VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Likai Ren

VAMK.HELP WEB APPLICATION BASED ON FLASK FRAMEWORK AND WEB CRAWLER

Information Technology

2016

ACKNOWLEDGEMENTS

First of all, the idea of this thesis was from a simple Python script “VAMK GPA Calculator”, which was written by me in the Python programming course taught by Mr. Ghodrat Moghadampour in the last year. Fortunately, he became my supervisor this year. I have to show my respect and gratitude to his patient instructions in the teaching as well as during the whole process of my final thesis. Without his help, I could not have finished this thesis in time.

Also, I would like to thank those students who tested my website and reported me with bugs so that I could have possibilities to fix them.

Besides, the enthusiastic fellows on “Stack Overflow” helped me a lot with some tricky problems which racked my brain, but I still could not solve. I would like to thank them very much for spending their precious time to help others selflessly.

Finally, thanks to the financial and mental support from my parents, I could have come to Finland to do the further studies. No words can express my appreciation to them.

Vaasa, 30.05.2016

Likai Ren

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Likai Ren
Title	VAMK.help Web Application Based on Flask Framework and Web Crawler
Year	2016
Language	English
Pages	75
Name of Supervisor	Ghodrat Moghadampour

The purpose of this thesis was to build a web application "VAMK.help" with services that could help students in VAMK deal with some practical problems.

This application was implemented to utilize the Flask web framework to serve the services from the web crawler which could make requests to the servers of Winha or Tritonia and parse the HTML documents to get desired data, and the data would be processed further and exposed to the student in a decent way.

The services were designed to include course reports, course calendars, Tritonia renewals, and GPA change notifications, automatic book renewals as additional services as well.

The project has achieved all devised services. The web application "VAMK.help" has been tested by many students in VAMK. With not only multi-functionalities, but also a decent user experience and visual effects, the application is highly praised by some students in VAMK.

It was a rewarding attempt to solve practical problems through programming in order to make life easier.

Keywords Python, Flask, Web Crawler, VAMK, Tritonia

CONTENTS

LIST OF FIGURES

LIST OF SNIPPETS

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
API	Application Program Interface
CLI	Command Line Interface
CRSF	Cross-site Request Forgery
CSS	Cascading Style Sheet
DES	Data Encryption Standard
GPA	Grade Point Average
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ID	Identification
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
ORM	Object Relational Mapping
PIN	Personal Identification Number
RDS	Relational Database Service
SPA	Single Page Application
SQL	Structured Query Language
SSL	Security Socket Layer

UI	User Interface
URI	Uniform Resource Identifiers
URL	Uniform Resource Locator
VAMK	Vaasan Ammattikorkeakoulu
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language
XSS	Cross-site scripting

1 INTRODUCTION

In this chapter, the introduction including background, motivations, and objectives of the thesis is described.

1.1 Background

One of the most important tasks of programming is to make the life easier. Solving practical problems by programming can be challenging but fun.

The web crawler technology is popular nowadays for grabbing the desired data from websites automatically by programs or scripts, instead of the manually repetitive and tedious work.

Python is one of the most widely used programming languages for various applications. Python not only has good support for the web crawler, but can also easily build a web application by many excellent web frameworks written in Python.

1.2 Motivations

The motivation for this project came from the practical problems for students in VAMK.

Winha is a platform for students in VAMK supplying services such as enrolling in courses, checking grades and credits. However, this system cannot give the students their GPA, which is one of the important academic indexes. Moreover, Winha does not have good support for the Chrome browser. When checking the completions in ISP, the list of courses cannot be displayed completely.

Students do not have the course timetable based on the enrolled courses, but only by the group, which makes it quite challenging for exchange students or students enrolling in courses of different groups to make their timetable manually.

Tritonia is the library from which students in VAMK borrow books. When the books are soon due, the students need to renew them online with their credentials (Login ID, last name, and PIN), which are hard to remember and type.

1.3 Objectives

Due to practical problems that students in VAMK face, the idea of a web application "VAMK.help" that can help them deal with some tough tasks came into being.

This application should implement the Flask web framework to serve the services from the web crawler which makes requests to the servers of Winha or Tritonia and parses the HTML documents to get the desired data. The data should be processed further and nicely exposed to the student.

The services include course reports, course calendars, Tritonia renewals, and GPA change notifications, automatic book renewals as well as additional services.

2 RELEVANT TECHNOLOGIES

In this chapter, a review of relevant technologies involved in the whole project is given.

2.1 Python

Python is a popular interpreted, object-oriented supporting scripting programming language, which was developed by Guido van Rossum in 1989. The first release of Python was in 1991. /1/

The syntax of Python is clear and simple, and one of the unique features is that mandatory whitespace acts as the statement indentation. By Python, it may take much fewer lines of code than C or Java to implement the same task. /1/

Except for its compressive standard libraries, Python also has rich and powerful third-party libraries which endow Python the ability to do almost everything, from mathematical modeling to web applications. Python has excellent web support, and many web crawler frameworks are written in Python.

There are two main versions of Python: Python 2 and Python3. Python 2 is the legacy version, while Python 3 is the future of this language. Python 3 is not compatible with Python 2. The project itself is the key in the selection of the version of Python. In this project, we use Python 2, because of the lack of Python 3 support in some third-party libraries. /2/

2.2 Pip and Virtualenv

Pip, standing for “Pip Installs Python” or “Pip Installs Packages”, is one of most used package management systems for Python, which helps install and manage Python packages. Pip is included by default in Python 2.7.9 and later, and Python 3.4 and later. PyPI (Python Package Index) includes most known packages. /3/

Virtualenv is a tool to make isolated Python Virtual Environments which are very useful when many different projects are requiring different package dependencies.

For instance, project A requires “version 1.x” package, yet project B requires “version 2.x” package. With virtualenv, this dilemma can be handled easily. Moreover, virtualenv can also keep the global “site-packages” clean, therefore, more manageable. /4/

Virtualenv creates a folder containing a copy of necessary Python library and its installation directories. The status of this virtual environment can be changed easily by the scripts of activating and deactivating in this folder. When the virtual environment is activated, the packages would be installed in the current virtual environment, and Python interpreter in this virtual environment would be used. /5/

2.3 Flask Framework and Jinja2

Flask is a minimalistic web application framework written in Python based on WSGI toolkit Werkzeug and template engine Jinja2. Although without major release (the latest version is Flask 0.10.1), Flask is highly recommended as a Python web application framework. /6/

Flask is known as a micro framework, because it is powered by the simple core, with the other desired features as extensions, as if they were implemented by Flask itself. For example, Flask has no database abstraction layer by default, one extension called Flask-SQLAlchemy can support SQLAlchemy perfectly. /6/

Flask has a default development server and is easy to debug. If the application mode is set to debug, whenever files change in this application, the server would detect them and reload automatically to get the application updated. At the same time, Flask provides us with a helpful debugger on the occurrence of exceptions. /7/

URL Routing technique is mapping URLs to the handlers. Route decorator is used to make a variety of routing rules in Flask. When a function is decorated with the route decorator and bound with a URL, this function will become a view function to give the response to the corresponding HTTP requests. /8/

Jinja2, as the default template engine of Flask, is a widely used and powerful templating language for Python programming language, with sandboxed template execution environment as an optional choice to ensure security. Also, Jinja2 has HTML escaping mechanism to prevent XSS attack. /9/

Jinja2 supports the template inheritance, which allows us to build a base template including the common parts of our website and insert some blocks that can be overridden by child templates. /9/

Moreover, Jinja2 is easy to debug. The corresponding line numbers in the template would be pointed when exceptions happen. /9/

With Jinja2, we can render the templates with Python data dynamically, but not mingling the presentational layer (HTML documents) with the logic layer.

2.4 SQLAlchemy

SQLAlchemy is one of the great ORM frameworks in Python. SQLAlchemy is open source written in Python programming language, providing SQL toolkit and ORM tool, under the MIT License. The first version of SQLAlchemy was released in Feb 2006 and quickly spread in the Python community as one of the most popular ORM tools. /10/

As the persistence layer in the application, SQLAlchemy implements ORM technology, which is a mechanism converting data between the relational database and Python programming language, mapping tables to classes, columns to attributes, and rows to objects in Python. /11/

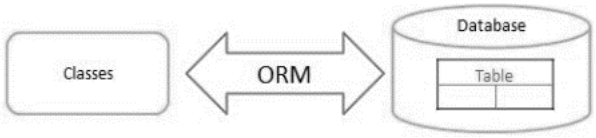

Figure 1. ORM illustration. /12/

2.5 Encryption, AES, and PyCrypto

Encryption is encoding the data in a certain way to make them inaccessible by the unauthorized parties, while only somebody with the decryption key has the access to the encrypted data. Encryption is an effective way to make data secure. /13/

There are two types of encryption: symmetric key encryption and public key encryption. Symmetric key encryption means the same keys are used for encryption and decryption. Public key encryption (or asymmetric cryptography) means the encryption and decryption use the different keys: public key and private key. /13/

Public key encryption is safer than symmetric key encryption but much slower it. One of the best symmetric key encryption AES would be used in this project.

AES is based on Rijndael encryption algorithm, which is a block cipher developed by Joan Daemen and Vincent Rijmen. At the beginning, AES was used by U.S. government, and soon widely used worldwide. /14/

Unlike DES with Feistel network adopted, the design principle of AES is known as a substitution-permutation network, resulting in the faster speed in not only software but also hardware. /14/

PyCrypto is a third-party Python library implementing various encryption algorithms and some secure hash function. With importing PyCrypto, we can easily use AES to encrypt and decrypt the sensitive data in our project. /15/

2.6 Requests

Requests is a third-party HTTP library written in Python. Requests makes HTTP requests simple and Pythonic, known as “HTTP for Humans”. Requests is released under the Apache2 license and is one of the most popular Python packages. /16/

Although Python standard library urllib2 provides most HTTP functionalities, its API is not so handy. Enormous work is needed to handle all kinds of stuff manually with urllib2. The sharp contrast in the lines of codes would illustrate how convenient Requests is compared with urllib2.

Requests is powered by urllib3 and has all features of it. Requests was created for modern web, supporting many features like keep-alive & connection pooling, sessions with cookie persistence, automatic content decoding. /16/

2.7 Beautiful Soup

Beautiful Soup is a third-party Python library for parsing HTML or XML documents and extracting desired data from them, which is very helpful for web scraping. The latest version is Beautiful Soup 4. /17/

Beautiful Soup works with a specific parser to turn even an invalid markup into a parse tree and provides idiomatic and straightforward methods for navigating, searching, and even modifying the parse tree. /17/

2.8 OAuth and Authomatic

OAuth is an open protocol for the secure authorization of third-party application with the accounts such as Facebook, Google, Twitter, but without exposure of their credentials. Authorized the application has limited and delegated access (on behalf of the resource owner) by an access token to resources on the resources server. /18/

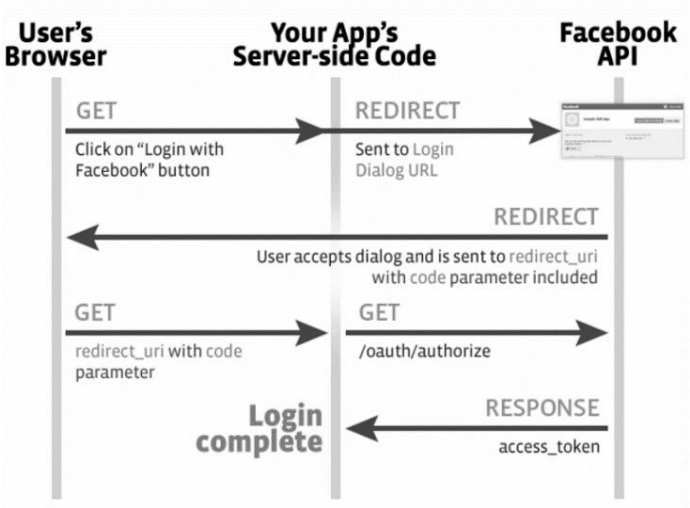

Figure 2. OAuth Login with Facebook process. /19/

The OAuth 2.0 authorization framework is the next version of the OAuth protocol, but it is not backward compatible with OAuth 1.0. Facebook's Graph API supports only Auth 2.0 as the authentication mechanism. /18/

Authomatic is the third-party Python library that has good support for Python web applications like Flask to implement OAuth by third-party providers like Facebook. Authomatic is under MIT license. In this project, Authomatic would be used to handle the Facebook authorization. /20/

2.9 Bower, Bootstrap, and jQuery

Bower is one of the most popular package management systems for the web components that composed of HTML, CSS, JavaScript.

Bower uses package registry mechanism to publish new packages. It is free for everybody to register the package via git repository on Bower server since there is no authentication or user management on the current Bower registry. /21/

With Bower, it is more convenient to install and manage all kinds of web components and handle their dependencies properly.

Bootstrap is one of the most popular front-end frameworks for developing mobile-first and responsive websites. With Bootstrap, we can easily design the front-end of websites without caring about too many details of CSS and JavaScript. /22/

jQuery is a widely-used lightweight JavaScript library that makes JavaScript programming simplified, which facilitates HTML document traversal and manipulation, events handling, animation and AJAX. /23/

2.10 Gunicorn and Supervisor

Gunicorn is a WSGI HTTP Server, which is simple, light and speedy, with various web frameworks compatible. /24/

Flask has its default development web server which is not suitable for production website for both performance and security concerns. There are many popular standalone WSGI containers written in Python, such as Gunicorn, Tornado, and Gevent. In this project, Gunicorn would be used as the Flask web server, and Nginx would proxy it to the Internet. /25/

Supervisor is a process control system that allows users to create, monitor and manage multiple processes on Linux. Supervisor would be used in this project to run the Gunicorn server in the background and start it on reboot automatically. /26/

2.11 Crontab

In Unix-like operating systems, the crontab is a plain configuration text file with commands to be executed at regular intervals in the system background driven by the job scheduler cron daemon. /27/

A job composed by a cron expression and a shell command makes up each line of a crontab file. The job is executed exactly when the current time matches the specific time. /28/

Each user has an individual crontab file, and the jobs are executed periodically no matter whether the user is logged in or not. The system administrators have privileges to edit the system-wide crontab, which is qualified for jobs demanding the administrative privileges. /27/

3 APPLICATION DESCRIPTION

In this chapter, a detailed description of the application is discussed.

3.1 General Description

The objective of this application is to build a web application for VAMK students in order to help them handle trivial and repeating stuff. The application includes the following services:

- Course Report
  Generates the GPA, grade distribution pie chart, and course details. Also supplies the automatic check of GPA changes service to subscribed students.
- Course Calendar
  Generates a course calendar based on the current courses in Winha. A Powerful tool for exchange students who have to enroll in courses of different groups.
- Tritonia Renewal
  An easy point to renew books in Tritonia. Also supplies an automatic renewal service to subscribed students.

3.2 Quality Function Deployment

The requirements of this application can be categorized into three types based on the priorities: must-have requirements, should-have requirements, nice-to-have requirements.

3.2.1 Must-have Requirements

- Checking GPA.
- Checking the grade pie chart.
- Checking the details of courses.
- Generating the personal course calendar.
- Checking books list borrowed in Tritonia.
- Renewing books borrowed in Tritonia.

3.2.2 Should-have Requirements

- Signing in with Facebook.
- Storing the credentials in the database.
- Renewing books borrowed in Tritonia automatically.
- Checking GPA changes automatically.

3.2.3 Nice-to-have Requirements

- Encrypting the credentials.
- Checking the profile of the student.
- Loading the saved course calendar.
- Sorting the course table.
- Downloading the course table by Excel format.
- Downloading the course table by pdf format
- Downloading the course table by png format.
- Supporting HTTPS.
- With Responsive UI.

3.3 Use Case Diagram

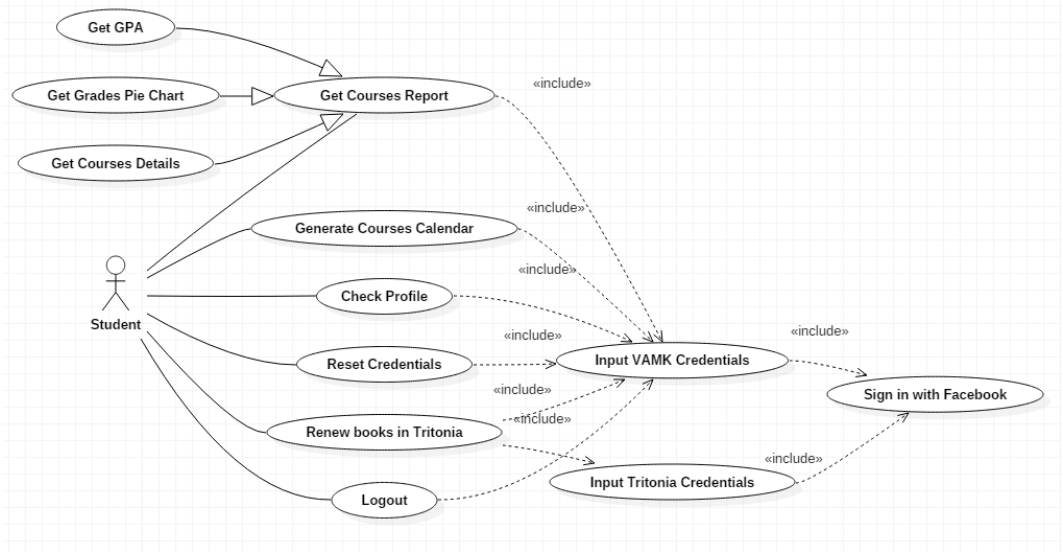

Figure 3. Use Case diagram for the student.

From this Use Case diagram, the student can get the course reports which are specialized by getting GPA, getting grade pie chart and getting course details. The student can also generate a personal course calendar, check the profile, reset credentials, renew books in Tritonia and log out. All of these operations require inputting of correct VAMK credentials. In addition, renewing books in Tritonia also requires the inputting of correct Tritonia credentials. Furthermore, signing in with Facebook is included on top of these.

3.4 Class Diagram

In this project, most modules are implemented in an object-oriented style, but the "views" module of Flask framework and scripts for crontab like "auto_winha", "auto_tritonia" are procedural style, considering the convention and convenience. The main class diagrams of this project are given below.

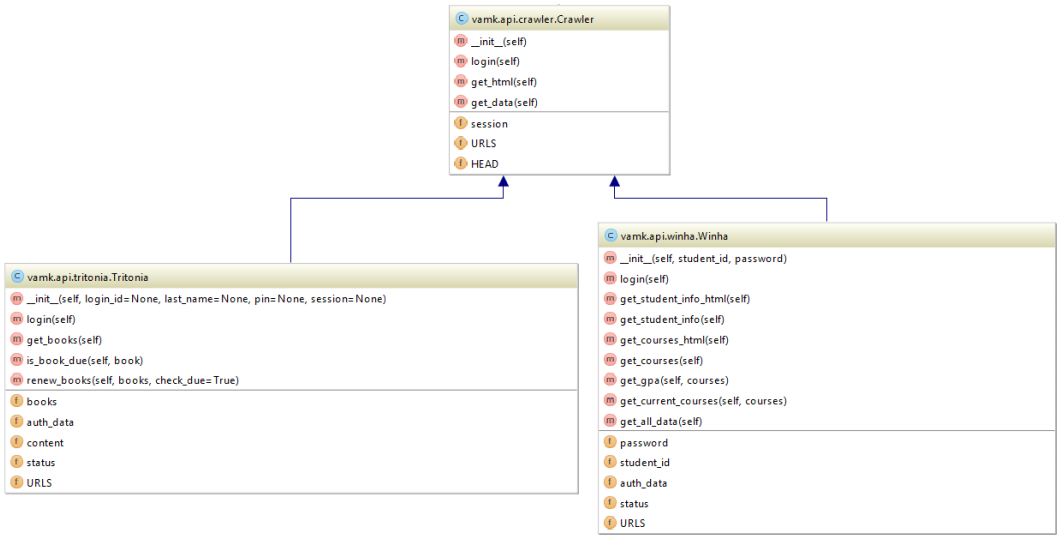

Figure 4. Class diagram for Crawler, Tritonia, Winha.

- Crawler class: Crawler base class, which has the methods for login to the web-site, requesting to get HTML documents and parsing HTML documents to get desired data, but not implemented.
- Winha class: Inherits Crawler class. Simulates the browser to login into the Winha student system, in order to get the desired data based on the HTML documents. The desired data, in this case, are student information, courses, and GPA & grade distribution (based on courses).
- Tritonia class: Inherits Crawler class. Simulates the browser to login into the Tritonia system, in order to get the desired data based on the HTML documents. The desired data, in this case, is the information of books borrowed from Tritonia library. And also includes the method for renewing books.

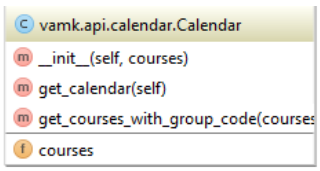

Figure 5. Class diagram for Calendar.

- Calendar class: Generates the course calendar events list based on the given courses.

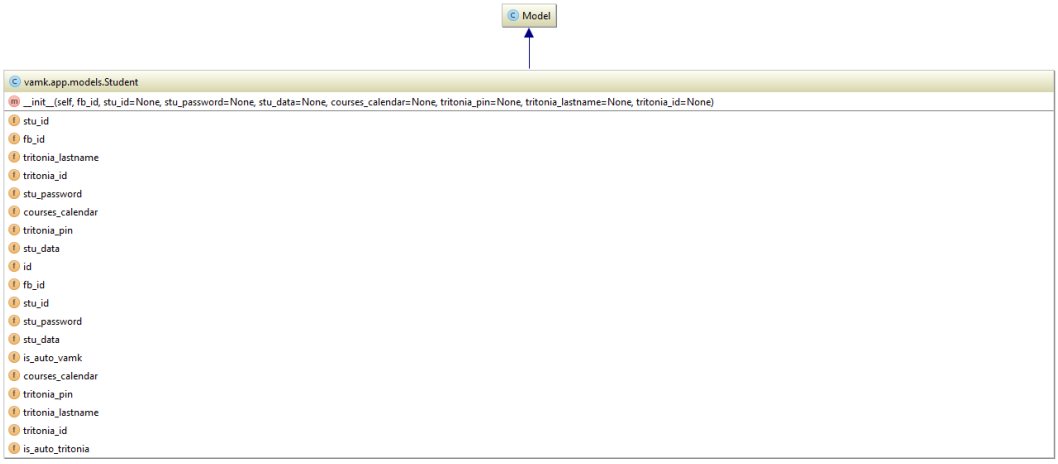

Figure 6. Class diagram for Student.

- Student class: Student model for mapping the student table containing fields fb_id, stu_id, stu_password, stu_data, courses_calendar, tritonia_pin, tritonia_lastname, and tritonia_id.

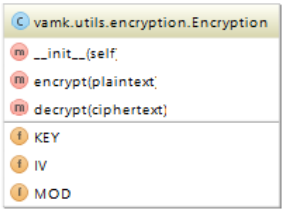

Figure 7. Class diagram for Encryption.

- Encryption class: Includes two static methods for encrypting and decrypting the text. Would be used to encrypt the student credentials to store in the database.

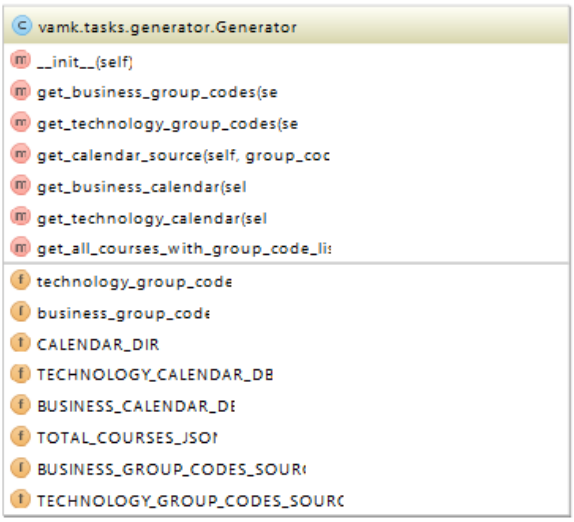

Figure 8. Class diagram for Generator.

- Generator class: Downloads calendars from the school server based on group codes and converts them to the courses events list that can be resolved by Full-Calendar (a jQuery plugin for displaying events on a calendar).

3.5 Sequence Diagram

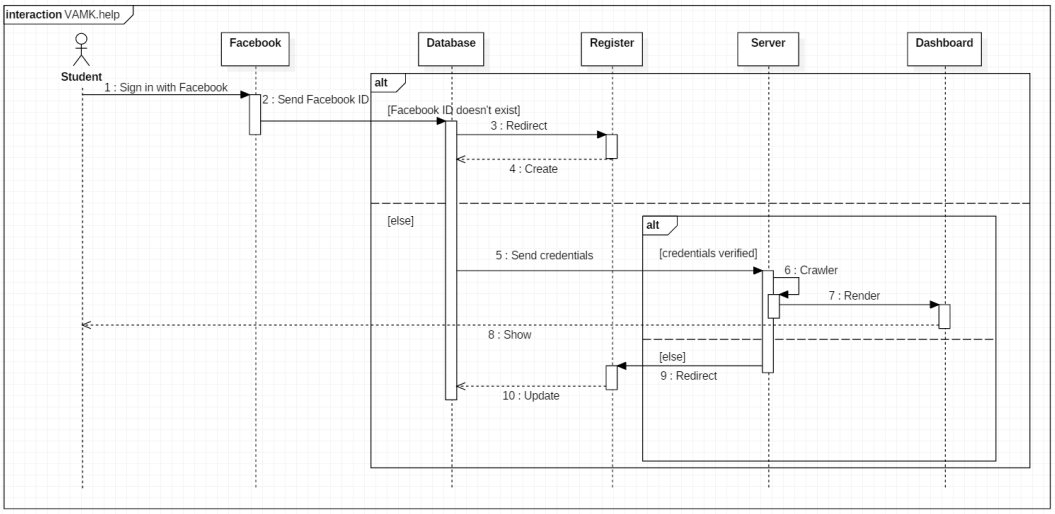

Figure 9. Sequence diagram.

This is the sequence diagram showing interactions of different objects with one another in the process that a student uses the application. At the beginning, the student signs in with his Facebook, and the Facebook ID obtained from the Facebook

server will be sent to the database. Based on the existence of the Facebook ID in the database, two branches are divided. If the Facebook ID does not exist, then the student will be redirected to the register page and feed the credentials to create a new record in the database. Else if the Facebook ID exists, the credentials of the student will be sent to the server. Based on the validity of the credentials, two branches are divided. If the credentials get verified in the Winha server, the server will use crawlers to get the desired the data from the Winha server and process the data, then render the data in a pretty way to the dashboard and display to the student. If the credentials are not valid, the student will be redirected to the register page to feed the credentials again and update the database, then repeat steps 5, 6, 7, 8.

3.6 Component Diagram

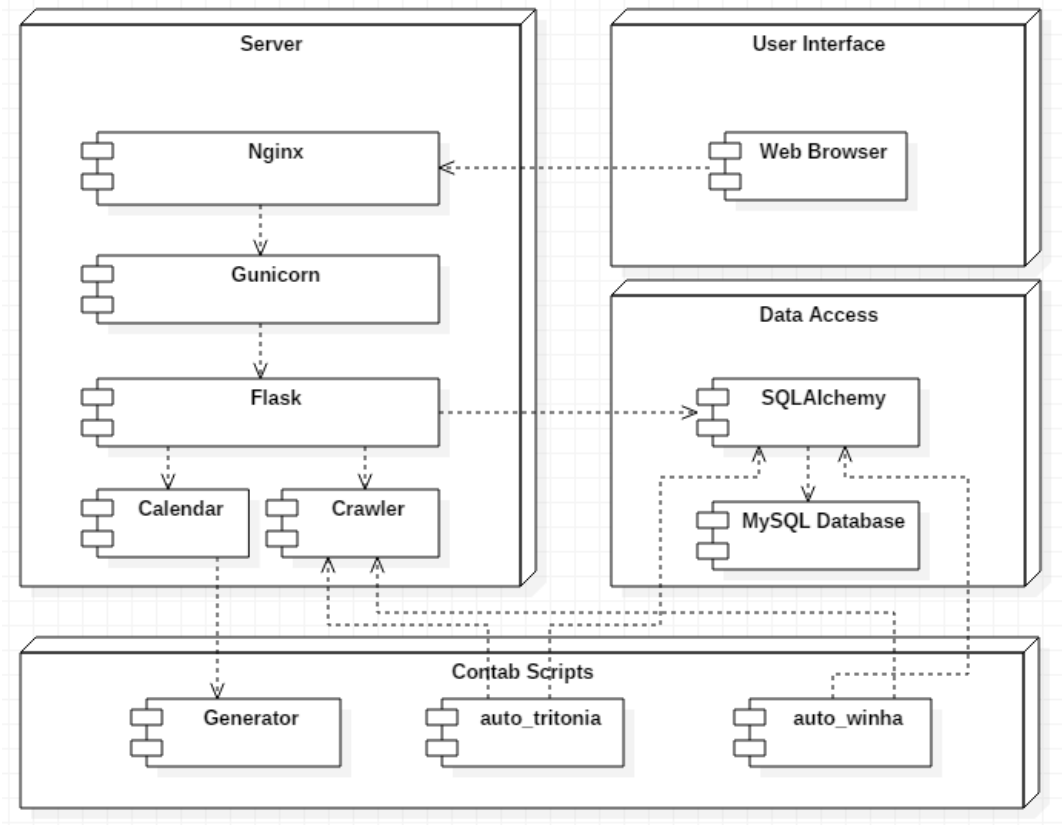

Figure 10. Component diagram.

The component diagram shows how components in this application are connected. This application includes mainly four nodes: “User Interface”, “Server”, “Data Access” and “Crontab Scripts”.

The “User Interface” is in the “Web Browser”. The HTTP requests are handled by “Nginx”, which gives a reverse proxy to “Gunicorn”. “Gunicorn” is a Python WSGI HTTP server as a container for “Flask”. “Crawler” and “Calendar” get the data for “Flask”. “Generator” generates the course calendar database for “Calendar”. “Crawler” also supports “auto_tritonia” and “auto_winha” to make corresponding automation. About “Data Access”, “SQLAchemy” is used for the Object Relational Mapper for the “MySQL Database”.

4 DATABASE AND GUI DESIGN

In this chapter, the design of database and GUI of the application is introduced.

4.1 Database Design

MySQL is used as the database of this application. The database in this application is quite simple. Only one `student` table is needed to store the credentials of VAMK and Tritonia, subscriptions and some important data.

#	Name	Datatype	Length/Set	Unsigned
1	**id**	**INT**	**11**	☐
2	fb_id	VARCHAR	32	☐
3	stu_id	VARCHAR	32	☐
4	stu_password	VARCHAR	64	☐
5	stu_data	TEXT		☐
6	is_auto_vamk	TINYINT	1	☐
7	courses_calendar	TEXT		☐
8	tritonia_pin	VARCHAR	32	☐
9	tritonia_lastname	VARCHAR	32	☐
10	tritonia_id	VARCHAR	32	☐
11	is_auto_tritonia	TINYINT	1	☐

Figure 11. Student table design.

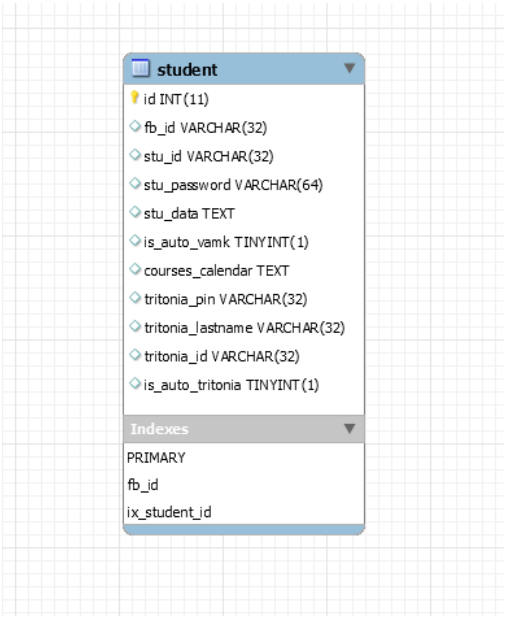

Figure 12. ER diagram.

The `student` table contains the following fields:

- id: The auto-increment id of this record.
- fb_id: Facebook ID.
- stu_id: VAMK student ID.
- stu_password: VAMK student password.
- stu_data: The data got by the Winha crawler.
- is_auto_vamk: Subscription of the VAMK service.
- course_calendar: The course calendar events list.
- tritonia_pin: Tritonia PIN.
- tritonia_lastname: Tritonia last name.
- tritonia_id: Tritonia ID.
- is_auto_tritonia: Subscription of the Tritonia service.

4.2 GUI Design

This application is a web application based on the browser. That is to say, the graphic user interface is implemented by HTML, CSS, and JavaScript. The web pages of this application are user-friendly and responsive. Jinja 2, the default template engine of Flask framework, is used to render the HTML documents. Popular front-end technologies like jQuery, Bootstrap are utilized, as well as some awesome plugins that enhance visual effect like Chart.js, countUp.js, font-awesome.

The main web pages of this application are listed below.

4.2.1 The Home Page

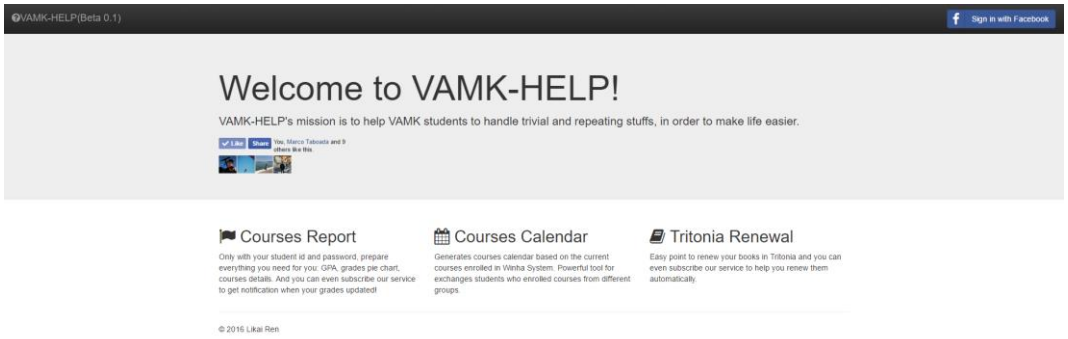

Figure 13. The home page (desktop).

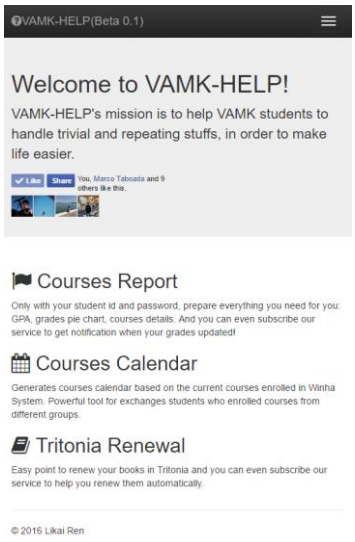

Figure 14. The home page (mobile).

In the home page, the “Sign in with Facebook” button is on the right-left corner. A brief introduction of this website is given, with the Facebook Like & Share plugin blow. Also, the three core services of this website are described in the home page.

4.2.2 The Facebook App Authorization Page

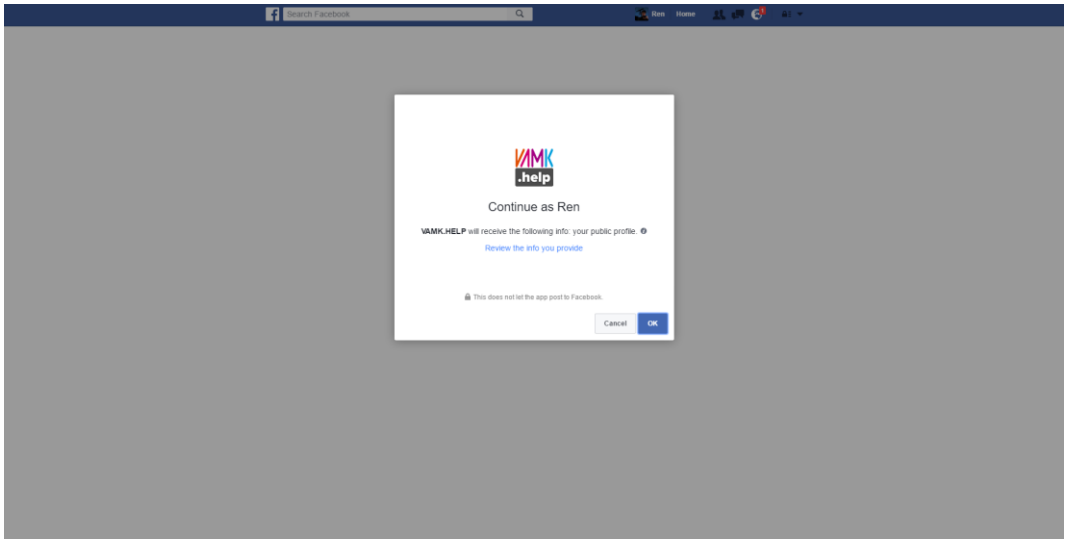

Figure 15. The Facebook app authorization page.

After signing in with a Facebook account, the app authorization page of Facebook would pop up.

4.2.3 The Register Page

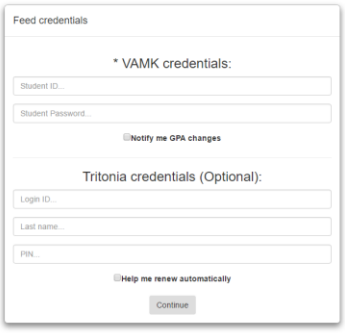

Figure 16. The register page.

If it is the first time for the student to sign in to the application or the VAMK credentials cannot be verified by the Winha server, this register page will be shown

with a modal dialog. In this form, the VAMK credentials are required to fill in, while the Tritonia credentials are optional. When the student checks the checkbox “Notify me GPA changes”, the student subscribes to the service of checking the GPA changes automatically. Moreover, the same applies to the checkbox “Help me renew automatically”. When the student checks it, he/she subscribes to the service of renewing books borrowed from Tritonia library automatically. After feeding the credentials, the “Continue” button should be clicked to send these data to the server.

4.2.4 The Waiting Page

Figure 17. The waiting page.

The waiting page would be shown when the server invokes the crawlers to work, which could take several seconds. To let the student know that the website is working but not dead, a waiting page with a spinning animation gif is designed.

4.2.5 The GPA Page

Figure 18. The GPA page.

The GPA page is the first page when the student gets access to his dashboard. CountUp.js plugin is used to animate the number to count from 0. The font size of the GPA is set to big enough to give the student an impressive visual impact.

4.2.6 The Grade Pie Chart Page

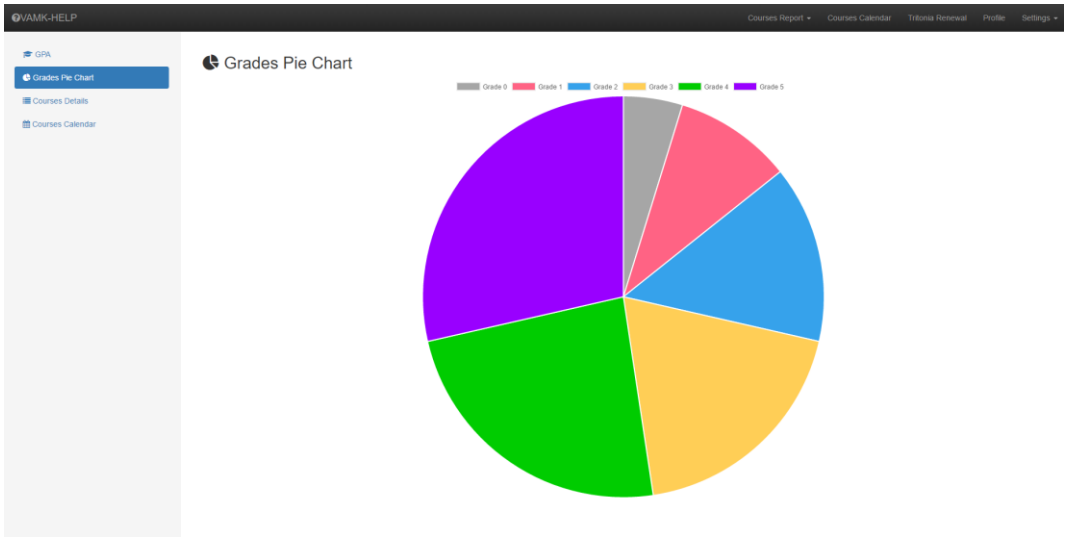

Figure 19. The grade pie chart page.

The grade pie chart page utilizes the Chart.js plugin to depict the pie chart for the courses grade distribution based on the course grades of the student. From 0 to 5

points, different colors are painted to the portions. When the pointer hovers on the specific portion, the number of courses with this grade will be shown. From this grade pie chart, the student can directly find the distribution of courses with different grades.

4.2.7 The Course Details Page

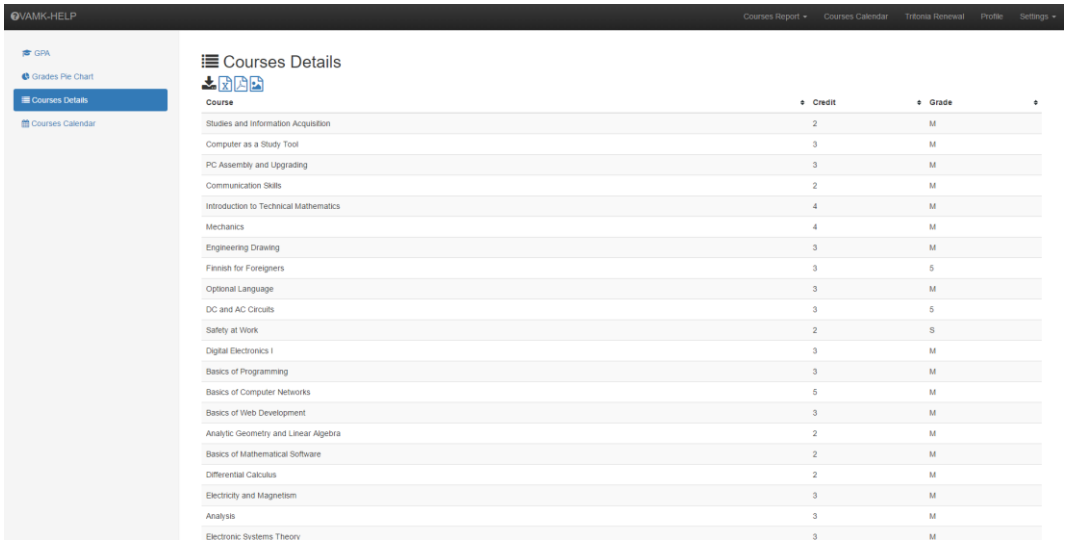

Figure 20. The course details page.

The course details page gives the student a table with all the courses that have been registered in the Winha system. The table has three columns: “Course”, “Credit”, and “Grade”, and supports sorting by every column. On the top of the table, there is a download icon and three file formats icons. The course table can be downloaded in these formats: Excel, pdf, and png (png is not supported by Chrome).

4.2.8 The Course Calendar Selection Page

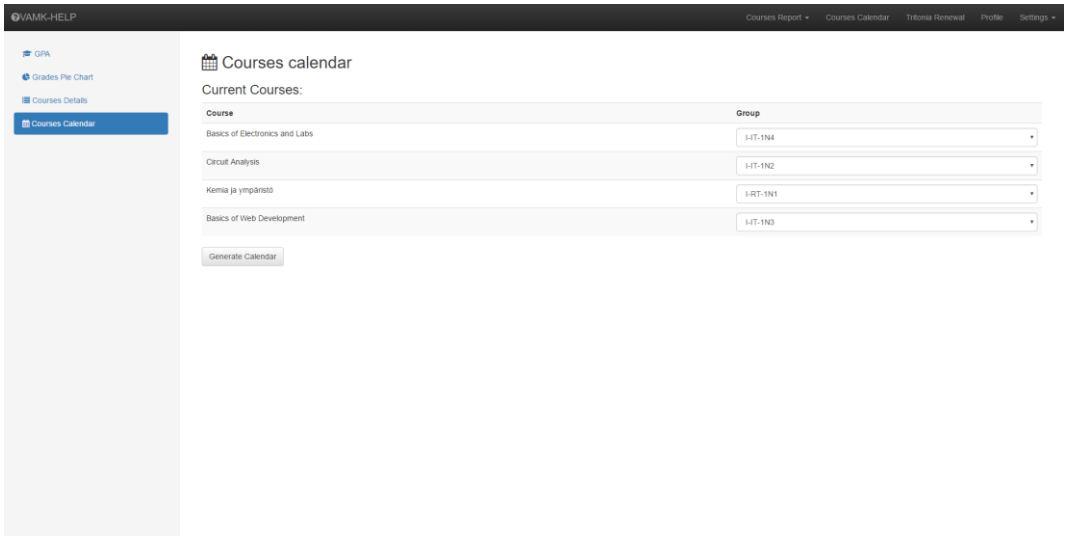

Figure 21. The course calendar selection page.

In the course calendar selection page, the current courses that the student is studying are listed with the corresponding group codes as a dropdown selection. The student can select the groups and then click the “Generate Calendar” button to generate the course calendar.

4.2.9 The Course Calendar Page

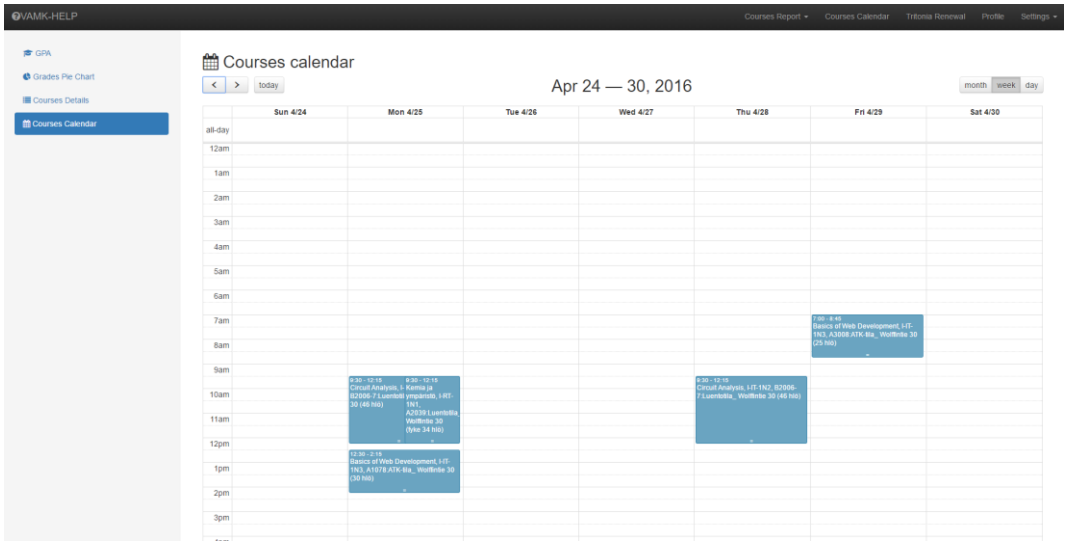

Figure 22. The course calendar page.

The course calendar page is powered by the FullCalendar plugin, which displays the person course calendar generated by the student in the course calendar selection page. The calendar can be shown by three views: month, week and day.

4.2.10 The Tritonia Renewal Page

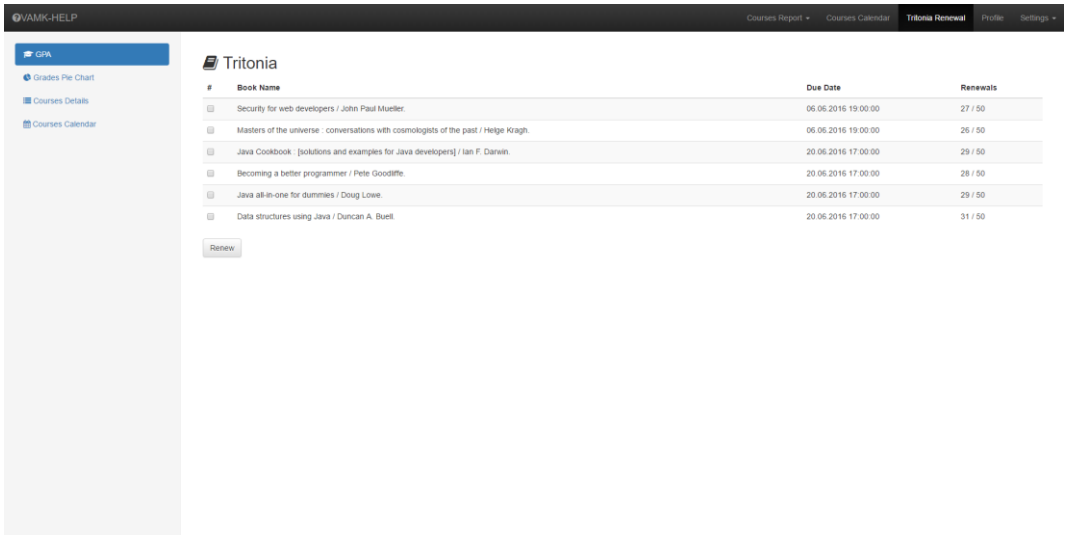

Figure 23. The Tritonia renewal page.

The Tritonia renewal page shows a table that consists of three columns: “Book Name”, “Due Date”, and “Renewals”. The books that the student borrowed from

the Tritonia library will be listed. The student can select one or more books and click the “Renew” button to renew the books.

4.2.11 The Profile Page

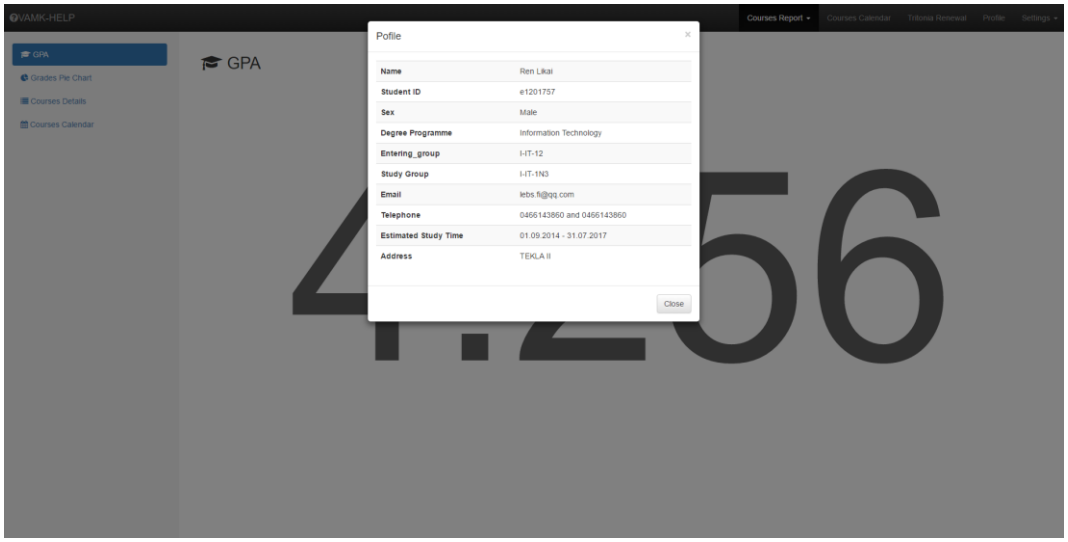

Figure 24. The profile page.

The profile page will pop up when the student clicks the “Profile” on the navigation bar of the dashboard. In the profile modal dialog, some information about the student is given, like name, student ID, sex, degree program, Email.

5 IMPLEMENTATION

In this chapter, detailed implementation of this project is introduced. First, the whole structure of this project is described, and after that, primary modules and the necessary methods are explained respectively.

5.1 Project Structure

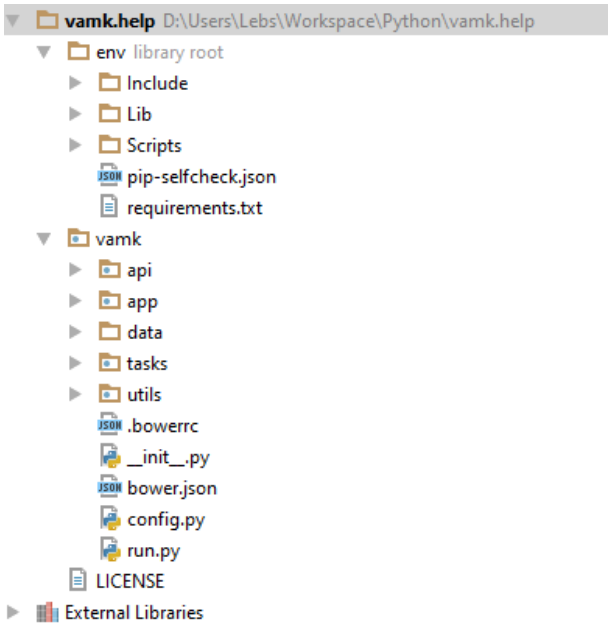

Figure 25. Project structure.

As shown in the project structure diagram, “vamk.help” is the project directory, with “env” and “vamk” directories, and “LICENSE” in it. “env” is the Python Virtual Environment directory, which is the library root of this project.

5.2 Package “vamk”

The package “vamk” is a Python package that contains many sub Python packages and some configuration files, as well as the Python modules. “LICENSE” is a file that declares the license type of this project.

5.2.1 Module “config”

- Configuration of the Flask application

```
# Configuration of Authomatic.
CONFIG = {
    'fb': {
        'class_': oauth2.Facebook,
        # Replacing stars to Facebook APP ID.
        'consumer_key': '******************',
        # Replacing stars to Facebook APP secret.
        'consumer_secret': '******************',
        'scope': [],
    }
}
# Configuration of Flask-SQLAlchemy.
# Replacing stars to Database URI.
SQLALCHEMY_DATABASE_URI = '******************'
SQLALCHEMY_TRACK_MODIFICATIONS = True
# Configuration of Flask.
# Replacing stars to the customized secret key.
SECRET_KEY = '******************'
# DEBUG = True
PORT = 8000
HOST='127.0.0.1'
THREADED = True
```

Snippet 1. Configuration of the Flask application.

5.2.2 Module “run”

- Starting point of the Flask application

```
# Starting the application on the default server (for debugging purpose only).
app.run(host=app.config['HOST'],
        threaded=app.config['THREADED'],
        debug=app.config['DEBUG'],
        port=app.config['PORT'])
```

Snippet 2. Starting point of the Flask application.

This snippet is for running the Flask default server. However, for production purpose of running on the Gunicorn server, the method “run” does not need to be invoked, and this line should be commented.

5.3 Package “api”

The package “api” includes the core modules behind the scene, which are invoked by the Flask.

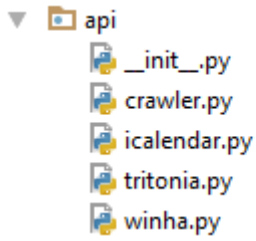

Figure 26. Package “api”.

5.3.1 Module “winha”

- Login the Winha server

```
def login(self):
    """Login the website by the credentials, and keep the session.

    :return: False if login fail, or True if success.
    """
    # Touching the login page to get the initialized session.
    self.session.get(Winha.URLS['ELOGON_URL'], headers=Winha.HEAD)

    # Posting the authentication data to the server.
    self.session.post(Winha.URLS['LOGIN_URL'], data=self.auth_data, headers=Winha.HEAD)

    # Touching the emainval page.
    response = self.session.get(Winha.URLS['EMAINVAL_URL'], headers=Winha.HEAD)

    # Determining the login status by the status code of the response.
    if response.status_code == 500:
        return False
    else:
        return True
```

Snippet 3. Login the Winha server.

The URLs used in the Winha crawler are put in a Python dict “URLS”.

Before posting the authentication data to the server, the login page must be accessed because of the CSRF protection of the Winha website. Otherwise, the crawler will not have the right to log in.

- Requesting the HTML document of courses

```
def get_courses_html(self):
    """Gets the HTML document of the courses page.

    :return: courses_html: the string of the HTML document of the courses page.
    """
    # Touching the ehopssis page.
    self.session.get(Winha.URLS['EHOPSSIS_URL'], headers=Winha.HEAD)

    # Requesting all of the courses information.
    response = self.session.get(Winha.URLS['EHOPSSIS_KAIKKI_URL'],
                                headers=Winha.HEAD)
```

Snippet 4. Requesting the HTML document of courses.

- Parsing the HTML document of courses

```
def get_courses(self):
    """Parses the HTML document of the courses page, in order to get the desired data.

    :return: a dict: {'courses': courses}, courses is a list containing course dicts:
    {'name': name, 'credit': credit, 'status': status, 'grade': grade}
    """
    courses_html = self.get_courses_html()

    # soup is a BeautifulSoup object, which represents the document as a nested data
structure.
    # 'html.parser' is used for HTML parsing.
    soup = bs4.BeautifulSoup(courses_html, 'html.parser')

    # Initializing the courses list.
    courses = []

    # Based on the HTML document, getting the desired data.
    for nobr in soup.find_all('nobr'):
        a_tags = nobr.find_all('a')
        if a_tags:
            if nobr.nobr is not None:
                name = nobr.nobr.string.strip()
            else:
                name = a_tags[0].string.strip()
            details = a_tags[1].string.strip()

            # Using regex group to match different parts: credit, status, grade.
            m = re.match(r'\(([\d,]+)\s*\S+\s*/\s*(\S+)\s*/\s*(\S+)\s*\)', details)

            # Notice that group(0) matches the whole regex expression.
            # Changing the decimal format for easily computing.
            credit = m.group(1).replace(',', '.')
            status = m.group(2)
            grade = m.group(3)

            # Appending the course into courses list.
            course = {'name': name, 'credit': credit, 'status': status, 'grade':
grade}
            courses.append(course)
    return {'courses': courses}
```

Snippet 5. Parsing the HTML document of courses.

Parsing the HTML documents, especially those not so well formed, to get the desired data is the trickiest part in the crawler. The HTML documents should be examined carefully to find the regular patterns, and the parsing utilities like Beautiful

Soup can be helpful to grab the desired data, combined with regular expressions sometimes if necessary.

- Calculating the GPA

```
def get_gpa(self, courses):
    """Calculates GPA and grade distribution based on the provided courses.

    :param courses: a dict: {'courses': courses}, courses is a list containing
course dicts:
    {'name': name, 'credit': credit, 'status': status, 'grade': grade}.

    :return: courses_result: a dict: {'gpa': gpa, 'grade_distribution':
grade_distribution}.
    """

    # Initializing gpa as float type.
    gpa = 0.0
    credits_sum = 0.0
    courses_result = {'gpa': 0, 'grade_distribution': [0, 0, 0, 0, 0, 0]}

    # Getting courses list
    courses = courses['courses']

    for c in courses:
        grade = c['grade']
        credit = eval(c['credit'])
        # Only number grade is considered.
        if grade.isdigit():
            grade = eval(grade)
            # When calculating gpa, the failed course won't be considered.
            if grade != 0:
                courses_result['grade_distribution'][grade] += 1
                credits_sum += credit
                gpa += credit * grade
            if grade == 0:
                courses_result['grade_distribution'][0] += 1
    # gpa = (grades * credits) / credits
    gpa /= credits_sum

    # Rounding gpa to 3 digits after the decimal point.
    courses_result['gpa'] = round(gpa, 3)
    return courses_result
```

Snippet 6. Calculating the GPA.

The further process of the crawled data is interesting. Calculating GPA is not difficult at all, but it should be noticed that the grade is possible to be a letter but not a digit. In this case, only the digit grade is considered to calculate the GPA.

- Getting the current courses

```
def get_current_courses(self, courses):
    """Gets the current courses that are Enrolled or Enrollemnet Accepted but
without grade given.

    :param courses: a dict: {'courses': courses}, courses is a list containing
course dicts:
    {'name': name, 'credit': credit, 'status': status, 'grade': grade}.

    :return: a dict: {'current_courses': current_courses}, current_courses is a
list containing course dicts:
    {'name': name, 'credit': credit, 'status': status, 'grade': grade}.
    """
    # Getting courses list.
    courses = courses['courses']
    current_courses = []
    for c in courses:
        # I = Enrolled; H = Enrollment accepted.
        if c['status'] == 'I' or c['status'] == 'H':
            current_courses.append(c['name'])
    return {'current_courses': current_courses}
```

Snippet 7. Getting the current courses.

The purpose of getting the current courses is for the generating of course calendar which is based on the current courses. Moreover, through the different status of the courses, the current courses are easy to be filtered.

5.3.2 Module “tritonia”

- Login Tritonia

```
def login(self):
    """Login the website by the credentials, and keep the session.
    :return: False if login fail, or True if success.
    """
    # Touching the login page to get the initialized session.
    self.session.get(Tritonia.URLS['LOGON_URL'], headers=Tritonia.HEAD)
    # Posting the authentication data to the server
    self.session.post(Tritonia.URLS['LOGON_POST_URL'], data=self.auth_data,
headers=Tritonia.HEAD)
    # Getting the response from my account page
    response = self.session.get(Tritonia.URLS['MY_ACCOUNT_URL'],
headers=Tritonia.HEAD)
    self.content = response.text
    # Determining the login status by a special string in the HTML document of my
account page.
    if '<title>Kirjaudu sis&auml;&auml;n</title>' in self.content:
        return False
```

Snippet 8. Login Tritonia.

When determining the login status of Tritonia, the status code of the response from the server cannot be used, because it would always be “200”. In this case, the special string from the HTML document is used to verify whether the login is failed.

- Parsing the HTML document of books

```
def get_books(self):
    """Parses the HTML document of my account page, in order to get the desired
data.

    :return: a list containing book dicts {'value': value, 'name': name, 'due':
due, 'renewals': renewals}.
    """
    # soup is a BeautifulSoup object, which represents the document as a nested
data structure.
    # 'html.parser' is used for HTML parsing.
    soup = bs4.BeautifulSoup(self.content, 'html.parser')
    # Initializing the borrowed books list.
    books = []
    # Based on the HTML document, getting the desired data.
    for tr in soup.findAll('tr', {'class': re.compile(r'resultListRow.*')}):
        value = tr.input['value']
        name = tr.find('td', headers='cellChargedItem').string.strip()
        due = tr.find('td', headers='cellChargedDueDate').string.strip()
        renewals = tr.find('td', headers='cellChargedRenewals').string.strip()
        # Appending the book to books list.
        books.append({'value': value, 'name': name, 'due': due, 'renewals':
renewals})
    return books
```

Snippet 9. Parsing the HTML document of books.

This HTML document is easy to parse because the special attributes in the tags really help.

- Checking if the book is due soon

```
def is_book_due(self, book):
    """Determines if a book is due soon by comparing the due date with current
date.
    :param book: a dict: {'value': value, 'name': name, 'due': due, 'renewals':
renewals}.
    :return: if the book is due soon then return True, or return False
    """
    # Converting the due date from a string to datetime object by the Format
'%d.%m.%Y %H:%M:%S',
    # for example: 25.05.2016 19:00:00.
    due_date = datetime.datetime.strptime(book['due'], '%d.%m.%Y %H:%M:%S')

    # Getting the current datetime object.
    now_date = datetime.datetime.now()

    # Two datetime objects subtracting gets a timedelta object to get the
available days before due.
    if (due_date - now_date).days < 2:
        return True
    else:
        return False
```

Snippet 10. Checking if the book is due soon.

Comparing the due date of the book with the current data to determine if the book is due soon. In this snippet, one day is set as the condition of due soon.

- Renewing the books

```
def renew_books(self, books, check_due=True):
    """Renews the provided books. If the check_due is True, the due date would be
checked before the renewals, or
    just renewing without condition.
    :param books: a list containing book dicts {'value': value, 'name': name,
'due': due, 'renewals': renewals}.
    :param check_due: boolean, determines if check the due date before the
renewal.
    :return: a list containing renewed book dicts {'value': value, 'name': name,
'due': due, 'renewals': renewals}.
    """
    due_books = []

    # Building the post data.
    renewal_data = [('renew', 'Request Renewal')]
    for book in books:
        if check_due:
            if self.is_book_due(book):
                due_books.append(book)
                renewal_data.append(('selectCharged', book['value']))
        else:
            due_books.append(book)
            renewal_data.append(('selectCharged', book['value']))

    # Posting the renewals data to the server.
    self.session.post(Tritonia.URLS['RENEWAL_URL'], renewal_data,
headers=Tritonia.HEAD)

    # Updating the latest books.
    self.content = self.session.get(Tritonia.URLS['MY_ACCOUNT_URL'],
headers=Tritonia.HEAD).text
    self.books = self.get_books()

    return due_books
```

Snippet 11. Renewing the books.

In the method “renew_books”, “check_due” is set as one of the parameters for considering two cases in which the method is used. One instance is unconditional renewing when the student renews in the dashboard, while another instance is conditional renewing requiring checking the due date when the “auto_tritonia” script is executed.

When building the post data for renewing information to the Tritonia server, Python date type tuple is used to store different items because there may be duplicated key “selectCharged”.

5.3.3 Module “icalendar”

- Getting the personal course calendar

```
def get_calendar(self):
    """Gets the course calendar events list based on the courses.

    :return: student_calendar: calendar events list.
    """
    student_calendar = []

    # Reading from the course calendar database generated beforehand.
    with io.open(CALENDAR_DB, 'r', encoding='utf-8') as f:
        courses_db = json.loads(f.read())
        for c in self.courses:
            course_name = c['course_name']
            group_code = c['group_code']
            # if the course exists in the course calendar database.
            if course_name in courses_db:
                events = courses_db[course_name][group_code]
            else:
                events = []
            # Do not use append here, because events is a list to be connected
with another list.
            student_calendar += events
    return student_calendar
```

Snippet 12. Getting the personal course calendar.

With the course calendar database, it is super easy to generate the personal calendar.

- Getting courses with group codes

```
def get_courses_with_group_code(courses):
    """Gets the courses with all of their group codes based on the course calendar
database.

    :param courses: a list: [course_name1, course_name2]
    :return: courses_with_group_code: a list containing dicts {course_name1:
[group_code1, group_code2]}.
    """

    courses_with_group_code = []

    # Reading from the course calendar database generated beforehand.
    with io.open(CALENDAR_DB, 'r', encoding='utf-8') as f:
        courses_db = json.loads(f.read())
        for c in courses:
            # if the course exists in the course calendar database.
            if c in courses_db:
                # courses_db[c].keys() is the group_code list.
                courses_with_group_code.append({c: courses_db[c].keys()})
    return courses_with_group_code
```

Snippet 13. Getting courses with group codes.

The reason to get the courses with group codes is that the student can select different course group corresponding to the course from the group codes list.

5.4 Package “app”

The Package “app” is the Flask framework directory including “static” and “templates” directories, as well as modules “models.py” and “view.py”. The directory “static” is for the static files, such as HTML, CSS, and JavaScript files. The directory “templates” is for the Jinja 2 templates to be rendered by Flask.

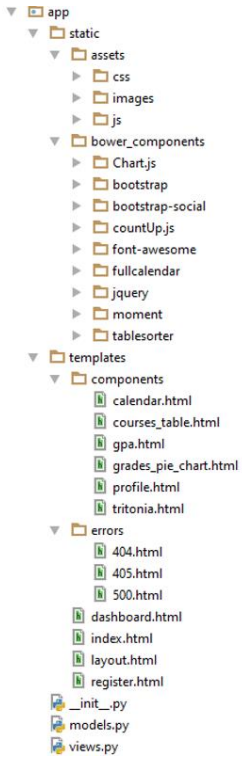

Figure 27. Package “app”.

5.4.1 Module “views”

- Signing in with Facebook

```
@app.route('/login/<provider_name>/', methods=['GET', 'POST'])
def login(provider_name):
    # response object for the WerkzeugAdapter.
    response = make_response()

    # Log the user in, pass it the adapter and the provider name.
    result = authomatic.login(WerkzeugAdapter(request, response), provider_name)

    # If there is no LoginResult object, the login procedure is still pending.
    if result:
        if result.user:
            # Updating the user to get more info.
            result.user.update()

            # Reading the facebook id of the user.
            fb_id = result.user.id

            # If there is no this facebook id in the database then add a new user.
            if Student.query.filter_by(fb_id=fb_id).first() is None:
                student = Student(fb_id)
                db.session.add(student)
                db.session.commit()

            # Saving fb_id to the session.
            session['fb_id'] = fb_id

            # Redirecting to the dashboard.
            return redirect(url_for('dashboard'))
    return response
```

Snippet 14. Signing in with Facebook.

Authomatic library is used to implement the OAuth 2.0, in order to login with Facebook. In this case, only Facebook ID is accessed by the app.

- POST request for student data in JSON format

```
@app.route('/api/student', methods=['POST'])
@login_required
def get_student_info():
    # Getting credentials from request by json format.
    student_id = request.json.get('student_id')
    password = request.json.get('password')

    # Instantiating a Winha crawler object by credentials.
    c = winha.Winha(student_id, password)

    # Returning json format data.
    return jsonify(c.get_all_data())
```

Snippet 15. POST request for student data in JSON format.

- POST request for course calendar in JSON format

```
@app.route('/api/calendar', methods=['POST'])
@login_required
def get_calendar():
    # Getting json format of courses from request
    courses = request.get_json()

    # Instantiating an ICalendar object by courses list.
    i = icalendar.Calendar(courses['courses'])

    # Getting the course calendar events list.
    calendar = i.get_calendar()

    # Save course calendar to the database.
    fb_id = session['fb_id']
    student = Student.query.filter_by(fb_id=fb_id).first()
    student.courses_calendar = json.dumps({'calendar': calendar})
    db.session.commit()

    # Returning json format data.
    return jsonify({'calendar': calendar})
```

Snippet 16. POST request for course calendar in JSON format.

- POST request for Tritonia books data in JSON format

```
@app.route('/api/tritonia/books', methods=['POST'])
@login_required
def get_books():
    # Getting credentials from request by json format.
    login_id = request.json.get('login_id')
    last_name = request.json.get('last_name')
    pin = request.json.get('pin')

    # Instantiating a Tritonia crawler by credentials.
    c = tritonia.Tritonia(login_id, last_name, pin)

    # Returning json format data.
    return jsonify({'books': c.books})
```

Snippet 17. POST request for Tritonia books data in JSON format.

- POST request for renewing Tritonia books

```
@app.route('/api/tritonia/renew', methods=['POST'])
@login_required
def renew_books():
    # Getting credentials from request by json format.
    credentials = request.json.get('credentials')
    login_id = credentials['login_id']
    last_name = credentials['last_name']
    pin = credentials['pin']

    # Getting books list from request by json format.
    books = request.json.get('books')

    # Instantiating a login_id crawler object by credentials.
    c = tritonia.Tritonia(login_id, last_name, pin)
    c.renew_books(books, check_due=False)

    # Returning json format data.
    return jsonify({'books': c.books})
```

Snippet 18. POST request for renewing Tritonia books.

- POST request for registering

```
@app.route('/api/register', methods=['POST'])
@login_required
def update_student():
    # Because login is required, reading fb_id from the session.
    fb_id = session['fb_id']

    # Getting the student based on the fb_id.
    student = Student.query.filter_by(fb_id=fb_id).first()

    # Handling the Winha information.
    student.stu_id = request.form['stu_id']
    # Encrypting the password.
    student.stu_password = Encryption.encrypt(request.form['stu_password'])
    if 'checkbox-vamk' in request.form:
        student.is_auto_vamk = True
    else:
        student.is_auto_vamk = False

    # Handling the Tritonia information.
    student.tritonia_id = request.form['tritonia_id']
    student.tritonia_lastname = request.form['tritonia_lastname']
    student.tritonia_pin = Encryption.encrypt(request.form['tritonia_pin'])  #
encrypting the pin
    if 'checkbox-tritonia' in request.form:
        student.is_auto_tritonia = True
    else:
        student.is_auto_tritonia = False

    # Committing the data into the database.
    db.session.commit()

    # Redirecting to the dashboard page.
    return redirect(url_for('dashboard'))
```

Snippet 19. POST request for registering.

When Storing the credentials to the database, the student password, and Tritonia PIN are encrypted for security reasons.

- Dashboard view

```
@app.route("/dashboard", methods=['GET', 'POST'])
@login_required
def dashboard():
    # Because login is required, reading fb_id from the session.
    fb_id = session['fb_id']

    # Getting the student based on the fb_id.
    student = Student.query.filter_by(fb_id=fb_id).first()
    books = None
    credentials = None

    # if stu_id or stu_password is None, Redirecting to the register page.
    if not student.stu_id or not student.stu_password:
        return render_template("register.html")
    else:
        # Instantiating a Winha crawler object.
        c = winha.Winha(student.stu_id, Encryption.decrypt(student.stu_password))
        # Winha Login fail, redirecting to the register page.
        if c.status is False:
            return render_template("register.html")
        # Winha login success.
        else:
            # Crawling all of te data of the students from the school server.
            data = c.get_all_data()
            # Converting dict to string so as to store in the database.
            student.stu_data = json.dumps(data)
            # Committing the changes to the database.
            db.session.commit()
            # Getting the current courses with group codes based on the data
crawled just now.
            current_courses =
icalendar.Calendar.get_courses_with_group_code(data['current_courses'])
            # If the Tritonia credential is valid.
            if student.tritonia_id and student.tritonia_lastname and
student.tritonia_pin:
                # Building the credentials dict.
                credentials = {'login_id': student.tritonia_id,
                               'last_name': student.tritonia_lastname,
                               'pin': Encryption.decrypt(student.tritonia_pin)}
                # Instantiating a Tritonia crawler object.
                c = tritonia.Tritonia(student.tritonia_id,
student.tritonia_lastname,
                                      Encryption.decrypt(student.tritonia_pin))
                # books would be None if login fail.
                books = c.books
            return render_template("dashboard.html",
                                   data=data,
                                   current_courses=current_courses,
                                   books=books,
                                   credentials=credentials,
                                   courses_calendar=student.courses_calendar)
```

Snippet 20. Dashboard view.

The Dashboard view is the core view for this Flask application. There are many necessary conditions in this routing method to achieve the goal to display all information in the dashboard. Many parameters are passed to the template including the student data, current courses, books, Tritonia credentials and course calendar.

5.4.2 Module “models”

- Model Student

```
class Student(db.Model):
    """Student model for mapping the student table containing fields fb_id,
stu_id, stu_password, stu_data, courses_calendar,
    tritonia_pin, tritonia_lastname, and tritonia_id.
    """
    id = db.Column(db.Integer, primary_key=True, index=True)
    fb_id = db.Column(db.String(16), unique=True, nullable=False)
    stu_id = db.Column(db.String(32))
    stu_password = db.Column(db.String(32))
    stu_data = db.Column(db.Text)
    is_auto_vamk = db.Column(db.Boolean)
    courses_calendar = db.Column(db.Text)
    tritonia_pin = db.Column(db.String(32))
    tritonia_lastname = db.Column(db.String(32))
    tritonia_id = db.Column(db.String(32))
    is_auto_tritonia = db.Column(db.Boolean)

    def __init__(self, fb_id, stu_id=None, stu_password=None, stu_data=None,
courses_calendar=None, tritonia_pin=None,
                 tritonia_lastname=None, tritonia_id=None):
        self.fb_id = fb_id
        self.stu_id = stu_id
        self.stu_password = stu_password
        self.stu_data = stu_data
        self.courses_calendar = courses_calendar
        self.tritonia_pin = tritonia_pin
        self.tritonia_lastname = tritonia_lastname
        self.tritonia_id = tritonia_id
```

Snippet 21. Model Student.

The class Student maps to the table ‘student’ in the MySQL database through the ORM technology implemented by the Flask-SQLAlchemy extension.

5.5 Directory “data”

The directory “data” is for storing the generated course calendar related files.

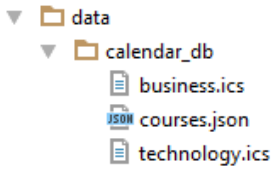

Figure 28. Package “data”.

5.6 Package “tasks”

The package “tasks” is for some Python scripts that repeatedly run by the Crontab.

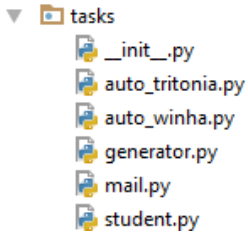

Figure 29. Package “tasks”.

5.6.1 Module “auto_vamk”

- Checking GPA changes automatically

```
if __name__ == '__main__':
    # Checking if the student subscribes the service.
    for student in session.query(Student).filter_by(is_auto_vamk='1'):
        if student.stu_data:
            print student.stu_id + ': '
            # Instantiating a Winha crawler object by credentials.
            c = Winha(student.stu_id, Encryption.decrypt(student.stu_password))
            # If login success.
            if c.status is True:
                # Crawling all of te data of the students from the school server.
                data = c.get_all_data()
                # Checking if the grades change by the grade distribution.
                if data['grade_distribution'] !=
json.loads(student.stu_data)['grade_distribution']:
                    student.stu_data = json.dumps(data)
                    # Updating the student information in the database.
                    session.commit()
                    print 'updated'
                    content = """Hi! You have new course(s) grade updated. Check
here: https://vamk.help
                    """
                    mail.mail(student.stu_id + '@edu.vamk.fi', content)
                else:
                    print 'no update'

            # If login fail.
            else:
                print 'Login fail!'
```

Snippet 22. Checking GPA changes automatically.

This script easily implements checking GPA change automatically by the grade distribution change of the specific student, and sending Email to the school mail of the student to give a notification if the GPA changes.

5.6.2 Module “auto_tritonia”

- Renewing books automatically

```
if __name__ == '__main__':
    # Checking if the student subscribes the service.
    for student in session.query(Student).filter_by(is_auto_tritonia='1'):
        print student.tritonia_id + ': '
        # Instantiating a Tritonia crawler object by credentials.
        c = Tritonia(student.tritonia_id, student.tritonia_lastname,
Encryption.decrypt(student.tritonia_pin))
        # If login success.
        if c.status is True:
            # Printing the renewed books to the console.
            print 'Renewed: ' + repr(c.renew_books(c.books))

            # Mailing to the student to notify the results.
            content = """Hi! Your books were renewed. Check here:
https://vamk.help
            """
            mail.mail(student.stu_id + '@edu.vamk.fi', content)
        # If login fail.
        else:
            print 'Login fail!'
```

Snippet 23. Renewing books automatically.

This script easily implements renewing the books in Tritonia automatically by checking if the books are soon due, and sending an Email to the school mail of the student with a notification if some books are renewed successfully.

5.6.3 Module “mail”

- Mailing by the MailGun

```
def mail(email, text):
    """Mails by mailgun.

    :param email: the email address of the recipient.
    :param text: the content of the mail
    """
    # Replcacing the stars with the key.
    key = '******************'
    # Replcacing the stars with the sandbox.
    sandbox = '******************'
    recipient = email
    request_url = 'https://api.mailgun.net/v2/' + sandbox + '/messages'

    request = requests.post(request_url, auth=('api', key), data={
        'from': 'noreply@vamk.help',
        'to': recipient,
        'subject': 'News from VAMK.help',
        'text': text
    })
```

Snippet 24. Mailing by the MailGun.

5.6.4 Module “generator”

- Getting the group codes of the Department of Technology

```
def get_technology_group_codes(self):
    """Gets the group codes of the department of technology by regex.

    :return: technology_group_codes: a list: [GROUP_CODE1, GROUP_CODE2...]
    """
    technology_group_codes = re.findall(r'(I-\S*?):',
Generator.TECHNOLOGY_GROUP_CODES_SOURCE)
    return technology_group_codes
```

Snippet 25. Getting the group codes of the Department of Technology.

The Technology group codes source can be found at http://www.bet.puv.fi/schedule/P4_15_16/week044.htm for example. Then the regular expression is utilized to parse the groups (start with “I-”).

- Getting the group codes of the Department of Business

```
def get_business_group_codes(self):
    """Gets the group codes of the department of business by regex.

    :return: business_group_codes: a list: [GROUP_CODE1, GROUP_CODE2...]
    """
    business_group_codes = re.findall(r'(T-\S*?):',
Generator.BUSINESS_GROUP_CODES_SOURCE)
    return business_group_codes
```

Snippet 26. Getting the group codes of the Department of Business.

The Business group codes source can be found at http://www.bet.puv.fi/studies/lukujarj/2015-2016/K16/week043.htm for example. Then the regular expression is utilized to parse the groups (start with “T-”).

- Downloading the calendar source

```
def get_calendar_source(self, group_code):
    """Download the *.ics from the school server.

    :param group_code: a list: [GROUP_CODE1, GROUP_CODE2...]

    :return:
    """
    # Building the calendar url by the first Character of the group code.
    # For the department of technology.
    if group_code[0] == 'T':
        calendar_url = 'http://www.bet.puv.fi/studies/lukujarj/iCal/' + group_code
+ '.ics'
    # group_code[0] == 'I', for the department of business.
    else:
        calendar_url = 'http://www.bet.puv.fi/schedule/ical/' + group_code +
'.ics'

    # Getting the calendar from the school server.
    r = requests.get(calendar_url)
    return r.text
```

Snippet 27. Downloading the calendar source.

VAMK supplies iCal format calendars for the students. However, the Department of Business and the Department of Technology have the different calendar URL format. Based on the first character of the group code, the type of the department can be determined, in order to get the proper calendar URL to download.

- Generating the course calendar database

```
def get_all_courses_with_group_code_list():
    """Builds the course calendar database.

    :return:
    """
    # courses is a dict: {COURSE_NAME1:{GROUP_CODE1:EVENTS_LIST1,
GROUP_CODE2:EVENTS_LIST2...}
    courses = {}

    # Converting ics to the format of json that can be resolved by fullcalendar.
    for cd in [Generator.BUSINESS_CALENDAR_DB, Generator.TECHNOLOGY_CALENDAR_DB]:
        with io.open(cd, 'r', encoding='utf-8') as f:
            calendar_source = f.read()
            # Building the regex pattern.
            # Parsing the data like [('2016', '04', '04', '11', '30', '2016',
'04', '04', '14', '15', 'A3006',
            # 'I-EM-3N', 'Basics of Mathematical Software')]
            pattern =
r'BEGIN.*?DTSTART:(\d{4})(\d{2})(\d{2})T(\d{2})(\d{2})\d{2}Z.*?DTEND:(\d{4})' \

r'(\d{2})(\d{2})T(\d{2})(\d{2})\d{2}Z.*?LOCATION:(.*?)\n.*?SUMMARY:\[(.*?):.*' \
                    r'?\]\s(.*?)\s?\n.*?END:VEVENT'

            # Finding all strings matches the pattern.
            # DOTALL flag means '.' matches any character including a newline.
            # all_courses_calendars is a list containing tuples.
            all_courses_calendars = re.findall(pattern, calendar_source,
re.DOTALL)

            # Further processing the data to populate courses dict.
            for c in all_courses_calendars:
                course_name = c[12]
                group_code = c[11]
                location = c[10]
                event = {'title': course_name + ', ' + group_code + ', ' +
location,
                         'start': c[0] + '-' + c[1] + '-' + c[2] + 'T' + c[3] +
':' + c[4] + 'Z',
                         'end': c[5] + '-' + c[6] + '-' + c[7] + 'T' + c[8] + ':'
+ c[9] + 'Z'}

                if courses.has_key(course_name):
                    if courses[course_name].has_key(group_code):
                        courses[course_name][group_code].append(event)
                    else:
                        courses[course_name][group_code] = [event]
                else:
                    courses[course_name] = {}
                    courses[course_name][group_code] = [event]

    # Writing to the file to persist the course calendar database.
    with io.open(Generator.TOTAL_COURSES_JSON, 'w', encoding='utf-8') as f:
        f.write(json.dumps(courses, ensure_ascii=False))
```

Snippet 28. Generating the course calendar database.

The course calendar database, from which the student can fleetly generate the personal course calendar, is stored in a JSON format file.

To generate the course calendar database, a regex pattern is used to parse the iCal files, in order to get the desired data which would be rearranged to the JSON format that the FullCalendar plugin can resolve.

5.7 Package “utils”

The package “utils” is for the utility, now only with the module “encryption” in it.

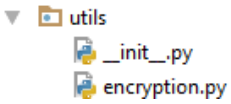

Figure 30. Package “utils”.

5.7.1 Module “encryption”

- Encryption class

```
class Encryption:
    """Includes two static methods for encrypting and decrypting the text. Would
be used to encrypt the student
    credentials to store in the database.
    """
    #  The secret key to use in the symmetric cipher. It must be 16 (AES-128), 24
(AES-192), or 32 (AES-256) bytes long.
    KEY = 'VAMK YOUSELF!!!#'
    # The initialization vector to use for encryption or decryption.
    IV = 'VAMK MYSELF!!!##'
    # The chaining mode to use for encryption or decryption.
    MOD = AES.MODE_CFB

    def __init__(self):
        pass

    @staticmethod
    def encrypt(plaintext):
        """Encrypts the plaintext.

        :param plaintext: string

        :return: ciphertext: string
        """
        cipher = AES.new(Encryption.KEY, Encryption.MOD, Encryption.IV)
        ciphertext = cipher.encrypt(plaintext).encode('hex')
        return ciphertext

    @staticmethod
    # Decryption
    def decrypt(ciphertext):
        """Decrypts the ciphertext.

        :param ciphertext: string

        :return: plaintext: string
        """
        cipher = AES.new(Encryption.KEY, Encryption.MOD, Encryption.IV)
        plaintext = cipher.decrypt(ciphertext.decode('hex'))
        return plaintext
```

Snippet 29. Configuration of the Flask application.

6 DEPLOYMENT

In this chapter, the deployment of this project is discussed.

Although Flask supports a default server, considering a better performance and security, the Gunicorn is used to serve the Flask instead as one of the schemes of the production website. On top of that, Nginx is used to reverse proxy the Gunicorn. For running the Gunicorn server in the background and starting it on reboot automatically, Supervisor is used.

The following operations are assuming that Ubuntu is used as the operating system, and the directory of this application is "/var/www/vamk.help".

6.1 Python Virtual Environment

Creating the Python Virtual Environment for this application:

- `/var/www/vark.help# virtualenv env`
- `/var/www/vark.help# . env/bin/activate`

Installing the Python packages by "requirements.txt":

- `(env) /var/www/vark.help# pip install vamk/requirements.txt`

6.2 PYTHONPATH Environment Variable

Adding the Python package "vamk" to the PYTHONPATH Environment Variable by appending the following line to the end of ".bashrc" file in the home directory.

- `export PYTHONPATH=$PYTHONPATH:/var/www/vamk.help/`

6.3 MySQL Database

In this project, an instance in the RDS of Amazon Web Service is used as the supplier for the MySQL database.

After creating a database instance, the endpoint can be obtained from the RDS dashboard. The endpoint is used to connect to the instance.

With the connection of the database instance, the database ‘vamk_help_db’ should be created manually.

6.4 Nginx

The configuration of Nginx is as following. In this configuration, all HTTP requests are redirected to HTTPS.

```
server {
        listen 443 ssl;
        server_name vamk.help;
        ssl_certificate /etc/nginx/ssl/1_vamk.help_bundle.crt;
        ssl_certificate_key /etc/nginx/ssl/vamk.key;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers ALL:!DH:!EXPORT:!RC4:+HIGH:+MEDIUM:!LOW:!aNULL:!eNULL;

        # Handle all locations
        location / {
        root /var/www/vamk.help;
                proxy_pass http://127.0.0.1:8000;
                # Set some HTTP headers so that our app knows where the request
really came from
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
}

server {
        listen 80 default_server;
        listen [::]:80 default_server;
        server_name vamk.help;
        return 301 https://$server_name$request_uri;
}
```

Snippet 30. Configuration of Nginx.

6.5 Supervisor

Creating a Supervisor configuration file:

- `(env) /var/www/vark.help# echo_supervisord_conf > supervisor.conf`

Appending the following snippet to the end of the file:

```
[program: vamk.help]
command = /var/www/vamk.help/env/bin/gunicorn run:app -w 4
directory = /var/www/vamk.help/vamk
user = root
stdout_logfile = /var/www/vamk.help/gunicorn/gunicorn_stdout.log
stderr_logfile = /var/www/vamk.help/logs/gunicorn/gunicorn_stderr.log
redirect_stderr = True
environment = PRODUCTION=1
```

Snippet 31. Configuration of Supervisor.

The following are the basic usage of Supervisor.

To start Supervisor by the Supervisor configuration file:

- supervisord -c supervisor.conf

To reload Supervisor:

- supervisorctl -c supervisor.conf reload

To check the status of Supervisor:

- supervisorctl -c supervisor.conf status

To reload Supervisor:

- supervisorctl -c supervisor.conf reload

To start all apps or a specific app in a Supervisor configuration file:

- supervisorctl -c supervisor.conf start [all]|[appname]

To stop all apps or a specific app in a Supervisor configuration file:

- supervisorctl -c supervisor.conf stop [all]|[appname]

6.6 Crontab

Editing the crontab file:

- `(env) /var/www/vamk.help# crontab –e`

Appending the following snippet to the end of the file.

```
00 00 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/generator.py"

00 00 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 09 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 12 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 14 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 16 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 18 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 20 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"
00 22 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_vamk.py"

00 00 * * * bash -l -c "python /var/www/vamk.help/vamk/tasks/auto_tritonia.py"
```

Snippet 32. Tasks in the crontab.

“generator.py” is set to be run on 00.00 every day.

“auto_vamk.py” is set to be run on 00.00, 09.00, 12.00, 14.00, 16.00, 18.00, 20.00, 22.00 every day.

“au_tritonia.py” is set to be run on 00.00 every day.

“.bashrc” is loaded for adding the project Python packages to PYTHONPATH Environment Variable before running the scripts.

7 TESTING

In this chapter, test cases for this application are given.

7.1 Signing in for the New Student

- Test Step
  Go to the home page and click the button “Sign in with Facebook”. With the state of signing in with Facebook, the authorization dialog will pop up, and click “OK”.
- Expected Result
  It should redirect to register page.
- Actual Outcome
  It redirects to register page.

7.2 Feeding Credentials in the Register page

- Test Step
  In the register page, fill the VAMK credentials as required, and fill other parts as optional. And click “Continue”.
- Expected Result
  If the VAMK credentials are not filled, it should show “Please fill out this field”. If the VAMK credentials cannot be verified in the Winha Server, it should stay on the register page.
- Actual Outcome
  If the VAMK credentials are not filled, it shows “Please fill out this field”. If the VAMK credentials cannot be verified in the Winha Server, it stays on the register page.

7.3 Signing in for the Registered Student

- Test Step
  Go to the home page and click the button “Sign in with Facebook”.
- Expected Result

If the VAMK credentials can be verified in the Winha system, it should be redirecting to dashboard page with the loading animation; or redirecting to the register page.

- Actual Outcome

  If the VAMK credentials can be verified in the Winha system, redirecting to dashboard page with the loading animation; or redirecting to the register page.

7.4 Checking GPA

- Test Step

  After signing in, in the dashboard page, click “GPA”.

- Expected Result

  The GPA should be calculated correctly based on the course grades.

- Actual Outcome

  The GPA is calculated correctly based on the course grades.

7.5 Checking Grade Pie Chart

- Test Step

  After signing in, in the dashboard page, click “Grade Pie Chart”.

- Expected Result

  The Grade Pie Chart should show correctly based on the distribution of grades.

- Actual Outcome

  The Grade Pie Chart shows correctly based on the distribution of grades.

7.6 Checking Course Details

- Test Step

  After signing in, in the dashboard page, click “Course Details”.

- Expected Result

  The detailed course table should be shown correctly containing the course list with course name, credit, and grade.

- Actual Outcome

The detailed course table should be shown correctly containing the course list with course name, credit, and grade.

7.7 Sorting the Course Table

- Test Step
  After signing in, in the dashboard page, click “Course Details”, and click the column head “Course”, “Credit”, “Grade” respectively.
- Expected Result
  When the “Course” is clicked, the table should be sorted by course name; when the “Credit” is clicked, the table should be sorted by credit; when the “Grade” is clicked, the table should be sorted by grade.
- Actual Outcome
  When the “Course” is clicked, the table is sorted by course name; when the “Credit” is clicked, the table is sorted by credit; when the “Grade” is clicked, the table is sorted by grade.

7.8 Downloading the Course Table by Excel Format

- Test Step
  After signing in, in the dashboard page, click “Course Details”, and click the icon of Excel.
- Expected Result
  The course table should be downloaded as an Excel document.
- Actual Outcome
  The course table is downloaded as an Excel document.

7.9 Downloading the Course Table by Pdf Format

- Test Step
  After signing in, in the dashboard page, click “Course Details”, and click the icon of pdf.
- Expected Result
  The course table should be downloaded as a pdf document.

- Actual Outcome
  The course table is downloaded as a pdf document.

7.10 Downloading the Course Table by Png Format

- Test Step
  After signing in, on the dashboard page, click “Course Details”, and click the icon of the picture.
- Expected Result
  The course table should be downloaded as a png picture.
- Actual Outcome
  The course table is downloaded as a png picture.

7.11 Generating Course Calendar

- Test Step
  After signing in, on the dashboard page, click “Course Calendar”. After selecting the group of the corresponding course, click “Generate Calendar”.
- Expected Result
  A course calendar rendered by the FullCalendar plugin should be inserted before the generating table. Moreover, there should be the correct events of the current courses of the student.
- Actual Outcome
  A course calendar rendered by the FullCalendar plugin is inserted before the generating table. Moreover, there are the correct events of the current courses of the student.

7.12 Loading the Saved Course Calendar

- Test Step
  After signing in, on the dashboard page, click “Course Calendar”.
- Expected Result
  If the student has generated course calendar before, it should show the course calendar generated last time.

- Actual Outcome

  If the student has generated course calendar before, it shows the course calendar generated last time.

7.13 Checking Books in Tritonia

- Test Step

  After signing in with the correct Tritonia credentials, in the dashboard page, click “Tritonia Renewal”.
- Expected Result

  The books table that consists of three columns: Book Name, Due Date, and Renewals should show. The books that the student borrowed from the Tritonia library should be listed.
- Actual Outcome

  The book table that consists of three columns: Book Name, Due Date, and Renewals shows. The books that the student borrowed from the Tritonia library are listed.

7.14 Renewing Books in Tritonia

- Test Step

  In the Tritonia Renewal page, select one or more books and click the “Renew” button.
- Expected Result

  The books selected should be renewed successfully, which can be noticed by the increment of the renewals.
- Actual Outcome

  The books selected are renewed successfully, which can be noticed by the increment of the renewals.

7.15 Checking Profile

- Test Step

  After signing in, on the dashboard page, click “Profile”.

- Expected Result
  The profile page containing the student information should pop up as a modal dialog.
- Actual Outcome
  The profile page containing the student information pops up as a modal dialog.

7.16 Resetting Credentials

- Test Step
  After signing in, on the dashboard page, click “Setting”, then click “Reset Credentials”.
- Expected Result
  It should redirect to the register page.
- Actual Outcome
  It redirects to the register page.

7.17 Logging out

- Test Step
  After signing in, on the dashboard page, click “Setting”, then click “Logout”.
- Expected Result
  The session should be over, and it should redirect to the home page.
- Actual Outcome
  The session is over, and it redirects to the home page.

7.18 Generating the Course Calendar Database

- Test Step
  Run the Python script “generator.py”.
- Expected Result
  Three files “business.ics”, “technology.ics”, and “courses.json” should be generated in the directory “./vamk/data/calendar_db”.
- Actual Outcome

Three files “business.ics”, “technology.ics”, and “courses.json” are generated in the directory “./vamk/data/calendar_db”.

7.19 Checking GPA Changes Automatically

- Test Step
  Run the Python script “auto_winha.py”.
- Expected Result
  The subscribed students should receive the mail when GPA changes.
- Actual Outcome
  The subscribed students receive the mail when GPA changes.

7.20 Renewing Books Automatically

- Test Step
  Run the Python script “auto_tritonia.py”.
- Expected Result
  The subscribed students should receive the mail when books get renewed.
- Actual Outcome
  The subscribed students receive the mail when books get renewed.

8 CONCLUSIONS

With the aim to facilitate the use of Winha for students in VAMK, the project was well implemented, evolving from a CLI (Command Line Interface) script at the very beginning to a modern web application with not only multi-functionalities but also decent user experience and visual effect. The initial goals have been successfully achieved.

With this application, students in VAMK can easily and expediently calculate the GPA, get the grade distribution, get the complete course lists, generate the course calendars, check the profile, renew books, and get automatic services like GPA change notifications and book renewals.

The web application is easy to use but was not easy to implement. One of the most challenging parts was generating the course calendar database and finding a solution for displaying the calendar on the web page. The FullCalendar was utilized to do the job for embedding an event calendar to the web page. But the FullCalendar cannot resolve the iCal format of the calendar; the calendar source got from the school server should be converted to the JSON format that the FullCalendar supports. Another hard part was the UI design, for the better user experience, it took a long time to debug.

Although the initial goals have been achieved generally, there are still some improvements to be done in the future works. Firstly, implementing the FullCalendar events to export as iCal format that can be imported by the Google calendar. Secondly, implementing the front-end as a SPA (Single Page Application) with React.js or Angular 2. Last but not least, improving the security to protect from malicious actions.

REFERENCES

/1/ Python (programming language). Accessed 20.05.2016.

https://www.wikiwand.com/en/Python_(programming_language)

/2/ Python2orPython3. Accessed 20.05.2016.

https://wiki.python.org/moin/Python2orPython3

/3/ pip (package manager). Accessed 20.05.2016.

https://www.wikiwand.com/en/Pip_(package_manager)

/4/ Virtualenv Introduction. Accessed 20.05.2016.

https://virtualenv.pypa.io/en/latest/

/5/ Virtualenv User Guide. Accessed 20.05.2016.

https://virtualenv.pypa.io/en/latest/userguide/

/6/ Flask (web framework). Accessed 20.05.2016.

https://www.wikiwand.com/en/Flask_(web_framework)

/7/ Flask Quickstart Debug Mode. Accessed 20.05.2016.

http://flask.pocoo.org/docs/0.10/quickstart/#debug-mode

/8/ Flask Quickstart Routing. Accessed 20.05.2016.

http://flask.pocoo.org/docs/0.10/quickstart/#routing

/9/ Jinja (template engine). Accessed 20.05.2016.

https://www.wikiwand.com/en/Jinja_(template_engine)

/10/ SQLAlchemy. Accessed 20.05.2016.

https://www.wikiwand.com/en/SQLAlchemy

/11/ Object-relational mapping. Accessed 20.05.2016.

https://www.wikiwand.com/en/Object-relational_mapping

/12/ ORM picture. Accessed 20.05.2016.

http://www.entityframeworktutorial.net/Images/ORM.png

/13/ Encryption Accessed. 20.05.2016.

https://www.wikiwand.com/en/Encryption

/14/ Advanced Encryption Standard. Accessed 20.05.2016.

https://www.wikiwand.com/en/Advanced_Encryption_Standard

/15/ Python Cryptography Toolkit (pycrypto). Accessed 20.05.2016.

https://pypi.python.org/pypi/pycrypto

/16/ Requests: HTTP for Humans. Accessed 20.05.2016.

http://docs.python-requests.org/en/master/

/17/ Beautiful Soup Documentation. Accessed 20.05.2016.

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

/18/ OAuth. Accessed 20.05.2016.

https://www.wikiwand.com/en/OAuth

/19/ Why OAuth itself is not an authentication framework? Accessed 20.05.2016.

https://www.javacodegeeks.com/2013/02/why-oauth-it-self-is-not-an-authentication-framework.html

/20/ Authomatic Official Website. Accessed 20.05.2016.

http://peterhudec.github.io/authomatic/

/21/ Bower Creating Packages Register. Accessed 20.05.2016.

http://bower.io/docs/creating-packages/#register

/22/ Bootstrap Official Website. Accessed 20.05.2016.

http://getbootstrap.com/

/23/ jQuery Official Website. Accessed 20.05.2016.

https://jquery.com/

/24/ Gunicorn Official Website. Accessed 20.05.2016.

http://gunicorn.org/

/25/ Standalone WSGI Containers. Accessed 20.05.2016.

http://flask.pocoo.org/docs/0.10/deploying/wsgi-standalone/

/26/ Supervisor: A Process Control System. Accessed 20.05.2016.

http://Supervisord.org/index.html

/28/ CronHowto Introduction. Accessed 20.05.2016.

https://help.ubuntu.com/community/CronHowto

/29/ Cron. Accessed 20.05.2016.

https://www.wikiwand.com/en/Cron