Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

User Profile Module for matfyz.sk Portal

Bachelor's Thesis

2014 **Mária Šormanová**

Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

User Profile Module for matfyz.sk Portal

Bachelor's Thesis

Study programme:	Computer Science
Study field:	2508 Computer Science
Study department:	Department of Computer Science
Supervisor:	doc. RNDr. Zuzana Kubincová, PhD.
Consultant:	RNDr. Martin Homola, PhD.

2014 **Mária Šormanová**

65511495

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mária Šormanová
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: User Profile Module for matfyz.sk Portal

Aim: The goal of the thesis is to propose, develop and implement a profile module of the portal matfyz.sk. The main challenge in this project is to design suitable integration patterns of the profile module with the other existing parts of the portal and associated applications. The profile should serve the purpose of unifying the whole matfyz.sk portal and thus the logging users in and out of the system will be taken over by it. It will also collect and visualize the data from all of the associated applications providing the reader with the newest and actual information about user's activity at the portal (apart from the personal information and enrolled courses, e.g. recently published blog posts, comments or wiki contributions, articles read lately, latest course activities, etc.). Since it will also show badges and achievements collected by the user as well as her progress in study, the profile could serve as a tool for motivation boosting in students.

Supervisor:	doc. RNDr. Zuzana Kubincová, PhD.
Consultant:	RNDr. Martin Homola, PhD.
Department:	FMFI.KZVI - Department of Informatics Education
Head of department:	doc. RNDr. Zuzana Kubincová, PhD.
Assigned:	28.10.2013
Approved:	29.10.2013

doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

....
Student

....
Supervisor, Consultant

Acknowledgements

I would like to thank my supervisor doc. RNDr. Zuzana Kubincová, PhD. and consultant RNDr. Martin Homola, PhD. for all their help, advice and support during writing of this thesis.

I hereby declare I wrote the thesis *User Profile Module for matfyz.sk Portal* by myself, only with the help of the referenced literature, under the careful supervision of my thesis supervisor.

Mária Šormanová

Abstrakt

Cieľom tejto práce bolo vytvoriť a včleniť nový subportál *profile.matfyz.sk* do existujúceho komunitného portálu Fakulty matematiky, fyziky a informatiky UK - matfyz.sk. Používateľské dáta, ktoré boli predtým ukladané na rôznych subportáloch vo viacerých kópiách, sú teraz uložené centrálne na jednom mieste - v databáze tohto subportálu, kde sa zobrazujú na profilovej stránke používateľa. Aby bol presun efektívny, spravili sme analýzu a kategorizáciu používateľských dát. Subportály, ktoré si doteraz dáta, ktoré využívali, spravovali sami, teraz potrebujú prístup k užívateľským dátam uloženým na *profile*. Na tento účel sme navrhli a vytvorili rozhranie, pomocou ktorého vedia subportály pristupovať k dátam o používateľoch a taktiež ohlasovať aktivitu používateľa na danom subportáli. Aktivity sa zobrazujú na jeho profile a slúžia na prepojenie celého portálu a zvyšovanie záujmu o ostatných používateľov. Na profile sa taktiež zobrazujú nadobudnuté ocenenia a odznaky, ktoré používatelia získavajú za svoju aktivitu na portáli, čím sa ich snažíme motivovať k tvorbe hodnotného obsahu. V práci sme navrhli metódu, ako implementovať tento mechanizmus. Subportál *profile.matfyz.sk* taktiež prebral funkciu prihlasovania a odhlasovania používateľov. Zavádza jednotné prihlásenie do všetkých subportálov naraz (single sign-on) a tým spríjemňuje prácu s celým systémom. Na tento účel sme implementovali protokol OAuth 2.0, ktorý sme prispôsobili a rozšírili tak, aby vyhovoval našim potrebám. Implementácia celej práce je dostupná na webovej stránke subportálu: `http://profile.matfyz.sk` a na priloženom CD.

Kľúčové slová: centalizácia, užívateľské dáta, single sign-on, gamifikácia

Abstract

The goal of this thesis was to create and incorporate a new subportal *profile.matfyz.sk* into the existing community portal of Faculty of Mathematics, Physics and Informatics of Comenius University - matfyz.sk. The user data, which was previously stored on different subportals in multiple copies, is now stored centrally in one place - in the database of this subportal, where it is displayed on the profile web page of the user. The analysis and categorization of the user data was conducted to make this transfer effective. Subportals, which until now took care of the user data they operated on and required for their work, now need to get an access to the user data stored on *profile*. For this purpose, an application programming interface was designed and created, through which the subportals can access the data and also report the users' activity on particular subportal. Activities are consecutively shown on the profile web page and thus interconnect the whole portal and encourage users to be engaged in other users on the portal. On the profile web page also achievements and badges are shown, which users earn for their activity on the portal and are thus motivated to create valuable content. We came up with the method to implement this mechanism. Subportal *profile.matfyz.sk* also took over the logging users in and out. In order to make the portal more convenient, the single sign-on solution was designed, which enables users to log in at once to all the subportals. For this purpose the protocol OAuth 2.0 was adjusted and extended to fit our requirements. Implementation of the whole thesis is available at `http://profile.matfyz.sk` and on the attached CD.

Keywords: centralization, user data, single sign-on, gamification

Contents

Introduction

The community portals on universities, where students can meet and share their knowledge, experience and notions have become commonplace. The Faculty of Mathematics, Physics and Informatics of Comenius University is not an exception and provides students with the portal matfyz.sk. The portal is developed exclusively as parts of Bachelor and Master Thesis of the students of the faculty and serves two main purposes: it is a tool for teachers to facilitate the teaching of their courses and a place where students can express their notions by writing a blog post or a wiki article. The portal consists of more subportals, each running one of the mentioned services.

What the subportals have in common are the registered users. The subportals share a small amount of the user data, but generally they store the pieces of information about users independently, which can lead to inconsistency and is insufficient for the correct operation of such a large portal. Therefore a new system for storing and sharing the user data needed to be developed. This thesis introduces and describes the new categorization and division of the user data among subportals. The new subportal *profile.matfyz.sk* was implemented and incorporated to matfyz.sk to store the general user data. The user data is presented on the profile website of the user, where they can edit it centrally in one place and the change takes effect on all the subportals. The method and way of accessing and exchanging this data needed to be developed as well, since otherwise the other subportals could not display and operate on the data. The profile subportal thus provides the other subportals with the application programming interface (API), through which they can access the information they need about the particular user.

One of the main goals of *profile.matfyz.sk* is also to unify the whole portal. To reach this goal two new features are presented on the profile webpage of the user:

Achievements and Badges Rewards earned by the user for their activity and engagement on the portal. They should serve as a motivation for the user to create valuable content on the subportals.

Activities The newest and up-to-date information about the activities of the user on the portal. It should interconnect the subportals and encourage the interaction among users. For collection of this information from the subportals, the API includes endpoints to post the user's activity on their user profile webpage.

The other tool which unifies and makes the portal more usable, is the single sign-on solution, designed and implemented as a part of this thesis. The single sign-on enables the users to log in only once at the beginning of their work with the system and not to fill in their credentials, to every subportal they want to visit, separately.

In the first chapter, we explain the current condition of matfyz.sk portal and identify the problems. We formulate the requirements on the new subportal. In the second chapter, we explain the principles of the single sign-on and gamification, analyse the available solutions and compare them to choose the one best suited for our portal. In the third chapter, we describe the designed structure of the user data administration and our single sign-on solution's technique of operation. In the fourth chapter, we introduce the data model and API for accessing and providing information about users. And in the last chapter, we present the implementation of this thesis, which is available on `http://profile.matfyz.sk` and on the attached CD.

Chapter 1

Portal Matfyz.sk

1.1 What is it

Matfyz.sk is an unofficial homepage of the Faculty of Mathematics, Physics and Informatics of the Comenius University. It is a project which was and is being created by students as parts of their Bachelor and Master theses. More about the matfyz.sk portal can be found on [7]. At the moment, the portal facilitates more subportals:

blog.matfyz.sk The community portal for former and present students and graduates from the faculty. The ambition is to gather the people interested in events and activities going on at the faculty in one place, and provide them with the informal environment, where they can share and exchange their thoughts and opinions not necessarily only about the faculty.

courses.matfyz.sk E-learning platform which facilitates some courses offered by the faculty.

wiki.matfyz.sk Community wiki, whose content is created by the students, offers additional information to the official faculty and university websites.

beania.matfyz.sk,... Websites of various students' activities.

In the future there is a plan to include more subportals which will meet the philosophy of matfyz.sk, and transform it into the large community portal.

1.2 Current condition

The individual subportals coexist on the same domain - matfyz.sk. To get the full use of each of them, user has to create an account and log in when visiting. However, as they were created successively by different people to fulfil different requirements, they work as relatively independent units. The greatest amount of the user data is saved in the blog's database, and the other subportals have only the information user has given them by filling out the registration form. This means that each portal has it's own database of user data. It is an unwanted state mainly because of the two reasons: data can be inconsistent - user can fill in different data on different subportals, and the change of the information is very inconvenient - user has to make the desired change on every portal separately that can possibly lead to already mentioned inconsistency.

Single sign-on (for explanation see: 2.1) is present and implemented via shared cookie. That is a technique where more subdomains use the same session-cookie. They check, whether the session-cookie is present and thereby determine which user is currently logged in. However, this solution only works across subdomains of one domain because of the same-origin policy. The Same-origin policy restricts browsers to allow a document retrieved from one site to access some document from the site of a different origin (origin is defined [16] as scheme, host, and port of the URL). Since there is a possibility of incorporating also third-party portals (hosted on different domains) into matfyz.sk in the future, this solution is no longer sufficient.

1.3 Requirements

The problem with the scattered user data and single sign-on resulted in the need of a new subportal, which would take care of both these problems. Since the linkage to the "user" and the information about them, we chose the name "profile" for this new subportal of matfyz.sk.

Figure 1.1: profile.matfyz.sk logo

We have divided the requirements on the profile portal into three categories:

1.3.1 Userdata

1. The userdata, used on all the subportals, would be stored in the central database maintained by profile subportal. The analysis of this data is required to determine which data is general and used across all the subportals and which is subportal-specific.

2. Other subportals would access this data via API provided by profile subportal.

3. The webpage with user data would be created. Each user would be assigned a subdomain username.profile.matfyz.sk, where a webpage consisting of user's profile information would be placed. There would be a possibility to edit the data by the user.

1.3.2 Single Sign-on

1. Logging users in and out of the portal and other issues around single sign-on.
2. Registering new users to the system

1.3.3 Other profile functions

1. Implementation of giving and receiving of badges and achievements as a gamification element for the portal (see 2.2).
2. The list of the best users, according to the badges, achievements, and teachers' decisions would be put together.
3. On the profile webpage of the user, the activities of the user in the other parts of the portal would be shown (see 3.3.1).

Chapter 2

Background

2.1 Single Sign-on

2.1.1 Motivation

Imagine having a university website consisting of more internet services, which the site provides to students and employees. For example, it could offer webmail, calendar, library website, schedule etc. These services are practically independent from each other and offer users a functionality which needs two following attributes to function correctly and properly:

authentication Who the user is. This is standardly done by giving user the username and password combination. When authenticating themselves, user logs in via these assigned credentials. The site consecutively knows which data from the database belongs to this particular user.

authorization What the user is allowed to do. In the terms of the site - which data from the database is the user authorized to manipulate on.

This would usually get to the point that every service has it's own database and it's own set of users (actually the sets are very similar, since almost everybody uses all the services at the university). This system leads to users having many different usernames and passwords combinations - one for each service. They have to remember all these combinations and furthermore, to preserve security, passwords have to be changed often, which makes it impossible to keep them in memory. Users than write passwords down and so make them vulnerable to being stolen and abused. This is not the only problem of this approach. Both users and IT managers are having hard time. For IT manager it is cumbersome to create all the accounts needed, once a new user has to be added to the system. Deleting a user from the system is also causing a problem, since one needs to delete not just one but several accounts scattered around. Users become frustrated by the tedious login processes and tend to pay less attention to it, which is a potential security risk as well.

2.1.2 Specification

Single sign-on (SSO) solves the problem with users having to remember many passwords, by giving them just one universal. The identity of the user is checked centrally against the authorization server and only once at the beginning of the work with the system. The user is assigned an electronic identity, which is then automatically transferred to all the services in the same system. This means that the user, once logged in to the central system, is automatically logged in to all the services and does not have to authenticate themselves with the username and password when entering another service. Single sign-on simplifies the administration of the user accounts, since in this approach user credentials are stored in one central database, which is much easier to administrate, and reduces the risks of the passwords being captured, as they are passed to the system just once. Other userdata and permissions can be also stored centrally and exchanged automatically between the central database and the other services. However, the single sign-on has its drawbacks as well; the advantage of the universal password means that when this password has been corrupted, the attacker gains access to the whole system in the name of the user whose password has been stolen. In the system without single sign-on, only one part of the system in the name of one user would be impacted by the leaked password.

2.1.3 Available Implementations

There are many implementations at disposal. They vary from commercial products, free and open-source software, to plain protocols which we could decide to implement. For the purpose of matfyz.sk we have considered several options which could be suitable for our purposes. The closely considered options were as follows (open-source, free software and protocols):

CoSign

CoSign is an open source project developed at the University of Michigan [12], originally created to provide single sign-on for the university. It is based on two types of cookies: login and service cookie. Login cookie is created by the CoSign server and it is the central cookie, which says that the particular user is logged in. Service cookies are issued when user logs in to a new service (service is the part of the system to which the user can log in). CoSign consists of three components:

CGI script The central script responsible for logging users into each service. It creates the login cookie and ties it up with the service cookies, each time user logs into new service.

Daemon The central daemon responsible for keeping track of all cosign sessions on the server.

Filter Resides on a service server and knows which parts of the service website are protected by CoSign. If the user wants to access protected part, filter checks if the user is authenticated and gets their username, authentication realm, IP address etc. It creates the service cookie.

The flow of the authentication is as follows: User tries to access protected resource on the service server. The filter on the server finds out that there is no service cookie present and so redirects user to the main CGI script on the CoSign server. The CGI finds out that there is no login cookie present and so shows the login page, where user is prompted to authenticate themselves by username and password. CGI verifies the credentials, and if they are correct the login cookie is set and the user is redirected back to the service (when user tries to log in to other service, login cookie will already be present, and so the authentication part will be skipped). Service filter afterwards creates the service cookie and redirects to the CGI, which registers the service cookie with the login cookie and redirects back (the central daemon needs to keep track of all the services user is logged into). In the last step, the filter verifies the service cookie and allows user to access the protected resource.

Shibboleth

Shibboleth is an open-source software package for Single sign-on [14]. It implements Security Assertion Markup Language, which is an XML-based open standard data format for authorization and authentication data exchange. The adherence to standards is the main advantage of Shibboleth, since it can cooperate also with services outside of the user's organization. The elements which appear in the authorization flow are:

Web Browser Represents the user.

Resource Content that user wants and which has restricted access.

Service Provider Element which facilitates the SSO process for the resource.

Identity provider Element which provides the authentication of the client.

The authentication flow, similar to that of a CoSign, works as follows: The user accesses the protected Resource. The monitor responsible for this resource finds out that the user does not have any active session. To get one, monitor redirects the user to the Service Provider. Service Provider issues an Authentication request, which is subsequently sent along with the user to the Identity provider. The Identity Provider checks for a session (this is an Identity Provider's session and so not the same one as the monitor has checked for). If it is present, Identity Provider skips the authentication part and continues to the next step. Otherwise, user is prompted to authenticate themselves (f.e. with their credentials). When successful, session is created. Having the user identified, the Authentication response and the user are sent back to the Service provider. Service Provider validates the response and creates the session for the user. The handling is given back to the Resource, which again, as in the beginning, checks if the user has an active session and since this time they do, the requested data will be sent to them.

OAuth 2.0

Unlike the previous two options, OAuth is not a software package, but a mere protocol. It is the second evolution of the OAuth protocol [8]. The "auth" in the name stands for authorization (and not authentication) and it was designed to solve a different problem, though close to the one of the single sign-on. Still it can be used for the purpose of the SSO as well.

We will demonstrate the principles of the protocol on the example of a website which offers a photo depository, where some user created more photo albums. Another site offers photo lab service for printing and editing photos. The user wants to use this service to print out one album from their photo depository website. To make it easy for them, they do not have to download the photos from the depository site and afterwards upload it to the photo lab. The photo lab just asks them to give them their username and password for the depository site and downloads the album directly to its server on their behalf. What is bad about this approach? By giving their credentials to the third party, they do not only expose all the information they have on the depository site (f.e. all the other photos or their personal details), but they also give third-party an access to do what they want on their behalf. The third-party can now even change their password and prevent the user from ever getting their account back. Although the idea of relieving the user of the cumbersome procedure of first obtaining and then giving away the data they want is great, this implementation is definitely not a good way to do so. User do not want to give some third-party the full access to their data; they want to give it only a partial one - to the data, which the service really needs and operates on (in the example above, this would be the particular photo album). And this is exactly the problem OAuth solves. User can grant access to their resources to another site, without sharing other information about themselves.

We will now briefly explain the principles of the protocol; the complete explanation can be found in the protocol's specification [5].

There are four roles in the protocol:

resource owner The entity capable of granting access to a protected resource (this is the user).

resource server Server hosting protected resources.

client An application (e.g. website) which gains access from the resource owner and makes a request to the resource server on behalf of the resource owner.

authorization server Entity which authenticates the resource owner and, with their permission, issues access to some protected resources to the client. Authorization and resource servers are usually the same one.

The main concept used in this protocol is the principal of access tokens. Instead of giving client resource owner's credentials, an access token is issued by the authorization server, with which the client can request protected resource from the resource server. There are many advantages of this approach, besides others: the grant can be easily restricted by invalidating the token, tokens can have time-limited validity, and they are issued to have only a limited scope.

The protocol flow is as follows: The client requests an authorization from the resource owner (preferably via authorization server). Client receives the authorization grant (there are four types in the protocol, some of them will be explained in 2.1.5), which expresses resource owner's authorization for the client to access the protected resource. The client then authenticates itself with the authorization server and presents authorization grant from the resource owner. Based on these two credentials (they are both verified), authorization server issues an access token to the client. With this access token, client is now authorized to access the protected resource of the resource owner.

This idea can be used for a single sign-on as well, for we can set the tokens to have short lifetime (as long as usual idle time-out) and issue them only after the user has successfully logged in to the authorization server. The tokens will be issued to the individual services (clients) from this central server. Once they get the token, they know that a user must have been logged in (because the access was granted to them) and they create a local session. With the possession of the token, clients get all the information they need about this particular user from the central resource server (the same as authorization server in this case).

2.1.4 Comparison

CoSign and Shibboleth are ready made software solutions, which require only configuration and installation, while the last one is a protocol which has to be implemented (some libraries which facilitates the basic functionalities are available). From the short description above, we have seen that all of them offers the single sign-on functionality. More differences among the solutions arise with single log-out (the opposite of the single sign-on, where user is logged out of all the participating services with one click).

The Shibboleth's flow does not give a place for the single log-out, because after the Identity provider creates the Authorization response, it gives the handling back to the Service provider; hence the Service provider is the only one with the information that the particular user is logged in to this Service. Identity Provider does not have a possibility to force Service providers to log users out. CoSign offers this functionality, since Filters check regularly, if the user has not logged out; however, log-out of only one service is not possible. Either the user is logged in to all the services or none of them. In OAuth solution, we can implement this as we please. Single log-out is possible via restricting all the access tokens issued in the session, deleting the session on Authorization server and requesting clients to log out their users in their local sessions. Also logout of only one client could be implemented - the access token belonging to this client would be confined and the client would be given notice to log the user out of the local session on the client.

CoSign and Shibboleth are products for single sign-on and so support prevalently only this functionality, whereas when choosing OAuth, created for the authorization issue, we can manage, apart from single sign-on and single log-out, also the exchange of the data between the Resource server and clients, which we need for the other part of the profile subportal - the management of the userdata (see 1.3.1) and thus solving two issues with one protocol. The final decision was, consequently, to implement OAuth 2.0 protocol.

2.1.5 OAuth 2.0 - detailed

To build further on this protocol we first need to understand more in detail, what entities take part in the protocol workflow, what their roles are, and how do they interact. Only the parts which are relevant for the purpose of this thesis will be explained (complete explanation can be found in the OAuth 2.0 specification [5]).

As mentioned before, the four roles in the protocol are: resource owner (RO), resource server (RS), authorization server (AS) and client. Client obtains authorization grant from the resource owner via authorization server.

Grant types

There are four types of the authorization grant, from which we will have a closer look at two :

Authorization code The client directs the resource owner (their user-agent) to an authorization server. RO authenticates themselves and gives the AS authorization (e.g. via dialog form where the resource owner is asked to confirm the extent of access which would be given to the client). The RO is then directed back to the client with the authorization code. RO is authenticated only with the AS and so the credentials do not pass through the client.

Client credentials Every client has its own pair of credentials called `client_id` and `client_secret` (clients get these upon registration to the Authorization server). With this pair, client can be directly granted access from the AS without the intermediary of the RO. This grant type is usually used when client is acting on its own behalf (when client is also RO), or for protected resources for which client is authorized upon previous agreement with the AS.

Having obtained the authorization grant, client exchanges it for the access token. Access token is a string, usually opaque to the client, representing the authorization. Access token represents specific scope and duration of access granted to the client.

Endpoints

The AS has two endpoints to which requests and user-agent redirections are made by the client:

Authorization endpoint Used by the client to obtain authorization code (with authorization of the RO) via user-agent redirection.

Token endpoint Client exchanges the authorization grant (e.g. authorization code) for the access token here.

The client's endpoint is called redirection endpoint, where the AS sends its responses.

Authorization code request

The request for an authorization code is done by constructing URI in "application/x-www-form-urlencoded" (as defined in [11]) to the authorization endpoint with following parameters (All the following requests will have similar structure. We will hence list only the parameters):

`response_type` Set to "code".

`client_id` Client identifier.

`redirect_uri` For security purposes. This must match the URI registered to the client on AS. It is required to prevent false requests, and access tokens being sent to attackers' URIs.

`scope` The requested scope of the access token. Supported scopes are defined by the AS.

`state` Value used to prevent cross-site request forgery (An attacker sends client the access token associated with their protected resource rather than with the victim's. That results in client using attacker's access token when manipulating victim's data. E.g. the client sends some private information to the attacker's protected resource rather than the victims and so the attacker can easily access them on their own behalf.) This value is included when redirecting back to the client so the client can check that the response is really the response to their request and so that the access token origins from the authorization server and not from an attacker.

An example request(extra line breaks are added for better orientation):

```
GET /authorize?response_type=code
&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
HTTP/1.1
    Host: server.example.com
```

When the request was successfully verified and the RO gives the authorization, the authorization code is delivered as a parameter in a query component of the `redirection_uri`:

`code` It is valid for a very short period of time, since the purpose is only to exchange it for the access token.

`state` The state parameter, as described before.

When the request was not verified or when the RO denies access the error response is sent to the `redirection_uri`. The parameters are as follows:

`error` The values of this parameter can be: invalid_request - request does not have desired form; unauthorized_client - client is not authorized to request an authorization code; access_denied - RO has denied the access; unsupported_response_type - AS does not support this response type; invalid_scope - unknown or invalid scope; server_error - unexpected server error; temporarily_unavailable - server is unavailable.

`error_description` Human readable additional information about the error

`state` The state parameter, as described above.

Access token request

Exchanging Authorization code

The parameters are sent in HTTP request entity-body (request is sent via POST method):

`grant_type` Set to “authorization_code”.
`code` The obtained authorization code.
`redirect_uri` As described above.
`state` As described above.
`client_id` Client identification.
`client_secret` To authenticate the client.

When the request was successful, the following response is returned:

`access_token` The access token issued by the AS.
`token_type` “Bearer” token type will be used throughout the project. This token type means that the token is a string, which is the “key” to the protected resource without any further verifying.
`expires_in` The lifetime of the token in seconds.
`refresh_token` Refresh token is a token used to gain new access token after the first one has expired.
`state` As described above.

When the request is not accepted, the error response, in the form of the one described above, is sent back to the client. The error parameter can have following values: invalid_request, invalid_client, invalid_grant, unauthorized_client, unsupported_grant_type, invalid_scope.

Exchanging Refresh token

The client can ask for a new access token also with the valid refresh token. Refresh tokens are sent with the access tokens, to give client option to prolong the access. The request is sent to the token endpoint with the following parameters:

`grant_type` Set to “refresh_token”.
`client_id` Client identification.
`client_secret` To authenticate the client.
`refresh_token` The refresh token.

The answer is either an access token or an error response, as described sooner in this section.

Client credentials grant type

In this grant type no middle-step before acquiring the token is present. This grant type is usually used when the client is acting on its own behalf and not on behalf of some RO. The request is created by adding following parameters to the HTTP entity-body and sent to the token endpoint:

`grant_type` Set to “client_credentials”.
`client_id` Client identification.
`client_secret` To authenticate the client.
`scope` As described above.
`redirect_uri` As described above.

Afterwards either access token response or error response is sent as described sooner in this section.

Security

The OAuth protocol relies on the transport-layer security and the implementation should thus require to use e.g. The Transport Layer Security portocol.

2.2 Gamification

2.2.1 Motivation

As stated in [17], people generally love playing and games. They have accompanied us since the dawn of civilizations. According to Zichermann and Cunningham, the researchers even predict that we are hard-wired to play, since they have noticed the relationships between our brain, neural system and game play. By this mechanism, games, alongside fun, cause children to improve their abilities and skills. Playing games thus reminds people of the best memories of their childhood, because games imply fun and the distraction from the “real world”. As Zichermann and Cunningham state, turning experience into a game can cause a change in the behaviour of the person - the game elements increase the engagement in the activity. For this reason, recently many websites include game mechanics into their designs, which has proved to be profitable, since users become more engaged and tend to spend more time on these sites.

2.2.2 Description

The term “gamification”, as explained in [2], describes “the use of game design elements in non-game contexts”. This means that we take mechanics and game design techniques from video games and bring them to, for example, a web portal which has nothing to do with games in general. There are plenty of game elements which might be adopted to the webportal, but we will take a closer look at three of them, which we decided to incorporate into the matfyz.sk portal.

Achievements

An achievement is usually defined as a goal, which serves the purpose of rewarding a user for their efforts. It is represented as a picture with the title and description, which, upon earning, is displayed on the userprofile webpage. The exact conditions, which need to be fulfilled in order to get a particular achievement are set. When the user fulfil all these conditions, the achievement is automatically earned. Achievements should arouse the engagement into the activities on matfyz.sk portal and motivate users to be more active and creative when doing assignments.

Badges

A badge is similar to an achievement in a way, that it is also represented as a picture with the description and title, which is displayed on the userprofile webpage. However, badges are not generated by the system upon fulfilling certain conditions, but they are exchanged among users. A user can give another user some badge, with the comment, why they reward that user with this badge. Badges should arouse the interest in other users' content and increase interactivity among users thus encouraging the social learning.

Leaderboards

Leaderboards, a list of users put up according to some ranking, serve the purpose of showing the users how they stand in the comparison to the other users. The presence of a leaderboard should elicit the desire to raise the rating and thus move up the leaderboard, since people are by nature competitive and want to compete with their peers.

Chapter 3

Proposed solution

3.1 The OAuth adjustment

In the 2.1.4, we have decided to implement the OAuth 2.0 protocol and use it for the single sign-on and exchange of user data. The exact proposal of the additions and changes to the protocol flow and application of it to the matfyz.sk portal will be proposed here.

Roles

The profile.matfyz.sk subportal will become both authorization and resource server. The server will be thus responsible for both issuing the tokens and providing protected resources, which are user data in our case. The other subportals will become clients. The end-users of the portal will become resource owners.

Client registration

All the subportals will need to be registered to the authorization server. On registration, they will have to present their:

home uri For navigation purposes. This address will be linked as the subportal's homepage.

requests uri On this endpoint the subportal must be listening to the requests from the profile server. The types of requests are described later in this section.

redirect uri The endpoint to which the OAuth responses will be sent.

Furthermore, the `client_id` and `client_secret` will be stored (analogue to the username and password) alongside with the `scope` to which the subportal is authorized (at the moment, all subportals will be authorized to all the scopes - this option is left here for the future use). The registration of the client will be done by the administrator, for whom a form will be provided on profile website.

User registration

New users will register themselves via form on the website, where they will be prompted to choose their username and fill in some personal information.

3.2 User data

User data is all the data related to the user on a particular system. It is a large and various amount of entries, ranging from personal information to user interaction with the system. When the system consists of more subsystems interacting with each other, the important thing is to keep the pieces of user data, which are being operated on by more subportals, consistent throughout the portal. The easiest way to keep some data consistent is to store it only in one place and let all the other parts of the system look them up there. How to determine which data should be treated this way? The answer is simple: all the data that is used on more than one subsystem; because the fact that at least two different subsystems operates on the same user data, makes this data prone to inconsistency (each subsystem can e.g. store its own copy of this user data and thus the system do not know, when some other subsystem changes certain value). To prevent this, the data should be stored in one place.

3.2.1 Categories

The first step to manage the user data effectively is, as described above, a good classification. According to the description in the first paragraph, we can divide our user data into two categories:

General All the pieces of user data, which are used on more than one subportal.

Subportal-specific The user data related to the particular subportal, which are not used on any other subportal.

Now we will analyse the data currently stored on matfyz.sk portal and divide it into these two categories. All the data listed below is currently stored locally on particular subportal:

- Blog
  - General: username, first name, surname, email address, short text about the user, avatar
  - Subportal-specific: authority of the user, posted articles, comments, tags, etc.
- Courses

  - General: username, first name, surname
  - Subportal-specific: courses for which the user is enrolled, specific course data, etc.

- Wiki
  - General: username
  - Subportal-specific: posted articles, edited articles,...

As we can see, there are many pieces of user data, which belong to the general category. However, the fact that each subportal stores its own values of these items can lead to the already mentioned inconsistency (for example, the name of the user on blog and courses - although, it is the same user, they are free to fill in different surname to blog portal and different to courses). The user data is thus scattered to the subportals. As described in the first paragraph of this section, this is an unwanted state. We need to avoid having user data, which belongs to the general category to be stored on more than one place.

It is obvious that the exact distinction between the general and subportal-specific category of user data is critical to the improvement of the situation. When we state clearly, which user data belongs to which category, we can let the general category be stored only on the profile subportal and all the other subportals will ask for this information from the profile subportal. The subportal-specific data will be stored on subportals as before. To link the subportal-specific user data to the general user data of a particular user, the username of the user will be used as the primary identification of the user across the subportals. The username will so become the only intersection of the user data stored on the subportals, which makes no place for inconsistencies, because username cannot be changed.

General User data

Some items of the user data, which should belong to this category, were already mentioned. These are: username, first name, surname, avatar picture, short text about the user and their e-mail address. However, we have agreed that we want to add some additional information to this category, since matfyz.sk is a community portal and user identity is an important part of it and should thus give user the opportunity to build their portal identity more accurately. The proposed additional pieces of information are:

occupation This will be an enum value, which determines whether the user is a student, a teacher, or an external user (someone, who does not study or work at the faculty).

text about the user User will be given option to write more about themselves. This will be carried out via division of the about text into two parts: short one and long one. Short one will be sent to the subportals as a response to the request for the about text. And long one will be shown on the profile webpage of the user, to give them more space for the introduction of themselves to the community.

Subportal-specific User data

All the user data, which does not belong to the general category belongs to the subportal-specific category. The subportal is completely in charge of this data. All of this data will be linked to the user by their username, as described above.

3.2.2 API for subportals to get information

When the user data from the general category will be stored on the profile server, we need to provide other subportals with the facility to get the user data they need. For this purpose an application programming interface (API) need to be present on the profile server. There is a possibility that the subportals of matfyz.sk might be hosted on different servers in the future, so the API should be realized e.g. via HTTP requests. As described in 2.1.3, the OAuth protocol fits for this purpose and that is why we chose to use it for the realization of the API.

The subportals will send requests to the profile subportal to get the particular user data (for detailed design of the API see section 4.2) along with the username of the user, whose user data they demand. The profile subportal will look this information up in its database and send the response back to the subportal.

3.3 Profile website

Profile website needs to fulfil the requirements listed in 1.3.1 and in 1.3.3. According to them we can divide the web pages of the profile website into two categories:

Userprofile web page This page will be created for each user (the URL will be: username.profile.matfyz.sk), where all their user data from the general category (see sec. 3.2.1) will be displayed. The user will be able to edit this data here.

Main profile web page Here the list of users, search options, sort options and list of the best, new etc. users will be shown.

3.3.1 User activities

In order to interconnect the subportals and strengthen the feeling that all the subportals are just parts of one big matfyz.sk portal, we have decided to collect the activities that user does on each subportal and display them on their userprofile web page. Activity can be, for example, that the user has posted a new article on blog. We decided to display these activities in the so-called "card design". This design is based on cards, where each card represents a bit of some content with the link to the rest of it. In our case, each activity, will be represented as one card, where a short explanation of that activity will be present. The link will lead to the activity itself on the subportal, where it happened. This helps to interconnect the portal as a whole, because when somebody is browsing the user profile page of some user and gets interested in a particular activity of this user, they can follow the link of that activity's card and so get to the other subportal.

There will be several types of activities on the profile. The administrator will be responsible for adding new ones and deleting no longer needed ones via form on the profile administration web page. Each activity type will have a source subportal, title of the activity and identification number assigned to it. When a subportal sends the information about some user activity, this id number will be added to this request and thus profile will know how to handle the activity. The mechanism of posting activities to profile will be realized via API described in 3.3.3.

3.3.2 Badges and achievements

Badges and achievements will have several types. Administrator will be responsible for adding new types of badges and achievements to the system via form on the profile administration web page. To each type of badge and achievement, the title, picture and description will be required.

The actual assigned badges and achievements will be also stored on the profile server and displayed on the user profile web page.

Badges will be given from one user to another by visiting the user profile web page of the user, to whom the badge want to be given. On this page a "give badge" button will be made visible for logged in users. They can click on the button and choose a badge they want to give from the list of badge types. The giver can also write a comment, why they chose to give the user this badge.

Achievements will be automatically generated by the subportals. When some user earned an achievement on some subportal, this subportal sends a message about it to the profile. For this purpose an API is described in the next section.

3.3.3 API for subportals to send information

In achievements and user activities the situation is similar to the one in 3.2.2 with the difference that here the subportals need to send some information to the profile rather than receive it. Still API is needed to offer this function. We will again use the

OAuth. Subportals will send requests to the post endpoint of the profile server with the information about the activity user did on this subportal, or an achievement. The form of the exchanged information need to be defined as well. For detailed design of the API and format of the data see section 4.2.

3.3.4 Users lists

On the main profile web page some user lists will be places. The *new users* list will consist of the fixed number of recently registered users.

Another list will be the *best users* list - Leaderboard (see 2.2.2). To measure who the best users are, a simple points system will be implemented. Every achievement and badge will be assigned a number of points. Upon receiving either of them, this number will be added to the current number of points of the user. The users with the highest number of points will be listed as the best users.

The last list on the page will be the *random users* list, where some number of randomly chosen users will be present.

3.4 Single sign-on

The profile.matfyz.sk subportal will be the main server, where the user logs in. The login form will be hosted here for users to fill in their credentials. After successful verification of username and password, profile will create a session for this user which times out, when the user is inactive for a certain amount of time. This session will be stored in the database as well, so that the profile knows all the time, which users are currently logged in. A subportal's task, when the user wants to log in, is to ask profile server for the authorization code (in order to get the access token). The standard behaviour of an Authorization server now takes place. If everything goes fine, the profile server stores the just issued access token to the database along with the `client_id` and so knows that the user will be logged in to this subportal. The `last_activity` (timestamp tied up to the session of the particular user, which says when the user was active for the last time during this session) of the user is also set to the current time, since the user is apparently active, although on the other subportal of matfyz.sk. Having received the access token, subportal will know which user has logged in to the system (or has already been logged in for some time), because this information will be sent along in the response of the access token. The response contains two additional parameters in comparison to the standard response described in 2.1.5:

`session_id` The identifier of the session, which the subportal is obliged to remember for the time of the session.

`username` The username of the logged in user.

The subportal will then create a local session for the user with the same lifetime as the token has. Better said: The subportal does not check whether the local session is still active, but whether the access token has not expired yet, since the local session (e.g. when using php sessions) could prolong its lifetime when the user is active automatically. Although we want to implement the functionality of automatic prolonging of the session, in the Single Sign-on this issue is more complicated, because we need to synchronize the session expirations on more subportals. We will go through this issue in detail in the next section.

3.4.1 Time out

Both the sessions (on profile and on subportal) will now (according to the state described in the previous section) time out at the same time, if the user is inactive. However, when the user is active, these two need to exchange information about it, since we want the user to stay logged in everywhere they have logged in, when they are active. In this case being active means to be active on either of the subportals. If the subportals and profile would not communicate about user's activity, the session on one of them could time out sooner than the other and the user would then no longer be logged in to the former one, which violates the principle of the single sign-on (the user switches back to the profile website after spending some time on another subportal and the session there has timed out although the user was active on the other subportal - this is undesired behaviour). Subportal needs to notify the profile server when the user is active there. There are two possibilities how this could be done:

1. Subportal sends notification *everytime* the user is active. This would probably result in too much unnecessary traffic, since every click user makes on the subportal, would cause a request to be made to the profile server. Profile server would afterwards have to distribute even more requests to notify other subportals where the user is logged in that the user was active, so that they can also prolong the lifetime of their sessions.

2. The notifications could be sent only *once in the given amount of time* to reduce the amount of requests.

The second option is obviously better for our purposes, since we do not need to know when exactly the user was active. We just need to time-out all the sessions on subportals and profile subportal at the same time. We will solve this problem with the help of refresh tokens (see Exchanging Refresh token of 2.1.5). We will set the default lifetime of the profile session and access tokens (and so the local session on subportals as well) to 2 hours. When some subportal detects that the user is active and it is less then an hour to the expiration of the access token, subportal will ask for a new one with the refresh token. This is the sooner mentioned “notification” to the profile server that the user is active. The profile successively notifies the other subportals where the user is logged in (the information, in which subortals the user is logged in, is acquired from the database of issued and not expired access tokens tied up with the particular user) to prolong their sessions as well. These requests are done to the requests_uri endpoint of the subportals. The following parameters will be sent in the HTTP entity-body of this request:

`request` Set to ”prolong”.

`session_id` Set to the id of the session, which should be prolonged.

`username` Username of the user, whose session should be prolonged.

After receiving the request to prolong the session, the subportals will ask for a new access token upon presenting the refresh token. However, we do not want an endless line of prolong requests to follow (after each refresh token request, all the other subportals are notified that the sessions should be prolonged and these subportals all send refresh token requests etc.). In order to prevent this the additional attribute `response`, set to “true” is added to the refresh token request, which gives server the information, that a new access token is desired only because the prolong request was received and not because the user was active on that subportal. By this whole mechanism all the subportals will be synchronized again to time out their sessions at the same time along with the profile server session of the user.

3.4.2 Single Logout-out

We need to implement the single log-out functionality as well. We agreed that we want a user to be able to logout either from only one subportal, or from all the subportals to which they have logged in. This will be implemented via redirection to the logout endpoint of the profile server. The user, who wants to logout clicks on the logout button (either on profile website itself or other subportal). If user is logging out of some subportal the `client_id` of this subportal is added as the last element of the path of the redirect uri. After redirection profile server asks the user, whether they want to logout of only this particular subportal (server knows the name of the portal from the URI) or from all the subportals to which they are logged in (these will be listed there). Depending on the user decision the logout requests are sent to the affected subportals to their requests uri with the following parameters in the HTTP entity-body:

`request` Set to “logout”.

`session_id` Set to the id of the session, which should be prolonged.

`username` Username of the user, whose session should be prolonged.

Having received this request, the subportal is obliged to destroy the particular session and so log the user out. The profile server destroys the access tokens issued for these subportals and if the “logout from all the subportals” option is chosen by the user, destroys the session on the profile as well.

Chapter 4

Data model and API

4.1 Data model

As proposed in 3.2.1, all the user data from the general category needs to be stored on the profile subportal. Apart from this data, also subportal-specific user data for the profile subportal and all the data needed for the SSO will be saved on the profile. For this purpose a data model has to be designed. We will introduce it in this section.

4.1.1 SSO data

We decided to use the oauth2-server-php library, which is a PHP library developed by Brent Shaffer and available on `https://github.com/bshaffer/oauth2-server-php`. This library implements the basic work flow of the OAuth protocol on which we can further build our application. There are 6 default tables used by this library, which we will need for our application. They all have the prefix oauth_ to easily tell them apart from the rest of the tables in our data model. Here only the parts, which were extended by us, or which are important for the further understanding are introduced, the rest can be found in the documentation to the library [13].

The `oauth_clients` table stores the information about the client - in our case subportal: `client_id`, `client_secret`, `redirect_uri`, `grant_types`, `scope`. To these default fields, we have added:

`home_uri` The URI, to the home page of the subportal, which will be used for the navigation. User will be redirected to this uri upon clicking on the subportal's button in the navigation bar on the website of the profile.

`requests_uri` As described in 3.1, this is the URI to which prolong and logout requests will be sent.

The next three tables are: `oauth_authorization_codes` - here the issued authorization codes are stored, `oauth_access_tokens` - here the issued access tokens are

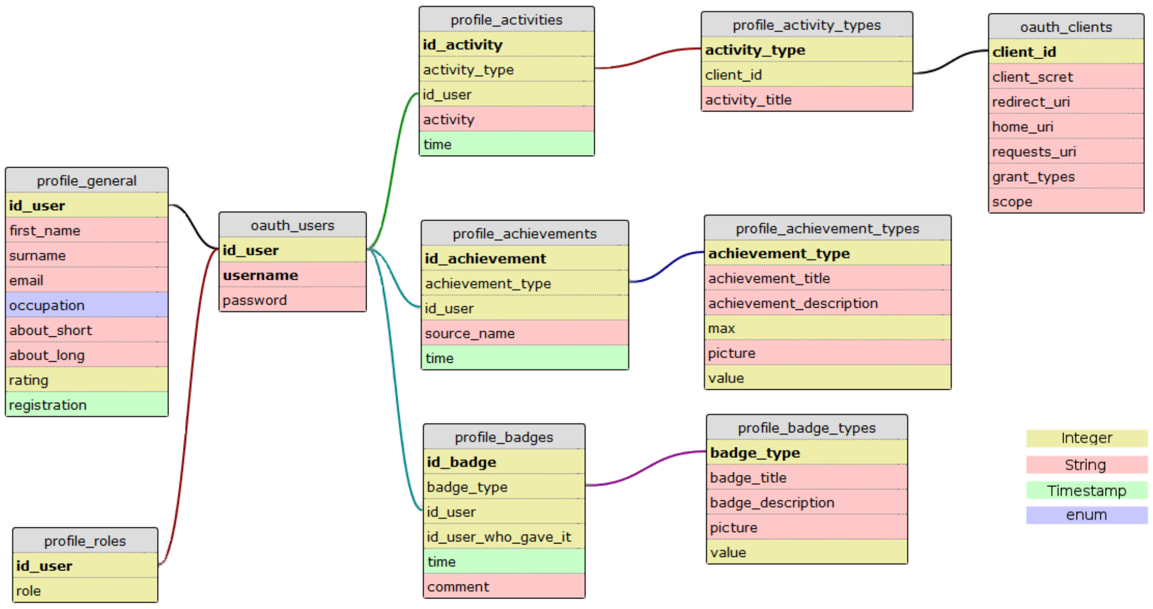


Figure 4.1: Data model of user data

stored, `oauth_refresh_tokens` - here the issued refresh tokens are stored. To every token (or authorization code) in all of these three tables the `user_id`, `client_id`, expiration time and scope are stored as well. The `oauth_scopes` table stores the information about defined scopes.

The last table is `oauth_users` where the `user_id`, `username` and `password` of the user are stored. The user authentication is not part of the library. That is why we were free to implement it our way. We chose to store passwords in an encrypted form (we will store only the hash of the actual password) to increase security using bcrypt [10] key derivation function for passwords. The user credentials are thus stored only in this table and they are also verified against it. The rest of the data will be stored in the tables, where the `user_id` will be the key to link the data to the particular user.

4.1.2 User data

The tables, which store user data and do not belong to the SSO solution will have the prefix profile_ to easily tell them apart. On the Fig. 4.1 is the complete data model of the profile without previously mentioned oauth_ tables. Only two of them (`oauth_users` and `oauth_clients`) are present in the figure, since they contain foreign keys for the other tables with user data. The individual tables will be introduced in this subsection.

The user data stored in the profile database can be divided into two groups: general and subportal-specific for the profile subportal:

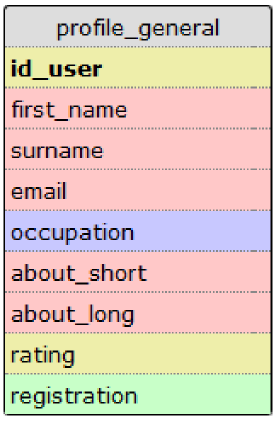


Figure 4.2: profile_general table

4.1.3 General user data

The general user data, is the data which will be shared across the whole portal. Special table called `profile_general` will be used for this purpose. The primary key is the `id_user` which is a foreign key, referencing the `id_user` field in the `oauth_users` table. The `occupation` field is an enum, with three possible options: student, teacher and other. Other value is provided because matfyz.sk is an open community portal not only for students and teachers, but for anybody interested in our faculty. The `rating` field contains the sum of the points collected from the received badges and achievements. According to this value a list of users will be put together. The `registration` field contains the timestamp when the user registered to the system. The other fields are self-explanatory and are shown on the Fig. 4.2.

4.1.4 Subportal-specific user data

The rest of the data on profile has to do with the other functions of the profile subportal. The structures of badges and achievements are similar to each other - see Fig. 4.3.

Badges and achievements

We will distinguish two terms: reward type (either achievement or badge type) and the actual reward assigned to some user (achievement or badge). Each reward type has title, description, assigned value and picture. Achievement type has additional attribute - maximum amount of this particular achievement type one user can obtain. Two tables called `profile_achievement_types` and `profile_badge_types` deal with this data. Each added reward type is assigned a number, which serves as a foreign key for the other two tables: `profile_achievements` and `profile_badges`, where the actual rewards will be stored. These two tables are slightly different from each other. The `profile_achievements` table contains following fields: `id_achievement` - the primary key, and id of this particular achievement, `achievement_type` - the

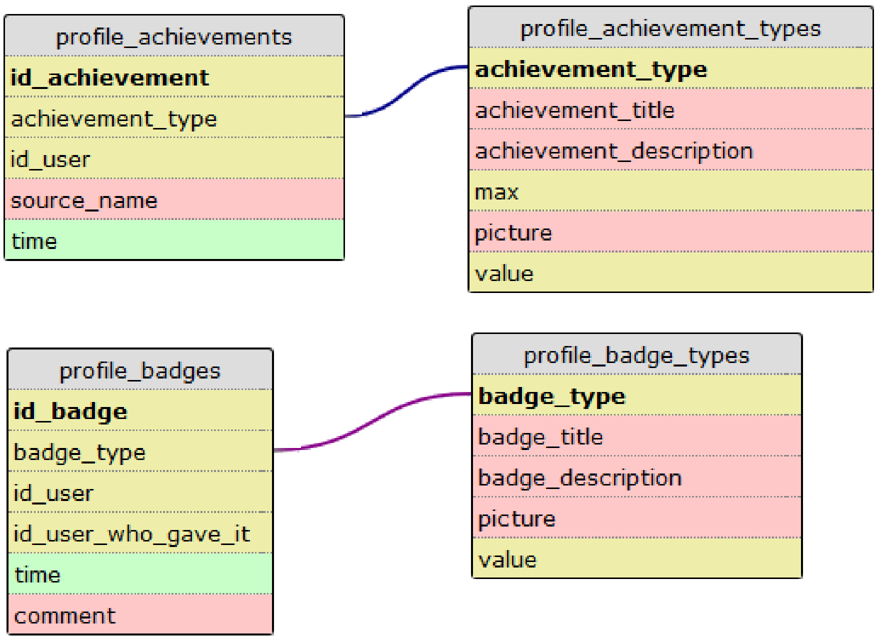


Figure 4.3: Rewards datamodel

foreign key referencing the `achievement_type` from the `profile_achievement_types`, `id_user` - foreign key referencing the `id_user` of the user to whom the achievement was given, `time` - timestamp saying when the achievement was assigned and `source_name` -name of the course or subportal on which the user has earned the achievement. This field is left up to the giving entity to fill in their name as they please. The `profile_badges` table has fields, which have the same function as the corresponding ones in the `profile_achievements` table: `id_badge`, `badge_type`, `id_user`, `time` and additional two: `id_user_who_gave_it` which is a foreign key that references the `id_user` of the user, who gave this badge and `comment` field, where the comment from the giver can be added.

Activities

As introduced in 3.3.1, the activities of the user on the whole portal will be displayed on the profile. We designed a similar model to the badges and achievements model as can be seen on the Fig. 4.4.

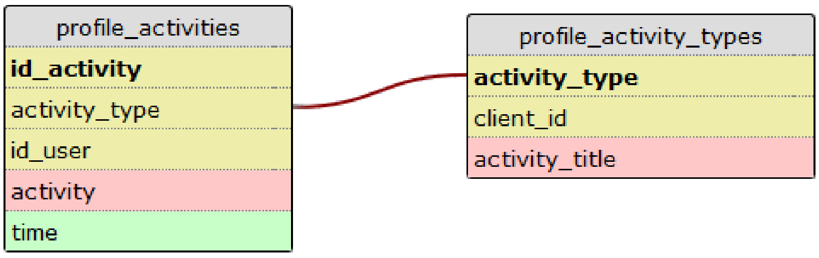


Figure 4.4: Activities data model

In the table `profile_activity_types` we will store the types of activities which exist on the subportals. For example on the blog subportal an activity "written articles" may exist. This means that a user can make an activity (in this example, post an article) on the subportal and this information will be then passed to the profile. The table contains three fields: `activity_type` - the id of this activity type, `client_id` - foreign key referencing the `client_id` of the subportal on which this activity exists and a `activity_title` - the title of the activity. The actual performed activities will be linked to these types.

The subportals will generate an HTML code, in which the more detailed description of the activity will be present. This HTML code must adhere the agreed form. The code consists of `div` elements, each with the following `id` attributes:

`ac_title` REQUIRED The title of the activity, preferably with a hyper-link to the actual activity on the subportal (nested in the `div` element as an `a` element)

`ac_picture` OPTIONAL In this `div` an `img` element with the picture corresponding to the activity can be added.

`ac_description` REQUIRED A closer description of the activity or short extract.

These activities will be stored in another table `profile_activities` (see Fig. 4.4) with the following fields: `id_activity` - the primary key and the id of the individual activity, `activity_type` - the foreign key referencing the type of the activity in the `profile_activity_types` table, `id_user` - foreign key referencing id of the user who did the activity, `time` - timestamp of the time when the activity was performed and a field called `activity` - the HTML code with the description of the activity as described above.

Roles

We needed to add some permission mechanism to distinguish which users are allowed to administrate the profile website, add badge, achievement and activity types, select best users, etc. For this purpose the `profile_roles` table was designed with only two fields: `id_user` - foreign key referencing the id of the user and `role` - integer value, which specifies what permission the user has. However till now only the distinction between administrator and standard user was needed, so the administrators are added to the table with the `role` value "1". If we need more permission levels in the future, the scale can be easily extended.

4.2 API model

If we want to collect the user related data in one place (on the profile subportal), we need to enable the other subportals to access and update these data. In order to provide them with this possibility two APIs will be made available for the subportals. One for receiving information about the users and one for sending information about them. Both will utilize OAuth protocol, which is described in 2.1.5. The subportals will be able to get and send the data upon presenting valid access tokens. The valid access token will be in this case acquired via Client credentials grant type, where the access token is issued to the subportal upon presenting, apart from other information, their `client_id` and `client_secret` to the token endpoint of the profile server.

4.2.1 API for subportals to get the information

The subportals need to access all the user data stored on the profile subportal. We have analysed in what kind of forms the subportals need to get the information. We have come up with the conclusion that it may vary from single strings, like first name or e-mail address of the user, to more complex data like HTML code with the short profile of the user, which can be placed on the destination subportal. To facilitates all these requirements, we have designed the following endpoints:

`get_firstname` returns the first name of the user

`get_surname` returns the surname of the user

`get_whole_name` returns the whole name of the user in one string

`get_photo` returns the URI, where the picture of the user is accessible

`get_about` returns the short version of the about - about_short

`get_email` returns the e-mail address of the user

`get_is_student` returns TRUE if the user is student, otherwise returns FALSE

`get_is_teacher` returns TRUE if the user is teacher, otherwise returns FALSE

`get_profile` returns an HTML code with the user profile, which can be placed on the destination website. The form of the code is following:

```
<div id="userprofile">
<a href="path to the userprofile">
<h3>Whole name of the user</h3>
</a>
<img src="path/to/the/picture" width=250 />'.
```

```
<p>short version of about</p>
</div>
```

`get_profile_short` returns the small version of the profile of the user, where only name and picture is present. The form of the code is as follows:

```
<div id="userprofile_short">
<img src="path/to/the/picture" width=50 />
<a href="path to the userprofile">
<h5>Whole name of the user</h5>
</a>
</div>
```

The requests for the information will be realized via POST requests to the wished endpoint of the profile server. The URI to which the request should be sent will be constructed from three segments as follows: `profile.matfyz.sk/resource/endpoint/username`, where endpoint is the desired endpoint and username is the username of the user, whose user data the subportal wants to obtain. The parameters of the request are encoded into the HTTP entity-body and are following:

`client_id` The id of the subportal, which is sending the request

`redirect_uri` The URI on the subportal to which the response should be sent

`access_token` The valid access token to prove the right to obtain the requested data

When the request is verified (when the access token is valid) the response is sent to the `redirect_uri` from the request. The response contains a `message` parameter encoded into the HTML entity-body, which contains the return value as described above. When the request is not verified or when it does not contain required parameters the error response is sent as described in 2.1.5.

4.2.2 API for subportals to send the information

The subportals need to send profile information about user activities, earned achievements and badges. The API works similarly to the API 4.2.1. Again requests are sent to the endpoints with the POST method and the parameters are encoded into the HTTP entity-body of the request. These parameters are `client_id`, `redirect_uri` and `access_token`. The URI consists of three segments: `profile.matfyz.sk/post/endpoint/username`, where endpoint is one of the below listed available endpoints and username is the username of the user to whom the information in the request belongs. To each endpoint the required additional parameters will be listed:

`activity` The required additional parameters are:

`activity_type` the type of the activity the user has done

`timestamp` timestamp of the time when the activity was performed

`message` the HTML code of the performed activity. The code must adhere to the following format:

```
<div id="ac_title">
The title of the performed activity -
a hyperlink to the actual activity is recommended
</div>
<div id="ac_description">
The description of the activity -
hyperlinks are recommended
</div>
<div id="ac_picture">
this is an optional div element,
which may contain an img tag with a hotlink
to the picture describing the activity
</div>
```

`achievement` The required additional parameters are:

`achievement_type` the type of the achievement we want to give to the user

`timestamp` timestamp of the time when the achievement was earned

`source` the name of the subportal or course, which gives the achievement - this parameter is left to be filled out freely by the subportal

When the request is not verified the error response, as described in 2.1.5 is sent. When the additional parameters are not present, the response is sent with the `status` parameter set to the description of the error. When the request is valid and the required additional parameters are present the response with the `status` parameter set to “OK” is sent.

Chapter 5

Implementation

5.1 Technologies

PHP

For the implementation, we chose to develop in PHP [9], which is a an open source general-purpose scripting language, widely used for developing interactive web applications. The PHP code is embedded into HTML, but the execution of the code is done on the server and only the outputted HTML is sent to the client. Therefore the client does not know what the code which had generated the particular output was.

Codeigniter

There are many available frameworks built on the top of the PHP language, which facilitates the programming of the web applications, as they offer generic functionality, which can be reused and expanded to create new software. Since our project is bigger web application we opted for using some framework. Some of the other subportals of matfyz.sk (e.g courses [15] [1]) are developed in Codeigniter framework [4]. That was the reason we also chose to use Codeigniter. This framework provides a rich set of libraries for commonly needed tasks in web applications. It is based on the MVC (Model-View-Controller) design pattern. MVC is a software pattern used for user interfaces. It divides the application into three interconnected components: Model, View and Controller. Model represents the data of the application and provides interface to access it for the controller. View is the output representation of the model, presented to the user. Controller serves as the link between user and the system; it collects the input from the user, satisfies their requirements, with the help of the model, and passes the output to the view, which presents it to the user. The Codeigniter framework follows this structure, and so will we.

MySQL

For storing data, we chose to use the MySQL open source relational database management system. The queries, run against the database, are written in SQL - Structured Query Language. It is most widely used solution for non-profit web projects and the integration between Codeigniter and this type of database is also very well.

CSS

Cascading Style Sheet (CSS) is a style sheet language, which is used for formatting and graphical presentation of the markup language document. The matfyz.sk portal is formatted in this language and for our project we will use and extend[1] the design developed by Roman Janajev as a part of his Master thesis [6].

5.2 The subportal profile.matfyz.sk

The whole subportal is written in Slovak language, therefore also the screenshots in this chapter will be in Slovak language. The localization to the English language will be done in the future in order to make the whole matfyz.sk bilingual (for the time being blog portal is already bilingual). For better orientation and understanding of the screenshots, the Slovak equivalents of English terms will be put in the parentheses.

On the top of each web page on profile subportal, a header with the links leading to all the subportals of matfyz.sk is placed along with the possibility to either log in, and register (when nobody is logged in - see Fig. 5.1), or to log out and the name of the currently logged in user.

Every user has a subdomain in the form of username.profile.matfyz.sk. However, these are only aliases for the profile.matfyz.sk, so the controllers have to resolve the URI by themselves to find out whose subdomain was called.

We will now, step by step, describe each controller we used in our project, since they are the elements which are accessible from outside and thus are good to demonstrate the implemented structure. The associated views and models will be described alongside them. In Codeigniter, the controllers are accessed via their name, written as the first segment of the resource part of the URI. The second segment indicates the function of the controller which will be called. When no second segment is present, the index function will be called.

In this section we will use the term “visitor”, when referring to the visitor of the profile subportal.

[1]Some of the used icons are created by Freepik from www.flaticon.com under the Creative commons licence.

5.2.1 Homepage of profile.matfyz.sk

Profile controller

This controller is responsible for the homepage of profile.matfyz.sk (see Fig. 5.1[2]) along with the `profile` model and view. The page (presented via the view) is divided into two columns. Left column serves as the navigation for searching users. The search box is placed on the top, where the name, username, or some part of them can be written and the corresponding user will be found. The users can be also sorted alphabetically according to their surname ("podľa priezviska"). The last search option is to sort users according to their rating ("podľa ratingu"). In the right column, by default, the three lists of users are shown (the central part of the web page). First list is a leaderboard consisting of the 10 users with the highest rating ("Používatelia s najvyšším ratingom"). The next is the list of the 5 newest users according to their registration time. And the last one consists of the 5 randomly chosen users.

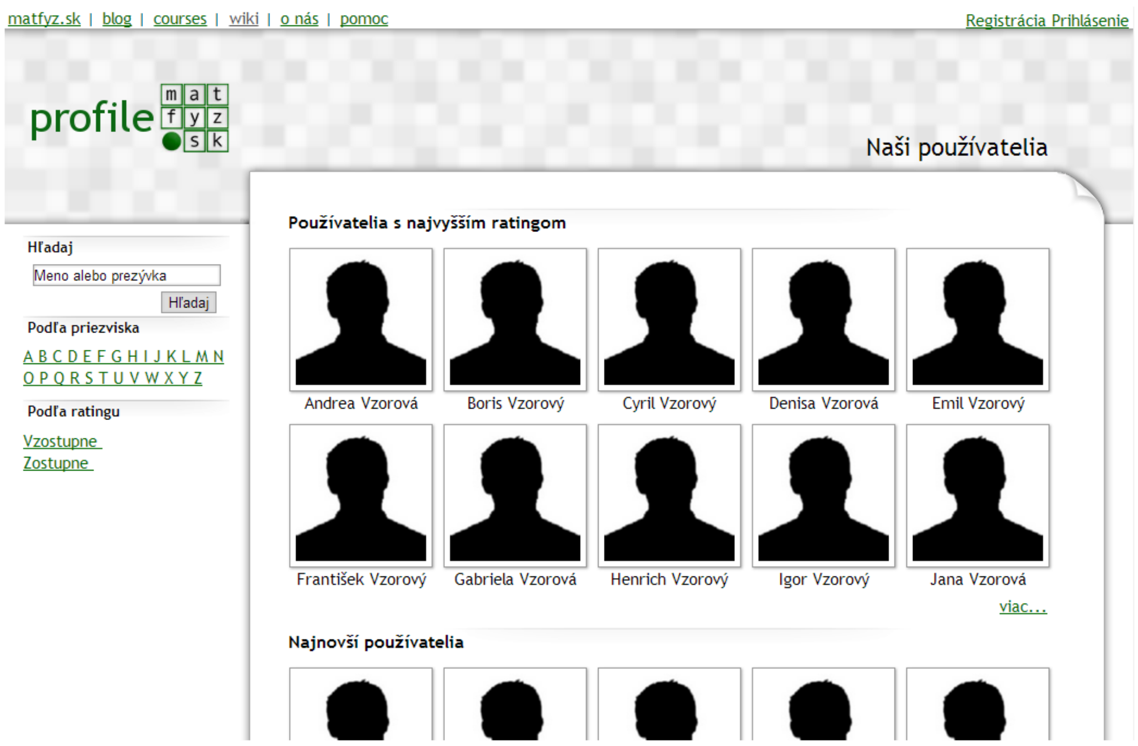


Figure 5.1: Homepage of profile.matfyz.sk

5.2.2 User profile

There are two controllers directly related to the user profile: `userprofile` controller and `edit_userprofile` controller.

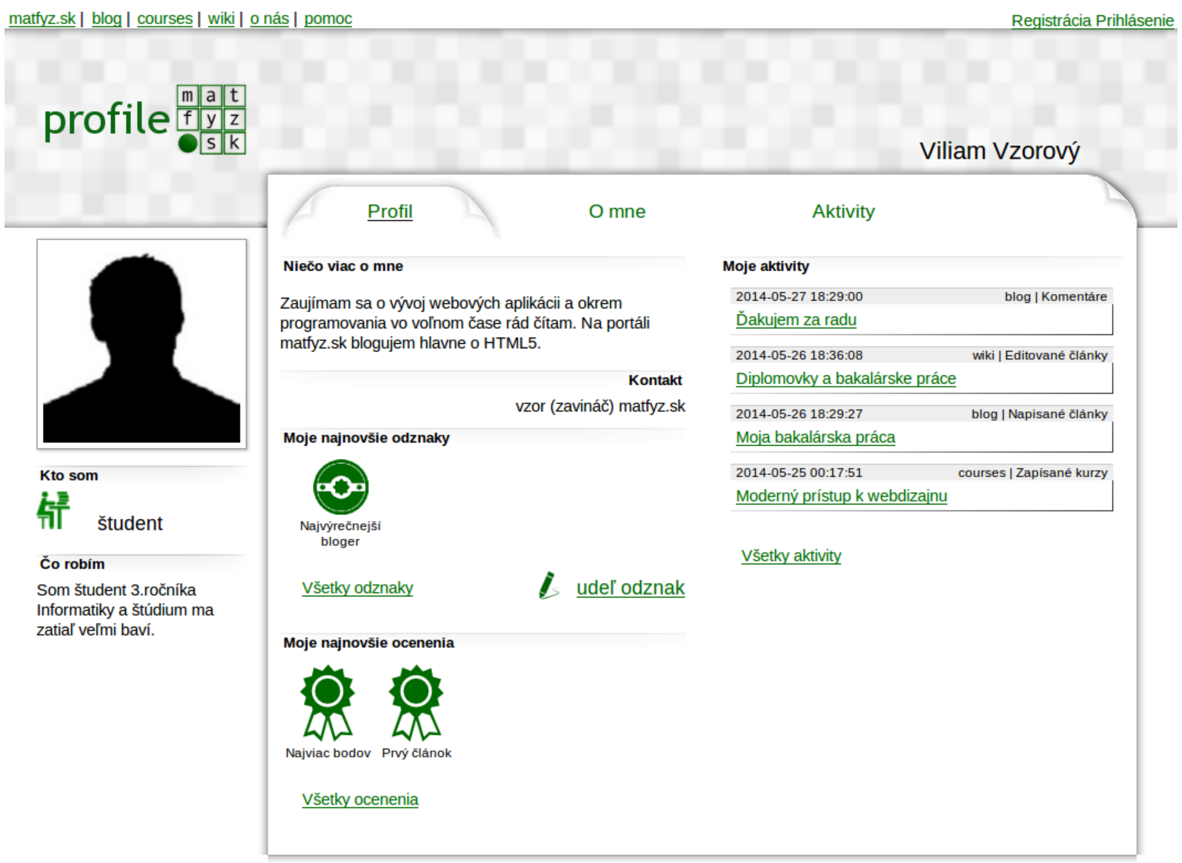
[2]The users and their avatars are fictional and only for illustration purpose

Userprofile controller

The `userprofile` controller resolves the URI to find out whether some user profile should be loaded. When there is no username in front of the .profile in the URI, the `profile` controller is called. If some username is found, the respective profile is loaded and presented to the visitor of the web page with the help of the `userprofile` model and view. The web page (presented via the view) consists of two columns. In the left column, the picture of the user, their occupation and short version of the about (short text, the user has filled in about them) is present. When the currently logged in user browses their own user profile, the link to edit the user data is also accessible.

In the right column, the main content of the web page is present, which is divided into three parts, represented as tabs, among which the visitor can switch:

Profile ("Profil") This tab is loaded as a default option after the user profile web page is visited. It is the summary of the other two tabs. In the left column, the longer version of about and email address are present and the last 6 earned rewards are exhibited. However, only the small pictures and titles of the achievements ("ocenenia") and badges ("odznaky") are shown here, but both reward types sections include a link to all the rewards ("Všetky") which takes user to the next tab - About ("O mne"), where the rewards are exhibited more in detail.


matfyz.sk | blog | courses | wiki | o nás | pomoc

Registrácia Prihlásenie

profile m a t f y z s k

Viliam Vzorový

Profil O mne Aktivity

Kto som

študent

Čo robím

Som študent 3.ročníka Informatiky a štúdium ma zatiaľ veľmi baví.

Niečo viac o mne

Zaujímam sa o vývoj webových aplikácii a okrem programovania vo voľnom čase rád čítam. Na portáli matfyz.sk blogujem hlavne o HTML5.

Kontakt

vzor (zavináč) matfyz.sk

Moje najnovšie odznaky

Najvýrečnejší bloger

Všetky odznaky

udeľ odznak

Moje najnovšie ocenenia

Najviac bodov

Prvý článok

Všetky ocenenia

Moje aktivity

2014-05-27 18:29:00 blog | Komentáre
Ďakujem za radu

2014-05-26 18:36:08 wiki | Editované články
Diplomovky a bakalárske práce

2014-05-26 18:29:27 blog | Napísané články
Moja bakalárska práca

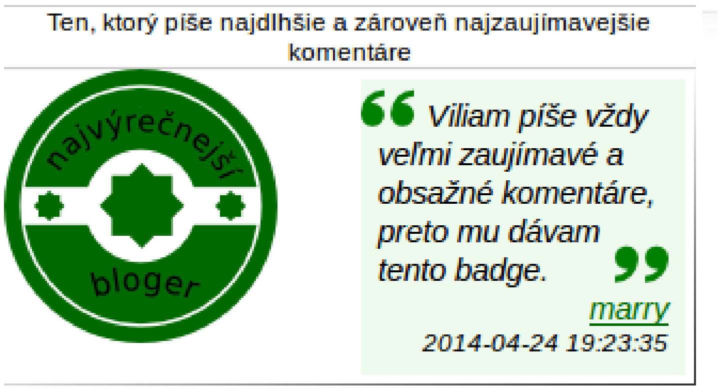
2014-05-25 00:17:51 courses | Zapísané kurzy
Moderný prístup k webdizajnu

Všetky aktivity

Figure 5.2: Userprofile

Also the link to give a badge to the user ("udeľ mi odznak") is shown above the earned badges, which leads the visitor to the `give` controller (described later in this section), where the visitor (when logged in) can award the user with some badge. The described parts and the general layout of the user profile web page is shown on the Fig. 5.2

About ("O mne") Here the long version of about and email address is shown. All the earned rewards are exhibited with large pictures, on which the title of the particular reward is written. The description and the time when the reward was earned are present. Next to the achievement, the source of it is stated. Next to the badge, the comment and the username of the user who had given this badge is written (see Fig. 5.3). Also the link allowing to give a badge is present.



Ten, ktorý píše najdlhšie a zároveň najzaujímavejšie komentáre

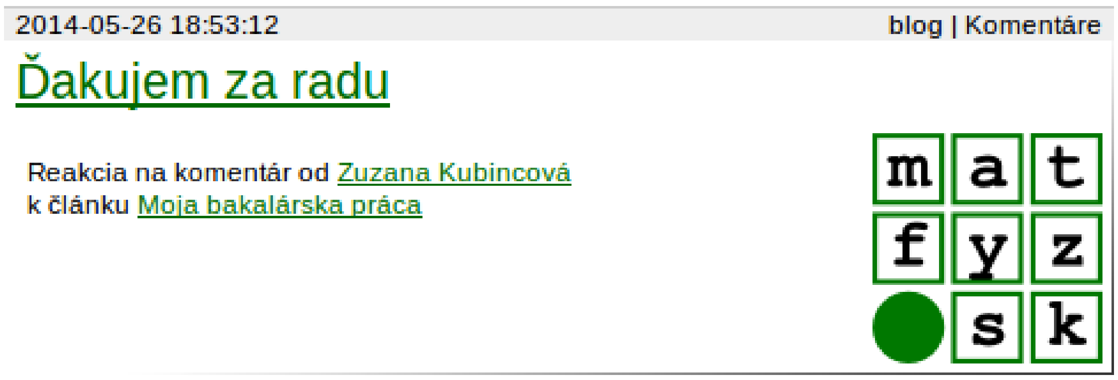
"Viliam píše vždy veľmi zaujímavé a obsažné komentáre, preto mu dávam tento badge."

marry

2014-04-24 19:23:35

Figure 5.3: Badge - detail

Activities ("Aktivity") Here the detailed view of each user activity is shown as one card. The card consists of the time, type of activity, subportal, title, description, and optionally a picture, as can be seen on Fig. 5.4.



2014-05-26 18:53:12 blog | Komentáre

Ďakujem za radu

Reakcia na komentár od Zuzana Kubincová
k článku Moja bakalárska práca

Figure 5.4: Activity card

Edit_userprofile controller

The controller resolves the URI and when the username in the URI matches the username of the currently logged in user, the `Edit_userprofile` controller gives user the opportunity to change their information stored in the system. In the left column, a form for changing the password is located. In the main part of the web page, the forms for changing the avatar picture and changing the other information is placed. The actual change of the information in the database is done via `membership_model`, which provides functions to do so.

Give controller

The `give` controller resolves the URI and when username is present, this controller enables logged in visitors to give the user, on whose subdomain they are, badges. The form with existing badge types as a radio option list is in the central part of the web page along with the text area to place a comment. Upon submitting the form the `userprofile_model` is called to store the just given badge to the user.

5.2.3 Administration

For viewing the administration web pages, the user needs to be logged in and have a record in the `profile_roles` table in the database with the value “1” in the role field. Only after checking these two conditions, the content of these pages is loaded. Otherwise, the user is redirected to the `not_permitted` controller.

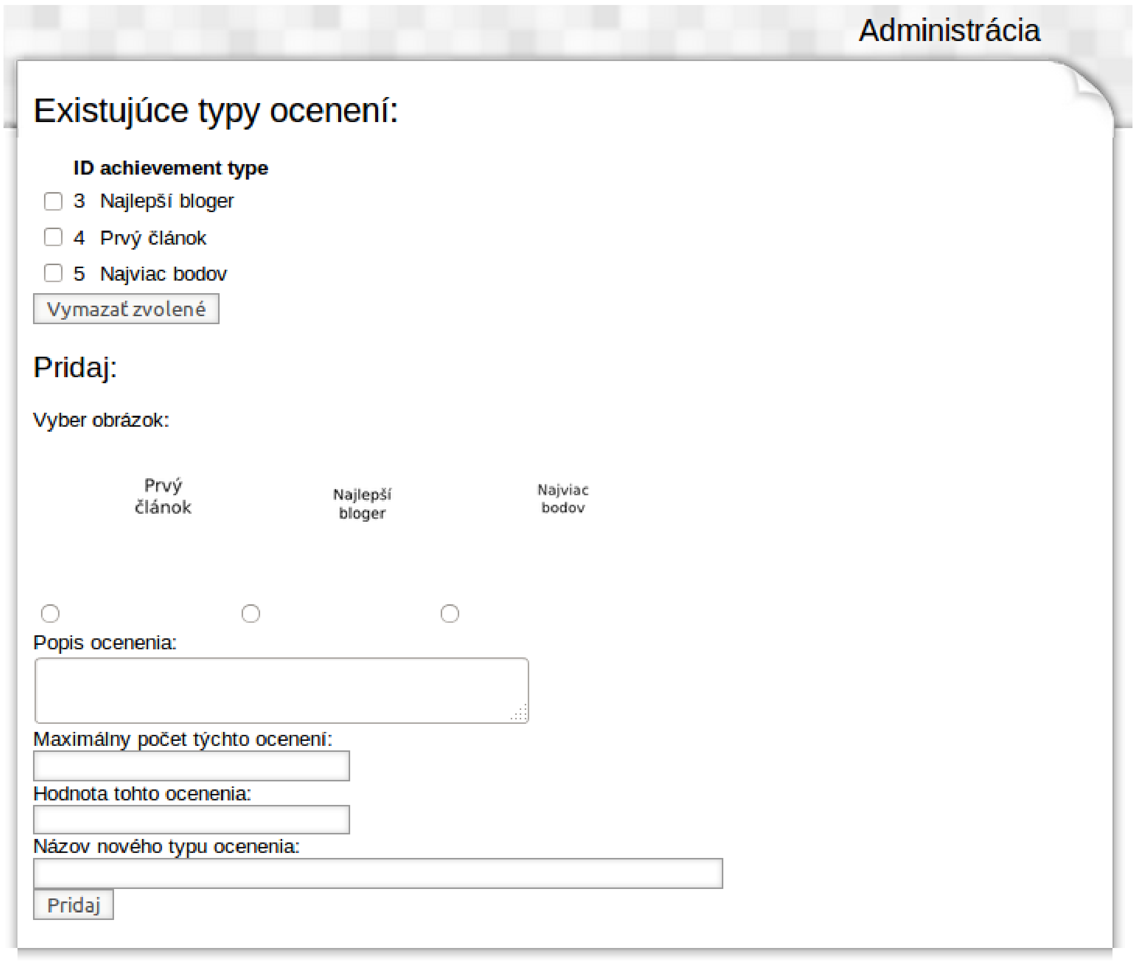
Edit controller

Here the administrator have the opportunity to alter or add content to the profile subportal. The links in the list, each adds the second segment to the URI, which specifies which function of the `edit` controller should be called. The links are :

subportals Upon clicking, the `clients` function of the controller is called (because subportals have the clients role in the OAuth protocol - see Roles in 3.1). In this function the `edit_view`is called, which presents the form in which the information about the new client can be filled in. The submitted data is stored into the database with the help of the `edit_model`.

achievements Upon clicking, the `achievements` function is called, which shows the form (with the help of the `edit_view` and `edit_achievements_view`), in which the existing types could be deleted or a new type can be added.

badges The same as in the previous link, just the called function and used view are called differently: `badges` function and `edit_badges_view`.


Administrácia

Existujúce typy ocenení:

ID	achievement type
☐ 3	Najlepší bloger
☐ 4	Prvý článok
☐ 5	Najviac bodov

Pridaj:

Vyber obrázok:

Popis ocenenia:

Maximálny počet týchto ocenení:

Hodnota tohto ocenenia:

Názov nového typu ocenenia:

Figure 5.5: Editing of the achievement types

activities Upon clicking, the `activities` function is called, which loads the `edit_activity_view`. This view presents a form where, similarly to achievements and badges, the existing activity types can be deleted or a new one can be added.

Not_permitted controller

This controller loads the view which says that the user does not have a permission to view the requested page. The user is redirected to this controller upon requesting a page for which they lack the appropriate permissions.

5.2.4 Single sign-on

We have already mentioned in 4.1.1, that we chose to use the oauth2-php-server PHP library for the OAuth implementation. This library is listed as one of the available PHP implementation on the official website of the OAuth community [5]. For sending the HTTP requests described in 3.1, we chose to use the Guzzle PHP HTTP client, developed by Michael Dowling [3], which contains HTTP adapters, by which the requests can be sent easily.

First, we will describe more in detail how the login and logout of the user looks like, and than we will describe the mentioned controllers.

The detailed workflow of the login

When user visits any subportal and wants to log in, they click on the login button. To log the user in, subportal creates the request for an authorization code, as described in 2.1.5 to the `profile.matfyz.sk/authorize` endpoint, which calls the `authorize` controller. The user-agent is then redirected to the profile server alongside the request. The authorize controller takes the request and validates it. If the request is found acceptable the process of user authentication is initiated:

1. Check if there is some session opened and whether some user is logged in. If true, go to step 5.
2. If no user is logged in, redirect to `profile.matfyz.sk/login` - `login` controller is made. The return URI to the authorize endpoint alongside with the request made by the client (encoded into this URI) is encoded as a parameter into this redirect URI.
3. The user is here prompted to fill in their username and password to authenticate themselves.
4. If successfully authenticated, the user is redirected back to the return URI - `authorize` controller.
5. The `authorize` controller revalidates the request (to check if it was not changed).
6. If the request is not valid, the error response is sent. If the request is valid, the authorization code is issued and sent back to the client's `redirect_uri` with additional parameter `username` (as an extension to the normal response described in 2.1.5). Where `username` is the username of the currently logged in user. The `session_id` is also sent as a parameter, to bind the sessions on clients to the session on profile server.

The detailed workflow of the logout

When the user wants to logout, they click on the logout button in the subportal from which they want to log out. The subportal redirects the user-agent to `profile.matfyz.sk/login/logout/subportal`, where "subportal" segment is the `client_id` of the subportal from which the client wishes to logout. The logout process is then initiated:

1. The `login` controller's function logout is called.

2. The controller checks whether the third segment is present in the URI and if yes, than it asks the user whether they want to logout only from this subportal or from all the subportals.

3. User clicks on the link according to their decision.

4. Depending on the choice of the user, the controller either sends the logout requests with the help of the Guzzle client, to all the subportals to which the user is logged in and destroys the session on profile as well or it sends the logout request only to the one particular subportal, leaves the session on profile server untouched, and redirects the user back to the subportal from which they just logged out.

5. When the user wished to logout from all the subportals, they are redirected to the index function of the `login`. Otherwise, they have already been redirected it the previous step.

Login controller

The web page which is viewed upon calling this controller consists of two columns. In the left column, the small lists with the pictures of the best users are shown. In the right column, the main content of the page is placed. On Fig. 5.6 on the right, the login form is placed and on the left, the best users lists is located [3]

The login controller checks whether some user is logged in or not. When no user is logged in, it shows the login form with the possibility to log in or create an account, as on the Fig. 5.6. In the opposite case, the controller loads a view which says that the user is already logged in. When user fills in their credentials and tries to log in, the `membership_model`'s function `validate_credentials` is called, which checks whether the username-password combination is present in the database. If the answer is positive, the controller creates the session for the user.

When the signup is present as the second segment of the URI, the `signup_form` view is loaded, where a form to fill in the required information for the registration of a new user is presented.

When logout is present as the second segment of the URI, the controller's `logout` function checks whether some user is logged in, and if yes, it asks the user whether they want to logout from all the subportals (or from one particular subportal, when the third segment of the URI is present). Upon clicking on the button, the logout process continues as described above.

[3]The users and their avatars are fictional and only for illustration purpose

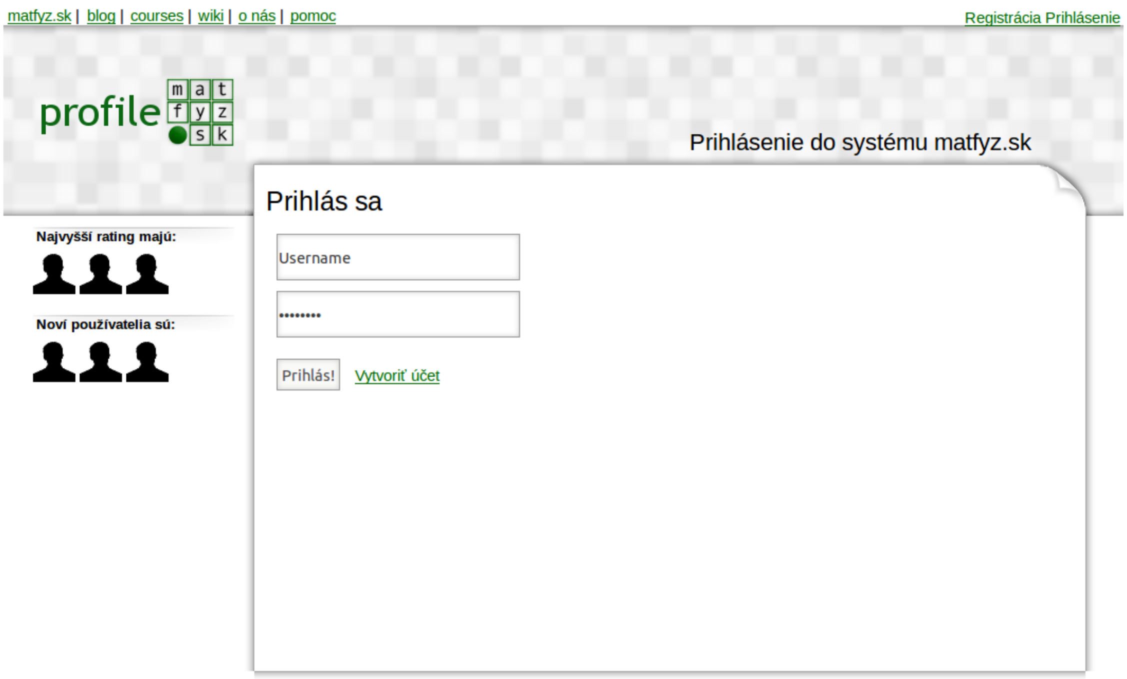


Figure 5.6: Login form

Authorize controller

This controller serves as the authorize endpoint for the OAuth protocol and thus its behaviour is described in 3.4. It uses `token_model` to manipulate the data associated with the tokens, authorization codes etc. Upon issuing an authorization code, it stores the `session_id` of the current session of the user who is logged in to this code, so that it can be stored to the respective token as well. The `session_id` needs to be stored in the database, since tokens are issued via requests and not redirection. The controller thus does not have an access to the `session_id` stored in the cookie. Without the record in the database, the controller would not know what the `session_id` was.

Token endpoint

This controller serves as the token endpoint for the OAuth protocol and thus the behaviour is descibed in 3.4. When the request is received by this controller, it checks what kind of a request it is and whether it is valid. If the request is valid, it issues the access token and sends back the HTTP response. With the help of the `token_model` it stores additional information to the token to the database - the `session_id` (if the authorization code grant type was used). When the grant type was refresh token, it checks the `response` parameter of the request and if it is not present or is set to “false”, the controller sends the prolong requests to other subportals with the help of Guzzle.

5.2.5 API

For the implementation of the API, designed in 4.2.1 and 4.2.2 we have created two controllers: `post` and `resource`.

Resource controller

The `resource` controller serves the purpose of the API to get the information. It contains the functions as designed in 4.2.1, which are called via URI, where the second segment is the name of the function the subportal wants to call. The third parameter is the username of the user, whose data the subportal wants to access. In the request, also the valid access token must be attached as the parameter. The controller checks whether the token is valid, and if it is, than it constructs the designed answer with the help of the `userprofile_model`. This answer is sent as a response via Guzzle.

Post controller

The `post` controller functions similarly to the `resource` controller. It provides the subportals with the functions described in 4.2.2 - achievement and activity. The functions are also called via URI, in which the name of the function is written as the second segment, and the username of the user to whom the sent data should be assigned is the third parameter. Again the valid access token must be present, so that the controller can check whether the request is valid and relevant. If the request is valid, the sent data is stored to the database with the help of the `userprofile_model` and the response with the `status` parameter set to “ok” is sent back to the subportal.

Conclusion

In our thesis we dealt with two main topics: the administration of user data on matfyz.sk portal and single sign-on solution for the portal.

Firstly, we have analysed the condition of matfyz.sk portal - above all, the user data storage. We have divided the data into two categories: General and Subportal-specific user data. General user data is the data shared across all the subportals and thus we moved this data to the new subportal *profile.matfyz.sk*. Now it is stored only in one place and other subportals access the data via application programming interface, which we have designed and described in this thesis. This API also enables subportals to post information about user activities, which are then stored and shown on the user profile web page. The rewarding mechanism for users was also designed - the users can now earn badges and achievements by being active on the portal, which, we hope, will increase their interest and engagement in the portal. For storing of all these information about users, the data model was designed. Each user is now assigned a subdomain *username.profile.matfyz.sk*, where the personal information, rewards, and activities of the user are shown; the profile thus consists of pieces of content from all the subportals and serves the purpose of unifying the whole portal.

In order to take over the logging of users in and out of the whole portal, we have first analysed the available solutions for the single sign-on for web sites, and then chosen the OAuth 2.0 protocol. We have designed the changes and additions to the protocol to fit the needs of matfyz.sk portal and implemented it.

All the designed parts have been successfully implemented and are available on the `profile.matfyz.sk` as a part of the portal matfyz.sk. The source code is also available on the attached CD.

This thesis creates the design and implements the user profile with all the mentioned features. However, the list of user-related information could be expanded and many additions to the current condition of the profile subportal could be proposed and added in the future.

Bibliography

[1] Adam Bilisics: *Electronic Notes*, Bachelor Thesis, Comenius University in Bratislava, 2014.

[2] Sebastian Deterding et al.: *From game design elements to gamefulness: defining gamification*, in: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, ACM, 2011, pp. 9–15.

[3] Michael Dowling: *Guzzle*, 2014, URL: `http://guzzlephp.org`.

[4] EllisLab, Inc.: *Codeigniter/EllisLab*, 2014, URL: `http://ellislab.com/codeigniter`.

[5] Dick Hardt: *The OAuth 2.0 authorization framework*, RFC 6749, Internet Engineering Task Force, Oct. 2012, URL: `http://tools.ietf.org/html/rfc6749.html`.

[6] Roman Janajev: *Customizable and Usable Layout of blog.matfyz.sk Portal*, Master Thesis, Comenius University in Bratislava, 2014.

[7] matfyz.sk team: *about*, Nov. 2007, URL: `http://www.matfyz.sk`.

[8] OAuth: *OAuth:Introduction*, Sept. 2007, URL: `http://oauth.net/about`.

[9] PHP Community: *What is PHP?*, cited on 15.5.2014, URL: `http://www.php.net/manual/en/intro-whatis.php`.

[10] Niels Provos, David Mazieres: *A Future-Adaptable Password Scheme.* In: *USENIX Annual Technical Conference, FREENIX Track*, 1999, pp. 81–91.

[11] Dave Raggett, Arnaud Le Hors, Ian Jacobs, et al.: *HTML 4.01 Specification*, in: *W3C recommendation* 24 (1999).

[12] Regents of the University of Michigan: *CoSign: Collaborative single sign on*, Aug. 2012, URL: `http://weblogin.org/`.

[13] Brent Shaffer: *OAuth 2.0 Server PHP*, URL: `http://bshaffer.github.io/oauth2-server-php-docs/`.

[14] Shibboleth: *How Shibboleth Works*, 2014, URL: `https://shibboleth.net/about/basic.html/`.

[15] Jakub Čulík: *Integrovaný LMS systém*, Bachelor Thesis, Comenius University in Bratislava, 2013.

[16] W3C: *Same origin policy*, Jan. 2010, URL: `http://www.w3.org/Security/wiki/Same_Origin_Policy`.

[17] Gabe Zichermann, Christopher Cunningham: *Gamification by design: Implementing game mechanics in web and mobile apps*, O'Reilly Media, Inc., 2011.