МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2022 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ЗАСТОСУНОК ДЛЯ АНАЛІЗУ І КОНТРОЛЮ ОСОБИСТИХ ВИТРАТ ТА ФІНАНСІВ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 401.21810303

Виконав студент 4-го курсу, групи 401
_____ *В. П. Бондаренко*
«___» _____ 2022 р.

Керівник: *ст. викладач*
_____ *В. В. Кошовий*
«___» _____ 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2022 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи

Видано студенту групи 401 факультету комп'ютерних наук

Бондаренку В'ячеславу Петровичу.
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Застосунок для аналізу і контролю особистих витрат та фінансів»

Керівник роботи Кошовий Віталій Володимирович, старший викладач.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затв. наказом Ректора ЧНУ ім. Петра Могили від «07» 12 2021 р. № 318

2. Строк представлення кваліфікаційної роботи студентом «03» 06 2022 р.

3. Вхідні (початкові) дані до роботи: програма була створена с нуля.

Очікуваний результат роботи: розроблена програма в якій користувач може слідкувати, записувати, перевіряти та використовувати свої фінансові дані, щоб максимально комфортніше і продуктивніше розуміти своє фінансове становище.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки)____

__дослідження та аналіз сучасного стану задачі створення бухгалтерського звіту, а також розробка графічного інтерфейсу та проектування баз даних та бекенду_____.

5. Перелік графічних матеріалів __презентація_____

_____.

6. Завдання до спеціальної частини ___опис основних питань охорони праці___ _пов'язаних з професійною діяльністю та використання обрахунків для роботи_ _в дома та в офісі_____.

7. Консультанти:

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	ст. викладач О. В. Макарова	

Керівник роботи _ _ст. викладач, Кошовий Віталій Володимирович,_____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ____Бондаренко В.П._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «____» _____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: __«Застосунок для аналізу і контролю особистих витрат та фінансів»_____

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	27.10.2021	27.10.2021	Виконано
2	Отримання завдання на виконання БКР	14.04.2021	15.04.2021	Виконано
3	Складання календарного плану роботи на весь період виконання БКР	16.04.2021	01.05.2022	Виконано
4	Отримання завдання на переддипломну практику	02.04.2022	03.04.2022	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	19.04.2022	08.05.2022	Виконано
6	Розробка звіту з переддипломної практики	09.05.2022	11.05.2022	Виконано
7	Виконання БКР: аналіз існуючих альтернативних веб-додатків , огляд існуючих технологій, розробка ПЗ	06.06.2022	19.06.2022	Виконано
8	Попередній захист БКР на засіданні комісії кафедри	30.05.2022	30.05.2022	Виконано
9	Доробка та остаточне оформлення БКР	02.06.2022	20.06.2022	Виконано
10	Подання БКР рецензенту	16.06.2022	18.06.2022	Виконано
11	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	20.06.2022	22.06.2022	Виконано
12	Захист БКР перед екзаменаційною комісією (ЕК)	27.06.2022	29.06.2022	Виконано

Розробив студент _____Бондаренко В.П. _____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи __старший викладач,_____
_____Кошовий Віталій Володимирович____ _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 2021

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 401 ЧНУ ім. Петра Могили

Бондаренка В'ячаслава Петровича

Тема: «Застосунок для аналізу і контролю особистих витрат та фінансів»

Об'єкт роботи – технології та алгоритми для досягнення поставленого завдання, а саме розробки програми для контролю особистих витрат. Система для передачі даних між користувачем і банківської картки.

Предмет роботи – принципи та технології побудови бази даних в яку будуть приходити дані із банківської картки.

Метою роботи є автоматизація та спрощення програми, в якому користувач може користуватися на інтуїтивному рівні.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків та додатків.

У першому розділі дипломної роботи представлені основні положення та проаналізовано методи створення додатків для аналізу і контролю витрат. Аналізуються існуючі представлення банківські аналоги та порівнюються між собою.

У другому розділі дипломної роботи представлена послідовність розробки програми. Описується технологій, який був обраний для виконання завдань, архітектура додатку та база даних, обрана для цього проекту.

У третьому розділі дипломної роботи описано процес розробки програми та бази даних до нього.

У четвертому розділі описується технічна документація.

У п'ятому розділі описується аналіз умов праці, безпеки праці, гігієни праці та виробничої гігієни. Також описані вимоги пожежної безпеки та освітлення і мікроклімату.

За результатами роботи зроблено висновки та пропозиції щодо створення програми для спостереження за своїми фінансами.

Бакалаврська кваліфікаційна робота містить 75 сторінки, 51 рисунок, 1 таблицю, 25 використаних джерел та 0 додатків.

Ключові слова: ПЗ , Java, Python, C#, тулбар, wrapper, handle, сервер, клієнт.

ABSTRACT

bachelor's degree work of a student of group 401 at Petro Mohyla Black Sea National University

Bondarenko Viacheslav Petrovich

On topic: « Creating an application for analysis and control of personal expenses and finances »

Object of work - technologies and algorithms to achieve the task, namely the development of a program to control personal costs. System for data transfer between the user and the bank card.

The subject of work - the principles and technologies of building a database, which will receive data from the bank card.

The aim is to automate and simplify the program, which the user can use on an intuitive level.

The work consists of a professional section and a special section on labor protection. The explanatory note consists of an introduction, four chapters, conclusions and appendices.

The first section of the thesis presents the main provisions and analyzes the methods of creating applications for cost analysis and control. Existing ideas about banking counterparts are analyzed and compared with each other.

Another section of the thesis presents the sequence of program development. Describes the technology that was selected to perform the tasks, the architecture and the database selected for this project.

The third section of the thesis describes the process of program development and database to it.

The fourth section describes the analysis of working conditions, occupational safety, occupational health and industrial hygiene. The requirements of fire safety and lighting and microclimate are also described.

Based on the results of the work, conclusions and proposals were made to create a program to monitor their finances.

The bachelor's thesis contains 75 pages, 51 figures, 1 table, 25 sources used and 0 appendices.

Keywords: software, Java, Python, C #, toolbar, wrapper, handle, server, client.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

СУБД – система управління базами даних

API – application programming interface

ASP – Active Server Pages

CRUD – Create Read Update Delete

EFC – Entity Framework core

HTTP – Hypertext Transfer Protocol

MVC – model-view-controller

ПЗ – програмне забезпечення

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«ЗАСТОСУНОК ДЛЯ АНАЛІЗУ І КОНТРОЛЮ ОСОБИСТИХ ВИТРАТ ТА ФІНАНСІВ»

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 401.21810303

Виконав студент 4-го курсу, групи 401
_____ *В. П. Бондаренко*
(підпис, ініціали та прізвище)
«___» _____ 202_ р.

Керівник: *старший викладач*
(наук. ступінь, вчене звання)
_____ *В. В. Кошовий*
(підпис, ініціали та прізвище)
«___» _____ 202_ р.

Миколаїв – 2022

ВСТУП

У сучасному суспільстві взаємодія із фінансами є важливою формою нашому житті. Тож для людей це дуже є важливим отримувати та слідкувати за своє фінансове становище. Саме тому, коли світ швидко розвивається а технології стають все доступнішими за допомогою швидкого інтернету, створення фінансових програм є найдоречнішим вирішенням питанням щоб побороти фінансову неграмотність.

Найважливішою причиною для розвитку нашого суспільства є створенням таких програм, який би допоміг людям в їх фінансових становищах. Можна знайти велике різноманіття представлень о фінансовій грамотності в інтернеті, але зазвичай вони не дуже добре зроблені і найчастіше люди пишуть про фінансову грамотність коли самі нічого не розуміють у цій сфері, але вони ніяк не допоможуть тобі слідкувати за своїм фінансовим становищем.

Контроль витрат допомагає розібратися, на що ми витрачаємо гроші і чи ми не виходимо за межі власного бюджету. Ми часто не контролюємо свої витрати, особливо дрібні, а потім не розуміємо, куди пішли всі гроші. Якщо вести облік витрат, то можна визначити такі «відпливи» та підкоригувати свої витрати.

В нашому світі є дуже багато великих компаній які мають десятки тисяч професійних програмістів які роками роблять продукти, і мені дуже тяжко конкурувати із цими компаніями тим паче із банківськими кампаніями, розробкою яких займалися спеціалісти які мають досвід багато років, тому я вважаю що не треба гнатися за цими компаніями і зробив звичайну програму яка може легко виконувати свою функцію для вирішення своїх задач.

Для реалізації моєї роботи було записано та виконано декілька задач, а саме:

– проаналізував завдання моєї дипломної роботи та розробки середовища, та вибрав найкращі з них;

- проаналізовав та знайшов деякі особливості цієї програми та виявити особливості фінансових програм;
- розробити унікальний дизайн додатку;
- провести тести додатку;
- створення базу даних користувача;
- створення бекенду який буде відповідати на запити, в якому користувач може знайти всю необхідну йому інформацію та зміг її міняти по своєму бажанню;
- створення графічного інтерфейсу;
- зв’язок між банківською карткою та користувачем.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Принципи створення ПЗ

При створенні та розвитку ПЗ рекомендується застосовувати такі загальносистемні принципи:

- принцип включення, який передбачає, що вимоги до створення, функціонування та розвитку ПЗ визначаються з боку складнішої, що включає його в себе системи;

- принцип системної єдності, який у тому, що у всіх стадіях створення, функціонування та розвитку ПЗ його цілісність забезпечуватиметься зв'язками між підсистемами, і навіть функціонуванням підсистеми управління;

- принцип розвитку, який передбачає у ПЗ можливість його нарощування та вдосконалення компонентів та зв'язків між ними;

- принцип комплексності, який у тому, що ПЗ забезпечує пов'язаність обробки інформації, як окремих елементів, так всього обсягу даних загалом усім стадіях обробки;

- принцип інформаційної єдності, тобто у всіх підсистемах, засобах забезпечення та компонентах ПЗ використовуються єдині терміни, символи, умовні позначення та способи представлення;

- принцип сумісності полягає в тому, що мова, символи, коди та засоби програмного забезпечення узгоджені, забезпечують спільне функціонування всіх підсистем та зберігають відкриту структуру системи в цілому;

- принцип інваріантності визначає інваріантність підсистем та компонентів ПЗ до оброблюваної інформації, тобто є універсальними або типовими.

Технології програмування – це апробовані стратегії створення програм, що викладаються у вигляді методик з інформаційними фондами, описами проектних процедур та проектних операцій. Існують технологія структурного програмування, технологія проектування програм із раціональною структурою даних, технологія об'єктно-орієнтованого програмування, технологія

візуального програмування. У кожній із цих технологій використовується одна або кілька парадигм програмування (концепцій, систем поглядів). Останні є різними підходами до написання програм. Для кожної з них необхідні: свій тип мислення, особлива школа навчання, прийоми та способи програмування, що визначаються мовою, що використовується. Існують чотири основні парадигми, які описують більшість сьогоднішніх методів програмування: імперативна, аплікативна, заснована на системі правил та об'єктно-орієнтована. Імперативна парадигма. Ця модель випливає з особливостей апаратної частини стандартного комп'ютера, яка виконує інструкції (команди) послідовно. Основним видом абстракції, що використовується у даній парадигмі, є алгоритми. На основі її розроблено безліч операторно-орієнтованих мов програмування. Програма такою мовою складається з послідовності операторів, виконання кожного з яких спричиняє зміну значення в одному або кількох осередках пам'яті. Загалом синтаксис такої мови має вигляд:

Оператор_1:

Оператор_2:

...

Зазвичай при першому знайомстві з концепціями програмування люди стикаються саме з цією моделлю, і багато широко поширених мов підтримують саме її (наприклад, C, C++, FORTRAN, ALGOL, PL/I, Pascal, Ada, Smalltalk і COBOL).

Аплікаційна парадигма. Являє собою інший погляд на обчислення, що виробляються за допомогою мови програмування. В основі цієї парадигми покладено розгляд функції, яку виконує програма. Тут не розглядається послідовність станів, якими має пройти обчислювальна машина. Питання ставиться інакше: яку функцію необхідно застосувати до початкового стану машини (шляхом вибору початкового набору змінних та комбінування їх певним чином), щоб отримати бажаний результат?

Мови, в яких акцентований саме цей погляд на обчислення, називаються аплікативними, чи функціональними. Синтаксис такої мови, як правило, виглядає так:

Функція_n (... функція_2 (функція_1 (дані))...)

Таку модель підтримують такі мови як ML та LISP. Парадигма, заснована на системі правил. Мови, засновані на цій парадигмі, здійснюють перевірку наявності необхідної умови, що вирішує, і у разі його виявлення виконують відповідну дію. Найбільш відомою мовою, заснованою на системі правил, є Prolog. Він називається також мовою логічного програмування.

Принципи розробки ПЗ - це набір певних правил і рекомендацій, яким потрібно слідувати при написанні вихідного коду програми, якщо хочеш написати красивий, зрозумілий і редагований код. Тобто, не існує чарівної палички, за допомогою якої можна було б перетворити мішанину зі змінних, класів та методів на ідеальний лістинг, але існують деякі підказки, які допомагають програмісту визначити, чи все правильно він робить. Досить простий, але при цьому дуже корисний принцип, який говорить, що повторення одного й того ж коду в кількох місцях дуже погана ідея. Це пов'язано насамперед із необхідністю подальшої підтримки та зміни коду. Якщо якийсь певний шматок лістингу повторюється в кількох місцях програми, то велика ймовірність виникнення двох плачевних ситуацій. При необхідності внести навіть найменші виправлення у вихідний код, вам доведеться заглянути у всі місця де він використовується, що вимагатиме додаткових витрат часу та сил.

З першого пункту випливає другий, ви або інший розробник може випадково пропустити одне з виправлень і зіткнутися з помилками в роботі програми.

У зв'язку з цим є рекомендація, якщо будь-який код зустрічається в лістингу більше двох разів, його потрібно виносити в окремий метод. Це загальна рекомендація, насправді потрібно задуматися про виділення методу, навіть якщо

ви зустрічаєте повторення вдруге. Всі принципи розробки ПЗ важливі, але це особливо!

1.2 Огляд та аналіз наявних аналогів

На сьогоднішній день фінансові програми які можуть відслідковувати ваш фінансовий стан – це банківські компанії, особливо коли використовуєш їх банківську карту. Таких банків які мають цю функцію дуже багато, але не всі вони зроблені добре, а ті якими досить комфортно користуватися ви можете додати тільки банківську карту того банку. Зазвичай ви не можете добавити банківську карту іншого банку і в цьому не зручність.

Ще основне, якою якості цієї програма. Бо якщо стоятиме вибір між двома майже однаковими програмами, то користувач візьме найбільш зручну програму для нього. Більшість користувачів надають перевагу програмі, в якому буде мати підтримку мов, а саме російську мову і англійську, ще один із найголовніших критеріїв це простий та інтуїтивний інтерфейс. Інтерфейс має бути привабливим та зрозумілим. Ну і найважливіше, щоб у програма мала попит, бо нікому не потрібна програма яка не потрібна людству. Зараз я наведу декілька критеріїв та поясню кожен із них.

Спочатку приведемо приклад доступності, а саме це мається на увазі пошуку потрібній інформації. Це дуже важливо для кожній програми бо користувач повинен легко знаходити будь яку інформацію. Також кожен зможе знаходити інформацію незважаючи на вік, з людьми обмеженими можливостями і так далі. Також доступність – це добре зроблена програма в якій користувач може зробити збільшення тексту(якщо у вас поганий зір) або зовсім зробити так, щоб текст озвучили (якщо є проблеми зі слухом)ю. Це все потрібно зробити для доступного користування програмою. Сьогодні треба пам’ятати, що в інтернеті є дуже багато інформації і саме через те плагіату не можливо уникнути. Будь яка інформація повинна шифруватися або подавання цього документу для можливості дублікату тексту з неможливістю його видалення. Програма має бути написана зрозуміло, щоб користувач зміг легко знаходити йому потрібну

інформацію. Зробити це потрібно так, щоб основне поле пошуку було добре видно у головного вікна програми. Також требо зробити посилання для зручного користування для знаходження інформації. Це все треба зробити, щоб користувач не зміг заблудитися в програмі і тим паче не викликало ніяких незручностей. Все це дозволить використовувати додаток як добре продуманий інструмент як і в серйозних закладах, так і в суспільстві.

Також немало важливим є й дизайнерське рішення оформлення додатку, якщо програма оформлена дуже кольоровим та із різким забарвленням, то це сильно буде відштовхувати користувача, тим паче причиняти дискомфорт та іноді може погано впливати на здоров'я читача. Якщо не зауважити на це увагу на оформлення сайтів в яких дуже погано проведено комбінації кольорів, то можна не чекати на успіх програми. Саме тому важливо триматися колористики в розроблених програм, бо саме це і формує початок успіху програми, бо якщо у конкурента буде схожа ідея а оформлення буде ніяке, то успіх у конкурента буде невеликим або його зовсім не буде. Дуже важливо, щоб усі вікна програми були одного колориту та дизайну, бо саме це впливає на сприйняття користувача. Та й для користувача буде краще запам'ятає цей продукт по кольорам. Саме це і буде ключовим в створені цієї програми. Звісно якщо це говориться про створювача, який хоче аби його програма зазнала величезного успіху.

1.3 Постановка задачі

Актуальність. В нашому сучасному світі із швидким інтернетом та новими технологіями програмне забезпечення становлять невід'ємну частину життя кожної людини. За допомогою сучасних технологій ми можемо легко написати програму яка відслідковує ваші фінанси. Як фінансова грамотність впливає на людину та її життя? Кожного для ми приймаємо десятки фінансових рішень, проте досить велика кількість людей не замислюється над питанням, як часто ми слідкуємо за нашими грошима і як вона впливає на нас? Чи є у нас знання щоб правильно керувати власними грошима? Було проведено дослідження яке

показує, що фінансова грамотність українців становить 11.6 пунктів з 21. Це досить не неплохий результат але можна краще. Саме тому я вирішив написати програму яка допоможе кожній людині краще розуміти куди витрачається гроші і на що. Це допоможе людям краще розбиратися в своїй фінансовій грамотності. Це має покращити життя всіх людей, бо наш цивілізований світ складається з економіки.

Об’єкт роботи – створення програми яка здатна відслідковувати та аналізувати своє фінансове становище.

Предмет роботи – принципи та технології побудови бази даних в яку будуть приходити дані із банківської картки.

Метою роботи є автоматизація та спрощення програми, в якому користувач може користуватися на інтуїтивному рівні.

Завдання для реалізації цілей:

1. Огляд та аналіз сучасного стану задачі та створення програми для фінансової грамотності.

2. Дослідження технологій та алгоритмів для вирішення декількох задач для домашнього користування.

3. Створення графічного інтерфейсу та базу даних для фінансів на мові Java,

Етапи які допоможуть вирішити поставлену задачу:

- створення базу даних користувача;
- створення бекенду який буде відповідати на запити, створити структуру запитів які допоможуть допомагати користувачу знайти необхідну інформацію;
- створення графічного інтерфейсу;
- зв’язок між банківською карткою та користувачем.

Висновки до розділу 1

Тема БКР присвячена розробці програми яка має аналізувати та показувати свої фінансові витрати. В першому розділі було розглянута актуальність та свіжість ідеї для застосунку та її аналогів.

Метою є створення програми фінансової грамотності з використанням бази даних за допомогою сучасних інструментів та технологій, який може буде використовуватися будь-ким. А також для задоволення своїх потреб.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ТА СТВОРЕННЯ ДОДАТКУ

2.1 Обрання мови програмування

Спочатку треба вибрати мову програмування для написання застосунку. Було обрано найпопулярніші та найбільш цікаві мови програмування:

- TypeScript;
- Python;
- PHP;
- JavaScript;
- Java;
- C#;
- C++.

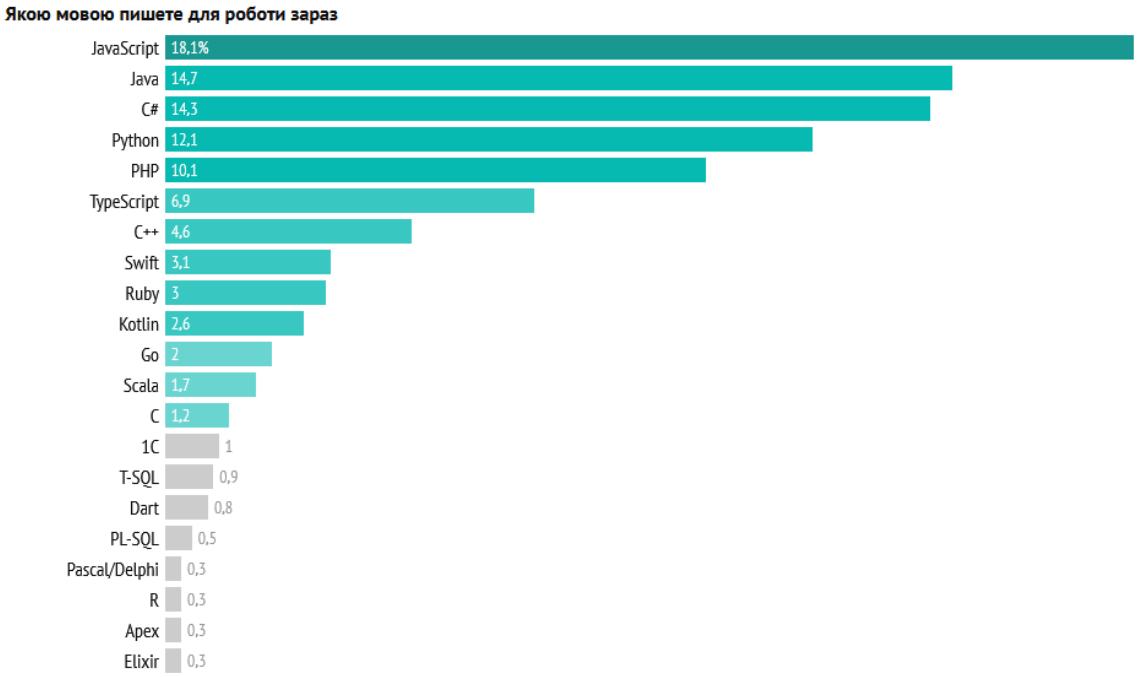

Рисунок 2.1 - Топ мов програмування на 2022 рік

До написання додатку було проведено аналіз в якому було проведено порівняння між усіма найпопулярнішими мовами програмування.

C++ та Java

Якщо порівнювати їх за всіма категоріями то можна помітити, що Java набагато зручніша за C++. Java дуже просто можна зробити класи та підкласи, інтерфейси і має дуже велику підтримку бібліотек. Взагалі Java дуже проста мова програмування ніж C++, бо на C++ треба підключати багато бібліотек та пам'ятати багато речей. Це дуже складна мова програмування. На C++ найчастіше пишуть програми низьких рівнях. Це програми які використовують у забезпечення автопілоту автомобіля або управління літака, тощо. Але для додатку фінансів вона не дуже підходить. Зрозуміло, що можна написати але ця програма буде в рази складніша та не зовсім оправдана. Саме тому було прийнято рішення не використовувати його для вирішення моїх задач.

Python против Java, C #

Для початку є дві типізовані мови – це статична та динамічна. Як всім відома, що статична типізована мова має більше переваг ніж динамічна.

Але треба розуміти, що ці типи допомагає програмістам запобігати велику кількість помилок. Зазвичай це запобігання звичайних орфографічних помилок.

У статична типізація є декілька переваг, а саме це підтримка IDE, підтримка плагінів, рефакторинг та генерація коду. В C# найкращий плагін рефакторингу є Re-Sharper. Це найкращий плагін для розроблення статичної структури.

Найкращий інструмент в C# є NCover. Це інструмент для модульного тестування коду.

Один із найкращих IDE є Visual Studio. В ній можна використовувати багато інших інструментів і мов програмування от C++ до C#.

C# або Java?

C# та Java дуже схожі між собою, але на мою думку Java набагато краще, тому що у неї є багато переваг для моєї програми. Так буде комфортніше для написання додатку.

Але C# - це об'єктно-орієнтована мова програмування яка була розроблена компанією Microsoft. Ще C# називают CSharp та має ініціативу .NET.

C# є однією з мов програмування мовної інфраструктури. C# дуже сильно схожий синтаксично Java і простий для користувача, особливо ті хто знає мови програмування на C, C ++ і Java.

C# підходить якщо ви розробляєте на платформи Windows. Для цієї платформи більшість програм йдуть за замовчуванням. Тому це підходить тим людям які полюбляють .Net та Windows.

Java та Python

Це схожі між собою мови програмування але в даному проекті я візьму Java Чому я не обрав Python? Тому що у Python є один серйозний мінус, і це на порядок повільніше працює ніж Java. Це дуже важливий аспект, бо Python треба набагато більше часу для оброблення даних. Тому, якщо писати велику програму то треба розуміти, що краще використовувати не Python.

Але у Python є одна велика перевага. І це графічний інтерфейс. В Python дуже класно і інтуїтивно зрозуміло використання різних інструментів. Тому якщо писати якісь інтерфейси то було обрано би Python, але в моїй програмі потрібно робити декілька розрахунків. І це важливо для мене.

Python в основному використовується для виконання спам повідомлень та розробки штучного інтелекту. Ця мова має дуже багато різними фреймворками, як Flask, Django, Pyramid і CherryPy, завдяки чому розробка стає дуже ефективною.

Кожна мова програмування має свої недоліки та переваги, але для кваліфікаційної робити я обрав лише одну Java. Вона найбільш зручніша для мене і моєї роботи, я уже використовував її та в повній мірі можу розкрити їх потенціал.

2.3 Що таке Entity Framework Core

Entity Framework Core – це платформа яка є полегшеною, розширюваною, відкритою між платформна версія популярної технології доступу до даних Entity Framework.

Рисунок 2.2 - Entity Framework Core

Entity Framework Core дозволяє працювати з базами даних, а також має більш високий рівень абстракції. Якщо на фізичному рівні ми оперуємо таблицями, індексами та ключами, але на концептуальному рівні, який нам пропонує Entity Framework Core ми вже працюємо з об'єктами.Entity Framework Core дозволяє абстрагуватися від самої бази даних та її таблиць і працювати з даними незалежно від типу сховища.

Entity Framework Core має підтримку декілька різних систем баз даних. Тому, ми можемо через EF Core працювати з будь-якої СУБД, тим паче якщо є потрібний провайдер.

Сьогодні в Microsoft є декілька вбудованих провайдерів: для роботи з MS SQL Server, для PostgreSQL для SQLite. Також є провайдери від інших різних сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. Це дуже зручно коли нам треба замінити базу даних, то можна буде легко замінити конфігурацію підключення до бази даних. А сам код не треба міняти під час цієї операції.

EF Core може бути як об'єктно-реляційним відображенням, а саме це:

– дозволяє програмістам працювати з різними базами даних за допомогою об’єктів EF;
– усуває проблему в якому не треба відкривати великий код і мати доступ до даних.

Entity Framework Core - це одна із найголовніших бібліотек, яка може забезпечити доступ до бази даних. Є декілька способів для роботи із цією бібліотекою. Один із них це об’єктно-реляційний мапер (O / RM). O / RM працюють за допомогою переносу одних даних між двома елементами: реляційною базою даних зі своїм власним API та об’єктно-орієнтованим програмним прикладом класів та програмного коду. Основна перевага цієї бібліотеки є дозвіл програмістів написання та забезпечення програми для швидкого доступу коду до баз даних.

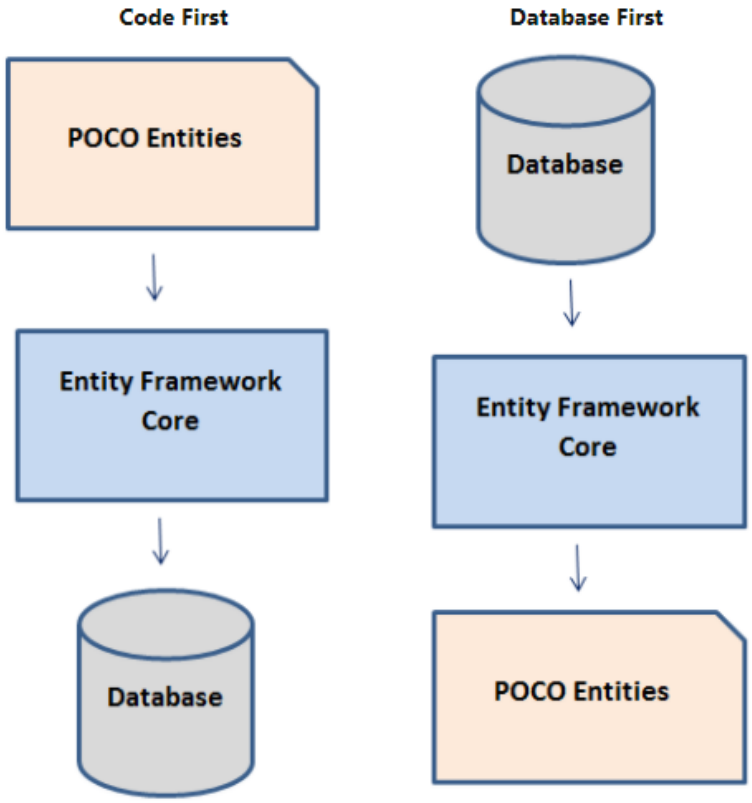

Рисунок 2.3 - Дизайн робочих процесів Entity Framework Core

Entity Framework Core зручним тим, що здійснює доступ до даних за допомогою моделей. Ці моделі складаються із декілька класів та об’єктів. Вони представляють між собою об’єднання між базами даними які можуть представляти собою запити та зберігання даних.

Розробки моделі EF:

- Створення класів які входять в моделі із існуючої бази даних;
- Створення вручну модель відповідно до бази даних;
- Використання бази даних після створення моделі, для створення бази даних із моделі.

2.4 Swagger

Swagger – це інструмент в якому є відкритий елемент із кодом, в якому може допомагати у розробці API створюючи документацію, та виклику кінцевих пунктів API і швидкий доступ.

Рисунок 2.4 - Swagger

Даний інструмент дозволяє описати структуру API, щоб машини могли їх читати. Здатність API описувати власну структуру моделі – все це є у Swagger. Це дуже важливо, бо прочитавши структуру API, Swagger може автоматично створювати красиву та інтерактивну документацію API. Swagger також може автоматично створювати клієнтські бібліотеки для API із різними мовами та досліджувати різні можливості, такі як автоматичне тестування. Swagger робить це, просячи API повернути YAML або JSON, який містить детальний опис всього

API. Цей файл є по суті списком ресурсів API, який відповідає особливій специфікації OpenAPI.

Специфікація має включати таку інформацію, як:

- операції які запобігають та підтримують API;
- параметри API і що вони повертають;
- наявність авторизації у API;
- умови, контактна інформація та ліцензія на використання API.

Swagger має широкий список інструментів таких як “ Editor”. Це один із інструментів які допоможуть нам знайти деякі помилки в нашій документації на YAML або JSON. Треба просто відкрити та завантажити нашу документацію на платформу і вона порівняє документацію із специфікаціями Swagger Editor. Він допоможе знайти помилки які були допущені а також запропонує альтернативи для того, щоб наша документація була без помилок.

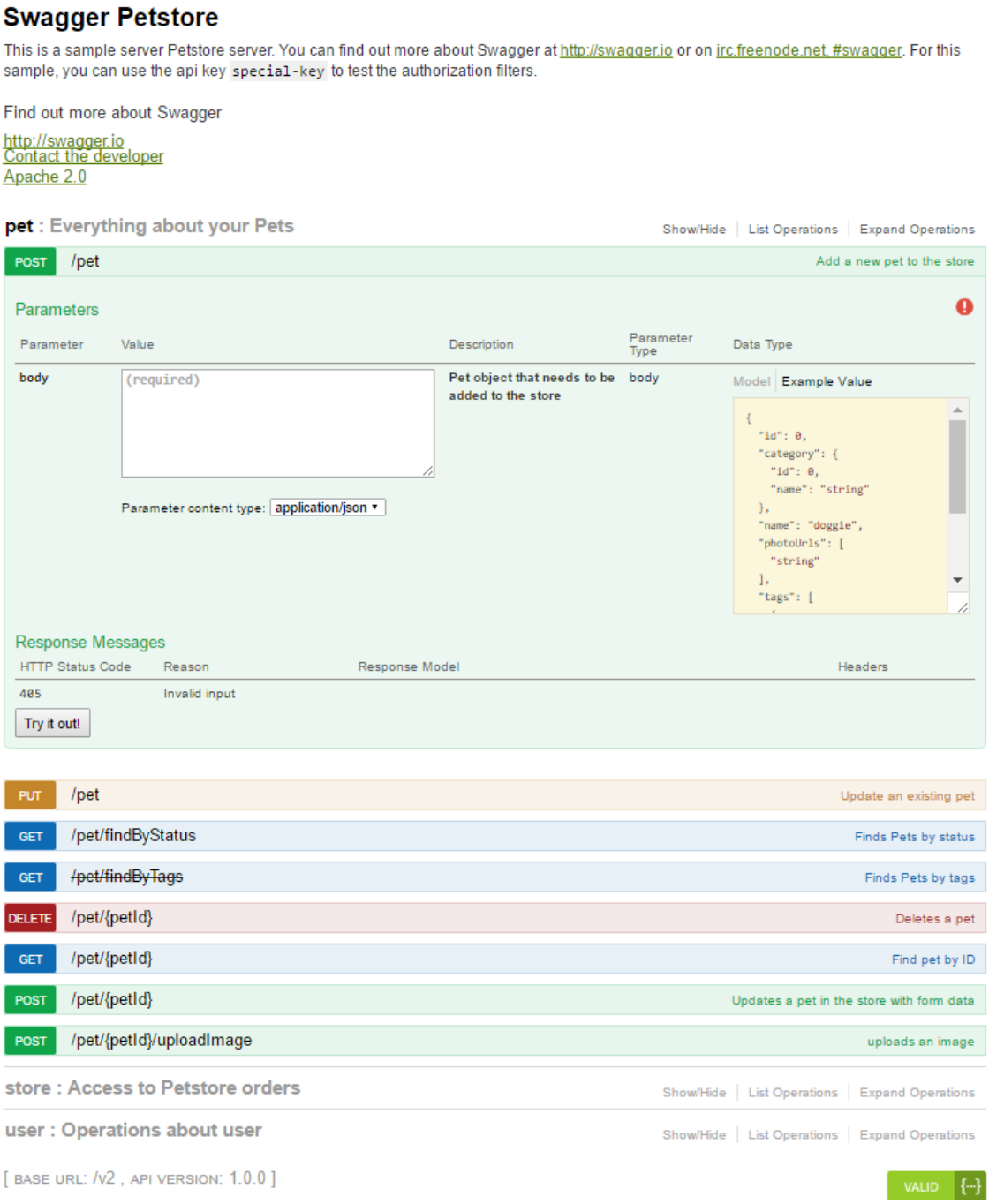

Рисунок 2.5 - Приклад роботи Swagger

Swagger для API можна було написати вручну, або зробити це все автоматично згенеровано з анотацій у вихідному коді.

Висновки до розділу 2

Під час цієї роботи над розділом було продемонстровано список програмних мов які допоможуть створити нашу програму. Також було виконано детальний аналіз популярних мов програмування відносно виконання поставленої задачі.

Для реалізації додатку було обрано написати програму с нуля на мові Java використовуючи клас Handler, Filter, Wrapper, SaveData та Swagger, за допомогою середовища розробки NetBeans та IntelliJ IDEA .

Клієнтська частина буде реалізована на мові Java, за допомогою середовища розробки IntelliJ IDEA, з використанням таких бібліотек, як:

```
import java.awt.Toolkit;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.KeyStroke;
import personalfinance.gui.EnableEditDelete;
import personalfinance.gui.MainFrame;
import personalfinance.gui.Refresh;
import personalfinance.gui.handler.Handler;
import personalfinance.gui.handler.MenuEditHandler;
import personalfinance.gui.handler.MenuFileHandler;
import personalfinance.gui.handler.MenuHelpHandler;
import personalfinance.gui.handler.MenuSettingsHandler;
import personalfinance.gui.handler.MenuViewHandler;
import personalfinance.settings.Settings;
import personalfinance.settings.Style;
import personalfinance.settings.Text;
```

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура проекту

Даний проект складається із 7х частин:

- створення моделі. Це створення проекту, класів с текстовими константами, створення базових класів, створення класів для об’єктів, створення фільтра, створення класів для роботи зі статистикою, створення класів основних характеристик стиля, створення форматування даних та створення класів с кодами для обробки подій;

- загрузка та збереження даних. Це створення класу Wrapper, створення механізмів загрузки та збереження даних, створення класу SaveData, додавання тестових даних, обновлення курсу валют;

- створення базового інтерфейсу користувача. А саме це вивід основного вікна, створення основних інтерфейсів, створення меню, створення класу кнопки, створення тулбара, створення тулбара с функціями, створення компонент з вибором дати та вибором файлу;

- діалогові вікна. Це створення вікна з інформацією о помилки, створення вікна із підтвердженими операціями, створення вікна “О программе”, створення абстрактного вікна для редагування даних, створення вікна для редагування рахунку, створення вікна для редагування статей, створення вікна для редагування транзакції, створення вікна для редагування переводів, створення вікна для редагування валют;

- вивід даних. А саме це вивід лівої панелі с балансом, створення класу для правої панелі, створення моделі таблиці, створення класу таблиці, вивід таблиці з останніми переводами, вивід таблиці із рахунками, вивід

таблиці зі статтями, вивід таблиці зі переводами, вивід таблиці зі транзакціями, вивід таблиці з валютами, створення меню для таблиці, створення та вивід панелі фільтрів;

- вивід статистики. Створення класу для виводу гістограми, вивід панелі зі гістограмою, створення панелі для перемикання типу статистики;

- обробка подій. Це обробка меню “Файл”, обробка подій тулбару із функціями, обробка меню “Правка”,”Вид” та ”Помощь”. Обробка основного тулбару, обробка події фільтра, панелі вибору типу статистики та головного вікна. А також обробка події меню та діалогового вікна.

3.2 Принцип роботи програми

3.2.1 Створення моделі

Спочатку ми створимо базовий клас

```
package personalfinance.model;

import personalfinance.saveload.SaveData;


abstract public class Common {

    public Common() {}

    public String getValueForComboBox() { return null; }

    public void postAdd(SaveData sd) {}
    public void postEdit(SaveData sd) {}
    public void postRemove(SaveData sd) {}

}
```

Рисунок 3.1 - Створення базового класу

Потім ми створимо фільтр щоб краще бачити своє фінансове становищ

```java
package personalfinance.model;

import ...


public class Filter {

    public static final int STEP_DAY = 0;
    public static final int STEP_MONTH = 1;
    public static final int STEP_YEAR = 2;

    private int step;
    private Date from;
    private Date to;

    public Filter() { this(STEP_MONTH); }

    public Filter(int step) {
        this.step = step;
        setFromTo(new GregorianCalendar());
    }

    public int getStep() { return step; }

    public Date getFrom() { return from; }

    public Date getTo() { return to; }

    public void next() { offset(1); }
```

Рисунок 3.2 - Створення фільтру(1)

```java
public void prev() { offset(-1); }

public void nextPeriod() {
    step += 1;
    if (step > STEP_YEAR) step = STEP_DAY;
    setFromTo(new GregorianCalendar());
}

public boolean check(Date date) { return (date.compareTo(from) > 0) && (date.compareTo(to) < 0); }

private void setFromTo(Calendar calendar) {
    switch (step) {
        case STEP_DAY:
            this.from = new GregorianCalendar(
                    calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH),
                    0, 0, 0).getTime();
            this.to = new GregorianCalendar(
                    calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH),
                    23, 59, 59).getTime();
            break;
        case STEP_MONTH:
            YearMonth yearMonth = YearMonth.of(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH) + 1);
            this.from = new GregorianCalendar(
                    calendar.get(Calendar.YEAR),
                    calendar.get(Calendar.MONTH),
                    1,
                    0, 0, 0).getTime();
```

Рисунок 3.3 - Створення фільтру(2)

```
        this.to = new GregorianCalendar(
                calendar.get(Calendar.YEAR),
                calendar.get(Calendar.MONTH),
                yearMonth.lengthOfMonth(),
                23, 59, 59).getTime();
        break;
    case STEP_YEAR:
        this.from = new GregorianCalendar(
                calendar.get(Calendar.YEAR),
                0,
                1,
                0, 0, 0).getTime();
        this.to = new GregorianCalendar(
                calendar.get(Calendar.YEAR),
                11,
                31,
                23, 59, 59).getTime();
```

Рисунок 3.4 - Створення фільтру(3)

```
private void offset(int i) {
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(from);
    switch (step) {
        case STEP_DAY:
            calendar.add(Calendar.DAY_OF_MONTH, i);
            break;
        case STEP_MONTH:
            calendar.add(Calendar.MONTH, i);
            break;
        case STEP_YEAR:
            calendar.add(Calendar.YEAR, i);
    }
    setFromTo(calendar);
}
```

Рисунок 3.5 - Створення фільтру(4)

Наступний крок буде написання коду для спостереження статистики своїх витрат

```java
public class Statistics {

    public static double getBalanceCurrency(Currency currency) {
        SaveData sd = SaveData.getInstance();
        double amount = 0;
        for (Account account : sd.getAccounts()) {
            if (currency.equals(account.getCurrency())) amount += account.getAmount();
        }
        return amount;
    }

    public static double getBalance(Currency currency) {
        SaveData sd = SaveData.getInstance();
        double amount = 0;
        for (Account account : sd.getAccounts()) {
            amount += account.getAmount() * account.getCurrency().getRateByCurrency(currency);
        }
        return amount;
    }

    public static HashMap<String, Double> getDataForChartOnIncomeArticles() { return getDataForChartOnArticles(true); }

    public static HashMap<String, Double> getDataForChartOnExpArticles() { return getDataForChartOnArticles(false); }

    private static HashMap<String, Double> getDataForChartOnArticles(boolean income) {
        List<Transaction> transactions = SaveData.getInstance().getFilterTransactions();
        HashMap<String, Double> data = new HashMap();
        for (Transaction t : transactions) {
            if ((income && t.getAmount() > 0) || (!income && t.getAmount() < 0)) {
                String key = t.getArticle().getTitle();
                double summa = 0;
                double amount = t.getAmount();
```

Рисунок 3.6 - Створення класу статистики(1)

```java
                if (!income) amount *= -1;
                if (data.containsKey(key)) summa = data.get(key);
                summa += amount * t.getAccount().getCurrency().getRateByCurrency(SaveData.getInstance().getBaseCurrency())
                data.put(key, round(summa));
            }
        }
        return data;
    }

    private static double round(double value) { return (double) Math.round(value * 100) / 100; }

}
```

Рисунок 3.7 - Створення класу статистики(2)

Це був перший крок в якому було створені базові класи для того щоб можна було підготувати програму для наступного кроку.

3.2.2 Загрузка та збереження даних.

Для початку ми створимо клас Wrapper. Клас Wrapper надає механізм для перетворення примітивних типів даних на об’єкти класу-оболонки. Нижче наведено еквівалентні об’єкти класів-оболонок примітивних типів даних.

```
public Wrapper() {
}

@{...}
public List<Article> getArticles() { return this.articles; }

public void setArticles(List<Article> articles) { this.articles = articles; }

@XmlElement(
    name = "accounts"
)
public List<Account> getAccounts() { return this.accounts; }

public void setAccounts(List<Account> accounts) { this.accounts = accounts; }

@XmlElement(
    name = "transactions"
)
public List<Transaction> getTransactions() { return this.transactions; }

public void setTransactions(List<Transaction> transactions) { this.transactions = transactions; }

@XmlElement(
    name = "transfers"
)
public List<Transfer> getTransfers() { return this.transfers; }

public void setTransfers(List<Transfer> transfers) { this.transfers = transfers; }
```

Рисунок 3.8 - Клас Wrapper

Далі було створено клас для загрузки та збереження даних. Дані будуть загружатися із файлу а потів цю інформацію клас буде зберігати.

```
public class SaveLoad {
    public SaveLoad() {
    }

    public static void load(SaveData sd) {
        try {
            JAXBContext context = JAXBContext.newInstance(new Class[]{Wrapper.class});
            Unmarshaller um = context.createUnmarshaller();
            Wrapper wrapper = (Wrapper)um.unmarshal(Settings.getFileSave());
            sd.setAccounts(wrapper.getAccounts());
            sd.setArticles(wrapper.getArticles());
            sd.setTransactions(wrapper.getTransactions());
            sd.setTransfers(wrapper.getTransfers());
            sd.setCurrencies(wrapper.getCurrencies());
        } catch (JAXBException var4) {
            System.out.println("Файл не существует!");
        }

    }
```

Рисунок 3.9 - Створення класу загрузки

```
public static void save(SaveData sd) {
    try {
        JAXBContext context = JAXBContext.newInstance(new Class[]{Wrapper.class});
        Marshaller m = context.createMarshaller();
        m.setProperty("jaxb.formatted.output", true);
        Wrapper wrapper = new Wrapper();
        wrapper.setAccounts(sd.getAccounts());
        wrapper.setArticles(sd.getArticles());
        wrapper.setTransactions(sd.getTransactions());
        wrapper.setTransfers(sd.getTransfers());
        wrapper.setCurrencies(sd.getCurrencies());
        m.marshal(wrapper, Settings.getFileSave());
    } catch (JAXBException var4) {
        Logger.getLogger(SaveLoad.class.getName()).log(Level.SEVERE, (String)null, var4);
    }
```

Рисунок 3.10 - Створення класу зберігання

Наступним етапом буде створення оновлення курсу валют. Чому ми його не вставили відразу? Це потрібно зробити тому що ці дані змінюється кожен день, бо сьогодні курс долару може бути 28 гривні, а завтра вже 29 гривні. Це можна легко зробити. Треба тільки оставити посилання на сайт курсу валют, і якщо дані змінюються то в програмі теж буде інший курс валют як і на сайті.

```
public RateCurrency() {
}

public static HashMap<String, Double> getRates(Currency base) throws Exception {
    HashMap<String, NodeList> result = new HashMap();
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
    String url = "http://www.cbr.ru/scripts/XML_daily.asp?date_req=" + dateFormat.format(new Date());
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    Document doc = factory.newDocumentBuilder().parse((new URL(url)).openStream());
    NodeList nl = doc.getElementsByTagName("Valute");

    for(int i = 0; i < nl.getLength(); ++i) {
        Node node = nl.item(i);
        NodeList nlChilds = node.getChildNodes();

        for(int j = 0; j < nlChilds.getLength(); ++j) {
            if (nlChilds.item(j).getNodeName().equals("CharCode")) {
                result.put(nlChilds.item(j).getTextContent(), nlChilds);
            }
        }
    }

    HashMap<String, Double> rates = new HashMap();
    Iterator var17 = result.entrySet().iterator();
```

Рисунок 3.11 - Посилання на сайт курсу валют

3.2.3 Створення базового інтерфейсу

Наступним кроком буде створення меню для програми. Так же програма буде на двох мовах, російською та англійською мовою. Було створено головне меню та інтерфейс. Також було створено кнопки та основний тулбар. В тулбар

було добавлено декілька функцій для зручної роботи. Ще було створено компонент який дає змогу вибирати дату та файл в який ця інформація буде зберігатися.

```
public class MainMenu extends JMenuBar implements Refresh, EnableEditDelete {
    private JMenuItem menuEdit;
    private JMenuItem menuDelete;
    private final MainFrame frame;

    public MainMenu(MainFrame frame) {
        this.frame = frame;
        this.init();
    }

    private void init() {
        JMenu file = new JMenu(Text.get("MENU_FILE"));
        JMenu edit = new JMenu(Text.get("MENU_EDIT"));
        JMenu view = new JMenu(Text.get("MENU_VIEW"));
        JMenu settings = new JMenu(Text.get("MENU_SETTINGS"));
        JMenu help = new JMenu(Text.get("MENU_HELP"));
        file.setIcon(Style.ICON_MENU_FILE);
        edit.setIcon(Style.ICON_MENU_EDIT);
        view.setIcon(Style.ICON_MENU_VIEW);
        settings.setIcon(Style.ICON_MENU_SETTINGS);
        help.setIcon(Style.ICON_MENU_HELP);
        this.add(file);
        this.add(edit);
        this.add(view);
        this.add(settings);
        this.add(help);
```

Рисунок 3.12 - Створення меню(1)

```
    MenuFileHandler fileHandler = new MenuFileHandler(this.frame);
    MenuEditHandler editHandler = new MenuEditHandler(this.frame);
    MenuViewHandler viewHandler = new MenuViewHandler(this.frame);
    MenuSettingsHandler settingsHandler = new MenuSettingsHandler(this.frame);
    MenuHelpHandler helpHandler = new MenuHelpHandler(this.frame);
    this.addMenuItem(file, fileHandler, Text.get("MENU_FILE_NEW"), Style.ICON_MENU_FILE_NEW,  action: "MENU_FILE_NEW",  key: 78);
    this.addMenuItem(file, fileHandler, Text.get("MENU_FILE_OPEN"), Style.ICON_MENU_FILE_OPEN,  action: "MENU_FILE_OPEN",  key: 79);
    this.addMenuItem(file, fileHandler, Text.get("MENU_FILE_SAVE"), Style.ICON_MENU_FILE_SAVE,  action: "MENU_FILE_SAVE",  key: 83);
    this.addMenuItem(file, fileHandler, Text.get("MENU_FILE_UPDATE_CURRENCIES"), Style.ICON_MENU_FILE_UPDATE_CURRENCIES,  action: "MENU_FILE_UPDATE_CURRENCIES");
    this.addMenuItem(file, fileHandler, Text.get("MENU_FILE_EXIT"), Style.ICON_MENU_FILE_EXIT,  action: "MENU_FILE_EXIT");
    this.addMenuItem(edit, editHandler, Text.get("MENU_EDIT_ADD"), Style.ICON_MENU_EDIT_ADD,  action: "MENU_EDIT_ADD");
    this.menuEdit = this.addMenuItem(edit, editHandler, Text.get("MENU_EDIT_EDIT"), Style.ICON_MENU_EDIT_EDIT,  action: "MENU_EDIT_EDIT");
    this.menuDelete = this.addMenuItem(edit, editHandler, Text.get("MENU_EDIT_DELETE"), Style.ICON_MENU_EDIT_DELETE,  action: "MENU_EDIT_DELETE");
    this.menuEdit.setEnabled(false);
    this.menuDelete.setEnabled(false);
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_OVERVIEW"), Style.ICON_MENU_VIEW_OVERVIEW,  action: "MENU_VIEW_OVERVIEW");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_ACCOUNTS"), Style.ICON_MENU_VIEW_ACCOUNTS,  action: "MENU_VIEW_ACCOUNTS");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_ARTICLES"), Style.ICON_MENU_VIEW_ARTICLES,  action: "MENU_VIEW_ARTICLES");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_TRANSACTIONS"), Style.ICON_MENU_VIEW_TRANSACTIONS,  action: "MENU_VIEW_TRANSACTIONS");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_TRANSFERS"), Style.ICON_MENU_VIEW_TRANSFERS,  action: "MENU_VIEW_TRANSFERS");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_CURRENCIES"), Style.ICON_MENU_VIEW_CURRENCIES,  action: "MENU_VIEW_CURRENCIES");
    this.addMenuItem(view, viewHandler, Text.get("MENU_VIEW_STATISTICS"), Style.ICON_MENU_VIEW_STATISTICS,  action: "MENU_VIEW_STATISTICS");
    this.addMenuItem(help, helpHandler, Text.get("MENU_HELP_ABOUT"), Style.ICON_MENU_HELP_ABOUT,  action: "MENU_HELP_ABOUT");
    JMenu language = new JMenu(Text.get("MENU_SETTINGS_LANGUAGE"));
    language.setIcon(Style.ICON_MENU_SETTINGS_LANGUAGE);
    settings.add(language);
    ButtonGroup group = new ButtonGroup();
    JRadioButtonMenuItem menuRussian = new JRadioButtonMenuItem(Text.get("MENU_SETTINGS_LANGUAGE_RUSSIAN"));
    JRadioButtonMenuItem menuEnglish = new JRadioButtonMenuItem(Text.get("MENU_SETTINGS_LANGUAGE_ENGLISH"));
    group.add(menuRussian);
    group.add(menuEnglish);
```

Рисунок 3.13 - Створення меню(2)

```
    group.add(menuEnglish);
    menuRussian.setIcon(Style.ICON_MENU_SETTINGS_LANGUAGE_RUSSIAN);
    menuEnglish.setIcon(Style.ICON_MENU_SETTINGS_LANGUAGE_ENGLISH);
    menuRussian.setActionCommand("MENU_SETTINGS_LANGUAGE_RUSSIAN");
    menuEnglish.setActionCommand("MENU_SETTINGS_LANGUAGE_ENGLISH");
    menuRussian.addActionListener(settingsHandler);
    menuEnglish.addActionListener(settingsHandler);
    if (Settings.getLanguage().equals("ru")) {
        menuRussian.setSelected(true);
    } else if (Settings.getLanguage().equals("en")) {
        menuEnglish.setSelected(true);
    }

    language.add(menuRussian);
    language.add(menuEnglish);
}

private JMenuItem addMenuItem(JMenu menu, Handler listener, String title, ImageIcon icon, String action, int key) {
    JMenuItem item = new JMenuItem(title);
    item.setIcon(icon);
    item.setActionCommand(action);
    item.addActionListener(listener);
    if (key != 0) {
        KeyStroke shortKey = KeyStroke.getKeyStroke(key, Toolkit.getDefaultToolkit().getMenuShortcutKeyMask());
        item.setAccelerator(shortKey);
    }

    menu.add(item);
    return item;
}
```

Рисунок 3.14 - Створення меню(3)

```
private JMenuItem addMenuItem(JMenu menu, Handler listener, String title, ImageIcon icon, String action) {
    return this.addMenuItem(menu, listener, title, icon, action, key: 0);
}

public void setEnableEditDelete(boolean enable) {
    this.menuEdit.setEnabled(enable);
    this.menuDelete.setEnabled(enable);
}

public void refresh() {
    this.removeAll();
    this.init();
}
```

Рисунок 3.15 - Створення меню(4)

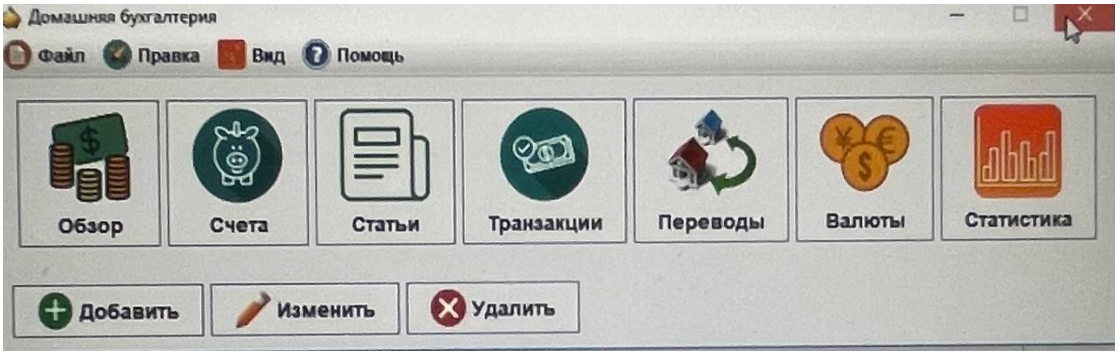

Рисунок 3.16 - Вигляд меню

3.2.4 Діалогове вікно.

Діалогове вікно - це елемент інтерфейсу користувача, тип вікна, яке використовується для забезпечення зв'язку та взаємодії між користувачем та програмною програмою. Діалогове вікно з'являється тоді, коли програмі потрібно або терміново давати інформацію про користувача, яка передбачає переривання, наприклад, коли виникає помилка, або якщо програма вимагає негайного введення або прийняття рішення від користувача, наприклад, коли програма закривається і потребує щоб знати, чи потрібно внести зміни, чи ні. Саме тому ми добавимо діалогове вікно.

```
public abstract class AddEditDialog extends JDialog {
    private final MainFrame frame;
    protected LinkedHashMap<String, JComponent> components = new LinkedHashMap();
    protected LinkedHashMap<String, ImageIcon> icons = new LinkedHashMap();
    protected LinkedHashMap<String, Object> values = new LinkedHashMap();
    protected Common c;

    public AddEditDialog(MainFrame frame) {
        super(frame, Text.get("ADD"), true);
        this.frame = frame;
        this.addWindowListener(new AddEditDialogHandler(frame, this));
        this.setResizable(false);
    }

    public Common getCommon() { return this.c; }

    public void setCommon(Common c) { this.c = c; }

    public final void showDialog() {
        this.setDialog();
        this.setVisible(true);
    }
```

Рисунок 3.17 - Створення діалогового вікна

```
public final void closeDialog() {
    this.setVisible(false);
    this.c = null;
    this.components.clear();
    this.icons.clear();
    this.values.clear();
    this.dispose();
}

public boolean isAdd() { return this.c == null; }

public abstract Common getCommonFromForm() throws ModelException;

protected abstract void init();

protected abstract void setValues();
```

Рисунок 3.18 - Добавлена функція закривання діалогового вікна

```
private void setDialog() {
    this.init();
    if (this.isAdd()) {
        this.setTitle(Text.get("ADD"));
        this.setIconImage(Style.ICON_ADD.getImage());
    } else {
        this.setValues();
        this.setTitle(Text.get("EDIT"));
        this.setIconImage(Style.ICON_EDIT.getImage());
    }

    this.getContentPane().removeAll();
    this.getContentPane().setLayout(new BoxLayout(this.getContentPane(), 1));
    ((JPanel)this.getContentPane()).setBorder(Style.BORDER_DIALOG);
    Iterator var1 = this.components.entrySet().iterator();

    while(var1.hasNext()) {
        Entry<String, JComponent> entry = (Entry)var1.next();
        String key = (String)entry.getKey();
        JLabel label = new JLabel(Text.get(key));
        label.setIcon((Icon)this.icons.get(key));
        label.setFont(Style.FONT_DIALOG_LABEL);
        JComponent component = (JComponent)entry.getValue();
        if (component instanceof JTextField) {
            component.setPreferredSize(Style.DIMENSION_DIALOG_TEXTFIELD_SIZE);
            if (this.values.containsKey(key)) {
                ((JTextField)component).setText("" + this.values.get(key));
            }
        } else if (component instanceof JComboBox) {
            if (this.values.containsKey(key)) {
                ((JComboBox)component).setSelectedItem(this.values.get(key));
```

Рисунок 3.19 - Створення настройки діалогового вікна(1)

```
            }
        } else if (component instanceof JDatePickerImpl && this.values.containsKey(key)) {
            ((UtilDateModel)((JDatePickerImpl)component).getModel()).setValue((Date)this.values.get(key));
        }

        component.addKeyListener(new AddEditDialogHandler(this.frame, this));
        component.setAlignmentX(0.0F);
        this.add(label);
        this.add(Box.createVerticalStrut(10));
        this.add(component);
        this.add(Box.createVerticalStrut(10));
    }

    MainButton ok = new MainButton(Text.get("ADD"), Style.ICON_OK, new AddEditDialogHandler(this.frame, this), "ADD");
    if (!this.isAdd()) {
        ok.setActionCommand("EDIT");
        ok.setText(Text.get("EDIT"));
    }

    MainButton cancel = new MainButton(Text.get("CANCEL"), Style.ICON_CANCEL, new AddEditDialogHandler(this.frame, this), "CANCEL");
    JPanel panelButtons = new JPanel();
    panelButtons.setLayout(new BorderLayout());
    panelButtons.setAlignmentX(0.0F);
    panelButtons.add(ok, "West");
    panelButtons.add(Box.createRigidArea(Style.DIMENSION_DIALOG_PADDING_BUTTON), "Center");
    panelButtons.add(cancel, "East");
    this.add(panelButtons);
    this.pack();
    this.setLocationRelativeTo((Component)null);
}
```

Рисунок 3.20 - Створення настройки діалогового вікна(2)

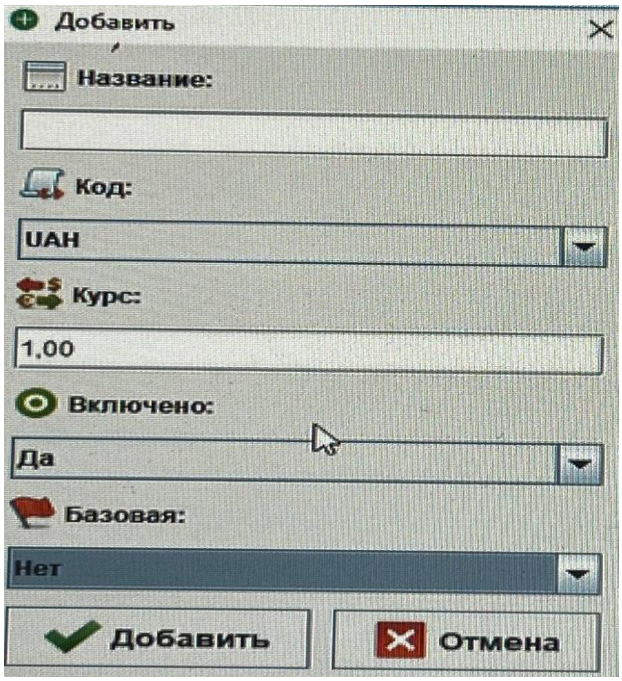

Рисунок 3.21 - Приклад діалогового вікна

Також була добавлена функція помилки діалогового вікна. Якщо цю функцію не добавити, і програма буде працювати не коректно то нічого не буде відбуватися. А якщо добавити цю функція то вийде помилка і користувачу відразу стане зрозуміло в чому справа.

```
public class ErrorDialog {
    public ErrorDialog() {
    }

    public static void show(MainFrame frame, String text) {
        JOptionPane.showMessageDialog(frame, Text.get(text), Text.get("ERROR"), 0);
    }
}
```

Рисунок 3.22 - Приклад написання коду, якщо вийде помилка діалогового вікна

3.2.5 Вивід даних.

Спочатку треба зробити ліву і праву панель, щоб краще було використовувати програму.

```
public final class LeftPanel extends AbstractPanel {
    public LeftPanel(MainFrame frame) {
        super(frame);
        this.init();
    }

    protected void init() {
        this.setLayout(new BoxLayout(this, 1));
        this.setBorder(Style.BORDER_LEFT_PANEL);
        JLabel headerBC = new JLabel(Text.get("BALANCE_CURRENCIES"));
        headerBC.setFont(Style.FONT_LABEL_HEADER);
        headerBC.setIcon(Style.ICON_LEFT_PANEL_BALANCE_CURRENCIES);
        headerBC.setAlignmentX(0.5F);
        this.add(headerBC);
        this.addBalanceCurrency();
        this.add(Box.createVerticalStrut(20));
        JLabel headerB = new JLabel(Text.get("BALANCE"));
        headerB.setFont(Style.FONT_LABEL_HEADER);
        headerB.setIcon(Style.ICON_LEFT_PANEL_BALANCE);
        headerB.setAlignmentX(0.5F);
        this.add(headerB);
        this.addBalance();
    }
```

Рисунок 3.22 - Створення лівої панелі

```
public RightPanel(MainFrame frame, TableData td, String title, ImageIcon icon, JPanel[] panels) {
    super(frame);
    this.td = td;
    this.title = title;
    this.icon = icon;
    this.panels = panels;
    this.init();
}

public RightPanel(MainFrame frame, TableData td, String title, ImageIcon icon, AbstractToolBar tb) {
    this(frame, td, title, icon, new JPanel[]{tb});
}

public RightPanel(MainFrame frame, TableData td, String title, ImageIcon icon) {
    this(frame, td, title, icon, new JPanel[0]);
}

protected void setPanels(JPanel[] panels) { this.panels = panels; }
```

Рисунок 3.23 - Створення правої панелі

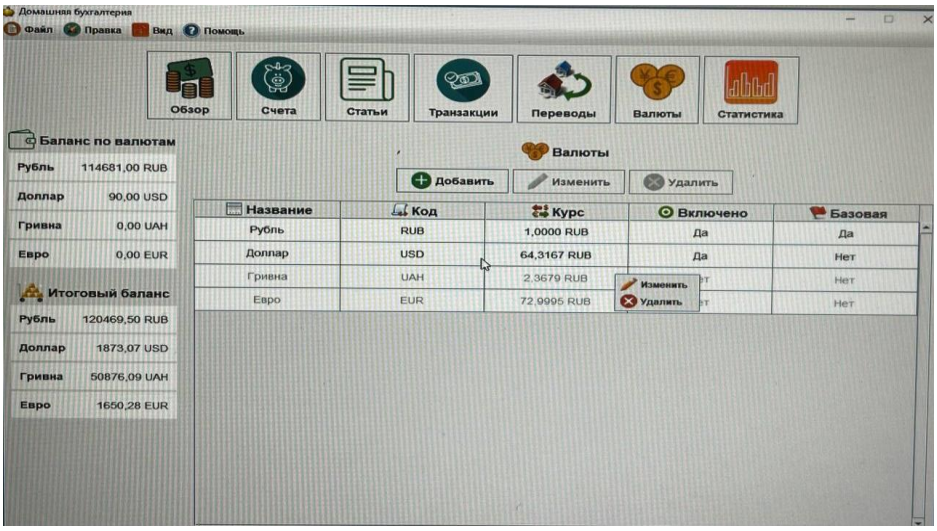

Рисунок 3.24 - Результат виходу даних за допомогою панелей

3.2.6. Вивід статистики.

Вивід статистики буде у вигляді гістограми. Гістогра́ма — спосіб графічного представлення табличних даних, приблизне представлення розподілу числових даних. Являє собою діаграму, що складається з прямокутників без розривів між ними. Кількісні співвідношення деякого показника представлені у вигляді прямокутників, площі яких пропорційні. Найчастіше для зручності сприйняття ширину прямокутників беруть однакову, при цьому їх висота визначає співвідношення відображуваного параметра.

```java
public final class Chart {
    private DefaultPieDataset dataset;
    private final String title;
    private final String currencyTitle;

    public Chart(HashMap<String, Double> data, String title, String currencyTitle) {
        this.setData(data);
        this.title = Text.get(title);
        this.currencyTitle = currencyTitle;
    }

    private void setData(HashMap<String, Double> data) {
        this.dataset = new DefaultPieDataset();
        Iterator var2 = data.entrySet().iterator();

        while(var2.hasNext()) {
            Entry<String, Double> entry = (Entry)var2.next();
            this.dataset.setValue((Comparable)entry.getKey(), (Number)entry.getValue());
        }

    }
```

Рисунок 3.25 - Створення гістограми

```java
public JPanel getPanel() {
    JFreeChart chart = ChartFactory.createPieChart3D(this.title, this.dataset, true, true, false);
    PiePlot plot = (PiePlot)chart.getPlot();
    plot.setToolTipGenerator(new StandardPieToolTipGenerator("{0}: {1} " + this.currencyTitle + " ({2})"));
    JPanel panel = new ChartPanel(chart);
    panel.setPreferredSize(Style.DIMENSION_CHART);
    if (this.dataset.getItemCount() == 0) {
        panel.setLayout(new GridBagLayout());
        JLabel label = new JLabel(Text.get("CHART_NO_DATA"));
        label.setFont(Style.FONT_LABEL_HEADER);
        panel.add(label);
    }

    return panel;
}
```

Рисунок 3.26 - З’єднання даних із гістограмою

Рисунок 3.27 - Результат гістограми

3.2.7. Обробка подій.

Обробка подій полягає в моделі делегування подій - джерело повідомляє про подію однієї чи кілька слухачів. Слухач чекає доти, доки не отримає повідомлення про подію. При отриманні слухач обробляє його та повертає управління. Слухач повинен реєструватися джерелом, щоб отримувати повідомлення про події. Таким чином повідомлення надсилаються лише тим слухачам, які бажають їх отримувати.

Подія - це спеціальний об'єкт, який описує зміну стану джерела. Це може бути, наприклад, клацання кнопки, введення символу з клавіатури, вибір елемента у списку тощо. Подія може відбуватися без участі користувача, наприклад, при використанні таймера. Також можна створювати власні події.

```
public abstract class Handler implements ActionListener {
    protected final MainFrame frame;

    public Handler(MainFrame frame) { this.frame = frame; }

    public void actionPerformed(ActionEvent ae) { this.frame.refresh(); }
}
```

Рисунок 3.28 - Створення абстрактного класу Handler

```java
public class AddEditDialogHandler extends Handler implements WindowListener, KeyListener {
    private final AddEditDialog dialog;

    public AddEditDialogHandler(MainFrame frame, AddEditDialog dialog) {
        super(frame);
        this.dialog = dialog;
    }

    public void actionPerformed(ActionEvent ae) {
        String var2 = ae.getActionCommand();
        byte var3 = -1;
        switch(var2.hashCode()) {
        case 64641:
            if (var2.equals("ADD")) {
                var3 = 0;
            }
            break;
        case 2123274:
            if (var2.equals("EDIT")) {
                var3 = 1;
            }
            break;
        case 1980572282:
            if (var2.equals("CANCEL")) {
                var3 = 2;
            }
        }
```

Рисунок 3.29 - Обробка подій(1)

```java
        switch(var3) {
        case 0:
            this.addEdit(true);
            break;
        case 1:
            this.addEdit(false);
            break;
        case 2:
            this.closeDialog();
        }

        super.actionPerformed(ae);
    }

    private void addEdit(boolean add) {
        try {
            if (add) {
                SaveData.getInstance().add(this.dialog.getCommonFromForm());
            } else {
                SaveData.getInstance().edit(this.dialog.getCommon(), this.dialog.getCommonFromForm());
            }

            this.closeDialog();
        } catch (ModelException var3) {
            ErrorDialog.show(this.frame, var3.getMessage());
        }

    }
```

Рисунок 3.30 - Обробка подій(2)

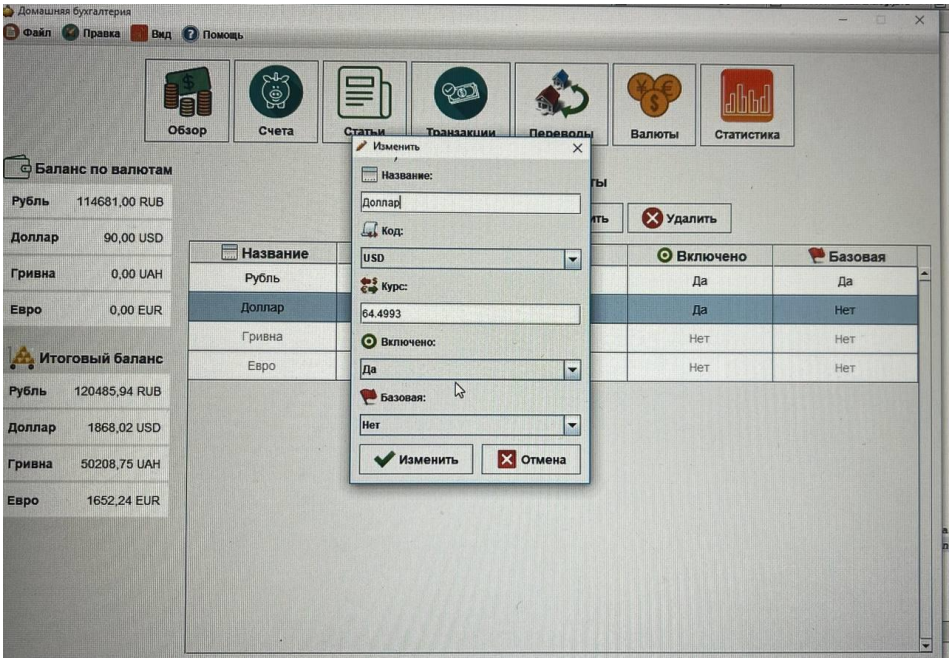

Рисунок 3.31 - Результат обробки подій

3.3 Аналітична частина програми.

Java — це об'єктно-орієнтована мова програмування, яка була розроблена компанією Sun Microsystems (надалі придбаною компанією Oracle). Дата офіційного випуску – 23 травня 1995 року.

Програми Java зазвичай транслюються в байт-коді, який потім через віртуальну машину виконується Java (JVM). JVM – це програма, яка може обробляє любий байтовий код та передає інструкції обладнання як інтерпретатор. Це зручно тому що дає перевагу в реалізації байтового коду від операційної системи та обладнання, що дозволяє легко виконувати Java-програми на будь-якому пристрої, для якого існує JVM. Тому можна вважати, що за допомогою особливості технології можна зробити гнучку систему безпеки. Саме тому багато банківських компанії їх ПЗ написані на мові Java. Також дуже легко можно керувати даними із віртуальною машиною. Будь-які операції, які можуть перевищувати встановленими обмеженнями програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають швидке переривання. Часто до недоліків концепції віртуальної машини відносять те, що виконання байт-коду віртуальною

машиною може негативно скасуватися на продуктивність програм та алгоритмів, реалізованих мовою Java. Java має репутацію не дуже швидких а зазвичай повільніших і більше оперативної пам'яті програм, ніж іншими мовамипрограмування. Однак, якщо порівнювати Java з мовами, що інтерпретируются, які найчастіше використовуються у веб-розробці. Під час написання програми було використано 78 інтерфейсів і класів, 5 патернів, 14 пакетних класів та 62 іконки.

В Java є документація в якому розписані багато інформацій про патерни, класи, виключення та багато чого іншого. Наприклад там можна дізнатися про клас handler який я використовував у програмі. Handler – це клас, який використовується для роботи з чергою повідомлень, пов'язаних із потоком. Handler дозволяє надсилати повідомлення в інші потоки із затримкою або без, а також обробляти отримані повідомлення. Handler завжди пов'язаний з looper, який у свою чергу пов'язаний із будь-яким потоком. При створенні handler в конструктор можна передати об'єкт Looper. Якщо використовується дефолтний конструктор, handler створюється на поточному потоці. Якщо з потоком не пов'язаний looper, то при створенні handler кидається RuntimeException.

Клас Wrapper — це клас, об’єкт якого обгортає або містить примітивні типи даних. Коли ми створюємо об’єкт для класу обгортки, він містить поле, і в цьому полі ми можемо зберігати примітивні типи даних. Іншими словами, ми можемо загорнути примітивне значення в об’єкт класу обгортки. Вони перетворюють примітивні типи даних в об’єкти. Об’єкти потрібні, якщо ми хочемо змінити аргументи, передані в метод (оскільки примітивні типи передаються за значенням). Класи в пакеті java.util обробляють лише об'єкти, і, отже, класи обгортки також допомагають у цьому випадку. Структури даних у структурі Collection, такі як ArrayList і Vector, зберігають лише об’єкти (посилальні типи), а не примітивні типи. Для підтримки синхронізації в багатопотоковому режимі потрібен об’єкт.

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Рисунок 3.32 - Приклад Wrapper

Діалогове вікно є незалежним підокном, призначеним, щоб перенести тимчасове повідомлення крім основного вікна програми Swing. Більшість діалогових вікон представляють повідомлення про помилку або попередження користувачеві, але діалогові вікна можуть представити зображення, дерева каталогів, або приблизно що-небудь сумісне з основною програмою Swing, яка керує ними. Для зручності кілька компонентних класів Swing можуть інсталювати і вивести на екран діалогові вікна. Щоб створити прості, стандартні діалогові вікна, Ви використовуєте JOptionPane class. ProgressMonitor class може підняти діалогове вікно, яке показує просування роботи. Два інших класи, JColorChooser та JFileChooser, також надайте стандартні діалогові вікна. Щоб перевести діалогове вікно друку в робочий стан, можна використовувати API Друк. Щоб створити власне діалогове вікно, треба використовувати JDialog class безпосередньо.

Висновки до розділу 3

Такі мова програмування як Java у сучасному світі мають великий вплив і розвиваються з великою швидкістю. Java має велику базу даних і інформації яку можна подивитися. Було написано велика кількість програм які і на сьогоднішній день є актуальними. Можна легко розробити сайт товарів, чатів, форумів та інші будь-які теми. Сьогодні це зробити дуже легко і зробити це все можна дуже швидко. За відносно не великий проміжок часу можна створити програму яка дає змогу повністю реалізувати всі потреби користувачів.

Під час написання цього розділу було показано реалізацію архітектури програми. Було ознайомлено і реалізовано багато технологій для реалізації завдання.

В результаті тестування роботи фінансової програми було зроблено закономірні висновки що програма працює справно і має перспективі для розвитку.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ДОКУМЕНТАЦІЇ

4.1 Опис програмної реалізації

4.4.1 Обробка подій

Обробка подій полягає в моделі делегування подій (delegation event model) - джерело повідомляє про подію однієї чи кілька слухачів (listener). Слухач чекає доти, доки не отримає повідомлення про подію. При отриманні слухач обробляє його та повертає управління. Слухач повинен реєструватися джерелом, щоб отримувати повідомлення про події. Таким чином повідомлення надсилаються лише тим слухачам, які бажають їх отримувати.

Подія - це спеціальний об'єкт, який описує зміну стану джерела. Це може бути, наприклад, клацання кнопки, введення символу з клавіатури, вибір елемента у списку тощо. Подія може відбуватися без участі користувача, наприклад, при використанні таймера. Також можна створювати власні події.

Джерело реєструє слухачів через окремі методи реєстрації. Як правило, імена методів мають форму addТипListener(ТипListener listener) або setТипListener(ТипListener listener). Тип – це ім'я події, а listener – посилання на слухача подій.

Слухач (listener) - це об'єкт, що повідомляється про подію. Він повинен бути зареєстрований джерелом подій та реалізовувати методи для отримання та обробки повідомлень.

Основний клас для подій - клас EventObject, який містить два методи getSource() та toString().

Підкласи: ConnectionEvent, HandshakeCompletedEvent, NodeChangeEvent, PreferenceChangeEvent, PropertyChangeEvent, RowSetEvent, , StatementEvent.

4.4.2 Моделі

Спочатку було створено папку проекту в якому знаходиться директорії. Ціль цих директорій має керувати всіма даними цього проекту. Приклад цих директорій:

Database - Шаблони баз даних з установками проекту та конфігурацією, а також файли записаних баз даних (.db).При аналізі записаних баз даних створюється і зберігається копія бази даних як файл з розширенням (.filt.db);

DtmData - Файли даних цифрової моделі дна (DTM data) вибрані у налаштуваннях Session Setup. Файли з розширенням *.qpd зберігаються у піддиректорії з назвою судна після завантаження файлів у Processing Manager;

DxfToQxf - Дані картки в обмінному форматі DXF (наприклад, файли AutoCAD) або у форматі QXF, на який дані перетворюються за допомогою програмної утиліти DXF Convertor;

Export - Дані, експортовані з режиму обробки (processing session) до іншого формату, як ASCII;

Geoid - Місце для рівнів річкової моделі створених за допомогою програмної утиліти Geoid Model Utility;

Graphics - Папка для збереження миттєвих знімків кадру або зображень GeoTIFF;

GridData - Файли сітки глибин Sounding Grid створені за допомогою програмної утиліти Sounding Grid Utility;

Import - Імпортовані дані можна зберігати тут;

- LineData - Бази даних запланованих галсів створені за допомогою програмної утиліти Processing Manager. (*.qgf);

- LogFiles - Файли запису подій (Logfiles), що створюються при працюючій програмі Alert Display або при використанні опції «Людина за бортом» (Man Over Board option), або при включеній опції Generic Logging з розширенням (*.txt);

- Mapping - Файли проекту QINSy Mapping (Terramodel) (*.pro);

- Results - Результати розрахунку позиціонування (параметри позиції та вузла) (*.res);

- Settings - Файли встановлення, що використовуються програмами Display Manager, Generic Logging або Sound Velocity Processor;

- Support - Різні спільні файли, наприклад файл форми судна (shape file);

- TEMP - Усі часові файли;

- Tide - Файли даних рівня моря створені в програмній утиліті Tide Data Manager.

4.4.3 Введення та вивід даних.

На відміну від більшості класів введення/виводу, клас File працює не з потоками, а безпосередньо з файлами. Цей клас дозволяє отримати інформацію про файл: права доступу, час і дату створення, шлях до каталогу. А також здійснювати навігацію з ієрархій підкаталогів.

При роботі з даними введення/виводу вам часто траплятиметься термін Потік (Stream). Потік – це абстрактне значення джерела або приймача даних, які здатні обробляти інформацію. Ви насправді не бачите, як дійсно йде обробка даних у пристроях вводу/виводу, тому що це складно, і вам це не потрібно. Це як із телевізором - ви не знаєте, як сигнал із кабелю перетворюється на картинку на екрані, але цілком можете перемикатися між каналами через пульт. Є два типи потоків: байтові та символьні. У деяких ситуаціях символьні потоки ефективніші, ніж байтові.

За введення та виведення відповідають різні класи Java. p align="justify"> Класи, похідні від базових класів InputStream або Reader, мають методи з іменами read() для читання окремих байтів або масиву байтів (відповідають за введення даних). p align="justify"> Класи, похідні від класів OutputStream або Write, мають методи з іменами write() для запису одиночних байтів або масиву байтів (відповідають за виведення даних).

Клас OutputStream – це абстрактний клас, що визначає потоковий байтовий висновок. У цій категорії знаходяться класи, що визначають, куди прямують ваші дані: в масив байтів (але не безпосередньо в String; передбачається, що ви зможете створити їх з масиву байтів), у файл або канал. BufferedOutputStream- Буферизований вихідний потік. Створює буфер у пам'яті. Всі дані, що надсилаються в цей потік, розміщуються у створеному буфері. Абстрактний

клас, що надає інтерфейс для класів-надбудов, що додають до існуючих потоків корисні властивості.

Методи класу:

- int close() - закриває вихідний потік. Наступні спроби запису передадуть виключення IOException

- void flush() - фіналізує вихідний стан, очищуючи всі буфери виводу;

- abstract void write (int oneByte) - записує єдиний байт у вихідний потік;

- void write (byte[] buffer) - записує повний масив байтів у вихідний потік;

- void write (byte[] buffer, int offset, int count) - записує діапазон з count байт із масиву, починаючи зі зміщення offset.

Клас BufferedOutputStream не сильно відрізняється від класу OutputStream, за винятком додаткового методу flush(), що використовується для забезпечення запису даних в потік, що буферизується. Буфери виведення потрібні підвищення продуктивності.

Клас ByteArrayOutputStream використовує байтовий масив у вихідному потоці. Метод close() можна викликати.

Клас DataOutputStream дозволяє писати елементарні дані в потік через інтерфейс DataOutput, який визначає методи, що перетворюють елементарні значення форму послідовності байтів. Такі потоки полегшують збереження у файлі двійкових даних. Клас DataOutputStream розширює клас FilterOutputStream, який, у свою чергу, розширює клас OutputStream.

Клас FileOutputStream створює об'єкт класу OutputStream, який можна використовувати для запису байтів у файл. Створення нового об'єкта залежить від того, чи існує заданий файл, оскільки створює його перед відкриттям. У разі спроби відкрити файл, доступний лише для читання, буде передано виняток.

Методи класу Reader:

abstract void close() – закриває вхідний потік. Наступні спроби читання передадуть виключення IOException;

void mark(int readLimit) - поміщає мітку у поточну позицію у вхідному потоці;

boolean markSupported() - повертає true, якщо потік підтримує методи mark() та reset();

int read() - повертає ціле уявлення наступного доступного символу викликає вхідного потоку. Після досягнення кінця файлу повертає значення -1. Є й інші перевантажені версії методу;

boolean ready() - повертає значення true, якщо наступний запит очікувати не буде;

void reset() - скидає покажчик уведення в раніше встановлену позицію мітки;

logn skip(long charCount) - пропускає вказану кількість символів введення, повертаючи кількість дійсно пропущених символів.

Клас FileReader, похідний класу Reader, можна використовувати для читання вмісту файлу. У конструкторі класу необхідно вказати шлях до файлу, або об'єкт типу File.

Клас Writer – абстрактний клас, що визначає символьний потоковий висновок. У разі помилок усі методи класу передають виняток IOException.

Методи класу:

Writer append(char c) - додає символ наприкінці вихідного потоку. Повертає посилання на потік, що викликає;

Writer append(CharSequence csq) - додає символи в кінець вихідного потоку, що викликає. Повертає посилання на потік, що викликає;

Writer append(CharSequence csq, int start, int end) - додає діапазон символів наприкінці вихідного потоку. Повертає посилання на потік, що викликає.

Клас BufferedWriter - це клас, похідний від класу Writer, який буферизує висновок. З його допомогою можна підвищити продуктивність рахунок зниження кількості операцій фізичної записи у вихідний пристрій.

Клас FileWriter створює об'єкт класу, похідного від класу Writer, який можна використовувати для запису файлу. Існують конструктори, які дозволяють додати висновок у кінець файлу. Створення об'єкта не залежить від наявності файлу, він буде створений у разі потреби. Якщо файл існує і він доступний тільки для читання, передається виняток IOException. Існує безліч класів та методів для читання та запису файлів. Найбільш поширені з них – класи FileInputStream та FileOutputStream, які створюють байтові потоки, пов'язані з файлами. Щоб відкрити файл, потрібно створити об'єкт одного з цих файлів, вказавши ім'я файлу як аргумент конструктора.

4.2 Керівництво користувача

Бібліотека Swing включає багатий вибір стандартних діалогових вікон, що істотно спрощують і прискорюють виведення простої інформації типу повідомлень про роботу програми, помилки та нестандартні ситуації. Для виведення в графічний інтерфейс програми різноманітної інформації та вибору простих даних призначений клас JOptionPane, робота з яким пов'язана з викликом одного з численних статичних методів, що створюють та виводять на екран модальне діалогове вікно стандартного вигляду. У діалогових вікнах JOptionPane можна виводити найрізноманітнішу інформацію та, за необхідності, розміщувати в них додаткові компоненти.

JOptionPane успадкований від базового класу JComponent бібліотеки Swing, отже можна працювати з нею безпосередньо, тобто. створювати екземпляри класу JOptionPane та налаштовувати їх властивості. Використання стандартних діалогових вікон суттєво спрощує розробку програми та дозволяє прискорити процес освоєння користувачем інтерфейсу.

Всі стандартні діалогові вікна Swing мають власні UI-представники, які відповідають за інтерфейс вікна у додатку. Це особливо важливо для зовнішніх видів вікон, що імітують відомі платформи, користувачі яких не повинні відчувати значної різниці при переході від «рідних» додатків до Java-додатків.

Рядок меню JMenuBar представляє звичайний контейнер, що нічим не відрізняється від панелі JPanel і володіє тими самими властивостями. JMenuBar може бути використана не тільки для роботи з меню, що випадає. У рядок меню можна додавати всілякі компоненти, наприклад написи зі значками або списки, що розкриваються. Як менеджер розташування JMenuBar використовує BoxLayout з розташуванням компонентів по горизонталі.

Панель інструментів надає користувачам часто використовувані функції програми. Зазвичай ми розміщуємо панель інструментів безпосередньо під панелями меню у верхній частині рамки. Панель інструментів діє як контейнер для інших компонентів, включаючи кнопку, поле зі списком і меню. Ми часто використовуємо панель інструментів як інструмент вибору, наприклад, для вирівнювання тексту в програмі офісних пакетів. Крім того, ми використовуємо панель інструментів для заміни часто використовуваних функцій меню, щоб користувачі могли швидше отримувати доступ до функцій програми. У деяких програмах ви побачите, що ми використовуємо панель інструментів для навігації, наприклад, у веб-браузерах.

Клас JToolBar. Щоб створити панель інструментів у Java Swing, ви використовуєте клас JToolBar. Клас JToolbar підтримує дві орієнтації: вертикальну та горизонтальну. Ви використовуєте атрибут orientation для підтримки поточної орієнтації панелі інструментів. Ви можете додати будь-який компонент на панель інструментів, включаючи кнопку, поле зі списком і меню. Порядок компонентів на панелі інструментів визначається цілим індексом. Якщо ви хочете відокремити групу пов’язаних компонентів на панелі інструментів, ви можете використовувати власний роздільник JToolBar, викликавши метод addSeperator(). JToolBar також підтримує плаваючу панель інструментів. Якщо ви не хочете, щоб панель інструментів була плаваючою, ви можете використовувати метод setFloatable(false).

Висновки до розділу 4

В цьому розділі було розглянуто основні принципи роботи програми, а саме створення основних класів та файлів які працюють в цій програмі. Також було використано багато інструментів які допомагають створювати та реалізовувати меню, тулбари та діалогові вікна.

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«ЗАСТОСУНОК ДЛЯ АНАЛІЗУ І КОНТРОЛЮ ОСОБСТИХ ВИТРАТ ТА ФІНАНСІВ»

Спеціальність 122 – Комп’ютерні науки

122-БКР-401.21810303

Виконав студент 4 курсу, групи 401

_____В.П. Бондаренко

«___» _____2022р

Консультант: ст. викладач

_____О. В. Макарова

«___» _____ 2022 р

Миколаїв – 2022

5 ОХОРОНА ПРАЦІ

У нашому світі охорона праці є дуже актуальною темою, бо це піклування про наше здоров'я. Охорона праці - правова, соціально-економічна, організаційно-технічна системи, санітарно-гігієнічні та технічно-профілактичні заходи та засоби, спрямовані на збереження життя та працездатності в процесі праці. Цей розділ дуже сильно перетинається із моєю дипломною роботою, бо теж зв'язаний із здоров'ям. Тому ми розглянемо два важливих аспекти. Це мету і завдання яка буде стосуватися моєї роботи.

Мета – продемонструвати та показати суспільству о важливості такого предмету як охорона праці. А саме розказати про безпеку з роботою персонального комп'ютера, та запобігти від погіршення здоров'я.

Завдання – розрухувати мікроклімат робочого приміщення та написати декілька допустимих значень які повинні бути у робочого місця. Також розповісти про безпеку під час роботи, про освітленість, гігієну, захист від електромагнітного випромінювання та захист від пожежі.

Основи науки про охорону праці існують багато століть і пройшли довгий шлях свого становлення. Початком наглядової діяльності щодо захисту робітників можна вважати виданий у 1719 році Петром I, Указ про створення Берг-колегії (гірничої колегії).

Відомство було засновано для управління гірничим виробництвом указом Петра I від 10 грудня 1719 року «Про заснування Берг-колегіуму для ведення в даній справі про руди та металів». До цього часу гірничою справою завідував Наказ рудокопних справ, заснований 24 серпня 1700 року. Наказ не мав в своєму розпорядженні відповідного апарату управління і діяв через місцеві адміністрації, що не дозволило йому стати централізованим органом влади.

Вперше необхідність встановлення нової форми управління гірничо видобуванням, а саме форми колегіальної, була заявлена в 1712 році І. Ф.

Блюером в поданому ним особисто Петру I меморіалі, де він докладно встановлював виробництво нового відомства, його права, склад і т. д. Хоча уявлення Блюера найближчого успіху і не мало, але, як цілком відповідне просвітнім планам імператора, лягло в основу при установі Берг-колегії, що діяла спочатку (з 1720 до 1722 роки) спільно з Мануфактур-колегією по «подібністю їх справ і обов'язків». Але в міру збільшення числа гірських заводів, посилюється з часу видання Берг-привілеї, а отже і справ з управління ними, виявилося незручним спільне управління гірськими заводами і мануфактурами.

А в 1720 році на вугільній шахті Петро I наказав дотримуватися правил безпечного користування сходами та покращувати умови робітників.

18 квітня 1720 року указом Петра I при Берг-колегії була створена лабораторія для досліджень руд і металів на чолі з Я.В. Брюсом. Він керував нею до 1726 року поліпшив справу видобутку і переробки корисних копалин.

В 1722 Берг-колегія була відділена від Мануфактур-колегії і отримала статус самостійної установи зі своїми певними правами і обов'язками, складом та іншим.

У функції Берг-колегії входило безпосереднє управління казенними заводами і управління їхньою продукцією, відведення місць під заводи і рудники, рекомендації з технічного і технологічного розвитку заводів, лабораторні дослідження проб, розгляд суперечок між власником заводу, збір податків і контроль над поставками металів, приписка селян до заводам і видача дозволів на покупку кріпаків, наймання зарубіжних фахівців, здійснення наглядових і каральних заходів стосовно працівників заводів.

В 1742 р. Була опублікована праця М. В. Ломоносова «Перший металургійний фундамент або рудна справа», яка розробила теорію природної вентиляції шахт, а також рекомендації що стосується техніки безпеки при використанні сходів і драбин, а також використання облицьованих робіт. Після скасування кріпосного права в Російській імперії в 1861 р. Здійснювався нагляд

за спеціальним органом, який контролював безпеку робочих місць у гірничодобувній промисловості, який називався Гірнича поліція.

Охорона праці як дисципліна була вперше введена в 1929 році в Московському залізничному транспортному інституті. У 1966 році було запропоновано створити відділ охорони праці при вищому технічному закладі.

Міжнародна організація праці (МОП) зробила важливий внесок у розвиток охорони праці в цілому.

Згодом, з розширенням державного нагляду «безпеки поведінки» в інших галузях, цей орган перетворився на Державний нагляд за гірничодобувною промисловістю з питань охорони праці в Україні (Держгірпромнагляд).

Охорона праці - правова, соціально-економічна, організаційно-технічна системи, санітарно-гігієнічні та технічно-профілактичні заходи та засоби, спрямовані на збереження життя та працездатності в процесі праці.

5.1 Аналіз умов праці

Кімната, на яку було подано заявку, знаходилась на другому поверсі двоповерхового будинку. Розмір кімнати $a \times b \times h = 3.0 \times 2.2 \times 2,7$ м. Кімната розміром з одне вікно $c \times d = 1,2 \times 1,6$, кімната має сучасний інтер'єр, стелю та стіни, поверхня яких майже рівна.

Розмір площі для одного робочого місця оператора персонального комп'ютера має бути не менше 6 кв. м, а об'єм — не менше 20 куб. м [14]. Отже, дане приміщення цілком відповідає зазначеним нормам.

Вікна оснащені світлозахисним пристроєм зі шторами, які можна регулювати горизонтально.

У кімнаті є одне робоче місце з персональним комп'ютером. Є також один матрац та полиця для книг.

Напруга живлення обладнання становить 220В. Електрична коробка виконана з трьох проводів, що відповідають всім вимогам NPAOP 40.1-1.01-97 та NAPB A.01.001-2004. Відповідно до ризику ураження електричним струмом, кімната показує простір без ризику ураження електричним струмом.

Мікрокліматичні умови влітку (частково в перехідний період) забезпечуються роздільною системою кондиціонування, обладнаною лабораторною кімнатою, достатньою для забезпечення комфортних умов праці. Взимку опалення забезпечується центральною системою.

Зовнішнє шумове навантаження мінімізоване завдяки сучасній конструкції вікон. Внутрішній шум дозволений і відповідає рівню для конкретного простору, який вважається трудовим процесом: наявність оргтехніки та зв'язок між персоналом.

5.2 Техніка безпеки

Сьогодні комп'ютерні технології використовуються у всіх сферах людської діяльності. Під час роботи з комп'ютером на деяких людей впливають шкідливі та небезпечні фактори виробництва: електромагнітне поле (діапазон радіочастот: ВЧ, УВЧ та мікрохвильова піч), інфрачервоне випромінювання та іонізація, шум та вібрація, статична електрика, та інші впливаючи фактори можливого ризику. Робота за комп'ютером характеризується значним розумовим напруженням та нервово-емоційним навантаженням оператора, високою інтенсивністю зорової роботи та досить великим м'язовим навантаженням при роботі з клавіатурою комп'ютера. Дуже важливим є дизайн та розташування елементів на робочому місці , що важливо для підтримання оптимальної робочої пози для людини-оператора, згідно до [15].

У процесі роботи з комп'ютером потрібно дотримуватися правильного режиму праці та відпочинку.

5.3 Безпека під час роботи з персональним комп'ютером

Перед використанням, монітори слід щодня очищати від пилу та інших забруднень. По завершенню, ПК та периферійні пристрої слід видалити. В екстрених випадках негайно вийміть ПК та периферійні пристрої.

Не дозволяється:

• виконувати технічне обслуговування, ремонт та налаштування персональних комп’ютерів та периферійних пристроїв безпосередньо на робочому місці оператора;

• папір для зберігання, будь-які носії (диски, флешки тощо), запасні частини, деталі тощо. У безпосередній близькості від персональних комп’ютерів та периферійних пристроїв, якщо вони не використовуються для поточної роботи;

• вимкнути захисні пристрої, мимовільно вносячи зміни в конструкцію та склад персональних комп’ютерів та периферійних пристроїв або технічні налаштування;

• робота з персональними комп’ютерами, на яких під час роботи з’являються незвичні сигнали, нестабільні зображення на моніторі, тощо;

• працювати з матричними принтерами, якщо відсутній вібраційний килимок, а верхня кришка піднята;

5.4 Освітлення

Що стосується вікон, робоче місце слід розміщувати так, щоб природне світло було збоку, бажано зліва [16]. Потрібно розмістити робоче місце, обладнане ПК, щоб уникнути прямого потрапляння світла в очі.

Штучне освітлення приміщення повинно бути обладнане загальною системою освітлення [17]. Комбіновані системи освітлення можна використовувати в приміщеннях, де переважають документи (крім загального освітлення встановлюються місцеві освітлювальні прилади).

Рекомендується розміщувати штучні джерела світла по обидва боки екрану паралельно напрямку зору. Щоб уникнути глобального відблиску в напрямку очей, необхідно використовувати проти бликові екрани, спеціальні фільтри для екрану. Щоб забезпечити нормальні значення освітленості, вікна та лампи слід чистити принаймні двічі на рік, а згорілі лампи слід своєчасно замінювати.

5.5 Мікроклімат

Робочі приміщення повинні бути обладнані вентиляцією або кондиціонером для організованого обміну повітрям. Допускаються параметри мікроклімату (температура - 19,5 ± 0,5 ° С, відносна вологість -60 ± 5%, швидкість повітря не більше 0,1 м / с). Вологе прибирання слід проводити щодня, тому підлога в кімнаті не повинна бути килимовим покриттям. До і після використання комп’ютера протирайте екран спеціальною тканиною. У таблиці 5.1 наведено допустимі параметри мікроклімату в приміщенні, згідно до [17].

Таблиця 5.1 - Оптимальні та допустимі параметри мікроклімату

Період року	Параметр мікроклімату	Значення		
		Оптимальне	Допустиме	Фактичне
Холодний	Температура повітря в приміщенні	21,0-23,4°С	23,5-25,4°С	16,1-18,0°С
	Відносна вологість	40-60%	75%	35%
	Швидкість руху повітря	0,1м/с	до 0,1м/с	0,1м/с
Теплий	Температура повітря в приміщенні	21,0-23,4°С	23,5-25,4°С	26,7-27,4°С
	Відносна вологість	40-60%	55%	55%
	Швидкість руху повітря	0,1 м/с	0,2-0,1м/с	0,1м/с

5.6 Гігієна праці і виробнича санітарія

Охорона праці охороняє виробничі ризики на робочому місці з метою розробки та впровадження захисних заходів для забезпечення безпечних умов праці. Промислова гігієна - вивчає вплив виробничого процесу та навколишнього середовища на організм людини для розробки санітарно-гігієнічних та лікувально-профілактичних заходів, спрямованих на створення сприятливих умов праці, забезпечення здоров’я та високого рівня працездатності людини.

Промислова гігієна - це система організаційних заходів та технічних засобів, що запобігають або зменшують вплив шкідливих виробничих факторів на робітників, що за певних умов може призвести до травм або професійних захворювань. Основна мета - зменшити або усунути вплив негативних та шкідливих факторів виробництва на організм людини. Оскільки безпека праці є центральним заходом охорони праці, заходи щодо поліпшення умов праці та побуту працівників можуть не тільки зменшити нещасні випадки на виробництві, професійні та загальні захворювання, а й підвищити продуктивність праці та якість роботи.

Якщо ви працюєте з одним ПК, дотримуйтесь кількох правил, щоб мати здоровий вплив на виробничий процес:

– провітрюйте кімнату не менше двох разів на день;

– утримуйте робоче місце в чистоті;
– налаштуйте час, проведений перед комп'ютером, щоб уникнути надмірного впливу негативного випромінювання;
– зменшить рівень шуму до 50-65 дБА.

5.7 Захист від пожежі

Особливої уваги заслуговують заходи протипожежного захисту. Так, лінії електропередачі у всьому офісі повинні бути захищені від коротких замикань та коливань напруги, що може спричинити несправність комп'ютерів [18].

Будинки (без сервера) повинні бути обладнані автоматичною системою пожежної сигналізації та вогнегасниками.

Під час монтажу та експлуатації ліній електропередачі необхідно повністю запобігти утворенню джерела електричного займання через коротке замикання та перевантаження ліній, обмежити використання горючих ізольованих ліній і, по можливості, використовувати їх. протипожежна ізоляція. Одночасно працює більше п'яти комп'ютерів, на видному та доступному місці встановлений аварійний резервний вимикач, а це означає, що електроживлення приміщення поза освітленням можна повністю відключити.

5.8 Безпека електромагнітного випромінювання

Електромагнітне випромінювання (ЕМВ) - це процес створення вільного електромагнітного поля, яке випромінює швидко рухаються заряджені частинки. Основними джерелами електромагнітного випромінювання є трансформатори, лінії високої напруги (ЛЕП), промислові електроприлади, високочастотні радіо- і телевізійні антени та радіо станції, електромобілі, електронні пристрої, комп'ютери, стільникові телефони тощо. та інші електричні пристрої, які працюють на широкій радіочастоті [19].

Вплив ЕМВ на організм людини може бути фіксованим або проміжним і впливати на тип впливу. Переривчасте поле характеризується періодичністю та періодичністю впливу різної інтенсивності та випромінювання.

Під впливом ЕМВ і опромінення призводить до загальної слабкості, підвищеної стомлюваності, пітливості, сонливості та розладів сну, головних болів, болів у серці, змін артеріального тиску. Опромінення організму може спричинити деструктивні зміни в тканинах та органах гострого або хронічного характеру, в основному впливаючи на нервову систему. Люди роздратовані, це явище частіше зустрічається в міських районах, ніж у сільській, що пов’язано з будівництвом великих телефонних, телевізійних, електричних та кабельних мереж у містах.

Контроль за рівнями ЕМВ покладено на органи санітарного нагляду і інспекцію електрозв'язку, а на підприємствах - на службу охорони праці.

Щоб забезпечити безпеку персоналу від ЕМВ, використовуйте цілий ряд заходів захисту, поділених на організаційні, інженерні, лікувальні та профілактичні. Технічні заходи захисту залежать від типу, сили та призначення джерела випромінювання. До них належать:

– зменшення інтенсивності та щільності енергетичних потоків за допомогою узгоджених навантажень та споживачів енергії;
– захист робочого місця матеріалами з високою електропровідністю (мідь, латунь, алюміній та сплави, сталь);

Існує адміністративні та контролюючі органи - інспекція по радіозв'язку (на Україні, наприклад, Укрчастотнагляд), яка регулює розподіл частотних діапазонів для різних користувачів, дотримання виділених діапазонів, відстежує незаконне користування радіоефіром.

Захист від дії ЕМВ:

– екранування (активне і пасивне; джерела електромагнітного випромінювання або ж об'єкта захисту; комплексне екранування);
– видалення джерел з ближньої зони; з робочої зони;
– конструктивне вдосконалення обладнання з метою зниження використовуваних рівнів ЕМВ, загальною споживаної і випромінюваної потужності обладнання;

– обмеження часу перебування операторів або населення в зоні дії ЕМВ.

Висновки до розділу

Аналіз умов праці на робочому місці показав, що умови праці відповідають вимогам, оскільки площа та об’єм не перевищують нормативних значень нормативних значень, рівня шуму, вібрацій.

Термін служби запропонованих світлодіодних ламп становить 50 - 20 тис. Годин, що краще, ніж люмінесцентні лампи із терміном служби 10 - 20 тис. Годин, залежно від кількості вимикачів. З іншого боку, лампи більш економічні на 44% (світлодіодна лампа 20 +/- 1 Вт, флуоресцентна 36 +/- 1 Вт), більш ударостійкі, не містять токсичних речовин і не мають особливих вимог до утилізації. Ці лампи створюють оптимальні умови для візуальної роботи інженера-програміста, а відносно низька температура нагрівання підвищує рівень пожежної безпеки.

Значення фактичної вологості в приміщенні в холодний період - 35%, не потрапляє в допустимий діапазон. Тому слід використовувати зволожувачі повітря в приміщенні в холодну пору року і встановлювати додатковий обігрів для підвищення температури. Для зниження температури в жарку пору року встановлений кондиціонер.

ВИСНОВКИ

Під час написання бакалаврської кваліфікаційної роботи в рамках дослідження було створено додаток ураховуючи особливості проектування, використовуючи бази даних та аналітичну частину за допомогою потужних та сучасних інструментів та технологій. Основною метою залишалася навчити людей фінансової грамотності.

Висновком даної роботи, є дослідження присвячені розробці додатку для аналізу і контролю особистих витрат та фінансів.

Було досліджено та розібрано основні принципи створення додатку. Визначено від яких принципів створення залежи успіх фінансової програми (це й структурованість, динамічність, якість і тематичність).

Під час цієї роботи можна зробити висновки, що в першому розділі розглянута актуальність застосунку та її аналоги.

Також було продемонстровано повну теорію та склад програми з повним описом програмного забезпечення для підготовки створення додатку, виконано детальний аналіз кожного з них відносно виконання поставленої задачі.

Для реалізації застосунку було обрано написати програму на мові Java використовуючи IntelliJ та Neatbeans.

Також, працюючи над розділом третім і четвертим дипломної роботі, було продемонстровано сучасні тенденцій розвитку технологій створення та функціонування додатку було створено за наступними критеріями:

- створення моделі. Це створення проекту, класів;
- загрузка та збереження даних. Це створення механізмів загрузки та збереження даних;
- створення базового інтерфейсу користувача. А саме це вивід основного вікна та меню;
- діалогові вікна. Це створення вікна з інформацією о помилки та підтвердженими операціями;

- вивід даних. А саме це вивід лівої панелі с балансом, створення класу для правої панелі, створення моделі таблиці, створення класу таблиці;
- вивід статистики. Створення класу для виводу гістограми;
- обробка подій. Це обробка меню та обробка подій тулбару із функціямию.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. API Documentation & Design Tools for Teams | Swagger. *API Documentation & Design Tools for Teams | Swagger*. URL: https://swagger.io (дата звернення: 20.06.2022).
2. Сімейний бюджет. *Сімейний бюджет*. URL: https://simeinyi-budzhet.ua (дата звернення: 20.06.2022).
3. 6,500,000+ free and premium vector icons, illustrations and 3D illustrations. *Iconfinder*. URL: https://www.iconfinder.com (дата звернення: 20.06.2022).
4. API Editor - Download or Try it in the Cloud. *API Documentation & Design Tools for Teams | Swagger*. URL: https://swagger.io/tools/swagger-editor/ (дата звернення: 20.06.2022).
5. CODE BLOG. *CODE BLOG*. URL: https://shwanoff.ru (дата звернення: 20.06.2022).
6. GeeksforGeeks | A computer science portal for geeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org (дата звернення: 20.06.2022).
7. How to Create ToolBar by Using JToolBar - zentut. *zentut*. URL: https://www.zentut.com/java-swing/how-to-create-toolbar-by-using-jtoolbar (дата звернення: 20.06.2022).
8. Icy Science. *Icy Science*. URL: https://uk.theastrologypage.com (дата звернення: 20.06.2022).
9. Автор проектів вікімедіа. Гістограма. *Вікіпедія*. URL: https://uk.wikipedia.org (дата звернення: 20.06.2022).
10. Обробка подій. *Обробка подій*. URL: http://developer.alexanderklimov.ru (дата звернення: 20.06.2022).

11. Interview Review Java ¸ Android. *Interview Review Java ¸ Android*. URL: https://itsobes.ru (дата звернення: 20.06.2022).
12. Java і інтерфейси. *Java і інтерфейси*. URL: https://ask-dev.ru (дата звернення: 20.06.2022).
13. Java | класи та об'єкти. *METANIT.COM*. URL: https://metanit.com/java/tutorial/3.1.php (дата звернення: 20.06.2022).
14. Java Swing. *zentut*. URL: https://www.zentut.com/java-swing (дата звернення: 20.06.2022).
15. MC.today, Media for Creators. *MC.today, Media for Creators*. URL: https://mc.today (дата звернення: 20.06.2022).
16. Moved. *Moved*. URL: https://docs.oracle.com (дата звернення: 20.06.2022).
17. Spec-Zone.ru. *Spec-Zone.ru*. URL: https://spec-zone.ru (дата звернення: 20.06.2022).
18. Web Creator. Мова програмування. *Web Creator*. URL: https://web-creator.ru/articles/java (дата звернення: 20.06.2022).
19. Welcome to The Apache Software Foundation!. *Welcome to The Apache Software Foundation!*. URL: https://www.apache.org (дата звернення: 20.06.2022).
20. Законодавство України про охорону праці //Збірник нормативних документів у 4 т. -К.: Основа, 2014 р.
21. Гандзюк М. П., Желібо Е. П., Халімовський М. О. Основи охорони праці / За ред.. Гандзюка М. П. - К.: Каравела 2010 - 405 с.
22. «Правила охорони праці під час експлуатації електронно-обчислювальних машин»(закон від 26.03.2010);
23. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, складності та

напруженості трудового процесу, (Наказ МОЗ від 27.12.2006 р. № 528).

24. Москальова В. М. Основи охорони праці. Підручник. - Київ: ВД Професіонал, 2005.-666 с.
25. Ткачук К. Н., Халімовський М. О., Зацарний В.В., та інші. Основи охорони праці: Підручник. -К.: Основа, 2013. -444 с.