FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Homework management system in Jupyter
Student:	Bc. Dmitry Vanyagin
Supervisor:	doc. Ing. Štěpán Starosta, Ph.D.
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 goal of the thesis is to design and implement a homework management system using Jupyter notebook system. The system may be composed of open source components and must fulfill the following requirements:

1) Manage (distribute, collect) homeworks for specific courses. A homework is a single Jupyter notebook with possible data file attachments. The distribution and collection of homeworks should be modifiable to fit the workflow of a course.
2) Provide feedback support: grading and comments of individual cells. The feedback should be in a form of a thread, i.e., with multiple possible responses.
3) The management system should have a possibility to be provided for students without the need of local installation of any software.
4) The system and its usage should be in maximum compliance with other faculty systems (the login should be done using Usermap credentials, the system should use student's list from KOS, it should provide means to export the grading to a grading system).

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 8, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Homework management system in Jupyter

Dmitry Vanyagin Bc.

Department of Software Engineering
Supervisor: Štěpán Starosta

February 10, 2020

Acknowledgements

I would like to thank my friends and colleagues for support and my supervisor Štěpán Starosta for help and guidance during work on this thesis.

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 Prague on February 10, 2020 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vanyagin, Dmitry. *Homework management system in Jupyter.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato práce popisuje proces analýzy a implementace správy úkolů v systému Jupyter. Hlavním cílem je umožnit vyučujícímu připravit, distribuovat a sbírat zadání ve formě Jupyter sešitů. Výsledkem práce je fungující prototyp splňující tyto požadavky. Práce důkladně popisuje všechny kroky vývoje: analýzu, implementaci, konfiguraci a nasazení.

Klíčová slova JupyterHub, Kubernetes, nbgrader, Systém

Abstract

This thesis describes the process of analysis and implementation of the Homework Management system in Jupyter. The main goal of this system is to allow teachers to prepare, distribute and collect homework assignments, which are prepared as Jupyter notebooks. Result of this work is a working prototype of the system that fulfills the requirements. All the steps of analysis, implementation, configuration, and deployment are thoroughly described.

Keywords JupyterHub, Kubernetes, nbgrader, System

Contents

Citation of this thesis vi

1 Introduction **1**
1.1 Motivation and goals 1
1.2 Requirements 2
1.2.1 Functional requirements 2
1.2.2 Non-functional requirements 2

2 Analysis and design **3**
2.1 Distribution 3
2.1.1 Manual installation 3
2.1.2 Virtualization 4
2.1.3 Docker 6
2.1.4 Centralization 8
2.2 Authentication 9
2.2.1 OAuth 2.0 9
2.3 KOS 15
2.4 Feedback module 20
2.4.1 Notebook file format 20
2.5 Infrastructure 26
2.5.1 How the Subsystems Interact 26
2.5.2 User login 27
2.5.3 Kubernetes 28
2.6 Managing the homeworks 30
2.6.1 Teacher's use cases 30
2.6.2 Student's use cases 33
2.6.3 nbgrader 35

3 Implementation **39**
3.1 Authentication 40

3.1.1 Making custom Authenticator . . . 41
3.1.2 Making custom Docker image . . . 44
3.2 Homework management . . . 45
3.2.1 Feedback module . . . 45
3.2.2 nbgrader installation . . . 47
3.3 Kubernetes . . . 48
3.3.1 Helm . . . 48
3.4 JupyterHub . . . 49
3.4.1 KOSAPI . . . 49
3.4.2 Spawner . . . 50
3.4.3 Deployment . . . 54
3.4.4 Testing . . . 55

4 Conclusion **57**

Bibliography **59**

A User manual **63**
A.1 Authentication . . . 63
A.1.1 Login . . . 63
A.1.2 Logout . . . 63
A.2 Student's guide . . . 64
A.2.1 Working with assignments . . . 64
A.3 Teacher's guide . . . 66
A.3.1 Managing assignments . . . 66

B Glossary **69**

C Acronyms **71**

D Contents of enclosed flash drive **73**

List of Figures

2.1 Using virtual machine 5
2.2 Using Docker containers 7
2.3 Using Hub for distribution 8
2.4 OAuth2.0 protocol flow 11
2.5 Authorization code flow 12
2.6 GET /students/{studyCodeOrId} response body 18
2.7 GET /teachers/{usernameOrId}/courses response body 19
2.8 Example of top level schema 20
2.9 Example of cell schema 21
2.10 Example of markdown cell schema 21
2.11 Example of code cell schema 22
2.12 Reserved cell metadata keys 22
2.13 Feedback module use cases 24
2.14 Schema of the Feedback module 24
2.15 Minimal code for Jupyter Notebook extension 25
2.16 Installation of an extension 25
2.17 JupyterHub subsystems 27
2.18 Management of homework use cases 36
2.19 Directory structure of a teacher 37
2.20 Directory structure of a student 37
2.21 nbgrader diagram 38

3.1 Authentication process 41
3.2 Custom spawner sequence diagram 53
3.3 Jupyter Homework Management system 56

A.1 Fetching assignment 64
A.2 Downloaded assignment 64
A.3 Assignment list expanded 65
A.4 Assignment passed tests 65

A.5 Formgrader extension 66
A.6 Creating assignment 66
A.7 Assignment toolbar 67
A.8 Generate student version 67

Chapter 1

Introduction

With the growing amount of easily accessible computation power, more and more researchers are enabled to use every day computers for processing of data.

There is a wide selection of software tools that allow people to use their favorite programming language for it. One of the most popular solutions on the market is Jupyter.[1] Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages.[2]

Jupyter Notebook - one of their solutions, is an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.[1]

1.1 Motivation and goals

The task of the thesis is to design and implement a homework management system using Jupyter. Such a tool would allow teachers to efficiently create, distribute and collect homework. Students will have immediate access to their assignment and will be using features of JN to better understand the problem by analyzing and visualizing it.

Moreover, another benefit of it is that students will be able to receive feedback from the teacher and follow a discussion thread. All of that plus seamless integration into existing university information system by using Usermap credentials and KOSAPI.

The main goal will be to design a set of small applications (plugins) that would solve individual requirement while being decoupled from the Jupyter Notebook. All the plugins should be extendable and replaceable in case of future changes in requirements.

1.2 Requirements

In this section listed the set of functional and non-functional requirements, which were extracted from the thesis assignment specification.

1.2.1 Functional requirements

The set of functional requirements consists of processes which needs to be implemented in the Jupyter ecosystem in order for teachers and students to manage homework assignments.

- Teacher should be able to manage (create, distribute and collect) homework assignments for specific courses.
- Teacher should be able to export list of grades from the system.
- Students should be able to fetch, implement and submit the homework assignment.
- System should allow homeworks to be prepared as Jupyter notebook file with possible data attachments.
- Teacher should be able to grade the assignment and provide feedback to a student. The feedback should be in a form of a thread.
- User's authentication should be done using University system.
- List of subjects for a logged in teacher should be fetched from University API.
- List of students for a given subject should be fetched from University API.

1.2.2 Non-functional requirements

- System should be scalable to handle different workloads.
- The management system should have a possibility to be provided to a students without need of local installation of any software.
- System should be fast and responsive.
- System should be composed of open source components.

Chapter 2

Analysis and design

This chapter will demonstrate the analysis of the requirements and derivation of the possible solution that would satisfy them. Of course there might exist multiple approaches for individual features of the application, they will be thoroughly described and compared between each other. Key topics will be analyzed:

- Distribution of software among the students
- User authentication
- Connection with university system (KOSAPI)
- Communication between student and teacher
- Export of grades from the system

2.1 Distribution

Choice of distribution highly depends on specific requirements, in this case it should be avoided to ask students to install or configure any software on their own devices. Jupyter notebook – is a solution which is built as a web application, so it could potentially be used to allow students to access it via the internet. On top of that, custom functionality (plugins) will need to be distributed, for example the one that will be connecting software with university API. Although manual installation is not an acceptable solution, it would still be a good idea to start the analysis from. By doing so, it will demonstrate the complexity of the process from beginning till the end.

2.1.1 Manual installation

Installation process of Jupyter Notebook is not trivial for regular user.[3] Main prerequisite is the Python installed on a target device, thus there could be used

some tools that ease the process of setup. One of those tools is **Anaconda**. Anaconda Distribution – is the open source platform for data science using Python and R programming languages. It allows user to manage libraries, dependencies and environments.[4] Steps to install Jupyter Notebook using Anaconda are following:

- Download Anaconda. It's recommended to download Anaconda's latest Python 3 version (currently Python 3.7).
- Install the version of Anaconda which was downloaded, following the instructions on the download page.

Another alternative for experienced Python users is to use **pip** tool. Pip – is a standard package-management system which is used to install and manage software packages written in Python. A lot of the packages can be found in the default source for packages and their dependencies - Python Package Index. Python Package Index – is a repository of software for the Python programming language. Currently it contains more than two hundred thousand projects.

In order to install Jupyter Notebook using pip, it's enough to run:

```
pip install jupyter
```

Both alternatives are good to install the basic configuration of Jupyter Notebook, but the goal is to distribute custom build of it. Among those customizations are features which fulfill other requirements, like integration with university authentication system, homework review module, etc. Those "packages" will need to be installed separately and definitely will make whole setup more complicated and time consuming.

In the following section will be discussed only approaches which somehow eliminate the need to install or configure the environment and deliver final working "bundle" to end users (students).

2.1.2 Virtualization

To minimize any custom installation, one approach could be to create a virtual machine image with whole environment installed and configured, in this case student will only need to download it from university server and run. Virtual machine is basically an emulation of whole computer system. They are based on computer architectures and provide functionality of a physical computer.

There exist multiple kinds of virtual machines:

- System virtual machine
- Process virtual machine

System virtual machine provide a substitute for real machine, which means that it is possible to run whole operating system on them. In order to achieve this **hypervisor** is used to share and manage hardware and allows multiple environments to to exist (and be isolated from one another) on the same physical machine.

The term ”virtual machine” was originally defined by Popek and Goldberg as ”an efficient, isolated duplicate of real computer machine”.[5]

The physical, ”real-world” hardware running the VM is generally referred to as the ”host”, and the virtual machine emulated on that machine is generally referred to as the ”guest”. A host can emulate several guests, each of which can emulate different operating systems and hardware platforms.

Virtual machine image that was referred to before, could be a snapshot of a hard disk which contains whole environment installed and configured. Student then could run it as a ”guest” and have immediate access to Jupyter Notebook with all the packages ready.

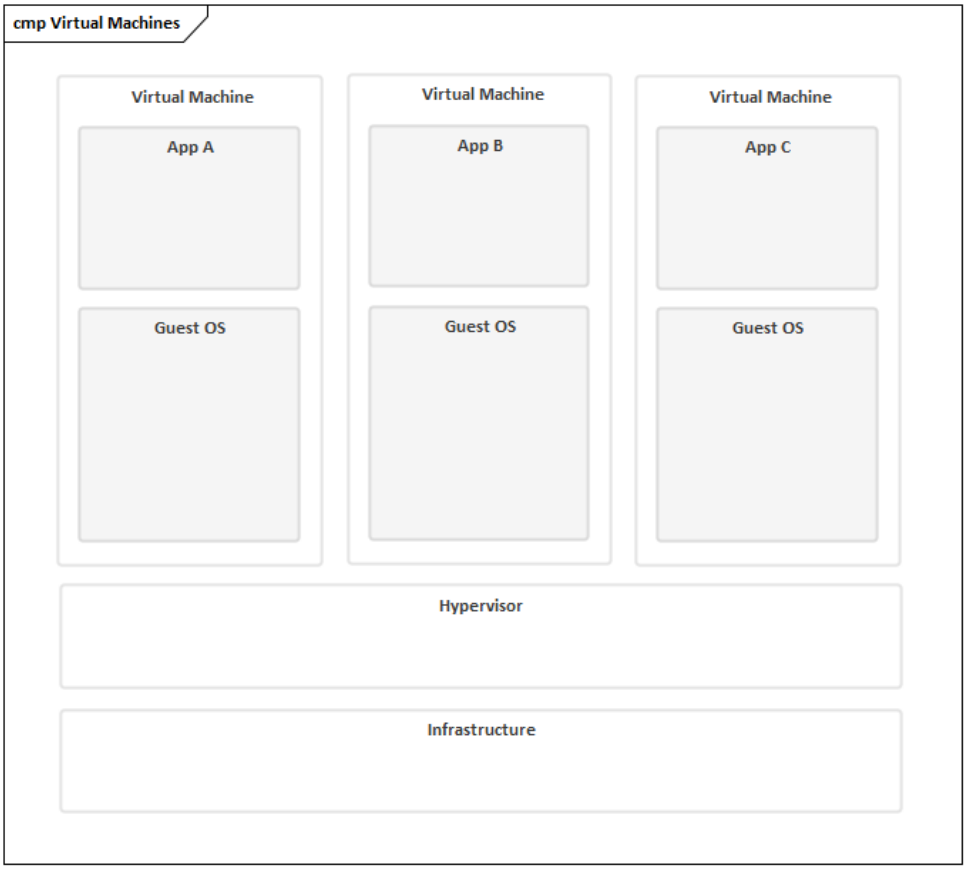

Figure 2.1: Using virtual machine

This approach has several advantages and disadvantages.
Advantages:

- Students don't need to configure environment
- For distribution it's enough to just send a link for downloading the virtual machine image

Disadvantages:

- Big size of image
- Not every OS supports virtualization
- To update the image, everyone need to delete old one and download new version, thus whole process needs to be controlled by teacher in order to ensure that every student has the most recent version.
- Students still need to install some software (virtual machine)

In order to get rid of several disadvantages, another approach could be used: **containerization**.

With containers, instead of virtualizing the underlying computer like a virtual machine (VM), just the OS is virtualized. Containers sit on top of a physical server and its host OS — typically Linux or Windows. Each container shares the host OS kernel, binaries and libraries. Shared components are read-only. Sharing OS resources such as libraries significantly reduces the need to reproduce the operating system code, and means that a server can run multiple workloads with a single operating system installation. Containers are thus exceptionally light — they are only megabytes in size and take just seconds to start. Compared to containers, VMs take minutes to run and are an order of magnitude larger than an equivalent container.

In contrast to VMs, all that a container requires is enough of an operating system, supporting programs and libraries, and system resources to run a specific program.[6]

By using containerization it will be possible to run Jupyter Notebook environment within the containers, to which only part of system resources is allocated. This approach effectively reduces the overhead compared to full virtualization, because applications do not need to be subjected to emulation or run on intermediate virtual machine.

2.1.3 Docker

Docker – is a one one type of containers which was launched in 2013 as an open source Docker Engine.

It leveraged existing computing concepts around containers and specifically in the Linux world, primitives known as cgroups and namespaces. Docker's

technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

Technology available from Docker and its open source project, Moby has been leveraged by all major data center vendors and cloud providers. Many of these providers are leveraging Docker for their container-native IaaS offerings. Additionally, the leading open source serverless frameworks utilize Docker container technology.[7]

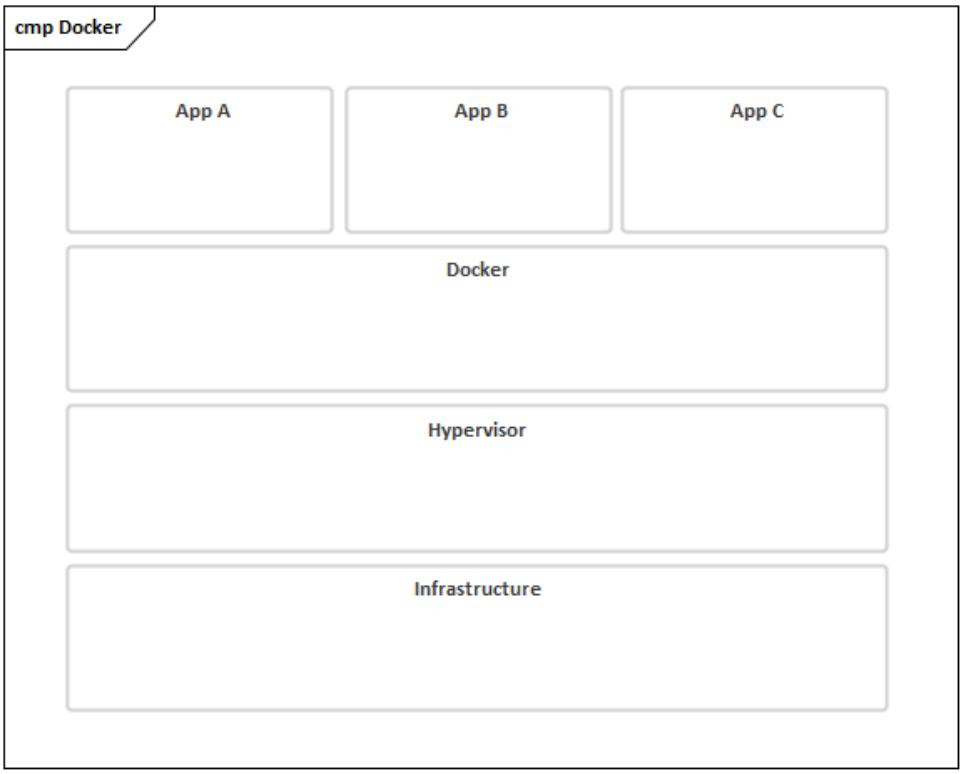

Figure 2.2: Using Docker containers

The benefit of using Docker is that it would be possible to create a custom docker image which will contain preinstalled and configured Jupyter notebook software with custom plugins and be available for students to use.

Choosing this option is effectively solving the majority of issues with previous approach, such as size of the image and burden of distribution of updated image to students, however here are still two main drawbacks:

- Students need to install docker on their computers
- Not every OS supports virtualization (thus Docker won't be supported as well)

2.1.4 Centralization

The only approach which would completely eliminate any need in installation or other manual work from students – is to create a centralized solution (hub) accessible over the internet, that will be hosting all student's notebooks.

This way it's possible to integrate authentication system of the hub with University, effectively fulfilling another requirement. Each user will be provided with JN instance that will contain all the necessary plugins and university API integration. The configuration and update of those instances will be realized by webmaster and all the students will be getting updated Jupyter notebooks after next login to the system.

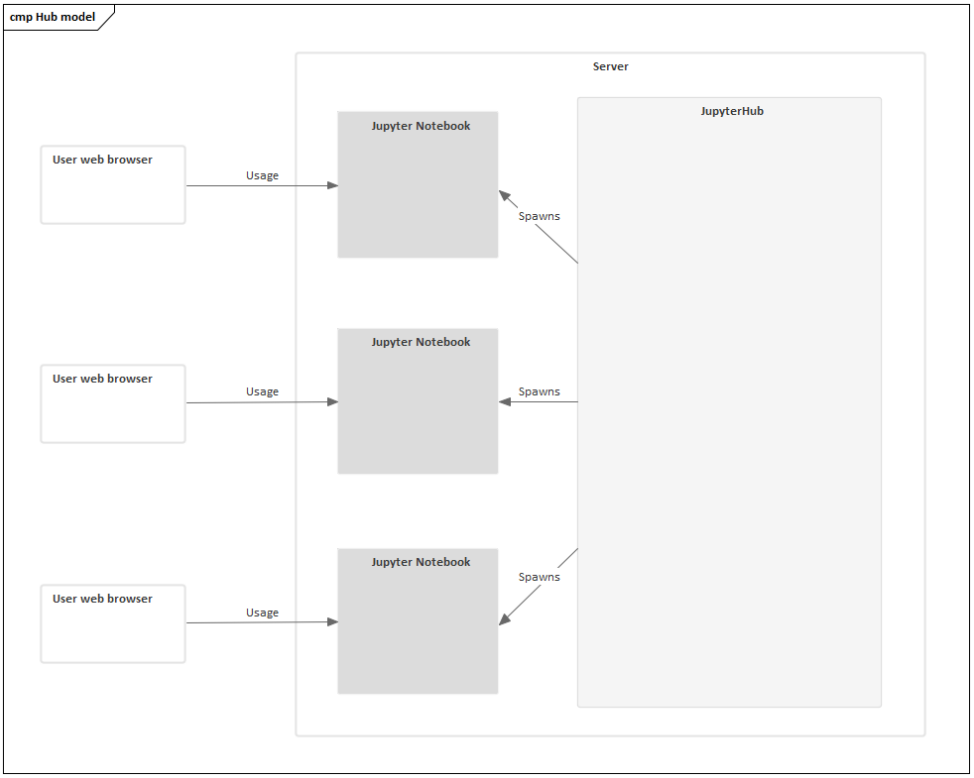

Figure 2.3: Using Hub for distribution

Unfortunately, this is solution is not perfect as well - since all the computation will be realized on server side, this hub needs to be hosted on a high performance hardware with plenty of system resources. In case of a very complex computations inside Jupyter notebook, system administrator would need to ensure that CPU cores have high frequency, for bigger amount of simultaneous computations - high number of CPU cores themselves. Bigger data

sets would need a lot of RAM to be effectively processed (to minimize read amount from hard disk).

2.2 Authentication

Another important topic to analyse is authentication. Authentication – is a process of verification of identity of a person. In this case, teacher or a student.

From the requirements, it's needed to identify individual student or a teacher because of multiple factors like:

- Teacher will be sending specific homework to specific subset of students (enrolled in the class)
- Teacher and student will be exchanging messages regarding homework solution
- Teacher will be grading homework of a student

The most popular way of authentication is a by providing a pair of login and password. Each user has it's associated username and secret password which she needs to supply to prove that account belongs to her. This pair is usually stored in database of a web server which is responsible for authentication.

Taking into account the requirements, authentication process needs to be integrated with university one, meaning that both student and teachers should be able to authenticate in the application by using their university credentials. In order to do that, it is necessary to use university server as an **identity provider**.

Identity provider – is a system that creates, maintains and manages identity information of so-called principals while providing authentication services to allowed applications. Identity providers offer user authentication as a service. Relying party applications, such as web applications, outsource the user authentication step to a trusted identity provider.

2.2.1 OAuth 2.0

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.[8]

Generally, OAuth provides to clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner.

The third party then uses the access token to access the protected resources hosted by the resource server.[9]

OAuth 2.0 defines four roles:

- Resource owner
- Client
- Resource server
- Authorization server

Resource owner – is the user who authorizes an application to access their account. The application's access to user's account is limited to the "scope" of the authorization granted.

Both **resource server** and **authorization server** are located on the side of identity provider. They host the protected user accounts (in this case university accounts) and verify the identity of the user. Once identity is verified, authorization server issues access tokens to the application. From current perspective it is possible to fully delegate authentication mechanism to identity provider hosted by the University and work with access tokens returned from it. Each access token will be used to communicate with other university systems.

Client – is the application that wants to access the user's account. Before it may do so, it must be authorized by the user.

Flow of the OAuth protocol is very straightforward. It consists of the following steps:

1. The application requests authorization to access service resources from the user
2. If the user authorized the request, the application receives an authorization grant
3. The application requests an access token from the authorization server (API) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the authorization server (API) issues an access token to the application. Authorization is complete.
5. The application requests the resource from the resource server (API) and presents the access token for authentication
6. If the access token is valid, the resource server (API) serves the resource to the application

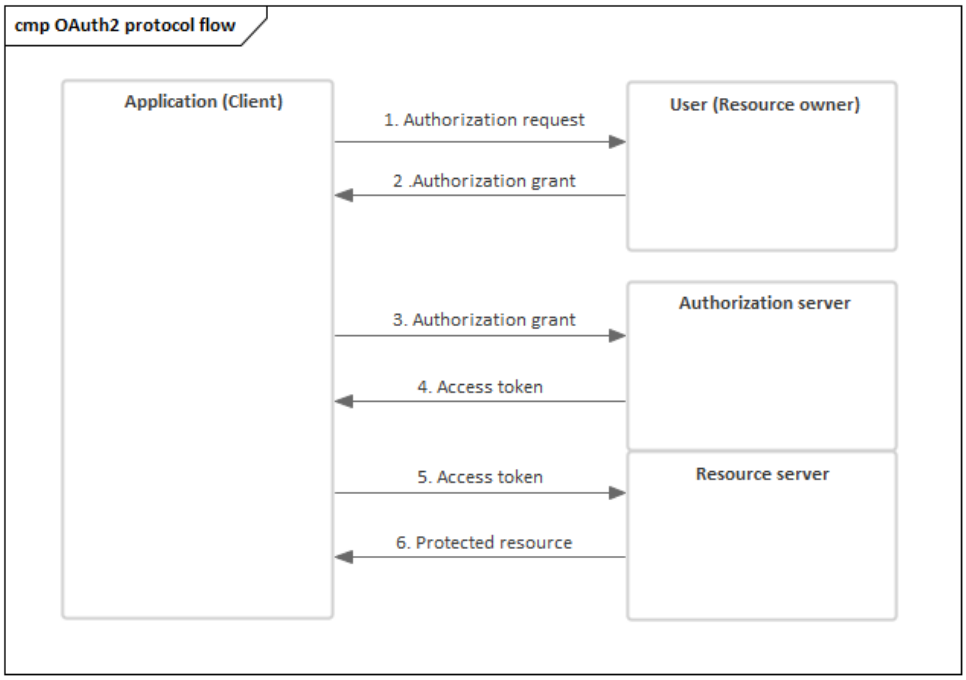

Figure 2.4: OAuth2.0 protocol flow

In order to use University identity provider, it's needed to register the application first. Registration consists of filling in some information about the application, most importantly:

- Application name
- Application website
- Callback URL

Callback URL is the URL where the service will redirect the user after they authorize or deny the application to access their account. Application should handle both cases.

After registration, it will obtain client ID and client secret – client credentials. Client ID is a public string that is used by the service API to identify the application. The client secret – is a string which is used to authenticate the identity of the application to the service (university identity provider API) when application requests to access user's account. Client secret must be kept private.

In the figure 2.4 another important entity is demonstrated – **authorization grant**. OAuth 2 defines four grant types:

- **Authorization code**: used with server-side applications
- **Implicit**: used with mobile or web applications
- **Resource owner password credentials**: used with trusted applications, such as those owned by service itself
- **Client credentials**: used with applications API access

According to the university OAuth 2 server documentation, it supports authorization code and client credentials grant types. Since the goal is to authenticate the app as a *user* (student or teacher), the only grant type which can be used is **authorization code**.

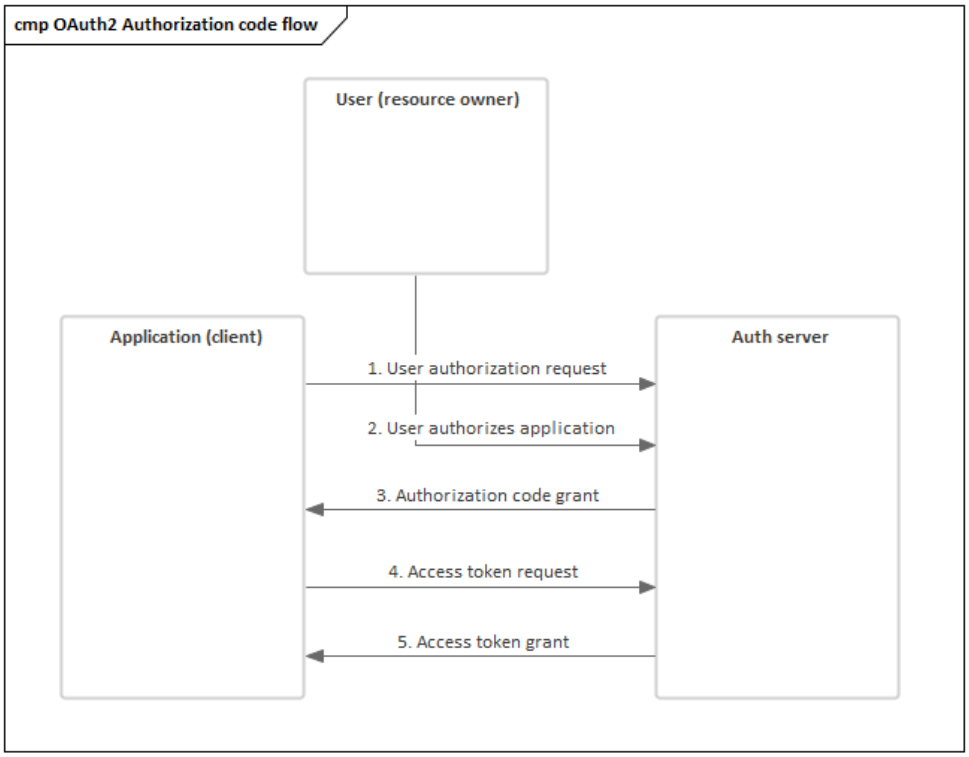

Figure 2.5: Authorization code flow

The authorization code grant type – is the most commonly used because it is optimized for server-side applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained. This is a redirection-based flow, which means that the application must be capable of interacting with the user-agent (i.e. the user's web browser) and receiving API authorization codes that are routed through the user-agent.[10]

Flow consists of following steps:

1. **Authorization code link**

   First, the user is redirected to specific authorization URL with a set of following query parameters. The user is prompted to login and then asked to approve *authorization request*.

Parameter	**Description**
response_type	A value of **code** must be used.
client_id	Indicates the client that is making the request. The client ID is obtained during the client registration.
redirect_uri	URL where users will be sent after authorization.
scope	(Optional) A space delimited set of scopes the client requests. It might be all scopes registered for the client or just a subset of them. If not provided then all registered scopes will be issued.
state	(Optional) A string value used by the client to maintain state between the request and callback. This value is included when redirecting the user back to the client. It should be used for preventing cross-site request forgery attacks.

2. After that user is redirected back to the client, to URL specified by the redirect_uri. If the user approves the authorization request, then the response contains an authorization code and state parameter (if included in the request). If the user does not approve the request, the response contains an error message.

3. **Obtaining of access token**

   After the client receives the authorization code, it may exchange it for an access token and a refresh token. This request is an HTTPS post and includes the following parameters:

Field	**Description**
code	The authorization code returned from the initial request.
grant_type	A value of **authorization_code** must be used.
redirect_uri	The URI registered with the application.

Additionally extra request header needs to be sent:

```
Authorization: Basic ZHVtbXktY2xpZW50OnRvcC1zZWNyZXQ=
```

Where value is Base64 encoded string, combined as *client_id:client_secret*

A **successful response** to this request contains the following fields:

Field	Description
access_token	The token that can be used to access resources on a resource provider.
expires_in	The remaining lifetime of the access token, in seconds.
refresh_token	A token that may be used to obtain a new access token. Refresh tokens are valid until the user revokes access.
scope	A space delimited set of scopes the token was issued for.
token_type	At this time, this field will always have the value Bearer.

4. **Refreshing of access token**

When using grant authorization code, a refresh token is returned with an access token. Once the original access token expires, the corresponding refresh token can be sent to the OAuth 2 server to obtain a fresh access token without requiring the user to re-authenticate.

To obtain a new access token this way, the client performs an HTTPs POST to URL:

```
https://auth.fit.cvut.cz/oauth/token
```

The request must include the following parameters:

Field	Description
refresh_token	The refresh token returned from the authorization code exchange.
grant_type	A value of **refresh_token** must be used.

Additionally header *Authorization* should be used again as before. In response new access_token will be returned.

2.3 KOS

Distribution of homework via the system would require another important integration with University infrastructure, it needs to be able to pull the information about students and their courses in order to know who is enrolled where. Luckily, there is a system which is used by university for management of this information: KOS.

KOS – is a system which stores information about students, courses, grades, final projects etc. Student can enroll subjects and manage her time schedule. There exist multiple ways of interaction with the system, most popular one is via web user interface. Another one is via it's API (KOSAPI).

KOSAPI – is a web service which conforms with REST architectural style, style that defines a set of constraints to be used for creating web services.

RESTful web services (services which conform with REST style) allow the client (requesting system) to access and manipulate textual representations of **web resource** by using predefined set of **stateless** operations.

Lets discuss RESTful web service architecture in more detail, the backbone of whole concept is **the resource oriented architecture** (ROA). The main term here is a **resource** – it's a something that can be stored on a computer and can be represented as a stream of bits. What makes resource a resource – it has to have at least one URI, which is the name and address of a resource.[11]

Each URI should be descriptive, for example:

```
https://example.org/users/1231
```

Could be an address of user with ID 1231. In this case resource is the the information about this user.

Another important features of ROA are addressability and statelessness. An application is addressable if it exposes the interesting aspects of its data set as resources. Since resources are exposed through URIs, an addressable application exposes a URI for every piece of information it might conceivably serve. This is usually an infinite number of URIs.

From the end-user perspective, addressability is the most important aspect of any web site or application. To fully understand what addressability is, let's look at this URI:

```
https://example.org/search?q=cat
```

It is clear that this URI could lead to the page with search results (cats). By incorporating query into URI, user can directly be navigated to this page any time.

Example of non-addressable website would have URI for the same search results page look like this:

```
https://example.org/search
```

Which would present a search page where it is needed to input ”cat” manually to get the same list of cats.

Addressability is one of the four features of ROA, second one is statelessness. Statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request. The server never relies on information from previous requests. If that information was important, the client would have sent it again in this request.

More practically, statelessness could be considered in terms of addressability. Addressability says that every interesting piece of information the server can provide should be exposed as a resource, and given its own URI. Statelessness says that the possible states of the server are also resources, and should be given their own URIs.[11]

Statelessness also brings new features. It's easier to distribute a stateless application across load-balanced servers. Since no two requests depend on each other, they can be handled by two different servers that never coordinate with each other. Scaling up is as simple as adding more servers into the load balancer. A stateless application is also easy to cache: software can cache the response returned from given URI.

Third feature of ROA called ”Hypermedia as the engine of application state” (HATEOAS).[12] This means that the current state of an HTTP “session” is not stored on the server as a resource state, but tracked by the client as an application state, and created by the path the client takes through the Web. The server guides the client's path by serving “hypermedia”: links and forms inside hypertext representations.

Fourth feature is the uniform interface of a web service. In order to control resource, it is possible to use one of the several HTTP methods, where each of them has a specific semantic:

- Retrieve a representation of a resource: **HTTP GET**
- Create a new resource: HTTP **PUT** to a new URI, or HTTP **POST** to an existing URI
- Modify an existing resource: HTTP **PUT** to an existing URI
- Delete an existing resource: HTTP **DELETE**

In the case of a GET request, the server sends back a representation in the response body. For a DELETE request, the response body may contain a status message, or nothing at all. To create or modify a resource, the client sends a PUT request that usually includes a body. The body contains the

client's proposed new representation of the resource. What data this is, and what format it's in, depends on the service.

Since the goal is to pull the data about students, teachers and their courses, GET requests are the ones to be used. More specifically, these URIs:

- **GET /students/{studyCodeOrId}** – returns information about given student. Each student can have multiple studies and it is needed to know which is the actual one (that information is part of response)
- **GET /students/{studyCodeOrId}/enrolledCourses** – returns information about enrolled courses for given student. It is needed to know which subjects are accessible to current user.
- **GET /teachers/{usernameOrId}/courses** – returns information about which courses given teacher is teaching. This endpoint is needed to determine which courses the current teacher can manage in the system.

Format from all URIs is Atom, extension of XML.[13]

Field	Data type	Description
branch	Branch	Branch of studies
department	Division	Department of student (only for PhD students)
email	string	Email
startDate	date	Date of beginning of studies
faculty	Division	Information about faculty of studies
firstName	string	First name
grade	integer	Year of studies
interruptedUntil	date	Date till when studies are interrupted
lastName	string	Last name
personalNumber	string	Personal number
programme	Programme	Program of studies
endDate	date	Date of finishing of studies
studyForm	StudyForm	Form of studies
studyGroup	integer	Number of study group
studyPlan	StudyPlan	Study plan
studyState	StudyState	State of studies
supervisor	Teacher	Supervisor of thesis (only for PhD students)
supervisorSpecialist	Teacher	Supervisor of specialist (only for PhD students)
studyTerminationReason	StudyTermination	Reason of termination of studies
titlesPost	string	Titles after name
titlesPre	string	Titles before name
username	string	Username

Figure 2.6: GET /students/{studyCodeOrId} response body

Field	Data type	Description
allowedEnrollmentCount	integer	How many times student can sign up for this subject during whole studies
approvalDate	date	Date of approval
classesLang	ClassesLang	Language of teaching
classesType	ClassesType	Type of teaching (lectures, seminars etc.)
code	string	Unique identificator
completion	Completion	Method of completion
credits	integer	How many credits
department	Division	Department
description	string	Description
homepage	string	Web page of the course
keywords	string	Keywords
lecturesContents	string	Content of lectures
literature	string	Literature list
name	string	Name
note	string	Note
objectives	string	Goals and objectives
programmeType	ProgrammeType	Type of programme
range	string	Range of teaching time
requirements	string	Requirements
season	Season	Season of the year
state	CourseState	State of subject
studyForm	StudyForm	Form of study
superiorCourse	Course	Parent course
subcourses	Course	Subcourses
tutorialsContents	string	Content of seminars
instance	Instance	Instance of course

Figure 2.7: GET /teachers/{usernameOrId}/courses response body

2.4 Feedback module

Feedback module should allow simple communication between teacher and student in the scope of a *cell*. In order to continue the analysis, it is needed first to determine how data is being store in Jupyter Notebook and what is a cell in particular.

2.4.1 Notebook file format

The official Jupyter Notebook format is defined as a JSON schema. At the highest level, notebook is a dictionary with a few keys:

- metadata
- nbformat
- nbformat_minor
- cells

On the following figure 2.8 JSON schema is demonstrated. The most interesting entities are *cells*, which needs to be described in more details.

```
{
"metadata" : {
   "kernel_info": {
      # if kernel_info is defined, its name field is required.
      "name" : "the name of the kernel"
   },
   "language_info": {
      # if language_info is defined, its name field is required.
      "name" : "the programming language of the kernel",
      "version": "the version of the language",
      "codemirror_mode": "The name of the codemirror mode to use"
   }
},
"nbformat": 4,
"nbformat_minor": 0,
"cells" : [
   # list of cell dictionaries
],
}
```

Figure 2.8: Example of top level schema

There are few basic cell types for encapsulating code and text. All cells have the following structure:

```
{
   "cell_type" : "type",
   "metadata" : {},
   "source" : "single string or [list, of, strings]",
}
```

Figure 2.9: Example of cell schema

First cell type is **markdown**, those cells are used for body of notebook and contain markdown formatted text. Markdown – is a lightweight markup language, which allows to structure a text. The result of it is similar to HTML, for example it is possible to specify headings, paragraphs, lists, links, images, etc. Features of markdown allow to nicely format the body text of Jupyter notebook.

The basic structure of markdown cell:

```
{
   "cell_type" : "markdown",
   "metadata" : {},
   "source" : "[multi-line *markdown*]",
}
```

Figure 2.10: Example of markdown cell schema

Second cell type is **code**. Code cells are the primary content of Jupyter notebooks.[14] They are used to contain a source code and outputs associated with executing that code. Example of a code cell structure is demonstrated on the following figure:

According to notebook format documentation [14], **metadata** field which is present in both cell types, can be used to store arbitrary JSON information. Some metadata keys are already defined at the cell level and it is prohibited to use their names:

```
{
   "cell_type" : "code",
   "execution_count": 1, # integer or null
   "metadata" : {
      "collapsed" : True, # whether the output of the cell is
          collapsed
      "scrolled": False, # any of true, false or "auto"
   },
   "source" : "[some multi-line code]",
   "outputs": [{
      # list of output dictionaries
      "output_type": "stream",
      ...
   }],
}
```

Figure 2.11: Example of code cell schema

Key	Value type	Description
collapsed	bool	Whether the cell's output container should be collapsed
scrolled	bool or "auto"	Whether the cell's output is scrolled, unscrolled, or autoscrolled
deletable	bool	If False, prevent deletion of the cell
editable	bool	If False, prevent editing of the cell (by definition, this also prevents deleting the cell)
format	"mime/type"	The mime-type of a Raw Cell
name	string	A name for the cell. Should be unique across the notebook. Uniqueness must be verified outside of the json schema.
tags	list of string	A list of string tags on the cell. Commas are not allowed in a tag

Figure 2.12: Reserved cell metadata keys

As specified in requirements, feedback will be provided in a scope of a cell, meaning that any cell could have a possible conversation thread between student and a teacher. Please note, that it's not planned to use schema to provide *grading* functionality, it will be realize using open source plugin which fulfills that requirement and will be discussed in section 2.6. Taking this information and the list of requirements into account, using metadata field for storing of this conversation looks like a possible solution. In order to proceed with it, several things needs to be defined:

- Unique namespace of a key, which will prevent collision with other custom modules of Jupyter Notebook.
- Key for storing of data of the feedback module
- Feedback module JSON schema

At first, JSON schema should be defined. Decision on how should it look like must be derived from analysis of particular use cases for this module:

Write a message for specific cell: Both student and teacher can write a message into the cell of Jupyter notebook.

1. user selects the cell (focuses)
2. user types the text of a message
3. user clicks submit button to add message to the thread

Delete a message from specific cell: Both student and teacher can delete a message that is owned by them from the thread. In order to keep the replies of deleted message, the text of it will change to "deleted" instead of removing of whole record.

1. user clicks "delete" icon on a message
2. user confirms that she wants to delete a message

Main element of the schema is a **message**. Each message is defined by content, author and a time of creation - these are the three basic attributes. All the messages will be stored as an array defined by **messages** key.

Final JSON schema will look like this:

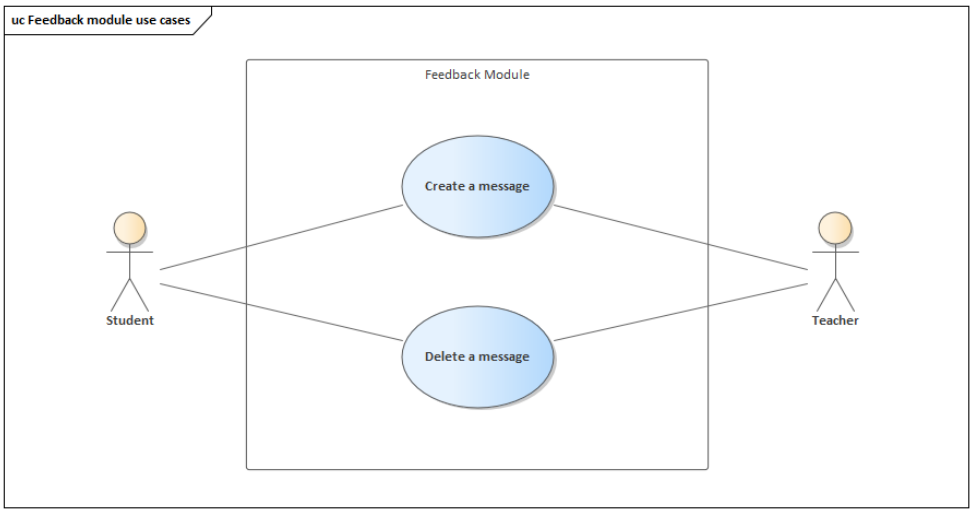

Figure 2.13: Feedback module use cases

```
{
   ...
   "metadata" : {
      ...
      "feedback_md" : {
         "messages" : [
            ...
            {
               "id" : "unique id of the message",
               "text" : "contents of the message",
               "username" : "username of the author".
               "datetime" : "date and time of creation"
            },
            ...
         ]
      },
      ...
   },
   ...
}
```

Figure 2.14: Schema of the Feedback module

Decision to store messages as metadata value fulfills the requirements 1.2, however it has some disadvantages. The main disadvantage is that the source

of data is located at client's side, meaning that theoretically user can alter or remove this data if she edited the JSON structure of notebook directly.

The proper solution for that problem would be to transfer the source of data to a remote server (backend) and store the messages in dynamic storage like database. In this case Feedback module would be loading messages from remote URI and user's actions on local JSON data won't be affecting records on remote server. This solution although a superior one, but greatly extends the scope of this thesis.

The functionality of this module will be implemented as an **extension**. Jupyter Notebook provides possibility to create custom front-end extensions, which allow to modify the behavior of the various pages like the dashboard, the notebook, or the text editor.[15]

A front-end extension is a JavaScript file that defines a module which exposes at least a function called **load_ipython_extension**, which takes no arguments.[15] The minimal code needed for a working extension:

```
// file my_extension/main.js

define(function(){

   function load_ipython_extension(){
      console.info('this is my first extension');
   }

   return {
      load_ipython_extension: load_ipython_extension
   };
});
```

Figure 2.15: Minimal code for Jupyter Notebook extension

After extension is implemented, it can be installed and enabled via following commands:

```
jupyter nbextension install path/to/my_extension/
jupyter nbextension enable my_extension/main
```

Figure 2.16: Installation of an extension

2.5 Infrastructure

As was described earlier 2.1.4, the best solution for distribution would be the centralized one. Luckily, there exists a project which exactly fulfills the requirements - **JupyterHub**.

It is a software that solves the problem with centralization of Jupyter notebook instance management. It spawns, manages, and proxies multiple instances of the single-user Jupyter notebook server. It can be used to serve notebooks to a class of students, a corporate data science group, or a scientific research group. This means that it can deliver custom instances of notebooks (customized with different plugins) to teachers and students.

JupyterHub is a set of processes that together provide a single user Jupyter Notebook server for each person in a group. It consists of a three major subsystems:

- Multi-user Hub (tornado process)
- Configurable http proxy (node-http-proxy)
- Multiple single-user Jupyter notebook servers (Python/IPython/tornado)

JupyterHub performs the following functions:

1. The Hub launches a proxy
2. The proxy forwards all requests to the Hub by default
3. The Hub handles user login and spawns single-user servers on demand
4. The Hub configures the proxy to forward URL prefixes to the single-user notebook servers

2.5.1 How the Subsystems Interact

Users access JupyterHub through a web browser, by going to the IP address or the domain name of the server.

The proxy is the only process that listens on a public interface. The Hub sits behind the proxy at /hub. Single-user servers sit behind the proxy at /user/[username].

Different authenticators control access to JupyterHub. The default one (PAM) uses the user accounts on the server where JupyterHub is running. In order to use it, it is needed to create a user account on the system for each person. Using other authenticators, it is possible to allow users to sign in with e.g. a GitHub account, or with any single-sign-on system of the organization.[16]

Existing authenticator can be used to connect the instance with University OAuth server.

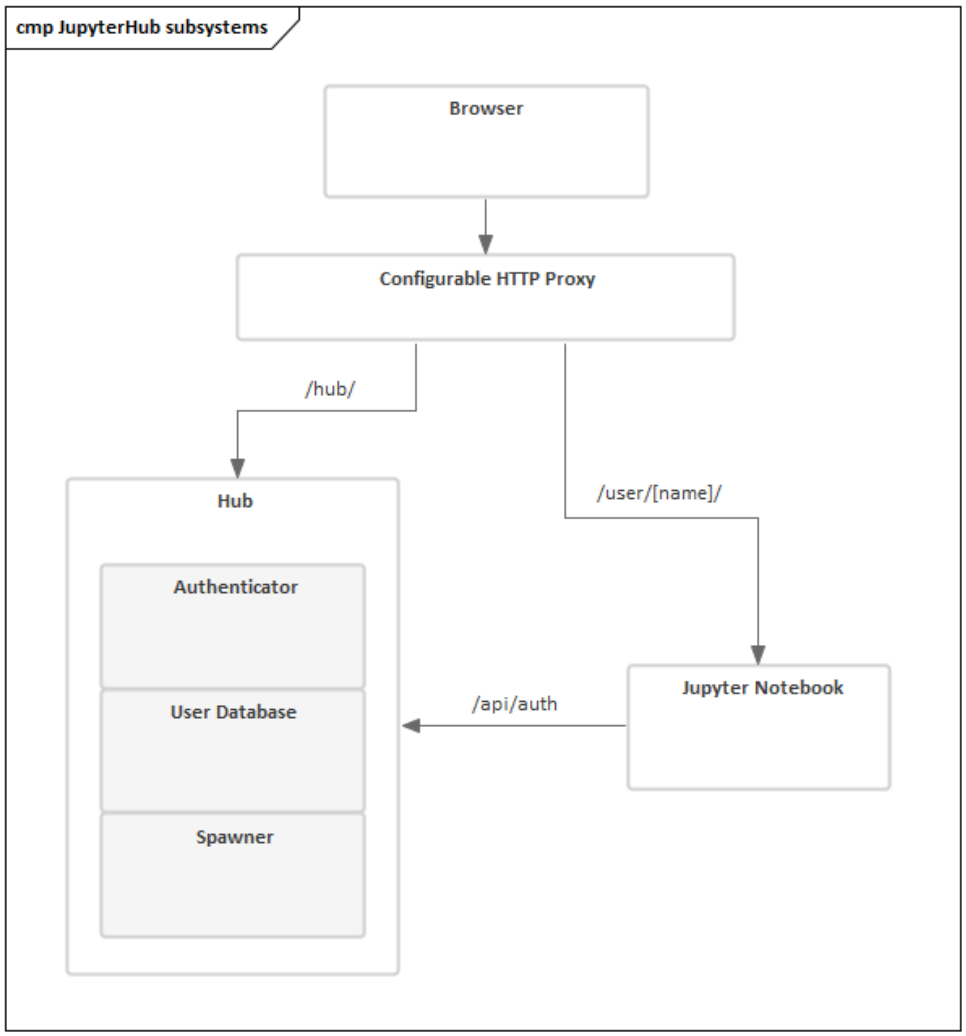

Figure 2.17: JupyterHub subsystems

Next, spawners control how JupyterHub starts the individual notebook server for each user. The default spawner will start a notebook server on the same machine running under their system username. The other main option is to start each server in a separate container, often using Docker 2.1.3.

2.5.2 User login

When a user accesses JupyterHub, the following events take place:

1. Login data is handed to the Authenticator instance for validation

2. The Authenticator returns the username if the login information is valid
3. A single-user notebook server instance is spawned for the logged-in user
4. When the single-user notebook server starts, the proxy is notified to forward requests to /user/[username]/* to the single-user notebook server.
5. A cookie is set on /hub/, containing an encrypted token.
6. The browser is redirected to /user/[username], and the request is handled by the single-user notebook server.

The single-user server identifies the user with the Hub via OAuth:

1. on request, the single-user server checks a cookie
2. if no cookie is set, redirect to the Hub for verification via OAuth
3. after verification at the Hub, the browser is redirected back to the single-user server
4. the token is verified and stored in a cookie
5. if no user is identified, the browser is redirected back to /hub/login

Support of OAuth allows to integrate JupyterHub with the University authentication server, which means that users will be able to login using University credentials as specified in requirements 1.2.

2.5.3 Kubernetes

There are exist several deployment options for JupyterHub, it's possible to install it via different package managers like **pip** and **conda** directly to running OS. Environment also possible to deploy via docker or build from the ground up by setting up every module of JupyterHub manually.[17]

In order to have the most robust and flexible instance of JupyterHub, it's preferred to deploy it on a Kubernetes cluster.

Kubernetes is an open-source container-orchestration system for automating application deployment, scaling and management. It defines a set of "building blocks", which provide mechanisms for deployment, maintaining and scaling of applications based on CPU, memory or custom metrics.[18]

The key objects are:

- **Pods**

  Pod is a higher level of abstraction grouping containerized components. A pod consists of one or more containers that are guaranteed to be co-located on the host machine and can share resources.[19] Each pod is assigned a unique Pod IP address within the cluster, which allows application to use ports without the risk of conflict.[20] When application

- **Services**

  Service is a set of pods function together. Services are discoverable via Kubernetes DNS or via environmental variables. Each service is assigned a stable IP address and load balancer decides in round-robin manner which pod inside a service is being used. By default all service are exposed only inside cluster.

- **Volumes**

  By default, storage inside Kubernetes is not "persistent" – it will be wiped out after restart of the pod. In order to store data persistently, volumes should be used. Each volume acts as a storage which can be shared between containers inside a pod. Each volume is mounted to specific mount points which is defined by configuration.

- **Namespaces**

  Kubernetes provides possibility to partition resources into non-overlapping sets which are called namespaces.[21] They are very helpful in environments with many users spread across multiple teams. This way these people can work independently on different namespaces without affecting each other.

Architecture of Kubernetes follows the master/slave architecture, where one device or process controls one of more other devices or processes and acts as a communication hub between them. The components of Kubernetes can be divided into those that manage an individual node and those that are part of the control plane.[22]

The Kubernetes master is the main unit which controls and manages communication across the system. The Kubernetes control plane consists of various components, that can run both on single node or multiple masters.[22] These are some components of the control plane:

- **etcd**

  etcd is a persistent, lightweight, distributed key-value store which stores the configuration data of the cluster and represents the overall state of it at any given time. For example, if deployer specifies that three instances of a given pod need to be running, this fact is stored in etcd. If happens, that only two instances are running, then Kubernetes will schedule pod creation of the missing one.[22]

- **API server**

  The API server is the key component which provides API using JSON over HTTP for internal and external interface to Kubernetes.[19] The API server processes requests and updates the state of objects in etcd and by doing so, allows to configure workloads and containers across the nodes.[19]

- **Scheduler**

  The scheduler is a component which selects the node that will be used to run a new pod, based on resource availability. The scheduler monitors resource use on each node to ensure that workload is not scheduled in excess.[19]

- **Controller manager**

  A controller is a reconciliation loop which is driving the cluster state towards the desired one, communicating with API server to create, update, and delete the resources.[23] The controller manager is a process that manages a set of Kubernetes controllers. One type of controllers is a **Replication Controller**, which handles replication and scaling by running a specified number of copies of a pod across the cluster. If some pod fails, it handles creation of replacement pods.[23]

Kubernetes node is a machine where containers (workloads) are deployed. Every node in a cluster runs a container runtime such as Docker and other components listed below:

- **Kubelet**

  Kubelet is responsible for the state of each node, ensuring that every container in the node is "healthy". It takes care of starting, stopping and maintaining application containers.[19]

- **Kube-proxy**

  Kube-proxy is an implementation of a network proxy and a load balancer. It's responsible for routing traffic to the appropriate container based on IP and port of incoming request.[19]

2.6 Managing the homeworks

Technical requirements specify several main features which are required for course teachers to create, update, distribute, collect, and grade the homeworks. At the same time, students needs to be able to receive, implement, submit and view the results of those homeworks. Let's structure it into well defined use cases:

2.6.1 Teacher's use cases

Teacher's use cases should cover all the core functionality regarding creation, distribution, collection, and grading of the homeworks.

- **Select a course**

Each teacher should be able to select a course she is teaching. Courses should be fetched from University system.

Scenario:

1. Teacher is presented with a list of her courses.
2. Teacher selects a course. All other actions will be made in the scope of selected course.

Postconditions:

- Teacher sees all homework related data for selected course.

- **Create homework**

  Each teacher should be able to create homework for a course. Homeworks are created as Jupyter notebook files.

  Scenario:

  1. Teacher prepares Jupyter notebook file with homework assignment.
  2. Teacher uploads the file with assignment into the system as a homework.
  3. (Alternative flow) Teacher creates empty Jupyter notebook file inside the system and writes the assignment.

  Postconditions:

  - Created homework is stored into the system as a draft.

- **Update homework**

  Each teacher should be able to modify already created homework assignment.

  Preconditions:

  - Teacher has selected already created homework in the system.

  Scenario:

  1. Teacher modifies contents of it directly inside the system.
  2. Teacher saves the changes.

  Postconditions:

  - Changes which were made by teacher are stored into the system

- **Delete homework**

  Each teacher should be able to delete already created homework.

  Preconditions:

- Teacher has selected already created homework in the system.

Scenario:

1. Teacher executes delete action via user interface.

Postconditions:

- Homework record is deleted from selected course.

- **Publish homework**

  Each teacher should be able to publish a homework draft, so that students have access to it. Once it's published, students should be able to fetch it and start working.

  Preconditions:

  - Teacher has selected already created draft of a homework assignment.

  Scenario:

  1. Teacher executes publish action via user interface.

  Postconditions:

  - Published homework is available for students of this course.

- **Fetch student's solution**

  Each teacher should be able to fetch and view student's solution for selected homework assignment.

  Preconditions:

  - Teacher has selected already created and published homework assignment.

  Scenario:

  1. Teacher executes fetch action via user interface.
  2. Teacher sees homework solutions which were submitted by students.
  3. Teacher can view the contents of a solution.

  Postconditions:

  - Teacher sees the contents of a chosen homework solution.

- **Grade student's solution**

  Each teacher should be able to grade a student's solution for a homework assignment.

  Preconditions:

  – Teacher fetched and opened a solution to a homework assignment

  Scenario:

  1. Teacher input grade for a homework solution.
  2. Teacher submits the grading.

  Postconditions:

  – Grading for a student's solution is stored in the system and available for student.

- **Export grades**

  Each teacher should be able to export a list of grades from the system for selected course.

  Preconditions:

  – Teacher did grading of homework solutions for a specific course.

  Scenario:

  1. Teacher executes export action via system user interface.

  Postconditions:

  – Export of grades in text format is available for teacher to download.

2.6.2 Student's use cases

Student's use cases cover processes of fetching, implementing, and submitting of the homeworks.

- **Select a course**

  Each student should be able to select a course she is enrolled in.

  Scenario:

  1. Student sees a list of courses he is enrolled in.
  2. Student selects a course from the list.

  Postconditions:

  - Course is selected. All other actions will be done in the scope of this course.

- **Fetch homework assignments**

  Each student should be able to fetch a homework assignment that was already published by the teacher of the course.

  Preconditions:

  - Student has selected a course.

  Scenario:

  1. Student executes fetch action via user interface.

  Postconditions:

  - Student can see the contents of the fetched homework assignment.

- **Implement a homework solution**

  Each student can implement a solution for a homework assignment.

  Preconditions:

  - Student has fetched and opened homework assignment.

  Scenario:

  1. Student implements a solution for a fetched homework.
  2. Student saves the solution.

  Postconditions:

  - Solution for a homework is saved into the system and is ready to be submitted.

- **Submit the solution**

  Each student should be able to submit a solution for a homework assignment.

  Preconditions:

  - Student has implemented and saved a solution for a homework assignment.

  Scenario:

  1. Student executes submit action via user interface.

  Postconditions:

  - Student's solution to a homework is available for teacher to grade.

- **Fetch a grade for a homework**

  Each student should be able to fetch a grade for a homework solution she submitted.

  Preconditions:

  - Student has implemented and submitted homework solution.
  - Teacher has graded the solution.

  Scenario:

  1. Student executes grading fetch action via user interface.

  Postconditions:

  - Student can see the grading of her solution.

2.6.3 nbgrader

In order to provide the required functionality, it is decided to use an open source solution **Nbgrader** - system for assigning and grading of Jupyter notebooks. Core functionality of it covers the requirements regarding management of the homework.

It organizes the courses and files using specific structure on the filesystem:

```
{course_dir}/{step}/{stud_id}/{homework_id}/{notebook_id}.ipynb
```

Example of the course data for a teacher account on the filesystem:

Example of the course data for a student account on the filesystem:

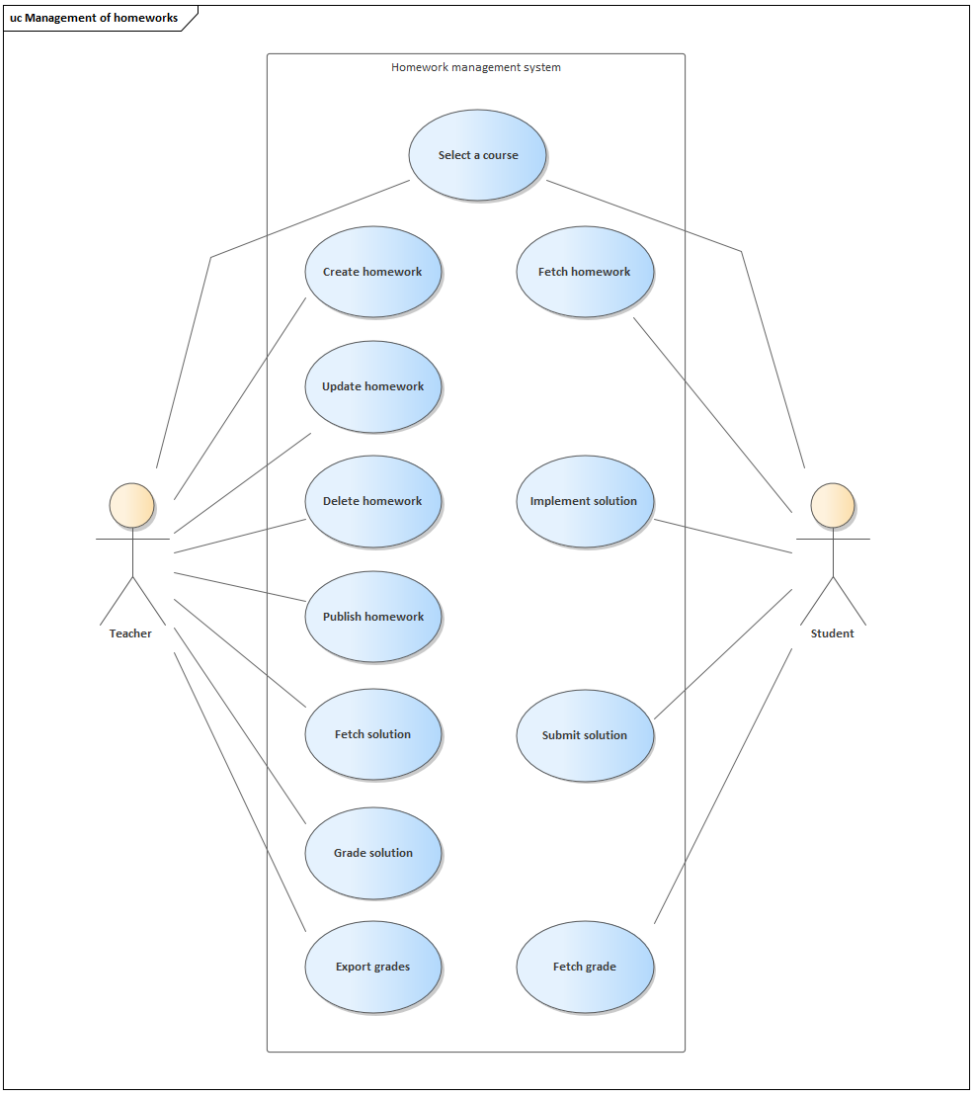

Figure 2.18: Management of homework use cases

Nbgrader contains several extensions to provide all it's functionality to users. Teachers are utilizing **Formgrader** and **Create Assignment** extensions in order to manage and grade the assignments, while students need to use just **Assignment List** extension to have access to the assignments prepared by teachers.

Although nbgrader is a tool for Jupyter Notebook (client-side application), it also integrates with JupyterHub (server-side application). It's doing so via another set of extensions, server-side versions of the ones listed above.

```
course1 ..... the course directory
├── gradebook.db ..... database of homeworks and students
├── nbgrader_config.py ..... configuration file of a course
├── source ..... homework source
│   └── hw1 ..... homework directory
│       ├── problem1.ipynb ..... Jupyter Notebook file with homework
│       └── problem2.ipynb
├── release ..... The directory with published homeworks
│   └── hw1
│       ├── problem1.ipynb
│       └── problem2.ipynb
├── submitted ..... The directory with submitted homework solutions
│   ├── student1 ..... Student's directory with solutions
│   │   └── hw1
│   │       ├── problem1.ipynb ..... Jupyter Notebook file with solution
│   │       └── problem2.ipynb
│   ├── student2
│   │   └── hw1
│   │       ├── problem1.ipynb
│   │       └── problem2.ipynb
│   └── student2
├── autograded ..... Directory with autograded solutions
└── feedback ..... Directory with teacher's grading results
```

Figure 2.19: Directory structure of a teacher

```
course1 ..... the course directory
├── hw1 ..... homework directory
│   ├── problem1.ipynb ..... Jupyter Notebook file with homework
│   ├── problem2.ipynb
│   └── feedback ..... Directory with teacher's grading results
│       └── 2019-05-31-11-48-34 ..... Timestamp of feedback
│           ├── problem1.html ..... Teacher's feedback with grade
│           └── problem2.html
└── hw2
```

Figure 2.20: Directory structure of a student

When it's integrated with JupyterHub, one of the most important components of it becomes so-called **Exchange**. Exchange acts as a "middleman" between teacher and student. When teacher releases the assignment, it is being stored in exchange directory. Later, when student will trigger fetch operation, it will pull it from the same exchange location.

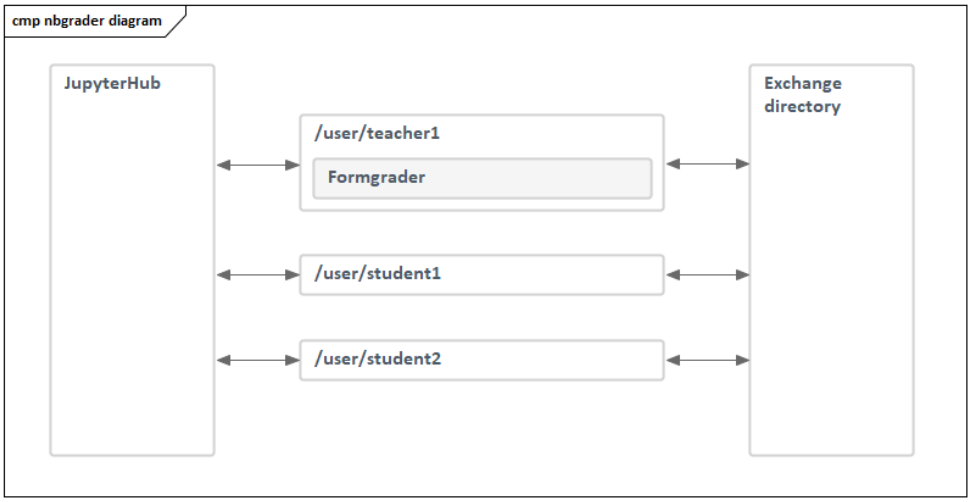

Figure 2.21: nbgrader diagram

In order to get the course results from the system, there exists the **Export plugin** which allows teachers to export course grades into CSV format, so that it can be later imported into school information system (KOS). This plugin can also be extended if other format is needed.

Chapter 3

Implementation

In this chapter implementation part of the requirements will be described. Following observations and conclusions which were made in previous chapter, full scope of work will include:

- Implementation of the authentication process against University system
- Implementation of the homework management system
- Setting up infrastructure and preparation of deployment scripts
- Installation and configuration of JupyterHub

In order to not ”reinvent the wheel”, it is advised to reuse existing approaches and solutions.

At first, authentication part of the solution would be described.

3.1 Authentication

Authentication process in University follows OAuth2 protocol, which was described in subsection 2.2.1. The target is to implement this process into JupyterHub, which will be used as a main component in whole system.

JupyterHub supports various OAuth2 providers out of the box, this list includes:

- Auth0
- Bitbucket
- CILogon
- GitHub
- GitLab
- Globus
- Google
- MediaWiki
- Okpy
- OpenShift

It is not required to use any of these services, so more general solution should be found. Luckily, there exists a "generic" implementation of the Authenticator.[24]

In order to test this Authenticator, it is needed to register application in University Apps Manager [25], where **client ID** and **client secret** will be obtained which will be used as credentials for client application.

Lets summarize the process which needs to happen in order to fully authenticate the user against University system:

It is clear from figure 3.1, Authenticator needs to handle the process from receiving the authorization code till retrieving user's details. Unfortunately, generic Authenticator bundled with JupyterHub uses different naming of parameters in requests and because of that University server is not able to process them.

In order to make it work, it is necessary to implement custom Authenticator and bundle it with JupyterHub Docker image.

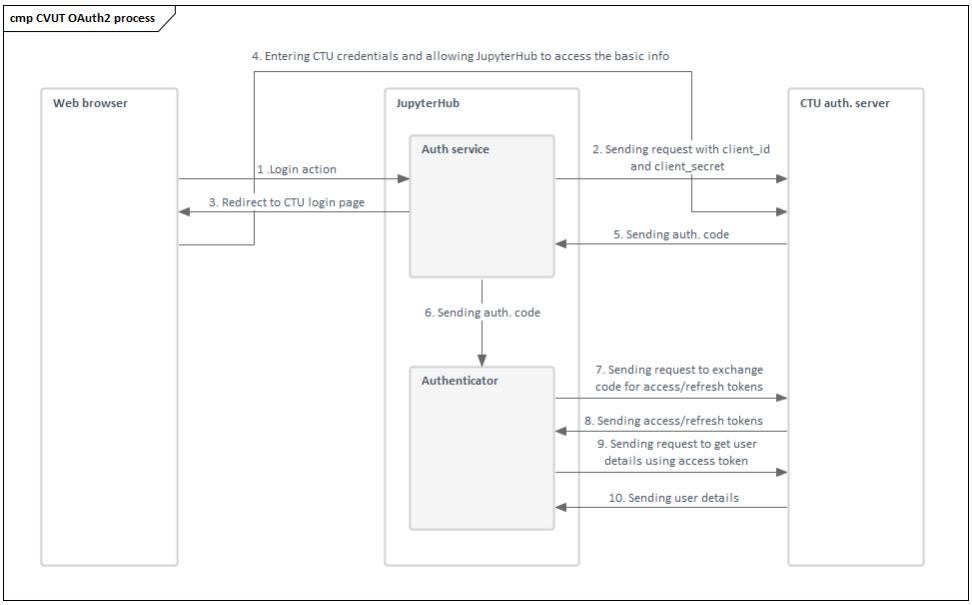

Figure 3.1: Authentication process

3.1.1 Making custom Authenticator

Making of custom Authenticator for JupyterHub starts from extending of existing Authenticator class and overriding **authenticate** method:

```
from IPython.utils.traitlets import Dict
from jupyterhub.auth import Authenticator

class DictionaryAuthenticator(Authenticator):

    passwords = Dict(config=True,
        help="""dict of username:password for authentication"""
    )

    async def authenticate(self, handler, data):
        if self.passwords.get(data['username']) == data['password']:
            return data['username']
```

Here depicted the example of a custom Authenticator which uses simple dictionary to do authentication.

For the Authenticator it is possible to reuse existing class **OAuthenticator** which is bundled with JupyterHub, it already has some required logic implemented.

```
from .oauth2 import OAuthenticator

...

class ZuulOAASOAuthenticator(OAuthenticator):

    ...

    @gen.coroutine
    def authenticate(self, handler, data=None):

        ...
```

First step is getting the **code** value from URL. It will be used to get access/refresh token pair from University system as per figure 3.1.

```
def authenticate(self, handler, data=None):
    code = handler.get_argument("code") #getting code value

    http_client = httpclient.AsyncHTTPClient()

    params = dict(
        redirect_uri=self.get_callback_url(handler),
        code=code,
        grant_type='authorization_code'
    )

    ...

    req = httpclient.HTTPRequest(url,
                         method="POST",
                         headers=headers,
                         body=urllib.parse.urlencode(params)
                         )
    #getting response with access/refresh tokens
    resp = yield http_client.fetch(req)
```

Next step is to fetch user details (username of the user is the most important). Once it is fetched, whole process is finished.

```
...

def authenticate(self, handler, data=None):

    ...
```

```
resp = yield http_client.fetch(req)

resp_json = json.loads(resp.body.decode('utf8', 'replace'))

access_token = resp_json['access_token']
refresh_token = resp_json.get('refresh_token', None)
token_type = resp_json['token_type']

...

req = httpclient.HTTPRequest(url,
                             method=self.userdata_method,
                             headers=headers
                             )
resp = yield http_client.fetch(req)
resp_json = json.loads(resp.body.decode('utf8', 'replace'))
name = resp_json.get(self.username_key) #getting username

# return auth data to be used by the rest of the system
return {
   'name': name,
   'auth_state': {
      'access_token': access_token,
      'refresh_token': refresh_token,
      'oauth_user': resp_json,
      'scope': scope,
   }
}
```

The Authenticator passed testing and successfully completed authentication process against University system. Next step is to make it available inside JupyterHub build.

3.1.2 Making custom Docker image

There are multiple methods of JupyterHub installation, each of them would require different way to embed custom Authenticator. As was concluded in previous chapter, the best way to setup JupyterHub for current use case will be to do so inside Kubernetes cluster.

As was already written in section 2.5.3, Kubernetes nodes operate Docker containers, thus all applications which are deployed should be containerized.

The plan is to make ZuulOAASOAuthenticator available inside JupyterHub during it's runtime on a cluster. To do so, it is necessary to use official JupyterHub Docker image [26] and customize it according to the needs.

What is required is to modify **Dockerfile** of the image and to copy custom Authenticator inside of it, then run installation process:

```
...

# copy folder with custom Authenticator inside
COPY Zuul-OAAS-Authenticator /tmp/Zuul-OAAS-OAuthenticator

# install it to be available in the system
RUN pip3 install -e /tmp/Zuul-OAAS-OAuthenticator

...
```

After Dockerfile was modified, it was built locally and pushed to DockerHub [27] repository. DockerHub is used for storing of Docker images and later will be used to pull modified JupyterHub image by Kubernetes cluster.

3.2 Homework management

Homework management functionality will be realized by installation and configuration of **nbgrader** - open source solution which is developed and maintained under Project Jupyter and currently is one of the most popular solutions for schools which use Jupyter Notebook.[28]

Whole process of implementation will consists of several steps:

1. Implementation of the Feedback module
2. Preparation of custom Docker image for Jupyter Notebook which will include the module
3. Installation of nbgrader

3.2.1 Feedback module

Feedback module will be implemented as a Jupyter Notebook extension which will provide a simple user interface for writing of messages into the notebook cell as described in section 2.4.

Since Jupyter Notebook is a web application, it is necessart to use web technologies (JavaScript, HTML, CSS) to create custom extension. The basic structure consists of just one function which needs to be exposed:

```
// file my_extension/main.js

define(function(){

   function load_ipython_extension(){
      console.info('this is my first extension');
   }

   return {
      load_ipython_extension: load_ipython_extension
   };
});
```

In this function callback will be registered which will add the new option into the cell toolbar.

```
var CellToolbar = celltoolbar.CellToolbar;

CellToolbar.register_callback(
   'feedback_md.show_count',
   show_message_count
);
```

```
CellToolbar.register_preset(
   'Feedback module',
   ['feedback_md.show_count']
);
```

After this option is used, new button will appear above the cells. This button will toggle chat window on mouse press. Everything is implemented using JQuery library, it's one of the most popular front-end JavaScript libraries which provides functions that cover most of the front-end manipulations. The button code is a simple JQuery snippet:

```
var button = $('<button/>')
   .addClass("btn btn-default btn-xs")
   .text("Display reviews (" + message_count + " messages)")
   .click( function () {
      if ($(div).find('#feedback-form').length > 0) {
         hide_chat(div, cell);
      } else {
         show_chat(div, cell);
      }

      return false;
   });
```

In order to modify metadata of the cell, it is needed to pass it inside the function which will manipulate metadata directly:

```
var add_message = function(cell, message, username) {
     cell.metadata.feedback_md.messages.push({
      'id': uuidv4(),
         'text': message,
         'username': username,
         'datetime': Date().toLocaleString()
     });
 }
```

Once extension is implemented, it is necessary to create a custom Docker image of Jupyter Notebook which will have the extension installed and enabled. The process is similar to the one which was described in section 3.1.2. Dockerfile will be started from one of Jupyter Notebook images, the source code of extension will be copied inside and installation is executed:

```
...

COPY ./feedback_md /tmp/feedback_md
```

```
RUN jupyter nbextension install /tmp/feedback_md/ --sys-prefix && \
jupyter nbextension enable feedback_md/feedback_md --sys-prefix

...
```

Final built image will be pushed to DockerHub as well.

3.2.2 nbgrader installation

Installation of nbgrader is a very simple process, modification of it's functionality is not needed, so it's just enough to add setup process to Dockerfile of Jupyter Notebook:

```
RUN pip install nbgrader

RUN jupyter nbextension install --sys-prefix --py nbgrader
    --overwrite && \
jupyter nbextension enable --sys-prefix --py nbgrader && \
jupyter serverextension enable --sys-prefix --py nbgrader
```

Configuration of nbgrader itself is done via YAML files and will be described in section 3.4.2.

During the testing of nbgrader with the feedback module it was discovered, that due to specifics of nbgrader it's not possible for a teacher and a student to access the same notebook file in order to exchange messages with each other. This is due to the fact, that student's volume is not available when she is not using the system (volume is unmounted). The only time teacher gets access is when student submits it for grading. However even then, nbgraded doesn't have the possibility to return the notebook back to student with added messages into cells.

All of this makes us introduce a work-around for this problem. Each teacher needs to be added to list of admin users:

```
c.Authenticator.admin_users = {'mal', 'zoe'}
```

Or assigned to admin group:

```
c.PAMAuthenticator.admin_groups = {'wheel'}
```

Then **admin_access** option should be enabled which will allow teacher to access student's volume even when student is logged out.

```
JupyterHub.admin_access = True
```

Those configuration values should be placed into JupyterHub configuration file **jupyterhub_config.py**

3.3 Kubernetes

Deployment will be realized on Kubernetes cluster, by doing so it will solve the non-functional requirements, as well as providing fully working system to students without any installation on their computer and utilizing scalability of Kubernetes cluster, it will guarantee that application will handle any amount of load.

Main tool which is used to manipulate the cluster is **kubectl**. Kubectl is the command-line tool which allows to run commands against Kubernetes clusters. Kubectl can be used to deploy applications, inspect and manage cluster resources, and view logs.[29]

Installation process of this tool is very simple and includes two main 2 steps:

1. Download latest release of the tool
2. Make downloaded file executable

All configuration for Kubernetes cluster has to be specified using YAML files. It is necessary to create the configuration file which will enable **dynamic provisioning** of disks. That will allow to automatically assign a disk per user when they login in to JupyterHub.[30]

```
# file storageclass.yml

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
   annotations:
      storageclass.beta.kubernetes.io/is-default-class: "true"
   name: gp2
provisioner: kubernetes.io/aws-ebs
parameters:
   type: gp2
```

This config can be applied on the cluster by entering command:

```
kubectl apply -f storageclass.yml
```

Next step is to install **Helm** which is used for deployment of applications to cluster.

3.3.1 Helm

Helm, the package manager for Kubernetes, is a useful tool for installing, upgrading and managing applications on a Kubernetes cluster. Helm packages

are called charts. Charts are abstractions describing how to install packages onto a Kubernetes cluster. When a chart is deployed, it works as a templating engine to populate multiple YAML files for package dependencies with the required variables, and then runs kubectl apply to apply the configuration to the resource and install the package.[31]

Helm has two parts: a client (Helm) and a server (tiller). Tiller runs inside of Kubernetes cluster as a pod in the kube-system namespace. Tiller manages both, the releases (installations) and revisions (versions) of charts deployed on the cluster. When helm commands are executed, local Helm client sends instructions to tiller in the cluster that in turn makes the requested changes.[31]

Installation of the Helm consists of several steps, mainly:

1. Install Helm locally
2. Create account for Tiller in the cluster
3. Initialize both Helm and Tiller

Once both kubectl and Helm installed, JupyterHub application is ready to be deployed.

3.4 JupyterHub

In this section will be described the implementation of the custom JupyterHub installation. In order to fulfill technical requirements, it is needed to create a client, which is able to communicate with KOSAPI for fetching of information about students, teachers and courses. Another part of the build was a custom Kubernetes spawner which enables JupyterHub to spawn single-user notebook servers on the cluster.[32]

3.4.1 KOSAPI

Implementation of KOSAPI client will consists of writing a class KOSApi-Client which has 4 methods:

- **is_teacher(username)** – method which checks if given username is teacher or a student
- **get_courses_for_teacher(username)** – method which fetches course list for given teacher
- **get_courses_for_student(username)** – method which fetches course list for given student
- **get_students_for_course(coursename)** – method which fetches student list for given course. Used to populate database of students.

Part of get_courses_for_student(username) method:

```
class KOSApiClient:

    def __init__(self, access_token):
        self.access_token = access_token
      self.token_type = 'Bearer'

   ...

   async def get_courses_for_student(self, username):
      kosapi_base_url = 'https://kosapi.fit.cvut.cz/api/3'
      url = '{}/students/{}/enrolledCourses'.format(
         kosapi_base_url,
         username
      )
      http_client = httpclient.AsyncHTTPClient()

      headers = {
         "User-Agent": "JupyterHub",
         "Authorization": "{} {}".format(
            self.token_type,
            self.access_token
         )
      }
      try:
         req = httpclient.HTTPRequest(url,
                           method='GET',
                           headers=headers,
                           )
         resp = await http_client.fetch(req)

      ...
```

3.4.2 Spawner

Spawner is one of the most important parts of whole system, it spawns pods in Kubernetes cluster for each logged in user. For current implementation, KubeSpawner [33] is chosen. It is shipped with JupyterHub installation. Each server which was spawned by a user has it's own KubeSpawner instance, so it is possible to use that fact in order to customize the environment of newly created pod.

In order to "inject" the custom logic into the spawner, it is needed to implement custom spawner class which will extend KubeSpawner and override some methods.

Firstly, it is required to be able to mount different storage volumes for different users, so that each two different users never share one volume, thus noone has access to someone else's data. Secondly, it is necessary to generate dynamically different configuration files for nbgrader in order to prepare the environment for the chosen course. List of courses should be fetched from KOSAPI, thus KOSAPI client needs to be used by spawner. In order to authenticate agains University API, spawner instance needs to receive access and refresh tokens from custom Authenticator 3.1.1.

According to KubeSpawner documentation [33], it is possible to override following methods in order to implement required logic:

- KubeSpawner.options_form
- KubeSpawner.pre_spawn_hook

KubeSpawner.options_form – An HTML form for options a user can specify on launching their server. The value returned from this method could be a string (HTML) or callable, which will be called asynchronously.

This call will be utilized in order to fetch the list of courses, for student those are the courses she has this semester; for teacher, the ones she is teaching. Returned list of courses will be stored in **KubeSpawner.profile_list** – list of profiles which user can select on start of the pod. Each profile will correspond to one course:

```
async def options_form(self, spawner):
    auth_state = await self.user.get_auth_state()

    client = KOSApiClient(auth_state['access_token'])

    # check if logged in user is a teacher
    is_teacher = await client.is_teacher(
        auth_state['oauth_user']['user_name']
    )
    my_courses = []

    if is_teacher is True:
        # if a teacher, then get list of courses she is teaching
        my_courses = await client.get_courses_for_teacher(
            auth_state['oauth_user']['user_name']
        )
    else:
        # if a student, then get courses in current semester
        my_courses = await client.get_courses_for_student(
            auth_state['oauth_user']['user_name']
        )

    ...
```

```
    self.profile_list = profile_list

    return super().options_form
```

Another method, KubeSpawner.pre_spawn_hook – is an optional hook function that can be implemented to do some bootstrapping work before the spawner starts. For example, create a directory for a user or load initial content.[33]

Here the logic will be implemented which will receive the chosen course name, prepare configuration for the environment and nbgrader. For example, here it will specify which volumes need to be mounted and where in the system:

```
async def pre_spawn_hook(self, spawner):
    await self.load_user_options()

    ...

    self.volume_mounts = [
        {
            'mountPath': home_mount_path,
            'name': volume_name_template
        },
        {
            "mountPath": "/srv/nbgrader/exchange",
            "name": 'nbgrader-exchange',
            "subPath": "exchange/{}".format(self.course_id),
            "readOnly": False
        }
    ]
```

This spawner is set to be used by JupyterHub via configuration file jupyterhub_config.py:

```
c.JupyterHub.spawner_class = CustomKubeSpawner
```

There are always required two volumes for nbgrader to function properly, user home volume and exchange one 2.21.

Nbgrader configuration consists of creation of **nbgrader_config.py** files, which contain information about current course and directory with assignments:

```
for line in [
    '',
    'c = get_config()',
```

```
        'c.CourseDirectory.course_id = "{}"'.format(self.course_id),
        'c.Exchange.course_id = "{}"'.format(self.course_id),
        'c.Exchange.assignment_dir = "{}/assignments/{}"'.format(
            home_mount_path,
            self.course_id
        ),
        'c.AssignmentList.assignment_dir = "{}/assignments/{}"'.format(
            home_mount_path,
            self.course_id
        )
    ]:
        cmds.append(
            r"echo '{}' >> {}/.jupyter/nbgrader_config.py".format(
                line,
                home_mount_path
            )
        )
```

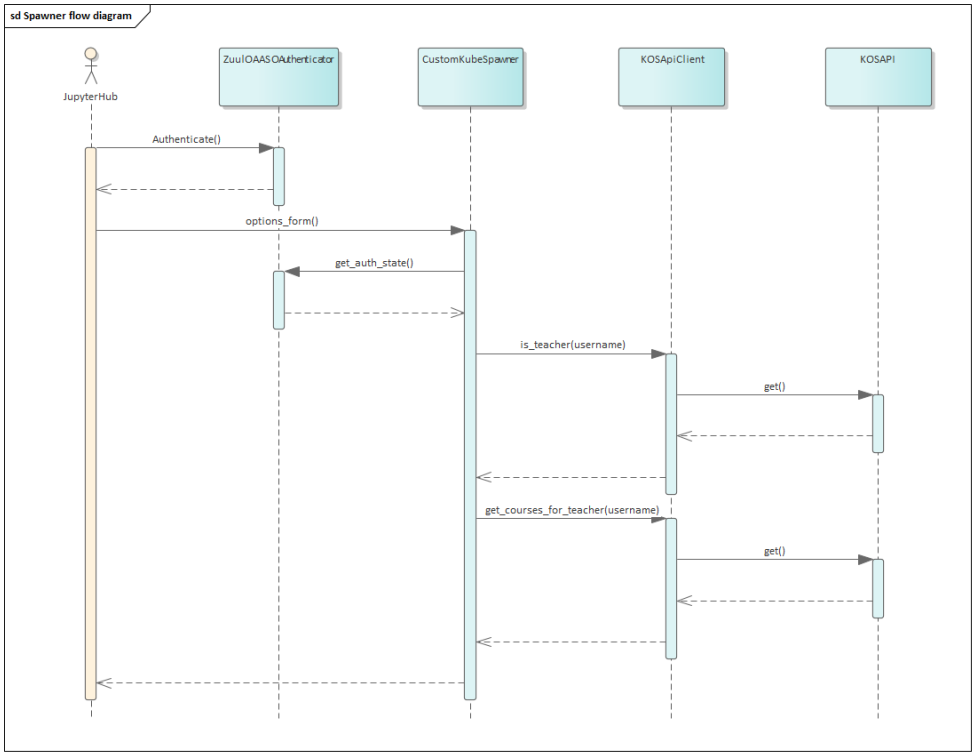

Figure 3.2: Custom spawner sequence diagram

3.4.3 Deployment

Deployment of JupyterHub to Kubernetes cluster will be done using official Helm chart.[34]

Firstly, it is needed to build the final version of JupyteHub Docker image, which will contain all customizations and jupyterhub_config.py file:

```
...

RUN mkdir -p /srv/jupyterhub/
RUN mkdir /etc/jupyter
RUN chmod 775 /etc/jupyter

COPY ./jupyterhub_config.py /srv/jupyterhub_config.py
COPY ./z2jh.py /srv/z2jh.py
COPY ./cull_idle_servers.py /etc/jupyterhub/cull_idle_servers.py
WORKDIR /srv/jupyterhub/

...
```

In order to use this chart it is necessary to prepare YAML configuration file which will specify settings of individual components that will be deployed. Here settings of Authenticator 3.1.1, custom JupyterHub Docker image, and some general JupyterHub parameters will be placed:

```
proxy:
   secretToken: "28d792014a3df3e6129272a5f3e1ff60c....."
   https:
      enabled: true
      type: "letsencrypt"
      letsencrypt:
      contactEmail: "vanyadmi@fit.cvut.cz"
      hosts:
      - "hostname"

# hub settings, here is specified custom JupyterHub Docker image
hub:
   image:
      name: "akasummer/jupyterhub-cvut-v3"
      tag: "latest"
# Environment auth related settings
extraEnv:
   OAUTH2_AUTHORIZE_URL: "https://auth.fit.cvut.cz/oauth/authorize"
   OAUTH2_TOKEN_URL: "https://auth.fit.cvut.cz/oauth/token"
   OAUTH_CALLBACK_URL: "https://hostname/hub/oauth_callback"
   OAUTH2_BASIC_AUTH: true
```

```
    OAUTH2_USERDATA_URL: "https://auth.fit.cvut.cz/oauth/check_token"
    OAUTH2_USERNAME_KEY: "user_name"
    OAUTH_CLIENT_ID: "e09aad91-f3d4-4db0-8b29-xxxxxxxxxx"
    OAUTH_CLIENT_SECRET: "SUSQU4jffxqaM3ZLbfSxxXXXxXxxx"
    OAUTH2_LOGIN_SERVICE: "CVUT"

auth:
    type: "custom"
    custom:
      className: zuul.ZuulOAASOAuthenticator.ZuulOAASOAuthenticator
      config:
        login_service: "CVUT"
        userdata_token_method: "url"
```

Once configuration file is created, custom JupyterHub and Jupyter Notebook images are pushed to DockerHub registry, a Helm command that will tell Tiller to start deployment process should be invoked:

```
helm upgrade --install jhub jupyterhub/jupyterhub \
  --namespace jhub \
  --version=0.8.2 \
  --values config.yaml
```

After several minutes final Homework Management System is running on Kubernetes cluster.

3.4.4 Testing

In order to validate the functionality of the final solution, testing methodologies should be applied to some degree. Since the majority of the used tools are the open source applications, it is enough to launch provided test suite. Once all automated tests passed, it can be concluded that those applications are working and nothing was broken during customizations. For the added functionality, user acceptance testing was used.

JupyterHub allows to specify administrator accounts which were used to impersonate teacher and students. While being logged in as a teacher whole process from creation of the first homework assignment till release of the feedback was manually tested. Student accounts were used to fetch, implement and submit the homework. Feedback module proved to be working, but with the limitation described in section 3.2.1.

KOSAPI connection was manually tested during the student/teacher scenario, list of courses and students was fetched from API successfully.

During testing, some user experience related issues were found. For-mgrader extension is not visually integrated into the overall system, it is launched in separate browser tab, thus making it harder to navigate back.

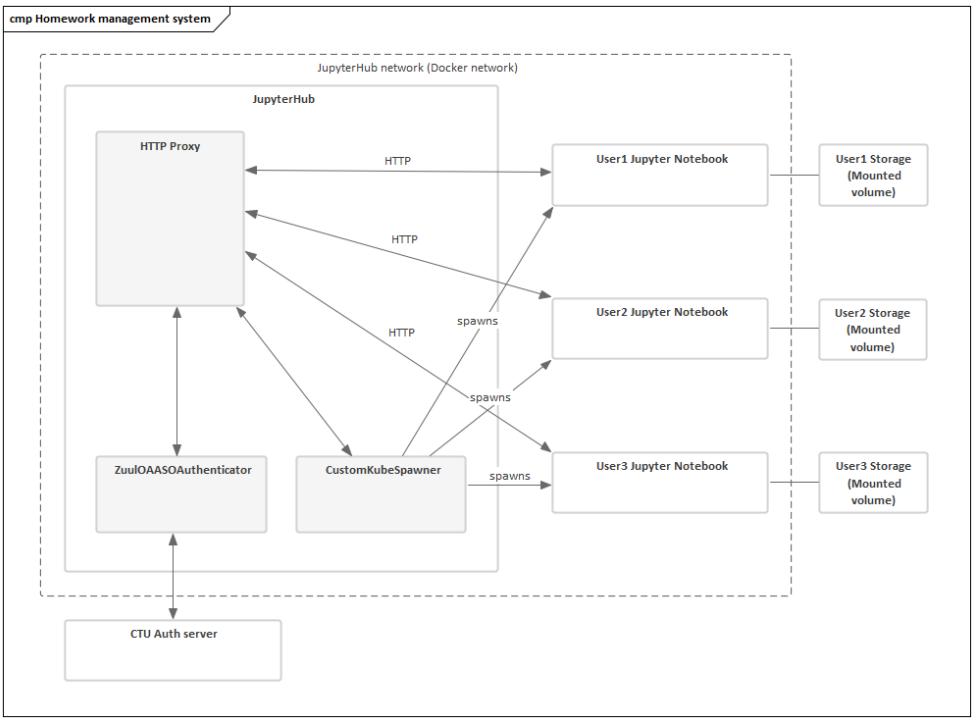

Figure 3.3: Jupyter Homework Management system

Chapter 4

Conclusion

The goal of this thesis was to analyse and implement a Homework Management system according to various functional and non-functional requirements. Results of this work indicate that the goal was reached and requirements are fulfilled. All the core ascpects of implementation, configuration, and deployment of the system were thoroughly described.

Unfortunately not all implemented use cases are done in the most user-friendly manner, the limitations were introduced by specifics of open source applications which were used.

Nbgrader application is heavily dependent on filesystem access and relies on passing of files between directories, this forced us to add extra steps needed for teacher to switch the selected course in the system. Also the same reasons made us to introduce work-around solution for the feedback module.

Result of this work is opening a lot of posibilities for future improvement. Along with new functionality which can be added, existing features can be refined in order to make them more intuitive to users.

It is possible to get new experience and knowledge about installation and configuration of Kubernetes clusters, helm charts and Python programming.

Current ecosystem around project Jupyter is developing very rapidly and all the parts are being improved along the way. It already enables schools and universities to introduce Jupyter products into the study process, and hopefully results of this thesis will help to integrate them into various courses at the university.

Bibliography

[1] Project Jupyter. Jupyter. (accessed: 30.12.2019). Available from: https://jupyter.org/

[2] Project Jupyter. About. (accessed: 30.12.2019). Available from: https://jupyter.org/about

[3] Project Jupyter team. Installing Jupyter Notebook. (accessed: 30.12.2019). Available from: https://jupyter.readthedocs.io/en/latest/install.html

[4] Anaconda, Inc. Anaconda Distribution. (accessed: 02.01.2020). Available from: https://www.anaconda.com/distribution/

[5] Popek, G. J.; Goldberg, R. P. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, volume 17, no. 7, July 1974: p. 412–421, ISSN 0001-0782, doi:10.1145/361011.361073. Available from: https://doi.org/10.1145/361011.361073

[6] Blackblaze, I. Docker Containers vs. VMs: Pros and Cons of Containers and Virtual Machines. (accessed: 15.04.2019). Available from: https://www.backblaze.com/blog/vm-vs-containers/

[7] Docker Inc. Docker. (accessed: 14.04.2019). Available from: https://www.docker.com/

[8] Gordon, W. Understanding OAuth: What Happens When You Log Into a Site with Google, Twitter, or Facebook. (accessed: 25.12.2019). Available from: https://lifehacker.com/understanding-oauth-what-happens-when-you-log-into-a-s-5918086

[9] D. Hardt, E. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, 10 2012. Available from: https://tools.ietf.org/html/rfc6749

[10] Anicas, M. An Introduction to OAuth 2. (accessed: 24.12.2019). Available from: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

[11] Richardson, L.; Ruby, S.; et al. *RESTful Web Services*. O'Reilly, 2007, ISBN 0596529260.

[12] Fielding, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. Available from: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[13] KOSapi contributors. KOSapi. (accessed: 30.12.2019). Available from: https://kosapi.fit.cvut.cz/projects/kosapi/wiki

[14] Jupyter Development Team. The Notebook file format. (accessed: 01.01.2020). Available from: https://nbformat.readthedocs.io/en/latest/format_description.html

[15] Jupyter Development Team. Custom front-end extensions. (accessed: 17.01.2020). Available from: https://jupyter-notebook.readthedocs.io/en/stable/extending/frontend_extensions.html

[16] Project Jupyter team. Technical Overview. (accessed: 22.01.2020). Available from: https://jupyterhub.readthedocs.io/en/stable/reference/technical-overview.html

[17] Project Jupyter team. Quickstart. (accessed: 22.01.2020). Available from: https://jupyterhub.readthedocs.io/en/stable/quickstart.html

[18] Bitnami. Configure Kubernetes Autoscaling With Custom Metrics. (accessed: 22.01.2020). Available from: https://docs.bitnami.com/kubernetes/how-to/configure-autoscaling-custom-metrics/

[19] Ellingwood, J. An introduction to Kubernetes. (accessed: 21.01.2020). Available from: https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

[20] Langemak, J. Kubernetes 101 - Networking. (accessed: 21.01.2020). Available from: http://www.dasblinkenlichten.com/kubernetes-101-networking/

[21] The Kubernetes Authors. Namespaces. (accessed: 01.02.2020). Available from: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

[22] OpenShift Commons Authors. Kubernetes infrastructure. (accessed: 01.02.2020). Available from: https://docs.okd.io/latest/architecture/infrastructure_components/kubernetes_infrastructure.html

[23] Red Hat, Inc. Authors. Overview of a Replication Controller. (accessed: 01.02.2019). Available from: https://coreos.com/kubernetes/docs/latest/replication-controller.html

[24] Project Jupyter team. Authenticators. (accessed: 02.02.2020). Available from: https://jupyterhub.readthedocs.io/en/stable/reference/authenticators.html

[25] České vysoké učení technické v Praze. Apps Manager. (accessed: 02.02.2020). Available from: https://auth.fit.cvut.cz/manager/index.jsf

[26] jupyterhub. JupyterHub. (accessed: 02.02.2020). Available from: https://hub.docker.com/r/jupyterhub/jupyterhub/

[27] Docker Inc. DockerHub. (accessed: 02.02.2020). Available from: https://hub.docker.com/

[28] Project Jupyter. Project Jupyter. (accessed: 03.02.2020). Available from: https://github.com/jupyter

[29] The Kubernetes Authors. Install and Set Up kubectl. (accessed: 03.02.2020). Available from: https://kubernetes.io/docs/tasks/tools/install-kubectl/

[30] Project Jupyter Contributors. Kubernetes on Amazon Web Services (AWS). (accessed: 03.02.2020). Available from: https://zero-to-jupyterhub.readthedocs.io/en/latest/amazon/step-zero-aws.html

[31] Project Jupyter Contributors. Setting up Helm. (accessed: 03.02.2020). Available from: https://zero-to-jupyterhub.readthedocs.io/en/latest/setup-jupyterhub/setup-helm.html

[32] Project Jupyter Contributors. kubespawner. (accessed: 04.02.2020). Available from: https://github.com/jupyterhub/kubespawner

[33] Project Jupyter team. KubeSpawner. (accessed: 04.02.2020). Available from: https://jupyterhub-kubespawner.readthedocs.io/en/latest/spawner.html

[34] Project Jupyter team. JupyterHub and BinderHub Helm charts for Kubernetes. (accessed: 04.02.2020). Available from: https://jupyterhub.github.io/helm-chart/

[35] Jupyter Development Team. Creating and grading assignments. (accessed: 05.02.2020). Available from: https://nbgrader.readthedocs.io/en/stable/user_guide/creating_and_grading_assignments.html

[36] Jupyter Development Team. nbgrader. (accessed: 05.02.2020). Available from: https://nbgrader.readthedocs.io/en/stable/index.html

Appendix A

User manual

A.1 Authentication

A.1.1 Login

Authentication is done using Usermap credentials. To Login:

1. Press the login button
2. You should see login page of university, enter you Usermap credentials and submit the form
3. You should see the list of courses you have in this semester, choose one and submit.

A.1.2 Logout

In order to logout, click the button in the upper-right corner of the screen.

A.2 Student's guide

A.2.1 Working with assignments

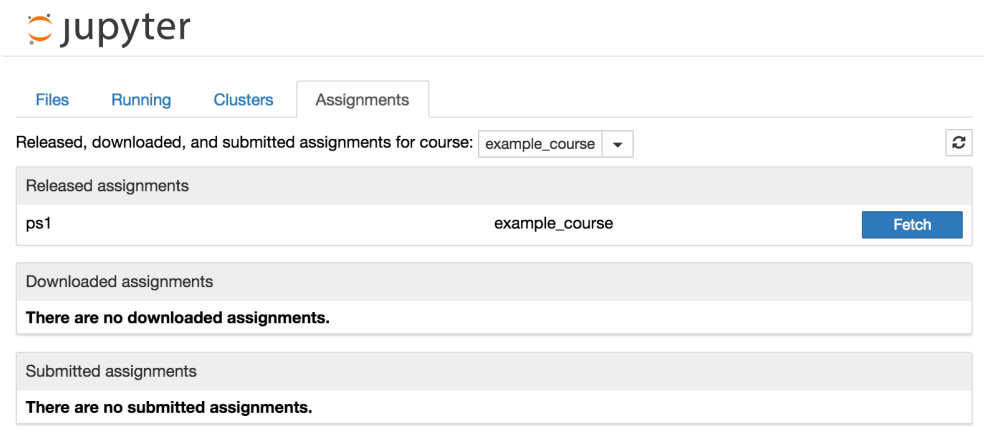

Figure A.1: Fetching assignment

The image above shows that there has been one assignment released ("ps1") for the class "example_course". To get this assignment, click the "Fetch" button

After the assignment is fetched, it will appear in the list of "Downloaded assignments":

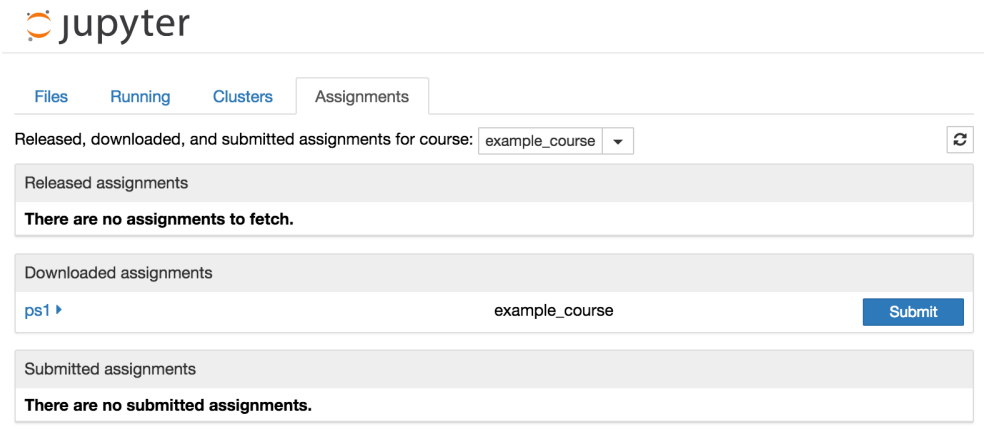

Figure A.2: Downloaded assignment

Click on the name of the assignment to expand it and see all the notebooks in the assignment:

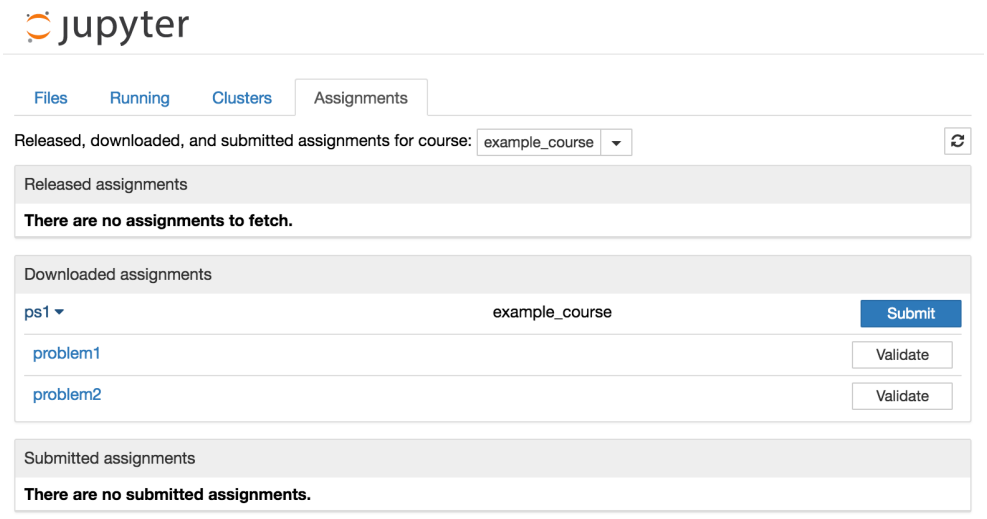

Figure A.3: Assignment list expanded

Clicking on a particular notebook will open it in a new tab in the browser.

After some work on the assignment, but before submitting, you can validate that notebooks pass the tests by clicking the "Validate" button.

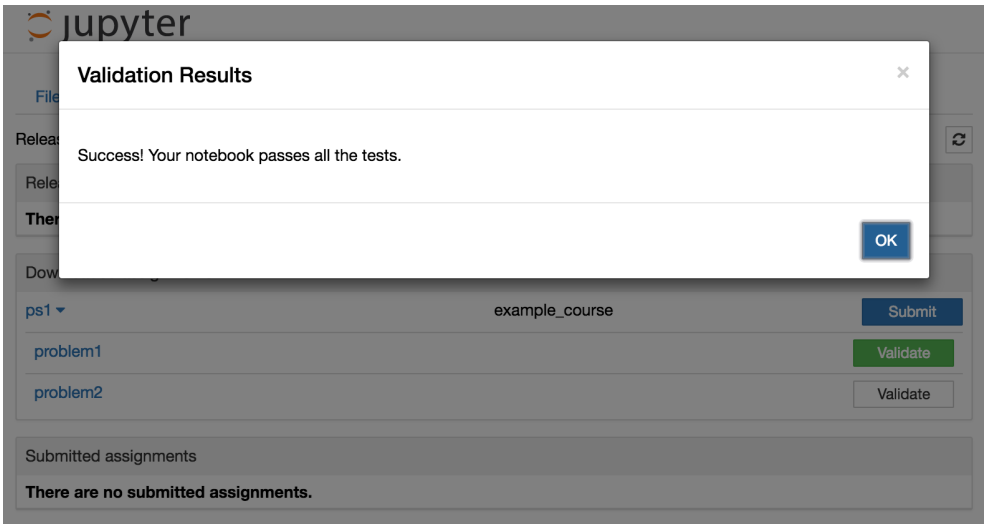

Figure A.4: Assignment passed tests

Once notebooks are validated, you can click the "Submit" button to submit the assignment.

A.3 Teacher's guide

A.3.1 Managing assignments

The formgrader extension provides the core access to nbgrader's instructor tools. You can access it through the tab in the notebook list:

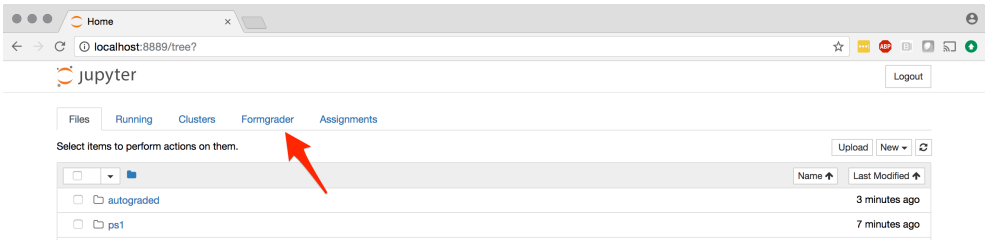

Figure A.5: Formgrader extension

To create a new assignment, open the formgrader extension and click the "Add new assignment..." button at the bottom of the page and fill in the form. Then, you can add files to the assignment and edit them by clicking the name of the assignment:

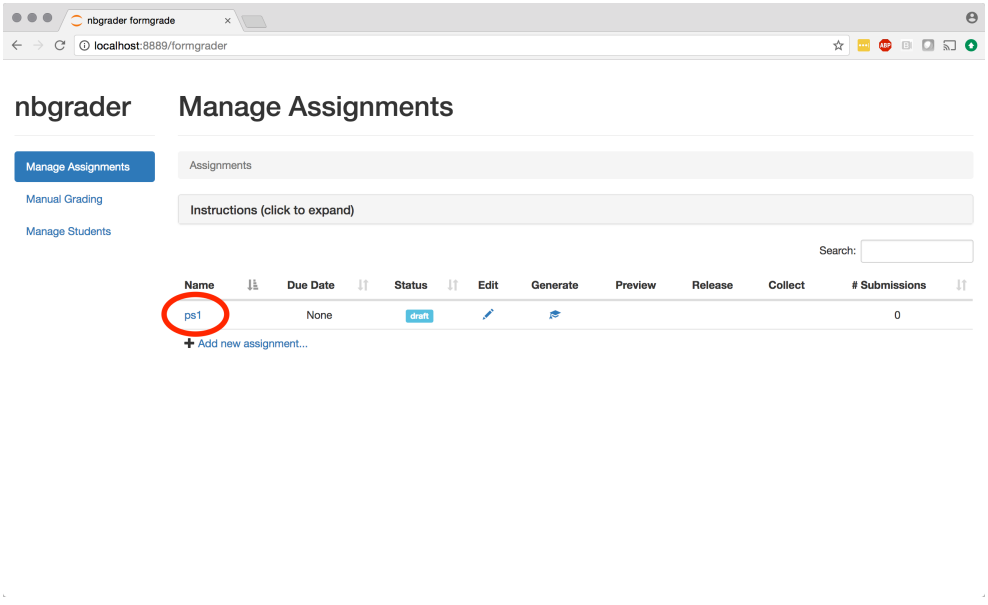

Figure A.6: Creating assignment

Use "Create Assignment" toolbar to create assignment tasks:

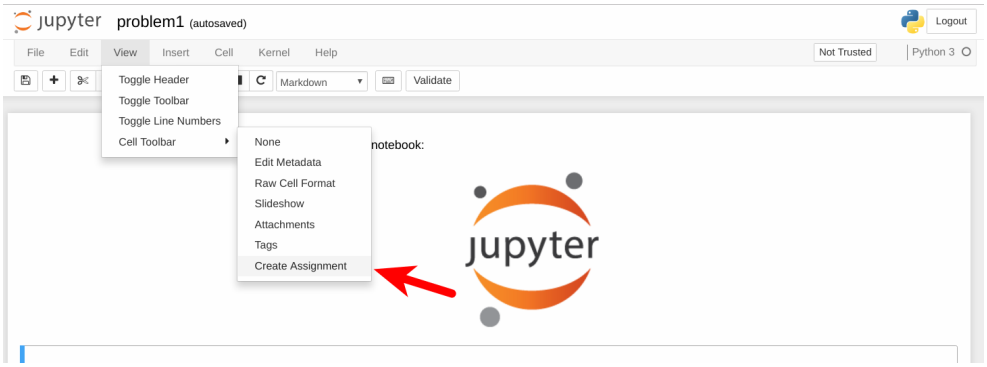

Figure A.7: Assignment toolbar

For more details see [35]

After an assignment has been created with the assignment toolbar, you will want to generate the version that students will receive. You can do this from the formgrader by clicking the "generate" button:

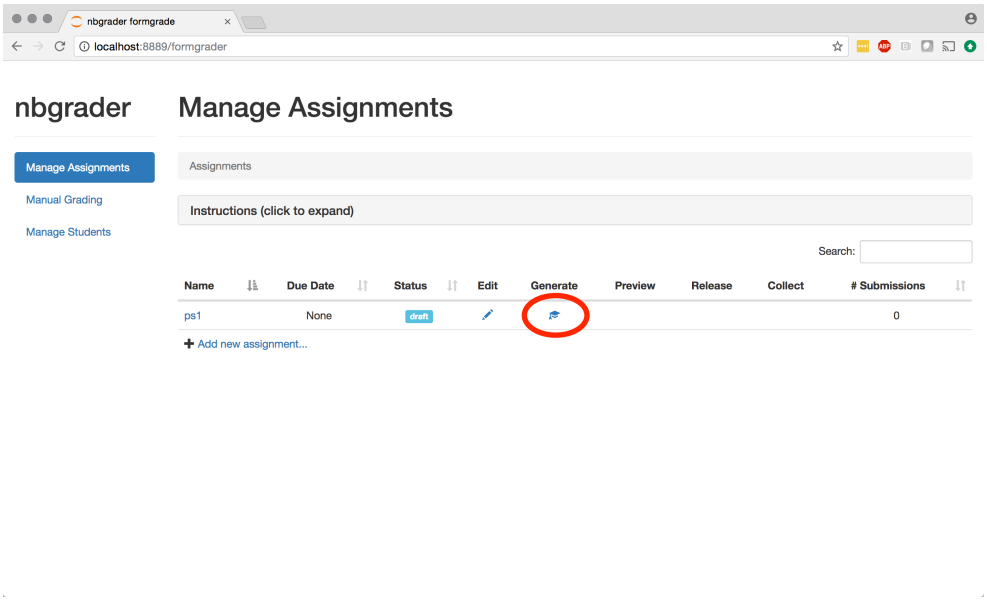

Figure A.8: Generate student version

You may release assignments by clicking on the "release" button.
For more information, please see [36]

Appendix B

Glossary

API Application Programming Interface is an interface or communication protocol between different parts of a computer program.

JSON JavaScript Object Notation is an open standard file format of data, which consists of key-value pairs and array data types.

HTTP Hypertext Transfer Protocol is an application protocol for distributed hypermedia information systems. It's the foundation of data communication for the World Wide Web.

Python Python is an interpreted, high-level, general-purpose programming language.

Package-management system Package-management system is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner.

Repository Repository is a data structure which stores metadata for a set of files or directory structure.

Snapshot Snapshot is the state of a system at a particular point in time.

XML Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

HTML HyperText Markup Language is the most basic building block of the Web. It defines the meaning and structure of web content.

JavaScript Often abbreviated as JS, is a high-level, multi-paradigm programming language that conforms to the ECMAScript specification.

YAML YAML (a recursive acronym for "YAML Ain't Markup Language") is a human-readable data-serialization language.

Appendix C

Acronyms

JN Jupyter notebook

CSV Comma separated values

OS Operating system

VM Virtual machine

LXC Linux containers

CPU Central Processing Unit

RAM Random access memory

URL Uniform Resource Locator

CSS Cascading Style Sheets

Appendix D

Contents of enclosed flash drive

- `readme.txt` ..... the file with flash drive contents description
- `src` ..... the directory of source codes
  - `feedback_md` ..... feedback module sources
  - `helm` ..... helm configuration files
  - `jupyterhub` ..... jupyterhub customization sources
  - `kube` ..... kubernetes configuration files
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format