|  |  |
| --- | --- |
| 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 на софтуер за генериране на справки и отчети**

**Научен ръководител:**

Подполковник   
Драгомир Драгнев

**Дипломант:** Николай

Живков Николов

**Специалност:** Киберсигурност

**фак. №:** 626-201-13

**гр. Варна**

**2022 г.**

Съдържание

[1. Увод 4](#_Toc96372085)

[2. Изложение 5](#_Toc96372086)

[2.1. Сравнение на възможни имплементации 5](#_Toc96372087)

[2.1. Използвани технологии 5](#_Toc96372088)

[2.1.1. Visual Studio Code 5](#_Toc96372089)

[2.1.2. GitHub 6](#_Toc96372090)

[2.1.3. Python 7](#_Toc96372091)

[2.1.3.1. Pip installer 7](#_Toc96372092)

[2.1.3.2. pyinstaller 8](#_Toc96372093)

[2.1.3.3. tkinter 8](#_Toc96372094)

[2.1.3.4. requests 9](#_Toc96372095)

[2.1.3.5. re 9](#_Toc96372096)

[2.1.3.6. configparser 10](#_Toc96372097)

[2.1.3.7. multiple dispatch 10](#_Toc96372098)

[2.1.3.8. datetime 10](#_Toc96372099)

[2.1.3.9. logging 11](#_Toc96372100)

[2.1.3.10. openpyxl 11](#_Toc96372101)

[2.1.3.11. python-docx 11](#_Toc96372102)

[2.2. Реализация 11](#_Toc96372103)

[2.2.1. Графичен интерфейс 12](#_Toc96372104)

[2.2.1.1. Месечни отчети 13](#_Toc96372105)

[2.2.1.2. Седмични програми & индекс на седмиците 13](#_Toc96372106)

[2.2.1.3. Технически подробности 13](#_Toc96372107)

[2.2.2. Прихващане на заявка 14](#_Toc96372108)

[2.2.2.1. За месечни отчети 14](#_Toc96372109)

[2.2.2.2. За седмични разписания 14](#_Toc96372110)

[2.2.2.3. Изпълнение на седмични заявки 15](#_Toc96372111)

[2.2.3. Филтриране на информация 15](#_Toc96372112)

[2.2.3.1. Основни шаблони и изпълнение 16](#_Toc96372113)

[2.2.3.2. Структура на данните и извличане 16](#_Toc96372114)

[2.2.3.3. Технически подробности 17](#_Toc96372115)

[2.2.4. Експортиране на данни за месец 18](#_Toc96372116)

[2.2.1. Логическа реализация 20](#_Toc96372117)

[2.2.1.1. Налични и нужни данни 21](#_Toc96372118)

[2.2.1.2. Зависимости 21](#_Toc96372119)

[2.2.1.3. Валидност на данните 21](#_Toc96372120)

[2.2.1.4. Конфигурация 21](#_Toc96372121)

[2.2.1.5. Логър за информация 21](#_Toc96372122)

[2.2.1.6. Разпространение 21](#_Toc96372123)

[2.2.1.7. Блок схема на действие 21](#_Toc96372124)

[Конфигурация и техническа информация. 20](#_Toc96372125)

[Хеширане на файлове 21](#_Toc96372126)

[3. Заключение 21](#_Toc96372127)

[4. Библиография 23](#_Toc96372128)

[5. Приложение 25](#_Toc96372129)

[5.1. Логическа блок схема 25](#_Toc96372130)

[5.2. Техническа блок схема 26](#_Toc96372131)

[5.2.1. Интерфейс 26](#_Toc96372132)

[5.2.2. Прихващане на заявки 27](#_Toc96372133)

[5.3. Графичен интерфейс 28](#_Toc96372134)

[5.3.1. Прототип 28](#_Toc96372135)

[5.3.2. Месечни отчети 28](#_Toc96372136)

[5.3.3. Седмична програма 28](#_Toc96372137)

# 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

# 2.1. Сравнение на възможни имплементации

Python v Java v C++

# 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.

TODO: sravnenie mejdu python I drugi ezici

# Реализация

Реализацията ще разгледа имплементацията на програмата, разрешените проблеми, както и техническите подробности. Ще бъдат правени референции към местата от кода, където се намират съответните части, а той може да бъде намерен 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. Всички детайли на използването трябва да са обявени и инициализирани преди да бъдат извикани.

Реализацията на проекта се състои от няколко фази. Първоначалната фаза е разработка за дисциплината „курсов проект“ през втори семестър, след което е надградена за целите и нуждите на дипломната работа. Съответната версия съдържа 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, която притежава разширена функционалност в количество и качество. Използваните данни са структурирани и логически разделени. Има значително по-голяма яснота при манипулация на данните (пр: взима се lecturer – името на водещият лектор, вместо елемент на трето място от масив – data[2]). Добавени са и две допълнителни функционалности – създаването на форматирана програма от седмица **А** до седмица **B**, съдържаща информацията за всички занятия през това време. Съответната програма притежава избор на вид файл за съхранение, и е удобна за пренос и принтиране. 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.

## 2.2.1. Графичен интерфейс

Програмата започва логически от графичният интерфейс в ExporterInterface.py. Той трябва да бъде удобен за използване, елементите му да са разпознаваеми и интуитивни за тяхната функция, а пътят на изпълнението на програмaта да е възможно най-кратък. Блокова схема на функционалноста му може да бъде намерена в приложението ([интерфейс](#_5.2.1._Интерфейс)).

Първоначалният интерфейс на програмата ([прототип](#_5.3.1._Прототип)) е сравнително малък, с ограничена функционалност. Подчинява се на блок схема, представляваща квадратна матрица от блокове, които биват запълвани от различни елементи. Причините за това са две – първо, по този начин е изключително лесно да се скицира и одобри промяна по дизайна. Всеки елемент си има строго определено място, може лесно да бъде разместен, и няма нужда да бъде тестван програмно всеки път, тъй като лесно се репродуцира на хартия. Второ – това е един от трите вградени начина за подредба на визуални елементи на използваната библиотека **Tkinter** (Roseman). Това улеснява работата върху интерфейса, като след приемане на финален вид на дизайна той може лесно и точно да бъде прехвърлен върху програмата. Дизайнът е разпределен на три части:

### 2.2.1.1. Месечни отчети

Дизайнът на [месечните отчети](#_5.3.2._Месечни_отчети) е продължение на прототипът. В него всички полета са преведени на български език, като в самата програма са добвени конвертори. Самият сървър работи с латински стойности, а също така низове с кирилизирани символи създават допълнителна сложност при написването на програмата. В същото време, за потребителят е важно да разпознава лесно елементите на дизайна, което е по-лесно на кирилица. Чрез конверторите е възможно да се запазят оригиналните латински стойности на променливите за подаване към хода на програмата, без допълнителна сложност. Освен старите полета са добавени и нови такива – избор на име на файл и неговото разширение. В случая на месечните отчети, полетата за преподавател и разширение са сложени с една единствена стойност, тъй като те не могат да бъдат създавани за студенти или стаи, и вече има приет формат на разработка. По време на разработката на дизайна се появяват усложнения с точната подредба на нужните елементи. Тяхното разрешение идва в използването на логото на университета като попълващ елемент, което допълнително подчертава функцията на програмата.

### 2.2.1.2. Седмични програми & индекс на седмиците

Дизайнът на седмичните програми е почти идентичен на този на отчетите, като месеците са сменени за седмици, а предишно заключените полета този път са избираеми. Тъй като в този случай програмата може да бъде използвана както от преподавателите, така и от студентите, видът заявка може да бъде избиран. Допълнително, оригиналното разширение от прототипа **.xlsx** не е подходящо за създаване на седмични програми, така че бива заменен от **.txt/.docx** – съответно елементарен текстови файл, и такъв на **Word**.

Считайки, че не може да се разчита да се помнят седмични индекси е добавен допълнителен екран за помощ, в който лесно може да се свери принадлежността на седмица към месец, и да се идентифицират желаните индекси. Индексите се генерират при стартиране на програмата за денят, в който е отворена. Съответно те са валидни за годината към която принадлежат.

### 2.2.1.3. Технически подробности

Интерфейсът е създаден с помощта на **Tkinter** (Python Software Foundation) и хартиени източници за прототипи. Всички елементи са част от библиотеката на модула, като стойността и подредбата им се задава преди стартирането на програмата, след което се извиква *root\_frame.mainloop()* (ExporterInterface.py, ред 286). Това казва на библиотеката да поддържа елементите на екрана като ги актуализира на всеки кадър, което прави програмата да изглежда постоянна на екрана, противно на това да се загаси веднага след стартиране.

При стартиране първоначално се взимат всички съществуващи елементи в конфигурационният файл. Това позволява консистентен графичен вид между различните изпълнения на програмата, както и удобство на потребителя. В идеален случай ще е нужно само промяна на месеца/седмицата при следващото стартиране.

Функционалните елементи са бутоните на програмата, които изпълняват съответната команда при натискане. Освен избирането на директория за съхранение, те изпълняват и самите експорти, като са обвързани с функции, извикващи останалите логически части на програмата. Важно е да се споменат две неща тук. Първо, след изпълнение на експорт, потребителят получава известие във формата на диалогов прозорец. То информира заедно с името на създаденият файл дали действието е успешно или не, след което връща контрола на програмата. В случай на успех, това не изисква допълнителни действия, но в случай на проблем, тъй като не е препоръчително да се препращат стек трейсове и кодове за грешки директно към потребителят, самата грешка няма да бъде показана, а само генерично съобщение. След това потребителят има отговорността да прегледа зададената от него информация и да я изпрати заедно със стъпките за пресъздаване към програмиста, който да разбере как и защо програмата не работи правилно.

Един от недостатъците на графичният интерфейс е липсата на прогресен показател. Това се оказва проблематично, тъй като създаването на месечни отчети минава през няколко фази, някои от които са времеемки, отнемащите между 5-10 секунди. Въпреки че потребителят няма контрол над програмата през това време, за да се опита да пуснет отчет наново или да изключи програмата, като по този начин се намеси в правилното и протичане, той няма никакъв индикатор дали съответното протичане върви по план, или нещо в програмата се е объркало. Това може да бъде неудобно и е отчетено като недостатък, които не е разрешен поради ограничето количество време.

## 2.2.2. Прихващане на заявка

Прихващането на заявките става от следващата част на програмата (ExporterRequestProcess.py), която приема зададените параметри от графичният интерфейс и формулира серия от заявки спрямо тях.

### 2.2.2.1. За месечни отчети

В случаят на месечните отчети, първото нещо, което се изпълнява е да бъде взето името на месеца и да бъде превърнато в индекс (пр. December -> 12) с помощта на времевата библиотека (Python Software Foundation). Той се използва за намиране на индекса на първата му седмица, тъй като функциите на библиотеката работят с номер, а не с наименование на месец. След това по подобен начин биват намерени и общият брой седмици в съществуващият месец. Генерира се списък с желаните седмични индекси, след което продължава изпълнението.

### 2.2.2.2. За седмични разписания

В случаят на седмичните разписания, самият списък от седмици вече е генериран от самият потребител, когато посочи начално и край на разписанието. Хубаво е да се знае, че самият сървър на университета успешно тълкува седмичните индекси без значение дали стойността им превишава 52. Това е често срещан проблем при работа с време, където последните няколко дни от една година могат да навлязат в седмица 53, или в нулева (0) от следващата година. Тъй като този проблем вече е разрешен вътрешно от някаква част на сървъра, програмата няма нужда да се справя с него. След избиране на индексите продължава изпълнението.

### 2.2.2.3. Изпълнение на седмични заявки

Създава се първоначално празна структура от данни, в която да се съдържа ежедневната информация. Започвайки от индекса на първата зададена 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, с номер626201 за първата седмица на Октомври - 39, тя ще изглежда така:

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

Октомври от 2021 има 5 седмици, съответно ще бъдат създадени 5 различни заявки в подобна форма. Всяка заявка бива създадена и изпълнена с помощта на модул **requests** (Reitz). 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 човешки четим. За да бъде полезна, информацията трябва да се преработи, което се случва по време на филтрацията.

## 2.2.3. Филтриране на информация

След прихващане, информацията се препраща към *sanitize\_weekly\_data(raw\_data, month, query\_type)* (ExporterRequestProcess.py, ред 69), където се извлича нужната информация и се добавя към временна седмична структура. След края на всяка седмична заявка, тази структура бива добавена към пълната месечна или много-седмична, след което автоматично се унищожава преди създаването на следващата заявка.

Извличането става чрез използването на модул **re** (Python Software Foundation). Използват се няколко различни регулярни израза, в зависимост от състоянието на информацията в таблицата, и наличието или липсата на данни. Всеки израз се състои от комбинация изрично обособени части и шаблонни такива. Примерно при търсене на програма за „понеделник“ ще бъде търсено мястото в таблицата, където пише точно думата „понеделник“ – точно обособена част, за чието наличие е известно. Докато при търсене на дата ще се използва шаблонна част – комбинация от символи от избран вид (числа, букви, празно място и др.) с определена дължина. Ако първият понеделник на Октомври е **2021-10-04**, тогава регулярният израз, на който отговаря е *[0-9]{4}-[0-9]{2}-[0-9]{2}*. Това значи, че модулът ще търси текст с поредица от четири цифри, тире, две цифри, тире, и отновно две цири. По подобен начин са формулирани и останалите изрази, но тъй като биват сравнително по-сложни, а регулярните изрази не са известни с лесната си четимост, те нямат да бъдат разглеждани детайлно. Вместо това ще бъде коментиран начинът им на използване. Съществуват два случая, които трябва да бъдат покрити от изразите:

* ден без налични лекции/упражнения
* дни със налични занятия

### 2.2.3.1. Основни шаблони и изпълнение

Спрямо наличието на занятия самият ден има различна структура в суровият **HTML** код, което изисква третирането му по различен вид. На база на това са изградени четири различни шаблонни изрази, които да прихващат всички разновидности на информация. Те могат да бъдат намерени във ExporterUtil.py (започвайки от ред 133) и са следните:

* *daily\_regex\_template(day, month)* – структура на на ден от седмицата в таблицата.
* *daily\_schedule\_regex\_template()* – обща структура на ден с лекции
* *no\_lecture\_regex\_template()* – обща структура на ден без лекции
* *lecture\_regex\_template()* – структура на лекция.

Логическата последователност на обхождането на седмична информация е следната – за всеки ден от седмицата се намира структурата на съответният ден, след това се търси структура на ден без лекции. Ако е намерен такъв, изпълнението продължава към следващият ден като се попълва единствено името и дата на деня в следващото място на масива. Ако не е намерен се търси обща структура на ден с лекции, след което тя се извлича за изолирана обработка, тъй като в противен случай следващата част на изпълнението неправилно засяга и лекции от други дни. След извличането и се прилага лекционна структурата, която връща като резултат всички налични лекции за даденият ден, елементите на които са логически разделени на групи. Всяка лекция се препраща за индивидуално извличане и преработка към *sanitize\_weekly\_data()*(ред 47).

### 2.2.3.2. Структура на данните и извличане

В прототипа на програмата всеки отделен елемент бива добавян като символен низ към масив. Това бива достатъчно за нуждите на дисциплината „Курсов проект“, но надграждането му изисква независима структура от данни. Тъй като различните видове заявки имат различни елементи, не може да се разчита на взимането на елементи от масив по индекс, тъй като не се знае какво представлява всеки един от тях. Съответната структура трябва да е способна да съдържа желаните елементи поименно.

За тази цел са създадени класовете *Day* & Lecture, намиращи се в DayData.py. Те са контейнери, които позволяват лесно и удобно съхранение на използваната информация. Всеки обект от тип *Day* държи в себе си списък от лекции, името и датата на съответният ден, които по-късно служат за вписване в седмичните разписания, както и в логването на информация. Всеки елемент от списъкът е от тип *Lecture*, като самият клас е направен да поддържа всички седем вида информация, които могат да бъдат извлечени. Спрямо видът заявка, различни полета от лекцията се попълват (пр. при лекция от тип *преподавател*, полето за преподавател ще остане празно). Съответните класове позволяват много по-смислено и удобно използване на информацията.

След структурно извличане на съществуваща лекция във формата на **HTML** код, информацията в него бива санирана спрямо вида заявка. Това става посредством използване на групите на регулярните изрази. Това са логически разделени части от израза, обособени в самият него, чрез които може да се отдели и извлече всяка нужна част. По този начин структурата на една лекция съдържа общо седем групи. В тях са включени постоянните:

* номер на часа
* продължителност
* наименование на дисциплина

И условните:

* номер на класно
* преподавател
* пореден номер на занятие
* номер на стая.

Попълват се задължителните полета, след което и избираемите такива спрямо заявката.

### 2.2.3.3. Технически подробности

По време на разработката се стигна до информация, съдържащата неправилно форматирани елементи – всичко, което е на български език бива показано на екрана като битова поредица от символи, тъй като по подразбиране се използва вид кодиране, което не поддържа кирилица. С тази цел са направени две промени: всеки отворен файл по време на изпълнението на програмата (лог, експорт и др.) бива отворен с кодиране от тип UTF-8 (Yergeau, 1998), и е инсталиран допълнителен модул **regex** (Barnett, 2022), който допълва липсващи функции на оригиналният **re**, като добавя флаг за **UNICODE** символи, като по този начин става възможно да се извлича безпроблемно кирилизирана информация от заявките.

В случай на проблем по време на заявката са добавени кодови блокове, които да прихващат различен вид грешки. При наличието на такава заявката спира, а резултатът бива записан в логовият файл.

## 2.2.4. Експортиране на данни за месец

След обхождане на цялата входна информация и извличане на всички елементи, програмата преминава към експортиране на данните към желаният формат, кодът за което се намира в *FileExporter.py*. Месечните отчети включват голямо количество сложност в сравнение със седмичните такива. За по-лесно проследяване е предоставена блок схема на процеса TODO.

### 2.2.4.1. Допълнителна обработка

Първоначално се взима филтрираната информация, подадена като списък от *DayData* обекти - *format\_lecture\_data\_for\_monthly\_report(data)* (ред 48). Тъй като месечният отчет изисква много различни стойности, които не са налични в първоначалната информация, тя трябва да се преработи повторно. Подобно на класовете *Day* и*Lecture* е нужна допълнителен клас за работа със занятия. За това се създава класът *LectureDataForExport.py.* В него се съхраняват всички възможни стойности, които могат да бъдат извлечени от дадена заявка и допълнителните файлове. Всяка лекция се преобразува в такъв елемент.

При създаването му се записват името на дисциплината и списък от номерата на групите, като допълнително от името се извличат ключови думи показващи дали съответното занятие е лекционно, практическо или изпит, и се записват като негов тип в обекта. Всички останали предишни елементи вече не са нужни. След изхвърлянето им, много от новите обекти имат еднакви стойности, което е изключително удобно за следващата част– сортиране и преброяване на обектите. От една страна това помага за оптимизацията, като значително намаля броя на обектите, които трябва да бъдат обхождани, от друга страна е нужно, защото отчета съдържа и брой на дадените занятия. След сортирането, всеки дублиран обект бива преброен и добавен към структурата, преди дубликатите да бъдат премахнати.

После се зарежда файла с данни, съдържащ кодовете на дисциплините и броят и типът на студентите за всяка група. Обхождат се и се сравнява стойноста на името на дисциплината със оригиналното извлчеено име което е комбинация от името на дисциплината и типът му – лекция, уипражнание и т.н.

### 2.2.4.?. Технически подробности

Тъй като не е наличен конкретен индикатор показващ това, отново се разчита на библиотеката **re** (Python Software Foundation), с която се търсят низове съдържащи думите „пз“, „практическо занятие“ или „упражнение“.

Докато сравняването на атомарни променливи като числа и низове е вградено във всеки език, сравняването на обекти по подразбиране се базира не на стойността им, а на адресите им. Ако два обекта са създадени един след друг с еднаква стойност, то всеки от тях заема различна част от паметта на компютъра, и всеки има различен адрес там, което води до логическо разминаване в очакваният резултат при сравнение. Това е проблем при сравнението на обекти, който се решава чрез пренаписването на някои от вградените функции в **Python**. Това са такива функции, които присъстват във всеки сложен обект, и служат за инциализация, сравнение, сортиране и други основни операции. В случаят, се интересуваме от две определени – хеш функцията - \_\_*hash\_\_()* и тази за сравнение - \_\_*eq\_\_()* (основните функции в питон могат да се разпознаят по наличието на двойни долни черти от всяка страна на името).

Хеширащата функция се използва за създаване на еднопосочна числена стойност наречена „хеш“, произлизащата от комбинацията на всички хешове на вече съществуващите елементи в един обект. По този начин се стига до хеш на атомарни стойности, които имат конкретна имплементация и връщат очакван резултат. Чрез съответната преработка се стига до единствено число, което винаги ще бъде едно и също при еднакви входни данни. Чрез комбинирано хеширане на всички елементи на класът *LectureDataForExport.py* се възвръща логическата точност при сравняване на елементи от съответният клас.

Подобно на това се имплементира и функцията за сравнение, която приема като параметър втори обект. Основната проверка тук, е видът на вторият обект, след което се сравняват комбинираните стойностите на двата класа.

Важно е да се отблежи, че функционалността е строго зависима от състоянието на шаблонният файл. Тя е направена специфично с даденият шаблон в предвид, и всяка промяна в неговата структура има голям шанс да засегне успеха на експортирането. Това е така основно поради наличието на видове студенти – курсанти, бакалвари, магистри и прочие. Всяко занятие на определената група си има място в шаблона, поради което за най-голямо улеснени, номерата на съответните места (редове в **Excel**) са строго зададени в кода на програмата. В заден план, това е разпознато като грешен подход на действие, като едно от възможните решения за това е да се добави конфигурационна секция, в която да бъдат попълвани ръчно подобни настройки. По този начин те ще могат лесно да бъдат променяни от по-опитните потребители на програмата, но едновременно с това няма да присъстват в графичният интерфейс, тъй като ще представляват излишни елементи.

Също така, това е и най времемката част на програмата. Тъй като не съществува закачен прогрес бар за показване на прогреса на изпълнение няма как да се проследи с точност нейното изпълнение, но по време на създаването на месечни отчети се губи най-много време, което е потвърдено от наблюдения по време на разработката. Това се случва най-вече във вторичната преработка на информация, и в записването на самата информация в шаблонният файл. Това се дължи на работата със сложен тип файл ­- **.xlsx** и на многократното обхождане на елементи и вградени цикли.

TODO: скорост O(n)

От цялата информация се извличат различни масиви със занятия, като за филтрация се използва видът студенти (пр. бакалаври), тъй като от редът им зависи правилното функциониране на програмата. За да няма разминаване се започване в обратен ред – от магистри към бакалаври – като групите намиращи се най-долу в шаблона са с предимство. Причина за това е имплементацията на програмата, в която са изнесени номера на редове, от където да започне попълването на всяко занятие.

Спрямо тях се извлича всичко останало от допълнителни файлове, след което се зарежда шаблонният файл, в който да бъдат попълнени занятията за даденият месец.

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.

2.2.5. Експортиране на програма за седмицата

2.2.6. Компилация към изпълним код

2.2.?. Конфигурация

Позволяване на прехвърляне на конфигурация

2.2.?. Логване на информация

2.2.?. Unit testing

2.2.?. Хеширане на информация

2.2.?. Code style

Самият код цели да е 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:

2.3. Нереализирани възможности

2.3.1. Мобилно приложение

2.3.2. PDF Експорт

2.3.3. Хеширане във реално време спрямо GitHub

2.4. Наръчник за използване

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/> 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.

## 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

### 5.2.1. Интерфейс

Graphical user interface, application, Teams

Description automatically generated

### 5.2.2. Прихващане на заявки

Graphical user interface

Description automatically generated

## 5.3. Графичен интерфейс

### 5.3.1. Прототип



Graphical user interface, application

Description automatically generated

### 5.3.2. Месечни отчети

### Graphical user interface, application Description automatically generated 5.3.3. Седмична програма

Graphical user interface, application

Description automatically generated