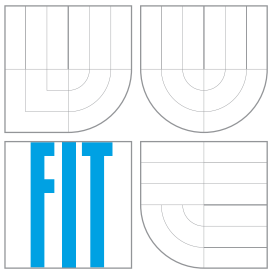
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# INOVATIVNÍ WEBOVÁ APLIKACE PRO ORGANIZACI GENERICKÝCH INFORMACÍ

INNOVATIVE WEB APPLICATION FOR ORGANISING GENERIC DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN PŘICHYSTAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZÖKE, Ph.D.
SUPERVISOR

BRNO 2013

# Abstrakt

Práce se zabývá vytvořením obecné, inovativní aplikace pro správu dat, která bude běžet v prostředí webového prohlížeče s úložištěm na vzdáleném serveru. Důraz je kladen na vhodný návrh architektury a implementaci s využitím nejnovějších technologií.



# Abstract

This thesis focuses on a creation of a multipurpose innovative application that runs in a web browser with a remote repository. Emphasis is put on clean architecture design, implementation and utilizing modern technologies.



# Klíčová slova

Webová aplikace, Ext JS framework, MVC, organizace dat, prohlížeč, vzdálené úložiště.

# Keywords

Web application, Ext JS framework, MVC, organization of data, browser, remote repository.

# Citace

Jan Přichystal: Innovative web application for organising generic data, bakalářská práce, Brno, FIT VUT v Brně 2013

# Innovative web application for organising generic data

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szöke, Ph.D.

.....
Jan Přichystal
May 8, 2013

# Contents

# Chapter 1

# Introductions

In this chapter, the intention is to cover author's incentives leading to the choice of this topic, provide broader view on contemporary course of application development, define general goals and advocate his choices in selected technologies.

## 1.1 Goals

The ultimate goal of this work is to implement a versatile web application that allows its users to store any kind of data. This data must be easy to find and categorize, the application itself must serve multiple purposes (case studies are presented further) and provide user-friendly, intuitive interface. Strong focus is targeted to overcome the fact that the application is browser-based and provide interface that is close native destop application interfaces as regards user experience.

In order to provide a good organizational system, the concept of tags and object has been introduced. Every new item put into the system should be labeled with at least one tag. Semantically, these tags are then used to categorize objects. By combining multiple tags, the objects are filtered and because many object can be labeled by the same tag, it's possible to choose an intersection or a unification of those sets.

An object is a unit that holds all the actual information. For convenient orientation in an object list, every object is optionally assigned an image, which is displayed as a thumbnail. What's more, every object can be assigned an unrestricted number of attachments, rich text description, media and, of course, a name.

## 1.2 Use cases

Despite the fact that the intention of the application is supposed to be as generic as possible to target as wide audience as possible, let's outline possible use cases. After all, it would be hardly possible to address users directly without pivoting the application guidelines towards their needs.

A large group of people that can take good advantage of this piece of software are collectors, in the general meaning of the word. We could even say that it serves the purpose of a framework that provides space for users to create neat electronic records of their specific collection. The application environment is greatly suited to fulfil this task: real world objects are modeled by object units carrying information and swiftly divided to categories by assigning tags to them, allowing to flexibly filter those objects.

To give but one example, consider you want to keep track of a butterfly collection. There would be tags covering the biological scientific classification and areas of occurrence. Then, every time you want to add a new specimen to the collection, you just select the appropriate tags an create an object with a name of the butterfly, short description about where you got it from and upload photos and/or more information.

## 1.3 Web applications

The development of computer applications for the end-users has undergone a huge shift in the last few years. The platforms that used to dominate the market are becoming obsolete due to vast accessibility of the Internet and common devices gaining more computational power than ever before. Instead of focusing on optimization and getting the best possible performance, development speed and ability to deploy to as many platforms as possible without rewriting the code are accented.

Traditionally, these features have been achieved by running an application in a virtual machine (VM) such as Java VM, which has became very popular all over the world. Although JVM can provide high-performance and is supported on all major platforms, application written for it have never been able to provide the user experience that native applications have. Additionally, it requires users to have a JVM installed on their devices.

With the Internet making its way to our everyday lives, huge amount of time spent using the computer consists of using a browser. Combined with devices with higher performance that we would have never been able to imagine in the past, this lead to a huge spread of JavaScript's popularity in the browser. JavaScript, which formerly started as a simple scripting language to validate forms, is becoming one of the most used programming languages in the world with countless complex frameworks around it. According to [6], a study of programming languages popularity considering number of projects on GitHub and number of tags on StackOverflow.com, JavaScript was in top 5 most popular languages in February 2012. New development frameworks such as PhoneGap or Titanium also enable applications written in JavaScript (using HTML5 and CSS3 for user interface) to be deployed for various mobile platforms: iOS, Android, webOS, Windows Phone or BlackBerry. The browser has become the new VM.

For the reasons stated in the preceding text, implementing the application as a web service was a clear choice: there are technologies to achieve it with and wider user base can be accessed via web rather than traditional native applications.

## 1.4 Comparison with existing products

This application usage lies somewhere between these kinds of products: to-do list applications, application for storing notes and knowledge base software. It takes some feature from all of them and could serve their purpose if required.

Let's consider Microsoft's OneNote web app and look at the differences. Whereas OneNote gives its users a free hand, my application enforces a certain structure. That could be perceived both negatively and positively. I believe, however, that insisting on storing information one way avoids confusion.

Additionally and most importantly, I took advantage of the graphical elements to help with orientation. The intention is to make it instantly obvious what kind of information objects or tags contain by applying an image to them.

A web app that comes the closest to the one of my own is certainly Evernote. It allows using tags to label notes, but it is less obvious and the tag system does not create the very core. Apart from that, their user interface is very different — I tried to get as close to a native application behavior as possible.

## 1.5 Summary

The goal is to create a maintainable web application that will allow users to neatly organize their information and files. It will be in a form of single-page web site, so user actions will evoke asynchronous requests (AJAX), manipulating data in a remote database. Emphasis is to be placed on friendly user-interface that follows practices in native applications and creating a clear, maintainable architecture.

In order to accomplish this, the program is split in two parts: the back-end and the front-end. The back-end is a database and a script layer that accesses it and serves requests. The core of the application is the front-end, JavaScript application that runs in the browsers.

In the upcoming chapters, there are parts dealing with the back-end and then the front-end implementation aspects.

# Chapter 2

# Tools

A wrong choice of tools to build a project can lead to its very failure; hence it is essential to put a lot of thought into it. We need to take into account programmer's knowledge of given programming environments, accessibility (Are those tools free? Can we run a specific interpreter on our servers?) and suitability. All these directly impact speed of development and resultant quality.

## 2.1 Front-end

Front-end represents the client-side, in this case the JavaScript running in a client's browser. The programming language for this part is given, but we need to choose a framework very carefully. Whereas in less complicated applications it is possible to avoid using a framework at all, for more complex tasks a solid framework is an absolute necessity to keep the code structured, readable and maintainable. Some frameworks provide libraries that can save a lot of work and the ability to hide browser differences comes in handy as well. We need the following:

- A way to organize code, preferably using some form of a Model-View-Controller pattern
- Abstraction from differences among browsers
- A library for DOM manipulation and AJAX calls
- User interface manager

Some of these items require further elaboration, which is provided in the paragraphs below.

### 2.1.1 Code structure

It is not uncommon for JavaScript code to become unreadable very quickly. To properly structure JS code is hard on its own and mixing HTML & CSS with scripts, storing data in the DOM, chaining events and other bad practices make it virtually impossible to read and maintain. That is the reason why the Model-View-Controller architectural pattern is so widely used. It allows the programmer to clearly separate presentational and business logic code. The model represents the data, the view describes how the model should be

presented to the user (user interface) and the controller binds the two together by defining events and relevant actions to them. In this scenario, some frameworks automatically projects changes in the model to the view, in others it needs to be evoked manually. This concept is illustrated by Figure 2.1.

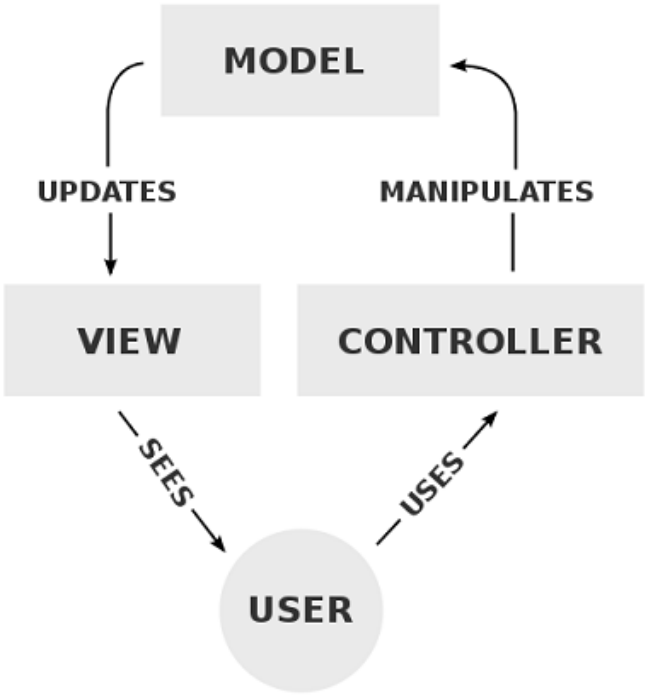



Figure 2.1: Model-View-Controller model

Among frameworks that offer some kind of code structuring patterns belong Backbone.js, Spine.js, Ember.js and many more. However, they very often require libraries such as JQuery for extended functionality.

### 2.1.2 Browser differences abstraction

Because browsers — our JavaScript interpreters in this case — are provided by multiple vendors, the syntax of JavaScript code can sometimes differ from browser to browser. This applies also to the DOM manipulation. Solution can be achieved be either using an external library such as JQuery or using a JavaScript framework that provides this functionality.

### 2.1.3 User interface manager

Truly dynamic web applications are still very new and it results in lack of good tools to design a complex user interface. Even though it would be possible to design the components or combine existing from multiple sources, it would have taken much more time to build. That's why I decided to look for a framework that also has this feature available.

### 2.1.4 Conclusions

After extensive search for a framework suitable for this task, the selection was narrowed to two candidates: Ext JS and Dojo Toolkit. I found Dojo to be more lightweight, but on the other hand, Ext JS more visually pleasing, robust and with a bigger community around it. That should lead to much quicker issue solving, so I decided to go with Ext JS. It provides all the required qualities: native support for MCV architecture, notable graphical interface designer, vast set of libraries, dynamic loading for optimization and it does not require any support from external vendors.

## 2.2 Back-end

The back-end in this case means a usual server-side scripts and a database to provide standard CRUD (create, read, update, delete) functionality. The back-end services requests by a user. Actions taken by a user in the front-end that affect data must result in changes of the user's database to ensure persistence: although smaller amounts of data could be stored inside the browser itself (using HTML5 local storage or cookies), it is not desired behavior for the users might want to access their data from multiple devices. For this reason, all data must be stored on server.

### 2.2.1 Scripting language and a database platform choice

The data are sent from the Ext JS framework in the form of JSON objects and thus need to be transformed in order to be used to change the state of user's database. There is a broad variety of languages to achieve this: Python, Ruby, Perl or Java and all of them would be fine to implement the back-end with, but I decided to choose PHP, because of its availability on most of servers. The same reasoning was used to choose a database platform - MySQL. The back-end is not the most complicated part of this project, so this open-source and well-accessible database platform is the best choice. Using a NoSQL databases such as MongoDB could lead to a better performance, but the deployment possibilities were more important.

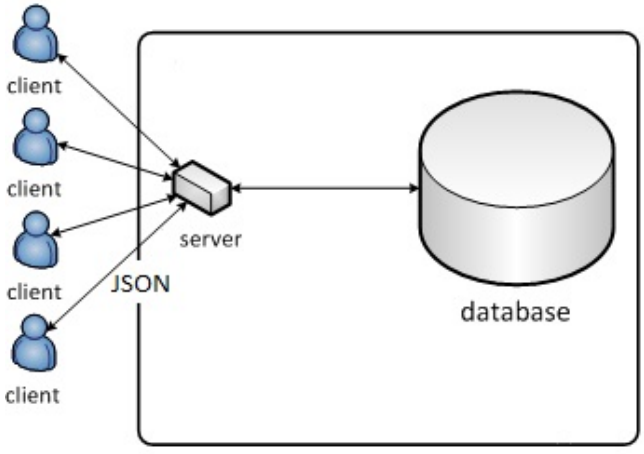



Figure 2.2: Server accepts and returns JSON objects

# Chapter 3

# The back-end

In the scope of this work, the server side makes up for the less complicated part. In spite of the fact that without the server side the application could never exist, from the implementation point of view it appears not to be overly complicated.

The back-end functions as an intermediary between the database storage and the front-end JavaScript application running within the browser. Front-end application sends the server requests, server scripts analyze them, perform appropriate actions on the database and returns a message containing information ab out the result. That means either an error message describing why the action has failed or a success message.

The application is programmed using PHP scripting language. To make the code more readable, objective design was used. This would not be absolutely neccessary, because the complexity of the back-end is not too extensive and because it does not take advantage of the abstraction aspect of object-oriented programming, but the encapsulation and modularization possibilities could be utilized very well.

## 3.1 Back-end structure

The application's entry point must be used to evoke any action in the back-end, otherwise no return values can be reached. In this case, the point of entry is „index.php“ file. In accordance with the GET values of a HTTP request, the control flow is redirected to a newly created object or a static method of a class, which is defined to deal with requested kind of action. Every time a request is made to the point of entry, it analyzes the „action“ parameter, instantiates necessary object, invokes appropriate method and returns a message with the results back to the front-end application. Other parameters associated with the request are passed in JSON format.

The following diagram illustrates the process of inserting a new object into the database:

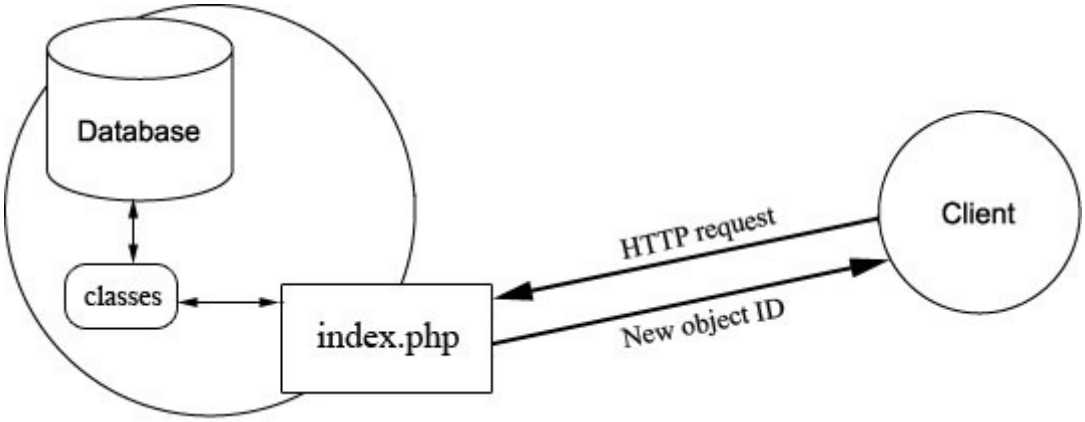



Figure 3.1: Inserting a new object

First, a HTTP request with is send with all the information about the newly created object. The back-end returns the objects new ID. This is generated by the RDBMS and it's necessary to pass it back to the front-end application for future reference and manipulation.

## 3.2 Autoloading classes

One of the good practices of object oriented programming is creating one source file per class definition to keep the sources well-structured and easily manageable. This may lead to long list of includes in the beginning of scripts. This is no longer inevitable in PHP 5, because an autoloading function was introduced in this version. When an autoloading function is registered in a script, it is called every time a class that has not been defined yet is used. While normally accessing such a class would result in an error, the autoloading function can try to find the class and load it before it does. The function gets the the class name as a parameter. If classes are categorized and stored in multiple folders, it is possible to recursively search the directory that contains all the classes, but this is would lead to very poor performance. For this reason, naming convention is used for each class in the back-end system, that the name of a class contains the path and the name, delimited by underscores. File in which the class definition is stored is named the same as the class name. The class name is then just split by underscores to construct the path and the actual loading takes place.

## 3.3 Point of entry

Another important issue is chaining includes - including a file withing another one. It is a feature of PHP programming language that the it takes either an absolute file path or, in the case of relative paths, the directory of the first file that is being included. If a file should be included in several scripts that are stored in different directories, it would be necessary to always use absolute path or change the path in every file to a location-specific relative one. This can be very inconvenient and get annoying when a file is moved to a different directory. Introducing the „point of entry“ approach is very helpful in this situation, because if the program always starts withing a file which is known in advance, files with classes can be easily included from this point in every application file regardless of their physical location.

This approach solves some of the security issues. If the application backend could be accessed from more than one places, it would be more susceptible to an attack. The checking whether the user is authenticated can be done before any action is taken and the logic taking care of this issues doesn't need to be placed anywhere else.

## 3.4 Database access

MeekroDB, a light-weight MySQL access library for PHP, was used in this project. It simplifies most database use cases and prevents SQL injections. Accessing the database directly with SQL statements from PHP is possible but includes serious security risks associated with SQL injections. Default approach is to use PDO (PHP Data Objects) library, which provides similar functionality. This option was ruled out because of high verbosity of its interface. MeekroDB, on the other hand, has most of its functions in well-known „printf" format and enables to write most functions in one line. As was already mentioned, the back-end is not the major part of the project and thus does not require vast frameworks for database abstraction that would just decrease performance: MeekroDB addresses important security issues, is light-weight, easy to use, free and open source as well.

## 3.5 High level entity-relationship model

The database data structure can be simplified as follows:

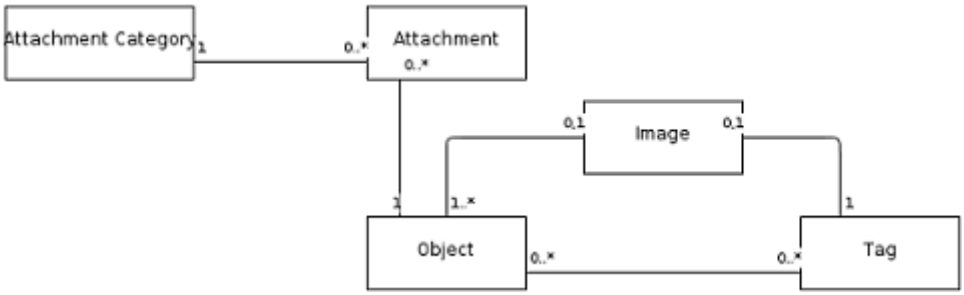



Figure 3.2: High level ERD

The diagram conceptually shows how the data are stored, while all the implementation details are hidden.

## 3.6 Back-end security issues

### 3.6.1 SQL injection

Direct SQL Command Injection is a technique where an attacker creates or alters existing SQL commands to expose hidden data, or to override valuable ones, or even to execute dangerous system level commands on the database host. This is accomplished by the application taking user input and combining it with static parameters to build an SQL query.[2] The-backend accepts parameters in the form of integers (IDs to find rows in a database) or strings (e.g., tag names, object descriptions, etc.) that form parts of the SQL queries. To prevent SQL injectin, we must check if the input type is actually the type we

expect and escape strings so they are treated as strings and don't add any new logic to the SQL statement. The MeekroDB [1] takes care of these issues for you. All you need to do is specify an input type in the WHERE clause. When inserting into a database, MeekroDB infers the types on its own. Escaping strings is done implicitly.

```
DB::update('objects',
    array('desc' => $item->desc),
    "id=\%d", $item->id);
```

This code sample updates the *objects* table. The field(s) to update is specified by the second parameter (a hash of key-value pairs) and the third one is a WHERE clause. The input parameters are referenced by symbols as in a printf function in C. Before the SQL query is compiled and executed, it checks if the data type which is being inserted fits the column type in a database and so it does with the input parameters in the WHERE clause.

### 3.6.2 File access

Users of the application upload images and file attachments, which are stored in the server's file system. The issue here is how to make these accessible from a web browser and maintain users's privacy. If these files were to be accessed directly, everyone could just type a URI of that file to download it. Even if they didn't know the filename, they could try to guess in order to retrieve data. Despite the fact it would be possible to store files in a database instead of the file system, this is not an optimal option, because that could lead to significant performance drops if the userbase grew larger.

A rather simple solution to this problem is to use a PHP script that checks if the user accessing a file has been authenticated and if he has, it just reads the file and sends it to him. The actual files are then stored someplace where they cannot be reached by the public users.

# Chapter 4

# Front-end

The front-end is the core of the whole project. The main goal is to achieve the comfort of a desktop application in a browser. Many tasks emerge with this: designing a user-friendly layout, making the application intuitive to use, choosing the right techniques to keep the application swift etc. Contemporary browsers were not intended to provide dynamic widgets as we know them from desktop environment, so every task related to this gets much more complicated. Fortunately, most of the work for designed a UI has been done by the creators of Ext JS framework.

## 4.1 Layout

The picture figure 4.1 shows the first draft of the application layout.
1. On the top there is a toolbar with text buttons. Those are to be replaced by buttons with pictures, but for the prototype are sufficient. They are used to add content: objects and tags.
2. The section with tags is on the left side and serves as a navigation panel. Every change in tag selection affect displayed objects.
3. The main area shows objects that contain tags selected in area. It allows to user to select multiple objects at once either by pressing control and clicking on the objects or by creating a selecting area with a mouse. 2.
4. When the details panel is clicked, it expands over the main area and shows details about selected objects.

## 4.2 Code organization

### 4.2.1 More on MVC

JavaScript is an example of a language with support of object oriented design. It uses prototype-based model, but this does not apply when working with Ext JS framework. In Ext JS, they changed this to class-based model, which is well-known to most developers. There will be more about that in the next chapter. As mentioned earlier, to keep the code well-structured and maintainable, Ext JS allows programmers to use Model-View-Controller architectural pattern. This means that the code resides in three kinds of classes, each with different semantics:

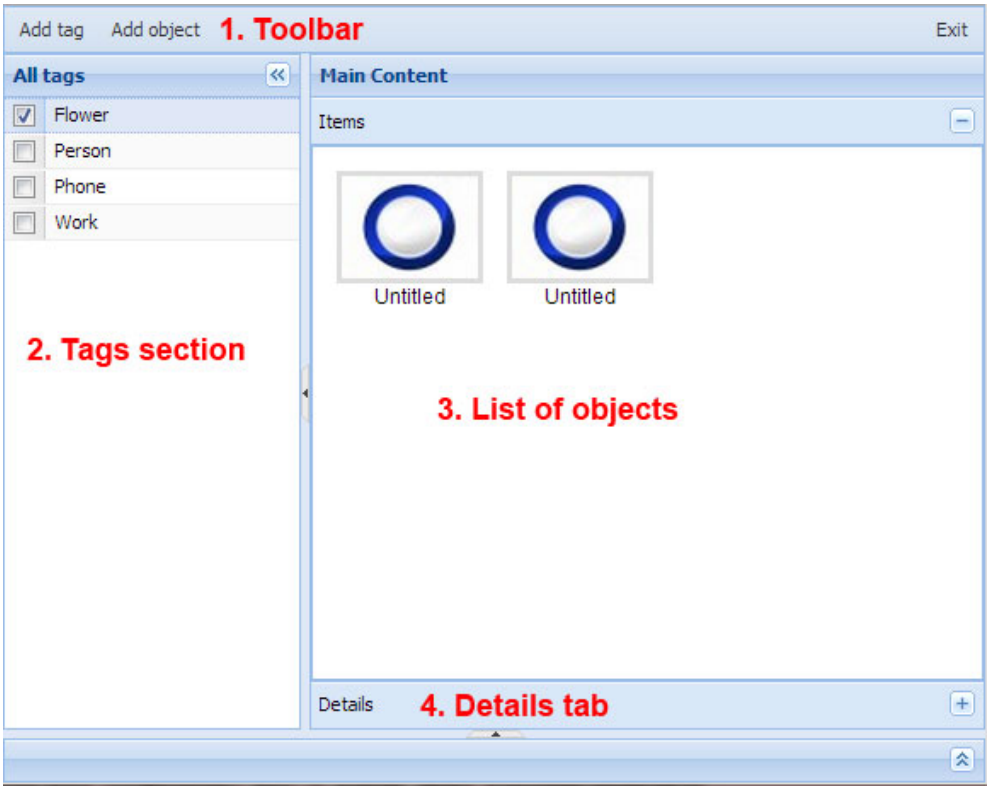



Figure 4.1: Application layout

- *Model* is a collection of fields and their data (e.g. a User model with username and password fields). Models are normally used with Stores to present data into grids and other components
- *View* is any type of component - grids, trees and panels are all views. Views are updated when a change in its related model occurs.
- *Controllers* are special places to put all of the code that makes the app work - whether that's rendering views, instantiating Models, or any other app logic. [9]

### 4.2.2 Class file organization

In the front-end, the standard practice of one file per one class is also followed. The file name is directly mapped to the class name, e.g. My.class.Name is located in My/class/Name.js. This is important for the Ext JS class loader to work. The application is being loaded when index.html is accessed. The file is almost empty, but contains important dependencies: Ext JS framework, plugins, css files and file „app.js“, which bootstraps the entire application. The layout is coded and controllers are registered in this file. The rest of the application code is then located in MVC classes.
The directory structure looks as shown below:

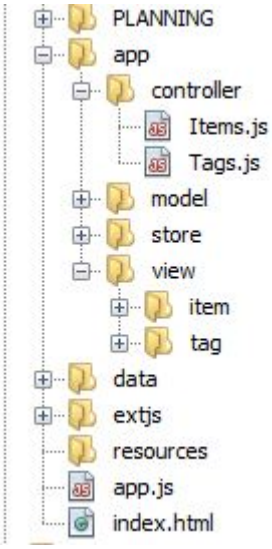



Figure 4.2: File structure

# Chapter 5

# Ext JS OOP

As it was mentioned in the previous chapter, the object-oriented system has been reworked in Ext JS framework and the changes were quite wide and important for Ext JS development. This chapter deals with this matter in more detail. First, the classical JavaScript object-oriented paradigm is briefly explained and then the approach of Ext JS is presented and explained how it was taken advantage of in this particular project.

## 5.1 JavaScript's OOP model

Unlike the majority of languages that support object-oriented programming, JavaSript does not implement class-based model and supports prototype-oriented OOP model instead. That is important to know when we want to discuss behavior reuse, because most programmers are familiar with class-based model and thus get confused easily, even though the principles are not hard to grasp. In JavaScript, there is no special keyword for class definition. New classes are defined the same way functions are. [5]

```
function NewClass (arg1) {
    this.first_argument = arg1;
}

var newClassInstance = new NewClass('value');
```

The code above defines a class and creates an instance of this class. There is also no explicit keyword for a class constructor, because the function's code is a constructor itself. It gets executed every time a new instance of this class is created. As there are no keywords for class definition and constructor method, there is no keyword for inheritance. Inheritance in JavaScript is put into practice by chaining prototype properties. Every class has a prototype object containing its methods and static variables. When a new instance is created, it automatically gets a `[[prototype]]` property pointing to it's class prototype. When we want to implement inheritance, we change the prototype property to point the superclass. In the new class' constructor we must manually call the superclass' constructor before any other code is executed.

The following code shows what this process looks like in an actual implementation:

```
//Shape - superclass
function Shape() { }
// add a method to Shape class
```

```
Shape.prototype.move = function(x, y) { /* move the shape */ };
 // Rectangle - subclass
function Rectangle() {
  Shape.call(this); //call super constructor.
}
// change the prototype to point to superclass
Rectangle.prototype = Object.create(Shape.prototype);
```

## 5.2 Class system in Ext JS

The creators of Ext JS framework decided to go with a different approach that would be more suitable for what what actually happens when an Ext JS application is developed and to simplify this whole process. It makes the framework more accessible to the majority of programmers, but also makes the process less verbose. They introduced two functions: `Ext.define` and `Ext.create` [8]. The former takes two parameters: name of the class and a data object. The name usually consists of a vendor's initials, packages (path to the file) and the actual name of the class (filename without extesion), e.g. *Ext.form.field.Textfield*. Note that this is just a convention, so it is not mandatory yet highly recommended. [7] The data object contains key-value pairs of methods and properties that are to be copied to the newly created class as static. Example of creating a new component extending the Window class.

```
Ext.define('MyApp.view.user.Add',
    extend: 'Ext.window.Window'
});
```

Inheritance is then added very elegantly by adding a pair where the key is named „extend“ and the value is the name of the class to be extended, the superclass. Everything is then taken care of automatically by Ext JS engine.

`Ext.create` function is used to instantiate objects by the class definition. It takes two parameters as well: a class name and a data object, whose properties will be injected to the new instance. The framework adds them to the constructor they will consequently become private properties of the instance.

```
var window = Ext.create('Ext.window.Window', { width: 600 });
```

Every new class in this project is created this way: all views, controllers, models, stores, custom components etc.

## 5.3 Application bootstrapping

Speaking of classes definition and instantiation is a good invitation to discuss how these classes are initially loaded — application bootstrapping. The entry point to the front-end is a simple html file with CSS and JavaScript includes. There is no HTML markup apart from the very skeleton, all element of the „body“ are created and rendered on the fly. Loading of the CSS files is quite tedious, but it gets interesting with the JavaScript part. There are just two JS files to be initially loaded: `ext.js` and `app.js`. The former is supplied by Sencha and contains the foundation of Ext JS framework — class system, layout managers etc. The `app.js` is the very core of the program that load everything necessary dynamically.

### 5.3.1 Dynamic loading

Dynamic loading means that whenever a class that has not been loaded is accessed, Ext JS will try to load it dynamically according to its name from a file (according to naming convention described in section 5.2. This is a supported feature, which results in faster initial loading time, but it also has drawbacks: whenever such an event occurs, the users need to wait until the file is downloaded from server and executed before it is possible to continue, thus creating unwelcome delays.

### 5.3.2 Static loading

Alternatively and more standardly, static loading takes place. There is a function in Ext JS that takes care of including classes in a program: `Ext.require`. It takes a string parameter — name of the required class — and synchronously loads it. I chose static loading for this project after some testing. Despite longer loading time, the whole loading process takes about five seconds and every further loading on the same device is faster because of caching. After taking into account that it was hosted on a server in America and almost all users were from the Czech Republic, I found this sufficient. Over one hundred of different people tried this application, so this should be statiscically valid sample. Additionally, when all classes are loaded statically, it is possible to make use of the Sencha SDK and pack all classes into one minified file.

# Chapter 6

# Front-end structure

We will have a look at some core classes, especially Controllers, Views and Models and how this system used in practice. This project's front-end uses four controllers, three models, four stores and eleven views.

## 6.1 Controllers

Application logic is stored in controllers and before any application is started to be implemented, a programmer must decide how to name them and what responsibilities should be assigned to respective controllers.

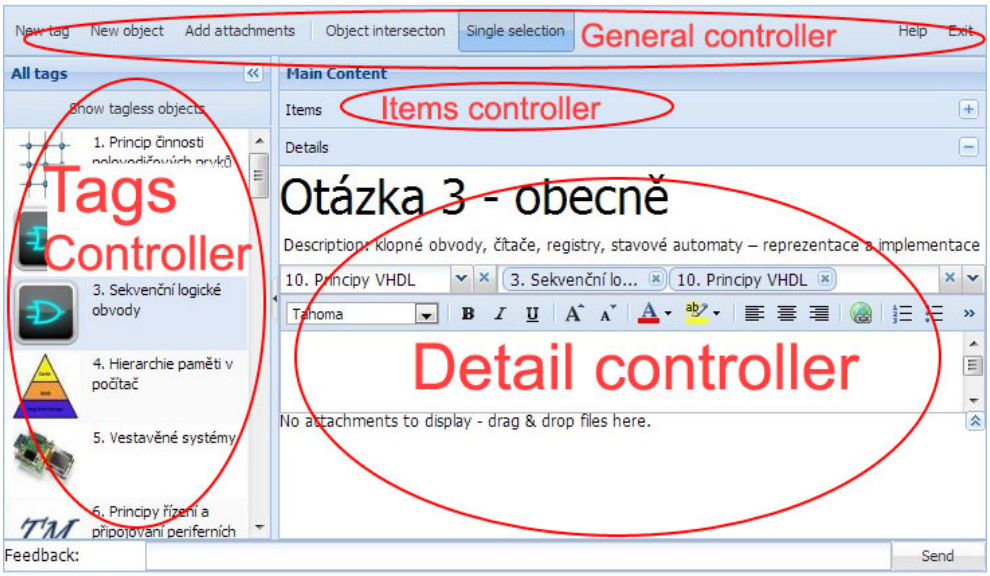



Figure 6.1: Responsibilities of controllers associated with UI

Figure 6.1 shows how the functionality is divided in this particular project by associating every controller to a specific part of the user interface. This way, it is very easy to figure out where to put a new function or where to look for an existing one, e.g. the function that takes care of event that is fired when a tag is selected in a tag list is to be found in „Tags" controller located in a file within a folder with controllers. There are three controllers that correspond directly with the UI components: Tags, Detail, and Items controller, but there

is also a „General controller“. It serves miscellaneous purposes: handling events in the toolbar, registering hotkeys or saving/loading of application state.

Event handling is also taken care of in controllers, there is even a special way to treat events to make this process easy and concise. There is a function static method `control` in a Controller class that takes care of adding listeners for events. Vast majority of events is associated with some component (clicks, rendering, hiding, etc.), so we need to specify this component. This is done by specifying a string to find the component by component query method provided by the framework. An example taken from the project:

```
this.control({
    'viewport > taglist': {
        viewready: this.checkSelectedRecords,
        itemdblclick: this.editTagDblClick
    }
});
```

This piece of code registers two event listeners for a *taglist* component, a descendant of a *viewport*. If there were more components that fit the description, it would register events listeners and handlers for all of them. The `this` keyword refers to the controller, so the handler functions are to be found in the same controller. However, an anonymous function could be uses instead as well.

## 6.2 Models and stores

Before we dive into views in Ext JS, models and stores should get mentioned, because they are tightly connected with them. In models, we specify what record in a data store should look like, it is a definition of a data structure, similar to what we know from other programming languages such as C, Java etc. Models define the patterns, it is a class, and *records* are instances with actual data. A data store is a collection of records. When we want to manipulate what is displayed in the application, we access a particular store and change the data there, which automatically result in a change of what is presented to the user.

One of the features of data stores is a *proxy*. It is utilized by stores to load and save data, either from from/to a server or locally. This saves a lot of time, because instead of handling every event for every CRUD operation on the store and creating the requests (in case of a server transfer) manually, the programmer defines just URIs and parameters to handle respective events. I took advantage of this feature with manipulatings tags and object in this project and it saved a lot of time and code lines.

## 6.3 Views

Views represent the presentational logic of an application. The first purpose they serve is grouping components that are semantically connected together, for example a window with a texfield and a confirm button should be members of one view. This is not a rule though, even one component, assumably heavily customized, can form a view too.

In Ext JS, the term *view* is a little different from what we imagine under this word in other programming languages and frameworks. Usually, a view means an instance of a View class. This can true in Ext JS as well, but not necessarily, because a class that extends

some component considered to be a *view* likewise. Usage of this kind of views is abundant, but less interesting than customized instances of the View class, which were used in this project as well.

### 6.3.1 The View class

Even though extending a view class also allows to add pre-defined components, its main purpose is to create brand new look with HTML markup and connect with a data store. It uses a templating mechanism. Templates are HTML fragments with added markup elements.

```
new Ext.XTemplate(
    '<p>Kids: ',
    '<tpl for=".">', // iterate over the data source
        // use current array index to
        '<p>{#}. {name}</p>', autonumber
    '</tpl></p>'
);
```

Let's describe what happens with help of the example above: when such a template is assigned to a View class, it will be used to when the view is rendered. The Ext JS engine will automatically take a store assigned to the View class, analyze the template and fill it with data from the store by replacing the variables (in curly braces) with actual values. That results in complete HTML that can be rendered.

Another feature of the View class that is worth to be noted is that the view also provides built-in behavior for many common events that can occur for its contained items (rendered by the template) including click, doubleclick, mouseover, mouseout, etc.

### 6.3.2 File app.js

This file is the entry point to the front-end where everything is initiated from. The first lines are `Ext.require` commands to load the classes from the Ext JS framework that are utilized in the project. Some of them are used directly and some of them are further extended to provide additional functionality. After that there is specified a list of controllers. When the controllers are loaded, they further require their own views, stores and models until the program is fully processed.

The main layout is defined in this file as well and is rendered the processing is finished. Some of the customized views are listed as composite components that should be rendered in the layout.

# Chapter 7

# Front-end implementation details

## 7.1 Storing application state

I decided not to store information about the application state — meaning tags, that the user has selected, that should become selected next time the app is loaded again — on server, but to utilize some of the possibilities to archive content in the web browser itself. The first think that would come to mind is to use cookies. Cookies allow to save up to 4kB of data, which would be sufficient. On the other hand, cookies are sent to the server every time a HTTP request is made resulting in more data being sent over the network. Another solution, which actually got implemented, is a HTML5 feature `localStorage` object. It has been present in all major browsers since 2010 and allows to save key-value pairs in the browser without the server knowing. However, there is no support of advanced data types such as arrays or objects. Luckily, Ext JS framework contains functions that can deal with this by decoding the object or array to a string before writing and encoding it back when the value is read. Every time a tag is selected/deselected in this application, it is pushed to/removed from an array of selected tags in this persistent localStorage objects. When the application is loaded, it read that array, selects appropriate tags, that fires the select events, so the list of associated objects is updated as well.

## 7.2 Deployment

There are about 275 classes in the Ext JS framework and all of them are heavily commented. It is highly unlikely that all of them would be used in a project, so there is no need to copy them to a deployment environment — in fact, they take up almost six megabytes. For this reason, Sencha introduced a product Sencha SDK Tools, that takes care of this and makes it simple to get a project ready for deployment. Using a command line, the tool goes through your project folder and seeks for dependencies — used classes of the framework, controllers, models, views etc. This produces a JSON file with a JSON-formatted list of all the classes that must be included in the production version. The second step is to use this in order to generate a single, minimized file containing the whole application code. It takes all the file, strips the comments, redundant whitespace characters and changes identifiers (function arguments, variable names etc.) to short versions everywhere possible.

Deploying the application in such manner significantly reduces the amount of data that needs to be transferred and as everything resides in one file, the number of HTTP request is minimized as well.

## 7.3 Error logging

Even after the application is put into production bugs may occur and should be dealt with. However, it is not possible to expect users to be sending the error description that appears in the browser's console, ideally with the function call stack. Luckily, browsers provide an event that is fired whenever an error occurs with information about that particular error passed to the handler function in an argument. Error then can be sent to a server and stored for further analysis. Instead of creating a brand new solution addressing this issue, I decided to use an existing one. I went with an application called Muscula available at http://www.muscula.com. It provides a free service with cloud storage of logged errors, email notifications and graphs showing when and what errors occurred.

## 7.4 Functional programming in JavaScript

JavaScript is a multiparadigm programming language; it supports object-oriented, imperative and to some degree even functional paradigm. This turned out to be quite helpful for managing arrays. In the project I needed to do some filtering of objects according to currently selected tags and so I took advantage of JavaScript's functional nature. The algorithm must check if the set of tags selected in the left view is a subset of the object's tags — that is if the application runs in intersection mode, otherwise it check if at least one of the selected tags is a member of the object's tag set. If the condition is not met, the object will be discarded.

Standardly, this would include branching to decide which operation is to be performed and then looping through the arrays, but this can be avoided by using Ext JS functions for array filtering, higher-order functions `Ext.Array.some` and `Ext.Array.every`. Note that these are usually also natively built-in in the *Array* object and then Ext JS just calls them, but are also implemented in the framework library in case they the user uses an older browser that doesn't support them. These functions are similar to list manipulation functions common in functional languages, e.g. filter or map function in Haskell. Their first argument is an array to be worked upon and the second one is a function. They execute the specified function for each array element until the function returns a falsy or truhty value, respectively. If such an item is found, the function will return false (true if `Ext.Array.some`) immediately. Otherwise, it will return true (false). The code snippet then look something like to following:

```
// mode variable contains true/false according to current
// selection mode; arrayLoopFunction is uninitialized
mode ? arrayLoopFunction =
        Ext.Array.every : arrayLoopF = Ext.Array.some;

var isValid = arrayLoopFunction(
    selectedTags,
    function(i) { // is the item "i" member of that array?
        return Ext.Array.indexOf(objectTagsArray, i) >= 0;
    }
);
```

First, the ternary operator decides which reference to a function should be stored in the *mode* variable. The reference is that used to evoke the function and is given two arguments:

array of selected tags and a function that decides whether an element is a member of an object array and return true or false accordingly. The `every` function return true only if all the return values of its argument function return true, while the `some` function succeeds after one element from the selected tags has been found in the object's tag array.

## 7.5 Ext JS themes

A problem that comes with using a well-established framework such as Ext JS is that the application look it produces is rather similar. The light blue design may become irritating after a while. Sencha designers decided to address this issue by introducing a very sophisticated theming system that takes employs SASS language [4] and Compass compiler [3]. SASS is a language that extends regular CSS expressing ability and compiles back to CSS.

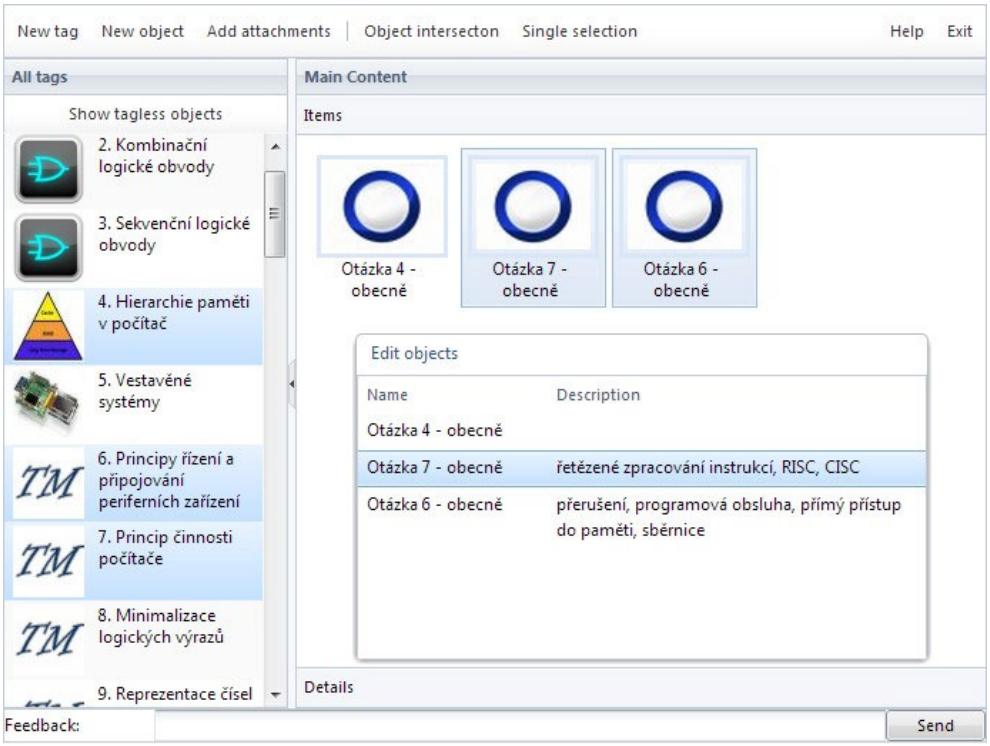



Figure 7.1: New theme

Working with SASS allows to use variables, functions, selector inheritance, nested rules, etc. Basically, we write the code in more generic way. For instance, instead of defining a color for each element, the color is defined once in a global variable. First, the generic color scheme is changed and then redesigning each Ext JS component that is used in the project takes place. Sencha provides a template that serves as a base theme to be customized. After the editing has been done, the Compass compiler utility is used to generate minimized css file that can be put into production.

The process can be found in full detail in documentation. [10]

The theme I created as a part of this project is to be seen in figure 7.1. It is not completely finished, but most of the component have been redesigned.

# Chapter 8

# Conclusions

## 8.1 Published resources

In order to share the product with the public, I registered a domain and hosted the application online. Furthermore, a login system has been set up, so everyone can register their own account and use the application in his or her own way. All the source codes are present on an attached disc — there is a minified version of the application online, which is not really readable.

- www.thingsmaster.com/ - a website with forms to register a separate user space and log-in the web application
- www.thingsmaster.com/statnice.php - shared user space for gathering knowledge needed for bachelor final state exams. This also serves as a demo, no login is needed.
- www.thingsmaster.com/app/statnice_theme.php - the previous with a custom made theme
- http://www.thingsmaster.com/manual.html - concise manual with images to demonstrate how the application should be used

## 8.2 Software metrics

External libraries and framework apart, the actual application itself consists of:

- 2680 lines of code
- divided in 34 files

SQL statements to construct a database and the website for login and registration are not included as they are not a concern of the assignment.

## 8.3 Summary of personal gains

The work on this project led not only to an interesting and useful product, but also provided me a really great opportunity to get familiar with the world of single page dynamic web applications. During the research stage I learnt about many frameworks that are used for

this purpose. Then, I was struggling with the Ext JS framework for a very long time, because its learning curve is rather unfriendly for newcomers. However, after I finally got comfortable with it, I appreciate the flexibility and countless tools and shortcuts it provides for developers, saving enormous amount of time and resources. From the very beginning, I kept in mind to keep the code open for future extensions as well and I hope it is evident by looking into it.

Lastly, this project gave me an opportunity to get my hands on some amazing new technologies and tools, being it SASS language, Muscula error logger service or the Google Chrome Developer tools, that make debugging this kind of application possible and much less painful.

## 8.4 Future development

My intentions are to keep the development going on in my free time. The improvements that can be done are indeed numerous. They range from user experience and design to performance optimization. To name just a few: extend support of drag & drop to increase user productivity, finish the custom theme to distinguish the application, assigning priority to objects and many more. It is a great project to further develop my skills with Ext JS, which can be also utilized on the job market in the future, because dynamic web application are a hot topic these days and Ext JS is a great tool to get the job done.

# Bibliography


[1] Meekrodb. http://www.meekro.com/.

[2] Php manual. http://php.net/manual/en/security.database.sql-injection.php.

[3] Eppstein C. Compass. http://compass-style.org/.

[4] Weizenbaum N. Catlin H. and Eppstein C. Sass language. http://sass-lang.com/.

[5] Mozilla. Javascript guide. https://developer.mozilla.org/en-US/docs/JavaScript/Guide.

[6] Stephen O'Grady. The redmonk programming language rankings: February 2012. http://redmonk.com/sogrady/2012/02/08/language-rankings-2-2012/.

[7] Ashworth S. and Duncan A. *Ext JS 4 Web Application Development Cookbook*. PACKT Publishing, 2012. ISBN 1849516863.

[8] Sencha Development Team. Ext js documentation. http://docs.sencha.com/extjs/4.1.3/!/api/Ext.

[9] Sencha Development Team. The mvc application architecture. http://www.sencha.com/learn/the-mvc-application-architecture/.

[10] Sencha Development Team. Theming guide. http://docs.sencha.com/extjs/4.1.3/!/guide/theming.

# Appendix A

# Content of the enclosed disc

- application/ - the actual application source codes and database layout
  - application/data - back-end source codes
  - application/app - front-end source codes
  - application/app.js - minified front-end in one file
- manual/ - manual in HTML
- theme/ - theme source code
- theme/css - compiled theme
- video/ - short video presentation about the project in Czech

# Appendix B

# Manual

## B.1 The main screen

- First, you need to choose tags from the list on the left or create you own by clicking „New tag“ button from the toolbar
- After that, doubleclick existing tags to see them in detail or create new, name them and add text or attachments.

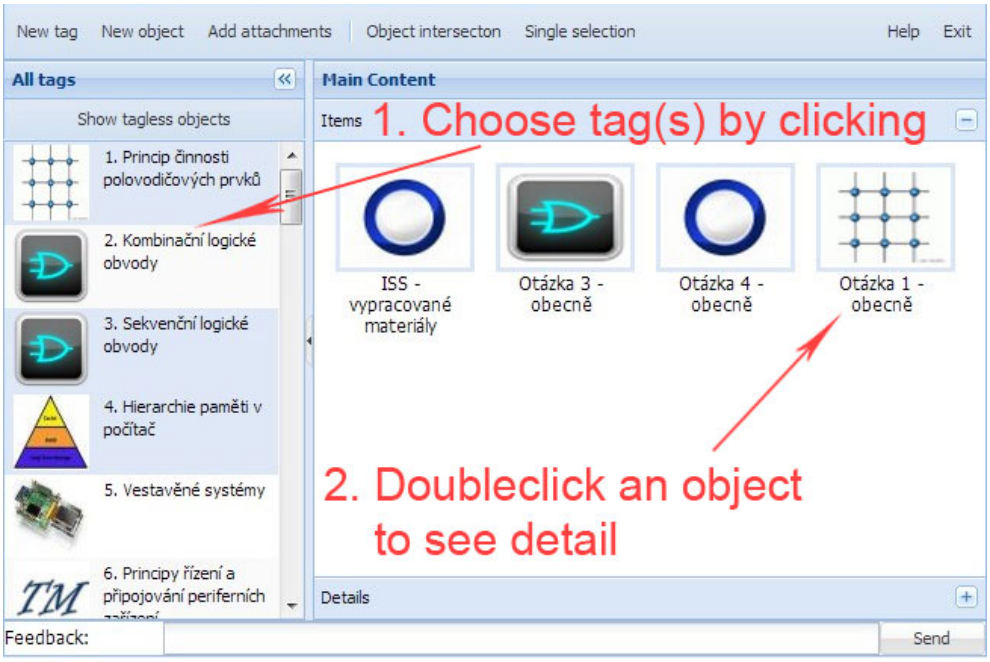



Figure B.1: Main screen - manual

## B.2 Object detail

- Doubleclick on the name or desctription to edit it. You can write anything you want into the textfield and style it. The text is saved automatically three seconds after you finish writing.

- Below the description, there is a box where you can edit tags associated with this object. This affects what is displayed when you select these tags. Additionally, you can choose a primary tag - the object will be shown in the object list with the same image as this tag's image.
- Finally, there is a field for attachments. It's possible to drag & drop them or invoke upload dialog by clicking 'Add attachment' button from the toolbar.

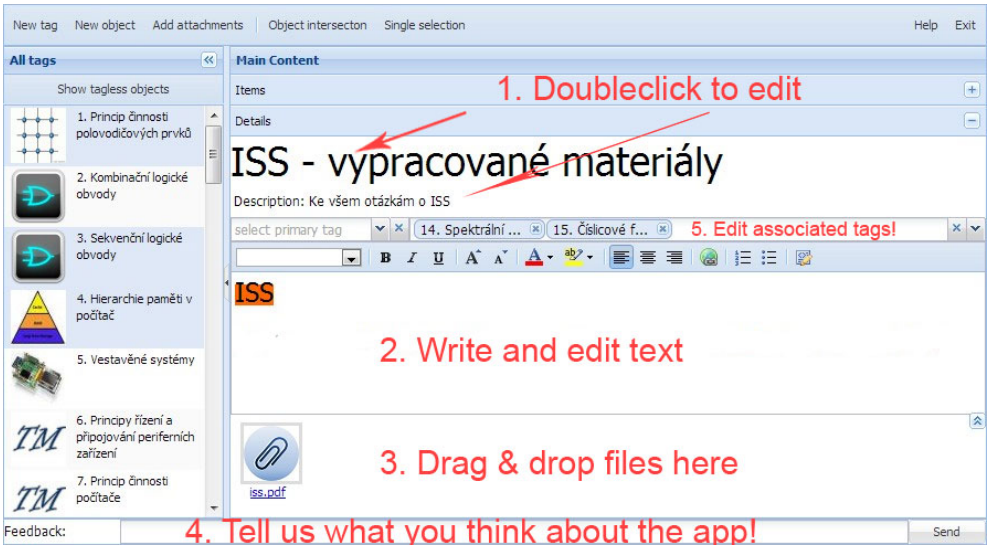



Figure B.2: Object detail - manual