|  |  |
| --- | --- |
| 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.

TODO: Кратко описание на анализът

## Сравнение на възможни имплементации

За създаването на програма за отчети могат да бъдат използвани множество инструменти и езици. Някой от тях са специализирани за работа в уеб обстановка, приложения или нещо друго, докато други могат да бъдат пригодени към голямо количество цели, спрямо нуждите на проекта. Съответно, разликата между тях е, че генеричните езици имат доста по-обширна функционалност и по-малки ограничения, но в каквато и посока да се използват няма как да изпълняват финалната си цел толкова добре, колкото специализираните. Нека бъдат разгледани някои от широко използваните варианти:

### Java

**Java** (Oracle) е програмен език съществуващ от 1995 година и е един от най-широко използваните езици. Името му идва от кафените зърна отглеждани на остров Джава, около което се изгражда и бранда на езика в последствие (Murphy, 1996). Джава е език от високо ниво, използващ класове и принципа на обектно-ориентираното програмиране. Той няма тясна специализация и може да бъде използван за различни цели. Включва автоматично почистване на обектите в паметта, познато като garbage cleaner. Така спестява на програмиста нуждата сам да контролира паметта на програмата, което изисква детайлно познаване на паметта на компютрите и начините за оптимизация. Това помага да се фокусира върху нужните функционалности, но едновременно е и ограничение, което не позволява фин контрол върху оптимизацията на програмата. Също така, езикът се компилира до байтов (byte) код, който позволява изпълнението на програмата върху всяка машина без значение от операционната система. Това улеснява работата и спестява разработката на различни версии за различните системи. За използването на програмата след това е нужна джава виртуална машина – **JVM** (Hartman, 2022), която тълкува съответният код и позволява изпълнението му.

### C++

Почти всичко споменато за **Java** също важи и за **C++** (The C++ Resouces Network). Разликата е в компилацията на кода и ръчното менажиране на паметта. Езикът не позволява автоматично изчистване на вече ненужните обекти, като се очаква от програмиста да контролира внимателно отделянето и използването на паметта. В противен случай програмата може да изчерпи наличната памет и да блокира, или да заключи части от паметта като неизползваеми по време на изпълнение. Самият код се компилира до битов код (bit), също познат като машинен код, които е едно от най-ниските възможни нива на компилация, и е изключително оптимизиран. Той комуникира директно с процесора и не е нужен междинен интерпретатор.

### 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. Тъй като той самият е без специализирано предназначение 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 работата им бива споделена в интернет. По този начин днес **Python** е от най-широко използваните и поддържани езици.

Също така поддържа множество стилове и методологии на програмиране. Подобно на **Java**, той има вградени инструменти, които автоматично почистват обекти от паметта след тяхното използване. Особено е, че използва стандарт за кодиране **PEP-8** (Rossum, et al., 2001). В него кода се структурира чрез табулатори, които служат както за четливост и компилация. Чрез тях Питон разбира къде има логическа или структурна промяна.

### 2.1.4. Избор

В сравнение с другите езици, стандартен код написан на Питон е около 3-5 пъти по-къс от еквивалента му на **Java**, и от 5-10 пъти по-къс от еквивалента му на **C++**. Това ще рече, че работата, която може да бъде свършена от програмист на Питон за 1 месец, може да бъде написана на **C++** за повече от половин година (The Python Software Foundation). Програми написани на Питон се очаква да вървят и по-бавно от Джава или Си, тъй като не са типизирани. По време на компилация, в другите два езика се задава точният тип на един обект, което значи, че самата програма няма да бъде компилирана в случай на несъответствие. При изпълнение, програмата не губи време в проверки изчислява желаната функция много по-бързо. Спрямо тях, Питон притежава много по-голяма свобода по време на компилация. Всичко може да бъде описано в код, и стига логически да има смисъл, ще бъде компилирано успешно. Но когато по време на изпълнението на програмата се стигне до съответното действие, променливите от двете му страни (пр. А+В) трябва да бъдат анализирани за тип – дали и двете са числа, символни низове или обекти, и при несъответствие програмата ще хвърли грешка. Това води до дилемата за баланса между скорост по време на разработка и оптимизация по време на изпълнение.

В идеален случай е най-удобно използването на комбинация от езиците. Различни части на приложението могат да бъдат написани на Джава или Си, а Питон да бъде използван като „лепило“ между тях, улеснявайки логическата връзка между различните части, чиито технически подробности не са важни при комбинирането им.

Освен тези сравнения на практика изборът зависи от реални ограничения като цена, опит, лицензи, както и личен избор. В случаят те не играят голяма роля, тъй като и трите езика са, или имат версия, която е напълно безплатна и разработвана с отворен код, като наличната функционалност е повече от достатъчна за целите на този проект.

За нуждите и целите на дипломната работа се избира да се работи с Питон. Този избор е обособен от :

* скоростта на разработка и създаване на прототипи спрямо другите езици
* големият набор на допълнителни модули, които лесно могат да бъдат използвани и сменяни
* лично предпочитание

## 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.

### Модули на Python

#### 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.

#### 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.

#### 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.

#### 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**.

#### 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.

#### 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.

#### 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.

#### datetime & time

**Datetime** е един от основните времеви модули на **python** (Python Software Foundation), работещ с дати. 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.

**Time** е допълнителен модул, с които ще бъде измерено времето, което отнема на програмата за операции, като по този начин ще бъде сравнено изпълнението на отделните и части. **Time** работи с хронологично време, но се използва за точни измервания, като може да бъде настроен какво да включва – основният процес на програмата, или да включва и допълнителни процеси като автоматичното чистене на използвани елементи.

#### 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 (The Python Software Foundation) и е част от основните модули.

#### 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.

#### 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.

#### Стандарт за писане

В написването на една програма е важно да се спазва единен стандарт. Това подпомага яснотата и четливостта на кода както между различните файлове на една програма, така и между всички различни приложения написани на съответният език от други хора. Голяма част от работата не един програмист не е писане и имплементация на нова функционалност, а поддръжка и тестване на стара (Grams, 2019). Съществуването и спазването на стандарти са част от нещата, които помагат това да се случва по-бързо. В случаят на **python** ще се спазва вече споменатият стандарт **pep8** (Rossum, et al., 2001), а инструмента, които ще улесни това е **autopep8** (Hattori, 2021). Той е способен да бъде приложен върху файл с разширение *.py* като автоматично поправи всички грешки свързани със стандарта на писане.

## 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.

### Обобщение

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.

### Логически проблеми

#### Налична информация

Нека се разгледа информацията, от която зависи цялата програма. Това е публично достъпна информация, която всеки студент и лектор използват за ориентир. Тя се намира в разписанието (ВВМУ "Никола Йонков Въпцаров")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 занятия през зададената седмица спрямо избраните параметри. В нея съществуват седем различни елемента за всяко едно занятие. Три от тях винаги се срещат:

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

А останалите четири зависят от видът на заявката:

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

Спрямо видът се появяват тези елементи, които не и съответстват. Тоест, при избиране на разписание за **класно отделение**, тогава е логично то да не излиза във финалната таблица, тъй като ще е еднакво навсякъде. Извън това, изключение прави **пореден номер на лекцията** се появява само при заявка от вид **преподавател**.

#### Нужна информация

Нека бъде разгледано и 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 дисциплина
* номера на класни отделения
* брой лекции/упражнения на съответната група за месеца
* номер и разделение по групи
* изпити и други…

Видимо е, че информацията е по-обширна от наличната в заявките. За правилното изпълнение на целите на проекта и постигане на максимален процент автоматизация ще са нужни допълнителни средства. След консултация с ръководител се оказва, че такива са налични, отново, в публичното пространство на университета. Те показват съотношението между номерата и наименованията на дисциплините, както и това между номерата, видът и броят на студентите. Чрез тях ще може да бъде попълнена значително по-голяма част от нужната информация, но е невъзможно да бъде достигната надеждна пълна автоматизация. Също така, съответните документи най-вероятно се препокриват през годините, но ще трябва да бъдат преразглеждани или напълно сменяни при всяка следваща година за гарантиране дейността на програмата. Има нужда всеки файл да отговаря на формата, които присъства в сегашните, така че да се разчита на консистентно поведение.

#### Зависимости

Програмата е значително 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 оказва критично влияние върху функционалността и. При промяна малки промени винаги е възможно кода да бъде променен, така че да пасва на новите изисквания. Но програмата би била напълно обезсмислена при пълна смяна на наличният софтуер на университета. Считайки, че това са рядко и трудно случващи се събития, рискът от подобен проблем е нисък.

Също така програмата ще бъде зависима от наличните университетски файлове. Без тях правилното изпълняване на месечни отчети няма да бъде възможно, а единствената работеща функционалност ще е тази за създаване на седмични програми, която няма нужда от допълнителни файлове.

#### Наличие и 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. Такова е поведението и в сайта. Тоест, възможно е да съществува празен отчет или програма. В този случай е невъзможно да бъде направена разлика между липсващи данни за седмица и грешно въведени параметри без съответната проверка. След създаването им е добре такава да бъде извършена от потребителя. Така той може да потвърди наличието на информация във файла, както и да поправи и преформатира всякакви нередности.

Също така е възможно да бъде изпратена невалидна информация в заявката, като резултатът отново ще е грешен файл. За да се разреши този проблем е възможно добавянето на валидация на данните. Това става посредством ограничение на входни полета за дължина и формат, както и наличието на подсказки за очакваният вид данни.

#### 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 големина на екрана или качество, 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 отчети. 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 последно зададената стойност на всички полета (номер на преподавател, вид заявка, папка за съхранение и др.), за да спести време и да улесни работата на крайният потребител.

#### Логване на 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 да се анализира какъв е проблема. Стандартното действие на библиотеката **logging** (The Python Software Foundation) е да записва информацията директно в конзолата, като нейният формат и съдържание могат да бъдат всичко. Съществуват и различни нива на логване – DEBUG, INFO, WARN, ERROR, наименувани спрямо сериозността на съответното действие. Ползват се 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 разширение и персонализация, но в случаят е достатъчно да бъде пренасочена информацията от конзолата към текстови файл, за да може да бъде лесно споделяна от потребителите в случай на проблем.

#### Разпространение

След създаването на програмата е нужен и начин за разпространение. Желателно е той да е удобен, бърз и разбираем, без наличието на хардуерни устройства, тъй като това добавя допълнителна сложност и проблеми свързани със сигурността. Програмата ще трябва да се разпространява заедно с наръчник, които оказва очакваният начин на работа, и стъпките, които да се следват за достигането и, както и какво да се прави при наличието на проблем.

#### Блок схема на действие

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

## 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:

### 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.

### 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.

### 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.

Тук се засяга и валидацията на елементите. Не всичко, което потребителят може да впише във входните полета се счита за валидна информация. Въпреки това, за него не съществува полза от използването на такава, освен като опит програмата да бъде счупена. Дори при успех, нейното действие при следващо стартиране няма да бъде нарушено, като в резултат играта с данни по този начин не води до никаква облага, и не може да бъде използвана за злоупотреба. Спрямо този факт, и допълнителната сложност, която валидацията изисква при използването и с избраната библиотека, такава не е добавена. Вместо това се разчита на диалоговият прозорец за грешка, когато потребителят трябва да разпознае, че не е въвел правилните елементи.

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.

## Прихващане на заявка

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

### За месечни отчети

В случаят на месечните отчети, първо се изпълнява взимане името на месеца и се превръща в индекс с помощта на вградените функции на времевата библиотека (Python Software Foundation). Той се използва за намиране на индекса на първата му седмица, тъй като следващите функциите на библиотеката, които ще бъдат използвани 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.

Ако се вземе за пример месец **Февруари** на **2022**, името ще бъде преобразувано в индекс **-** **2**, след което ще бъде взет индекса на първата му седмица, които се явява номер **6**. **Февруари** се изчислява да притежава общо **5 седмици**, което значи, че заявки ще бъдат генерирани за списък от седмиците – [**6, 7, 8, 9, 10]**.

### За седмични разписания

В случаят на седмичните разписания, самият списък от седмици вече е генериран от самият потребител, когато посочи начало 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 прелеят в седмица **53**, или в нулева **0** от следващата година. Тъй като този проблем вече е разрешен вътрешно от непознат инструмент на сървъра, програмата няма нужда да се справя с него. След избиране на индексите продължава изпълнението.

### Изпълнение на седмични заявки

Без значение от използваният вид функционалност, изпълнението на седмични заявки се осъществява по един и същи начин с подаване на списък от седмични индекси. 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:

*Основен път към сайта + идентификационен номер + вид заявка + номер на седмица.*

Ако искаме да направим заявка за **класно отделение**, с номер **626201** за първата седмица на Февруари с индекс **6**, тя ще изглежда така:

*https://nvna.eu/wp/?group=****626201****&queryType=****group****&Week=****6***

Февруари 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. 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.

Преди продължаване дейността на програмата се извършва проверка за върнатият код от сървъра. Всяка заявка в интернет връща един от множество кодове, определени по стандарт (MDN Contributors, 2022). Стандартната очаквана стойност при успешен резултат е код 200. Проверката удостоверява наличността му, след което се продължава нормалното изпълнение. В противен случай се повдига код за грешка и изпълнението се връща до графичният интерфейс, а грешката се логва във файла.

## Филтриране на информация

След прихващане, информацията се препраща към санираща функция *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 от вградените инструменти в Питон 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 или шаблонни части. 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 Февруари е във формат **2022-02-07**, 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 три случая, които трябва да бъдат покрити от изразите:

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

### 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 заглавен ред 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).

### 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.

### 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.

## Експортиране на данни за месец

След обхождане на цялата входна информация и извличане на всички занятия, 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 програми. За по-лесно проследяване е предоставена блок схема на процеса TODO.

### 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 бива преобразувана в такъв елемент.

При създаването на всеки такъв елемент се записват името на дисциплината и списък от номерата на групите. Допълнително от името се извличат ключови думи показващи дали съответното занятие е лекционно, практическо или изпит, и се записват като негов тип в обекта. Съществуват 3 вида занятие индексирани със символи (*LectureDataForExport.py*, ред 4):

* **L** – lecture
* **P** – practice
* **E** - exam

Всички останали предишни елементи вече не са нужни и биват изтрити. Тъй като Питон не е типизиран език, същата променлива, която до сега е държала списъкът с непреработени обекти, сега ще държи списъкът с преработени такива. След изхвърлянето на излишният елементи, много от новите обекти придобиват 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 очаквано, защото един от елементите на отчета е брой на дадените занятия през месеца. След сортирането, всеки дублиран обект бива преброен и добавен като стойност, преди дубликатите да бъдат премахнати.

След това се зарежда файл с данни, служещите за придобивани на допълнителна информация. Отново е от тип **.xlsx** и съдържа два списъка – един с кодовете и имената на дисциплините, и втори с броят и типът на студентите за всяко класно отделение. Обхождат се всички лекции и всички редове в съответната колона, като се търси символният низ репрезентиращ името на дисциплината във файла. Не се търси точно равенство, тъй като името, което дисциплината притежава в оригиналният си вид, идващ от заявките, е комбинация с типът му – лекция, упражнение или изпит. При успешно наличие на името се взима стойността на колоната до него, която показва идентификационният номер на дисциплината, и се вписва като следващ елемент в обекта.

По подобен начин се обхождат и вторият списък, които съдържа информацията за студентите. Разликата е в увеличеният брой обхождания, тъй като търсене трябва да бъде направено за всяка група. След намиране в съответният обект се попълват сумата от броя на студентите и техният тип – бакалаври, магистри или курсанти. Те се индексират с цифри по следният принцип (*LectureDataForExport.py*, ред 10):

* **0** – курсанти
* **1** – бакалаври редовно
* **2** – бакалаври задочно
* **3** – магистри

### Примерна обработка

Ако имаме обект *DayData*, в който имаме налична следната информация за практическо занятие:

* **Ден:** Понеделник
* **Дата:** 2022-02-21
* **Списък от лекции с една налична:**
  + **Продължителност**: 9:50-11:30
  + **Дисциплина:** Курсов проект от дисциплина от 2ри или 3ти семестър - пз група 1
  + **№ на занятие:** 4
  + **Зала:** виртуална стая
  + **Групи:** група/групи – 126202

Тогава след преработка ще имаме обект *LectureDataForExport* с данни за дисциплина, групи и вид занятие, като видът се удостоверява от наличието на „пз“ в името на дисциплината. След взимане на данните от файловете ще бъде намерено съдържанието на „Курсов проект от дисциплина от 2ри или 3ти семестър“ в наименованието на дисциплината под код „26131“, а до номера над класното отделение ще бъдат взети броя на студенти – „26“, и номер „1“. Финалният обект ще има данни със следните стойности:

* **Дисциплина:** Курсов проект от дисциплина от 2ри или 3ти семестър - пз група 1
* **Номер на дисциплина:** 26131
* **Групи**: [126202]
* **Тип** **занятие:** P (Практическо)
* **Брой студенти:** 26
* **Тип студенти:** 1 (Бакалаври редовно)

### Попълване на информацията

След извличането на тази информация се продължава към попълването в шаблонният файл. Редът на попълване е строго определен от шаблонният файл. Той е разделен на части спрямо видовете студенти (пр. курсанти), затова и попълването на информацията е разделена спрямо това. В самият файл не се прави разлика между редовни и задочни бакалаври, затова номера показващи „1“ и „2“ в типовете студенти се комбинират. Взимат се три константи индекса показващи номера на реда, от който всеки от типовете трябва да започне да се попълва. За да не се развали този ред, попълването става в обратен ред от магистри към курсанти.

Попълват се неизменимите полета – име на дисциплината и списък от групи, след което се проверява вида занятие. Ако е лекция се попълват следващите полета за общ брой студенти и занятия, ако е упражнение се взима половината брои студенти и същият брои занятия.

### Технически подробности

#### Извличане на вид занятие

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

#### Сравняване на обекти

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 в паметта. Ако два обекта са създадени един след друг с еднаква стойност, то всеки от тях заема различна част от нея и притежава различен адрес, 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**. Те 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 (*\_\_hash\_\_(self)*, ред 70) се използва за създаване на еднопосочна числена стойност наречена „хеш“, произлизащата от комбинацията на всички изчислени 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 – числата връщат себе си, а символните низове връщат манипулация от числените стойности на отделните символи. Чрез съответната преработка се стига до единствено число, което винаги ще бъде едно и също при еднакви входни данни. Чрез хеширане на елементите на класът *LectureDataForExport.py* се възвръща логическата точност при сравняване на елементи от съответният клас.

Подобно на това се имплементира и функцията за сравнение (*\_\_eq\_\_(self, other)*, ред 73), която приема като параметър втори обект. Основната проверка в случаят е видът на вторият обект, след което се сравняват комбинираните стойностите на двата класа (*\_\_key\_\_(self)*, ред 67).

#### Зависимости

Важно е да се отбележи, че функционалността на попълването 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. Всяко занятие на определената група си има място в шаблона, поради което 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.

#### Производителност

Скоростта на алгоритмите в програмирането се мери с горната граница на изпълнение, или най-лошият случай (Huang, 2020). При се анализ на кода от *FileExporter.py* (ред 48) се вижда, че има налични няколко вложени цикъла. Всеки от тях разчита на различни променливи, така че за изчисленията ще бъдат използвани следните символи:

* – брой дни
* – брой занятия
* – брой групи
* – брой редове
* – брой колони

Резултатът може да се разглежда по следната формула:

Считайки, че се търси най-лошият случай, търсим опростена версия на формулата, която да зависи от най-важните елементи. Броят на дните никога не може да е по-голям от броят лекции, а обхожданията винаги се случват върху една колона. Тези елементи отпадат и тогава формулата може да бъде опростена до следният вариант, които показва, че най-лошият вариант за скорост зависи от броят лекции, редове и групи:

Това е най-времеемката част от програмата. Потвърждава се от наблюдения направени при многократно изследване на производителността ([таблица за производителност](#_Производителност)). В сравнение с него, седмичните отчети за същото количество седмици се изпълняват два пъти по-бързо. Спрямо направените изследвания за един стандартен месец със 5 седмици, средната скорост за създаване на месечен отчет е 2,06 секунди, а тази на създаването на програма за същото време е 1,05 секунди. Разбира се, скоростта зависи основно от броят занятия, а не от броят на седмиците. Празна седмица би създала празен експорт за много по-кратко време, но точното такова не е подлагано на тест. Зависимостта между занятия и скорост е линейна. В таблицата може да се види резултатът за тестове 4-6, където количеството занятия във всеки следващ тест средно се удвоява, което води и до средно двойно увеличение на времетраенето на изпълнението. Ако времето за минаване през всички други стъпки на програмата не се взима под внимание, може да се стигне до заключението, че средната тежест на едно занятие се явява на **0,024** секунди. Ако се вземат наличните занятия в тестваните месеци – Януари, Февруари и Март на 2022 година – се стига до **55** средни занятия на месец на брой. Със средно отклонение **11%** може да се заключи, че средното нужно време за създаването на месечен отчет (**T**) е в рамките на:

Възможно решение за редуцирането на това време е изнасянето на всички данни във елементарен текстови файл **.txt**, които да поддържа информацията като двойки ключ и стойност, като по този начин се намали сложността и съответно се ускори изпълнението. От друга страна оригиналните използвани файлове са с разширение **.xlsx** и са с минимални промени. В случаят се задава въпросът дали е по-добре всеки нов файл да замени с минимални промени предишните, или да бъде изцяло пренаписван в друг формат.

От към страна на многократното обхождане, решение може да се намери в техническо анализиране на кода на програмата или смяна на използваната библиотека. **Openpyxl** не позволява директни търсещи операции върху избран списък от клетки, редове или колони. Единственият начин за обхождане е чрез избиране на съответният списък и минаването през него по редове и колони, което гарантира начална сложност от O(n2).

## Експортиране на програма за седмицата

Създаването на седмични програми работи по различен начин от седмичните отчети. След обработка на данните се извиква *export\_simple\_report(data, folder, file\_name, file\_type, weekly\_indices)*(*FileExporter.py*, ред 35). От него се викат различни функции спрямо типа файл, който потребителят е избрал. Има опции за файлове от тип **.docx** и **.txt**.

### Текстови файлове

При избиране на нормален текстови файл, информацията за всеки ден бива записана като се добавя нов ред между всеки ден и лекция за визуална четливост. Информацията, която се добавя е предварително форматирана в методите *info()* на класовете *Day* и *Lecture*, които връщат наличните си данни във форматиран начин. Примерен файл може да бъде намерен в приложението TODO.

### Файлове на Word

При избиране на файл от тип **.docx** информацията се разделя на таблици за всяка седмица. Създава се заглавен ред, в който се съхранява информация за номера на седмицата, както и началната и крайната и дата. След това за всяко занятие се добавя нов ред, в който се записва съответната му информация. След записване на цялата информация, всички клетки в таблицата се обхождат и стилизират, като всеки заглавен ред бива стилизиран по различен начин от информационните. По време на стилизирането се настройват формата, цвета и големината на клетките, техният фон и широчината на колоните. Финалният резултат е примерен и е извлечен от стойностите на съответните атрибути след първоначално ръчно наместване. Примерен файл отново може да бъде намерен в приложението.

### 2.8.3. Технически подробности

Формата, който се използва при създаване на файлове за **Word – docx** следва стандарта за разработка на файлове във формат **XML**, като използва библиотеките на **Word** за да създаде елементи, които могат да бъдат разчетени от програмата. Това прави използваната библиотека по програмен начин - **python-docx** (Canny, 2013). Тя позволява лесното отваряне и внасяне на елементи. За съжаление не е напълно разработена, като има липсващи функционалности за стилизиране и форматиране. Някои от стандартните възможности не са имплементирани и липсват, докато други са сериозно ограничени. За да компенсира това, тя позволява директното вграждане на **XML** тагове, което позволява използването на всички стандартно налични елементи. Тъй като резултатът от всички нейни функции е **XML** код, вграждането е един вид прескачане на тези функции, или компенсирането за тяхната липса, където ги няма. По този начин е използвано и стилизирането на заглавни редове, като техният цвят е вкаран допълнително с таг *<w:shd {} w:fill="A6A6A6"/>* (*FileExporter.py*, ред 221), където стойността на *w:fill* е шестнайсетично число оказващо цвета на клетката във формат **RGB**.

Отново може да се счита, че настройките на таблицата могат да бъдат добавени в отделна част на конфигурационният файл, където да бъдат избирани настройки като цвят, големина и шрифт на текста, но по време на проекта не е държало висок приоритет.

## Компилация към изпълним код

След като кодът е написан и тестван, той трябва да превърнат в изпълнима програма. Това става чрез използването на модул **pyinstaller** (PyInstaller Development Team, 2021). Той се използва от командният ред, като му се задават скриптовете, които да бъдат превърнати в програма, заедно с всички желани настройки. Ще бъдат разгледани тези, които са използвани в проекта и техният резултат.

Има няколко начин да се задейства създаването на програма през командният ред. Първият е директно вписване на името и желаните настройки, заедно с техните стойности. По този начин беше компилирана програмата по време на етап „курсов проект“, когато имаше ограничена функционалност и по-ниски цели. След разрастването на програмата се появи нужда от по-удобен начин на изпълнение поради използването на повече настройки и честото тестване. Едно от решенията е копирането на използваната команда в текстови файл и повторното и използване, но самият модул позволява по елегантни решения.

Има два варианта, от които може да бъде избрано решение. Първият е да се използва **.spec** файл, в който са описани същите настройки по формат, който модулът разбира. При стандартното вписване в команден ред, първото нещо, което се случва е да се генерира такъв файл със зададените стойности, така че този вариант може да се разглежда като премахване на горният слой и по-директна манипулация. Този вариант е препоръчан при по-голямо наличие на настройки и нужда от по-фина манипулация, но е и по-труден за използване.

Вторият вариант е подобен на вписването в текстови файл, но е по-удобен за използване от него и елиминира сложността на **.spec** файла. Това е пренасянето на командите в изпълним **.bat** файл, който да бъде задействан от командният ред с елементарно извикване. В него са посочени всички команди, настройки и техните стойности. При желание за промяна различни части от него могат да бъдат добавяни или коментирани, а самият файл се съхранява с програмата в системата за контрол на версиите. След като се знае начинът на създаване, нека се разгледат споменатите настройки

### 2.9.1. Файлове за компилация

Основното, което се избира при компилация е файлът, който да бъде превърнат в програма (Скрипт за компилация, ред 1). Могат да бъдат избрани няколко такива, но в случаят логическото начало на програмата се бележи от графичният интерфейс. Всеки друг скрипт е свързан директно или индиректно с него, което ще рече, че добавянето им е излишно и би добавило излишна сложност.

### 2.9.2. Път за изходни файлове

По време на компилация се създават много временни файлове, както и се подменят последните такива от предишното изпълнение. Тези файлове затормозяват съхранението чрез **GitHub** като запълват папката за съхранение с множество вторични файлове. По този причина са изместени пътищата на папките за временни и изходни файлове (Скрипт за компилация, ред 2-4). Оказани са пътища в самостоятелни папки, където могат да бъдат разделени трите основни типа файлове. В папката *build* се намират временните файлове за компилация. В папката *spec* се намират конфигурационният файл, който беше споменат по рано. Той се създава спрямо зададените настойки в началото на компилация и не съдържа нищо друго. В последната папка *dist* се намират изходните файлове от компилацията. Там се намира и самата изпълнима програма.

### 2.9.3. Генерация на еднолично приложение

За най-голямо удобство на потребителя и абстракция от допълнителни файлове, програмата се създава като еднофайлово приложение (Скрипт за компилация, ред 5). Също така при стартирането на програмата се създава и конзола, която може да се използва за проследяване на случващото се. Тъй като това нарушава визуалният вид на приложението, а същото проследяване може да се види в логовите файлове, се използва допълнителна команда (ред 6), която я премахва.

### 2.9.4. Прибавяне на файлове

По време на втората фаза са добавени много вторични файлове за използване на програмата. Към това число се включват изображенията и шаблоните. Чрез тях се задава логото на графичният интерфейс, изображението налично в екраните за отчети и програми, както и цялата информация, която се ползва при създаването на отчети. Без нея то няма да може да работи, като не може да се очаква потребителя да ги предоставя. Тези елементи са добавени в последните 4 реда на скрипта, като по този начин те се предоставят на модула за използване по време на компилация.

Те служат за още една цел. Приложението стартира сравнително бавно , когато се изпълнява от файл, спрямо изпълнение от програмна среда. За да не е объркващо за потребителя е добавен начален екран, който стои отворен докато приложението стартира нормално и графичният интерфейс се задейства.

### 2.9.5. Технически подробности

След успешна компилация приложението се генерира, но при стартиране то няма да може да се задейства успешно поради липсата на вторични файлове. Беше очаквано, че добавянето им в скрипта ще ги добави към самото еднофайлово приложение за вътрешно използване. Това не беше истинският резултат, поради което те са добавени в папка със същото наименование, което имат и в програмната среда. Те вървят ръка за ръка с приложението, и по този начин дори могат да бъдат лесно обменяни, без нужда от нова компилация.

## Конфигурация

Конфигурацията (*ExporterConfig.py*) 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. При всяко действие се изпълнява функция, която запаметява новите стойности на параметрите, като първо се проверява дали файлът съществува. Ако не, се създава нов такъв с разширение *.cfg*, след което се добавя секция с име *[request\_parameters]*, където ще бъдат съхранявани всички желани стойности. След това продължава стандартното изпълнение на програмата. По този начин също се елиминира и нуждата от пренасяне на шаблонен конфигурационен файл с изпълнимата програма, като може да се разчита, че при неговата липса той ще бъде създаден. Обратно на казаното, въпреки че може да се разчита липсата на файл да не е проблемна, наличието на такъв може да се окаже полезно при споделянето на файл между преподаватели, като новият му потребител ще има предварително заредени данни и ще може да го използва с минимални промени.

## Logging

Следващата част на приложението се грижи за събирането на техническа информация за случващото се в програмата по време на изпълнение и се нарича логър. Важно е да се отбележи, че тя по никакъв начин не събира или анализира информация за потребителя, 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 (*ExporterLogger.py*), към които реферират всички останали части на програмата, които имат някаква изходна информация. Този файл наследява и инкорпорира оригиналният модул за логване на **Python**, след което се настройва различните му елементи спрямо нуждите на приложението. Пренасочва се изходната информация към файл, които се създава наново с всяко пускане на програмата. От възможните нива на логване са предоставени две – *INFO, ERROR*. Използван е стандарт за кодиране от вид *UTF-8* за поддръжка на изходящата информация, тъй като тя включва кирилизирани низове. Форматът на изходящите данни е следният:

*НИВО:ИМЕ\_НА\_ЛОГЪР:ИНФОРМАЦИЯ*, като името винаги е еднакво.

Информацията, която се логва зависи от логическите граници на приложението. Логват се данни при стартирането на програмата, при изпълнение на експорт, в случай на грешка и нарушаване на стандартният ход на изпълнение, 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. Също така се съхранява информацията за скорост на изпълнение, която бе използвана по рано в техническият анализ.

## Unit testing

За установяване на грешки в кода той често бива детайлно тестван. Тестването в изолация помага за ранното намиране на такива преди те да стигнат до крайният потребител. Въпреки че това е честа практика, за този проект не остана достатъчно време за детайлен тест. Приложението е тествано по стандартен начин с проба и грешка по време на работа, а за информация в случай на грешка се разчита на логъра. Единственият клас, които е детайлно тестван е при създаване на обекти от тип *LectureDataForExport*, като се тества типът занятие след създаване. По време на разработката имаше проблеми с правилното удостоверяване на типовете, но тестването с нов отчет всеки път не беше достатъчно ясно и забавяше процеса, като добавяше и допълнителни елементи, които влияят на резултата. Поради тази причина проблемът беше изолиран и разрешен в тестови клас. Това е показателно за нуждата от наличие на тестове още при създаването на всеки скрипт и ще бъде взето под по-строго внимание при следващата разработка.

## Хеширане на информация

Целта на проекта е да създаде изпълним файл с конкретно изпълнение. Това поражда брой проблеми със сигурността. Програмата има достъп до интернет и изпраща заявки, съдържа изпълним код във формат .**exe**, и се разпространява публично в интернет пространството, като не се изисква автентификация при изтеглянето и. Ако се приеме, че оригиналният и източник е сигурен, нека бъдат разгледани възможните атаки към всеки един от тези елементи, и решенията, които могат да бъдат предприети.

### Уеб уязвимости

Програмата има достъп до интернет, което автоматично увеличава броят възможни уязвимости. Заявките се изпълняват по конкретно определен начин, което увеличава предсказуемостта им. Въпреки това, информацията, която те достъпват е публично достъпна и не изисква автентификация. Ако имаше нужда от такава, тя щеше да бъде използвана и в кода на програмата след съгласие от страна на университета за целите на дипломна работа. Тъй като функционалността е имплементирана без такава, рискът от атака от тип Man in the middle или друг вид нападение свързано с уеб трафик е зачетен като нисък, а последствия не са идентифицирани.

### Изпълним код

Компилирането на кодът в изпълнима програма води до няколко точки на уязвимости. Тъй като по време на работа се използва **pyinstaller** за компилацията, това води до въпросът за количество доверие, което може да се гласува на чужд модул при работа с него. Първият начин, по които това може да бъде измерено е чрез анализ на кода на приложението, екипът му и времето, от което съществува. Въпреки че няма как целият код да бъде прегледан, лесно може да се види, че цялото му количество е публично наличен на **GitHub** страницата на модула (PyInstaller Development Team). Екипът му се състои от няколко основни персонажа, но бива допълнен и от голямо количество самостоятелни разработчици, като на страницата са оказани всичките 384 сътрудници. Модулът започва съществуването си в публичното пространство малко преди 2006 година, което също може да се види на страницата. За предишна история не е ясно, но приблизително тогава е направено първото качване върху платформата. Тези факти вдъхват достатъчно доверие в публичният имидж на модулът и намалят всякакви тревоги свързани с него.

Въпреки това е напълно възможно да има чисто технически проблеми. Не най-малко поради наличието на толкова много сътрудници. След сверяване с два сайта за проверка на уязвимостите (Vulmon) (Snyk) е идентифицирана една такава. Уязвимост **CVE-2019-16784** е от тип ескалация на привилегии и е валидна за програми стартирани на операционна система Windows, които са компилирани с опция –*onefile* (с каквато е компилиран и този проект). Чрез нея е възможен непривилегирован достъп, когато програмата е пусната от системен акаунт като сървиз или с планирано стартиране, с допълнението, че уязвимостта е валидна единствено при вторично стартиране. Казано по друг начин, ако компилираният код се пуска със стартирането на компютъра той няма да бъде уязвим, но след рестартиране има опасност. Уязвимостта е идентифицирана за версии на модула от 3.6. надолу. Използваната версия за този проект е последната налична – 4.9., което води до заключението, че тя не представлява риск за изпълнението на програмата.

### Свободно разпространение

Изпълнимото приложение и оригиналният код на програмата, заедно с цялата документация са налични на сървъра на **GitHub**, където те се поддържат. Всеки е свободен да изтегли, използва и модифицира програмата по желан начин, стига той

която изисква поглед над възможността за атака от тип Man in the middle. 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.

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.

# 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

## Логическа блок схема



## Техническа блок схема

### Интерфейс

Graphical user interface, application, Teams

Description automatically generated

### Прихващане на заявки

Graphical user interface

Description automatically generated

### Саниране на седмична информацияDiagram Description automatically generated

### Месечни отчети

### Седмични заявки

### Конфигурация

A picture containing diagram

Description automatically generated

## Графичен интерфейс

### Прототип



Graphical user interface, application

Description automatically generated

### Graphical user interface, application Description automatically generatedМесечни Отчети

### Graphical user interface, application Description automatically generatedСедмична програма

### Graphical user interface Description automatically generatedGraphical user interface Description automatically generatedСедмици

### Graphical user interface, text, application Description automatically generatedGraphical user interface, text, application, chat or text message Description automatically generatedДиалогови прозорци

## Производителност

### 5.4.1. Използвани данни

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер на тест | 1 | 2 | 3 | 4 | 5 | 6 |
| Вид дейност | Отчет | Отчет | Отчет | Програма | Програма | Програма |
| Вид заявка | Преподавател | Преподавател | Преподавател | Преподавател | Преподавател | Преподавател |
| Номер | 2613 | 2613 | 2613 | 2613 | 2613 | 2613 |
| Месец/ Седмица | Януари | Февруари | Март | 1-5 | 1-10 | 1-20 |

### 5.4.2. Резултати

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер на тест | 1 | 2 | 3 | 4 | 5 | 6 |
| Седмици | 5 | 5 | 5 | 5 | 10 | 20 |
| Занятия | 36 | 61 | 68 | 34 | 99 | 228 |
| Брой повторения | 10 | 10 | 10 | 10 | 10 | 10 |
| Минимален резултат (сек.) | 1.578 | 2.062 | 2.015 | 0.926 | 1.966 | 4.353 |
| Максимален резултат (сек.) | 1.828 | 2.593 | 2.328 | 1.276 | 2.597 | 4.949 |
| Среден резултат (сек.) | 1.670 | 2.237 | 2.202 | 1.052 | 2.323 | 4.759 |

## Примерни файлове

### 5.5.1. Текстови файл

### 5.5.2. Word файл

## Скрипт за компилация

