АНОТАЦІЯ

Магістерська робота «Розробка об’єктно-орієнтованої системи обробки персоніфікованих даних на платформі iOS для малого та середнього бізнесу» Винник Максим Володимирович, Тернопільський національний технічний університет імені І. Пулюя, факультет комп’ютерно–інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм–61, Тернопіль, 2019.

Пояснювальна записка містить: 74 с. 9 рис., 7 табл., 3 дод..

Метою роботи є розробка об’єктно-орієнтованої системи обробки персоніфікованих даних на платформі iOS.

В ході роботи досліджено та проаналізовано предметну область, визначено ключових акторів системи, спроектовано ефективну базу даних, розроблено ефективні алгоритми обробки даних, застосовано сучасний підхід до розробки, створено зручний дизайн, виконано тестування.

Розроблена система написана на мовах програмування Swift, Objective-C та SQL.

Ключові слова: ФРЕЙМВОРК, SWIFT, COREDATA, ПЕРСОНІФІКОВАНА ІНФОРМАЦІЯ, IOS.

ABSTRACT

Master thesis «Development of an object-oriented system for processing personal data on the iOS platform for small and medium businesses» Maxym Vynnyk, I. Pulyu Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPM–61 Group, Ternopil, 2019.

The explanatory note contains: 74 p. 9 Figure, 7 Table, 3 Add..

The purpose of the work is to object-oriented system for processing personal data on the iOS platform.

In the course of the work the subject area was researched and analyzed, key system actors were identified, an efficient database was designed, effective data processing algorithms have been developed a modern approach to development was applied, a convenient design was created, testing was performed.

The system is written in Swift, Objective-C and SQL programming languages.

Keywords: FRAMEWORK, SWIFT

ЗМІСТ

ВСТУП

У наш час дані – один з найважливіших ресурсів людства, адже у правильному вигляді ці дані можуть надати безцінну інформацію. Вона показує спади та підйоми певних аспектів ведення бізнесу. Обробка даних починається з даних у сирому вигляді та перетворення їх у більш читабельний формат (графіки, документи тощо), надаючи йому форму та контекст, необхідні для інтерпретації кінцевому користувачу.

Дана система використовує спрощені алгоритми та працює з малою кількістю даних, що дозволяє користувачу максимально мінімізувати витрати. Дана система буде надзвичайно корисною починаючим підприємцям, які хочуть побачити свій бізнес в цифровому варіанті, не витрачаючи великі кошти, при цьому зрозумівши над якими аспектами підприємства потрібно попрацювати щоб збільшити свій дохід.

Найважливішими кроками є обробка та інтерпретація даних, адже важливо щоб система обробила дані правильно, з мінімальною кількість похибок. Також не менш важливим є інтерпретація цих даних для кінцевого користувача, бо якою б якісною не була інформація, вона нічого не варта, якщо ніхто не може її зрозуміти.

Призначення системи для малого та середнього бізнесу має на меті економію коштів підприємців. Адже система, яка призначена для великої кількості користувачів потребує просунутих алгоритмів обробки даних, машинного навчання, великих обчислювальних потужностей та зберігання надзвичайно великої кількості інформації. Для цього всього потрібна велика кількість коштів, яких може не бути у підприємців малого чи середнього бізнесу. Тому ця система використовує спрощені алгоритми та працює з малою кількістю даних, що дозволяє користувачу максимально мінімізувати витрати.

1. ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

1.1.1.1 Обробка даних

Обробка даних - це збирання та маніпулювання даними у використаній та бажаній формі. Маніпуляція - це не що інше, як обробка, яка здійснюється або вручну, або автоматично в заздалегідь заданій послідовності операцій. Раніше це робиться вручну, що забирає багато часу і може мати можливість помилок при обробці, тому зараз більшість обробку проводиться автоматично за допомогою комп'ютерів, що робить швидку обробку і дає правильний результат.

Наступним моментом є перетворення в потрібну форму, зібрані дані обробляються та перетворюються в потрібну форму відповідно до вимог програми, що означає перетворення даних у корисну інформацію, яка може використовуватись у додатку для виконання певного завдання. Вхід обробкою - це збір даних з різних джерел, таких як дані текстових файлів, дані файлових файлів Excel, база даних, навіть неструктуровані дані, такі як зображення, аудіокліпи, відеокліпи, дані GPRS тощо.

Результатом обробки даних є змістовна інформація, яка може бути в різних формах, таких як таблиця, зображення, діаграми, графік, векторний файл, аудіо та ін., Всі формати, отримані залежно від необхідного додатку чи програмного забезпечення.

Існує п'ять різних типів обробки даних. Перші два - наукова та комерційна обробка даних - це специфічні типи обробки даних, другі три - це конкретні типи обробки даних.

Спочатку короткий підсумок обробки даних: Обробка даних визначається як процес перетворення необроблених даних у змістовну інформацію.

Обробку даних можна визначити за допомогою наступних кроків:

- Збір даних або збір даних.
- Зберігання даних.
- Перетворення даних (зміна у придатному або єдиному форматі).
- Очищення даних та усунення помилок.
- Перевірка даних (перевірка перетворення та очищення).
- Розділення та сортування даних (малювання моделей, взаємозв'язків та створення підмножини).
- Узагальнення даних та агрегація даних (комбінування підмножин у різних групуваннях для отримання додаткової інформації).
- Представлення даних та звітність.

Існують різні типи методи обробки даних, залежно від того, для чого потрібні дані. Типи обробки даних на стендовому рівні можуть включати статистичну, алгебраїчну, картографічну та графічну схему, метод лісів і дерев, машинне навчання, лінійні моделі, нелінійні моделі, реляційну та нереляційну обробку. Це методологія та методи, які можна застосувати в рамках основних типів обробки даних.

Також п'ять основних ієрархічних типів обробки даних. Це загальні типи систем в аналізі даних.

Перші два ключових типи обробки даних - це наукова обробка даних та комерційна обробка даних.

Наукова обробка даних. При використанні в науковому дослідженні або дослідно-конструкторській роботі набори даних можуть вимагати зовсім інших методів, ніж комерційна обробка даних.

Наукові дані - це особливий тип обробки даних, який використовується в академічній та дослідницькій сферах.

Для наукових даних дуже важливо, щоб не було суттєвих помилок, що сприяють неправильним висновкам. Через це етапи очищення та перевірки можуть зайняти значно більше часу, ніж для комерційної обробки даних.

Наукова обробка даних повинна робити висновки, тому етапи сортування та узагальнення часто потрібно виконувати дуже обережно, використовуючи широкий спектр інструментів обробки, щоб забезпечити відсутність упередженості вибору або неправильних зв’язків.

Наукова обробка даних часто потребує спеціаліста з питань тематики, додатково до експерта з даних, щоб працювати з кількістю.

Комерційна обробка даних має багаторазове використання, і це не обов'язково потребує складного сортування. Вперше він широко використовувався в галузі маркетингу, для застосувань для управління взаємовідносинами з клієнтами, а також для банківських, платіжних та платіжних функцій. Більшість даних, що потрапляють у ці програми, є стандартизованими та дещо захищеними від помилок. Тобто поля захоплення усувають помилки, тому в деяких випадках необроблені дані можуть бути оброблені безпосередньо, або з мінімальним і значною мірою автоматизованою перевіркою помилок.

Комерційна обробка даних зазвичай застосовує стандартні реляційні бази даних та використовує пакетну обробку, проте деякі, зокрема технологічні програми, можуть використовувати нереляційні бази даних.

У комерційній обробці даних все ще існує багато застосунків, які схиляються до наукового підходу, наприклад, прогнозування ринку. Вони можуть вважатися гібридом двох методів.

У основних галузях наукової та комерційної обробки застосовуються різні методи застосування стадій обробки до даних. Три основні типи обробки даних, про які ми будемо обговорювати, - це автоматична / ручна, пакетна та обробка даних у режимі реального часу.

Це може здатися неможливим, але навіть сьогодні люди все ще використовують ручну обробку даних. Функції обробки даних

бухгалтерського обліку можуть виконуватися з ведення книги, опитування клієнтів можуть збиратися та оброблятися вручну, і навіть обробка даних на основі електронних таблиць зараз вважається дещо ручною. У деяких складніших частинах обробки даних для інтуїтивного обґрунтування може знадобитися ручний компонент.

Першою технологією, яка призвела до розвитку автоматизованих систем в обробці даних, були перфокарти, що використовуються при підрахунку перепису. Панч-картки також використовувались в перші дні обробки даних про оплату праці.

Комп'ютери почали використовувати корпорації в 1970-х, коли почала розвиватися електронна обробка даних. Деякі перші програми для автоматизованої обробки даних за допомогою спеціалізованих баз даних були розроблені для управління взаємовідносинами з клієнтами (CRM), щоб сприяти кращим продажам.

Електронне управління даними набуло широкого поширення із впровадженням персонального комп’ютера у 1980-х роках. Електронні таблиці забезпечили просту електронну допомогу навіть для повсякденних функцій управління даними, таких як особистий бюджет та розподіл витрат.

Управління базами даних забезпечило більшу автоматизацію функцій обробки даних, саме тому я називаю електронні таблиці як тепер досить ручний інструмент управління даними. Користувач зобов’язаний обробляти всі дані в електронній таблиці, майже як у ручній системі, допомагають лише розрахунки. Тоді як у базі даних користувачі можуть витягувати відносини даних та звіти відносно легко, за умови правильного керування налаштуваннями та записами.

Автономні бази даних зараз виглядають як метод обробки даних у майбутньому, особливо у комерційній обробці даних. Oracle і Peloton готові запропонувати користувачам більше автоматизації з тим, що називається базою даних «самонавідки». Ця розробка в галузі автоматичної обробки даних у поєднанні з інструментами машинного навчання для оптимізації та

вдосконалення сервісу має на меті полегшити доступ та керування даними кінцевим користувачам без необхідності роботи спеціалізованих спеціалістів даних.

Для економії обчислювального часу до широкого використання архітектури розподілених систем або навіть після неї автономні комп'ютерні системи застосовують методи пакетної обробки. Це особливо корисно у фінансових програмах або там, де дані захищені, наприклад, медичні записи.

Пакетна обробка завершує цілий спектр процесів обробки даних як пакетна, спрощуючи окремі команди, щоб забезпечити дії для декількох наборів даних. Це дещо схоже на порівняння комп’ютерної таблиці з калькулятором. Обчислення можна застосувати за допомогою однієї функції, тобто одного кроку, до цілого стовпця або серії стовпців, даючи кілька результатів від однієї дії. Така ж концепція досягається і при пакетній обробці даних. Ряд дій чи результатів можна досягти, застосувавши функцію до цілого ряду даних. Таким чином часу на обробку комп'ютера набагато менше.

Пакетна обробка може завершити чергу завдань без втручання людини, а системи даних можуть запрограмувати пріоритети певних функцій або встановити час, коли пакетна обробка може бути завершена.

Банки зазвичай використовують цей процес для здійснення транзакцій після закриття бізнесу, коли комп'ютери більше не беруть участь у захопленні даних і можуть бути присвячені функції обробки.

Для комерційного використання багато великих програм для обробки даних потребують обробки в режимі реального часу. Тобто їм потрібно отримати результати з даних саме так, як це відбувається. Одне із застосувань цього, з яким більшість із нас може ідентифікувати, - це відстеження тенденцій на фондовому ринку та валюті. Дані потрібно негайно оновлювати, оскільки інвестори купують в режимі реального часу, а ціни оновлюються за хвилину. Дані про графіки авіакомпаній та придбання

квитків, а також програми GPS-відстеження в транспортних послугах мають аналогічні потреби в оновленнях у режимі реального часу.

Найпоширеніша технологія, що використовується в режимі реального часу - це потокова обробка. Аналітика даних береться безпосередньо з потоку, тобто в джерелі. Якщо дані використовуються для отримання висновків без завантаження та перетворення, процес відбувається набагато швидше.

Методи віртуалізації даних є ще однією важливою розробкою в обробці даних у режимі реального часу, де дані залишаються у вихідному вигляді, і єдину інформацію витягують для обробки даних. Краса віртуалізації даних полягає в тому, що трансформація не потрібна, тому помилка зменшується.

Віртуалізація даних та обробка потоків означають, що аналітику даних можна зробити в режимі реального часу набагато швидше, виграючи багато технічних та фінансових застосувань, скорочуючи час обробки та помилки.

Крім цих популярних технологій обробки даних, є ще три методи обробки, які згадуються нижче.

Ця техніка обробки даних походить від автоматичної обробки даних. Ця методика тепер відома як негайне або нерегулярне поводження з доступом. За цією методикою активність рамки готується під час роботи / обробки. Це можна легко переглянути при постійній підготовці наборів даних. Цей метод обробки підкреслює швидкий внесок обміну даними та безпосередньо з'єднується з базами даних.

Це найчастіше використовується техніка обробки даних. Однак він використовується по всьому світу, де у нас є комп'ютерні установки для збору та обробки даних. Як випливає з назви - багатопроцесорна робота не пов'язана з одним центральним процесором. Завдяки цьому вона має набір декількох процесорів. Оскільки в цей метод включені різні набори пристроїв обробки, тому ефективність результатів дуже корисна. Завдання розбиваються на фрейми і потім надсилаються багатопроцесорам на обробку.

Очікується, що отриманий результат буде за менший час, а вихід збільшиться. Додатковою перевагою є те, що кожен технологічний блок незалежний, тому відмова будь-якого не вплине на роботу інших технологічних блоків.

Цей вид обробки даних повністю базується на Часі. При цьому одну одиницю обробки даних використовують кілька користувачів. Кожному користувачеві виділяються задані терміни, в які їм потрібно працювати над тим же процесором / процесорним блоком. Інтервали поділяються на сегменти і, таким чином, для користувачів, так що не відбувається краху часу, що робить його системою з багатодоступним доступом. Ця техніка обробки також широко застосовується і в основному розважається в стартапах.

1.1.1.2 Персоніфіковані дані

Використання цих даних важливіше, ніж будь-коли. З недавнього переходу від однозначного повідомлення на високо персоналізовані кампанії (ми говоримо більше про те, щоб додати FNAME до електронної пошти), не використовувати дані - це просто не варіант. Клієнти вимагають більше уваги, і вони хочуть знати, що ви слухаєте. Якщо вони відчують, що ви просто надсилаєте їм кампанії з вирізання файлів cookie, ви ризикуєте їх відчужити. Хоча це стосується кожного типу бізнесу незалежно від того, чи є це B2B чи B2C, статистика електронних комерційних інтернет-магазинів вражає.

У опитуванні споживачів персонального опису компанії Magnetic 2015 було встановлено, що половина всіх споживачів бажає, щоб їх особиста інформація використовувалася для координації кращого загального досвіду покупок. Згідно з доповіддю 2014 року від Janrain та Blue Research, надсилання чогось такого простого, як помилкові електронні листи, може мати серйозні наслідки. Серед опитаних цього звіту 94% споживачів, які отримали електронні листи, які не врахували їхні особисті переваги,

вживають заходів. У колосальних 68% звітів вони автоматично видаляють електронні листи, 54% автоматично підписуються, 29% заявляють, що рідше купують товари цієї компанії, а 10% вирішують, що більше ніколи не відвідуватимуть цей веб-сайт.

Давайте розглянемо, як можна використовувати дані для створення персоналізованих кампаній і, що ще важливіше, як ви можете збирати, отримувати доступ та компілювати всі дані для використання в корисному стані для підвищення рентабельності інвестицій.

Терміни та актуальність є основою для будь-якої маркетингової кампанії, незалежно від того, висувається вона через електронну пошту, соціальні медіа, нативну рекламу чи інше. Більшість, якщо не всі, компанії збирають дані з декількох каналів (веб-сайт компанії, система CRM, програмне забезпечення для автоматизації маркетингу, сайти соціальних медіа, програмне забезпечення кол-центру тощо). Усі ці дані при ефективному використанні дадуть маркетологам зрозуміти, які можна використовувати для створення більш вигідних рекламних кампаній. Наприклад, якщо інтернет-магазин взуття знає, що потенційний клієнт шукав пару пішохідних черевиків, компанія може показати акцію переорієнтації «один на один», що рекламує «Пішохідні черевики в продажу».

Завдяки більш доступним даним, маркетологи можуть судити про ефективність вмісту до окремої публікації у Facebook чи Twitter, дозволяючи їм додатково оптимізувати вміст на основі тем, які примушують і залучають клієнтів. Використання даних у реальному часі з соціальних сигналів, таких як подобається, слідкує та твіти, може дати вам набагато більш релевантну та цінніснy інформацію про наміри людини, а не покладатися лише на такі фактори, як історія покупок чи стандартна демографічна характеристика.

Наприклад, ви можете визначити набір людей, які спілкуються з дописом у Facebook про найкращі місця для походу, та надіслати їм електронний лист із найпопулярнішими похідними продуктами, а не

викладати це на всю базу даних та руйнувати стосунки з тими, хто у вашій країні базу даних, які не зацікавлені в конкретній пропозиції.

Пошук потенційної нової аудиторії раніше ґрунтувався на визначенні загальних факторів, таких як географія, вік чи сімейний стан, і припускаючи, що людина, яка поділяє ці подібні риси, мала схожі інтереси. Але цей тип узагальнення не враховує індивідуальних уподобань та поведінки людей - факторів, які набагато важливіші для процесів прийняття рішень. Можливість бачити, як ваша потенційна аудиторія взаємодіє з певним вмістом, може призвести до створення цінного нового колу клієнтів, про якого ви ніколи не знали.

Оскільки компанії, що збирають мільйони точок даних на різних платформах та інструментах, маркетологи стикаються з бар'єрами, але всі вони вирішувані. У прикладі, який ми використовуємо, маркетологу потрібно отримати як оперативні дані, так і дані соціальних медіа (а може бути, дані з ще більшої кількості джерел), щоб скласти ідеальну персоналізовану кампанію. Більшість сучасних сайтів електронної комерції зберігають свої оперативні дані (продажі) в оперативному магазині, як MySQL, а також дані своїх клієнтів у CRM, наприклад Salesforce. Щоб зробити це більш складним, маркетологи можуть мати свої дані онлайн-кампаній у Google Analytics і Google Adwords, Facebook Ads, Twitter Ads і, можливо, навіть у такій службі автоматизації маркетингу, як HubSpot. Щоб отримати повне уявлення про ваших клієнтів, всі дані з усіх цих джерел повинні бути інтегровані.

Пошукові технології вже деякий час використовують обробку великих даних та машинне навчання для покращення результатів пошуку, але лише нещодавно ми зрозуміли, що ці самі методи можна використовувати для персоналізації. Крім того, ми вважаємо, що це підвищує процес персоналізації від спеціального процесу проб і помилок до статистично достовірного процесу, керованого даними, який може забезпечити послідовну та вимірювану віддачу від інвестицій.

Використання історичних даних для створення статистичних моделей. Цей крок вимагає збору історичних даних від усіх користувачів та використання цих даних у межах великої бази даних для створення статистичних моделей, які передбачають ймовірність того, що користувач знайде предмет (документ, веб-сторінку, товар, пункт призначення тощо) корисним.

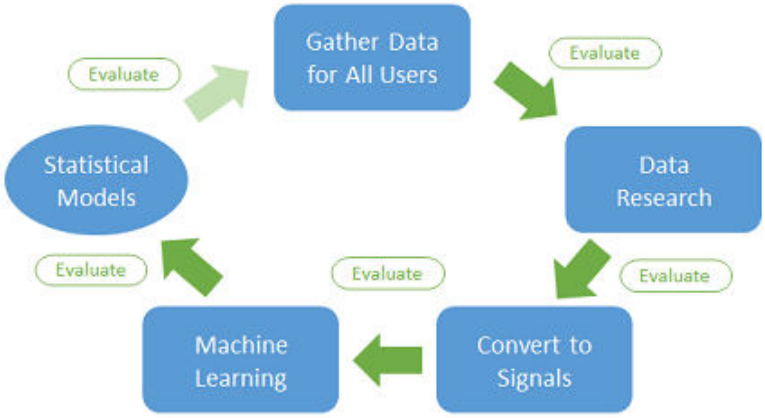

Рисунок 1.1 – Циклічних процес оцінки даних

"Сигнали" у вищезазначеному процесі можуть включати вектори, скаляри, кластери, категорії тощо, які представляють (наскільки це можливо) розміри персоналізації, які сильно співвідносяться з ймовірністю зацікавлення. Зауважте, що генерація хороших сигналів зазвичай становить 90 +% від загальних зусиль для проекту.

Зауважте також, що процес триває, циклічно і вимагає оцінки на кожному кроці. Це процес обміну даними, який вимагає глибокого розуміння сукупності користувачів та пропонування вмісту системи. Дані статистики приведуть до статистичних моделей, які приведуть до більше розуміння, що призведе до кращих моделей у процесі вдосконалення та збільшення релевантності.

Впровадження персоналізації. Як тільки дійсні, перевірені статистичні моделі для персоналізації стануть доступними, їх потрібно використовувати в режимі реального часу для персоналізації результатів для окремих користувачів.

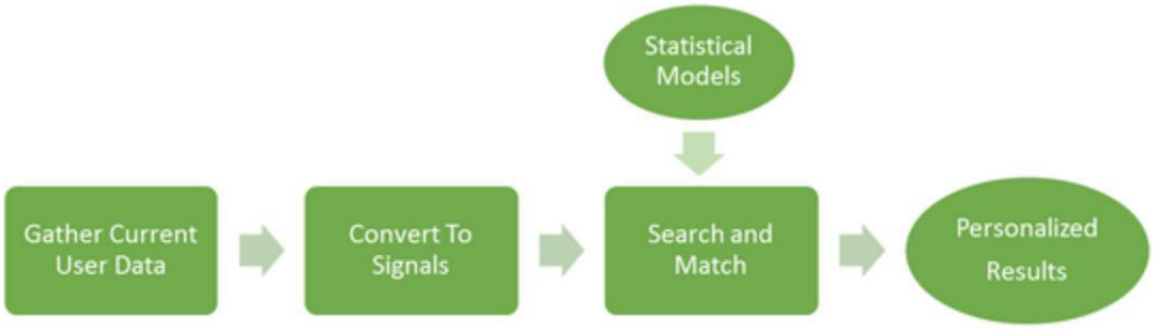

Рисунок 1.2 – Кроки персоналізації інформації для окремого користувача

В ідеалі всі ці кроки повинні відбуватися під час просування користувача через систему. Історичні дані поєднуються з даними з початкового з'єднання, а потім кожна подія (клацання, пошук, покупка тощо) додає до моделі та генерує новий набір сигналів, які додаються до системи пошуку та відповідності.

На практиці деякі сигнали можуть бути занадто обтяжливими для обчислення в режимі реального часу, і, можливо, це потрібно зробити у фоновому режимі, наприклад, як періодичний пакетний процес, який готує сигнали до наступної зустрічі користувача з системою. На практиці деякі сигнали можуть бути занадто обтяжливими для обчислення в режимі реального часу, і, можливо, це потрібно зробити у фоновому режимі, наприклад.

Генерування високоякісних сигналів, які добре співвідносяться з рівнем інтересу користувача та які можна обчислити в режимі реального часу, є нетривіальною проблемою, яка вимагає винахідливості, терпіння та постійної оцінки.

1.1.2 Постановка задачі

Завданням магістерської роботи є розробка об’єктно-орієнтованої системи обробки персоніфікованих даних для малого та середнього бізнесу.

Результатом розробки повинна бути програма, яка працює під управлінням мобільної операційної системи iOS.

Інтерфейс повинен бути легким та інтуїтивно зрозумілим. Важливо щоб уся статистична інформація була подана в максимально зрозумілій формі. Усі графіки повинні відображати мінімально необхідну інформацію, зоб не перенавантажувати кінцевого користувача. Дизайн повен бути мінімалістичним, щоб не відволікати користувача від основного функціоналу, також дизайн повинен слідувати основним принципам системи iOS. Навігація повинна бути мінімалістичною та срозумілою, щоб зменшити запутаність застосунку та зменшити вхідний поріг для майбутніх користувачів. Програма повинна працювати швидко, плавно та без необхідності постійного підключення до інтернет мережі.

Після аналізу предметної області було визначено основний функціонал системи, що повинен бути реалізований в процесі розробки програмної системи.

Ключовими модулями було визнано: модуль обробки даних, модуль відображення графіків, модуль класифікації даних та модуль виведення статистики.

Обробка інформації повинна виконуватись швидкими та спрощеними алгоритмами. Алгоритми та технології обробки даних буде визначено на одному з наступних етапів проектування програмної системи.

Порядок задач та кроків, які необхідно вирішити:

- Визначення акторів та варіантів використання системи
- Обрати модель розробки застосунку
- Обрати архітектурний шаблон додатку
- Побудувати діаграму класів
- Обрати середовище розробки та мову програмування
- Обрати СУБД
- Реалізувати систему
- Повести тестування системи

1.1.3 Пошук акторів та варіантів використання

Ознайомившись з вимогами, було визначено лише одно актора – Користувач. Оскільки в системі не передбачені будь які ролі, рівні доступу, тощо. Усі дії необхідні для повноцінного користування застосунком можливі за участі лише одного користувача. Усі дії необхідні для повноцінного користування застосунком можливі за участі лише одного користувача.

Користувач повинен мати можливість внесення даних в систему, яку далі ця система повинна обробити. Надалі оброблена інформація буде доступна користувачу для перегляду в кількох форматах, які легко сприймаються людиною.

Після визначення акторів, можна приступити до визначення варіантів використання системи. Більшість варіантів використання зображені в таблиці 1.1

Таблиця 1.1 Виявлення варіантів використання

Актор	Назва	Опис
Користувач	Внесення даних для обробки	Користувач повинен могти внести в систему масив даних, записаних в певному форматі, які система надалі проаналізує
Користувач	Видалення існуючих даних	Користувач може видаляти дані, які система використовує в даний момент, щоб мати змогу внести нові дані
Користувач	Перегляд історії маніпуляції даними	Користувач повинен мати змогу переглядати історію змін даних, коли дані були оновлені останній раз, дати коли були видаленні попередні дані тощо.
Користувач	Поділитися даними	Користувач повинен мати змогу поділитися своїми даними через системні елементи, тобто будь які

		програми які підтримують пересилання файлів
Користувач	Перегляд графіка	Користувач повинен мати змогу переглянути графік, який зображує статистичну інформацію, яка базується на даних, внесених Користувачем
Користувач	Зміна типу графіка	Користувач повинен мати змогу змінювати вид графіка, обраний з наданого списку доступних типів, для покращення сприйняття інформації у тій чи іншій ситуації
Користувач	Зближення графіка	Користувач повинен мати змогу зближувати графік, щоб краще розгледіти дрібні деталі
Користувач	Віддалення графіка	Користувач повинен мати змогу віддалити графік, щоб краще бачити більш загальну картину зображеної інформації

Продовження таблиці 1.1

Користувач	Змінна часових проміжків графіка	Користувач повинен мати змогу змінювати часові проміжки, за які відображається даних графік
Користувач	Зміна шкали графіка	Користувач повинен мати змогу змінювати шкалу графіка, за якої дані відображаються
Користувач	Зміна кольору ліній графіка	Користувач повинен мати змогу змінювати колір ліній на графіку, щоб краще виділити певні елементи
Користувач	Зміна заднього фону графіка	Користувач повинен мати змогу змінювати колір заднього фону

		графіка, щоб уникнути зливання кольорів
Користувач	Додавання рисунків користувача	Користувач повинен мати змогу додавати власно створенні рисунки на графік, щоб краще виділити для себе певні аспекти, які відображає графік
Користувач	Збереження налаштувань графіка	Користувач повинен мати змогу зберігати усі свої налаштування графіка, такі як шкала, тип, кольори, рисунки тощо.
Користувач	Скидання налаштувань графіка	Користувач повинен мати змогу скидати налаштування графіка до стандартного, передбаченого системою
Користувач	Паралельний перегляд кількох графіків	Користувач повинен мати змогу відкрити кілька графіків, які будуть знаходитись один над одним, що дасть змогу користувачу їх порівнювати.
Користувач	Накладання графіків	Користувач повинен мати змогу накладати графіки один на одного, щоб точніше порівняти різницю в графіках

Продовження таблиці 1.1

Користувач	Створення зображення з обраного графіка	Користувач повинен мати змогу створити зображення, яке відображає обраний графік
Користувач	Видалення рисунків користувача	Користувач повинен мати змогу видаляти рисунки, раніше створенні ним
Користувач	Перегляд спрощеної статистики	Користувач повинен мати змогу переглядати спрощений варіант статистики, який створений на основі обраних даних

Користувач	Перегляд розширеної статистики	Користувач повинен мати змогу переглядати розширений варіант статистики, створеної на основі обраних даних
Користувач	Генерування документу xls формату	Користувач повинен мати змогу згенерувати документу xls формату, який можна відкрити з допомогою Exel
Користувач	Перегляд статистики обраного елемента даних	Користувач повинен мати змогу обрати певний елемент даних щоб переглянути його статистику
Користувач	Фільтрування елементів даних	Користувач повинен мати змогу фільтрувати елементи які враховуються в статистиці
Користувач	Сортування елементів даних	Користувач повинен мати змогу відсортувати дані за певним показником

Після виявлення акторів та варіантів використання, було створено UML-діаграму варіантів використання(рис. 1.3). На цій діаграмі зображенні усі актори та основні варіанти використання, можливі у даній системі.

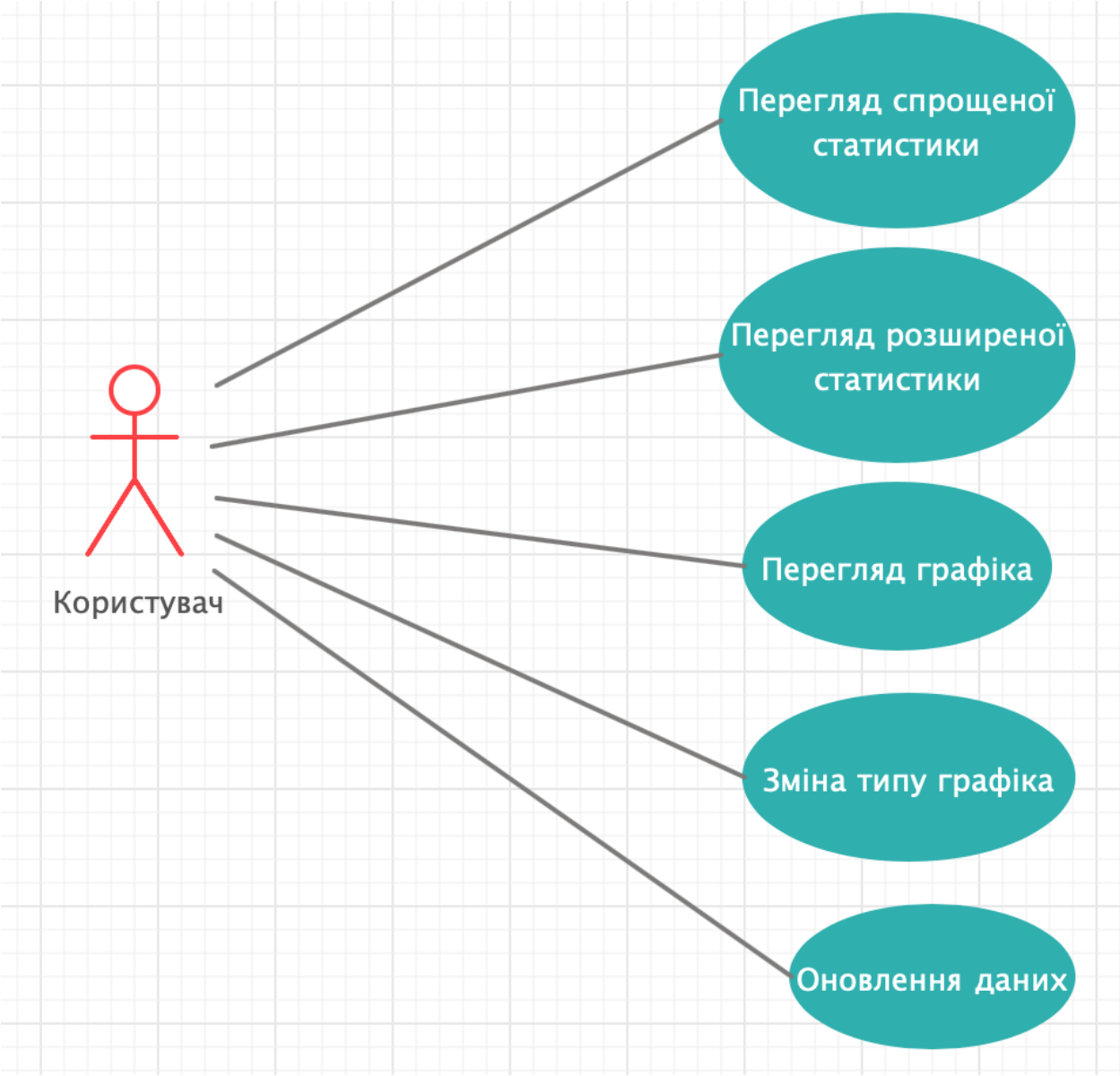

Рисунок 1.3 – Діаграма варіантів використання

1.1.4 Опис ключових варіантів використання

Після цього, було виділено кілька основних варіантів використання, які є ключовими для даної системи: перегляд графіка, додавання даних, перегляд спрощеної статистики.

Варіант використання «Перегляд графіка» описаний у таблиці 1.2

Таблиця 1.2 Опис варіанту використання «Перегляд графіка»

Дійові особи	Користувач, Система
Цілі	Переглянути графік певного елементу даних
Передумова	Система повинна мати коректний масив даних
Успішний сценарій: 1. Користувач відкриває застосунок. 2. Користувач обирає елемент даних. 3. Користувач натискає «Переглянути графік». 4. Система дістає оброблені дані з пам'яті телефону. 5. Система створює масив даних на основі обраного елементу даних. 6. Система підтягує попередні налаштування графіка. 7. Система малює графік.	
Результат	Система успішно збудувала графік
Альтернативні сценарії	
2а	Немає даних у системі. Результат: Система повідомляє користувача що необхідно обрати дані.
6а	Немає попередніх налаштувань графіка Результат: Система застосує стандартні налаштування графіка

Варіант використання «Додавання даних» описаний у таблиці 1.3

Таблиця 1.3 Опис варіанту використання «Додавання даних»

Дійові особи	Користувач, Система
Цілі	Додати нові дані в систему

Передумова	Дані повинні знаходитись у користувача в телефоні. Користувач повинен надати системі доступ до файлів.
Успішний сценарій: 1. Користувач відкриває застосунок. 2. Користувач натискає «Налаштування». 3. Користувач натискає «Додати данні». 4. Система надає запит користувачу про доступ до файлів. 5. Користувач надає права на перегляд файлів. 6. Система показує файловий провідник. 7. Користувач обирає масив даних. 8. Система обробляє дані. 9. Систем показує користувачу повідомлення що дані прийнято.	
Результат	Користувач успішно змінив дані у системі

Альтернативні сценарії	
4а	Користувач не надав доступ до файлів. Результат: Система повідомляє користувача що неможливо обрати файл.
8а	Файл пошкоджено або він невідповідного формату Результат: Система повідомляє користувача, що обрані дані неможливо використати

Варіант використання «Перегляд спрощеної статистики» описаний у таблиці 1.4.

Таблиця 1.4 Опис варіанту використання «Перегляд спрощеної статистики»

Дійові особи	Користувач, Система
Цілі	Переглянути спрощеної статистики про обрані дані
Передумова	Система повинна мати коректний масив даних
Успішний сценарій: 1. Користувач відкриває застосунок. 2. Користувач натискає «Переглянути спрощену статистику». 3. Система дістає оброблені дані з пам’яті телефону. 4. Система генерує спрощену статистику з даних застосунку. 5. Система відкриває нове вікно з відображення спрощеної статистики.	
Результат	Користувач успішно переглянув спрощену статистику

Альтернативні сценарії	
2а	Немає даних у системі. Результат: Система повідомляє користувача що необхідно обрати дані.

1.2 Проектування програмної системи

1.2.1 Вибір моделі розробки

Існує кілька моделей розробки програмного забезпечення, тому потрібно проаналізувати основні з них ті вибрати найбільш актуальну для даного програмного продукту.

Існує численні способи організації процесу розробки та написання коду. Хоча неможливо покласти один шлях в якості єдиного шляху, є чому багато чому навчитися. Дізнайтеся про переваги роботи у різних форматах та процедурних підходах у цьому огляді щодо методів створення коду.

Методології розвитку - це битва між догматизмом та прагматизмом. Догматизм - це люди, які просто мають завзяття - вони кажуть, що цей шлях - це шлях, якщо відхилитися від цього шляху, все втрачено. Прагматизм, зближуючи те, що працює в даний момент.

Для обох є певні переваги. Я вважаю, що люди, які є більш догматичними проти прагматичних, покращують рівень розуміння системи; тому що вони дійсно витрачають багато часу, зосереджуючись на своєму інструменті і що він може зробити та як оптимізувати його. Вони дають кращий сирий продукт. Прагматики можуть подивитися на всі ці сировинні продукти і сказати: я візьму цей шматочок звідти і цей шматочок звідти, і я можу бути більш ефективним, коли справа стосується змін вимог та змін проектів.

Методологія розробки програмного забезпечення - це процес або ряд процесів, що використовуються при розробці програмного забезпечення. Знову ж таки, досить широкий, але це такі речі, як фаза проектування, фаза розробки. Це способи роздуму над речами, як водоспад, як не ітераційний процес. Зазвичай він має форму визначених фаз. Він розроблений для опису життєвого циклу програмного забезпечення.

Це також кодифіковане спілкування. Таким чином, ви фактично встановлюєте набір норм між групою людей, які говорять про те, як ви збираєтеся працювати, і так ви збираєтеся певним чином передавати інформацію між вами; чи це документація, чи це обговорення, чи це малюнки на папері.

Звичайно, не могло бути багато різних методологій розробки програмного забезпечення. Є стільки, скільки ви, можливо, знайдете, і майже будь-коли, коли хтось має, і вирішить навіть трохи змінити його від

існуючого, вони накладуть на нього новий ярлик і назвуть його чимось новим. Це робить його досить важким для всіх різних типів.

Найбільша брехня, коли говорять про методології розробки програмного забезпечення, полягає в тому, що є один переможець - що якщо ви будете використовувати цей процес, він буде отримувати досконаліть кожного разу. І якщо ви не отримаєте хорошого результату, ви самі неправильно застосували процес у якійсь формі. Це найважча частина. Це означає, що це спрацювало б, якби ви дотримувалися цього процесу більш ретельно.

Одне напевно, написання коду та об'єднання його в робочу форму вимагають дисципліни та організованості. Необхідні навички включають команду, яка зосереджена навколо плану дій. Навіть якщо одна людина пише коду, повинна бути певна форма управління, щоб планувати підхід, який працює для успішного результату.

Scrum - це легка гнучка рамка управління проектами, яка широко застосовується для управління та контролю ітеративних та поступових проектів усіх типів. Кен Швабер, Майк Бідл, Джефф Сазерленд та інші зробили вагомий внесок у еволюцію Scrum за останнє десятиліття. Компанія Scrum здобула все більшу популярність у спільній розробці програмного забезпечення завдяки своїй простоті, доведеній продуктивності та здатності виконувати функції обгортки для різних інженерних практик, що просуваються іншими спритними методологіями.

За методологією Scrum "Власник продукту" тісно співпрацює з командою, щоб визначити та визначити пріоритетність функціональності системи у вигляді "Затримки продукту". Блокування продукту складається з функцій, виправлень помилок, нефункціональних вимог тощо - все, що потрібно зробити для успішної доставки працюючої системи програмного забезпечення. Завдяки пріоритетам, керованим Власником продукту, багатофункціональні команди оцінюють та підписуються, щоб доставляти "потенційно придатні для збільшення" програмні засоби під час послідовних

спринтів, як правило, тривалістю 30 днів. Щойно Блокування продукту Sprint буде здійснено, додаткові функції не можуть бути додані до Sprint, за винятком команди. Після доставки Sprint, при необхідності аналізується та репріоритизується «Product Backlog», а для наступного Sprint вибирається наступний набір функціональних можливостей.

Доказано, що методологія Scrum розширюється до декількох команд в дуже великих організаціях з 800+ людьми. Подивіться, як VersionOne підтримує Scrum Sprint Planning, полегшивши керування реєстром продуктів.

Lean Development Software - це ітеративна гнучка методологія, спочатку розроблена Мері та Томом Поппендієком. Lean Software Development значною мірою зобов'язана руху Lean Enterprise та практиці компаній, таких як Toyota. Lean Software Development фокусує свою команду на наданні вартості клієнту та на ефективності "потоку вартості", механізмів, які надають цю цінність. Основні принципи методу Lean включають:

- Усунення марнування часу
- Посилення навчання
- Вирішити як можна пізніше
- Виконати якомога швидше
- Розширення можливостей команди
- Побудова цілісності
- Бачити ситуацію в цілому

Leand методологія виключає лишні трати часу за допомогою такої практики, як вибір лише справді цінних функцій для системи, надання пріоритету вибраним та доставку їх невеликими партіями. Він підкреслює швидкість та ефективність робочого процесу розвитку та покладається на швидкий та надійний зворотний зв'язок між програмістами та замовниками.

Lean використовує ідею, щоб робочий продукт "витягувався" через запит клієнта. Він фокусує повноваження та можливості прийняття рішень на окремих людей та невеликих колективах, оскільки дослідження показують, що це швидше та ефективніше, ніж ієрархічний потік контролю. Lean також

зосереджується на ефективності використання командних ресурсів, намагаючись забезпечити, щоб всі були максимально продуктивними якнайбільше часу. Він зосереджений на одночасній роботі та найменших можливих залежностях робочого процесу в команді. Lean також наполегливо рекомендує одночасно писати автоматизовані одиничні тести.

Метод Kanban використовується організаціями для управління створенням продуктів з акцентом на постійну доставку, не перевантажуючи команду розробників. Як і Scrum, Kanban - це процес, призначений допомогти командам ефективніше працювати разом.

Канбан базується на 3 основних принципах:

- Візуалізуйте, чим займаєтесь сьогодні (робочий процес): бачити всі предмети в контексті один одного може бути дуже інформативним
- Обмежте кількість незавершеної роботи (WIP): це допомагає збалансувати підхід, орієнтований на потоки, щоб команди не починали і не брали на себе занадто багато роботи відразу
- Підвищити потік: коли щось закінчено, наступна найвища річ із відставання запускається в роботу

Kanban сприяє постійній співпраці та заохочує активне, постійне навчання та вдосконалення, визначаючи найкращий можливий робочий процес у команді. Подивіться, як VersionOne підтримує розробку програмного забезпечення Kanban.

Екстремальне програмування (XP), спочатку описаний Кентом Беком, став однією з найпопулярніших і суперечливих спритних методологій. XP - це дисциплінований підхід до швидкої та постійної доставки якісного програмного забезпечення. Це сприяє високій участі клієнтів, швидкому циклу зворотного зв’язку, постійному тестуванню, постійному плануванню та тісній командній роботі для доставки робочого програмного забезпечення з дуже частими інтервалами, як правило, кожні 1-3 тижні. Оригінальний

рецепт XP базується на чотирьох простих значеннях - простота, спілкування, зворотній зв'язок та сміливість - та дванадцять допоміжних практик:

- Невеликі випуски
- Тести прийняття клієнта
- Простий дизайн
- Парне програмування
- Тестова розробка
- Рефакторинг
- Постійна інтеграція
- Право власності на колективний код
- Стандарти кодування
- Метафора
- Стійкий темп

Дон Веллс зобразив процес XP на популярній схемі. У XP "Замовник" дуже тісно співпрацює з командою розробників для визначення та визначення пріоритетності деталізованих одиниць функціональності, згаданих як "Історії користувачів". Команда розробників оцінює, планує та надає користувацькі історії з найвищим пріоритетом у формі робочого, перевіреного програмного забезпечення на основі ітерації за ітерацією. Для того, щоб досягти максимальної продуктивності, практика забезпечує сприятливу та легку основу для керівництва командою та забезпечення якісного програмного забезпечення.

Методологія Crystal - це один із найлегших, адаптованих підходів до розробки програмного забезпечення. Crystal насправді складається з сімейства гнучких методологій, таких як Crystal Clear, Crystal Yellow, Crystal Orange та інші, унікальні характеристики яких обумовлені кількома факторами, такими як розмір команди, критичність системи та пріоритети проекту. Ця сім'я Crystal стосується усвідомлення того, що кожен проект може вимагати дещо адаптованого набору політик, практик та процесів, щоб відповідати унікальним характеристикам проекту. Деякі з основних

принципів Crystal включають командну роботу, спілкування та простоту, а також роздуми, щоб часто коригувати та вдосконалювати процес. Як і інші гнучкі методології процесів, Crystal сприяє ранній, частої доставці робочого програмного забезпечення, високій участі користувачів, адаптованості та усуненню бюрократії чи відволікань. Алістер Кокберн, засновник компанії Crystal, випустив книгу «Кришталево ясна: Методологія для малих команд».

Метод розвитку динамічних систем (DSDM), починаючи з 1994 р., Виникла з необхідності забезпечити стандартну рамкову структуру для реалізації проектів для того, що тоді називалося швидким розвитком додатків (RAD). Хоча RAD був надзвичайно популярним на початку 1990-х років, підхід RAD до доставки програмного забезпечення розвивався досить неструктуровано. Як результат, Консорціум DSDM був створений та скликаний у 1994 році з метою розробки та просування загальної галузевої бази для швидкої доставки програмного забезпечення. Починаючи з 1994 року, методологія DSDM розвивалася і дозрівала, щоб забезпечити всебічну основу для планування, управління, виконання та масштабування гнучких процесів розробки програмного забезпечення та ітераційних програм. DSDM базується на дев'яти ключових принципах, які в основному обертаються навколо потреб / цінності бізнесу, активного залучення користувачів, розширених повноважень команд, частої доставки, інтегрованого тестування та співпраці зацікавлених сторін. DSDM спеціально називає "придатність для ділових цілей" як основний критерій доставки та прийняття системи, акцентуючи увагу на корисних 80% системи, які можна розгорнути за 20% часу. Вимоги визначаються на високому рівні на початку проекту. Переробка вбудовується в процес, і всі зміни в розвитку повинні бути оборотними. Вимоги плануються та доставляються у короткі строки фіксованої довжини, які також називаються ітераціями, а вимоги до проектів DSDM визначаються пріоритетом за допомогою правил MoSCoW:

M - повинен мати вимоги S - повинен мати, якщо це можливо, C - міг би мати, але не критично W - не мав би цього разу, але, можливо, пізніше

Усі критичні роботи повинні бути завершені у проекті DSDM. Важливо також, що не кожна вимога в проекті чи часовій рамці вважається критичною. У межах кожного часового вікна включаються менш критичні елементи, щоб у разі необхідності їх можна було зняти, щоб не впливати на вимоги з більш високим пріоритетом на графік. Рамка проектів DSDM не залежить і може бути реалізована разом з іншими ітераційними методологіями, такими як Екстремальне програмування та Раціональний єдиний процес.

Функціональна розробка (FDD) гнучкої методології був спочатку розроблений та сформульований Джефф Де Лука, за участю М. А. Раджасіма, Ліма Бака Ві, Поля Сего, Джона Керна та Стівена Палмера. Перші втілення FDD сталися в результаті співпраці між Де Лука і лідером OOD Пітером Коадом. FDD - це керований моделлю процес короткої ітерації. Починається з встановлення загальної форми моделі. Потім він продовжує серію двотижневих ітерацій "конструювання за особливістю, побудова за характеристикою". Особливості невеликі, «корисні в очах клієнта» результати. FDD розробляє решту процесів розробки навколо доставки функцій, використовуючи наступні вісім практик:

- Моделювання об'єктів домену
- Розробка за особливістю
- Власність компонента / класу
- Функціональні команди
- Перевірки
- Управління конфігурацією
- Регулярні побудови
- Видимість прогресу та результатів

FDD рекомендує конкретні практики програмістів, такі як "Регулярні збірки" та "Компонент / клас власності". Прихильники FDD стверджують, що вона масштабується більш просто, ніж інші підходи, і краще підходить для великих команд. На відміну від інших гнучких методів, FDD описує

конкретні, дуже короткі фази роботи, які необхідно виконати окремо за особливістю. До них відносяться проходження домену, дизайн, перевірка дизайну, перевірка коду, перевірка коду та сприяння створенню.

Agile - одна з найбільш широко використовуваних та визнаних рамок у розробці програмного забезпечення у світі.

Більшість організацій прийняли його в тій чи іншій формі, але ще існує довгий шлях до зрілості програм їх прийняття.

Існуючі на той час методи водоспаду були занадто громіздкими і не мали можливості для зворотного зв'язку, поки готовий продукт не був готовий до доставки. Його називали моделлю розвитку водоспаду, оскільки спочатку команди повністю виконали один крок і лише після цього вони перейшли до наступного кроку.

Це означало, що коригування курсу не було, і замовник не мав погляду на прогрес, поки весь продукт не був готовий. І саме цього хотіли уникнути ці експерти. Вони хотіли рішення, яке могло б мати можливість для постійного зворотного зв’язку, щоб уникнути витрат на переробку на більш пізньому етапі.

І тому гнучкість також полягає в тому, щоб бути адаптивним і постійним удосконаленням, наскільки це стосується постійного зворотного зв'язку та швидкості доставки.

1.2.2 Вибір архітектурного паттерну програми

Шаруватий візерунок, мабуть, є одним з найбільш відомих моделей архітектури програмного забезпечення. Багато розробників використовують його, не знаючи насправді його назви. Ідея полягає в тому, щоб розділити свій код на "шари", де кожен шар несе певну відповідальність і надає послугу більш високому шару.

Немає заздалегідь визначеної кількості шарів, але саме такі ви бачите найчастіше:

- Презентаційний або користувальницький шар
- Прикладний шар
- Бізнес або доменний шар
- Постійність або рівень доступу до даних
- Шар бази даних

Ідея полягає в тому, що користувач ініціює фрагмент коду в шарі презентації, виконуючи певні дії (наприклад, натискаючи кнопку). Потім презентаційний шар викликає нижчий рівень, тобто прикладний шар. Потім ми переходимо до бізнес-шару і, нарешті, шар стійкості зберігає все в базі даних. Тож вищі шари залежать від та здійснюють дзвінки до нижчих. Схема шарів зображена на рисунку 1.4.

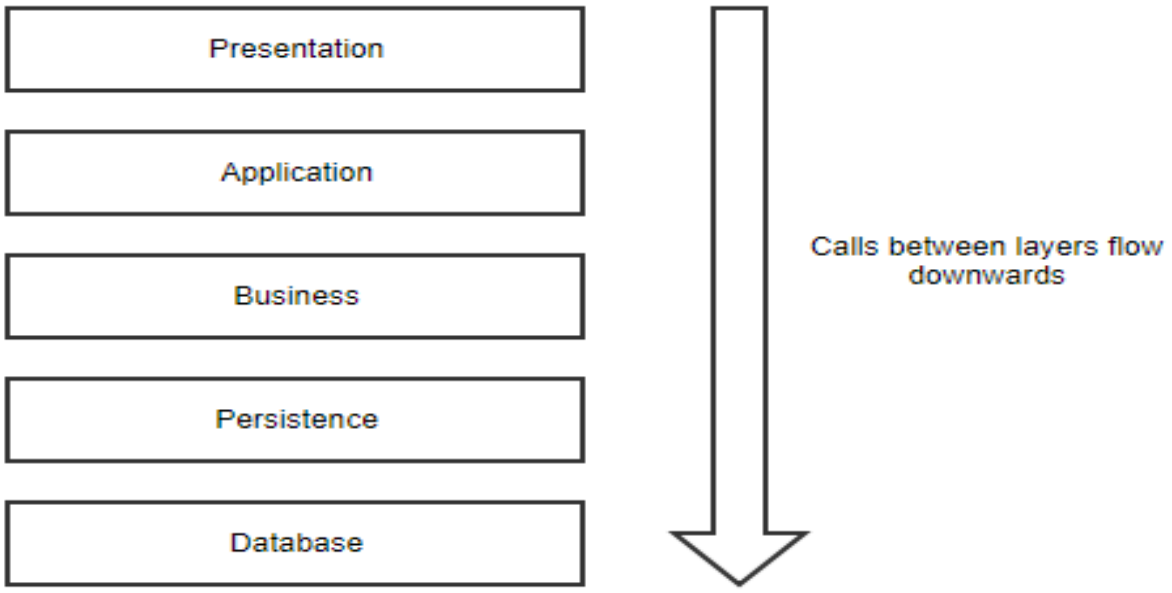

Рисунок 1.4 Схема викликів між шарами

Ви побачите варіанти цього, залежно від складності програм. Деякі програми можуть опускати рівень програми, а інші додавати шар кешування. Можна навіть злити два шари в один. Наприклад, шаблон ActiveRecord поєднує шари бізнесу та наполегливості.

Як було сказано, кожен шар несе свою відповідальність. Презентаційний шар містить графічний дизайн програми, а також будь-який

код для обробки взаємодії з користувачем. Ви не повинні додавати логіку, яка не є специфічною для користувальницького інтерфейсу в цьому шарі.

У бізнес-шарі ви розміщуєте моделі та логіку, характерну для бізнес-проблеми, яку ви намагаєтеся вирішити.

Прикладний шар розташований між шаром презентації та бізнес-рівнем. З одного боку, він забезпечує абстракцію, щоб презентаційному шару не потрібно було знати бізнес-рівень. Теоретично ви можете змінити стек технологій презентаційного шару, не змінюючи нічого іншого у вашій програмі. З іншого боку, у прикладному шарі передбачено місце для встановлення певної логіки координації, яка не вписується в бізнес або презентаційний шар.

Нарешті, рівень стійкості містить код для доступу до рівня бази даних. Шар бази даних є базовою технологією бази даних. Шар стійкості - це набір коду для управління базою даних: оператори SQL, деталі з'єднання тощо.

Переваги:

- Більшість розробників знайомі з цією схемою.
- Це забезпечує простий спосіб написання добре організованої та перевіреної програми.

Недоліки:

- Це, як правило, призводить до монолітних застосувань, які важко розділити згодом.
- Часто розробники виявляють велику кількість коду, щоб пройти через різні шари, не додаючи значення в ці шари. Якщо все, що ви робите, це написання простої програми CRUD, шаруватий візерунок може бути для вас непосильним.

Ідеально підходить для:

- Стандартні додатки для бізнесу, які роблять більше, ніж просто CRUD-операції

Мікрокернел. Шаблон мікроядра або модуль плагінів корисний, коли у вашої програми є основний набір обов'язків і колекція змінних деталей збоку.

Мікроядер забезпечить точку входу та загальний потік програми, навіть не знаючи, що роблять різні плагіни. Схема мікроякдра зображена на рисунку 1.5.

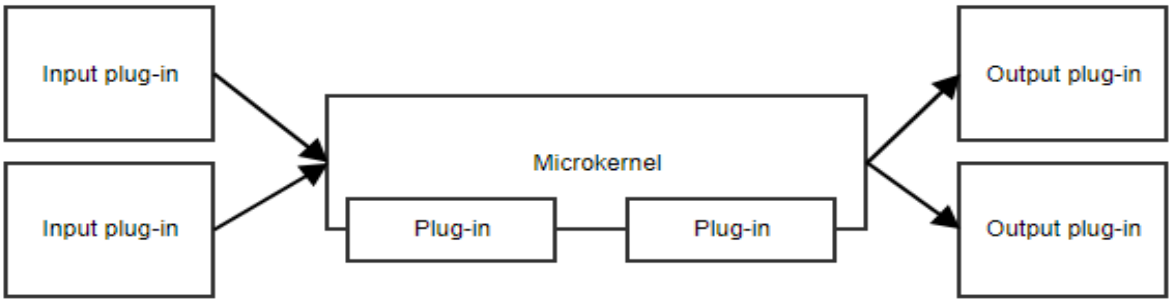

Рисунок 1.5 Схема мікроядра

Прикладом є планувальник завдань. Мікроядер може містити всю логіку планування та запуску завдань, тоді як плагіни містять конкретні завдання. Поки плагіни дотримуються заздалегідь заданого API, мікроядер може запускати їх, не знаючи деталей реалізації.

Інший приклад - робочий процес. Реалізація робочого процесу містить такі поняття, як порядок виконання різних етапів, оцінка результатів кроків, вирішення того, що буде наступним кроком тощо. Конкретна реалізація кроків менш важлива для основного коду робочого процесу.

Переваги:

- Ця модель забезпечує велику гнучкість та розширюваність.
- Деякі реалізації дозволяють додавати плагіни під час роботи програми.
- Microkernel та плагіни можуть бути розроблені окремими командами.

Недоліки:

- Визначити, що належить до мікроядер, а що ні, може бути важко.
- Заздалегідь визначений API може не підходити для майбутніх плагінів.

Ідеально підходить для:

- Програми, які беруть дані з різних джерел, перетворюють ці дані та записують їх у різні напрямки
- Програми робочого процесу
- Додатки та програми планування завдань

CQRS - абревіатура для розділення відповідальності за команду та запити. Основна концепція цієї схеми полягає в тому, що додаток має операції зчитування та запису, які повинні бути повністю розділені. Це також означає, що модель, що використовується для операцій запису (команд), буде відрізнятися від моделей читання (запитів). Крім того, дані будуть зберігатися в різних місцях. У реляційній базі даних це означає, що будуть таблиці для командної моделі та таблиці для прочитаної моделі. Деякі реалізації навіть зберігають різні моделі в абсолютно різних базах даних, наприклад SQL Server для командної моделі для моделі читання.

Ця модель часто поєднується з пошуку подій, про які ми розповімо нижче.

Коли користувач виконує дію, програма надсилає команду службі команд. Командна служба отримує будь-які потрібні їй дані з бази даних команд, здійснює необхідні маніпуляції та зберігає їх у базі даних. Потім він повідомляє службу читання, щоб модель читання могла бути оновлена. Цей потік можна побачити на рисунку 1.6.

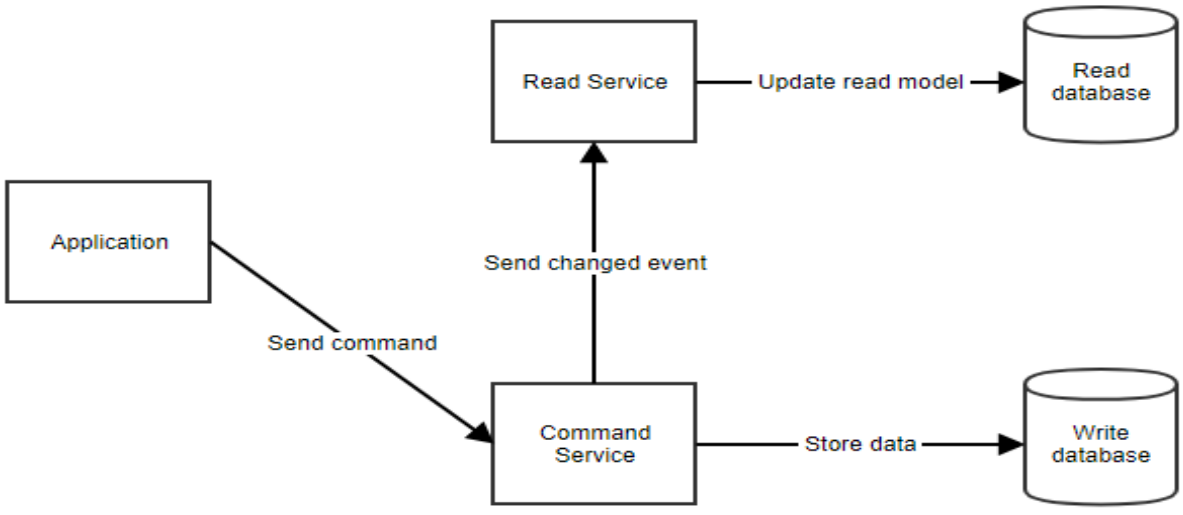

Рисунок 1.6 Потік команд

Архітектура Clean Swift використовує цикл VIP, щоб допомогти вам розділити логіку у вашій програмі. Цикл VIP складається з ViewController, Interactor та Presenter. Усі класи відповідають за певну логіку. ViewController відповідає за логіку відображення, Інтерактор відповідає за логіку бізнесу, а Презентатор відповідає за логіку презентації. Зображення нижче описує потік циклу VIP. Схема паттерну зображена на рисунку 1.7.

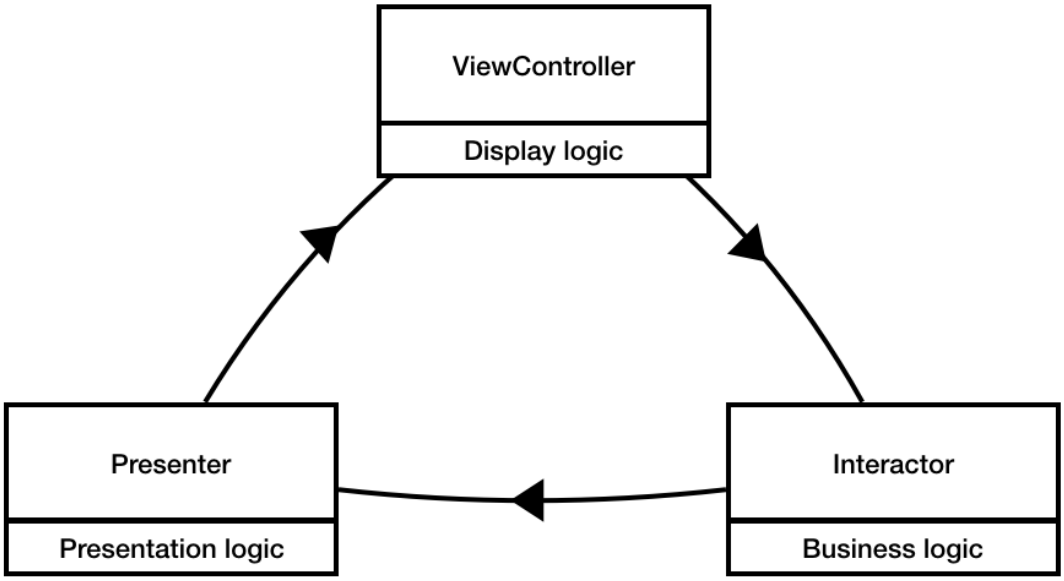

Рисунок 1.7 Схема паттерну VIP

ViewController є першим класом, де виконується дія. Клас відповідає за керування видами в ViewController. У ViewController повинні бути вказані всі розетки та функції IBAction.

Як тільки розпочнеться дія у ViewController, ViewController передасть цю дію Інтерактору. Інтерактор - це клас, в якому повинні бути реалізовані всі випадки використання. Роблячи це, Interactor - це клас, який містить всю логіку бізнесу. Це велика перевага під час написання одиничних тестів,

оскільки при тестуванні всіх інтеракторів перевіряється вся логіка бізнесу у вашому додатку.

Інтерактор несе відповідальність за обробку запиту і поверне об'єкт презентатору. Презентатор - це клас, який відповідає за представлення об'єкта, що генерується Інтерактором. Презентатор розбере цей об'єкт на об’єкт ViewModel і поверне об'єкт ViewController для відображення.

Використовуючи архітектуру Clean Swift, ви точно знаєте, яка логіка повинна розташовуватися в якому класі. Це робить ваш код більш ретельним, тому що, коли вам потрібно вирішити помилку або хочете додати більше функціональних можливостей, ви точно знаєте, де у вашому коді слід внести зміни.

Під час циклу VIP кожен клас додасть об'єкт даних при запиті дії від іншого класу циклу. Коли ViewController просить інтерактор запросити дію від Інтерактора, ViewController додасть об’єкт "Запити". Цей об'єкт містить усі дані, необхідні інтерактору для ділової логіки.

Інтерактор буде обробляти запит. Після того як дані будуть оброблені, він поверне об’єкт "Відповідь" презентатору. Презентатор розбере дані, необхідні об’єкту "ViewModel", який буде надісланий ViewController. Коли ViewModel приймає ViewController, ViewController потрібно лише оновити елементи інтерфейсу з даними в ViewModel.

У вашій програмі можуть виникнути ситуації, що ViewController представить інший ViewController. У Clean Swift навігація між ViewControllers здійснюється маршрутизатором.

Якщо для ViewController доступні варіанти навігації, до ViewController додається клас маршрутизатора. Клас маршрутизатора містить усі параметри навігації, де цей конкретний ViewController може переходити через коридори. Роблячи це, майже кожен VIP-цикл матиме маршрутизатор, що дає зрозуміти, які варіанти навігації має ViewController.

Презентатор відповідає за логіку презентації. Коли відбудеться перехід екрану, ViewController повинен запитати про це Інтерактора, а потім

Інтерактор повинен запитати це у презентатора. Звідти презентатор вирішить, що ViewController може перейти до наступного ViewController за допомогою маршрутизатора.

Якщо вся бізнес-логіка знаходиться в Інтеракторі, може статися така ситуація, коли Інтерактор буде дуже великим класом. Щоб запобігти цьому, Інтерактор може скористатися кількома працівниками. Робітник - це помічник Інтерактора, який може допомогти в отриманні даних.

Worker несе відповідальність за створення об'єктів та здійснення мережевих дзвінків. Крім того, Worker може використовуватися для впровадження сторонніх програм SDK у вашій заявці. Наприклад, якщо ви використовуєте Alamofire для виконання мережевих запитів, але робите всі мережеві запити у працівників, Alamofire SDK потрібно лише імпортувати у працівника.

Працівники повинні бути загальними, а це означає, що декілька інтеракторів можуть використовувати їх у разі потреби для обробки даних.

Отже зважаючи на переваги та недоліки кожної з представлених архітектур, було прийнято рішення використовувати архітектуру Clean Swift (VIP). Так як вона найкраще підходить для написання застосунку на платформі iOS.

1.2.3 Побудова діаграми класів

Оскільки основними сутностями програми будуть дані, графік, проста статистика, розширена статистика та оброблювач інформації, то було вирішено створити відповідні класи для цих сутностей: UserData, DataChart, SimpleStatistic, ExtendedStatistic, DataParser.

Також для роботи з користувачем необхідні класи представлення, які будуть мати змогу відображати необхідну інформацію. Для цього було створено класи: DataChartViewController, SimpleStatisticViewController,

ExtendedStatisticViewController. Для відображення можливості зміни даних також було створено клас DataSelectorViewController.

2 Реалізація програми

2.1 Конструювання програмної системи

2.1.1 Вибір мови та середовища розробки

В написанні програми одним з основних питань є те якою мовою програмування буде написана ця програма. Для написання програми на платформі iOS існує дві основні мови програмування: Objective-C та Swift. Тому необхідно розглянути недоліки та переваги кожної мови щоб зрозуміти яка з них підходить найкраще.

Для того, щоб порівняння було максимально неупередженим, його потрібно проводити, використовуючи ті самі критерії. Короткі результати кожного етапу порівняння представлені в таблиці нижче та детально описані в наступних параграфах.

Розглянемо детальніше кожну з цих відмінностей та знайдемо відповіді на кілька часто заданих питань.

Найбільш очевидним критерієм порівняння програмування Swift та кодування Objective-C є швидкість отримання додатків. Objective-C набагато більш зрілий і добре перевірений, ніж Swift. З цієї причини "Це об'єктивний C швидше Swift?" - одна з найбільш обговорюваних тем серед початківців розробників iOS. Так, у своєму нинішньому стані Swift більш-менш перевершує Objective-C у більшості сценаріїв використання. Але давайте поглибимось у цю проблему: відповідь може вас здивувати.

Як випливає з назви, Swift був розроблений таким чином, що забезпечує високу швидкість як один із основних пріоритетів. Apple заявляє, що Swift приблизно в 2,6 рази швидше, ніж Objective-C і в 8,4 рази швидше, ніж Python. Однак фактичний досвід використання не завжди відповідає офіційній статистиці.

Спочатку Swift пропонував приблизно таку ж продуктивність, що і мови на базі C, включаючи Objective-C. Однак, завдяки постійним вдосконаленням, впровадженим у нових випусках, ця молода мова помітно швидша, ніж її старий конкурент, а також попередні версії. З іншого боку, слід зазначити, що Objective-C все ще компілює код швидше, ніж Swift.

Завдяки своєму традиційному синтаксису кодування в Objective-C передбачає часте використання багатьох службових символів, особливо - крапки з комою, парентези, дужки тощо. На відміну від цього, Swift пропонує простий та виразний синтаксис, який має багато особливо затребуваних розробниками функцій. Наприклад, Apple розробила Swift як "безпечну для типу" мову, яка має власні варіанти типів, присутні в Objective-C, а також представила кілька унікальних, наприклад, кортежі та поняття необов’язкового типу.

Простота синтаксису приносить ряд важливих переваг, таких як менша кількість рядків коду, краща читабельність, низький рівень входу для розробників тощо. Ці отримані переваги будуть описані більш детально нижче.

Swift був розроблений з огляду на максимальну чіткість коду. Простий синтаксис значно полегшує програмування, покращує читабельність коду та робить Swift легшим для засвоєння мови порівняно з Objective-C. Однак нижчий рівень для нових програмістів не перетворив Swift у якусь "мову кодування iPhone для дітей"; вона все ще потребує великої відданості, старанності та таланту.

З іншого боку, вражаючий вік Objective-C призвів до створення величезної бази знань, яка містить посібники та поради для широкого кола проблем та рівнів кваліфікації. Хоча Swift вважається загалом легшим для читання та вивчення, ніж «Objective-C», останній все ще має більший обсяг навчальних матеріалів для будь-якого потенційного учня.

Крім кращої читабельності, код Swift також є більш компактним порівняно з Objective-C. Перш за все, він має засоби для позбавлення від префіксів класу, і для цього не потрібні файли заголовків; ця можливість зменшує кількість файлів проекту вдвічі порівняно з Objective-C. Крім того, простота синтаксису Swift дозволяє реалізувати функцію з меншими рядками коду, ніж за допомогою функції Objective-C для тієї ж функції.

У Objective-C неправильне використання нульових покажчиків може потенційно зламати код, не фактично вибиваючи додаток або показуючи помилку програмісту. Що ще гірше, вказівники можуть бути потенційними вразливими кодами, які загрожують безпеці програми через їх "мовчазну" поведінку, наприклад, коли метод викликається нульовим покажчиком.

На відміну від цього, кожен раз, коли Swift намагається обробити поганий фрагмент коду, він запускає помилку компілятора. Ця функція спрощує відстеження помилок та дозволяє виправляти помилки коду безпосередньо перед тим, як програмний проект перейде до подальшої стадії розробки. Цей підхід, що підходить до помилок, що є результатом методик поводження з помилками у поєднанні з покращеною системою набору тексту, підвищує безпеку та економить час, необхідний для забезпечення стабільності програми.

За свою 35-річну історію «Objective-C» природно створила численну спільноту відданих програмістів. Через тривалість даного періоду багато з них є експертами, які мали достатньо часу, щоб відточити свою майстерність. Ще одна важлива особливість, яка приваблює як розробників програмного забезпечення, так і їх клієнтів до Objective-C, - це доведена стабільність мови в руках майстерного майстра, який може обробляти складний синтаксис та використання покажчиків, схильних до помилок.

Відносно коротка історія Свіфта характеризується сплеском популярності, що призвело до швидко зростаючої кількості підписників і однаково зростаючого обсягу навчального контенту, доступного як у мережі, так і поза нею. Всього через рік після виходу Swift вдалося завоювати титул "Найулюбленіша мова кодування" на відомому веб-сайті Stack Overflow, а через рік, у 2016 році, зайняв друге місце.

Станом на 2019 рік Свіфт вже не серед лідерів у категоріях "Найпопулярніші" або "Найпопулярніші"; однак він все ще перемагає Objective-C значною мірою в цих рейтингах. Варто згадати, що з 2016 року Objective-C пробивається через список мов програмування "Найстрашніші" та потрапив на друге місце у 2019 році.

І Swift, і Objective-C підтримують автоматичний підрахунок посилань як функцію управління пам'яттю. Спочатку Objective-C 2.0 мав функцію збору сміття, яку згодом було замінено на користь ARC, тоді як Swift підтримував ARC з самого початку. Хоча ARC вважається вдосконаленням у порівнянні з відстеженням сміття, він не виконує автоматичну обробку еталонних циклів. Тобто, якщо об’єкт має чіткі посилання, що ведуть до нього, ARC не може перерозподілити згаданий об'єкт.

Objective-C використовує статичні бібліотеки, які, як правило, інтегруються у виконуваний файл програми і не можуть бути змінені без перекомпіляції відповідного файлу .exe. Таким чином, така інтеграція може захистити бібліотеку від підробок або пошкоджень, зробивши додаток більш надійним. Крім того, вбудовані статичні бібліотеки збільшують швидкість

виконання відповідних програм, оскільки вони є невід'ємною частиною програмного забезпечення.

Навпаки, Swift використовує динамічні бібліотеки, які знаходяться поза файлом .exe програми, і ними можна ділитися між різними програмами. Зважаючи на їх розташування, динамічні бібліотеки можуть оновлюватися окремо від відповідних додатків і частіше, ніж статичні бібліотеки, що є величезною вигодою як для розробників, так і для користувачів програмного забезпечення. Одна динамічна бібліотека може використовуватися декількома програмами та може завантажуватися на вимогу. Ця функція дозволяє розробникам зменшувати розмір програм, оскільки їм не потрібно інтегрувати бібліотеки в додатки під час компіляції.

Обидві мови офіційно підтримуються Apple як справді рідні мови для їх операційних систем. Вони обидва використовують ID Xcode IDE, який із кожним оновленням приносить нові функції та виправлення всьому інструментарію розробки, включаючи компілятор Clang та компілятор Swift. Наприклад, Apple пропонує основу Core ML 3 для побудови програм машинного навчання за допомогою Xcode. Варто також зазначити, що Apple Cocoa Touch, рамка UI для розробки додатків для iOS, в основному написана на Objective-C.

У 2019 році Swift 5 нарешті досяг довгоочікуваної стійкості ABI, а це означає, що фаза хаотичного розвитку та хворобливих переходів між мовними версіями закінчилася. Стабільний бінарний інтерфейс програми означає кращу сумісність для поточної та майбутніх версій Swift та дозволяє включати стандартні бібліотеки в операційні системи.

З технічної точки зору можна комбінувати Swift та Objective-C в одній програмі незалежно від мови оригіналу проекту. Apple увімкнула цю можливість у Xcode за допомогою з'єднання файлів заголовків. Таким чином, розробники можуть використовувати найоптимальнішу мову для кожної конкретної потреби та просто змішувати їх разом у компіляторі. Однак, якщо

проект значною мірою покладається на велику кількість бібліотек C ++, об'єктив-C буде кращим, ніж Swift.

Ще один важливий факт, про який повинні пам’ятати розробники, це те, що Swift підтримує лише наступні або новіші версії цільових операційних систем: macOS 10.9.0, iOS 7.0, watchOS 2.0 та tvOS 9.0. Якщо додаток необхідно розробити для старих платформ, програмістам доведеться використовувати Objective-C.

Порівняння Swift з Objective-C доводить, що на цей момент зарано нехтувати Objective-C як ефективної мови розробки додатків iPhone. Хоча Свіфт виявився трохи кращим майже в усіх аспектах цієї відповідності, його перевага далеко не переважна. Наприклад, Objective-C вважається більш стійкою мовою, яка додатково дозволяє використовувати бібліотеки C і C ++, а також більш швидкий час компіляції.

Підведемо підсумки: Обидві мови мають свої плюси і мінуси. Свіфт набагато новіший, дещо простіший і все ще активно розвивається. Він більш швидкий, ніж «Об’єктив-C» - це більшість сценаріїв використання, хоча різниця у продуктивності не завжди помітна. Обидві мови підтримуються Apple, мають численні спільноти відданих шанувальників та значні бази знань.

З цієї причини прагнучі розробники мобільних телефонів неодмінно повинні використовувати Swift як мову коду iOS, насамперед тому, що це простіше. Однак багато програмістів, які освоїли Objective-C, все ще вважають, що Swift у своєму нинішньому стані не варто переходити на нього, оскільки його поточні та потенційні переваги можуть не покрити витрати на його вивчення та відмову від Objective-C. Без сумніву, команда кваліфікованих розробників Objective-C більш ніж здатна запропонувати конкурентне рішення, навіть якщо жоден з них ще не освоїв Swift.

Отже зважаючи на всі вище зазначені факти було вирішено використовувати мову програмування Swift, як сучаснішу, швидшу та безпечнішу мову.

IDE (Integrated Environment Environment) - це програмне забезпечення, яке допомагає розробникам у розробці мобільних додатків. Це графічний інтерфейс користувача, який допомагає розробникам створювати програмні програми в інтегрованому середовищі. Багато пристроїв iOS, таких як iPhone, iPad, TvOS та інші, працюють на мобільній операційній системі iOS.

Xcode та AppCode - це два модні інструменти розробки додатків для iOS. Дебати з приводу Xcode проти AppCode - гаряча тема серед кращих розробників додатків для iPhone, і його порівняння неминучі.

Ось деякі особливості обох інструментів IDE для порівняння та прийняття розумного рішення щодо того, яким шляхом рухатися.

Що стосується налаштування, AppCode випереджає свою суть. Стильний шрифт, кольори та стиль коду - це потужні функції, які AppCode пропонує розробникам. Ви також можете змінити параметри меню та порядок їх появи. Це дуже відверто і безглуздо одночасно. Навпаки, налаштування в Xcode посереднє. Тож не дивно, що розробники витрачають ці кілька додаткових хвилин на налаштування AppCode так, як вони хочуть.

Що стосується написання коду, AppCode має незначну перевагу над Xcode. Перший вимикає невикористану змінну та властивості, підказує, коли потрібно перетворити блок коду "if ... else" на потрійні операції та попереджає, коли ви створюєте код, який ніколи не буде потрапляти. Тому рекомендація та доповнення коду, запропоновані AppCode, допомагають написати бездоганний код. Тоді як Xcode також добре впорядкований і точний, ніж AppCode, але останній має більше функцій. Таким чином, AppCode має перевагу в написанні коду між Xcode проти AppCode.

Рефакторинг - це процес зміни внутрішньої структури комп'ютерної програми, не впливаючи на зовнішню поведінку з метою поліпшення внутрішньої нефункціональної властивості програмного забезпечення. У AppCode є різноманітні параметри, такі як зміна підпису, сама змінна властивість та перетворення вибору коду в метод або блог. Крім того, він також може редагувати методи та властивості у "Перейменуванні".

Перебуваючи в Xcode, для перейменування імен класів у глобальному масштабі потрібно використовувати команду - це може бути повільним і невизначеним процесом. Отже, якщо мова йде про Xcode vs. AppCode, перейменування є найбільшою перевагою AppCode перед Xcode у розробці додатків iPhone.

Важливою частиною будь-якого програмування є налагодження. Процес налагодження в Xcode неефективний, оскільки він перевіряється через точки прориву та рядки за рядком. Це стало жартівливістю того, як погано Xcode справляється з налагодженням. Перебуваючи в AppCode, він повідомляє програмісту, який блок виконується кожен раз. Xcode також не в цьому розділі. Помітною перевагою AppCode над Xcode в контексті розробки мобільних додатків, звичайно, є можливість виявлення значень властивості навіть тоді, коли вона прихована. Безумовно, результат цього раунду також йде на користь AppCode в битві між Xcode проти AppCode.

AppCode ніде не близький до Xcode за інтерфейсом користувача та його динамічними можливостями.

Оскільки останній власний IDE Apple, Apple має повний контроль над ним. Вставлення ліцензії на додаток - це лише кілька хвилин у Xcode. Набір параметрів збірки Xcode набагато кращий, ніж його конкурент. Жодна з перерахованих вище функцій не включена в AppCode, тому вона відстає від Xcode у цій області.

Коли виникає ситуація між Xcode і AppCode, розробники iOS повинні знати, що більшість додатків iPhone широко використовують розгортки, і підтримка цього в Xcode є найбільшою перевагою порівняно з AppCode. Підтримка для них посередня в AppCode. Завдання, прості в Xcode, неможливі в AppCode. Якщо ви шукаєте дошку розкадрів, то Xcode - ваш найкращий супутник.

Однак AppCode має більше переваг у певній області, і він не може пройти довгий шлях, не залежно від Xcode. Навіть кращим розробникам iPhone потрібен Xcode перед встановленням AppCode, оскільки в певний

момент їм потрібен Xcode для використання розкадров і класифікації розмірів. Зрештою, розробники не можуть повністю викинути Xcode під час використання AppCode.

Оскільки Xcode розроблений самим Apple, це безкоштовний інструмент для розробки програмного забезпечення для macOS, iOS, watchOS та tvOS. Навпаки, AppCode розробляється JetBrains на платформі IntelliJ IDEA, яка не є безкоштовною.

JetBrains пропонує 30-денну пробну версію, а після цього AppCode коштує 89 $ у перший рік та 71 $ у наступному році.

Загалом, обидва інструменти IDE мають неабияку перевагу, що може покращити досвід користувачів та допомогти їм у процесі розробки. Крім того, це залежить від зручності розробника. Якщо він очікує більшого комфорту при написанні коду, тоді для нього найкраще підійде AppCode, з іншого боку, якщо він шукає дошки та набори функцій інтерфейсу користувача, то Xcode - правильний вибір.

Зробивши висновок з описаного вище, було прийнято рішення використовувати Xcode, адже це безкоштовна нативна IDE.

2.2 Реалізація основних класів та методів

Для написання програми було використано мову Swift, а написання візуальних елементів використано новітній фреймворк SwiftUI.

Оскільки основну інформацію, яку буде бачити користувач, буде представлено у вигляді графіків, було прийнято рішення виділити саме ці класи, які займаються створенням графіків.

Ключовим елементом у побудові графіка є клас який займається малюванням одиничної лінії. Його код представлений у лістингу 1.1.

```
struct DataLine: View {
var body: some View {
    ZStack {
        GeometryReader { geometry in
```

```
                Path { path in
                    let maxX = geometry.size.width
                    let maxY = geometry.size.height

                    path.move(to: .init(x: 0, y: maxY))

                    return path.addCurve(
                        to: .init(x: maxX, y: maxY),
                        control1: .init(x: 0,    y: maxY * 0.3),
                        control2: .init(x: maxX, y: maxY * 0.3)
                    )
                }
                .foregroundColor(Color(red:  0.9,  green:  0.7,  blue:
0.7))

            }
            .foregroundColor(.primary)

            GeometryReader { geometry in
                Path { path in
                    let maxX = geometry.size.width
                    let maxY = geometry.size.height

                    path.move(to: .init(x: maxX/2, y: maxY/2))
                    return path.addLine(to: .init(x: maxX/2, y: maxY *
0.66))
                }
                .stroke(lineWidth: 2)
                .foregroundColor(.pink)
            }
        }

    }
}
```

Лістинг 1.1 Код для малювання лінії графіка

Також важливим є клас який агрегує в собі інші елементи графіків та будує простий графік. Код даного класу представлений у лістингу 1.2.

```
      struct SimplaChart: View {

       var body: some View {
           GeometryReader { geometry in
               ZStack(alignment: .bottom) {
                    Grid()

                VStack(spacing: 0) {
                    DataLine()
                        .aspectRatio(1, contentMode: .fit)
```

```
                        .frame(width:            geometry.size.width/3,
alignment: .bottom)

                    Capsule()
                        .frame(
                            width:        geometry.size.width/3        +
geometry.size.width/30,
                            height:            min(15,            max(5,
geometry.size.width/15))
                        )
                        .offset(y: -2)
                        .foregroundColor(Color(red:   1,   green:   0.7,
blue: 0.7))

                    Rectangle()
                        .frame(width: geometry.size.width/3)
                        .offset(y: -2)
                }
                   .offset(y: -1)
               }
           }
       }
}
```

Лістинг 1.2 Код елемента який будує простий графік

2.3 Використання системи

2.3.1 Системні вимоги

Оскільки застосунок призначений для мобільних телефонів під управлінням операційної системи iOS, то це основна вимога. На телефонах з ОС Android застосунок працювати не буде. Оскільки він написаний з використанням найновіший технологій, вимагається використання найновішої версії ОС, а саме iOS 13.0 та вище.

Також застосунок отримає хороший приріст від новіших телефонів, які мають потужніші процесори та відеоядра, це допоможе швидше аналізувати дані та рендерити графіки. Тому робота на старіших телефонах буде повільною. Мінімальні вимоги описані у таблиці 2.1.

Таблиця 2.1 Мінімальні системні вимоги

Критерії	Мінімальні вимоги
Операційна система	iOS 13.0
Модель телефону	Apple iPhone 6s, Apple iPhone 6s Plus
Вільне місце у пам’яті	58 мб

2.3.2 Огляд системи

На головному екрані (рисунок 2.1) Користувач бачить спрощену інформацію, з якої відомо коли останній раз дані були оновлені, скільки відвідувань було минулого року, минулого місяця, та цього місяця і відповідно до цього видно відсотковий приріст відвідувань. Також Користувач бачить аналогічну статистику про покупки.

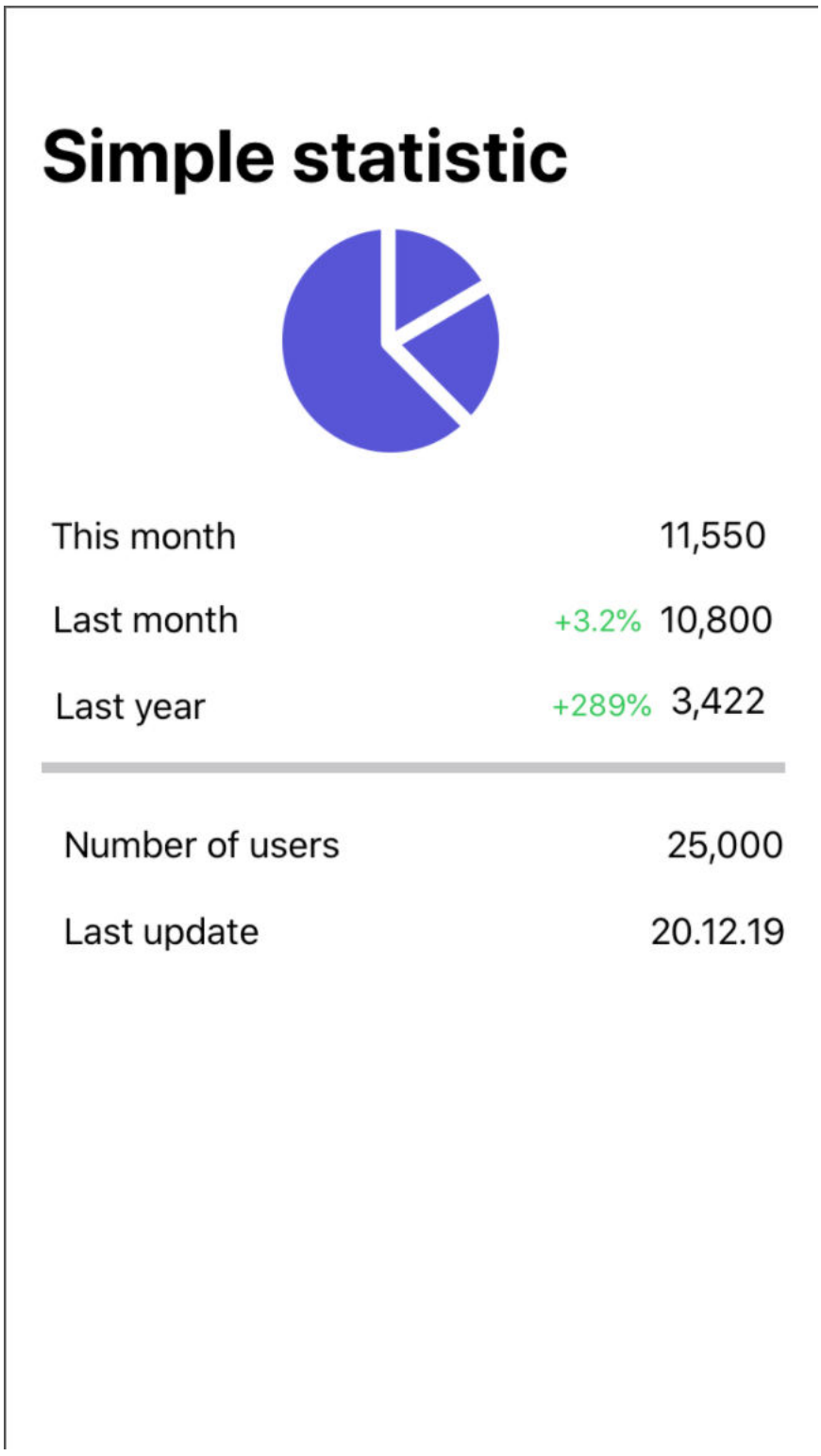

Рисунок 2.1 Спрощена інформація

На екрані графіка Користувач бачить зміну певного параметра з часом, в даному випадку відвідувань. Користувач може змінити параметр по якось будується графік та проводити певні маніпуляції з ним (збільшувати\зменшувати, пересувати).

Рисунок 2.2 Графік

3 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНА ЧАСТИНА

3.1 Загальний підхід до визначення економічної ефективності розробки.

Головною метою розділу виступає встановлення економічної доцільності розробки удосконалення програмного забезпечення, що буде використовуватись бібліотеками для адміністрування даних.

Розробка нового програмного продукту вимагає свого управління і контролю з боку керівника. Таким чином, складання та організація економічної частини є актуальною проблемою сучасного менеджменту.

Планування потребує будь-яке підприємство, будь-яке виробництво, економіка в цілому. Спланувати – означає оцінити можливості, необхідність і обсяги випуску конкурентоспроможної продукції, визначити місткість ринку і його конкретного сегмента, оцінити попит на продукцію, що випускається підприємством, результативність його роботи на ринку.

Економічне обґрунтування доцільності розробки програмного забезпечення. Зокрема розраховується комплексний показник якості проектного рішення, який показує його переваги в порівнянні з аналогами. А також на основі показника якості та ціни споживання проектного рішення та його аналога визначається коефіцієнт конкурентоздатності, який показує спроможність даного проектного рішення конкурувати з аналогами.

Приступаючи до розробки такого програмного забезпечення перш за все потрібно обґрунтувати його з точки зору економічної доцільності. Дане обґрунтування необхідне для того, щоб розробка була економічно ефективною і визначити чи варто розробляти новий програмний продукт, чи вигідніше модернізувати застарілий.

Економічний ефект розробленого продукту визначається на основі економічних показників, які дають можливість прогнозувати результат від впровадження даної програми. Враховуючи інтенсивний розвиток

комп'ютерної техніки і широкий вибір програмного забезпечення, на сьогодні економічний аналіз є невід'ємною частиною попереднього аналізу, а кошти, що затрачаються, повинні бути еквівалентними тому ефекту, який принесе конкретне нововведення.

Оцінка вартості дослідницьких розробок базується на витратному підході: використанні первісної вартості об'єктів, виходячи з фактичних витрат на розробку та доведення до комерційного використання з урахуванням амортизації.

Згідно Статті 8 Закону № 3792-12 передбачено, що твори наукового характеру та комп’ютерні програми є об’єктами авторського права. У разі реєстрації виконаної роботи як інтелектуальної власності та авторського права у державній службі інтелектуальної власності України необхідно буде сплатити збори за державну реєстрацію.

Для отримання відмінних результатів експериментів та доцільності розробки такого спеціалізованого ПЗ потрібні відповідні затрати на дослідження та розробку. Це і становитиме основу витрат, які будуть здійснені протягом підготовки та виконання реалізації даного рішення. Уявно модель витрат можна поділити на дві основні частини: витрати, пов’язанні на дослідження предметної області, побудову математичних моделей, отримання попередніх результатів експериментів, та частину реалізації програмної системи, архітектури та тестування.

Беручи до уваги залежність якості кінцевого продукту від кваліфікації програмістів, потрібно сконцентрувати увагу на якості та результативності розробки. Для цього потрібно провести ґрунтовний аналіз предметної області, залучити найновіші та інноваційні технології, провести ґрунтовне тестування та оцінку результатів розробки. Для забезпечення хорошої результативності при розробці доводиться йти на додаткові заходи заохочення та стимулювання у вигляді преміювання працівників.

До створення ПЗ можуть бути залучені позаштатні програмісти як зареєстровані, так і не зареєстровані підприємцями. В обох випадках

співпраця з ними здійснюється на підставі цивільно-правового договору, найчастіше – договорами підряду. Щодо оподаткування виплат за договором підряду, то все залежить від того, чи зареєстрований виконавець підприємцем. Важливим етапом розробки ПЗ є його тестування, яке виконується тестувальником за певну винагороду. Якщо тестувальники не перебувають з підприємством у трудових відносинах, оплата виконується на основі договору підряду на виконання робіт з тестування ПЗ. Сам же результат розробки не оподатковується, адже не є комерційним проектом і не спрямований на продаж.

3.2 Розрахунок вартості процесу розробки та оцінка економічної ефективності проекту

Для оцінки нематеріальних активів використовують міжнародні стандарти оцінки розрахунку вартості об’єктів інтелектуальної власності, розроблені TIAVIS (The International Assets Vabulation Standarts Commitie).

Виконання розробки програмного забезпечення з огляду економічної моделі можна виконувати двома способами: процедурним та об’єктно-орієнтованим. Обидва підходи потребують залучення ресурсів у вигляді програмістів-розробників, тестувальників, керівника проекту, наукового ресурсу. Різниця виникає в самій схемі розробки, тривалості періоду розробки та відповідній вартості. Процедурний підхід для розробки ПЗ в основі якого лежать процедури і функції передбачає розробку ПЗ як монолітного композиту, що в подальшому, як правило, вимагає великих витрат на супровід та модернізацію. Об’єктно-орієнтований підхід, що ґрунтується на основі об’єктів певних класів, що описують певну область, описують певну поведінку (методи) та володіють властивостями (атрибутами), орієнтовані на варіанти використання та покроковий процес розробки.

Для початку робіт необхідно скласти технічне завдання на розробку, яке є основним документом, що регламентує подальшу роботу, та містить докладний опис необхідних функцій програми, інтерфейс, технології, інше. Вартість складання технічного завдання переважно складає до 10% від планованої вартості розробки. Роботу зі складання технічного завдання веде керівник проекту разом із програмістами та консультуючись із замовником.

Усі програмісти, що працюють у штаті підприємства-розробника мають встановлено певний посадовий оклад. Місячний оклад, денна заробітна плата, трудомісткість (днів) і основна заробітна плата кожного учасника техпроцесу представлено у таблиці 3.1. Всі суми наведені в національній валюті – в гривні.

Таблиця 3.1 – Розрахункова вартість технологічного процесу розробки

Посада	Місячний оклад, грн.	Денна зар. плата, грн.	Об'єктно-орієнтований підхід		Процедурний підхід	
			Днів	Сума, грн.	Днів	Сума, грн.
Керівник проекту	13300,00	570,00	18	10260,00	19	10830,00
Програміст	12500,00	530,00	27	14310,00	31	16430 ,00
Інженер - тестувальник	10120,00	440,00	10	4400,00	10	4400,00
Всього			55	28970 грн.	60	31660 грн.

Витрати на науково-дослідницьку роботу та здійснення розробки програмних продуктів і об'єктно-орієнтованим, і процедурним способом включають:

Основна заробітна плата:

$$ЗП_{осн\ 1} = 28970 \text{ грн}; \qquad ЗП_{осн\ 2} = 31660 \text{ грн}.$$

Додаткова заробітна плата обчислюється як:

$$ЗП_{дод} = 0,2 \cdot ЗП_{осн} \quad (3.1)$$

$$ЗП_{дод\,1} = 0,2 \cdot 28970 = 5794 \text{ грн};$$

$$ЗП_{дод\,2} = 0,2 \cdot 31660 = 6332 \text{ грн}.$$

Таким чином загальний фонд заробітної плати, що обчислюється за формулою:

$$ФЗП = ЗП_{осн} + ЗП_{дод}. \quad (3.2)$$

$$ФЗП_1 = 28970 + 5794 = 34764 \text{ грн.};$$

$$ФЗП_2 = 31660 + 6332 = 37992 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи:

– єдиний соціальний внесок – 3,6 %;

– військовий збір – 1,5 %;

– ПДФО (прибутковий податок) – 15 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$Відр_{ЄСВ1} = 0,036 \cdot ФЗП = 1251,50 \text{ грн.};$$

$$Відр_{ЄСВ2} = 0,036 \cdot ФЗП = 1367,71 \text{ грн.};$$

$$Відр_{вз1} = 0,015 \cdot ФЗП = 521,46 \text{ грн.};$$

$$Відр_{вз2} = 0,015 \cdot ФЗП = 569,88 \text{ грн.}$$

$$Відр_{ПДФО1} = 0,15 \cdot ФЗП = 5214,6 \text{ грн.};$$

$$Відр_{ПДФО2} = 0,15 \cdot ФЗП = 5698,8 \text{ грн..}$$

Нарахування на фонд оплати праці, які включають відрахування до Пенсійного фонду, фонду з тимчасової втрати працездатності, фонду з безробіття і фонду страхування від нещасних випадків на виробництві; для бюджетної організації тариф на фонд оплати праці встановлено на рівні 36,3%.

Зокрема, видання програмного забезпечення – 36,77%.

Нарахування на Фонд оплати праці (ФОП): $ФОП_{ЄСВ} = 0,3677 \cdot ФЗП$

$$ФОП_{ЄСВ1} = 0,3677 \cdot ФЗП_1 = 12782,72 \text{ грн.};$$

$$ФОП_{ЄСВ2} = 0,3677 \cdot ФЗП_2 = 13969,66 \text{ грн}$$

Всього витрат:

$$В_{ЗП1} = ФЗП_1 + ФОП_{ЄСВ1} = 34764 + 12782{,}72 = 47546{,}72 \text{ грн.};$$

$$В_{ЗП2} = ФЗП_2 + ФОП_{ЄСВ2} = 37992 + 13969{,}66 = 51961{,}66 \text{ грн.};$$

Також важливою складовою витрат є матеріальні витрати. Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{в.і} = q_і \cdot p_і , \quad (3.3)$$

де: $q_і$ – кількість витраченого матеріалу і–го виду;

$p_і$ – ціна матеріалу і–го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в} = \sum M_{в.і} \quad (3.4)$$

Проведені розрахунки занесемо у таблицю:

Таблиця 3.2 – Зведені розрахунки матеріальних витрат

Найменування ресурсу	Кількість, шт.	Ціна одиниці, грн	Загальна сума, грн
Флешки	2	227,00	454,00
Папір для друку А4, арк	500	0,2	100,00
Тонер для принтера	1	60,00	60,00
Всього			614,00

Отже загальна сума матеріальних витрат становить 614 гривень.

Розрахунок витрат на електроенергію одиниці обладнання визначаються за формулою:

$$З_в = W * T * S, \quad (3.5)$$

де W – необхідна потужність, кВт; T – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії, S = 2,50 грн/кВт·год.

$З_{в1}$ =0.7 * 440 * 2.50 = 770 грн;

$З_{в2}$ =0.7 * 480 * 2.50 = 840 грн;

Розрахунок суми амортизаційних відрахувань. Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 %, вартість яких перевищує 1000 грн. і визначається:

$$A = \frac{C_Б \cdot N_A \cdot T_{ФАК}}{T_{ГОД}} \qquad (3.6)$$

де $C_Б$ – балансова вартість обладнання, грн; N_A – норма амортизаційних відрахувань в рік, %; $T_{ФАК}$ – фактичний час роботи обладнання по написанню програми, год; $T_{ГОД}$ – річний робочий фонд часу, год.

У даній формулі норма відрахувань на амортизацію рівна N_A = 0,6. Балансова вартість обладнання вказана в таблиці 3.3 і рівна $C_Б$ = 16045 гривень. Річний робочий фонд часу приймемо за $T_{ГОД}$ = 2120 годин. З них реальний фактичний робочий час становить $T_{ФАК}$ = 440 години згідно об'єктно-орієнтованого підходу та $T_{ФАК}$ = 480 годин згідно процедурного підходу.

Таблиця 3.3 – Перелік необхідного обладнання

Найменування	Кількість, шт	Ціна, грн	Сума, грн
Комп'ютер	1	12545,00	12545,00
Принтер	1	3500,00	3500,00
Середовища розробки	2	безкоштовно	безкоштовно
Операційна система	2	безкоштовно	безкоштовно
Всього більше 1000 грн.			16045,00
Всього			16045 грн.

$$A_1 = (16045 \cdot 0{,}6 \cdot 440)/2120 = 1998{,}1 \text{ грн.}$$

$$A_2 = (16045 \cdot 0{,}6 \cdot 480)/2120 = 2179{,}7 \text{ грн.}$$

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління та створення необхідних умов праці та закупівлю ресурсів та обладнання для розробки наведені в таблиці 3.3.

Залежно від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$НВ = 0{,}5 \cdot ЗП_{осн} \tag{3.7}$$

де $НВ$ – накладні витрати.

Отже, накладні витрати:

$$НВ_1 = 28970 \cdot 0{,}5 = 14485 \text{ грн.}$$

$$НВ_2 = 31660 \cdot 0{,}5 = 15830 \text{ грн.}$$

Проведемо розрахунок вартості створюваного програмного продукту. Вартість продукції включає у собі собівартість і планований прибуток. Найважливішим моментом для розробника, з економічної точки зору, є процес встановлення ціни.

Можна отримати загальні значення витрат на розробку та реалізацію проекту, враховуючи всі вище описані затрати та нарахування. Цей вид витрат складається з сум витрат на оплату праці (всього витрати на оплату праці), матеріальні затрати, затрати на електроенергію, накладні витрати, витрати на обладнання, враховуючи амортизації обладнання на час виконання проекту.

Собівартість продукції – це сума грошових витрат підприємства (фірми) на виробництво і збут одиниці продукції, виконання робіт та надання послуг.

Повна собівартість програмного продукту дорівнює сумі усіх витрат на його виробництво(таблиця 3.4):

Св1 = 59619,82 грн.; Св2 = 65093,36 грн.

Таблиця 3.4 – Розрахунок собівартості

Зміст витрат	Сума, грн	
	Об'єктно-орієнтований підхід	Процедурний підхід
Витрати на оплату праці	28970	31660
Відрахування на соціальні заходи	12782,72	13969,66
Матеріальні витрати	614	614
Витрати на електроенергію	770	840

Продовження таблиці 3.4 – Розрахунок собівартості

Амортизаційні відрахування	1998,1	2179,7
Накладні витрати	14485	15830
Собівартість	59619,82	65093,36

Приймемо прибуток на рівні 30%. Для нових інноваційних продуктів, що користуються високим попитом на ринку, ринкову вартість Вр можна встановити вищу.

Отже, вартість розробленого програмного забезпечення:

Вр1 = Св1 + 0,3 · Св1 = 59619,82+ 0,3 · 59619,82= 77505,8 грн.;

Вр2 = Св2 + 0,3 · Св2 = 65093,36+ 0,3 · 65093,36= 84621,4 грн.

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність (Е) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E = \frac{П}{Св} \tag{3.8}$$

де *П* – прибуток, *П= В - Св; Св* – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = В_р - С_в \tag{3.9}$$

Розраховуємо плановий прибуток:

$$П_{пл1} = 77505,8 – 59619,82 = 17627,99 \text{ грн.}$$

$$П_{пл2} = 84621,4 – 65093,36 = 19528 \text{ грн.}$$

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_р = \frac{П_{пл}}{С_в}. \tag{4.10}$$

$$Е_{р1} = 17627,99 / 59619,82 = 0,3.$$

$$Е_{р2} = 19528 / 65093,36 = 0,3.$$

Якщо ринкова вартість програмного продукту рівна прийнятій, то економічна ефективність визначається встановленим рівнем прибутку. Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_{ок}$):

$$Т_{ок} = \frac{1}{Е} \tag{3.10}$$

Термін окупності дорівнює:

$$Т_{ок} = 1 / 0,3 = 3,3 \text{ роки}$$

У нашому випадку $Т_{ок1} = Т_{ок2} = 1/0,30 = 3,33$ років, що є нормальним, оскільки допустимим вважається термін окупності до 5 років.

Даний розрахунок виконаний у розрахунку на 1 екземпляр програмного продукту без врахування його тиражування.

Чистий приведений дохід (ЧПД) визначається як різниця між сукупними доходами (сукупний грошовий потік) і сукупними витратами (сукупними інвестиціями) взятими за весь період життя інвестицій і дисконтова ними в кожному році на фактор часу. Дисконтування являє собою визначення вартості майбутніх грошових потоків у теперішній момент часу. Ефективним вважається той проект, який забезпечує максимум ЧПД, оскільки при цьому досягається найвища дохідність власників інвестицій.

Визначення ЧПД відбувається за формулою:

$$ЧПД = \sum_{i=1}^{t} ГП_i \alpha_{TBi} - \sum_{i=1}^{t} ІК_i \alpha_{TBi} \quad (3.11)$$

де $ГП_i$ – грошовий потік і-го розрахункового року;

$ІК_i$ – сума інвестицій і-го розрахункового року;

α_{TBi} – коефіцієнт дисконтування (коефіцієнт приведення інвестицій і грошового потоку до теперішньої вартості).

Грошовий потік – це фінансовий показник, що характеризує ефект інвестицій у вигляді грошових коштів, які повертаються інвестору.

Коефіцієнт дисконтування показує, яку величину грошових коштів ми отримаємо з урахуванням фактору часу та ризиків. Він дозволяє перетворити майбутню вартість у вартість на даний момент. Для розрахунку коефіцієнта дисконтування (коефіцієнта приведення) грошових потоків за роками періоду економічного життя інвестицій використовується формула:

$$\alpha = \frac{1}{(1+i)^n} \quad (3.12)$$

де i – ставка дисконтування або норма дисконту, i = 0,2;

n – час або кількість періодів (років), протягом якого планується отримання доходу.

$\alpha_0=1, \qquad \alpha_1=\frac{1}{(1+0.2)^n}=0,83$

Вважатимемо, що обидва програмних продукта одинаково забезпечують потреби і вимоги споживача, і тому придбання першої чи другої програми одинаково вплинуть на розмір його додаткових доходів на вкладений капітал.

Тому приймемо цю величину за постійну, а порівняння дохідності двох проектів проведемо тільки за витратами.

$ЧПД_1' = ГП+0,83*ГП-76387,95 - 0,83*17627,99 = 1,83ГП - 91019,19$ грн.;

$ЧПД_2' = ГП+0,83*ГП-83437,59- 0,83*19254,83 = 1,83ГП - 99419,10$ грн.

Чим менші витрати, тим більша дохідність проекту.

Економія витрат у випадку придбання, супроводу і одноразової модернізації програмного продукту, створеного за об’єктно-орієнтованим підходом, становить 8652,2 грн. Однак варто зауважити, що розробка спрямована на короткотривалу підтримку та не прогнозує модернізації. У разі ж необхідності розробки такого ж або суміжного ПЗ можна вважати за доцільно розпочинати розробку з початкових етапів, що потягне за собою нові витрати у повній мірі.

Здана в експлуатацію система не завжди цілком завершена, її треба змінювати протягом терміну експлуатації. Об’єктно-орієнтоване представлення програми дозволяє навіть середньому програмісту швидко і ефективно супроводжувати і модернізувати програми, що значно скорочує подальші витрати на супровід і модернізацію.

Сумарні дані економічного розрахунку розробки даного проекту наведені в таблиці 3.5.

Таблиця 3.5 – Загальні витрати

Вид витрат	Об’єктно-	Процедурний

	орієнтований підхід, грн	підхід, грн
Зарплата основна	28970	31660
Зарплата додаткова	5794	6332
Фонд заробітної плати, грн.	34764	37992
Відрахування на ФОП	12782,72	13969,66
Разом на оплату праці	82310,72	89953,66
Матеріальні витрати	614,00	614,00
Електроенергія	770	840
Амортизація	1998,1	2179,7
Накладні витрати	14485	15830
Разом на ін. витрати	17867,1	19403,7
Собівартість	59619,82	65093,36
Вартість розробленого ПЗ	77505,8	84621,4
Економічна ефективність	0,30	0,30
Термін окупності, років	3,33	3,33
Загальні витрати на розробку	95372,9	104025,1
Економія ($ЗВ_2$-$ЗВ_1$)	**8652,2**	

Загальна вартість пропонованих робіт становить 104025,1 гривень для процедурного і 95372,9 гривень для об’єктно-орієнтованого підходів розробки. В даному випадку реалізації проекту варто вибрати об’єктно орієнтований підхід для розробки даного ПЗ, адже фінансово це більш вигідно. Також у даній методиці розробки кращі часові рамки та перспективи підтримки і модернізації. У оцінці вартості продукту варто враховувати можливість фінансування та спонсорування проекту, що дозволить гнучко та ефективно підійти до розробки та організації праці.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці.

У даній роботі ведеться розробка об’єктно-орієнтованої системи обробки персоніфікованих даних. Дана система може використовуватись на комп’ютерних системах та смартфонах. Тому користувачі даної системи повинні дотримуватися правил та норм поведінки з комп’ютерною технікою та смартфонами.

Тому важливим пунктом є правильний вибір екрану для роботи та правила вибору екранної техніки. Розглянемо вимоги безпеки до екранних пристроїв:

1. Екранні пристрої не мають бути джерелом ризику для працівників.

2. Усе випромінювання, за винятком видимої частини електромагнітного спектра, має бути зведене до незначного рівня з погляду безпеки і охорони здоров'я працівників.

3. Символи на екранних пристроях мають бути чіткими, відповідного розміру. Між символами і рядками символів має бути належна відстань.

4. Зображення на екрані має бути стабільним, без миготінь або інших видів нестабільності.

5. Яскравість та/або контрастність символів має легко регулюватися працівником під час роботи з екранними пристроями, а також швидко адаптуватися до навколишніх умов.

6. Вибираючи екрани, слід надавати перевагу таким екранам, які легко та вільно повертаються і нахиляються відповідно до потреби працівника.

7. За необхідності може використовуватись окрема підставка або регульований стіл для розміщення екрана.

8. Екран не має відблискувати або відбивати світло, щоб не викликати дискомфорту у працівника під час роботи з екранними пристроями.

Також не менш важливим є правила розміщення працівників які працюють за комп'ютерами та вимоги безпеки під час роботи з екранними пристроями. Розглянемо перелік вимог:

1. Щодня перед початком роботи необхідно очищати екранні пристрої від пилу та інших забруднень.

2. Після закінчення роботи екранні пристрої слід відключати від електричної мережі.

3. У разі виникнення аварійної ситуації необхідно негайно відключити екранний пристрій від електричної мережі.

4. Не допускається:

- виконувати технічне обслуговування, ремонт і налагодження екранних пристроїв безпосередньо на робочому місці працівника під час роботи з екранними пристроями;
- відключати захисні пристрої, самочинно проводити зміни у конструкції та складі екранних пристроїв або їх технічне налагодження;
- працювати з екранними пристроями, у яких під час роботи виникають нехарактерні сигнали, нестабільне зображення на екрані та інші несправності.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп’ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні.

Під час розробки, тестування та впровадження автоматизованої системи були дотримані всі вимоги, норми та державні стандарти з охорони праці. Вся техніка відповідає вищезазначеним вимогам та усі працівники були розміщені таким чином, що мінімізує ризики здоров’ю.

4.2. Освітлення виробничих приміщень для роботи з ВДТ

Однією із характерних особливостей сучасного розвитку суспільства є зростання сфер діяльності людини, в яких використовуються інформаційні технології, до цих сфер також відноситься автоматизація роботи бібліотеки. Широке розповсюдження отримали персональні комп'ютери. Однак їх використання загострило проблеми збереження власного та суспільного здоров'я, вимагає вдосконалення існуючих та розробки нових підходів до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів.

Серед причин, що зумовлюють виникнення професійних захворювань користувачів ВДТ, значне місце посідають умови праці.

Перенапруження здорового аналізатора при роботі з ВДТ можуть бути з таких причин:

– неправильна орієнтація робочого місця відносно світлових отворів;

– неадекватні світлові характеристики світильників, неправильне їх просторове розташування відносно робочих місць;

– засліплююча дія яскравих предметів, що перебувають у полі зору користувача (пряма блискість);

– дзеркальне відбиття на екрані предметів з високою яскравістю, які за спиною користувача (відбита блискість):

– неправильний розподіл яскравості в полі зору користувача;

– засвітлення екрана прямим чи розсіяним світлом світильників або небосхилу через світлові отвори.

У запобіганні дискомфортним умовам важливе місце належить оптимізації кількісних та якісних показників освітлення.

Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2006 (На заміну СНиП II-4-79).

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природної освітленості (КПО) не нижче ніж 1,5%. Розраховується КПО за методикою, викладеною в ДБН В.2.5-28-2006.

За виробничої потреби дозволяється експлуатувати ЕОМ у приміщеннях без природного освітлення за узгодженням з органами державного нагляду за охороною праці та органами і установами санітарно-епідеміологічної служби.

Вікна приміщень з ВДТ повинні мати регулювальні пристрої для відкривання, а також жалюзі, штори, зовнішні козирки тощо.

Штучне освітлення приміщення з робочими місцями, обладнаними ВДТ ЕОМ загального та персонального користування, має бути обладнане системою загального рівномірного освітлення. У виробничих та адміністративно-громадських приміщеннях, де переважають роботи з документами, допускається вживати систему комбінованого освітлення (додатково до загального освітлення встановлюються світильники місцевого освітлення).

Світловий клімат визначає зоровий дискомфорт. Істотне значення має те, в якому світловому поясі розміщується будівля, тому що природне освітлення залежить від кількості сонячних днів у році, снігового покриву взимку та інших факторів. Для врахування цих обставин використовується поняття світлового клімату. Світловий клімат — сукупність умов природного освітлення в місцевості (освітленість і якість освітлення на горизонтальній та орієнтованих за сторонами горизонту вертикальних поверхнях, що створюються розсіяним світлом неба і прямим світлом сонця, тривалість сонячного сяйва, відбиваючі властивості земного покриву) за період понад десять років.

Запобігти шкідливому впливу освітлення можна шляхом правильного підбору системи освітлення, джерел світла (за їх спектрального складу випромінювання), світильників. Коли штучне світло змішується з природним, рекомендується використовувати лампи за спектральним складом найбільш близькі до сонячного світла. Світильники слід вибирати з розсіювачами, а блискучі деталі освітлювального обладнання, що можуть потрапити в поле зору оператора, повинні бути замінені на матові. Розташовувати робоче місце, обладнане дисплеєм, необхідно таким чином, щоб у полі зору оператора не потрапляли вікна або освітлювальні прилади; вони не повинні знаходиться і безпосередньо за спиною оператора. Світловий клімат може бути поліпшений шляхом встановлення спеціальних антивідблискових контрастних фільтрів, однак при виборі типу фільтра необхідно враховувати умови роботи з комп'ютером, оскільки оптимальні

значення коефіцієнтів пропускання і дзеркального відображення фільтрів залежать від освітленості робочого місця і типу джерела світла.

Загальне освітлення має бути виконане у вигляді суцільних або переривчатих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників. Допускається застосовувати світильники таких класів світлорозподілу:

– світильники прямого світла – П;
– переважно прямого світла – Н;
– переважно відбитого світла – В.

При розташуванні відеотерміналів ЕОМ за периметром приміщення лінії світильників штучного освітлення повинні розміщуватися локально над робочими місцями.

Для внутрішнього оздоблення приміщень з ВДТ слід використовувати дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі 0,7 - 0,8, для стін 0,5 - 0,6. Покриття підлоги повинне бути матовим з коефіцієнтом відбиття 0,3 - 0,5.

Для загального освітлення необхідно застосовувати світильники із розсіювачами та дзеркальними екранними сітками або віддзеркалювачами, укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Допускається застосовувати світильники без ВЧ ПРА тільки при використанні моделі з технічною назвою "Кососвітло". Застосування світильників без розсіювачів та екранних сіток забороняється.

Як джерело світла при штучному освітленні повинні застосовуватися, як правило, люмінесцентні лампи типу ЛБ. При обладнанні відбивного освітлення у виробничих та адміністративно-громадських приміщеннях можуть застосовуватися металогалогенні лампи потужністю до 250 Вт. Допускається у світильниках місцевого освітлення застосовувати лампи розжарювання.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50º до 90º відносно вертикалі в подовжній і поперечній

площинах повинна складати не більше 200 кд/м^2, а захисний кут світильників повинен бути не більшим за 40º.

Коефіцієнт запасу ($K_з$) відповідно до ДБН В.2.5-28-2006 для освітлювальної установки загального освітлення слід приймати рівним 1,4.

Коефіцієнт пульсації повинен не перевищувати 5 % і забезпечуватися застосуванням газорозрядних ламп у світильниках загального і місцевого освітлення.

За відсутності світильників з ВЧ ПРА лампи багатолампових світильників або розташовані поруч світильники загального освітлення необхідно підключати до різних фаз трифазної мережі.

Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300…500 лк. У разі неможливості забезпечити даний рівень освітленості системою загального освітлення допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану та збільшення освітленості екрану більше ніж до 300 лк.

Світильники місцевого освітлення повинні мати напівпрозорий відбивач світла з захисним кутом не меншим за 40º.

Необхідно передбачити обмеження прямої блискості від джерела природного та штучного освітлення, при цьому яскравість поверхонь, що світяться (вікна, джерела штучного світла) і перебувають у полі зору, повинна бути не більшою за 200 кд/м^2.

Необхідно обмежувати відбиту блискість шляхом правильного вибору типів світильників та розміщенням робочих місць відносно джерел природного та штучного освітлення. При цьому яскравість відблисків на екрані відеотерміналу не повинна перевищувати 40 кд/м^2, яскравість стелі при застосуванні системи відбивного освітлення не повинна перевищувати 200 кд/м^2.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору осіб, що працюють з відеотерміналом, при цьому відношення значень

яскравості робочих поверхонь не повинно перевищувати 3:1, а робочих поверхонь і навколишніх предметів (стіни, обладнання) – 5:1.

Необхідно використовувати систему вимикачів, що дозволяє регулювати інтенсивність штучного освітлення залежно від інтенсивності природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення.

Для забезпечення нормованих значень освітлення в приміщеннях з відеотерміналами ЕОМ загального та персонального користування необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік, та своєчасно проводити заміну ламп, що перегоріли.

Виробниче освітлення. За висновками експертів ВООЗ, під час роботи на ВДТ найбільше навантаження припадає на зоровий аналізатор, тому в забезпеченні роботи користувача важливе місце займає раціональне освітлення робочих місць.

Аварійне освітлення ділиться на два види: освітлення для продовження роботи й для евакуації людей. Освітлення для продовження роботи обладнується у виробничих приміщеннях підприємств, у яких неприпустимі перерви в роботах при відключенні робочого освітлення. Найменша освітленість робочих місць в аварійному режимі повинна становити не менше 5% нормованої робочої освітленості. Аварійне освітлення для евакуації людей встановлюється в місцях, небезпечних для проходу людей, коридорах, на сходових клітках, їдальні, конференц-залах і виробничих приміщеннях. Аварійне освітлення повинне забезпечувати освітленість не менше 0,5 лк на рівні підлоги основних проходів і сходів.

ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено програмну систему для обробки персоніфікованих даних, які несуть велику цінність для малого та середнього бізнесі, і приватних підприємців.

Призначення системи для малого та середнього бізнесу має на меті економію коштів підприємців. Адже система, яка призначена для великої кількості користувачів потребує просунутих алгоритмів обробки даних, машинного навчання, великих обчислювальних потужностей та зберігання надзвичайно великої кількості інформації. Для цього всього потрібна велика кількість коштів, яких може не бути у підприємців малого чи середнього бізнесу. Тому ця система використовує спрощені алгоритми та працює з малою кількістю даних, що дозволяє користувачу максимально мінімізувати витрати.

Доцільність розробки базується на дешевизні та простоті розробки, в сукупності з великою користю для отримання прибутку. Також дана область є надзвичайно цікавою і корисною у нашому капіталістичному суспільстві.

Підсумовуючи весь матеріал можна дійти висновку, що дана система буде надзвичайно корисною починаючим підприємцям, які хочуть побачити свій бізнес в цифровому варіанті, не витрачаючи великі кошти, при цьому зрозумівши над якими аспектами підприємства потрібно попрацювати щоб збільшити свій дохід.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is data processing? [Електронний ресурс]. – Режим доступу: https://www.talend.com/resources/what-is-data-processing/ – Назва з екрану.
2. Swift vs. Objective-C: A Look at iOS Programming Languages [Електронний ресурс]. – Режим доступу: https://www.upwork.com/hiring/mobile/swift-vs-objective-c-a-look-at-ios-programming-languages – Назва з екрану.
3. Что такое Agile и Scrum? [Електронний ресурс]. – Режим доступу: https://www.pmoffice.by/blog/agile/agile-approach.html – Назва з екрану.
4. Swift [Електронний ресурс]. – Режим доступу: https://swift.org – Назва з екрану.
5. Історія мобільних пристрій та їх використання \. [Електронний ресурс] Режим доступу https://westele.com.ua/ua/blog/135_istoria- mobilnogo-telefona.html
6. Методичні вказівки до виконання магістерської роботи освітнього рівня “магістр” студентами усіх форм навчання для напряму підготовки 121 – “Інженерія програмного забезпечення” / Укладачі : Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 26 с.
7. Методичні рекомендації по виконанню розділу техніко- економічного обґрунтування дипломних робіт студентами технічних спеціальностей напряму підготовки 8.05010302 «Інженерія програмного забезпечення» освітньо-кваліфікаційного рівня «Магістр» / Укладачі: Петрик М.Р., Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с.

8. Android vs. iOS: Which is more secure? [Електронний ресурс] Режим доступу: https://us.norton.com/internetsecurity-mobile-android-vs-ios-which-is-more-secure.html

ДОДАТКИ