FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Academic Collaboration Information System
Student:	Bc. Petr Jirásko
Supervisor:	Ing. Marek Suchánek
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2019/20

Instructions

The aim of the thesis is to create an easily extensible information system supporting scientific research activities within academic institutions internally as well as externally (with partners from other institutions). The system must support the whole life-cycle of collaborative production of various types of publications.

- Analyze types of scientific publications and the process of their preparation and creation up to publishing and indexing. Specify requirements for the system.
- Briefly research current solutions for collaboration upon publications.
- Design a system supporting the process of publishing activities. During the designing, take into account extensibility of the system and consider integrations of suitable external services.
- Implement the designed system and test it thoroughly. Justify selection of the programming language and other technologies (e.g. a web framework).
- Evaluate benefits of the system for users and compare it with alternative applications.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 20, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Academic Collaboration Information System

Bc. Petr Jirásko

Katedra softwarového inženýrství
Supervisor: Ing. Marek Suchánek

May 9, 2019

Acknowledgements

I would like to thank my tutor, Ing. Marek Suchánek, for all the care, immediate answers to my emails and great help with this thesis. I would also like to thank Ing. Robert Pergl, Ph.D. for helping me with design of the system.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In In Prague on May 9, 2019 ……

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Jirásko, Petr. *Academic Collaboration Information System.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Cílem mé diplomové práce je navržení a implementace aplikace, která bude usnadňovat vědeckou spolupráci při psaní publikací. Práce obsahuje popis vytváření vědeckých článků, návrh systému, jeho implementaci a testování.

Klíčová slova Vědecká spolupráce, usnadnění, integrace

Abstract

The aim of my diploma thesis is design and implementation of an application, that would facilitate scientific collaboration during process of creating publications. This work contains description of publication creation process, design of the system as well as its implementation and testing.

Keywords Scientific collaboration, facilitation, integration

Contents

List of Figures

List of Tables

Introduction

Aim of this thesis

Writing of scientific publications is, as every other activity demanding human cooperation, a challenging process. There are various tools for cooperation, such as emails, Kanban boards, different cloud drives etc., however their disadvantage is, that they are not connected and the principle investigator of a project has to manually add new collaborators to all of these sources.

The aim of this thesis is to create a system, that would reduce the ammount of unnecessary manual work while managing these 3rd party tools.

Structure of this thesis

This thesis is logically divided into 4 main chapters: Analysis of current situation and requirements on the application, Design, Implementation and Testing.

Analysis of current situation and requirements on the application describes types of publications as well as their creation and contains analysis of current tools for scientific collaboration. There also is analysis of requirements on the appplication along with their use cases.

Design describes selection of the type of the application and based on that selection of used technologies and architecture. There is also OntoUML model thoroughly describing the application domain. Design chapter also contains relational modal, which is based on OntoUML model and serves as a structure of the database. Last, but not least, this chapter contains wireframes.

Implementation describes all principles and peculiarities during implementation. It contains description of authentication and authorization, my own serializer, templates and mainly how I dealt with 3rd party resources integration.

Testing describes unit testing, functional testing as well as manual walk-through of the system.

Chapter 1

Analysis of current situation and requirements on the application

1.1 Scientific publications

In today's world, regardless whether scientific or not, there are countless information sources. Key part of spreading information you researched among target audience is publishing.

According to Öchsner [1] we can distinguish following types of publications (the higher it is in the list, the larger the volume of information and the more recognized the publication is):

- books – authorship
- books – editorship
- book chapters
- journals
- conference proceedings
- theses
- patents, technical reports
- others (web pages, non-scientific, etc.)

Disregarding where in the above mentioned list the publication is, it potentially needs a lot of cooperation.

As mentioned in Öchsner [1] it is possible to distinguish two broad types of publications according to the fact, whether their content is professionally reviewed.

So-called Grey literature, that is not professionally reviewed, is defined by *Luxembourg convention* [2] as a literature, that *which is produced on all levels of government, academics, business and industry in print and electronic formats, but which is not controlled by commercial publishers.*

Among reviewed literature there are two main types of publications – journals and books. Since publishing anything in reviewed literature is more complex than in Grey literature and therefore it needs more actions to perform, in the following subsection I will take closer look to journal publication type as a representative of a reviewed publication type and clarify the process of publication.

1.1.1 Identification of publications

Before I will clarify more about journals, let me explain, how reviewed publications are identified.

In order to identify a source well for later on citation, it is necessary to have a unique identifier. For journals there is an 8 digit number called ISSN (International Standard Serial Number [3]) and for books there is a 13 digit number called ISBN (International Standard Book Number [4]).

It is quite peculiar to evaluate scientific quality of articles. One of the ways is to measure the impact factor, which should tell about a journal, how prestigeous it is and therefore how reliable the information in it are. However, there is a lot of criticism of such an attitude [5, 6].

1.1.2 Journals and magazines

There are two basic types of periodical publications – journals and magazines. While magazines focus on more popular content, contains a lot of advertisements and the articles are only reviewed by an editor, journals are often go through a peer review process [7]. However, the border between these to types is not strict. The following subsection is addressed to content of scientific journals.

1.1.2.1 Types of contributions to journals

According to Öchsner [1] the basic division of types of contributions to journals, that are reviewed, is

- *Research paper* is the most common type of contribution to journal and describes significant advancement in a particular research field. It is original and therefore it was not published before. Every *research paper* is judged according to its novelty, contribution and quality of scientific content. *Alternative terms are 'Original Paper', 'Original Article' or 'Research Article'* [1].

- *Review article* does not provide only original content of the authors, however it rather summarizes content of other articles and its purpose is to broaden reader's knowledge through critical comparison. *Review articles* are highly cited source of information. *Alternative terms are 'Critical Review' and 'Critical Literature Review'.* [1].

- *Rapid communication* is an announcement of a breakthrough scientific results, that are meant to be widespread as much as possible in order to stimulate further research. *Alternative term by some journals: 'Letter'* [1].

- *Short communication* has a similar style as an article, however its length is limited and it only needs to demonstrate the *proof of principle* [1].

- *Technical note describes noteworthy improvements, significant novel applications, or practical solutions to problems in an (established) technique* [1].

In the following section subsection I will briefly explain the process of publishing a contribution in a journal.

1.1.2.2 Contributing into a journal

Writing a contribution to a journal is a very complex process. Depending on the institution you are working for there might be slight differences in the work flow of writing the contribution, however according to Ellison [8] writing a paper consists roughly of following stages.

- *Assignment* might be general or more concrete. Such an assignment should clarify what are the expectations from the paper, such as its length, due date etc., analyze the audience and choose a topic. Such an assignment can be result of your own idea or a call for paper, e.g. for a conference.

- *Research* might be done after creating your assignment as well as precede it. E.g. if there is a conference concerning a field of research you are involved it and there is a call for paper, you probably have done the research already.

- *Writing.* The vital part of making of an article. Tight cooperation among team members is necessary, as well as setting out the responsibilities, outlines etc. In order to ensure high standard, citing of good source is necessary [9].

- *Peer review.* To ensure the quality, the article is after finishing reviewed by experts in given research field [10]. The reviewer also might give suggestions, what should be updated. Also the article might be rejected.

- *Submission.* After a successful review and implementing of all demanded changes, an article is submitted and published.

Off course, as Winkler and Metherell say [9], a very important part of making an article is choosing a suitable topic, that is not too broad, neither too narrow.

1.2 Current tools for cooperation

In todays world, there are many tools supporting collaboration among researchers on a publication, some of which are traditional and used by almost everybody, while others offer groundbreaking features, however they are complicated to use and they are not widespread. In the following section I discuss some of the tools, that I discovered through interviewing researchers and web research. The overview, off course, is not complete and some of tools have overlaps in their features and usage.

1.2.1 Email

Email is the oldest form of electronic communication of which predecessors reach back into the 60's [11]. Its major advantage is, that everybody owns an email address and is willing to use it. However, this aspect might also be counteraproductive, because it might discourage email users to use anything else. Its main disadvantage is complete absence of any versioning tool as well as lack of organization. Some services, such as Gmail [12] offer a possibility to organize emails into conversations. There is also a browser extension called Streak [13] (unluckily available only for Chrome), that supports putting emails concerning one publication together and divide them into different phases.

1.2.2 Instant messaging tools

There are various instant messaging services, such as Slack [14], WhatsApp [15], Telegram [16] or Fleep [17]. Their main advantage is, that they allow collaborators to communicate with each other in real time. Even though with mentioned tools you can organize collaborators in groups, the main disadvantage is lack of structure causing peculiarities if somebody needs to extract a particular piece of information after a longer time period.

1.2.3 Kanban

Kanban boards are systems that are helping researchers to organize their tasks, put them into various categories and set deadlines and assignees. In figure 1.1 you can see an example of such a Kanban board – Trello [18].

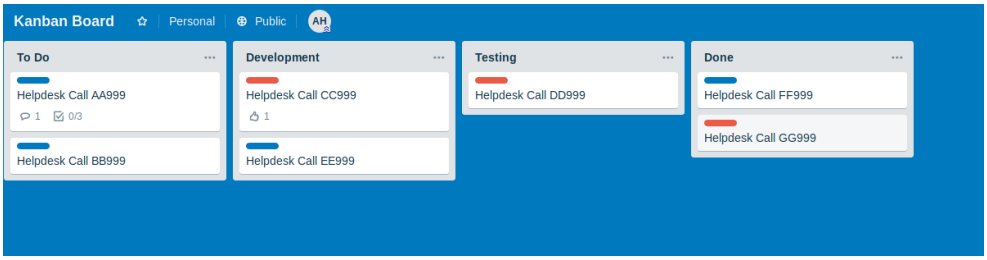

Figure 1.1: Trello kanban board

1.2.4 GIT

GIT is a powerful tool for versioning widely used by technically skilled researchers [19]. There are a lot of implementations, like GitHub [20] or GitLab [21]. Its main advantage is, that it is almost a perfect tool for versioning of LaTeXcodes. Its main disadvantage is, that manipulation through CLI demands a deeper knowledge of the system. However, there are various clients with GUI. Even though some researchers still refuse to use GIT and they share their versions by sending files through email.

As a bonus, GitHub and GitLab offer more tools such as CI, a Kanban board, issues (if there is in issue anywhere in the source code/text, an issue is assigned to responsible person to solve it).

1.2.5 Google Drive

Google family of services outstand with their user friendliness and therefore are quite popular. Google Drive is an online file system, which main advantage is that it is a part of Google ecosystem and almost everybody has an account. Among disadvantages of Google Drive belong absence of versioning.

1.2.6 Google Documents

Google Documents are WYSIWIG editor quite similar to Microsoft Word with very limited features, however with two advantages – they are part of Google ecosystem and there is no need to install any particular client, since you can edit documents directly in your browser simultaneously with other collaborators. They are widely use thanks to their simplicity, however there is only limited versioning available.

1.2.7 Research Gate

Research Gate is a social network for for scientists, where a researchers can share publications, ask questions (analogue of Stack Overflow [22] in the scope

of research world), follow other researchers updates, get statistics concerning your papers or create projects (however, without connection to any 3rd party services integrated) [23].

Its main advantage is you can stay in touch with your colleagues, however its effectiveness during publication process is discutable.

1.2.8 Open Science Framework

Open Science Framework is quite complex platform enabling users to create more complex projects and invite collaborators [24]. It contains a file system, where users can upload files, and a communication tool. Its main disadvantage is, that it is not connectible to any other external service.

1.3 SciCol

Even though, that there are various tools supporting scientific collaboration, none of them is ideal, however combined together they might be more powerful.

Therefore, in the scope of this thesis I will develop a system, whose idea is depicted in figure 1.2. It should basically connect various 3rd party tools in one system and combine their advantages.

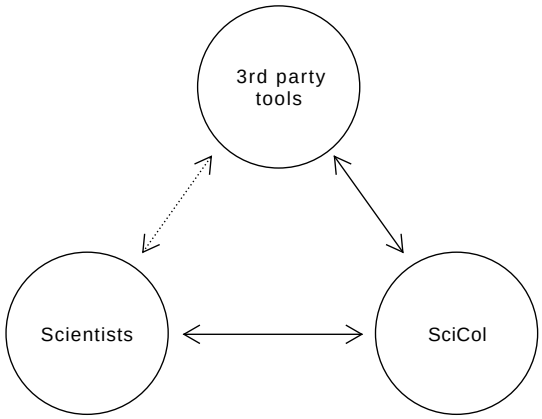

Figure 1.2: Basic idea of SciCol

1.4 Requirements

1.4.1 Functional requirements

In this section I will list functional requirements on the system from the user's point of view. In other words all functionality available to the user. I will

divide this section logically according to fields that the requirements are concerning.

1.4.1.1 Organization management

F-A-1: **It will be possible to add and modify scientific organizations.** A chosen user (hereinafter referred to as *super administrator*) will be able to add or modify scientific institutions (hereinafter referred to as *scientific organizations*), which will be identified by their domain name, such as "cvut.cz".

F-A-2: **It will be possible to set one or more administrators for a given organization.** Such administrators (hereinafter referred to as *organization administrator*) can be added or removed by other *organization administrators* within the *scientific organization* or by *super administrator*.

F-A-3: **Super administrator will be able to add other super administrators.** However, a super administrator will not be able to remove other *super administrators*.

1.4.1.2 Registration

F-B-1: **A registration into system will be possible for users possessing an email account belonging to an enlisted scientific organization.** Users will be able to register with their email, that is on a domain of an *scientific organization* that is added to the system.

F-B-2: **User will be able to log into the system with registered email.** Such an email will have to belong to one of the *scientific organizations* in the system and also will determine, that the user belongs to a particular organization. Log in with any SSO will not be provided within the scope of this thesis.

F-B-3: **Password recovery will be possible.** User will be able to recover his password through a link sent to his email address.

F-B-4: **Organization administrator will be able to inactivate and activate users.** *Organization administrator* will be able to set status of a user to inactive. That means, that all data entered into the system by this user will be preserved, however the user will not be able log in and/or actively participate in projects anymore. *Organization administrator* will be able to make this user active again.

1.4.1.3 Project ownership

F-C-1: **User belonging to a scientific organization can start a new project.** Under the term new project (hereinafter referred as *project*) we understand a new publication. By founding such a *project* a user will become a principal investigator of this *project* (hereinafter referred to as as *principal investigator*). There can be only one *principal investigator* at once.

F-C-2: **A principal investigator will be able to add or remove project collaborators.** Collaborator (hereinafter referred to as *project collaborator*) will then have to approve that he wishes to be a part of the project.

F-C-3: **A principal investigator will be able to add or remove project administrators.** Only the *principal investigator* of the project will be able to remove other administrators (hereinafter referred to as *project administrators*). A *project administrator* will otherwise have the same rights as the *principal investigator*. Therefore, all actions, that can be performed by *project administrator*, can be performed as well by *principal investigator*. As well, any action, that can be performed by *project collaborator* can be performed by *project administrator*.

F-C-4: **A principal investigator can pass the ownership of the project.** *Principal investigator* can pass the ownership to any of *project collaborators* or *project administrators*. Such a step will result into a loss of ownership for the former *principal investigator* (due to the constraint, that ownership can be held by just one user at a time).

1.4.1.4 Project settings and visibility

F-D-1: **A project administrator will be able to set visibility of the project.** There will be 4 levels of visibility: visible for *project collaborators* only, visible for users within *organization*, visible for every user logged in the system and visible to visitors. By visibility is meant to display *project* name and *project* description.

F-D-2: **A user will be able to list projects visible to him.** User will be able to filter the displayed projects by *organization* or display only projects he is involved in.

F-D-3: **A user can ask to join any project visible to him.** To start working on the *project* the *project administrator* will have to approve his request.

F-D-4: **System will contain set of templates.** Templates will determine the work flow of the project. There will be 2 types of templates: the

first editable by *super administrator* defining a type of publication, the second editable by *organization administrator* redefining the common one with organization specific tasks added.

F-D-5: **A template will contain set of phases.** Each *phase* will have a name, a deadline, list of preset tasks and one or more *resources* such as on-line editor.

F-D-6: **A user will set up project settings according to the template.** After selecting a template, *project phases* and *tasks* will be automatically set up. The user can then adjust the settings and then establish a new *project*.

F-D-7: **All tasks and deadlines will be automatically synchronized with chosen task system and calendar.** As a part of project settings the *project administrator* will choose a preferred task system (such as Trello) and a calendar (such as Google calendar). These will be synchronized among each other and change in any of these 3 will result into automatic change in 2 others.

F-D-8: **3rd party applications needed in a project will be initialized by credentials given by users.** That means, that where necessary, accounts provided by users will be added to a 3rd party application (e. g. after establishing a *project* a new Trello board will be created and *collaborators* added to their tasks).

1.4.1.5 Project work flow

F-E-1: **All collaborators will see project phases.** Every *collaborator* will be able to see *project phases* and their status (done, opened, not opened yet). By clicking on the *project phase* a *collaborator* will be able to see *phase* details.

F-E-2: **Project administrator will be able to switch the phase of the project.** *Project administrator* will be able to manually switch between *project phases*.

F-E-3: **Project administrator will be able to suspend a project.** *Project administrator* will be able to temporarily suspend the project by putting it into suspended state. Later on any of the *project administrators* will be able to put it back into life or set it as archived. A suspended state means, that nobody can work on the project anymore, however it will still appear among active projects.

F-E-4: **After finishing the project can be archived.** *Project administrator* can archive the *project*. Archived *projects* will be removed from *visible* projects to *archived*.

F-E-5: **A new project can be established from an archived project.** A user can create a new *project* from an archived one.

1.4.2 Non-functional requirements

N-1: **The system will be a web application.** It will not be necessary for the user to install anything but a web browser.

N-2: **The system will be mainly optimized for desktop computers.** The application is primarily intended to be used in a desktop environment. Even though selected functions will be optimized for mobile devices.

N-3: **All the data in the system will be stored in a relational database.** Off course, vast part of the project will be stored in 3rd parties applications. The system database will only contain information necessary to reach them.

N-4: **The whole system will be in English language.** No support for translations will be provided.

N-5: **Calendar, task system and other resources will be loosely coupled with the system.** That means, that an arbitrary calendar, task system or any kind of resource in the system will be accessed by the system through a given facade. Particular implementation of such a facade for given resource will be provided as a module.

N-6: **Super administrator will be added before launching the system.** Since *scientific organization* can be added only by super administrator, there will be at least one super administrator account added before launching of the system.

N-7: **There will be application's account for 3rd party systems.** I. e. there will be shared Google drive or Trello account for the needs of application.

N-8: **Collaborators can use 3rd party task system in every phase.** Whenever they mark task as done, it will also appear done in the system.

1.5 Use cases

1.5.1 Actors

There will be following actors in the system.

- **Unregistered user.** The only actions he will be able to perform is to register and view publicly visible projects.

- **Inactive user.** User set by the organization administrator as inactive. He can not perform any actions, however all data in a system generated by him are preserved.
- **Active user.** User, that has registered with an email belonging to one of the scientific organizations that are in the system.
- **Collaborator.** A user, that is participating in a project.
- **Project administrator.** A user, that can set project collaborators and phases with tasks and deadlines.
- **Principal investigator.** A user, that have initiated a project or have been given the ownership by a principal investigator. Has the same privileges as a project administrator, however a principal investigator is the only person that can add or remove project administrator.
- **Organization administrator.** Can activate and inactivate users within their organization as well as add or remove other organization administrators.
- **Super administrator.** A person that can add or remove scientific organizations and organization administrators and do basically anything in the system.

1.5.2 List of use cases

In this subsection the use cases are also logically divided into sections as in case of functional requirements.

1.5.2.1 Organization management

UC-A-1: **Add scientific organization**

Actors: Super administrator

Scenario:

1. Super administrator requires to add a new scientific organization.
2. Super administrator fills in details about scientific organization, such as its name, location and most importantly its domain.
3. Super administrator requests saving of the new scientific organization.
4. System saves the new organization.

Outcome: Now it will be possible to register with an email of added scientific organization.

UC-A-2: **Modify scientific organization**

Actors: Super administrator

Scenario:

1. Super administrator asks the system to list scientific organizations.
2. Super administrator chooses scientific organization to modify.
3. Super administrator modify details of scientific organization.
4. Super administrator requests saving of the scientific organization.
5. System saves the details of the scientific organization.

UC-A-3: **Add organization administrator**

Actors: Super administrator, Organization administrator

Scenario:

1. Super administrator asks the system to list scientific organizations.
2. Super administrator selects a scientific organization.
3. Super administrator selects a user from the selected scientific organization.
4. System adds a Organization administrator privilege to given user.

UC-A-4: **Add organization administrator**

Actors: Organization administrator, Active user

Scenario:

1. Organization administrator selects a user from given Scientific organization.
2. System adds a Organization administrator privilege to given user.

UC-A-5: **Remove organization administrator**

Actors: Super administrator, Organization administrator, Active user

Scenario:

1. Super administrator asks the system to list scientific organizations.
2. Super administrator selects a scientific organization.

3. Super administrator selects Organization administrator from selected scientific organization to be removed.
4. System removes the Organization administrator privilege from given user.

Outcome: Selected Organization administrator becomes Active user.

UC-A-6: **Remove organization administrator**

Actors: Organization administrator, Active user

Scenario:

1. Organization administrator selects organization administrator to be removed.
2. System removes the Organization administrator privilege from given user.

Outcome: Selected Organization administrator becomes Active user.

UC-A-7: **Add Super administrator**

Actors: Super administrator, Active user

Scenario:

1. Super administrator select an Active user from a pool of all Active users.
2. System give the active user the privileges of Super administrator.

1.5.2.2 Registration

UC-B-1: **Register**

Actors: Unregistered user, Active user

Scenario:

1. Unregistered user requests registration with his email.
2. System checks, if the email is on a domain of any Scientific Organization in the system. In case it is a registration link is sent to the email.
3. Unregistered user follows the link and sets a password and fills in his details, such as name.
4. System adds the user to list of users.

UC-B-2: **Log in**

Actors: Active user

Scenario:

1. Active user fills in his credentials.
2. System checks, if the email and password are correct.

Outcome: Active user is logged in.

UC-B-3: **Password recovery**

Actors: Active user

Scenario:

1. Active user requests password recovery on the login page.
2. System send a link to Active user's email.
3. Active user follows the link and sets up a new password.

UC-B-4: **Deactivate users**

Actors: Organization administrator, Active user, Inactive user

Scenario:

1. Organization administrator selects a user in his Scientific organization.
2. Organization administrator turns the Active user into an Inactive user.

Outcome: The inactivated user can not log in anymore, however, the data he has entered into the system will be preserved.

UC-B-5: **Activate Users**

Actors: Organization administrator, Active user, Inactive user

Scenario:

1. Organization administrator selects users from the list of Inactive users in his Scientific organization.
2. Organization administrator actives selected users.

Outcome: Activated users can again log in and participate in projects.

1.5.2.3 Project ownership, settings and visibility

UC-C-1: **Start a new project**

Actors: Active User, Principal investigator, Project administrator

Scenario:

1. Active user asks the system to start a new project.
2. Active user selects a template. The system creates phases and tasks of the project.
3. Active user fills in details of the project.

4. Active user adds collaborators. They will be added, however in order to participate in the project they will have to accept it.
5. Active user asks the system to save the project.
6. System saves the project.
7. Active user becomes the principal investigator of the project.

UC-C-2: **Accept project**

Actors: Active User

Scenario:

1. Active user lists his projects.
2. Active user asks the system to accept a project.
3. Active user fills in his credentials for 3rd party systems.
4. System checks, if provided credentials are valid.
5. System adds user as a project collaborator.

UC-C-3: **Start a new project from old project**

Actors: Active user, Principal investigator

Scenario:

1. Active user lists archived and frozen projects.
2. Active user selects a project.
3. Active user establishes a new project based on the old one.
4. Active user adjusts the settings of the project.
5. Active user becomes principal investigator of this project.

UC-C-4: **Remove project administrators**

Actors: Principal investigator, Project administrator

Scenario:

1. Principal investigator list projects he owns.
2. Principal investigator selects project.
3. Principal investigator removes the project administrator.
4. Selected project administrators become collaborators of the project.

UC-C-5: **Pass project ownership**

Actors: Project Owner, Project administrator, Collaborator

Scenario:

1. Principal investigator selects one of the project collaborators or project administrators.
2. Principal investigator then passes the ownership to the selected person.

Outcome: The former principal investigator loses project ownership and becomes project administrator.

UC-C-6: **Join a project**

Actors: Active User, Project administrator

Scenario:

1. Active user list all projects visible to him.
2. Active user asks to join a project.
3. Project administrator then accepts or rejects the request.
4. Active user gets a notification about the result.

Outcome: Active user becomes collaborator of the project.

UC-C-7: **List visible projects**

Actors: Active User, Project administrator

Scenario:

1. User list all projects visible to him.

1.5.2.4 Project work flow

UC-D-1: **Display project phases**

Actors: Collaborator

Scenario:

1. Collaborator lists all active projects he is part of.
2. Collaborator selects a project.
3. System displays a chart with all phases of the project.

UC-D-2: **Switch project phase**

Actors: Project administrator

Scenario:

1. Project administrator displays project phases.
2. Project administrator clicks on the current phase.
3. Project administrator switches the phase.

UC-D-3: **Suspend a project**

Actors: Project administrator

Scenario:

1. Project administrator selects an active project.
2. Project administrator suspends the project.

Outcome: The suspended project will not be displayed among the active projects anymore. Nobody can work on it anymore.

UC-D-4: **Resume a project**

Actors: Project administrator

Scenario:

1. Project administrator lists all suspended projects.
2. Project administrator selects the suspended.
3. Project administrator resumes the project.

UC-D-5: **Archive a project**

Actors: Principal investigator

Scenario:

1. Principal investigator lists his active and suspended projects.
2. Principal investigator selects a project.
3. Principal investigator archives the project.

Outcome: Archivf

1.5.3 Use cases diagrams

I prepared 2 use case diagrams, since there were to many use cases to display them in just one diagram.

In diagram 1.3 there are use cases concerning *Organization administrators* and *Super administrators*. In diagram 1.4 there are use cases concerning *Unregistered users*, *Active users*, *Collaborators*, *Project administrators* and *Principal investigators*. The link between those 2 diagrams is, that *Organization administrator* inherits from *Active user*.

1.5.4 Use cases fulfilling requirements

In order to be sure use cases cover all requirements, in tables 1.1 and 1.2 there is described, which use case fulfill which reqirement.

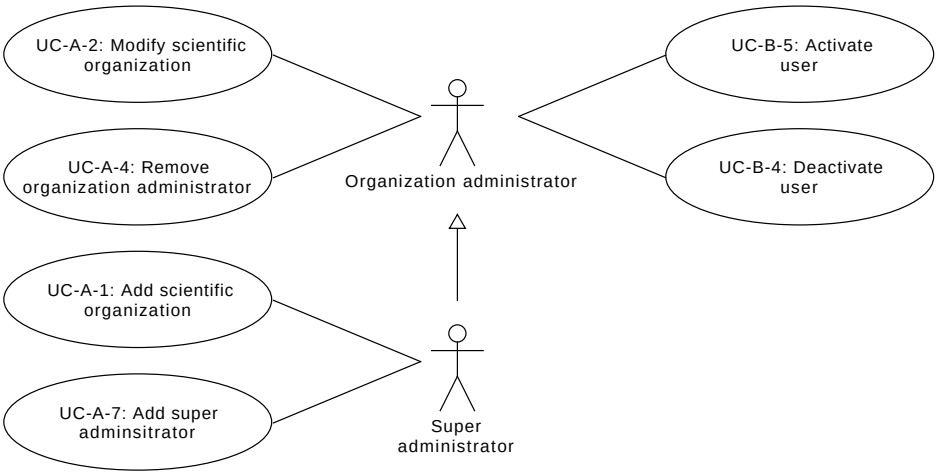

Figure 1.3: Adminstrator use cases

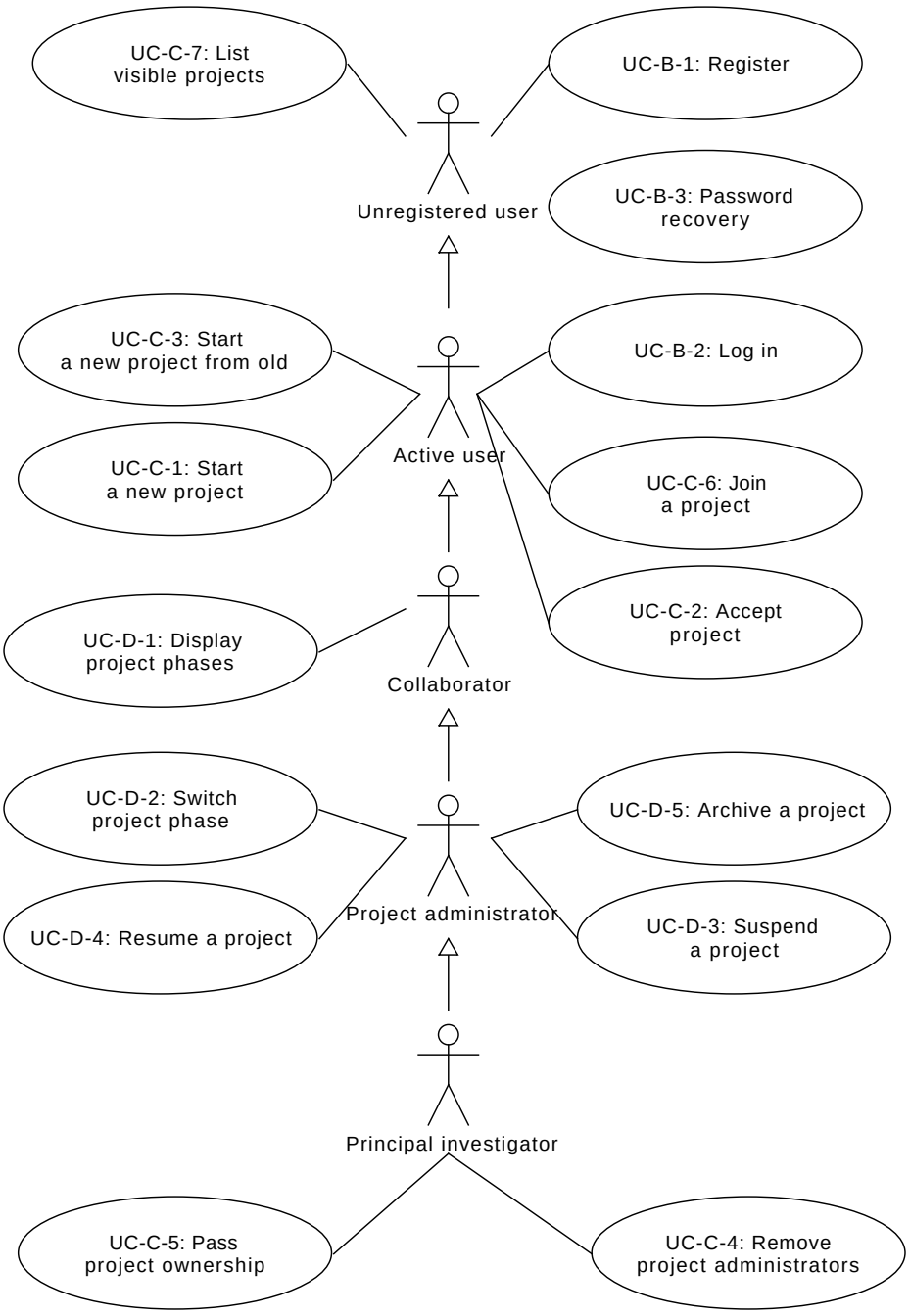

Figure 1.4: Active users use cases

Table 1.1: Table of requirements fulfilled by use cases 1

		Use cases										
		UC-A-1: Add scientific organization	UC-A-2: Modify scientific organization	UC-A-4: Add organization administrator	UC-A-5: Remove organization administrator	UC-A-7: Add super administrator	UC-B-1: Register	UC-B-2: Log in	UC-B-3: Password recovery	UC-B-4: Deactivate users	UC-B-5: Activate users
Reqs	F-A-1:	*	*								
	F-A-2:			*	*						
	F-A-3:					*					
	F-B-1:						*				
	F-B-2:							*			
	F-B-3:								*		
	F-B-4:									*	*

Table 1.2: Table of requirements filled by use cases 2

		Use cases												
		UC-C-1: Start a new project	UC-C-2: Accept a project	UC-C-3: Start a new project from old one	UC-C-4: Remove project administrators	UC-C-5: Pass project ownership	UC-C-6: Join a project	UC-C-7: List visible projects	UC-D-1: Display project phases	UC-D-2: Switch project phase	UC-D-3: Suspend a project	UC-D-4: Resume a project	UC-D-5: Archive a project
	F-C-1:	*		*									
	F-C-2:	*		*									
	F-C-3:	*		*	*								
	F-C-4:					*							
	F-D-1:	*		*									
	F-D-2:		*					*					
	F-D-3:					*							
	F-D-4:	*		*									
Reqs	F-D-5:	*		*									
	F-D-6:	*		*									
	F-D-7:	*		*									
	F-D-8:						*						
	F-E-1:							*					
	F-E-2:									*			
	F-E-3:									*	*		
	F-E-4:											*	
	F-E-5:			*									

Chapter 2

Design

In the previous chapter I specified requirements that the system must meet. In order to be able to implement the application a good design is necessary. In this chapter I will describe and justify the selection of technologies and introduce OntoUML model, wireframes and data model.

2.1 Type of application

The application should serve variety of users, from those who are technically competent to those that have no connection to the world of technologies. Therefore usage of the application should be as simple as possible and there should not be any barrier in a form of a need to install any sort of client. Therefore, according to non-functional requirement N-1: the application will be a web application, which does not require any specific software but a web browser.

2.2 Used technologies

Before I was going to design the architecture, I found it beneficial to choose the technologies I would use in the application before designing the architecture, since the architecture is strongly dependent on the language and framework used.

2.2.1 Language selection

There is a large variety of programming languages around the world that are more or less useful to develop a web application. Some of them are very well known and are often place on the top ranks in lists of web programming languagues, such as PHP, Java, Python for back end and JavaScript and HTML for front end, while others are rather rarity, such as Rust or Go (

[25–27]). Therefore during the selection of a programming language I took into account more criteria.

The very first, rather personal criterion, in order to write the code effectively, was, that I have to have a certain knowledge of the language I would use. This quite shrank the selection to Java, Python and PHP for the back end and JavaScript accompanying HTML and CSS for the front end.

Along with this criterion of personal knowledge comes another one – good documentation and support from the community. The quality of documentation is hard to measure, but I find all of the documentations of languages mentioned above (Java [28], Python [29], PHP [30], JavaScript [31] and HTML [32]) as satisfying. When it comes to community support, I took as a measure the number of tags of selected languages on Stack Overflow [33] (1,539,953 for Java, 1,156,341 for Python, 1,278,731 for PHP, 818,421 for HTML, 583,114 for CSS and 1,798,449 for JavaScript). Those numbers indicate, that there probably is an answer to any question if a problem appears in the development.

Another criteria I took into consideration are the easiness of coding (which is debatable) and performance. According to my experience, as well as some other sources [34], Java is more robust and secure than Python and PHP, however it takes more time to code in it. Personally, I find PHP is slightly easier to code and therefore I chose PHP for the back end. For the front end I chose JavaScript, CSS and HTML, as they are a long-term, irreplacable standard.

2.2.2 Back end framework selection

Working with a pure language without any framework might be a solution for a very restricted project, however for a project of a size of this thesis it is an absolute necessity to have a framework. There are dozens of frameworks and in the decision, which one to select, I took into consideration following criteria: it must have a good documentation (hard to measure), it must have reasonable performance, support from community, well-working ORM, simple router and vast selection of libraries.

The requirement to have a well-working ORM and simple router disqualifies the most popular Czech framework Nette [35]. Due to the requirement, I was looking for a popular framework, so in the end, I was deciding among Laravel, Zend, Symfony and CodeIgniter ([36–39]). In the end I decided for Symfony framework, since it meets all criteria I demand and as bonus, I already had worked with it previously.

2.2.3 Front end framework selection

There are also numerous JavaScript frameworks. I had similar requirements on the JavaScript framework as on the PHP framework: it must have a good com-

munity support, good performance, vast selection of libraries and a perspective to be maintained in the future. From the most popular frameworks [40, 41] I found the most appropriate Angular.js and React.js. I decided to select React.js, because I already worked with it and it is fast to develop as well as it offers satisfying performance.

2.2.4 Relational database

To fulfill the non-functional N-3:All the data in the system will be stored in a relational database, there must be a relational database engine. The selection of relational database engines is not as wide as selection of JavaScript and PHP frameworks, however it is still quite complicated to select database that perfectly suits application needs.

Therefore I set a few criteria, that the database must meet. It must run on a Linux server as well as it cannot have a proprietary license, which disqualifies Oracle [42] and Microsoft SQL server [43]. It also should be among popular databases, since in case of any problem it is easier to find a solution, so in the end I was deciding among PostgreSQL, MySQL and MariaDB [44]. Since the application will not presumably use the database excessively and I wanted a database as easy to set up as possible, I chose MariaDB, which is a fork from MySQL and is as easy to maintain as MySQL, however offers more features [45].

2.3 Architecture

2.3.1 Back end architecture

The architecture of the system is determined by used frameworks. Architecture of the Symfony framework is based on Model-View-Controller pattern.

The Symfony architecture, according to [46], is depicted in figure 2.1. The work flow is following:

1. The user sends a HTTP GET request through webrowser to the server.

2. The server forwards the request to PHP layer - Symfony framework.

3. HttpKernel resolves, which Controller should process the requests and forwards the request to appropriate Controller.

4. Controller calls the Model structures to perform the business logic. When it is necessary to load anything from database, the Doctrine ORM queries the database and automatically transforms the selection results into PHP entities [47].

5. Controller then can return HTTP response directly (e.g. in form of JSON [48]) or render a View through a template enginge, in our case Twig [49].

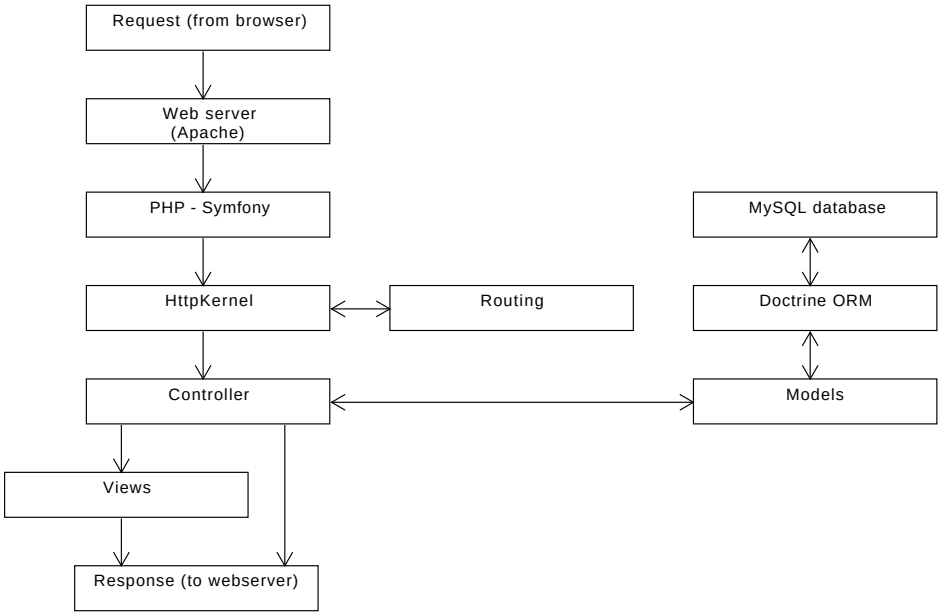

Figure 2.1: Symfony architecture

2.3.2 Front end

The front end of the application will be implemented in React [50]. In order to work, react needs an HTML element, where it can render the HTML tree. Therefore there will be an HTML Twig template [49], in which will be an element `<div id="react-root">`, which will be a container and the whole content will be render by React. The data necessary will be provided through the template as a variable.

2.3.3 Integration with other systems

The application will be connected to 3rd party systems and that is possible only through an API. The applications connected to the system will be Trello [18], Google Calendar [51], Google Documents [52] and Google Drive [53]. All of these offer such an API [54, 55].

However, to achieve extendability of the system, I do not want to hardwire any of these APIs into the application logic. Instead, I will introduce a resource

- an abstract class with abstract methods, that every resource will have to implement. As you can see in the figure 2.2, for each type of resource, such as a calendar or a task system, there will be another abstract class, that will implement the abstract resource method by using new abstract methods - i.e. for a calendar resource a method `addEvent`. These abstract methods will be then implemented in a particular resource class.

Such a structure ensures, that it not only will be possible to add a new resource, but there will be also possibility to introduce a new resource type by just implementing a naw resource type abstract class.

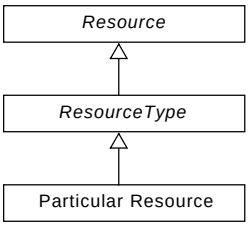

Figure 2.2: Resource structure

2.4 OntoUML model

In order to be able to design the application well, there was a need to design a good quality model. When deciding about the modeling language, the first thought was to use widely accepted standard in form of UML [56]. However, the standard UML model does not fully reflect ontological aspects and therefore I decided to use OntoUML.

OntoUML is a graphical conceptual modelling language aimed at constructing ontologically well-founded conceptual models. It is based on Unified Foundational Ontology (UFO) [57]. UFO is composed of three parts, each of which describes different aspects [57]. UFO-A deals with theory of various types of things, UFO-B with theory of events and processes and last, but not least, UFO-C intentional and social aspects of business processes. [57]. In my model I will use all three UFO ontologies.

2.4.1 Sortals

2.4.1.1 Kinds and Subkinds

During creating of the OntoUML model I was following materials of CTU's subject Conceptual Modeling [58].

I started the model with Sortal types providing identity – Kinds and Subkinds. The Kind `User` represents unregistered user, because even though there

is no database record necessary for such a user, he has his own ontological identity. The Subkind `Registered User` is from ontological view just `User` extension. I created a separate Subkind of `User` for `Super Administrator`, since he is not connected anyhow to any `Organization`.

The kind `Project` is a Kind representing work on one publication and it has its `Phases` and these have their `Tasks`. A `Phase` and `Task` can have their `Resource`, which is a connection to a 3rd party service.

2.4.1.2 Phases and Roles

A `Registered User` has 2 disjoint Phases – he can either be `Inactive` or `Active`. An `Inactive` user has been inactivated by administrator and can not perform any actions, while `Active` can perform all actions according to his permissions.

A `Project` can be in 3 disjoint phases – `Active`, `Suspended` and `Archived`. While `Active` project can be switched into `Suspended` one and vice versa, once a `Project` gets into Phase `Archived`, it cannot get into any other Phase.

A `Registered User` can have a Role `Organization Administrator` in relation with an `Organization` and Role `Collaborator`, `Project Administrator` or `Principal Investigator` in relation with a `Project`. These Roles are in hierarchy, since they only add permissions to their parent Roles.

2.4.2 Relations Whole-Part

There are only 2 Collectives – sets of `Templates` and `Team`, that always belongs to a certain `Project` and as members it has Roles `Collaborator`, `Project Administrator` and `Principal Investigator`.

2.4.3 UFO-C concepts

From UFO-C conceptes there are 2: UFO-C action and UFO-C organization. I have chosen UFO-C organization for `Organization`, since it is a spot on match concept of it.

2.5 Relational Model

From the OntoUML model, that describes the domain more deeply and in more details I needed to create a relational model of database, respectively class model of entities used in Symfony. I decided to follow Zdeněk Rybola's doctoral dissertation [57] with a vast amount of adjustments to the needs of the application (`names in bold` are concerning OntoUML model, *names in italics* are names of UML classes). The resulting class diagram is to be seen in the figure 2.4.

2.5.1 Transformation of Kinds and Subkinds

I transformed Kinds into UML classes with following adjustments:

- `User`, `Registered User` and `Super Administrator` were transformed into one class *User*. Even though a `User`, that is not registered, has his own ontological identity, I omitted it in the class model, since he can not perform any action except listing project and therefore he does not need any database entry. I degraded `Super Administrator` to be a single system role in the list of *User's* roles property.

- `Resource` has its reflection into class structure only in the form of *ResourceUsername* a *ResourceSettings*, since the `Resources` are not an item to store in a database, but they are loaded on the fly from the classes with particular implementations, that can handle a 3rd party service (see also subsection 2.3.3).

- No adjustments were made during transformation of `Project`, `Phase` and `Task`.

2.5.2 Transformation of Roles

`Collaborator`, `Project Administrator` and `Principal Investigator` were shrank them into one class connected to *Project* and *User*, where the Role in the `Project` is described by the property *role*. Role `Organization Administrator` was degraded to be a system role in the list of *User's* roles property.

2.5.3 Transformation of Phases

All the Phases were degraded to a single value in their respective entities. Since they are all complete and disjoint, they are introduced as enumeration (*phase* in *Project* and) and a boolean value (*active* in *User*).

2.5.4 Transformation of relations Whole-Part

The Collective `Team` was transformed into *Project* property *projectInvolved*, which is a collection of *ProjectInvolved*. Template will be stored as a textual settings (see also section 3.3) and therefore there will be only the *Template* entity and the *Phases* will be created from the textual settings.

2.5.5 Transformation of UFO-C concepts

All the Actions, that are part of the model, will be implemented as method of respective entities. The `Scientific Organization` was degraded to a single entity *Organization*.

2.6 Wireframes

Wireframes are a prototyping tool, that not have any precise colors or design, however, they are there in order to depict the layout and type of control structures such as text inputs, check boxes, switches, menus and others [59].

For this prototyping I used the tool called Pencil [60]. Since the application is mainly proposed for desktop user, I chose the layout with classical menu on a side the main container next to it. Along with that comes the user settings menu in the right top corner.

In figure 2.5 you can see establishing of a new project, in figure 2.6 you can see, how such a project will be displayed.

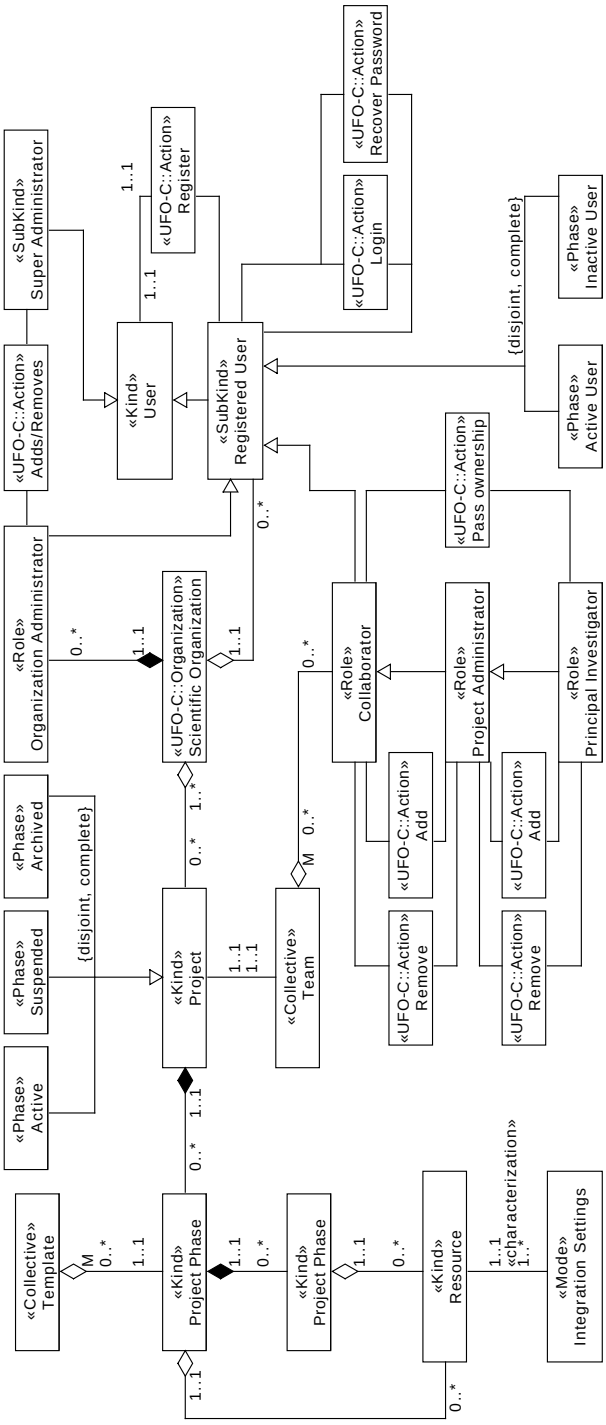

Figure 2.3: OntoUML model

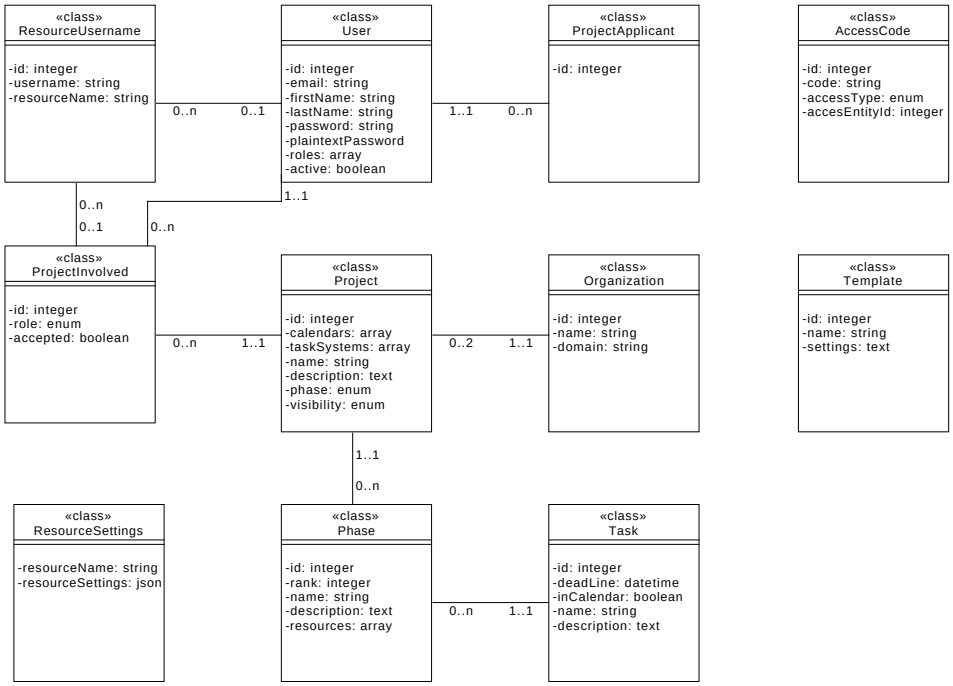

Figure 2.4: Class diagram

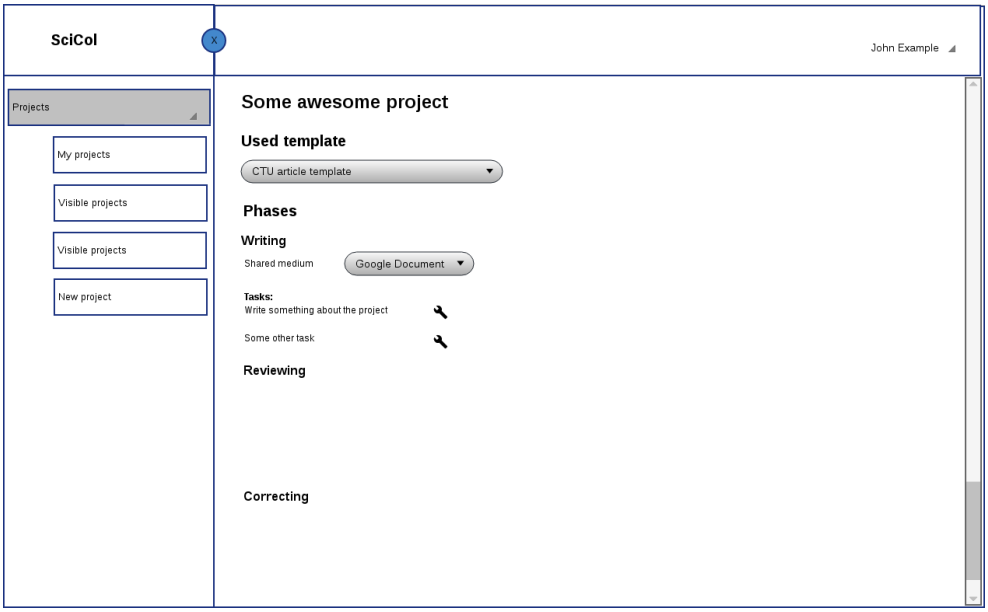

Figure 2.5: Configuring new project

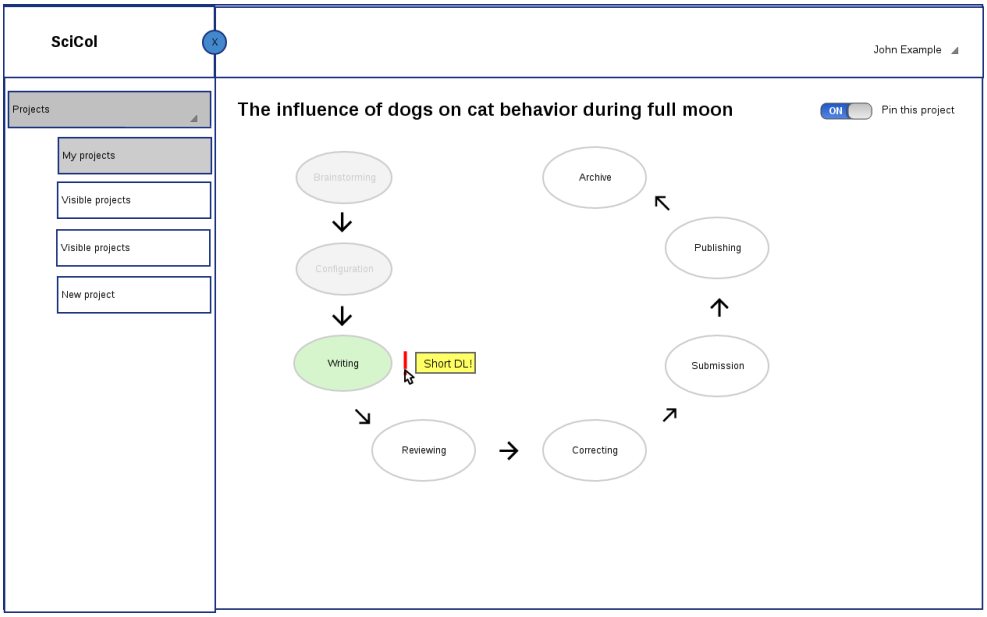

Figure 2.6: Viewing a running project

Chapter 3

Implementation

3.1 Authentication and authorization

In general, authentication is verifying identity of the user while authorization is checking, whether selected user can perform a particular action [61]. A real life example: a police check – if a policeman stops a driver, firstly he authenticates the driver by his id and then the authorization comes – the policemen checks whether the driver has the permission to drive a motor vehicle.

Since the system is based on collaboration among multiple scientific organizations (and number of organizations is virtually not limited), we encounter 2 issues concerning authentication and authorization. Firstly, the usage of SSO systems is problematic as there would be a need to provide a particular implementation for every organization. Secondly, for every user we need to specify not only list of his authorizations (i.e. list of actions a user can perform), but also the organization connected to these rights (e.g. a user is an organization administrator at Czech Technical University in Prague, however he cannot perform any administrator actions at Brno Univesity of Technology).

3.1.1 Registration

To avoid usage of SSO systems on one hand and the necessity of approving every registration manually on the other hand I decided to provide a registration through email on domain of selected scientific organization. An organization has saved its domain (e.g. `cvut.cz`) and when a new user registers, the system checks, whether his email has a domain of a known scientific organization.

In such a case a new `User` entity is made with `active` property set to false and in the same time a verification email with activation code is sent to entered email address.

3.1.2 Authentication

Symfony provides so called Security bundle [62]. This provides either tools for authentication as well as for authorization.

The authentication is provided based on users email and password, which is stored in the database in form of salted hash [63], which is a prevention against any password stealing. The `plaintextPassword` property of `User` entity is used only for processing sign in form.

In Symfony it is possible to user your own class for user (in my case `App\Entity\User`), that has to implement `UserInterface` and there must be an instance of `UserProvider` [64], that tells the framework, where and how to load and authenticate users. Thanks to this mechanism the usage of SSO systems is possible (even though it is not in the scope of this thesis).

3.1.3 Authorization

There are two approaches to authorization in a Controller – either the *@IsGranted* notatation or calling `isGranted` method. The difference is, that if the function is called and user is unauthorized, *@IsGranted* notatation will cause 403 HTTP status, while is `isGranted` method just returns a boolean value, whether the user has a given role:

```
/**
 * @IsGranted("AUTH_SUPER_ADMIN")
 */
 public function someMethod(){
      if($this->isGranted('AUTH_SUPER_ADMIN')){
          ...
      }
 }
```

Both of methods mentioned above use use authorizations and to perform a check Symfony uses classes implementing `VoterInterface` [65], which have 2 methods – `supports` tells you, whether the particular voter should decide about the authorization, `userCanReachAuth` then tells you, if user should be granted or not. The Voters are implemented in `App\Security` and there are two basic approaches I used to implement them.

3.1.3.1 Roles and role hierarchy

The first approach to implement Voters is, that they check if a `User` has a certain role or if the demanded role is in his role hierarchy. Role hierarchy is a tree of roles, where some roles have a list of other roles, that are included in them. Role hierarchy used in SciCol is in the YAML 3.1.3.1. For some roles, the authorization is granted if a user has a certain role (i.e. if a user has the role `ROLE_SUPER_ADMIN`, he will be granted `AUTH_SUPER_ADMIN`), for others connect to a certain organization the role contains also id of the respective

organization (i.e. the user will be granted as `AUTH_ORG_SPEC` for organization with id equal to 1, if he has a role `ROLE_1_ORG_SPEC`).

```
role_hierarchy:
        AUTH_SUPER_ADMIN: [AUTH_ORGANIZATION_ADMIN]
        AUTH_ORGANIZATION_ADMIN: [AUTH_ORGANIZATION_MEMBER]
```

3.1.3.2 Dedicated entities

Another approach to implement Voters is to have dedicated entities. I use them only in connection with a `Project` object, where `ProjectInvolved` connected to a particular instance of `Project` entity contains information about user's authorization in relation to the project – whether the user is only a collaborator or project administrator / principal investigator.

3.2 Serialization

Since the PHP back end of the application runs on the server and most of the front end runs in JavaScript in a web browser of the user, there is a need to serialize the entities in the back end into some format such as JSON or XML, sending them through HTTP protocol to the client, deserialize and process them on the client side and then serialize, send and deserialize them again. Therefore I needed a (de)serializer.

3.2.1 Symfony (de)serializer

Symfony offers its own serializer, which serializes entities (firstly it normalizes an entity into an array and then encodes the array into the chosen format) [66]. However I have encountered 3 major issues that made the built-in serializer unusable.

Firstly, it is impossible to choose which properties of an entity will be serialized and followingly send to the front end. In some entities I store information that should stay just in the back-end (such as security tokens for resources, see 3.4).

Secondly, in a lot of entities I had a problem with circular references (e.g. a `Project` has a list of `Phases` and a `Phase` has its `Project`). In order to avoid this you can provide a built-in `ObjectNormalizer` object, where you can set a circular reference limit and function that handles the cases of circular references [66]. However, those two methods are deprecated since Symfony 4.2 [67].

Thirdly, I could not find any tool, that would transform PHP code of entities into JavaScript, to be able to use them on the front end side.

3.2.2 Own serializer

Due to issues mentioned above I decided to implement my own serializer, that is located in `App\Serialization\Serializer`.

Every entity, that can be serialized must implement `ISerializableEntity` interface:

```
interface ISerializableEntity {
    public static function getProperties();

    public function getId() : ?int;

    public function setId(?int $id);
}
```

The `getProperties` function returns an array of `App\Serialization\Property`, that are supposed to be sent to the front end. `Property` contains information about

- type of property – either primitive (scalar types as defined in PHP [68]), entity (object implementing `ISerializableEntity`) or array, that can either consist of primitives or entities.
- getter function – by default set to `getPropertyName`, however it is possible to set it to something else. This function is used to retrieve the value of the property.
- setter function – by default set to `setPropertyName`, which is used during deserialization to set a value of either primitive value or an entity.
- adder function – by default set to `addPropertyName`, which is used during deserialization to add elements to array properties.
- class name – in case of an entity the class name, that is stored also in the JavaScript side object and helps deserialization when retrieved back.

During serialization the `Serializer` transforms an entity into an array, which is then encoded into JSON format by standard PHP function *php_encode* [69] and send to the front end.

During deserialization the `Serializer` the serializer distinguishes two cases. If the entity coming from the front end already has an id, that is not null or negative, the `Serializer` loads the entity from repository and updates properties that are not null in the serialized entity. If the entity's id is null or is negative, a new entity is formed. Negative ids are added to support deserialization of complex new entities to signalize, that the entities with the same negative ids are the same.

3.2.3 Front end part of serialization

Since I needed some tool, that would transform the entities code from PHP to JavaScript, I decided to implement a Symfony command [70], which is run from the command line by `php bin/console serializer:js` and is located in `App\Command\JSSerialization`.

It automatically transforms all classes implementing `ISerializableEntity` interface and located in `App\Entity` into a JavaScript class, that has a constructor taking either an array (that initializes the entity from JSON, that is reconstructed from the string sent over HTTP by standard JavaScript function `JSON.parse`) or nothing (that creates a class with all properties set to null or an empty array in case of array properties) and a method `deserialize`, that deserializes the object back to JavaScript JSON (it also recursively deserializes subentities).

3.3 Templates

When a user decideds to establish a new project, he does not have to define all the phases and tasks inside of them, however, he can use a template. When a template is selected, phases and tasks within the phases are filled in according to the template.

3.3.1 Template entity structure

The `Template` entity contains properties `name`, `organization` and `settings` and represents a template of phases and tasks for a certain type of publication, such as article or book.

If the `organization` is null, it means the template is a general template not connected to any organization and is editable only by a super administrator. On the other hand, if the `organization` is not null, it belongs to an organization and can be edited by a super administrator or a respective organization administrator.

The `settings` is a text in YAML format and represents a recipe, what all should be added to a project if it is base on the particular template.

3.3.2 Template settings structure

To describe the structure of a template let's use settings in YAML 3.1.

`ICalendar` and `ITaskSystem` contains names of resources, that will be used for as calendars and task systems. After establishing of the project the system will automatically add events to all listed calendars and put tasks into all task systems.

`Brainstorming` and `Planning`, are 2 phases that will be added to the project. `Brainstorming` has a rank equal to 1, which is a human readable

```
ICalendar: Google Calendar
ITaskSystem: Trello
Brainstorming:
    rank: 1
    description: "Description of brainstorming phase"
    leadsTo: [2]
    "Do brainstorming":
        rank: 1
        IDocument: GoogleDocument
        inCalendar: true
        description: "Some description of brainstorming part"
Planning:
    rank: 2
    description: "Plan the deadlines for the whole project"
    leadsTo: [3]
    "Plan who will do what":
        rank: 1
        inCalendar: false
        description: "Assign people to all tasks"
...
```

Figure 3.1: Template settings

and short identification of the phase within the project. The `leadsTo` attribute contains list of ranks of phases, that the phase can be switched into – in this case `Brainstorming` can be switched to `Planning`.

`Do brainstorming` is a task, which has a unique `rank` within its phase – `Brainstorming`. `IDocument` tells, that there will be automatically made a document for this task, that will be shared to all participants (however, you can use any kind of resource except those representing task systems and calendars). The `inCalendar` attribute determines, whether the particular task should be added to calendars, since if there would be all tasks, calendars would be labyrithine.

3.3.3 Creating a new project from a template

The whole process of transforming a template into a `Project` entity is depicted in the figure 3.2. When a user selects a template, a JavaScript method `TemplateParser.createNewProjectFromTemplate` is called and an instatnce of JavaScript class `Project` with pre-filled phases and tasks is created. User then adjusts the `Project`, which is then deserialized (since `Project` is a class created by serializer described in subsection 3.2.3), sent over HTTP, where the PHP `Serializer` turns it into a PHP entity `Project`

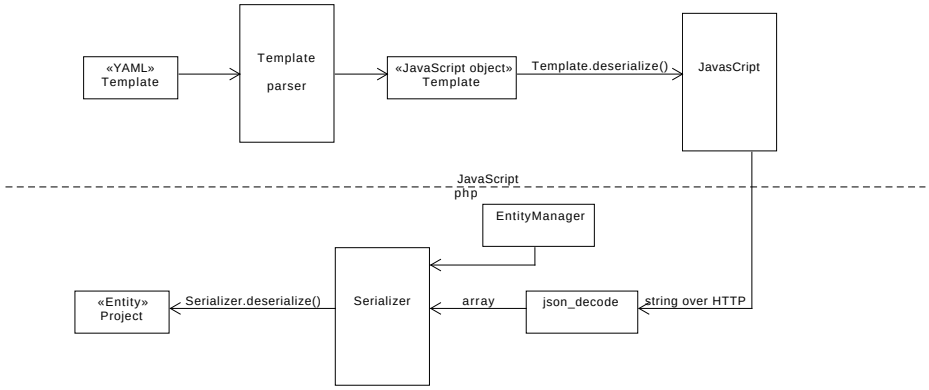

Figure 3.2: Template parsing diagram

3.4 Resources

The most important part of the system are, as I called them, resources. They represent a connection layer between the system and 3rd party services.

3.4.1 Abstract resource structure

As you can see in diagram 3.3, there is an abstract class called `Resource`, that defines all methods, that every single resource needs to implement, since they are necessary for synchronization of the resources during manipulation with projects, their phases and tasks (see also subsections **??** and 3.4.4).

Every resource has a name (i.e. `Trello` or `Google Document`) and is of a certain type – in our case a document, calendar, task system or a file system. Each type is represented by an abstract class extending the class `Resource` and all of these classes, `Document`, `Calendar`, `TaskSystem` and `FileSystem`, implement the abstract methods of `Resource` and introduce new abstract methods specific for the particular resource type, that are then implemented in the particular implementation.

In case anybody would like to add another resource type, i.e. GIT, he just needs to add an abstract class, that will extend `Resource`.

To make the matter clear, let's take a look on an example. The method `addProjectInvolved` is called, when a `ProjectInvolved` accepts the offer to participate in the project and we need to assign him his tasks in our `TaskSystem`, particularly `Trello`. The method `addProjectInvolved` is implemented in class `TaskSystem` and calls the abstract methods of `TaskSystem`, namely `addProjectInvolvedToBoard` and `addProjectInvolvedToTask`. These methods are then implemented in class `Trello`, that is connected to Trello API (see subsubsection 3.4.5.2).

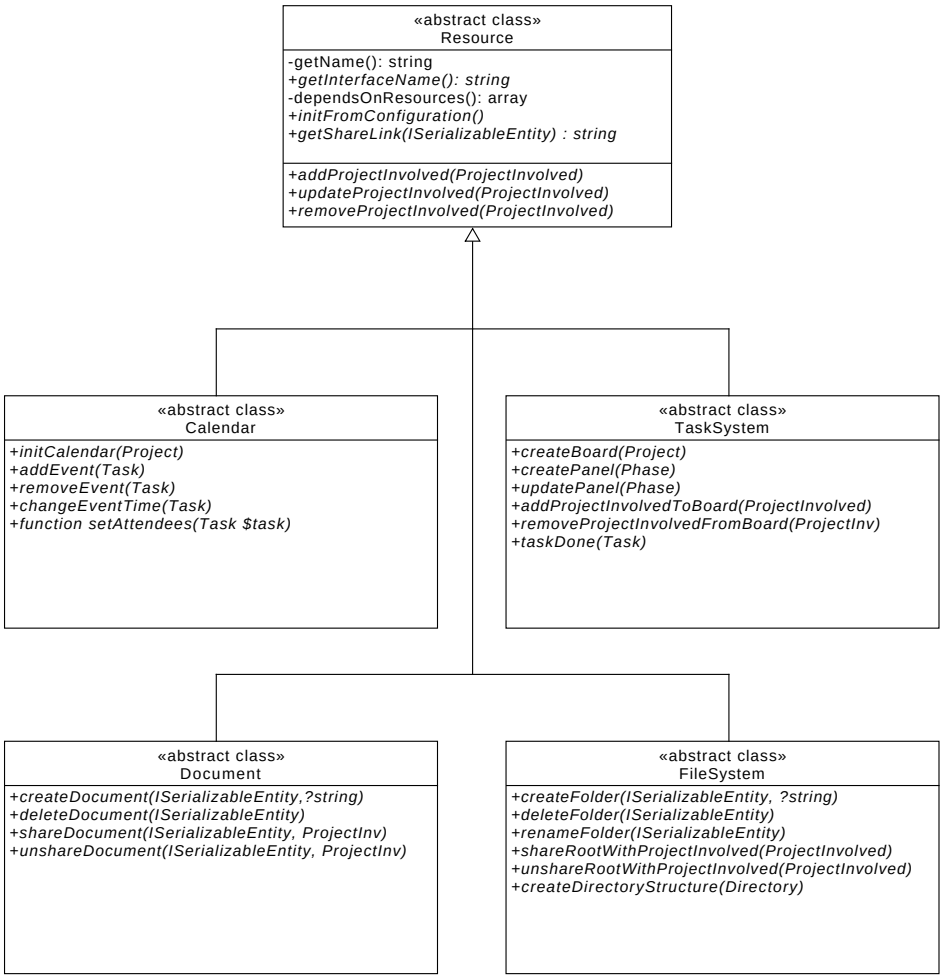

Figure 3.3: Resources structure

3.4.2 Pairing of application classes with resources

Parts of `Project`, namely parts represented by classes `ProjectInvolved`, `Phase` and `Task` are always bound to certain entities within the resource, e.g. a `Task` is represented by an event in `Google Calendar`. In order to be able to pair the application entities with the resource entities, there is an entity `EntityResourceRelator`.

`EntityResourceRelator` consists of four pieces of information – resource name (e.g. `Trello`), entity class (e.g. `Task`), SciCol entity id and resource entity id. Then `EntityResourceRelatorRepository` can provide resource entity id after providing of SciCol entity and vice versa.

3.4.3 Resource types

As mentioned earlier, there is an abstract class for every type of a resource. In case somebody would like to extend the application and add some other resource type, the only step necessary is to add such a class. Let's describe the resource types present in SciCol.

3.4.3.1 Calendar

Class `Calendar` represents a calendar system. `Project` is bound to a calendar and `Task` is bound to an event placed in such a calendar. `Phase` does not have a calendar counterpart.

3.4.3.2 TaskSystem

`TaskSystem` represents a tool for managing tasks. In `TaskSystem` a `Project` represents a board. Board consists of lists, each of which is a counterpart of a `Phase`. A list then contains arbitrary number of cards representing `Tasks`. If a `ProjectInvolved` is assigned to a `Task`, then in the task system he is assigned to the bounded card.

3.4.3.3 Document

A `Document` can be bound to either of `Phase` and `Task`.

3.4.3.4 FileSystem

`FileSystem` is a specific resource. If bound to any `Phase` or `Task`, the system creates a root folder for the whole `Project`, sub-folder of the root folder for a bounded `Phase` and a sub-folder of it for a bounded `Task`.

3.4.4 Resource Loader

When working with resources, the application somehow needs to manage them. For that purpose there is the class `ResourceLoader`. It implements following functions:

- `getResource` – return the resource with given name
- `getResourcesImplementing` – return all resources the are of a certain type, e.g. `Calendar`
- `getAllResources` – return all resource available in the application
- `checkAllUsernames` – for a given `ProjectInvolved` insepcts, whether all `ResourceUsername` entities have a valid username for the given resource

- `callFunctionOnAllResources` – calls the given function on all resources available. E.g. after `ProjectInvolved` accepts a `Project`, the `Resource` function `addProjectInvolved` is called on every Resource

3.4.5 Implementations

For every resource, that is used in the application, there must be a particular implementation, that connects to an external service through an API a performs the synchronization. The necessary configuration of every resource is stored in a YAML file, from where its loaded by `ResourceLoader` when a certain resource is required.

3.4.5.1 Common Google framework

Even though I use three different Google services (namely `Google Documents` [52], `Google Calendar` [51] and `Google Drive` [53]), they share the a common part to secure the communication with the services through the Google REST API.

That has the advantage, that for all Google service I can use the same core functionality. Therefore, every Google related resource, `Google Calendar`, `Google Drive` and `Google Document`, extend their respective parent classes (`Calendar`, `Document` and `FileSystem`) and use `GoogleCommon` trait, since there is not allowed multiple inheritance in PHP [71].

In SciCol the Google PHP client [72] is used, since it makes the usage of the Google REST API much simpler. In the common Google core the `Google_Client` is initilized and in `Calendar`, `Document` and `FileSystem` the corresponding services (`Google_Service_Calendar` and `Google_Service_Drive`, since Google does not distinguish much between folders and files [73]) are used.

3.4.5.2 Trello

Trello offers a REST API [54] to create, update and delete its boards, lists and cards as well as to assign users to cards.

In order to call the API, you can use a PHP HTTP client, such as Guzzle [74], however I decided to use Steven Maguire's Trello client [75]. It contained most of the methods I needed to work with Trello. I implemented the rest of them in the class `TrelloClient`, that extends Steven Maguire's client.

3.4.6 Savers

When addding, updating or removing an entity, that is a part of a project there is also a need to call associated services. From that reason there is an abstract class `Saver`, that has the abstract method `save`. Every type of entity related to `Project` (`Phase`, `Task` and `ProjectInvolved`) has an own implementation of this method.

This method accepts 2 arguments – the original entity instance, that is stored in the database, and the new serialized entity, that has updated information in it. Both of them can be null. The new entity with updated information cannot be serialized, since thanks to Symfony ORM the original entity would get updated as well. There are three allowed combinations:

- The original entity instance is null and the new entity array is not null. In such a case the `ResourceLoader` calls a function `add*` on all services, since we just added the entity. The new entity is saved into the database.
- The original entity instance is not null and the new entity array is not null. The function `update*` is called and the new entity instance is saved in the database.
- The original entity instance is not null and the new entity array is null. The function `remove*` is called and the entity instance is removed from the SciCol database.

3.4.7 Webhooks

In case there is a need for a bi-directional synchronization, the webhooks come to the word. When a change occurs in any resource, e.g. Trello, the resource makes a GET request on pre-defined address to notify, that some changes have occured.

For this purpose I have created `WebhookController`, that takes care about this all incoming requests. I chose the a `Controller`, since a method taking care about the triggered webhooks will have to accept HTTP requests.

3.4.7.1 Trello webhooks

Trello webhooks are accessible only through its API [76]. In order to assign webhook to any Trello entity (in case of Trello boards, lists and cards), a call containing `modelId` (`idBoard`, `idList` or `idCard`) and webhook callback address must be made. For this purpose there is the `createWebhook` of class `Trello`. This method is called always, when any new Trello entity is added. There is no need to delete the webhook when deleting the entity, since it is automatically done by Trello itself.

Whenever something in Trello changes (e.g. dead line or name of a card), Trello calls `http://scicol.fit.cvut.cz/trelloCallbacks/`. In SciCol this invokes the method `trelloWebhook` of `WebhookController`, which saves all the changes and initiates changes in other resources.

3.4.7.2 Google webhooks

Google API offers webhooks as well, just under the name push notifications [77]. Originally, I wanted to implement them as well, however, Google

push notifications require a callback adress with HTTPS. However, so far the SciCol server as well as my dev server do not have HTTPS yet. I prepared `createWebhook` method of `GoogleCalendar`, which is called, but now still has empty body. The `WebhookController` method for `GoogleCalendar` callbacks still has to be implemented.

Chapter 4

Testing

4.1 Testing environment

Symfony offers concept so-called environments [78]. This feature helps to divide configuration of Symfony packages and system variables for different environments. By default there are 3 different environments: prod for production, dev for development and test for testing.

For testing purposes I created a package of Symfony fixtures [79], that can be found in `App\Fixtures`. When the fixtures are loaded, the loader purges the database and inserts entities defined in the fixtures. These data are then used in the tests. For test purposes I created a separate database `scicol_test` (production and development use database `scicol`).

In order to automate everything I created a bash script `tests\runtest.sh` that runs migrations if any new, than loads the fixtures and then executes all defined tests.

4.2 Unit testing

Symfony has integrated tests for both, unit testing as well as functional testing [80]. Unit testing tests a single functionality, that do not need any other interaction with other resources, e.g., database.

In SciCol there is a limited set of functional tests, since most of the system backend logic is relatively complex and cannot be covered by simple unit tests.

Unit tests cover testing of expected behavior of entities, e.g., for `ProjectInvolved` it tests whether you do not want to assign two *Prinicipal Investigators* to a `Project` or for `Task`, that you are not trying to add `ProjectInvolved` belonging to another project.

4.3 Functional testing

Functional tests are more complex than unit tests and use various different resources, such as database [80].

For functional testing, that requires also `enityManager` to load the entities from the database, a Symfony `Kernel` is needed. I used the same `Kernel` as in the production withou any adjustments.

I divided functional tests into 2 logical parts: testing of serialization and testing of resources.

4.3.1 Testing of serialization

`Serializer` does a quite complex operations and therefore needs to be tested thoroughly.

The tests cover the basic functionality of `Serializer` by loading various entities from database, their serialization followed immediately by their deserialization and comparison with the original entities.

It also test storing of new entities recieved from the front end into database. The front end assigns to new entities, that do not have any database id yet, negative ids. The `Serializer` than should store entities of the same type with same negative id as one entity.

4.3.2 Testing of resources

All the resources are working with APIs. It is definitely not a good idea to do any automated tests against live API. Therefore I used PHP-VCR [81] library, that records all the HTTP communication during the first iteration of the test and replays it during the next iterations.

In the tests, various entities are added to resources (e.g. when a `Project` is added to `GoogleCalendar` resource, a new Google Calendar is created) and the database is checked for presence of all `EntityResourceRelator` entities.

4.4 UI testing

User Interface is an important part of the application as well and therefore should be tested as well.

4.4.1 Manual walk-through

Since the application is based on integration with 3rd party tools through various APIs, the static tests with recorded HTTP communication might not reflect edge cases. Therefore I have decided to make a manual walk-through and make tests against live API. I was checking all 3rd party tools, whether

they reflect the changes in SciCol as intended. Before executing the walk-through, I have initialized database with fixtures located in `src\Fixtures`.

Here is the list of exact steps I made:

- I started by creating a new project. I selected template `Article`, that is in `TemplateFixture`. I filled in the project name, description and set visibility to *Only organization members*. I added one collaborator and to task *Do brainstorming* I added deadline and assigned myself. I saved the project and checked, if a new calendar with 1 event was made in `Google Calendar`, if a new board with appropriate lists and tasks was made in `Trello` and corresponding `Google Drive` structures were made.
- Then in *My projects* I accepted the project. During the first run I added all usernames, during the second run only `Trello` and `Google calendar`. I checked, whether I was added correctly to the task in `Trello` and `Google Calendar` and if I could access `Google Drive`.
- Then I viewed the project. Firstly I clicked on *General settings*, changed the project name and checked whether the project name was changed in all 3rd party tools.
- Afterwards I opened the *Brainstorming* phase and changed the deadline (*DL*), turn on and off *inCalendar* and changed the deadline again. During this I was checking in `Trello` if the deadline changes in corresponding card and in `Google Calendar`, whether the event appears and disappears as it should.
- In *Collaborators* I added Marek Suchánek as a collaborator and turned on and off *Project Administrator* option.
- In the phase modal window I removed and added `Google Document` to `Phase` and to `Task`.
- I added and removed myself to Task and observed, whether I will be added and removed to the card in `Trello` and event in `Google Calendar`.
- I pinned and unpinned the project.
- I suspended and unsuspended the project, afterwards I archived it.

4.4.2 Problems found during manual walk-through

During first few iterations of the manual walk-through I was getting large ammounts of internal server error. Thanks to the testing I found following bugs:

- The `Serializer` was not serializing correctly properties of datetime type as well as the `JSerializerCommand` was not initializing them at all.

- The mechanism of saving changed entities with classes inherited from `Saver` was completely wrong, since I was trying to compare entities deserialized with `Serializer` with those loaded from the database. I found out, that any two instances of any doctrine entity are synchronized.
- I was doing changes in Google services wrongly. E.g., when you want to change a file name of `Google Document`, you have to create a brand new `Google Document` instance and set only the values, that you want to update.
- There was no mechanism of checking, whether the user has saved the username for given service or not.
- There was no check, if there already exists an equivalent of entity in 3rd party tools (e.g. `Trello` card for a `Task`), so in certain situations it happened, that one entity had more equivalents.

All of the described problems were fixed.

Chapter 5

Further possibilities

The scope of this thesis is relatively large and many more hundreds of hours could be spend on developing the system. In the following sections I want to outline the possibilities of further development based on my ideas as well as ideas provided by others.

5.1 New external services

Understandably, there are many tools that the researchers might use during the publication process. It is almost impossible to satisfy everybody, however thanks to emphasis on extendability it is relatively uncomplicated to implement a new `Resource` representing a new external tool.

In the application I included four different external tools: Trello, Google Calendar, Google Documents and Google Drive. It might be beneficial to implement connection to other external tools of the same type (calendar, shared drive, task system, document) or a brand new type, such as GIT.

5.2 Tighter integration with services

Many researchers use only email or some type of shared documents and are so used to it, that they refuse to learn anything new.

Therefore it might be beneficial to make some tighter integration with external services, e.g., include messages from Gmail directly into the application.

Conclusion

Benefits and omparison with other applications

In comparison with other applications, that are supposed to guide collaborators throughout the whole process of creating a publication, such as ResearchGate or Open Science Framework mentioned in section 1.2, SciCol is not trying to implement the whole process, however it relies on external services that the collaborators ought to already be used to and is not trying to do it better than Google.

It is also relatively straight forwared to use.

Dissuades of application

Among the dissuades of current implementation belongs, in my opinion, the fact, that there is a lack of implemented connections to external services.

Fullfilment of assignment

During designing and implementation I was following the assignment. In the following list I specify fullfilment of every point of the assignment:

- **Analyze types of scientific publications and the process of their preparation and creation up to publishing and indexing. Specify requirements for the system.**
- **Briefly research current solutions for collaboration upon publications.** In the chapter 1.2 I summarized the most used applications during creating and publishing an article upon my own research as well as upon interviews with people from the faculty.

- **Design a system supporting the process of publishing activities. During the designing, take into account extensibility of the system and consider integrations of suitable external services.** I designed the system with the help of OntoUML (see section 2.4) along with usage and extendability of external services as described in subsection 2.3.3.
- **Implement the designed system and test it thoroughly. Justify selection of the programming language and other technologies (e.g. a web framework).** The language and framework were selected and justified already during the designing phase (see section 2.2). Then the system was implemented (chapter 3) and tested (chapter 4). I admit lack of user testing as well as not fulfilling the requirements to pass project ownership and start a new project based on an old one. However, I plan to continue working on the project even after submission of this thesis.
- **Evaluate benefits of the system for users and compare it with alternative applications.** Please see the upper part of this chapter.

Bibliography

[1] Öchsner, A. *Introduction to Scientific Publishing*. ISBN 978-3-642-38646-6, Springer, 2013, 9-13 pp.

[2] Schöpfel, J. *Grey Literature in Library and Information Studies*. ISBN 978-3-598-11793-0, De Gruyter, 2010, 2-3 pp.

[3] International Serial Standard Serial Number International Center. *What is an ISSN?* [cited 2019-04-24]. Available from: `https://www.issn.org/understanding-the-issn/what-is-an-issn/`

[4] Bradley, P. Book numbering: the importance of the ISBN. *The Indexer*, volume 18, no. ISSN 1756-0632, 1992. Available from: `https://www.theindexer.org/files/18-1/18-1_025.pdf`

[5] Kochel, T. Impact Factor Distortions. *Science*, volume 340, no. ISSN 1095-9203, 2013: p. 787. Available from: `https://science.sciencemag.org/content/340/6134/787`

[6] Somnath Saha, D. A. C., Sanjay Saint. Impact factor: a valid measure of journal quality? *Journal of the Medical Library Association*, volume 91, 2003: pp. 42–46. Available from: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC141186/`

[7] Georgetown University Library. *What's the Difference between Scholarly Journals and Popular Magazines?* 2019, [cited 2019-05-07]. Available from: `https://www.library.georgetown.edu/tutorials/scholarly-vs-popular`

[8] Ellison, C. *McGraw-Hill's Concise Guide to Writing Research Papers*. ISBN 978-0-07-162990-4, The McGraw-Hill's Companies, 2010.

[9] Anthony C. Winkler, J. R. M. *Writing the research paper: A Handbook*. ISBN: 978-0-495-79964-1, Wadsworth, Cangage Learning, 2012.

[10] Jacalyn Kelly, K. A., 1 Tara Sadeghieh. Peer Review in Scientific Publications: Benefits, Critiques, & A Survival Guide. *Journal of the International Federation of Clinical Chemistry and Laboratory Medicine*, volume 25, no. ISSN 1051-2292, 2014: pp. 227–243. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4975196/

[11] (U.S.)., N. R. C. Electronic Message Systems for the U.S. Postal Service: A Report. Technical report, National Academy Of Sciences, Washington D.C., 1976.

[12] Google. *Group emails into conversations*. [cited 2019-05-07]. Available from: https://support.google.com/mail/answer/5900?co=GENIE.Platform%3DDesktop&hl=en

[13] Rewardly, Inc. *Features - Streak*. [cited 2019-05-07]. Available from: https://www.streak.com/features

[14] Slack. *Where work happens | Slack*. [cited 2019-05-07]. Available from: https://slack.com/intl/en-cz/?eu_nc=1

[15] WhatsApp Inc. *WhatsApp Features*. [cited 2019-05-07]. Available from: https://www.whatsapp.com/features/

[16] Telegram. *Telegram F.A.Q.* [cited 2019-05-07]. Available from: https://telegram.org/faq

[17] Fleep Technologies. *Fleep*. [cited 2019-05-07]. Available from: https://fleep.io/features

[18] Atlassian. *About Trello*. [cited 2019-05-07]. Available from: https://trello.com/about

[19] Software Freedom Conservancy. *About - Git*. [cited 2019-05-07]. Available from: https://git-scm.com/about

[20] GitHub, Inc. *About GitHub*. [cited 2019-05-07]. Available from: https://github.com/about

[21] GitLab. *The first single application for the entire DevOps lifecycle - GitLab*. [cited 2019-05-07]. Available from: https://about.gitlab.com/

[22] Stack Exchange Inc. *Tour - Stack Overflow*. [cited 2019-05-07]. Available from: https://stackoverflow.com/tour

[23] ResearchGate GmbH. *ResearchGate*. [cited 2019-05-07]. Available from: https://www.researchgate.net/about

[24] Center For Open Science. *OSF*. [cited 2019-05-07]. Available from: https://osf.io/

[25] Franklin, C. 10 Hot Programming Languages To Build Web Apps. *InformationWeek*, 2016, [cited 2019-05-07]. Available from: https://www.informationweek.com/software/10-hot-programming-languages-to-build-web-apps/d/d-id/1327471

[26] Shiotsu, Y. Web Development 101: Top Web Development Languages to Learn in 2018. *Upwork Blog*, 2017, [cited 2019-05-07]. Available from: https://www.upwork.com/blog/2017/11/top-web-development-languages-2018/

[27] James, N. 8 Best Programming Languages to Develop an Ecommerce Website in 2017. *WebEcommerce Pros*, volume April, 2017, [cited 2019-05-07]. Available from: https://www.webecommercepros.com/best-programming-language-ecommerce-website-development-2017

[28] Oracle. *The Jave Tutorials*. [cited 2019-05-07]. Available from: https://docs.oracle.com/javase/tutorial/

[29] Python Software Foundation. *Python 3.7.3 documentation*. [cited 2019-05-07]. Available from: https://docs.python.org/3/

[30] PHP Group. *PHP - HyperText Preprocessor*. 2019, [cited 2019-05-07]. Available from: https://www.php.net/

[31] Mozilla Developer Network. *JavaScript Documentation*. [cited 2019-05-07]. Available from: https://devdocs.io/javascript/

[32] W3Schools. *HTML Reference*. [cited 2019-05-07]. Available from: https://www.w3schools.com/tags/

[33] Stack Exchange Inc. *Stack Overflow - tags*. [cited 2019-05-07]. Available from: https://stackoverflow.com/tags

[34] Hornostaiev, M. Java, Python, and PHP: Which is Better for Server Backends? *Erminesoft*, volume February, 2019, [cited 2019-05-07]. Available from: https://erminesoft.com/java-python-and-php-which-is-better-for-server-backends/

[35] Nette Foundation. *Nette Framework*. [cited 2019-05-07]. Available from: https://nette.org/en/

[36] Morris, W. 8 Best PHP Frameworks for Web Developers. *Hostinger Tutorials*, 2019, [cited 2019-05-07]. Available from: https://www.hostinger.com/tutorials/best-php-framework

[37] Machač, M. 10 nejlepších PHP frameworků pro vývojáře. *interval.cz*, 2015, [cited 2019-05-07]. Available from: https://www.interval.cz/clanky/10-nejlepsich-php-frameworku-pro-vyvojare/

[38] Reigns, S. 11 Best PHP Frameworks for Modern Web Developers in 2019. *Coders Eye*, 2019, [cited 2019-05-07]. Available from: https://coderseye.com/best-php-frameworks-for-web-developers/

[39] Njenga, A. 10 Popular PHP frameworks in 2019. *Raygun*, volume November, 2018, [cited 2019-05-07]. Available from: https://raygun.com/blog/top-php-frameworks/

[40] Goel, A. 10 Best JavaScript Frameworks to Use in 2019. *Hackr.io*, volume March, 2019, [cited 2019-05-07]. Available from: https://hackr.io/blog/10-best-javascript-frameworks-2019

[41] Smith, J. 9 Popular JavaScript Frameworks for 2019. *Raygun*, volume January, 2019, [cited 2019-05-07]. Available from: https://raygun.com/blog/popular-javascript-frameworks/

[42] Oracle. *Database Licensing*. [cited 2019-05-07]. Available from: https://www.oracle.com/assets/databaselicensing-070584.pdf

[43] Microsoft Corporation. *SQL Server 2017 Licensing Data Sheet*. [cited 2019-05-07]. Available from: https://download.microsoft.com/download/B/C/0/BC0B2EA7-D99D-42FB-9439-2C56880CAFF4/SQL_Server_2017_Licensing_Datasheet.pdf

[44] DB-Enginges Ranking. Available from: https://db-engines.com/en/ranking

[45] MariaDB. *MySQL vs. MariaDB: Comprehensive Differences*. [cited 2019-05-07]. Available from: https://mariadb.com/kb/en/library/mariadb-vs-mysql-features/

[46] SensioLabs. *Symfony - AbstractNormalizer*. 2019, [cited 2019-05-07]. Available from: https://www.tutorialspoint.com/symfony/symfony_architecture.htm

[47] SensioLabs. *Symfony - Databases and the Doctrine ORM*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/doctrine.html

[48] SensioLabs. *Symfony - The HttpFoundation Component*. 2019, [cited 2019-04-26]. Available from: https://symfony.com/doc/current/components/http_foundation.html#creating-a-json-response

[49] SensioLabs. *Twig - introduction*. 2019, [cited 2019-05-07]. Available from: https://twig.symfony.com/doc/2.x/intro.html

[50] Facebook Inc. *Tutorial: Intro to React*. [cited 2019-05-07]. Available from: https://reactjs.org/tutorial/tutorial.html#what-is-react

[51] Google. *Google Docs: Free Calendar App for Personal Use*. [cited 2019-05-07]. Available from: https://www.google.com/calendar/about/

[52] Google. *Google Docs: Free Online Documents for Personal Use*. [cited 2019-05-07]. Available from: https://www.google.com/intl/en_US/docs/about/

[53] Google. *Google Docs: Free Cloud Storage for Personal Use*. [cited 2019-05-07]. Available from: https://www.google.com/drive/

[54] Atlassian. *Introduction - Trello API*. [cited 2019-05-07]. Available from: https://developers.trello.com/reference

[55] Google. *Google APIs Explorer*. [cited 2019-05-07]. Available from: https://developers.google.com/apis-explorer/

[56] Fowler, M. *UML Distilled, Third Edition*. ISBN 0-321-19368-7, Addison-Wesley, 2004.

[57] Ing. Zdeněk Rybola, P. *Towards OntoUML for Software Engineering: Transformation of OntoUML into Relational Databases*. Dissertation thesis, Czech Technical University in Prague, 2017.

[58] Konceptuální modelování. [cited 2019-05-07]. Available from: https://moodle.fit.cvut.cz/course/view.php?id=32

[59] Guilizzoni, P. *What are wireframes*. Balsamiq Studios, LLC, [cited 2019-05-07]. Available from: https://balsamiq.com/learn/resources/articles/what-are-wireframes/

[60] Evolus. *Home - Pencil project*. [cited 2019-05-07]. Available from: https://pencil.evolus.vn/

[61] Siddiqui, A. Authentication vs. authorization. *Data Driven Investor [online]*, September 2018, [cited 2019-05-07]. Available from: https://medium.com/datadriveninvestor/authentication-vs-authorization-716fea914d55

[62] SensioLabs. *Symfony - Security*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/security.html

[63] Wolfram Research, I. Hash Function. 2019, [cited 2019-05-07]. Available from: http://mathworld.wolfram.com/HashFunction.html

[64] SensioLabs. *Symfony - Security User Providers*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/security/user_provider.html

[65] SensioLabs. *Symfony - Authorization*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/components/security/authorization.html

[66] SensioLabs. *Symfony - The Serializer Component*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/components/serializer.html

[67] SensioLabs. *Symfony - AbstractNormalizer*. 2019, [cited 2019-05-07]. Available from: https://github.com/symfony/symfony/blob/4.2/src/Symfony/Component/Serializer/Normalizer/AbstractNormalizer.php

[68] PHP Group. *PHP - Introduction*. 2019, [cited 2019-05-07]. Available from: https://php.net/manual/en/language.types.intro.php

[69] PHP Group. *PHP - json_encode*. 2019, [cited 2019-05-07]. Available from: https://www.php.net/manual/en/function.json-encode.php

[70] SensioLabs. *Symfony - Console Commands*. 2019, [cited 2019-05-07]. Available from: https://symfony.com/doc/current/console.html

[71] PHP Group. *PHP - Traits*. 2019, [cited 2019-05-07]. Available from: https://www.php.net/manual/en/language.oop5.traits.php

[72] Google. *Getting Started - API Client Library for PHP (Beta)*. [cited 2019-05-07]. Available from: https://developers.google.com/api-client-library/php/start/get_started

[73] Google. *Share files and folders | Drive REST API | Google Developers*. [cited 2019-05-07]. Available from: https://developers.google.com/drive/api/v3/folder

[74] *Guzzle, an extensible PHP HTTP client*. [cited 2019-05-07]. Available from: https://github.com/guzzle/guzzle

[75] Maguire, S. *A php client for consuming the Trello API*. [cited 2019-05-07]. Available from: https://github.com/stevenmaguire/trello-php

[76] Atlassian. *Trello - webhooks*. [cited 2019-05-07]. Available from: https://developers.trello.com/page/webhooks

[77] Google. *Push Notifications*. [cited 2019-05-07]. Available from: https://developers.google.com/calendar/v3/push

[78] SensioLabs. *How to Master and Create new Environments*. [cited 2019-05-07]. Available from: https://symfony.com/doc/current/configuration/environments.html

[79] SensioLabs. *DoctrineFixturesBundle*. [cited 2019-05-07]. Available from: https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html

[80] SensioLabs. *Symfony - testing*. [cited 2019-05-07]. Available from: https://symfony.com/doc/current/testing.html

[81] PHP-VCR. *PHP-VCR | Record HTTP interactions while testing*. [cited 2019-05-07]. Available from: https://php-vcr.github.io/

Appendix A

List of used abbreviations

CLI Command Line Interface

GUI Graphical User Interface

UI User Interface

CI Continuous Integration

WYSIWIG What You See Is What You Get

PHP Hypertext Preprocessor

HTML Hypertext Markup Language

CSS Cascade Style Sheets

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secured

SQL Structured Query Language

ORM Object-Relational mapper

JSON JavaScript Object Notation

API Application Programming Interface

UFO Unified Foundational Ontology

CTU Czech Technical University

UML Unified Modelling language

SSO Single Sign On

YAML YAML Ain't Markup Language

ISSN International Standard Serial Number

ISBN International Standard Book Number

Appendix B

Content of attached SD card

- `readme.txt` ..... short description of sd card content
- `src`
  - `impl` ..... source code of the implementation
  - `thesis` ..... source code of this thesis LaTeX
- `text` ..... text of this thesis
  - `thesis.pdf` ..... this thesis in PDF format

Appendix C

Installation manual

The application was developed under Debian 9.8 Stretch and this manual is intended for an installation on a server running on Linux.

C.1 How to get the source code

The source code of this project is available on GitLab of the Czech Technical University, namely under `https://gitlab.fit.cvut.cz/jiraspe2/scicol`. In case you would like to install the application your server, please contact me vie email `jiraspe2@fit.cvut.cz` and send me your public SSH key.

Once the SSH key is present in the GitLab, clone the repository `git@gitlab.fit.cvut.cz:jiraspe2/scicol.git` into any folder on the server, e.g. /home/scicol/. For further explanation, let's consider the folder with the repository as the `root_path`.

C.2 Dependencies

The application needs following programs for its run:

- PHP 7.2
- MySQL 15.1 or any other relational database compatible with Symfony ORM.
- Apache2 server
- yarn and node.js
- composer

After installation of above mentioned programs it is necessary to set their configuration.

In the database machine create 2 databases called `scicol` and `scicol_test` and create a user with all privileges granted for these databases. In files `root_path/.env` and `root_path/.env.test` set the configuration your databases.

In Apache 2 server you will need to adjust you configuration file, on Debian usually `/etc/apache2/sites-available/000-default.conf`. Set the configuration as following:

```
<VirtualHost *:80>
    ServerName domain.tld
    ServerAlias www.domain.tld

    DocumentRoot root_path/public
    <Directory root_path/public>
        Order Allow,Deny
        AllowOverride All
        Allow from All
    </Directory>
    ErrorLog /var/log/apache2/project_error.log
    CustomLog /var/log/apache2/project_access.log combined
</VirtualHost>
```

In some Apache2 versions you will need to replace `Order Allow,Deny` with `Require all granted`. Also enable mod rewrite.

C.3 Installation

After installing all necessary programs, go to `root_path` and do following steps:

1. Download and install all necessary PHP dependencies by calling `composer install`.

2. Call `yarn install`.

3. And then `yarn encore dev`, which encores the JavaScript front-end.

4. Run all migrations by calling `php bin/console doctrine:migrations:migrate`.

5. Run tests by running the bash script `tests/runtests.sh`. Be sure you run them from `root_path` folder.

6. In order to get a super administrator into database, load the `UserFixture` by calling `php bin/console doctrine:fixtures:load --fixture=/src/DataFixtures/User` Adjust it to your needs.

Now you should be able to run the server.

C.4 Updating

For updating always pull the newest content from `master` branch on GitLab. Then repeat steps 1-5 from previous section.