|  |  |
| --- | --- |
| C:\Users\Емил Йорданов\Downloads\logo.png | **ВИСШЕ ВОЕННОМОРСКО УЧИЛИЩЕ „Н. Й. ВАПЦАРОВ“**  ***9002 Варна, ул. „В. Друмев“ 73, тел.052/632-015, факс 052/303-163*** |
| ***“FILII MARIS SUMUS”***  **ФАКУЛТЕТ „ИНЖЕНЕРЕН“ – КАТЕДРА „ИНФОРМАЦИОННИ ТЕХНОЛОГИИ“** |
|  |  |

****

**ДИПЛОМНА РАБОТА**

**за придобиване на ОКС „МАГИСТЪР”**

**теМА: Разработка и анализ на   
изпълнима програма за   
автоматизация на месечни отчети**

**Научен ръководител:**

Подполковник   
Драгомир Драгнев

**Дипломант:** Николай

Живков Николов

**Специалност:** Киберсигурност

**фак. №:** 626-201-13

**гр. Варна**

**2022 г.**

Съдържание

[1. Увод 3](#_Toc95929865)

[2. Изложение 4](#_Toc95929866)

[2.1. Използвани технологии 4](#_Toc95929867)

[2.1.1. Visual Studio Code 4](#_Toc95929868)

[2.1.2. GitHub 5](#_Toc95929869)

[2.1.3. Python 6](#_Toc95929870)

[2.1.3.1. Pip installer 6](#_Toc95929871)

[2.1.3.2. pyinstaller 7](#_Toc95929872)

[2.1.3.3. tkinter 7](#_Toc95929873)

[2.1.3.4. requests 8](#_Toc95929874)

[2.1.3.5. re 8](#_Toc95929875)

[2.1.3.6. configparser 8](#_Toc95929876)

[2.1.3.7. multiple dispatch 9](#_Toc95929877)

[2.1.3.8. datetime 9](#_Toc95929878)

[2.1.3.9. TODO: logging 9](#_Toc95929879)

[2.1.3.10. openpyxl 9](#_Toc95929880)

[2.1.3.11. python-docx 10](#_Toc95929881)

[2.2. Реализация 10](#_Toc95929882)

[2.2.1. Логическа реализация 11](#_Toc95929883)

[Техническа имплементация 15](#_Toc95929884)

[Графичен Интерфейс 16](#_Toc95929885)

[Прихващане на заявка 17](#_Toc95929886)

[Експортиране 19](#_Toc95929887)

[Конфигурация и техническа информация. 20](#_Toc95929888)

[Хеширане на файлове 20](#_Toc95929889)

[3. Заключение 21](#_Toc95929890)

[4. Библиография 22](#_Toc95929891)

# Увод

В този проект ще бъде разгледано създаването на изпълнима програма и изследването и за възможни уязвимостти. Програмата ще бъде създанена посредством езикът за програмиране Python, а анализът ще бъде чисто теоритичен.

Проектът представлява изготвяне на програма, с която лесно, бързо и практично да се изготвят месечни времеви отчети, относно преподадените лекции и упражнения, във формата на таблица на Microsoft Excel. Приложение, с чиято помощ се автоматизира частично или пълно създаването на месечни отчети на лекторите от ВВМУ „Никола Йонков Вапцаров“. По този начин може да бъде спестено значително време в изготвяне на документи, които зависят от вече съществуващ софтуер. Цялата информация, която се използва е публично достъпна, не изисква автентификация и не подлежи на засекретяване. Спрямо това се извежда заключението, че напълно удачно документирането и да бъде автоматизирано. Самото написване на програмата ще се случи с език за програмиране Python, използвайки съществуващи основни и допълнителни модулни библиотеки.

Тъй като написването на програма не е пряко свързано с дисциплината „киберсигурност“, дипломната работа ще бъде допълнена от теоритичен анализ на уязвимостите на такава. Той ще служи за допълване целите на методиката на разработка, както и затвърждаване на знанията придобите по време на обучение. Анализът ще бъде с по-тясна обвързаност към възможните уязвимости при програма създадена на Python, като значението се явява в съществуването на различни способи за атака спрямо различните софтуерни имплементации.

# Изложение

В изложението ще бъдат представени използваните технологии за разработка, референции към всички използвани източници, процеса на самата работа и резултата, както и анализа на уязвимостите.

По време на практическата част на разработката не е имало литературен озбор. Вместо това проектът е разработван на части, за нуждите на които се оказва достатъчно да се посъветва документацията на софтуера. При нужда от технически обяснения, това ще бъде направено на момента в съответният параграф.

TODO: Кратко описание на анализът

# Използвани технологии

Следва списъкът на всички използвани технологии в разработката на този проект, основните такива биват разпределени в главни отдели, а допълнителните свързани с тях в подотдели.

## Visual Studio Code

**Visual studio code** (Microsoft) е платформата, която се използва за написването и компилацията на самият код. Без значение на какво е написана една програма, тя трябва да има достъп до интерпретаторите и компилаторите на един език, за да може компютъра да разбере как да манипулира написаният код, за да създаде функционалност. Това са неговите указания как да разчита написаното от нас. Теоретично погледнато, кодът може да бъде написан и в един елементарен **notepad** и после компилиран отделно, но това би било непрактично и неудобно. Платформата позволява събирането на всички тези елементи на едно място за удобно използване. Освен това включва:

* **Intelisense** – възможност за интелигентно автоматично довършване на кода. Чрез сканиране на вече съществуващите библиотеки може да предложи всички възможни обекти и функционалности принадлежащи на един обект, заедно с документация за използването им, ако такава съществува. Подобно на стандартният табулатор в една конзола, с разликата, че предлага повече от едно решение, той улеснява работата на програмиста като спестява време от правописни грешки и пълно изписване, както и предлага нужната информация за използването на един обект на място, вместо тя да бъде търсена в първоначалните източници. **Intelisense** е част от всяка подобна платформа в днешно време и е неразделна част от набора инструменти на един програмист.
* **Портативност** – самата програма е портативна версия на основната такава **Visual** **Studio**. Превърната е в не-натоварваща модулна платформа, която може да бъде използвана за изключително голям набор от езици спрямо желанието и нуждите на използващите я. Не изисква инсталация, а всички файлове се съдържат в папката при изтегляне.
* **Extensions** – Програмата включва „допълнения“, които и позволяват да върши стандартната си работа. В този случай, **Python** също играе роля на такъв екстеншън. Той е просто един от многото начин да се използва **Visual Studio Code**. Той от своя страна, съдържа пакетни мениджъри, чрез които да се инсталират останалите модули, които са създадени или пригодени за работа с питон. Тези екстеншъни могат да бъдат инсталирани и деинсталирани в рамките на минути или секунди с няколко клика, което позволява и бързото рефакториране на програмата за нови цели, въпреки че това рядко се налага.

## GitHub

**GitHub** (GitHub, Inc) е разширение, с графичен интерфейс и по-големи възможности, на системата за контролиране на версиите **git** (Software Freedom Foundation). Подобна система поддържа списък от всички налични версии на една програма, т.е. в нея се съхраняват сегашното и състояние, както и всичките и минали такива до началото на нейното съществуване. Тъй като доста програми са не-малки, тази задача би била изключително натоварваща върху операционните системи и паметта на сървърите, ако се помнят всички файлове във всяка нова версия на една програма. Затова git пази „снимка“ на състоянието на файловете използвайки криптиращият алгоритъм SHA1 (Eastlake, и др., 2011), за да придобие символично състояние на всеки един файл.

Освен това **git** разрешава проблема с екипната работа върху еднакви файлове. Той позволява разработката на различни части на програмата в отделни „клони“. По този начин всеки човек от екипа може да работи самостоятелно върху файлове от програмата без да се изисква постоянен хардуерен трансфер на данни, и с минимална опасност от препокриване. Единственият проблем, е че той трудно поддържа файлове с големи размери. Съществуват допълнения, които разрешават това като **git lfs**, но тъй като за проекта това няма да е нужно, той няма да бъде разглеждан.

Последното предимство на използването на тази система е, че тя може да бъде закачена към сървъра на **GitHub**, вместо да бъде съхраняван на локална машина. Това от една страна елиминира риска от загуба на информация по време на работа в случай на авария,а от друга позволява лесно теглене на проекти, което значи, че всеки преподавател може да достъпи и използва проекта, правейки разпространението му лесно и бързо.

## Python

**Python** (Python Software Foundation) или питон е език за програмиране на високо ниво, без специализирано предназначение. Създаден е през 80-те години и в момента е един от най-известните езици за програмиране. Създателите му целят той да е приятен за използване, което е подчертано от името му, което идва от известната британска комедийна трупа Monty Python (Python (Monty) Pictures Limited). Отличава се с леснота на използване и модулност. Предназначен е да се използва и допълва спрямо нуждите на програмиста. Също така поддържа множество стилове на програмиране. Особено е, че използва стандарт за кодиране **PEP-8** (Rossum, et al., 2001). Това е стандарт, който използва табулатори, които служат както за четимост, така и за структура. Чрез използване им питон разбира къде има логическа или структурна промяна. Тяхната липса не позволява на програмата да се компилира правилно. Тъй като той самият е без специалзиирано предназначение, той съдържа набор от основни библиотеки (пр. Datetime за манипулация на времеви данни), както и огромно количество допълнителни такива, създадени както от професионални, така и от програмисти ентусиасти, с цел допълване на липсваща функционалност или просто за лично използване, след което споделят работата си с останалите. По този начин днес **Python** е от най-широко изпозлваните и поддържани езици. Модулите, които ще бъдат използвани в този проект, без специфичен ред, са:

### 2.1.3.1. Pip installer

**Pip installer** (The pip developers) е част от основните модули на **Python** и се инсталира, заедно с питон. Той е главният инструмент, чрез който биват добавяни всички останали модули от библиотеките на питон. Той осигурява достъп до тях, както и лесен мениджмънт. Това се случва чрез няколко кратки команди:



Първата команда е главният начин за инсталиране на пакети чрез инсталатора. Втората е подобна, но съдържа допълнителен аргумент *--upgrade*, който ръчно проверява за наличие на нови версии и, ако такива съществуват, ги инсталира. Ако той не бъде използван ще бъде проверено наличието на пакета като цяло. В случай, че се опитваме да използваме нова или променена функционалност, първата команда няма да бъде достатъчна, след което ще се чудим защо програмата ни не работи. Последната команда премахва съществуващ модул, както се деинсталират допълнения от **Visual Studio Code**. Въпреки че тях рядко бива използвана, има допълнителни случаи, в които тя се оказва доста полезна. Един подобен ще бъде описан в този проект. Освен това има и различни аргументи при използването на базовите, какъвто е *--upgrade*, те позволяват разнообразни неща като задаване на конфигурационен файл при инсталация, избиране на специфична версия и други. Съществуват и още много други команди спрямо нуждите на използващият го, но те няма да са нужни за разработката на проекта, затова няма да бъдат разглеждани.

### 2.1.3.2. pyinstaller

**Pyinstaller** (PyInstaller Development Team, 2021), съкратено от python installer, е модул от допълнителните библиотеки на python, който позволява компилирането на скриптове в изпълними програми, т.е. превръща файловете с *.python* разширения в .*exe*. Има и други такива модули, но този се отличава с поддръжка на новите версии на питон, създаването на по-малки на големина файлове, и идентично действие независимо от платформата, върху която файловете са създадени или изпълнени. Всеки файл може да бъде компилиран еднакво както на **Windows**, така и на **Linux**, след което да бъде изпратен на друга система и да работи по очакваният начин. Също така е способен да събере цялата описана функционалност на една програма в единичен файл, без нуждата от допълнителни директории и разнообразни конфигурационни файлове.

За изискванията на този проект, **pyinstaller** ще бъде използван за превръщане на python скриптовете в изпълним файл с графичен интерфейс, който е по-интуитивен за използване от командният ред.

### 2.1.3.3. tkinter

**Tkinter** (Python Software Foundation) или tk interface е стандартно допълнение за функционалност на графичен интерфейс към езикът за програмиране **Tcl** и собственото му разширение за интерфейс – **Tk**. **Tkinter** е **Python** обвивка на съответната библиотека. За да може тя да съответства на езика възможно най-добре, тя включва и голямо количество самостоятелна логика, вместо да е просто тънка обвивка около съществуващата библиотека.

Библиотеката служи за създаване на графични интерфейси за програми. Имайки доста възможности, с нея може да бъде създаден както елементарен интерфейс като калкулатор, така и сложни интерфейси с множество функционалности, различни екрани и логически разделени пътища на изпълнение, както примерно би бил интерфейсът на един инсталатор. Едно от основните преимущества на библиотеката е, че дава възможност за използване на вече съществуващи елементи, наречени widgets. Това са често използвани елементи в дизайна, като надписи, полета за вход на данни, бутони, изображения и други. Това помага за лесното и бързо създаване на прототип или дори на пълна програма за сравнително по-малко време, отколкото би било нужно при написването на всички тези елементи от нулата. Тъй като не би могло да се разчита единствено на тези елементи, библиотеката позволява специфично наместване на съществуващите елементи по множество начини, включително тяхното точно разположение, размер, промяната на такъв при промяна на размера на екрана и други. Всички тези елементи превръщат библиотеката в изключително полезен инструмент при изграждането на проекта.

### 2.1.3.4. requests

Библиотеката за интернет заявки – **requests** (Reitz) е точно това, което звучи. Чрез нея можем да осъществяваме всички съществуващи стандартни заявки с протокол – *HTTP/HTTP*S (пр. *GET*/*POST*) по програмируем начин. Тя също е допълнителен модул за питон, който може да бъде инсталиран чрез pip installer. Тя ще бъде използвана за прихващане на съществуващата информация в публичният домейн на университета относно седмичните програми на учителите. Резултатът от прихващането на заявката е чист *HTML* код, което ще рече, че тя трябва да бъде детайлно обработена и интересуващата ни информация да бъде изчистена и структурирана, преди да е възможно тя да бъде пренасочена и съхранена към файл на **Excel**.

### 2.1.3.5. re

**Re** (Python Software Foundation), съкратено от regex, е основна библиотека в модулите на питон, която служи за извличане и обработка на стандартни изрази от сложни данни. Името на библиотеката буквално идва от съкращението и обединението на „стандартен израз“ (**r**egular **e**xpression). Тя съществува в много близки, почти еднакви, състояния в почти всички езици за програмиране, тъй като е широко използвана за намиране на информация, валидация, и други операции в много програми. Въпреки, че за по-сложни изрази, нивото на четимост рязко пада, скоростта на regex го кара да бъде често избиран (пр. валидация на телефон, пощенски код, имейл). След прихващането на заявка, тази библиотека ще бива използвана за изграждането на регулярни изрази, спрямо които да бъде извлечена необходимата ни информация.

### 2.1.3.6. configparser

**Config** **parser** (Python Software Foundation) е основен модул, който позволява по-детайлна манипулация на файлове от стандартният модул **os**, със специализация към конфигурационни файлове. Той ще бъде използван за съхраняване на настройките на програмата при всеки следващ експорт. Тоест, за улеснение на използването, той ще запаметява и зарежда последно използваните настройки за експорт в програмата. Това ще доведе до създаването на всеки следващ експорт да бъде сведено до отваряне на програмата, смяна на месеца и натискане на бутон „експорт“, което е доста по-удобно от ръчното търсене и въвеждане на цялата информация.

### 2.1.3.7. multiple dispatch

Multiple dispatch е допълнителен модул, който позволява използването на питон по малко по-различен начин. Той помага за променяне на използваната методология на програмиране, и позволява съществуването на функции с еднакви имена и различни параметри, нещо което питон не позволява по подразбиране. Обяснението на това е следното: при компилация питон намира мястото, където е декларирана една функция и тя вече има такава стойност при повикването и. Ако вземем за пример умножение с две числа, то ще имаме функция, която взима **А** и **B** и връща резултат. Но ако искаме да имаме същата функция, само с един параметър **А**, която да връща умножението на едно число само по себе си, то тогава ще имаме проблем. Питон ще се компилира правилно, но единствената функция за умножение ще бъде тази, която е написана последно в кода. При намиране на първата функция, питон ще и зададе съответната стойност, но при намирането на втората такава със същото име, тя просто ще бъде презаписана. Това не е грешка в кода на питон, а просто ограничение, тъй като той не е създаден, за да спазва определени методологии, а съществуването на множество функции с еднакви имена и различни параметри е функционалност на Обектно-Ориентираното Програмиране, така наречният overloading (W3Schools). Тъй като, това е нещо, което е полезно за запазването на чистотата, четимостта и късата дължина на кода ще използваме този модул, който позволява питон да третира двете функции по отделно.

### 2.1.3.8. datetime

Datetime която е от по-голямо значение е основният времеви модул на python (Python Software Foundation). Тъй като заявките на университета работят с номер на седмица, а времевите отчети са месечни, е необходима такава програма, която надеждно да извършва сложни времеви манипулации. В случая – да изчислява кои седмици участват в даден месец, след което да извлича информацията за всяка една от тях, без да излиза от рамките на месеца, в случаите, в които една седмица бива споделена от два месеца. Съответната библиотека позволява подобни изчисления да бъдат извършени сравнително лесно, тъй като повечето от тях вече са налични като вградени функции.

### 2.1.3.9. logging

Python съдържа като всеки друг език за програмиране, библиотека за логване на информация. Чрез нея се проследява всяка част от изпълнението на програмата, както в конзолата по време на разработка, така и в реалното и приложение. Основната полза на логъра е, че записва всичко случващос се, като така значително по-лесно може да се разбера причината за един проблем, когато той се случи. Записва се т.нар. stack trace, който съдържа подробна информация за проблема и мястото, от което произлиза. Чрез него може да бъде стеснен кръга на възможностите или директно да бъде намерена причината за него. Библиотеката за логване в питон се нарича logging и е част от основните модули.

### 2.1.3.10. openpyxl

Името openpyxl(Gazoni, et al.) идва от комбинацията open source python excel, т.е. свободно разпространена библиотека за работа с екселски файлове използвайки питон. Чрез нея се осъществява манипулация на файлове от съответният тип на множество нива – файл, таблица, ред, колона или клетка. Използвайки тази библиотека е възможно цялата информация от заявките да бъде запаметена и форматирана в екселски файл.

### 2.1.3.11. python-docx

**Python-docx** (Canny, 2013)е допълнителен модул, който се занимава със създаването и манипулацията на данни за файлове с разширение .docx. Тъй като програмата има възможност за допълнителен експорт на седмични данни за лесна справка, този модул ще бъде използван за експорт в уърд. Самото име на разширението идва от комбинацята “**x**ml **doc**ument – **doc/x**”, тъй като самият документ, когато бъде прочетен с елементарен четец като notepad++ се вижда, че е структуриран във вид на XML файл. Това позволява лесната му манипулация, считайки че всяка част от него – заглавие, параграф, таблица, както и всичките им атрибути – шрифт, фонов цвят, рамки и други, са програмируеми елементи.

2.1.3.12. Unit testing

# Реализация

Реализацията на проекта се състои от няколко фази. Първоначалната му версия е разработвана за дисциплината „курсов проект“ през втори семестър, след което е надградена за целите и нуждите на дипломната работа. Съответната версия имаше достатъчно сложност дотолкова, че да създаде елементарен експорт на данните за даден месец накуп, без обработка или разделение.

Тези елементи са преразгледани и довършени във втората версия на проекта, като данните вече са структурирани и разделени, има възможност за взимане на специфична част от данните по име (пр: взима се lecturer – името на водещият лектор, вместо елемент на трето място от масив – data[2]). Освен това са добавени две допълнителни функционалности – създаването на форматирана програма от седмица **А** до седмица **B**, съдържаща информацията за всички лекции и упражнения през това време. Съответната програма има избор за вид на финален файл за съхранение, и е удобна за пренос и принтиране. Тя е разработена по съвет на Подполк. Драгнев, считайки нейната финална цел, както и разработката на цялостна нова система във ВВМУ. Вторият елемент е изглед на всички седмици през дадената година. Тъй като изкарването на програма от седмица до седмица става с използването на индекси, те са изкарани в допълнителен екран за лесна справка.

Третата фаза на проекта е сравнително малка практически, но логически може да се разглежда като отделна – това е превръщането на проекта в изпълнима програма и тестването за желан резултат, както и написването на кратък наръчник за използването и.

TODO: да напиша наръчника, да кача архив в гитхъб, и да тествам на други системи.

Четвъртата фаза и последната е създаването на подробен анализ за уязвимости. Той ще едновременно за обогатяване и затвърждаване на знанията, както и за целта спокойно разпространяване и използване на програмата.

## 2.2.1. Логическа реализация

### 2.2.1.1. Налични и нужни данни

Логическата реализация ще разгледа проекта без технически подробности, а в качеството си на това какви са задачите на програмата, кои са възможните спънки по пътя, и как да бъдат разрешени.

За начало, нека се разгледа информацията, от която зависи цяата програма. Това е публично достъпна информация, която всеки студент и лектор използват за ориентир. Тя се намира на [https://nvna.eu/wp/](https://nvna.eu/wp/%20) и създава заявки към сървъра на университета с три части – номер, седмица, и вид заявка, с избор от:

* класно отделение
* преподавател
* стая

След въвеждане се генерира заявка, която връща таблица с всички дейности за съответният номер и вид заявка през зададената седмица. В таблицата съществуват общо седем различни елемента. Три, които винаги се срещат:

* номер на час (от 1-13, всеки разделен на 45мин.)
* времетраене (пр: от 10:00 – 11:30)
* име на занятието

И други четири в зависимост от видът на заявката:

* преподавател
* пореден номер на лекцията
* група/класно отделение
* стая

Като правилото е, че елементът не се появява, тогава когато отговоря на видът заявка, а „пореден номер на лекцията“ се появява само при заявка от вид *преподавател*.

Сега нека бъде разгледано каква информация е нужна, за автоматизацията на месечни отчети и изготвянето на програми. За нуждите на този проект бе осигурен шаблон на стандартен месечен отчет във формат, какъвто подават лекторите на университета. Всички данни в него са примерни. Информацията, която се попълва в тези отчети е:

* номер на дисциплината
* номер на групата
* брой занятия на съответната група за месеца
* номер и разделение по групи
* изпити
* и други

След като се знае каква информация е налична и каква е необходима, лесно се вижда какво може и не може да бъде направено с нея. Възможно е да бъде създадена лесна справка, но няма как, само със съществуващата информация, напълно да се автоматизира създаването на месечни отчети. За постигането на съответната цел има нужда от допълнителни документи, който показват отношението между наличните елементи към нужните такива. Допълнително, по време на работата по този проект бе разбрано, че в университета се създава нова вътрешна система, която, измежду други неща, ще може да служи за съответното пълно автоматизиране. Спрямо това, практическата част на проектът няма да набляга толкова на месечният отчет, колкото до създаването на полезен инструмент за лесно изкарване на информацията за седмични програми. Това би било полезно не само на лекторите, а и на студентите, на които често им се налага да сверяват програмата си в сайта.

### 2.2.1.2. Зависимости

Програмата е зависима зависима от състоянието на сайта на университета, както и с интернет връзката. Това значи, че всяка промяна в структурата на заявките, или в състоянието на сървъра съответно оказва критично влияние върху функционалността на програмата. При промяна, ако тя не е пълна смяна на софтуера на университета, винаги е възможно кода да бъде променен, така че да пасва на новите изисквания. Считайки, че това са рядко и трудно случващи се събития, рискът от подобен проблем е нисък. Дори и да бяха чести събития, не е нещо, което зависи от страна на разработчика на програмата, така че няма какво да бъде предприето за тях.

### 2.2.1.3. Валидност на данните

Заявките винаги връщат резултат, било то празен или не. Без значение дали съществуват реални данни за съответното класно отделение, преподавател или зала, файл винаги ще бъде създаден. Т.е. възможно е да съществува празен отчет или програма, така че след създаването им е добре да има ръчна проверка от използващият програмата, дотолкова че да се потвърди наличието на информация във файла. В този случай е невъзможно да бъде направена разлика между липсващи данни за седмица и грешно въведени параметри без съответната проверка.

За да се разреши този проблем е възможно добавянето на валидация на данните, във вид на ограничение на всяко поле за дължина, вид данни и формат. Това допълнително усложнява програмата, и също така изисква повече промени при предишно споменятият проблем с промяна на структурата на заявката. Финалното решение ще бъде взето спрямо удобството и сложността, която валидацията добавя към програмата.

### 2.2.1.4. Конфигурация

Програмата ще се нуждае от конфигурационен файл за максимално улеснение на работата на потребителя. Самият файл няма да служи за настройване на системни елементи като големи на екрана или резолюция, а за записване на последно използваната информация. Ако бъде представен един реален случай на използване, то това може да бъде преподавател, който месец за месец, или седмица за седмица използва програмата за създаване на отчет или програма. Единственото нещо, което той ще промени при всяко следващо използване е номера на съответният месец или седмица. Водено от този ред на мисли, има логика да бъде създаден конфигурационен файл, който да записва всичко друго (номер на преподавател, вид заявка, папка за съхранение и др.), за да спести време и да улесни работата на крайният потребител.

### 2.2.1.5. Логър за информация

Както вече бе споменато, записването на информацията по време на процеса на работа е изключително полезен. Чрез него може да се проследи дали изпълнението на програмата протича правилно, и ако не, какъв е проблема. Стандартното действие на библиотеката е да записва информацията директно в конзолата, като нейният формат и съдържание могат да бъдат всичко. Допълнително съществуват и различни нива на логване – DEBUG, INFO, WARN, ERROR. Всяко от тях се ползва спрямо видът информация, която искаме да запишем и са сравнително само-обясними, като в самият изход на информацията (конзола, файл или др.) се филтрират нивата спрямо желаната информация. Честа практика е да се записва всичко от INFO нагоре (WARN & ERROR), считайки че всяко ниво има и съответстващият му индекс. Библиотеката разполага с много възможности за допълнимост, но в случаят е достатъчно само да пренасочим информацията от конзолата към текстови файл, за да може да бъде лесно споделяна от потребителите в случай на проблем.

### 2.2.1.6. Разпространение

TODO:

### 2.2.1.7. Блок схема на действие

В [логическата схема](#_5.1._Логическа_блок) към приложението може да бъде проследено действието на програмата в основни стъпки.

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

## 2.2.2. Техническа имплементация

В тази част ще бъдат разгледани техническите детайли на имплементацията на програмата. Референции към кода на програмата могат да бъдат намерени в приложението. Възможно е логическото обяснение на програмата да е в обратен ред на написаният код, тъй като в **Python** не е възможно да се използва променлива или функция, преди да бъде създадена, т.е. всички детайли на използването трябва да са обявени и инициализирани преди да бъдат извикани от друго място.

## Графичен Интерфейс

Програмата започва логически от графичният интерфейс [(1)](#_Графичен_интерфейс), използвайки библиотеката **Tkinter**, създаваме начален екран и съвместими променливи за всеки желан параметър. Тъй като репортите са месечни, а не седмични, интерфейса показва поле за избиране на месец, вместо седмица. Съществуват съответно основните три параметри, заедно с допълнителен такъв за избиране на папка, в която да бъде създаден крайният експорт.

След това се използва графичната библиотека за създаване на всички нужни елементи – полета за вход на данни и техните наименования, бутони и други, и тяхното разположение и форматиране. Библиотеката съдържа няколко вида инструменти за геометрично подреждане на елементи. Основният е grid geometry manager. Чрез него елементите могат много лесно да бъдат разпределени в квадратна мрежа с избран брой редове и колони, като последният ред е извикването на *root\_frame.mainloop()* [(1, ред 107)](#_Графичен_интерфейс), което казва на библиотеката да поддържа елементите на екрана (т.е. няма да се загаси веднага, след като тръгне). Програмата цели да е разположена по удобен начин като също така по този начин дава възможност за лесно разширение при бъдеща нужна от промяна (пр. избиране на наименование на файл). Разпределението и се води от следният шаблон:



Graphical user interface, application

Description automatically generatedИ в резултат изглежда така:

Спрямо споменатите проблеми с валидацията на параметрите на заявката, избирането на вид на заявка и месец представляват избиране на стойности, заредени от статични списъци с променливи. Списъците съдържат само стойности, които са сигурни да работят, освен ако няма промяна в начина на действие на уебсайта. Потребителят няма право избор на друг елемент и, тъй като точно това са стойностите с най-голяма вероятност за грешка в сравнение с номера, рязко пада възможността за изпращане на заявки съдържащи невалидни елементи. Със същата цел е добавен бутон към избирането на съдържаща папка за финалният експорт. При директно избиране на съществуваща папка няма възможност за правописна грешка. Въпреки това може да има ситуация, при която потребителят копира целият път от файлов браузър, след което го копира директно в програмата. За такъв случай, полето не е направено с единствен достъп от бутона.

От там нататък е счетено за излишно добавянето на валидация на полетата за номер или папка. Отговорността за верни данни се пренася върху потребителят. Считайки, че информацията съдържана в заявките е публична и не крие нищо тайно, а главният и път е твърдо зададен [(2)](#_Прихващане_на_заявки) в недостъпна част на програмата и единственото, което може да бъде променено са параметрите и, се счита, че опасност от кибератака не съществува, или при проба за такава е невъзможно да бъде извлечена полезна информация. След проверка на информацията и натискане на бутон експорт, закачената към него функция се изпълнява, стойностите в конфигурационният файл се презаписват и хода на програмата се предава към прихващача [(2)](#_Прихващане_на_заявки).

## Прихващане на заявка

Прихващането на заявките става от следващата част на програмата [(2)](#_Прихващане_на_заявки), която приема зададените параметри от графичният интерфейс и формулира серия от заявки спрямо тях. Първото нещо, което се случва е да бъде взето името на месеца и да бъде превърнато в индекс с помощта на времевата библиотека. Това се прави, за да бъде използван след това, за намиране на първата седмица от този месец, тъй като функциите на библиотеката работят с номер, а не с наименование на месец. След това по същият начин биват намерени и общият брой седмици в съществуващият месец. Създава се празна структура от данни, в която да се съдържа ежедневната информация и, започвайки от първата седмица, заявки се формулират по следната формула:

*Основният път към сайта + идентификационен номер + вид заявка + номер на седмица*

Или, ако искаме да направим заявка за класно отделение 626201 за първата седмица на Октомври (39), тя ще изглежда така:

*https://nvna.eu/wp/?****group****=626201&****queryType****=group&****Week****=39*

Октомври има 5 седмици, съответно ще бъдат създадени 5 различни заявки в подобна форма. Всяка заявка ще бъде създадена и изпълнена с помощта на модул **requests**. Отговорът от всяка е *HTML* кодът на цялата страница, т.е. това което иначе би видял потребителят в сайта е показано в оригиналният му вид преди браузърът да го превърне в смислен вид. След прихващане информацията се препраща към друга функция, която извлича нужната ни информация и я добавя към временна структура. След края на всяка заявка, временната структура бива добавена към месечната, след което автоматично се унищожава преди създаването на следващата заявка.

Извличането става чрез използването на модул **re**. Използват се няколко различни регулярни израза, самостоятелно или в комбинации за постигане на желаната цел. Най-кратко казано, изразите се състоят от очаквани точни части (пр. при търсене на програмата за понеделник се търси мястото в таблицата, където пише точно „Понеделник“), комбинирано с шаблонни части (пр. след името на всеки ден е неговата дата, ако първият понеделник на Октомври е **2021-10-04**, тогава ще се търси като **[0-9]{4}-[0-9]{2}-[0-9]{2}**, което ще търси поредица от четири цифри, после две, и после още две, с тире между всяка група). По подобен начин са формулирани и останалите изрази, но изглеждат по-сложно и е излишно да бъдат обяснявани детайлно. Съществуват три основни случая, които трябва да бъдат покрити от изразите:

* ден без часове
* академичен час от деня, които е начало на лекция/упражнение
* академичен час от деня, които е продължение на вече започнала лекция/упражнение

Причината случаите да са три, а не два (с/без лекция), е че кодът изглежда по-различен начин спрямо дали един ден е празен, или ако не дали всеки негов час е начало или продължение на лекция, като структурата на празните часове, които не са продължение на лекция съвпадат със стандартният код за начало на такава, но липсва информация в тях. Съществуват 4 различни шаблона изрази:

* Структурата на заглавният ред на таблицата, показващ деня и датата (пр. Понеделник, 2021-10-04).
* Структурата на ден, в които няма лекции.
* Структурата на ден, в които има лекции.
* Структурата на самостоятелен час.

Разделени са по този начин за лесно комбиниране, тъй като се препокриват. За всеки ден от седмицата се извършват следните действия - Тъй като е най-семпло, първо се изпълнява търсенето за заглавен ред, заедно с ден без лекция. Ако върне резултат, към временната структура се добавя списък съхраняващ деня и изречението „Няма занятия“. Ако не, програмата използва комбинация от заглавният ред и структурата на цял ден, в които има лекции. Тук не се интересуваме от самата информация, а дали такъв ден ще бъде намерен, тъй като това улеснява бъдещото извличане на информация да бъде фокусирано върху резултата от това търсене. То ще бъде кодът на един единствен ден, вместо да бъде правено търсене из цялата седмица, тъй като е излишно и може да има грешно намерени изрази. При резултат, програмата използва структурата на самостоятелният час и намира всички такива попадения. Към временната структура се добавят името на деня и информацията за всеки час разделен в четири части – номер на академичният час, в които е започнал, времевото продължение, дисциплина и преподавател. Така временната структура представлява списък от списъци – в случай, че за един ден не е имало лекции, списъка е с дължина два елемента, ако да е с дължина *n*, за която винаги е валидно условието *n-1=4\*x*, където *x* е броят часове. Това ни позволява по-нататък в програмата лесно да предскажем какво е нужно и не е нужно да бъде добавено в крайният файл спрямо големината на всеки списък. Последно, в случай, че програмата не върне отговор за нито един от изразите, нищо не бива добавено. Това се случва само в първата или последната седмица на даден месец, където има дни от други месеци, тъй като индексът на всеки месец участва в търсенето.

След като всички заявки бъдат направени успешно и информацията е събрана в списък, тя се препраща към следващата част на програмата и съхранението и в екселски файл.

## Експортиране

Експортирането е сравнително по-просто от останалите части на програмата. Извикването на функцията за създаване на нов файл става с три параметъра – списъка с информация, месеца, и папката за съхранение. Използвайки модул **openpyxl**, програмата създава нов файл, чието има е съчетание от статична фраза, датата на създаване и името на месеца. Файл, създаден на първи Декември, за месец Октомври би изглеждал така: *NvnaExport\_October\_2021-12-01.xlsx*. Всеки нов файл с подобни параметри би презаписал предишният. Към момента програмата не позволява избирането на име за файла, което значи, че репорти за различни класни отделения с препокриващ се месец направени в един и същи ден изискват ръчно преименуване, за да не бъдат затрити. Но това не е голяма промяна и при нужда може да бъде променено сравнително лесно.

При извикването на функцията, програмата създава името на крайният файл, и за всеки елемент от списъка, добавя информацията в него в последователни колони, като всеки следващ елемент е на нов ред. Ако всички действия до тук са били успешни, тук програмата връща съответният отговор към графичният интерфейс и се показва съответният диалог, потвърждаващ на потребителя, че репортът е създаден без проблеми, заедно с папката, където е създаден.

## Конфигурация и техническа информация.

Освен основната логика съществуват още две части на програмата – конфигурационната и информационната.

Конфигурацията [(4)](#_Конфигурация) се грижи за създаване и запазване на използваните параметри за по-лесно бъдещо използване. Тъй като е най-добре програмата да се разпространява с минимално количество файлове, тя не разчита, че потребителят има или знае как да конфигурира съответният файл. Затова при всеки експорт се изпълнява функция, която запаметява новите стойности на параметрите, но първо проверява дали файлът съществува. Ако не, се създава нов такъв с разширение *.cfg*, след което се добавя секция *[request\_parameters]*, където ще бъдат съхранявани всички желани стойности. След това продължава стандартното изпълнение на програмата.

Информационната част се грижи за събирането на техническа информация за случващото се в програмата по време на изпълнение. Важно е да се отбележи, че тя по никакъв начин не събира или анализира информация за потребителят, тя служи само и единствено за проследяване на държанието на програмата в случай на проблем. Основната логика за конфигурирането на логъра се съдържа в отделен файл [(5)](#_Логър), към които реферират всички останали. Съхранява се информация при стартирането на програмата, при изпълнение на експорт, както и самата информация от заявките, филтрирана и подредена за всеки ден. По този начин може лесно да се проследи точно къде е настъпил проблемът, при наличие на такъв.

За да може всичко да е четимо и на български език, всичко свързано с отваряне, затваряне или модификация на файлове е със статично зададен стандарт за кодиране *UTF-8*.

## Хеширане на файлове

Тъй като проекта ще се разпространява с изпълним файл и оригиналният код, като допълнение против зловредна промяна във файловете е добавен скрипт, които изчислява хешираната сума на всеки *.py* скрипт, както и на самият изпълним файл .*exe* по алгоритъм SHA-256 (Eastlake, et al., 2011). Изчислените суми за всеки файл се съхраняват в текстови файл и могат лесно да бъдат пресметнати с инструменти каквито се намират на всяка операционна система. Промени по кода съответно ще укажат влияние върху сумите. За най-голяма сигурност е добре изчисленията да се правят спрямо файлът в сървъра на **GitHub**, откъдето може да бъде изтеглен проекта.

# Заключение

В този доклад е представен проект за разработка на програма, която служи за автоматично създаване на месечни отчетни доклади. Целта на проекта е да спести време на учителите във Военноморско училище „Никола Йонков Вапцаров“, като изпълнява това вместо тях, спестявайки време. Отбелязани са всички използвани технологии и са референции към мястото, от където са взети, и където е описан начинът на тяхното използване. Целият използван софтуер по време на проекта е безплатен и свободно-разпространим за не-комерсиални цели. След описание на логическото поведение на проекта е показано реалното протичане на имплементацията. Показани са логическите проблеми идващи от разчитането на външен източник на информация – сайтът на университета, както и специфичните разминавания между стандартното изкарване на седмична програма в <https://nvna.eu/wp>, и начинът, по които това се прави в програмата. Рискът от невалидни данни е намален максимално спрямо допустимото от страна на разработка, останалото зависи от личното потвърждение на потребителя. Счетено е, че поради съществото на използваната информация, риск от кибератака не съществува, или изпълнението на такава е невъзможно да окаже смислено влияние върху потребителят или университета. Въпреки това има шанс от зловредно разпространение на проекта. Той винаги трябва да се взима от първоизточникът му, който е зададен в апендиксът на този доклад. Дори тогава, за по-голяма сигурност е добавен скрипт с резултат от хеширане на всички използвани файлове чрез алгоритъм *SHA-256.*

# Библиография

**Eastlake, Donald и Hansen, Tony. 2011.** *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF).* неизв. : IETF, 2011.

**Gazoni, Eric и Clark, Charlie.** openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files. *openpyxl.* [Онлайн] [Цитирано: 15 December 2021 r.] https://openpyxl.readthedocs.io/en/stable/.

**GitHub, Inc.** GitHub. *GitHub.* [Онлайн] [Цитирано: 13 December 2021 r.] https://github.com.

**Microsoft.** Visual Studio Code. *Visual Studio Code.* [Онлайн] [Цитирано: 13 December 2021 r.] code.visualstudio.com.

**PyInstaller Development Team. 2021.** PyInstaller. *PyInstaller.* [Онлайн] PyInstaller Development Team, 11 August 2021 r. [Цитирано: 15 December 2021 r.] https://www.pyinstaller.org/.

**Python (Monty) Pictures Limited.** Monthy Python's Official Website. *Monty Python.* [Онлайн] Python (Monty) Pictures Limited. [Цитирано: 13 December 2021 r.] http://www.montypython.com/.

**Python Software Foundation.** configparser — Configuration file parser. *Python Documentation.* [Онлайн] [Цитирано: 15 December 2021 r.] https://docs.python.org/3/library/configparser.html.

**—.** datetime — Basic date and time types¶. *Python Documentation.* [Онлайн] [Цитирано: 15 December 2021 r.] https://docs.python.org/3/library/datetime.html.

**—.** Python. *python.* [Онлайн] [Цитирано: 14 December 2021 r.] https://www.python.org/.

**—.** re — Regular expression operations. *Python Documentation.* [Онлайн] [Цитирано: 14 December 2021 r.] https://docs.python.org/3/library/re.html.

**—.** tkinter — Python interface to Tcl/Tk¶. *Python Documentation.* [Онлайн] [Цитирано: 15 December 2021 r.] https://docs.python.org/3/library/tkinter.html.

**Reitz, Kenneth.** Requests: HTTP for Humans™. *Requests.* [Онлайн] [Цитирано: 14 December 2021 r.] https://docs.python-requests.org/en/latest/.

**Rossum, Guido van, Warsaw, Barry и Coghlan, Nick. 2001.** Style Guide for Python Code. *Python.* [Онлайн] 05 July 2001 r. [Цитирано: 13 December 2021 r.] https://www.python.org/dev/peps/pep-0008/.

**Software Freedom Foundation.** git. *git.* [Онлайн] https://git-scm.com/.

**The pip developers.** pip documentation v21.3.1. *pip.* [Онлайн] [Цитирано: 13 December 2021 r.] https://pip.pypa.io/en/stable/.

# Приложение

## 5.1. Логическа блок схема



## 5.2. Техническа блок схема