Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи цифрових інформаційно-рекламних панелей”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Гончаренко О.П.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук
_____ Лисенко І.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *123 "*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Гончаренку Олександру Павловичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи цифрових інформаційно-рекламних панелей*

2. Керівник роботи *Лисенко Ірина Анатоліївна, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 46-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи цифрових інформаційно-рекламних панелей*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Лисенко І.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Гончаренко О.П.
(прізвище та ініціали)

АНОТАЦІЯ

Гончаренко О.П. Програмне забезпечення системи цифрових інформаційно-рекламних панелей. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи цифрових інформаційно-рекламних панелей.

Метою розробки є програмне забезпечення системи цифрових інформаційно-рекламних панелей.

Результат роботи – програмна реалізація системи цифрових інформаційно-рекламних панелей.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, цифрові інформаційно-рекламні панелі

ABSTRACT

Goncharenko O.P. Software for the system of digital information and advertising panels. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the system of digital information and advertising panels.

The purpose of the development is the software for the system of digital information and advertising panels.

The result of the work is the software implementation of the system of digital information and advertising panels.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with Windows 10/11.

The program was developed in the Python environment.

Keywords: computer engineering, digital information and advertising panels

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ВКЗ	–	відеоконференцзв’язок
ПЗ	–	програмне забезпечення
CMP	–	Codian Management Platform
DNS	–	Domain Name System
GMS	–	Global Management System
GRE	–	Generic Routing Encapsulation
ISDN	–	Integrated Services Digital Network
MCU	–	Multipoint Control Unit
MPLS	–	Multiprotocol Label Switching
MXM	–	Media eXchange Manager
SSL	–	Secure Socket Layer
TLS	–	Transport Layer Security
TMS	–	Tandberg Management Suite
VPN	–	Virtual Private Network
USENET		User Network

ВСТУП

Актуальність теми. Системи інформаційно-рекламних вивісок, фахівцям більше відомі як Digital Signage, – один з найбільш ефективних інструментів цифрової трансформації бізнесу та й всього нашого життя. Варто відволіктися на секунду від всюдисущого Інтернету на смартфоні або планшеті, і навколо розкривається інформаційний мир Digital Signage – у торговому центрі, виставочному комплексі, на стадіоні, транспортному вузлі й просто на вулицях міста.

Відповідно до дослідження, щорічно проведеному аналітичним агентством OSP Data, частка компаній, що використовують системи Digital Signage, постійно збільшується. Якщо в 2024 році таких компаній було всього 6%, то в поточному, 2025-м – уже 11%. І це незважаючи на кризу, що охопила український ринок. Як змінилися потреби й запити замовників щодо систем Digital Signage у поточних економічних умовах? Яка роль Digital Signage у цифровій трансформації бізнесу компаній?

Девальвація гривні привела до істотного подорожчання технічних рішень Digital Signage. Відповідно, питання ціни стало ключовим, і замовники почали шукати способи зниження фінансових витрат. Замовники стали ще ретельніше прораховувати всі ризики й прибутковість проектів, вони дуже обережно приймають рішення. Багато хто серйозно урізують бюджет, скорочуючи масштаби проектів, відмовляючись від «надмірностей» і намагаючись відкласти всі витрати, якщо тільки вони не пов'язані з життєво важливими процесами. Тим часом системи Digital Signage встигли міцно ввійти в повсякденне життя й бізнес, і сьогодні, у непростій економічній ситуації, затребувані в багатьох галузях, включаючи торгівлю, транспорт, корпоративний сектор, утворення й т.д.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи цифрових інформаційно-рекламних панелей.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем цифрових інформаційно-рекламних панелей.

– Дослідження системи цифрових інформаційно-рекламних панелей.

– Програмна реалізація системи цифрових інформаційно-рекламних панелей.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі цифрових інформаційно-рекламних панелей.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи цифрових інформаційно-рекламних панелей, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розгортання масштабної системи Digital Signage, наприклад, у торговельній мережі припускає досить більші інвестиції, тому компанії, особливо в ретейлі, перейшли в режим очікування. У свою чергу, підвищився попит з боку організацій, де Digital Signage не опція, а необхідність. Це кінотеатри й музеї, де мультимедійні технології використовуються усе ширше.

У якості одного зі способів мінімізації витрат називається відмова від мультивендорного підходу й відхід від використання набору «дисплей – платформа – ПЗ». Все більшою популярністю користуються закінчені коробкові рішення, пропоновані розроблювачами самих дисплеїв. Сучасні платформи, які вбудовуються, стали дуже продуктивними: вони вже здатні не тільки вирішувати базові завдання клієнта, але й реалізовувати більше складні функції. Серед них – керування розкладом виводу контенту або навіть керування контентом при побудові відеостін.

На більше активний попит у кризу на інтегровані рішення класу System on Chip (SOC) указують експерти. На їхню думку, вони хоча й не надають всіх можливостей повноцінних, виділених плеєрів, але можуть виконувати основні завдання системи Digital Signage.

Інший шлях для економії бюджету – перехід на рішення від азіатських виробників дисплеїв. Сьогодні ми спостерігаємо підвищення попиту на встаткування китайських і південно-корейських вендорів. При цьому вони активно інвестують як у розробку, так і в розвиток партнерської мережі на території України. І звичайно, ціна на їхні рішення поки залишається одними з найнижчих.

1.2 Область застосування

Для розподілу контенту в системах Digital Signage всі частіше використовується технологія передачі відео по IP. З безлічі технологій передачі відеоконтенту в системах Digital Signage найцікавіші ті, що надають можливість простого переналаштування системи. Така потреба викликана частими змінами схеми трансляції контенту до різних пристроїв відображення у зв'язку з постійними перепланування ми приміщень у сфері ретейла, зміною орендарів, зміною площ, переносом пристроїв відображення на нові місця для більше широкого охвату цільової аудиторії й т.п.

IP-рішення надають подібну гнучкість. До їхніх переваг ставиться також можливість використовувати наявну мережу, створивши для систем Digital Signage окрему незалежну віртуальну локальну мережу (VLAN). Це дозволяє заощадити істотні засоби. Рішення на базі IP легко масштабуються – можна нарощувати як число крапок трансляції, так і кількість пристроїв відображення. А далекість пристрою відображення (монітора) від джерела визначається топологією мережі, причому у випадку оптичних ліній зв'язку відстань між ними практично нічим не обмежується.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи цифрових інформаційно-рекламних панелей, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Scala Designer

Scala Designer це найшвидше й рентабельне рішення для створення динамічного контенту для цифрових табло.

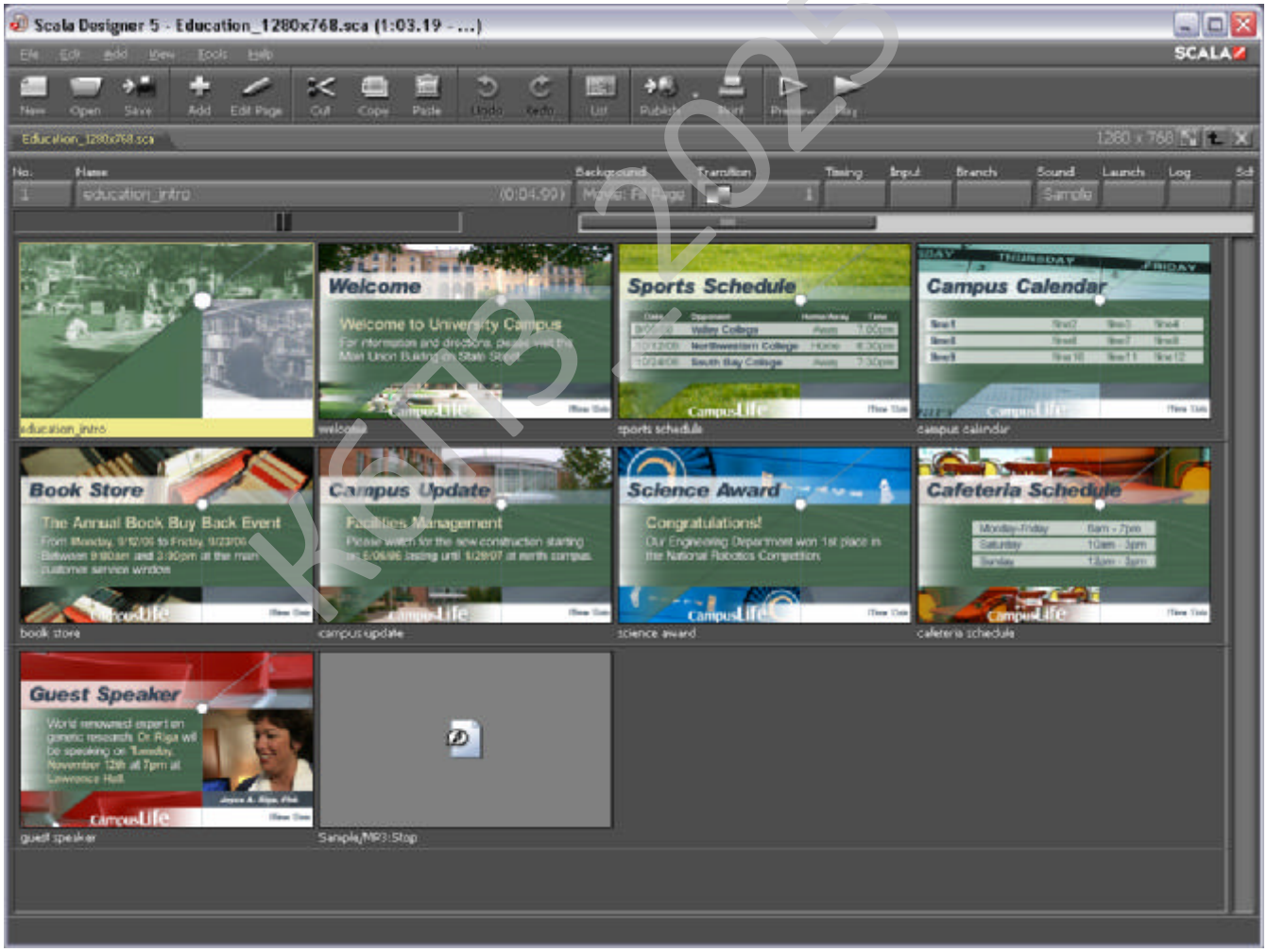

Рисунок 2.1 – Інтерфейс користувача Scala Designer

Функції:

– Використовуйте ваші улюблені інструменти – Використовуйте улюблені інструменти створення відео й зображень для створення повного продукту або як елементи для поліпшення за допомогою Designer, щоб створити динамічний контент, якому можна швидко й просто міняти.

– Плагіни Photoshop – Designer поставляється із плагіном для Adobe Photoshop, що дозволяє створювати верстку в Photoshop і експортувати в Designer, де можна додати переходи й інтерактивність.

– Агрегація інших форматів – Designer поєднує велику кількість форматів файлів зображень, відео й аудіо без перетворення або повторної візуалізації на сервері.

– Зображення: BMP, JPG, GIF, PNG, TIFF.

– Відео: AVI, SWF, MPG, WMV, H.264.

– Звук: WAV, MP3.

– Зробіть його динамічним – Плавно поєднуйте текст, графіку, звук і відео в якісну мультимедійну програму. Попередній перегляд у точності як це буде виглядає, навіть на повному екрані.

– Забудьте про повторну візуалізацію – Designer не проводить повторної візуалізації ваших відео й іншого контенту, надаючи більше творчого контролю над якістю фінального відтворення. Відтворення візуалізується в реальному часі й на ходу, не міняючи оригінального формату.

– Дивні переходи – Designer дозволяє вам використовувати сотні плавних повноекраних ефектів і переходів, убудованих у механізм відтворення Scala.

– Відтворення в реальному часі – Вмонтуйте Windows Streaming Video або Adobe Flash у реальному часі.

– Верстка – Створіть більше привабливий і помітний екрани з інструментами розміщення елементів, сітками й лінійками.

– Мовна підтримка – Створюйте свій контент на десятках мов з підтримкою більшості міжнародних систем листа (включаючи європейські й азіатські мови, арабський, іврит і т.д.).

– Гнучкість дизайну – Розробляйте для ландшафтних і портретних дисплеїв або відеостін. Контент можна автоматично обертати й масштабувати для ваших дисплеїв.

– Рядки, що біжать – Додайте рядки, що біжать, даних з керованим напрямком і швидкістю.

– Повний запас – Scala Designer поставляється з безліччю кліп-арта, стокових зображень, шаблонів, фонів і стокових відео, щоб допомогти пожвавити ваш контент.

– Дизайн шаблонів – Розробляйте відмінні шаблони й зберігайте ясність і незмінність стилю, у той же час дозволяючи користувачам в інтернеті з легкістю додавати контент і текст.

– Інтерактивність – Розробляйте інтерактивний контент із кнопками, змінними й розгалуженням для терміналів і інших цілей.

– Легкість у налаштуванні – Багато додатків можна писати прямо в зручному графічному інтерфейсі Scala.

– Можливості скриптів – Програмісти можуть розібратися в скриптовій мові ScalaScript або використовувати підтримувані Windows скриптові мови, такі як BScript, JavaScript або Python.

– Зручність розробки – Designer поставляється з безліччю інших корисних можливостей, таких як скасування, перевірка правопису, печатка, збір файлів і експорт.

– Модулі розширення – Необов'язкові доповнення для керуючих перемикачів, пристроїв відтворення відео й інших пристроїв.

– Опублікуйте свою роботу – Закінчивши, опублікуйте свій контент в Scala Content Manager, де він вноситься в розклад і розподіляється в програми Scala Player.

Менеджер контенту

З Scala's Content Manager керування й контроль ваше мережі поліпшується й спрощується – скорочуючи кількість необхідних для цього ресурсів. Підтримка шаблонів, планове поширення контенту, розклади й списки відтворення – управляйте своєю мережею з будь-якої крапки миру, з будь-якого комп'ютера, підключеного до інтернету.

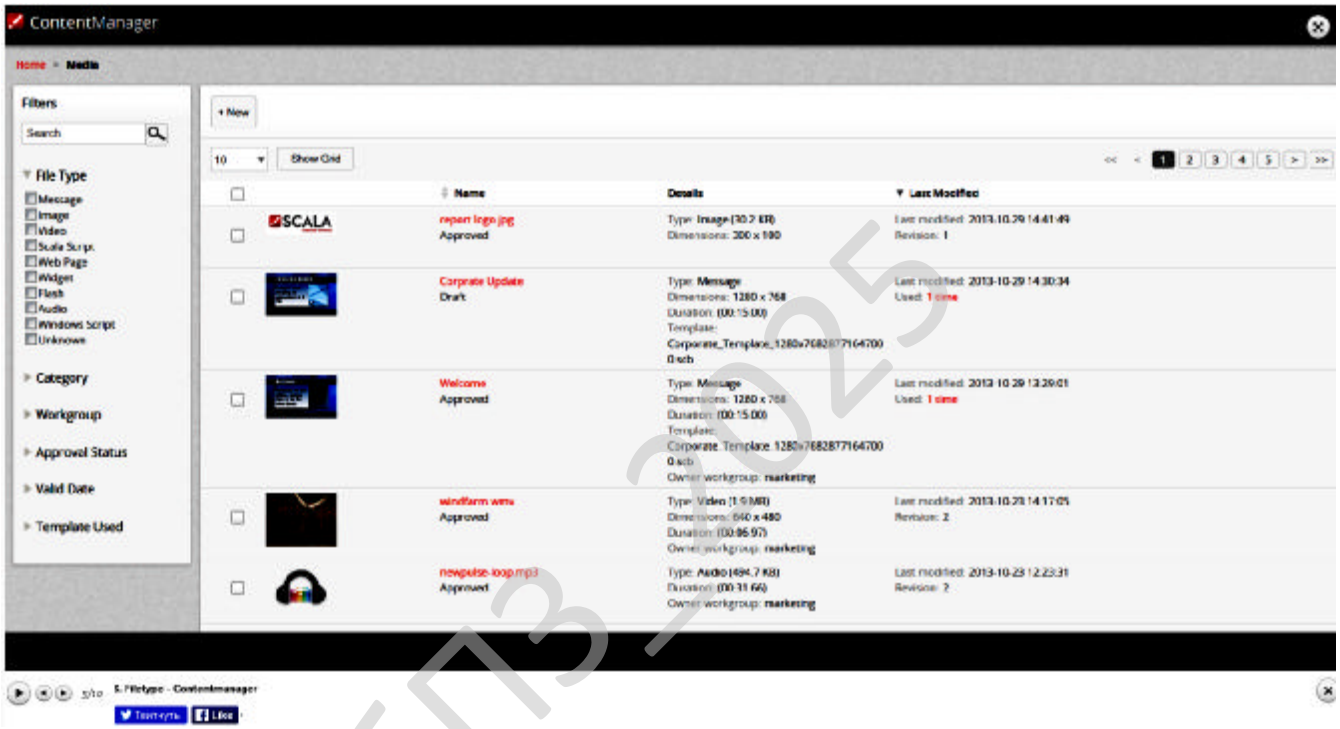

Рисунок 2.2 – Інтерфейс користувача Scala's Content Manager

Функції:

– Простій у використанні інтерфейс – Швидко й просто орієнтуйтеся в інтуїтивному й дружньому веб-інтерфейсі.

– Шаблони – Користувальницькі шаблони полегшують швидке створення прекрасних відображень, простим додаванням власних зображень і тексту. Ваші дизайнери можуть створити змінні поля даних, які авторизовані користувачі зможуть надалі заповнити й переглянути через інтерфейс Content Manager.

– Керування середовищем – Організуйте свій контент по типах, категоріям і метаданим, щоб усе було легше знайти.

– Списки відтворення – Створіть списки відтворення за допомогою скриптів Scala, повідомлень, відео, flash і статичних зображень.

– Розкладу – Визначите й редагуйте тимчасові інтервали списків відтворення в простому графічному відображенні. З різними схемами повтору й датами початку/завершення, складання розклад контенту повинне бути нескладним.

– Термінові повідомлення – Можна настроїти повідомлення, які перекриють відображення в певний час або коли відбудеться зовнішня подія, таке як надзвичайна подія.

– Розумні списки відтворення – Використовуйте метадані програвача для динамічного налаштування контенту на кожному етапі відтворення.

– Схвалення – Схвалюйте контент перед відтворенням. Обрані користувачі одержать електронне повідомлення із запитом на схвалення.

– Розумний розподіл контенту – Контент доставляється автоматично й кожний програвач розумно вибирає нові або змінені файли.

– Кілька незалежних кадрів – З легкістю створюйте кадри на дисплеї, з незалежним контентом у кожному.

– Підтримка багатоканальності – Запускайте два окремих канали з одного програвача, знижуючи витрати на апаратні засоби, ліцензування ОС і адміністрування.

– Аудіо керування – Створюйте й аудіо канали й відтворюєте фонову музику, яку можна автоматично відключити при відтворенні відео із власної аудіо доріжкою.

– Надійна й масштабована – Проста в установці й масштабуванні. Випробувана в польових умовах надійність із більшою кількістю мереж, чим у будь-якого конкурента.

– Зв'язок на ваших умовах – Підтримує наземні й супутникові (багатоадресні) IP комунікації.

– Поліпшена сумісність – Сумісне із проксі й ACNS і може доставляти контент через HTTP/HTTPS через IPv4 або IPv6, відповідаючи вимогам IT-безпеки.

– Бази даних/Обчислювальні бази даних – Виберіть сервер і базу даних, оптимальні для вашого оточення В числі підтримуваних баз даних MySQL, Microsoft SQL Server і PostgreSQL.

– Опції програвача – Виберіть пристрою відтворення, що відповідають вашим потребам, від повноцінного PC до мультимедійного пристрою або цифрової фоторамки.

– Керування програвачем – Групуйте програвачі за критеріями (географія, демографія й т.д.), установіть опції відтворення й виберіть контент.

– Моніторинг мережі – Перевірте статус вашої мережі за допомогою моніторингу стану програвача.

– Розклад обслуговування – Виконуйте завдання по вилученому обслуговуванню, такі як перезавантаження, відправлення/одержання файлів і установка відновлень Scala/ Завдання можуть запускатися в певний час із мінімальним порушенням роботи вашої мережі цифрових табло.

– Веб-послуги – Одержуйте доступ до програмного "движкам" для всіх, хто хоче розробити власні клієнтські програми для керування доставкою контенту або доступом. Content Manager відмінно працює не тільки для власних клієнтських програм, але корисний і для інтеграції багатьох інших процесів і додатків.

– Робочі групи – Управляйте доступом до контенту в різних відділах, щоб кожна група бачила тільки повідомлення, призначені їй.

– Засновані на допусках ролі користувачів – Користувачам можуть бути призначені різні права доступу для поліпшення надійності й робочого процесу.

Програвач

– Стабільне й надійне відтворення цифрових табло 24 години на добу, 7 днів у тиждень.

– Чудові технології відтворення із гладким рухом суб-пікселів без "брижів" і "спотикания".

– Програвач буде завантажувати тільки зміни, а не весь контент.

– Підтримується зв'язок з Content Manager для відстеження станів Player і ведення журналу подій для виставляння рахунків за рекламу й звітів.

– Підтримка Windows Script Host дозволяє створити надійний зв'язок між Player і іншими системами/даними через JavaScript, Python або VBScript.

– Підтримує Windows Streaming Video або відео в прямому ефірі через Модулі розширення й відповідні апаратні продукти.

– Відтворює або вільнойдущий, або інтерактивний контент за допомогою сенсорного екрана.

– Убудована підтримка широкоекранних дисплеїв і автоповорот на портретних екранах.

– Можливість відтворення тільки аудіо, якщо ви хочете управляти й візуальними й аудіо повідомленнями.

– У випадку відключення вашої мережі, Player продовжить використовувати технологію зберігання й передачі.

– Можливості ліцензування Player доступна для різноманітних апаратних засобів, від найпростіших до самих нових.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – високорівнева мова програмування, яку називають другою за популярністю в світі. Її використовують для розробки вебзастосунків, програмного забезпечення, машинного навчання. Python застосовують для

вирішення робочих завдань у компаніях Google, Instagram, Facebook, IBM, NASA, Dropbox, Netflix та інших. Розробники цінують цю мову програмування за простоту у вивченні, ефективність та мультиплатформність.

Python – скриптова мова програмування з досить простим синтаксисом. Для розуміння достатньо порівняти принципи написання найпростішої програми, яка виводить на екран текстове повідомлення. Саме тому мова програмування Python більш доступна для новачків, а професіонали встигли адаптувати її для вирішення великої кількості завдань. Це мультиплатформне рішення, тому знання Python дає можливість працювати у різних сферах: від розробки мобільних застосунків до ігрової індустрії та штучного інтелекту.

У мови програмування динамічна типізація: є можливість передавати до функцій будь-який тип даних без попереднього вказання. Інтерпретованість дозволяє знаходити помилки у коді ще до повної збірки у робочий застосунок. При цьому Python дуже чітко дає зрозуміти, де та через що виникла помилка.

Це мова об'єктно-орієнтованого програмування (ООП). Програмне забезпечення на Python оформлене у вигляді моделей, які можуть бути зібраними у пакети. Тип та структуру кожного об’єкта можна запитати під час виконання програми. Для кожного з об’єктів можна отримати всю інформацію щодо його внутрішньої структури. Окрім того:

– у мови логічний синтаксис, завдяки чому вихідний код легко читати та розуміти;

– гнучкість та масштабованість Python дозволяє адаптувати високорівневу логіку та розширяти складні застосунки, як тільки виникне така необхідність;

– розробка на Python у більшості випадків проходить швидше, ніж на інших мовах програмування;

– Python – інтерпретована мова програмування. Це значить, що код можна написати у будь-якому текстовому файлі на будь-якій платформі, і потім успішно запустити;

– у Python – колосальна спільнота однодумців. Тож будь-які складнощі конкретних розробників вирішуються колективно.

Проте є декілька особливостей, які можна віднести до недоліків. Це повільність (ця мова програмування хоч і універсальна, проте повільніша за інші), велика кількість ресурсів, необхідних для роботи та «прив'язаність» до системних бібліотек.

Мова програмування Python використовується у наступних сферах:

1. Розробка програмних застосунків будь-якого напрямку.

2. Розробка серверної частини мобільних застосунків (найпопулярніший напрямок).

3. Ігри. Багато сучасних ігор для комп'ютерів (наприклад, World of Tanks) частково чи повністю написані на Python.

4. Вбудовані системи для різних пристроїв. Дуже часто Python використовують для написання внутрішніх платформ управління банкоматами.

5. Скрипти та плагіни до уже реалізованих програм для автоматизації процесів чи створення інших рішень.

6. Тестування (автоматизація цього процесу).

7. Машинне навчання. – основна мова для написання алгоритмів і аналітичних застосунків у сфері Machine Learning.

Бібліотеки Python

Різні бібліотеки Python використовують для виконання конкретних завдань. Наприклад, Matplotlib підходить для відображення даних у двовимірній та тривимірній графіці. Pandas підходить для зручної роботи з даними. NumPy дозволяє створювати масиви та керувати ними. Requests використовується для веброзробки. OpenCV-Python відкриває можливості для обробки зображень з метою оптимізації систем «машинного зору».

Найвідоміші фреймворки для мови програмування Python

Фреймворки Python допомагають створити зручне та функціональне середовище для розробки. У них міститься набір інструментів, модулів та

бібліотек, корисних для виконання конкретних завдань. Це значно полегшує роботу: наприклад, дає змогу не витрачати час на розписування дій, які повторюються, а використати релевантний інструмент. Тож є можливість позбутися рутинних процесів та сконцентруватися на логіці проєкту.

Серед найпопулярніших фреймворків для Python:

– Django – найстаріший та найвідоміший. Створений для реалізації великих інтерактивних проєктів;

– Pyramid – зручний у налаштуваннях, і дає можливість реалізувати складні нестандартні ідеї;

– Web2py – підходить в першу чергу для вебзастосунків і може використовуватись на будь-яких архітектурах.

Популярні Python IDE

IDE або інтегровані середовища розробки – це програмне забезпечення, яке надає розробникам необхідні інструменти для написання, редагування, тестування та налаштування коду. Для розробки на Python найчастіше використовують IDE PyCharm, IDLE, Spyder та Atom.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи цифрових інформаційно-рекламних панелей.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі.

Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Засоби відображення

Незважаючи на прагнення замовників заощадити, світові тенденції розвитку засобів відображення, безумовно, проявляються й в Україні. Так, більшість експертів відзначають ріст популярності технології 4K. Дисплеїв з високим дозволом використовується усе більше. При цьому вони стають яскравіше й підтримують розширений колірний простір, що дозволяє зробити контент більше «глянсовим» і «соковитим»».

Ще одна тенденція – збільшення діагоналі екранів. Якщо раніше самими «ходовими» були дисплеї 42″, 46″, 55″, то зараз середній сегмент змістився до 65″, 85″, 98″ і навіть 105″. Але при цьому відзначається, що розширився затребуваний діапазон невеликих екранів, які дуже зручно використовувати в ретейлі на полках і крапках POS – тепер він становить від 7″ до 21″.

Разом з тим 3D-рішення усе ще рідко використовуються в проектах. Зв'язано це навіть не стільки з високою вартістю самих систем, скільки з дорожнечею виробництва відповідного контенту: Ми вже більше двох років пропонуємо екрани, на яких можна демонструвати 3D без необхідності використання спеціальних окулярів, але за все це час не було ні одного великого проекту саме через те, що клієнти не в змозі підготувати 3D-контент. Проте поява автостереоскопічних дисплеїв (не потребуючі використання спеціальних окулярів) із принципово більше високою якістю відтворення 3D здатне згодом змінити відношення ринку до тривимірного зображення.

Самим яскравим ноу-хау на ринку за останнім часом стала поява екранів нестандартної форми. Сучасні рішення на базі OLED-технологій дозволяють будувати відеостіни з екранів, зібраних у вигляді хвиль, півсфер і арок. Крім того,

серед інноваційних рішень він виділяє дисплеї, на яких зображення транслюється із двох сторін. Для вузького сегмента ринку з'являються рішення на основі прозорих дисплеїв. Але всі ці новинки поки тільки починають виходити на ринок професійних систем Digital Signage, їхня вартість досить висока.

Екрани нестандартних форм і розмірів, двосторонні й прозорі екрани можуть стати важливої складового інтер'єра або навіть його основою. У цілому, сьогодні в замовників є досить багатий вибір: це, зокрема, рідкокристалічні екрани, LED-екрани (у тому числі екрани високої чіткості, HDLED, напрямок що бурхливо розвивається), OLED-дисплеї (включаючи двосторонні, вигнуті й прозорі варіанти), а також проекційні технології. Останні теж не варто скидати з рахунків завдяки розвитку лазерних і гібридних технологій, що забезпечують тривалу безперервну роботу, високий світловий потік і великі убудовані можливості процесінгу.

Експерти відзначають ріст інтересу замовників до засобів відображення, що забезпечує можливість інтерактивної взаємодії. Люди вже звикли взаємодіяти з дисплеями, а замовники, у свою чергу, усвідомили перспективи використання накопиченого клієнтами користувальницького досвіду. По даним IMS, уже з'явилися запити на величезні відеостіни із сенсорними екранами, хоча технічно подібні проекти реалізувати непросто. Але поки основними видами інтерактивних пристроїв залишаються кіоски самообслуговування й навігації.

Сучасні дисплеї із сенсорними екранами називаються « iPad-подібними» у тому розумінні, що якість відпрацьовування торкань сталася порівнянно із забезпечуваним планшетами. У сполученні зі зниженням вартості інтерактивних систем це повинне сприяти більше широкому їхньому поширенню.

Засоби відображення з можливістю інтерактивної взаємодії знаходять застосування в основному в культурно-досуговій (музеї, виставки) і освітній сферах. У корпоративному секторі найбільший інтерес до таких рішень проявляє ретейл для залучення клієнтів – наприклад, для створення віртуальних примірювальних або систем інформування.

До переліку замовників інтерактивних систем додаються готелі, яким, зокрема, цікаві інтерактивні столи з набором ігор для дітей. Так, стандартами мережі готелів Novotel уже давно передбачається розміщення в дитячих куточках столів з набором ігор для дітей від 5 до 12 років. І у всіх нових готелях даної мережі передбачається установка таких рішень виробництва французької компанії Humelab.

Залучення користувачів у процес одержання інформації або надання їм додаткових сервісів, що припускають зворотний зв'язок, – важливий напрямок розвитку систем Digital Signage. Для такого «залучення» запропоновано багато підходів. Оборотний увага на технологію Light ID від компанії Panasonic. Ідея полягає в тому, що за допомогою LED-джерела світла самого дисплея (РК-панелі або проектора) на мобільний пристрій передається ідентифікаційний номер (якщо камеру смартфону навести на демонструєме зображення), по якому відповідний додаток мобільного пристрою знаходить потрібну інформацію. Ця технологія заснована на тих же принципах, що й QR-коди, тільки код тепер не потрібно виділяти візуально – він взагалі невидимий для очей. Система, побудована на основі Light ID, може виявитися дуже ефектною: коштує користувачеві навести камеру смартфону на його зображення, що зацікавило, або відеоролик на екрані – і він одержить додаткову інформацію. Очевидно, що й власник такої системи у виграші: він одержує дані про інтереси користувачів.

Одне з головних переваг сучасних систем Digital Signage – максимально точне таргетування споживача з підбором найбільш релевантного для нього контенту в реальному часі. У свою чергу, один з головних інструментів таргетування – визначення місця розташування людини, або геолокація. Класичний приклад використання геолокації – передача на смартфон користувача пропозицій конкретного магазина (акції, знижки та ін.) у той момент, коли він перебуває поблизу цього магазина або безпосередньо в ньому.

Існує кілька методів геолокації, кожний має свої переваги й обмеження. Усередині будинків для визначення місця розташування користувачів в

основному використовуються дві технології: Wi-Fi і Bluetooth Low Energy (LE). Але для їхньої ефективної роботи необхідно побудувати усередині будинку розгалужену інфраструктуру (мережа Wi-Fi або мережа з датчиками-маяками), що може виявитися досить витратною витівкою.

Точність різних методів геолокації

Дані про точність і обмеження різних технологій геолокації:

– GPS/ГЛОНАСС. Точність близько 8 м, але в приміщеннях сигнал часто недоступний або зашумлен, що знижує точність позиціонування пристрою в будинках. Таким чином, часто можна лише визначити, що користувач перебуває в конкретному будинку або поблизу від певних місць. Варто відзначити, що не в кожної людини є в наявності телефон з GPS (або ця функція включена), що також обмежує можливість застосування такого типу геоданих.

– Позиціонування за допомогою вишок стільникового зв'язка. Точність порядку 600 м, тому можна лише визначити, у якому місті й районі міста перебуває людина, але не його точне місце розташування.

– Wi-Fi. Точність може досягати 70 м. За допомогою цього виду геоданих можливо визначити, у якому приміщенні перебуває пристрій, але де конкретно в приміщенні – установити не вдасться.

– iBeacon/Bluetooth. Точність до 5 см, що точніше, ніж у будь-якого іншого типу геоданих, але застосування відповідних систем має свої нюанси. Для визначення положення виміряється сила сигналу протягом певного часу. Наприклад, для досягнення точності 5 см необхідно 10 с. Крім того, надійність вимірів сильно залежить від відстані до об'єкта й сили сигналу. Так, на відстані 30 см до об'єкта точність визначення його положення може скласти зазначені 5 див, при видаленні на 10 м – 50 см. Для зменшення помилки потрібно використовувати більше маяків. Таким чином, вартість подібної системи може виявитися високої для покриття, наприклад, торгового центра середнього розміру.

Для досягнення максимальної точності система геолокації може задіяти комбінацію різних технологій поряд з інерційними навігаційними системами (реалізуються за допомогою акселерометрів), які в деяких ситуаціях можуть давати виправлення й збільшити точність до декількох сантиметрів.

Активний розвиток систем для геолокації усередині будинків стимулює ріст інтересу замовників до даних технологій. Але поки на ринку немає чіткого подання про переваги використання геолокаційних сервісів для підвищення ефективності бізнес-процесів і наступної монетизації. Тим часом користувач, увійшовши в торговий центр, зможе за допомогою системи Digital Signage і встановленого на смартфоні додатка одержувати в інтерактивному або напівінтерактивному режимі на мобільний пристрій інформацію про магазини, акції, а також прокладати індивідуальний маршрут з урахуванням своїх переваг.

Аналітика

Крім інформації про місцезнаходження людини, для більше точного таргетування бажано знати його стать, вік і інші відомості. Один з варіантів одержання подібних даних – ідентифікація людини й звертання до його аккаунтів у соціальних мережах. Але це далеко не завжди можливо й не всім людям подобається. Разом з тим за останні кілька років сильно просунулися вперед системи відеоаналітик, що дозволяють визначати стать, вік і навіть емоцію людини, що перебуває перед екраном.

Істотний прорив у даній області був зроблений зовсім недавно, в 2010 році, коли застосування графічних процесорів (GPU) дозволило тренувати нейронні мережі з більшою кількістю шарів і зв'язків. Зараз в Інтернеті доступно досить багато відкритих даних з розміченими зображеннями (стать, вік, емоції) і є велика кількість статей з описом і тестами різних методів розпізнавання. Більше того в середині 2015 року Microsoft відкрив сайт how-old.net, що дозволяє на будь-якій фотографії знайти обличчя людей і визначити їхній вік. Американська компанія пішла далі й запропонувала цілий набір сервісів по розпізнаванню зображень, об'єднавши їх у проекті Project Oxford (www.projectoxford.ai). Проект

включає сервіси по розпізнаванню осіб, статі, віку й емоцій людей по фотографії; їх можна використовувати на комерційній основі через відкрите API.

Наявні рішення по визначенню статі, віку й емоцій є досить зрілими. Звичайно, одиничні помилки можливі, але статистично результати цілком прийнятні для практичного використання. В Україні поки подібні рішення поширюються в'ялуно в силу того, що звичайно виявляються в списку тих самих «надмірностей», від яких замовники відмовляються в кризу. Ми зі своєї сторони проробляємо й пропонуємо варіанти застосування подібних систем у надії, що справа дійде й до їхньої практичної реалізації.

Технології визначення статі, віку й емоцій людини – дуже потужний інструмент аналізу ефективності рекламних кампаній, але з ним треба вміти звертатися. Уже в найближчому майбутньому можна буде побачити перші масштабні впровадження подібних систем в Україні.

У цілому системи Digital Signage стрімко розвиваються, і головний вектор розвитку – забезпечити надання інформації, що цікава й корисна конкретній людині в цей момент у даному місці. Причому надаватися вона може через будь-який підходящий засіб відображення – від великої відеостіни до маленького екрана смартфону. Для рішення цього завдання використовуються найсучасніші технології, включаючи аналітикові Больших Даних, методи машинного навчання й т.д. Уже сьогодні відвідувачів «просунутих» торгових центрів зустрічають роботи, які ідентифікують їх через свою відеокамеру, відповідають на питання й дають персоніфіковані ради, наприклад, пропонуючи продукти, цікаві конкретній людині.

3.2 Розробка структурної схеми

Світлодіодні рекламні панелі це одне із кращих рішень для організації рекламних відеостін. Сучасні системи керування рекламними моніторами, дозволяють відрегулювати яскравість рекламного екрана автоматично відповідно

до зміни висвітлення в приміщенні й на відкритому повітрі, заощаджувати енерго-споживання рекламно-інформаційних дисплеїв для того, щоб значно зменшити експлуатаційні витрати.

Рекламні відеостіни звичайно розділяють по типах телевізійних панелей:

– РК рекламні панелі;

– LED рекламні панелі;

– рекламні плазмені панелі.

Світлодіодні рекламно інформаційний екрани для зовнішньої реклами будуються на базі LED панелей за рахунки того, що LCD рекламні панелі мають менше енергоспоживання, більшим терміном служби, не мають стиків і т.д. Комерційні рекламні лід панелі найбільше підходять для компаній, що займаються зовнішньою рекламою. LED відеостіни підходять для відтворення комерційної реклами в зовнішнім навколишнім середовищі. Висока швидкість відновлення, чітка картинка, висока контрастність і кольоровість, у тому числі насичене чорне світло зроблять картину на рекламній відеостіні більше реалістичної. Рекламні відео панелі відповідають підвищеному попиту рекламодавців з метою доставки до клієнта контенту високої візуальної якості. Рекламні екрани також підходять для всіх сфер телевізійного віщання й комерційного використання.

Рекламний або інформаційної вміст, відображуваний на рекламних відеостінах, може бути змінене в будь-який час віддалено, а так само в режимі реального часу може відображати різні рекламні блоки для різних клієнтів у різних рекламних оголошеннях.

Вилучене керування дозволяє настроїти всю інформацію рекламного дисплея, мережі, і змінити розташування рекламних блоків на екрані, у яких місцях вони б не перебували, декількома натисканнями миші в центрі керування. З рекламними ЛІД дисплеями досягти кластеризації міських і регіональних оголошень стало як ніколи просто й зручно. Контролювати рекламні екрани, а

також змінити вміст їхнього відображення в будь-який час і в різних містах можна з одного місця із центра керування.

Системи моніторингу навколишнього середовища, убудовані в рекламні відеостіни, вуличні стенди, вуличні рекламні дисплеї, вуличні рекламні панелі дозволять довідатися операції відображення в будь-який час і в будь-якому місці. Блоки автоматизації рекламних панелей дозволяють в автоматичному режимі настроїти функції включення й відключення LED екранів, регулювання яскравості й т.п.

Вимоги, пропоновані до рекламних світлодіодних дисплеїв:

– Надійний захист вуличних рекламних LED дисплеїв, здатна витримати екстремальні погодні умови.

– Енергозбереження – дозволяє знизити енергоспоживання й заощадити на експлуатаційних витратах на зміст рекламних відеостін

– Висока яскравість – для того щоб рекламні й інформаційні блоки були чіткими й контрастними навіть при сонячному світлі

– Широкий кут огляду – збільшує діапазон перегляду рекламного контенту й залучає більше уваги перехожих

– Однорідна й стійка картинка – видатна продуктивність рекламних дисплеїв дозволяє відображати відео високого (Full-HD) або надвисокого (Ultra-HD) дозволу на практично не обмежених розмірами площах у вигляді одного суцільного екрана

– Тривалий термін служби рекламних дисплеїв – дозволяє максимізувати переваги для клієнтів за 5 – 8 років безперервної роботи або навіть довше

Технічне рішення для рекламних LED дисплеїв

Високоякісні компоненти led панелей для реклами**:**

– Світлодіодні лампи: Висока яскравість LED лампи із широким спектром перегляду й низьким струмом, споживають менше енергії.

– Висока точність постійного струму: Світлодіодне підсвічування рекламних відео дисплеїв з контролем посилення, дозволяє підтримувати стабільну яскравість.

– Живлення: З функцією PFC (поправочний коефіцієнт потужності) світлодіодні панелі для реклами мають підвищену ефективність і енергопостачанням (за рахунок меншого виділення тепла заощаджують енергію).

– Оптичне волокно: Скорочення затримки сигналу, викликані передачею телевізійних інформаційних і рекламних блоків високого й ультра високого дозволу на більші відстані.

Сучасні технології й контроль виробництва рекламних блоків**:**

– Калібрування: піксель за пікселем перевіряється на яскравість із технікою колірного калібрування для того, щоб рекламні панелі мали рівномірну яскравість і природні квіти при зборі в загальний рекламний світлодіодний дисплей або рекламну відеостіну.

– Моніторинг: Убудована система керування може контролювати температуру, яскравість, інформацію про стан кожного світлодіодного дисплея в режимі реального часу.

– Дистанційне керування рекламними екранами: Керування й зміна інформації дисплеїв, контролюються покази на кілька міст із одного місця. Рекламні екрани можуть змінити вміст у будь-який час, включення й відключення живлення світлодіодного дисплея може бути зроблене по команді в будь-який час або в автоматичному режимі в певний час.

– Регульована яскравість LED: Регулювання яскравості рекламного екрана виробляється автоматично відповідно до навколишнього висвітлення на основі чутливої системи керування.

– Регулювання вручну або за допомогою програмувальних меню також можливі для кожного з рекламних дисплеїв відеостіни.

– Системи резервування: У випадку відмови системи керування, резервна система відразу починає роботу на підставі даних автоматичного резервування.

– Великий рекламний екран і керування: Керування супервеликим рекламним екраном, що складається з безлічі LED дисплеїв, виробляється стабільно, без чорних екранів, вібрацій кадрів або ефектів зменшення.

– Висока швидкість відновлення, високі градації сірого, чіткий чорний і висока контрастність комерційних ЛІД екранів: Більше реалістичні картинки оптимізовані під комерційну цінність оголошення (рекламного або інформаційного блоку). Запатентована конструкція установки й захисту світлодіодів дозволяє уникнути відбиття прямих сонячних променів і відкладення бруду для поліпшення контрастності.

Ефективні рішення відображення реклами на більших і надвеликих екранах**:**

– Енергозбереження: Світлодіодні лампи з низьким струмом, регульованою яскравістю, ефективним живленням з PFC, меншим тепловиділенням, спеціальною конструкцією охолодження й ідеальним розподілом потужності від системи керування рекламними екранами.

– Охолодження й дизайн: За допомогою спеціальних конструкцій, світлодіодних ламп на платі, радіатора, повітряних потоків, вентиляції й кондиціювання повітря, світлодіодний рекламний дисплей може продовжувати працювати у відповідній температурі.

– Тривалий термін служби рекламних екранів: зроблені методи енергозбереження, висока якість світлодіодних ламп, ефективні конструкції охолодження, регулювання яскравості, всебічний захист і ефективне живлення з PFC.

– Захист рекламних панелей: IP65 стандартні шафи для захисту світлодіодних дисплеїв в екстремальні погодні умовах, відомі брендові пристрої анти-блискавки. захисний шар спеціального захисного покриття на друкованій платі.

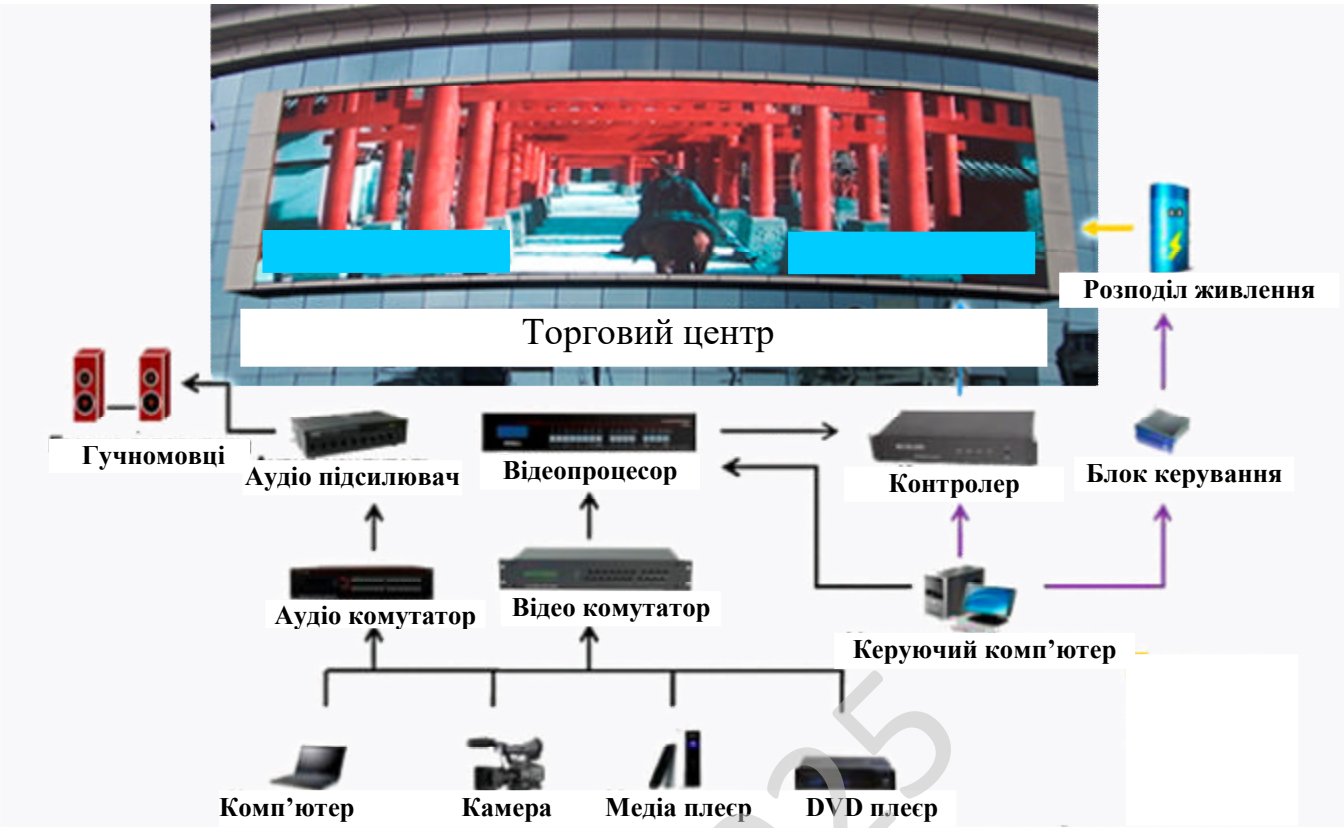

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Робочі місця учасників процесу відеоконференцій системи цифрових інформаційно-рекламних панелей

Загальні вимоги

Під «робочим місцем» розуміється комплект основного, додаткового й супутнього технологічного встаткування, а також інших засобів, розміщених на спеціально виділених площах і необхідних учасникові відеоконференцій системи цифрових інформаційно-рекламних панелей для виконання всіх передбачених технологією функцій і процесів. Робочі місця для різних учасників процесу відрізняються друг від друга по составі встаткування і їхньому оформленню залежно від характеру технологічних процесів, виконуваних кожним з учасників. Робоче місце оснащується технологічним устаткуванням і технологічними

меблями, засобами висвітлення, електропостачання й життєзабезпечення (опалення, вентиляції й кондиціювання повітря).

Загальні вимоги містять у собі також вимоги до технологічності, ергономіці, електропостачанню, висвітленню, шуму, безпеці, гігієні праці й профілактиці профзахворювань.

Робоче місце оператора сервера відеоконференцій системи цифрових інформаційно-рекламних панелей

В состав устаткування оператора сервера ВК системи цифрових інформаційно-рекламних панелей входить система резервування й керування відеоконференціями, професійний персональний комп'ютер з модемом і точка підключення до Інтернету.

Робочі місця користувачів системи відеоконференцій системи цифрових інформаційно-рекламних панелей

Види робочих місць. Залежно від видів конференції, учасників і категорії термінального встаткування можуть бути виділені чотири основних види робітників місць.

Індивідуальне робоче місце абонента (користувача) ВК системи цифрових інформаційно-рекламних панелей збігається з його постійним робочим місцем в офісі. Групове робоче місце перебуває або в безпосередній близькості від постійного робочого місця одного з учасників ВК системи цифрових інформаційно-рекламних панелей, або розміщається в окремо виділеному або пристосованому для цього приміщенні.

Робоче місце оператора ВК системи цифрових інформаційно-рекламних панелей територіально розташовується або поблизу групового робочого місця абонента ВК системи цифрових інформаційно-рекламних панелей, або на ділянці архіву ВК системи цифрових інформаційно-рекламних панелей, залежно від виконуваної роботи. До складу встаткування робочого місця оператора ВК системи цифрових інформаційно-рекламних панелей входить комплект устаткування монтажу, кодування відеофонограм і виготовлення архівних копій,

а також термінал керування архівом ВК системи цифрових інформаційно-рекламних панелей.

Робоче місце оператора сервера ВК системи цифрових інформаційно-рекламних панелей розташовується на ділянці зв'язку поблизу сервера ВК системи цифрових інформаційно-рекламних панелей і має термінал системи керування відеоконференціями.

Склад устаткування й структурні схеми. Робоче місце користувача системи відеоконференцій системи цифрових інформаційно-рекламних панелей включає абонентський термінал ВК системи цифрових інформаційно-рекламних панелей індивідуального або групового застосування з відеокамерою, мікрофоном, відеотерміналом, гучномовцями й електронними блоками; додаткове встаткування (документальну відеокамеру, комп'ютер, додаткові відеокамеру, мікрофон і відеомагнітофон) і супутнє встаткування (відеопроектор, системи спецосвітлення й звукопідсилення). Залежно від призначення й цілей застосування робочого місця користувача додаткове й супутнє встаткування використовують у різній комплектації..

Вимоги до розміщення встаткування й людей

Розміщення встаткування й людей – користувачів системи ВК системи цифрових інформаційно-рекламних панелей – повинне відповідати ряду суперечливих вимог, частина з яких нормована. По-перше, повинні бути виконані норми МСЕ-Р (сектора Р Міжнародного союзу електрозв'язку) і ЄСВ (Європейського союзу віщання), пропоновані до умов перегляду відеоматеріалу й прослуховуванню звукового матеріалу, по-друге, – технічні вимоги розміщення кожного з видів устаткування, по-третє, враховані норми СНіП у частині електро- і пожежебезпеки. Параметри приміщення й умови розміщення встаткування й людей підлягають розрахунку з урахуванням наступних норм і документів:

– співвідношення сторін приміщення – відповідно до «благозвучного» співвідношеннями Болта [1, 2];

– обсяг приміщення й час реверберації – відповідно до норм МСЕ-Р (МККР) серії BS [3, 4] і вимогами ЄСВ [5];

– мінімальна відстань гучномовців термінала ВК системи цифрових інформаційно-рекламних панелей від задньої стіни й розташування людей щодо гучномовців – відповідно до вимог МЕК [6] і МСЕ-Р [3, 7];

– шумові характеристики приміщення – відповідно до норм МСЕ-Р і ЄСВ [2, 3, 7];

– розташування людей щодо екрана відеомонітора – відповідно до вимог рекомендацій МСЕ-Р серії ВТ [8 -10];

– розміщення людей щодо стін приміщення – відповідно до вимог СНіП [11-13], а також з урахуванням вимог, пропонованих до висвітлення тла при відеозйомці.

Вимоги до розмірів приміщення визначаються з урахуванням акустичних вимог до часу реверберації, частотній характеристиці й раннім відбиттям [5, 7].

Оптимальне планування робочого місця досягається на підставі інженерного розрахунку.

В випадку розміщення учасників відеоконференції в залі використовують додаткове й супутнє встаткування. Основну відеокамеру, установлену звичайно на відеотерміналі, направляють убік доповідача на трибуні, а додаткову – у зал. Відеотерминал розташовують поблизу доповідача й оператора ВК системи цифрових інформаційно-рекламних панелей, щоб вони могли стежити за зображенням. Зображення для учасників у залі проектується на екран. Для них у залі встановлюють додаткові мікрофони й передбачають систему звукопідсилення. Для забезпечення правильної передачі кольору в залах використовують систему спецосвітлення.

Особлива увага варто приділяти акустиці приміщень. Акустична обробка поверхонь стін, стелі й підлоги повинна бути такий, щоб запобігти раннім відбиттям звуку, видаваного учасниками й вихідного від гучномовців. Авторові доводилося зіштовхуватися з випадками, коли через зневагу акустичними

вимогами помилка в наведенні відеокамери на робочому місці користувача ВК системи цифрових інформаційно-рекламних панелей перевищувала 200. При цьому камера наводилася не на промовця, а на його сусіда.

Для проведення групових відеоконференцій системи цифрових інформаційно-рекламних панелей важливо ретельно проробити світлотехнічні рішення. Так, колірна температура штучних джерел світла в залі повинна бути однакової щоб уникнути порушення передачі кольору.

Устаткування системи ВК системи цифрових інформаційно-рекламних панелей

Можливі схеми з'єднання терміналів ВК системи цифрових інформаційно-рекламних панелей Термінали відеоконференцій системи цифрових інформаційно-рекламних панелей з'єднуються один з одним за допомогою каналів зв'язку й коммунікаторів. Для багатоточечної системи відеоконференцій системи цифрових інформаційно-рекламних панелей головним коммунікатором є багатоточечний сервер ВК системи цифрових інформаційно-рекламних панелей. Залежно від взаємного розташування терміналів і сервера ВК системи цифрових інформаційно-рекламних панелей, режимів роботи пристроїв, а також використовуваних мереж зв'язки можливі різноманітні схеми з'єднання терміналів і серверів ВК системи цифрових інформаційно-рекламних панелей. При значній взаємній далекості терміналів ВК системи цифрових інформаційно-рекламних панелей зв'язок між ними й сервером, а також між ними самими (в окремих випадках) здійснюється з використанням каналів магістральних, міжрегіональних і регіональних мереж зв'язку.

У системі відеоконференцій системи цифрових інформаційно-рекламних панелей багатоточечний відеоконференцзв’язок може вироблятися по протоколах Н.320 і Н.323 МСЕ-Т. Зв'язок терміналів ВК системи цифрових інформаційно-рекламних панелей, що працюють по протоколі Н.323, із сервером ВК системи цифрових інформаційно-рекламних панелей може здійснюватися через вузли

(Gateway). З'єднання вузла Gateway із сервером ВК системи цифрових інформаційно-рекламних панелей виробляється по протоколах BRI і V.35/RS336.

Підключення сервера ВК системи цифрових інформаційно-рекламних панелей до магістральних, міжрегіональних і регіональних каналів зв'язку може вироблятися через маршрутизатор, магістральний мультиплексор і цифрову АТМ. З'єднання сервера із цими пристроями може здійснюватися з використанням протоколів PRI/E1, G.703. Підключення терміналів ВК системи цифрових інформаційно-рекламних панелей до магістральних, міжрегіональних і регіональних каналів зв'язку може відбуватися через мультиплексори земних станцій (HUB) і абонентських станцій (VSAT) супутникового зв'язку, магістральні мультиплексори (MUX) і АТМ. Залежно від відстані між терміналами й цим устаткуванням з'єднання можуть здійснюватися або безпосередньо з використанням протоколу V.35, або із застосуванням модемів або інверсних мультиплексорів.

Деякі особливості роботи підключення терміналів ВК системи цифрових інформаційно-рекламних панелей до каналоутворюючому встаткування

Безпосереднє підключення термінала ВК системи цифрових інформаційно-рекламних панелей до каналоутворюючому встаткування допускається в тих випадках, коли довжина сполучних кабелів невелика. У випадках, коли термінали й каналоутворююче встаткування вилучені друг від друга, для їхнього з'єднання повинні використовуватися модеми або інверсні мультиплексори.

Для входу в супутникову мережу термінали ВК системи цифрових інформаційно-рекламних панелей повинні бути підключені до мультиплексору центральної, вузлової або абонентської земної станції (ЗС) з використанням інтерфейсної плати внутрішнього модуля ЗС. Залежно від відстані між терміналом ВК системи цифрових інформаційно-рекламних панелей і внутрішнім модулем земної станції підключення виробляється прямо або із застосуванням

мультиплексора (модему). З'єднання рекомендується здійснювати з використанням протоколу V.35. Подібні способи підключення мають на увазі застосування твердої структури з'єднань, слабко пристосованої до подальшого розвитку. Вони доречні для використання в кінцеві (не вузлових) пунктах мережі відеоконференцзв’язку.

У тих випадках, коли об'єкт відеоконференцзв’язку розглядається як вузловий об'єкт, підключення термінала ВК системи цифрових інформаційно-рекламних панелей до внутрішнього модуля ЗС супутникового зв'язку варто здійснювати через мультиплексор.

Термінал ВК системи цифрових інформаційно-рекламних панелей підключається до інтерфейсному модуля мультиплексора прямо або через модем або інверсний мультиплексор, залежно від відстані між терміналом і мультиплексором.

На вузлових об'єктах з розвитою телефонною мережею доцільно підключати термінали до магістральних мереж через цифрову АТМ. При використанні цифровий АТМ у якості коммунікатора потоків даних відеоконференцзв’язку варто мати на увазі, що ці дані не повинні піддаватися стиску в АТМ, передбаченому для звичайних телефонних сигналів. Інакше кажучи, на період сеансу відеоконференцзв'язку повинна бути як би виділена смуга на вимогу для передачі цифрового потоку з необхідною бітною швидкістю (у загальному випадку 256 кбіт/с) для кожного з напрямків відеоконференцзв’язку.

На центральній ЗС або в центральному вузлі інформатизації з розгалуженими комунікаціями підключення термінала ВК системи цифрових інформаційно-рекламних панелей може бути здійснене через магістральний мультиплексор, цифрову телефонну станцію або маршрутизатор.

Підключення терміналів ВК системи цифрових інформаційно-рекламних панелей до магістральних, міжрегіональних і регіональних каналів зв'язку через мультиплексори, цифрові АТМ або маршрутизатори забезпечує ряд переваг у

порівнянні із прямим підключенням до ЗС. По-перше, добре розвинені комунікаційні функції цих пристроїв дозволяють їх використовувати як вузли відеоконференцзв’язку (хоча з деякими обмеженнями). По-друге, завдяки широкому діапазону інтерфейсів, властивим цим пристроям, досягається можливість зв'язку терміналів ВК системи цифрових інформаційно-рекламних панелей, що діють у різнорідних мережах (наприклад, супутникових і наземних, телефонних і мультимедійних і т.п.). При цьому забезпечується гнучкість структури системи відеоконференцзв’язку й подальше нарощування користувальницьких послуг, в основному програмними, а не апаратними засобами.

Системи цифрових інформаційно-рекламних панелей можливо використовувати також для організації відеоконференцій. Багатоточечний сервер відеоконференцій системи цифрових інформаційно-рекламних панелей (ВК) разом із системою керування відеоконференціями є гнучким модульним багатофункціональним засобом для підтримки багатоточечних ВК системи цифрових інформаційно-рекламних панелей. Функціональна схема сервера ВК системи цифрових інформаційно-рекламних панелей наведена на рисунку 3.2.

Крім модулів каналів зв'язку, сервер ВК системи цифрових інформаційно-рекламних панелей містить модулі звукового процесора й відеопроцесора, синхронізатора зображення й звуку, транскодера Н.261/Н.263, а також набір інтерфейсних модулів, зв'язаних системною шиною. Керуючі модулі керують роботою всієї системи. На «транковом» рівні сервер ВК системи цифрових інформаційно-рекламних панелей звичайно підтримує інтерфейси IDNX-PRI (Integrated Services Digital Network Primary Rate Interface) і T1 (E1) BBS (Robbed-Bit Signaling), на лінійному рівні – інтерфейс ISDN-BRI (Integrated Services Digital Network Basic Rate Interface), протокол цифрових з'єднань DCP (Digital Communications Protocol) і аналогові лінійні канали для з'єднання через модеми. Звичайно сервер ВК системи цифрових інформаційно-рекламних панелей заснований на стандартах Н.320 і Т.120 MCE-T (Сектора Т Міжнародного союзу

електрозв'язку) і має широкі можливості транскодування, що забезпечує сумісність його з найрізноманітнішими засобами відеоконференцій системи цифрових інформаційно-рекламних панелей. В таблиці 3.1 для приклада наведені деякі характеристики одного із серверів ВК системи цифрових інформаційно-рекламних панелей (MCU Lucent Technologies), що визначають його сумісність із іншим устаткуванням.

Сервер ВК системи цифрових інформаційно-рекламних панелей підтримує відеостандарти, звукові стандарти й стандарти даних, перераховані в таблицях 3.2-3.3.

Функціональна схема сервера відеоконференцій системи цифрових інформаційно-рекламних панелей зображена на рисунку 3.2.

Термінали відеоконференцій системи цифрових інформаційно-рекламних панелей

У системі відеоконференцій системи цифрових інформаційно-рекламних панелей можуть бути використані моделі групових напольних, компактних і настільних відеотерміналів різних виробників. Властивості систем різних фірм у кожній із груп досить близькі.

Ці засоби дозволяють працювати по протоколах H.320 і H.323 і поряд із груповими й компактними терміналами ВК системи цифрових інформаційно-рекламних панелей можуть бути включені в корпоративну систему ВК системи цифрових інформаційно-рекламних панелей.

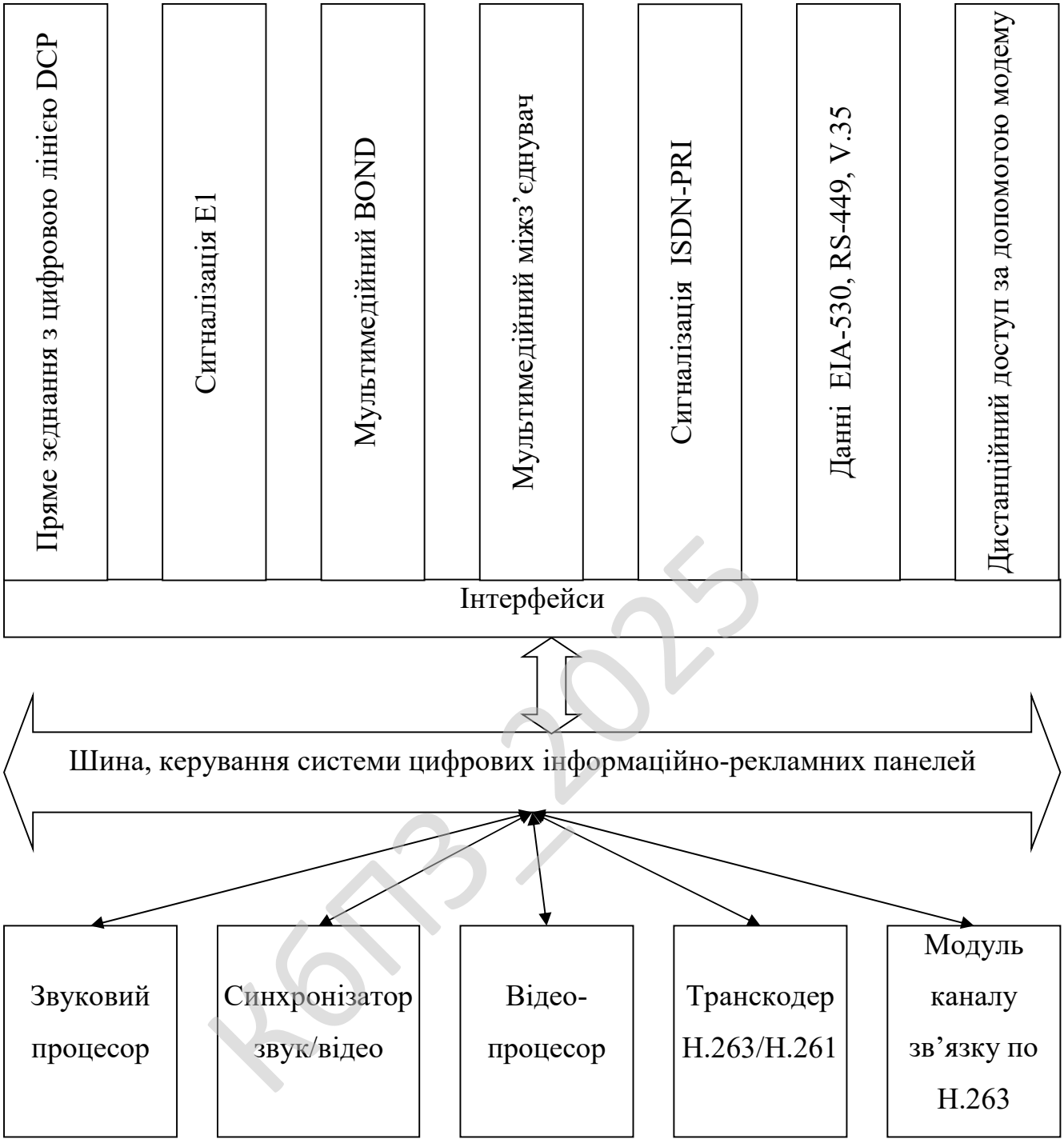

Рисунок 3.2 – Функціональна схема системи

Таблиця 3.1 – Можливості транскодування

Категорія	Стандарти й параметри інтерфейсів
Транскодування сигналів зображення	G.711 і G.722
	G.711 і G.728
	G.711 і G.723 (через Gateway)
	Частота кадрів – від 7,5 до 30 кадр/с Розрішення – CIF/QCIF Стиск – H.261/H.263
Доступ до мережі	Цифрова швидкість – від 56 до 768 кбіт/с Режими – BONDing, багатошвидкісний, багатоканальний
Багатоточечні протоколи	H.320 і H.323
Конференція даних (T.120)	Допускається в комбінованих конференціях Змішання H.320/H.323 (через Gateway)

Комунікаційні властивості встаткування системи ВК системи цифрових інформаційно-рекламних панелей

Основою технологічного встаткування відеоконференцзв’язку служать термінали ВК системи цифрових інформаційно-рекламних панелей різних конфігурацій і сервер ВК системи цифрових інформаційно-рекламних панелей із системою керування відеоконференціями. Сервер ВК системи цифрових інформаційно-рекламних панелей є головним телекомунікаційним засобом багатоточечної системи ВК системи цифрових інформаційно-рекламних панелей і має продуктивні цифрові інтерфейси для високошвидкісної передачі голосових повідомлень, зображень і даних. Він містить інтерфейси цифрових мереж інтегрованих служб (ISDN): для базової цифрової швидкості (BRI) і первинної цифрової швидкості (PRI). Крім того, він підтримує протокол прямих цифрових з'єднань (DCP). При з'єднанні ISDN-PRI на рівні первинної групи группостворення E1 застосовується формат 30B + D, а при ISDN-BRI, – формат

2B + D, де B (64 кбіт/с) – потік для основної інформації служби, а D (16 кбіт/с) – потік керування й сигналізації для приєднаних каналів B. Протокол цифрових з'єднань DCP застосовується для зв'язку терміналів ВК системи цифрових інформаційно-рекламних панелей із системою керування за допомогою високошвидкісних і мультимедійних з'єднань (MML).

Таблиця 3.2 – Відеостандарти, підтримувані сервером ВК системи цифрових інформаційно-рекламних панелей

Відеостандарт МСЕ-Т	Найменування, зміст
H.221	Стандарт структури кадрів для відеоконференцзв’язку
H.230	Стандарт кадрової синхронізації для відеоконференцзв’язку
H.231	Стандарт відеоконференцзв’язку, що визначає з'єднання між звуковізуальними терміналами
H.242	Стандарт, що визначає системи для подання з'єднань між звуковізуальними терміналами
H.243	Рекомендації ITU-T: процедура встановлення зв'язку між трьома або більше аудіовізуальними терміналами, що використовують канали зі швидкістю передачі цифрової інформації до 2 Мб/с.
H.261	Стандарт відеокодека для звуковізуальних служб зі швидкістю H.320
H.263	Кодування й декодування зображення для передачі з низькою швидкістю з поліпшеними характеристиками і якістю по каналах H.261

Таблиця 3.3 – Звукові стандарти, підтримувані сервером ВК системи цифрових інформаційно-рекламних панелей

Звуковий стандарт МСЕ-Т	Параметри якості	Смуга, бітна швидкість
G.711	3,5 кГц	48/56/64 кбіт/с
G.722	7 кГц	48 кбіт/с на швидкості N*56 кбіт/с
		56 кбіт/с на швидкості N*64 кбіт/с
G.728	3.5 кГц	16 кбіт/с

Таблиця 3.4 – Стандарти даних, підтримувані сервером ВК системи цифрових інформаційно-рекламних панелей

Стандарти МСЕ-Т	Найменування, зміст
Т. 122	Багатоточечна служба зв'язку для звукографії й відеоконференцзв’язку
Т.123	Стеки протоколів ISDNProfile-MLP для застосувань звукографії й відеоконференцзв’язку
Т.124	Специфікація GCC (Genetic Conference Protocol)
Т. 125	Протокол MC5
Т.126	Протокол для нерухливих зображень і описів при багатоточечному зв'язку
Т.127	Протокол перетворень багатоточечних бінарних файлів

Сервер ВК системи цифрових інформаційно-рекламних панелей може підключатися до кодеків H.320, H.CTX для передачі даних через інтерфейси EIA-530, RS449 або V.35 із застосуванням RS366 для сигналізації (дозвона). У випадку надання смуги на вимогу (BONDing) можна здійснювати проведення відеоконференцій системи цифрових інформаційно-рекламних панелей без використання каналів ISDN-PRI і широкополосних каналів типу H0.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

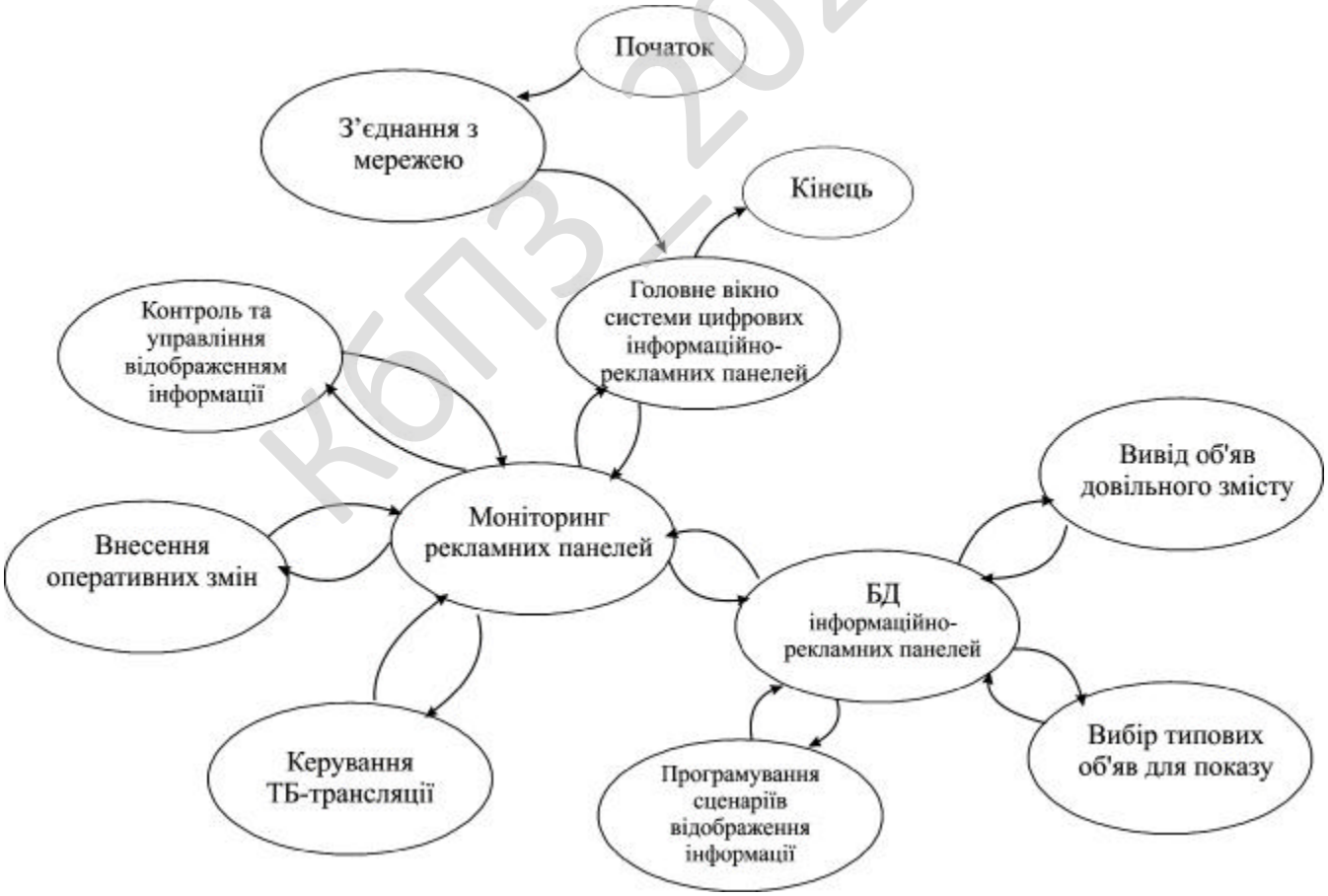

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є основою ПЗ. Тому від точності і детальності проробки блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації, також те, що при розробці програми слід надати особливу увагу модулю системи цифрових інформаційно-рекламних панелей.

Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні блоки можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірки поточного стану та поверненням на початок схеми чи з завершенням роботи розробленого ПЗ.

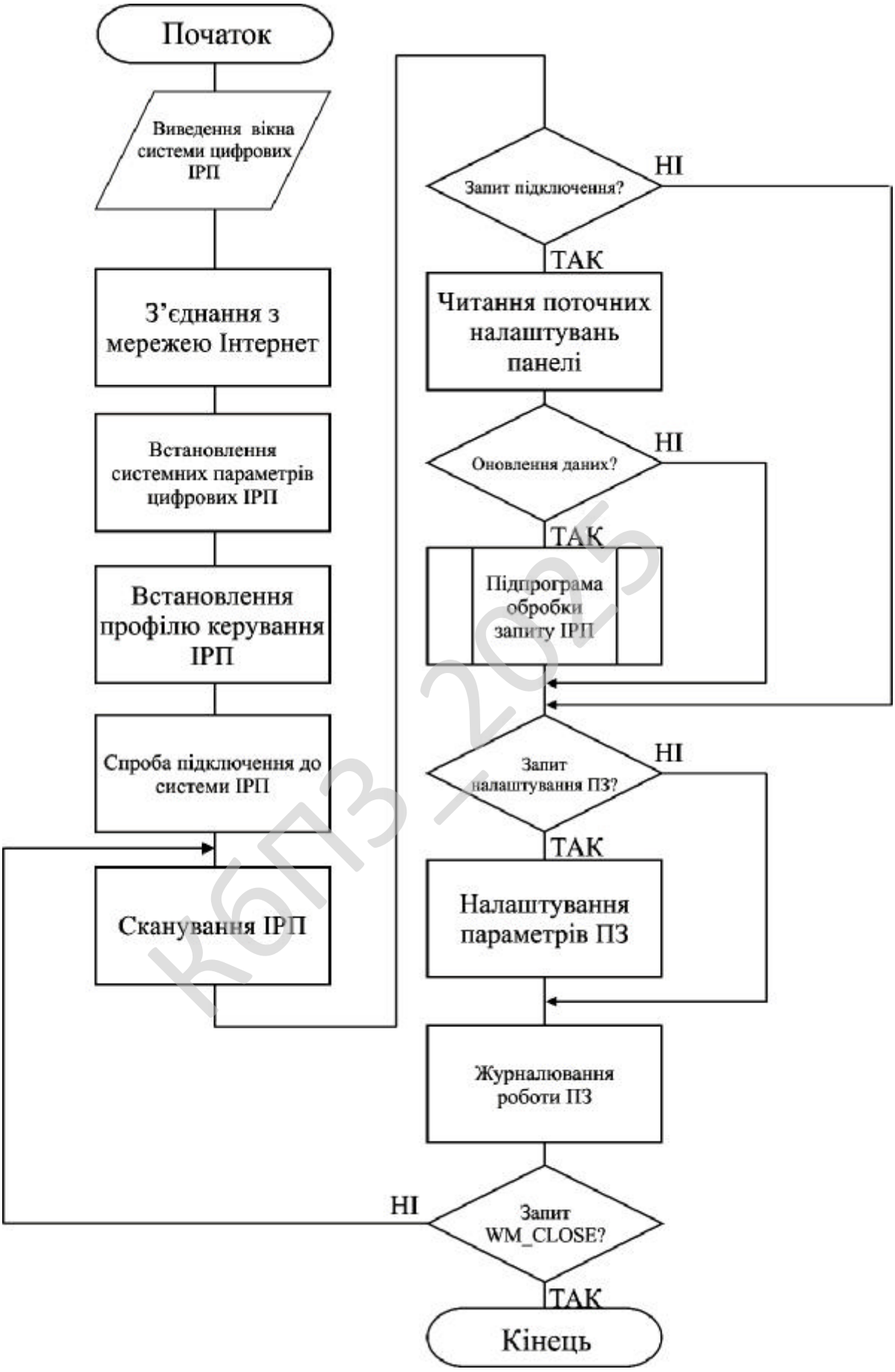

Рисунок 4.1 – Блок-схема основної програми

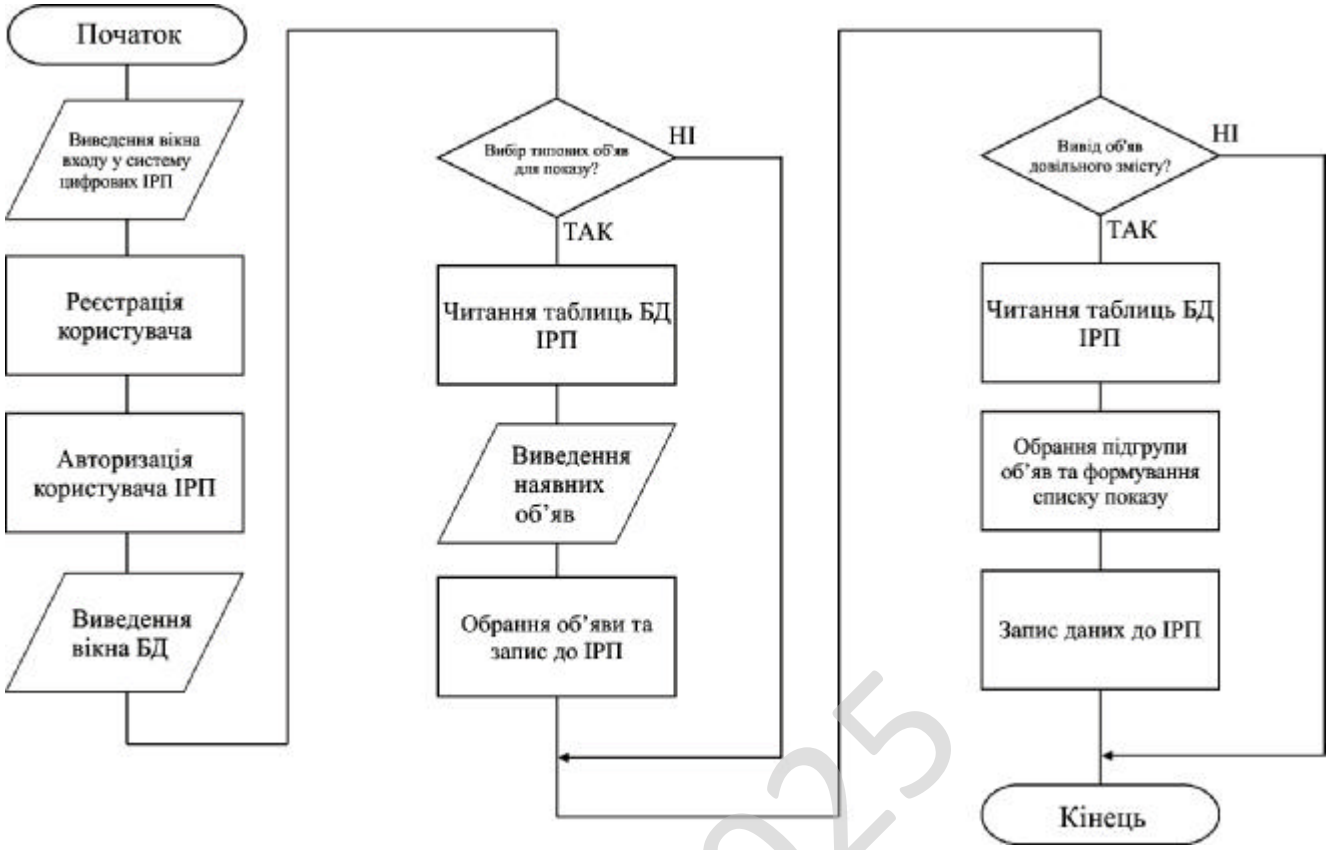

Рисунок 4.2 – Блок-схема роботи підпрограми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

Розглянемо використані технології та їх основні компоненти що підтверджують правильність використаних проектних рішень.

Була використана водоспадна (каскадна) модель життєвого циклу ПЗ (waterfall model) – послідовний метод розробки програмного забезпечення, названий так через діаграму схожу на водоспад.

Ця модель розробки запозичена з системної інженерії у виробництві та будівництві – областях, в яких зміни на пізніх етапах дуже дорогі, або неможливі. Наприклад, для створення складних інженерних конструкцій (споруд, літаків, мостів і т.п.). Зміни в проекті фундаменту будинку після того, як покладений дах коштують дуже дорого, тому перфекціонізм на початкових етапах проектування просто необхідний. Інженери, які починали займатись розробкою програмного забезпечення перейшовши з інших галузей, просто адаптували звичну модель, тому що на ранніх етапах розвитку комп'ютерної техніки не було методологій створених саме для програмування. Проте, схожі методології застосовуються для програмного забезпечення й далі, у випадках коли вимоги фіксовані, і вимагається висока якість та надійність, наприклад в системах для військових чи медичних потреб.

Перший формальний опис водоспадної моделі, після якої вона стала популярною був здійснений В. В. Ройсом у 1970. Попри те, що стаття містить переважно критику методу, на неї часто посилаються.

Переваги методу:

– Ніяких переробок.

– Гарна специфікація перетікає в гарну документацію.

– Зрозуміла модель.

– Розробники можуть мати низьку кваліфікацію.

Недоліки:

– Необхідний перфекціонізм на кожному етапі.

– Важко вносити зміни (якщо взагалі можливо).

– Надлишкове проектування.

– Поділ розробників на "perfect" та "code monkeys".

Модифікації. Через те що цей метод погано підходить для розробки саме ПЗ, частіше використовують його модифікації.

Найвідоміша модифікація – Sashimi. Названа так через японську страву сашимі (суші нарізане і сервіроване так, що складені рядочком шматочки накладаються один на одного). В моделі розробки Сашимі фази життєвого циклу йдуть одна за одною, але при цьому перекриваються одна з одною в часі.

Опис системи цифрових інформаційно-рекламних панелей передбачає створення програмного забезпечення для керування панелями, розподілу контенту та обробки даних. У цій системі виділяються основні модулі:

1. Модуль управління контентом – забезпечує додавання, видалення та редагування інформаційного контенту.

2. Модуль розподілу контенту – відповідає за передачу контенту на різні інформаційні панелі в режимі реального часу.

3. Модуль моніторингу і статистики – забезпечує збір і аналіз даних про роботу панелей та взаємодію користувачів з контентом.

Архітектура системи базується на клієнт-серверній моделі. Серверна частина відповідає за зберігання даних, обробку запитів і управління контентом. Клієнтська частина представлена дисплеями інформаційних панелей, які відображають контент.

Пропонований код ілюструє основну архітектуру системи на Python.

```
import sqlite3
import datetime
import time
from flask import Flask, request, jsonify

# Створюємо додаток Flask
app = Flask(__name__)

# Ініціалізація бази даних для зберігання контенту панелей
def init_db():
    connection = sqlite3.connect('panels.db')
    cursor = connection.cursor()
```

```
    # Таблиця контенту
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS content (
        id INTEGER PRIMARY KEY,
        title TEXT NOT NULL,
        description TEXT,
        display_time INTEGER NOT NULL,
        created_at TEXT NOT NULL
    )
    ''')
    connection.commit()
    connection.close()

# Додаємо новий контент до бази даних
def add_content(title, description, display_time):
    connection = sqlite3.connect('panels.db')
    cursor = connection.cursor()
    created_at = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute('INSERT INTO content (title, description, display_time,
created_at) VALUES (?, ?, ?, ?)',
                   (title, description, display_time, created_at))
    connection.commit()
    connection.close()

# Отримуємо весь контент для відображення
def get_all_content():
    connection = sqlite3.connect('panels.db')
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM content')
    result = cursor.fetchall()
    connection.close()
    return result

# Відображення контенту на інформаційних панелях
@app.route('/display', methods=['GET'])
def display_content():
    content = get_all_content()
    return jsonify(content)

# Маршрут для додавання контенту
@app.route('/add_content', methods=['POST'])
def add_content_route():
```

```
    data = request.get_json()
    title = data.get('title')
    description = data.get('description', '')
    display_time = data.get('display_time', 10)

    add_content(title, description, display_time)
    return jsonify({'message': 'Content added successfully'})

# Ініціалізуємо базу даних при старті додатку
if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Опис роботи системи

База даних містить інформацію про контент панелей (заголовок, опис, тривалість показу і час створення).

API сервер створений на основі Flask. Він дозволяє додавати новий контент через POST-запити і отримувати список контенту для відображення через GET-запити.

Інтерфейс для інформаційних панелей підключається до сервера і відображає контент у реальному часі.

Розрахунок часу відображення контенту дозволяє визначити оптимальну тривалість показу кожного елемента для уникнення перевантаження панелей.

Тестування системи показує, що кожна інформаційна панель здатна обробляти і відображати контент з мінімальною затримкою. Припустимо, що середній розмір контенту становить 1 МБ. Час завантаження контенту через локальну мережу (зі швидкістю 100 Мбіт/с) не перевищує 0,1 секунди, що є допустимим значенням.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми.

Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є

важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ.

Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення –звичайний браузер.

Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої

обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму Camellia – блоковий шифр на основі мережі Фейстеля. У криптографії, Camellia – це симетричний ключ блоковий шифр із розміром блоку 128 біт і розмірами ключа 128, 192 і 256 біт. Він був розроблений спільно Mitsubishi Electric і NTT з Японії. Шифр був схвалений для використання ISO / IEC, проектом Європейського Союзу NESSIE і Японським CRYPTREC проект. шифр має рівні безпеки й можливості обробки, порівнянні з Advanced Encryption Standard.

Шифр був розроблений, щоб підходити як для програмних, так і для апаратних реалізацій, від недорогих смарт-карти для високошвидкісних мережних систем. Він є частиною криптографічного протоколу Transport Layer Security (TLS), призначеного для забезпечення безпеки зв'язки в комп'ютерній мережі, такий як Інтернет

Camellia – це шифр Фейстеля з 18 раундами (при використанні 128-бітних ключів) або 24 раундами (при використанні 192- або 256-бітних ключів). Кожні шість раундів застосовується шар логічного перетворення: так звана «FL-функція» або її зворотна. Camellia використовує чотири 8 × 8-бітних S-блоку із вхідними й вихідними афінними перетвореннями й логічними операціями. Шифр також використовує введення й вивід відбілювання клавіш. Шар дифузія використовує лінійне перетворення на основі матриці з номером галузей 5.

Аналіз безпеки

Камелія вважається сучасним надійним шифром. Навіть при використанні параметра меншого розміру ключа (128 біт) уважається неможливим зламати його за допомогою атаки грубої сили на ключі за допомогою сучасних технологій. Немає відомих успішних атак, що значно послабляють шифр. Шифр був схвалений для використання ISO / IEC, проектом Європейського Союзу NESSIE і Японським CRYPTREC проект. Японський шифр має рівні безпеки й можливості обробки, порівнянні із шифром AES/Rijndael.

Camellia – це блоковий шифр, який може бути повністю визначені мінімальними системами багатомірних багаточленів:

– Камелія (а також AES) S-блоки можуть бути описані системою 23 квадратних рівнянь в 80 членах.

– Розклад ключів можна описати 1120 рівняннями в 768 змінні з використанням 3328 лінійних і квадратичних членів.

– Увесь блоковий шифр можна описати 5104 рівняннями в 2816 змінні з використанням 14 592 лінійних і квадратичних членів.

– Усього потрібно 6224 рівняння з 3584 змінними з використанням 17 920 лінійних і квадратичних членів.

– Кількість вільних членів становить 11 696, що приблизно таке ж число, що й для AES.

Теоретично, такі властивості можуть дозволити зламати Camellia (і AES) за допомогою алгебраїчної атаки, такий як розширена розріджена лінеаризація, у т Майбутнє за умови, що атака стане можливою.

Хоча Camellia запатентована, вона доступна за безоплатною ліцензією. Це дозволило шифру Camellia стати частиною проекту OpenSSL під ліцензією з відкритим вихідним кодом з листопада 2006 року. Це також дозволило йому стати частиною Mozilla Модуль NSS (Служби мережної безпеки).

Підтримка Camellia була додана в остаточний випуск Mozilla Firefox 3 в 2008 році (за замовчуванням відключене починаючи з Firefox 33 в 2014 році в дусі «Пропозиції по зміні стандартних наборів шифрів TLS, пропонованих браузерами», який був виключено з версії 37 в 2015 році). Pale Moon, відгалуження Mozilla / Firefox, продовжує пропонувати Camellia і розширив свою підтримку, включивши в нього набори Galois / Counter mode (GCM) із шифром, але вилучив GCM знову у випуску 27.2.0, пославшись на очевидну відсутність інтересу до них.

Пізніше, в 2008 році, група розробки релізу FreeBSD оголосила, що цей шифр також був включений в FreeBSD 6.4. Крім того, Йошисато Янагисава додав підтримку шифру Camellia у дисковий клас зберігання geli FreeBSD.

У вересні 2009 року GNU Privacy Guard додала підтримку Camellia у версії 1.4.10.

Veracrypt (відгалуження Truecrypt) включав Camellia як один з підтримуваних алгоритмів шифрування.

Крім того, різні популярні бібліотеки безпеки, такі як Crypto ++, Gnutls, mbed TLS і Openssl також включають підтримку Camellia.

26 березня 2013 р. було оголошено, що Camellia була знову обрана для включення в новий список рекомендованих шифрів для електронного уряду Японії як єдиний 128-бітний алгоритм блокового шифрування, розроблений у Японії. Це збігається з тим, що список CRYPTREC обновляється вперше за 10 років. Вибір був заснований на високій репутації Camellia у плані простоти придбання, а також характеристик безпеки й продуктивності, порівнянних з такими з Advanced Encryption Standard (AES). Камелія залишається незмінною у своєму повному втіленні. Неможлива диференціальна атака на Camellia з 12 раундами без шарів FL / FL дійсно існує.

Продуктивність

S-блоки, використовувані Camellia, мають структуру, аналогічну S-блоку AES. У результаті можна прискорити реалізацію програмного забезпечення Camellia за допомогою наборів команд ЦП, розроблених для AES, таких як x86 AES-NI.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи цифрових інформаційно-рекламних панелей яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; БД об’яв; Налаштування; Довідка.

– Функції представлені у графічному вигляді.

– Розділу обрання панелі.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

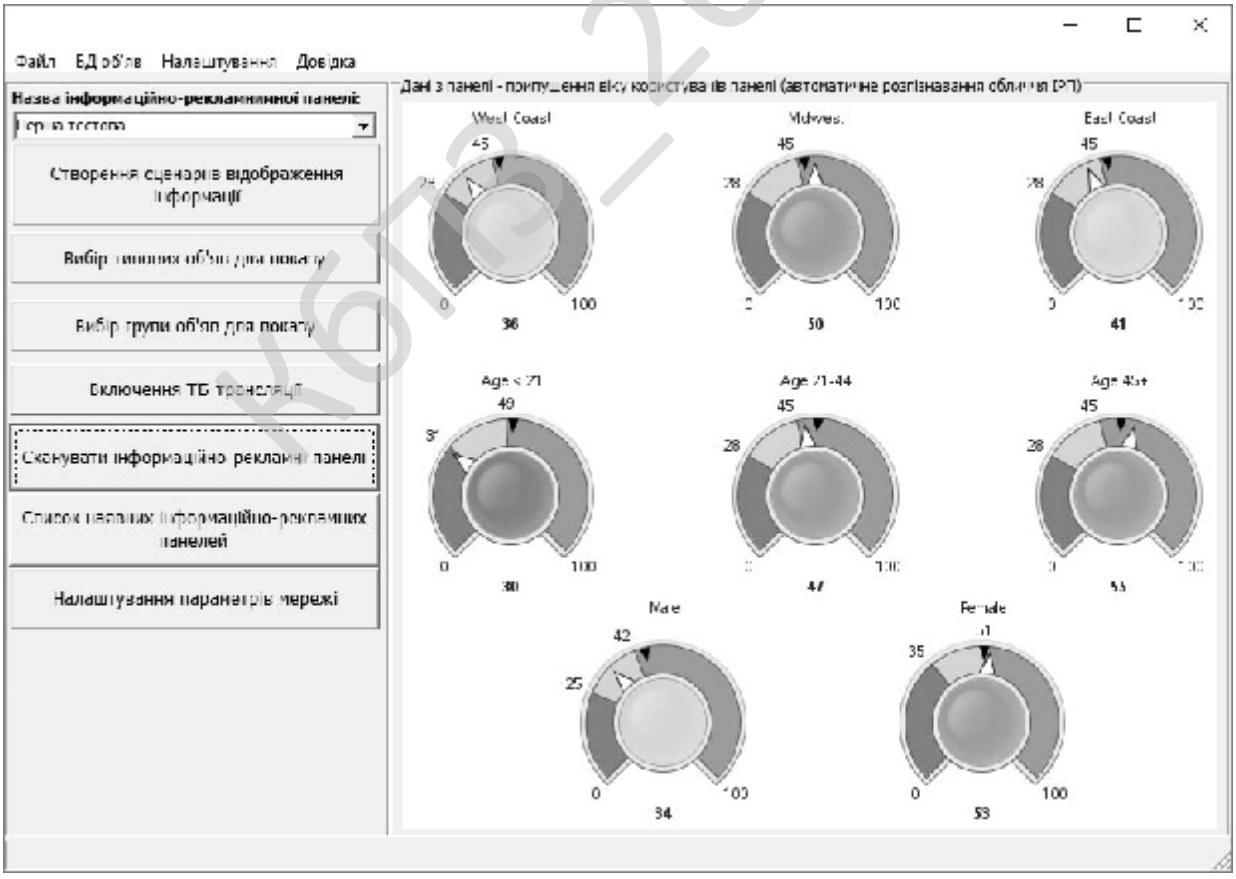

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

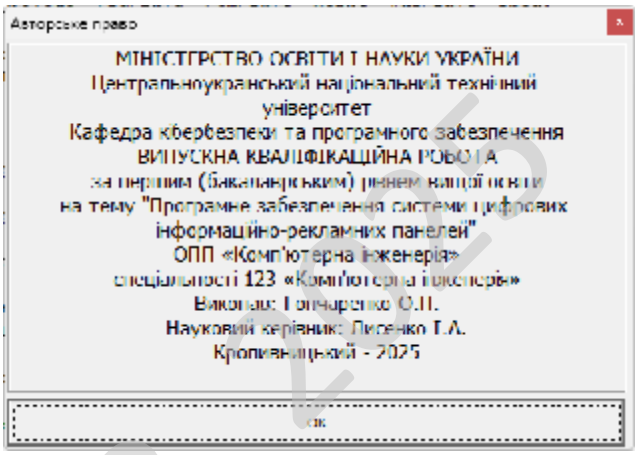

Рисунок 5.2 – Авторське право

Для розподілу контенту в системах Digital Signage всі частіше використовується технологія передачі відео по IP. З безлічі технологій передачі відеоконтенту в системах Digital Signage найцікавіші ті, що надають можливість простого переналаштування системи. Така потреба викликана частими змінами схеми трансляції контенту до різних пристроїв відображення у зв'язку з постійними перепланування ми приміщень у сфері ретейла, зміною орендарів, зміною площ, переносом пристроїв відображення на нові місця для більше широкого охвату цільової аудиторії й т.п.

IP-рішення надають подібну гнучкість. До їхніх переваг ставиться також можливість використовувати наявну мережу, створивши для систем Digital

Signage окрему незалежну віртуальну локальну мережу (VLAN). Це дозволяє заощадити істотні засоби. Рішення на базі IP легко масштабуються – можна нарощувати як число крапок трансляції, так і кількість пристроїв відображення. А далекість пристрою відображення (монітора) від джерела визначається топологією мережі, причому у випадку оптичних ліній зв'язку відстань між ними практично нічим не обмежується.

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати ІТ рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Обрано умови розповсюдження – Freeware.

Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи цифрових інформаційно-рекламних панелей.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем цифрових інформаційно-рекламних панелей.

– Досліджена система цифрових інформаційно-рекламних панелей.

– На основі отриманих результатів досліджень створена програмна реалізація системи цифрових інформаційно-рекламних панелей.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання цифрових інформаційно-рекламних панелей.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи цифрових інформаційно-рекламних панелей. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Camellia.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.

2. Інженерна комп’ютерна графіка: підручник / В.В. Проців [та ін.] / М-во освіти і науки України, Нац. гірн. унт-т. – Дніпро: НГУ, 2017. – 247 с.

3. Проців В.В. Прикладна комп'ютерна графіка [Текст]: Навч. посібник / В.В. Проців, К.А. Зіборов, К.М. Бас, Г.К. Ванжа; М-во освіти і наук, Нац. гірн. ун-т. - Д.: НГУ, 2016. - 187 с.

4. Kopf, Johannes and Lischinski, Dani. Depixelizing Pixel Art (англ.) // ACM Trans. Graph. – 2011. – Vol. 30, no. 4. – P. 99:1--99:8.

5. Giachetti, Andrea and Asuni, Nicola. Real-Time Artifact-Free Image Upscaling (англ.) // Trans. Img. Proc.. – 2011. – Vol. 20, no. 10. – P. 2760—2768.

6. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

7. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447

8. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

9. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

10. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

11. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022,

12. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

13. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w

14. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021, Cracow, Poland, 22-25 September 2021. P. 414-418.

15. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». 4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

16. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

17. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

18. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». Journal of theoretical and applied information technology Vol.98. No 21, 2020, P. 3334-3346.

19. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». CEUR Workshop Proceedings Volume 2654, 2020, Pages 1-14.

20. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

21. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

22. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

23. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

24. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

25. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», CEUR Workshop Proceedings Volume 2608, 2020, Pages 646-660.

26. Zhurakovskyi, B., Tsopa, N., Batrak, Y., Odarchenko, R., Smirnova, T «Comparative analysis of modern formats of lossy audio compression». Workshop Proceedings, 2020, 2654, стр. 315-327.

27. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

28. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

29. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

30. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

31. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and

Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

32. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

33. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

34. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

35. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». CEUR Workshop Proceedings Volume 2732, 2020, Pages 214-227.

36. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

37. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

38. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

39. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Хмарна інформаційна система оцінювання шорсткості з використанням дискретного частотного аналізу макрофотографій». IV міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 15-16 квітня 2021р. – Кропивницький: ЦНТУ. – 2021. – С. 30.

40. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у Кібербезпека та інформаційні технології: монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

41. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

42. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

43. О. Смірнов, Є. Деменко, О. Онікійчук, А. Арищенко, Л. Горбачова, «Формування псевдовипадкових послідовностей для приховування даних в зображеннях» Комп’ютерні науки та кібербезпека. № 4. С. 30-37. 2019.

44. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп’ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

45. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI’2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

46. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна

безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

47. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

48. Смірнов О.А., Кавун С.В., Коваленко О.В., Дрєєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.

49. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

50. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.

51. Смірнов О.А., Дрєєв О.М. Порівняння бітових щільностей при використанні різних методів кодування інформації. Збірник наукових праць "Системи обробки інформації". - Випуск 2 (118). т.2. - Х.: ХУПС - 2014. - С. 64-67

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи цифрових інформаційно-рекламних панелей.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 46-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи цифрових інформаційно-рекламних панелей.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи цифрових інформаційно-рекламних панелей;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 70 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 2.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Лисенко І.А.

Програмне забезпечення системи цифрових інформаційно-рекламних панелей

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 34

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
# Основний код системи цифрових інформаційно-рекламних панелей

# Імпорт стандартних модулів
import sqlite3
import time
import threading
import random
import datetime
import requests
import json
import os
import sys

# Підключення файлів розширень
import extensions_1
import extensions_2
import extensions_3

# ============================================================================
# Основні класи та функції системи цифрових інформаційно-рекламних панелей
# ============================================================================

# Клас для управління базою даних
class DatabaseManager:
# Ініціалізація менеджера бази даних та встановлення з'єднання
    def __init__(self, db_file):
        # Ініціалізація параметрів бази даних
        self.db_file = db_file
        self.connection = None
        self.cursor = None
        self.connect_db()

    def connect_db(self):
        # Підключення до бази даних SQLite з урахуванням потокобезпеки
        self.connection = sqlite3.connect(self.db_file, check_same_thread=False)
        self.cursor = self.connection.cursor()
        self.create_tables()

    def create_tables(self):
# Створення таблиці панелей, якщо вона не існує
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS panels (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                location TEXT NOT NULL,
                last_updated TIMESTAMP
            )
        ''')
# Створення таблиці рекламних оголошень, якщо вона не існує
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS advertisements (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                panel_id INTEGER,
                content TEXT,
                duration INTEGER,
                start_time TIMESTAMP,
                end_time TIMESTAMP,
                active INTEGER,
                FOREIGN KEY (panel_id) REFERENCES panels (id)
            )
        ''')
        self.connection.commit()

    def add_panel(self, name, location):
        # Додавання нової панелі до бази даних
        now = datetime.datetime.now()
```

```
        self.cursor.execute(
            'INSERT INTO panels (name, location, last_updated) VALUES (?, ?,
?)',
            (name, location, now)
        )
        self.connection.commit()
        return self.cursor.lastrowid

    def update_panel(self, panel_id):
        # Оновлення часу останнього оновлення для панелі
        now = datetime.datetime.now()
        self.cursor.execute(
            'UPDATE panels SET last_updated = ? WHERE id = ?',
            (now, panel_id)
        )
        self.connection.commit()

    def add_advertisement(self, panel_id, content, duration, start_time,
end_time, active=1):
# Додавання нового рекламного оголошення для певної панелі
        self.cursor.execute(
            'INSERT INTO advertisements (panel_id, content, duration,
start_time, end_time, active) VALUES (?, ?, ?, ?, ?, ?)',
            (panel_id, content, duration, start_time, end_time, active)
        )
        self.connection.commit()
        return self.cursor.lastrowid

    def update_advertisement_status(self, ad_id, active):
# Оновлення статусу рекламного оголошення (активне/неактивне)
        self.cursor.execute(
            'UPDATE advertisements SET active = ? WHERE id = ?',
            (active, ad_id)
        )
        self.connection.commit()

    def get_active_advertisements(self, panel_id):
# Отримання всіх активних рекламних оголошень для заданої панелі
        now = datetime.datetime.now()
        self.cursor.execute('''
            SELECT id, content, duration, start_time, end_time
            FROM advertisements
WHERE panel_id = ? AND active = 1 AND start_time <= ? AND end_time >= ?
            ORDER BY start_time ASC
        ''', (panel_id, now, now))
        return self.cursor.fetchall()

    def get_all_panels(self):
# Отримання списку всіх панелей з бази даних
        self.cursor.execute('SELECT id, name, location, last_updated FROM
panels')
        return self.cursor.fetchall()

    def close(self):
        # Закриття з'єднання з базою даних
        self.connection.close()

# Клас для представлення цифрової інформаційно-рекламної панелі
class Panel:
    def __init__(self, panel_id, name, location, db_manager):
        # Ініціалізація параметрів панелі
        self.panel_id = panel_id
        self.name = name
        self.location = location
        self.db_manager = db_manager
        self.current_ad_index = 0
        self.advertisements = []
        self.lock = threading.Lock()
```

```
    def refresh_advertisements(self):
        # Оновлення списку рекламних оголошень з бази даних
        ads = self.db_manager.get_active_advertisements(self.panel_id)
        self.lock.acquire()
        self.advertisements = ads
        self.lock.release()

    def display(self):
        # Відображення рекламного оголошення на панелі
        self.lock.acquire()
        if not self.advertisements:
            print(f"[{datetime.datetime.now()}] Панель '{self.name}'
({self.location}): Немає активних оголошень.")
            self.lock.release()
            return
        ad = self.advertisements[self.current_ad_index]
        ad_id, content, duration, start_time, end_time = ad
        print(f"[{datetime.datetime.now()}] Панель '{self.name}'
({self.location}) відображає оголошення: {content}")
        self.current_ad_index = (self.current_ad_index + 1) %
len(self.advertisements)
        self.lock.release()
        time.sleep(duration)

# Функція для симуляції отримання рекламного оголошення з віддаленого джерела
def fetch_remote_advertisement():
    # Імітація мережевої затримки
    time.sleep(random.uniform(0.5, 2.0))
    # Список можливих рекламних повідомлень
    ad_contents = [
        "Купи один, отримай другий безкоштовно!",
        "Обмежена пропозиція: знижка 50%!",
        "Нова колекція вже в продажу!",
        "Ексклюзивна пропозиція для наших клієнтів!",
        "Літній розпродаж розпочато!"
    ]
    return random.choice(ad_contents)

# Функція для оновлення рекламних оголошень для конкретної панелі
def update_advertisements(db_manager, panel_id):
    now = datetime.datetime.now()
    for i in range(5):
        content = fetch_remote_advertisement()
        duration = random.randint(5, 15)
        start_time = now + datetime.timedelta(seconds=random.randint(0, 30))
        end_time = start_time + datetime.timedelta(seconds=duration * 3)
        db_manager.add_advertisement(panel_id, content, duration, start_time,
end_time)
        time.sleep(0.1)

# Функція циклічного відображення оголошень на панелі
def panel_loop(panel):
    while True:
        panel.refresh_advertisements()
        panel.db_manager.update_panel(panel.panel_id)
        panel.display()

# Функція для запуску системи панелей: створення потоків для кожної панелі
def start_panels_system(db_manager):
    panels_data = db_manager.get_all_panels()
    panels = []
    threads = []
    for data in panels_data:
        panel_id, name, location, last_updated = data
        panel_instance = Panel(panel_id, name, location, db_manager)
        panels.append(panel_instance)
        t = threading.Thread(target=panel_loop, args=(panel_instance,))
        t.daemon = True
        t.start()
```

```
        threads.append(t)
    return panels, threads

# Функція для періодичного оновлення рекламних оголошень для всіх панелей
def advertisement_updater(db_manager):
    while True:
        panels = db_manager.get_all_panels()
        for panel in panels:
            panel_id, name, location, last_updated = panel
            update_advertisements(db_manager, panel_id)
        time.sleep(30)

# Функція для логування подій системи у файл та консоль
def log_event(event_message):
    now = datetime.datetime.now()
    log_message = f"[{now}] {event_message}"
    with open("system_log.txt", "a", encoding="utf-8") as log_file:
        log_file.write(log_message + "\n")
    print(log_message)

# Клас для управління конфігурацією панелей (файл panel_config.json)
class PanelConfig:
    def __init__(self, config_file="panel_config.json"):
        # Ініціалізація об'єкта конфігурації панелей
        self.config_file = config_file
        self.config_data = {}
        self.load_config()

    def load_config(self):
        if not os.path.exists(self.config_file):
            self.config_data = {"panels": []}
            self.save_config()
        else:
            with open(self.config_file, "r", encoding="utf-8") as f:
                self.config_data = json.load(f)

    def save_config(self):
        with open(self.config_file, "w", encoding="utf-8") as f:
            json.dump(self.config_data, f, indent=4, ensure_ascii=False)

    def add_panel_config(self, name, location):
        panel_config = {"name": name, "location": location, "created":
str(datetime.datetime.now())}
        self.config_data["panels"].append(panel_config)
        self.save_config()

    def get_panel_configs(self):
        return self.config_data.get("panels", [])

# Клас для моніторингу стану системи
class SystemMonitor:
    def __init__(self, db_manager, panel_config):
        self.db_manager = db_manager
        self.panel_config = panel_config

    def print_status(self):
        print("Звіт стану системи")
        print("====================")
        panels = self.db_manager.get_all_panels()
        print("Список панелей:")
        for panel in panels:
            panel_id, name, location, last_updated = panel
            print(f"ID: {panel_id}, Назва: {name}, Локація: {location}, Останнє
оновлення: {last_updated}")
        print("====================")
        print("Конфігурації панелей:")
        configs = self.panel_config.get_panel_configs()
        for config in configs:
```

```
            print(f"Назва: {config['name']}, Локація: {config['location']},
Створено: {config['created']}")
        print("====================")

# Функція для обробки команд користувача через консоль
def command_interface(db_manager, panel_config, system_monitor):
    while True:
        command = input("Введіть команду (status/add_panel/add_config/exit):
").strip()
        if command == "status":
            system_monitor.print_status()
        elif command == "add_panel":
            name = input("Введіть назву панелі: ").strip()
            location = input("Введіть локацію панелі: ").strip()
            panel_id = db_manager.add_panel(name, location)
            log_event(f"Додано нову панель: {name} у {location} з ID
{panel_id}")
        elif command == "add_config":
            name = input("Введіть назву панелі для конфігурації: ").strip()
            location = input("Введіть локацію панелі для конфігурації:
").strip()
            panel_config.add_panel_config(name, location)
            log_event(f"Додано нову конфігурацію панелі: {name} у {location}")
        elif command == "exit":
            log_event("Завершення роботи командного інтерфейсу.")
            break
        else:
            print("Невідома команда. Спробуйте ще раз.")

# Функція для запуску командного інтерфейсу в окремому потоці
def start_command_interface(db_manager, panel_config, system_monitor):
    cmd_thread = threading.Thread(target=command_interface, args=(db_manager,
panel_config, system_monitor))
    cmd_thread.daemon = True
    cmd_thread.start()

# Додаткові функції для системи
def extra_functionality_1():
    for i in range(10):
        log_event(f"Виконується додаткова логіка 1, ітерація {i}")
        time.sleep(0.5)

def extra_functionality_2():
    for i in range(5):
        log_event(f"Виконується додаткова логіка 2, ітерація {i}")
        time.sleep(1)

def extra_functionality_3():
    try:
        response = requests.get("https://api.github.com")
        if response.status_code == 200:
            log_event("Отримано дані з GitHub API.")
        else:
            log_event("Помилка при зверненні до GitHub API.")
    except Exception as e:
        log_event(f"Виняток при виконанні мережевого запиту: {e}")

def start_extra_functionalities():
    t1 = threading.Thread(target=extra_functionality_1)
    t2 = threading.Thread(target=extra_functionality_2)
    t3 = threading.Thread(target=extra_functionality_3)
    t1.daemon = True
    t2.daemon = True
    t3.daemon = True
    t1.start()
    t2.start()
    t3.start()
```

```
# Функція для запуску розширених модулів файлів
def start_extensions():
    # Запуск розширених можливостей з файлу
    ext1_thread = threading.Thread(target=extensions_1.run_all_extensions)
    ext1_thread.daemon = True
    ext1_thread.start()

    # Запуск розширених можливостей з файлу
    ext11_thread = threading.Thread(target=extensions_2.run_all_extensions_2)
    ext11_thread.daemon = True
    ext11_thread.start()

    # Запуск розширених можливостей з файлу
    ext21_thread = threading.Thread(target=extensions_3.run_extensions_3)
    ext21_thread.daemon = True
    ext21_thread.start()

# Головна функція для ініціалізації та запуску системи
def main():
    db_file = "advertising_panels.db"
    db_manager = DatabaseManager(db_file)
    existing_panels = db_manager.get_all_panels()
    if not existing_panels:
        db_manager.add_panel("Головний вхід", "Будівля A")
        db_manager.add_panel("Лобі", "Будівля B")
        db_manager.add_panel("Конференц-зал", "Будівля C")
    panels, threads = start_panels_system(db_manager)
    updater_thread = threading.Thread(target=advertisement_updater,
args=(db_manager,))
    updater_thread.daemon = True
    updater_thread.start()
    panel_config = PanelConfig()
    system_monitor = SystemMonitor(db_manager, panel_config)
    start_command_interface(db_manager, panel_config, system_monitor)
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        print("Зупинка системи.")
        db_manager.close()
```

Файл extensions_1.py

```
#!/usr/bin/env python3
# Модуль розширення функціоналу системи цифрових інформаційно-рекламних панелей

import time
import datetime
import threading
import json
import sqlite3
import random
import requests
import logging
import gettext
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from flask import Flask, render_template, request

# Налаштування системного логування
logging.basicConfig(level=logging.DEBUG, filename='extension_features.log',
filemode='a', format='%(asctime)s - %(levelname)s - %(message)s')

# Аутентифікація користувачів та управління ролями ---

class User:
    def __init__(self, username, password, role):
        # Ініціалізація користувача
        self.username = username
        self.password = password
        self.role = role

class AuthManager:
    def __init__(self):
        # Ініціалізація менеджера автентифікації
        self.users = {}
        self.logged_in_users = {}
        self.load_users()

    def load_users(self):
        # Завантаження користувачів (імітація бази даних)
        self.users = {
            "admin": User("admin", "admin123", "admin"),
            "manager": User("manager", "manager123", "manager"),
            "viewer": User("viewer", "viewer123", "viewer")
        }
        logging.info("Користувачі завантажені у систему автентифікації.")

    def authenticate(self, username, password):
        # Аутентифікація користувача
        user = self.users.get(username)
        if user and user.password == password:
            self.logged_in_users[username] = user
```

```
            logging.info(f"Користувач '{username}' автентифікований успішно.")
            return True
        logging.warning(f"Невдала спроба автентифікації для '{username}'.")
        return False

    def get_user_role(self, username):
        # Отримання ролі користувача
        user = self.users.get(username)
        if user:
            return user.role
        return None

    def logout(self, username):
        # Вихід користувача з системи
        if username in self.logged_in_users:
            del self.logged_in_users[username]
            logging.info(f"Користувач '{username}' вийшов із системи.")

# Розширений графічний інтерфейс користувача (GUI) ---

class AdvancedGUI:
    def __init__(self, auth_manager):
        # Ініціалізація розширеного GUI з менеджером автентифікації
        self.auth_manager = auth_manager
        self.root = Tk()
        self.root.title("Розширений GUI панелей")
        self.create_widgets()

    def create_widgets(self):
        # Створення основних віджетів для авторизації
        self.login_frame = Frame(self.root)
        self.login_frame.pack(pady=10)

        self.username_label = Label(self.login_frame, text="Ім'я користувача:")
        self.username_label.grid(row=0, column=0, padx=5, pady=5)
        self.username_entry = Entry(self.login_frame)
        self.username_entry.grid(row=0, column=1, padx=5, pady=5)

        self.password_label = Label(self.login_frame, text="Пароль:")
        self.password_label.grid(row=1, column=0, padx=5, pady=5)
        self.password_entry = Entry(self.login_frame, show="*")
        self.password_entry.grid(row=1, column=1, padx=5, pady=5)

        self.login_button = Button(self.login_frame, text="Увійти",
command=self.login)
        self.login_button.grid(row=2, column=0, columnspan=2, pady=10)

        self.info_label = Label(self.root, text="Не увійшли в систему",
fg="red")
        self.info_label.pack(pady=5)

        # Панель керування для подальшої роботи
        self.control_frame = Frame(self.root)
```

```
        self.control_frame.pack(pady=10)
        self.role_label = Label(self.control_frame, text="Роль: ")
        self.role_label.grid(row=0, column=0, padx=5, pady=5)
        self.dashboard_button = Button(self.control_frame, text="Перейти до
панелі", command=self.open_dashboard)
        self.dashboard_button.grid(row=0, column=1, padx=5, pady=5)

    def login(self):
        # Обробка події входу користувача
        username = self.username_entry.get()
        password = self.password_entry.get()
        if self.auth_manager.authenticate(username, password):
            role = self.auth_manager.get_user_role(username)
            self.info_label.config(text=f"Успішний вхід: {username} (Роль:
{role})", fg="green")
        else:
            self.info_label.config(text="Невірні дані", fg="red")

    def open_dashboard(self):
        # Відкриття панелі керування після входу
        dashboard = Toplevel(self.root)
        dashboard.title("Панель керування")
        msg = Label(dashboard, text="Вітаємо в панелі керування!")
        msg.pack(padx=20, pady=20)
        close_btn = Button(dashboard, text="Закрити", command=dashboard.destroy)
        close_btn.pack(pady=10)

    def run(self):
        # Запуск графічного інтерфейсу
        self.root.mainloop()

# --- Підтримка мобільних пристроїв ---

# Ініціалізація Flask додатку для мобільного інтерфейсу
app = Flask(__name__)

@app.route('/')
def mobile_home():
    # Головна сторінка мобільного інтерфейсу
    return "<h1>Мобільний інтерфейс цифрових рекламних
панелей</h1><p>Вітаємо!</p>"

@app.route('/login', methods=['GET', 'POST'])
def mobile_login():
    # Сторінка входу для мобільного інтерфейсу
    if request.method == 'POST':
        username = request.form.get("username")
        password = request.form.get("password")
        auth_manager = AuthManager()
        if auth_manager.authenticate(username, password):
            return f"<h1>Ласкаво просимо, {username}!</h1>"
        else:
            return "<h1>Невірні дані</h1>"
```

```
    return '''
        <form method="post">
            Ім'я користувача: <input type="text" name="username"><br>
            Пароль: <input type="password" name="password"><br>
            <input type="submit" value="Увійти">
        </form>
    '''

def run_mobile_interface():
    # Запуск мобільного інтерфейсу у окремому потоці
    app.run(port=5001)

# --- Розширена аналітика та звітність ---

class AnalyticsManager:
    def __init__(self, db_file="advertising_panels.db"):
        # Ініціалізація менеджера аналітики з підключенням до бази даних
        self.db_file = db_file

    def fetch_data(self):
        # Завантаження даних рекламних оголошень з бази даних
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM advertisements")
        data = cursor.fetchall()
        conn.close()
        logging.info("Дані для аналітики завантажено з бази даних.")
        return data

    def generate_report(self):
        # Генерація звіту на основі отриманих даних
        data = self.fetch_data()
        report = "Звіт рекламних оголошень\n"
        report += "========================\n"
        for row in data:
            report += f"ID: {row[0]}, Панель: {row[1]}, Зміст: {row[2]},
Тривалість: {row[3]}, Початок: {row[4]}, Кінець: {row[5]}, Активність:
{row[6]}\n"
        logging.info("Аналітичний звіт.")
        return report

# --- Інтеграція з API постачальників рекламного контенту ---

class AdContentProviderAPI:
    def __init__(self, api_url):
        # Ініціалізація API постачальника рекламного контенту
        self.api_url = api_url

    def fetch_advertisements(self):
        # Отримання рекламного контенту через API
        try:
            response = requests.get(self.api_url)
            if response.status_code == 200:
```

```
                ads = response.json()
                logging.info("Дані з API постачальника рекламного контенту
отримано.")
                return ads
            else:
                logging.error("Помилка отримання даних з API: статус код не
200.")
                return []
        except Exception as e:
            logging.error(f"Виняток при запиті до API: {e}")
            return []

# --- Автоматичне оновлення рекламного контенту ---

class AutoContentUpdater:
    def __init__(self, ad_provider_api, update_interval=60):
        # Ініціалізація автоматичного оновлювача контенту
        self.ad_provider_api = ad_provider_api
        self.update_interval = update_interval
        self.running = False

    def update_content(self):
        # Автоматичне оновлення рекламного контенту за встановленим інтервалом
        while self.running:
            ads = self.ad_provider_api.fetch_advertisements()
            logging.info(f"Оновлення контенту: отримано {len(ads)} оголошень.")
            # Тут можна додати логіку для оновлення бази даних
            time.sleep(self.update_interval)

    def start(self):
        # Запуск процесу оновлення контенту в окремому потоці
        self.running = True
        t = threading.Thread(target=self.update_content)
        t.daemon = True
        t.start()

    def stop(self):
        # Зупинка процесу оновлення контенту
        self.running = False

# --- Налаштування розкладу відображення реклами ---

class AdScheduler:
    def __init__(self):
        # Ініціалізація планувальника рекламних оголошень
        self.scheduled_ads = []

    def schedule_ad(self, ad_id, display_time):
        # Планування показу оголошення у визначений час
        self.scheduled_ads.append((ad_id, display_time))
        logging.info(f"Заплановано показ оголошення {ad_id} на {display_time}.")

    def get_schedule(self):
```

```
        # Отримання списку запланованих оголошень
        return self.scheduled_ads

    def run_scheduler(self):
        # Безкінечний цикл для відстеження часу показу оголошень
        while True:
            now = datetime.datetime.now()
            for ad, display_time in list(self.scheduled_ads):
                if now >= display_time:
                    logging.info(f"Показ запланованого оголошення {ad} о
{display_time}.")
                    print(f"Відображення оголошення {ad} згідно розкладу.")
                    self.scheduled_ads.remove((ad, display_time))
            time.sleep(1)

# --- Підтримка мультимовності ---

class MultilingualSupport:
    def __init__(self, locale_dir='locales', domain='messages'):
        # Ініціалізація підтримки мультимовності
        self.locale_dir = locale_dir
        self.domain = domain
        self.translators = {}

    def load_language(self, language_code):
        # Завантаження перекладів для заданої мови
        try:
            translator = gettext.translation(self.domain,
localedir=self.locale_dir, languages=[language_code])
            translator.install()
            self.translators[language_code] = translator
            logging.info(f"Мову '{language_code}' завантажено.")
        except Exception as e:
            logging.error(f"Помилка завантаження мови '{language_code}': {e}")

    def translate(self, text, language_code):
        # Переклад тексту на обрану мову
        translator = self.translators.get(language_code)
        if translator:
            return translator.gettext(text)
        return text

# ---Інтеграція з соціальними мережами ---

class SocialMediaIntegration:
    def __init__(self, api_key):
        # Ініціалізація інтеграції з соціальними мережами
        self.api_key = api_key

    def post_update(self, message):
        # Симуляція публікації оновлення в соціальній мережі
        logging.info(f"Публікація в соціальній мережі: {message}")
        # Реальна інтеграція може використовувати API соцмережі
```

```
        print(f"Соціальна мережа: {message}")

# --- Система управління рекламними кампаніями ---

class Campaign:
    def __init__(self, campaign_id, name, start_date, end_date, ads):
        # Ініціалізація рекламної кампанії
        self.campaign_id = campaign_id
        self.name = name
        self.start_date = start_date
        self.end_date = end_date
        self.ads = ads

class CampaignManager:
    def __init__(self):
        # Ініціалізація менеджера рекламних кампаній
        self.campaigns = {}
        self.next_campaign_id = 1

    def create_campaign(self, name, start_date, end_date, ads):
        # Створення нової рекламної кампанії
        campaign_id = self.next_campaign_id
        self.next_campaign_id += 1
        campaign = Campaign(campaign_id, name, start_date, end_date, ads)
        self.campaigns[campaign_id] = campaign
        logging.info(f"Створено кампанію {campaign_id}: {name}")
        return campaign_id

    def update_campaign(self, campaign_id, name=None, start_date=None,
end_date=None, ads=None):
        # Оновлення даних кампанії
        campaign = self.campaigns.get(campaign_id)
        if campaign:
            if name:
                campaign.name = name
            if start_date:
                campaign.start_date = start_date
            if end_date:
                campaign.end_date = end_date
            if ads is not None:
                campaign.ads = ads
            logging.info(f"Кампанію {campaign_id} оновлено.")
            return True
        logging.warning(f"Кампанію {campaign_id} не знайдено.")
        return False

    def delete_campaign(self, campaign_id):
        # Видалення рекламної кампанії
        if campaign_id in self.campaigns:
            del self.campaigns[campaign_id]
            logging.info(f"Кампанію {campaign_id} видалено.")
            return True
        logging.warning(f"Кампанію {campaign_id} не знайдено.")
```

```
        return False

    def get_campaign(self, campaign_id):
        # Отримання інформації про кампанію
        return self.campaigns.get(campaign_id)

    def list_campaigns(self):
        # Повернення списку всіх кампаній
        return list(self.campaigns.values())

# --- Головна функція для запуску всіх розширених можливостей ---

def run_all_extensions():
    # Ініціалізація менеджера автентифікації
    auth_manager = AuthManager()

    # Запуск розширеного GUI в окремому потоці
    gui_thread = threading.Thread(target=lambda:
AdvancedGUI(auth_manager).run())
    gui_thread.daemon = True
    gui_thread.start()

    # Запуск мобільного інтерфейсу в окремому потоці
    mobile_thread = threading.Thread(target=run_mobile_interface)
    mobile_thread.daemon = True
    mobile_thread.start()

    # Ініціалізація аналітики та звіту
    analytics_manager = AnalyticsManager()
    report = analytics_manager.generate_report()
    print(report)

    # Інтеграція з API постачальника рекламного контенту
    ad_api = AdContentProviderAPI("https://api.example.com/ads")
    ad_api.fetch_advertisements()

    # Запуск автоматичного оновлення рекламного контенту
    auto_updater = AutoContentUpdater(ad_api, update_interval=30)
    auto_updater.start()

    # Запуск планувальника рекламних оголошень
    scheduler = AdScheduler()
    scheduler.schedule_ad(1, datetime.datetime.now() +
datetime.timedelta(seconds=10))
    scheduler_thread = threading.Thread(target=scheduler.run_scheduler)
    scheduler_thread.daemon = True
    scheduler_thread.start()

    # Підтримка мультимовності
    multilingual = MultilingualSupport()
    multilingual.load_language("uk")
    translated_text = multilingual.translate("Hello, world!", "uk")
    print(f"Переклад: {translated_text}")
```

```
    # Інтеграція з соціальними мережами
    social_integration = SocialMediaIntegration("dummy_api_key")
    social_integration.post_update("Нова рекламна кампанія запущена!")

    # Управління рекламними кампаніями
    campaign_manager = CampaignManager()
    campaign_id = campaign_manager.create_campaign("Кампанія 1",
datetime.datetime.now(), datetime.datetime.now() + datetime.timedelta(days=7),
[1, 2, 3])
    campaigns = campaign_manager.list_campaigns()
    for campaign in campaigns:
        print(f"Кампанія {campaign.campaign_id}: {campaign.name}")
```

Файл extensions_2.py

```
#!/usr/bin/env python3
# Файл розширень функціоналу
# Цей файл містить реалізації інтеграції з CRM-системами, geo-таргетинг,
підтримку різних форматів реклами,
# реалізацію push-повідомлень, інтеграцію з платіжними системами, систему
резервного копіювання даних,
# розширення API для сторонніх розробників, підтримку IoT-пристроїв, інтеграцію
з хмарними сервісами,
# а також автоматичне виявлення та усунення помилок.

import time
import datetime
import threading
import logging
import sqlite3
import json
import random
import requests
import os
import math

# Налаштування логування для цього модуля
logging.basicConfig(level=logging.DEBUG, filename='extension_11_20.log',
filemode='a', format='%(asctime)s - %(levelname)s - %(message)s')

# ----------------------------------------------------------------------
# Інтеграція з CRM-системами
# ----------------------------------------------------------------------
class CRMIntegration:
# Ініціалізація інтеграції з CRM-системою
    def __init__(self, crm_endpoint, api_key):
        # Збереження кінцевої точки та ключа API
        self.crm_endpoint = crm_endpoint
        self.api_key = api_key
        # Ініціалізація внутрішнього кешу даних
        self.leads = {}
        logging.info("CRMIntegration ініціалізовано з кінцевою точкою: " +
crm_endpoint)

# Надсилання даних потенційного клієнта в CRM
    def send_lead(self, lead_data):
        # Формування запиту до CRM
        logging.debug("Надсилання даних потенційного клієнта: " +
json.dumps(lead_data))
        # Симуляція мережевого запиту з затримкою
        time.sleep(random.uniform(0.2, 0.5))
        # Імітація успішної відповіді
        response = {"status": "success", "lead_id": random.randint(1000, 9999)}
        # Збереження отриманого ідентифікатора
        self.leads[lead_data.get("email", "unknown")] = response["lead_id"]
```

```
        logging.info("Потенційного клієнта додано з ID: " +
str(response["lead_id"]))
        return response

# Отримання інформації про клієнта за його ідентифікатором
    def fetch_customer_info(self, customer_id):
        # Симуляція отримання даних з CRM
        logging.debug("Запит інформації про клієнта з ID: " + str(customer_id))
        time.sleep(random.uniform(0.1, 0.3))
        # Повернення імітованих даних клієнта
        customer_info = {"customer_id": customer_id, "name": "Ім'я_Клієнта",
"email": "client@example.com"}
        logging.info("Отримано дані клієнта: " + json.dumps(customer_info))
        return customer_info

# Симуляція роботи CRM інтеграції з різними запитами
    def simulate_crm_interaction(self):
        # Створення даних потенційного клієнта
        lead_data = {"name": "Олександр", "email": "oleksandr@example.com",
"phone": "+380501234567"}
        # Надсилання даних до CRM
        response = self.send_lead(lead_data)
        # Отримання інформації про клієнта, використовуючи отриманий ID
        customer_info = self.fetch_customer_info(response["lead_id"])
        # Логування завершення симуляції
        logging.info("CRM інтеграція завершена. Дані клієнта: " +
json.dumps(customer_info))
        print("CRM інтеграція: Дані клієнта отримано.")

# ----------------------------------------------------------------------
# Реалізація geo-таргетингу
# ----------------------------------------------------------------------
class GeoTargeting:
# Ініціалізація geo-таргетингу
    def __init__(self):
        # Імітація бази даних регіонів за IP адресами
        self.ip_region_mapping = {
            "192.168.1.1": "Київ",
            "192.168.1.2": "Львів",
            "192.168.1.3": "Одеса",
            "10.0.0.1": "Харків"
        }
        logging.info("GeoTargeting ініціалізовано з базою даних IP-регіонів.")

# Отримання регіону за IP адресою
    def get_region_by_ip(self, ip_address):
        # Пошук регіону в імітованій базі даних
        region = self.ip_region_mapping.get(ip_address, "Невідомий регіон")
        logging.debug("IP адреса " + ip_address + " відповідає регіону: " +
region)
        return region

# Обчислення відстані між двома координатами за допомогою формули Хаверсайна
```

```
    def calculate_distance(self, coord1, coord2):
        # Розпакування координат
        lat1, lon1 = coord1
        lat2, lon2 = coord2
        # Перетворення градусів у радіани
        lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2])
        # Розрахунок різниці
        dlat = lat2 - lat1
        dlon = lon2 - lon1
        # Формула Хаверсайна
        a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) *
math.sin(dlon/2)**2
        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
        # Радіус Землі (кілометри)
        distance = 6371 * c
        logging.debug("Обчислено відстань: " + str(distance) + " км")
        return distance

# Симуляція функціоналу geo-таргетингу
    def simulate_geo_targeting(self):
        # Імітація отримання IP адреси
        test_ip = random.choice(list(self.ip_region_mapping.keys()))
        region = self.get_region_by_ip(test_ip)
        # Обчислення відстані між двома випадковими координатами
        distance = self.calculate_distance((50.4501, 30.5234), (49.8397,
24.0297))
        logging.info("Geo-таргетинг: IP " + test_ip + " визначено як " + region
+ "; Відстань обчислено: " + str(distance) + " км")
        print("Geo-таргетинг виконано. Регіон: " + region)

# ---------------------------------------------------------------------
# Підтримка різних форматів реклами (відео, анімація, інтерактив)
# ---------------------------------------------------------------------
class AdFormatsSupport:
# Ініціалізація підтримки форматів реклами
    def __init__(self):
        # Список підтримуваних форматів реклами
        self.supported_formats = ["video", "animation", "interactive", "static"]
        logging.info("AdFormatsSupport ініціалізовано з форматами: " + ",
".join(self.supported_formats))

# Рендеринг оголошення в залежності від формату
    def render_ad(self, ad_content, format_type):
        # Перевірка чи формат підтримується
        if format_type not in self.supported_formats:
            logging.warning("Непідтримуваний формат: " + format_type)
            return "Формат не підтримується."
        # Симуляція рендерингу оголошення
        logging.debug("Рендеринг оголошення в форматі: " + format_type)
        time.sleep(random.uniform(0.1, 0.3))
        rendered = f"[{format_type.upper()}] {ad_content}"
        logging.info("Оголошення відрендерено: " + rendered)
        return rendered
```

```
# Симуляція роботи з різними форматами реклами
    def simulate_ad_formats(self):
        # Імітація контенту оголошення
        ad_content = "Нова пропозиція: знижка 20%!"
        for fmt in self.supported_formats:
            rendered_ad = self.render_ad(ad_content, fmt)
            print("Результат рендерингу (" + fmt + "): " + rendered_ad)
            time.sleep(0.2)
        logging.info("Симуляція підтримки форматів реклами завершена.")

# ----------------------------------------------------------------------
# Реалізація push-повідомлень
# ----------------------------------------------------------------------
class PushNotificationSystem:
# Ініціалізація системи push-повідомлень
    def __init__(self):
        # Список запланованих push-повідомлень
        self.notifications = []
        logging.info("PushNotificationSystem ініціалізовано.")

# Надсилання push-повідомлення користувачу
    def send_notification(self, message, user_id):
        # Симуляція відправлення повідомлення з затримкою
        logging.debug("Відправка push-повідомлення користувачу " + str(user_id)
+ ": " + message)
        time.sleep(random.uniform(0.1, 0.3))
        logging.info("Push-повідомлення надіслано користувачу " + str(user_id))
        print("Push-повідомлення надіслано користувачу " + str(user_id) + ": " +
message)

# Запланування push-повідомлення через затримку
    def schedule_notification(self, message, user_id, delay):
        # Розрахунок часу відправлення повідомлення
        scheduled_time = datetime.datetime.now() +
datetime.timedelta(seconds=delay)
        self.notifications.append((scheduled_time, message, user_id))
        logging.info("Заплановано push-повідомлення для користувача " +
str(user_id) + " на " + str(scheduled_time))

# Функція для періодичної перевірки запланованих повідомлень
    def process_scheduled_notifications(self):
        while True:
            now = datetime.datetime.now()
            for notification in list(self.notifications):
                scheduled_time, message, user_id = notification
                if now >= scheduled_time:
                    self.send_notification(message, user_id)
                    self.notifications.remove(notification)
            time.sleep(0.5)

# Симуляція роботи системи push-повідомлень
    def simulate_notifications(self):
```

```
        # Запланування декількох повідомлень
        self.schedule_notification("Акційна пропозиція!", 101, 2)
        self.schedule_notification("Нове оновлення системи.", 102, 4)
        self.schedule_notification("Перевірте свої налаштування.", 103, 6)
        # Запуск окремого потоку для обробки повідомлень
        notif_thread =
threading.Thread(target=self.process_scheduled_notifications)
        notif_thread.daemon = True
        notif_thread.start()
        # Симуляція надсилання миттєвого повідомлення
        self.send_notification("Ласкаво просимо!", 100)
        time.sleep(8)
        logging.info("Симуляція push-повідомлень завершена.")

# ----------------------------------------------------------------------
# Інтеграція з платіжними системами
# ----------------------------------------------------------------------
class PaymentIntegration:
# Ініціалізація інтеграції з платіжною системою
    def __init__(self, payment_api_url, api_key):
        # Збереження URL API та ключа доступу
        self.payment_api_url = payment_api_url
        self.api_key = api_key
        logging.info("PaymentIntegration ініціалізовано з API: " +
payment_api_url)

# Обробка платежу певної суми для користувача
    def process_payment(self, amount, user_id):
        # Логування початку процесу платежу
        logging.debug("Обробка платежу на суму " + str(amount) + " для
користувача " + str(user_id))
        time.sleep(random.uniform(0.2, 0.5))
        # Імітація відповіді з транзакційним ID
        transaction_id = random.randint(10000, 99999)
        logging.info("Платіж оброблено, транзакційний ID: " +
str(transaction_id))
        return transaction_id

# Імітація повернення платежу
    def refund_payment(self, transaction_id):
        logging.debug("Початок процесу повернення для транзакції " +
str(transaction_id))
        time.sleep(random.uniform(0.1, 0.3))
        logging.info("Повернення здійснено для транзакції " +
str(transaction_id))
        return True

# Перевірка статусу платежу за транзакційним ID
    def check_payment_status(self, transaction_id):
        logging.debug("Перевірка статусу транзакції " + str(transaction_id))
        time.sleep(random.uniform(0.1, 0.2))
        status = random.choice(["completed", "pending", "failed"])
        logging.info("Статус транзакції " + str(transaction_id) + ": " + status)
```

```
        return status

# Симуляція повного циклу обробки платежу
    def simulate_payments(self):
        transaction_id = self.process_payment(150.75, 201)
        status = self.check_payment_status(transaction_id)
        if status != "completed":
            self.refund_payment(transaction_id)
        logging.info("Симуляція платіжної інтеграції завершена.")
        print("Платіжна інтеграція: процес платежу симульовано.")

# ---------------------------------------------------------------------
# Система резервного копіювання даних
# ---------------------------------------------------------------------
class DataBackupSystem:
# Ініціалізація системи резервного копіювання
    def __init__(self, backup_directory="backups"):
        self.backup_directory = backup_directory
        # Перевірка наявності директорії резервних копій
        if not os.path.exists(self.backup_directory):
            os.makedirs(self.backup_directory)
            logging.info("Створено директорію для резервного копіювання: " +
self.backup_directory)
        else:
            logging.info("Директорія для резервного копіювання існує: " +
self.backup_directory)

# Резервне копіювання даних у файл
    def backup_data(self, data, backup_name):
        backup_path = os.path.join(self.backup_directory, backup_name + ".json")
        with open(backup_path, "w", encoding="utf-8") as f:
            json.dump(data, f, ensure_ascii=False, indent=4)
        logging.info("Дані збережено в резервній копії: " + backup_path)
        return backup_path

# Відновлення даних з резервної копії
    def restore_data(self, backup_name):
        backup_path = os.path.join(self.backup_directory, backup_name + ".json")
        if os.path.exists(backup_path):
            with open(backup_path, "r", encoding="utf-8") as f:
                data = json.load(f)
            logging.info("Дані відновлено з резервної копії: " + backup_path)
            return data
        else:
            logging.error("Резервну копію не знайдено: " + backup_path)
            return None

# Симуляція процесу резервного копіювання та відновлення
    def simulate_backup(self):
        sample_data = {"ads": [1, 2, 3, 4], "panels": ["A", "B", "C"]}
        backup_name = "backup_" +
datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_file = self.backup_data(sample_data, backup_name)
```

```
        time.sleep(1)
        restored_data = self.restore_data(backup_name)
        logging.info("Резервне копіювання завершено. Відновлені дані: " +
json.dumps(restored_data))
        print("Резервне копіювання: дані збережено та відновлено.")

# ----------------------------------------------------------------------
# Розширення API для сторонніх розробників
# ----------------------------------------------------------------------
class ThirdPartyAPI:
# Ініціалізація розширеного API для сторонніх розробників
    def __init__(self):
        self.endpoints = {}
        logging.info("ThirdPartyAPI ініціалізовано.")

# Реєстрація нового API-ендпоінту
    def register_endpoint(self, endpoint_name, function):
        self.endpoints[endpoint_name] = function
        logging.info("Зареєстровано ендпоінт: " + endpoint_name)

# Виклик зареєстрованого ендпоінту з параметрами
    def call_endpoint(self, endpoint_name, *args, **kwargs):
        if endpoint_name in self.endpoints:
            logging.debug("Виклик ендпоінту: " + endpoint_name)
            result = self.endpoints[endpoint_name](*args, **kwargs)
            logging.info("Ендпоінт " + endpoint_name + " повернув результат: " +
str(result))
            return result
        else:
            logging.error("Ендпоінт не знайдено: " + endpoint_name)
            return None

# Симуляція роботи API для сторонніх розробників
    def simulate_api_calls(self):
        # Реєстрація декількох тестових ендпоінтів
        self.register_endpoint("get_ad_stats", lambda: {"views":
random.randint(100, 1000), "clicks": random.randint(10, 100)})
        self.register_endpoint("update_ad_content", lambda ad_id, content:
f"Оновлено оголошення {ad_id} з контентом: {content}")
        # Виклик ендпоінтів
        stats = self.call_endpoint("get_ad_stats")
        update_result = self.call_endpoint("update_ad_content", 1, "Новий
контент оголошення")
        logging.info("API симуляція завершена з результатами: " +
json.dumps({"stats": stats, "update": update_result}))
        print("API для сторонніх розробників: запити виконано.")

# ----------------------------------------------------------------------
# Підтримка IoT-пристроїв
# ----------------------------------------------------------------------
class IoTDeviceSupport:
# Ініціалізація підтримки IoT-пристроїв
    def __init__(self):
```

```
        self.devices = {}
        logging.info("IoTDeviceSupport ініціалізовано.")

# Реєстрація нового IoT-пристрою
    def register_device(self, device_id, device_info):
        self.devices[device_id] = {"info": device_info, "status": "offline"}
        logging.info("Зареєстровано IoT-пристрій з ID: " + str(device_id))

# Оновлення статусу пристрою
    def update_device_status(self, device_id, status):
        if device_id in self.devices:
            self.devices[device_id]["status"] = status
            logging.info("Оновлено статус пристрою " + str(device_id) + " на: "
+ status)
            return True
        else:
            logging.error("Пристрій не знайдено для оновлення статусу: " +
str(device_id))
            return False

# Надсилання команди IoT-пристрою
    def send_command_to_device(self, device_id, command):
        if device_id in self.devices:
            logging.debug("Надсилання команди '" + command + "' до пристрою " +
str(device_id))
            time.sleep(random.uniform(0.1, 0.3))
            logging.info("Команда виконана для пристрою " + str(device_id))
            return "Команда виконана"
        else:
            logging.error("Неможливо надіслати команду, пристрій не знайдено: "
+ str(device_id))
            return "Пристрій не знайдено"

# Симуляція роботи з IoT-пристроями
    def simulate_iot_devices(self):
        # Реєстрація кількох пристроїв
        self.register_device("iot_001", {"model": "SensorX", "location":
"Будівля A"})
        self.register_device("iot_002", {"model": "ActuatorY", "location":
"Будівля B"})
        # Оновлення статусу та надсилання команд
        self.update_device_status("iot_001", "online")
        response = self.send_command_to_device("iot_002", "Перезапустити")
        logging.info("IoT симуляція завершена з відповіддю: " + response)
        print("IoT-пристрої: симуляція виконана.")

# ----------------------------------------------------------------------
# Інтеграція з хмарними сервісами
# ----------------------------------------------------------------------
class CloudIntegration:
# Ініціалізація інтеграції з хмарними сервісами
    def __init__(self, cloud_provider, api_key):
        self.cloud_provider = cloud_provider
```

```
        self.api_key = api_key
        logging.info("CloudIntegration ініціалізовано для провайдера: " +
cloud_provider)

# Завантаження файлу в хмарне сховище
    def upload_to_cloud(self, file_path):
        logging.debug("Завантаження файлу до хмарного сховища: " + file_path)
        time.sleep(random.uniform(0.2, 0.5))
        cloud_file_id = "cloud_" + str(random.randint(1000, 9999))
        logging.info("Файл завантажено до хмари з ID: " + cloud_file_id)
        return cloud_file_id

# Завантаження файлу з хмарного сховища
    def download_from_cloud(self, file_identifier):
        logging.debug("Завантаження файлу з хмари з ID: " + file_identifier)
        time.sleep(random.uniform(0.1, 0.3))
        file_content = "Зміст файлу з хмарного сховища"
        logging.info("Файл завантажено з хмари, ID: " + file_identifier)
        return file_content

# Синхронізація локальних даних з хмарою
    def sync_data(self, local_data):
        logging.debug("Синхронізація даних з хмарою.")
        time.sleep(random.uniform(0.2, 0.4))
        synced = True
        logging.info("Дані успішно синхронізовано з хмарою.")
        return synced

# Симуляція операцій з хмарними сервісами
    def simulate_cloud_operations(self):
        # Створення тимчасового файлу для завантаження
        temp_file = "temp_data.txt"
        with open(temp_file, "w", encoding="utf-8") as f:
            f.write("Деякі локальні дані для синхронізації з хмарою.")
        # Завантаження файлу в хмару
        cloud_id = self.upload_to_cloud(temp_file)
        # Завантаження файлу з хмари
        downloaded_content = self.download_from_cloud(cloud_id)
        # Синхронізація даних
        sync_status = self.sync_data({"file_id": cloud_id, "content":
downloaded_content})
        logging.info("Симуляція хмарних операцій завершена. Статус
синхронізації: " + str(sync_status))
        print("Інтеграція з хмарними сервісами: операції виконано.")
        # Видалення тимчасового файлу
        os.remove(temp_file)

# ----------------------------------------------------------------------
# Автоматичне виявлення та усунення помилок
# ----------------------------------------------------------------------
class ErrorDetectionAndResolution:
# Ініціалізація системи виявлення помилок
    def __init__(self):
```

```
        self.error_log = []
        logging.info("ErrorDetectionAndResolution ініціалізовано.")

# Моніторинг системи для виявлення помилок
    def monitor_system(self):
        # Симуляція постійного моніторингу з випадковим генеруванням помилок
        logging.debug("Початок моніторингу системи.")
        for _ in range(5):
            time.sleep(random.uniform(0.1, 0.3))
            if random.choice([True, False]):
                error = "Помилка: випадковий збій компонента " +
str(random.randint(1, 10))
                self.error_log.append(error)
                logging.error("Виявлено помилку: " + error)
        logging.debug("Моніторинг системи завершено.")

# Виявлення помилок у системі
    def detect_errors(self):
        self.monitor_system()
        if self.error_log:
            logging.info("Виявлено " + str(len(self.error_log)) + " помилок.")
            return self.error_log
        else:
            logging.info("Помилок не виявлено.")
            return []

# Спроба автоматичного усунення помилок
    def attempt_resolution(self, error):
        logging.debug("Спроба усунення помилки: " + error)
        time.sleep(random.uniform(0.2, 0.4))
        resolution_success = random.choice([True, False])
        if resolution_success:
            logging.info("Помилку усунуто: " + error)
            return True
        else:
            logging.warning("Не вдалося усунути помилку: " + error)
            return False

# Симуляція процесу виявлення та усунення помилок
    def simulate_error_handling(self):
        errors = self.detect_errors()
        for error in errors:
            result = self.attempt_resolution(error)
            if result:
                print("Усунено: " + error)
            else:
                print("Не вдалося усунути: " + error)
        logging.info("Симуляція виявлення та усунення помилок завершена.")

# ----------------------------------------------------------------------
# Головна функція для запуску розширених функцій
# ----------------------------------------------------------------------
def run_all_extensions_11_20():
```

```
    # Симуляція CRM інтеграції
    crm = CRMIntegration("https://crm.example.com/api", "crm_api_key_123")
    crm.simulate_crm_interaction()

    # Симуляція geo-таргетингу
    geo = GeoTargeting()
    geo.simulate_geo_targeting()

    # Симуляція підтримки форматів реклами
    ad_formats = AdFormatsSupport()
    ad_formats.simulate_ad_formats()

    # Симуляція роботи push-повідомлень
    push_system = PushNotificationSystem()
    push_system.simulate_notifications()

    # Симуляція інтеграції з платіжними системами
    payment = PaymentIntegration("https://payments.example.com/api",
"payment_api_key_456")
    payment.simulate_payments()

    # Симуляція системи резервного копіювання даних
    backup_system = DataBackupSystem()
    backup_system.simulate_backup()

    # Симуляція розширення API для сторонніх розробників
    third_party_api = ThirdPartyAPI()
    third_party_api.simulate_api_calls()

    # Симуляція підтримки IoT-пристроїв
    iot_support = IoTDeviceSupport()
    iot_support.simulate_iot_devices()

    # Симуляція інтеграції з хмарними сервісами
    cloud_integration = CloudIntegration("ExampleCloud", "cloud_api_key_789")
    cloud_integration.simulate_cloud_operations()

    # Симуляція автоматичного виявлення та усунення помилок
    error_system = ErrorDetectionAndResolution()
    error_system.simulate_error_handling()

    # Затримка для завершення всіх процесів симуляції
    time.sleep(5)
```

Файл extensions_3.py

```
#!/usr/bin/env python3
# Розширення функціоналу системи цифрових інформаційно-рекламних панелей
# Використання машинного навчання для оптимізації розкладу реклами
# Розширене логування та моніторинг системи
# Підтримка розподіленої архітектури
# Інтеграція з системами сповіщень та алертів

import time
import datetime
import threading
import logging
import random
import json
import os
import math
import queue

# Спроба імпорту бібліотек для машинного навчання
try:
    import numpy as np
    from sklearn.linear_model import LinearRegression
except ImportError:
    np = None
    LinearRegression = None

# ================================================================================
# Використання машинного навчання для оптимізації розкладу реклами
# ================================================================================

class AdScheduleOptimizer:
    # Клас для оптимізації розкладу рекламних оголошень за допомогою машинного
    # навчання
    def __init__(self):
        # Ініціалізація оптимізатора розкладу
        self.model = None
        self.trained = False
        self.historical_data = []
        self.features = []
        self.targets = []
        logging.basicConfig(level=logging.DEBUG, filename='ml_optimizer.log',
filemode='a', format='%(asctime)s - %(levelname)s - %(message)s')
        logging.info("AdScheduleOptimizer ініціалізовано.")

    def load_historical_data(self):
       # Завантаження імітованих даних для тренування моделі
       # Дані містять інформацію про годину доби та ефективність оголошень
       # (наприклад, кількість кліків)
        logging.info("Завантаження історичних даних для оптимізації розкладу
реклами.")
        self.historical_data = []
        for hour in range(24):
```

```
            # Імітація даних: випадкова ефективність оголошення для кожної години
            clicks = random.randint(50, 500)
            self.historical_data.append((hour, clicks))
            logging.debug("Історичні дані завантажено: " +
str(self.historical_data))

    def prepare_data(self):
        # Підготовка даних для тренування моделі
        self.features = []
        self.targets = []
        for record in self.historical_data:
            hour, clicks = record
            self.features.append([hour])
            self.targets.append(clicks)
        logging.info("Дані підготовлено для тренування моделі.")
        logging.debug("Фічі: " + str(self.features))
        logging.debug("Цілі: " + str(self.targets))

    def train_model(self):
        # Тренування моделі для прогнозування ефективності оголошень
        if LinearRegression is None:
            logging.error("Бібліотека sklearn не доступна. Модель не може бути
натренована.")
            print("Модель не може бути натренована через відсутність sklearn.")
            return
        self.load_historical_data()
        self.prepare_data()
        self.model = LinearRegression()
        self.model.fit(self.features, self.targets)
        self.trained = True
        logging.info("Модель тренування завершено. Коєфіцієнти: " +
str(self.model.coef_) + ", Перетин: " + str(self.model.intercept_))

    def predict_optimal_schedule(self):
        # Прогнозування оптимального часу для показу реклами
        if not self.trained:
            logging.warning("Модель не тренована. Запуск тренування.")
            self.train_model()
        optimal_hour = None
        max_predicted = -1
        # Прогнозування для кожної години доби
        for hour in range(24):
            predicted_clicks = self.model.predict([[hour]])[0]
            logging.debug("Прогноз для години " + str(hour) + ": " +
str(predicted_clicks))
            if predicted_clicks > max_predicted:
                max_predicted = predicted_clicks
                optimal_hour = hour
        logging.info("Оптимальна година для показу реклами: " +
str(optimal_hour) + " з прогнозованими кліками: " + str(max_predicted))
        return optimal_hour, max_predicted

    def simulate_optimization(self):
```

```
        # Симуляція процесу оптимізації розкладу реклами
        print("Початок оптимізації розкладу реклами за допомогою машинного
навчання...")
        self.train_model()
        optimal_hour, predicted = self.predict_optimal_schedule()
        print(f"Оптимальний час для показу реклами: {optimal_hour}:00,
прогнозовані кліки: {predicted:.2f}")
        time.sleep(1)

# ========================================================================
# Розширене логування та моніторинг системи
# ========================================================================

class AdvancedLogger:
    # Клас для розширеного логування та моніторингу системи
    def __init__(self, log_file="advanced_system.log"):
        # Ініціалізація розширеного логера з кількома обробниками
        self.log_file = log_file
        self.logger = logging.getLogger("AdvancedLogger")
        self.logger.setLevel(logging.DEBUG)
        self.setup_handlers()
        logging.info("AdvancedLogger ініціалізовано.")

    def setup_handlers(self):
        # Налаштування обробників для логування у файл та консоль
        file_handler = logging.FileHandler(self.log_file, mode='a',
encoding='utf-8')
        file_handler.setLevel(logging.DEBUG)
        console_handler = logging.StreamHandler()
        console_handler.setLevel(logging.INFO)
        formatter = logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s')
        file_handler.setFormatter(formatter)
        console_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)
        self.logger.addHandler(console_handler)
        self.logger.debug("Обробники логування налаштовано.")

    def log_event(self, message, level="info"):
        # Метод для запису подій з різними рівнями важливості
        if level == "info":
            self.logger.info(message)
        elif level == "debug":
            self.logger.debug(message)
        elif level == "warning":
            self.logger.warning(message)
        elif level == "error":
            self.logger.error(message)
        elif level == "critical":
            self.logger.critical(message)

    def monitor_system_metrics(self):
        # Симуляція моніторингу системних метрик
```

```
        cpu_usage = random.uniform(10.0, 90.0)
        memory_usage = random.uniform(1000.0, 8000.0)
        metrics = {
            "cpu_usage": cpu_usage,
            "memory_usage": memory_usage,
            "timestamp": datetime.datetime.now().isoformat()
        }
        self.logger.info("Системні метрики: " + json.dumps(metrics))
        return metrics

    def simulate_monitoring(self):
        # Симуляція періодичного моніторингу системи
        for _ in range(5):
            self.monitor_system_metrics()
            time.sleep(1)

# ================================================================================
# Підтримка розподіленої архітектури
# ================================================================================

class Node:
    # Клас для імітації окремого вузла в розподіленій системі
    def __init__(self, node_id):
        self.node_id = node_id
        self.task_queue = queue.Queue()
        self.active = True
        self.thread = threading.Thread(target=self.run)
        self.thread.daemon = True
        self.thread.start()
        logging.info("Вузол " + str(self.node_id) + " ініціалізовано.")

    def add_task(self, task, *args, **kwargs):
        # Додавання завдання до черги вузла
        self.task_queue.put((task, args, kwargs))
        logging.debug("Завдання додано до вузла " + str(self.node_id))

    def run(self):
        # Основний цикл виконання завдань у вузлі
        while self.active:
            try:
                task, args, kwargs = self.task_queue.get(timeout=1)
                logging.debug("Вузол " + str(self.node_id) + " виконує завдання
" + str(task.__name__))
                task(*args, **kwargs)
                self.task_queue.task_done()
            except queue.Empty:
                continue

    def shutdown(self):
        # Зупинка вузла
        self.active = False
        logging.info("Вузол " + str(self.node_id) + " зупинено.")
```

```
class DistributedSystem:
    # Клас для управління розподіленою системою
    def __init__(self, num_nodes=3):
        self.nodes = []
        self.num_nodes = num_nodes
        self.setup_nodes()
        logging.info("DistributedSystem ініціалізовано з " + str(self.num_nodes)
+ " вузлами.")

    def setup_nodes(self):
        # Створення заданої кількості вузлів
        for i in range(self.num_nodes):
            node = Node(node_id=i+1)
            self.nodes.append(node)
        logging.debug("Всі вузли розподіленої системи створено.")

    def distribute_task(self, task, *args, **kwargs):
        # Розподіл завдань між вузлами у круговому режимі
        selected_node = random.choice(self.nodes)
        logging.info("Завдання " + str(task.__name__) + " розподілено на вузол "
+ str(selected_node.node_id))
        selected_node.add_task(task, *args, **kwargs)

    def shutdown_system(self):
        # Завершення роботи всієї розподіленої системи
        for node in self.nodes:
            node.shutdown()
        logging.info("Розподілена система зупинена.")

    def simulate_distributed_processing(self):
        # Симуляція обробки завдань у розподіленій системі
        def sample_task(task_id):
            logging.info("Виконання завдання " + str(task_id) + " у розподіленій
системі.")
            time.sleep(random.uniform(0.5, 1.5))
            logging.info("Завдання " + str(task_id) + " виконано.")

        for i in range(10):
            self.distribute_task(sample_task, task_id=i+1)
            time.sleep(0.2)
        # Очікування завершення завдань
        time.sleep(5)
        self.shutdown_system()

# ================================================================================
# Інтеграція з системами сповіщень та алертів
# ================================================================================

class AlertSystem:
    # Клас для інтеграції з системами сповіщень та алертів
    def __init__(self):
        self.scheduled_alerts = []
        self.alert_lock = threading.Lock()
```

```
        logging.info("AlertSystem ініціалізовано.")

    def send_alert(self, message, level="info"):
        # Надсилання миттєвого сповіщення або алерту
        alert_time = datetime.datetime.now().isoformat()
        alert_message = f"[{alert_time}] [{level.upper()}] {message}"
        # Симуляція інтеграції з зовнішніми сервісами сповіщень
        logging.info("Надсилання алерту: " + alert_message)
        print("ALERT:", alert_message)

    def schedule_alert(self, message, delay, level="info"):
        # Планування алерту через задану затримку
        scheduled_time = datetime.datetime.now() +
datetime.timedelta(seconds=delay)
        with self.alert_lock:
            self.scheduled_alerts.append((scheduled_time, message, level))
        logging.info("Алерт заплановано на " + str(scheduled_time) + " з
повідомленням: " + message)

    def process_scheduled_alerts(self):
        # Обробка запланованих алертів
        while True:
            now = datetime.datetime.now()
            with self.alert_lock:
                pending_alerts = list(self.scheduled_alerts)
            for alert in pending_alerts:
                scheduled_time, message, level = alert
                if now >= scheduled_time:
                    self.send_alert(message, level)
                    with self.alert_lock:
                        if alert in self.scheduled_alerts:
                            self.scheduled_alerts.remove(alert)
            time.sleep(0.5)

    def simulate_alerts(self):
        # Симуляція надсилання та планування алертів
        alert_thread = threading.Thread(target=self.process_scheduled_alerts)
        alert_thread.daemon = True
        alert_thread.start()
        # Надсилання миттєвих алертів
        self.send_alert("Система працює в нормальному режимі.", level="info")
        self.send_alert("Виявлено незначну проблему з модулем A.",
level="warning")
        # Планування алертів на майбутнє
        self.schedule_alert("Перевірте систему охолодження.", delay=3,
level="critical")
        self.schedule_alert("Запланована перевірка серверів через 5 секунд.",
delay=5, level="info")
        # Затримка для демонстрації обробки алертів
        time.sleep(7)
```

```
# ============================================================================
# Головна функція для запуску розширених функцій
# ============================================================================

def run_extensions_21_24():
    # Запуск симуляції машинного навчання для оптимізації розкладу реклами
    optimizer = AdScheduleOptimizer()
    optimizer.simulate_optimization()

    # Запуск розширеного логування та моніторингу системи
    adv_logger = AdvancedLogger()
    adv_logger.log_event("Початок розширеного моніторингу системи.",
level="info")
    adv_logger.simulate_monitoring()

    # Запуск симуляції розподіленої архітектури
    distributed_system = DistributedSystem(num_nodes=4)
    distributed_system.simulate_distributed_processing()

    # Запуск інтеграції з системами сповіщень та алертів
    alert_system = AlertSystem()
    alert_system.simulate_alerts()

    # Завершення роботи
    time.sleep(2)
```