SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

FCHPT-5415-70165

Timetable and Interactive Faculty Plan – Database Design and Data Processing

BACHELOR THESIS

2014 Pavol Ďurina

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF CHEMICAL AND FOOD TECHNOLOGY

Timetable and Interactive Faculty Plan – Database Design and Data Processing

BACHELOR THESIS

FCHPT-5415-70165

Study programme:	Automation, Information Engineering and Management in Chemistry and Food Industry
Study field number:	2621
Study branches combination:	5.2.14 Automation, 5.2.52 Industrial Engineering
Department:	Department of Information Engineering and Process Control
Supervisor:	Prof. Ing. Miroslav Fikar, DrSc.

Bratislava 2014 **Pavol Ďurina**

Slovak University of Technology in Bratislava
Ústav informatizácie, automatizácie a matematiky

Faculty of Chemical and Food Technology
Academic year: 2013/2014
Reg. No.: FCHPT-5415-70165

STU FCHPT

BACHELOR THESIS TOPIC

Student:	**Pavol Ďurina**
Student's ID:	70165
Study programme:	Automation, Information Engineering and Management in Chemistry and Food Industry
Study branches combination:	5.2.14 automation, 5.2.52 industrial engineering
Thesis supervisor:	prof. Ing. Miroslav Fikar, DrSc.

Topic: **Timetable and Interactive Faculty Plan – Database Design and Data Processing**

Specification of Assignment:

The aim of the work is to create a complex web application for interactive faculty plan. Moreover, it will provide navigation in student timetables. Also, it will simplify contact with teachers using their location data. The thesis is a part of a team project.

This project part is primarily focused on database layer from its creation to data filling. It also contains transformation of available data to desired form

Tasks:

1 – Implementation of database technologies for web application
2 – Parsing of various data sources
3 – Application of PHP framework with MVC architecture

Length of thesis: 40

Selected bibliography:

1. DuBois, P. *MySQL profesionálně : Komplexní průvodce použitím, programováním a správou MySQL.* Brno: Mobil Media, 2003. 1071 s. ISBN 80-86593-41-X.

Assignment procedure from: 17. 02. 2014

Date of thesis submission: 24. 05. 2014

L. S.

Pavol Ďurina
Student

prof. Ing. Miroslav Fikar, DrSc.
Head of department

prof. Ing. Miroslav Fikar, DrSc.
Study programme supervisor

Acknowledgement

I am very grateful for the support, supervision, encouragement, and patience of prof. Miroslav Fikar. I thank my colleague Rudolf Halás for fruitful cooperation on this project.

Abstract

Thesis is part of a team project resulting in web application that is be used as interactive faculty plan. In addition to providing layout it contains information about faculty staff offices thus simplifying student-teacher contact. Among other information it provides timetable information. This part of the project is focused on back-end of the web application that provides data required for visual front-end. Thesis describes database design, REST API that was created to interact with front-end and database and Python consumers used to fill the API with provided data.

Key words:

RESTfull, API, floor plan, database

Súhrn

Táto práca je časťou tímového projektu, ktorého úlohou je vypracovať interaktívny plán budovy fakulty. Webová aplikácia okrem rozloženia budovy poskytuje aj informácie o kanceláriách zamestnancov, čo v konečnom dôsledku uľahčuje kontakt medzi pedagógom a žiakom. Okrem základných informácií aplikácia poskytuje aj rozvrh.

Zameraním tejto časti projektu je dátová časť webovej apkikácie, ktorá poskytuje informácie vizualizačnej časti. Práca opisuje navrhnutú databázu, REST API, pomocou ktorého aplikácia spravuje dáta a API konzumentov napísaných v jazyku Python, pomocou ktorých sa aplikácia plní dostupnými dátami.

Kľúčové slová:

RESTfull, API, orientačný plán, databáza

Contents

List of Abbreviations

AIS	–	Academic Information System
API	–	Application Programming Interface
ASCII	–	American Standard Code for Information Interchange
CI	–	CodeIgniter
CRUD	–	Create, Read, Update, Delete
CSV	–	Comma-Separated Values
cURL	–	Client URL Request Library
DTD	–	Document Type Definition
FCFT	–	Faculty of Chemical and Food Technology
GPS	–	Global Positioning System
GUI	–	Graphical User Interface
HATEOAS	–	Hypermedia as the Engine of Application State
HTML	–	HyperText Markup Language
HTTP	–	Hypertext Transfer Protocol
ISO	–	International Organization for Standardization
JSON	–	JavaScript Object Notation
MVC	–	Model–View–Controller
PHP	–	Hypertext Preprocessor
RDBMS	–	Relational Database Management System
RELAX NG	–	Regular Language for XML Next Generation
REST	–	Representational State Transfer
RPC	–	Remote Procedure Call
SGML	–	Standard Generalized Markup Language
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
STU	–	Slovak University of Technology in Bratislava
STU	–	Slovak University of Technology in Bratislava
UCS	–	Universal Character Set

UTF-8	–	UCS Transformation Format—8-bit
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locators
W3C	–	World Wide Web Consortium
XML	–	Extensible Markup Language
XSD	–	XML Schema Definition

Chapter 1

Introduction

Cartography and maps are important part of human society for thousands of years. [24] As the technology progressed so did the maps. Our need for them is still apparent. This can be seen in integration of GPS in everyday use devices e.g. cellphones. There were 150 million global shipments of GPS capable units in 2009 with forecast of 750 million shipments in year 2014. [32]

Every year there are roughly 1300 new students who are enrolled to FCFT. [53] You can see them roaming through seemingly endless corridors of our faculty looking for classrooms, student laboratories or offices. If we include visitors, the need for some kind of solution becomes clear.

Maps help us with today's need for efficiency and interactive maps can provide us with additional information that would have to be searched for elsewhere. My colleague Rudolf Halás and I have decided to create an interactive faculty plan which will help to locate staff offices, lecture rooms and laboratories. Classroom occupancy will be amongst additional information which will help to locate vacant room to study in during free periods.

Publicly available computers with Internet connection in faculty lobby, decent wireless coverage across faculty, and Internet capable cellphones ensure comfortable use and accessibility.

Faculty plan web application should be based on widely used web technologies with modern approach in mind. Application will be divided into two parts: visual front-end and data back-end for which my colleague and I will be responsible, respectively.

Chapter 2

Goals

Our ultimate goal is to create Web application that will serve as floor plan and will offer additional information about the faculty. This application will need data and interface to access this data. My goals are:

- Design database structure that will contain necessary information for the floor plan of our buildings, faculty staff, and timetable events in the classrooms.
- Create API that will be used by application front-end to request, insert, and modify the data in database.
- Prepare example API consumers and populate API with available data.
- Use MVC architecture within PHP framework to maintain structure of the application.

Chapter 3

Used Web Technologies

3.1 Representational State Transfer

The Representational State Transfer (REST) was introduced in 2000 by Roy Fielding at the University of California, Irvine in his dissertation thesis [17]. Even though REST did not attract that much attention in the beginning, now, years after its introduction, major frameworks for REST have started to appear. For example Representational state transfer is part of Java since version 6. [22]. There are various Python/Django, C/C++/C# and even Ruby implementations [44].

Even though REST is not a standard, being just an architectural style, it uses HTTP and URL standards. Another standards with sufficient vocabulary can be used or created to be used within REST application [18]. In comparison with Simple Object Access Protocol, Remote Procedure Call over HTTP, proper utilization of well-defined HTTP protocol verbs (e.g. PUT, GET, POST) or status codes, gives REST big advantage [45].

Fielding in his dissertation defined six constraints that are applied to the architecture, while leaving the implementation of the individual components free to design: [17]

- Client-server separation: allowing their independence.
- Stateless: no client context being stored on the server between requests.
- Cacheable: clients can cache responses.
- Layered system: intermediate servers may improve system scalability or enforce security policies.
- Code on demand (optional): custom code (compiled components such as Java

applets or client-side scripts such as JavaScript) may be passed to extend functionality.

- Uniform interface: between clients and servers thus simplifying and decoupling the architecture.

3.1.1 RESTful Web Services

RESTful web service is a web service implemented using HTTP and adhering to the principles of REST architecture. Richardson [44] gives us Table 3.2 which shows how the HTTP methods are typically used to implement a web service. This leaves GET method safe and CRUD (Create, Read, Update, Delete) mapping between database and HTTP is listed in Table 3.1.

It is important that PUT and DELETE methods remain idempotent and GET method as nullipotent as defined in the standard [16].

3.1.2 RESTfull APIs on the Rise

The concept of API is not a completely new invention. However with the rise of of front-end JavaScript frameworks like Angular, Backbone.js and many others, smart phones, apps, digital businesses the need for constant improvements becomes obvious. Where would Facebook, Twitter, Snapchat be without so many applications? Simple, understandable and still powerfull API is the key that enables developers to create these apps.

As Brian Mulloy notes in his popular Teach a Dog to REST [34], RESTfull APIs are a way forward, but industry standard is still far from perfect implementation of REST. Even he suggests some changes in implementation of Table 3.2, as seen in Table 3.3.

Table 3.1: CRUD representation in SQL and HTTP

Operation	SQL	HTTP
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE	DELETE

Table 3.2: HTTP methods typically used when implementing a web service

Resource / Method	Collection URI: http://example.com/	Element URI: http://example.com/id
GET	List the URIs and perhaps other details of the collection's members.	Retrieve a representation of the addressed member of the collection.
PUT	Replace the entire collection with another collection.	Replace the addressed member of the collection, or if it doesn't exist, create it.
POST	Create a new entry in the collection.	Treat the addressed member as a collection in its own right and create a new entry in it.
DELETE	Delete the entire collection.	Delete the addressed member of the collection.

Table 3.3: HTTP methods suggested when implementing a web service

Resource / Method	Collection URI: http://example.com/	Element URI: http://example.com/id
GET	List the URIs and perhaps other details of the collection's members.	Retrieve a representation of the addressed member of the collection.
PUT	Bulk update collection	If addressed element exists update, if not return error.
POST	Create a new entry in the collection.	Return error.
DELETE	Delete the entire collection.	Delete the addressed member of the collection.

3.1.3 Accessing REST API

PHP CURL

PHP function *file_get_contents()* can be used to perform a basic GET requests. Curl URL Request Library (cURL) [49] is more flexible way to interact with a REST API as it was designed for use in cases like this. It allows to set HTTP headers, parameters and other properties of the request. Example 3.1 shows PHP function used to update a user with example API and cURL to make a POST request.

```php
function native_curl($new_name, $new_email){
    // Set up and execute the curl process
    $curl_handle = curl_init();
    curl_setopt($curl_handle, CURLOPT_URL, 'http://localhost/
        restserver/index.php/example_api/user/id/1/format/json');
    curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, 1);
    curl_setopt($curl_handle, CURLOPT_POST, 1);
    curl_setopt($curl_handle, CURLOPT_POSTFIELDS,
                array('name' => $new_name,
                    'email' => $new_email));
    $buffer = curl_exec($curl_handle);
    curl_close($curl_handle);
    $result = json_decode($buffer);
    if(isset($result->status) && $result->status == 'success'){
      echo 'User has been updated.';
    }
    else{
      echo 'Something has gone wrong';
    }
}
```

Example 3.1: Simple cURL Request

Python 2

In Python 2 it is possible to utilize (Example 3.2) standard urllib2 module [19]. However, a more simple to use (Example 3.3) is library requests[41] dubbed Requests: HTTP for Humans.

Browser Plugins

Browsers Chrome, Firefox, Safari and others support instalation of plugins like Advanced REST client [42], Postman - REST Client [4] and Simple REST Client [46].

3.2 MySQL

Relation model for databases as proposed by Codd [12] has since developped into multiple commercial and open-source applications. MySQL is world's most popular open source relational database management system (RDBMS) [13]. Structured Query Language (SQL) is programming language and ISO standard for managing RDBMS [5].

```
20 #!/usr/bin/env python
21 # -*- coding: utf-8 -*-
22 import urllib2
23 gh_url = 'https://api.github.com'
24 req = urllib2.Request(gh_url)
25 password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
26 password_manager.add_password(None, gh_url, 'user', 'pass')
27 auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
28 opener = urllib2.build_opener(auth_manager)
29 urllib2.install_opener(opener)
30 handler = urllib2.urlopen(req)
31 print handler.getcode()
32 print handler.headers.getheader('content-type')
33 # ------
34 # 200
35 # 'application/json'
```

Example 3.2: urllib2 request

```
36 #!/usr/bin/env python
37 # -*- coding: utf-8 -*-
38 import requests
39 r = requests.get('https://api.github.com', auth=('user', 'pass'))
40 print r.status_code
41 print r.headers['content-type']
42 # ------
43 # 200
44 # 'application/json'
```

Example 3.3: urllib2 request

During the lifetime of database it has to serve many requests. In order to avoid data inconsistency and anomalies, such as update, insertion or deletion anomaly, process of table normalization was developed to minimize these [10][11].

Normal Forms (NF) of relational database theory provide us with criteria for determining a table's affinity to logical inconsistencies and anomalies. While each table has highest normal form it has to meet requirements for lower normal forms. Third normal form is predominantly satisfactory for normal use even though with little effort higher forms can be achieved [14]. Short desctiptions of first three normal forms are:

- First NF: Table faithfully represents a relation and has no repeating groups [10].

- Second NF: No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key [11].

- Third NF: Every non-prime attribute is non-transitively dependent on every candidate key in the table [11] .

Denormalization is process where table is intentionally restructured to lower highest normal form in order to increase speed [14]. This can bee seen in databases that predominantly perform read operations.

3.3 Extensible Markup Language

The Extensible Markup Language (XML)[7] is a subset of Standard Generalized Markup Language (SGML) [26]. XML and several other related specifications (XSLT, XPath) were developped by World Wide Web Consortium (W3C) Working Group in late 1990s. Working group was chaired by Jon Bosa.

Some of the design properties are: [7]

- straightforward usage over the Internet,

- human and machine readable,

- easy to design and create.

XML processor is a software module that is used to read XML documents and provide access to their content and structure on behalf of application. In order for the document to be parsed it has to meet well-formedness constraints defined in specification [7].

XML document can be divided into of Markup and Content. Markup strings can by identified by the rules stated in specification [7]. Tags begin with "<" and end with ">" while escape entities begin and end with "&" and ";", respectively.

Tag is a markup construct of elements which are building blocks of XML document. There are three types of tags: [15]

- start-tag: for example <h1>

- end-tag: for example </h1>

- empty element tag: for example <img/>

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <examples>
3   <example>
4     <h1>
5       Hello World!
6     </h1>
7   </example>
8   <example>
9     <img src="smiley.gif" alt="Smiley face" height="42" width="42" />
10  </example>
11 </examples>
```

Example 3.4: Simple XML Document

Element is a logical XML document component. It is composed by start-tag, content and end-tag (example 3.4 lines 4-6). Well formed content may contain another so called *child elements*. Element *example* beginning at line 3 (Example 3.4) has one child element *h1*.

Element *img* has name/value pair markup construct that exists within a start-tag or empty-element tag called attribute.

XML declaration is the first line in example 3.4. Even though it is not made mandatory in specification [7] it is recommended [37] and good practice.

3.3.1 Validation

Document markup of XML describes its storage and logical structure, associates attribute name-value pairs with its logical structures.

Document Type Declaration (DTD) is mechanism defining constraints on the logical structure of XML document and thus supporting the use of predefined storage units.

An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it. [7] This means that DTD provides grammar for XML.

Several other solutions like XSD and RELAX NG have been developed providing namespace awareness, much greater specificity, and better data-type handling.

Regular Language for XML Next Generation

RELAX NG is schema language for XML. Main reason for use are that it is easier to write, it is easier to generate, and it is easier for applications to use. Another XML

schemas like XML 1.0 DTD or XSD can be automatically generated from RELAX NG schema. [52]

RELAX NG has highly restricted built-in datatype library. This should not be seen as a limitation, it is a fundamental design decision. Structure and content validation is seen as different problem.

Most of the disadvantages posed by XML can be overcome by using JSON.

3.4 JavaScript Object Notation

JavaScript Object Notation (JSON) is an open standard format that uses human-readable text to transmit data objects. It is used as an alternative to XML mostly to transmit data between a server and web application. Official media type is "application/json".

Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages. [28] However the licence [30] contains clause “The Software shall be used for Good, not Evil.”, which can couse trouble [31]. The JSON format was originally specified by Douglas Crockford and is currently described by two competing standards RFC 7159 [6] and ECMA-404 [1].

The size advantage over XML due to absence of closing tags can be easily diminished by good compression. However the data-type advantage of JSON over XML can be evident [29].

JSON is built on two structures: [28]

object A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

array An ordered list of values. Usually, this is realized as an array, vector, list, or sequence.

3.4.1 Validation

JSON Schema defines the media type "application/schema+json", a JSON based format for defining the structure of JSON data. JSON Schema provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data. [21]

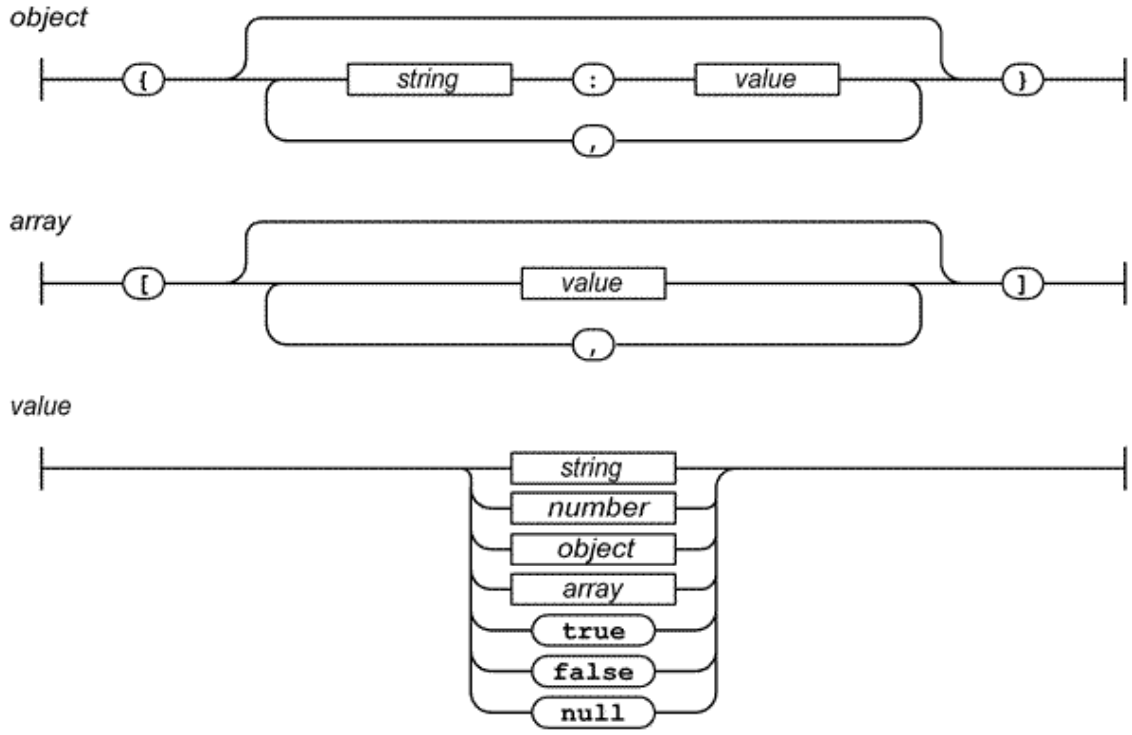

Figure 3.1: JSON forms

JSON Schema is based on the concepts from XML Schema and RelaxNG [9], but is JSON-based. Currently it is an Internet Draft version 4 and there are several validators available for different programming languages.

3.5 PHP

PHP is open-source server side scripting language capable of procegural or object oriented programming especially suited for web development. [3] In 1995 Rasmus Lerdorf released the first version to public. Since then it has become very popular. PHP can be used on all major operating systems and has support of most web servers today [3]. Currently released version is 5.5. New major release PHP 6 has been abandoned due to problems with intended UTF-16 implementation but mostly PHP works well and there are no burning issues pushing for new major release [25].

3.5.1 Model–View–Controller

Model–View–Controller as a design pattern was first described in 1979 [50]. It was designed for Smalltalk, an object-oriented, dynamically typed, reflective pro-

gramming language. Next step of MVC evolution was influenced mainly by NeXT, company founded in late 1980s by Steve Jobs. Early Apple GUIs were very similar with Smalltalk-80 v2 [35].

As the name suggests, MVC is a design pattern that allows developers to cleanly separate code into three parts: [51]

- **Models** maintain data.
- **Views** display data.
- **Controllers** handle user events affecting models and views.

In various frameworks slightly different approach to MVC can be seen but basic concept is upheld.

3.5.2 Laravel

Laravel is a web application framework with expressive, elegant syntax. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, Laravel combines the very best of what can be seen in other web frameworks, including frameworks implemented in other languages.Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications such as expressive migration system, and tightly integrated unit testing support. [38]

3.5.3 Libs and Packages

league/fractal

Fractal provides a presentation and transformation layer for complex data output, the like found in RESTful APIs, and works really well with JSON. [36]

soapbox/laravel-formatter

This package will help you to easily convert between various formats such as XML, JSON, CSV, and others.[47]

DataTables and Twitter Bootstrap

DataTables is a table enhancing plug-in for the jQuery Javascript library, adding sorting, paging and filtering abilities to plain HTML tables with minimal effort. The stated goal of DataTables is to enhance the accessibility of data in HTML tables. [27]

Bootstrap is the most popular front-end framework for developing responsive, mobile first projects on the web. [2]

Chapter 4

Realisation

4.1 Database Design

Creating the database and later changes are easily done by utilizing Laravel migrations. It is a type of version control for database. They allow a team to modify the database schema and stay up to date on the current schema state. [39]

As an example several table columns were renamed during initial stages. As an example *rooms.floor* was changed from integer to string.

Combining fzaninotto/faker package with Laravels Seeder class [39] initial data for development purposes were easily created. The only remaining seed is the initial user for the API administration part.

Both the first consumer and API were to be in Slovak and English so the database has to reflect that. From the multitude solutions for multilingual database I chose to implement separate translations table for each resource that would be translated.

Users and password reminders tables was created for the Authorization of POST, PUT and DELETE requests and access to administration part of API.

Migrations table is where Laravel keeps information on migrations and revisions table is polymorphic relations table where package venturecraft/revisionable keeps log of changes in database.

4.2 Input Output Data Format and Structure

API input and output is mostly HTTP requests and responses. When the HTTP request comes to the API, it's header is analyzed. If *Accept: application/xml* is found, response will be appropriately formated. Otherwise default *Content-type: application/json* will be used. Similarly POST and PUT request headers are checked for

Content-type header field where application/json or application/xml are accepted.

Amongst emerging standards {json:api} [48] was the one I took the most from. The most significant exception that is usually made is the structure of *data* value. Standard [48] proposes two possibilities i.e.both array and object depending on number of returned resources, more or just one respectively. It is easier for consumers when it is always easily iterable array as seen in returned response example 4.1. When the collection is empty, so is the 200 response data array.

Separation between API Output and database for each resource is handled by extended classes of league/fractal package [36] transformer class. Conversions between PHP array structure and JSON and XML is done by soapbox/laravel-formatter package [47]. During conversion to and from XML natural language is used to represent array structure e.g. *data* element will contain *datum* child elements. It might be beneficial to human readability, however programatically it is not ideal as long as the same pluralization library is not used on the other end as well.

```
1 {    "meta": {...},
2      "data": [...],
3      "embeds": [...]   }
```

Returned response 4.1: Structure of JSON 200 response

4.3 REST API

At the moment v1 API is deployed at `http://bc.durina.cc/api/v1`. Language can be selected by next URL segment /en for English and default /sk does not have to be present.

Available resources can be seen in response 4.2. All routes served are listed in appendix A.1.

Requests using verbs POST, PUT and DELETE have to contain HTTP Basic Authentification as defined by RFC 2617 [20]. Protocol over which route is served can be set in application configuration file routeProtocol.php.

4.3.1 API Requests and Responses

Object *meta* (returned response 4.3) in response with code 200 (4.1) contains Link Relations [33] providing pagination information. When addressing collection, query parameters *limit* and *offset* can be used and *next* and *previous* values are not empty

```
{    "links": [
        {   "rel": "self",
            "uri": "{en|sk}?/"},
        {   "rel": "people",
            "uri": "/people"},
        {   "rel": "rooms",
            "uri": "/rooms"},
        {   "rel": "events",
            "uri": "/events"},
        {   "rel": "courses",
            "uri": "/courses"}   ]}
```

Returned response 4.2: API v1 root URL

```
{   "self": "http://bc.durina.cc/api/v1/people/1107?",
    "next": "",
    "previous": "",
    "totalCount": 1   }
```

Returned response 4.3: meta object

strings when applicable. When these parameters are not present, sensible default is used which may differ depending on collection addressed.

Array *embeds* contains values for query parameter *embed*. Response data object then contains object with the same name and it's data value is array of objects that are related to the parent. There can be more than one comma separated requested resource. Embedable resources can be nested using dot notation, e.g. */rooms?embed=todayEvent,person.email*. Embed parameter is analyzed to eager load necessary related models so that respective Transformer class does not have to make additional database calls. Resulting data object can be seen in returned response 4.5

Related resources do not have to be embedded, They can be accessed via GET request. For example GET `http://bc.durina.cc/api/v1/en/rooms{roomid}/people` will return collection of people related to the room. These relationships are formed using POST requests. POST `http://bc.durina.cc/api/v1/en/rooms{roomid}/people/{personId}` will create relationship between person and room and will receive 201 response.

To remove relationship use DELETE request and receive 204 response. POST requests to this relationship URLs result in 405 error message (returned response 4.7).

New resource 201 response header *Location* can be subsequently used to create

```
[    "person",
     "todayEvent"    ]
```

Returned response 4.4: embeds array

```
{    ... ,
     "data": [{    ...,
                   "links": [],
                   "person": {"data": [{...
                                        "email":{"data":[...]}   }]   },
                   "todayEvent": { "data": [] } }] }
```

Returned response 4.5: embeded content

```
{    "data": [{    ...,
                   "relationships":[
      {"relationship":"rooms", "relationshipId":123},
      {"relationship":"courses", "relationshipId":456} ] }] }
```

Request body 4.6: relationships

```
{    "error": {    "message": "Method not allowed",
                   "status_code": 405 } }
```

Returned response 4.7: Error response

```
{    "data": [{    ...,
                   "translations":[
      {"lang":"sk", "translation":"sk text"},
      {"lang":"en", "translation":"en text"} ] }] }
```

Request body 4.8: translations

new relationships or relationships array can be included in POST payload 4.6. New resource will be created and related both to room 123 and course 456.

In addition to creating resource in one language and making PUT request to update translatable fields in other language, translations array can be included in data object. This takes precedes over translatable data in object (as in 4.8).

4.3.2 Resource Events

Resource URI is `/events/[id]`. This resource has no translatable fields. Only *scheduledFor* is not required. Possible relationships are with *rooms, people* and

courses.

Additional query parameters are *startsAt, startsBefore, startsAfter, endsAt, endsBefore, endsAfter, date* and *eventType*.

Event type 1 is for lectures, 2 for seminars, 3 is laboratory work.

```
{   "id": 1,
    "starts": "11:00:00",
    "ends": "12:00:00",
    "date": "2014-02-17",
    "eventType": 2,
    "scheduledFor": [{ "year": 1,
                       "group": 32 }],
    "links": [] }
```

JSON object 4.9: events resource data

Resource event.notes

Every *event* can have zero or more *notes*. Resource URI is `/events/id/notes/[id]`. This resource has one translatable field *note*.

Additional query parameter is *note*, which performs case insensitive LIKE search in database.

```
{   "id": 1,
    "note": "note example",
    "links": [] }
```

JSON object 4.10: notes resource data

4.3.3 Resource Rooms

Resource URI is `/rooms/[id]`. This resource has one translatable filed *tagCloud*. Possible relationships are with *people* and *events*. Building 2 is old building. Basement floor is s1, then it is numbered from 0. Room types are 1-unknown generic, 5-known generic, 2-office, 3-classroom, 4-restroom. Field *tagCloud* is pipe separated list of alternative names and tags (e.g. "SCHK|library")

Additional query parameters are *idLike, aisId, aisName, aisNameLike, number, building, floor, roomType* and *tagCloud*. All string searches in database are case insensitive.

```
{ "id": "r1-s1-s152",
  "aisId": 0,
  "aisName": "NB s152",
  "number": "s152",
  "building": 1,
  "floor": "s1",
  "tagCloud": null,
  "roomType": 1,
  "links": [] }
```

JSON object 4.11: rooms resource data

4.3.4 Resource People

Resource URI is `/people/[id]`. This resource has no translatable fields. Only *name* and *surname* are required. Possible relationships are with *rooms* and *events*.

Additional query parameters are *firstName, lastName, name* and *aisId*, where name performs LIKE search in fields surname and name.

```
{ "id": 1107,
  "aisId": 3374,
  "prefixTitle": "prof. Ing.",
  "name": "Miroslav",
  "surname": "Fikar",
  "suffixTitle": "DrSc.",
  "links": [] }
```

JSON object 4.12: people resource data

Resource person.emails

Every *person* can have zero or more *emails*. Resource URI is `/people/id/emails/[id]`. This resource has no translatable fields.

Additional query parameter is *email*, which performs case insensitive LIKE search in database.

```
{   "id": 1,
    "email": "email@example.com",
    "links": [] }
```

JSON object 4.13: person emails resource data

Resource person.phones

Every *person* can have zero or more *phones*. Resource URI is `/people/id/phones/[id]`. This resource has no translatable fields.

Additional query parameter is *phone*, which performs case insensitive LIKE search in database.

```
{   "id": 1,
    "phoneNumber": "+421 123 456 789",
    "links": [] }
```

JSON object 4.14: person phones resource data

Resource person.details

Every *person* can have zero or more *details*. Resource URI is `/people/id/details/[id]`. This resource has two translatable fields *name* and *text*. Translations object is looks like JSON object 4.16.

Additional query parameter is *detail*, which performs case insensitive LIKE search in *name* and *text*.

```
{   "id": 1,
    "name": "Out of office",
    "text": "untill 1.2. 3456",
    "links": [] }
```

JSON object 4.15: person details resource data

```
{ "nameTranslation": "Out of office",
  "textTranslation": "untill 1.2. 3456",  "lang":"en" }
```

JSON object 4.16: person details translation

4.3.5 Resource Courses

Resource URI is `/courses/[id]`. This resource has one translatable field *name*. Possible relationship is with *events*.

Additional query parameters are *name, code* and *aisId*.

```
{ "id": 1,
  "code": "41600_4P",
  "aisId": 257747,
  "name": "Bachelor Project",
  "links": [] }
```

JSON object 4.17: events resource data

4.4 API Administration

Api administration is available at http://bc.durina.cc/login. It utilizes theme Flatly [40] from Bootswatch. Responsive behavior on small screen devices can be seen in Figure 4.1.

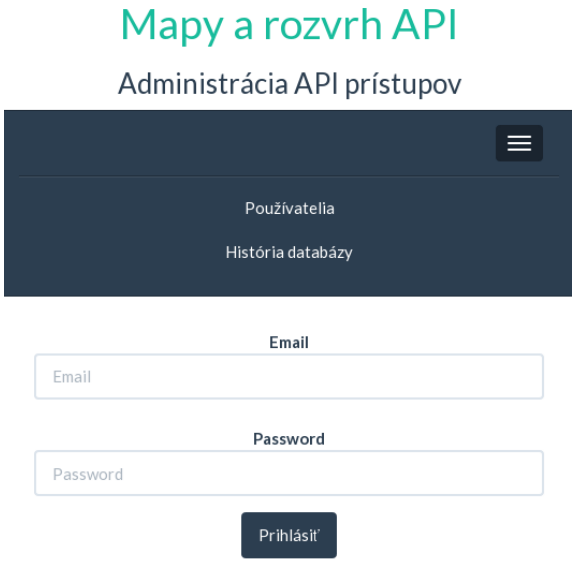

Figure 4.1: Responsive administrator login page

For editing users remote content bootstrap modal is loaded. When new user is created, standard password reminder link is sent to their email address, where they can set new password. Administrator credentials can be used to access API, however security implications of using the same credentials for HTTP Basic authentication have to be considered. Creating separate non administrator account is advised.

Database is initially seeded with credentials deleteme@admin:admin. These should be used to create new administrator and subsequently deleted as soon as possible.

Use of DataTables [27] can be seen in Figure 4.3. Package venturecraft/revisionable [8] was used to track changes in database. It tracks only updates and soft

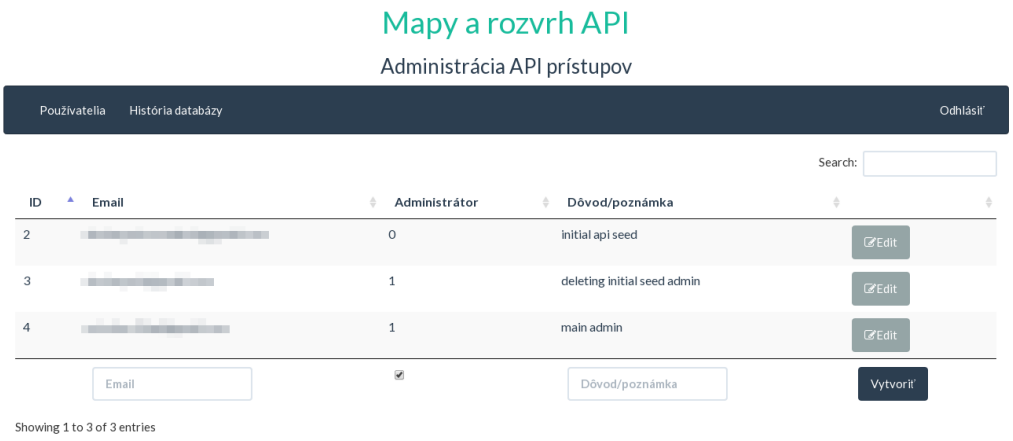

Figure 4.2: Users page

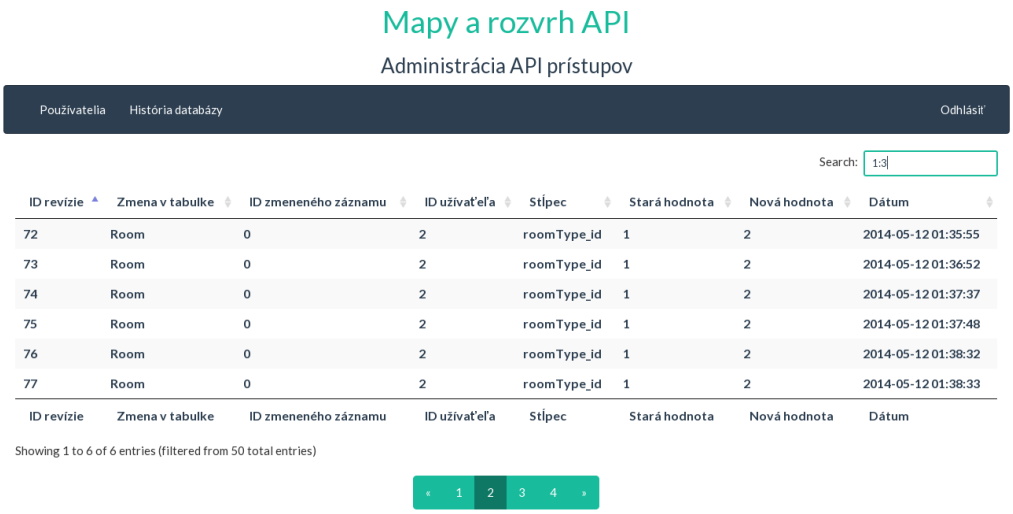

Figure 4.3: Revisions page

deletes which are not currently enabled.

Towards the end of this version of API *rooms.id* was changed from integer to string, which this package does not understand, thus id of changed row is 0 for rooms in database history overview (Figure 4.3).

4.5 Python API Consumers

To populate the API with available data, four python 2.7 classes were created. They utilize python packages *requests* [41], *requests_cache* [23] and *Beautiful Soup* [43]. Class *ApiConsumer* contains mostly helper methods that are used to create *.log* and *.err* files, retrieve user credentials, manipulate files and send POST or PUT requests to API. These classes assume same protocol for GET, POST, PUT requests to API.

4.5.1 PeopleCreator class

When instantiating, this class takes two parameters:

apiUrl Location of the API.

peopleUrl Location of FCFT employees list. Default is `http://is.stuba.sk/pracoviste/zamestnanci.pl?id=40`

GET requests with 200 response are forever permanently stored in SQLite based cache (file people_cache.sqlite). Method *clearCache()* can be used to clear it when necessary. When method *createPeople()* is called *peopleUrl* is downloaded and parsed for information.

Method *aisWait* is used to throttle non-cached requests for vCard info. Lowering the wait time is not advised as it can result in IP ban.

When room is found its type is checked and changed to office when applicable. When no room is found, person is placed into special room with id *r1-u-u2*.

4.5.2 CoursesCreator Class

When instantiating, this class takes these parameters:

apiUrl Location of the API.

predmetySkFile File location of downloaded Slovak page of public courses catalog in AIS. File encoding must be ASCII or UTF-8.

predmetyEnFile File location of downloaded English page of public courses catalog in AIS. File encoding must be ASCII or UTF-8.

When method *createCourses()* is called the two files are parsed and courses are bulk created in API.

4.5.3 TimetableCreator Class

When instantiating, this class takes these parameters:

apiUrl Location of the API.

startDate *datetime.date* object representation of the beginning of the term.

endDate *datetime.date* object representation of the end of the term.

Table 4.1: roomsFile structure

room id	room name
10	NB CH17

Table 4.2: coursesFile structure

course code	course name
42401_4B	P_Filozofia

ttFile Timetable data file location. File encoding must be ASCII or UTF-8. Default is *"../rozvrhy/rozvrhDrBoor.txt"*.

coursesFile Courses file location timetable data. File encoding must be ASCII or UTF-8. Default is *"../rozvrhy/predmetyDrBoor.txt"*.

roomsFile Rooms file location. File encoding must be ASCII or UTF-8. Default is *"../rozvrhy/miestnostiDrBoor.txt"*.

genericOccupiedCourseApiCode Course code for unrecognized courses. Default is *"xxxxx_00"*

When method *createEvents()* is called *roomsFile* is parsed to create dictionary to translate ids between these files and API, When room is found, type is checked and changed to classroom when needed. Generic room id "r1-u-u1" is used when no corresponding room is found. When *coursesFile* is parsed, there is no explicit id, so record number beginning with 1 is used. When course with corresponding code is not found in API, *genericOccupiedCourseApiCode* is used. Finally *ttFile* is parsed, ids translated and appropriate events for each day between *startDate* and *endDate* are bulk created.

This class uses forever permanent SQLite based cache (file timetable_cache.sqlite) for GET requests with 200 response. Method *clearCache()* can be used to clear it when necessary.

Table 4.3: ttFile structure

row id	room id	day	start time	duration	year	type	course id	group
58	89m	4d	11h	1p	2r	3t	891k	34

Chapter 5

Conclusions

Designed database proved to be sufficient for our needs. Table *rooms* was denormalized because some knowledge of buildings and floors is expected, and are highly improbable to change. If really needed in version 2 artificial resources *buildings* and *buildings.floors* could be made based on SQL DISTINCT queries. Translations column could be renamed to the column name it is translating to simplify translations object (request body 4.8 and JSON object 4.16).

API v1 is successfully deployed at `http://bc.durina.cc/api/v1` and consumed by front end visual application. Initial problem of longer responses was solved by production deployment (disabling debugging features of Laravel and generating optimized class loader).

Uniform data structure for GET, POST, PUT requests can be in JSON or XML. Some schema might be created in the future, however they will be either too restrictive or too general. They might provide more information in case of badly formated data resulting in 400 response.

API was successfully populated with provided data. However there is still high percentage of FCFT employees that do not have their office in AIS profile. It would be possible to create visual application where they can easily change the generic office they are automatically assigned to their actual office. Another application that could be built on top of this API, is for reporting mislabeled rooms.

MVC architecture can be best seen in the API administration part. Depending on the usefulness of the database history part *revisionable* package could be forked and modified to accept both integer and string keys, log the creation of new row and hard deletion.

Source code for both the API and python consumers can be found in Bitucket repository `https://bitbucket.org/hrpd/fchpt_api`.

Bibliography

[1] *The JSON Data Interchange Format*. ECMA International, 2013.

[2] Bootstrap. Retrieved May 5, 2014 from http://getbootstrap.com/, 2014.

[3] Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, and Jakub Vrana. What can PHP do? Retrieved May 5, 2014 from http://www.php.net/manual/en/intro-whatcando.php, 2012.

[4] Abhinav Asthana. Postman - REST Client. Retrieved May 5, 2014 from https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcmn.

[5] Alan Beaulieu. *Learning SQL*. O'Reilly, Beijing Sebastopol, 2009.

[6] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159 (Proposed Standard), March 2014.

[7] Tim Bray. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation, W3C, November 2008. http://www.w3.org/TR/2008/REC-xml-20081126/.

[8] Chris Duell. revisionable. Retrieved May 5, 2014 from https://github.com/venturecraft/revisionable, 2014.

[9] James Clark. RELAX NG specification. Retrieved May 5, 2014 from http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, 2001.

[10] E.F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.

[11] E.F. Codd. Further Normalization of the Data Base Relational Model. *IBM Research Report, San Jose, California*, RJ909, 1971.

[12] E.F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *SIGMOD Rec.*, 38(1):17–36, June 2009.

[13] Oracle Corporation. Market Share. Retrieved May 5, 2014 from http://www.mysql.com/why-mysql/marketshare/, 2012.

[14] C.J. Date. *Database in depth : relational theory for practitioners.* O'Reilly, Sebastopol, CA, 2005.

[15] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell, Third Edition.* O'Reilly Media, Inc., 2004.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[17] R.T. Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, 2000.

[18] R.T. Fielding and N. Richard. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[19] The Python Software Foundation. 20.6. urllib2 — extensible library for opening URLs. Retrieved May 5, 2014 from https://docs.python.org/2/library/urllib2.html.

[20] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[21] Francis Galiegue. JSON Schema: core definitions and terminology. Retrieved May 11, 2014 from http://json-schema.org/latest/json-schema-core.html.

[22] Marc Hadley and Paul Sandoz. SR-000311 JAX-RS: The Java API for RESTful Web Services . Retrieved May 5, 2014 from http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html, 2011.

[23] Roman Haritonov. Requests-cache documentation. Retrieved May 5, 2014 from https://requests-cache.readthedocs.org/.

[24] J.B. Harley and D. Woodward. *The history of cartography.* University of Chicago Press, Chicago, 1987.

[25] Tasman Hayes. PHP 6: Features, Release Date, Hosting and Download. Retrieved May 5, 2014 from http://smartwebdeveloper.com/php/php-6-features-release-date-hosting-download, 2011.

[26] ISO. *ISO 8879:1986: Information processing — Text and office systems — Standard Generalized Markup Language (SGML)*. August 1986.

[27] Allan Jardine. DataTables. Retrieved May 5, 2014 from https://github.com/DataTables/DataTables, 2014.

[28] json.org. Introducing JSON. Retrieved May 5, 2014 from http://json.org.

[29] json.org. JSON Example. Retrieved May 5, 2014 from http://json.org/example.html.

[30] json.org. JSON Licence. Retrieved May 5, 2014 from http://www.json.org/license.html.

[31] Lior Kaplan. Bug #63520 JSON extension includes a problematic license statement. Retrieved May 15, 2014 from https://bugs.php.net/bug.php?id=63520.

[32] A. Malm. GPS and Mobile Handsets - 4th edition. Retrieved May 5, 2014 from http://www.researchandmarkets.com/research/8dc21b/gps_and_mobile_han, 2010.

[33] Jan Algermissen Mark Nottingham, Julian Reschke. Link Relations. Retrieved May 11, 2014 from http://http://www.iana.org/assignments/link-relations/link-relations.xhtml.

[34] Brian Mulloy. RESTful API Design: Teach a Dog to REST. Retrieved May 5, 2014 from https://blog.apigee.com/detail/slides_for_restful_api_design_second_edition_webinar, 2010.

[35] Thomas Myer. *Professional CodeIgniter*. Wrox Press Ltd., Birmingham, UK, UK, 2008.

[36] The League of Extraordinary Packages. Fractal. Retrieved May 5, 2014 from http://fractal.thephpleague.com, 2014.

[37] Uche Ogbuji. Always use an XML declaration. Retrieved May 5, 2012 from http://www.ibm.com/developerworks/xml/library/x-tipdecl/index.html, 2004.

[38] Taylor Otwell. Laravel Philosophy. Retrieved May 5, 2014 from http://laravel.com/docs#laravel-philosophy, 2014.

[39] Taylor Otwell. Laravel Philosophy. Retrieved May 5, 2014 from http://laravel.com/docs/migrations, 2014.

[40] Thomas Park. Flatly. Retrieved May 11, 2014 from http://bootswatch.com/flatly/.

[41] Kenneth Reitz. Requests: HTTP for Humans. Retrieved May 5, 2014 from http://docs.python-requests.org/.

[42] restforchrome.blogspot.com. Advanced REST client. Retrieved May 5, 2014 from https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo/details.

[43] Leonard Richardson. Beautiful Soup. Retrieved May 5, 2014 from http://www.crummy.com/software/BeautifulSoup/.

[44] Leonard Richardson. *RESTful web services*. O'Reilly, Farnham, 2007.

[45] Kenn Scribner. *Effective Rest Services Via .Net: for .Net Framework 3.5*. Addison-Wesley Professional, Reading, 2009.

[46] Jeremy Selier. Simple REST Client. Retrieved May 5, 2014 from http://github.com/jeremys/Simple-Rest-Client-Chrome-Extension.

[47] SoapBox. laravel-formatter. Retrieved May 5, 2014 from https://github.com/SoapBox/laravel-formatter, 2014.

[48] Yehuda Katz Steve Klabnik. {json:api}A standard for building APIs in JSON. Retrieved May 11, 2014 from http://jsonapi.org.

[49] The PHP Group. Client URL Library. Retrieved May 5, 2012 from http://php.net/manual/en/book.curl.php, 2012.

[50] Trygve M. H. Reenskaug. MVC XEROX PARC 1978-79. Retrieved May 5, 2012 from http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

[51] David Upton. *CodeIgniter for rapid PHP application development*. Packt Pub, Birmingham, U.K, 2007.

[52] Eric Vlist. *RELAX NG*. O'Reilly, Sebastopol, CA, 2004.

[53] Š. Antalíková and J. Kmec. Príjmacie konanie na vysoké školy na akademickyý rok 2010/2011 v číslach a grafoch. Retrieved May 5, 2014 from http://www.uips.sk/sub/uips.sk/images/PKvs/Statista/r2010pk1.pdf, 2011.

Resumé

Naša fakulta sa novým študentom, ale aj návštevníkom, môže zdať ako bludisko. Častokrát nemajú inú možnosť, ako sa spoľahnúť na znalosti okoloidúcich študentov vyšších ročníkov. Preto sme sa spolu s kolegom Rudolfom Halásom rozhodli vytvoriť interaktívny plán budov fakulty. Tento plán okrem základného rozloženia učební a kancelárií obsahuje aj informácie o zamestnancoch fakulty, čo uľahčí kontakt medzi pedagógom a žiakom. Ďalšími zobrazovanými informáciami sú rozvrh a obsadenie tried. Tieto požiadavky som pretransformoval do návrhu databázy, ktorá tieto informácie obsahuje a API, ktoré komunikuje s vizuálnou časťou a prípadnými ďalšími aplikáciami.

Vytvorenie a úpravy databázy ako napríklad premenovanie alebo zmena dátového typu stĺpca *rooms.id* z celého čísla na textový reťazec prebieha veľmi jednoducho pomocou migrácií [39]. Je to jedna z výhod moderného a elegantného frameworku Laravel [38]. Tabuľka *rooms* je denormalizovaná, nakoľko sa predpokladá istá znalosť budov a poschodí a zmeny v týchto citlivých častiach tabuľky sú nepredpokladané. Databáza bola navrhnutá tak, aby umožňovala viacjazyčnosť API a tým aj vizuálnej nadstavby. Výsledná databáza je v prílohe A.1.

Typ internetového média pre výmenu dát API sleduje použitím HTTP hlavičiek *Accept* a*Content-type*. Implementované sú dve možnosti XML [7] a JSON [6]. Základnú štruktúru odpovede 4.1 možno rozdeliť na tri časti, kde *meta* poskytuje HATEOS informácie o stránkovaní, *data* je rad vrátených objektov a *embeds* poskytuje zoznam objektov, ktoré sa dajú vložiť ako príbuzné.

API poskytuje štyri základné vstupné miesta:

ľudia URI `/people/[id]` podkapitola 4.3.4

miestnosti URI `/rooms/[id]` podkapitola 4.3.3

udalosti URI `/events/[id]` podkapitola 4.3.2

predmety URI `/courses/[id]` podkapitola 4.3.5

Verzia 1 v slovenskom preklade je prístupná na adrese http://bc.durina.cc/api/v1.

Metódy POST, PUT, DELETE vyžadujú HTTP Basic autentifikáciu [20]. Za účelom spravovania používateľov bola vytvorená administračná časť (podkapitola 4.4). Je to implementácia responzívnej Bootstrap [2] témy Bootswatch Flatly [40] (obrázok 4.1). Po vytvorení nového užívateľa je mu odoslaný email s odkazom, kde si môže zmeniť heslo. Administrátor môže využívať svoje heslo aj na prístup do API, avšak je potrebné zvážiť bezpečnosť takéhoto kroku vzhľadom na HTTP Basic autorizáciu. V časti História databázy (obrázok 4.3) je možné prehliadať zmenené záznamy.

Na spracovanie dostupných dát a ich následné vloženie do API som vytvoril niekoľko knižníc v jazyku Python 2.7. Trieda *PeopleCreator* spracováva dáta z dvoch zdrojov, a to zoznamu zamestnancov FCHPT a následne ich vCard. Opis potrebných parametrov a najdôležitejších metód je v podkapitole 4.5.1. Trieda *CoursesCreator* spracováva súbory z verejného katalógu predmetov pre daný rok a semester (podkapitola 4.5.2). Trieda *TimetableCreator* spracováva zdrojové súbory z tvorby rozvrhov, kde najprv preloží ID zdrojových súborov na id z API a následne vytvorí potrebné udalosti.

Vysoké percento zamestnancov nemá v AIS vloženú kanceláriu, preto si do budúcnosti viem predstaviť aplikáciu, ktorá im jednoducho umožní vložiť si túto úpravu do API. Aplikácia, ktorá bude mať v najbližšej dobe význam, je nahlasovanie chybne označených miestností. Ak sa časom ukáže potreba historických záznamov z databázy, bude zaujímavé venovať sa zvýšeniu funkcionality balíčka, ktorý sa k tomuto využíva.

Zdrojový kód pre API a nástroje v jazyku Python je možné nájst v Bitucket repozitári na adrese https://bitbucket.org/hrpd/fchpt_api.

Appendix A

Appendices

A.1 Routes

Table A.1: A simple longtable example

URI
GET|HEAD users
GET|HEAD users/create
POST users
GET|HEAD users/users
GET|HEAD users/users/edit
PUT users/users
PATCH users/users
DELETE users/users
GET|HEAD /
GET|HEAD login
POST login
GET|HEAD logout
GET|HEAD password/reset
POST password/reset
GET|HEAD password/reset/token
POST password/reset/token
GET|HEAD revisionable
GET|HEAD api/v1/people
GET|HEAD api/v1/people/id

Continued on next page

Table A.1 – *Continued from previous page*

URI
GET|HEAD api/v1/relationship/relationshipId/people
GET|HEAD api/v1/relationship/relationshipId/people/id
GET|HEAD api/v1/relationship/relationshipId/emails
GET|HEAD api/v1/relationship/relationshipId/emails/id
GET|HEAD api/v1/relationship/relationshipId/details
GET|HEAD api/v1/relationship/relationshipId/details/id
GET|HEAD api/v1/relationship/relationshipId/phones
GET|HEAD api/v1/relationship/relationshipId/phones/id
GET|HEAD api/v1/rooms
GET|HEAD api/v1/rooms/id
GET|HEAD api/v1/relationship/relationshipId/rooms
GET|HEAD api/v1/relationship/relationshipId/rooms/id
GET|HEAD api/v1/events
GET|HEAD api/v1/events/id
GET|HEAD api/v1/relationship/relationshipId/events
GET|HEAD api/v1/relationship/relationshipId/events/id
GET|HEAD api/v1/relationship/relationshipId/notes
GET|HEAD api/v1/relationship/relationshipId/notes/id
GET|HEAD api/v1/courses
GET|HEAD api/v1/courses/id
GET|HEAD api/v1/relationship/relationshipId/courses
GET|HEAD api/v1/relationship/relationshipId/courses/id
POST api/v1/people
POST api/v1/people/id
POST api/v1/relationship/relationshipId/people
POST api/v1/relationship/relationshipId/people/id
POST api/v1/relationship/relationshipId/emails
POST api/v1/relationship/relationshipId/emails/id
POST api/v1/relationship/relationshipId/details
POST api/v1/relationship/relationshipId/details/id
POST api/v1/relationship/relationshipId/phones
POST api/v1/relationship/relationshipId/phones/id
POST api/v1/rooms
POST api/v1/rooms/id

Continued on next page

Table A.1 – *Continued from previous page*

URI
POST api/v1/relationship/relationshipId/rooms
POST api/v1/relationship/relationshipId/rooms/id
POST api/v1/events
POST api/v1/events/id
POST api/v1/relationship/relationshipId/events
POST api/v1/relationship/relationshipId/events/id
POST api/v1/relationship/relationshipId/notes
POST api/v1/relationship/relationshipId/notes/id
POST api/v1/courses
POST api/v1/courses/id
POST api/v1/relationship/relationshipId/courses
POST api/v1/relationship/relationshipId/courses/id
PUT api/v1/people
PUT api/v1/people/id
PUT api/v1/relationship/relationshipId/people
PUT api/v1/relationship/relationshipId/people/id
PUT api/v1/relationship/relationshipId/emails
PUT api/v1/relationship/relationshipId/emails/id
PUT api/v1/relationship/relationshipId/details
PUT api/v1/relationship/relationshipId/details/id
PUT api/v1/relationship/relationshipId/phones
PUT api/v1/relationship/relationshipId/phones/id
PUT api/v1/rooms
PUT api/v1/rooms/id
PUT api/v1/relationship/relationshipId/rooms
PUT api/v1/relationship/relationshipId/rooms/id
PUT api/v1/events
PUT api/v1/events/id
PUT api/v1/relationship/relationshipId/events
PUT api/v1/relationship/relationshipId/events/id
PUT api/v1/relationship/relationshipId/notes
PUT api/v1/relationship/relationshipId/notes/id
PUT api/v1/courses
PUT api/v1/courses/id

Continued on next page

Table A.1 – *Continued from previous page*

URI
PUT api/v1/relationship/relationshipId/courses
PUT api/v1/relationship/relationshipId/courses/id
DELETE api/v1/people
DELETE api/v1/people/id
DELETE api/v1/relationship/relationshipId/people
DELETE api/v1/relationship/relationshipId/people/id
DELETE api/v1/relationship/relationshipId/emails
DELETE api/v1/relationship/relationshipId/emails/id
DELETE api/v1/relationship/relationshipId/details
DELETE api/v1/relationship/relationshipId/details/id
DELETE api/v1/relationship/relationshipId/phones
DELETE api/v1/relationship/relationshipId/phones/id
DELETE api/v1/rooms
DELETE api/v1/rooms/id
DELETE api/v1/relationship/relationshipId/rooms
DELETE api/v1/relationship/relationshipId/rooms/id
DELETE api/v1/events
DELETE api/v1/events/id
DELETE api/v1/relationship/relationshipId/events
DELETE api/v1/relationship/relationshipId/events/id
DELETE api/v1/relationship/relationshipId/notes
DELETE api/v1/relationship/relationshipId/notes/id
DELETE api/v1/courses
DELETE api/v1/courses/id
DELETE api/v1/relationship/relationshipId/courses
DELETE api/v1/relationship/relationshipId/courses/id
GET|HEAD api/v1/testing
POST api/v1/testing
GET|HEAD|POST|PUT|PATCH|DELETE api/v1
GET|HEAD|POST|PUT|PATCH|DELETE api/v1/all

A.2 Database Entity Relationship Diagram