МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

РОЗРОБЛЕННЯ ІНТЕГРОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ UVS З ФУНКЦІЄЮ ВІДСТЕЖЕННЯ СТАТУСУ ЧЛЕНСТВА ТА ОПЛАТОЮ ЧЕРЕЗ РІЗНІ ПЛАТІЖНІ ПЛАТФОРМИ

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Виконав студент групи МгІТ 1-23
Прохоренко А.Л.

Науковий керівник к.т.н., доц.
Мельник Г.В.

Рецензент к.т.н., доц. Астістова Т.І.

Київ 2024

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет Мехатроніки та комп'ютерних технологій

Кафедра Комп’ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп’ютерні науки

Освітня програма Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН

_____ Наталія ЧУПРИНКА

«_____»_____ 20____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Прохоренку Артему Леонідовичу

1. Тема кваліфікаційної роботи *Розроблення інтегрованої системи управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи*
Науковий керівник роботи *Мельник Геннадій Валерійович*, к.т.н., доц.
затверджені наказом КНУТД від “03” вересня 2024 року № 188-уч.
2. Вихідні дані до кваліфікаційної роботи Розробка кафедри комп’ютерних наук. *Інтегрована система управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи на PHP*
3. Зміст кваліфікаційної роботи: Розділ 1 Теоретичні основи розробки системи управління членством; Розділ 2 Методологічні аспекти розроблення інтегрованої системи управління членством; Розділ 3 Практична реалізація інтегрованої системи управління членством на PHP.
4. Дата видачі завдання 08.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапу кваліфікаційної роботи	Орієнтовний термін виконання	Примітка про виконання
1	Вступ	09.10.2024	
2	Розділ 1. Теоретичні основи розробки системи управління членством	11.10.2024	
3	Розділ 2. Методологічні аспекти розроблення інтегрованої системи управління членством	18.10.2024	
4	Розділ 3. Практична реалізація інтегрованої системи управління членством на PHP	22.10.2024	
5	Висновки	28.10.2024	
6	Оформлення (чистовий варіант)	30.10.2024	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку	30.10.2024	
8	Подача кваліфікаційної роботи для рецензування (за 14 днів до захисту)	07.11.2024	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату та текстових співпадінь (за 10 днів до захисту)		
10	Подання кваліфікаційної роботи завідувачу кафедри (за 7 днів до захисту)		

З завданням ознайомлений:

Студент _____ Артем ПРОХОРЕНКО

Науковий керівник _____ Геннадій МЕЛЬНИК

АНОТАЦІЯ

Прохоренко А.Л. Розроблення інтегрованої системи управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи.

Кваліфікаційна робота за спеціальністю 122 - «Комп'ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2024 рік.

Мета роботи – дослідити теоретичні засади інтегрованої системи управління членством та обґрунтування практичних інструментів розроблення інтегрованої системи управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи.

На підставі проведеного дослідження та аналізу існуючих систем управління членством та різних методів практичної реалізації інтегрованої системи управління членством розроблено інтегровану систему управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи на PHP.

Сформульовано рекомендації щодо застосування основних практичних інструментів для розроблення інтегрованої системи управління членством та написано код для реалізації функції відстеження статусу членства на мовах Python та PHP.

Зроблено огляд та ілюстрацію (див. додаток В) функціоналу розробленої інтегрованої системи управління членством UVS. Деталізовано послідовність реєстрації в системі управління членством UVS, а також проведення онлайн-оплати через інтегровані платіжні платформи.

Робота містить 22 рисунки, 2 схеми, 2 таблиці та три додатки. Об'єм магістерської роботи становить 90 сторінок.

Ключові слова: інтегрована система, код, управління членством, онлайн-оплата, функціонал, архітектура, особистий кабінет, меню, технології.

ANNOTATION

Prokhorenko A.L. Development of an Integrated Membership Management System for UVS with Membership Status Tracking and Payment via Various Payment Platforms.

Qualification thesis in the specialty 122 - "Computer Science." – Kyiv National University of Technologies and Design, Kyiv, 2024.

Objective: The aim of the thesis is to investigate the theoretical foundations of an integrated membership management system and to justify practical tools for the development of an integrated UVS membership management system with membership status tracking and payment via various payment platforms.

Based on the conducted research and analysis of existing membership management systems and various methods for practical implementation, an integrated UVS membership management system with membership status tracking and payment via various payment platforms was developed in PHP.

Recommendations for applying key practical tools for the development of an integrated membership management system were formulated, and code for implementing the membership status tracking function was written in Python and PHP.

A review and illustration (see Appendix B) of the functionality of the developed integrated UVS membership management system were provided. The sequence of registration in the UVS membership management system and conducting online payments via integrated payment platforms were detailed.

The work includes 22 figures, 2 diagrams, 2 tables, and three appendices. The total volume of the master's thesis is 90 pages.

Keywords: integrated system, code, membership management, online payment, functionality, architecture, personal account, menu, technologies.

ЗМІСТ

ВСТУП

Сьогодні в умовах швидкого розвитку цифрових технологій організації стикаються з необхідністю ефективного управління членством, що включає контроль за статусом членів та зручне здійснення платежів. Багато систем управління членством мають обмежені функціональні можливості або недостатню інтеграцію з платіжними платформами, що ускладнює моніторинг і знижує ефективність процесів. З огляду на це, постає потреба вирішення **проблеми розробки інтегрованої системи управління членством (UVS)**, яка дозволить не лише відстежувати статус членства, але й забезпечить багатоканальну платіжну підтримку. Дослідження в галузі інтегрованих систем керування членством розвинуте недостатньо, зокрема, стосовно адаптації до сучасних українських умов і використання локальних платіжних рішень.

Актуальність дослідження зумовлена потребою в створенні високоякісних, зручних для користувачів систем управління членством, що відповідають сучасним вимогам до безпеки, зручності та функціональності. Розробка UVS з урахуванням інтеграції з платіжними платформами та функцією відстеження членства стане корисним рішенням для багатьох українських організацій, включно з комерційними підприємствами, навчальними закладами та неприбутковими організаціями. Така система покращить досвід користувачів і дозволить вчасно оновлювати інформацію про статуси членства.

Метою роботи є розроблення інтегрованої системи управління членством з функцією відстеження статусу членства та підтримкою різних платіжних платформ. **Основні завдання** кваліфікаційної роботи включають:

- аналіз існуючих систем управління членством та їхніх функціональних можливостей;
- визначення вимог до системи, які включають підтримку різних платіжних платформ;
- розробка архітектури системи та впровадження основних компонентів;

- тестування системи на відповідність вимогам та функціональним характеристикам.

Об’єктом дослідження є процеси управління членством у сучасних організаціях, що потребують автоматизації.

Предмет дослідження — методи та інструменти, які забезпечують відстеження статусу членства, інтеграцію з платіжними платформами, а також засоби для впровадження такої інтегрованої системи.

Наукова новизна роботи полягає у розробці унікальної інтегрованої системи управління членством, яка поєднує функціонал відстеження статусу з мультиплатформною платіжною підтримкою. Застосування сучасних технологій і методів забезпечує унікальну гнучкість та адаптивність системи для українських організацій, дозволяючи керувати процесами та забезпечувати зручний для користувачів досвід.

Розроблена система матиме **практичне застосування** для українських організацій, що бажають автоматизувати управління членством і спростити процеси оплати. Це дозволить забезпечити зручний та безпечний спосіб керування членськими даними і підвищити прозорість фінансових операцій. Впровадження системи UVS сприятиме підвищенню рівня лояльності користувачів завдяки гнучкості та багатоканальній підтримці оплати.

У роботі використовуються такі **методи наукових досліджень:**

- аналіз наявних інформаційних систем для управління членством та платіжних платформ;
- методи системного аналізу для визначення ключових вимог до функціональних компонентів системи;
- проєктування та моделювання для розробки архітектури системи;
- тестування для перевірки функціональності та стабільності роботи системи.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ

1.1. Аналіз існуючих систем управління членством.

Системи управління членством (Membership Management System) – це сукупність взаємопов'язаних елементів (людей, процесів, технологій), спрямованих на ефективне управління взаємовідносинами організації з її членами (членами асоціацій, профспілок, організацій, громад, спілок, партій тощо). [1, с.12]

У нашому інформаційному суспільстві система управління членством – це інтегрована платформа або програмне забезпечення, призначена для автоматизації процесів управління членськими організаціями, клубами, професійними асоціаціями, громадськими об’єднаннями, благодійними фондами та іншими спільнотами.

Основною метою таких систем є оптимізація комунікації з членами організації, спрощення процесів прийому та обліку членства, обробка фінансових транзакцій (наприклад, внески, пожертвування, навчання тощо), зберігання інформації про членів та управління взаємодією з ними. [1, с.20]

Сучасні системи управління членством виконують такі основні функції:

1. **Облік та зберігання інформації про членів**: уся інформація про членів організації централізовано зберігається у захищеній базі даних, що дає можливість легко знайти необхідну інформацію та забезпечує конфіденційність.
2. **Оплата членських внесків і фінансова інтеграція**: системи підтримують інтеграцію з платіжними платформами, що дозволяє членам організації легко здійснювати внески, оплачувати членство, проходити навчання чи робити благодійні внески.
3. **Управління подіями**: можливість організувати заходи, семінари, тренінги, вести реєстрацію учасників і розсилати запрошення та нагадування.

4. **Автоматизація комунікації**: використання автоматичних листів, повідомлень або нагадувань дозволяє підтримувати активний зв'язок із членами. [2, с.10]
5. **Аналітика та звітність**: надання інформаційних звітів для оцінки залученості членів, аналізу зростання/зменшення кількості членів, аналізу фінансових потоків тощо.
6. **Залучення нових членів**: розробка стратегій залучення, створення привабливого іміджу організації.
7. **Утримання членів**: забезпечення задоволеності членів, розробка програм лояльності, вирішення конфліктів.
8. **Розвиток членства**: надання можливостей для професійного зростання, організація навчальних заходів, делегування повноважень.
9. **Представництво інтересів членів**: захист прав і інтересів членів перед третіми особами, участь у переговорах.

Електронні системи управління членством активно використовуються в Україні для автоматизації процесів взаємодії з членами різноманітних організацій: професійних об'єднань, громадських організацій, спортивних клубів, освітніх установ тощо. [3, с.40] Ці системи дозволяють значно спростити управління фінансами, комунікацію, організацію подій та обробку великих обсягів інформації. Окремі українські розробники пропонують локалізовані рішення, що враховують специфіку національного ринку. Серед найпопулярніших платформ можна виділити My Membership, Clubeo та SWE Membership.

My Membership: спеціалізована платформа, яка надає комплексні рішення для управління членством. Ця система підходить для великих громадських і благодійних організацій. Розглянемо детальніше функціонал платформи. [4, с.80]

Можливості My Membership.

- Платформа надає повний спектр можливостей для обліку та адміністрування членів, включаючи зберігання персональних даних у захищеній базі, ведення історії внесків, автоматизацію комунікації.

- Вбудована інтеграція з популярними платіжними системами, що дозволяє організаціям приймати оплату онлайн, зручно обробляючи членські внески, пожертвування та інші кошти.
- Має інструменти для організації подій із можливістю реєстрації учасників, що актуально для асоціацій, які проводять семінари або конференції.
- На платформі є аналітичні модулі для відстеження активності членів, динаміки росту/зниження членської бази, фінансових потоків.

Технічний аналіз системи My Membership.

- Система розроблена на основі PHP із застосуванням сучасних фреймворків (наприклад, Laravel), що забезпечує стабільність і швидку обробку запитів.
- Використання MySQL для обробки великих обсягів даних дозволяє забезпечити надійне зберігання та швидкий доступ до інформації. [5, с.45]
- Інтеграція API для підтримки онлайн-платежів (Wayforpay, LiqPay) забезпечує безпечні фінансові операції.

Переваги My Membership: широкий функціонал, надійна система безпеки, адаптація до українського ринку.

Недоліки My Membership: висока вартість впровадження для невеликих організацій, потреба у технічній підтримці при налаштуванні платіжних модулів.

Платформа Clubeo: орієнтована на спортивні та громадські клуби.

Функціонал Clubeo:

- Основний акцент платформи на зручність управління спортивними та клубними організаціями; система має інтуїтивний інтерфейс для обліку членів.
- Вбудована функціональність для реєстрації на події та інтеграції з соціальними мережами, що сприяє популяризації подій і забезпечує швидку комунікацію.
- Проста аналітика та можливість налаштування шаблонів електронних розсилок для підтримки комунікації з членами клубу.

Технічний аналіз Clubeo:

- Розробка базується на CMS, що спрощує впровадження системи для невеликих організацій із мінімальними потребами в технічній підтримці.
- База даних побудована на SQLite, що оптимізує витрати на інфраструктуру, але обмежує обсяги даних, які можуть зберігатися. [6, с.65]

Переваги Clubeo: простота впровадження, мінімальні витрати на підтримку, зручна інтеграція із соціальними мережами.

Недоліки Clubeo: обмежений функціонал для фінансових операцій, недостатня гнучкість для великих організацій.

SWE Membership: українська платформа, що пропонує можливість інтеграції з державними реєстрами.

Функціонал SWE Membership:

- Система забезпечує гнучке налаштування обліку членів, інтеграцію з державними реєстрами, що важливо для офіційних об'єднань та профспілок.
- Наявність підтримки багатоканальних комунікацій дозволяє організаціям вибудовувати індивідуальні сценарії взаємодії з членами, включаючи e-mail, SMS та месенджери.
- Платформа пропонує функціонал для фінансового обліку, включаючи можливість налаштування різних типів платежів, знижок та промокодів. [7, с.90]

Технічний аналіз SWE Membership:

- Побудована на основі Node.js, що підвищує масштабованість і дає змогу обробляти численні запити одночасно.
- Використання бази даних PostgreSQL забезпечує можливість зберігання великого обсягу даних, що актуально для організацій із великою кількістю членів.
- Інтеграція з державними API та підтримка OAuth2 для підвищення безпеки доступу.

Переваги SWE Membership: масштабованість, високий рівень безпеки, підтримка різних каналів комунікації. [8, с.190]

Недоліки SWE Membership: складність налаштування, потреба у високій кваліфікації технічного персоналу для впровадження та підтримки.

Електронні системи управління членством в Україні адаптовані до потреб організацій різних масштабів і типів. Вибір платформи залежить від специфічних вимог організації, фінансових можливостей і потреб у функціоналі. [9, с.198]

1.2. Моделі та методи управління членством.

Моделі управління членством – це спрощені зображення реальних процесів управління, які дозволяють зрозуміти їхню логіку та взаємозв'язки. [10, с.298]

Методи управління членством – це інструменти та прийоми, за допомогою яких реалізовуються моделі управління.

Моделі та методи управління членством в асоціаціях та інших організаціях є фундаментальними елементами в забезпеченні ефективного функціонування організацій, що об’єднують учасників із спільними інтересами, цілями або професійною діяльністю. [11, с.305] **Метою** таких систем є підтримка оптимальної взаємодії між учасниками, спрощення адміністративних процесів, забезпечення ефективного менеджменту членських внесків, а також підвищення загальної продуктивності асоціації.

Основними завданнями управління членством є організація комунікацій між членами, облік фінансових операцій, зокрема членських внесків, а також відстеження статусу учасників. Завдяки використанню ефективних моделей управління створюються умови для стратегічного розвитку організації, що включає залучення нових членів, підвищення їхньої залученості та забезпечення ефективної комунікації. [12, с.425] Системи управління членством дозволяють асоціаціям автоматизувати та спростити ці процеси, зробивши організацію гнучкішою та здатною швидко реагувати на зміни.

У сучасному середовищі управління системами членства особливого значення набуває цифровізація. Це забезпечує асоціаціям можливість ведення

обліку, управління базами даних, комунікації та автоматизованих звітів, що значно спрощує роботу та покращує обслуговування членів спільнот. Тож, можна стверджувати, що моделі та методи управління членством є важливими компонентами, які дозволяють оптимізувати діяльність асоціацій та підвищити їхню ефективність.

Існує декілька типів моделей та методів управління членством, кожен із яких має свої особливості, переваги, обмеження та сфери застосування. [13, с.525] Основні з них включають:

- Моделі CRM-систем;
- Моделі AMS-систем;
- Моделі LMS-систем;
- Гібридні моделі.

Моделі CRM-систем. CRM (Customer Relationship Management) системи [14, с.480] набули широкої популярності як інструмент управління взаємовідносинами з членами асоціації. Вони дозволяють створювати централізовану базу даних, в якій зберігається інформація про членів, їхня активність, історія внесків та інше. Основні функції CRM-систем включають відстеження залученості членів, надання персоналізованих пропозицій, автоматизацію розсилок та повідомлень.

Переваги CRM-систем: централізована база даних, автоматизація процесів, можливість персоналізації.

Недоліки CRM-систем: потреба в постійному оновленні даних, витрати на впровадження та підтримку.

Моделі AMS-систем. **AMS (Association Management Software) системи** розроблені спеціально для потреб асоціацій. [15, с.380] Ці системи дозволяють управляти членськими внесками, організовувати заходи, вести облік фінансових операцій та підтримувати комунікацію з членами. AMS-системи також надають можливість аналізувати дані про членів та їхню залученість, що сприяє прийняттю стратегічних рішень.

Переваги AMS-систем: широкий спектр функцій, спеціалізація на управлінні асоціаціями, зручність у використанні.

Недоліки AMS-систем: обмеження у персоналізації, високі витрати на впровадження.

Моделі LMS-систем. **LMS (Learning Management Systems) системи** [16, с.580] зазвичай використовуються для організації навчання в асоціаціях, де передбачено розвиток членів через освітні програми. Вони дозволяють структурувати процес навчання, організовувати вебінари, курси, створювати оцінки для учасників.

Переваги LMS-систем: структуроване навчання, доступ до освітніх ресурсів.

Недоліки LMS-систем: вузький фокус на навчанні, можливі труднощі з інтеграцією інших функцій.

Гібридні моделі. Гібридні моделі об'єднують функціональні можливості декількох систем для створення комплексного інструменту управління членством. [17, с.280] Такі моделі дозволяють асоціаціям одночасно вирішувати завдання CRM, AMS та LMS.

Переваги гібридних систем: комплексність, ширший спектр можливостей.

Недоліки гібридних систем: висока вартість, складність інтеграції.

Порівняння моделей управління членством можна здійснювати за кількома ключовими параметрами: функціональні можливості, спеціалізація, зручність у використанні, вартість впровадження та підтримки. [18, с.365] CRM-системи орієнтовані на управління взаємовідносинами з членами, що робить їх оптимальними для персоналізованого підходу та маркетингових активностей. Водночас AMS-системи мають чітко визначену спеціалізацію на управлінні членством, що робить їх придатними для більш комплексного обліку членських внесків та управління заходами.

LMS-системи, у свою чергу, добре підходять для асоціацій, що зосереджені на навчанні своїх членів, але мають обмеження щодо управління іншими аспектами

діяльності асоціації. Гібридні моделі є універсальним рішенням, але можуть бути дорогими в реалізації та підтримці.

1.3. Технології розробки програмного забезпечення для систем управління.

Технологія – це комплекс правил, методологій та засобів, [19, с.565] що забезпечують упорядковане функціонування виробничого процесу, спрямованого на створення певного продукту. Вона охоплює також процеси планування, вимірювання показників, оцінювання якості, визначення відповідальності виконавців та інші ключові аспекти.

Сьогодні існує розбіжність у поглядах на поняття «технологій програмування». З одного боку, технології трактують як інтеграцію широкого спектру інструментальних засобів, що полегшують розробку. З іншого – під технологією розуміють чітко структурований набір формальних методологій і регламентів, який дає змогу на кожному етапі виконувати експертизу, архівування, а також контролювати обсяги та якість виконаної роботи. Перший підхід підтримують фахівці-програмісти, тоді як другий розділяють керівники проєктів. [20, с.470]

Процес розробки програмного забезпечення на замовлення охоплює питання документування, ціноутворення, методів регламентації та контролю за виконанням робіт. Проте основним результатом застосування відповідних технологій є створення добре налагодженої, правильно задокументованої програми, яка функціонує в заданому обчислювальному середовищі, є зрозумілою для подальшого вдосконалення та підтримки. [21, с.493]

Сучасні замовники потребують комплексних рішень для виконання своїх завдань, що зумовлює необхідність розробки складних програмних систем. Для створення таких систем залучення тільки кваліфікованих програмістів є недостатнім; потрібні також системні аналітики, які здійснюють аналіз замовлення та розробляють проєкт системи, і системні інженери, що забезпечують реалізацію завдань у вигляді комплексної системи. Застосування стандартизованих підходів і

технологій є необхідним для успішної реалізації таких проєктів. Виходячи з вищезазначеного, можна дати визначення технології програмування. [22, с.393]

Таким чином, **технологія програмування (ТП)** – це сукупність методів та інструментів для розробки програмного забезпечення та встановлений порядок їхнього застосування. [23, с.397]

Усі технології, що використовуються програмістами, спрямовані на досягнення основних цілей:

- забезпечення високої якості продукту;
- дотримання виділеного бюджету;
- виконання робіт у встановлені терміни.

Технологія програмування складається з системи технологічних інструкцій, що містять:

- визначення послідовності виконання технологічних операцій;
- умови, за яких виконується кожна операція;
- опис операцій із вказівкою вхідних даних, очікуваних результатів, а також інструкцій, нормативів, стандартів, критеріїв та методів оцінки тощо.

Процес створення програмного забезпечення можна розподілити на чотири основні етапи або стадії (рис. 1.1): [24, с.370]

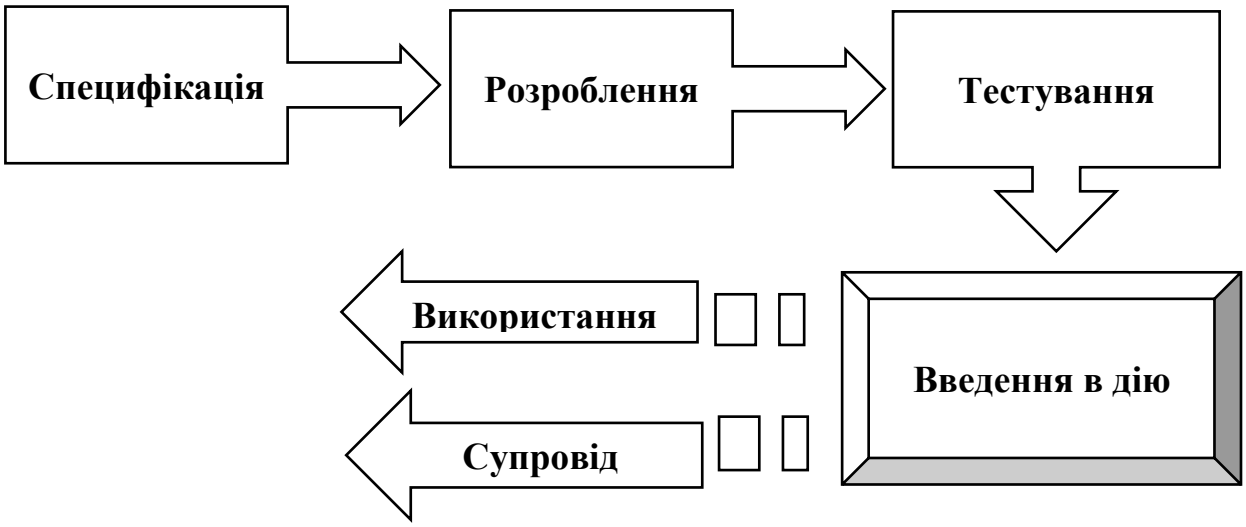

Рисунок 1.1. Схема процесу створення ПЗ

1. **Специфікація** – формулювання та визначення ключових вимог до програмного забезпечення.
2. **Розроблення** – реалізація програмного забезпечення, орієнтована на задоволення вимог, визначених у специфікаціях.
3. **Тестування** – контроль якості, що передбачає перевірку програмного продукту на відповідність вимогам замовника.
4. **Супровід та модернізація** – адаптація і вдосконалення програмного забезпечення у відповідь на нові потреби клієнта.

Ці стадії є базовими етапами життєвого циклу ПЗ, кожен з яких сприяє створенню продукту, що відповідає заданим специфікаціям і очікуванням користувачів.

Розроблення будь-якої програми, незалежно від її масштабу – від простої процедури обробки даних, що надходять на консоль, до складного програмного продукту – складається з кількох етапів, ретельне дотримання яких є необхідною умовою для досягнення якісного результату. [25, с.570] Суворе слідування етапам розроблення програмного забезпечення, які перевірені на практиці, стає основним критерієм як для компаній-розробників ПЗ, так і для їхніх клієнтів, які очікують на надійну програму, що відповідає заданим умовам. Розглянемо кожен етап загальновизнаної методології розроблення ПЗ, щоб оцінити їх важливість для досягнення поставленої мети.

Процес створення будь-якої програми поділяється на наступні етапи: [26, с.635]

1. Постановка завдання.
2. Аналіз завдання та моделювання.
3. Розробка або вибір алгоритму вирішення завдання.
4. Проєктування загальної структури програми.
5. Кодування.
6. Налагодження та тестування програми.
7. Аналіз результатів.
8. Публікація результатів роботи та передача замовнику.

9. Супровід програми.

1. Постановка завдання – здійснюється фахівцем у відповідній галузі природною мовою (наприклад, українською чи англійською). На цьому етапі визначається мета завдання, його зміст та загальний підхід до розв'язання. У деяких випадках завдання може бути вирішено аналітично без використання комп'ютера. Вже на етапі постановки важливо враховувати ефективність алгоритму вирішення на комп'ютері, а також обмеження, що можуть накладатися апаратним і програмним забезпеченням (АТ і ПЗ).

При підготовці до проєктування вирішуються ключові організаційні аспекти:

- визначається, що саме клієнт може надати (технічне завдання, макети, дизайн), а також наскільки ці вихідні матеріали є повними та які етапи робіт вони покривають — це дозволяє сформувати склад майбутніх робіт;
- уточняються бюджет і терміни: на основі наявних матеріалів визначаються приблизна вартість, загальна тривалість проєкту, а також терміни і точна вартість найближчого етапу. [27, с.715] Лише після цього можна укладати контракт, отримувати передоплату та всі необхідні для виконання роботи матеріали.

Під час початкових обговорень замовлення на розроблення програмного забезпечення замовник часто має лише загальне уявлення про свої потреби. Зазвичай він надає кілька сторінок опису завдання і одразу запитує оцінку часу виконання та вартості. Проте без чіткого визначення процесів, для яких необхідна автоматизація, неможливо навіть приблизно оцінити обсяг робіт. [28, с.710] Початкова оцінка за умови мінімальної кількості вхідної інформації може містити значні похибки, що негативно впливає на точність визначення строків виконання та загальної вартості проєкту.

Щоб уточнити обсяг робіт, замовнику слід запропонувати надати або розробити детальне технічне завдання. Це дозволить системним аналітикам більш глибоко ознайомитися із завданням, застосувати інструментальні засоби для

декомпозиції системи на окремі компоненти, приблизно визначити обсяги цих компонентів і, відповідно, необхідний час для їх реалізації.

Процес "формалізації постановки завдання" передбачає розробку послідовних моделей, кожна з яких описує систему та її оточення з різних точок зору із поступовим збільшенням рівня деталізації. Важливо, щоб усі представлені моделі системи зберігалися в єдиному репозиторії (спеціалізованій базі даних). Це забезпечить наскрізний процес проєктування, коли кожна наступна модель використовує результати попередньої без суперечностей між ними. Всі можливі перевірки також мають бути наскрізними. [29, с.610]

2. Аналіз завдання та моделювання – на цьому етапі визначаються вхідні дані та очікуваний результат, а також виявляються обмеження на їх значення. Виконується формалізація завдання та побудова (або вибір) математичної моделі, що придатна для оброблення на комп’ютері.

Метою аналізу та моделювання є виявлення чіткої та відносно спрощеної внутрішньої структури проєкту (також відомої як архітектура проєкту), яка: [30, с.520]

- відповідає встановленим функціональним характеристикам і специфікаціям;
- узгоджена з обмеженнями, що накладаються апаратним забезпеченням;
- задовольняє як явні, так і неявні експлуатаційні вимоги;
- відповідає явним та прихованим критеріям дизайну;
- відповідає вимогам процесу розробки, включаючи часові й фінансові обмеження.

На цьому етапі розробки програмного забезпечення важливим завданням є всебічний аналіз вимог замовника, висунутих до створюваного програмного продукту, щоб чітко визначити основні цілі та завдання кінцевого рішення. У рамках цього процесу забезпечується ефективна комунікація між клієнтом, що потребує програмного рішення, та командою розробників, що сприяє точному формулюванню вимог до програмного забезпечення. [31, с.529] Наслідком цього

аналізу є створення основного регламентуючого документа — технічного завдання (ТЗ) на розробку програмного забезпечення. ТЗ повинно всебічно описувати поставлені перед розробниками задачі та чітко окреслювати кінцеву мету проєкту з погляду замовника.

Продуктом етапів аналізу та проєктування стають моделі, які дозволяють як розробникам, так і замовникам зрозуміти структуру майбутньої системи, узгодити вимоги та розробити план реалізації. [32, с.429] Під час розробки таких моделей використовуються сучасні методології та інструменти об'єктно-орієнтованого аналізу та проєктування, що дозволяє ефективно реагувати на зростаючі потреби у проєктуванні складних бізнес та технологічних процесів.

3. Розробка або вибір алгоритму вирішення завдання – цей етап базується на математичному описі завдання. Багато задач можуть бути вирішені кількома методами, тому програміст має обрати оптимальне рішення. [33, с.229] Недоліки на етапах постановки, аналізу або розроблення алгоритму можуть призвести до прихованих помилок – програміст може отримати хибний результат, вважаючи його коректним.

Розробка програмного забезпечення, як і будь-яка інша діяльність, включає виконання операцій і проєктів, що мають спільні риси, зокрема використання людських ресурсів та обмеженість доступних ресурсів. Проте ключова різниця між ними полягає в тому, що операції характеризуються безперервністю і повторюваністю, тоді як проєкт є тимчасовим та унікальним. У зв’язку з цим, проєкт визначається як обмежена у часі діяльність, спрямована на створення неповторного продукту або послуги. Поняття «тимчасовий» означає, що кожен проєкт має чітко визначені дати початку та завершення. Унікальність продукту передбачає його суттєві відмінності від усіх аналогічних рішень або послуг. [34, с.376]

Виходячи з цього, розробка програмного забезпечення відповідає визначенню проєкту, а тому для управління цим процесом доцільно використовувати проєктні методи та інструменти. Наприклад, створення веб-сайту є проєктом, тоді як його довгострокова підтримка відноситься до операційної

діяльності. Кожен проєкт має чітко встановлені початок і завершення, що настає або після досягнення всіх визначених цілей, або у випадку неможливості їх досягнення. Тимчасовий характер проєкту не завжди означає його коротку тривалість — розробка складних програмних систем може тривати роками, хоча зазвичай проєкти мають обмежені часові рамки, враховуючи обмежений період ринкової кон'юнктури для цих продуктів. Після завершення проєкту його команда розформовується, і її учасники переходять до нових проєктів.

Вибір або розробка алгоритму і чисельного методу для розв'язання задачі є критичними факторами успішного виконання роботи над програмою. Надійно розроблений алгоритм є необхідною умовою для ефективного створення програмного продукту. [35, с.386]

4. Проєктування загальної структури програми – формується модель рішення із подальшою деталізацією та поділом на підпрограми, визначається архітектура програми, а також спосіб зберігання інформації (зокрема, змінні, масиви тощо).

Наступний важливий етап розробки програмного забезпечення – це стадія проєктування, що охоплює моделювання теоретичних засад майбутнього продукту. [36, с.396] Сучасні програмні засоби дозволяють частково поєднати процеси проєктування та кодування, тобто технічної реалізації проєкту, оскільки базуються на об’єктно-орієнтованому підході. Однак, для повного й точного планування необхідно ретельніше моделювання, яке враховує всі можливі нюанси.

Якісний аналіз перспектив та функціональних можливостей створюваного продукту є основою для забезпечення його стабільної роботи та виконання комплексу покладених на нього завдань. Одним із важливих аспектів етапу проєктування є вибір інструментів та операційної системи, які нині представлені на ринку в широкому асортименті.

На цьому етапі здійснюється «архітектурне» опрацювання проєкту: визначаються частини алгоритму, які доцільно виділити у вигляді підпрограм чи модулів, а також обирається спосіб зберігання даних – у формі простих змінних, масивів чи інших структур.

На даному етапі сторони зобов'язані провести:

- оцінку результатів початкового аналізу, включно з ідентифікацією виявлених обмежень та пошуком критичних елементів проєкту; [37, с.401]
- розроблення остаточної архітектури створюваної системи; аналіз доцільності застосування програмних модулів або готових рішень від сторонніх розробників;
- проєктування ключових компонентів продукту, зокрема моделі бази даних, процесів та програмного коду;
- вибір середовища програмування й інструментів розробки, затвердження інтерфейсу застосунку з акцентом на елементи графічного відображення даних;
- визначення основних вимог щодо безпеки розроблюваного програмного забезпечення.

5. Кодування – перетворення алгоритму у програмний код. [38, с.411] Сучасні середовища програмування дозволяють прискорити цей процес, автоматично генеруючи частини коду, але фундаментальна творча робота залишається за програмістом. Для успішного досягнення цілей проєкту необхідне використання структурного програмування.

Кодування являє собою процес фіксації алгоритму мовою програмування. Якщо алгоритм розв'язання задачі, структура програми та структура даних детально опрацьовані й чітко задокументовані, це суттєво зменшує час на кодування та знижує ймовірність помилок на цьому етапі. Досліджувати всі особливості найскладнішого етапу розробки немає потреби; достатньо зазначити, що успіх реалізації будь-якого проєкту напряму залежить від якості попереднього аналізу та оцінки альтернативних рішень, з якими створюваний продукт змагатиметься за лідерство у своїй ніші.

У випадку написання коду для виконання спеціалізованих задач конкретного підприємства, від якості та професійності кодування значною мірою залежить ефективність роботи компанії-замовника. [39, с.311] Кодування може

здійснюватися паралельно з наступним етапом — тестуванням програмного забезпечення, що дозволяє вносити корективи безпосередньо в процесі написання коду.

Узгодженість дій між програмістами, тестувальниками та проєктувальниками є вирішальною для досягнення високої якості проєкту. [40, с.211] Якість коду суттєво впливає на працездатність і надійність системи. Для забезпечення якісного кодування необхідно постійно оновлювати знання та вдосконалювати навички. Код слугує не лише функціональним елементом, але й містить стислі та зрозумілі відомості про стан проєкту. Програмісти розробляють код відповідно до стандартів, що сприяє ефективній комунікації через кодові інструкції.

6. Налагодження та тестування програми – налагодження передбачає виправлення помилок у коді, тоді як тестування дозволяє знайти ці помилки та, зрештою, впевнитись, що налагоджена програма виконує завдання правильно. [40, с.415] Для цього розробляється система тестів – підібраних контрольних прикладів з параметрами, для яких відоме рішення задачі. Тестування повинне охоплювати всі можливі варіанти розгалужень у програмі, включаючи дані, для яких рішення неможливе. Перевірка виняткових ситуацій є необхідною для оцінки коректності програми, наприклад, відмова у видачі коштів, якщо вони відсутні на рахунку клієнта банку. Відповідальні проєкти вимагають стійкості програми до некоректного користувацького вводу, що передбачає "захист від дурня". Використання спеціальних відлагоджувальних програм, які дозволяють виконувати програму покроково із переглядом значень змінних, значно полегшує цей етап.

Процес тестування дає змогу відтворити умови, за яких програмний продукт може припинити коректно функціонувати. [39, с.415] Після цього відділ налагодження виявляє та усуває знайдені дефекти в коді, доводячи його до майже бездоганного стану. Ці два етапи займають не менше 30% від загального часу, витраченого на проєкт, оскільки від якості їх виконання залежить подальша успішність розробленого програмного забезпечення. Часто функції тестувальника

та відладчика виконує один відділ, однак найдоцільнішим є розподіл цих завдань між окремими виконавцями, що підвищує ефективність виявлення помилок у програмному коді.

7. Аналіз результатів – якщо програма виконує моделювання певного процесу, необхідно порівняти результати обчислень із фактичними спостереженнями. При значних розбіжностях слід змінити модель.

Одержання та інтерпретація результатів із можливим подальшим коригуванням моделі є важливим етапом процесу розробки. [38, с.545] Після завершення перевірки програми та усунення основних помилок виникає обґрунтована впевненість у тому, що вона, в межах обраної моделі, забезпечує достовірний результат. На цьому етапі необхідно провести детальний аналіз отриманих даних. Якщо програма моделює природний процес, результати комп'ютерного моделювання варто співставити з даними спостережень. Такий аналіз відомий як інтерпретація результатів розрахунку. Однак програміст може зіткнутися з ситуацією, коли результат не відповідає очікуваному, що може вимагати внесення змін у саму модель для підвищення її реалістичності. Після цього, отримавши файл виконуваної програми у форматі *.exe, ми запускаємо його для повторної перевірки коректності роботи програми. На цьому етапі розробка програми вважається завершеною.

8. Публікація результатів роботи та передача замовнику – програма передається в експлуатацію після завершення розробки та тестування.

Завершальним етапом процесу програмування є документування, яке включає широкий спектр описів, що сприяють полегшенню розробки програмного забезпечення та узагальненню підсумкового продукту. [37, с.505] Постійне створення документації має бути невід'ємною частиною кожного етапу розробки. До основних документів належать опис завдання, проєктна документація, алгоритми та програми. Внутрішня документація, включена безпосередньо в код, забезпечує полегшене читання та розуміння програмного коду.

Документація програмного забезпечення (англ. software documentation) [36, с.205] – це сукупність супровідних документів, які містять відомості про

загальні принципи та рекомендації для користувачів перед початком роботи з програмним забезпеченням. Така документація є надзвичайно важливою, адже містить не лише інструкції з використання, а й пояснення до основних застосованих алгоритмів.

Залежно від складності, специфіки та ліцензійних умов кожного програмного продукту, обсяг і зміст документації можуть значно відрізнятися. [36, с.705] У процесі розробки програмного забезпечення створюється значна кількість різноманітних документів, що сприяють як передачі інформації між розробниками, так і управлінню процесом створення та розвитку програмного продукту, а також забезпечують користувачів необхідною інформацією для його подальшого використання та обслуговування.

9. Супровід програми – передбачає надання консультацій представникам замовника щодо роботи з програмою та навчання персоналу. Недоліки та помилки, виявлені під час експлуатації, повинні бути усунені.

Документація з супроводу програмного забезпечення детально висвітлює процес його розробки та необхідна у випадках, коли потрібно зрозуміти його структуру та принципи функціонування. Супровід є продовженням розробки, тому, коли вдосконалення та оновлення програмного продукту здійснюють не його автори, залучається спеціальна команда розробників, яка займається супроводом. Ця команда користуватиметься тією самою документацією, проте з акцентом на ретельний перегляд та аналіз матеріалів, створених початковими розробниками. [35, с.705] Це дозволяє зрозуміти змінювану архітектуру програми та особливості її розробки, а також зробити відповідні корективи у документацію. Такий підхід дозволяє повторити значну частину технологічних процесів, що застосовувалися під час створення початкової версії програмного забезпечення.

Інструменти для розробки та аналізу вимог. Інструменти для розробки та аналізу вимог – це спеціалізоване програмне забезпечення, яке допомагає команді проєкту визначати, документувати, відстежувати, пріоритизувати та керувати вимогами до інформаційних систем. [34, с.654] Ці інструменти полегшують процеси збору інформації від зацікавлених сторін, аналізу їхніх потреб

та трансформації цих потреб у конкретні вимоги до системи. Вони особливо важливі у великих проєктах, де висока складність системи вимагає чіткої організації та структуризації вимог.

Основні **функції цих інструментів** включають інтеграцію з іншими системами для тестування та забезпечення якості, підтримку командної роботи над вимогами та полегшення комунікації між технічними спеціалістами та бізнес-аналітиками.

Інструменти для роботи з вимогами виконують широкий спектр функцій, включаючи такі: [33, с.354]

- **Документування вимог** – забезпечення єдиного середовища для зберігання та документування вимог, що спрощує доступ до інформації і її актуалізацію.
- **Управління змінами вимог –** можливість відслідковувати зміни в вимогах у реальному часі та забезпечувати автоматизоване оновлення у випадку модифікації.
- **Відстеження стану вимог –** інструменти дозволяють відстежувати, на якому етапі знаходиться кожна вимога, від її початкового формулювання до повного виконання.
- **Пріоритизація вимог –** надають можливість встановлювати пріоритети, що допомагає команді зосередитись на найбільш важливих аспектах проєкту.
- **Інтеграція з іншими системами –** часто підтримується інтеграція з інструментами для тестування, управління проєктом, а також з системами контролю версій.
- **Спільна робота та комунікація –** підтримка колективної роботи, де учасники проєкту можуть спільно працювати над вимогами в режимі реального часу.

Документування вимог до системи є одним із ключових етапів у процесі розробки програмного забезпечення, що забезпечує створення чіткої і структурованої основи для подальшої розробки, тестування та впровадження

системи. [32, с.454] Вимоги до системи визначають функціональність і характеристики, які вона повинна мати для задоволення потреб користувачів і досягнення бізнес-цілей організації. Для системи управління членством, таке документування дає змогу визначити специфічні завдання, які система повинна виконувати: зберігання даних про користувачів, управління членськими статусами, надання інструментів для взаємодії з членами, сповіщення про статуси тощо.

Процес документування включає створення документів, які описують вимоги до системи з технічного і бізнес-поглядів, у формі текстових описів, діаграм, таблиць тощо. Такий підхід дає можливість чітко фіксувати, що має бути розроблено, з мінімізацією ризику виникнення непорозумінь між командою розробників, замовниками та кінцевими користувачами.

Вимоги до системи управління членством можуть включати: [31, с.454]

Функціональні вимоги: описують функції, які система повинна підтримувати (наприклад, можливість автоматичного оновлення членського статусу).

Нефункціональні вимоги: зосереджуються на атрибутах, як-от швидкість, безпека, масштабованість системи.

Бізнес-вимоги: висвітлюють стратегічні цілі компанії, які система має підтримувати.

Документування вимог **здійснюється** в кілька етапів і включає різні методи збору, аналізу та фіксації інформації.

1. **Збір вимог**. На цьому етапі розробники та аналітики співпрацюють із зацікавленими сторонами, щоб зрозуміти їхні потреби. Методи збору можуть включати опитування, інтерв'ю, робочі наради та аналіз бізнес-процесів. Наприклад, для системи управління членством це можуть бути зустрічі з командою підтримки членів і маркетологами, які відповідають за роботу з базою даних членів.
2. **Аналіз вимог**. Після збору інформації аналітики аналізують вимоги, виявляючи конфлікти між різними запитами та уточнюючи деталі. Вимоги повинні бути повними, коректними, узгодженими та актуальними.

3. **Візуалізація вимог**. Це може включати використання діаграм та інших візуальних засобів, наприклад, діаграм потоків даних (Data Flow Diagrams), UML-діаграм для опису основних компонентів та їхніх взаємозв'язків. Візуалізація допомагає зацікавленим сторонам легше зрозуміти структуру системи та її функціональність.
4. **Складання та оформлення документації**. Основним документом для системи управління членством може стати специфікація вимог до програмного забезпечення (Software Requirements Specification, SRS), яка детально описує функціональні та нефункціональні вимоги, передбачувані сценарії використання, архітектурні рішення та технічні характеристики. Інші супровідні документи можуть включати бізнес-аналіз вимог та документ щодо забезпечення якості.
5. **Перевірка та затвердження вимог**. Всі вимоги необхідно узгодити з командою розробників і замовником, щоб уникнути непорозумінь на пізніх етапах. Це зазвичай включає обговорення вимог, отримання затверджень та офіційного затвердження документації перед початком розробки.

Розглянемо основні **типи інструментів**, які використовуються для роботи з вимогами: [30, с.454]

- **Інструменти для збору вимог –** такі інструменти, як JIRA, дозволяють зберігати зібрані вимоги від користувачів, які згодом аналізуються та трансформуються у технічні специфікації.
- **Інструменти для відстеження вимог –** такі як IBM Rational DOORS та HP ALM. Вони дозволяють відстежувати стан вимог, зміни, що до них вносяться, а також підтримують аналітиків у контролі відповідності вимог реальним потребам проєкту.
- **Інструменти для управління вимогами –** спеціалізовані платформи (наприклад, ReqIF Studio) полегшують процеси управління вимогами, включаючи їхню документацію, відстеження змін та співпрацю.

- **Інструменти для моделювання вимог –** засоби на кшталт Enterprise Architect надають можливість моделювати вимоги у вигляді діаграм і сценаріїв, що значно полегшує їх розуміння.

При виборі інструментів для управління вимогами до системи управління членством, необхідно звертати увагу на наступні критерії:

1. Сумісність з іншими інструментами проєкту – важливо, щоб обраний інструмент легко інтегрувався з іншими системами, такими як системи тестування, управління проєктами та бази даних.
2. Гнучкість налаштувань – інструмент повинен мати можливість адаптації до специфічних вимог і процесів конкретного проєкту. [29, с.365]
3. Можливості для спільної роботи – якщо команда розподілена, важливо забезпечити доступ до вимог у режимі реального часу з будь-якої точки.
4. Відповідність стандартам безпеки – обраний інструмент повинен мати належний рівень захисту даних, особливо якщо проєкт передбачає роботу з конфіденційною інформацією.

Інструменти для розробки та аналізу вимог відіграють критичну роль у забезпеченні якості та успішності проєкту системи управління членством. Вони спрощують процеси збору та аналізу вимог, забезпечують їх структуровану документацію та зменшують ймовірність виникнення помилок під час реалізації. Вибір відповідного інструмента залежить від потреб проєкту, ресурсів команди та обсягу вимог, які необхідно обробити.

1.4. Висновок до розділу 1.

У цьому розділі проаналізовано основи теорії та методології, необхідні для розробки систем управління членством. Проаналізовано деякі існуючі системи управління членством, досліджено моделі та методи управління членством. Було розглянуто технології розробки програмного забезпечення для систем управління. Важливими аспектами визначено структуру даних, безпечне зберігання та обробку інформації про користувачів. Застосування теоретичних принципів, таких як

модульність, масштабованість та ефективність, створює умови для побудови надійної системи, яка забезпечує безперервне і надійне управління членством.

РОЗДІЛ 2
МЕТОДОЛОГІЧНІ АСПЕКТИ РОЗРОБЛЕННЯ ІНТЕГРОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ

2.1. Функціональні вимоги до системи управління членством.

Функціональність програми— це здатність програмної системи (ПС) виконувати свої функції згідно з встановленими вимогами та очікуваннями користувачів в умовах її використання. Підкатегорії включають: [28, с.365]

Функціональна повнота — здатність ПС надавати весь необхідний набір функцій для вирішення визначених завдань і досягнення цілей користувача.

Точність — здатність ПС забезпечувати коректні й узгоджені результати з необхідним рівнем точності.

Здатність до взаємодії — можливість ПС взаємодіяти з однією або кількома заданими системами.

Захищеність — здатність ПС захищати інформацію від несанкціонованого доступу або змін та забезпечувати доступність даних для авторизованих осіб чи систем.

Відповідність нормативам — узгодженість роботи ПС з нормативними вимогами та стандартами. [26, с.165]

Система управління членством призначена для автоматизації та спрощення обробки членських даних. Основні функції включають обробку інформації про членів, управління їх статусом, автоматичне оновлення даних та підтримку фінансових операцій. Система повинна відповідати сучасним стандартам кібербезпеки, адже містить конфіденційну інформацію про членів організації. Додатково, система має функціонувати як гнучкий інструмент, що може легко адаптуватися до змін у структурі організації, наприклад, при створенні нових видів членства або введенні нових категорій послуг. Важливо також, щоб система могла масштабуватись разом із зростанням кількості членів та

підтримувати інтеграцію з іншими IT-системами для автоматизації процесів на рівні всієї організації. [25, с.165]

Управління профілями передбачає комплексний набір інструментів для створення, редагування та оновлення особистих даних кожного члена. Це включає не лише основні контактні дані, але й додаткові атрибути, такі як історія участі в заходах, попередні та поточні категорії членства, а також відомості про внески. Автентифікація за допомогою багатофакторного захисту підвищує безпеку системи, а для кожного профілю передбачена можливість налаштування рівнів доступу залежно від ролі користувача. Важливим аспектом є забезпечення автоматичних сповіщень про зміни статусу членства або необхідність оновлення даних, що сприяє підтриманню актуальності інформації. [27, с.165]

Управління категоріями членства. Категорії членства дозволяють організації структурувати своїх учасників за рівнями доступу до ресурсів та послуг, що надаються. Це можуть бути, наприклад, категорії "Базова", "Преміум", "VIP" або спеціальні пропозиції, такі як знижки для студентів чи пенсіонерів. Для кожної категорії важливо встановити правила та привілеї, а також передбачити можливість коригування цих умов залежно від внутрішньої політики або зовнішніх факторів. Технічно це означає наявність спеціального модулю, де адміністратори можуть змінювати, додавати чи видаляти категорії, що динамічно оновлює права членів, які до них належать. Це також передбачає автоматичний перехід між категоріями, наприклад, при закінченні терміну дії пільг. [24, с.265]

Система управління членством повинна мати розширені можливості для моніторингу та обробки платежів. Це включає інтеграцію з платіжними сервісами, такими як LiqPay, WayforPay, або інші локальні сервіси, що дозволяють членам організації здійснювати оплату онлайн. Система має підтримувати гнучкі налаштування, які дозволяють автоматично нагадувати про наближення терміну платежу, надсилати сповіщення у разі пропущених оплат або зміни статусу платежу. Важливим є збереження інформації про минулі платежі, що дає змогу створювати фінансові звіти та аналізувати стабільність фінансових надходжень від членів.

Перевірка статусу членства є необхідною функцією для забезпечення актуального відображення прав доступу членів до певних ресурсів та послуг. Система має забезпечити чітке визначення статусів (наприклад, "Активний", "Прострочений", "Тимчасово призупинений") та автоматичне оновлення цього статусу на основі регулярних правил. Крім цього, система може надсилати нагадування членам про оновлення членства та забезпечувати адміністраторам швидкий доступ до інформації про статус кожного учасника, щоб мінімізувати час на обробку запитів. [22, с.265]

Управління подіями. Для організацій, які активно проводять різноманітні заходи, функція управління подіями є ключовою. Система має забезпечити можливість організації заходів, створення календаря подій, а також ведення реєстрації учасників. Додатково важливо передбачити можливість інтеграції із зовнішніми платформами для відеоконференцій, такими як Zoom чи Microsoft Teams, для проведення онлайн-заходів. Система повинна включати функції відстеження відвідуваності та збору відгуків, що дозволяє керівникам оцінювати активність учасників і ефективність подій.

Функція комунікації є важливим інструментом для підтримання зв’язку з членами та передачі їм важливої інформації. Це включає розсилки новин, нагадування про оплату, анонси подій, а також індивідуальні повідомлення. Технічний аспект передбачає можливість інтеграції з CRM-системою або електронною поштою для автоматичної відправки повідомлень. Також можна впровадити сегментацію членів, щоб комунікація була цільовою та персоналізованою. [20, с.765]

Управління документами. Ця функція є корисною для організацій, які зберігають різноманітні документи, пов’язані з членством, такі як контракти, згоди, фінансові документи. Система повинна включати репозиторій для завантаження та зберігання документів, а також забезпечувати інструменти для організації доступу до них. Це також передбачає можливість використання цифрових підписів і забезпечення конфіденційності при обробці документів. [21, с.295]

Система повинна забезпечувати широкі можливості для створення звітів, що дозволяють аналізувати ключові показники діяльності, зокрема, приріст кількості членів, фінансові надходження, популярність подій. Функціонал звітності має включати динамічні фільтри для формування різних типів звітів, що дозволяє керівникам отримувати саме ту інформацію, яка їм потрібна. Також важливим є можливість експорту звітів у різні формати (Excel, PDF) для подальшого аналізу.

Інтеграція з іншими системами. Для більшої ефективності система управління членством повинна підтримувати інтеграцію з іншими інформаційними системами, такими як CRM, ERP або бухгалтерські програми. [19, с.285] Це дозволяє організації уникати дублювання даних та забезпечує зручний обмін інформацією між різними відділами. Технічний аспект може включати інтеграцію через API або за допомогою спеціальних плагінів для обміну даними в режимі реального часу.

2.2. Архітектура системи та вибір технічних рішень.

Архітектура системи управління членством повинна відповідати бізнес-вимогам, забезпечуючи надійний доступ користувачів до інформації, функцій та даних, пов'язаних із членством. Така система передбачає взаємодію користувачів через фронтенд, безпечну обробку запитів через бекенд, а також збереження даних в оптимізованій базі даних. [15, с.385]

Основні компоненти архітектури. Система управління членством складається з різних компонентів, кожен з яких має свої особливі функції:

Фронтенд забезпечує користувацький інтерфейс, завдяки якому користувачі можуть взаємодіяти із системою. Він повинен бути адаптивним та зручним, підтримувати інтерактивність і швидкий відгук на дії користувача.

Бекенд відповідає за обробку запитів, управління бізнес-логікою та взаємодію з базою даних. Бекенд необхідний для обробки даних та забезпечення їхньої консистентності.

База даних зберігає основні дані про користувачів і їхню активність. Зазвичай для таких цілей використовуються реляційні бази (наприклад, PostgreSQL або MySQL) або нереляційні бази даних (наприклад, MongoDB).

Система авторизації та автентифікації забезпечує надійність доступу, перевірку прав користувачів та їхню ідентифікацію. Впровадження безпечних протоколів, як-от OAuth 2.0, допомагає забезпечити захист даних. [18, с.372]

Email-розсилка використовується для автоматизованого інформування користувачів про статуси членства, оновлення та інші події.

Інтеграції дозволяють з'єднувати систему з зовнішніми платформами, такими як платіжні шлюзи (Stripe, LiqPay), CRM-системи, аналітичні сервіси тощо.

Вибір технічних рішень. Вибір технологій визначається кількома критеріями: швидкодія, масштабованість, безпека та простота інтеграції. Для системи управління членством важливими є: [17, с.472]

- **Фреймворки для фронтенду:** React.js забезпечує компонентну структуру, що спрощує тестування і підтримку коду, тоді як Vue.js відомий легкістю навчання та використання, завдяки інтуїтивному синтаксису.
- **Фреймворки для бекенду:** Node.js є популярним вибором завдяки асинхронній моделі I/O, що забезпечує швидку обробку великої кількості запитів. Django (Python) підходить для побудови більш структурованих систем з комплексними вимогами до безпеки.
- **Бази даних:** реляційні бази, як-от PostgreSQL, пропонують розширені можливості транзакційної обробки, що корисно для ведення історії членства та транзакцій. MongoDB, з його NoSQL структурою, добре підходить для зберігання даних про профілі користувачів і нефіксованих полів.
- **Безпека:** для авторизації і аутентифікації, використання JWT (JSON Web Tokens) забезпечує швидкий і захищений доступ, дозволяючи масштабувати процес авторизації без додаткових витрат на сесійну підтримку.

Фронтенд. У розробці фронтенду важливим аспектом є зручність для користувача та адаптивність. Найбільш поширені інструменти: [16, с.572]

- **React.js** дозволяє створювати інтерфейси з використанням компонентного підходу. Це полегшує повторне використання компонентів, що скорочує час розробки і підвищує надійність.

- **Vue.js** забезпечує простий API та має невеликий розмір, що позитивно впливає на швидкість завантаження. Він ідеально підходить для малих і середніх проєктів.
- **CSS-фреймворки:** використання фреймворків типу Bootstrap або TailwindCSS дозволяє швидко створювати адаптивні інтерфейси, що зручно для проєктів із динамічними інтерфейсами.

Бекенд. Бекенд складає основу всієї бізнес-логіки системи. Інструменти, що використовуються, включають:

Express.js (Node.js) забезпечує легку і швидку конфігурацію сервера та дозволяє працювати з асинхронними запитами, що корисно для реального часу.

Django пропонує вбудовану ORM (Object-Relational Mapping), що полегшує роботу з базами даних і підходить для високозавантажених систем з великим обсягом даних. [12, с.432]

GraphQL може використовуватись замість REST API для оптимізації запитів, оскільки дозволяє вибирати тільки необхідні дані, зменшуючи обсяг трафіку.

База даних. Обрання бази даних впливає на ефективність і масштабованість системи:

PostgreSQL – надійний вибір для реляційної структури, що дозволяє зберігати структуровані дані і підтримує складні запити.

MongoDB підходить для NoSQL рішень, зокрема для проєктів, де дані можуть мати змінну структуру. Завдяки горизонтальному масштабуванню MongoDB легко адаптується під зростаючу кількість даних.

Система авторизації та автентифікації. Для забезпечення високого рівня безпеки використовують: [13, с.332]

1. **OAuth 2.0 –** стандартний протокол для авторизації, який дозволяє безпечно використовувати сторонні сервіси для входу.

2. **JWT (JSON Web Tokens)** – сучасний підхід до авторизації, де токени містять всю необхідну інформацію, що дозволяє масштабувати систему без зберігання сесій на сервері.
3. **Двофакторна аутентифікація (2FA)** додає додатковий рівень безпеки, який рекомендується для захисту конфіденційних даних.

Система email-розсилки. Для автоматизації email-розсилки можна використовувати сервіси, такі як SendGrid або Mailgun. Вони дозволяють надсилати повідомлення користувачам залежно від подій, зберігаючи надійність і масштабованість розсилки.

Інтеграції. Для інтеграцій необхідно врахувати інтерфейси API, що дозволяють взаємодіяти із зовнішніми сервісами. Це може включати платіжні сервіси, сервіси аналітики або інструменти для CRM-систем. [13, с.632]

Ключові фактори, які слід враховувати при виборі. До ключових факторів належать:

- **Масштабованість** – спроможність системи до розширення.
- **Безпека** – захист даних і персональної інформації.
- **Інтеграція** – можливість підключення до зовнішніх сервісів.
- **Зручність використання** – простота для користувачів та адміністраторів.

Приклад архітектури. Ось базова схема, що ілюструє взаємодію різних компонентів системи управління членством: [14, с.532]

1. Користувач -> Фронтенд (React.js/Vue.js) -> API (GraphQL/REST)

2. API -> Бекенд (Node.js/Django) -> База даних (PostgreSQL/MongoDB)

3. Сервіс авторизації (OAuth 2.0/JWT) підключається до бекенду для верифікації доступу.

4. Інтеграції (платіжні шлюзи, аналітика) комунікують із бекендом для обробки даних та операцій.

Графічне зображення архітектури системи управління членством. Схема ілюструє основні компоненти та їх взаємодію: користувач взаємодіє з

інтерфейсом (Frontend), який через API обробляє запити у бекенді, що, в свою чергу, комунікує з базою даних та системами авторизації та інтеграцій. [10, с.533]

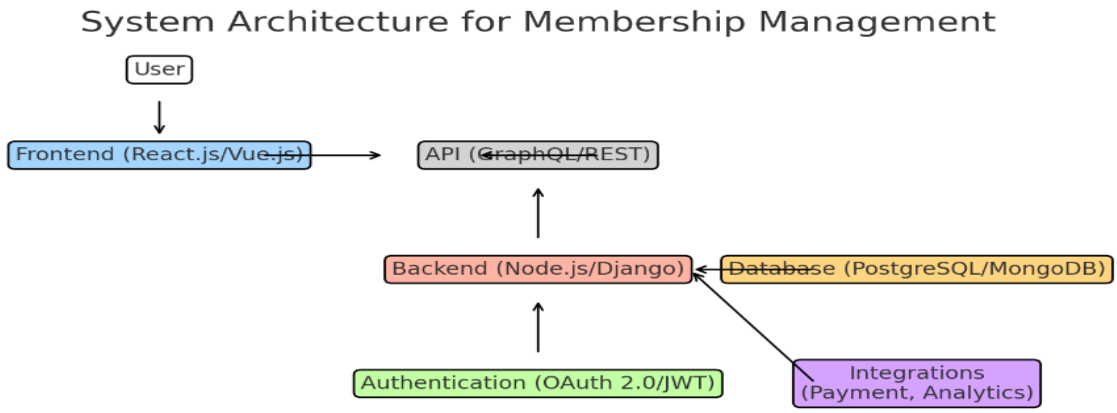

Схема 2.2. Архітектура системи управління членством

Рекомендації. Вибір технологій повинен базуватися на можливостях масштабування системи, рівні безпеки, а також вимогах до продуктивності. Рекомендується використовувати сучасні фреймворки та бази даних, які підтримують високі навантаження та мають спільноту підтримки. [9, с.533]

Система управління членством є важливим інструментом для будь-якої організації, яка прагне ефективно керувати членською базою та підвищувати якість обслуговування своїх учасників.

2.3. Тестування системи.

Тестування системи управління членством – це процес оцінки програмного забезпечення, що використовується для обробки та організації інформації про учасників певної організації або платформи. Основною метою тестування є перевірка коректності роботи всіх функцій системи, виявлення можливих помилок і забезпечення надійності та безпеки даних користувачів. У процесі тестування оцінюються такі аспекти, як функціональність системи, її продуктивність, безпека, зручність використання, а також відповідність вимогам і технічним стандартам.

Тестування системи управління членством є важливим етапом, що дозволяє виявити недоліки на ранніх стадіях і забезпечити ефективну роботу системи, що забезпечує стабільний облік і підтримку членства. [7, с.233]

Тестування управління членством включає різні види тестування.

Типи тестування

Функціональне тестування. Функціональне тестування забезпечує перевірку того, що система виконує всі задані функції відповідно до вимог. Для системи управління членством це може включати перевірку коректності реєстрації нових користувачів, функціонування системи авторизації, налаштування профілю, управління підписками, а також надання доступу до ресурсів залежно від ролі користувача. Основний метод функціонального тестування - це розробка тестових сценаріїв, які емулюють поведінку кінцевих користувачів.

Нефункціональне тестування. Нефункціональне тестування стосується властивостей системи, таких як продуктивність, безпека, масштабованість та зручність використання. Наприклад, тестування продуктивності дозволяє визначити, як система обробляє великий потік запитів під час пікових навантажень. Безпека є важливим аспектом для захисту персональних даних членів системи, що вимагає перевірки захищеності системи від потенційних вразливостей.

Інші види тестування. До інших видів тестування належать регресійне тестування, димове тестування та тестування сумісності. Вони допомагають забезпечити стабільність системи після внесення змін, перевіряючи, чи не порушують нові зміни існуючий функціонал. [5, с.273]

Регресійне тестування. Регресійне тестування є важливим для виявлення помилок, які могли бути внесені після оновлень або покращень функціональності. Впровадження регулярного регресійного тестування допомагає знизити ризик виникнення нових дефектів, особливо при великій кількості інтеграцій із зовнішніми системами або розширеннях функціональних можливостей. [6, с.273]

Димове тестування. Димове тестування є швидким тестуванням базових функцій системи після великих змін у коді, щоб пересвідчитися, що основні компоненти системи функціонують належним чином. Воно економить час, оскільки виявляє критичні проблеми, які можуть унеможливити подальше тестування.

Тестування сумісності. Система управління членством може використовуватися на різних платформах (наприклад, мобільних пристроях, настільних ПК) і у різних браузерах. Тестування сумісності забезпечує, що користувацький досвід залишатиметься стабільним незалежно від платформи або середовища використання.

Стратегія тестування.

Розробка тестових сценаріїв. Це процес, що включає створення детальних сценаріїв для покриття різних аспектів функціональності та нефункціональності системи. Наприклад, тестові сценарії можуть включати позитивні та негативні випадки, що дозволяють побачити реакцію системи на коректні та некоректні дії користувача.

Автоматизація тестів. Автоматизація скорочує час на тестування та підвищує точність перевірки функцій. Для цього можуть використовуватися спеціалізовані інструменти (наприклад, Selenium або Cypress), які дозволяють автоматизувати основні сценарії взаємодії з системою. [4, с.347]

Використання тестових даних. Ефективне тестування вимагає реалістичних тестових даних, які відтворюють типові сценарії роботи системи. Наприклад, для тестування системи управління членством це можуть бути облікові записи користувачів з різними рівнями доступу та правами.

Моніторинг результатів. Постійний моніторинг результатів тестування дозволяє вчасно виявити помилки та дотримуватись високого рівня якості. Цей етап включає аналіз виконаних тестів, відстеження тенденцій у виникненні дефектів та контроль за відновленням коректної роботи системи після виправлення помилок.

Інструменти для тестування:

- **Selenium -** потужний інструмент для автоматизації веб-тестування, що дозволяє записувати сценарії взаємодії користувачів з системою та відтворювати їх. [4, с.547]

- **Cypress** - популярний інструмент для тестування JavaScript-додатків, що забезпечує швидке налаштування та зручний інтерфейс для проведення тестування інтерфейсу.
- **Postman** - широко використовується для тестування API, що є важливим для перевірки роботи бекенд-сервісів системи.
- **JMeter** - застосовується для тестування продуктивності, дозволяє симулювати навантаження на систему та оцінювати її стабільність при великій кількості запитів.
- **Burp Suite** - один із основних інструментів для забезпечення безпеки, дозволяє перевіряти систему на вразливості до атак, таких як SQL-ін'єкції чи міжсайтовий скриптинг.

Висновок. Тестування є критично важливим етапом у процесі розробки та підтримки системи управління членством. Використання різноманітних видів тестування, стратегій та інструментів дозволяє забезпечити високу якість і надійність системи. Виконання функціонального і нефункціонального тестування, а також автоматизація процесів сприяють швидкому виявленню та усуненню помилок, покращують користувацький досвід і гарантують стабільність системи навіть при високих навантаженнях.

2.4. Висновок до розділу 2.

У розділі було проаналізовано основні підходи та принципи, що формують методологічну основу для розроблення інтегрованої системи управління членством. Дослідження показало, що успішне впровадження такої системи потребує комплексного підходу, який поєднує аналіз потреб користувачів, вибір сучасної технологічної інфраструктури та забезпечення захищеності даних. Використання модульної архітектури та гнучких методів розробки дозволяє підвищити адаптивність системи до змінних вимог та забезпечити її масштабованість. Таким чином, розроблена методологія є основою для створення ефективної, безпечної та надійної системи, що відповідає сучасним вимогам управління членськими відносинами.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНТЕГРОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ НА PHP

3.1. Вибір інструментів для розробки та аналізу вимог до системи.

Для успішного вибору інструментів для аналізу та управління вимогами в системі управління членством важливо зрозуміти потреби проєкту. Це означає детальне вивчення цілей, завдань, масштабів, а також технічних та бізнес-вимог системи. [3, с.446] Система управління членством часто потребує гнучкого підходу до розробки, оскільки функціональність може включати керування обліковими записами користувачів, авторизацію, обробку платежів, сегментацію користувачів та інтеграцію з іншими системами.

При виборі інструментів для управління вимогами до системи управління членством доцільно керуватися такими критеріями: [33, с.346]

Функціональні можливості. Інструмент повинен забезпечувати можливість документування вимог, їх аналізу, відстеження змін і забезпечення відповідності функціональних потреб системи. Найважливіші функції включають створення ієрархії вимог, обробку конфліктів, аналіз впливу змін, а також інструменти для відстеження виконання вимог на кожному етапі життєвого циклу проєкту.

Простота інтеграції. Інструмент має легко інтегруватися з іншими програмами, які використовуються для розробки та тестування системи управління членством (наприклад, з системами для управління проєктами або для підтримки циклу розробки програмного забезпечення, такими як Jira чи Confluence).

Користувацький інтерфейс та зручність використання. Користувачі повинні швидко освоювати інструмент та зручно працювати з його функціоналом без необхідності проходити тривале навчання. [37, с.646]

Можливості командної роботи. Інструмент повинен підтримувати колаборативне середовище, надаючи можливості для обміну коментарями,

розподілу ролей та прав доступу для різних учасників команди, включаючи розробників, аналітиків та менеджерів проєкту.

Вартість. Особливо важливо для проєктів з обмеженим бюджетом; бажано обрати рішення, яке відповідає фінансовим можливостям організації, враховуючи можливість довготривалої підтримки та оновлень інструменту.

Враховуючи попередні критерії, важливо розглянути практичні аспекти використання таких інструментів: [40, с.426]

Підтримка вимог до якості та надійності. Інструменти повинні дозволяти аналізувати вимоги щодо надійності системи, наприклад, обмеження доступу, шифрування даних для безпеки персональної інформації членів.

Гнучкість та адаптивність до змін. Системи управління членством часто змінюються відповідно до потреб організації або регуляторних вимог, тому інструменти повинні дозволяти швидке внесення змін без втрати узгодженості в документації вимог.

Можливості для автоматизації. Інструменти, що підтримують автоматизацію, дозволяють зменшити кількість рутинних операцій. Наприклад, інструменти на основі штучного інтелекту можуть автоматично перевіряти вимоги на відповідність формату чи виявляти потенційні конфлікти. [35, с.426]

Порівняння популярних інструментів для управління вимогами до систем управління членством. Для систем управління членством поширено використовувати такі інструменти:

Jama Software. Платформа забезпечує управління вимогами, підтримку рецензій, а також можливості для візуалізації залежностей. Ідеально підходить для проєктів, що потребують високої прозорості у відстеженні вимог.

IBM Engineering Requirements Management DOORS Next. Інструмент пропонує потужну інтеграцію з іншими продуктами IBM, що корисно для великих організацій, яким потрібна точність та повна прозорість у процесі розробки. [23, с.426]

Jira з Confluence. Jira популярна завдяки своїй гнучкості та можливості налаштування. У поєднанні з Confluence цей інструмент дозволяє детально

документувати вимоги та слідкувати за процесом їх виконання, забезпечуючи прозорість в команді.

ReqView. Більш спеціалізований інструмент для документування та відстеження вимог з інтегрованою підтримкою слідування, що може бути корисним для малих команд чи стартапів.

Аналіз вибору інструменту для управління вимогами до системи управління членством на прикладі. Для детального аналізу вибору інструменту управління вимогами до системи управління членством, варто звернути увагу на конкретний приклад використання Jira з Confluence. [27, с.506] Ці інструменти від Atlassian є популярними завдяки своїй гнучкості, адаптивності до змін та здатності інтегруватися з іншими компонентами розробницької екосистеми. У цьому прикладі розглянемо, як Jira та Confluence можуть оптимізувати процес управління вимогами до системи членства в організації, що прагне прозорості, безпеки та спрощеного доступу для своєї команди.

Jira – це система управління проєктами з акцентом на задачі, що дозволяє організувати та відслідковувати процеси розробки. У випадку з системою управління членством, Jira надає можливість детально документувати та класифікувати вимоги, починаючи від початкової ідеї до моменту впровадження. Особливо важливо, що Jira підтримує гнучкі методології розробки (Agile, Scrum), що дозволяє швидко адаптуватися до змін у вимогах або пріоритетах.

У Jira кожна вимога може бути представлена у вигляді "Epic" (для великих елементів) або "User Story" (для менших завдань), що дозволяє структуровано розділяти завдання. Наприклад, у проєкті системи управління членством великі функціональні вимоги, такі як авторизація користувачів, надання доступу до персональних даних членів чи система сповіщень, можна організувати у вигляді окремих "Epic". Це забезпечує розподіл відповідальності між учасниками проєкту та дозволяє команді зосередитись на конкретних функціях, зменшуючи ризик упущення деталей. [26, с.306]

Функції відстеження змін у Jira дозволяють команді контролювати історію змін, виконаних над вимогами, що особливо важливо для системи управління членством, де безпека та узгодженість вимог до конфіденційності є критичними.

Використання Confluence для документування вимог та підтримки **командної співпраці**. **Confluence** – інструмент для документування та зберігання інформації, який інтегрується з Jira, забезпечуючи єдину платформу для спільної роботи. У Confluence можна створити базу знань, яка дозволяє команді зберігати всі деталі щодо вимог, включаючи специфікації, шаблони та діаграми. Для системи управління членством це означає, що всі учасники проєкту, від менеджерів до розробників, мають доступ до актуальних і чітко документованих вимог. [23, с.206]

Інтерфейс Confluence дозволяє легко організувати документацію за допомогою сторінок і розділів, що дає можливість деталізувати кожну вимогу. Наприклад, для вимоги "Система авторизації" можна створити окрему сторінку, на якій зберігатиметься опис функціоналу, бізнес-логіка, сценарії використання та навіть можливі ризики. Confluence підтримує спільну роботу над документами в режимі реального часу, що дозволяє аналітикам, менеджерам та розробникам коментувати і уточнювати вимоги без необхідності окремих зустрічей.

Практичне застосування: сценарій використання для системи управління членством.

1. **Визначення основних вимог у Jira**. Розробка системи управління членством починається зі створення епік для основних компонентів, таких як модуль реєстрації користувачів, модуль підписок, модуль доступу до персональних даних тощо. Кожен з цих епік може містити окремі завдання, які деталізують підфункції, наприклад, "Підтримка багатоетапної авторизації", "Автоматичне сповіщення про закінчення підписки". [21, с.301]
2. **Зберігання специфікацій у Confluence**. Для кожного епік у Jira створюється відповідна сторінка у Confluence, де деталізуються вимоги до цього компонента. Наприклад, для модуля реєстрації користувачів можна

зберігати специфікацію, що включає описи полів форми, вимоги до валідації даних, специфікацію безпеки та сценарії тестування.

3. **Відстеження змін та сповіщення**. При внесенні змін до вимог, наприклад, додавання нового параметра у форму реєстрації, учасники проєкту отримують сповіщення. Jira автоматично оновлює завдання, пов’язані з новими вимогами, а Confluence дозволяє команді слідкувати за всіма оновленнями у відповідній документації.
4. **Інтеграція з іншими системами**. Важливою особливістю є можливість інтеграції Jira з іншими сервісами, такими як Bitbucket для керування версіями коду, що дозволяє відстежувати зміни коду, пов'язані з конкретними вимогами, забезпечуючи зв’язок між вимогами та їх реалізацією.

Переваги вибору Jira з Confluence [32, с.351]

- **Централізоване управління вимогами**. Наявність всіх вимог, документів та завдань в одній екосистемі дозволяє знизити ймовірність комунікаційних збоїв та втрат інформації.
- **Гнучка адаптація**. Можливість швидко вносити зміни та інтегрувати їх у процес розробки дозволяє адаптуватися до змін у вимогах, що особливо важливо для проєктів, де вимоги часто змінюються.
- **Простота у використанні та широкі можливості для кастомізації**. Інтерфейс Jira та Confluence дозволяє легко налаштовувати систему під специфічні потреби проєкту, при цьому забезпечуючи легкість навчання для нових членів команди.
- **Високий рівень безпеки**. Для проєктів з чутливими даними про членів, система Atlassian підтримує багатофакторну автентифікацію, контролі доступу та захищене зберігання даних.

Недоліки та обмеження. Попри значні переваги, Jira та Confluence мають деякі обмеження, зокрема: [39, с.352]

- **Вартість ліцензії** може бути високою для малих команд.

- **Складність налаштування** для користувачів, які не мають досвіду з інструментами Atlassian, може потребувати додаткового навчання.
- **Перевантаження функціоналом** може призводити до складнощів у навігації для простіших проєктів.

Вибір Jira з Confluence в ролі інструментів для управління вимогами до системи управління членством дозволяє забезпечити прозорість, злагодженість та централізований контроль над вимогами. Це рішення ідеально підходить для організацій, які потребують надійної платформи для зберігання вимог і можливості спільної роботи над проєктом. [37, с.452] Завдяки інтеграції з іншими інструментами розробки, можливостям налаштування та гнучкості, даний комплекс забезпечує ефективну підтримку на всіх етапах життєвого циклу проєкту.

3.2. Програмна реалізація функцій відстеження статусу членства.

Розуміння завдання. Мета цього завдання полягає в розробці програмного забезпечення, яке дозволяє керувати статусами членства користувачів у системі, відстежувати періоди активності та неактивності підписки, а також своєчасно нагадувати про потребу поновлення членства. Реалізація такого функціоналу вимагає обробки даних користувачів та їх підписок, а також інтеграції механізмів, що дозволять надсилати автоматизовані повідомлення для підвищення лояльності та утримання користувачів. Ключовими завданнями є забезпечення надійності зберігання інформації, захисту персональних даних, та надання інтуїтивно зрозумілого інтерфейсу для адміністрації. [34, с.257]

Основні функціональні вимоги. Функціональні вимоги для системи управління членством включають:

- **Створення і редагування профілів користувачів**, включаючи статус підписки, історію операцій, дату закінчення підписки;
- **Моніторинг та оновлення статусу членства** – система повинна автоматично змінювати статус користувачів відповідно до умов підписки;
- **Надсилання повідомлень про оновлення членства –** система повинна автоматично генерувати нагадування про закінчення терміну дії підписки;

- **Адміністративна панель** для управління інформацією про користувачів та перегляду аналітики членства;
- **Безпечне зберігання даних** та захист особистої інформації відповідно до вимог безпеки.

Технологічний стек. Для побудови такої системи можна використовувати наступний технологічний стек: [39, с.257]

Frontend: HTML, CSS, JavaScript (Frameworks: React, Vue.js або Angular).

Backend: Python з Django або PHP з Laravel для побудови надійного API.

База даних: PostgreSQL, MySQL або MongoDB для зберігання даних про користувачів та статус підписок.

Інструменти автоматизації: Celery (з Redis для Python) для організації автоматизованих повідомлень, або Cron для PHP.

Безпека: JWT (JSON Web Tokens) для автентифікації користувачів та SSL для шифрування даних.

Технічний аналіз системи управління членством UVS. Система UVS являє собою спеціалізовану систему управління членством з набором власноруч розроблених функцій, що дозволяє автоматизувати всі аспекти взаємодії з учасниками асоціації, зокрема: [40, с.255]

Реєстрація членів та зберігання їх даних. Система має модуль реєстрації, який забезпечує створення профілю для кожного учасника. Цей профіль містить ключову інформацію про статус, права та активність членства.

Керування статусом членства. UVS відстежує статуси учасників, що включає поточний статус, дати поновлення членства, оплату внесків та нагадування про оновлення.

Взаємодія з членами. Через особистий кабінет система надає можливість прямої комунікації з учасниками, повідомлення про важливі новини, події, а також інтеграцію зі сторонніми платформами.

Інтеграція з платіжними системами. Кастомний функціонал забезпечує можливість приймання онлайн-платежів через інтеграцію з популярними платіжними сервісами, що спрощує оплату членських внесків. [27, с.355]

Frontend, Backend, база даних, інструменти автоматизації та безпека у системі UVS.

- **Frontend:** Система використовує сучасні вебтехнології, як-от HTML5, CSS3 та JavaScript (з фреймворками типу Vue.js або React для покращення інтерфейсу). Завдяки адаптивному дизайну сайт зручний для користувачів як на комп'ютерах, так і на мобільних пристроях.
- **Backend:** Основний функціонал розроблено на PHP у поєднанні з фреймворком Laravel. Використання Laravel забезпечує чітку структуризацію коду, легку масштабованість та високу продуктивність. Завдяки REST API забезпечується зручність інтеграції з іншими платформами.
- **База даних.** UVS використовує реляційну базу даних MySQL. Ця база даних підходить для обробки великих обсягів інформації, що дозволяє ефективно управляти даними членів, зберігаючи їх безпечно та забезпечуючи швидкий доступ до інформації.
- **Інструменти автоматизації.** Система реалізує автоматизоване оновлення статусів членів, що базується на сценаріях CRON, автоматичні нагадування про необхідність поновлення членства, повідомлення про закінчення терміну членства, а також інтеграцію з електронними поштовими сервісами для розсилок.
- **Безпека.** UVS включає захист даних через SSL-шифрування для всіх транзакцій і авторизаційних сесій. Крім того, для забезпечення додаткової безпеки впроваджено двофакторну аутентифікацію, а також захист від SQL-ін’єкцій, Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Функції відстеження статусу членства у системі. UVS містить ефективну систему відстеження статусу членства, яка включає: [26, с.325]

1. **Моніторинг та оновлення статусу.** Система автоматично змінює статус членства відповідно до активності користувача: від оновлення термінів членства до нагадувань про оплату.
2. **Нагадування про оновлення статусу.** Завдяки інтеграції з електронними поштовими сервісами система надсилає сповіщення учасникам про необхідність поновлення членства, вчасно повідомляє про його завершення.
3. **Аналітика та звітність.** UVS дозволяє генерувати звіти по статусах членства, відстежувати зміни в обсягах та динаміці членства, що сприяє більш якісному стратегічному управлінню.

Приклад реалізації на Python з використанням Django функції відстеження статусу членства у системі

```
from django.db import models
from django.utils import timezone
from django.core.mail import send_mail

class Membership(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    start_date = models.DateTimeField(default=timezone.now)
    end_date = models.DateTimeField()
    status = models.CharField(max_length=20, choices=(('active', 'Active'), ('expired', 'Expired')))

    def check_status(self):
        if self.end_date < timezone.now():
            self.status = 'expired'
            self.save()
            send_mail(
```

```
            'Your membership has expired',
            'Please renew your membership.',
            'admin@site.com',
            [self.user.email],
        )
```

Пояснення коду. Цей код є частиною програми, написаної з використанням веб-фреймворку Django на Python. Він реалізує базову модель членства користувача (клас Membership) та містить метод для перевірки статусу членства. Далі наведено детальний опис кожної складової коду. [23, с.425]

Імпорт необхідних модулів

python

```
from django.db import models
from django.utils import timezone
from django.core.mail import send_mail
```

1. django.db import models — Модуль Django models містить основні класи для створення моделей у базі даних. Модель (Model) — це клас, що визначає структуру та поведінку об'єктів бази даних.

2. django.utils import timezone — Імпорт об'єкта timezone, який використовується для роботи з датою та часом з урахуванням часових поясів. Це корисно для відображення часу відповідно до налаштувань сервера або користувача.

3. django.core.mail import send_mail — Імпорт функції send_mail, яка дозволяє надсилати електронні листи безпосередньо з коду.

Опис класу Membership

python

```
class Membership(models.Model):
```

Клас Membership є моделлю Django, що наслідує від базового класу models.Model, дозволяючи створювати таблицю в базі даних із відповідними полями.

Поля моделі Membership

1. user — це поле представляє відношення багато до одного (foreign key) з іншим класом User , що передбачає існування об’єкта користувача для кожного членства: [23, с.625]

python

user = models.ForeignKey(User, on_delete=models.CASCADE)

- ForeignKey встановлює зв’язок з моделлю User.
- on_delete=models.CASCADE означає, що при видаленні користувача, пов'язані з ним записи членства також будуть видалені.

2. start_date — Це поле зберігає дату початку членства:

python

start_date = models.DateTimeField(default=timezone.now)

- DateTimeField означає, що поле приймає як дату, так і час.
- default=timezone.now автоматично встановлює поточну дату і час під час створення нового запису.

3. end_date — Це поле представляє дату закінчення членства:

python

end_date = models.DateTimeField()

- DateTimeField без параметра default, тобто значення має бути встановлене вручну під час створення об'єкта.

4. status — Поле, яке зберігає стан членства, визначений як рядок фіксованої довжини:

python

status = models.CharField(max_length=20, choices=(('active', 'Active'), ('expired', 'Expired')))

- CharField з обмеженням max_length=20 означає, що поле може містити до 20 символів.
- choices обмежує можливі значення для цього поля: active або expired, забезпечуючи контроль над даними. [24, с.627]

Метод check_status

python

```
def check_status(self):
    if self.end_date < timezone.now():
        self.status = 'expired'
        self.save()
        send_mail(
            'Your membership has expired',
            'Please renew your membership.',
            'admin@site.com',
            [self.user.email],
        )
```

Метод check_status перевіряє, чи не завершився термін членства користувача, і за потреби оновлює статус членства та надсилає повідомлення користувачеві. [24, с.727]

1. Перевірка дати закінчення:

python

```
if self.end_date < timezone.now():
```

Якщо значення end_date менше за поточну дату і час (timezone.now()), це означає, що членство завершено.

2. Оновлення статусу:

```
python
self.status = 'expired'
self.save()
```

Статус змінюється на expired , і об'єкт зберігається в базі даних для збереження змін.

3. Відправка електронного листа:

```
python
  send_mail(
     'Your membership has expired',
     'Please renew your membership.',
     'admin@site.com',
     [self.user.email],
  )
```

Якщо членство завершилось, користувачу надсилається лист із вказаною темою та текстом, що спонукає його оновити своє членство.

Приклад реалізації на **PHP** (код) функції відстеження статусу членства у системі та пояснення коду дивіться у додатку Б.

Цей код дозволяє автоматично відстежувати стан членства користувача та виконувати необхідні дії при завершенні членства, такі як оновлення статусу та надсилання повідомлення. [22, с.627]

Програмна реалізація системи управління членством включає багато компонентів, таких як-от фонові задачі, автоматичне оновлення статусів, та забезпечення безпеки даних. Обираючи відповідний стек технологій і архітектуру, можна створити систему, яка сприятиме ефективному управлінню та покращенню взаємодії з користувачами.

3.3. Інтеграція платіжних платформ для здійснення онлайн-оплати.

Інтеграція платіжних платформ у систему управління членством UVS (Ukranian Vascular Society — Українська судинна спільнота) є дуже важливою для

забезпечення зручності оплати і автоматизації фінансових операцій. Основні завдання інтеграції платіжної платформи включають: [20, с.427]

- автоматизацію обробки транзакцій,
- забезпечення безпеки користувачів,
- легкий процес оплати, що оптимізує взаємодію користувача з системою.

Процес інтеграції з UVS орієнтований на підтримку онлайн-платформ для оплати членських внесків, додаткових послуг та інших транзакцій.

LiqPay є однією з найпопулярніших платіжних систем в Україні, що активно використовується для здійснення онлайн-платежів. Впровадження LiqPay в систему управління членством UVS дозволяє автоматизувати процес збору членських внесків, пожертвувань та інших фінансових транзакцій, пов'язаних з діяльністю організацій.

Платіжна платформа LiqPay є одним із найзручніших рішень для UVS, оскільки забезпечує миттєву оплату та підтримує різні методи переказів, включно з картками Visa та Mastercard, гаманцями, Apple Pay та Google Pay. Переваги інтеграції LiqPay: [17, с.327]

1. **Миттєва обробка платежів –** транзакції проходять за лічені секунди, що дозволяє членам організації швидко активувати послуги.
2. **Зручність для користувачів.** Члени організації мають можливість здійснювати оплату членських внесків безпосередньо через особистий кабінет, використовуючи різні платіжні методи, такі як банківські картки, мобільні платежі та інші.
3. **Проста інтеграція API –** UVS може швидко впровадити LiqPay за допомогою REST API, яке дозволяє контролювати процес оплати і моніторити успішність транзакцій.
4. **Мобільна сумісність** – LiqPay добре адаптується для мобільних пристроїв, що особливо актуально для користувачів, які здійснюють оплату з мобільних додатків.

5. **Автоматизація процесів.** Завдяки інтеграції з LiqPay, система автоматично створює рахунки, надсилає нагадування про оплату, здійснює облік платежів та генерує звітність.

Особливості інтеграції LiqPay у систему управління членством. [26, с.437]

a. **Створення платіжних шлюзів.** Для ефективної інтеграції LiqPay в систему управління членством необхідно налаштувати платіжні шлюзи, що забезпечують з'єднання вашої платформи з LiqPay.
b. **Налаштування типів платежів.** Система дозволяє конфігурувати різноманітні платіжні опції, включаючи одноразові платежі, підписки та пожертвування.
c. **Генерація рахунків.** Платформа автоматично створює рахунки для кожного платежу, що включають всю необхідну інформацію, зокрема суму та деталі транзакцій.
d. **Інтеграція з особистими кабінетами членів.** Кожен учасник організації отримує доступ до персонального кабінету для перегляду історії платежів, оплати рахунків та управління підписками.
e. **Отримання сповіщень.** Система автоматично надсилає повідомлення про успішні та неуспішні транзакції, що дозволяє швидко реагувати на зміни у фінансових операціях.
f. **Звіти та аналітика.** Інтеграція з LiqPay забезпечує можливість отримання детальних фінансових звітів, що сприяє ефективному аналізу фінансової діяльності організації.

Реєстрація у системі управління членством UVS.

Реєстрація у UVS спрямована на спрощення входу нових членів у систему.

Процес реєстрації у системі UVS досить простий та не забирає багато часу. Користувач вводить базові дані для створення профілю, після чого отримує доступ до функцій системи, включно з оплатою. Ключовий аспект – це захист даних і мінімізація кроків реєстрації, що знижує ризик втрати інтересу з боку потенційних учасників. [21, с.537]

Розглянемо хід реєстрації детальніше.

1. Завантажити сайт системи управління членством UVS.

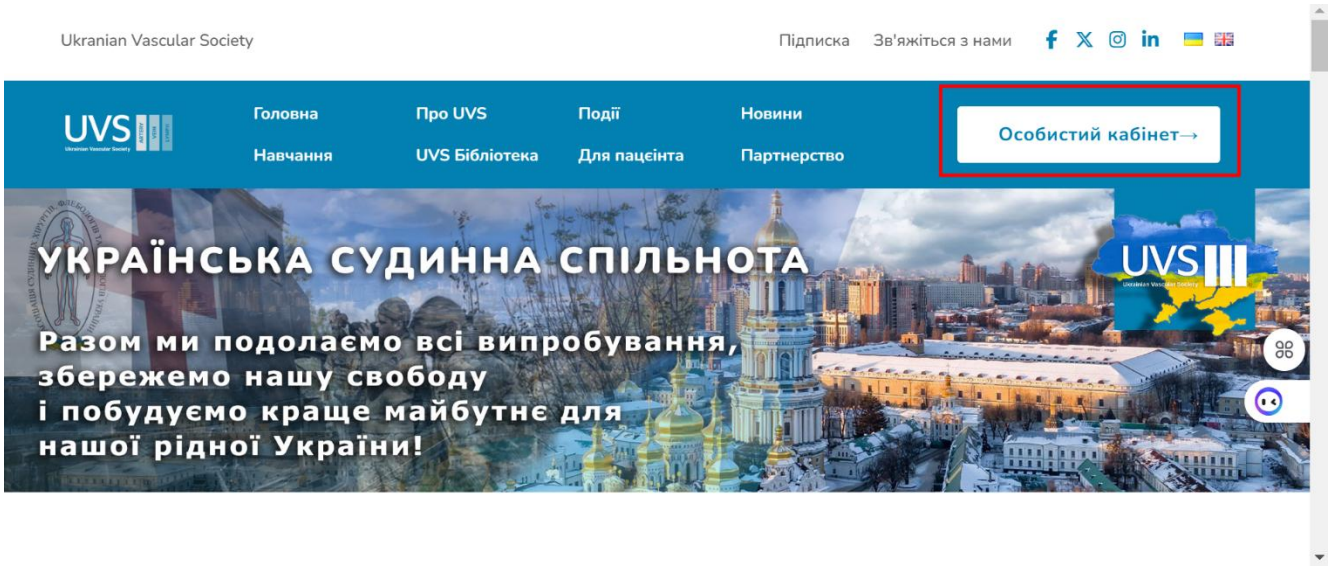

Рисунок 3.1.

2. Натиснути на кнопку Особистий кабінет, яка розміщена справа вгорі. Після цього відкриється сторінка сайту UVS, на якій є дві форми: форма входу на сайт зліва та форма для реєстрації справа. Потрібно у форму для реєстрації на сайті системи UVS увести адресу своєї електронної пошти та придумати і ввести надійний пароль, а тоді натиснути на кнопку Реєстрація.

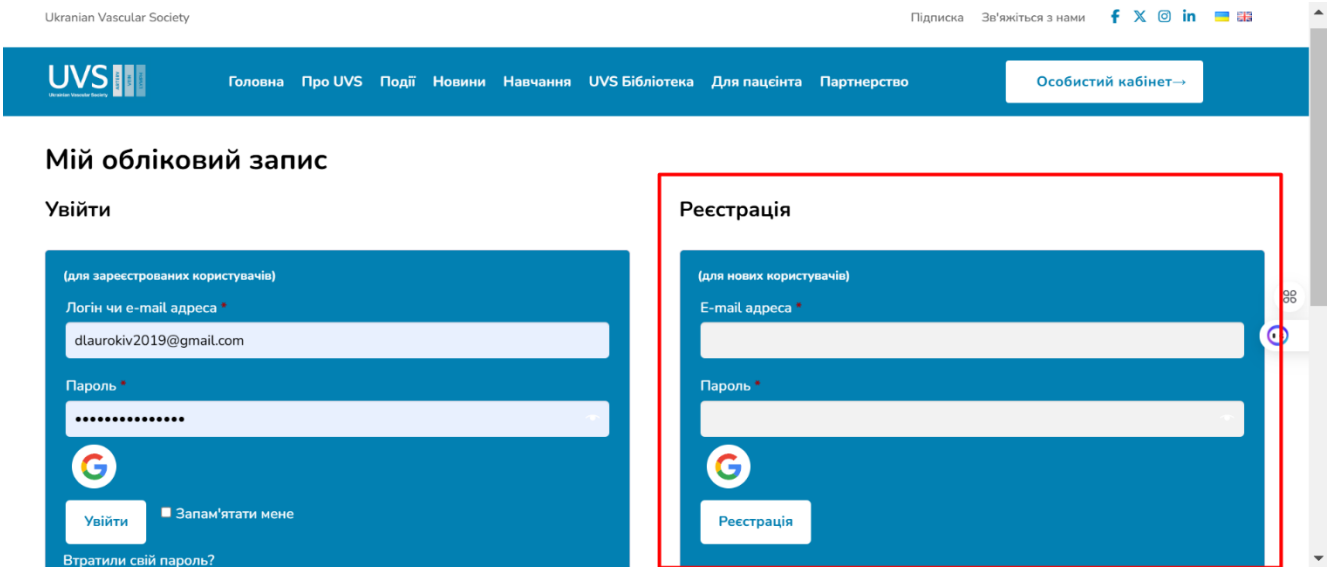

Рисунок 3.2.

3. Вгорі посередині нового вікна форми реєстрації на сайті системи UVS потрібно завантажити свою фотографію на аватар.

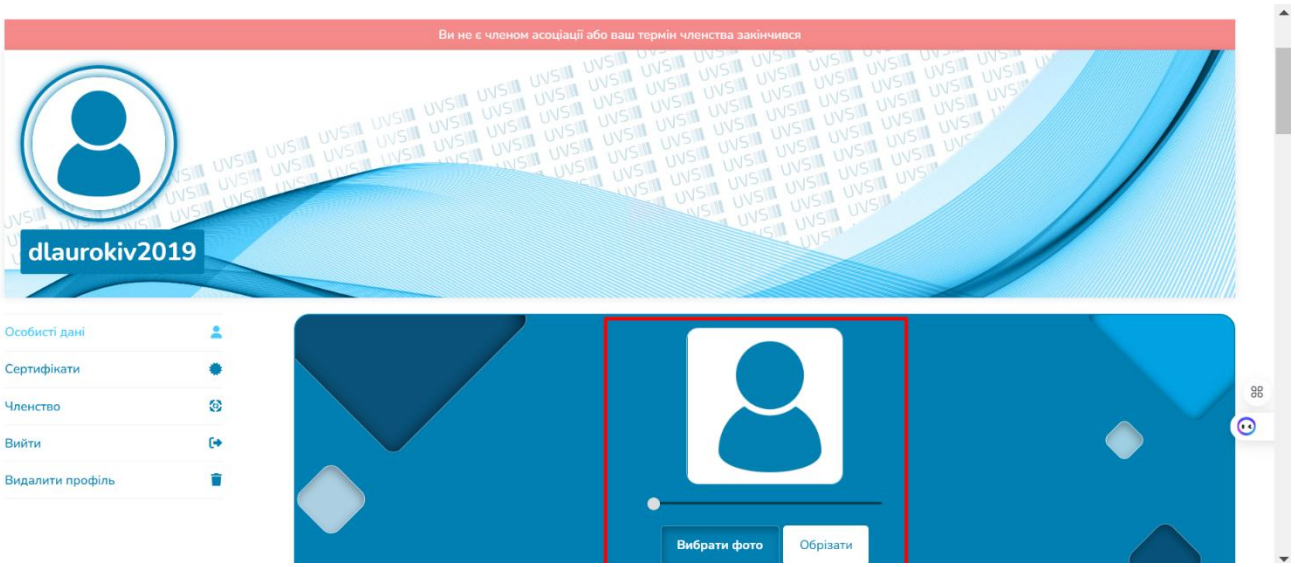

Рисунок 3.3.

4. У новому вікні форми реєстрації на сайті системи необхідно увести наступну інформацію: своє прізвище та ім’я, відображуване ім’я (нікнейм), Email, дату народження, дату народження, номер телефону, країну та місто проживання, місце роботи лікарем, займану посаду на теперішній час, кваліфікаційну категорію, науковий ступінь, спеціалізацію, за наявності і другу спеціалізацію вказати, початок роботи, вставити посилання на соціальні мережі та інші ресурси, увести основну інформацію про себе, максимум 5000 символів, за бажанням змінити пароль та натиснути кнопку Зберегти.

Рисунок 3.4.

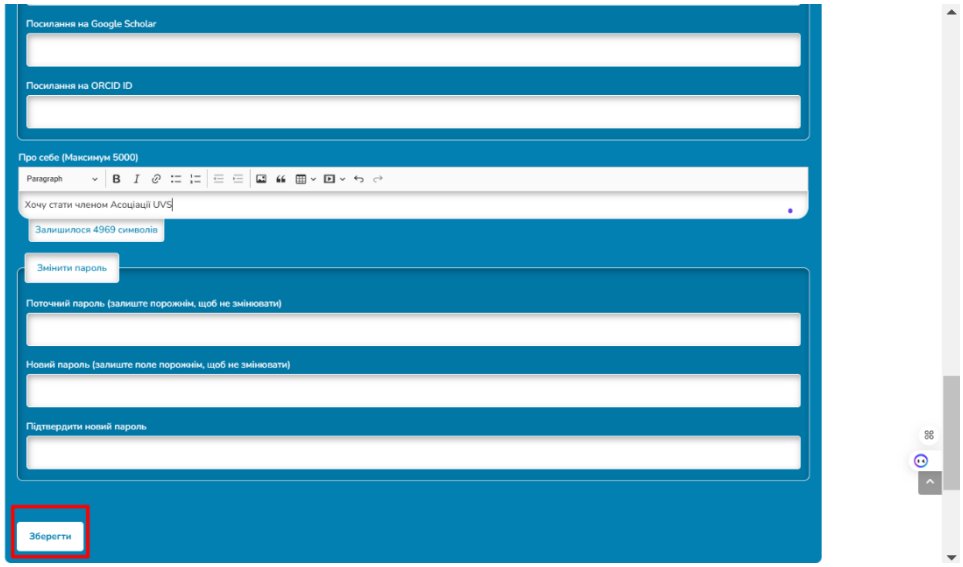

Рисунок 3.5.

5. Після перелічених вище дій можна користуватися усіма функціями особистого кабінету.

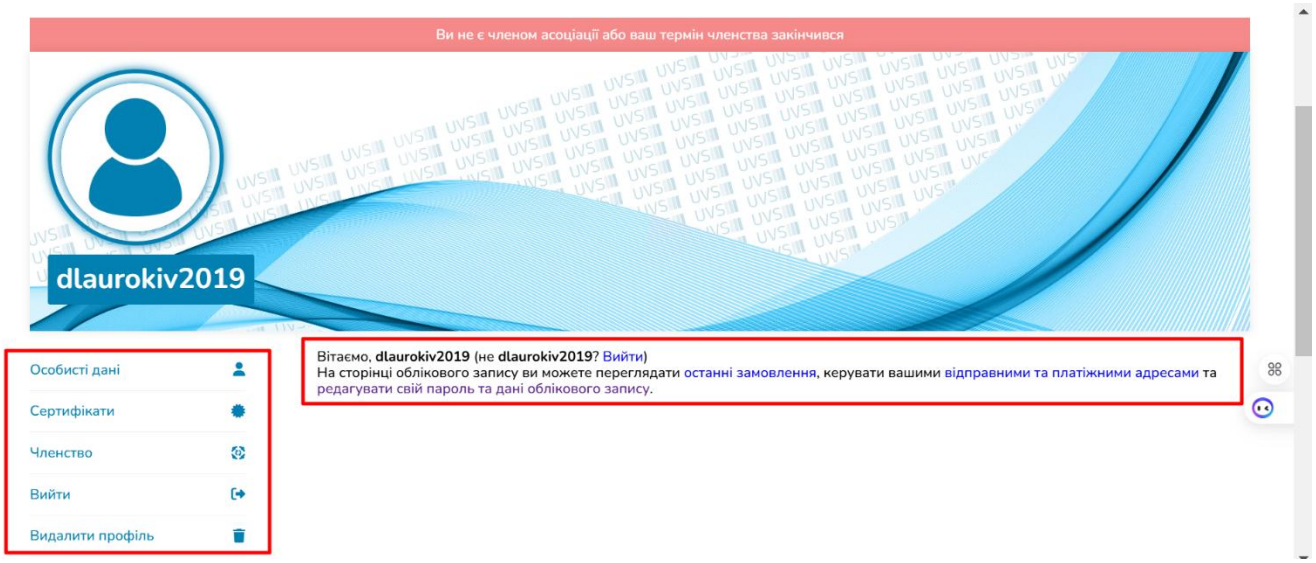

Рисунок 3.6.

Функціонування особистого кабінету у системі управління членством UVS. Вважаємо доцільним зазначити, що функціональні можливості системи управління членством UVS були значно розширені та вдосконалені завдяки впровадженню індивідуально розроблених модулів на PHP. Ці модулі створені з урахуванням унікальних вимог проєкту, що дозволило забезпечити їхню оптимальну інтеграцію та ефективну адаптацію до специфіки діяльності системи.

На відміну від готових платформ чи CMS, такий функціонал не має обмежень шаблонів та стандартних модулів. [29, с.737] Ілюстрація функціоналу кабінету представлена у додатку В.

PHP - одна з найпопулярніших мов для веб-розробки. Велика кількість хостингів підтримує PHP за замовчуванням. PHP дозволяє створювати рішення будь-якої складності, від простих сайтів-візиток до складних веб-додатків. Сучасні версії PHP демонструють високу продуктивність, що важливо для ресурсомістких сайтів. Існує безліч фреймворків (Laravel, Symfony, Yii2), бібліотек та інструментів, які значно спрощують розробку на PHP. PHP легко інтегрується з різними системами управління базами даних MySQL, PostgreSQL, що дозволяє створювати динамічні та масштабовані веб-додатки. [28, с.423]

Переваги кастомного функціоналу.

Індивідуальність. Розробка виконується під специфічні вимоги компанії, що дозволяє створити унікальний продукт.

Оптимізація. Код максимально адаптується до виконання конкретних завдань, що підвищує його продуктивність.

Безпека. Можливість впровадження власних механізмів захисту, що мінімізує ризики вразливостей.

Масштабованість. Система легко адаптується до зростаючих вимог бізнесу завдяки можливості додавати нові функції.

Незалежність. Відсутність залежності від сторонніх рішень дає змогу впроваджувати інновації та проводити експерименти без обмежень.

Можливості розширеного функціоналу на PHP. [25, с.323]

Динамічний контент. Генерація сторінок на основі даних з бази, що забезпечує оновлення інформації в реальному часі.

Інтерактивні елементи. Реалізація форм, калькуляторів, чатів, галерей та інших інструментів для взаємодії з користувачем.

Персоналізація. Забезпечення індивідуального підходу до кожного користувача.

Інтеграція. Підключення платіжних сервісів, аналітичних інструментів, соціальних мереж та інших зовнішніх систем.

Розробка API. Створення інтерфейсів для інтеграції з мобільними додатками та іншими платформами.

Особистий кабінет у UVS є центральним елементом взаємодії з користувачем, що включає:

Особисті дані – користувачі можуть переглядати свої дані, змінювати параметри профілю та керувати налаштуваннями доступу.

Сертифікати – учасники бачать інформацію про всі наявні в них сертифікати.

Членство – у кабінеті доступний розділ для швидкої оплати членських внесків, що дозволяє користувачам здійснювати транзакції без додаткових зусиль.

Вийти – вихід з особистого кабінету.

Видалити профіль – видалення власного профілю.

Інтерфейс особистого кабінету зрозумілий та простий у користуванні. Для вибору потрібної функції кабінету, достатньо скористатися меню, яке розташоване зліва внизу. [40, с.321]

Онлайн-оплата у системі управління членством. UVS пропонує функцію онлайн-оплати, яка працює в декілька етапів:

1. Перевірка статусу користувача – система аналізує, чи має користувач активний профіль і чи виконані всі умови для здійснення оплати.

2. Вибір платіжної системи – користувач вибирає з підтримуваних платформ (наприклад, LiqPay), яка є зручною для його потреб.

3. Підтвердження оплати – після вибору платіжної системи користувач підтверджує платіж, що автоматично активує відповідні функції в особистому кабінеті.

Механізм онлайн-оплати у системі UVS доволі простий та інтуїтивно зрозумілий. Зупинимося на оплаті детальніше. [23, с.321]

1. У меню особистого кабінету потрібно обрати Членство. Якщо людина ще не є членом асоціації **UVS,** то у розділі Членство появиться активна кнопка оплатити членство. Потрібно на неї натиснути.

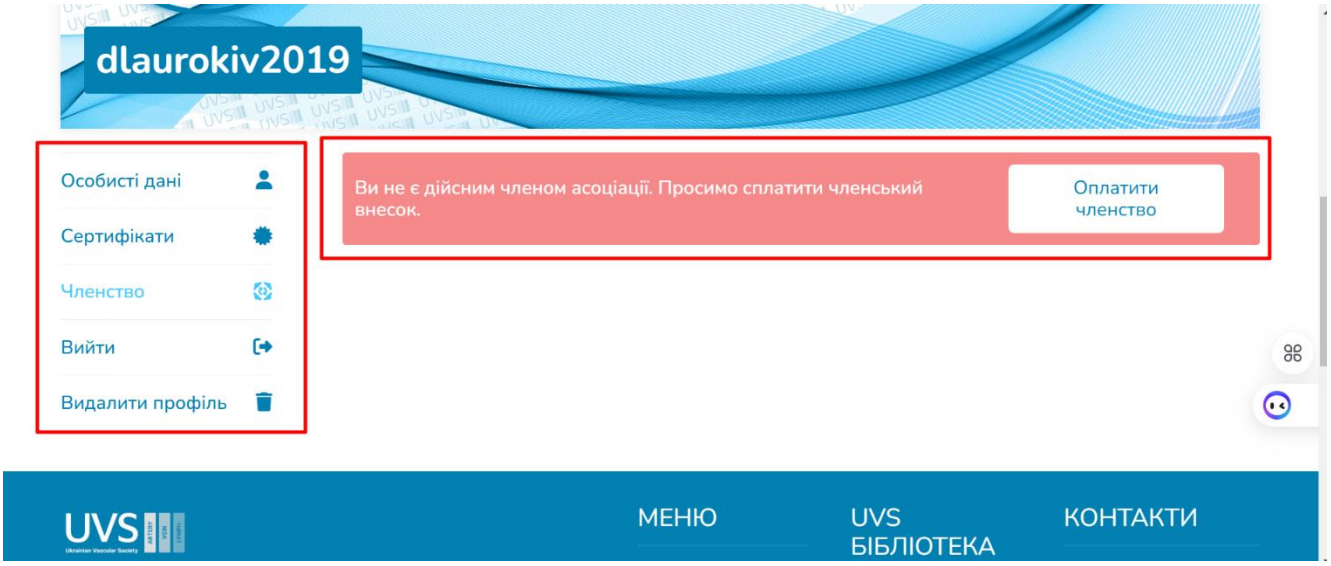

Рисунок 3.7.

2. Далі у новому вікні слід обрати платіжну платформу, наприклад, LiqPay.

3. Після цього необхідно вибрати спосіб оплати, скажімо, Google Pay та натиснути на відповідну кнопку.

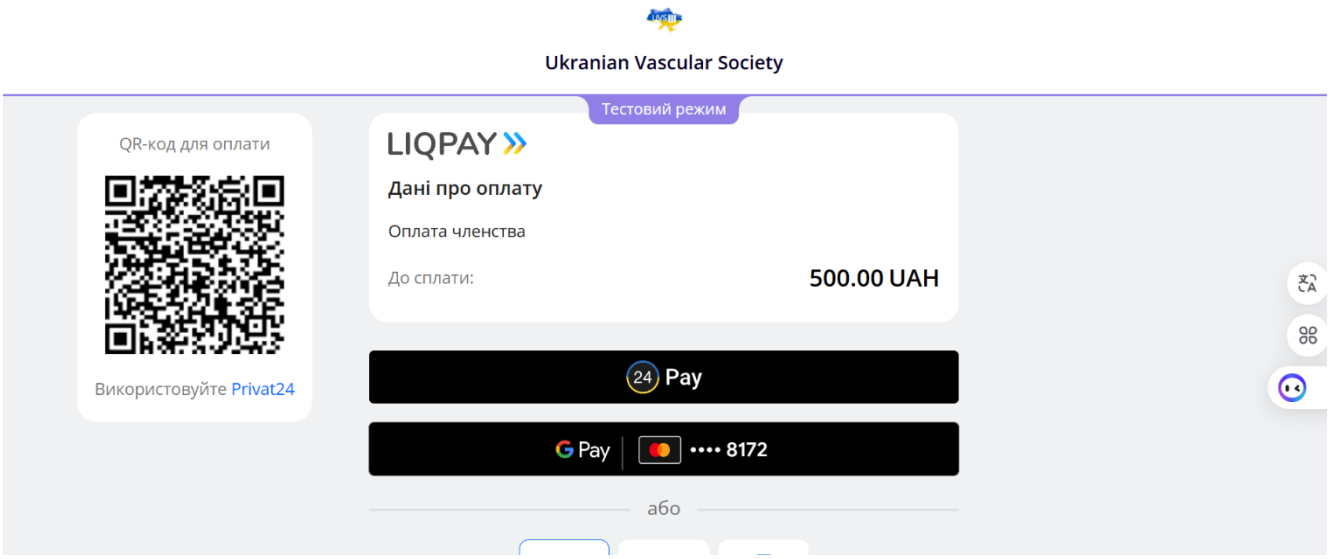

Рисунок 3.8.

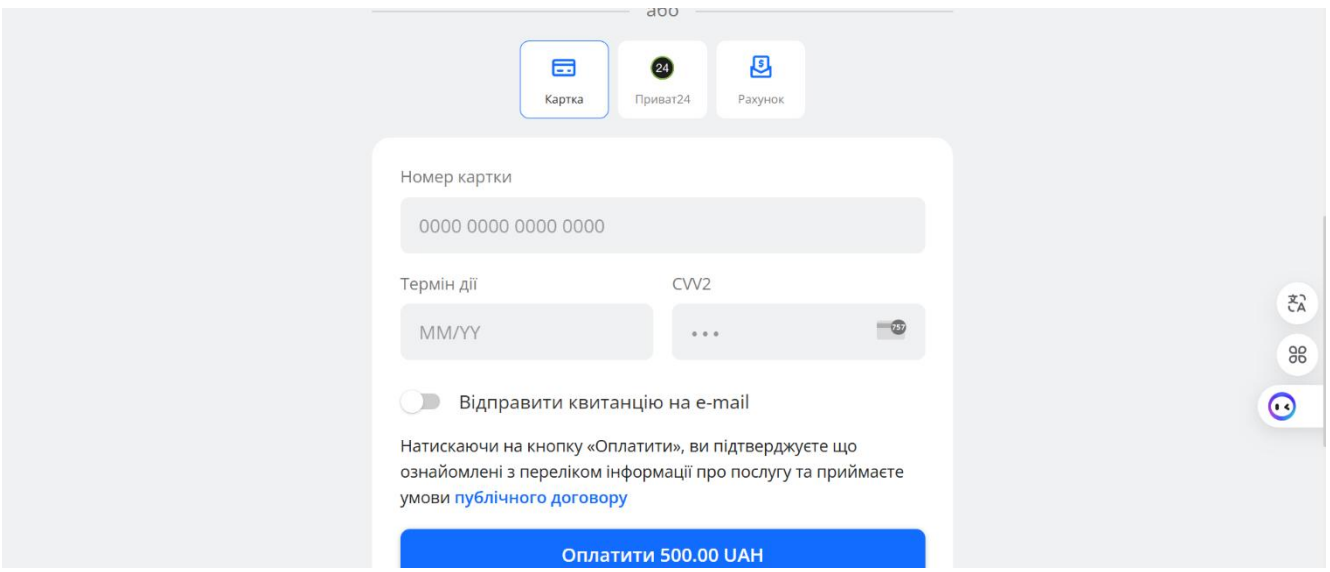

Рисунок 3.9.

4. У вікні, що з’явиться слід підтвердити оплату.

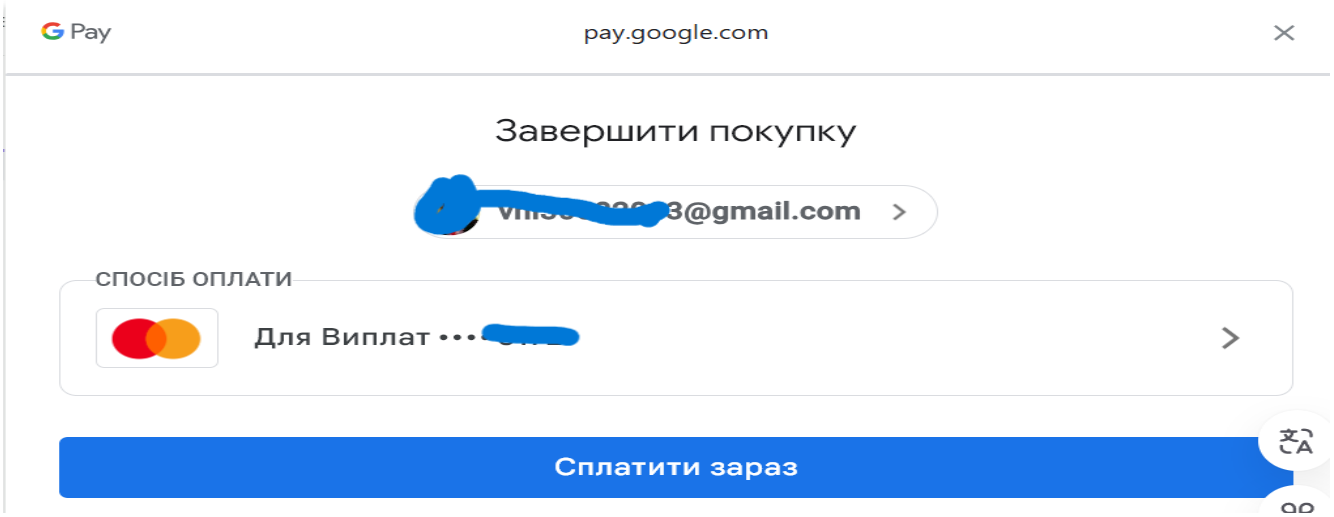

Рисунок 3.10.

Контроль доступу до контенту для користувачів без сертифіката у системі управління членством. UVS реалізує функцію контролю доступу на основі статусу оплати. Для користувачів без сертифіката система обмежує доступ до частини контенту або функцій, дозволяючи їм бачити лише загальнодоступну інформацію. Це стимулює користувачів до своєчасної оплати, забезпечуючи справедливий розподіл доступу до ресурсів на платформі. [25, с.322]

UVS пропонує сучасну систему управління членством, яка відповідає вимогам користувачів завдяки інтеграції з LiqPay, надійному функціонуванню особистого кабінету та контролю доступу, оптимізованому для підтримки активного членства.

3.4. Висновок до розділу 3.

У розділі, присвяченому практичній реалізації інтегрованої системи управління членством на PHP, було детально розглянуто процеси створення ефективного та надійного інструмента для управління членськими даними. За основу використано PHP, що дозволяє гнучко обробляти запити та надавати масштабовані рішення. Інтеграція різноманітних модулів забезпечила автоматизацію ключових функцій, таких як реєстрація, оновлення статусу та

управління платіжними операціями, що сприяло підвищенню функціональності системи. [26, с.322]

Розроблена система відповідає сучасним вимогам безпеки, стабільності та продуктивності, що робить її практичним рішенням для широкого кола підприємств.

ВИСНОВКИ

У **першому** розділі було досліджено теоретичні основи розробки систем управління членством, що включає аналіз концепцій, структурних компонентів і функціональних вимог до сучасних систем. Було підкреслено важливість таких компонентів, як управління членством, відстеження статусу учасників та інтеграція з платіжними платформами, що підвищує гнучкість та зручність використання для користувачів.

Встановлено, що успішна система управління повинна мати простий у використанні інтерфейс, безпечну базу даних для зберігання персональної інформації та механізми для своєчасного повідомлення про завершення членства.

Системи управління членством являють собою комплекс із людей, процесів і технологій для налагодження ефективної взаємодії організації з її учасниками. У сучасному інформаційному середовищі це платформи або програмні рішення, що автоматизують процеси в членських організаціях, клубах, асоціаціях чи фондах, спрощуючи облік, комунікації та фінансові операції.

Головні функції таких систем включають:

1. Облік і зберігання даних: централізоване зберігання інформації з високим рівнем захисту.
2. Оплата та фінансова інтеграція: інтеграція з платіжними сервісами для легкого обслуговування фінансових операцій.
3. Управління подіями: організація заходів, реєстрація учасників, автоматична розсилка запрошень.
4. Автоматизована комунікація: відправка нагадувань і повідомлень.
5. Аналітика та звітність: генерування звітів для аналізу активності членів та фінансових потоків.
6. Залучення та утримання членів: стратегії для залучення нових учасників, забезпечення задоволеності членів і програм лояльності.
7. Розвиток та представництво: підтримка професійного зростання членів і захист їхніх інтересів.

В Україні такі системи застосовуються для автоматизації діяльності різноманітних організацій, забезпечуючи зручне управління комунікаціями, фінансами та подіями. Популярні платформи включають локалізовані рішення, як-от My Membership, Clubeo, SWE Membership, адаптовані до місцевих умов.

Моделі управління членством відображають ключові процеси управління, дозволяючи зрозуміти їх логіку та взаємозв'язки. Методи управління – це інструменти, що реалізують ці моделі, забезпечуючи ефективність асоціацій, об'єднаних спільними інтересами або цілями. Мета управлінських систем полягає в оптимізації взаємодії між учасниками, спрощенні адміністративних процесів, ефективному контролі членських внесків та підвищенні продуктивності. Основними завданнями є налагодження комунікації, фінансовий облік та відстеження статусу членів. Сучасні системи з використанням цифровізації дозволяють автоматизувати ці процеси, сприяючи гнучкості та адаптивності організацій до змін. Основні типи моделей включають CRM, AMS, LMS та гібридні системи.

Технологія програмування – це набір методів і засобів для створення програмного забезпечення з чітким порядком їх застосування. Усі технології, що застосовуються, мають на меті забезпечити:

- високу якість продукту;
- дотримання бюджету;
- завершення роботи у встановлені терміни.

Технологія включає систему інструкцій, що визначає:

- послідовність технологічних операцій;
- умови виконання кожної операції;
- опис операцій з визначенням вхідних даних, очікуваних результатів, а також інструкцій, нормативів, стандартів і методів оцінки.

У **другому** розділі роботи проаналізовано методологічні підходи до розробки інтегрованих систем управління, зокрема, таких, що потребують адаптації під різні платіжні платформи та забезпечення безпеки особистих даних

користувачів. Досліджено ключові методологічні етапи — від визначення потреб і розробки архітектури до вибору програмного стеку. Особливу увагу було приділено методам інтеграції платіжних систем та захисту даних від несанкціонованого доступу, що є критичними для подібних систем.

Функціональні можливості програми включають здатність програмного забезпечення відповідати вимогам користувачів, забезпечуючи точність, захищеність, взаємодію з іншими системами та відповідність стандартам. Система управління членством автоматизує обробку даних членів, керування статусами, підтримує фінансові операції та кібербезпеку. Вона гнучка для адаптації до змін організаційної структури, підтримує інтеграцію з іншими IT-системами та масштабування.

Ключові функції включають:

Управління профілями — редагування та захист даних членів із багатофакторною аутентифікацією та автоматичними сповіщеннями.

Управління категоріями членства — встановлення рівнів доступу з можливістю коригування та автоматичного переходу між категоріями.

Платіжні операції — інтеграція з платіжними сервісами для онлайн-оплат, нагадування про платежі та збереження історії для звітності.

Перевірка статусу — актуалізація прав доступу членів із автоматичним оновленням статусу та сповіщеннями.

Управління подіями — організація подій, реєстрація, відстеження відвідуваності та інтеграція з онлайн-платформами.

Комунікація — підтримка зв'язку через персоналізовані розсилки та інтеграція з CRM.

Управління документами — зберігання, доступ до документів, цифровий підпис і захист конфіденційної інформації.

Архітектура системи управління членством має відповідати бізнес-вимогам, забезпечуючи надійний доступ до інформації та функцій для користувачів. Система складається з кількох компонентів:

Фронтенд створює зручний інтерфейс для користувачів, підтримуючи інтерактивність і швидкий відгук.

Бекенд відповідає за обробку запитів, управління логікою і роботу з базою даних для забезпечення консистентності даних.

База даних зберігає інформацію про користувачів; використовують реляційні (PostgreSQL, MySQL) чи нереляційні (MongoDB) бази.

Авторизація та автентифікація гарантують надійний доступ і захист даних через протоколи, як-от OAuth 2.0.

Email-розсилка автоматично інформує користувачів про оновлення статусів.

Інтеграції підключають систему до платіжних шлюзів, CRM, аналітичних сервісів.

Вибір технологій ґрунтується на таких критеріях: продуктивність, масштабованість, безпека та легкість інтеграції.

Тестування системи управління членством спрямоване на оцінку програмного забезпечення, що відповідає за обробку та впорядкування інформації про членів організації чи платформи. Основне завдання тестування полягає в перевірці коректності функціонування всіх компонентів, виявленні потенційних помилок і забезпеченні захисту та надійності даних користувачів. Під час тестування аналізуються функціональність, продуктивність, безпека, зручність використання системи, а також її відповідність вимогам та технічним стандартам. Цей процес дозволяє вчасно виявити недоліки та сприяти стабільній роботі системи для ефективного обліку і підтримки членства.

У **третьому** розділі наведено результати практичної реалізації системи на PHP, включаючи особливості вибору цього середовища для розробки. PHP був обраний через свою високу адаптивність до веб-застосунків та можливість роботи з різними базами даних, зокрема MySQL.

Було продемонстровано процес інтеграції функцій відстеження статусу членства, відправки нагадувань користувачам та обробки транзакцій через різні платіжні шлюзи.

Практичні результати свідчать про високу ефективність інтегрованої системи у підтримці постійної взаємодії з користувачами та забезпеченні стабільного функціонування всіх модулів.

Для ефективного вибору інструментів для аналізу та управління вимогами у системі управління членством необхідно чітко визначити потреби проєкту, включаючи цілі, завдання, обсяги та специфічні технічні й бізнес-вимоги. Зокрема, важливо, щоб система мала гнучкість для інтеграції та управління користувацькими обліковими записами, авторизацією, обробкою платежів, сегментацією та синхронізацією з іншими системами.

Критерії вибору інструментів управління вимогами:

Функціональні можливості – документування та аналіз вимог, відстеження змін, а також інструменти для контролю їх виконання.

Інтеграція – здатність легко інтегруватись із системами для розробки та тестування, такими як Jira або Confluence.

Зручність інтерфейсу – інтуїтивний інтерфейс, що спрощує освоєння та використання.

Підтримка командної роботи – можливість обміну коментарями, розподілу ролей і доступу.

Вартість – відповідність бюджету організації з урахуванням підтримки та оновлень.

Виконана робота спрямована на розробку ефективної та гнучкої системи управління членством, яка здатна задовольнити сучасні вимоги бізнесу і користувачів до управління статусом членства та інтеграції платіжних можливостей.

Теоретичне дослідження, аналіз існуючих методологічних підходів та практична реалізація системи на PHP довели доцільність вибраного підходу та підтвердили можливість створення інтегрованої системи, яка надає зручний і захищений інтерфейс для користувачів. Ця робота підкреслює важливість інтегрованого підходу, який включає як технічні, так і організаційні аспекти розробки, забезпечуючи надійність та безпеку користування системою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Азаров, І. Сучасні системи управління членством в організаціях. – Київ: Вид-во «Менеджмент», 2020.
2. Боровик, С. Інформаційні системи для управління членством: особливості, функціонал, перспективи. Електронний ресурс: Науковий журнал з менеджменту. – 2021. – № 3.
3. Поліщук, Н. Розробка та впровадження інформаційних технологій в обліку членства в Україні. – Науково-практичний журнал «Інформаційний менеджмент», 2022.
4. Коваль, О. М. Програмні рішення для громадських організацій: огляд сучасних платформ. – Харків: Вид-во «Технологія», 2023.
5. Басюк, Н. М. "Інформаційні системи управління: підходи та інструменти." Київ: Видавничий дім "Кондор", 2020.
6. Бондаренко, С. О. "Цифрові технології у системах управління підприємствами." Харків: Фоліо, 2021.
7. Микитенко, В. О. "Моделі та методи управління інформаційними ресурсами." Львів: Українська Академія Друкарства, 2019.
8. Олійник, М. Г. "Системи управління членством: сучасні підходи." Одеса: Політехнічний інститут, 2021.
9. Сидоренко, А. В. "Інтерактивні моделі в управлінні асоціаціями." Журнал "Менеджмент та адміністрування", №4, 2022, с. 33–48.
10. Кравченко, О. Системи управління взаємовідносинами з клієнтами (CRM) як інструмент ефективного управління // Бізнес Інформ. – 2020. – №3. – С. 30-35.
11. Савченко, М. Сучасні технології для автоматизації управління асоціаціями // Економіка і суспільство. – 2021. – №6. – С. 115-120.
12. Бойко, П., Залевський, Т. Управління асоціаціями: AMS-системи та їхні можливості // Науковий вісник. – 2022. – Т. 11. – С. 64-70.

13.Сидоренко, А. Використання гібридних моделей управління членством в корпоративних структурах // Менеджмент XXI. – 2021. – №4. – С. 88-92.

14.Карпенко, І. Г. Порівняння моделей CRM, AMS та LMS для управління асоціаціями // Вісник економічної науки України. – 2023. – №5. – С. 29-35.

15.Антонов В.В. Інформаційні системи в управлінні. Київ: Видавництво "Політехніка", 2021.

16.Данилюк С.М. Основи автоматизованих інформаційних систем. Львів: Видавничий центр ЛНУ, 2020.

17.Коваленко О.О. Сучасні інформаційні системи: управління та інтеграція. Харків: Вид. ХНУРЕ, 2019.

18.Петров В.М. Менеджмент інформаційних систем: теорія та практика. Одеса: Вид. ОНПУ, 2022.

19.Шевченко І.В. Основи інформаційного менеджменту. Київ: Видавничий дім "Кондор", 2023.

20.Мельник В. І. Основи тестування програмного забезпечення: підручник. Київ: Кондор, 2020. 300 с.

21. Бойко І. Б., Мазур О. Л. Тестування програмних систем. Підходи та інструменти // Вісник Харківського національного університету. – 2021. – № 6. – С. 43-49.

22. Кондратенко Ю. А. Тестування продуктивності та безпеки програмних систем. Харків: Університетська книга, 2019. 256 с.

23. Офіційний сайт інструмента Postman: https://www.postman.com.

24.Кузьмін О. Є., Білецька О. І. Управління ІТ-проектами: Підручник. Львів: Львівська політехніка, 2020. 256 с.

25.Грінченко Т. В., Коваленко С. І. Аналіз і документування вимог до інформаційних систем. К.: Вид-во НАУ, 2019. 312 с.

26.Офіційний сайт Державного університету телекомунікацій України. https://www.dut.edu.ua

27. Бузунов І. В. Системи управління вимогами в програмному забезпеченні: підручник. Київ: Видавничий центр "Академія", 2019. 512 с.

28. Тараненко О. М. Методи і засоби автоматизації процесів розробки та управління вимогами. Київ: НТУУ "КПІ", 2020. 432 с.

29. Яковенко В. В., Сорока Д. Ю. Розробка програмного забезпечення: теорія і практика. Харків: Видавництво ХНУРЕ, 2018. 615 с.

30.Онлайн платформа з розробки програмного забезпечення Atlassian. Доступно: https://www.atlassian.com/

31. Jama Software. Офіційний сайт: https://www.jamasoftware.com/

32.Брич В.Я., Корман М.М. Психологія управління: навч. посібник. Київ: Кондор, 2013. 384 с.

33.11. Василенко В.А. Теорія і практика розробки управлінських рішень: навч. посібник. Київ: ЦУЛ, 2013. 420 с

34.Карамишев Д. В. Програмно-цільовий підхід до реалізації державної політики у сфері охорони здоров'я / Д. В. Карамишев //

35.22. Книш С.В. Адміністративно-правові відносини у сфері охорони здоров'я в Україні: автореф. дис. … докт. юрид. наук: 12.00.07. Тернопіль, 2019. 36 с.

36.Кузьменко, І. В. Розробка веб-додатків на PHP: навчальний посібник. — Київ: Кондор, 2019. — 280 с.

37. Михайлов, М. А. PHP для професіоналів. — Львів: Видавничий дім "Магістр", 2018. — 350 с.

38.Грищук, О. Ю. Веб-програмування на PHP: посібник для початківців. — Харків: Фоліо, 2021. — 220 с.

39.Савчук, О. С. PHP: основи програмування. — Київ: Наукова думка, 2020. — 198 с.

40.Прохоренко А.Л., Мельник Г.В. Розробка інтегрованої системи управління членством UVS з функцією відстеження статусу членства та оплатою через різні платіжні платформи. Мехатронні системи: інновації та інжиніринг : тези доповідей VIII Міжнародної науково-практичної конференції, 7 листопада 2024 р., м. Київ : КНУТД, 2024, с. 143-144

ДОДАТОК А

КОД ДО СХЕМИ 2.1. АРХІТЕКТУРИ СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ НА PYTHON

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Create figure and axis
fig, ax = plt.subplots(figsize=(10, 6))

# Disable axis
ax.axis('off')

# Define colors
frontend_color = "#A3D5FF"
backend_color = "#FFB3A3"
database_color = "#FFD580"
auth_color = "#C3FFA3"
integration_color = "#D5A3FF"

# Define boxes
# Frontend
ax.text(0.2, 0.8, "Frontend (React.js/Vue.js)", ha="center", va="center", fontsize=10,
bbox=dict(facecolor=frontend_color, edgecolor="black", boxstyle="round,pad=0.3"))

# API Gateway
ax.text(0.5, 0.8, "API (GraphQL/REST)", ha="center", va="center", fontsize=10,
bbox=dict(facecolor="lightgrey", edgecolor="black", boxstyle="round,pad=0.3"))
```

```
# Backend
ax.text(0.5, 0.6, "Backend (Node.js/Django)", ha="center", va="center", fontsize=10, bbox=dict(facecolor=backend_color, edgecolor="black", boxstyle="round,pad=0.3"))

# Database
ax.text(0.8, 0.6, "Database (PostgreSQL/MongoDB)", ha="center", va="center", fontsize=10, bbox=dict(facecolor=database_color, edgecolor="black", boxstyle="round,pad=0.3"))

# Authentication
ax.text(0.5, 0.4, "Authentication (OAuth 2.0/JWT)", ha="center", va="center", fontsize=10, bbox=dict(facecolor=auth_color, edgecolor="black", boxstyle="round,pad=0.3"))

# Integrations
ax.text(0.8, 0.4, "Integrations\n(Payment, Analytics)", ha="center", va="center", fontsize=10, bbox=dict(facecolor=integration_color, edgecolor="black", boxstyle="round,pad=0.3"))

# User
ax.text(0.2, 0.95, "User", ha="center", va="center", fontsize=10, bbox=dict(facecolor="white", edgecolor="black", boxstyle="round,pad=0.3"))

# Arrows between components
arrow_props = dict(facecolor="black", arrowstyle="->", lw=1)

# User to Frontend
ax.annotate("", xy=(0.2, 0.83), xytext=(0.2, 0.9), arrowprops=arrow_props)

# Frontend to API
```

```
ax.annotate("", xy=(0.38, 0.8), xytext=(0.28, 0.8), arrowprops=arrow_props)
ax.annotate("", xy=(0.45, 0.8), xytext=(0.55, 0.8), arrowprops=arrow_props)

# API to Backend
ax.annotate("", xy=(0.5, 0.75), xytext=(0.5, 0.65), arrowprops=arrow_props)

# Backend to Database
ax.annotate("", xy=(0.62, 0.6), xytext=(0.72, 0.6), arrowprops=arrow_props)

# Backend to Authentication
ax.annotate("", xy=(0.5, 0.55), xytext=(0.5, 0.45), arrowprops=arrow_props)

# Backend to Integrations
ax.annotate("", xy=(0.62, 0.6), xytext=(0.72, 0.4), arrowprops=arrow_props)

# Show the diagram
plt.title("System Architecture for Membership Management")
plt.show()
```

ДОДАТОК Б

ПРИКЛАД РЕАЛІЗАЦІЇ НА PHP (КОД) ФУНКЦІЇ ВІДСТЕЖЕННЯ СТАТУСУ ЧЛЕНСТВА У СИСТЕМІ ТА ПОЯСНЕННЯ КОДУ

```
<?php
class Membership {
    public $user_id;
    public $start_date;
    public $end_date;
    public $status;

    public function checkStatus() {
        if (new DateTime() > new DateTime($this->end_date)) {
            $this->status = 'expired';
            // Send reminder email
            mail($this->user_email, 'Your membership has expired', 'Please renew your membership.');
        }
    }
}
?>
```

Пояснення коду. Цей код написаний на PHP і реалізує клас для управління членством. Нижче наведено детальне пояснення роботи цього коду на науковій українській мові.

Опис класу Membership. Клас Membership є шаблоном, який представляє структуру та методи для зберігання й обробки даних про членство користувача. Основними атрибутами цього класу є ідентифікатор користувача, дата початку членства, дата закінчення та поточний статус.

Оголошення атрибутів

1. public $user_id; – Це публічна змінна, яка зберігає унікальний ідентифікатор користувача. Тип даних не вказано, тому значенням може бути як число, так і рядок.
2. public $start_date; – Публічна змінна, що зберігає дату початку членства. Вона повинна містити значення, яке можна інтерпретувати як дату (наприклад, у форматі YYYY-MM-DD), щоб полегшити порівняння дати.
3. public $end_date; – Публічна змінна, що містить дату закінчення членства. Її значення використовується для визначення, чи є членство активним.
4. public $status; – Публічна змінна, яка зберігає поточний статус членства (наприклад, active , expired).

Метод checkStatus. Метод checkStatus виконує перевірку дати закінчення членства для визначення його поточного статусу.

1. Порівняння дати закінчення з поточною датою:

php

if (new DateTime() > new DateTime($this->end_date)) {

Цей рядок створює об'єкт DateTime , який містить поточну дату й час, і порівнює його з об'єктом DateTime , створеним на основі значення $this->end_date. Якщо поточна дата більша за дату закінчення членства, це означає, що членство прострочене.

2. Оновлення статусу:

php

$this->status = 'expired';

Якщо умова перевірки дати виконана, статус членства змінюється на expired.

3. Надсилання нагадування про продовження членства:

php

mail($this->user_email, 'Your membership has expired', 'Please renew your membership.');

Функція mail() надсилає електронний лист користувачеві з нагадуванням про необхідність продовжити членство. Проте, в коді не оголошено змінної $user_email , що може спричинити помилку під час виконання.

Висновок. Даний клас Membership дозволяє зберігати інформацію про членство, включно з перевіркою його дійсності, та надсилати повідомлення у випадку, якщо членство прострочене. Проте, для безперебійної роботи необхідно додати атрибут $user_email , який зберігатиме електронну адресу користувача.

ДОДАТОК В

ІНТЕРФЕЙС СИСТЕМИ УПРАВЛІННЯ ЧЛЕНСТВОМ UVS

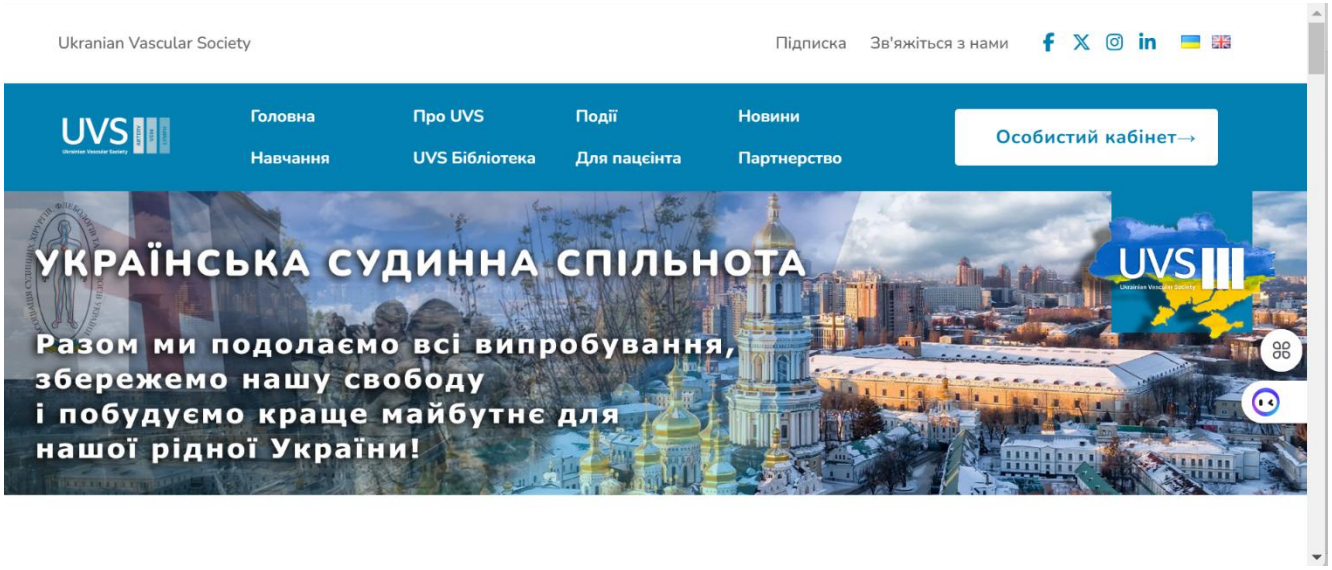

Рисунок 3.11. Головна сторінка інтегрованої системи управління членством UVS

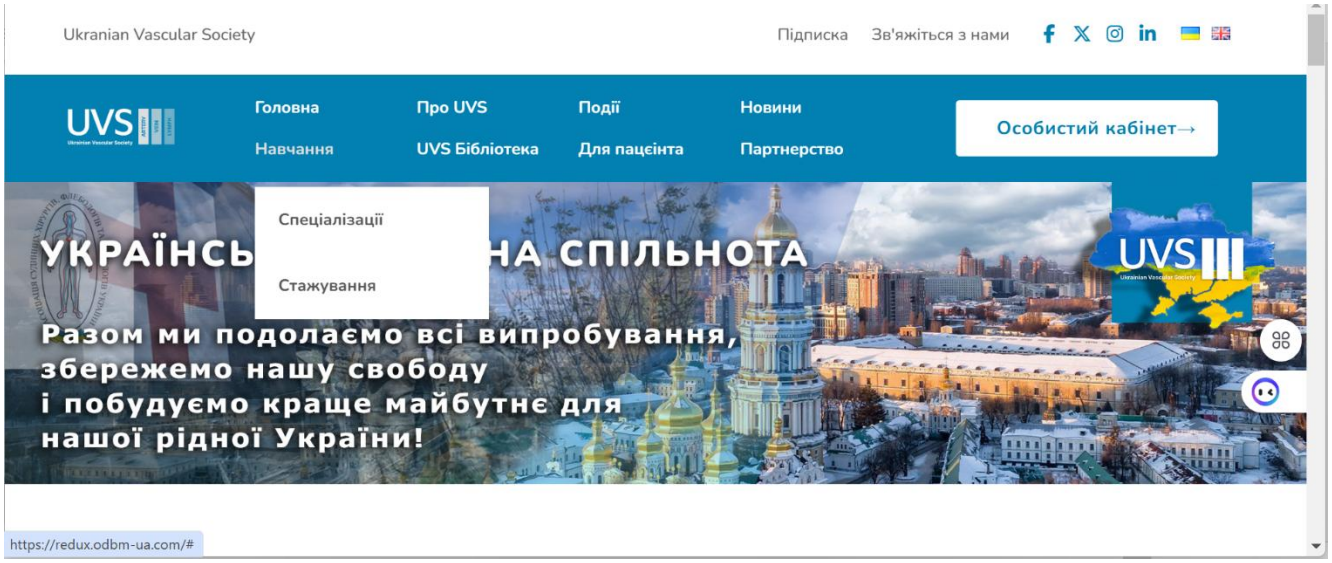

Рисунок 3.12. Пункт меню Навчання інтегрованої системи управління членством UVS

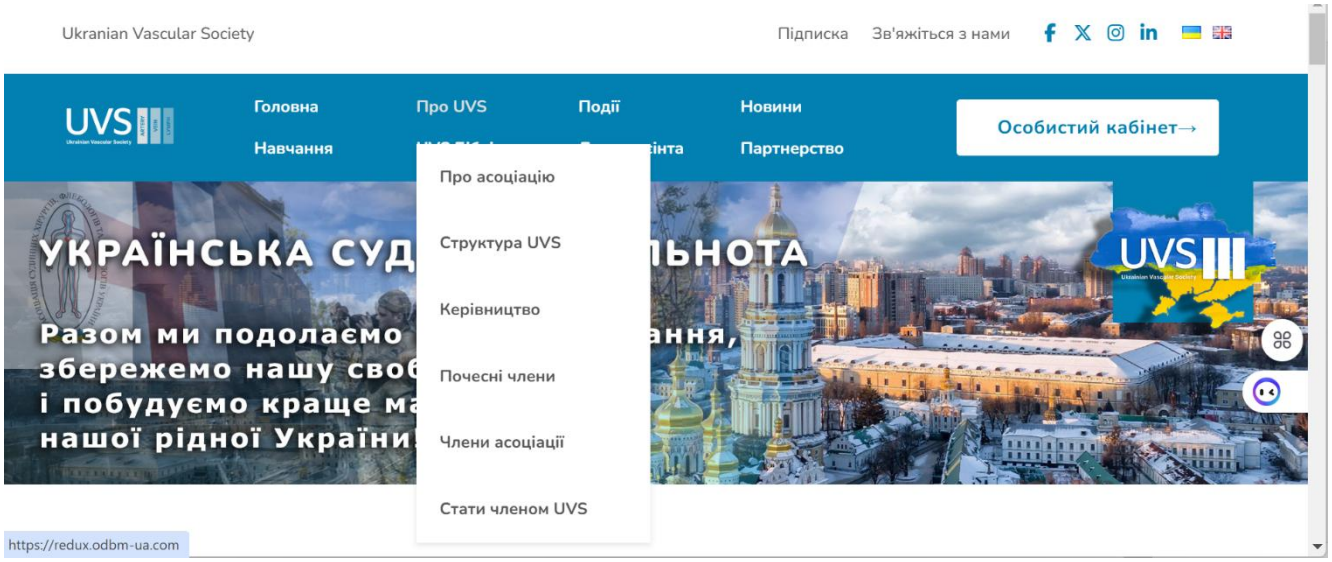

Рисунок 3.13. Пункт меню Про UVS інтегрованої системи управління членством UVS

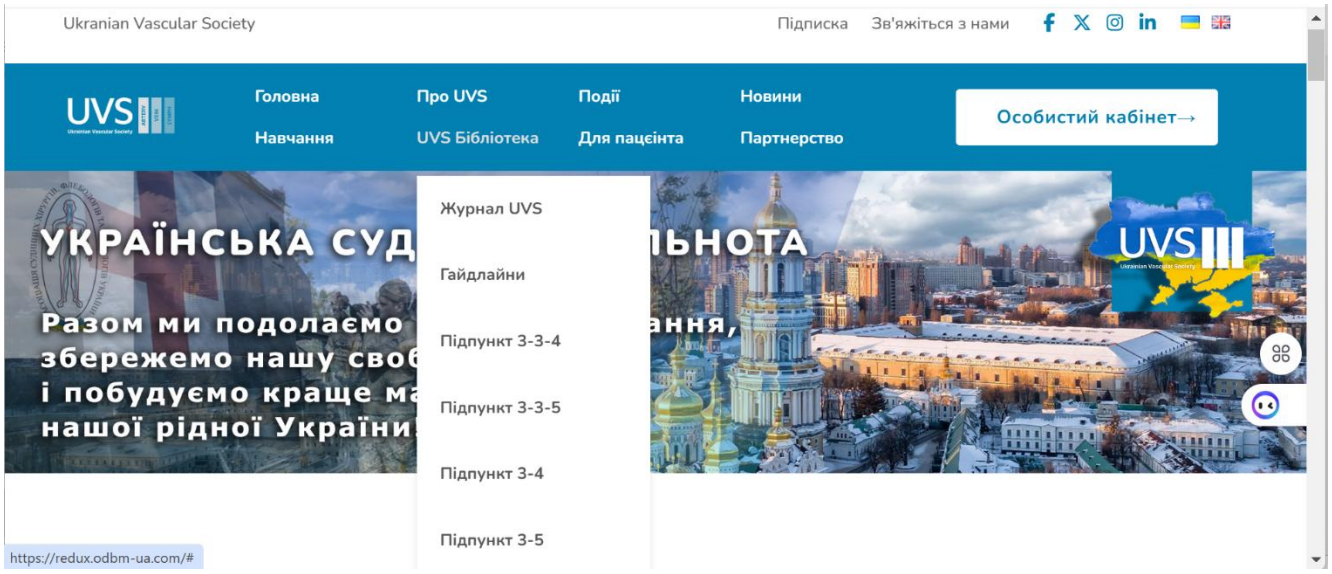

Рисунок 3.14. Пункт меню UVS Бібліотека інтегрованої системи управління членством UVS

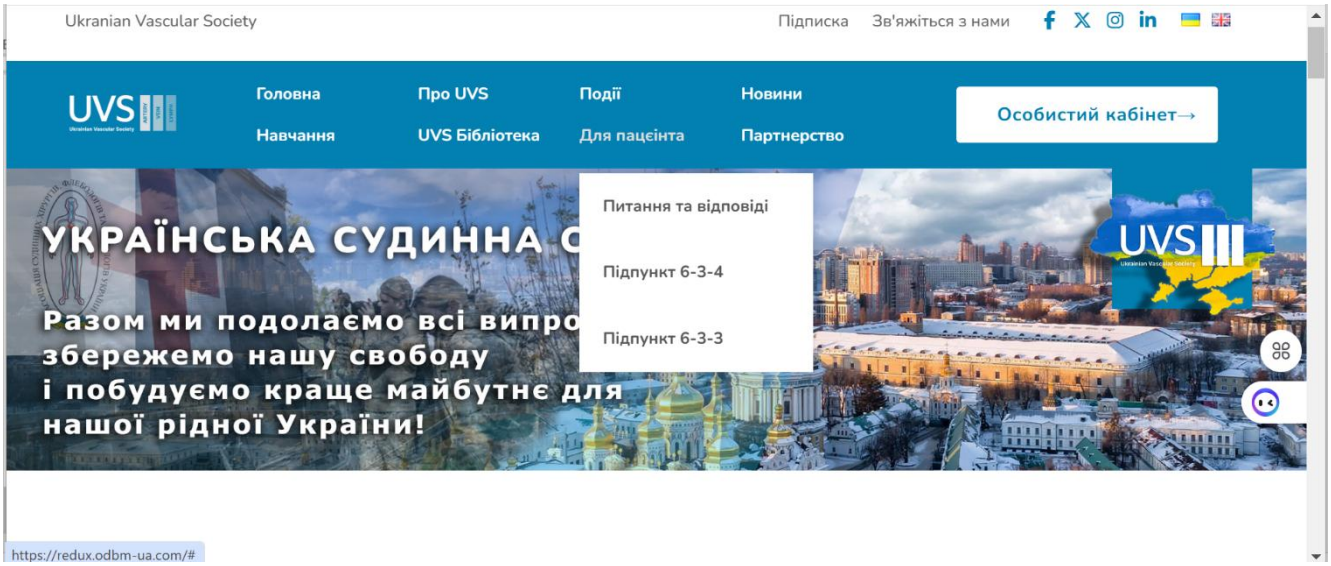

Рисунок 3.15. Пункт меню Для пацієнта інтегрованої системи управління членством UVS

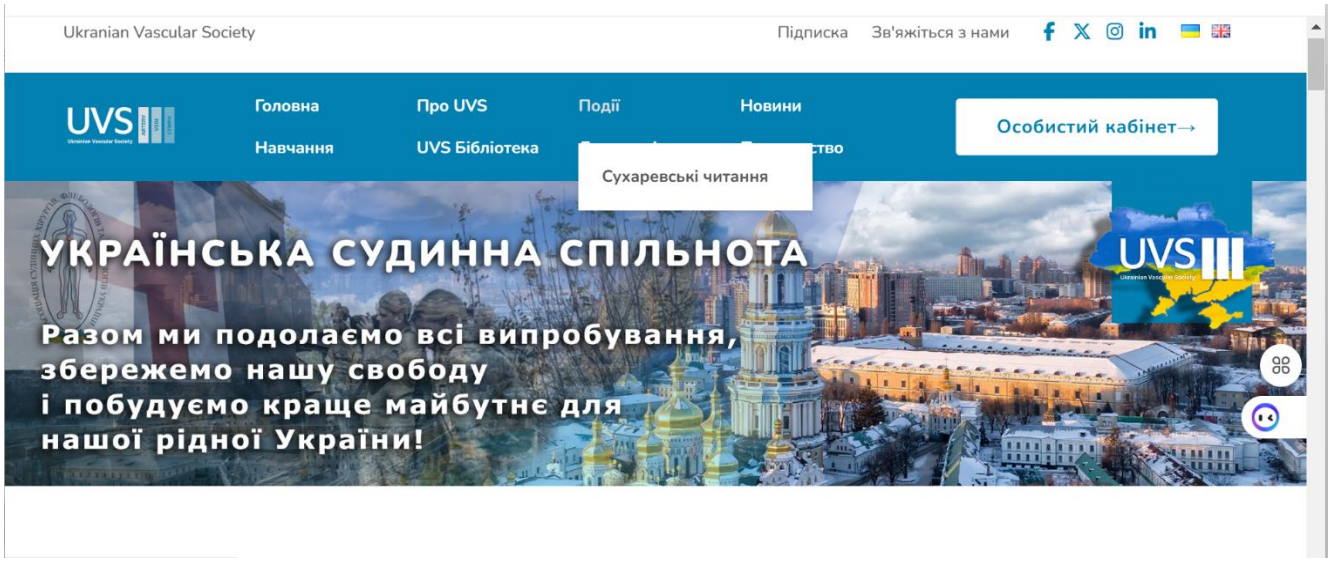

Рисунок 3.16. Пункт меню Події інтегрованої системи управління членством UVS

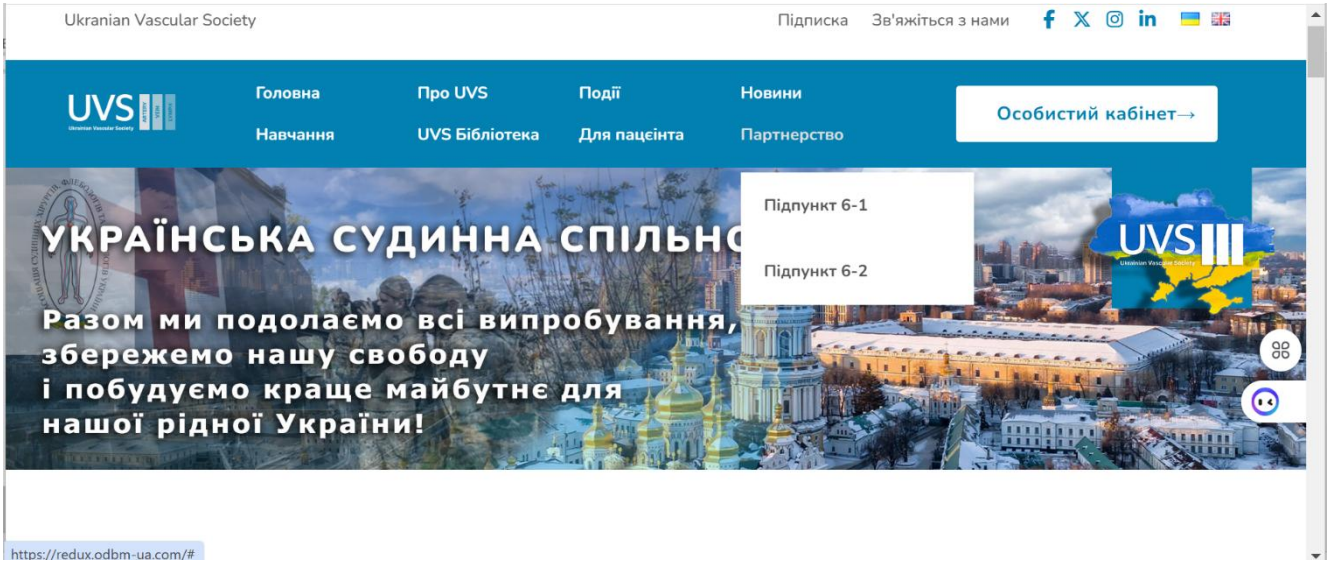

Рисунок 3.17. Пункт меню Партнерство інтегрованої системи управління членством UVS

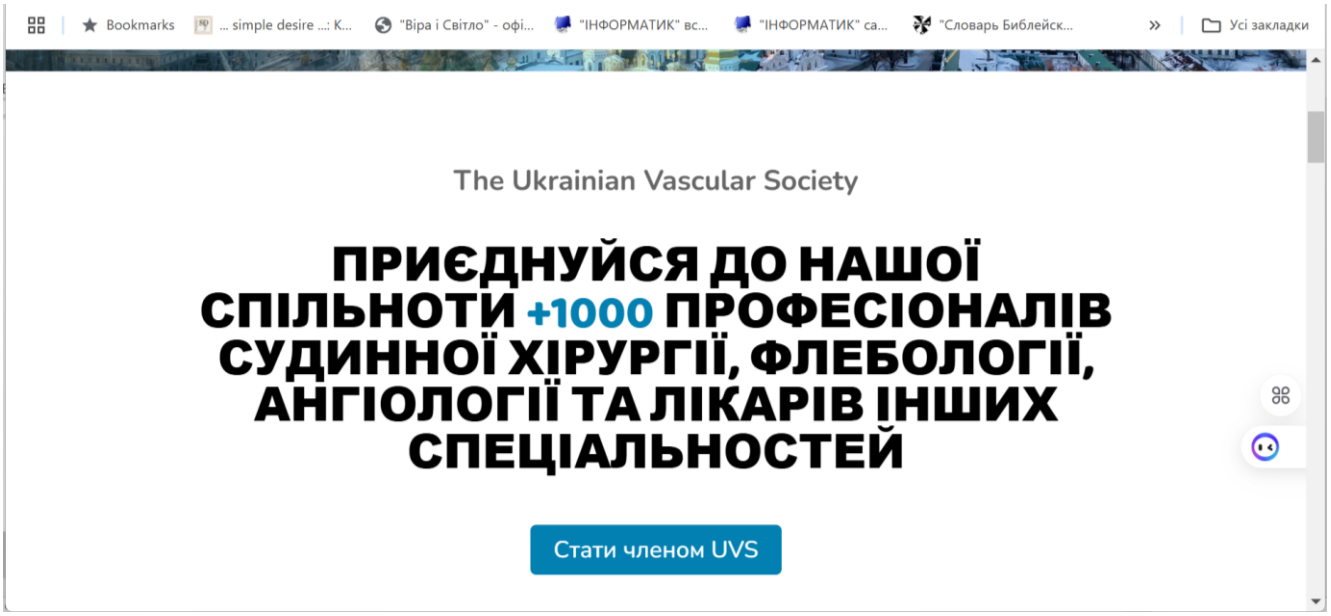

Рисунок 3.18. Головна сторінка інтегрованої системи управління членством UVS

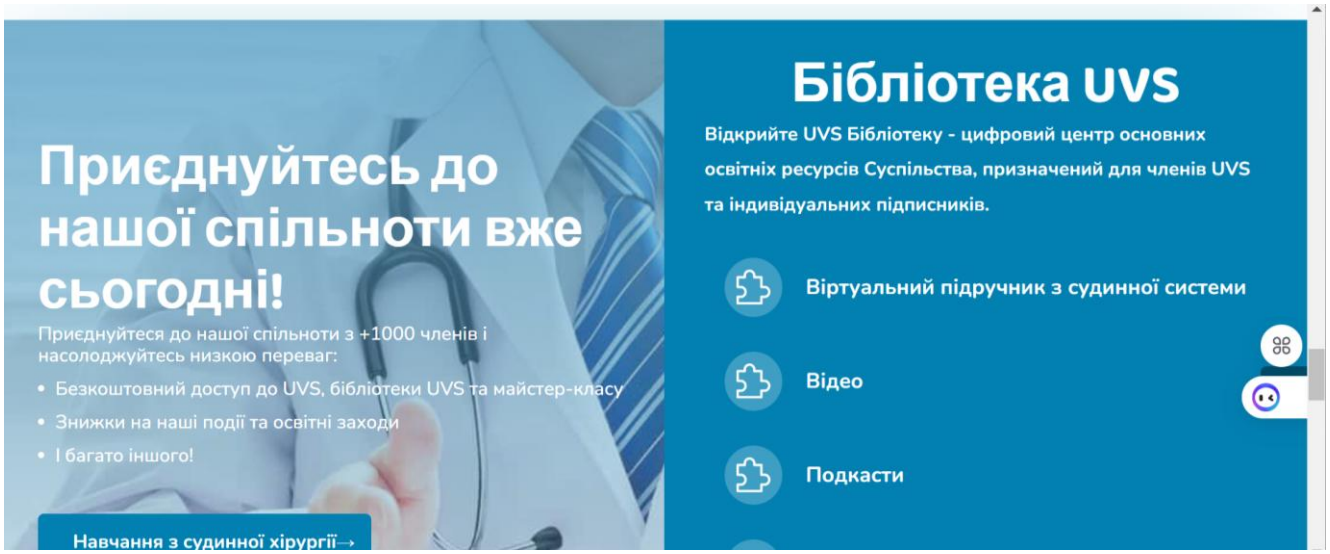

Рисунок 3.19. Сторінка Бібліотека інтегрованої системи управління членством UVS

Рисунок 3.20. Головна сторінка (низ) інтегрованої системи управління членством UVS

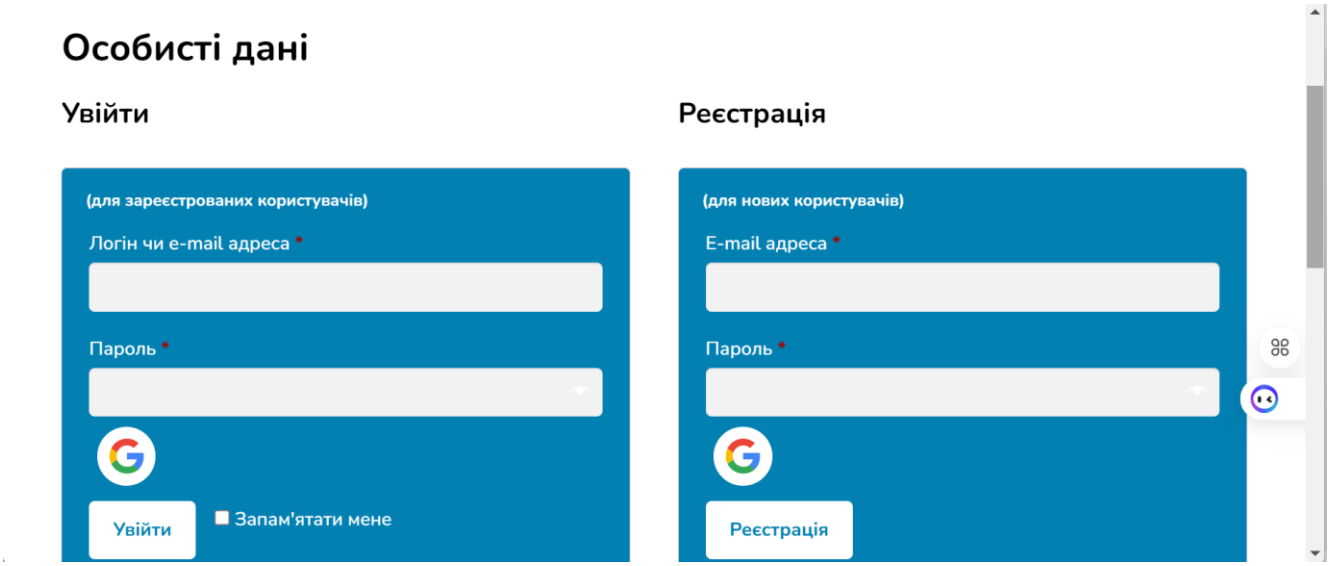

Рисунок 3.21. Форма для входу в особистий кабінет інтегрованої системи управління членством UVS

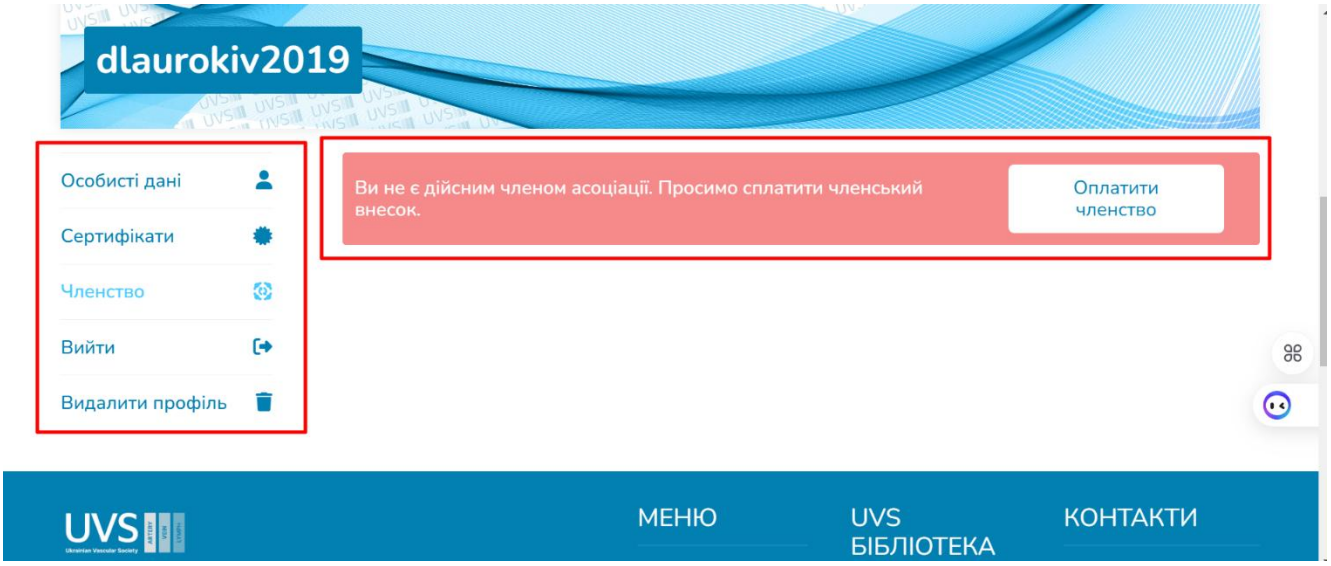

Рисунок 3.22. Розділ Членство в особистому кабінеті інтегрованої системи управління членством UVS