



**BRNO UNIVERSITY OF TECHNOLOGY**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

**WEB-BASED MANAGEMENT AND LENDING SYSTEM**

WEBOVÝ SYSTÉM PRO SPRÁVU A PŮJČOVÁNÍ ZAŘÍZENÍ

**BACHELOR'S THESIS**

BAKALÁŘSKÁ PRÁCE

**AUTHOR**

AUTOR PRÁCE

**JAKUB VITÁSEK**

**SUPERVISOR**

VEDOUCÍ PRÁCE

**Ing. JAKUB ŠPAŇHEL**

**BRNO 2017**

**Brno University of Technology - Faculty of Information Technology**

Department of Computer Graphics and Multimedia

Academic year 2016/2017

**Bachelor's Thesis Specification**

For: **Vitásek Jakub**  
Branch of study: Information Technology  
Title: **Web-Based Management and Lending System**  
Category: Image Processing

Instructions for project work:

1. Study the main principles of web-based application systems.
2. Analyze requirements of the web-based application for management and lending of mobile and other devices at FIT VUT.
3. Design web-based application based on these requirements.
4. Implement the proposed application and verify its functionality on a suitable sample of data.
5. Evaluate the results of user testing and further possible follow-up of this project.
6. Create a brief poster and video presenting your work.

Basic references:

- According to instructions of supervisor

Requirements for the first semester:

- The first three points of the assignment

Detailed formal specifications can be found at <http://www.fit.vutbr.cz/info/szz/>

The Bachelor's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

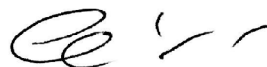
Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Špaňhel Jakub, Ing.**, DCGM FIT BUT

Beginning of work: November 1, 2016

Date of delivery: May 17, 2017

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
Fakulta Informačních technologií  
Ústav počítačové grafiky a multimédií  
612 66 Brno, Božetěchova 2



---

Jan Černocký  
*Associate Professor and Head of Department*

## Abstract

The aim of this thesis is to analyze and implement a web-based equipment checkout system, which provides an organized outlook on system data and equipment transactions history. The final application was implemented as a server-side application using Nette Framework under PHP, MySQL and AJAX. The first half of this thesis investigates existing solutions, describes the implementation phases and looks at design patterns present in the code. The second half of the paper examines the testing, mentions all used add-on components and their purpose, and finally assesses the usability of the implemented system. The result of this paper is an implemented equipment checkout system giving its users an effective way to manage item lending.

## Abstrakt

Cílem této práce je analýza a implementace webového systému pro správu a půjčování zařízení, které uživatelům zprostředkuje přehled dat v systému a historii transakcí zařízení. Výsledná aplikace byla implementována v PHP s využitím Nette Frameworku, MySQL a AJAXu. V první polovině této práce je probrána analýza existujících platforem, popis jednotlivých fází implementace a seznam využitých návrhových vzorů. Druhá polovina pojednává o testování, využitých komponentách a jejich opodstatnění, a nakonec sumarizuje využitelnost implementovaného systému. Výsledkem této práce je implementace systému pro správu a půjčování zařízení, který uživatelům umožňuje vést efektivní evidenci půjčování položek.

## Keywords

Web application, Management system, Lending system, Equipment checkout system, Nette

## Klíčová slova

Webová aplikace, evidenční systém, půjčování zařízení, Nette

## Reference

VITÁSEK, Jakub. *Web-Based Management and Lending System*. Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Špaňhel Jakub.

# Web-Based Management and Lending System

## Declaration

I hereby declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Jakub Špaňhel. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....  
Jakub Vitásek  
May 17, 2017



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Thesis Structure . . . . .	5
<b>2</b>	<b>Analysis</b>	<b>6</b>
2.1	Target Group . . . . .	6
2.2	Possible Solutions . . . . .	7
<b>3</b>	<b>Application Design</b>	<b>8</b>
3.1	Database Structure . . . . .	8
3.2	Use Case . . . . .	9
3.3	Server-side Approach . . . . .	10
3.4	Framework . . . . .	11
<b>4</b>	<b>Implementation</b>	<b>12</b>
4.1	Entities . . . . .	12
4.2	Presenters . . . . .	13
4.3	Content Management . . . . .	15
4.4	User Authentication . . . . .	16
4.5	User Interface (UI) . . . . .	17
4.6	Localization . . . . .	19
4.7	Notifications . . . . .	19
4.8	Routing . . . . .	20
4.9	Application Programming Interface (API) . . . . .	21
<b>5</b>	<b>Design Patterns</b>	<b>22</b>
5.1	Model-View-Controller (MVC) . . . . .	22
5.2	Abstract Class . . . . .	22
5.3	Access Control List (ACL) . . . . .	22
<b>6</b>	<b>Tools</b>	<b>23</b>
6.1	HTML template . . . . .	23
6.2	Nette Framework . . . . .	23
6.3	Database Abstraction Layer (DBAL) . . . . .	23
6.4	Components . . . . .	24
<b>7</b>	<b>Testing</b>	<b>25</b>
7.1	Presenter Testing . . . . .	25
7.2	Usability Testing . . . . .	25

<b>8</b>	<b>Deployment</b>	<b>26</b>
8.1	Statistics . . . . .	26
8.2	Benchmarking . . . . .	27
8.3	User Feedback . . . . .	27
<b>9</b>	<b>Extensions</b>	<b>29</b>
9.1	Calendar . . . . .	29
9.2	Updates . . . . .	30
<b>10</b>	<b>Conclusion</b>	<b>31</b>
	<b>Bibliography</b>	<b>32</b>
	<b>Appendices</b>	<b>33</b>
<b>A</b>	<b>Hotjar Heat Map</b>	<b>35</b>
<b>B</b>	<b>Poster</b>	<b>36</b>
<b>C</b>	<b>Entity–Relationship Diagram</b>	<b>37</b>
<b>D</b>	<b>API</b>	<b>38</b>
<b>E</b>	<b>CRON</b>	<b>39</b>

# List of figures

2.1	Google Trends statistical comparison of PHP, Javascript, jQuery and Angular hints that server-side languages trend more as search keywords. . . . .	7
3.1	Use Case Diagram presenting different user roles and capabilities with 3 actors and inheritance from User to Administrator. It also shows the scheduled CRON job. . . . .	10
4.1	Class Diagram showing the hierarchy of the class models and interfaces that some of the classes implement. . . . .	12
4.2	Data Grid as implemented in the application layout (item listing), showing two items and their data with administrator rights. . . . .	15
4.3	The layout of the application lock screen, the landing page of any unauthenticated user and the logout redirect destination. . . . .	16
4.4	The drag and drop process in a board, the item is being dragged from one user to another. . . . .	17
4.5	The search presenter view when searching for a user name. Sorting of search categories by result count is visible. . . . .	18
4.6	User listing with an active filter for the letter <i>J</i> . . . . .	18
6.1	Dibi Fluent with the use of placeholders and class to eliminate the <b>FROM</b> command . . . . .	24
8.1	Which parts of the application did you use? . . . . .	28
8.2	Functionality and layout satisfaction. . . . .	28
8.3	E-mail digests usability and application requirements inquiry. . . . .	28
A.1	This dashboard heat map shows heavy use of the left sidebar and quick-access board listing in the content section. . . . .	35
B.1	A poster showing mock-ups of the application and the respective implementation phases. . . . .	36
C.1	Entity–Relationship Diagram of the application database structure shows the relation between tables and junction tables. . . . .	37
D.1	API routes, separated into Crud routes and Resource GET routes, which serve as helpers for fast querying. . . . .	38
E.1	The CRON settings dashboard showing the CRON job will be executed every day of the month at 6 AM. . . . .	39

# List of Tables

- 2.1 Existing solutions with marked key software requirements which are met. . . 6
- 3.1 The tables used in the application with junction tables not displayed. . . . 8
- 3.2 Simplified board table violating the second normal form. . . . . 9
- 3.3 Simplified board table with a foreign key, meeting the second normal form. 9
  
- 8.1 Page load times showing the difference between different pages being rendered before and after optimizing the application with cache and image thumbnails. . . . . 27

# Chapter 1

## Introduction

Management and lending systems are a very specialized subset of booking systems. Some of the conventional categories of booking systems are reserving a flight, dinner or a spinning lesson. This type of systems is also the most widespread one, owing to its global commercial use.

However, the aim of this thesis is to create an unconventional booking system which enables users to borrow items while storing transaction data and handling e-mail notifications. Across the Internet, this type of system is also known as *equipment checkout* system. Since it is highly data-driven, it is beneficial to implement it as a web application, utilizing the accessibility, cross-platform compatibility and usability that web applications provide.

Within the framework of this criteria, the result of this thesis is an implemented device management system which makes organizing items and keeping track of transactions an effectively handled task.

### 1.1 Thesis Structure

This paper is organized as follows: the following chapter (2) gives a brief overview of current solutions and their usability, while analyzing the target group and presenting the possible solutions. In Chapter 3, the application design is presented with the help of a use case diagram and an entity-relationship diagram, while mentioning the chosen approach to implementing the application.

Chapter 4 examines the implementation phases in detail, going over system's entities, content management, user authentication, notification handling and the application programming interface. In Chapter 5, there is more on design patterns and their use across the application. The tools used during the implementation are mentioned in Chapter 6, with the emphasis on the framework and its components. Chapter 7 presents testing methods employed before launching the application and also outlines presenter testing, specific for the MVC structure.

Chapter 8 investigates the actual deployment of the application, including benchmark results and server environment specifications. The penultimate chapter (9) looks at possible extensions of the deployed system, with chapter 10 being the conclusion.

# Chapter 2

## Analysis

To make an educated decision on how to approach the implementation, there had to be a research for an existing application best suiting the software requirements. The investigation into non-commercial platforms found following projects:

**Trello** [7] is a project management application featuring item transactions and history.

**phpcheckout** [8] is an open-source project offering items, item history and fines.

**phpLabMan** [5] is an open-source project featuring items, groups (classes) and a calendar.

**Merci** [3] is a Drupal project featuring interchangeable items and groups.

Table 2.1: Existing solutions with marked key software requirements which are met.

	Items	Boards	Users	Teams
<b>Trello</b>	✓	✓	✓	✗
<b>phpcheckout</b>	✓	✗	✓	✗
<b>phpLabMan</b>	✓	✗	✓	✓
<b>Merci</b>	✓	✗	✓	✓

As visible from Table 2.1, there are some checkout and management systems satisfying different parts of the software requirements. However, the majority of these projects are not maintained anymore and are usually implemented in an old version of PHP without the use of design patterns and scalable code.

### 2.1 Target Group

An important part of analyzing the application is recognizing the end users and their needs. In this case, the focus group is a technical university (specifically the Department of Computer Graphics and Multimedia at FIT BUT), where several items need to be distributed across different teams while keeping their current position tracked.

In general, the target group is very skilled in using various website interfaces and has extensive knowledge of web technologies. The group also works in an academic environment, thus preferring a pragmatic, functional and a rather formal design. Their primary need for this application is an organized outlook on items. A Google document or a shared Excel spreadsheet is too low-level, since the system has to provide transfer history, team membership and e-mail notifications.

## 2.2 Possible Solutions

There are multiple ways to achieve satisfying results, utilizing different technologies and design patterns. Overall, these approaches can be divided into a client-side and a server-side approach. Both are described in the following subsections.

### 2.2.1 Client-side approach

It could be argued that the client-side approach is the more current one, since client-side frameworks like React and Angular are gaining massive popularity among developers worldwide<sup>1</sup>. The main benefit of implementing the application as a client-side system is fast rendering after the initial load. Only specific portions of the document object model (DOM)<sup>2</sup> are redrawn with dynamic data retrieved from the server, thus lowering the browser's workload when downloading the whole page again. Thanks to the high rendering speed, this approach is ideal for a highly interactive system (e.g., games). The cons of the client-side technologies are a higher initial loading time and usually the need for external libraries.

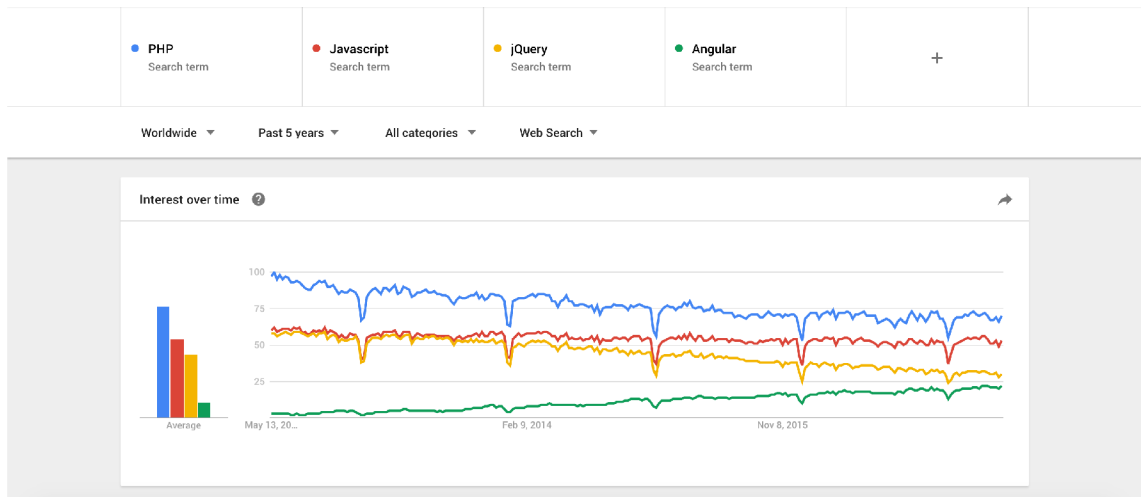


Figure 2.1: Google Trends statistical comparison of PHP, Javascript, jQuery and Angular hints that server-side languages trend more as search keywords.

### 2.2.2 Server-side approach

This approach is the more common and traditional one, as shown in Figure 2.1. Although the client-side technologies popularity has recently been rising, server-side is still considered the mature and robust approach to web applications. PHP is a widely used<sup>3</sup> server-side language, and its frameworks rank high in popularity<sup>4</sup>.

The benefits of using the server-side approach are primarily accessibility and security. Since the communication with the client is handled via HTTP responses, usually bearing the HTML of the requested page, the business logic is invisible to the client, making it more difficult to reverse-engineer the application or spoof the input based on the visible code.

<sup>1</sup>Google Trends <https://trends.google.com/trends/explore?cat=31&q=React,Angular&hl=en>

<sup>2</sup>Document Object Model <https://www.w3.org/DOM/>

<sup>3</sup>PHP usage <http://php.net/usage.php>

<sup>4</sup>PHP frameworks showcase <https://github.com/showcases/web-application-frameworks>

# Chapter 3

## Application Design

Before writing any code, it is beneficial to create an Entity-Relationship diagram and a Use Case diagram to streamline the implementation process. These diagrams will be shown and described in the next sections, along with the structural design pattern used in this system.

### 3.1 Database Structure

In the beginning of the development process, it is feasible to proceed with database architecture after the initial analysis. This will help with creating a better outlook on the general application structure. MySQL<sup>1</sup> is the database management system (DBMS) of choice for this system, since it is an open source relational database model and features the InnoDB storage engine, which can be used to check for existence of foreign keys.

The principal advantage of the final data structure are *associative entities*, implemented by junction tables, since they allow for a many-to-many relationship between entities. As Lloyd [9] shows, an associative entity will contain identifier attributes for each of the entities that it associates. The Entity-Relationship diagram in Figure C.1 on page 37 shows the database structure.

#### 3.1.1 Tables

Table 3.1 lists all standalone tables used in the system and their specification.

Table 3.1: The tables used in the application with junction tables not displayed.

Table	Specfication
board	Boards for devices and users.
item	The devices owned by different users.
team	Teams owning different boards and having different users.
user	Registered users.
log	The log of changes in the system.

---

<sup>1</sup>MySQL <https://www.mysql.com/>



### 3.1.2 Database Normalization

To reduce data redundancy, all normal forms were met while designing the database. As Kent shows [4], the normalization rules are designed to prevent update anomalies and data inconsistencies.

#### First Normal Form (1NF)

To meet the first normal form, all records in the database have to be atomic – in other words, inseparable. An example of atomicity can be shown on the user name in the `user` table, where user’s first name and surname are both stored individually (including the user’s degree).

#### Second Normal Form (2NF)

For the database to reach the second normal form, there can be no functional dependency on any candidate keys. The board table could be designed as in Table 3.2 and thus violating the 2NF. Instead, the table’s final design solves the problem by utilizing a foreign key, as can be seen in Table 3.3.

Table 3.2: Simplified board table violating the second normal form.

ID	title	owner_name	owner_email
1	Mobile devices	John Doe	john@doe.com

Table 3.3: Simplified board table with a foreign key, meeting the second normal form.

ID	title	owner_id
1	Mobile devices	5

#### Third Normal Form (3NF)

There can be no transitive functional dependency to meet the third normal form. The table `item` would have transitive dependency if it depended on any non-prime attributes (e.g., if the item’s board name depended on the board ID).

## 3.2 Use Case

The key attribute of any back-end based system are different user roles and capabilities. There are two roles in the system – `administrator` and `guest`.

A guest is allowed to create teams, boards and items. His view on boards and items is restricted to only his items and boards owned by either one of the user’s teams or by the user himself.

An administrator inherits all permissions from a guest, and extends them to creating users and accessing a minimalistic content management section. The administrator is also allowed to view all items, users, boards and teams with browse, read, edit, add and delete (henceforth BREAD) rights.

All of these constraints are best shown<sup>2</sup> in the Use Case Diagram in Figure 3.1.

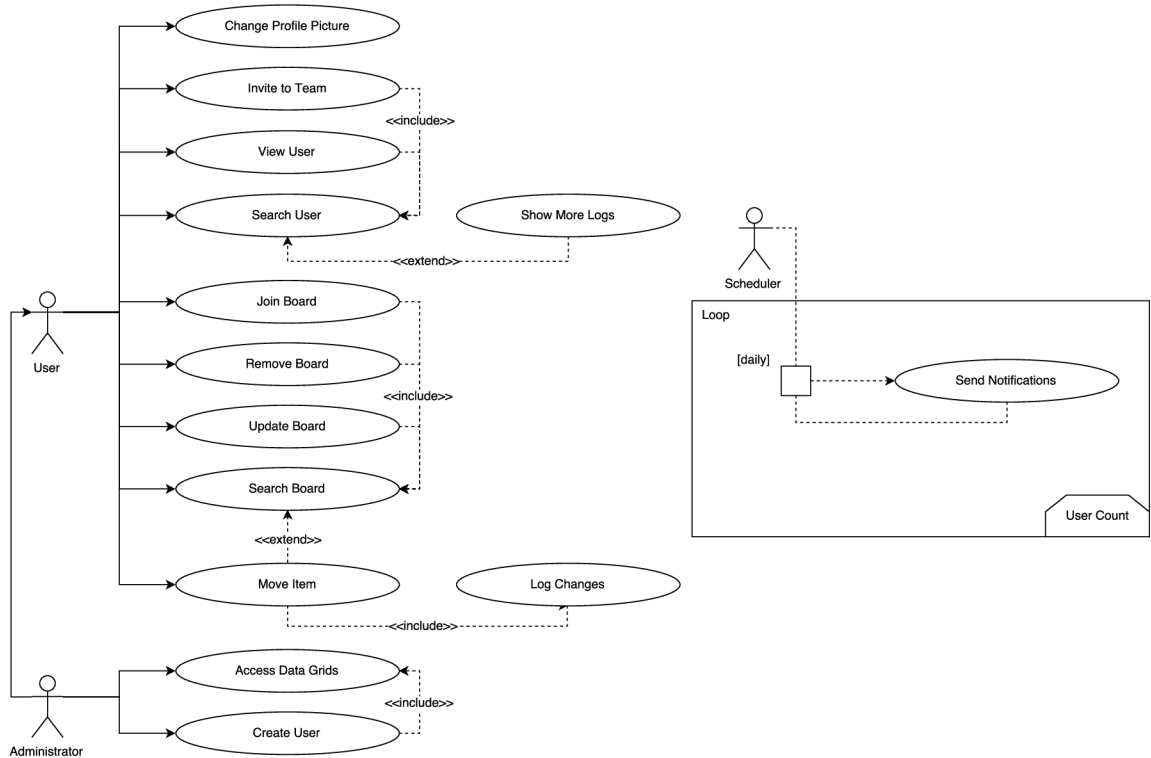


Figure 3.1: Use Case Diagram presenting different user roles and capabilities with 3 actors and inheritance from User to Administrator. It also shows the scheduled CRON job.

### 3.3 Server-side Approach

As it has been established in Chapter 2, there are two distinct approaches to implementing the application. In this case, the server-side approach with the use of PHP was chosen. The list below describes the rationale for this decision.

- Client-side technologies provide dynamic interfaces and a more fluent user experience, which is only needed marginally for item transactions and modal boxes.
- The application has to be accessible even from older versions of browsers, which can have JavaScript disabled or even miss it completely.
- The system is highly dependent on permanent data, which is fetched from the database. Since the database runs on the server side, the application is closer to it implementing the server-side approach.
- Many parts of the application can be cached and stored in a temporary directory to minimize the time-expensive queries to the database. Server-side applications provide more control over caching these requests.

<sup>2</sup>Free on-line diagram software <https://www.draw.io/>

The solution to the need of dynamic elements in a server-side application is AJAX, which allows the programmer to asynchronously execute PHP and receive parts of the website to redraw.

## 3.4 Framework

To avoid implementing a custom tool set and devote more time to meeting the specified software requirements, it is beneficial to use a framework. PHP frameworks offer a templating system, protection against known vulnerabilities and usually an effective database abstraction layer. For these reasons, Nette was chosen as the framework for this application.

### 3.4.1 Push-based MVC

Nette has a strong presence of the MVC design pattern, which is used to separate the system's logic and data presentation to make the application modular, scalable and organized. Owing to this segregation of responsibilities, each component can be tested separately and reused easily.

### 3.4.2 Diagnostic tools

A big part of the actual implementation is debugging and diagnostics. Nette offers a separate package Tracy, which generates detailed error and exception logs for both development and production environments. It also features an eloquent variable dumper with customizable maximal length and depth.

Tracy can be configured to automatically send e-mails when an error occurs. It also provides the option to switch itself to the strict mode. This means exceptions and notices are sent as well. Tracy also remembers it already sent an e-mail regarding a specific error and logs every error only once, so that the attacker cannot spam the website with server error (500) logs. Also, when a server error occurs and the application is in production mode, a custom user-friendly notice is shown to the user.

### 3.4.3 Forms

Forms are the only way for users of this application to alter the data stored in the database. Nette has its own Forms<sup>3</sup> class, which allows for an object-oriented composition and rendering via a macro `{control formName}` or via manual rendering in the template using `n:name` attributes for inputs and the form. The forms in templates are autowired, so that even with manual rendering, the form is passed through the component usually created in the presenter – unless the programmer is utilizing the factory design pattern<sup>4</sup>.

Nette Forms offer a wide range of validation and input types, which handle the rendering of the correct HTML element and also its rules. To process the form, a submit action with the process method callback is commonly required. Mostly, it is the submit button. The form is event driven and processes the callback of the event `onSuccess[]`. Nette Forms also have cross-site request forgery (CSRF) protection and handle all undesirable characters in the user input data.

---

<sup>3</sup>PHP framework Nette <https://doc.nette.org/en/2.4/forms>

<sup>4</sup>Factory Design Pattern <http://www.oodeesign.com/factory-pattern.html>

# Chapter 4

## Implementation

The actual implementation of entities and data structures drafted in the last chapter composes of multiple phases. All of these phases are addressed in following sections.

### 4.1 Entities

With respect to the strong object-oriented design of Nette Framework<sup>1</sup>, which was used to implement this application, all database tables have a model counterpart. As Deacon shows [1]: in object-oriented terms, this will consist of the set of classes which model and support the underlying problem, and which therefore will tend to be stable and as long-lived as the problem itself. Models then represent database tables and implement reusable methods used throughout the application.

In this case, each entity model implements the `IEntity` interface, which enforces that each model is capable of create, read, update and delete (hereafter CRUD) actions. The read method takes an optional parameter – the record identifier. If no ID is passed, the method fetches all records. The class hierarchy of the application models can be seen in Figure 4.1.

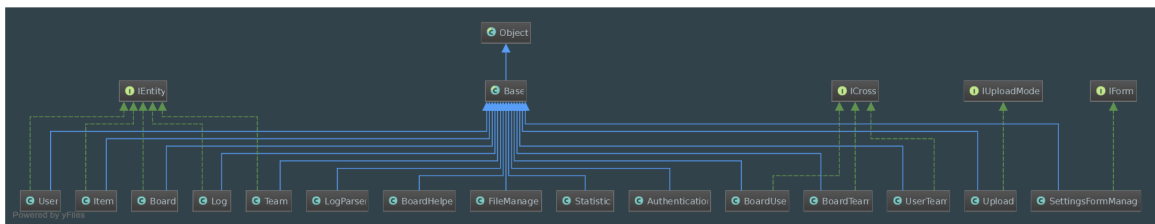


Figure 4.1: Class Diagram showing the hierarchy of the class models and interfaces that some of the classes implement.

#### 4.1.1 User Entity

The class `Entity\User` models the users table, and apart from the interface methods, it implements functions for password reset and profile picture manipulation. Since password reset needs to generate a random hash for both the confirmation e-mail and the new pass-

<sup>1</sup>PHP framework Nette <https://nette.org/en/>

word (more in section 3.3.2), the class `Random`<sup>2</sup> is used in these methods. The entity also handles changing states of flags and cascades over dependent rows on delete.

Furthermore, user's full name consists of four separate columns. Optional columns `degree_before` and `degree_after` are used to enable storing the entire user name without violating the first normal form of relational databases (more in section 3.1.2). Throughout the application, the custom filter `getFullName` is used to compose the full user name.

#### 4.1.2 Board Entity

The methods in the Board entity are predominantly fetching algorithms, which take an ID of a user or a team and return respective boards. Moreover, board model implements one of the most important methods – `changeItemLocation`. This function is called upon completing the AJAX request of a drag and drop transaction, with the help of a presenter signal.

#### 4.1.3 Item Entity

The item keeps its current borrower and its original owner, so that the ownership is clear immediately.

The item entity serves mainly for fetching items for boards or for a specific user. It also uses the `FileSystem`<sup>3</sup> class to manage removal of the item's picture.

#### 4.1.4 Team Entity

As the class name indicates, this entity is working with multiple users being grouped into zero or more teams – utilizing several junction tables to fetch membership information.

#### 4.1.5 Log Entity

The logger in this application is of a fairly generalized structure to make it reusable for other log events apart from an item transaction. There is a helper ID to aid in linking an entry to more than one different entity – e.g., storing the identifier of a created board while maintaining the creator's ID.

## 4.2 Presenters

The application itself uses models listed above selectively, by using dependency injection (DI container) in each presenter depending on the needs of the view. The final data is then handed over to the template via `$this->template->variable_name` from the respective presenter action.

### BasePresenter

This presenter is abstract, since it is only inherited by other non-abstract presenters. The benefit of using inheritance in presenters is having variables which are needed by more (or all) views only in the abstract presenter. However, models cannot be injected through the constructor, since it is never called. The solution is to call `final public function`

---

<sup>2</sup>Nette\Utils\Random <https://api.nette.org/2.4/Nette.Utils.Random.html>

<sup>3</sup>Nette\Utils\FileSystem <https://api.nette.org/2.4/Nette.Utils.FileSystem.html>

`inject` and pull models from there. Nette's autowiring automatically takes care of injecting the dependencies. `BasePresenter` also enforces global rules. In this system, all presenters (with the exception of `LoginPresenter`) cannot be accessed without an acquired user identity (more in section 4.4).

Nette allows the programmer to define custom Latte filters (formerly called helpers). These can be defined in the method `beforeRender` in each presenter and they propagate through inheritance as well. The two custom filters registered in this application are very compact, with one composing the full user name and the second one getting a language shorthand and transforming it into the full name of the language.

## BoardPresenter

There are two render methods in this presenter. One of them is `renderList`, which takes no arguments and assembles the variables needed by the list view. An abridged content of the `renderList` method, fetching boards from the model layer, adding an offset containing the board's users and injecting it into the template can be seen in the code below.

```
$boards = $this->boardModel->get()->fetchAll();
/** @var Row $b */
foreach($boards as $b) {
    $users = $this->boardModel->getAllUsers($b->id);
    $b->offsetSet('uzivatele_ids', $users);
}
$this->template->boards = $boards;
```

The second render method, `renderDefault($url)` takes one parameter – the board URL. The method searches for a board with the specified URL in the database. If it finds it, it proceeds with composing the template variables. However, if the database returns `false`, a new `BadRequestException` is thrown, causing Nette to redirect to `ErrorPresenter`, as visible in the code below. The view itself is the actual management canvas, where users can move devices from one column to another, with a column representing a user and his items pertaining to the respective board.

```
if($this->template->board) {
    // variables for the template
} else throw new BadRequestException();
```

This presenter also has the highest number of *handle methods*, which are functions called upon receiving a specific signal. For example, when the URL `/?do=removeBoard&id=1` is accessed, the signal calls the method `handleRemoveBoard($id)`. The signals can be called asynchronously with the help of Nette Ajax and snippet redrawing.

## HomepagePresenter

This presenter is the landing page of a successful log in. It lists all the user boards and also all the boards the user has access to through team membership. Owned boards can be edited from this view. Homepage also has a statistical module in the header, where the number of each entity is listed.

## ItemPresenter

This presenter is responsible for listing items in the system and also their update and delete actions. There is no item detail page, since the item data is displayed in a modal box shown upon clicking the item link.



## TeamPresenter

Team views offer both listing and detail, while the listing methods also handles creating a profile picture object. The object contains randomly ordered profile pictures of users in that team, which appear in the header of each team.

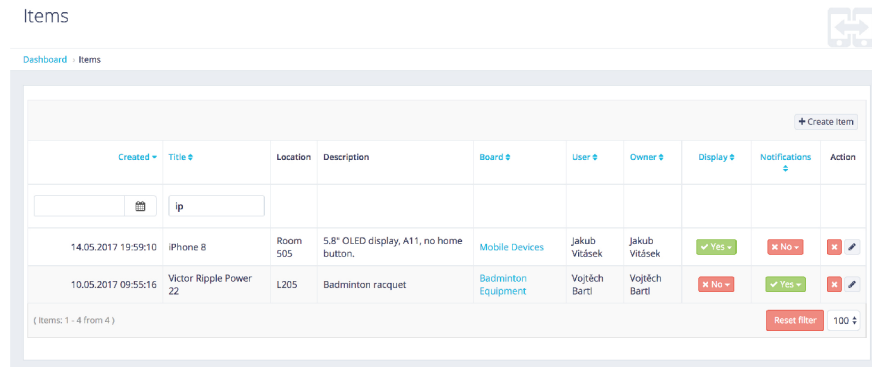
## UserPresenter

Users have a listing page, where they can be filtered by the first letter of their name. The filtering is strictly in JavaScript without the use of Nette Ajax. There is also a sidebar that enables the user to edit his information, giving an alternative to the settings modal box shown on clicking the cogwheel icon in the site's header.

User detail contains the user's complete information, including his e-mail, phone number, number of owned items, number of borrowed items and team membership. There is also a list of all transactions done by the user, where a load more button shows four more logs on click.

## 4.3 Content Management

Owing to the system back-end characteristics, it was not necessary to create a separate module solely for content management purposes – all the administration links reside within a subsection of the left sidebar. Granted the user is in the **administrator** role, the subsection is visible and each of the links lead to a *data grid* page as shown in Figure 4.2.



Created	Title	Location	Description	Board	User	Owner	Display	Notifications	Action
14.05.2017 19:59:10	iPhone 8	Room 505	5.8" OLED display, A11, no home button.	Mobile Devices	Jakub Vitásek	Jakub Vitásek	✓ Yes	✗ No	✗
10.05.2017 09:55:16	Victor Ripple Power 22	L205	Badminton racquet	Badminton Equipment	Vojtěch Bartl	Vojtěch Bartl	✗ No	✓ Yes	✗

Figure 4.2: Data Grid as implemented in the application layout (item listing), showing two items and their data with administrator rights.

### 4.3.1 Data Grid

A data grid is a dynamic table which lists data fetched from a *data source* in a well arranged way. Data grids come with many built-in features like sorting, filtering, data export and in-line data management. This application uses a data grid which supports AJAX invalidation of Latte snippets<sup>4</sup>. Snippets are parts of the template which can be asynchronously redrawn from the presenter, using the `redrawControl()` method.

Utilizing in-line create and update methods, it is possible to promptly and intuitively manage all data in the system. The user is able to fill in respective columns directly inside

<sup>4</sup>AJAX & snippets <https://doc.nette.org/en/2.4/ajax>

of the table, making it simpler to grasp the user interface. Deletion, sorting and filtering is handled by AJAX as well, which makes for a smooth user experience while maintaining features like input validation and notifications.

## 4.4 User Authentication

There is only one way to gain access to the system – a user has to complete a standard sign up form where he specifies a desired password. He then uses his e-mail and the password specified to log in to the system. This process begins on the lock screen seen in Figure 4.3.

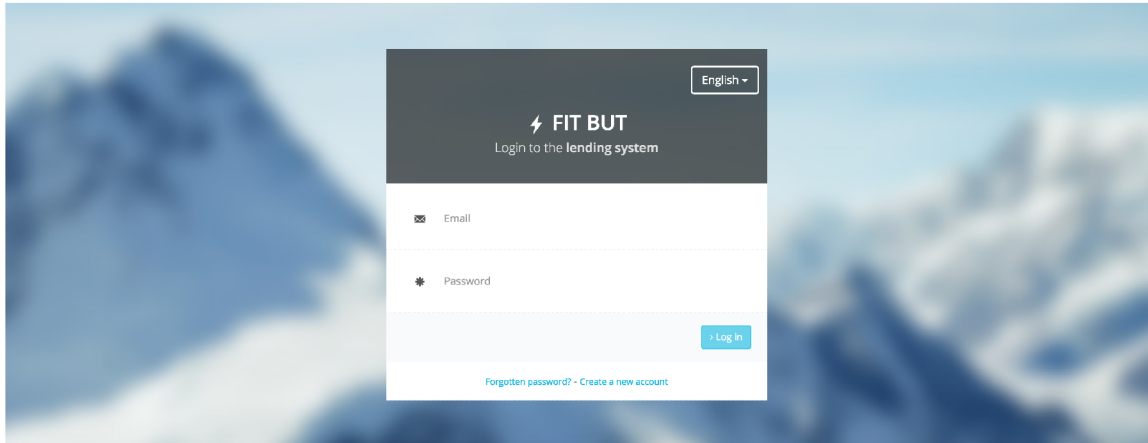


Figure 4.3: The layout of the application lock screen, the landing page of any unauthenticated user and the logout redirect destination.

The lock screen consists of three sections which are displayed exclusively – a *log in* form, a *sign up* form and a form used to retrieve a forgotten password. Switching views is handled by jQuery<sup>5</sup>, which uses the form’s HTML identifier to toggle the visibility of each form.

### 4.4.1 Obtaining User Identity

During a log in attempt, the authentication model compares user input against the database. If a row is retrieved, the model returns the user identity<sup>6</sup>. To optimize database queries, the identity is expanded by the user’s profile picture URL and the appropriate greeting, which is the vocative case of the user’s first name.

To determine the correct vocative case, a very lean API wrapper called Hi<sup>7</sup> was used. If the name is not recognized by the API, user identity stores the first name in its original form. The wrapper caches all loaded greetings.

### 4.4.2 Forgotten Password

In case a user forgets his password, a standard procedure was implemented to allow the user to reset the password. The user enters his e-mail, which triggers a mailing function<sup>8</sup>

<sup>5</sup>jQuery: a fast, small, and feature-rich JavaScript library <https://jquery.com/>

<sup>6</sup>Nette\Security\Identity <https://api.nette.org/2.4/Nette.Security.Identity.html>

<sup>7</sup>Greeting generator API PHP wrapper <https://github.com/ondrs/Hi>

<sup>8</sup>Nette Mailer <https://doc.nette.org/en/2.4/mailing>



which sends a new randomly generated password to the input e-mail. For the new password to take effect, the user has to click a confirmation link in the received e-mail.

The confirmation link uses a 6 character random hash which further prevents an attacker to reverse-engineer the structure of the confirmation links.

## 4.5 User Interface (UI)

While the user interface is given by the HTML template for the most part, the functionality of filters, item drag and drop, search and much more had to be customized and implemented. Each of these UI elements lead to a better user experience (UX) of the application.

### 4.5.1 Item Transactions

Item manipulation in a board interface is the key feature of this application. Behind this functionality lies a JavaScript library Dragula<sup>9</sup>, which is tied to the element ID of an item. A user then simply drags the item from the source column to another column, as can be seen in Figure 4.4.

Upon finishing an item transfer (*drag and drop* action), a custom presenter signal is called via the Nette Ajax<sup>10</sup> extension directly from the script. This signal handles the location change of the item.

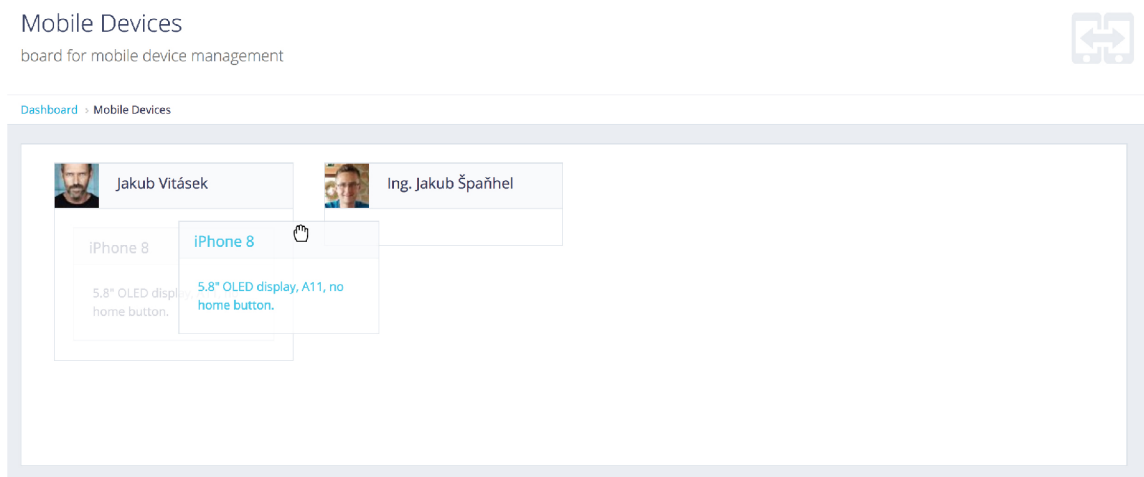


Figure 4.4: The drag and drop process in a board, the item is being dragged from one user to another.

The signal method parses the received JSON structure containing the final distribution of items and creates an ArrayHash<sup>11</sup> variable to be used in the log create method. It then uses snippet invalidation to fetch the latest log data to the sidebar log widget and redraw the data without refreshing the page.

<sup>9</sup>Dragula <https://bevacqua.github.io/dragula/>

<sup>10</sup>Nette Ajax <https://componette.com/vojtech-dobes/nette.ajax.js>

<sup>11</sup>Nette\Utils\ArrayHash <https://api.nette.org/2.4/source-Utils.ArrayHash.php.html#13-100>

## 4.5.2 Search

For the system to be able to scale well, it was important to provide the user with a solid search functionality. The search algorithm has to filter through all manageable entities (items, teams, boards and users), return relevant results and sort by the number of results in each entity.

With this algorithm, the system fetches users upon searching for a given name, and returns an item upon searching for a mobile device. This can be seen in Figure 4.5.

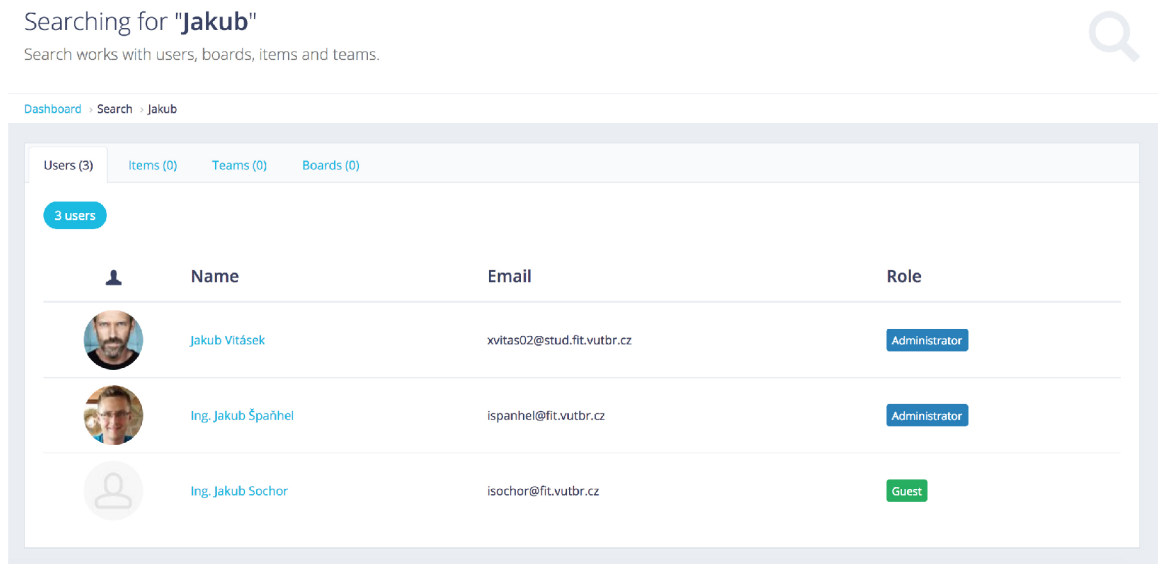


Figure 4.5: The search presenter view when searching for a user name. Sorting of search categories by result count is visible.

## 4.5.3 User listing

With the application's target group being teams of users not necessarily knowing each other, the list of users was made more user-friendly by implementing a name filter. This filter allows users to filter a specific letter which occurs in the given name of the searched user. When no letter is selected, all of the users are shown in a three column layout, as seen in Figure 4.6.

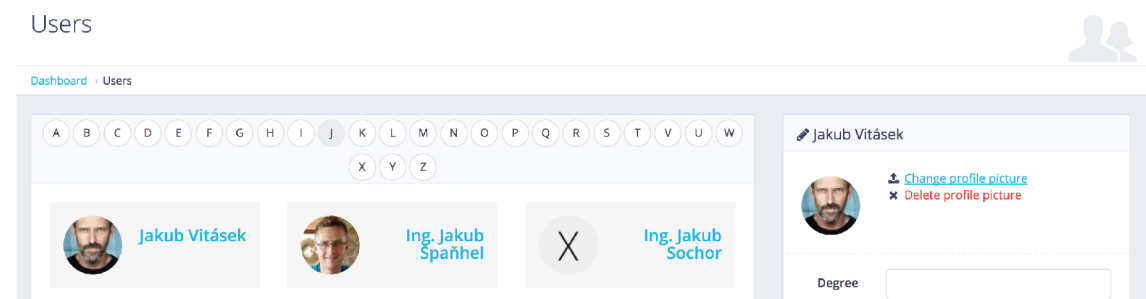


Figure 4.6: User listing with an active filter for the letter *J*

## 4.6 Localization

In order to make the system usable for the widest range of users, a translation system was implemented to offer suitable versions of the user interface data. To make that possible, all strings in presenters and views (including components) had to be replaced by a localization macro<sup>12</sup>. This macro passes the strings through a translation class implementing the `Nette\Localization\ITranslator` interface.

For this project, `Kdyby\Translation`<sup>13</sup> was chosen to provide translation functionality. It directly integrates Symfony's Translation and is easily used throughout Nette templates and presenters. It also automatically integrates with Tracy panel, where a module can be opened to see missing translations and loaded resources used for the localization.

The translation files reside in the directory `lang` inside the application directory. The files are formatted following the NEON syntax<sup>14</sup> also used in Nette configuration files. Its syntax is very similar to YAML, thus allowing for a structured localization file. The filename consists of the translation category and the combination of ISO 639-1 and ISO 3166-2 codes separated by an underscore (e.g., `homepage.cs_CZ.neon`), and then easily addressed in the code as `{_homepage.section.phrase}`. This macro finds the NEON file named `homepage` in the currently selected language, inside of which a category named `section` and a subcategory titled `phrase` is found. The found text is then printed in place of the macro.

## 4.7 Notifications

To keep users informed about manipulation with their items, the system enables a user to define if he wants to receive notifications globally. If the user has active notifications, he can also decide for which items the notifications apply. The granularity of notification options was lowered so that the user can turn off notifications globally while maintaining his selection of item notifications (e.g., a business trip).

If there were transactions with selected items, a CRON job running every day at 6 AM sends out digest to users where all the transactions made since the last notification are listed for each item.

### 4.7.1 E-mail Template

One of the key parts of sending e-mail notifications is composing the HTML of the actual e-mail. Nette uses its own templating engine `Latte`<sup>15</sup>, which is used in every view of the application. It can also be instantiated as a separate object and rendered to string, which is then passed as the e-mail's HTML. This allows for the complete Latte templating functionality in e-mails.

This application uses includes to have separate e-mail header and footer files, which are the same for each e-mail. Each e-mail template then includes the header and footer while having a unique body.

---

<sup>12</sup>Default Latte Macros <https://latte.nette.org/en/macros#toc-localization>

<sup>13</sup>`Kdyby\Translation` <https://github.com/Kdyby/Translation>

<sup>14</sup>`Nette\Neon` <https://ne-on.org/>

<sup>15</sup>`Nette\Latte` <https://latte.nette.org/en/>

Variables are passed to the Latte engine object, containing data for the notification – an associative array, where the key is the item identifier and the value are the logs for that item.

### 4.7.2 Mailer

The class used to compose the e-mail is `Nette\Mail\Message`<sup>16</sup>, which encapsulates all the functionality of the PHP mail function but without the need of low-level HTTP headers specification. The Message object allows to enclose attachments, specify recipients and also to define the subject and the HTML body.

Before instantiating the Message class, it is important to validate the recipient's e-mail address to prevent server errors caused by an uncaught exception thrown by the Message object.

When the Message object is fully defined, it needs to be passed into a class implementing the interface `Nette\Mail\IMailer`<sup>17</sup>. In this application, a custom SMTP connection is provided with the help of `Nette\Mail\SmtplibMailer`<sup>18</sup>. The send method, which finalizes the mailing procedure, needs to be enclosed in a `try-catch` block, since an exception can be thrown if the SMTP server does not respond.

### 4.7.3 CRON

Before setting up the CRON job for sending out notifications, it is important to decide on the time at which the mailing will occur regularly. Since a high percent of work with the system is happening throughout the day (approximately from 6 AM to 6 PM), the notification e-mails are sent each day at 6 AM.

To have a better outlook on how each CRON job ran, the notification method logs the times and amount of e-mails sent for every CRON run. This is more of a debugging step, but important for validating the functionality. The CRON job runs from an external server and accesses a hashed URL dedicated for CRON jobs. The CRON presenter also has an inclusive IP filter, which only allows logged-in users or the IP address of the CRON server to access the presenter's actions.

## 4.8 Routing

Semantic and explicit (also called *pretty*) URLs are usually not a necessity in back-end systems, since the content of the site is not indexed by search engines. However, for effective routing and user-friendly links, it was optimal to create URLs for users, boards and teams. The URL is created using the method `Webalize`<sup>19</sup>, which takes the full title of the record and encodes it to be used in links (web safe characters [a-z0-9-]). This URL is then used in the routes, as can be seen in Figure D.1.

---

<sup>16</sup>`Nette\Mail` <https://github.com/nette/mail>

<sup>17</sup>`Nette\Mail\IMailer` <https://api.nette.org/2.4/Nette.Mail.IMailer.html>

<sup>18</sup>`Nette\Mail\SmtplibMailer` <https://github.com/nette/mail/blob/master/src/Mail/SmtplibMailer.php>

<sup>19</sup>`Nette\Utils\Strings` <https://api.nette.org/2.3/source-Utils.Strings.php.html#207-223>

## 4.9 Application Programming Interface (API)

As a bonus extension to the application assignment, an API was created to aid in the implementation of multi-platform alternatives (e.g., Android, iOS). As Mitchell stated [6], an Application Programming Interface (API) lets computer programmers access the functionality of published software modules and services.

### 4.9.1 CRUD

The API is based around the CRUD model, allowing the programmer to use any methods of the entities. Junction tables provide only the read, create and delete methods, update is not available. As a controller for handling API requests, this application uses the Restful<sup>20</sup> library, which provides an abstract presenter `Resource`. Owing to several methods in this presenter, it is fairly simple to implement networking functions (e.g., sending a response, receiving a request, or returning an error code).

### 4.9.2 Implementation

Each API presenter (i.e., a presenter extending the `ResourcePresenter`) composes an `ArrayHash` variable which is sent to the respective model and method of the specified entity. That means the API calls use the same CRUD methods as the actual application, reducing duplicity and redundant separate code for the API.

The API response was programmed so that the caller would receive a corresponding message on success or error. Since each CRUD method in models is encapsulated into a `try-catch` block, it is fairly simple to detect errors and exceptions.

### 4.9.3 Feedback

The API was tested during the implementation phase. The feedback pointed to creating more helper API routes to allow for faster and simpler querying to get all items pertaining to a specific board or the contents of the log. The test subject also requested that create methods return the ID of the inserted row. This request was processed and it is a part of the final application.

### 4.9.4 Resulting API

As can be seen in Figure D.1, unique routes were created specifically for the API. `CrudRoute` automatically detects the header type (POST/GET/PUT/DELETE) and delegates the request to the appropriate action in the destination presenter. APIs were created for teams, users, boards and items, allowing the programmer to create and delete pairs in junction tables as well. The API actions can be called by accessing specific URLs with one of the aforementioned header types. The complete API documentation is enclosed in appendices on page 38.

---

<sup>20</sup>Drahak\Restful <https://github.com/drahak/Restful>

# Chapter 5

## Design Patterns

Design patterns are a big part of object-oriented programming. The Gang of Four [2] shows that using design patterns makes a programmer an expert designer: *One thing expert designers know not to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again.*

### 5.1 Model-View-Controller (MVC)

With Nette Framework being an MVC framework, it was desirable to leverage this feature and create an MVC valid application. There already is a strong tie between templates (View) and presenters (Controller), however, there is little built-in support for models.

That is why the model directory structure is modified by several namespaces<sup>1</sup> to give models a more organized layout.

### 5.2 Abstract Class

Each base class, either a presenter or a model, is an abstract class in this application. This means the class cannot be instantiated, only inherited from. Based on this concept, every class inheriting the base class receives the same public or protected attributes and methods, which can be overrode if needed.

In models, it is very beneficial to exploit the inheritance of the base model. For one, a database connection can be distributed throughout the model layer by only injecting<sup>2</sup> the dependency once.

### 5.3 Access Control List (ACL)

Since there are multiple user roles present in the application, the best way to solve authorization is via Access Control List (henceforth named ACL). A higher level access control model like attribute-based access control (ABAC) is too complex for the needs of this application. Moreover, Nette has a built-in ACL layer which is satisfactory for authorization in this case. More of user access differences can be seen in the Use Case Diagram (Figure 3.1).

---

<sup>1</sup>Namespaces overview <http://php.net/manual/en/language.namespaces.rationale.php>

<sup>2</sup>Nette DI <https://doc.nette.org/en/2.4/dependency-injection>



# Chapter 6

## Tools

To increase productivity, put more focus on the task at hand and make the implementation phase more effective, several tools were utilized. These tools can be found in sections in this chapter.

### 6.1 HTML template

Since PHP programming is not enough to make the system usable and user friendly, an HTML template<sup>1</sup> was chosen to provide styles and code for dashboard and page layouts. This solves an array of coding subproblems, ranging from mobile usability (responsive layout) to modern and functional design, giving the programmer more time to focus on back-end (PHP) and front-end (JavaScript) functionality.

### 6.2 Nette Framework

This application also uses a Czech-made PHP framework called Nette. This framework is very popular among Czech developers and has a strong community which maintains the momentum of the framework's upkeep and expansion. It comes with many useful built-in packages for debugging, form building, database querying or mailing.

One of the strong points of Nette is vulnerability protection<sup>2</sup>, securing the application against known security flaws like cross-site scripting (XSS) or cross-site request forgery (CSRF).

### 6.3 Database Abstraction Layer (DBAL)

A reservation system of these dimensions does not yet require an object-relational mapping (ORM). That is why a lighter and simpler DBAL called Dibi<sup>3</sup> was chosen to streamline database queries and make them reusable throughout the application.

The Dibi library enables the programmer to use the *fluent* notation, which chains individual commands together without enforcing a specific order of some SQL commands. An example of this notation is shown in Figure 6.1, where a log-in attempt is being handled.

---

<sup>1</sup>ProUI by Pixelcave <https://pixelcave.com/proui>

<sup>2</sup>Vulnerability protection in Nette <https://doc.nette.org/en/2.4/vulnerability-protection>

<sup>3</sup>Dibi: smart database abstraction layer <https://github.com/dg/dibi>

```

users::select('id, name')
    ->where('email = %s', $username)
    ->and('password = SHA1(%s)', $password)
    ->and('active = 1')
    ->fetch();

```

Figure 6.1: Dibi Fluent with the use of placeholders and class to eliminate the FROM command

## 6.4 Components

One of the most useful tools utilized in this application is Composer<sup>4</sup>, which is helpful in managing all extensions (also known as components) by keeping them updated and also fetching all their dependencies. Following are the components used in this application.

**Nette Ajax** An effective utility script<sup>5</sup> enabling AJAX links by giving them the HTML class `ajax`. This functionality was used in refreshing sidebar logs and triggering modal boxes without having to refresh the page.

**jQuery FileUpload** A component<sup>6</sup> used for AJAX handling of image upload. It has rather extensive customization possibilities and it uses an upload model interface, which makes it easy to implement into Nette framework.

**Thumbnail Helper** A component<sup>7</sup> for effective and simple thumbnail generating. It is used throughout the application for profile pictures and item images without stressing the client side with downloading the full sized file.

**Session Panel** This extension<sup>8</sup> is strictly a development tool, which allows the programmer to see each session in a clear debugging environment of the Tracy<sup>9</sup> panel.

**Ublaboo Data Grid** This data grid<sup>10</sup> has previously been mentioned in the content management section. It is used exclusively for BREAD administration, filtering and sorting of database entities.

**Hi** This API wrapper<sup>11</sup> is more of a bonus, providing the vocative case of Czech names to greet the user on the application dashboard.

**Restful** A powerful tool<sup>12</sup> to implement an API intuitively in Nette. More information can be found in the Application Programming Interface section (Section 3.6).

<sup>4</sup>Composer <https://getcomposer.org/>

<sup>5</sup>vojtech-dobes/nette.ajax.js <https://github.com/vojtech-dobes/nette.ajax.js>

<sup>6</sup>jzechy/jquery-fileupload <https://github.com/JZechy/jquery-FileUpload>

<sup>7</sup>kollarovic/thumbnail <https://github.com/Kollarovic/ThumbnailHelper>

<sup>8</sup>kdyby/nette-session-panel <https://github.com/fprochazka/Nette-Session-DebugBar>

<sup>9</sup>nette/tracy <https://github.com/nette/tracy>

<sup>10</sup>ubladoo/datagrid <https://github.com/ubladoo/datagrid>

<sup>11</sup>ondrs/hi <https://github.com/ondrs/Hi>

<sup>12</sup>drahak/restful <https://github.com/drahak/Restful>



# Chapter 7

## Testing

Testing was an important part of both development and deployment of the application. Unit tests streamline the implementation with test-driven development (TDD), while usability tests validate the application's functionality and layout.

### 7.1 Presenter Testing

The MVC design pattern allows for easy testing of separate modules. Testing models would be the task of unit testing, but a more special category of tests is presenter testing. This can be done by the use of Nette Tester<sup>1</sup>, which allows for asserting the equality of two presenter responses. This can be used for detection of errors, when the expected response is of the class `TextResponse` and the presenter responds with `RedirectResponse`.

Also, the DOM can be searched through, looking for specific elements by their class or ID, similarly as in CSS selectors in jQuery. The assertion returns true if the element is present in the DOM. That is beneficial when testing the view module – e.g., if the right components are included and the right Latte templates rendered. All the created presenter tests can be found in the `tests` directory and can be ran by an enclosed BAT script on Windows.

### 7.2 Usability Testing

Before deploying the application into production mode, a 5-day testing period was given to 12 test subjects (who are also potential future users of the system). The goals of this testing were to confirm or disprove the hypothesis that the implemented system solves the problem of managing device lending.

Test subjects were assigned their own accounts and were shown the principles of the application. Each of their sessions was recorded and stored in Hotjar for future analysis, including the heat maps of their behavior in the system. A short questionnaire was also sent out to the tests subjects to be used to assess the usability of the application and can be found on page 28.

---

<sup>1</sup>Nette\Tester <https://tester.nette.org/en/>

# Chapter 8

## Deployment

Before launching the application for production, a small set of test users was selected in order to validate system's functionality and design.

Also, several optimizations were made to accelerate the load time.

### 8.1 Statistics

During the testing period, Google Analytics<sup>1</sup> and Hotjar<sup>2</sup> were utilized as statistical collectors to maximize the aggregated data value.

**Hotjar** The free (Basic) version of Hotjar allows the user to view recordings of every session on the website, including clicks, typing and cursor flow. Recordings aid in detecting and disclosing usability and functionality problems of the application without the need of user generated bug reports. Hotjar also generates heat maps, where the general behavior flow and scrolling or moving patterns of users are visible. The dashboard heat map in appendices (Figure A.1 on page 35) is the result of a 10-day testing period.

**Google Analytics** This analytical tool primarily measures visits, but also records important attributes of visitors. From this data, it can be inferred what the most used operating system is, how high is the user bounce rate or how do users progress through the website (*users flow*).

#### 8.1.1 Results

Hotjar heat maps show that visitors use the left sidebar as the primary navigation point. They also frequently target their own boards, which are placed at the topmost position of the dashboard page content. User profile drop-down situated in the upper right corner of the page also gets clicked fairly often, indicating that users understand the drop-down functionality.

Google Analytics revealed that 90 % of visitors use Windows as their operating system, followed by Macintosh and Linux both at 5 %. Since the application was developed and fine-tuned for Windows, this fact is rather convenient. The same can be said about the browser, which is Google Chrome by 97 %.

---

<sup>1</sup>Google Analytics <https://www.google.com/analytics/>

<sup>2</sup>Hotjar <https://www.hotjar.com/>

## 8.2 Benchmarking

This section shows the render time of specific pages before and after optimization. The primary part of the optimization was image thumbnails, caching and database indexes. The page load times can be seen in the table 8.1.

Table 8.1: Page load times showing the difference between different pages being rendered before and after optimizing the application with cache and image thumbnails.

	<b>Before optimization</b>	<b>After optimization</b>
<b>Team List</b>	313.8 ms	235.4 ms
<b>Dashboard</b>	281.5ms	257.4 ms
<b>Board Detail</b>	421.1 ms	250.3 ms
<b>Create Item</b>	300.0 ms	300.0 ms
<b>Search</b>	354.3 ms	320.6 ms

## 8.3 User Feedback

Test subjects were given a questionnaire where they could report bugs and room for improvement. The questionnaire also served as validation to see if the users are able to work with the application and if the system satisfies their needs.

Listed below are the respective questions and selected answers. Some of the feedback was immediately processed and is part of the final application. The implemented feedback is marked with the **[processed]** flag.

### **What benefits does using the application offer?**

- Easy item management
- Overview of lent items per person/per category
- The option to see available devices, their owner and their current state
- Information about the current location of my items for lending

### **Is there anything you miss in the application?**

- Possibility to send a request for team invite
- **[processed]** Indicators of team membership, preferably differentiated by an icon or color
- **[processed]** Item listing could be sorted from my owned items, my borrowed items and then other items
- **[processed]** Filters for item, board and team listing
- **[processed]** Items on boards could have pictures
- **[processed]** Items on boards could show their owner
- Option to edit items directly on the board

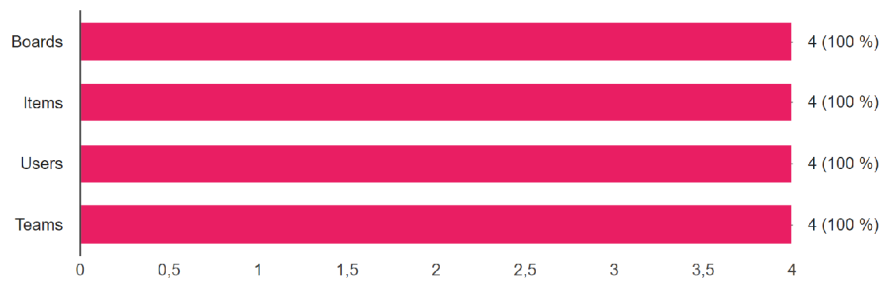
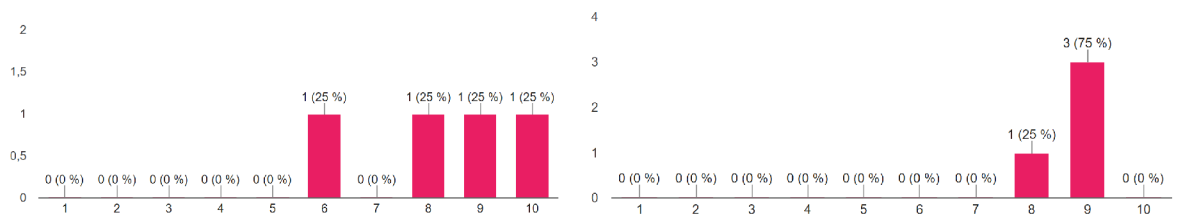


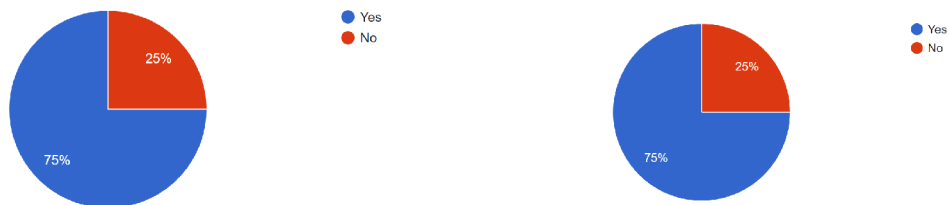
Figure 8.1: Which parts of the application did you use?



(a) On the scale of 1 to 10, say how much you are satisfied with the system's layout.

(b) On the scale of 1 to 10, say how much you are satisfied with the system's functionality.

Figure 8.2: Functionality and layout satisfaction.



(a) Do you find everyday email digests for trans- actions with selected items helpful?

(b) Does the application meet your requirements?

Figure 8.3: E-mail digests usability and application requirements inquiry.

## Chapter 9

# Extensions

Since it is expected that this application would be deployed across several different clients and thus run in many different instances, it is desirable to think about extending the application. Thanks to the strong presence of the MVC pattern in Nette projects, extensions could be implemented as separate modules. These modules could then be activated via the NEON configuration file for each instance of this application, making it more customizable and easily marketable.

**User Chat** A thread based chatting system allowing users to swiftly interact with each other in order to discuss the extent of the borrow.

**Item Gallery** Currently, the item detail only enables the upload of one image which is displayed as the main image (profile image) of the item. This extension would provide an interface to upload multiple images to be shown in an organized manner in the item detail modal box.

**Logger** The logger currently integrated in the application only logs item transactions and board membership changes. An extension of this logger could also save the destination of the item transaction and log the creation/editing of each entity with more specific information.

**Localized Database** The application is currently localized only on the static side, but lacks the ability to store multilingual data for each entity. This would mean creating a languages table and referencing the translation with a foreign key in a language column of the entity translated.

### 9.1 Calendar

To improve the booking functionality and allow users to book items ahead of time, a calendar module could be integrated. Each of the items could act as a single server system, busy only in already booked times. This means the user booking the item for a selected time would be granted exclusive availability of the item for that time, automatically assigning the item to the borrower and making it non-movable on the item's board.

## 9.2 Updates

As each client has different needs and expects customizability, a problem arises with every new module or even a bug fix – different instances of the application cannot be updated collectively, since that would overwrite the custom changes. A satisfactory solution would be to keep the core module (the only module getting batch updates) in a VCS repository and roll out updates through a custom updater model: e.g., PHPAutoUpdate<sup>1</sup>.

The updater model would then access a specific URL with a JSON (or XML) file, which would list an associative array of versions and paths to the update files. The update file would most likely be a ZIP, only containing new or updated directories. The updater model could then install the update into the root directory of each instance of the application, creating new folders and updating existing ones.

To inform the user about the availability of a new version, a CRON could run every day to compare the JSON file versions to the version of the application. If a higher number is detected, the client would be notified and could manually start the update (e.g., by clicking an update button). This process could even become automatic.

---

<sup>1</sup>VisualAppeal\PHP-Auto-Update <https://github.com/VisualAppeal/PHP-Auto-Update/>

## Chapter 10

# Conclusion

In summary, this thesis led to an analysis of the needs and requirements of an equipment checkout system where items can be tracked and borrowed on-line, and also its complete implementation. The web application is currently hosted on <http://bc.jvitasek.cz> and running in the beta mode. The implemented system meets all its software requirements and its usability can be assessed from the received feedback, in which majority of test subjects consider the application usable and indicate their further use of the system (see graphs on page 28).

# Bibliography

- [1] Deacon, J.: *Model-View-Controller (MVC) Architecture*. JOHN DEACON Computer Systems Development, Consulting and Training. 1995: pp. 1–6.
- [2] Gamma, E.; Helm, R.; Johnson, R.; et al.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. 1994. ISBN 978-0201633610.
- [3] Hiatt, B.: *MERCI (Manage Equipment Reservations, Checkout and Inventory)*. Drupal. January 2009. [On-line; visited 28.04.2017].  
Retrieved from: <https://www.drupal.org/project/merci>
- [4] Kent, W.: *A Simple Guide to Five Normal Forms in Relational Database Theory*. *Communications of the ACM* 26(2). 1983: pp. 120–125.
- [5] McMurray, B.; Rejfek, J.; Reynen, K.: *phpLabMan*. Sourceforge. July 2001. [On-line; visited 28.04.2017].  
Retrieved from: <https://sourceforge.net/projects/phplabman/>
- [6] Mitchell, B.: *Network Application Programming Interfaces (APIs)*. Lifewire. October 2016. [On-line; visited 25.04.2017].  
Retrieved from: <https://www.lifewire.com/network-application-programming-interfaces-818102>
- [7] Software, F. C.: *Trello*. Web. 2011. [On-line; visited 29.04.2017].  
Retrieved from: <https://trello.com/>
- [8] Turner-Harris, W.: *phpcheckout*. Github Repository. 2000. [On-line; visited 29.04.2017].  
Retrieved from: <https://github.com/wturnerharris/phpcheckout>
- [9] Williams, L. G.; Smith, C. U.: *Information requirements for software performance engineering*. Springer, Berlin, Heidelberg. 2005. ISBN 978-3-540-44789-4.



# Appendices

<b>A Hotjar Heat Map</b>	<b>35</b>
<b>B Poster</b>	<b>36</b>
<b>C Entity–Relationship Diagram</b>	<b>37</b>
<b>D API</b>	<b>38</b>
<b>E CRON</b>	<b>39</b>

# Appendix A

## Hotjar Heat Map

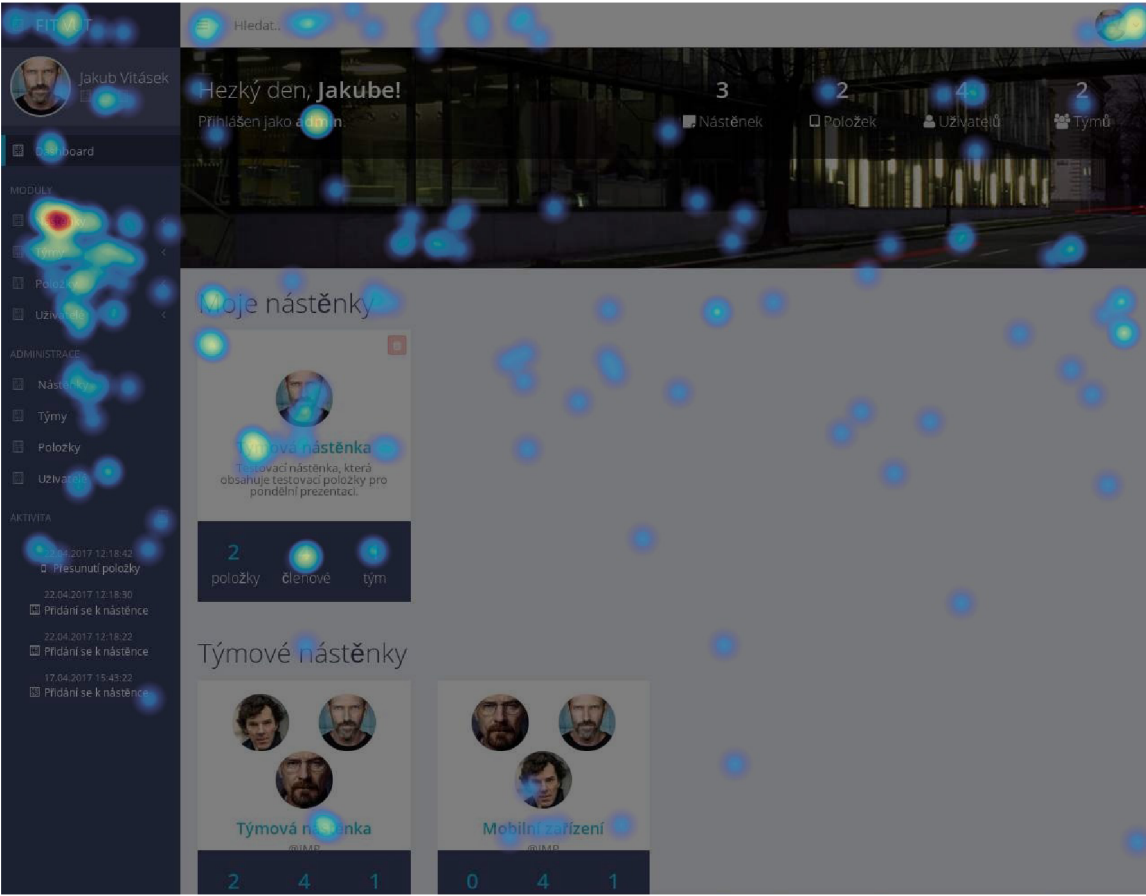


Figure A.1: This dashboard heat map shows heavy use of the left sidebar and quick-access board listing in the content section.

# Appendix B

## Poster

**T** BRNO FACULTY  
UNIVERSITY OF INFORMATION  
OF TECHNOLOGY TECHNOLOGY

WEB-BASED MANAGEMENT  
AND LENDING SYSTEM  
YEAR: 2017

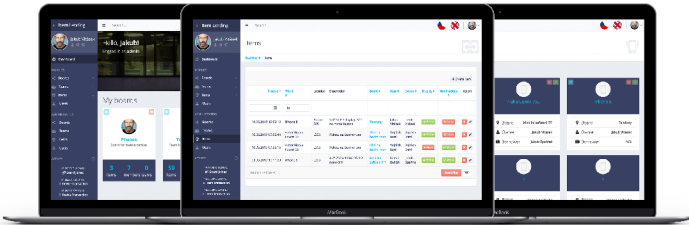
AUTHOR:  
JAKUB VITÁSEK






SUPERVISOR:  
ING. JAKUB ŠPAŇHEL

- AN EFFECTIVE DEVICE MANAGEMENT SYSTEM -

# EQUIPMENT CHECKOUT WITH ITEM HISTORY

WEB-BASED APPLICATION WITH INTEGRATED CMS  
SUPPORTS E-MAIL NOTIFICATIONS



IMPLEMENTATION PHASES				
 SYSTEM ANALYSIS	 APPLICATION DESIGN	 SYSTEM IMPLEMENTATION	 USABILITY TESTING	 APPLICATION DEPLOYMENT

**ABOUT THE SYSTEM**

The system offers a complete sign up process, internal CMS module for entity management, logging system and notifications handling. Every user is allowed to access entity lists and filter its elements by different attributes. Owing to a modular implementation and the use of design patterns, the system is extendable and usable on any server running PHP >= 5.6.

**MVC STRUCTURE**

This application uses design patterns and Nette Framework to achieve an organized and modular code, which can be extended to implement any features needed to make the system as effective as possible.

Try it now.  
<http://bcjvitasek.cz>

Figure B.1: A poster showing mock-ups of the application and the respective implementation phases.

# Appendix C

## Entity–Relationship Diagram

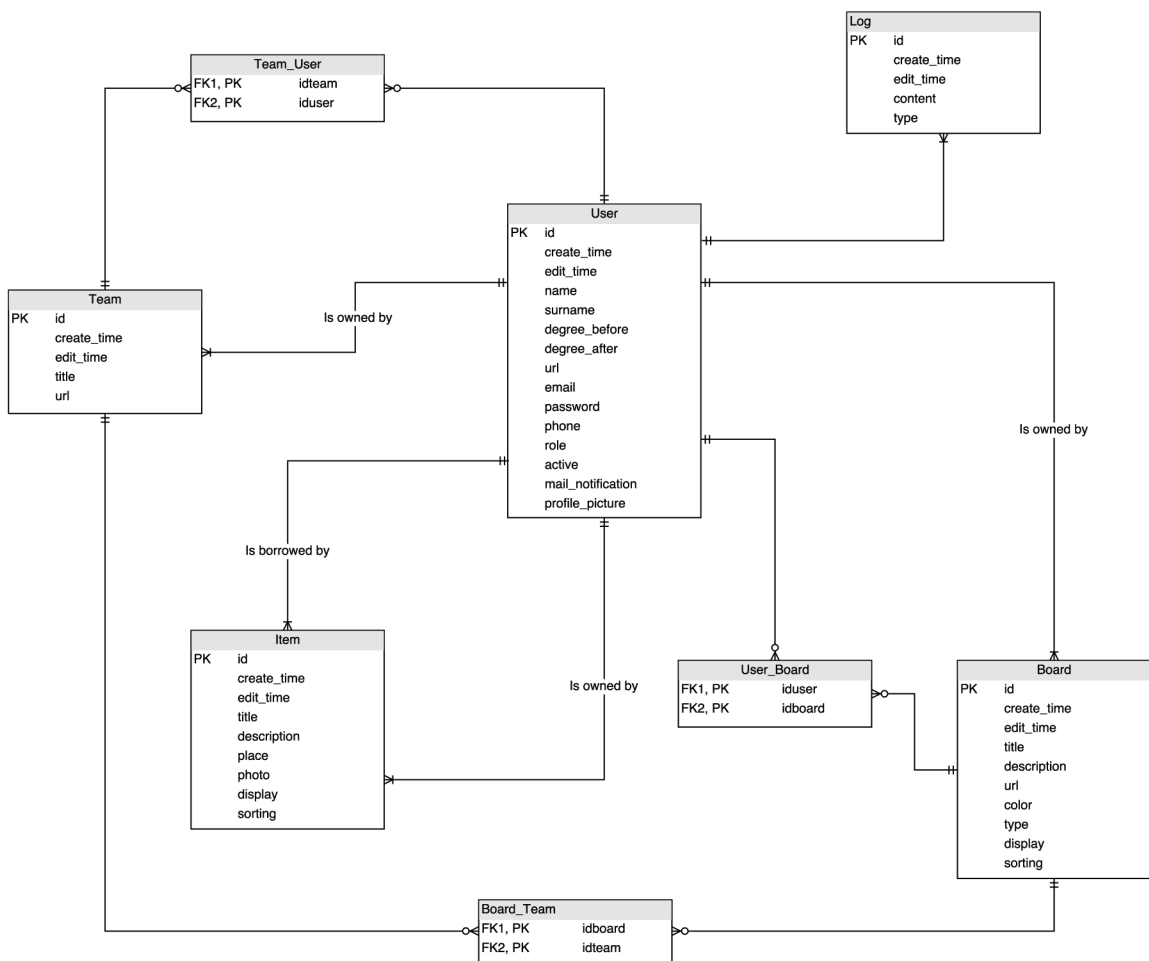


Figure C.1: Entity–Relationship Diagram of the application database structure shows the relation between tables and junction tables.

# Appendix D

## API

Enclosed below is the full list of routes available for this application and the parameters for each route action.

```
/** API */
$routeur[] = new CrudRoute('<module>/api/team/crud[/<id>]', 'TeamApi');
$routeur[] = new CrudRoute('<module>/api/board/crud[/<id>]', 'BoardApi');
$routeur[] = new CrudRoute('<module>/api/item/crud[/<id>]', 'ItemApi');
$routeur[] = new CrudRoute('<module>/api/user/crud[/<id>]', 'UserApi');
$routeur[] = new CrudRoute('<module>/api/log/crud[/<id>]', 'LogApi');

/** Cross API */
$routeur[] = new CrudRoute('<module>/api/c-board-team/crud[/<idnasteny>/<idtymy>]', 'CBoardTeamApi');
$routeur[] = new CrudRoute('<module>/api/c-board-user/crud[/<idnasteny>/<iduzivatele>]', 'CBoardUserApi');
$routeur[] = new CrudRoute('<module>/api/c-user-team/crud[/<iduzivatele>/<idtymy>]', 'CUserTeamApi');

/** Helpers API */
$routeur[] = new ResourceRoute('front/api/item-helper/<idnasteny>', 'Front:ItemHelperApi:content', ResourceRoute::GET);
$routeur[] = new ResourceRoute('front/api/item-helper/logs/<idpolozky>', 'Front:ItemHelperApi:log', ResourceRoute::GET);
$routeur[] = new ResourceRoute('front/api/board-helper/users/<idnasteny>', 'Front:BoardHelperApi:content', ResourceRoute::GET);
```

Figure D.1: API routes, separated into Crud routes and Resource GET routes, which serve as helpers for fast querying.

```
UserApi:
'name', 'surname',
'degree_before', 'degree_after',
'url', 'email',
'password', 'phone',
'role', 'active',
'mail_notifications'

ItemApi:
'title', 'description',
'location', 'idboard',
'iduser', 'idowner',
'display', 'sortby'

TeamApi:
'title', 'url'

CBoardTeamApi:
'idboard', 'idteam'

BoardApi:
'title', 'description',
'url', 'type',
'iduser', 'display',
'sortby'

LogApi:
'content', 'page',
'iduser', 'idhelper',
'type'

CBoardUserApi:
'idboard', 'iduser'

CUserTeamApi:
'iduser', 'idteam'
```

# Appendix E

# CRON

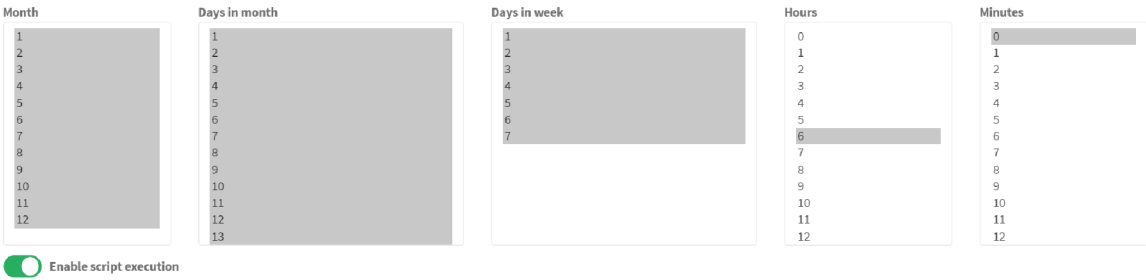


Figure E.1: The CRON settings dashboard showing the CRON job will be executed every day of the month at 6 AM.