEN -k KEY
```

Listing 4.29: `export.rb` usage.

The `-n` command-line property specifies the filename which is used to store the exported data from Trello.

The `export.rb` script at first has to accumulate the data. Unlike the other scripts the whole information is needed all at once and in the most compact structure. So, there are four big types of content: boards, lists, cards and members. The method `getBoardsByMember`'me')) determines the boards of the member. `getMembersByBoard()` determines the members by board, means, this is an array which contains the boards as hashes. One value of these hashes contains all members that are assigned to this board. For each board `getListsByBoard()` determines the related lists. The cards are collected by board as well with the `getCardsByBoard` method. But because these API response contains not the whole information about a card the card array must be extended with the `getCardsAsArray()` method. This method also downloads attachments, if any. When all the data of these four content types is avalable, each array is saved to a mutual hash:

```
hashBackup = Hash.new
hashBackup['boards'] = arrayBoards
hashBackup['members'] = arrayMembersByBoards
hashBackup['lists'] = arrayLists
hashBackup['cards'] = arrayCards
```

Listing 4.30: `joomlaMultiple.rb` usage.

`hashBackup` contains the whole data of a users Trello account. Now it have to be saved to the backup file.

Saving to JSON files

The JSON library for Ruby described in section 2.3 is able to generate JSON out of Ruby hashes and arrays.

```
1 backupFile = File.new(File.join(Dir.tmpdir, 'backup.json'
    ), "wb")
2 backupFile.puts JSON.generate(hashBackup)
3 backupFile.close()
```

Listing 4.31: Using the temporary directory of the operating system to save the JSON to file.

In line 1 of listing 4.31 a file called `backup.json` is opened. `Dir.tmpdir` determines the path to the temporary directory of the operating system. This works under every operating system Ruby supports. Using the temporary directory has several advantages. Even if the saving process is cancelled it wouldn't result in a cluttered directory. The temporary directory is cleaned by the operating system regularly. The `join` method of Rubys `File` class merges the path of the temporary directory and the file name to a correct path to a file in the temporary directory. `File.new` makes this new file accesible with the variable `backupFile`. To store the generated hash in the file the script first generates the JSON with the `generate` method of the JSON library. The result is directly stored in the `backup.json` file because the *puts* method applied to the `backupFile` variable accesses the file in the temporary directory. The last step closes the `backup.json` file.

Now there is one JSON file and a directory with attachments. For exchanging the data this combination is not practical. A single file is needed. In order to achive that and to minimise file size the script uses the gem `zippy`. `zippy` is used to easily generate ZIP archives.

```
1 Zippy.create @filename do |zip|
2   zip['backup.json'] = File.open(backupFile)
3
4   Dir.entries(directoryNameAttachments).each do |file|
5     fileName = File.new(File.join(
        directoryNameAttachments, file), "r")
```

```
6       if file != "." && file != ".."
7         zip['attachments/'+file] = File.open(fileName)
8         fileName.close
9         File.delete(fileName)
10      end
11    end
12  end
```

Listing 4.32: Creating a ZIP archive out of the backed up data.

In order to create a single ZIP file `zippy` opens a `create` block. To add a file to the ZIP archive it can be specified like in an array. In line 2 of listing 4.32 a new file with the name `backup.json` in the ZIP archive is created. In the same step it is filled with the contents of the `backupFile` variable, which is basicly the `backup.json` in the temporary directory. In line 4 the script iterates through attachment directory in the temporary directory. Each attachment is stored in an `attachments` directory in the ZIP archive. After that the attachments are deleted from the temporary directory. Unless the operating system will delete them anyway if they are not in use, the attachments directory and the `backup.json` file are also deleted.

4.5.2 Import

Usage:

```
1 ruby import.rb -n filename -t TOKEN -k KEY
```

Listing 4.33: `import.rb` usage.

In contrast to the `export.rb` script, with the `-n` the user can specify a file to import. At first the scipt checks if the file is a ZIP file. For this it doesn't use the file name but the MIME type of the file.

```
1 if `file -Ib #{@filename}`.gsub(/;.*\n/, "") != "
    application/zip"
2   puts "ERROR: The backup\index{Backup} file has to be a
      ZIP\index{ZIP} file!"
```

```
3   abort
4 end
```

Listing 4.34: Checking if the file has the MIME type "application/zip"

In line 1 the `file -Ib #{filename}` is a bash call for receiving the MIME type of a file. Ruby executes it and with the gsub-Method it cuts the MIME part out of the received string. This shell script part in a ruby file is a bit messy. But only for this small case it would be complicated to use a separate gem.

TODO: What's a MIME type?

TODO: Problem adding members

TODO: Import problems with comments, votes, subscriptions.

4.5.3 Member import

TODO: Solution with memberimport.rb.

4.5.4 Close all boards

`closeboards.rb` is more of a helper tool for developers. Do you mean: While developing Trello, several test accounts emerge which now and then get crowded with test boards. The execution of this script will close all boards in the specified Trello account. The CLI isn't employed here because it would be too dangerous. If the wrong account was accitently indicated all the boards would go.

Chapter 5

Conclusion

TODO: Conclusion?

Chapter 6

Outlook

6.1 Trello Alfred Extension

Alfred [alf12] is a small Mac application which simplifies the way one can search the web or access all sorts of applications. It constist just of a input field which one cann access with a keystroke combination. It's like an extended Spotlight (on Mac) or Windows Search (on Windows). Developers can write extensions to access other webservices and applications with Alfred. It's even possible to run scripts with Alfred. With that possibility given it's perfect for acessing Trello while working in a fast and easy way.

There are three commands to add or read cards with this extension:

1. `trello board-name` will return the card-names and statuses of this board.

2. `trello board-name list-name` will return card-names and statuses of this list in this board.

3. `trello board-name text for a new card` will add a new card with the specified text to the first list of this board.

4. `trello board-name list-name text for a new card` will add a new card with the specified text to this list of this board.

If you enter `trello Berlin Visit the Reichstag` in Alfred the extension looks for a board called *Berlin*. If it finds nothign it looks for *Berlin Visit* and so on. So your board names shouldn't end with an imperative. The thought behind this operating principle is that it's very unlikely that a board name ends with an imperative and that imperatives are often used for card titles because cards are sort of a command.

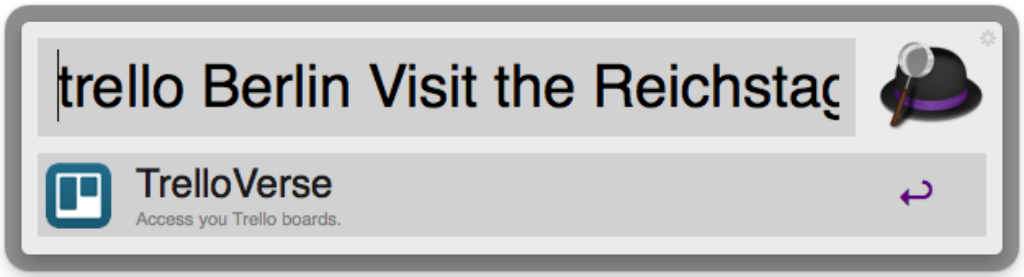

Figure 6.1: Alfred Extension for Trello: This command would add a card with the name *Visit the Reichstag* to the board called *Berlin*.

If you omit the text after the board name the extension will show you all card names of this board and its statuses.

Sometimes there are several boards with similar board names. In this case the extension will pick the "last"match. So if you have two boards called *Berlin* and *Berlin sightseeing* the extension will would pick *Berlin sightseeing*. This approach makes sense because if the extension would pick the first match, in this case *Berlin*, it wouldn't be possible to access *Berlin sightseeing*. In the case that one wants to access *Berlin* and add a new card beginning with *sightseeing* one has to put this board name betweet tick marks.

TODO: Maybe code it and verify practicability.

6.2 Native applications

Although Trello is an extremely good web-app, I'm of the opinion that a native application is always the better solution. The first reason is because it's a dedicated app and so it's integrated with the operationg system. Especially for to-do-applications it's an advantage that they can access the systems notification system, or that they could completely vanish in the background so they

don't bother the user while working. There are mobile applications for iOS [tre12a] and Android [tre12c] by Trello itself. But theres no Mac, Windows or Linux application.

A native application would even speed up the Alfred extension because the application could cache the data. So there hasn't to be an actual HTTP request for every command by the Alfred extension. And if a HTTP request necessary the user hasn't to wait because the application will handle the command in the background.

Bibliography

[alf12] Alfred app. http://www.alfredapp.com/, 08 2012.

[boo12] Twitter bootstrap. http://twitter.github.com/bootstrap/, 08 2012.

[erb11] An introduction to erb templating. http://www.stuartellis.eu/articles/erb/, 02 2011.

[erb12] Class: Erb (ruby 1.9.3). http://ruby-doc.org/stdlib-1.9.3/libdoc/erb/rdoc/ERB.html, 08 2012.

[goo12a] http://www.google.com, 08 2012.

[goo12b] Frequently asked questions - google data apis — google developers. https://developers.google.com/gdata/faq#AuthScopes, 08 2012.

[goo12c] Google apis console. https://code.google.com/apis/console/, 08 2012.

[goo12d] Google calendar api - google apps platform — google developers. https://developers.google.com/google-apps/calendar/, 08 2012.

[htm] W3c html5 logo. http://www.w3.org/html/logo/.

[htm11] W3c invites broad review of html5. http://www.w3.org/2011/05/html5lc-pr.html, 05 2011.

[htm12a] 4.4 sections — html standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-footer-element, 08 2012.

[htm12b] 4.4 sections — html standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-footer-element, 08 2012.

[htm12c] 4.4 sections — html standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/sections.html#the-article-element, 08 2012.

[htm12d] Html5 differences from html4. http://www.w3.org/TR/html5-diff/, 03 2012.

[ica03] icalendar specification excerpts. http://www.kanzaki.com/docs/ical/, 10 2003.

[ica12] File: Readme — documentation for icalendar (1.1.6). http://rubydoc.info/gems/icalendar/1.1.6/frames, 08 2012.

[joo10] The case for better architecture. http://people.joomla.org/groups/viewdiscussion/750-the-case-for-better-architecture.html?groupid=708, 10 2010.

[joo12a] Api - joomla! documentation. http://docs.joomla.org/API, 02 2012.

[joo12b] Developing a model-view-controller component/1.5/introduction - joomla! documentation. http://docs.joomla.org/Developing_a_Model-View-Controller_Component/1.5/Introduction, 08 2012.

[joo12c] www.joomla.de. http://www.joomla.de/, 08 2012.

[jso12a] Json. http://www.json.org/, 08 2012.

[jso12b] Json implementation for ruby. http://flori.github.com/json/doc/index.html, 04 2012.

[kra12] kramdown. http://kramdown.rubyforge.org/, 06 2012.

[les12] Less the dynamic stylesheet language. http://lesscss.org/, 08 2012.

[mar04] Daring fireball: Markdown. http://daringfireball.net/projects/markdown/, 12 2004.

[mar12] Bootstrap in a list apart no. 342 · deep thoughts by mark otto. http://www.markdotto.com/2012/01/17/bootstrap-in-a-list-apart-342/, 01 2012.

[mys08] Mysql :: Market share. http://www.mysql.com/why-mysql/marketshare/, 2008.

[mys12] File: Readme — documentation for mysql (2.8.1). http://rubydoc.info/gems/mysql/2.8.1/frames, 08 2012.

[oau12] Oauth community site. http://oauth.net/, 08 2012.

[res12] Restful web services: The basics. https://www.ibm.com/developerworks/webservices/library/ws-restful/, 08 2012.

[rfc98] Rfc 2445 - internet calendaring and scheduling core object specification (icalendar). http://tools.ietf.org/html/rfc2445, 11 1998.

[rfc09] Rfc 5545 - internet calendaring and scheduling core object specification (icalendar). http://tools.ietf.org/html/rfc5545, 09 2009.

[rub] Iap: Caffeinated crash course in ruby. http://sipb.mit.edu/iap/ruby/.

[rub00] [ruby-talk:02773] re: More code browsing questions. http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/2773, 05 2000.

[rub12a] About ruby. http://www.ruby-lang.org/en/about/, 08 2012.

[rub12b] Class: Optionparser (ruby 1.9.3). http://ruby-doc.org/stdlib-1.9.3/libdoc/optparse/rdoc/OptionParser.html, 08 2012.

[rub12c] Rubyforge: Rubygems: Projektinfo. http://rubyforge.org/projects/rubygems/, 08 2012.

[rub12d] Rubygems manuals. http://docs.rubygems.org/, 08 2012.

[sma11] One billion html5 phones to be sold worldwide in 2013. http://www.strategyanalytics.com/default.aspx?mod=pressreleaseviewer&a0=5145, 12 2011.

[tre12a] App store - trello. http://itunes.apple.com/us/app/trello/id461504587?mt=8, 08 2012.

[tre12b] Getting started — trello documentation. https://trello.com/docs/gettingstarted/index.html, 08 2012.

[tre12c] Trello android app available for download! — trello blog. http://blog.trello.com/trello-android-app-available-for-download/, 08 2012.

[tre12d] Trello documentation — trello documentation. https://trello.com/docs/index.html, 08 2012.

[vca] vcalendar overview. http://www.imc.org/pdi/.

Index

Statement of authorship

I certify that I have prepared this thesis independently and that I'm using only the tools mentioned here. All passages that have been taken from other works are characterized as borrowing. This thesis has been submitted in identical or similar form in any other program as an examination.

Place, Date Signature