Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальний інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

Розробка WEB-ресурсу управління родоводом

Виконав студент 2(4) курсу, групи КН-22мс
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Бойко А.В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Денисюк В. О.
(прізвище та ініціали)
« 07 » 06 2024 р.

Рецензент: к.т.н., доцент каф. САІТ
Козачко О. М.
(прізвище та ініціали)
« 07 » 06 2024 р.

Допущено до захисту
Зав. Кафедри Яровий І.А.
« 07 » 06 2024р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А. А.
«07» 03 2024 р

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Бойко Андрію Вікторовичу

1. Тема роботи: Розробка WEB-ресурсу управління родоводом.
Керівник роботи: Денисюк Валерій Олександрович, к.т.н., доц.
Затверджені наказом вищого навчального закладу "11" 03 2024 року №80

2. Термін подання студентом роботи 27.05.2024 р.

3. Вихідні дані до роботи:
Мова програмування фронтенду JavaScript:
Мова програмування бекенду PHP;
Середовище розробки PhpStorm.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
Вступ; Аналіз предметної області управління родоводом; Аналіз відомих програм аналогів; Постановка задачі; Обґрунтування засобів розробки; Розробка ER-діаграми; Проєктування основних алгоритмів; Обґрунтування вибору мови програмування; Створення серверної частини web-ресурсу управління родоводом; Створення клієнтської частини web-ресурсу управління родоводом; Висновки; Список використаних джерел.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): ER-діаграма бази даних; Алгоритм рендеру дерева роду; Алгоритм формування даних дерева; Алгоритм обрахунку поколінь персон.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Денисюк В. О., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07.03.2024 р.

Календарний план

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області управління родоводом	06.05.24	07.05.24	
2	Аналіз галузі генеалогії	08.05.24	10.05.24	
3	Аналіз відомих програм аналогів	11.05.24	12.05.24	
4	Постановка задачі	13.05.24	14.05.24	
5	Обгрунтування засобів розробки системи управління родоводом	15.05.24	16.05.24	
6	Розробка ER-діаграми	17.05.24	19.05.24	
7	Проєктування основних алгоритмів	20.05.24	24.05.24	
8	Обгрунтування вибору мови програмування	25.05.24	26.05.24	
9	Створення серверної частини web-ресурсу управління родоводом	27.05.24	31.05.24	
10	Створення клієнтської частини web-ресурсу управління родоводом	01.06.24	04.06.24	
11	Підготовка матеріалів БДР до захисту	05.06.24	06.06.24	

Студент _____ (підпис) **Бойко А.В.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Денисюк В. О.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 112 сторінок формату А4, на яких є 19 рисунки, список використаних джерел містить 20 найменувань.

Перед виконанням даної роботи поставлено певні задачі, яких потрібно досягти: аналіз предметної області управління родоводом, аналіз відомих програм аналогів, обґрунтування засобів розробки, розробка ER-діаграми, проєктування основних алгоритмів, обґрунтування вибору мови програмування, створення серверної частини web-ресурсу управління родоводом, створення клієнтської частини web-ресурсу управління родоводом.

У даній бакалаврській дипломній роботі досліджено процес розробки веб-ресурсу управління родоводом. У роботі проаналізовано сучасні сервіси-аналоги, що використовуються для управління родоводами та генеалогічними даними. Наведено переваги і недоліки існуючих сервісів-аналогів. Досліджено способи вирішення даної задачі. Результатом дипломного дослідження є система у вигляді веб-ресурсу, що реалізовано з використанням сучасних технологій веб-розробки, зокрема JavaScript, HTML, CSS, SvelteKit та бібліотеки Pixi.js для рендерингу дерева родоводу.

Ключові слова: веб-ресурс, родовід, програмування, ER-діаграма, SvelteKit, Pixi.js.

ABSTRACT

The bachelor's thesis consists of 112 pages of A4 format, with 19 figures, and a list of references containing 20 titles.

This thesis has certain tasks to be achieved: analysis of the subject area of pedigree management, analysis of known analog programs, justification of development tools, development of an ER diagram, design of basic algorithms, justification of the choice of programming language, creation of the server part of the pedigree management web resource, creation of the client part of the pedigree management web resource.

This bachelor's thesis investigates the process of developing a pedigree management web resource. The paper analyzes modern analog services used for managing pedigrees and genealogical data. The advantages and disadvantages of existing analog services are presented. The ways of solving this problem are investigated. The result of the thesis research is a system in the form of a web resource implemented using modern web development technologies, in particular JavaScript, HTML, CSS, SvelteKit and the Pixi.js library for rendering the family tree.

Keywords: web resource, family tree, programming, ER diagram, SvelteKit, Pixi.js.

Зміст

Вступ 3
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ УПРАВЛІННЯ РОДОВОДОМ 5
1.1 Аналіз галузі генеалогії 5
1.2 Аналіз відомих програм аналогів 7
1.3 Постановка задачі 20
1.4 Висновок 22
2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ УПРАВЛІННЯ РОДОВОДОМ 23
2.1 Обгрунтування засобів розробки системи управління родоводом 23
2.2 Розробка ER-діаграми 25
2.3 Проєктування основних алгоритмів 38
2.4 Висновок 42
3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ УПРАВЛІННЯ РОДОВОДОМ 43
3.1 Обґрунтування вибору мови програмування 43
3.2 Створення серверної частини web-ресурсу управління родоводом 47
3.3 Створення клієнтської частини web-ресурсу управління родоводом 57
3.4 Висновок 61
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63
ДОДАТОК А ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ 66
ДОДАТОК Б ІНСТРУКЦІЯ КОРИСТУВАЧА 67
ДОДАТОК В ЛІСТИНГ СЕРВЕРНОЇ ЧАСТИНИ. ЛІСТИНГ КЛІЄНТСЬКОЇ ЧАСТИНИ 74
ДОДАТОК Г ГРАФІЧНА ЧАСТИНА 103

Вступ

У сучасному світі все більше людей цікавляться своїм родоводом, прагнучи дізнатися про своїх предків та зберегти історію своєї сім'ї для майбутніх поколінь. Розуміння свого родоводу сприяє зміцненню сімейних зв'язків, формуванню національної ідентичності та збереженню культурної спадщини. У зв'язку з цим виникає потреба у зручних та функціональних інструментах для збирання, збереження та управління інформацією про родинні зв'язки.

Створення веб-ресурсу для управління родоводом надає користувачам можливість легко і зручно вводити, зберігати та візуалізувати інформацію про своїх предків, створювати генеалогічні дерева, додавати фотографії та документи. Такий ресурс дозволяє не тільки систематизувати родинні дані, але й ділитися ними з іншими членами родини, що сприяє спільному збереженню родинної історії.

Практична цінність одержаних результатів полягає в тому, що розроблені методи та інструменти можуть бути використані іншими розробниками для створення аналогічних веб-ресурсів, що сприятиме подальшому розвитку генеалогічних досліджень і збереженню культурної спадщини.

Реалізація такого веб-ресурсу передбачає вирішення складних технічних завдань, пов'язаних із забезпеченням високої продуктивності та масштабованості даних. Проектування ефективних механізмів для збереження родоводів та автентифікації користувачів вимагає глибокого розуміння технологій веб-розробки та баз даних. У цій роботі буде розглянуто питання, пов'язані з проектуванням та реалізацією архітектури веб-ресурсу, вибором оптимальних технологій та методів розробки, а також забезпеченням високої якості та стабільності роботи програмного забезпечення.

Об’єктом дослідження є процес дослідження родоводу, тоді як **предметом дослідження** є програмні засоби розробки та алгоритми підготовки даних та рендеру дерева роду.

Основною метою даної бакалаврської дипломної роботи є покращення доступності та гнучкості WEB-ресурсу управління родоводом.

Для досягнення цієї мети потрібно виконати такі **завдання**:

- Аналіз предметної області управління родоводом,
- Аналіз галузі генеалогії,
- Аналіз відомих програм аналогів,
- Постановка задачі,
- Обгрунтування засобів розробки системи управління родоводом,
- Розробка ER-діаграми,
- Проєктування основних алгоритмів,
- Обґрунтування вибору мови програмування,
- Створення серверної частини web-ресурсу управління родоводом,
- Створення клієнтської частини web-ресурсу управління родоводом,
- Тестування програмного забезпечення.

Отже, в рамках бакалаврської дипломної роботи необхідно виконати підготовчі роботи для розробки та реалізації веб-ресурсу управління родоводом, спираючись на методичні вказівки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ УПРАВЛІННЯ РОДОВОДОМ

1.1 Аналіз галузі генеалогії

Розробка сервісу для управління генеалогічним деревом відкриває широкі можливості для дослідження та інновацій у галузі генеалогії [1]. Для початку, слід ретельно проаналізувати потреби та очікування користувачів у генеалогічних дослідженнях. Це може включати як аматорів, які шукають свої родові зв'язки, так і професіоналів, які займаються родинними деревами для клієнтів або для наукових досліджень.

Основна функціональність сервісу повинна бути спрямована на зручність введення та візуалізацію генеалогічних даних. Це включає можливість додавання нових осіб до дерева, встановлення зв'язків між ними, додавання фотографій та інших документів, а також функції для створення приміток та відстеження історії змін.

Іншим важливим аспектом є забезпечення безпеки та конфіденційності даних користувачів. З урахуванням того, що генеалогічні дослідження можуть містити чутливу особисту інформацію, таку як дати народження, шлюбу та смерті, важливо вжити відповідних заходів для захисту цих даних від несанкціонованого доступу.

Для забезпечення ефективності та зручності користування, сервіс також може включати різноманітні інструменти та функції для аналізу генеалогічних даних. Це може включати автоматичне підрахунок статистики, виявлення повторень у родоводі, а також можливість відстеження та аналізу генетичних даних.

Крім того, важливо враховувати тенденції розвитку галузі генеалогії та інформаційних технологій. Нові технології, такі як штучний інтелект, блокчейн

та аналіз даних, можуть відкривати нові можливості для розробки інноваційних функцій та сервісів у галузі генеалогії.

Узагальнюючи, розробка сервісу для управління генеалогічним деревом вимагає ретельного аналізу потреб користувачів, розробки ефективної та зручної функціональності, забезпечення безпеки та конфіденційності даних, а також використання сучасних технологій для розвитку інноваційних рішень у галузі генеалогії.

Однією з ключових переваг розробки сервісу для управління генеалогічним деревом є можливість створення і підтримки генеалогічних досліджень у форматі, зрозумілому для користувачів будь-якого рівня. Це може означати створення інтуїтивно зрозумілого інтерфейсу, який дозволяє легко додавати нових членів родини, встановлювати зв'язки між ними та додавати додаткову інформацію, таку як дати подій та місця.

Також важливо враховувати потенціал для розширення функціональності за допомогою інтеграції з іншими сервісами та джерелами інформації. Наприклад, інтеграція з онлайн-архівами документів або генетичними базами даних може допомогти користувачам знаходити нові докази та розширювати свої дослідження.

Зокрема, враховуючи широкий спектр користувачів генеалогічних сервісів, важливо забезпечити адаптивний та інклюзивний дизайн, який би враховував потреби різних груп користувачів. Це може включати підтримку різних мов, включаючи менш розповсюджені, а також функції доступності для людей з обмеженими можливостями.

Урахування цих аспектів може допомогти створити сервіс, який буде корисним і привабливим для широкого кола користувачів. Крім того, такий сервіс може стати важливим інструментом для збереження сімейної спадщини та сприяти збереженню та передачі історії родини наступним поколінням.

Нарешті, важливо пам'ятати про можливості масштабування та розвитку сервісу з плином часу. Технології швидко змінюються, і для того, щоб сервіс залишався актуальним та конкурентоспроможним, потрібно постійно вдосконалювати його функціональність та впроваджувати нові можливості.

1.2 Аналіз відомих програм аналогів

Проаналізувати відомі програми-аналоги є важливим етапом при розробці будь-якого нового сервісу або продукту. Ось деякі ключові причини, чому це важливо:

1. Вивчення конкуренції: Аналіз аналогічних програм допомагає зрозуміти, які функції та можливості вже присутні на ринку і як вони реалізовані. Це дозволяє виявити слабкі та сильні сторони конкурентів та визначити, як можна покращити свій продукт або надати йому конкурентну перевагу.

2. Визначення нішевих можливостей: Аналіз аналогічних програм може виявити нішеві можливості, які ще не були задіяні на ринку. Виявлення цих можливостей може допомогти спрямувати розробку продукту у напрямку, який відповідає унікальним потребам або інтересам певної аудиторії.

3. Уникнення повторень помилок: Аналізуючи відомі програми-аналоги, можна виявити помилки або недоліки, які вже були зроблені конкурентами. Це дозволить уникнути повторення цих помилок у власному продукті та забезпечить його кращу якість та користувацький досвід.

4. Визначення трендів індустрії: Аналіз аналогічних програм допомагає визначити поточні тренди та напрямки розвитку відповідної індустрії. Це може бути корисно для планування стратегії розвитку продукту та впровадження нововведень, що відповідають потребам ринку.

5. Створення унікального пропозицій продажів: Шляхом вивчення аналогічних програм можна визначити, як зробити свій продукт унікальним та

привабливим для потенційних користувачів. Це дозволяє сформулювати чітку унікальну пропозицію продажів та залучити увагу цільової аудиторії.

Отже, повертаючись до аналізу, основними двома аналогами є сервіси MyHeritage та FamilySearch. Тому необхідно провести порівняльний аналіз цих двох сервісів.

Огляд існуючих веб-ресурсів та програмного забезпечення:

FamilySearch: Ця безкоштовна генеалогічна платформа належить Церкві Ісуса Христа Святих Останніх Днів і визначається своєю безплатністю та широким доступом до генеалогічних даних. FamilySearch володіє величезною базою даних, що містить мільйони записів про родоводи та інші документи, такі як акти народження, шлюбу та смерті. Він прагне зробити генеалогічне дослідження доступним для всіх, надаючи можливість користувачам внести свій внесок у створення та розширення бази даних.

MyHeritage: Це комерційна платформа з генеалогії та родинних дерев [2], що вирізняється своїм багатством функцій та можливостей для користувачів. MyHeritage пропонує інструменти для дослідження родоводів, створення та управління родинними деревами, а також можливість завантажувати фотографії та історії для збереження спогадів та історій родини.

Функціональність і можливості:

FamilySearch: На платформі FamilySearch користувачі можуть додавати та редагувати інформацію про своїх родичів, будувати генеалогічні дерева, використовувати інструменти для пошуку родоводів та зберігати цифрові копії документів. Крім того, FamilySearch активно використовується для спільної роботи між користувачами, дозволяючи їм спільно працювати над родоводами та обмінюватися інформацією.

MyHeritage: На платформі MyHeritage користувачі можуть додавати та редагувати інформацію про своїх родичів, завантажувати фотографії та історії,

використовувати технології автоматичного пошуку родоводів для швидкого розширення своїх дерев та знаходження нової інформації про своїх предків.

Інтерфейс і взаємодія з користувачем:

FamilySearch: Платформа має простий і зрозумілий інтерфейс, який дозволяє користувачам легко орієнтуватися та використовувати різноманітні функції без необхідності спеціальної підготовки. FamilySearch [3] акцентується на доступності та зручності використання для всіх категорій користувачів. Фрагмент інтерфейсу зображено на рисунку 1.1.

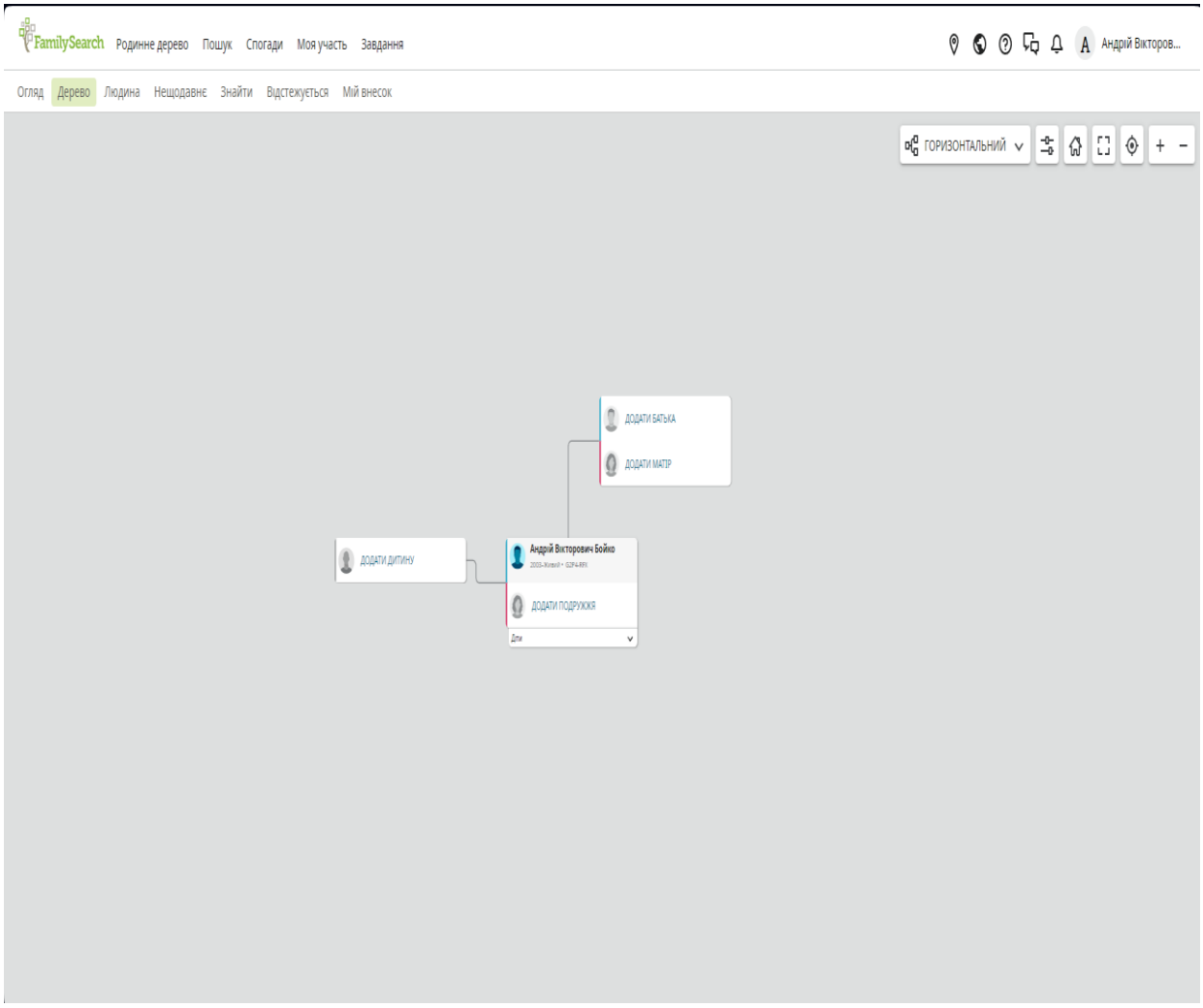

Рисунок 1.1 – Фрагмент інтерфейсу FamilySearch

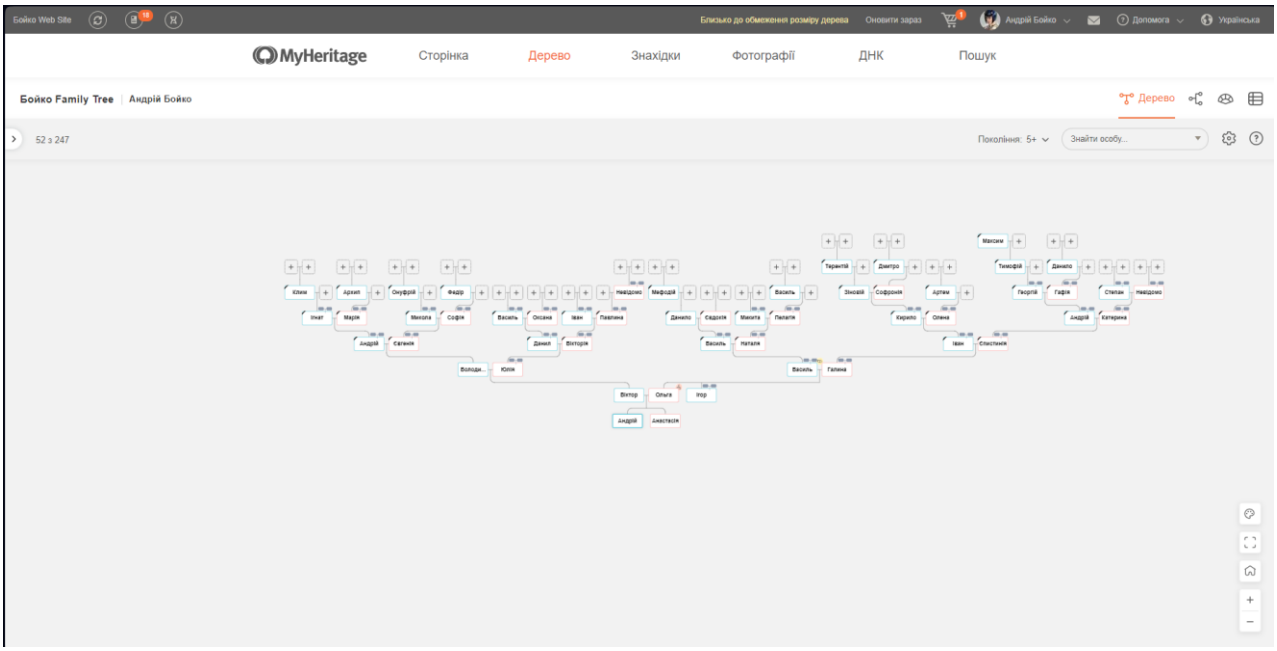

Рисунок 1.2 – Фрагмент дизайну сервісу MyHeritage

MyHeritage: Інтерфейс MyHeritage [1] має сучасний та привабливий дизайн, що привертає увагу користувачів. Фрагмент дизайну зображено на рисунку 1.2. Він пропонує багато інтерактивних елементів та можливостей персоналізації, що дозволяють користувачам налаштовувати платформу під свої потреби та вподобання.

Технологічні аспекти:

FamilySearch: Платформа FamilySearch використовує сучасні технології для зберігання та обробки великих обсягів генеалогічних даних, забезпечуючи при цьому безпеку та конфіденційність особистої інформації користувачів. Вона постійно оновлюється та вдосконалюється з урахуванням останніх досягнень у галузі інформаційних технологій.

MyHeritage: MyHeritage використовує різноманітні технології для автоматичного пошуку родоводів, обробки фотографій та побудови

генеалогічних дерев. Він активно розвивається і впроваджує нові технології для полегшення користування та покращення досвіду користувачів.

Аналіз потреб цільової аудиторії:

FamilySearch: Платформа FamilySearch спрямована на широке коло користувачів, які цікавляться своєю родоводною історією та генеалогією. Вона створена з метою забезпечення доступу до генеалогічних даних та сприяння спільному дослідженню та обміну інформацією між користувачами.

MyHeritage: MyHeritage звертається до користувачів, які шукають інструменти для дослідження своєї родоводної історії та створення родинних архівів. Платформа пропонує різноманітні функції та можливості для задоволення потреб користувачів різних вікових та соціальних груп.

Світовий ринок та тенденції:

Обидва сервіси, FamilySearch та MyHeritage, мають значну кількість користувачів по всьому світу і активно розвиваються, додавши нові функції та покращивши існуючі. Вони реагують на зміни у галузі генеалогії та інформаційних технологій, постійно вдосконалюючи свої продукти та сервіси, щоб задовольняти потреби користувачів і залишатися конкурентоспроможними на світовому ринку.

Також дуже важливим етапом аналізу аналогів даного продукту є оцінка їх недоліків. Розпочнемо із сервісу FamilySearch:

1. Обмежена функціональність для некомерційного користувача: У порівнянні з комерційними платформами, такими як MyHeritage, FamilySearch може мати обмеженіше функціональність та можливості, особливо щодо додаткових функцій, таких як автоматичний пошук родоводів або розширені аналітичні інструменти.

2. Менша база користувачів: У FamilySearch може бути менша кількість активних користувачів порівняно з комерційними платформами, що може вплинути на кількість доступної інформації та можливість знаходження нових зв'язків у родоводі.

3. Менша кількість інтегрованих сервісів: FamilySearch може бути менше інтегрованим з іншими генеалогічними сервісами та джерелами даних порівняно з іншими платформами.

Тепер необхідно розглянути недоліки сервісу MyHeritage:

1. Платні підписки: Деякі з продвинутих функцій та можливостей MyHeritage доступні лише за плату. Це може створювати бар'єр для деяких користувачів, особливо тих, хто шукає безкоштовні альтернативи. Варіанти тарифів зображені на рисунку 1.3.

2. Обмежені можливості для некомерційних користувачів: Деякі функції, такі як розширений пошук та аналітичні інструменти, можуть бути доступні лише за плату або за підпискою, що може обмежити можливості некомерційних користувачів.

3. Залежність від автоматичного збігу: Система автоматичного пошуку родоводів MyHeritage може призводити до неточностей або помилкових збігів, що вимагає уважного перегляду та перевірки результатів користувачами.

4. Конфіденційність даних: Оскільки MyHeritage є комерційною платформою, деякі користувачі можуть висловлювати занепокоєння щодо конфіденційності та безпеки своїх особистих даних.

Оцінивши недоліки аналогів, не варто забувати за їх вдалі рішення. Адже розглянувши правильні підходи до політик цих сервісів можна не лише убезпечити себе від повтору їх помилок, а й перейняти найкращий досвід попередників.

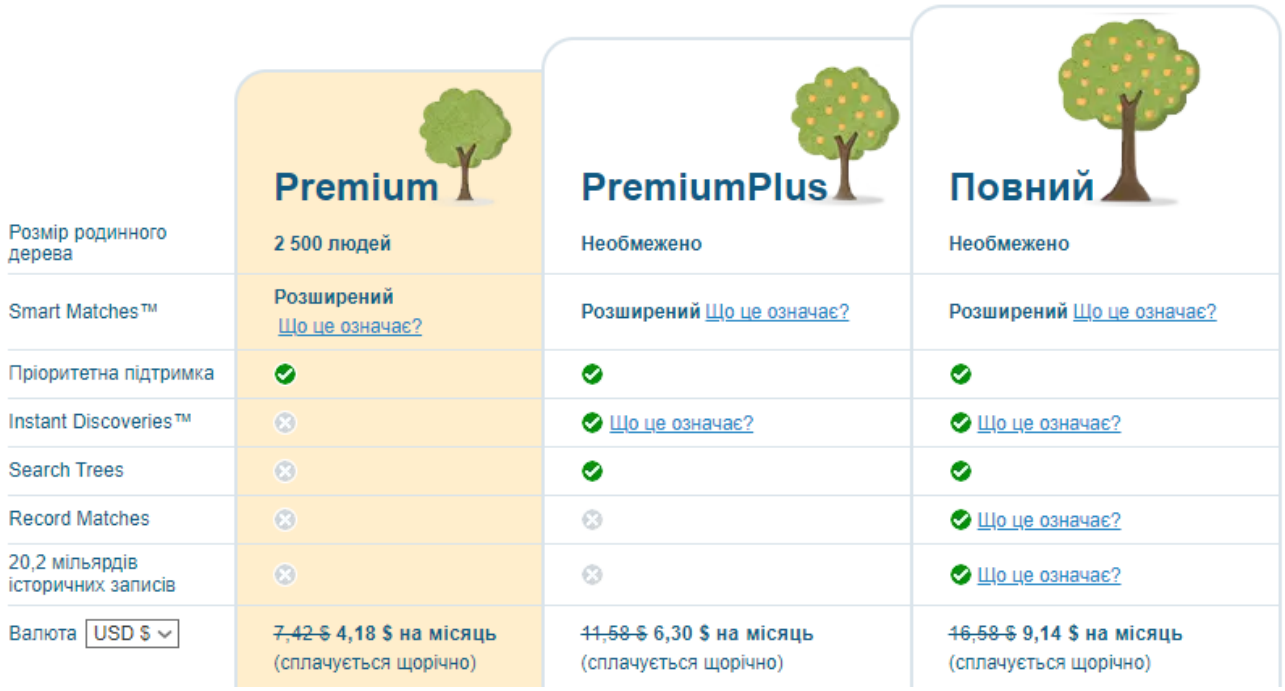

Рисунок 1.3 – Тарифи сервісу MyHeritage

Почнімо із сервісу FamilySearch:

1. Співпраця з добровольцями: FamilySearch активно залучає добровольців з усього світу для індексації та перегляду генеалогічних записів. Це допомагає розширювати базу даних та покращувати точність пошуку для користувачів.

2. Партнерство з іншими організаціями: FamilySearch встановлює партнерські зв'язки з іншими організаціями та архівами для доступу до додаткових джерел генеалогічної інформації, що допомагає розширювати обсяги доступної інформації та покращувати якість досліджень.

3. Оновлення та покращення інтерфейсу користувача: FamilySearch постійно працює над покращенням інтерфейсу користувача, роблячи його більш інтуїтивно зрозумілим та зручним для користувачів різних вікових категорій та рівнів досвіду.

4. Використання машинного навчання та штучного інтелекту: FamilySearch впроваджує технології машинного навчання та штучного інтелекту для полегшення індексації та пошуку генеалогічних записів, що допомагає покращити швидкість та точність пошуку.

Тепер розглянемо вдалі рішення сервісу MyHeritage:

1. Інноваційні технології: MyHeritage активно розвиває та впроваджує нові технології для полегшення генеалогічних досліджень, такі як технологія розпізнавання облич, яка допомагає користувачам ідентифікувати людей на старих фотографіях. Прикладом такої інноваційної технології є функціонал, що дозволяє «оживляти» фотографії родичів, рисунок 1.4.

2. Регулярні оновлення та покращення: MyHeritage постійно вдосконалюється та додає нові функції та можливості для користувачів, забезпечуючи їм широкий спектр інструментів для дослідження та управління родоводами. Приклад інтерфейсу дослідження історичних записів та метричних книг зображено на рисунку 1.5.

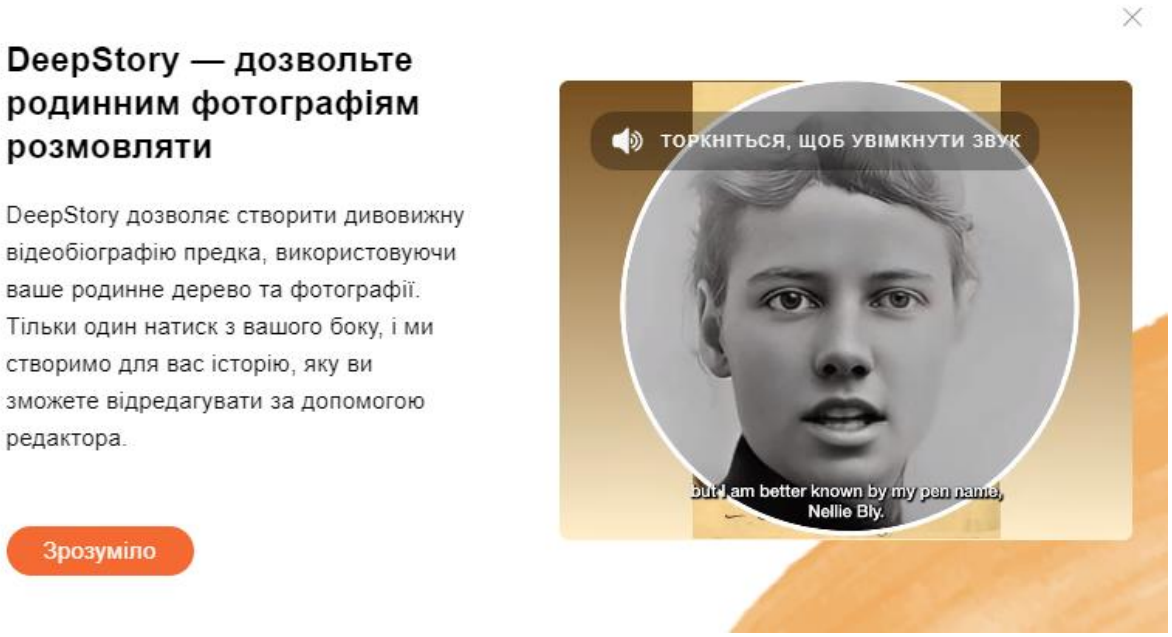

Рисунок 1.4 – Функціонал створення живих фотографій

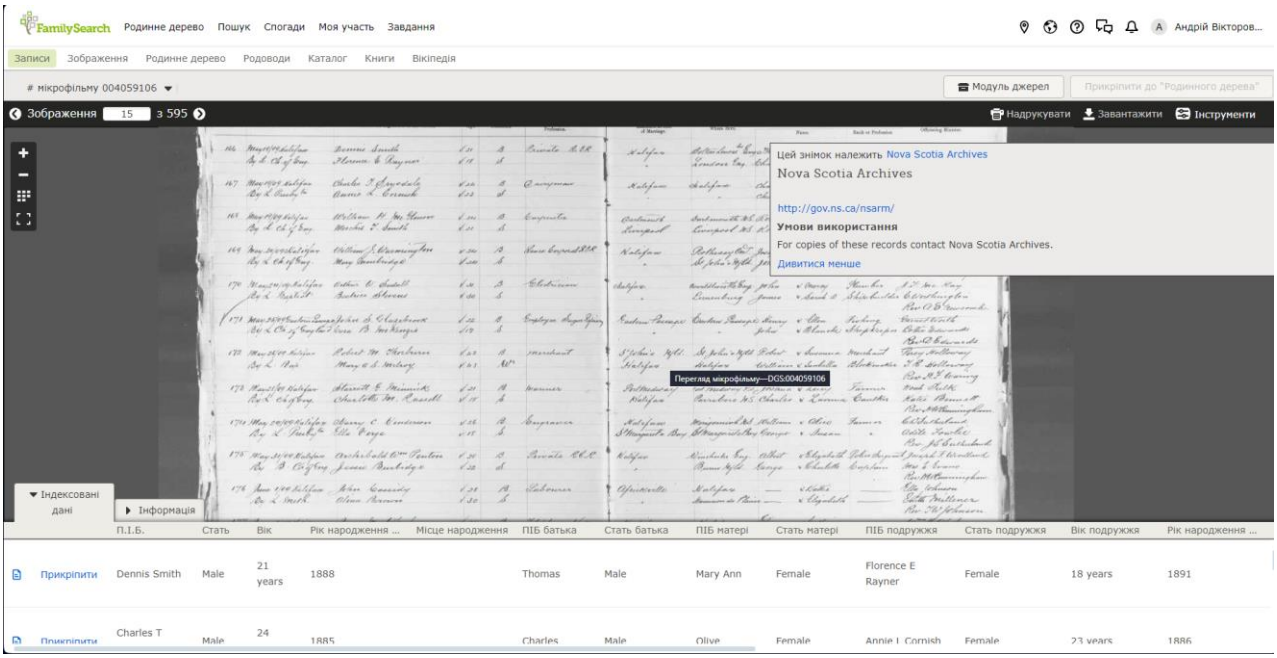

Рисунок 1.5 – Фрагмент інтерфейсу дослідження записів сервісу FamilySearch

3. Збільшення бази даних та джерел інформації: MyHeritage активно працює над розширенням своєї бази даних та партнерством з іншими архівами та організаціями для забезпечення доступу до нових джерел генеалогічної інформації для користувачів. Статистичні дані наведені на сайті FamolySearch зображені на рисунку 1.6.

Рисунок 1.6 – Статистика даних збережених сервісом FamilySearch

4. Забезпечення конфіденційності та безпеки даних: MyHeritage приділяє велику увагу захисту конфіденційності та безпеці особистих даних користувачів, надаючи різні можливості для керування приватністю та захистом даних.

Далі розглянемо переліки функціональних можливостей цих систем, що надаються користувачам у доступ безкоштовно, або ж за платну підписку на додаткові можливості сервісів.

Першим розглянем функціонал сервісу MyHeritage:

1. Створення родинного дерева: Користувачі можуть створювати свої власні генеалогічні дерева, додавати інформацію про предків, родичів та інших членів родини.

2. Пошук інформації про предків: MyHeritage забезпечує доступ до широкого спектру баз даних та джерел, які допомагають користувачам знаходити інформацію про своїх предків.

3. Спільний доступ до дерева: Користувачі можуть дозволити доступ до свого генеалогічного дерева іншим користувачам, щоб спільно працювати над родинним дослідженням.

4. Додавання фотографій та документів: Можливість додавати до генеалогічного дерева фотографії, документи та інші матеріали, що стосуються родини.

5. Автоматичний пошук фотографій предків: MyHeritage пропонує функцію автоматичного пошуку фотографій, яка допомагає ідентифікувати осіб на зображеннях та додавати їх до генеалогічного дерева.

6. Створення родинних історій: Можливість створювати різні історії та звіти про різні події, особливості та аспекти родинної історії.

7. Доступ до генетичних досліджень: MyHeritage також надає можливість здійснювати генетичні дослідження, що дозволяє користувачам дізнатися більше про своє генеалогічне походження та генетичні характеристики.

8. Побудова родинних зв'язків: Функціонал MyHeritage допомагає встановлювати зв'язки між членами родини та відображати їх у вигляді генеалогічного дерева.

9. Спільноти та форуми: Користувачі можуть обмінюватися досвідом, запитувати поради та спілкуватися з іншими учасниками генеалогічної спільноти через форуми та спільноти MyHeritage.

10. ДНК-зіставлення: MyHeritage пропонує можливість порівняння генетичних даних з іншими користувачами та встановлення родинних зв'язків на основі генетичних співпадінь [4].

11. Дослідження етнічного складу: Сервіс надає можливість дізнатися більше про своє етнічне походження шляхом аналізу генетичних даних та зіставлення їх з базами даних з різних регіонів світу.

12. Зберігання та резервне копіювання даних: MyHeritage забезпечує безпечне зберігання генеалогічних даних користувачів та можливість створення резервних копій для захисту від втрати інформації.

13. Доступ до архівних матеріалів: Користувачі мають можливість отримувати доступ до архівних документів, фотографій та інших матеріалів, які можуть бути корисними для родинного дослідження.

14. Календар подій: Функціонал календаря дозволяє користувачам відстежувати важливі дати та події у житті різних членів родини та відображати їх у генеалогічному дереві.

15. Поділ інформації: Можливість обміну даними та інформацією про родину з іншими користувачами MyHeritage шляхом відправки запрошень та дозволів на перегляд.

16. Мобільний додаток: Наявність мобільного додатка дозволяє користувачам отримувати доступ до функціоналу MyHeritage з будь-якого пристрою з підтримкою мобільних платформ.

Далі необхідно розглянути відповідний перелік можливостей сервісу FamilySearch:

1. Створення родинного дерева: Користувачі можуть створювати свої власні генеалогічні дерева, додавати інформацію про предків та родичів.

2. Пошук інформації про предків: FamilySearch надає доступ до обширної бази даних та документів, що дозволяє знаходити інформацію про родину та предків.

3. Доступ до цифрових копій документів: Користувачі можуть переглядати цифрові копії документів, включаючи метричні книги, церковні записи та інші джерела інформації.

4. Робота з іншими користувачами: FamilySearch дозволяє користувачам спільно працювати над деревом та обмінюватися інформацією з іншими дослідниками родинної історії.

5. Пошук унікальних колекцій: Користувачі можуть здійснювати пошук у спеціалізованих колекціях документів та джерел, що дозволяє знаходити унікальні документи та інформацію про предків.

6. Матеріали для навчання: FamilySearch пропонує навчальні матеріали та ресурси для допомоги користувачам в проведенні досліджень родинної історії та розумінні методів пошуку.

7. Додавання фотографій та історій: Користувачі можуть додавати до своїх профілів фотографії, історії та інші матеріали, що стосуються родинної історії.

8. Робота з архівними матеріалами: FamilySearch співпрацює з архівами та бібліотеками по всьому світу, що дозволяє користувачам отримувати доступ до обширних колекцій архівних документів.

9. Дослідження генеалогічного походження: Сервіс дозволяє користувачам дізнатися більше про своє генеалогічне походження шляхом аналізу генетичних даних та дослідження родинних ліній.

10. Пошук ресурсів для родинного історика: FamilySearch надає доступ до різноманітних ресурсів та матеріалів, що допомагають родинним історикам у їх дослідженнях.

11. Спільноти та форуми: Користувачі можуть обмінюватися досвідом, запитувати поради та спілкуватися з іншими учасниками генеалогічної спільноти через форуми та спільноти FamilySearch.

12. Спільні проекти: FamilySearch дозволяє користувачам об'єднуватися для спільних досліджень та проектів з родинної історії, ділитися ресурсами та допомагати один одному у вирішенні складних завдань.

13. Звіти та друковані матеріали: Сервіс дозволяє створювати звіти, друковані матеріали та інші документи на основі зібраної інформації про родину для подальшого використання або обміну з іншими.

14. Інтеграція з іншими сервісами: FamilySearch надає можливість інтеграції з іншими генеалогічними сервісами та додатками, що розширює можливості користувачів у дослідженні родинної історії.

15. Робота з сімейними історіями: Користувачі можуть створювати та зберігати історії своєї сім'ї, додавати важливі події та аспекти життя різних членів родини.

16. Співпраця з музеями та інститутами: FamilySearch співпрацює з різними культурними та науковими установами для збереження та розповсюдження історичних даних та документів.

17. Організація подій та зборів: Сервіс дозволяє користувачам організовувати сімейні збори, зустрічі та події, а також вести облік учасників та планувати програму подій.

18. Робота з управлінням документами: FamilySearch надає інструменти для ефективного управління документами та матеріалами, включаючи можливість створення та організації цифрових архівів.

Отже з визначених переліків можливостей сервісів, можна встановити, що не дивлячись на одну сферу діяльності, ці сервіси спрямовують свої зусилля на розвиток різних напрямків, а саме сервіс MyHeritage спрямовується на розвиток інноваційного функціоналу з використанням штучних інтелектів, а сервіс FamilySearch зосереджений на розвитку своїх баз даних та соціальної взаємодії між людьми та спільного дослідження своїх родоводів.

1.3 Постановка задачі

Виходячи з усього вищесказаного необхідно постановити задачу для розробки WEB-ресурсу управління родоводом.

Задача розробки сервісу управління родоводом полягає у створенні інноваційного та зручного інструменту для користувачів, який би враховував недоліки і вдали рішення сервісів FamilySearch та MyHeritage, а також відповідав сучасним потребам користувачів у генеалогічних дослідженнях.

WEB-ресурс управління родоводом повинен відповідати наступним вимогам:

1. Зручний інтерфейс користувача: Сервіс повинен мати інтуїтивно зрозумілий і зручний інтерфейс, що дозволяє користувачам легко додавати, редагувати та організовувати інформацію про своїх родичів та створювати генеалогічні дерева.

2. Можливість інтеграції з іншими сервісами: Сервіс повинен мати можливість інтеграції з іншими генеалогічними сервісами та джерелами даних для розширення доступної інформації для користувачів.

3. Захист конфіденційності даних: Велика увага має бути приділена захисту особистих даних користувачів і забезпеченню конфіденційності їхніх генеалогічних даних.

4. Функції пошуку та аналізу: Сервіс повинен надати розширені функції пошуку, фільтрації та аналізу генеалогічних даних для полегшення досліджень користувачів.

5. Можливість додавання мультимедійного контенту: Користувачам повинна бути доступна можливість додавати фотографії, відео та інші мультимедійні файли до профілів своїх родичів.

Також під час розробки WEB-ресурсу потрібно опиратись на наступні переваги та недоліки аналогів:

1. Широкий функціонал: Сервіс повинен мати розширений функціонал для дослідження родоводу, включаючи можливості для спільної роботи над деревом, автоматичного пошуку даних та аналізу родоводу.

2. Доступність даних: Велика база даних та доступ до різноманітних джерел генеалогічної інформації є важливою перевагою, оскільки вона забезпечує більш повну картину про предків користувачів.

3. Інноваційні технології: Використання інноваційних технологій, таких як розпізнавання облич, може значно полегшити і прискорити процес дослідження генеалогії.

4. Питання конфіденційності: Питання захисту конфіденційності особистих даних користувачів є критично важливим, оскільки генеалогічні дослідження можуть містити чутливу інформацію про особисте життя.

5. Точність даних: Недостовірна або неточна інформація може призвести до помилкових висновків та невірних з'єднань в генеалогічному дереві, що може призвести до недорозумінь та помилкових даних.

6. Доступність джерел даних: Обмежена доступність джерел генеалогічної інформації може ускладнити процес дослідження, особливо для користувачів з рідкісними або малодоступними джерелами.

1.4 Висновок

Отже за результатами роботи над даним розділом було розглянуто основні дані галузі генеалогії. Також було розглянуто сервіси аналоги до WEB-ресурсу управління родоводом, складено їх порівняльну характеристику та визначено їх переваги та недоліки. Опираючись на ці дані, було постановлено задачу до розробки WEB-ресурсу управління родоводом.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ УПРАВЛІННЯ РОДОВОДОМ

2.1 Обгрунтування засобів розробки системи управління родоводом

Перед початком реалізації будь-якого програмного продукту важливо обдумати та обгрунтувати вибір засобів розробки, які будуть використовуватися під час процесу створення. У цьому підрозділі ми детально розглянемо обрані інструменти та середовища розробки для реалізації системи управління родоводом. Кожен інструмент відіграє важливу роль у процесі розробки, забезпечуючи зручність, ефективність та надійність у роботі над проєктом. Обгрунтування вибору кожного засобу розробки допоможе зрозуміти переваги та перспективи його використання в контексті даного проекту.

1. IDE PhpStorm: PhpStorm [6] обрано в якості основного інтегрованого середовища розробки (IDE) для розробки як фронтенду, так і бекенду системи управління родоводом. Він забезпечує розширені можливості для написання, відлагодження та візуального налагодження коду, що значно полегшує процес розробки.

2. Локальний сервер PhpStorm та OpenServer: PhpStorm використовувався як локальний сервер для забезпечення можливості виконання запитів з фронтенду до бекенду локально. Особливо це було важливо під час розробки різних частин проєкту, де фронтенд та бекенд були окремими сутностями. Для запуску локального сервера бекенду використовувався також OpenServer [7], що дозволило забезпечити надійне середовище для роботи з базою даних та виконання серверного коду.

3. Postman: Postman [8] використовувався для тестування функціональності бекенду. Цей інструмент дозволяє легко відправляти HTTP-

запити до сервера та перевіряти відповіді, що дозволяє ефективно тестувати різні аспекти серверної частини додатку.

Також було розглянуто можливість використання альтернативних систем.

Інтегроване середовище розробки (IDE):

1. Visual Studio Code [9]: Безкоштовне та популярне середовище розробки з підтримкою багатьох мов програмування та численними розширеннями. Воно забезпечує високу продуктивність і гнучкість, але PhpStorm має кращу інтеграцію з PHP і пропонує більше спеціалізованих інструментів для розробки на PHP.

2. Sublime Text [10]: Легкий і швидкий текстовий редактор з можливістю встановлення численних плагінів. Проте, у порівнянні з PhpStorm, Sublime Text менш спеціалізований для PHP-розробки та не має такої потужної підтримки інтеграції з іншими інструментами розробки.

Локальний сервер:

1. XAMPP: Пакет, що включає Apache, MySQL, PHP та Perl. Пропонує просте встановлення та конфігурацію, проте OpenServer забезпечує більше гнучкості та підтримки різних конфігурацій баз даних та серверів.

2. WampServer: Подібно до XAMPP, цей серверний пакет включає Apache, MySQL та PHP для Windows. Хоча WampServer є потужним інструментом, OpenServer має більш інтуїтивний інтерфейс і кращу підтримку різних версій PHP та MySQL, що важливо для тестування та розробки.

Інструмент для тестування API:

1. Insomnia: Альтернатива Postman з простим і зрозумілим інтерфейсом для тестування RESTful API. Однак, Postman пропонує більш широкий набір функцій для автоматизації тестування, інтеграції з CI/CD інструментами та кращу підтримку колаборації.

2. SoapUI: Потужний інструмент для тестування API, який підтримує як RESTful, так і SOAP веб-сервіси. Проте, його інтерфейс може бути складнішим для освоєння у порівнянні з Postman, а також він не такий зручний для швидкого створення та тестування запитів.

Вибір PhpStorm, OpenServer та Postman був обумовлений наступними причинами:

1. PhpStorm: Забезпечує глибоку інтеграцію з PHP та інструментами для веб-розробки, що дозволяє швидше та ефективніше створювати та налагоджувати код.

2. OpenServer: Пропонує зручне управління різними версіями PHP та MySQL, що важливо для тестування різних конфігурацій та забезпечення сумісності.

3. Postman: Надійний інструмент для тестування API з можливістю автоматизації процесів та інтеграції з CI/CD, що робить його незамінним для розробників бекенду.

Використання цих засобів дозволило забезпечити зручне та продуктивне середовище для розробки системи управління родоводом, забезпечивши високу якість коду та функціональність продукту.

2.2 Розробка ER-діаграми

Розробка ER-діаграми є важливим етапом у процесі створення будь-якої бази даних. Entity-Relationship (ER) модель дозволяє візуалізувати структуру даних та зв'язки між ними, що допомагає розробникам зрозуміти та узгодити вимоги до системи з замовниками та іншими зацікавленими сторонами.

Обгрунтування вибору ER-моделі для відображення структури даних полягає у врахуванні її переваг та відповідності потребам проекту системи управління родоводом.

1. Простота та зрозумілість: ER-модель використовує прості концепції сутностей, атрибутів та зв'язків, що робить її легкою для сприйняття як розробниками, так і замовниками проекту. Вона дозволяє візуалізувати структуру даних у вигляді графічної схеми, що полегшує розуміння взаємозв'язків між об'єктами системи.

2. Універсальність: ER-модель може бути використана для будь-якого типу проекту, незалежно від його масштабів та складності. Вона підходить для моделювання відносин між об'єктами будь-якої галузі, що робить її універсальним інструментом для відображення структури даних.

3. Придатність для аналізу вимог: ER-модель дозволяє розробникам та аналітикам чітко визначити сутності, атрибути та їхні взаємозв'язки, що допомагає в аналізі вимог до системи та узгодженні їх зі всіма зацікавленими сторонами.

4. Підтримка процесу розробки: ER-модель може бути використана як основа для подальшого проектування та реалізації бази даних. Вона дозволяє систематизувати та структурувати дані, що сприяє розробці ефективної та надійної системи управління родоводом.

На рисунку 2.3. зображено повну ER-діаграму із промальованими зв’язками між різними модулями системи. ER-діаграма була розроблена з урахуванням майбутнього розширення системи різними модулями з метою забезпечення гнучкості та масштабованості системи. Під час проектування структури бази даних було враховано можливість додавання нових функціональних можливостей та розширення функціональності існуючих модулів без значних змін у базовій структурі даних.

Тому під час розробки проєкту буде використано лише головний модуль зображений на рисунку 2.4.

У головному модулі системи сконцентровані основні функції, пов'язані з взаємодією користувачів з деревами генеалогічних зв'язків. Сюди входить управління деревами та персонами, а також взаємодія між ними. Модуль включає у себе функції додавання, редагування та видалення інформації про осіб, а також управління фотографіями та особистою інформацією персон. Він також враховує взаємозв'язки між персонами, де може бути реалізована можливість, наприклад, додавання родинних зв'язків між особами.

У даному модулі міститься чотирнадцять сутностей, три з яких є асоціативними сутностями, для реалізації зв’язків багато-до-багатьох.

Сутність User має зв’язок один-до-багатьох до сутності Tree, оскільки екземпляр сутності User може мати зв’язки із багатьма екземплярами сутності Tree. Створено саме цей зв’язок через потребу реалізувати користувачу можливість створити кілька різних дерев. Дана потреба може виникнути у користувача у ситуації коли він є професійним генеалогом та повинен дослідити дерева багатьох своїх клієнтів.

У зв’язку із тим, що сутність Person має із сутністю Tree зв’язок багато-до-багатьох, тобто екземпляр сутності Person може міститись у багатьох екземплярах сутності Tree, та екземпляр сутності Tree може містити багато різних екземплярів сутності Person, було додано асоціативну сутність AvailabilityInTree. Завдяки цій сутності зв’язок багато-до-багатьох було перетворено на два зв’язки один-до-багатьох.

Дана реалізація зв’язків зумовлена можливою потребою копіювати персон із одного дерева у інше, оскільки вводити однакову інформацію однієї персони кілька разів у різних деревах може бути довготривалим процесом, за умови наявності багатьох дерев, що мають містити цю персону, або великим обсягом біографічної інформації про цю персону.

Також дана реалізація може стати у пригоді, коли постане потреба реалізувати функціонал пошуку збігів між деревами різних користувачів та їх подальшого дублювання або ж об’єднання.

Сутності Person та Attibute мають зв’язок один-до-багатьох, оскільки один екземпляр сутності Person може містити безліч екземплярів сутності Attribute.

Сутності Attribute та AttributeType мають зв’язок багато-до-одного, адже екземпляр сутності Attribute може мати лише один присвоєний екземпляр сутності AttributeType, а він у свою чергу може бути присвоєним безлічі екземплярів сутності Attribute.

Сутності Attribute та SubAttribute мають зв’язок один-до-багатьох, оскільки екземпляр сутності Attribute може містити безліч екземплярів сутності SubAttribute, а вони в свою чергу можуть мати лише один екземпляр сутності Attribute.

Дана структура сутностей сформована з метою максимізувати гнучкість збереження даних про персону. Ця мета досягнута шляхом розділення кожної події чи факта з життя персони на окремі атрибути, що розділені на певні категорії (типи атрибутів).

Зокрема така структура дозволить у майбутньому реалізувати функціонал імпортування родинного дерева у даний WEB-ресурс із GEDCOM файлу, отриманого із іншого аналогічного сервісу, та функціонал експортування родинного дерева із нашого ресурсу у GEDCOM файл з метою збереження, створення резервної копії або імпорту у сторонню систему управління родоводами. Оскільки GEDCOM файли мають схожу структуру із наведеною, дані процеси буде можливо реалізувати без зайвих проблем та реструктуризації бази даних.

Сутності Person та GalleryPhoto мають зв’язок багато-до-багатьох, оскільки один екземпляр сутності Person може з’являтись у безлічі екземплярів сутності

GalleryPhoto, які у свою чергу можуть кожен можуть бути присвоєні безлічі екземплярів сутності Person. Для усунення зв’язку багато-до-багатьох було створено сутність AvailabilityInPhoto. У результаті між сутностями Person та AvailabilityInPhoto було встановлено зв’язок багато-до-одного, оскільки один екземпляр сутності Person може мати безліч екземплярів сутності AvailabilityInPhoto.

Дана структура необхідна, щоб користувач міг на елементі галереї, що найймовірніше буде фотографією, виділити зони, на яких буде зображено необхідні користувачу персони, та таким чином присвоїти їм ці фотографії.

Сутність Person має із самою собою зв’язок багато-до-багатьох, оскільки один екземпляр сутності Person може бути пов’язаний із безліччю інших екземплярів сутності Person. Щоб вирішити цю проблему, було створено асоціативну сутність Relation. У результаті може здатись, що нічого не змінилось і ми так само маємо зв’язок багато-до-багатьох, але уже із сутностями Person та Relation. Насправді ж цей зв’язок набуває виду зображеного на рисунку 2.1

Згідно нової структури зв’язку між сутностями маємо наступне: було сформовано два різних зв’язки типу один-до-багатьох між сутностями Person та Relation, оскільки один екземпляр сутності Person може мати зв’язок із безліччю екземплярів сутності Relation, а екземпляр сутності Relation може мати лише один екземпляр сутності Person на наявний зв’язок між сутностями.

Сутності Relation та RelationType мають між собою зв’язок багато-до-одного, оскільки екземпляр сутності Relation може мати лише один екземпляр сутності RelationType, а цей екземпляр у свою чергу може мати безліч зв’язків із екземплярами сутності Relation.

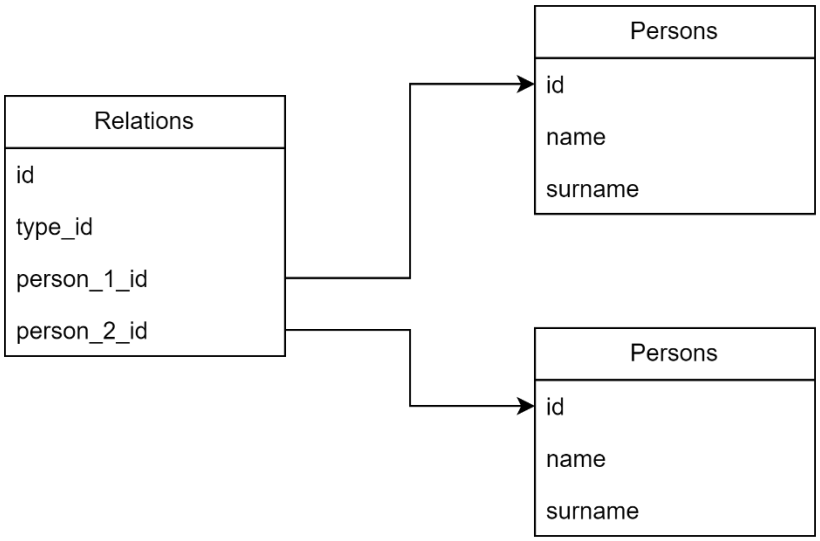

Рисунок 2.1 – Фактична структура зв’язків між сутностями

Сутності Relation та RelationProperty мають зв’язок один-до-багатьох, оскільки один екземпляр сутності Relation може мати безліч екземплярів сутності RelationProperty, а екземпляр сутності RelationProperty може мати лише один екземпляр сутності Relation.

Сутності RelationProperty та EventType мають зв’язок багато-до-одного, оскільки еземпляр сутності RelationProperty може мати лише один екземпляр сутності EventType, який у свою чергу може мати безліч екземплярів сутності RelationProperty.

Дана структура зв’язків персон є подібною до структури атрибутів персон та служить тій самій задачі, але виконує цю задачу над різними категоріями опису персони.

Модуль документообігу, зображений на рисунку 2.5., відповідає за присвоєння персонам з дерева фізичних та електронних документів, та забезпечує зв’язок цих документів із архівами, у яких зберігаються ці документи.

Модуль тарифних планів, зображений на рисунку 2.6., відповідає за управління наданням платних послуг та можливостей користувачам задля економічної ефективності сервіса.

Модуль чатів, зображений на рисунку 2.7., відповідає за роботу чатів між користувачами, враховує можливість створення групових чатів та навіть подальшої розробки системи чат ботів.

Модуль сповіщень користувачів, зображений на рисунку 2.8., відповідає за збереження та відправку користувачам сповіщень різними шляхами, для інформування користувачів про зміни у роботі сервісу, його політик, подій сервісу та генеалогії в цілому, для подачі користувачу корисної інформації з пошуку родичів для його родоводу.

Також на рисунку 2.2 зображено допоміжні сутності, що не можна віднести до конкретного модулю, але мають корисний та необхідний функціонал.

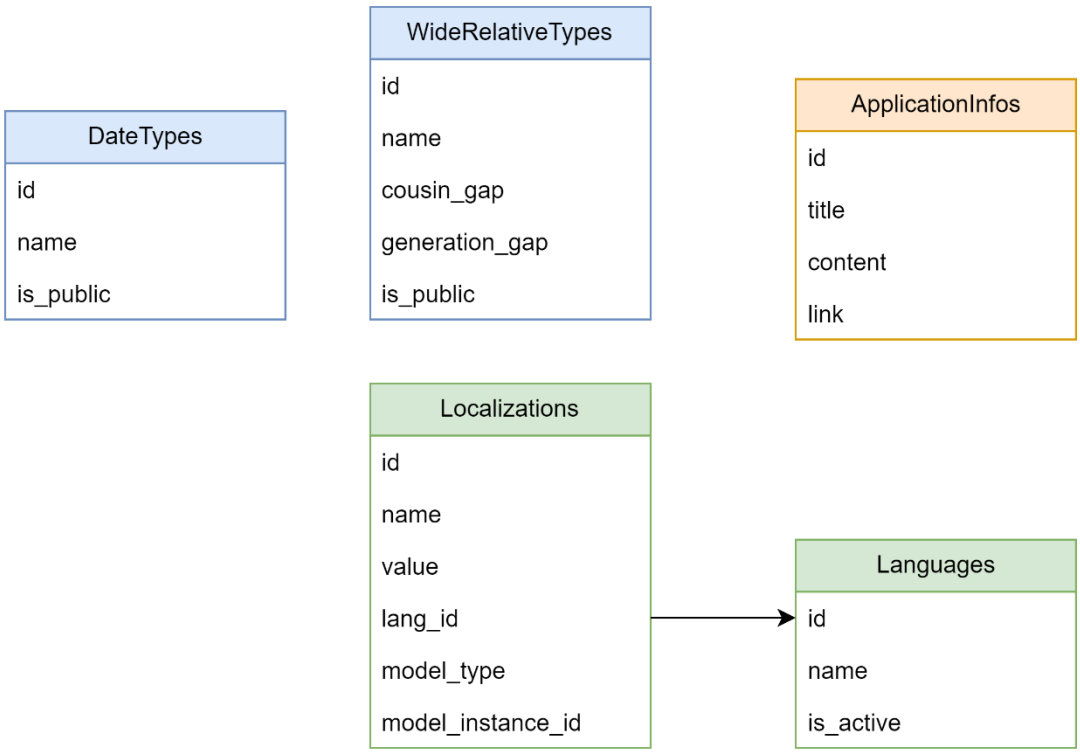

Рисунок 2.2 – Допоміжні сутності

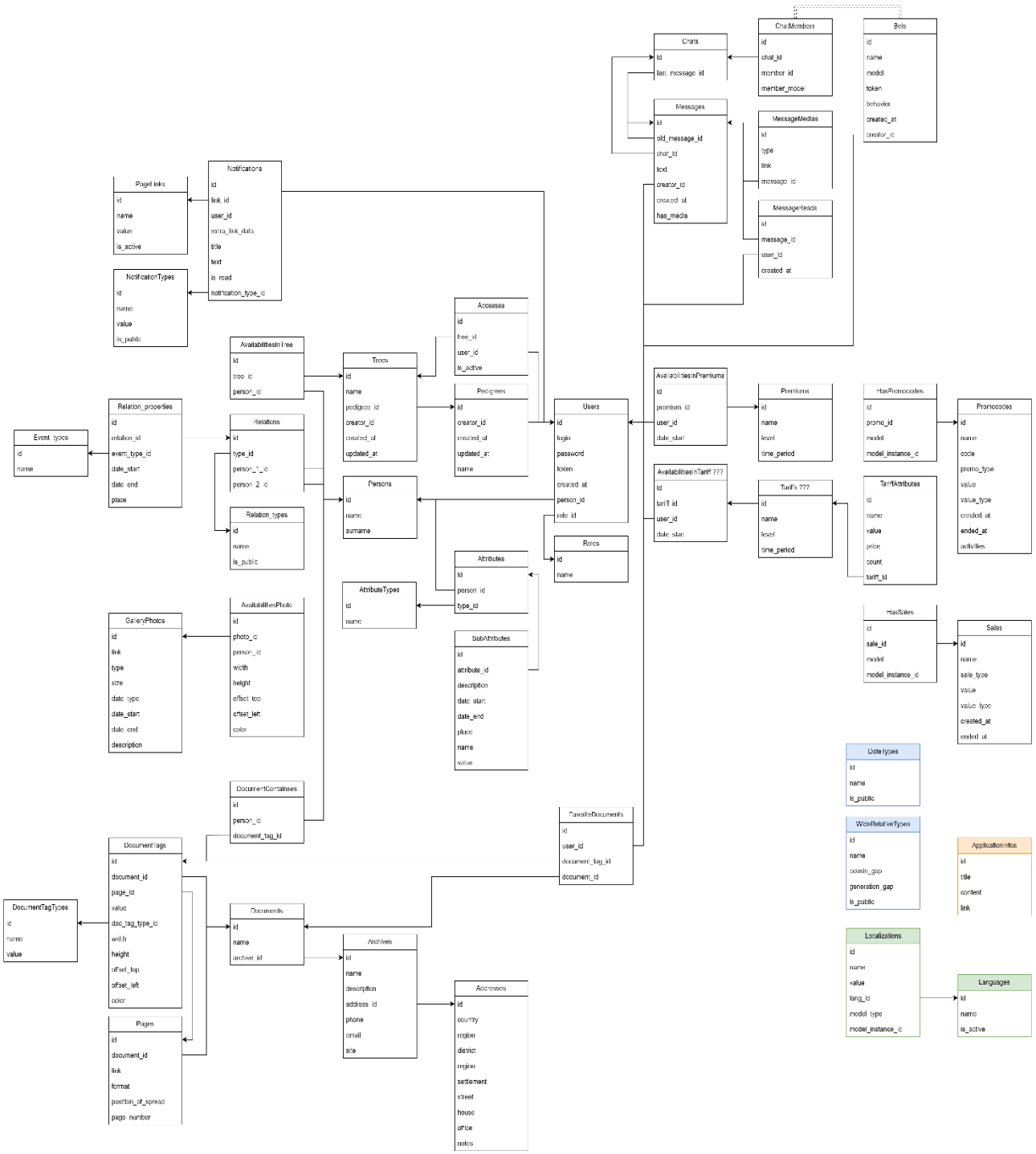

Рисунок 2.3 – Повна ER-діаграма

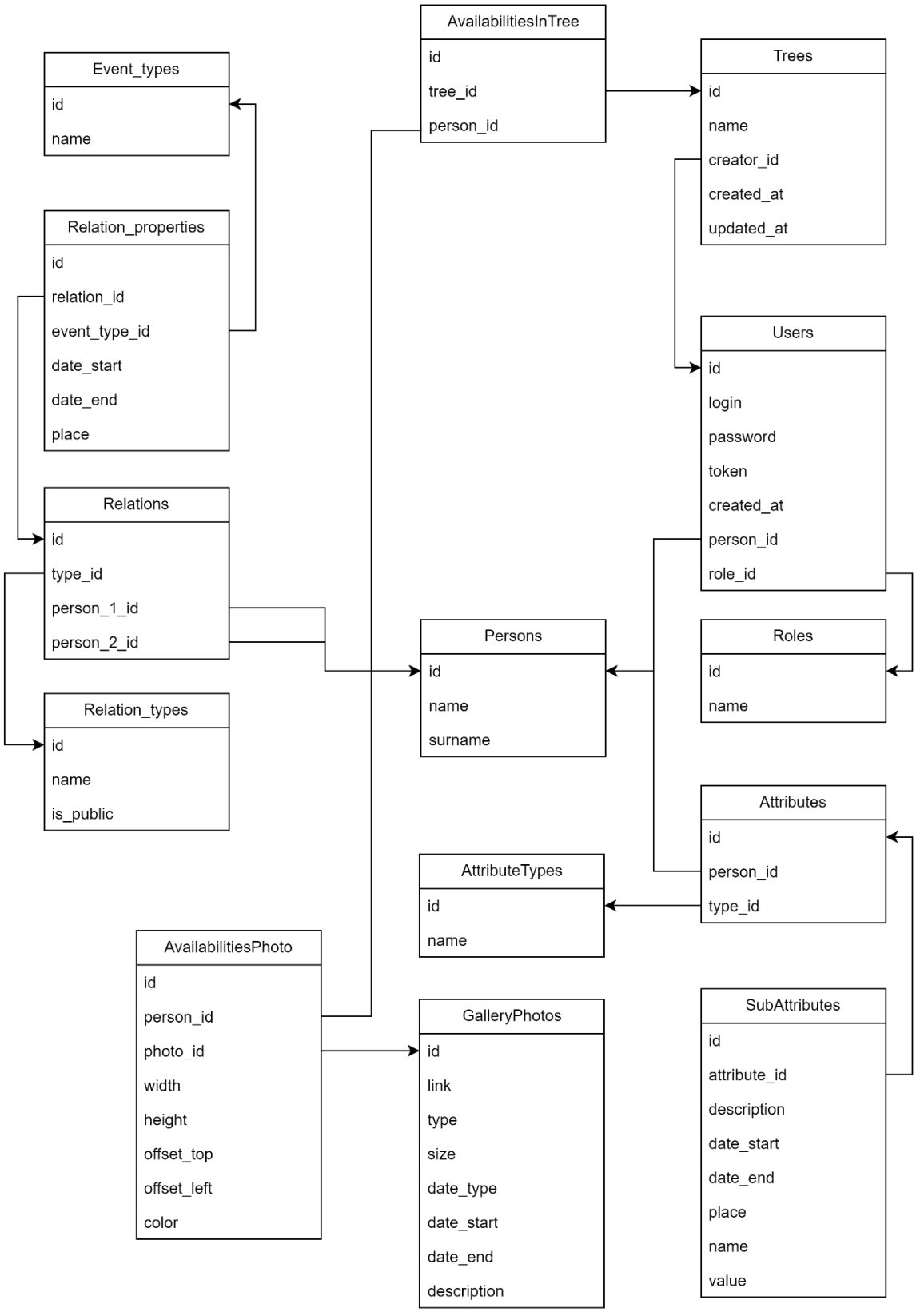

Рисунок 2.4 – Модуль керування деревом (Головний модуль)

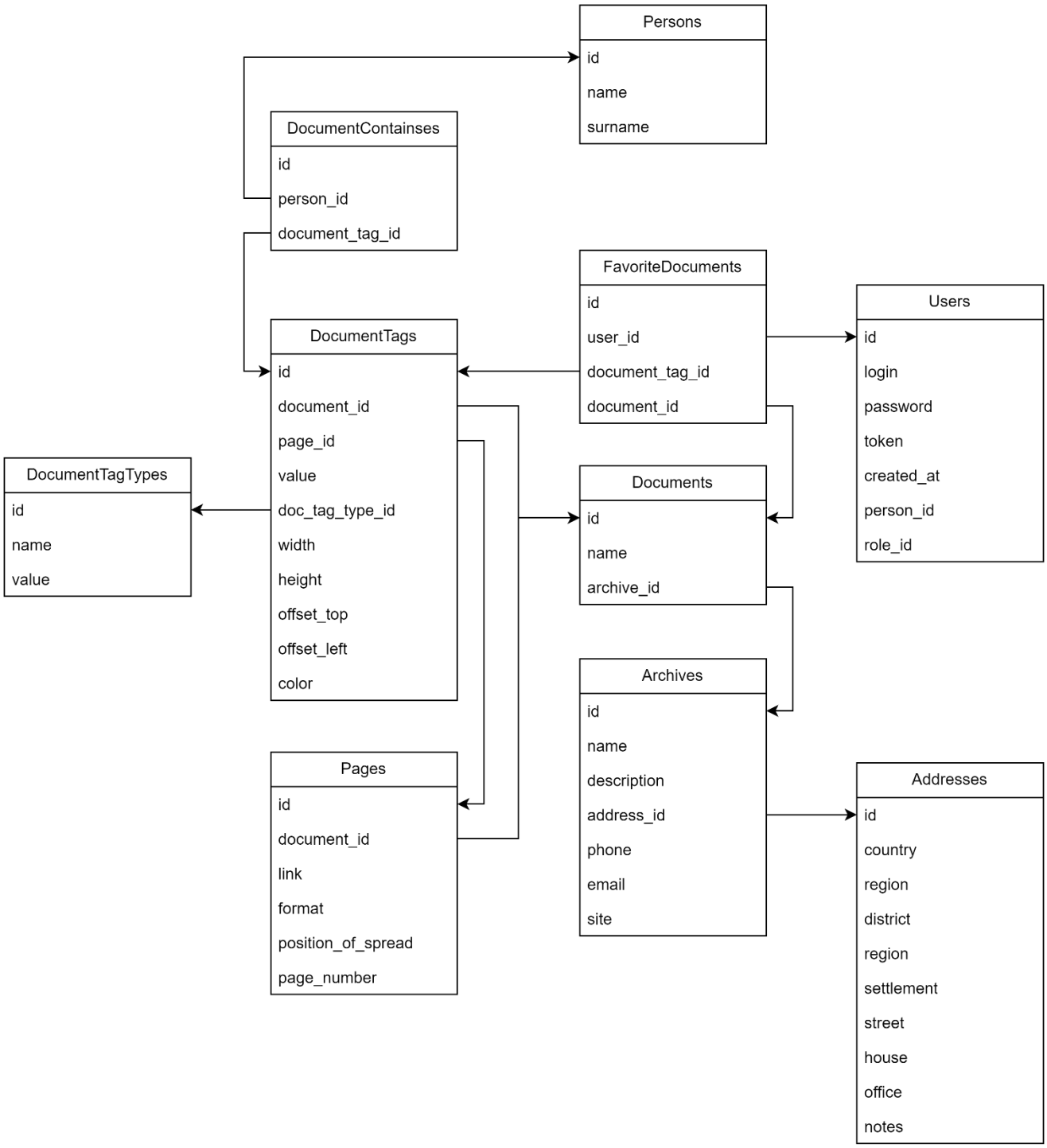

Рисунок 2.5 – Модуль документообігу

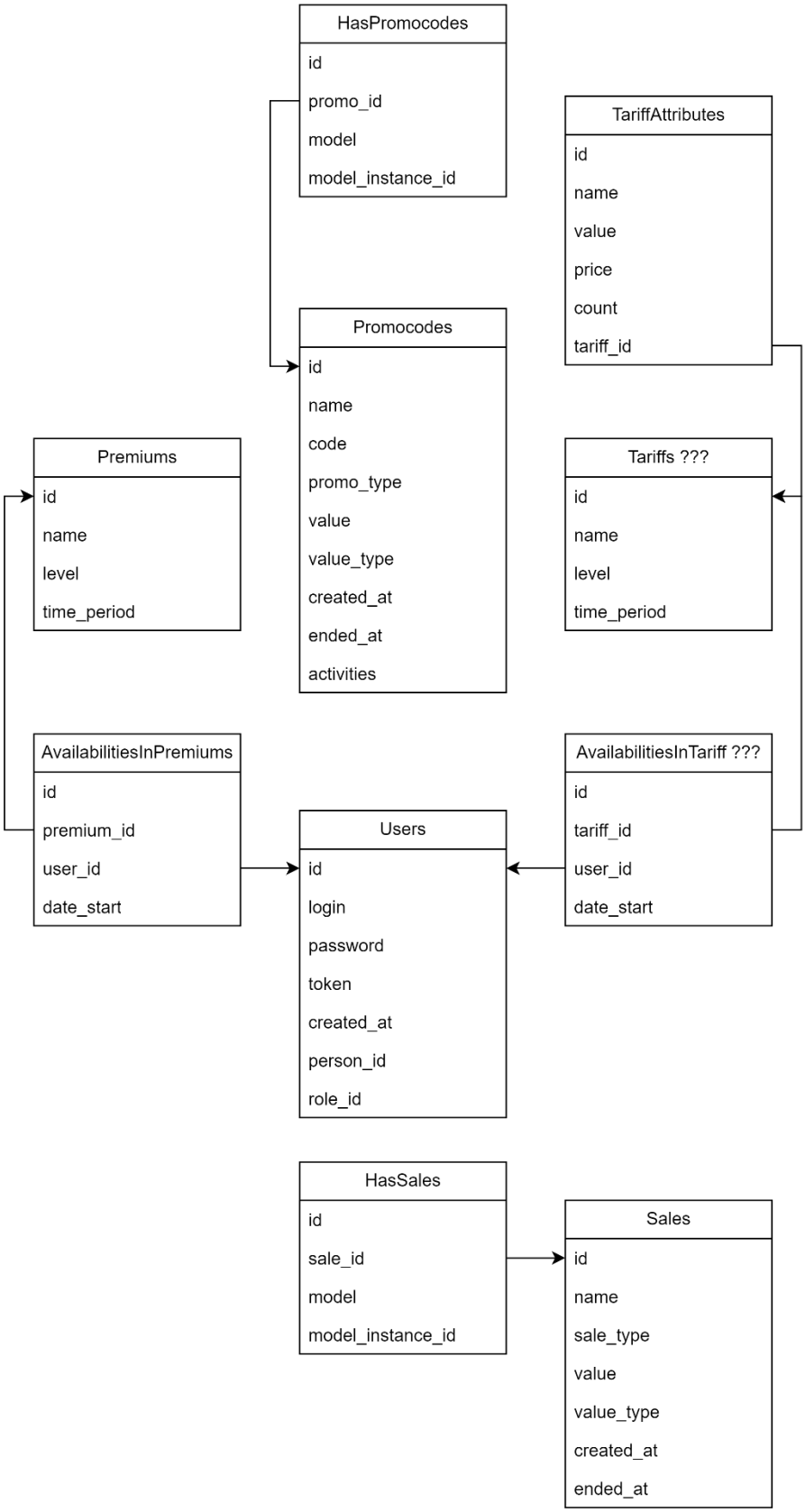

Рисунок 2.6 – Модуль тарифних планів

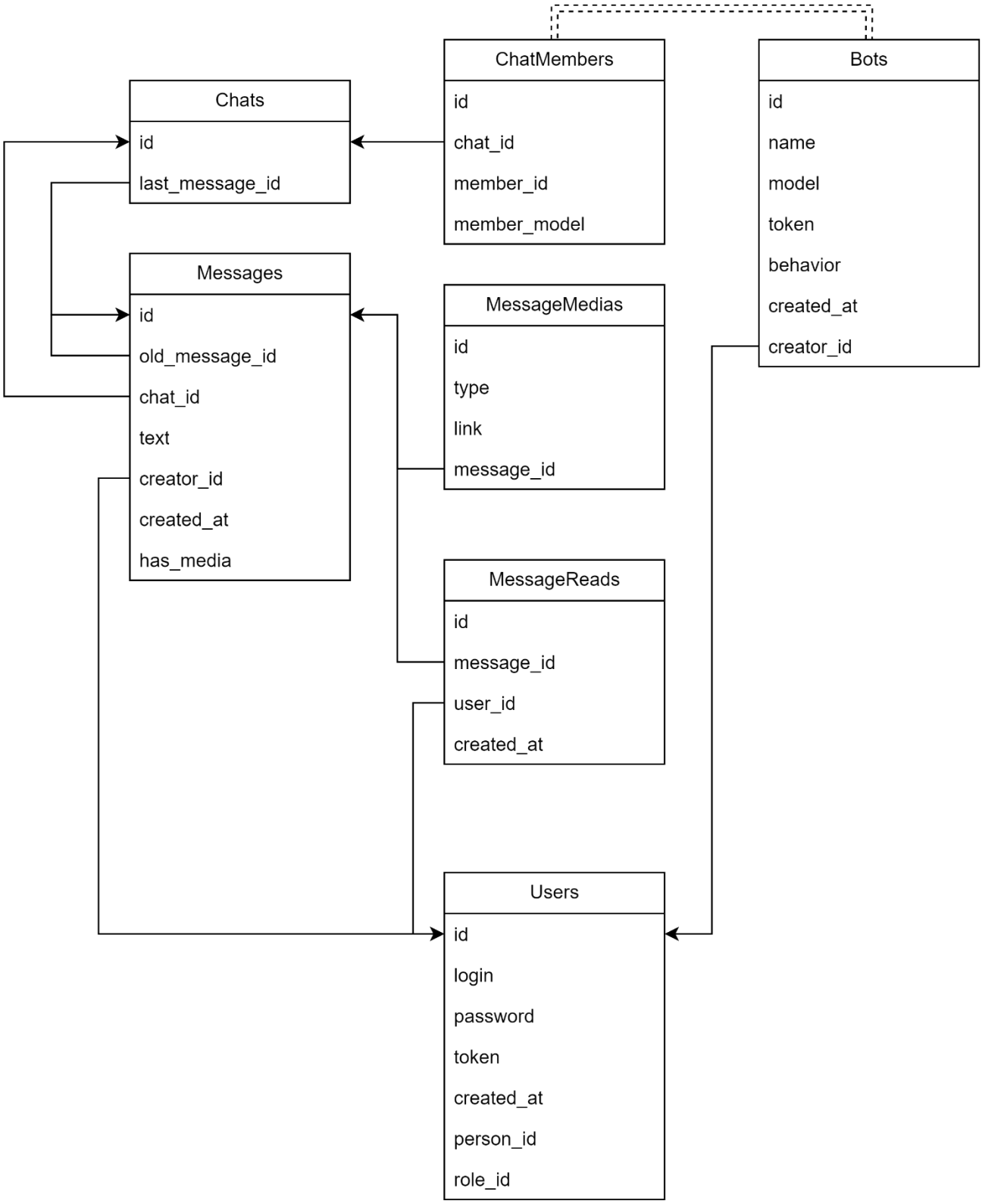

Рисунок 2.7 – Модуль чатів

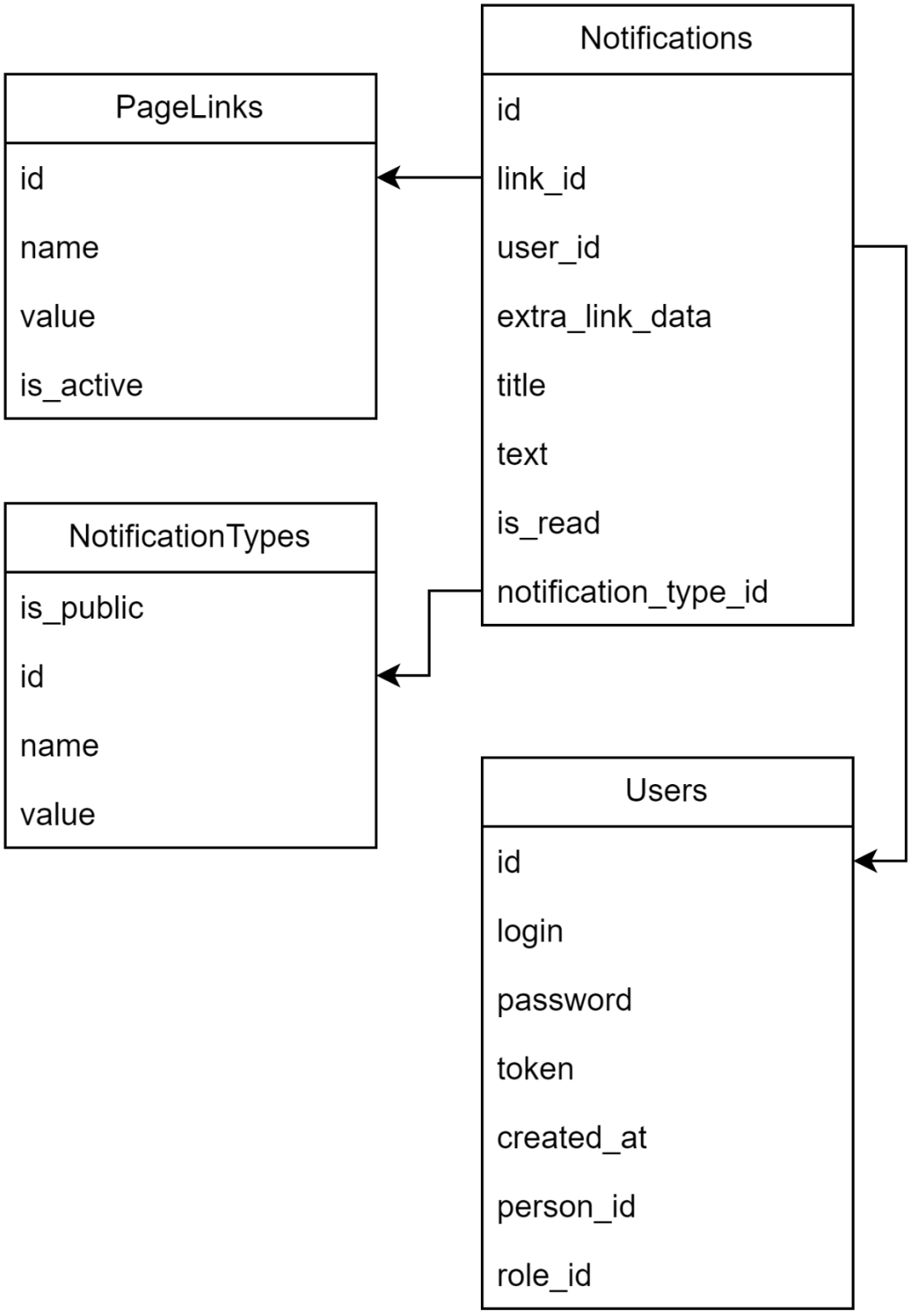

Рисунок 2.8 – Модуль сповіщень користувачів

2.3 Проєктування основних алгоритмів

Проектування алгоритму видачі даних про дерева роду на фронт є критично важливим етапом у розробці веб-ресурсу управління родоводом. Цей процес забезпечує чітку та логічну структуру даних, що полегшує їх обробку і відображення на фронтенді, дозволяючи користувачам легко і швидко переглядати родоводи.

Важливим аспектом є оптимізація продуктивності алгоритму, що мінімізує затримки при передачі даних з сервера на клієнт, забезпечуючи швидке завантаження та відображення інформації. Це особливо важливо при роботі з великими обсягами даних, де затримки можуть негативно вплинути на користувацький досвід.

Алгоритм, зображений на рисунку 2.9, відпрацьовує на сервері у відповідь на запит клієнтської частини. Його функція полягає у отримані даних дерева користувача, їх обробці та форматуванні потребам клієнської частини, та створенні відповіді на запит.

Цей алгоритм під час обробки даних виконує інший алгоритм, зображений на рисунку 2.10, необхідний для обрахунку поколінь кожної персони дерева. Він полягає у визначенні крайніх (найстарших) персон дерева, та перебору пов’язаних із ними персон, та їх подальшу обробку у такий ж спосіб. Якщо пов’язані персони є персонами типу “spouse” або “sibling” їх покоління встановлюється рівним обраній персоні. Якщо пов’язані персони є персонами типу “child” їх покоління встановлюється більшим за покоління обраної персони. Якщо ж пов’язані персони є персонами типу “parent” їх покоління встановлюється меншим за покоління обраної персони.

У випадку появи конфлікту поколінь під час виконання цього алгоритму, тобто якщо для персони під час перебору від одного найстаршого предка було вказано одне покоління, а під час перебору від іншого предка інше, обирається

пріорітетним покоління із найбільним номіналом, і всі предки поточної персони зазнають перерахунку поколінь відповідно до нового пріорітету.

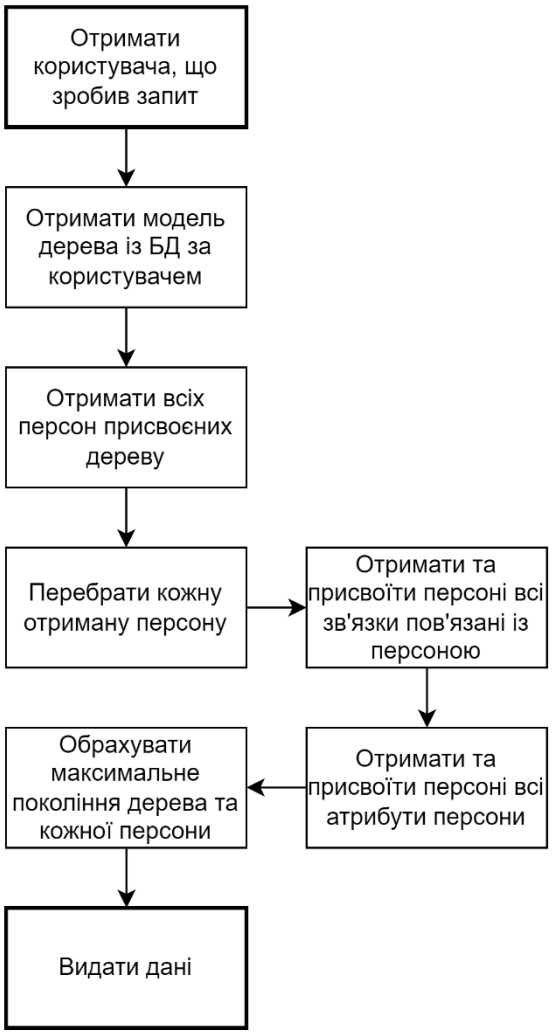

Рисунок 2.9 – Алгоритм для видачі даних дерева на клієнтську частину

Після отримання даних із серверної частини, клієнтська частина повинна застосувати ці дані для рендеру дерева роду. Алгоритм згідно якого відбувається цей рендер зображено на рисунку 2.11.

Спочатку відбувається рендер вузлів усіх персон дерева. Далі відбувається рендер персон, що є нащадками відносно головної персони дерева знизу до верху,

тобто від наймолодших персон до найстарших. Опісля відбувається рендер персон, що є предками відносно головної персони, зверху донизу, тобто від найстарших до наймолодших персон.

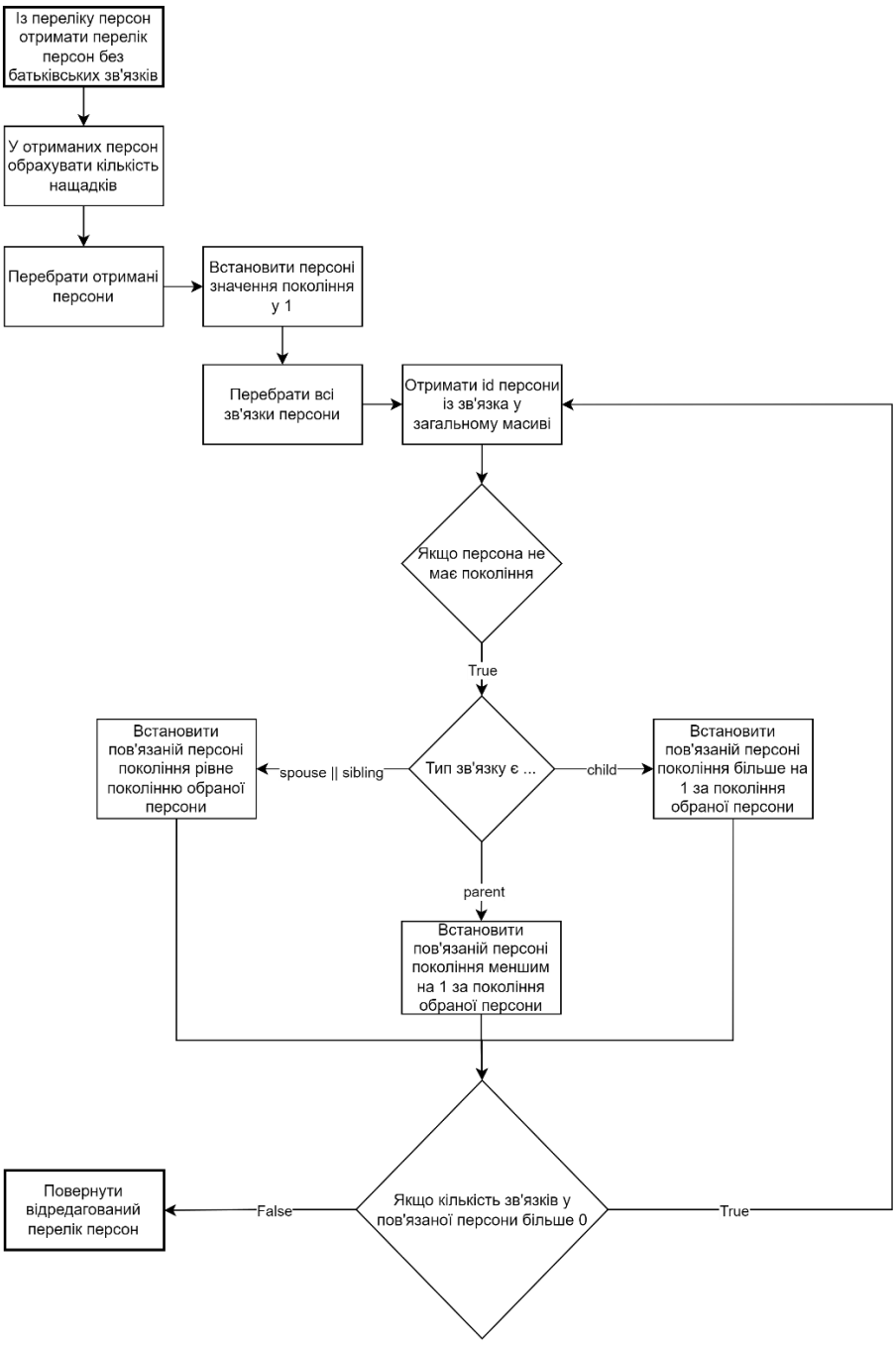

Рисунок 2.10 – Алгоритм обрахунку поколінь персон дерева

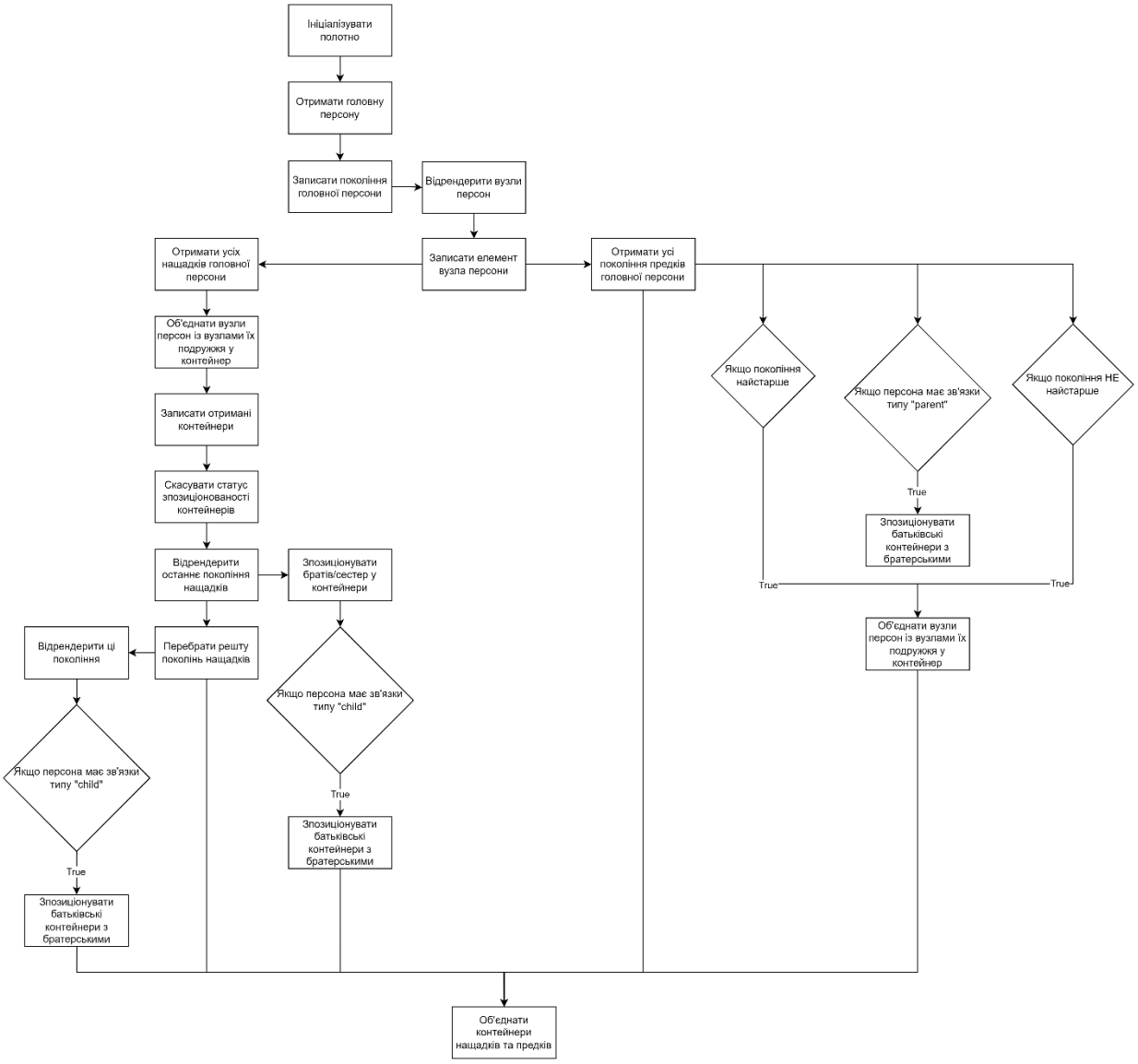

Рисунок 2.11 – Алгоритм рендеру дерева на клієнтській частині

Такий порядок рендеру та розділення персон на “нащадків” та “предків” зумовлена потребою відрендерити персон так, щоб жодна з них не перекривала одна одну. А оскільки дерево має вигляд пісочного годинника, тобто найширше на краях дерева та звужується на рівні головної персони, необхідно починати рендер із найширших країв дерева.

Після цього всього дві частини дерева об’єднуються у єдине дерево роду.

2.4 Висновок

Отже за результатами роботи над даним розділом було розглянуто та обґрунтовано обрано засоби для розробки системи управління родоводом. Також у цьому розділі було розроблено ER-діаграму бази даних проєкта та детально описано головний модуль проєкта. Як наслідок, далі було розроблено та описано алгоритми рендеру дерева роду та компонування даних дерева роду і їх видачі на клієнтську частину проєкта.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ УПРАВЛІННЯ РОДОВОДОМ

3.1 Обґрунтування вибору мови програмування

У процесі розробки будь-якого програмного продукту, вибір мови програмування є одним з ключових рішень, що впливають на всі етапи створення системи: від проектування і розробки до тестування та підтримки. Вибір мови програмування визначає, наскільки ефективно можна реалізувати необхідний функціонал, як швидко і легко буде здійснюватися розробка, а також які ресурси будуть потрібні для підтримки і масштабування системи. Тому обґрунтування цього вибору є важливим етапом у процесі створення програмного забезпечення.

WEB-ресурс управління родоводом складається з двох основних частин: серверної та клієнтської. Серверна частина відповідає за обробку запитів, управління базою даних та бізнес-логіку, тоді як клієнтська частина займається відображенням інформації та взаємодією з користувачем. Відповідно, для кожної з цих частин необхідно обрати окремі мови програмування, які найкраще підходять для виконання відповідних завдань.

JavaScript є основною мовою програмування [11] для створення інтерактивних веб-сторінок і додатків. Він дозволяє розробникам додавати динамічний контент, реагувати на дії користувачів та взаємодіяти з іншими веб-сервісами.

Переваги:

– Широке використання і підтримка у всіх сучасних браузерах,
– Велика кількість бібліотек і фреймворків, що полегшують розробку,
– Можливість створення динамічних і інтерактивних веб-сторінок.

Недоліки:

– Можливість виникнення проблем з продуктивністю при обробці великих обсягів даних,

– Деяка складність у налагодженні та відстеженні помилок.

HTML (HyperText Markup Language) є стандартною мовою розмітки для створення веб-сторінок. Вона визначає структуру контенту на веб-сторінці.

Переваги:

– Основа веб-розробки, підтримується всіма браузерами,

– Простота у використанні та розумінні,

– Семантична розмітка, що покращує SEO та доступність.

Недоліки:

– Обмеженість у створенні динамічного контенту без JavaScript,

– Необхідність використання CSS для стилізації.

CSS (Cascading Style Sheets) є мовою стилів для опису презентації документа, написаного мовою розмітки, такої як HTML. CSS описує, як елементи повинні відображатися на екрані, папері або в інших медіа.

Переваги:

– Відокремлення стилізації від структури, що полегшує підтримку і розробку,

– Підтримка адаптивного дизайну для різних пристроїв.

Недоліки:

– Можливі проблеми з кросбраузерною сумісністю,

– Складність в управлінні великими стилями без використання препроцесорів.

SvelteKit є сучасним фреймворком для побудови веб-додатків [12], який використовує компіляцію компонентів на етапі збірки для досягнення високої продуктивності.

Переваги:

– Висока продуктивність завдяки компіляції компонентів на етапі збірки,

– Простий синтаксис і легкість у вивченні,

– Вбудована підтримка для серверного рендерингу та генерації статичних сайтів.

Недоліки:

– Відносно новий фреймворк, тому менша спільнота і кількість готових рішень,

– Можливі проблеми з інтеграцією деяких сторонніх бібліотек.

SCSS (Sassy CSS) є препроцесором CSS, який додає додаткові можливості, такі як змінні, вкладеність, міксини та інші, що робить CSS більш динамічним і зручним у використанні [13].

Переваги:

– Додаткові можливості (змінні, вкладеність, міксини), які спрощують управління стилями,

– Підвищена продуктивність і підтримуваність коду.

Недоліки:

– Необхідність компіляції у звичайний CSS,

– Додаткова складність у налаштуванні інструментів для компіляції.

React є популярною бібліотекою для побудови користувацьких інтерфейсів [14]. Він дозволяє створювати компонентно-орієнтовані додатки з високою продуктивністю.

Переваги:

– Велика спільнота і підтримка,

– Широкий вибір бібліотек і інструментів,

– Підтримка компонентного підходу.

Недоліки:

– Більш складний синтаксис у порівнянні зі Svelte,

– Більший розмір бандла через додаткову абстракцію.

Angular є повноцінним фреймворком для побудови веб-додатків, який надає вбудовані рішення для багатьох задач, таких як управління станом, роутинг та багато іншого [15].

Переваги:

– Повноцінний фреймворк з вбудованими рішеннями для багатьох задач,

– Підтримка TypeScript.

Недоліки:

– Складність у вивченні та використанні,

– Великий розмір бандла.

Vue.js є прогресивним фреймворком для побудови користувацьких інтерфейсів [16]. Він відомий своєю легкістю у вивченні та гнучкістю.

Переваги:

– Легкість у вивченні,

– Підтримка компонентного підходу,

– Гнучкість у налаштуванні.

Недоліки:

– Менша спільнота у порівнянні з React,

– Можливі проблеми з масштабованістю для великих проектів.

Для розробки клієнтської частини системи управління родоводом були обрані JavaScript, HTML, та CSS разом з фреймворком SvelteKit та препроцесором SCSS. Ці технології забезпечують високу продуктивність, зручність у використанні та підтримці, а також гнучкість і адаптивність для розробки сучасного веб-додатку. Незважаючи на наявність альтернатив, саме ці

інструменти найкраще відповідають вимогам проекту і забезпечують ефективне вирішення поставлених задач.

3.2 Створення серверної частини web-ресурсу управління родоводом

Серверна частина web-ресурсу управління родоводом виконує ключову роль у функціонуванні системи. Вона відповідає за обробку запитів від клієнтської частини, управління базою даних, реалізацію бізнес-логіки та забезпечення безпеки і доступності даних.

Правильне проектування та реалізація серверної частини є критично важливими для забезпечення стабільної роботи всього web-ресурсу. Від ефективності роботи серверної частини залежить швидкість відповіді на запити користувачів, безперебійність роботи системи, а також можливість масштабування і подальшого розвитку проекту.

Архітектура серверної частини web-ресурсу управління родоводом побудована на фреймворку Laravel, який забезпечує структурованість і масштабованість додатку [17]. Laravel дотримується архітектурного шаблону MVC (Model-View-Controller), що дозволяє чітко розділяти логіку додатку, його презентацію та управління даними.

– Проект Laravel структурований таким чином, щоб забезпечити максимальну зручність у розробці та підтримці коду. Основні каталоги проекту включають:

– app/: містить основний код додатку, включаючи моделі, контролери та інші компоненти.

– config/: містить конфігураційні файли додатку.

– database/: містить файли міграцій для створення та зміни структури бази даних.

– routes/: містить файли маршрутизації, де визначаються всі маршрути додатку.

– resources/: містить ресурси додатку, такі як шаблони електронної пошти та інші статичні файли.

– public/: містить файли, до яких можна отримати доступ безпосередньо через веб-сервер, такі як зображення та інші статичні ресурси.

До основних компонентів серверної частини належать:

– Моделі. Моделі відповідають за роботу з базою даних. Вони визначають структуру даних, встановлюють взаємозв'язки між таблицями та забезпечують зручний доступ до даних через ORM (Object-Relational Mapping) Eloquent. Кожна модель представляє конкретну таблицю бази даних і дозволяє виконувати операції створення, читання, оновлення та видалення записів.

– Контролери. Контролери відповідають за обробку запитів, які надходять від клієнтської частини. Вони визначають бізнес-логіку додатку і керують даними, отриманими з моделей. Контролери приймають запити, викликають відповідні методи моделей, обробляють дані і повертають відповіді у вигляді JSON, які потім використовуються на фронтенді.

– Маршрути. Маршрути визначають, які дії виконуються при зверненні до певних URL-адрес. У Laravel маршрути задаються у файлах маршрутизації, де вказуються URL і відповідні контролери з методами. Це забезпечує гнучкість і легкість у налаштуванні обробки запитів.

Компоненти серверної частини тісно взаємодіють між собою для забезпечення коректної роботи додатку. Коли клієнтська частина надсилає запит до сервера, маршрутизатор визначає, який контролер і метод слід викликати для обробки цього запиту. Контролер, у свою чергу, взаємодіє з моделями для отримання або зміни даних у базі даних. Після обробки запиту контролер формує відповідь і повертає її клієнту.

Така архітектура забезпечує чіткий розподіл відповідальностей між компонентами, спрощує розробку і підтримку коду, а також робить додаток більш гнучким і масштабованим. В наступних розділах буде детально розглянуто процес розробки кожного з цих компонентів та їх інтеграція в рамках серверної частини web-ресурсу управління родоводом.

Розробка моделей та налаштування бази даних є важливим етапом у створенні серверної частини web-ресурсу управління родоводом.

Створення моделей у Laravel починається з виконання команди, описаної у лістингу 3.1, яка генерує файл моделі у каталозі app/Models.

Лістинг 3.1 – Шаблон команди для створення моделі

```
php artisan make:model ModelName
```

Наприклад, для створення моделі Person, яка представляє особу у родоводі, виконується команда, що описана у лістингу 3.2. У файлі моделі визначаються атрибути, що відповідають колонкам таблиці в базі даних, а також взаємозв'язки з іншими моделями.

Лістинг 3.2 – Команда для створення моделі Person

```
php artisan make:model Person
```

У результаті опису моделі Person, фрагмент коду якої зображено у лістингу 3.3, було задано базовий формат дати за допомогою функції serializeDate(), що тепер відображатиметься при отриманні екземпляра моделі..

Лістинг 3.3 – Команда для створення моделі Person

```
class Person extends Model
{
    use HasFactory;

    protected $table = 'persons';

    protected $fillable = [
        'first_name',
        'last_name',
    ];

    protected function serializeDate(DateTimeInterface $date){
        return $date->format('d-m-Y');
    }
    public function relations(){
        ...
    }
}
```

Також у моделі було реалізовано функцію relations(), зображену у окремому лістингу 3.4, що знаходить та формує для видачі дані про всі зв’язки екземпляра моделі, та використовує при цьому інші моделі проєкту, а саме моделі RelationType та моделі Relation

Лістинг 3.4 – Функція отримання зв’язків персони

```
    public function relations(){
            $parent_child_type = RelationType::where('name',
'parent-child')->first();
            $parentsRaw = Relation::where([['type_id',
$parent_child_type->id], ['last_person_id', $this->id]])->get();
            $parents = [];
            foreach ($parentsRaw as $parent) {
                $parents[] = [
                    'type' => 'parent',
                    'id' => $parent->first_person_id
                ];
            }
            $childrenRaw = Relation::where([['type_id',
$parent_child_type->id], ['first_person_id', $this->id]])->get();
            $children = [];
            foreach ($childrenRaw as $child) {
                $children[] = [
                    'type' => 'child',
                    'id' => $child->last_person_id
                ];
            }
            $sibling_type = RelationType::where('name',
'sibling')->first();
            $siblingsRaw = Relation::where([['type_id',
$sibling_type->id], ['first_person_id', $this->id]])->get();
            $siblings = [];
            foreach ($siblingsRaw as $sibling) {
                $siblings[] = [
                    'type' => 'sibling',
                    'id' => $sibling->last_person_id
                ];
```

```
            }
            $spouse_type = RelationType::where('name', 'spouse')-
>first();
            $spousesRaw = Relation::where([['type_id',
$spouse_type->id], ['first_person_id', $this->id]])->get();
            $spouses = [];
            foreach ($spousesRaw as $spouse) {
                $spouses[] = [
                    'type' => 'spouse',
                    'id' => $spouse->last_person_id
                ];
            }
            $response = array_merge($parents, $children,
$siblings, $spouses);
            return $response;
        }
```

Laravel використовує систему міграцій для управління структурою бази даних. Міграції дозволяють створювати і змінювати таблиці бази даних програмним шляхом, що спрощує підтримку і розгортання додатку.

Приклад команди для створення файлу міграції описано у лістингу 3.5.

Лістинг 3.5 – Приклад команди для створення міграції

```
php artisan make:migration create_persons_table
```

Для реалізації у базі даних таблиці persons було створено міграцію вищезазначеною командою. Код створеного файлу міграції зображено у лістингу 3.6.

Лістинг 3.6 – Приклад файлу міграції

```
class CreatePersonsTable extends Migration{
    public function up()
    {
        Schema::create('persons', function (Blueprint $table) {
            $table->id();
            $table->string('first_name')->default(null);
            $table->string('last_name')->default(null);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('persons');
    }
}
```

Однією з сильних сторін Eloquent є підтримка взаємозв'язків між моделями, таких як "один до одного", "один до багатьох", "багато до багатьох" та “має багато наскрізних”. У контексті управління родоводом важливо коректно налаштувати ці взаємозв'язки для моделювання переліку персон, що належать конкретному дереву. Приклад реалізації взаємозв’язку “має багато наскрізних” для моделі Tree зображено у лістингу 3.7.

Лістинг 3.7 – Приклад реалізації взаємозв’язку “має багато наскрізних”

```
public function people()
    {
        return $this->hasManyThrough(Person::class,
AvailableInTree::class, 'tree_id', 'id', 'id', 'person_id');
    }
```

Таким чином, налаштування моделей та взаємозв'язків у Laravel дозволяє створювати ефективну і гнучку структуру даних, яка легко масштабується і підтримується. Правильне моделювання даних є основою для стабільної і надійної роботи серверної частини web-ресурсу управління родоводом.

Контролери у Laravel створюються за допомогою команди аналогічної до команд створення моделі та міграцій. Наприклад, для створення контролера UserController, який буде обробляти запити, пов'язані з авторизацією та реєстрацією користувача, а також із отриманням та редагуванням даних користувача, використовується команда описана у лістингу 3.8.

Лістинг 3.8 – Приклад команди для створення контролера

```
php artisan make:controller UserController
```

Контролери реалізують методи для виконання CRUD операцій (створення, читання, оновлення, видалення) над даними таблиці бази даних. Метод контролера відповідає за виконання конкретної операції і взаємодіє з моделями для обробки даних. У лістингу 3.9 зображено фрагмент реалізованого контролера UserController.

Лістинг 3.9 – Приклад коду контролера

```
class UserController extends Controller{
    public function createUser(Request $request){
        try {
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password)
            ]);
            return response()->json([
```

```
                    'status' => true,
                    'status_code' => 2,
                    'message' => 'User Created Successfully',
                    'data' => [
                        'token' => $user->createToken("API
TOKEN")->plainTextToken,
                        'user' => [
                            'id' => $user->id,
                            'name' => $user->name,
                        ]
                    ],
                ], 200);
            } catch (\Throwable $th) {
                return response()->json([
                    'status' => false,
                    'message' => $th->getMessage(),
                ], 500);
            }
        }
    }
```

Маршрутизація визначає, які контролери і методи будуть обробляти певні HTTP-запити. У Laravel маршрути задаються у файлах маршрутизації, де вказуються URL і відповідні методи контролерів.

У Laravel маршрути визначаються у файлі routes/web.php для веб-інтерфейсу та routes/api.php для API-запитів. Для даного проекту використовується API-маршрутизація. Приклад реалізації маршрутів зображено у лістингу 3.10.

Лістинг 3.10 – Приклад реалізації маршрутів

```
    Route::post('/auth/register', [UserController::class,
'createUser']);
    Route::post('/auth/login', [UserController::class,
'loginUser']);
```

Основні маршрути включають шляхи для виконання CRUD операцій над даними родоводу:

– GET /person - отримання списку всіх осіб.
– POST /person - створення нової особи.
– GET /person/{id} - отримання інформації про конкретну особу.
– PUT /person/{id} - оновлення інформації про конкретну особу.
– DELETE /person/{id} - видалення конкретної особи.

Аутентифікація та авторизація є критично важливими аспектами для забезпечення безпеки веб-додатку. Laravel надає зручні інструменти для реалізації аутентифікації користувачів, включаючи механізми реєстрації, входу та виходу. Для цього у проєкті було використано готовий пакет Laravel Sanctum [18].

Приклад реалізації запитів, що потребують авторизації користувача зображено у лістингу 3.11.

Лістинг 3.11 – Приклад реалізації маршрутів, що вимагають авторизації

```
    Route::post('/auth/register', [UserController::class,
'createUser']);
    Route::post('/auth/login', [UserController::class,
'loginUser']);
```

3.3 Створення клієнтської частини web-ресурсу управління родоводом

Клієнтська частина є критично важливою складовою проекту, оскільки саме вона забезпечує зручний і інтуїтивно зрозумілий інтерфейс для користувачів. Вона дозволяє користувачам легко взаємодіяти з системою, переглядати родове дерево, додавати нових членів родини та редагувати інформацію про них. Створення ефективної та привабливої клієнтської частини є запорукою успіху всього проекту, оскільки саме вона формує перше враження у користувачів та забезпечує їхній комфорт при роботі з системою.

Проект на SvelteKit організований у такий спосіб, щоб забезпечити максимально зручну розробку та підтримку коду.

src/ - головний каталог з вихідним кодом проекту. Містить підкаталоги для компонентів, сторінок, стилів та інших ресурсів.

Основні підкаталоги проекту включають:

– components/ - містить компоненти Svelte, які використовуються для побудови інтерфейсу користувача. Кожен компонент представляє окремий елемент інтерфейсу (наприклад, кнопка, форма, таблиця).

– routes/ - містить файли, що відповідають за маршрутизацію. Кожен файл у цьому каталозі представляє окрему сторінку або групу сторінок додатку.

– styles/ - містить файли стилів, написані на SCSS. Включає глобальні стилі та специфічні стилі для окремих компонентів.

– assets/ - містить статичні ресурси, такі як зображення, шрифти та інші медіафайли.

SvelteKit використовує файлову систему для автоматичного створення маршрутів на основі файлів у каталозі routes/. Кожен файл або директорія у цьому каталозі відповідає окремому маршруту. Наприклад, файл src/routes/index.svelte

відповідає головній сторінці додатку, а файл src/routes/trees.svelte - сторінці "Дерева" [19].

SvelteKit у своєму розпорядженні має файли клієнтської частини та внутрішньої серверної частини. Для спрощення реалізації певних запитів із клієнтської частини до внутрішньої серверної частини SvelteKit було розроблено допоміжну функцію описану у лістингу 3.12.

Лістинг 3.12 – Допоміжна функція реалізації запитів між клієнтською та серверною частинами додатка

```
export function backQueryFront(data, token = null){
    if(token){
        if(!data.headers){
            data.headers = {
                'Authorization' : 'Bearer ' + token,
            }
        }
        else{
            data.headers['Authorization'] = 'Bearer ' + token;
        }
    }
    return axios(data);
}
```

Аналогічну функцію було створено для спрощення реалізації запитів між внутрішньою серверною частиною та зовнішньою серверною частину та зображено у лістингу 3.13.

Лістинг 3.13 – Допоміжна функція реалізації запитів між серверними частинами додатка

```
export function apiQueryBack(data, cookies){
    if(data.url??false){
        data.url = domain + data.url;
    } else {
        data.url = domain;
    }
    let token = cookies.get('token')??false;
    let lang = cookies.get('lang')??'en';
    if(token){
        if(!data.headers){
            data.headers = {
                'Authorization' : 'Bearer ' + token,
            }
        } else {
            data.headers['Authorization'] = 'Bearer ' + token;
        }
    } else {
        data.headers = {};
    }
    data.headers['Accept'] = 'application/json';
    data.headers['Accept-Language'] = lang;
    return axios(data).catch((e) => {
        if(e.response.status === -45){
            throw redirect(302, '/login');
        }
    });
}
```

Для вибору користувачем дерева, що потрібно відображати із наявних у користувача, було розроблено сторінку переліку дерев, що містить посилання на

сторінки конкретних дерев та функціонал створення нового дерева. Ця сторінка зображена на рисунку 3.1.

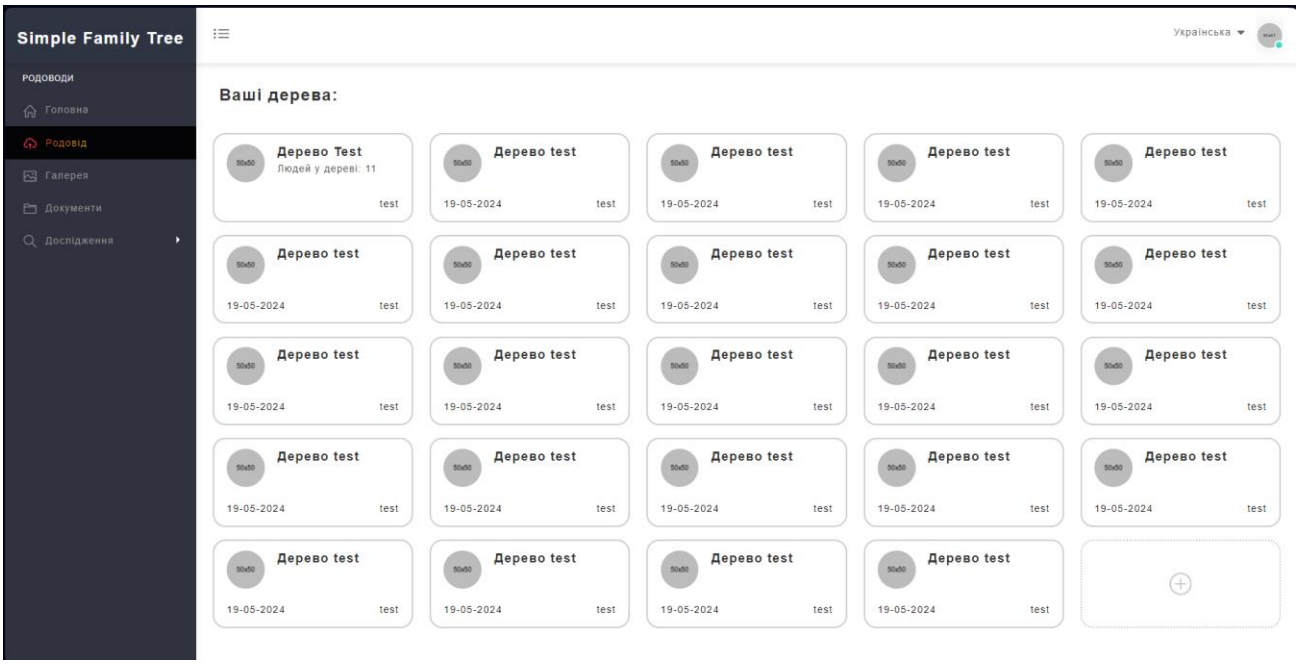

Рисунок 3.1 – Сторінка переліку дерев

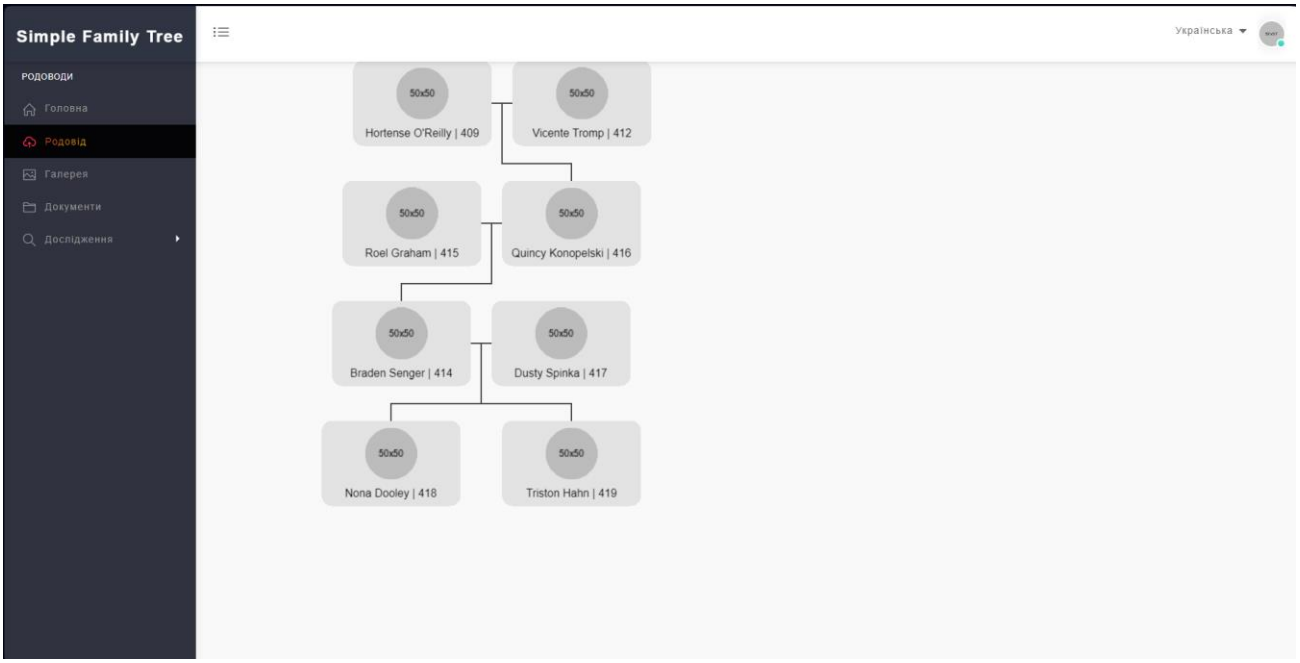

Рисунок 3.2 – Сторінка конкретного дерева

Рендер дерева було реалізовано із використанням бібліотеки PIXI.js, що полегшує роботу із HTML-елементом canvas [20]. Для спрощеного користування цією бібліотекою було створено перелік методів, наприклад для створення картки персони.

Приклад сторінки конкретного дерева зображено на рисунку 3.2.

3.4 Висновок

Отже за результатами роботи над даним розділом було розглянуто та обрано мови програмування та фреймворки для реалізації проєкту із обґрунтуванням цього вибору. Також було розглянуто процес реалізації серверної частини засобами Laravel та описано основні характеристики такого проєкту. Аналогічно було описано основні характеристики проєкту написано на основі SvelteKit, тобто клієнтської частини даного проєкту, та зображено деякі сторінки цієї частини.

ВИСНОВКИ

Під час виконання бакалаврської роботи було розроблено веб-ресурс управління родоводом. Усі завдання, поставлені перед бакалаврською роботою, виконані в повному обсязі: обґрунтовано доцільність створення веб-ресурсу, проаналізовано відомі рішення у створенні подібних веб-ресурсів, розроблено ER-діаграму функціонування та основні алгоритми роботи веб-ресурсу, обґрунтовано вибір мови програмування та засобів розробки, створено серверну та клієнтську частини веб-ресурсу.

У першому розділі охарактеризовано предмет проектування, визначено актуальність та доцільність реалізації веб-ресурсу управління родоводом. Проаналізовано сервіси-аналоги до реалізованої системи. Під час аналізу об’єкта проектування було сформульовано задачі розробки ресурсу, визначено основні завдання, напрямки розвитку та необхідний функціонал програми. Врахувавши всю інформацію, було вирішено, що доцільно створити власний веб-ресурс управління родоводом.

У другому розділі було обґрунтовано засоби розробки системи управління родоводом. Крім цього було розроблено та детально описано ER-діаграму, описано модулі з яких складається ER-діаграма та взаємодії у головному модулі. Також у даному розділі реалізовано схеми основних алгоритму веб-ресурсу.

У третьому розділі було обґрунтовано вибір мови програмування. Реалізовано клієнтську та серверну частини веб-ресурсу управління родоводом.

Мету роботи, яка є покращенням доступності та гнучкості WEB-ресурсу управління родоводом шляхом розробки гнучкої та продуманої наперед структури бази даних, а також реалізації простого клієнтського інтерфейсу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Генеологія Wikipedia [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Генеалогія
2. MyHeritage Wikipedia [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/MyHeritage
3. FamilySearch [Електронний ресурс] – Режим доступу: https://www.familysearch.org/uk/about/
4. MyHeritage [Електронний ресурс] – Режим доступу: https://www.myheritage.com.ua
5. MyHeritage DNA Review [Електронний ресурс] – Режим доступу: https://www.forbes.com/health/medical-supplies/myheritage-dna-review/
6. JetBrains PhpStorm [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/phpstorm/
7. OSPanel [Електронний ресурс] – Режим доступу: https://ospanel.io
8. About Postman [Електронний ресурс] – Режим доступу: https://www.postman.com/company/about-postman/
9. Visual Studio Code [Електронний ресурс] – Режим доступу: https://code.visualstudio.com
10. Sublime Text [Електронний ресурс] – Режим доступу: https://www.sublimetext.com
11. Dsnews.ua. JavaScript: як і чому варто почати [Електронний ресурс] – Режим доступу: https://www.dsnews.ua/ukr/future/javascript-kak-i-pochemu-stoit-nachat-23052023-480051
12. SvelteKit web development, streamlined [Електронний ресурс] – Режим доступу: https://kit.svelte.dev
13. CSS with superpowers [Електронний ресурс] – Режим доступу: https://sass-lang.com

14. React [Електронний ресурс] – Режим доступу: https://uk.legacy.reactjs.org
15. Angular [Електронний ресурс] – Режим доступу: https://angular.dev
16. Vue.js [Електронний ресурс] – Режим доступу: https://vuejs.org
17. Laravel. The PHP Framework for Web Artisans [Електронний ресурс] – Режим доступу: https://laravel.com
18. Laravel Sanctum [Електронний ресурс] – Режим доступу: https://laravel.com/docs/11.x/sanctum
19. SvelteKit. Routing [Електронний ресурс] – Режим доступу: https://kit.svelte.dev/docs/routing
20. PIXIjs. The HTML5 Creation Engine [Електронний ресурс] – Режим доступу: https://pixijs.com

ДОДАТКИ

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка WEB-ресурсу управління родоводом

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,35% Схожість 1,65%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Бойко А.В.

Керівник роботи Денисюк В.О.

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА

Для того, щоб розгорнути серверну частину web-сервісу на глобальному або локальному сервері необхідно виконати таку послідовність дій:

1) Скопіювати файли проекту в директорію.
2) Виконати команду composer install.
3) Виконати команду npm i.
4) Скопіювати файл .env.example та змінити його назву на .env
5) Запустити команду php artisan key:generate
6) Запустити команду php artisan storage:link
7) Створити базу даних
8) Заповнити в файлі .env параметри APP_URL, DB_DATABASE, DB_USERNAME та DB_PASSWORD
9) Виконати команду php artisan migrate

Також у випадку розгортання на локальному сервері необхідно виконати команду php artisan serve --port=8090

Тепер до серверної частини можна робити запити за посиланням http://127.0.0.1:8090/api

Далі необхідно розгорнути клієнтську частину web-сервісу, для цього необхідно виконати аналогічний шлях:

1) Скопіювати файли проекту в директорію.
2) Виконати команду npm i.
3) Скопіювати файл .env.example та змінити його назву на .env
4) Заповнити в файлі .env параметр API_LINK та встановити його значення у http://127.0.0.1:8090/api

У випадку розгортання на глобальному сервері необхідно виконати команду npm run build, а у випадку розгортання на локальному сервері команду npm run dev.

Тепер web-сервіс має бути доступний за домен глобального сервера або за посиланням http://localhost:5173/

При переході за цим посиланням нас перенаправляє на сторінку авторизації. Для того, щоб зареєструватись у сервісі необхідно натиснути на кнопку “Реєстрація” знизу форми, що зображено на рисунку 1.

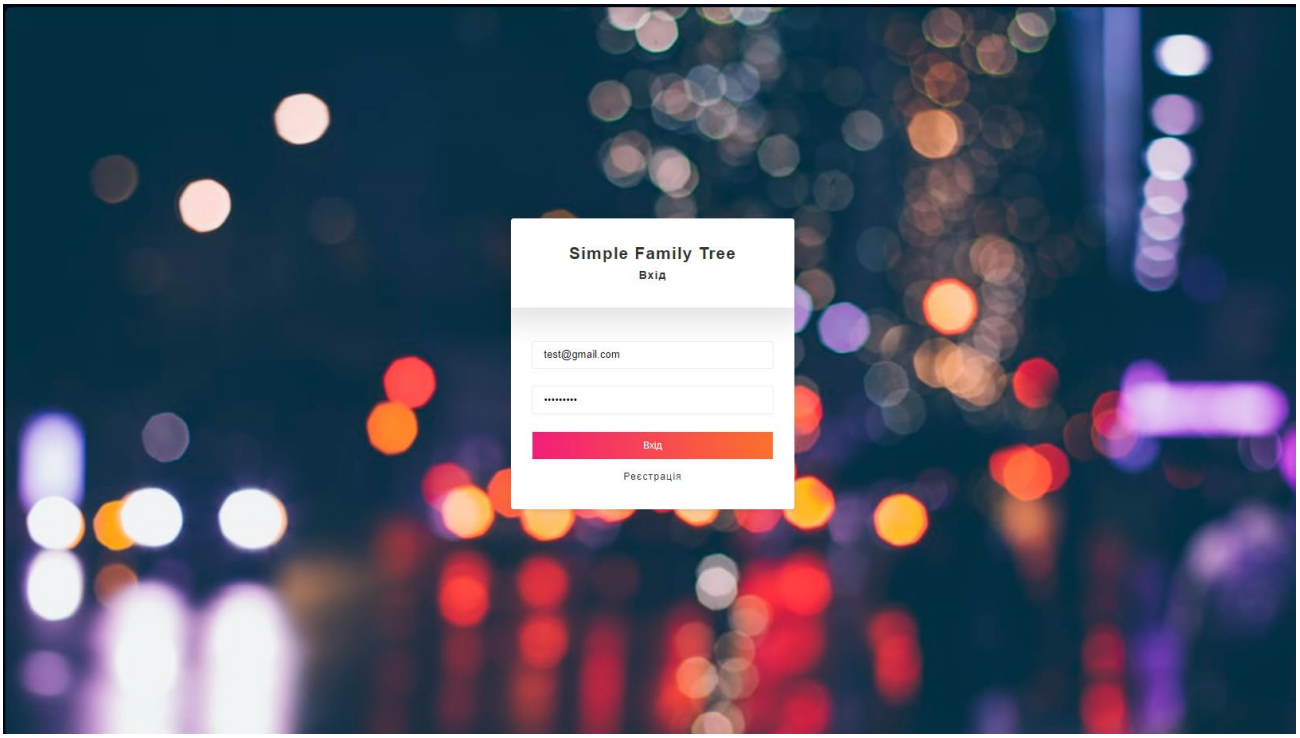

Рисунок 1 – Сторінка авторизації

На сторінці реєстрації необхідно ввести ім’я, пошту та пароль користувача і натиснути кнопку “Реєстрація”. Сторінку реєстрації зображено на рисунку 2. Якщо реєстрація пройшла успішно, нас перекине на головну сторінку, інакше виникне "поп-ап" із повідомленням, шо сталось не так.

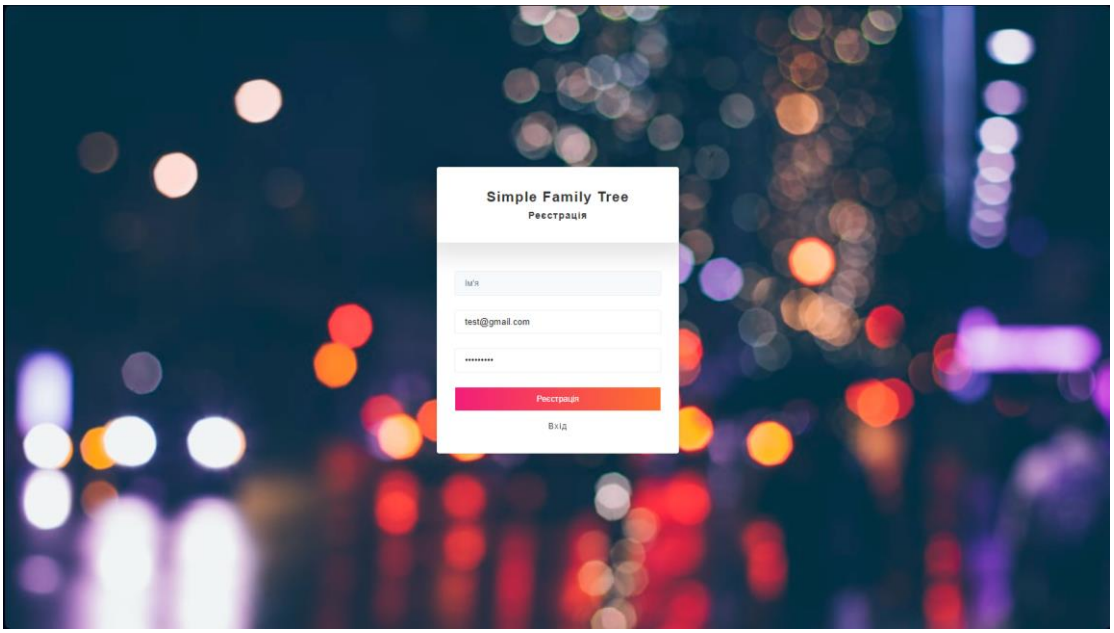

Рисунок 2 – Сторінка реєстрації

З головної сторінки, користуючись навігаційною панеллю зліва можна перейти на сторінку дерев користувача, що зображена на рисунку 3. Кожна сторінка містить вищезгадану навігаційну панель, а також перемикач мови та кнопку виходу.

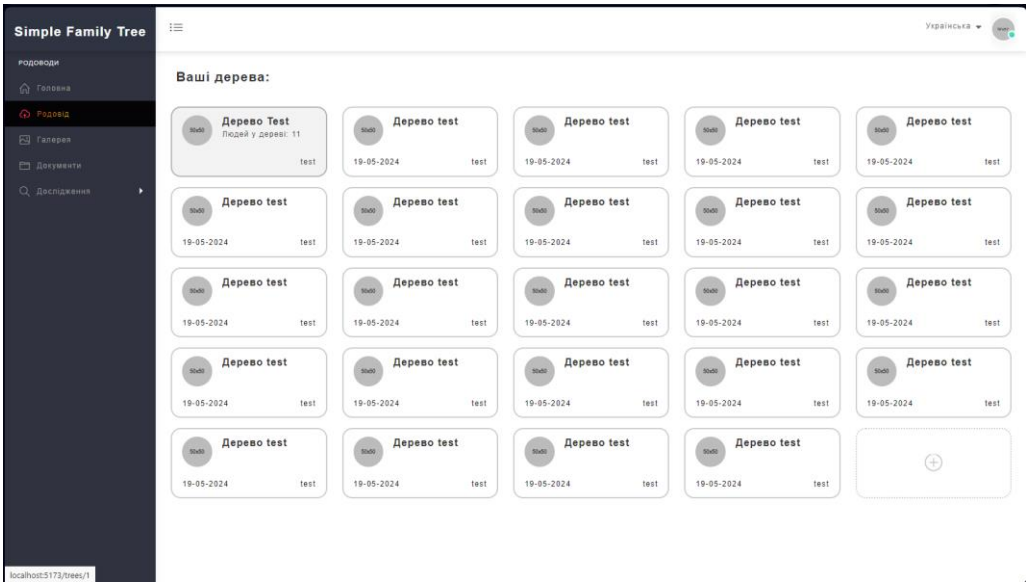

Рисунок 3 – Сторінка переліку дерев

На даній сторінці можна переглянути наявні дерева користувача, створити нове дерево та перейти на сторінку існуючого дерева, що зображена на рисунку 4.

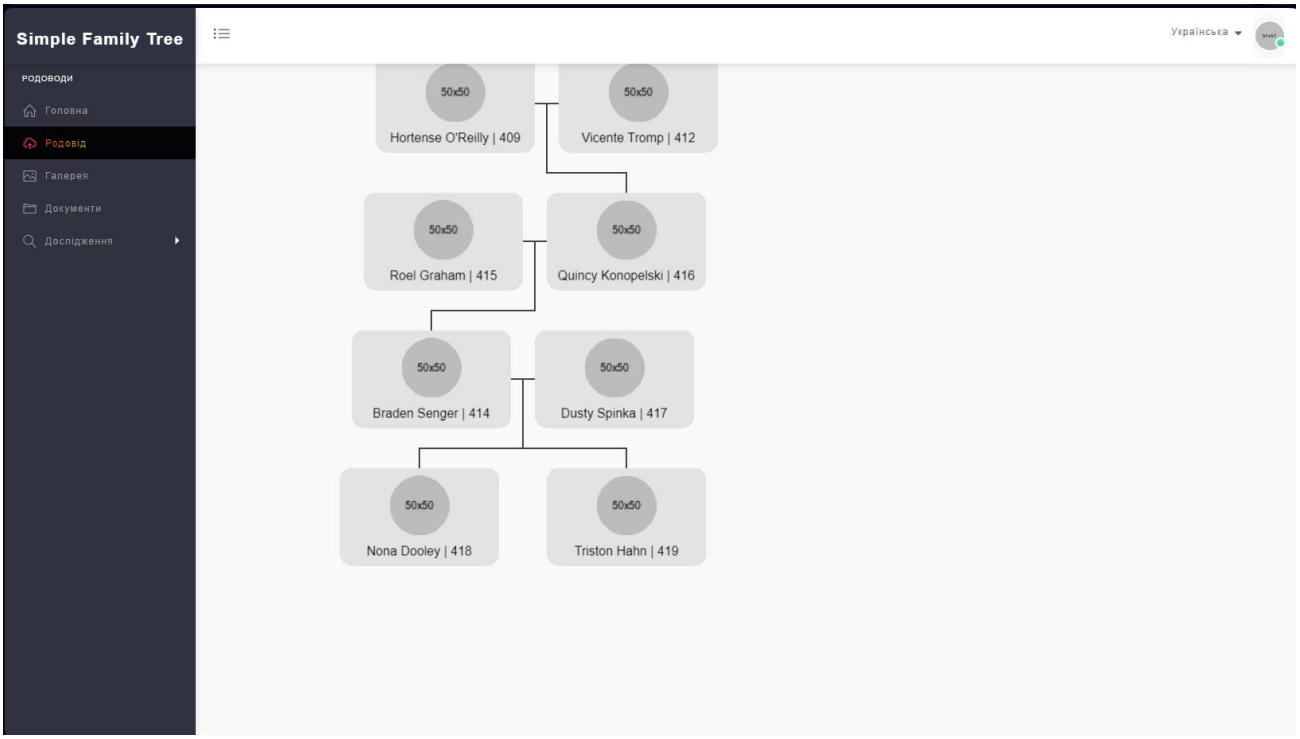

Рисунок 4 – Сторінка дерева

На даній сторінці зображено відрендерене дерево. При кліку на картку персони відкривається модальне вікно із інформацією про персону, яку можна редагувати. Модальне вікно складається із 4 вкладок, зображених на рисунку 5. Дане модальне вікно містить посилання для відкриття цього модального вікна у форматі повноцінної сторінки, зображеної на рисунку 6. На першій вкладці розташована основна інформація про персону.

На наступній вкладці, зображеній на рисунку 7, міститься перелік найближчих родичів персони. Також на цій сторінці можна створити нових персон пов’язаних із поточною персоною.

На третій вкладці, зображеній на рисунку 8, міститься галерея із

фотографіями прив’язаними до поточної персони. На цій вкладці можна додавати та видаляти фотографії пов’язані із цією персоною.

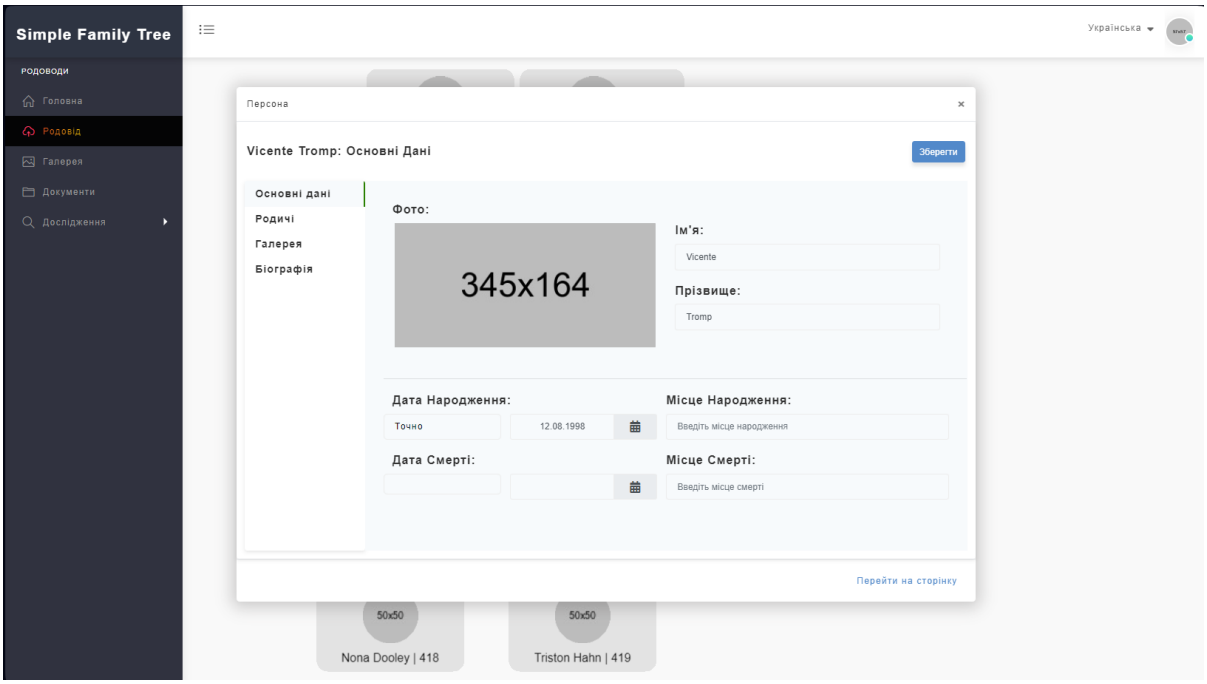

Рисунок 5 – Модальне вікно персони

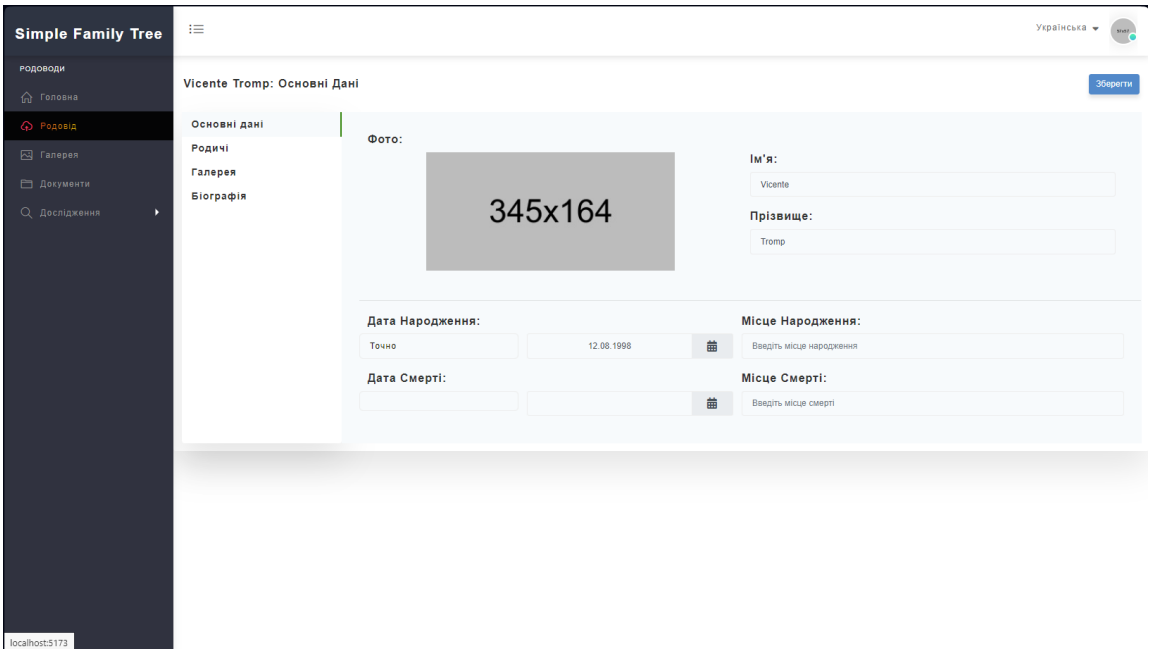

Рисунок 6 – Сторінка редагування персони

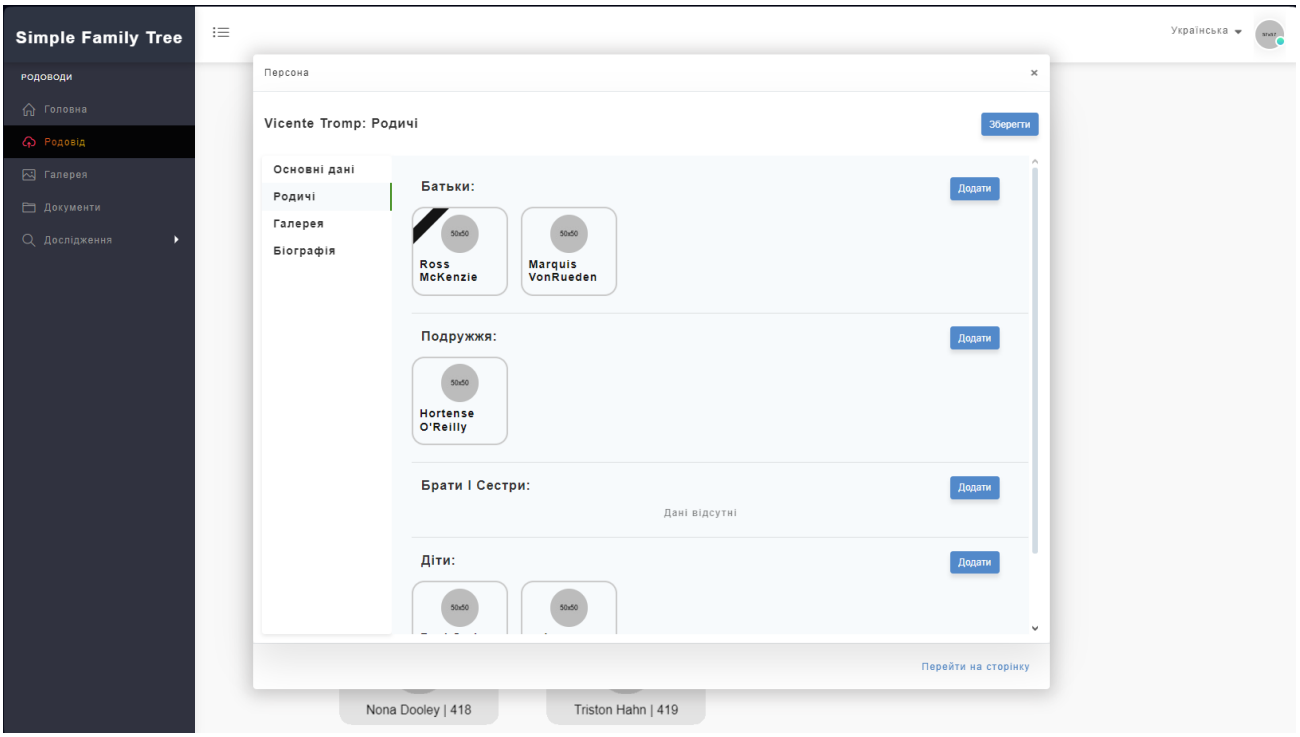

Рисунок 7 – Вкладка родичів модального вікна

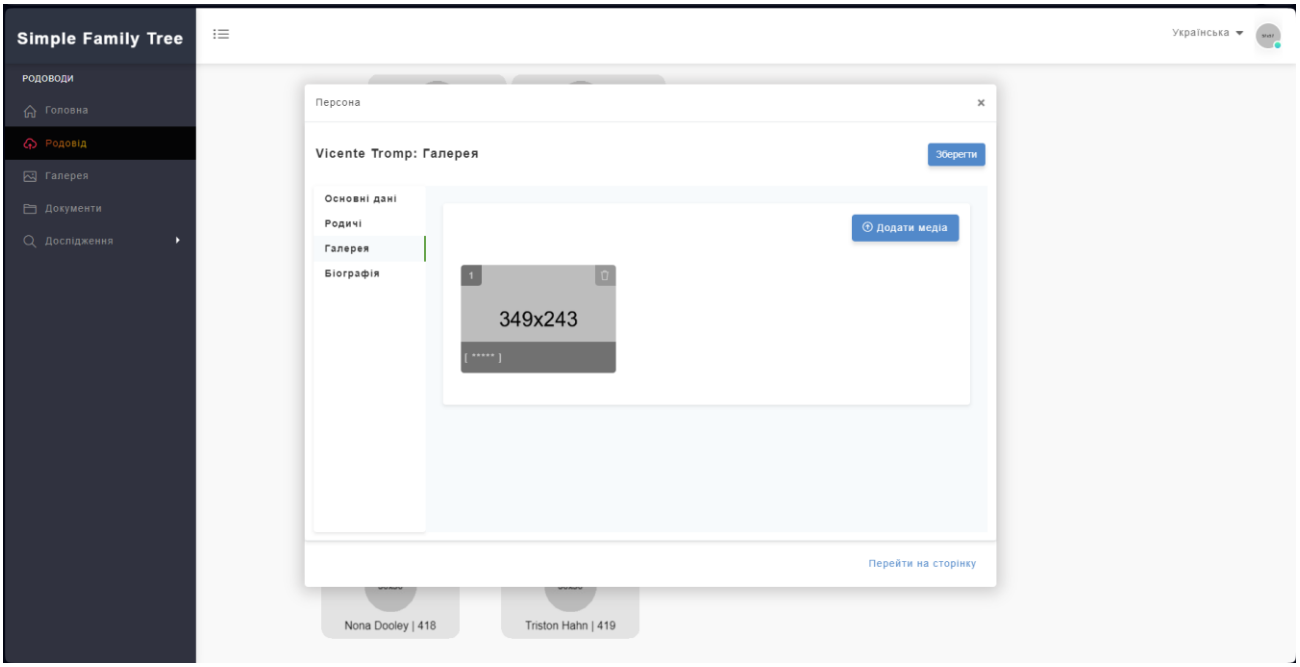

Рисунок 8 – Вкладка галереї модального вікна

На останній вкладці модального вікна, зображеній на рисунку 9, міститься

глобальний редактор тексту для заповнення біографії персони.

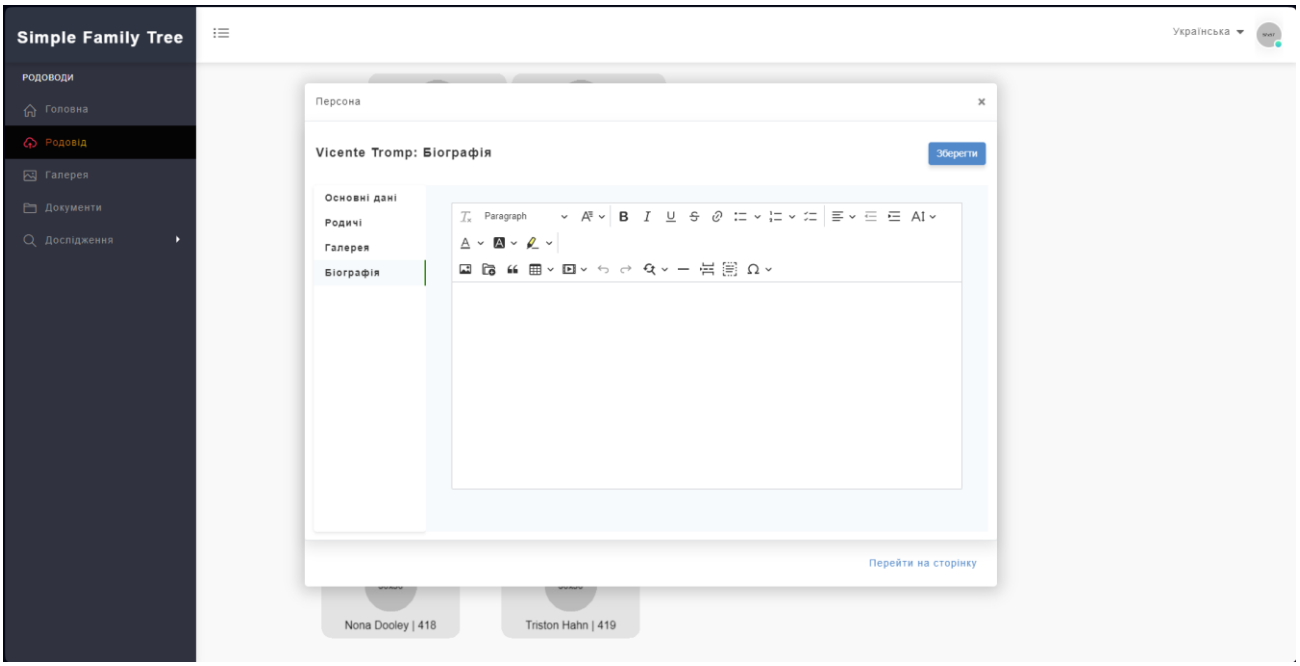

Рисунок 9 – Вкладка біографії модального вікна

ДОДАТОК В
ЛІСТИНГ СЕРВЕРНОЇ ЧАСТИНИ. ЛІСТИНГ КЛІЄНТСЬКОЇ ЧАСТИНИ

```
<?php

namespace App\Models;

use DateTimeInterface;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Person extends Model
{
    use HasFactory;

    protected $table = 'persons';

    protected $fillable = [
        'first_name',
        'last_name',
    ];

    protected function serializeDate(DateTimeInterface $date)
    {
        return $date->format('d-m-Y');
    }

    public function relations()
    {
        $parent_child_type = RelationType::where('name', 'parent-child')->first();

        $parentsRaw = Relation::where([['type_id', $parent_child_type->id], ['last_person_id',
$this->id]])->get();
        $parents = [];
        foreach ($parentsRaw as $parent) {
            $parents[] = [
                'type' => 'parent',
                'id' => $parent->first_person_id
            ];
        }

        $childrenRaw      =      Relation::where([['type_id',      $parent_child_type->id],
['first_person_id', $this->id]])->get();
        $children = [];
        foreach ($childrenRaw as $child) {
            $children[] = [
```

```
                'type' => 'child',
                'id' => $child->last_person_id
            ];
        }

        $sibling_type = RelationType::where('name', 'sibling')->first();
        $siblingsRaw = Relation::where([['type_id', $sibling_type->id], ['first_person_id',
$this->id]])->get();
        $siblings = [];
        foreach ($siblingsRaw as $sibling) {
            $siblings[] = [
                'type' => 'sibling',
                'id' => $sibling->last_person_id
            ];
        }

        $spouse_type = RelationType::where('name', 'spouse')->first();
        $spousesRaw = Relation::where([['type_id', $spouse_type->id], ['first_person_id',
$this->id]])->get();
        $spouses = [];
        foreach ($spousesRaw as $spouse) {
            $spouses[] = [
                'type' => 'spouse',
                'id' => $spouse->last_person_id
            ];
        }

        $response = array_merge($parents, $children, $siblings, $spouses);
        return $response;
    }

}
```

```
<?php

namespace App\Servises;

class Generations
{
    public function calcGenerations($persons)
    {
        $persons_without_parents = [];
        Generations::getPersonsWithoutParentsIds($persons, $persons_without_parents);

        Generations::calcMaxOffsprings($persons_without_parents, $persons);

        $max_offspring_count = max(array_column($persons_without_parents, 'offspring_count'));
        $older_persons_infos = array_filter($persons_without_parents, function($elem) use
($max_offspring_count) {
```

```
            return $elem['offspring_count'] == $max_offspring_count;
        });

        $older_persons = [];
        foreach ($older_persons_infos as $info) {
            $new_person = Generations::getPersonById($persons, $info["person_id"]);
            $new_person["generation"] = 1;

            $older_persons[] = $new_person;
        }

        foreach ($older_persons as $person) {
            Generations::setRelationsGenerations($persons, $person);
        }

        return [
            "max_generation" => $max_offspring_count,
            "persons" => $persons
        ];
    }

    protected function getPersonsWithoutParentsIds($persons, &$output = []): array
    {
        foreach($persons as $id => $person) {
            $hasParent = false;

            foreach ($person->relations as $relation) {
                if($relation["type"] == "parent") {
                    $hasParent = true;
                    break;
                }
            }

            if(!$hasParent) {
                $output[] = [
                    "array_id" => $id,
                    "person_id" => $person->id,
                    "offspring_count" => null
                ];
            }
        }

        return $output;
    }

    protected function calcMaxOffsprings(&$persons_infos, $persons)
    {
        foreach ($persons_infos as $id => $elem) {
            $offsprings = [];
```

```
            $offsprings[] = [$elem["person_id"]];

            Generations::hasChildren($offsprings, $elem["person_id"], $persons);
            $persons_infos[$id]["offspring_count"] = count($offsprings);
        }
    }

    protected function hasChildren(&$offsprings, $person_id, $persons)
    {
        $person = Generations::getPersonById($persons, $person_id);

        $currentOffspring = [];

        foreach ($person["relations"] as $relation) {
            if($relation["type"] == "child") {
                $currentOffspring[] = $relation["id"];
            }
        }

        if(count($currentOffspring) > 0) {
            $offsprings[] = $currentOffspring;
            foreach ($currentOffspring as $person_id) {
                Generations::hasChildren($offsprings, $person_id, $persons);
            }
        }
    }

    protected function getPersonById($persons, $id)
    {
        $resultArray = array_filter($persons->toArray(), function($person) use ($id) {
            return $person["id"] == $id;
        });

        return reset($resultArray);
    }

    protected function hasGeneration($person): bool
    {
        return $person["generation"] != null;
    }

    protected function setRelationsGenerations(&$old_persons, $person)
    {
        foreach ($person["relations"] as $relation) {
            $key = array_search($relation["id"], array_column($old_persons->toArray(), "id"));

            if(!Generations::hasGeneration($old_persons[$key])) {
                if ($relation["type"] == "sibling" || $relation["type"] == "spouse") {
                    $old_persons[$key]["generation"] = $person["generation"];
```

```
                }
                if ($relation["type"] == "parent") {
                    $old_persons[$key]["generation"] = $person["generation"] - 1;
                }
                if ($relation["type"] == "child") {
                    $old_persons[$key]["generation"] = $person["generation"] + 1;
                }

                if(count($old_persons[$key]["relations"]) > 0) {
                    Generations::setRelationsGenerations($old_persons, $old_persons[$key]);
                }
            }
        }
    }
}
```

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Relation extends Model
{
    use HasFactory;

    protected $table = 'relations';
    protected $fillable = [
        'type_id',
        'first_person_id',
        'last_person_id',
    ];

    public function sub_person_id($main_person)
    {
        switch ($main_person) {
            case $this->first_person_id:
                $response = $this->last_person_id;
                break;
            case $this->last_person_id:
                $response = $this->first_person_id;
                break;
            default:
                $response = null;
        }

        return $response;
    }
```

```
}<?php

use App\Http\Controllers\Api\TreeController;
use App\Http\Controllers\Api\UserController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::post('/auth/register', [UserController::class, 'createUser']);
Route::post('/auth/login', [UserController::class, 'loginUser']);

Route::middleware('auth:sanctum')->post('/trees/create',              [TreeController::class,
'createTree']);
Route::middleware('auth:sanctum')->get('/trees/get',                  [TreeController::class,
'getTreesByUser']);
Route::middleware('auth:sanctum')->get('/trees/detail', [TreeController::class, 'getTree']);

//Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
//    return $request->user();
//});


<script>
    import Tab from "$lib/components/structure/Tab.svelte";
    import MainTab from "./MainTab.svelte";
    import GalleryTab from "./GalleryTab.svelte";
    import BiographyTab from "./BiographyTab.svelte";
    import RelationsTab from "./RelationsTab.svelte";
    import {page} from "$app/stores";
    import {goto} from "$app/navigation";
    import {onMount} from "svelte";
    import {t} from "$lib/scripts/translations.js";
    import {defaultData} from "../values.js"

    export let data = defaultData();

    let tabsList = [
        {
            key: "main-tab",
```

```
        title: $t("static.titles.main-tab"),
        component: MainTab
    },
    {
        key: "relations-tab",
        title: $t("static.titles.relations-tab"),
        component: RelationsTab
    },
    {
        key: "gallery-tab",
        title: $t("static.titles.gallery-tab"),
        component: GalleryTab
    },
    {
        key: "biography-tab",
        title: $t("static.titles.biography-tab"),
        component: BiographyTab
    },
];

let tabTitle = 'Основна інформація';
let activeTab = tabsList[0].key;

function setTabTitle(tabName) {
    tabTitle = "";

    if(data?.person?.data?.name || data?.person?.data?.surname) {
        tabTitle += data?.person?.data?.name??"";
        tabTitle += data?.person?.data?.surname?" "+data?.person?.data?.surname:"";
        tabTitle += ": ";
    }
    tabTitle += tabName;
}

function openTab(tab) {
    $page.url.searchParams.set('tab', tab.key);

    setTabTitle(tab.title);
    if($page.url.pathname.includes("persons")) {
        goto('?' + $page.url.searchParams.toString(), {replaceState: true});
    } else {
        activeTab = tab.key;
    }
}

function saveNewData() {

}
```

```
    onMount(()=>{
        console.log($page);
        if($page.url.searchParams.get('tab') && $page.url.pathname.includes("persons")){
            activeTab = $page.url.searchParams.get('tab');
        } else {
            setTabTitle(tabsList[0].title);
        }
    });

</script>


<Tab
    {tabTitle}
    tabHeaderClasses="bg-dark-gray"
    tabNavClasses="tab-navs-container col-md-2 col-sm-3 col-xxl-3 mb-lg-0 mb-sm-3"
    tabContentClasses="tab-content-container col-md-10 col-sm-9 col-xxl-9 mb-sm-9"
>
    <svelte:fragment slot="header-buttons">
        <button              class="btn              btn-primary              raised-rounded"
on:click={saveNewData}>{$t('static.buttons.save')}</button>
    </svelte:fragment>

    <svelte:fragment slot="nav-tabs">
        {#each tabsList as tab}
            <li class="nav-item">
                <a
                    href="#{tab.key}"
                    class="nav-link global-tab text-left"
                    class:active={tab.key == activeTab}
                    on:click={()=>{openTab(tab)}} data-toggle="tab"
                >
                    {tab.title}
                </a>
            </li>
        {/each}
    </svelte:fragment>

    <svelte:fragment slot="tab-content">
        {#each tabsList as tab}
            <div
                class="tab-pane global-tab fade show"
                class:active={tab.key == activeTab}
                id="{tab.key}"
                role="tabpanel">

                <svelte:component
                        this="{tab.component}"
                        bind:data="{data}">
```

```
                </svelte:component>
            </div>
        {/each}
    </svelte:fragment>
</Tab>

<script>
    import Input from "$lib/components/form/Input.svelte";
    import {t} from "$lib/scripts/translations.js";
    import ImageInput from "$lib/components/form/ImageInput.svelte";
    import CustomSelector from "$lib/components/form/CustomSelector.svelte";
    import DatePicker from "$lib/components/form/DatePicker.svelte";
    import {onMount} from "svelte";
    import {browser} from "$app/environment";

    export let data = {};

    function getNewMainPhoto(e) {
        console.log(e.detail.event.target.files);
        let files = Array.from(e.detail.event.target.files);
        console.log(files);
        console.log(files[0]);
        data.person.data.profile_photo_id = files[0];
    }

    onMount(()=>{
        if(browser) {
            data["date_types"] = data["date_types"].map(item => {
                item.name = $t(`static.date-types.${item.value}`);
               return item;
            });
        }
    });
</script>

<div class="main-tab-container container">
    <div class="row">
        <div class="col-12">
            <div class="row">
                <div class="col-6">
                    <ImageInput
                            label="{$t('static.inputs.label.photo')}:"
                            placeholder="{$t('static.inputs.placeholder.photo')}"
                            dataImage="{data.person.data.profile_photo_id}"
                            on:imageInput={getNewMainPhoto}
                    />
                </div>
                <div class="col-6 pt-4">
                    <Input
```

```
                            type="text"
                            classes="col-12 pt-2"
                            label="{$t('static.inputs.label.name')}:"
                            placeholder="{$t('static.inputs.placeholder.name')}"
                            bind:value={data.person.data.name}
                            borderColor=""
                        />
                        <Input
                            type="text"
                            classes="col-12"
                            label="{$t('static.inputs.label.surname')}:"
                            placeholder="{$t('static.inputs.placeholder.surname')}"
                            bind:value={data.person.data.surname}
                            borderColor=""
                        />
                    </div>
                </div>
            </div>
            <hr class="col-12">

            <div class="col-6">
                <div class="row">
                    <div class="col-12">
                        <p class="card-title mb-1">{$t("static.inputs.label.birth_date")}:</p>
                    </div>
                    <CustomSelector
                            label=""
                            searchPlaceholder="{$t('static.inputs.placeholder.select-date-
type')}"
                            value="{data['date_types'].find(type      =>      type.value      ==
data.person.attributes['birth_date_type'])}"
                            items="{data['date_types']}"
                            options="{{}}"
                            wrapperClasses="form-group col-5"
                            classes="col-12 pr-0 pl-0"
                    />
                    <DatePicker
                            label=""
                            wrapperClasses="m-0 col-7"
                            inputClass="col-12 mb-2 p-0"
                            id="birth_date"
                            value="{data.person.attributes['birth_date']}"
                    />
                </div>
            </div>
            <Input
                    type="text"
                    classes="col-6"
                    label="{$t('static.inputs.label.birth_place')}:"
```

```
                placeholder="{$t('static.inputs.placeholder.birth_place')}"
                bind:value={data.person.attributes.birth_place}
                borderColor=""
        />

        <div class="col-6">
            <div class="row">
                <div class="col-12">
                    <p class="card-title mb-1">{$t("static.inputs.label.death_date")}:</p>
                </div>
                <CustomSelector
                        label=""
                        searchPlaceholder="{$t('static.inputs.placeholder.select-date-
type')}"
                        value="{data['date_types'].find(type      =>      type.value      ==
data.person.attributes['death_date_type'])}"
                        items="{data['date_types']}"
                        options="{{}}"
                        wrapperClasses="form-group col-5"
                        classes="col-12 pr-0 pl-0"
                />
                <DatePicker
                        label=""
                        wrapperClasses="m-0 col-7"
                        inputClass="col-12 mb-2 p-0"
                        id="birth_date"
                        value="{data.person.attributes['death_date']}"
                />
            </div>
        </div>
        <Input
                type="text"
                classes="col-6"
                label="{$t('static.inputs.label.death_place')}:"
                placeholder="{$t('static.inputs.placeholder.death_place')}"
                bind:value={data.person.attributes.death_place}
                borderColor=""
        />
    </div>
</div>

<script>
    import * as PIXI from 'pixi.js';
    import {onMount} from "svelte";
    import {Canvas} from "$lib/scripts/pixi/canvas.js";
    import    {selectedPersonId,    treeScale,    isCtrlPressed,    isShiftPressed}    from
"$lib/stores/store.js";
    import {onlyUnique} from "$lib/scripts/utils.js";
    import {drawLine} from "$lib/scripts/pixi/pixiUtils.js";
```

```
    export let data = {};

    let persons;
    let canvasElem = null;

    let app = null, mainContainer = null, canvas = null, card = null;

    $: persons = data.persons??[];
    let gapDist = {x: 30*$treeScale, y: 50*$treeScale};

    let containersArray = [];

    $: drawTree($treeScale);

    function drawTree() {
        if(canvasElem) {
            canvas = new Canvas(app, mainContainer, canvasElem, $treeScale);

            if (app) {
                let mainPersonId = 414;
                let mainPerson = persons.find(person => person.id == mainPersonId);
                let mainPersonGeneration = mainPerson.generation;

                makePersons();

                let offspringPersons = persons.filter(person => person.generation >=
mainPersonGeneration);
                offspringPersons.forEach((person) => {
                    makePersonAndSpouse(person);
                });
                containersArray.forEach(container => {
                    container.isPositioned = false;
                });
                makeLastOffspringGeneration(data.maxGen);
                for (let gen = data.maxGen-1; gen >= mainPersonGeneration; gen--) {
                    makeOffspringsGeneration(gen);
                }

                for (let gen = 1; gen <= mainPersonGeneration-1; gen++) {
                    makeAncestorsGeneration(gen);
                }
            }
        }
    }

    function makeFirstAncestorGeneration(generation) {

    }
```

```
    function makeAncestorsGeneration(generation) {
        let personsOfGen = persons.filter(person => person.generation == generation);

        if(personsOfGen.length > 1) {
            personsOfGen.forEach((person) => {
                if(generation == 1) {
                    makePersonAndSpouse(person);
                }

                if(person.relations.filter(rel => rel.type.includes('parent')).length > 0) {
                    positioningParentsAndChildrenForAncestors(person);
                }

                if(generation != 1) {
                    makePersonAndSpouseForAncestors(person);
                }
            });
        }
    }
    function makeLastOffspringGeneration(generation) {
        let personsOfGen = persons.filter(person => person.generation == generation);

        if(personsOfGen.length > 1) {
            personsOfGen.forEach((person) => {
                positioningSiblings(person);

                if(person.relations.filter(rel => rel.type.includes('child')).length > 0) {
                    positioningParentsAndChildren(person);
                }
            });
        }
    }
    function makeOffspringsGeneration(generation) {
        let personsOfGen = persons.filter(person => person.generation == generation);

        if(personsOfGen.length > 1) {
            personsOfGen.forEach((person) => {
                if(person.relations.filter(rel => rel.type.includes('child')).length > 0) {
                    positioningParentsAndChildren(person);
                }
            });
        }
    }

    function makePersons() {
        persons.forEach(person => {
            if(person.elem == null) {
                person.elem           =           canvas.createCard({x:250,           y:250},
'../../assets/images/avatar/1.jpg',
```

```
                    person['first_name'] + ' ' + person['last_name'] + ' | ' + person.id,
person.id);
            }
        });
    }

    function makePersonAndSpouse(person) {
        if(!person.elem) {
            // extra person making
        }
        if(!person.isPositioned) {
            person.elem.position.y = 0;
            person.elem.position.x = 0;
            person.isPositioned = true;

            let container = new PIXI.Container();
            container.name = 'spouses';
            container.addChild(person.elem);

            // many spouses

            let spouseRel = person.relations.find(rel => rel.type.includes('spouse'));
            if(spouseRel) {
                let spouse = persons.find(person => person.id == spouseRel.id);
                if (spouse && !spouse.isPositioned) {

                    let dots = [];
                    dots.push({
                        x: person.elem.getBounds().x + person.elem.getBounds().width,
                        y: person.elem.getBounds().y + person.elem.getBounds().height / 2,
                    });

                    spouse.elem.position.y = 0;
                    if (spouse.sex == 'man') {
                        spouse.elem.position.x   =   person.elem.position.x   -   gapDist.x   -
spouse.elem.getBounds().width;
                        dots.push({
                            x: spouse.elem.getBounds().x + spouse.elem.getBounds().width,
                            y: spouse.elem.getBounds().y + spouse.elem.getBounds().height / 2,
                        });
                    } else {
                        spouse.elem.position.x        =        person.elem.position.x        +
person.elem.getBounds().width + gapDist.x;
                        dots.push({
                            x: spouse.elem.getBounds().x,
                            y: spouse.elem.getBounds().y + spouse.elem.getBounds().height / 2,
                        });
                    }
```

```
                    let personRel = person.relations.find(rel => rel.id == spouse.id);
                    let spouseRel = spouse.relations.find(rel => rel.id == person.id);
                    let line = drawLine(dots);
                    line.name = 'spouse-line';
                    container.addChild(line);

                    personRel.elem = line;
                    spouseRel.elem = line;

                    spouse.isPositioned = true;
                    container.addChild(spouse.elem);

                    containersArray.push({
                        elem: container,
                        persons: [person.id, spouse.id],
                        containers: null,
                        generations:                                    [person.generation,
spouse.generation].filter(onlyUnique),
                        isPositioned: false,
                    });
                }
            } else {
                containersArray.push({
                    elem: container,
                    persons: [person.id],
                    containers: null,
                    generations: [person.generation].filter(onlyUnique),
                    isPositioned: false,
                });
            }

            mainContainer.addChild(container)
        }
    }
    function makePersonAndSpouseForAncestors(person) {
        let personContainer = containersArray.find(container =>
            container.persons.includes(person.id) &&
            !container.generations.includes(person.generation + 1));

        console.log(person, personContainer);
        return;
        if(!person.elem) {
            // extra person making
        }
        if(!person.isPositioned) {
            person.elem.position.y = 0;
            person.elem.position.x = 0;
            person.isPositioned = true;
```

```
            let container = new PIXI.Container();
            container.name = 'spouses';
            container.addChild(person.elem);

            // many spouses

            let spouseRel = person.relations.find(rel => rel.type.includes('spouse'));
            if(spouseRel) {
                let spouse = persons.find(person => person.id == spouseRel.id);
                if (spouse && !spouse.isPositioned) {

                    let dots = [];
                    dots.push({
                        x: person.elem.getBounds().x + person.elem.getBounds().width,
                        y: person.elem.getBounds().y + person.elem.getBounds().height / 2,
                    });

                    spouse.elem.position.y = 0;
                    if (spouse.sex == 'man') {
                        spouse.elem.position.x   =   person.elem.position.x   -   gapDist.x   -
spouse.elem.getBounds().width;
                        dots.push({
                            x: spouse.elem.getBounds().x + spouse.elem.getBounds().width,
                            y: spouse.elem.getBounds().y + spouse.elem.getBounds().height / 2,
                        });
                    } else {
                        spouse.elem.position.x        =        person.elem.position.x        +
person.elem.getBounds().width + gapDist.x;
                        dots.push({
                            x: spouse.elem.getBounds().x,
                            y: spouse.elem.getBounds().y + spouse.elem.getBounds().height / 2,
                        });
                    }

                    let personRel = person.relations.find(rel => rel.id == spouse.id);
                    let spouseRel = spouse.relations.find(rel => rel.id == person.id);
                    let line = drawLine(dots);
                    line.name = 'spouse-line';
                    container.addChild(line);

                    personRel.elem = line;
                    spouseRel.elem = line;

                    spouse.isPositioned = true;
                    container.addChild(spouse.elem);

                    containersArray.push({
                        elem: container,
                        persons: [person.id, spouse.id],
```

```
                        containers: null,
                        generations:                                    [person.generation,
spouse.generation].filter(onlyUnique),
                        isPositioned: false,
                    });
                }
            } else {
                containersArray.push({
                    elem: container,
                    persons: [person.id],
                    containers: null,
                    generations: [person.generation].filter(onlyUnique),
                    isPositioned: false,
                });
            }

            mainContainer.addChild(container)
        }
    }
    function positioningSiblings(person) {

        let persContainers = containersArray.filter(container =>
            container.persons.includes(person.id) &&
            !container.generations.includes(person.generation - 1));

        persContainers.sort((a, b) => b.generations.length - a.generations.length);
        let persContainer = persContainers[0];

        if(!persContainer.isPositioned) {
            persContainer.elem.position.y = 0;
            persContainer.elem.position.x = 0;
            persContainer.isPositioned = true;

            let container = new PIXI.Container();
            container.name = 'siblings';
            container.addChild(persContainer.elem);

            let allGenerations = [],
                allPersons = [];

            allGenerations = [...allGenerations, ...persContainer.generations];
            allPersons = [...allPersons, ...persContainer.persons];

            let siblingRels = person.relations.filter(rel => rel.type.includes('sibling'));

            if (siblingRels.length > 0) {

                let          xPos          =          persContainer.elem.position.x          +
persContainer.elem.getBounds().width + gapDist.x * 2;
```

```
                //    let    lastSibling    =    persons.find(person    =>    person.id    ==
siblingRels[siblingRels.length-1].id);

                siblingRels.forEach(siblingRel => {
                    let sibling = persons.find(person => person.id == siblingRel.id);

                    if (sibling) {
                        if (!sibling.isPositioned) {
                            makePersonAndSpouse(sibling);
                        }

                        let sibContainers = containersArray.filter(container =>
                            container.persons.includes(sibling.id) &&
                            !container.generations.includes(sibling.generation    +    1)    &&
!container.isPositioned);

                        sibContainers.sort((a,      b)      =>      b.generation.length      -
a.generation.length);
                        let sibContainer = sibContainers[0];

                        if (sibContainer && !sibContainer.isPositioned) {
                            sibContainer.elem.position.y = 0;
                            sibContainer.elem.position.x = xPos;


                            container.addChild(sibContainer.elem);
                            sibContainer.isPositioned = true;

                            allGenerations = [...allGenerations, ...sibContainer.generations];
                            allPersons = [...allPersons, ...sibContainer.persons];

                            xPos += (sibContainer.elem.getBounds().width + gapDist.x * 2);
                        }
                    }
                });

                // let firstLine = person.relations.find(rel => rel.type.includes('parent'));
                //      let      lastLine      =      lastSibling.relations.find(rel      =>
rel.type.includes('parent'));
                // let dots = [];
                //
                // if(firstLine) {
                //     dots.push({
                //         x: firstLine.elem.getBounds().x,
                //         y: firstLine.elem.getBounds().y,
                //     });
                // }
                // if(lastLine) {
                //     dots.push({
```

```
            //          x: lastLine.elem.getBounds().x + 2,
            //          y: lastLine.elem.getBounds().y,
            //      });
            // }
            //
            // if(dots.length >= 2) {
            //      let line = drawLine(dots);
            //      line.name = 'between-siblings-line';
            //      container.addChild(line);
            // }
        }

        containersArray.push({
            elem: container,
            persons: [...allPersons],
            containers: null,
            generations: [...allGenerations].filter(onlyUnique),
            isPositioned: false,
        });

        mainContainer.addChild(container)
    }
}

function positioningParentsAndChildren(parent) {

    let parentsContainer = containersArray.find(container =>
        container.persons.includes(parent.id) &&
        !container.generations.includes(parent.generation + 1));

    if (!parentsContainer.isPositioned) {
        parentsContainer.elem.position.y = 0;
        parentsContainer.elem.position.x = 0;
        parentsContainer.isPositioned = true;

        let container = new PIXI.Container();
        container.name = 'parents and children';
        container.addChild(parentsContainer.elem);

        let allGenerations = [],
            allPersons = [];

        allGenerations = [...allGenerations, ...parentsContainer.generations];
        allPersons = [...allPersons, ...parentsContainer.persons];

        let childrenRels = parent.relations.filter(rel => rel.type.includes('child'));

        if (childrenRels.length > 0) {
```

```
                childrenRels.forEach(childrenRel => {
                    let child = persons.find(person => person.id == childrenRel.id);

                    if (child) {
                        let childrenContainers = containersArray.filter(container =>
                            container.persons.includes(child.id) &&
                            !container.generations.includes(parent.generation)             &&
!container.isPositioned);

                        childrenContainers.sort((a,     b)     =>     b.generation.length     -
a.generation.length);
                        let childrenContainer = childrenContainers[0];

                        if (childrenContainer && !childrenContainer.isPositioned) {

                            let yPos = parentsContainer.elem.getBounds().height + gapDist.y;
                            let xPos = 0;

                            if(childrenRels.length > 1) {
                                if         (parentsContainer.elem.getBounds().width         <=
childrenContainer.elem.getBounds().width) {
                                    xPos   =   -((childrenContainer.elem.getBounds().width   -
parentsContainer.elem.getBounds().width) / 2);
                                } else {
                                    xPos    =    (parentsContainer.elem.getBounds().width    -
childrenContainer.elem.getBounds().width) / 2;
                                }
                            } else {
                                xPos         =         (child.elem.getBounds().width         -
parentsContainer.elem.getBounds().width) / 2;
                                xPos += child.elem.getBounds().x;
                                xPos = -xPos;
                            }

                            childrenContainer.elem.position.y = yPos;
                            childrenContainer.elem.position.x = xPos;
                            childrenContainer.isPositioned = true;

                            container.addChild(childrenContainer.elem);

                            allGenerations               =               [...allGenerations,
...childrenContainer.generations];
                            allPersons = [...allPersons, ...childrenContainer.persons];
                        }

                        drawParentsToChildLines(parent, child);

                        function drawParentsToChildLines(parent, child) {
                            let     secondParentId     =     child.relations.find(rel     =>
```

```
rel.type.includes('parent') && rel.id != parent.id)?.id ?? null;
                            let secondParent = null;

                            if (secondParentId || secondParentId == 0) {
                                secondParent   =   persons.find(person   =>   person.id   ==
secondParentId);
                            } else {
                                secondParent = persons.find(person => person.id == parent.id);
                            }

                            let   hasChildCurrentParent   =   child.relations.find(rel   =>
rel.type.includes('parent') && rel.id == parent.id);
                            let dotsParent = [], dotsSecondParent = [];
                            if(child.id == 5) {
                                console.log(parent,        secondParent,        child       ,
hasChildCurrentParent)
                            }
                            if(hasChildCurrentParent) {
                                if (parent.id != secondParent.id) {

                                    dotsParent.push({
                                        x: parentsContainer.elem.getBounds().width / 2,
                                        y: parent.elem.getBounds().height / 2,
                                    });
                                    dotsParent.push({
                                        x: parentsContainer.elem.getBounds().width / 2,
                                        y:     parentsContainer.elem.getBounds().height     +
gapDist.y / 2,
                                    });
                                    /*
                                                                                            //
dotsSecondParent.push({
                                                                                            //                  x:
parentsContainer.elem.getBounds().width / 2,
                                                                                            //                  y:
secondParent.elem.getBounds().height / 2,
                                                                                            // });
                                                                                            //
dotsSecondParent.push({
                                                                                            //                  x:
parentsContainer.elem.getBounds().width / 2,
                                                                                            //                  y:
parentsContainer.elem.getBounds().height + gapDist.y / 2,
                                                                                            // });
                                                                                            //
dotsSecondParent.push({
                                                                                            //                  x:
child.elem.getBounds().x + child.elem.getBounds().width / 2,
                                                                                            //                  y:
```

```
parentsContainer.elem.getBounds().height + gapDist.y / 2,
                                                                    // });
                                                                    //
dotsSecondParent.push({
                                                                    //                  x:
child.elem.getBounds().x + child.elem.getBounds().width / 2,
                                                                    //                  y:
child.elem.getBounds().y,
                                                                    // });
                                                                    //    let    line    =
drawLine(dotsSecondParent);
                                                                    //     line.name     =
'parent-children-line';
                                                                    //
                                                                    // let childRelParent
= child.relations.find(rel => rel.id == secondParent.id);
                                                                    //                  if
(childRelParent) {
                                                                    //
childRelParent.elem = line;
                                                                    // }
                                                                    // let parentRelChild
= secondParent.relations.find(rel => rel.id == child.id);
                                                                    //                  if
(parentRelChild) {
                                                                    //
parentRelChild.elem = line;
                                                                    // }
                                                                    // // set to rels

                                                                    //
container.addChild(line);
                                                                    */
                                } else {
                                    console.log(child, parent);

                                    console.log(childrenContainer);
                                    if(child) {
                                        let newXPos = child.elem.getBounds().x;
                                        if(child.elem.getBounds().width                    >
parent.elem.getBounds().width) {
                                            newXPos    +=    (child.elem.getBounds().width    -
parent.elem.getBounds().width) / 2;
                                        }
                                        parent.elem.position.x = newXPos;

                                        console.log(parentsContainer.elem.children.find(elem
=> elem.name == "spouse-line"));
                                        let                    spouseLine                   =
```

```
parentsContainer.elem.children.find(elem => elem.name == "spouse-line");
                                        if(spouseLine) {
                                            spouseLine.getBounds().width                    =
parentsContainer.elem.getBounds().width - parent.elem.getBounds().width * 2;
                                        }

                                        let spouseDots = [];
                                        spouseDots.push({
                                            x:      parentsContainer.elem.getBounds().x      +
parent.elem.getBounds().width,
                                            y: parentsContainer.elem.getBounds().height / 2,
                                        });
                                        spouseDots.push({
                                            x:      parentsContainer.elem.getBounds().x      +
parentsContainer.elem.getBounds().width - parent.elem.getBounds().width,
                                            y: parentsContainer.elem.getBounds().height / 2,
                                        });
                                        let line = drawLine(spouseDots);
                                        parentsContainer.elem.addChild(line);

                                        // записати лінію у rels
                                    }

                                    dotsParent.push({
                                        x:             parent.elem.getBounds().x             +
parent.elem.getBounds().width / 2,
                                        y: parent.elem.getBounds().height,
                                    });
                                    dotsParent.push({
                                        x:             parent.elem.getBounds().x             +
parent.elem.getBounds().width / 2,
                                        y: parent.elem.getBounds().height + gapDist.y / 2,
                                    });
                                }
                                dotsParent.push({
                                    x: child.elem.getBounds().x + child.elem.getBounds().width
/ 2,
                                    y: parentsContainer.elem.getBounds().height + gapDist.y /
2,
                                });
                                dotsParent.push({
                                    x: child.elem.getBounds().x + child.elem.getBounds().width
/ 2,
                                    y: child.elem.getBounds().y,
                                });
                                let line = drawLine(dotsParent);
                                line.name = 'parent-children-line';

                                let  childRelParent  =  child.relations.find(rel  =>  rel.id  ==
```

```
parent.id);
                                let parentRelChild = parent.relations.find(rel => rel.id ==
child.id);

                                if (!childRelParent?.elem || !parentRelChild?.elem) {
                                    container.addChild(line);
                                }

                                if (childRelParent) {
                                    childRelParent.elem = line;
                                }
                                if (parentRelChild) {
                                    parentRelChild.elem = line;
                                }
                            }
                            // set to rels

                            //  console.log(secondParent,  secondParent.relations.find(rel  =>
rel.type.includes('child') && rel.elem == undefined));
                            let   noDrewChildRel   =   secondParent.relations.find(rel   =>
rel.type.includes('child') && rel.elem == undefined);
                            if(noDrewChildRel) {
                                let  noDrewChild  =  persons.find(person  =>  person.id  ==
noDrewChildRel.id);
                                if(noDrewChild) {
                                    // потрібно якось враховувати, щоб не створювались повтори
старого parent
                                    drawParentsToChildLines(secondParent, noDrewChild);
                                }
                            }
                        }
                    }
                });
            }

            containersArray.push({
                elem: container,
                persons: [...allPersons],
                containers: null,
                generations: [...allGenerations].filter(onlyUnique),
                isPositioned: false,
            });

            mainContainer.addChild(container);
        }
    }
    function positioningParentsAndChildrenForAncestors(parent) {

        if (!parent.isPositioned) {
```

```
            let container = new PIXI.Container();
            container.name = 'parents and children';

            let allGenerations = [],
                allPersons = [];

            allGenerations = [...allGenerations, parent.generation];
            allPersons = [...allPersons, parent];

            let childrenRels = parent.relations.filter(rel => rel.type.includes('parent'));

            if (childrenRels.length > 0) {

                childrenRels.forEach(childrenRel => {
                    let child = persons.find(person => person.id == childrenRel.id);

                    if (child) {
                        let childrenContainers = containersArray.filter(container =>
                            container.persons.includes(child.id) &&
                            !container.generations.includes(parent.generation)             &&
!container.isPositioned);

                        childrenContainers.sort((a,     b)     =>     b.generation.length     -
a.generation.length);
                        let childrenContainer = childrenContainers[0];

                        if (childrenContainer && !childrenContainer.isPositioned) {

                            let yPos = childrenContainer.elem.getBounds().height - gapDist.y;
                            let xPos = 0;

                            if(childrenContainer.elem.getBounds().width                        >
parent.elem.getBounds().width) {
                                xPos          =          parent.elem.getBounds().x          -
(childrenContainer.elem.getBounds().width - parent.elem.getBounds().width) / 2;
                            } else {
                                xPos = parent.elem.getBounds().x;
                            }

                            childrenContainer.elem.position.y = yPos;
                            childrenContainer.elem.position.x = xPos;
                            childrenContainer.isPositioned = true;

                            container.addChild(parent.elem);
                            container.addChild(childrenContainer.elem);

                            allGenerations               =               [...allGenerations,
...childrenContainer.generations];
```

```
                allPersons = [...allPersons, ...childrenContainer.persons];
            }

            console.log(childrenContainer);
            drawParentsToChildLines(parent, child, childrenContainer);

            function drawParentsToChildLines(child, parent, parentsContainer) {

                let secondParentId = child.relations.find(rel =>
rel.type.includes('parent') && rel.id != parent.id)?.id ?? null;
                let secondParent = null;

                if (secondParentId || secondParentId == 0) {
                    secondParent = persons.find(person => person.id ==
secondParentId);
                } else {
                    secondParent = persons.find(person => person.id ==
parent.id);
                }

                let hasChildCurrentParent = child.relations.find(rel =>
rel.type.includes('parent') && rel.id == parent.id);
                let dotsParent = [];

                if(hasChildCurrentParent && parentsContainer) {
                    if (parent.id != secondParent.id) {

                        console.log(parentsContainer);
                        dotsParent.push({
                            x: parentsContainer.elem.getBounds().width / 2,
                            y: parent.elem.getBounds().height / 2,
                        });
                        dotsParent.push({
                            x: parentsContainer.elem.getBounds().width / 2,
                            y: parentsContainer.elem.getBounds().height +
gapDist.y / 2,
                        });

                    }
                    else {
                        console.log(child, parent);

                        if(child) {
                            let newXPos = child.elem.getBounds().x;
                            if(child.elem.getBounds().width >
parent.elem.getBounds().width) {
                                newXPos += (child.elem.getBounds().width -
parent.elem.getBounds().width) / 2;
                            }
```

```
                                        parent.elem.position.x = newXPos;

                                        console.log(parentsContainer);

console.log(parentsContainer.elem.children.find(elem => elem.name == "spouse-line"));
                                        let                  spouseLine                  =
parentsContainer.elem.children.find(elem => elem.name == "spouse-line");
                                        if(spouseLine) {
                                            spouseLine.getBounds().width                 =
parentsContainer.elem.getBounds().width - parent.elem.getBounds().width * 2;
                                        }

                                        let spouseDots = [];
                                        spouseDots.push({
                                            x:    parentsContainer.elem.getBounds().x    +
parent.elem.getBounds().width,
                                            y: parentsContainer.elem.getBounds().height /
2,
                                        });
                                        spouseDots.push({
                                            x:    parentsContainer.elem.getBounds().x    +
parentsContainer.elem.getBounds().width - parent.elem.getBounds().width,
                                            y: parentsContainer.elem.getBounds().height /
2,
                                        });
                                        let line = drawLine(spouseDots);
                                        parentsContainer.elem.addChild(line);

                                        // записати лінію у rels
                                    }

                                    dotsParent.push({
                                        x:           parent.elem.getBounds().x           +
parent.elem.getBounds().width / 2,
                                        y: parent.elem.getBounds().height,
                                    });
                                    dotsParent.push({
                                        x:           parent.elem.getBounds().x           +
parent.elem.getBounds().width / 2,
                                        y: parent.elem.getBounds().height + gapDist.y / 2,
                                    });
                                }
                                dotsParent.push({
                                    x:              child.elem.getBounds().x             +
child.elem.getBounds().width / 2,
                                    y:     parentsContainer.elem.getBounds().height     +
gapDist.y / 2,
                                });
                                dotsParent.push({
```

```
                                        x:              child.elem.getBounds().x             +
child.elem.getBounds().width / 2,
                                        y: child.elem.getBounds().y,
                                    });
                                    let line = drawLine(dotsParent);
                                    line.name = 'parent-children-line';

                                    let childRelParent = child.relations.find(rel => rel.id ==
parent.id);
                                    let parentRelChild = parent.relations.find(rel => rel.id
== child.id);

                                    if (!childRelParent?.elem || !parentRelChild?.elem) {
                                        container.addChild(line);
                                    }

                                    if (childRelParent) {
                                        childRelParent.elem = line;
                                    }
                                    if (parentRelChild) {
                                        parentRelChild.elem = line;
                                    }
                                }
                                // set to rels

                                //  console.log(secondParent,  secondParent.relations.find(rel
=> rel.type.includes('child') && rel.elem == undefined));
                                let   noDrewChildRel   =   secondParent.relations.find(rel   =>
rel.type.includes('child') && rel.elem == undefined);
                                if(noDrewChildRel) {
                                    let  noDrewChild  =  persons.find(person  =>  person.id  ==
noDrewChildRel.id);
                                    if(noDrewChild) {
                                        // потрібно якось враховувати, щоб не створювались
повтори старого parent
                                        drawParentsToChildLines(noDrewChild,    secondParent,
parentsContainer);
                                    }
                                }

                            }
                        }
                    });
            }

            containersArray.push({
                elem: container,
                persons: [...allPersons],
                containers: null,
```

```
                generations: [...allGenerations].filter(onlyUnique),
                isPositioned: false,
            });
            mainContainer.addChild(container);
        }
    }
    function selectCard(e) {
        $selectedPersonId = e.detail;
        console.log($selectedPersonId);
    }
    function scrollCanvas(e) {
        e.preventDefault();
        if(!$isShiftPressed && !$isCtrlPressed) {
            mainContainer.position.y = mainContainer.position.y + (e.deltaY > 0 ? 100 : -100);
        }
        if($isShiftPressed && !$isCtrlPressed) {
            mainContainer.position.x = mainContainer.position.x + (e.deltaY > 0 ? 100 : -100);
        }
        if($isCtrlPressed && !$isShiftPressed) {
            treeScale.set($treeScale * (e.deltaY < 0 ? 1.1 : 0.9));
        }
    }

    onMount(()=>{
        app     =     new     PIXI.Application({     background:     '#f9f9f9',     resizeTo:
document.querySelector("#tree-page") });

        globalThis.__PIXI_APP__ = app;

        canvasElem.appendChild(app.view);

        mainContainer = new PIXI.Container();
        mainContainer.name = "main";
        app.stage.addChild(mainContainer);

        drawTree();

        // mainContainer.x = (app.renderer.width - mainContainer.getBounds().width) / 2;
        mainContainer.x = mainContainer.getBounds().width/2;
        mainContainer.y = (app.renderer.height - mainContainer.getBounds().height) / 2;
    });
</script>

<div tabindex="0" role="button" aria-pressed="false"
     bind:this={canvasElem} on:selectCard={selectCard} on:mousewheel={scrollCanvas} ></div>
```

ДОДАТОК Г
ГРАФІЧНА ЧАСТИНА

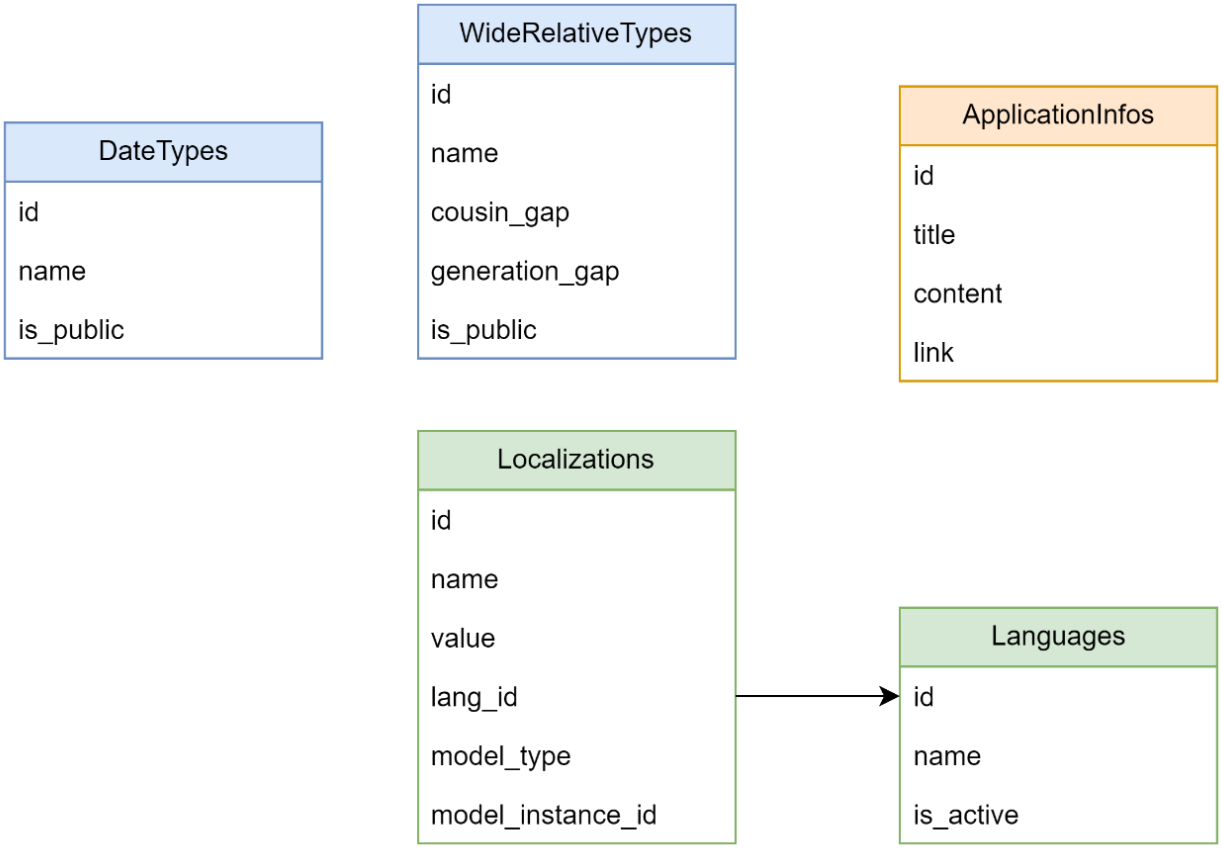

Рисунок Г.1 – Допоміжні сутності

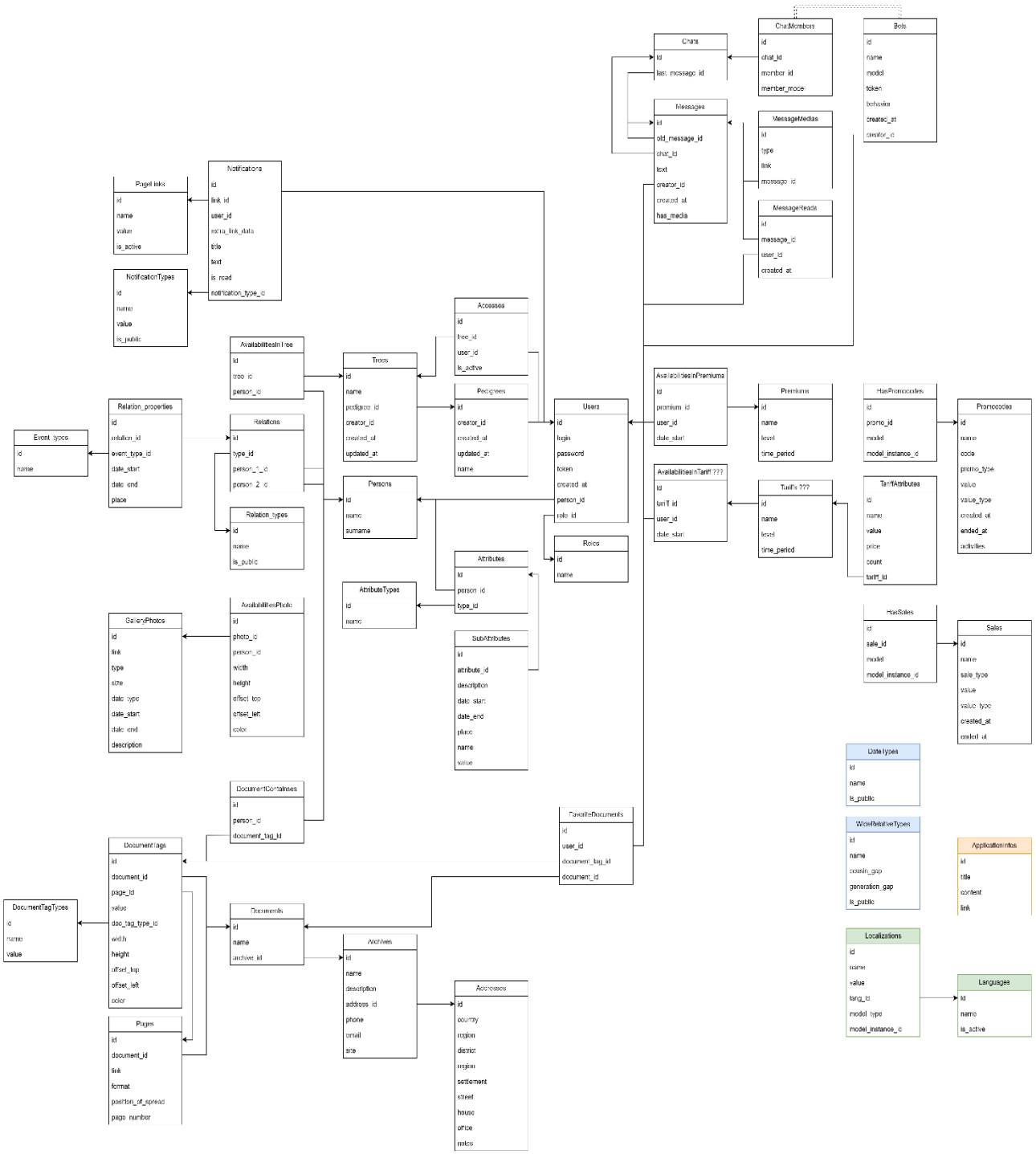

Рисунок Г.2 – Повна ER-діаграма

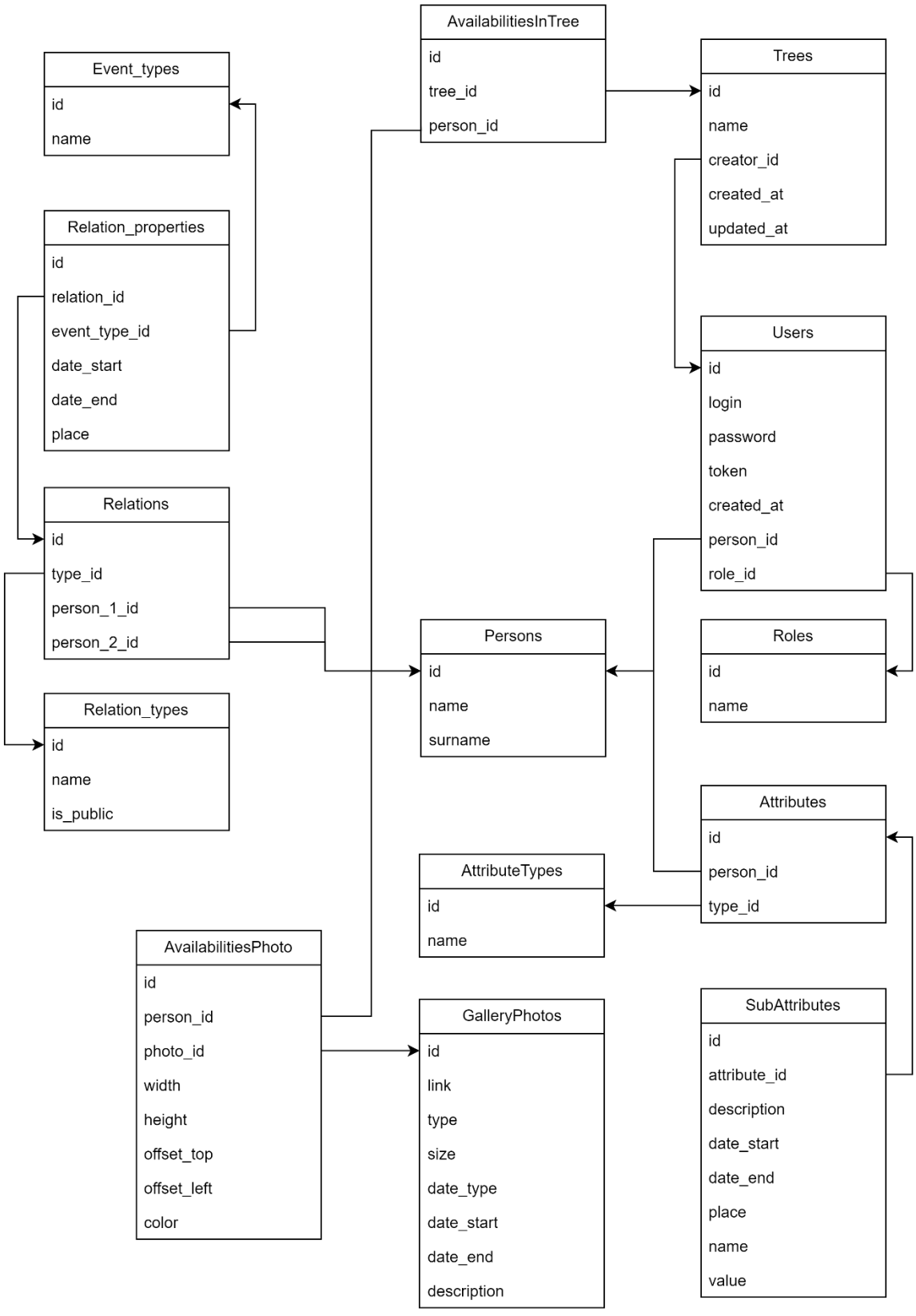

Рисунок Г.3 – Модуль керування деревом (Головний модуль)

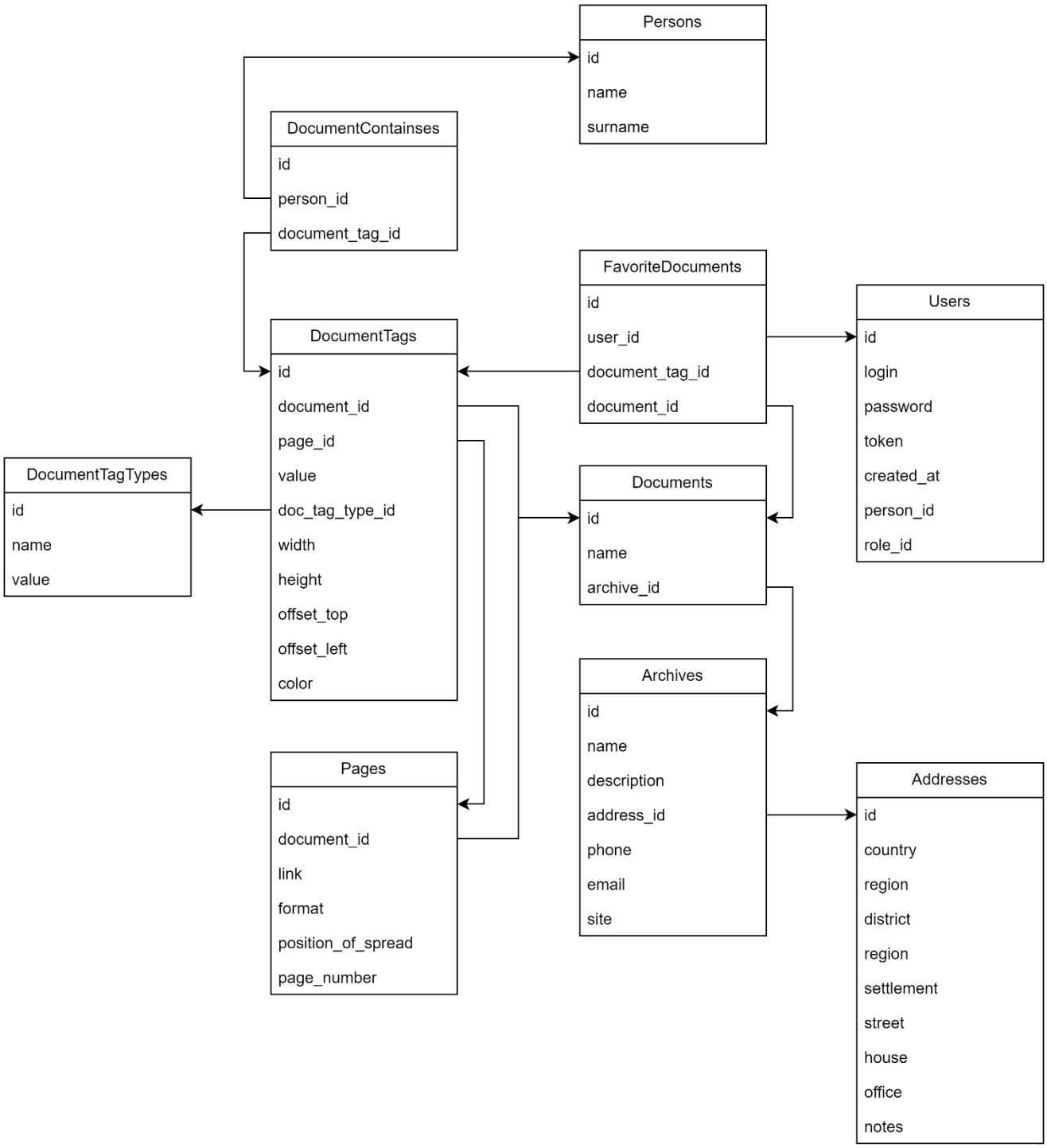

Рисунок Г.4 – Модуль документообігу

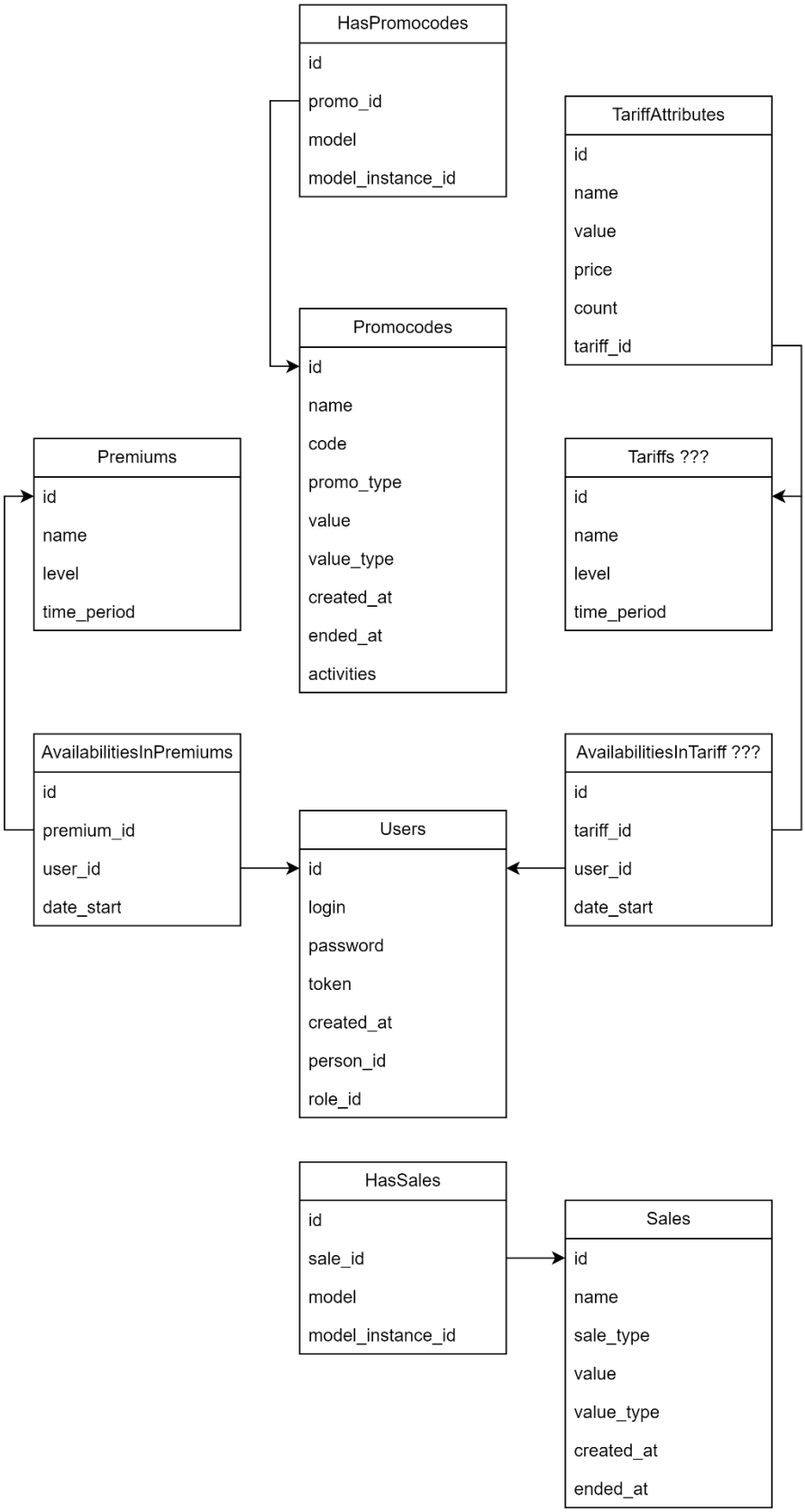

Рисунок Г.5 – Модуль тарифних планів

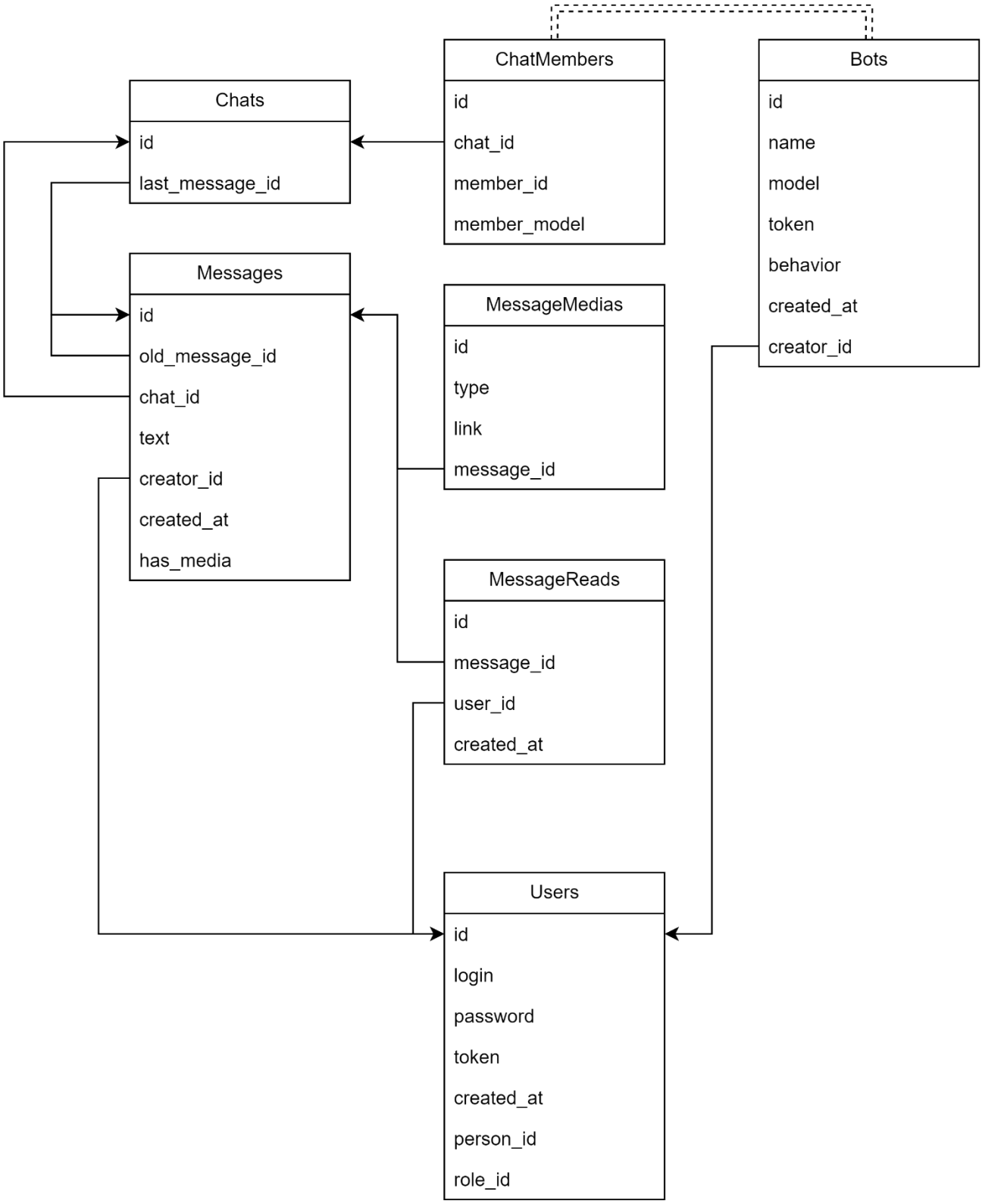

Рисунок Г.6 – Модуль чатів

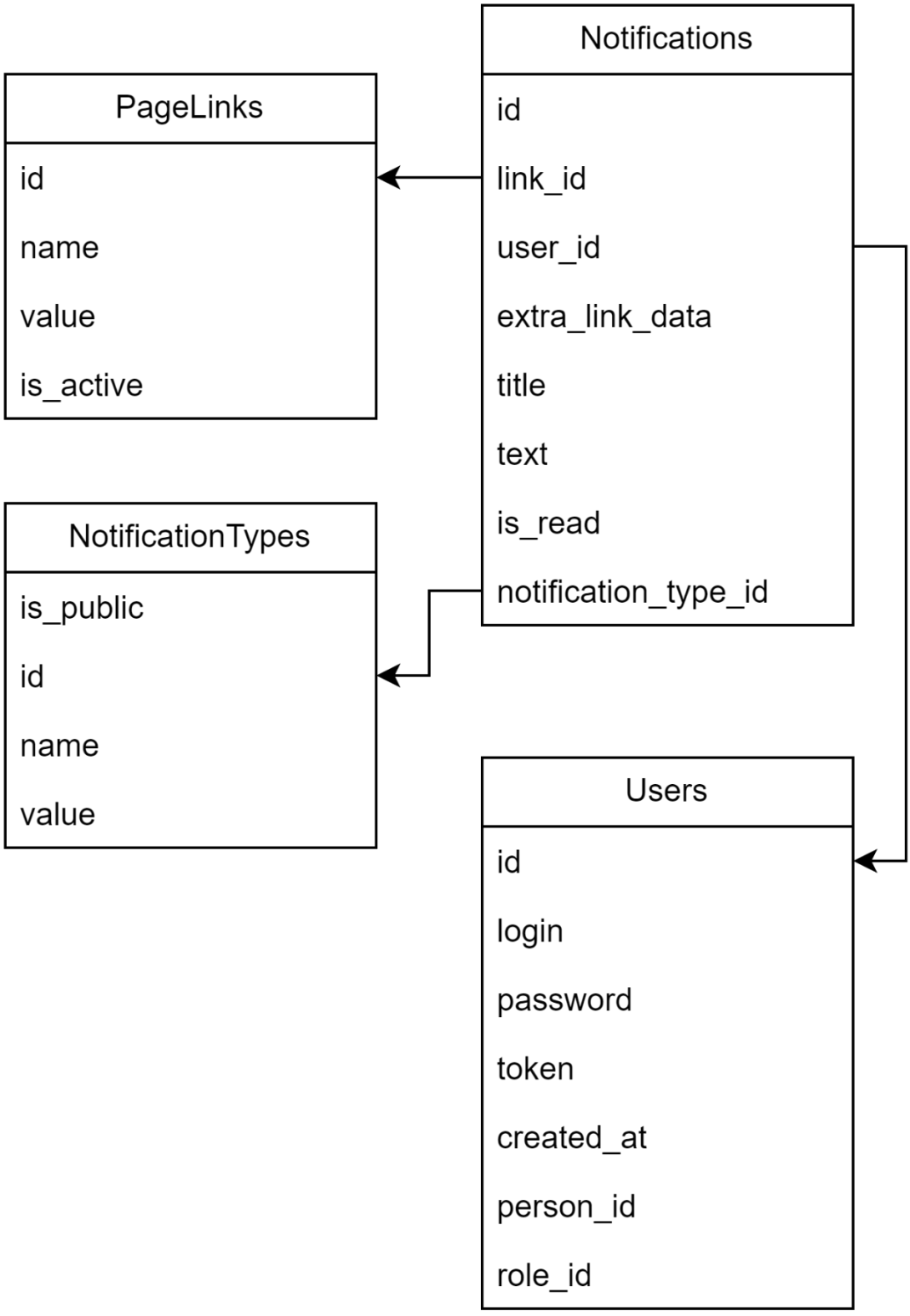

Рисунок Г.7 – Модуль сповіщень користувачів

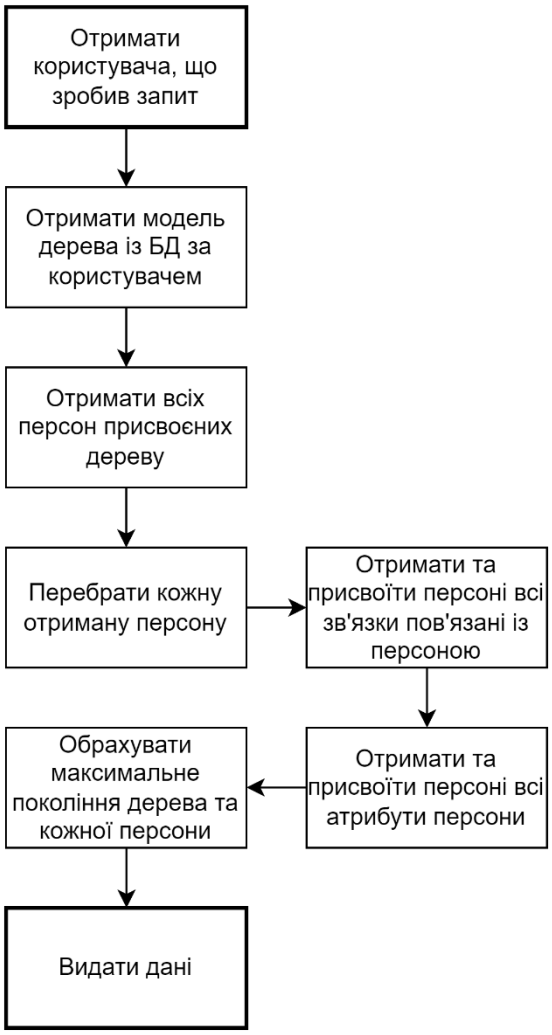

Рисунок Г.8 – Алгоритм для видачі даних дерева на клієнтську частину

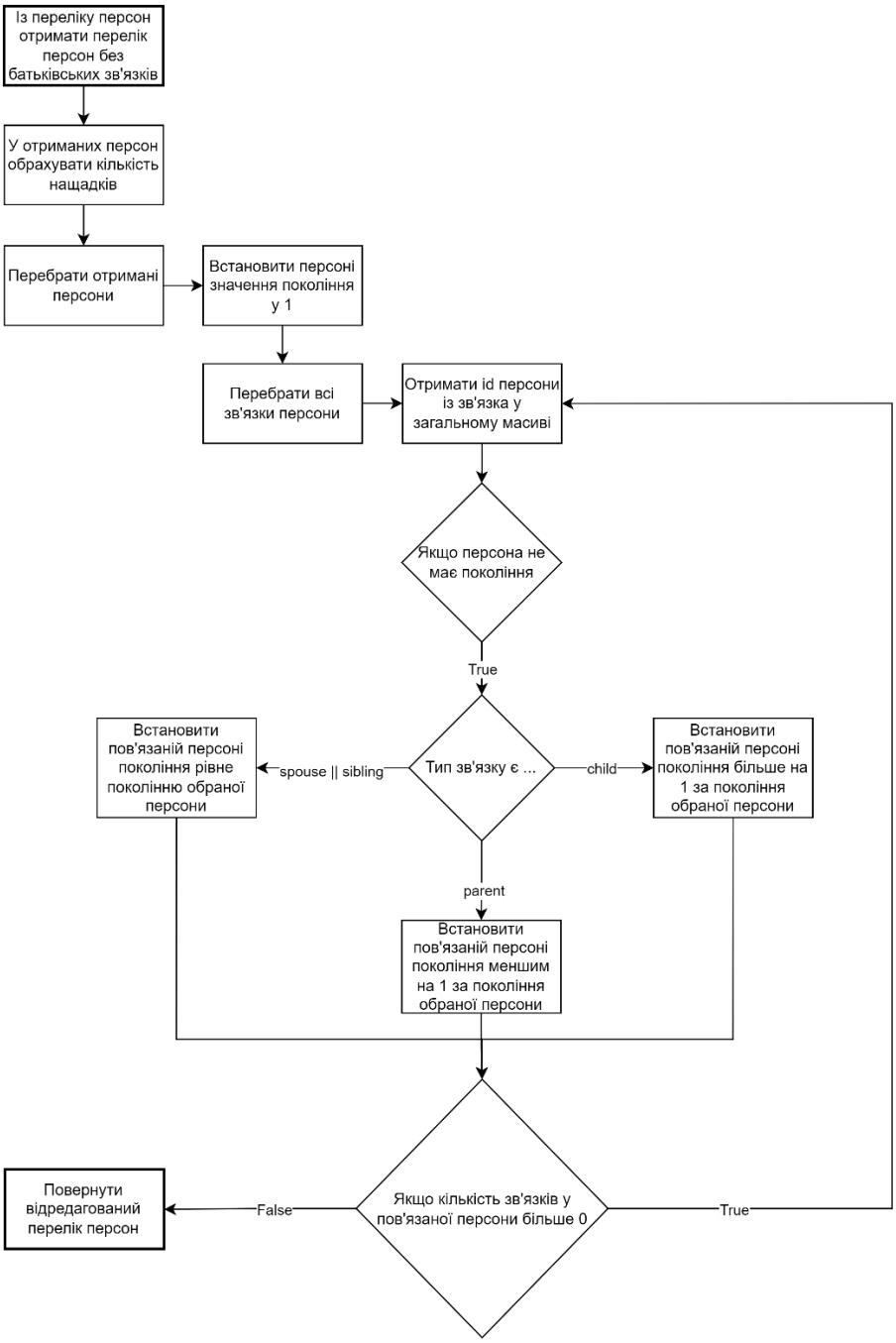

Рисунок Г.9 – Алгоритм обрахунку поколінь персон дерева

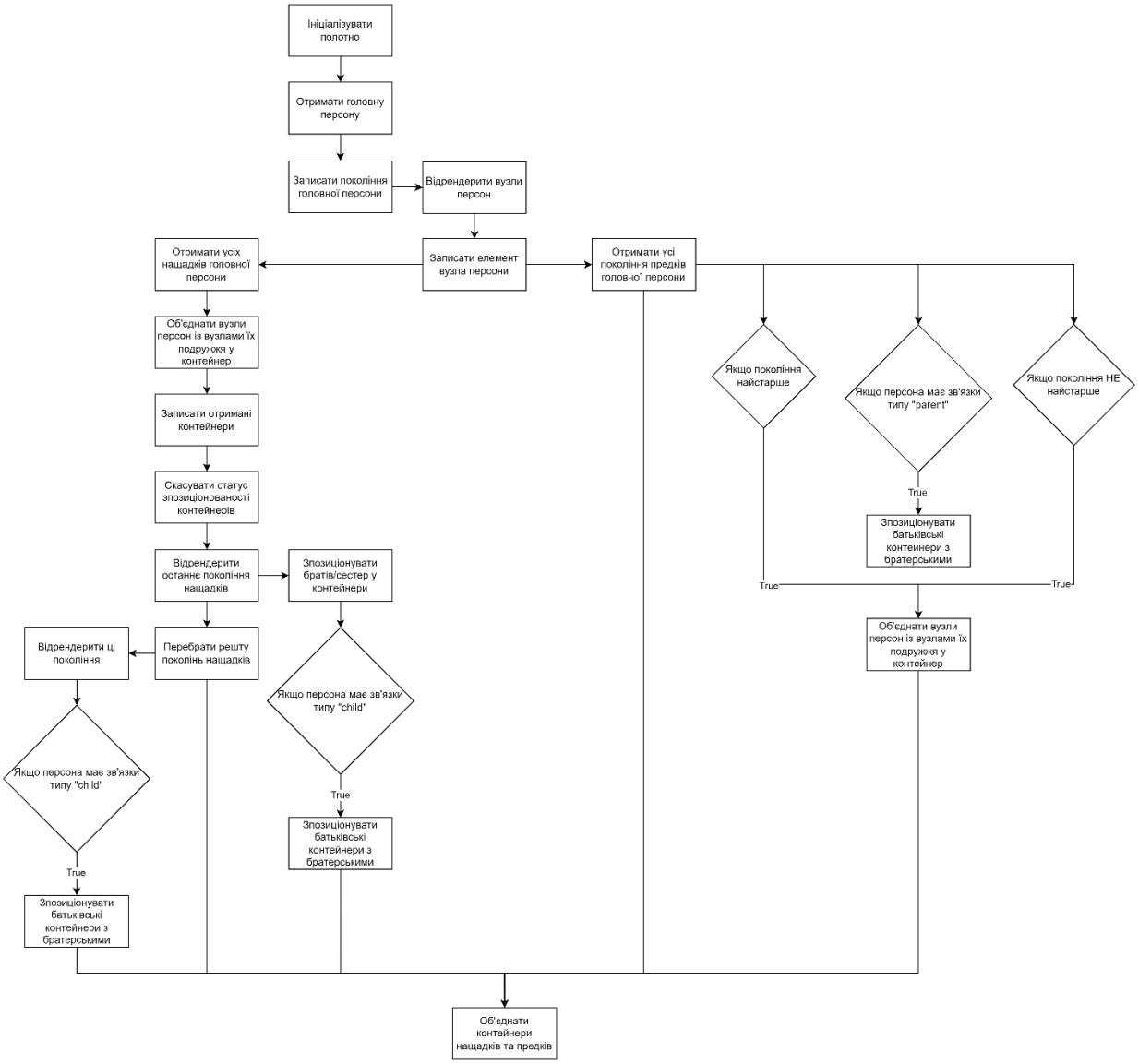

Рисунок Г.10 – Алгоритм рендеру дерева на клієнтській частині