Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Морара Микити Сергійовича*
(ПІБ)

академічної групи *122-17-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-сервісу для створення родинних дерев та вивчення родоводу*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*доц. Реута О.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-17-3* *Морара М.С.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-сервісу для створення родинних дерев та вивчення родоводу*

затверджена наказом ректора НТУ «ДП» від *07.06.2021 р.* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Реута О.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Морар М.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 97 с., 44 рис., 1 табл., 3 дод., 29 джерел.

Об'єкт розробки: веб-сервіс для створення родинних дерев та вивчення родоводу.

Мета кваліфікаційної роботи: розробка веб-сервісу для створення родинних дерев та вивчення родоводу, який надаватиме користувачам можливість створювати свої сім'ї, наповнювати їх людьми, заповнювати даними профілі людей, запрошувати людей в сервіс, проводити пошук людей по сервісу і переглядати сімейне дерево в зручній формі, а розробникам надасть зручний API інструмент для створення своїх клієнтів.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної області та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у створенні веб-сервісу, що надає можливість збереження даних про членів сім'ї, наповнення та зміни розгорнутих даних про людей, можливість графічного виведення родинного дерева і пошуку родичів по системі.

Актуальність інформаційної системи визначається великим попитом на подібні системи і відсутністю безкоштовного і зручного рішення, яке ми поєднувало в собі багато функцій, корисних для генеалогів і людей, які хочуть вивчити родовід.

Список ключових слів: ПРОЕКТУВАННЯ, КЛІЄНТ, СЕРВЕР, ІНФОРМАЦІЙНА СИСТЕМА, ЕОМ, ВЕБ-СЕРВІС, API, REST, БАЗА ДАНИХ, ТОКЕН, РОДИННЕ ДЕРЕВО, ГЕНЕАЛОГІЯ .

ABSTRACT

Explanatory note: 97 pp., 44 figs., 1 tabs., 3 apps, 29 sources.

Object of development: web service for creating family trees and studying genealogy.

Purpose of the qualification work: development of a web service for creating family trees and pedigree study, which will allow users to create their families, fill them with people, fill in people's profiles, invite people to the service, search for people on the service and view the family tree. convenient form, and developers will provide a convenient API tool for creating their clients.

The introduction analyzes the current state of the problem, clarifies the problem, the purpose of the qualification work and the scope of its application, substantiates the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development are carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

The economic section determines the complexity of the developed software product, calculates the cost of work to create an application and calculates the time to create it.

The practical significance lies in the creation of a web service that provides the ability to save data about family members, fill in and change detailed data about people, the ability to graphically display the family tree and search for relatives in the system.

The relevance of the information system is determined by the high demand for such systems and the lack of a free and convenient solution, which we have combined many features useful for genealogists and people who want to learn pedigree.

Keywords: DESIGN, CLIENT, SERVER, INFORMATION SYSTEM, COMPUTER, WEB SERVICE, API, REST, DATABASE, TOKEN, FAMILY TREE, GENEALOGY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ОС – операційна система;

ЕОМ – електронно-обчислювальна машина;

API – це Application Programming Interface (з англ. інтерфейс для програмування застосунків);

REST – Representational State Transfer (з англ. передача репрезентативного стану);

JWT – JSON Web Token (з англ. JSON веб токен).

ВСТУП

Завдання даної кваліфікаційної роботи та об'єкт її діяльності безпосередньо пов'язані з напрямом підготовки та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки та інформаційні технології».

Тематикою кваліфікаційної роботи є розробка Web-сервісу для створення родинних дерев та вивчення родоводу.

Метою кваліфікаційної роботи є створення архітектури веб-сервісу та розробка функціоналу для нього. Кваліфікаційна робота також передбачає розробку бази даних і архітектури веб-сервісу, використання REST API, як архітектурного стилю взаємодії компонентів розподіленого сервісу в мережі та аналіз і порівняльну характеристику аналогів.

Після аналізу аналогічних сервісів та додатків були виявлені плюси та мінуси кожного, що допомогло при проектуванні власного веб-сервісу, який буде мати весь необхідний функціонал для задоволення потреб користувачів.

Причиною виникнення необхідності розробки програмного забезпечення є відсутність безкоштовного і відкритого рішення для побудови сімейних дерев без лімітів на кількість учасників .

Виходячи з вищеописаної інформації, було прийнято рішення розробити зручний та функціональний веб-сервіс для створення родинних дерев. Дане програмне забезпечення є актуальним через те, що воно, на відміну від більшості подібних, надає весь функціонал безкоштовно та не ставить лімітів у користуванні.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Генеалогія - систематичний збір відомостей про походження, спадкоємство і спорідненість сімей, відстеження родоводів і сімейних історій; в більш широкому сенсі - наука про родинні зв'язки [10]. Фахівці з генеалогії використовують усні відомості, історичні документи, генетичний аналіз і інші методи для отримання інформації про сім'ю і продемонструвати родинні зв'язки її членів .

Результати досліджень часто відображаються у вигляді діаграм або генеалогічних схем [9]. Запис генеалогічного дослідження може бути представлена як генеалогічне древо - воно відстежує нащадків або предків однієї конкретної людини. Генеалогічне древо - поняття, відоме практично кожному ще з початкової школи. Генеалогія як історична дисципліна пропонує чотири варіанти по оформленню інформації про родичів в єдине ціле на вибір (рис. 1.1): низхідне вертикальне древо (1), висхідне вертикальне древо (2), горизонтальне древо (3), кругова таблиця (4).

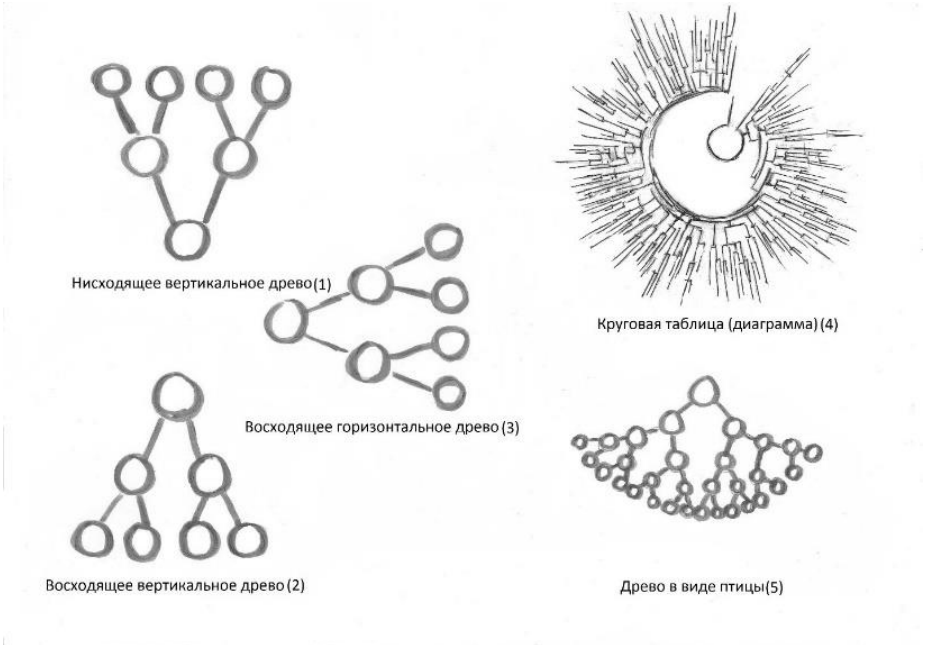

Рис. 1.1. Приклад схем відображення родинного дерева

Найпопулярнішим є поділ генеалогічних древ на висхідні і низхідні. При цьому найчастіше і той, і інший типи оформляються вертикально. У висхідному вертикальному дереві розпис будується від нащадків до предків, а в низхідному відповідно - від предків до нащадків.

В сучасному світі для вирішення задач генеалогії використовують програмне забезпечення, яке виконує таки функції, як накопичення, аналіз і публікація зібраних даних. Існує багато сервісів, які надають користувачеві можливість створення і відображення сімейних дерев, але багато хто з них вирішують тільки проблему відображення інформації у вигляді графіка, але не надають можливості заповнити інформацію для кожної конкретної людини.

Для аналізу проблем сучасних систем створення родинних дерев варто розглянути існуючі аналоги та виявити їх переваги та недоліки. Для дослідження були обрані такі відомі проекти як Family Tree Builder (MyHerritage), Agelong Tree, та GenoPro [8].

Древо Життя (Agelong Tree) - програма для побудови генеалогічних (родоводів) дерев для Windows і Mac (рис. 1.2.). Система надає такі функції, як зберігання, систематизацію та відображення зібраної інформації, в тому числі у вигляді генеалогічного дерева. Без покупки ліцензії не можна ввести більше 40 персон, всі інші можливості доступні повністю. Програма зберігає всі дані, що вводяться тільки локально, тобто на комп'ютері користувача, не завантажуючи їх в інтернет.

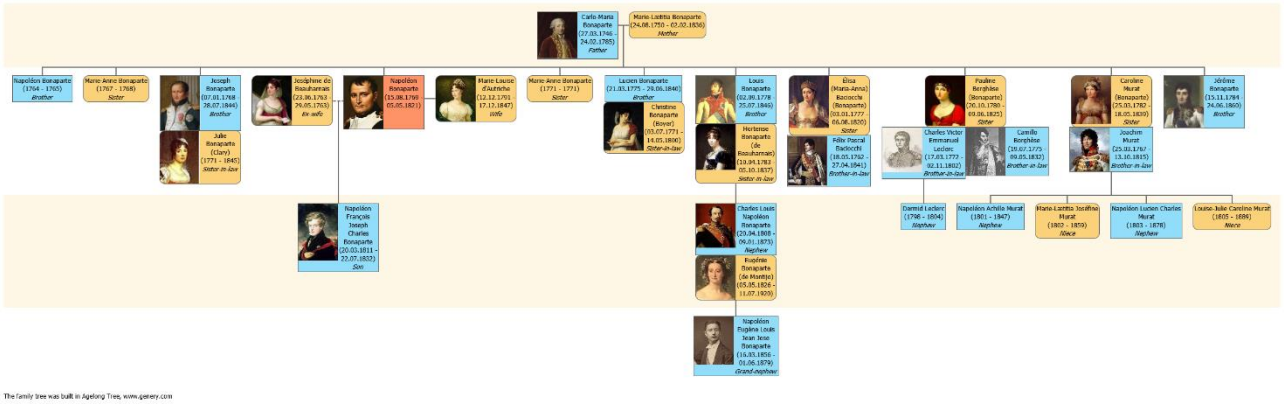

Рис. 1.2. Приклад програми Agelong Tree

Серед програм для побудови сімейного древа лідирує, згідно з відгуками користувачів, Family Tree Builder від ізраїльської компанії MyHeritage. Програма працює і зберігає локально на комп'ютері користувача (рис 1.3). Сервіс реалізує багато можливостей, з яких додавання картки інформації до кожної людини реалізовано дуже продумано. У картці кожної людини можна вказати джерела відомостей, життєві факти, архіви та їх адреси, телефони та документи. Програма дає можливість створити власний сімейний сайт і запросити на нього родичів.

Рис. 1.3. Приклад програми Family Tree Builder

GenoPro - це генеалогічна програма для складання генеалогічного древа і родоводів розписів (рис. 1.4.). Програма дає можливість відобразити повне графічне зображення генеалогічного древа. З GenoPro можна вільно створювати і змінювати графічну форму представлення всього древа предків і нащадків. Програма зручна і проста. GenoPro дає можливість побудувати фамільне древо декількома кліками миші. Ціна на програму починається від 42 євро і закінчується 1508 євро, в залежності від кількості учасником, які можуть користуватися деревом.

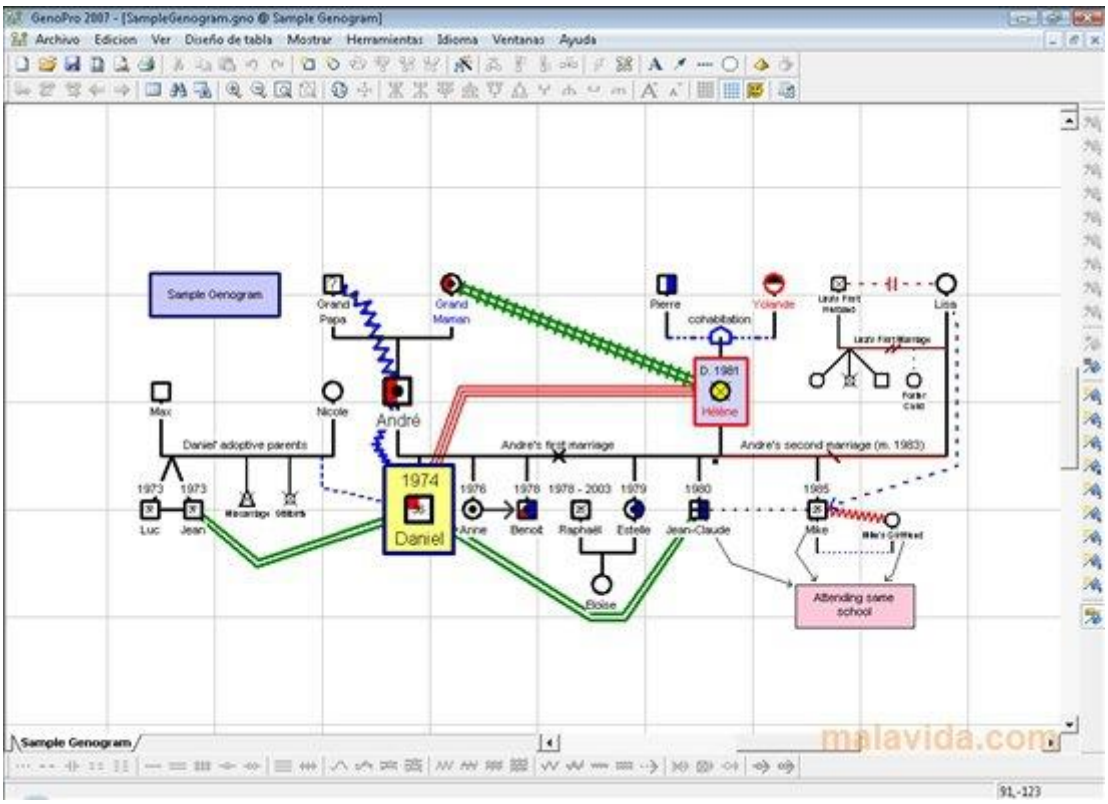

Рис. 1.4. Приклад програми GenPro

Після розгляду найпопулярніших систем для створення родоводу можна виділити плюси та мінуси кожної та вивести основні недоліки, які відносяться до кожної з представлених систем.

Family Tree Builder / MyHerritage – це на даний час золотий стандарт системи для створення родоводу, дуже логічно сформований сервіс, який дає купу можливостей користувачеві, від створення сімейного дерева та заповнення картки людини до колоризації старих фото (в останньому оновленні). Безперечним плюсом системи є те, що вона дає можливість вийти за рамки локального сховища та перенести дерево у веб-простір. Найбільшим мінусом цієї системи є вартість програмного забезпечення та ліміти на користування сервісом.

При досягненні ліміту в 250 чоловік у користувача обмежується доступ до можливостей сайту і продовжити роботу з родоводом можна тільки при оплаті річної передплати (вартістю від 100 до 189 доларів в рік.).

Agelong Tree – ця програма дозволяє заносити, редагувати, зберігати і візуалізувати генеалогічні дані. За внесеними даними може бути автоматично побудовано генеалогічне древо родичів будь-якої персони, причому не тільки прямих предків і нащадків, так і всіх кровних родичів персони. Програма також дозволяє зберігати мультимедійні дані (документи) з прив'язкою до персоналій. Програма має цікаву особливість, вона обчислює статистику по поколінням або по всьому дереву: кількість живих, чоловіків / жінок, середня кількість дітей, середню тривалість життя. У Agelong Tree є кілька мінусів. Перший і дуже істотний це те, що всі дані зберігаються локально і на одному комп'ютері, що наражається на ризик втратити всі дані при проблемах з жорстким диском. Другий - в програмі немає можливості поділитися доступом з іншими людьми, що ускладнює взаємодію з деревом. Безперечним мінусом є ціна і обмежена функціональність (до 40 людей в сімейному дереві) без оплаченої передплати.

GenoPro – ця програма є популярною через свою інтернаціональність (переведена і доступна на 25 мовах в 170 країнах світу). До плюсів можна віднести, що програма підтримує спектр можливостей для управління складними сценаріями і комплексного аналізу наявних даних. Що досить важливо, GenoPro має вражаючий генератор звітів, здатний створити повноцінні сторінки, інтерактивно пов'язані з зображеннями генеалогічних древ, виконаними за технологією SVG (Scalable Vector Graphics / Масштабуєма векторна графіка). До мінусів можна віднести те, що у програми зовсім немає безкоштовної або пробної версії, через що користувачеві складно оцінити надані можливості. Істотним мінусом так само можна назвати неінтуїтивний інтерфейс і непродумане відображення графіка з деревом, що ускладнює взаємодію з програмою.

Таблиця 1.1

Переваги та недоліки програм

Можливості	MyHeritage	Agelong Tree	GenoPro	Розроблювана система
Відображення членів сім'ї у вигляді дерева	Так	Так	Так	Так
Доступ через мережу Інтернет	Так	Ні	Ні	Так
Багатокористувацький доступ	Так	Ні	Ні	Так
Згортання гілок дерева	Ні	Так	Ні	Так
Ліміт на кількість людей у родинному дереві	Так (250)	Так (40)	Ні	Ні

З таблиці можливостей кожної з програм (табл. 1.1), можна зробити висновки, які можливості повинна надавати розроблювальна система, а яких недоліків вона повинна позбутися.

1.2. Призначення розробки та галузь застосування

Повна назва розробленої системи для кваліфікаційної роботи – «Web-сервіс для створення родинних дерев та вивчення родоводу».

Основна термінологія та ключові слова:

– Веб-сервіс – це веб-додаток, що надає ресурси в форматі, використовуваному іншими комп'ютерами. Веб-сервіси включають в себе різні типи API, в тому числі REST і SOAP API. Веб-сервіси – це, в основному, запити та відповіді між клієнтами і серверами (комп'ютер запитує ресурс, а веб-сервіс відповідає на запит). API, що використовують протокол HTTP для передачі запитів і відповідей, розглядаються як «веб-сервіси» [4].

– API (Application Programming Interface) – це контракт, який надає програма, забезпечує взаємодію між двома системами [3].

– REST (Representational State Transfer) (RESTful) – це архітектурний стиль, загальні принципи організації взаємодії додатка або сайту з сервером за допомогою протоколу HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами - в кожному запиті передається інформація, що ідентифікує користувача (наприклад, token, отриманий через OAuth-авторизацію) і всі параметри, необхідні для виконання операції [12].

Розроблений сервіс має використовуватися в галузі генеалогії генеалогами та зацікавленими людьми, які бажають зайнятися вивченням родоводу, створенням свого родинного дерева та упорядкуванням інформації про родичів на безкоштовній і відкритій основі.

Причиною виникнення необхідності розробки програмного забезпечення є відсутність безкоштовного і відкритого рішення для побудови сімейних дерев без лімітів на кількість учасників.

1.3. Підстави для розробки

Відповідно до освітньої програми , згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

- освітня програма спеціальності 122 «Комп'ютерні науки та інформаційні технології»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» №317-с від 07.06.2021 р;
- завдання на кваліфікаційну роботу на тему «Розробка веб-сервісу для створення родинних дерев та вивчення родоводу».

1.4. Постановка завдання

Метою завдання є розробка Web-сервісу для створення родинних дерев та вивчення родоводу, що дозволяє створювати родинні дерева, виводити їх у графічній формі, додавати інформацію в картку людини та шукати родичів. Розроблювальний web-сервіс повинен реалізувати наступні функції:

- додавання людей у дерево;
- додавання та редагування інформації про кожного члена родинного дерева;
- вивід родинного дерева у зручний графічний формат для перегляду його членів;
- запрошення людей до аккаунту родини на веб-сервісі;
- пошук людей серед користувачів веб-сервісу;
- можливість входу та виходу з сервісу.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблюваний сервіс повинен відповідати наступним функціональним характеристикам:

- дані повинні вводитися користувачем у форму та відправлятися на сервер за допомогою файлу формату JSON;
- сервер повинен обробляти інформацію та віддавати «відповідь» на кожен HTTP запит;
- при виникненні помилок, веб-сервіс повинен обробляти помилки та виводити правильні повідомлення з помилкою;
- повинна бути реалізована можливість додавати інформацію у картку людини: ім’я, прізвище, дати народження та смерті, біографія, тощо;
- повинна бути реалізована можливість додавати зв’язки між людьми у сімейному дереві;
- повинна бути реалізована можливість створення різних аккаунтів користувачів та повинна бути імплементована авторизація та аутентифікація;
- повинна бути реалізована можливість виведення згенерованого дерева в зручний графічний вигляд.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування системи необхідно реалізувати наступні вимоги:

- захист від несанкціонованого доступу до веб-сервісу за допомогою JWT (JSON Web Token) токенів доступу, заснованих на форматі JSON;
- контроль та валідація вхідної інформації при відправці запиту на сервер (перевірка на відповідність типів та на введення обов’язкових полів);
- контроль вихідної інформації, виключення випадків відправки важливої / секретної інформації з серверної частини на клієнтську;
- обробка виняткових ситуацій та виведення повідомлень у разі виникнення помилок.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормального функціонування веб-сервісу необхідно, щоб обчислювальна машина, на якій буде функціонувати веб-орієнтована підсистема, відповідала наступним вимогам:

- чотирьох ядерний процесор з тактовою частотою не менш 2.4 ГГц;
- оперативна пам'ять обсягом не менше 8 GB;
- SSD-накопичувач 120 GB.

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати веб-орієнтована система, відповідало наступним вимогам:

- Операційні системи: Windows (7, 8, 10), Mac OS, Linux.
- Браузери: Chrome, Safari, Mozilla Firefox та інші.
- Фреймворк для автоматизації збирання проектів: Maven (версія 3.2 і вище) або Gradle (версія 4.0 і вище).
- База даних PostgreSQL (версія 13.0 і вище).
- Комплект розробника Java: Java Development Kit (Oracle JDK 11).

Рекомендована швидкість інтернет-з'єднання: від 3 Мбіт / сек.

Основні мови програмування, які були використані: Java, JavaScript та мова запитів SQL.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Результатом даної кваліфікаційної роботи має бути веб-сервіс, а саме REST API, який дасть користувачам доступ до швидкого та зручного створення свого родинного дерева, заповнення інформації про членів родоводу та пошуку родичів.

Основне призначення сервісу:

- надання можливості створення родинного дерева;
- безпечне зберігання даних про родину.

Розроблювана система має дотримуватися наступних функціональних характеристик:

- авторизація і аутентифікація користувача через JWT-токен;
- можливість створення акаунту користувача;
- можливість створення родини та членів родини;
- можливість наповнення ключової інформації про члена родини (дати, біографія, події, тощо);
- можливість редагування та оновлення існуючої інформації;
- можливість збереження фотографій на віддаленому сховищі;
- можливість створення зв'язків між людьми;
- можливість запрошення нових користувачів до сервісу;
- можливість пошуку родичів по базі сервісу;
- можливість перегляду родинного дерева у зручному форматі;
- можливість видалення всіх даних з сервісу.

2.2. Опис застосованих математичних методів

У розробленій системі не використовуються математичні методи.

2.3. Опис використаних технологій та мов програмування

Цей веб-сервіс був розроблений з використанням таких технологій та мов програмування:

- Java;
- Spring Framework;
- Hibernate;
- Apache Maven;
- JPQL;
- SQL;
- JavaScript;
- D3.js.

Java – це строго типізована об'єктно-орієнтована мова програмування загального призначення. Програми Java транслюються в байт-код Java, який виконується віртуальною машиною (JVM) - програмою, що обробляє байтовий код і передає інструкції обладнанню як інтерпретатор. Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина [5].

Мова Java була вибрана для розробки, бо вона використовуватися для створення складних, безпечних, надійних та масштабованих веб-додатків. Аргументами до вибору саме цієї мови програмування:

– Масштабованість (Java легко масштабована для розробки веб-додатків. Це тому, що компоненти широко доступні. При масштабуванні веб-додатка горизонтально або вертикально, мова адаптується до потреб розробників).

– Кросплатформність (Програму на Java доведеться писати один раз, а потім використовувати код де завгодно. Після розробки веб-програми, її можна використовувати на будь-якому пристрої та в будь-якій операційній системі. Таким чином, власникам бізнесу не потрібно турбуватися про зміну обладнання або операційних систем і мати справу з клопотами та витратами на виправлення несумісності існуючого програмного забезпечення).

– Управління пам'яттю (У програмуванні Java майже всі об’єкти «живуть» у heap («купа») пам’яті. «Купа» створюється під час запуску веб-програми і може збільшуватися або зменшуватися під час її виконання. Якщо купа заповнена, то об’єкти, які більше не використовуються, видаляються. Все це відбувається автоматично системою управління пам’яті. Це, у свою чергу, допомагає керувати ефективністю та швидкістю роботи веб-програми).

Spring Framework – це універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Spring забезпечує вирішення багатьох завдань, з якими стикаються Java-розробники, які хочуть створити інформаційну систему, засновану на платформі Java. Spring Framework, ймовірно, найбільш відомий як джерело розширень, потрібних для ефективної розробки складних бізнес-додатків поза великовагових програмних моделей, які історично були домінуючими в розробці [11].

Під час розробки цього веб-сервісу були використані такі модулі фреймворка, як Spring Boot, Spring Web, Spring Data та Spring Security.

Spring Boot – фреймворк , що спрощує створення самостійних додатків на основі Spring, які можна швидко та без клопотів запустити.

Spring Web (MVC) – це веб-фреймворк, що побудований на API сервлетів.

Spring Data – фреймворк, що полегшує використання технологій доступу до даних, реляційних та нереляційних баз даних, хмарних служб даних. Це великий проект, який містить безліч підпроектів, що стосуються баз даних.

Spring Security – фреймворк, та потужна та легко налаштовувана система автентифікації та контролю доступу. Це фактичний стандарт захисту програм, зосереджений на забезпеченні автентифікації та авторизації програм Java.

Hibernate – це популярний фреймворк, мета якого зв'язати ООП і реляційну базу даних. Він дозволяє скоротити обсяги низькорівневого програмування при роботі з реляційними базами даних; може використовуватися як в процесі проектування системи класів і таблиць «з нуля», так і для роботи з уже існуючою базою. Відображення (mapping, зіставлення) Java-класів з таблицями бази даних здійснюється за допомогою конфігураційних XML-файлів або Java-анотацій. При використанні файлу XML Hibernate може генерувати скелет вихідного коду для класів тривалого зберігання. Також забезпечуються можливості по організації відносин між класами «один-до-багатьох» і «багато-до-багатьох». На додаток до управління зв'язками між об'єктами Hibernate також може управляти рефлексивними відносинами, де об'єкт має зв'язок «один-до-багатьох» з іншими екземплярами свого власного типу даних.

Apache Maven – фреймворк для автоматизації збирання проектів на основі опису їх структури в файлах на мові POM (Project Object Model). Maven може управляти збіркою проекту, звітуванням та документацією з центральної частини інформації. Об’єктна модель проекту надає всю конфігурацію для проекту, а загальна конфігурація охоплює залежності від інших проектів. Складні проекти розділяються на кілька модулів або підпроектів, кожен із яких має свій POM. Цими проектами керує кореневий POM, за допомогою якого можна скомпілювати всі модулі однією командою.

JPQL – це незалежна від платформи об'єктно-орієнтована мова запитів, що є частиною специфікації Java Persistence API. JPQL використовується для написання запитів до сутностей, що зберігаються в реляційній базі даних. JPQL багато в чому схожий на SQL, але на відміну від останнього, оперує запитами, складеними стосовно сутностей JPA, на відміну від прямих запитів до таблиць бази даних.

SQL – це мова програмування структурованих запитів (SQL, Structured Query Language), який використовується в якості ефективного способу збереження даних, пошуку їх частин, поновлення, вилучення з бази і видалення. Можливості обробки охоплюють команди визначення уявлень, вказівки прав доступу, схем відносин (в тому числі, їх видалення і зміни) , перевірку цілісності, завдання початку і завершення транзакцій

JavaScript – це повноцінна динамічна мова програмування, яка застосовується до HTML документу, і може забезпечити динамічну інтерактивність на веб-сайтах. Програми на цій мові називаються скриптами. Вони можуть вбудовуватися в HTML і виконуватися автоматично при завантаженні веб-сторінки. Скрипти поширюються і виконуються, як простий текст. Їм не потрібна спеціальна підготовка або компіляція для запуску.

D3.js – це JavaScript-бібліотека для обробки і візуалізації даних з неймовірно величезними можливостями. D3 це набір інструментів для візуалізації даних. Він складається з декількох десятків невеликих модулів, кожен з яких вирішує свою задачу. Крім модулів для побудови різних фігур, всередині D3 є модулі для роботи з елементами на сторінці, завантаженням даних, форматуванням і масштабуванням даних, математичними функціями і іншим [15].

У кваліфікаційній роботі використовується модифікована версія бібліотеки d3.js, яка називається js_family_tree, та базуються на модулі d3.js, який називається d3-hierarchy. Бібліотека d3-hierarchy реалізує алгоритми 2D-макетування для візуалізації ієрархічних даних, а бібліотека js_family_tree розширює її можливості [29]. Це надає можливість використовувати такі нові можливості, як:

– розкладні дочірні вузли;

– представлення вузлів об'єднання для з'єднання батьків та дітей за допомогою загальної точки шляху;

– розбірність вузлів в усіх напрямках: діти, батьки, партнери;

– використання d3-dag замість d3-ієрархії (дозволяє додати двох батьків);

– використання d3-tip, щоб показати метадані при наведенні;

– використання d3-zoom, щоб увімкнути масштабування та панорамування.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Опис архітектурного підходу

В якості основного архітектурного підходу для створення кінцевого програмного продукту був обраний архітектурний стиль (патерн), який називається REST. Створений веб-сервіс являє собою REST API (рис. 2.1.), до якого можуть звертатися абсолютно різні клієнти, такі як мобільний додаток, комп’ютерна програма, front-end частина веб-сайту та інші [13].

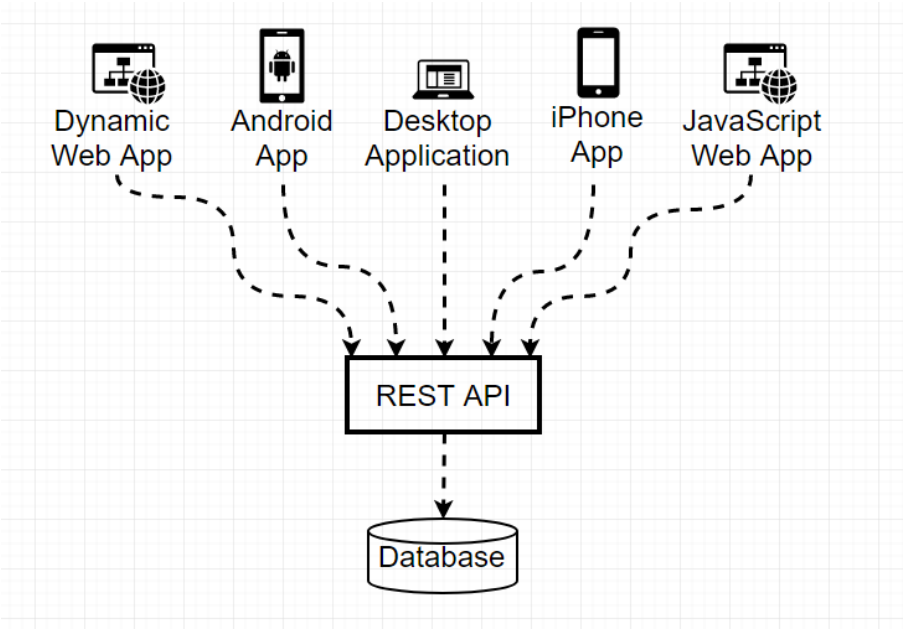

Рис. 2.1. Приклад архітектури REST API

Абревіатура REST розшифровується, як Representational State Transfer, що можна перекласти з англійської мови, як «передача репрезентативного стану».

Дані повинні передаватися між сервером та клієнтом у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). У цій кваліфікаційній роботі використовується файл формату JSON, бо це найпопулярніше розширення фалу для передачі інформації по мережі Інтернет на час виконання кваліфікаційної роботи. JSON – це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті та може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

Приклад JSON об'єкту, який передається через мережу (рис. 2.2.):

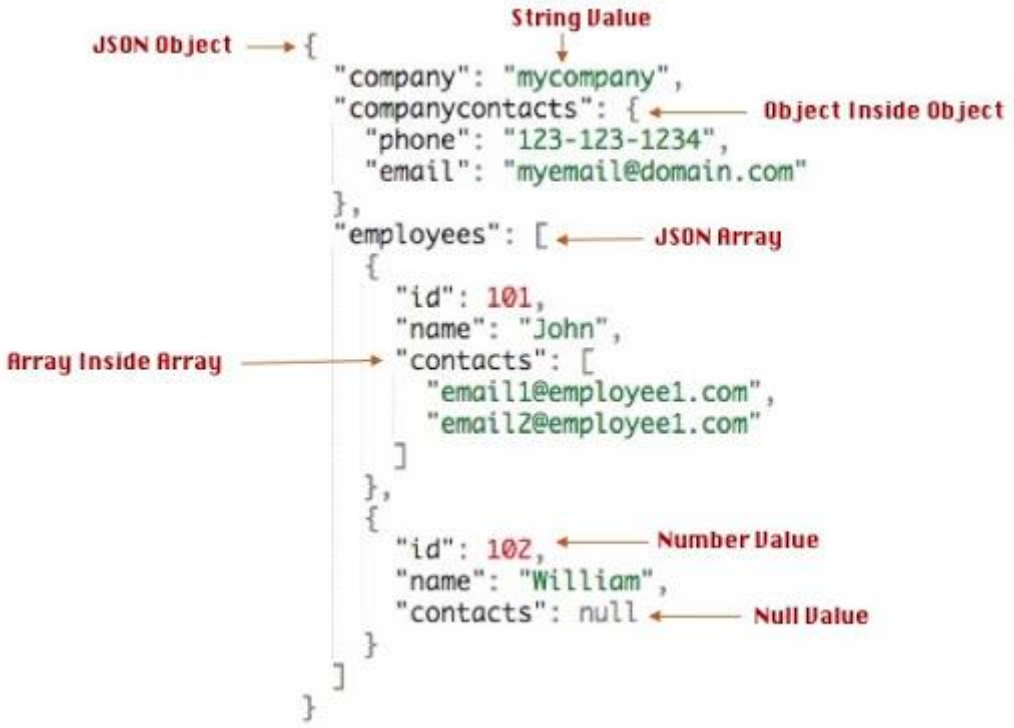

Рис. 2.2. Приклад JSON об'єкту

Для того, щоб веб-сервіс відповідав REST дизайну, потрібно, щоб були виконані наступні правила:

– Клієнт-серверна архітектура. Сервер - це комп'ютер, який має необхідні ресурси, а клієнт - це комп'ютер, якому потрібно взаємодіяти з ресурсами, що зберігаються на сервері;

– Використання явних методів HTTP. Щоб створити ресурс на сервері, потрібно використовувати метод POST. Для отримання ресурсу, використовувати GET. Щоб поміняти стан ресурсу або оновити його, використовувати PUT. Щоб видалити ресурс, використовувати DELETE;

– Відсутність стану. Поняття «без стану» не означає, що сервери і клієнти його не мають, у них просто немає необхідності відстежувати стан один одного. Коли клієнт не взаємодіє з сервером, сервер не має уявлення про його існування. Сервер також не веде облік минулих запитів. Кожен запит розглядається як самостійний;

– Одноманітність інтерфейсу. Обмеження гарантує, що між серверами і клієнтами існує спільна мову, яка дозволяє кожній частині бути замінюваною або змінною, без порушення цілісності системи. Це досягається через 4 субобмеження: ідентифікацію ресурсів, маніпуляцію ресурсами через уявлення, "самодостатні" повідомлення і гіпермедіа;

– Передача даних через JSON.

Розроблений веб-сервіс повністю відповідає обраному архітектурному стилю, тому можна сказати, що він являє собою RESTful API.

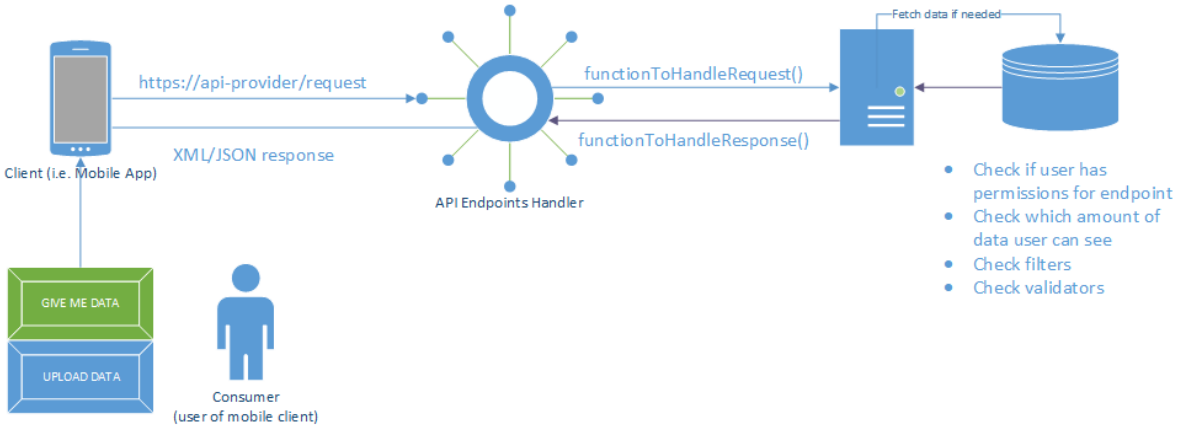

Рис. 2.3. Приклад RESTful API

Ще один патерн архітектурного проектування, який був задіяний в розробці, має назву Layered Architecture (з англ. багаторівнева архітектура). Це кієнт-серверна архітектура, в якій розділяються функції представлення, обробки і зберігання даних. Найбільш поширеною різновидом багаторівневої архітектури є трирівнева архітектура, яка і використовується в кваліфікаційної роботі. N-рівнева архітектура надає модель, по якій розробники можуть створювати гнучкі і повторно-використовувані програми. Поділяючи додаток на рівні абстракції, розробники набувають можливість внесення змін в якийсь певний шар, замість того, щоб переробляти весь додаток цілком [6].

Трирівнева архітектура (рис. 2.4.) забезпечує, як правило, більшу масштабованість (за рахунок горизонтальної масштабованості сервера додатків і мультиплексування з'єднань), більшу конфігурованість (за рахунок ізольованості рівнів один від одного).

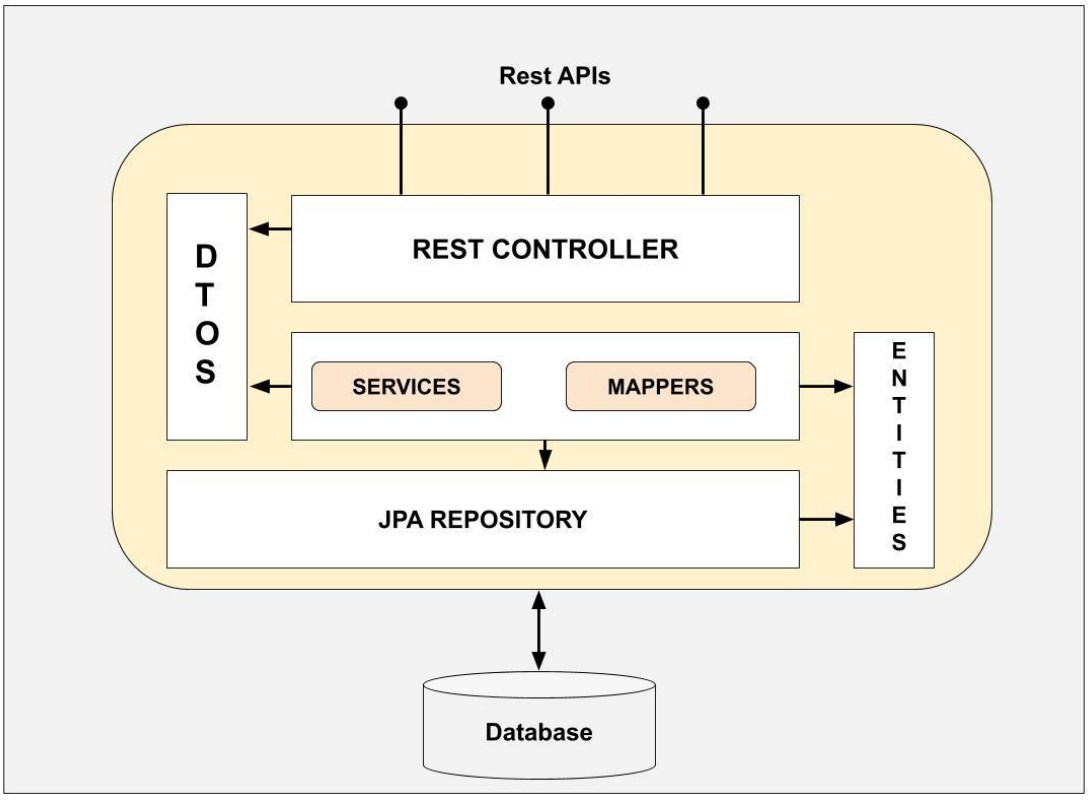

Рис. 2.4. Архітектура, що використовується у розроблювальному сервісі

Архітектура поділяється на три такі рівні:

– Репозиторій. Це шлюз між доменом та бізнес-рівнем, є шаром, який здійснює доступ до бази даних та виконує операції. Отже репозиторій є абстракцією для доступу до бази даних.

– Сервіс. Це проміжний шлюз між контролером та репозиторієм. Сервіс забирає дані з контролера, виконує перевірку та логіку, викликає сховища для маніпулювання даними.

– Контролер. Це шлюз, який приймає вхідні дані й перетворює їх на команди для подальшої роботи, також видає вихідні дані.

2.4.2. Опис структури бази даних

Перед створенням схеми бази даних для кваліфікаційної роботи з'явилася необхідність вибору типу бази даних. Спочатку для проектування був обраний графовий тип бази даних, та саме база даних Neo4j.

Графова база даних — це база даних, яка використовує структури графів для семантичних запитів з вершинами, ребрами та властивостями для представлення та зберігання даних (рис. 2.5.). Ключовим поняттям системи є «граф» (або «ребро» або «відносини»), який безпосередньо пов'язує елементи даних у сховищі. Відносини дозволяють безпосередньо об'єднувати дані у сховищі, а в багатьох випадках, вибрати однією операцією [27].

Спочатку цей тип бази даних був вибраний через те, що він явно відображає зв'язки між типами даних і не має обмежень в типах зав’язків. Також цей тип бази даних ідеально підходив для потреб кваліфікаційної роботи, як інструмент відображення готово родинного дерева. Вибрана база Neo4j, дані зберігає у власному форматі, спеціалізовано пристосованому для подання графової інформації, такий підхід в порівнянні з моделюванням графової бази даних

засобами реляційної СУБД дозволяє застосовувати додаткову оптимізацію в разі даних з більш складною структурою.

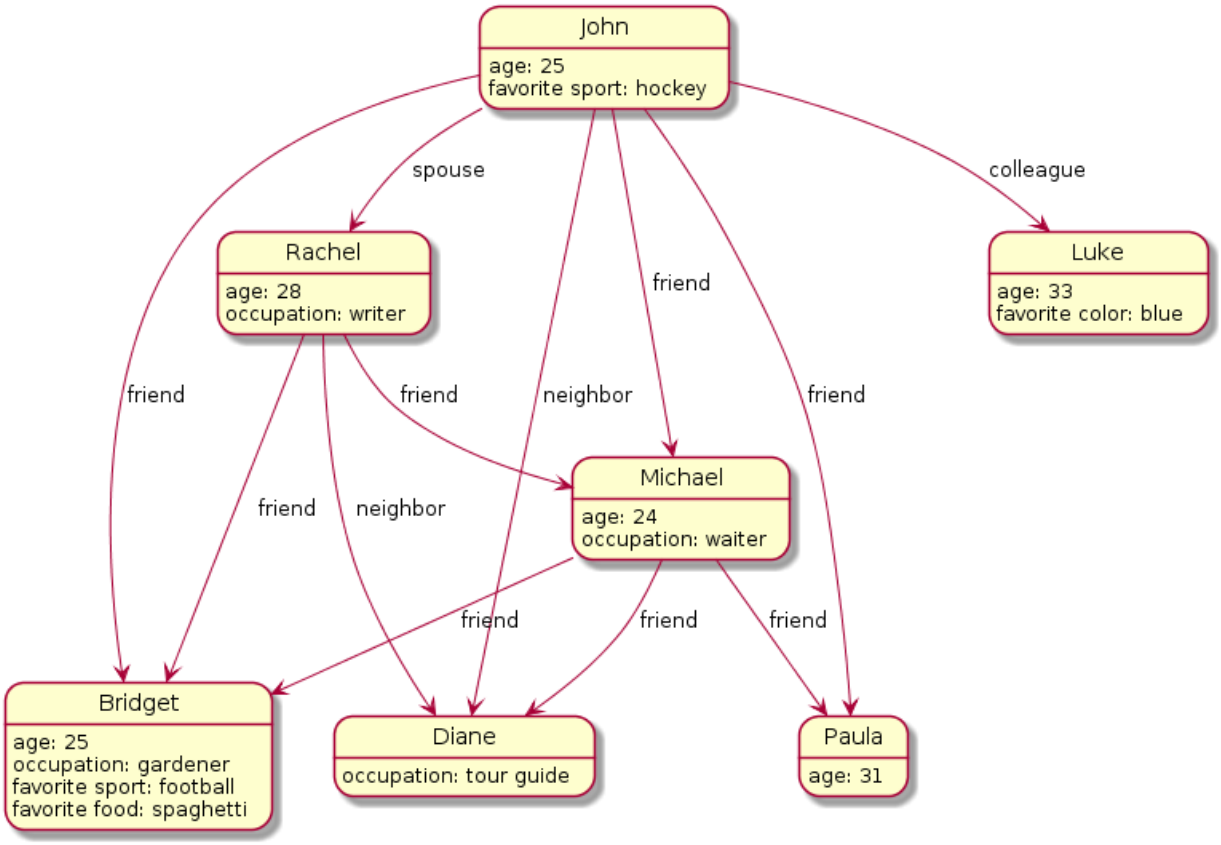

Рис. 2.5. Приклад відображення графової бази даних

Так як і в побудові сімейних дерев, так і в побудові графової бази даних використовується зв’язок між людьми, то на початку проектування була обрана і реалізована наступна структура бази даних. Вона реалізована через "вузли" (наприклад, людина, подія або сім'я і т.д.) і "відносини" (наприклад, батько, мати, місце народження і т.д.). Приклад спочатку спроектованої структури (рис. 2.6.).

Але після роботи з спроектованою базою даних було виявлено серйозні проблеми, які впливали на роботу веб-сервісу. По-перше, навіть при не дуже великий наповненості БД, відповіді на запити, що посилаються в БД, приходили дуже довго. Простий запит на отримання і її зв'язки займав близько 2-4 секунд, що неприпустимо довго для веб-сервісу, тим паче при низькій завантаженості БД. На момент тестування в базі знаходилося близько 20 "вузлів" типу "людина" і близько 15 "відносин", що є мінімальною кількістю в родинному дереві. Навіть

при такій кількості були проблеми з продуктивністю. По-друге, Neo4j-OGM (бібліотека зіставлення об’єктів Java для бази Neo4j) яка використовувалася на тестовій фазі мала багато недоліків і помилок в роботі. Багато відповідей на запити в базу даних відображалися некоректно.

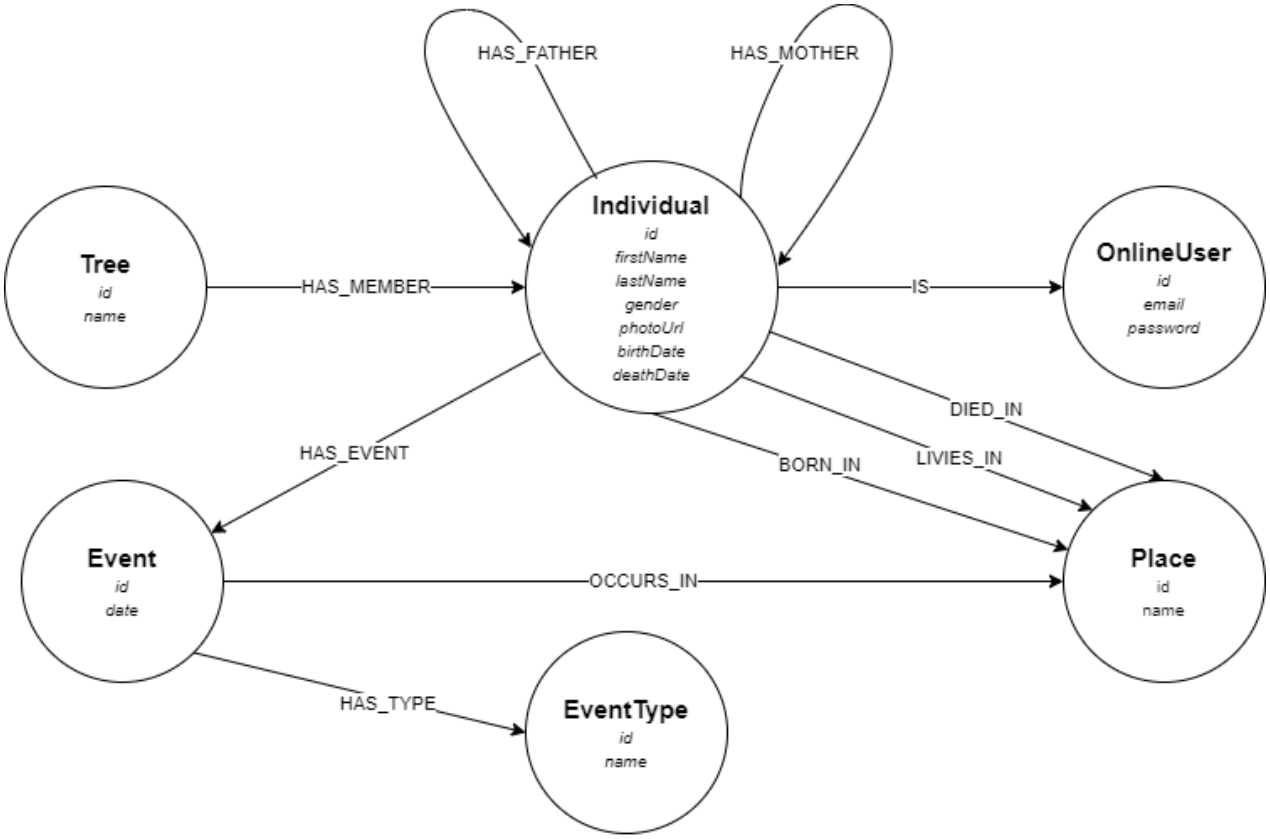

Рис. 2.6. Перший прототип графової бази даних для кваліфікаційної роботи

У результаті довелося дійти до рішення відмовитися від використання графової бази даних і пожертвувати можливістю зручного відображення графіків. Після дослідження ринку популярних рішень, для кваліфікаційної роботи була обрана реляційна база даних PostgreSQL, яка показувала відмінні результати в швидкості відповіді на запити.

Так само безумовним плюсом бази є підтримка типу даних UUID (англ. Universally unique identifier «універсальний унікальний ідентифікатор»), який є стандартом ідентифікації, який використовується в створенні програмного забезпечення. Основне призначення UUID — дозволити розподіленим системам

унікально ідентифікувати інформацію без центру координації [20]. Таким чином, кожен може створити UUID і використовувати його для ідентифікації чого-небудь із достатнім рівнем впевненості, що даний ідентифікатор не буде ненавмисно використано для чогось іншого. Тому інформацію, відмічену за допомогою UUID, можна пізніше додати до загальної бази даних без необхідності вирішення конфлікту імен, що є дуже зручним при величезній кількості людей і подій в базі даних родинних дерев.

Центральним місцем бази є таблиця individual, яка уособлює члена сімейного дерева і має такі стовпці, як first_name, last_name, birth_date, death_date і інші. Кожен член є учасником якоїсь сім'ї, тому в таблце individual є зовнішній ключ family_id на таблицю family. Таблиця family містить стовпці family_name і family_description, family_photo, а також зовнішній ключ на таблицю individual. У таблиці individual так само зберігається зовнішній ключ online_user_id на таблицю online_user. Таблиця online_user містить інформацію для входу користувача на сервіс, а саме стовпці email, password і інші. Так само в таблиці individual зберігаються зовнішні ключі на таблицю place, яка відповідає за зберігання різних місць, таких як місце народження, місце смерті та інших. Останнім зовнішнім ключем в таблиці individaul є ключ gender_id, який посилається на таблицю gender, в якій зберігяються гендери.

Наступною важливою таблицею бази є таблиця event, яка зберігає різні події з життя учасників сімейного дерева. У цій таблиці міститися стовпчики event_date і event_description, які характеризують дату і опис події відповідно. Так само в таблиці є три зовнішніх ключа на інші таблиці. Ключ і стовпець individual_id відповідає за прив'язку подій до певно ї людині і посилається на таблицю individual, ключ і стовпець place_id, посилається на табілцю place і описує місце проведення даної події. Так само в таблці зберігається стовпець і зовнішній ключ на таблицю event_type, в якій зберігаються типи подій (такі як, весілля, випускний і так далі).

Останньою зв'язуючою таблицею є таблиця marriage, яка зберігає дані про шлюби людей. Таблиця має стовпці person_id і spouce_id, які уособлюють двох людей у шлюбі. Між таблицями marriage і individual є дві проміжні таблиці. Одна з них - marriage_children, яка зберігає посилання на дитину, яка була народжена в певному шлюбі. Інша - individual_marriages, зберігає інформацію про шлюби певної людини. Так само абсолютно кожна таблиця має стовпець id - щоб мати унікальний ідентифікатор для кожного запису і стовпці creation_date і last_modified_date, за якими можна зрозуміти, коли був створений запис і коли він останній раз змінювався.

Дана схема бази даних (рис. 2.7.) дозволяє реалізувати багато різних варіантів шлюбів, як і буває в реальному житті і дозволяє зберігати повну інформацію про людей. Багато проміжних і додаткових таблиць дозволяють утримувати дані структуровано і уникати дублікатів. База даних зберігається на віддалених хмарних серверах Amazon, де використовується безкоштовний тарифний план, що дозволяє не прив'язуватися до локальної бази даних на комп'ютері розробника.

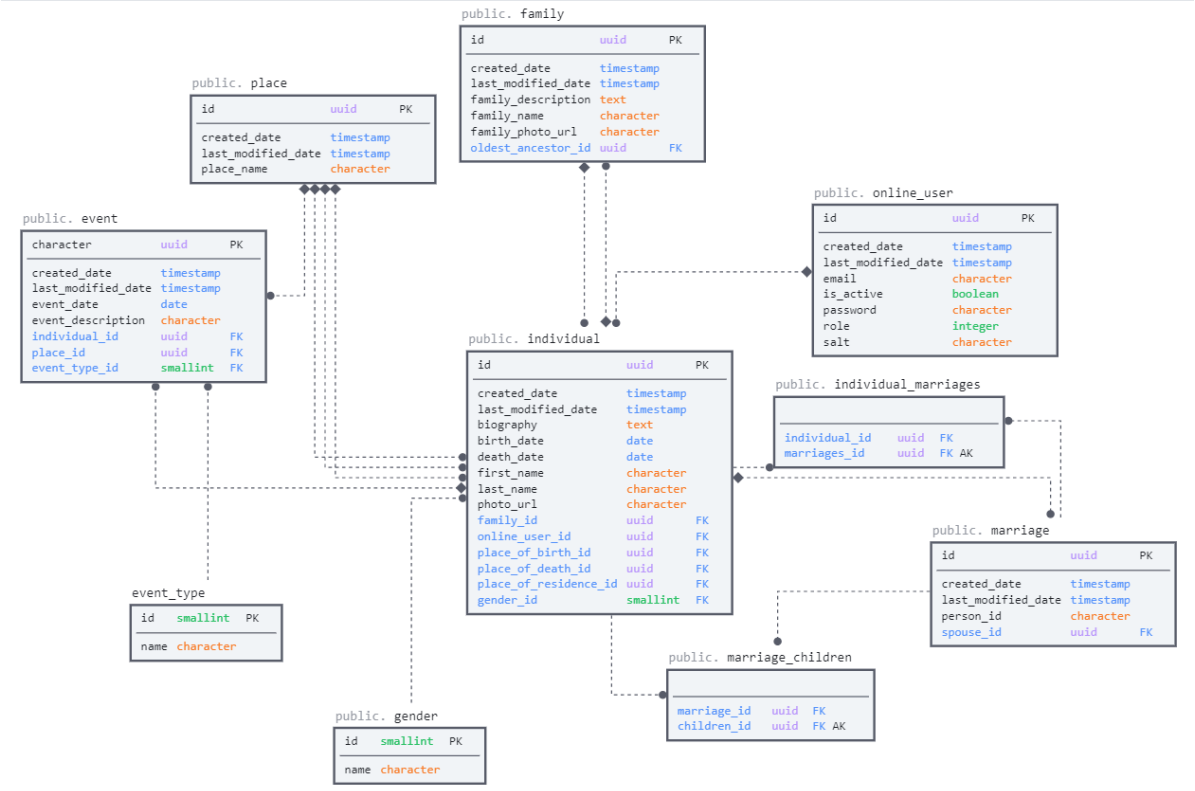

Рис. 2.7. Схема бази даних кваліфікаційної роботи

2.4.3. Короткий опис структури проекту

Так як у проекті використовується багаторівнева архітектура, сам розроблюваний проект теж є багатомодульним. Збирач проектів Maven дозволяє збирати проект з декількох модулів. Кожен програмний модуль включає свій проектний файл pom.xml, який описує всі залежності між модулями і встановлює зв'язок з бібліотеками. Один з проектних pom.xml файлів є кореневим і дозволяє об'єднати всі модулі в єдиний проект. Багатомодульний проект найпростіше уявити собі як дерево - у нього є спільне коріння, яке нічого не робить, а лише описує загальні параметри, і листя, які успадковують ці загальні параметри. Модулі розподіляються наступним чином:

- spadok-web. Цей підпроект відповідає за обробку запитів клієнта, передачу даних в сервісний шар, передачу даних назад клієнту і безпеку системи.
- spadok-service. Цей підпроект відповідає за логіку роботи веб-сервісу, містить класи-сервіси, які обробляють інформацію, відповідають за валідацію і конвертацію даних.
- spadok-persistance. Цей підпроект відповідає за опис доменних класів, які є базовими в даній розроблюваній системі.
- spadok-frontend. Це окремий підпроект, який відповідає за генерацію родинного дерева за виконує роль клієнта.

2.4.4. Детальний опис модулів розроблюваної системи

У модулі spadok-web знаходиться чотири пекеджа (папки): config, controller, exeptions та security. У папці config знаходяться класи AmazonConfig, SecurityConfig та SwaggerConfig, кожен з них призначений для конфігурації того чи іншого сервісу. AmazonConfig – це клас, в якому відбувається з'єднання веб-

сервісу з онлайн веб-службою Amazon S3 за рахунок двох секретних ключів. SecurityConfig – це клас, в якому настроюється безпека веб-сервісу, в тому числі закриття або відкриття ендпоінтів (це саме звернення до маршруту окремим HTTP методом). SwaggerConfig – це клас, в якому настроюється базова інформація та подання того, як буде виглядати інтерфейс користувача. У папці controller знаходяться класи, які відповідальні за отримання даних від клієнта і передачу цих дані всередину сервісу, а так само за віддачу відповіді на будь-який запит. У папці exeptions знаходяться класи, в яких описано те, як повинна працювати обробка винятків, які поля повинні віддаватися клієнту при помилці, а так само які людсько-зрозумілі помилки повинен бачити клієнт. Так само, як приклад, в класі CustomRestExceptionHandler описується те, при яких помилках в системі, які повідомлення повинні відправитися клієнту, це виключає випадки, що клієнт отримає нерелевантні інформацію про помилку. У папці security знаходяться один клас, метою якого є отримання заголовків HTTP методу для аутентифікації.

У модулі spadok-service знаходиться сім пакетів, які відповідають за бізнес-логіку сервісу і є ключовою ланкою роботи. В папці validation знаходяться класи, в яких описана логіка перевірки існуючого імейлу і отриманого jwt токена. В папці annotations знаходяться класи, які використовуються як індивідуальні анотації, які завдання, що ставляться над іншими класами і спрощують роботу. В папці constants знаходиться клас, в якому зберігаються константи, які використовуються в різних модулях системи. В папці dto знаходяться класи, які являють собою об'єкти на стороні сервера, які зберігають певні дані і використовуються для безпечної передачі цих дані клієнта [18]. В папці converter знаходяться класи, які конвертують доменні об'єкти додатка або сутності з бази даних в об'єкти передачі даних, які передаються на шар подання клієнту. В папці security знаходяться класи, які описують всю логіку створення та підтвердження jwt токенов, які використовуються для надання безпечного доступу до веб-сервісу. В папці service знаходяться класи, які описують саму основну бізнес-логіку додатка і відповідають за правильну обробку даних, правильне

збереження і передачу даних між модулями. Є свого роду серцем системи, де відбуваються вся обробка.

У модулі spadok-persistance знаходиться два пакети, domain та repository. В папці domain знаходяться класи, які описують доменні об'єкти системи, над якими проводяться різні операції, такі як збереження, зміна і видалення. Ці класи безпосередньо є об'єктним уявленням таблиць з реляційної бази даних. В папці repository знаходяться класи, мета яких надати зручний інтерфейс для з'єднання системи з базою даних, класи відповідають за створення і виконання індивідуальних і стандартних запитів до бази даних.

У модулі spadok-frontend знаходиться знаходяться html і js файли, які використовуються для генерування сторінки з родинним деревом.

2.4.5. Опис можливості збереження зображень

У кваліфікаційній роботі є можливість збереження зображень (для збереження фото людей з родинного дерева) на віддаленому хмарному сховищі. Після аналізу ринку доступних хмарних сховищ було вибрано сховище S3 від компанії Amazon. Сховище надає можливість зберігання та отримання будь-якого обсягу даних в будь-який час з будь-якої точки мережі. Це веб-служба є комерційною публічною хмарою Amazon Web Services, яка надає передплатникам послуги як по інфраструктурної моделі (віртуальні сервери, ресурси зберігання), так і платформного рівня (хмарні бази даних, хмарне програмне забезпечення, хмарні обчислення, засоби розробки). Amazon S3 пропонує прості у використанні інструменти адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно до потреб. Важливим фактором при виборі хмарного сховища також була ціна за послуги зберігання, так як розробка системи ведеться на некомерційній основі. Безкоштовний доступ Amazon S3 повністю відповідає потребам у віддаленій базі

для веб-сервісу, надаючи в місяць 5 гігабайт для зберігання файлів, 20000 запитів на отримання і 2000 запитів на оновлення даних.

Файли в цьому сервісі зберігаються в так званих buckets ("бакетах") – це аналог папки з доступом по шифру і ключу. Робота з бакетом відбувається через S3 Object Lambda, який використовується в AWS SDK для мови Java [24]. За допомогою функції S3 Object Lambda можна додавати власний код в запити S3 GET, щоб змінювати і обробляти дані, які повертаються в додаток. Можна застосувати для користувача код для зміни даних, що повертаються стандартними запитами S3 GET, для фільтрування рядків, динамічної зміни розміру зображень, видалення конфіденційних даних і багато чого іншого (рис. 2.8.). Виконання кодів на базі функцій AWS Lambda здійснюється в інфраструктурі, повністю керованої AWS, що дає можливість виключити необхідність створення і зберігання похідних копій даних або запуску дорогих проксі, при цьому без яких би то не було змін в додатках.

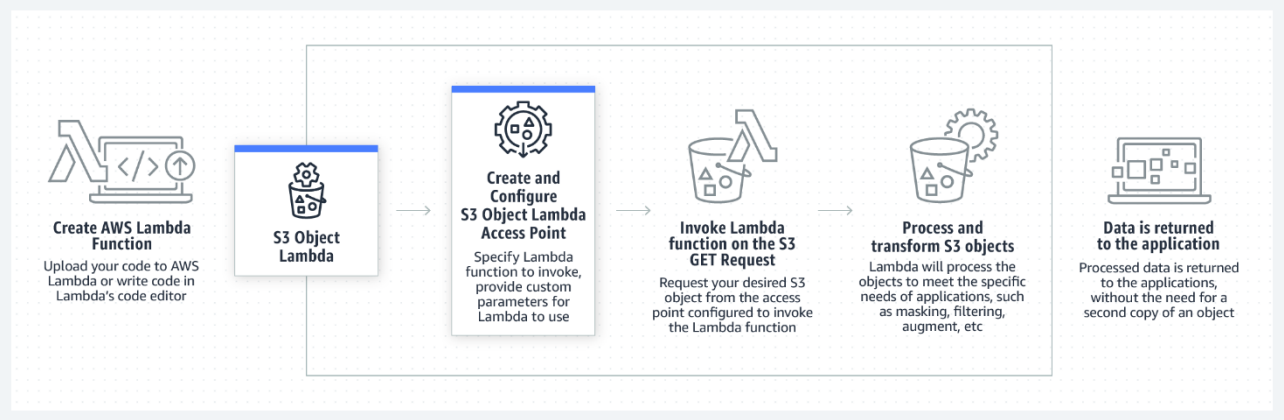

Рис. 2.8. Схема роботи з бакетом AWS

Так як всі файли зберігаються в Amazon S3, при додавання зображення через веб-сервіс, у віддалену базу даних веб-сервісу записується тільки ідентифікатор створеного файлу, за яким можна буде знайти файл в сховищі Amazon, що істотно зменшує навантаження на основну базу даних веб-сервісу.

2.4.6. Опис аутентифікації користувача у веб-сервісі

Для того, щоб отримати доступ до певних точок входу в веб-сервіс, потрібно посилати запити з клієнта з певним рядком, в якому зашитий токен для входу. Цей токен називається JWT токен. JSON Web Token (JWT) - це відкритий стандарт для створення токенів доступу, заснований на форматі JSON. Як правило, використовується для передачі даних для аутентифікації в клієнт-серверних додатках. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який в подальшому використовує даний токен для підтвердження своєї особи [14].

У найпростішому розумінні – це рядок в спеціальному форматі, яка містить дані, наприклад, ID і ім'я зареєстрованого користувача. Вона передається при кожному запиті на сервер, коли необхідно ідентифікувати і зрозуміти, хто надіслав цей запит.

JWT складається з трьох частин (рис. 2.9.): заголовок header, корисні дані payload та підпис signature.

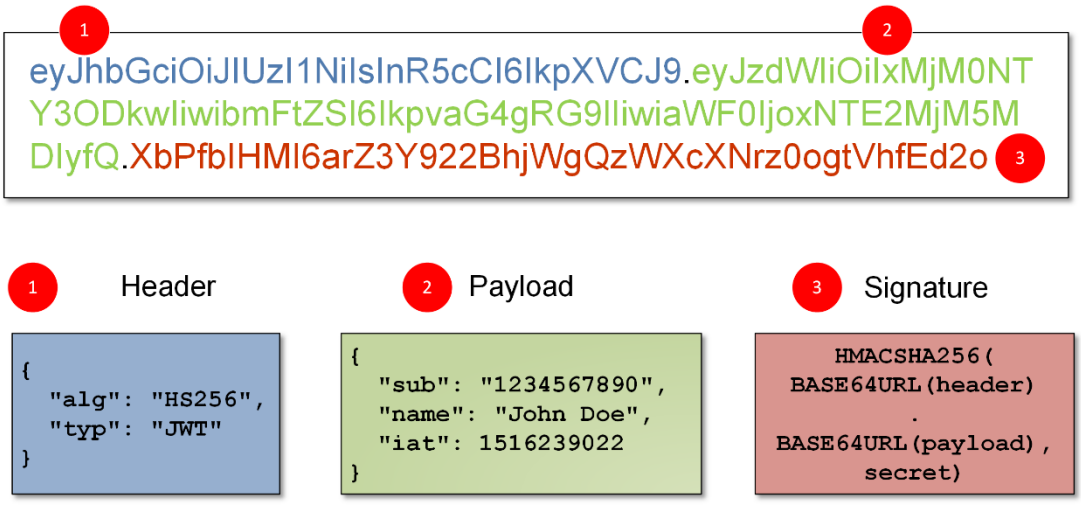

Рис. 2.9. Приклад структури JWT токена

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 – це перша частина токена – заголовок. Вона закодована в Base64 і якщо її розкодувати, отримаємо об'єкт,

який містить ключ typ - це тип токена, в даному випадку значення - JWT, і ключ alg - алгоритм шифрування, в даному випадку - HSH256. HS256 - не що інше, як HMAC-SHA256, для його обчислення потрібен лише один секретний ключ.

eyJ1c2VyX2lkIjoxLCJleHAiOjE1ODEzNTcwMzl9 – це друга частина токена – PAYLOAD або корисні дані. Ці дані також називають JWT-claims (заявки). В наведеному вище прикладі в payload передається ім'я користувача, а в кваліфікаційної роботі передається email користувача.

E4FNMef6tkjIsf7paNrWZnB88c3WyIfjONzAeEd4wF0 – це остання частина токена, найбільш важлива, сигнатура. Як можна було помітити, перші дані передаються практично у відкритому вигляді і розкодувати їх може будь-хто. Але шифрувати їх немає необхідності. Мета токена - підтвердити, що ці дані не були змінені. Ось для цих цілей і виступає сигнатура. І щоб її згенерувати потрібний приватний ключ або якась секретна фраза, яка знаходиться тільки на сервері. Тільки за допомогою цього ключа ми можемо створити сигнатуру і перевірити, що вона була створена саме за допомогою його.

Принцип перевірки токена такий. У прикладі використовується JWT, який підписаний за допомогою HS256 алгоритму і тільки сервер аутентифікації і сервер додатка знають секретний ключ. Сервер програми отримує секретний ключ від сервера аутентифікації під час установки аутентифікаційних процесів. Оскільки додаток знає секретний ключ, коли користувач робить API-запит з доданим до нього токеном, додаток може виконати той же алгоритм підписування до JWT. Додаток може потім перевірити цей підпис, порівнюючи її зі своєю власною, обчисленої хешуванням. Якщо підписи збігаються, значить JWT валідний, тобто прийшов від перевіреного джерела. Якщо підписи не збігаються, значить щось пішло не так - можливо, це є ознакою потенційної атаки. Таким чином, перевіряючи JWT, додаток додає довірчий шар (a layer of trust) між собою і користувачем.

Алгоритм роботи виглядає так (рис. 2.10.):

1. Спершу користувач заходить на сервер аутентифікації за допомогою ключа (це може бути пара логін / пароль).
2. Потім сервер аутентифікації створює JWT і відправляє його користувачеві.
3. Коли користувач робить запит до API сервісу, він додає до нього отриманий раніше JWT.
4. Коли користувач робить API запит, сервіс може перевірити за запитом JWT, чи є користувач тим, за кого себе видає.

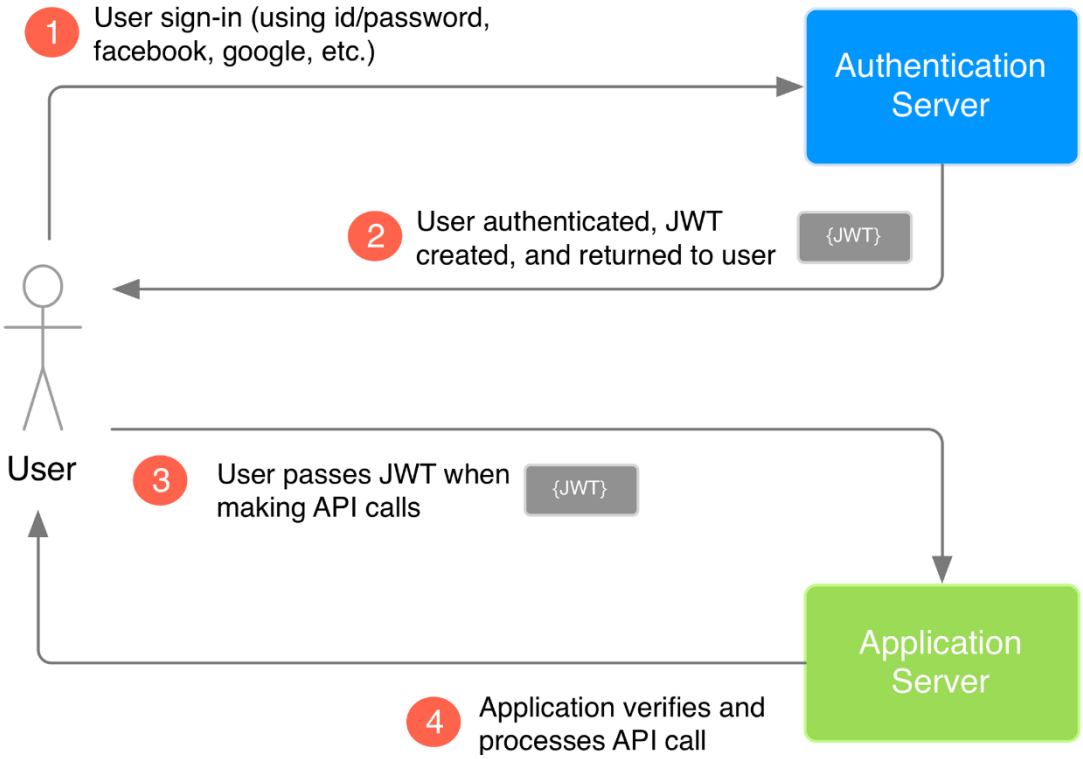

Рис. 2.10. Приклад алгоритму роботи

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані приходять на сервер від клієнта в форматі JSON файлу, в якому міститися об'єкти з парами ключ і значення. Відправлені дані проходять перевірку перед тим, як потрапити на сервер і почати використовуватися в логіці

сервісу. Проходить перевірка на правильність введених даних для певних полів. Наприклад, правильність написання формату email'у або правильність написання дати народження людини. Вихідні дані також відправляються назад в форматі JSON.

В роботі вхідних і вихідних даних використовується шаблон проектування DTO (Data Transfer Object), він використовується для передачі даних між підсистемами. Вони поділяються на ті, які використовуються при запиті (Request) і на ті, що повертатися в якості відповіді сервера (Response). DTO допомагає приховати так звані БО (бізнес-об'єкти). БО – це об'єкти, що містить дані і методи, які виконують операції над цим об'єктом. Коли БО виставляється на верхній рівень, він може викликати відкриті методи об'єкта. Цього треба уникати. DTO не повинен транспортувати всі дані БО. При правильному застосуванні цього шаблону проектування, доцільно створювати DTO для кожної операції, яка стосується вхідних і вихідних даних. Тобто якщо об'єкт має поля аудиту, такі як CreatedBy (ким створений) або CreatedDate (дата створення), вони не повинні виходити за шар бізнес логіки і не повинні повертатися клієнту на запити. Така ж ситуація і, наприклад, при оновленні об'єкта, запит з вхідними даними не повинен містити поля аудиту.

2.6. Опис розробленої системи

Розроблений програмний продукт являє собою REST API веб-сервіс для створення сімейних дерев і вивчення родоводу, який задовольняє всім поставленим задачам. Розроблений веб-сервіс надає можливість створення акаунтів користувачів, створювати сім'ї, створювати членів сімейного дерева, наповнювати даними профілі людей, запрошувати людей приєднатися до сімейного дерева, створювати зрозумілі і інформативні сімейні дерева.

2.6.1. Використані технічні засоби

Для правильного функціонування розробленої системи, потрібна ЕОМ з певними характеристиками, а саме потрібен сервер, на якому буде запущений даний проект. На даний момент веб-сервіс запущений на віддаленому сервері з наступними мінімальними характеристиками:

- ЦП [CPU]: Чотирьохядерний процесор.
- Накопичувач [HDD]: 500 МБ.
- Оперативна пам’ять [RAM]: 512 МБ.

Безпосередньо розробка і тестування велося на ЕОМ з наступними технічними характеристиками:

- ЦП [CPU]: Inter Core i5-8770.
- Відеопам'ять [VRAM]: 8 ГБ.
- Накопичувач [SDD]: 120 ГБ.
- Оперативна пам'ять [RAM]: 16 ГБ.

2.6.2. Використані програмні засоби

Під час розробки даної кваліфікаційної роботи були використані такі програмні засоби, за допомогою яких забезпечується функціонування розробленої системи:

- IntelliJ IDEA.
- Git.
- GitLab.
- Heroku.

– Amazon Web Services.

IntelliJ IDEA – це комерційне інтегроване середовище розробки для різних мов програмування від компанії JetBrains (рис. 2.11.). Починаючи з версії 9.0, середовище доступно в двох редакціях: Community Edition і Ultimate Edition. Community Edition є повністю вільною версією, доступною під ліцензією Apache 2.0, в ній реалізована повна підтримка Java SE, Kotlin, Groovy, Scala, а також інтеграція з найбільш популярними системами управління версіями. В редакції Ultimate Edition, доступною під комерційною ліцензією, реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду, а також підтримка інших систем управління версіями, мов та фреймворків. У розробці була використана версія Ultimate Edition, так як в цій версії є підтримка фреймворків Spring і Hibernate. Покращена версія середовища розробки була отримана безкоштовно по студентській підписці.

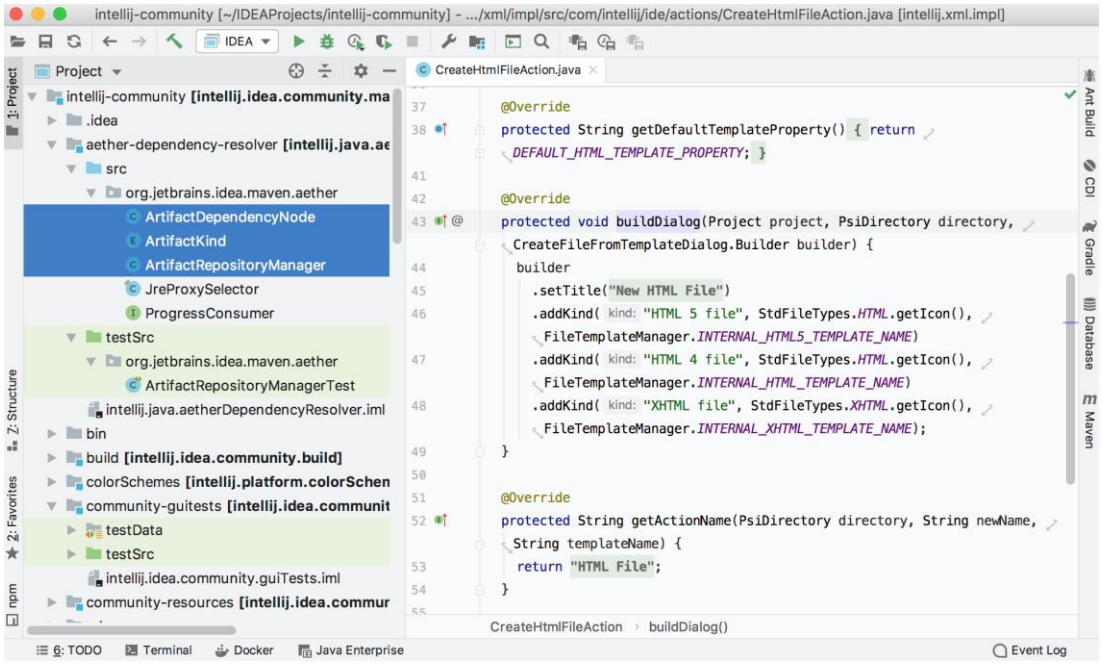

Рис. 2.11. Приклад середовища розробки

Git – це абсолютний лідер за популярністю серед сучасних систем управління версіями. Це розвинений проект з активною підтримкою і відкритим вихідним кодом. Git застосовується для управління версіями в рамках

колосальної кількості проектів з розробки ПЗ, як комерційних, так і з відкритим вихідним кодом. Система використовується безліччю професійних розробників програмного забезпечення. Вона чудово працює під управлінням різних операційних систем і може застосовуватися з безліччю інтегрованих середовищ розробки (IDE). Це система управління версіями з розподіленою архітектурою, в Git кожна робоча копія коду сама по собі є репозиторієм. Це дозволяє всім розробникам зберігати історію змін в повному обсязі. Розробка в Git орієнтована на забезпечення високої продуктивності, безпеки і гнучкості розподіленої системи. А Gitflow Workflow – це модель робочого процесу Git. Gitflow Workflow передбачає вибудовування суворої моделі розгалуження з урахуванням випуску проекту. Така модель забезпечує надійну основу для управління великими проектами (рис. 2.12.).

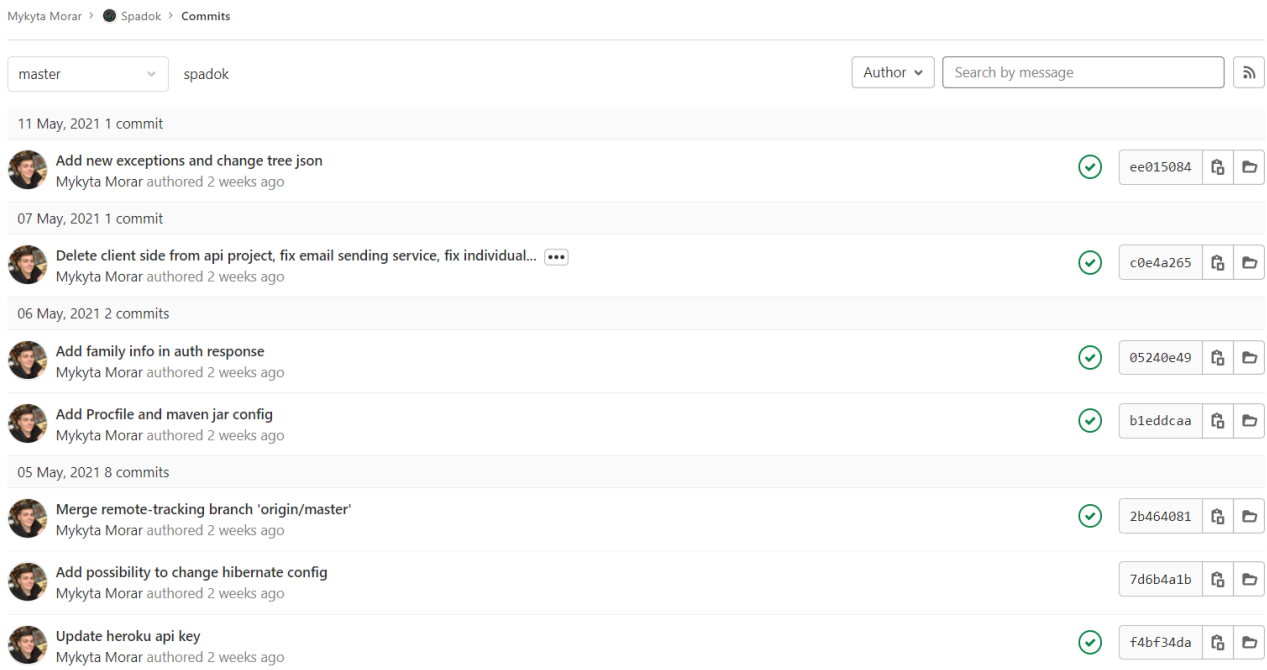

Рис. 2.12. Приклад коммів Git Flow

GitLab – це веб-додаток і система управління репозиторіями програмного коду для Git. GitLab пропонує рішення для зберігання коду та спільної розробки масштабних програмних проектів. Репозиторій (рис. 2.13.) включає в себе систему контролю версій для розміщення різних ланцюжків розробки та гілок,

дозволяючи розробникам перевіряти код і повератися до стабільної версії ПО в разі непередбачених проблем.

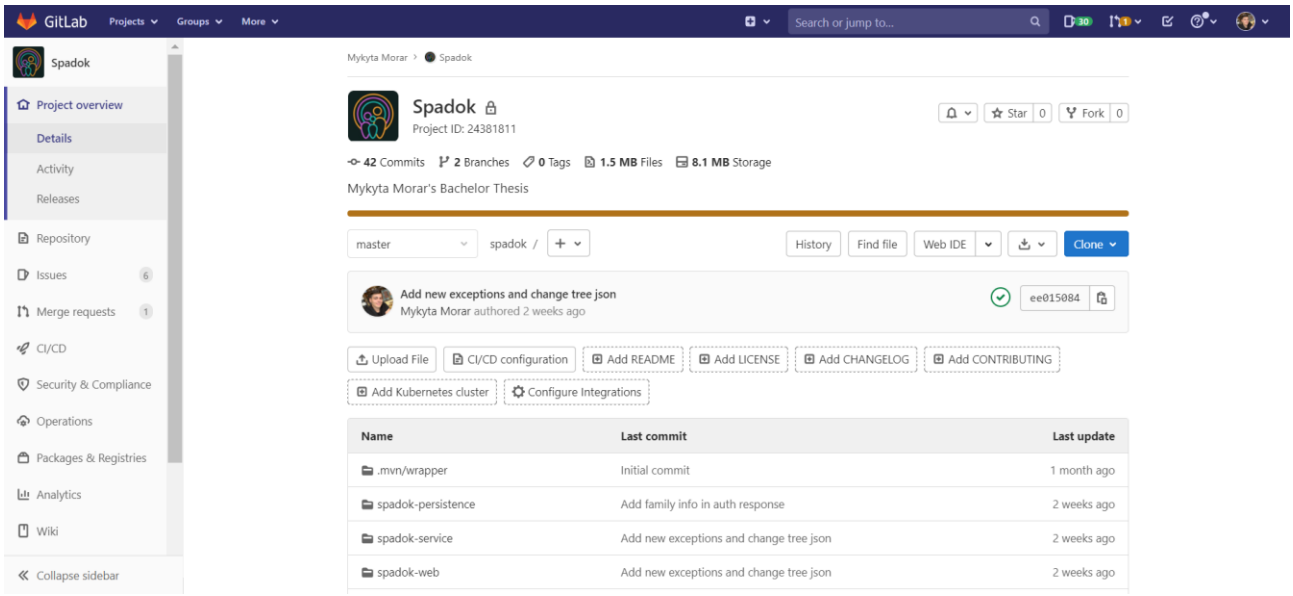

Рис. 2.13. Приклад проекту на GitLab

Heroku – це хмарна PaaS-платформа, що підтримує ряд мов програмування. Platform as a Service (PaaS, «платформа як послуга») – модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ: операційних систем, систем управління базами даних, сполучній програмного забезпечення, засобів розробки і тестування, розміщеним у провайдера. Використовується для запуску веб-сервісу на віддалених серверах (рис. 2.14.).

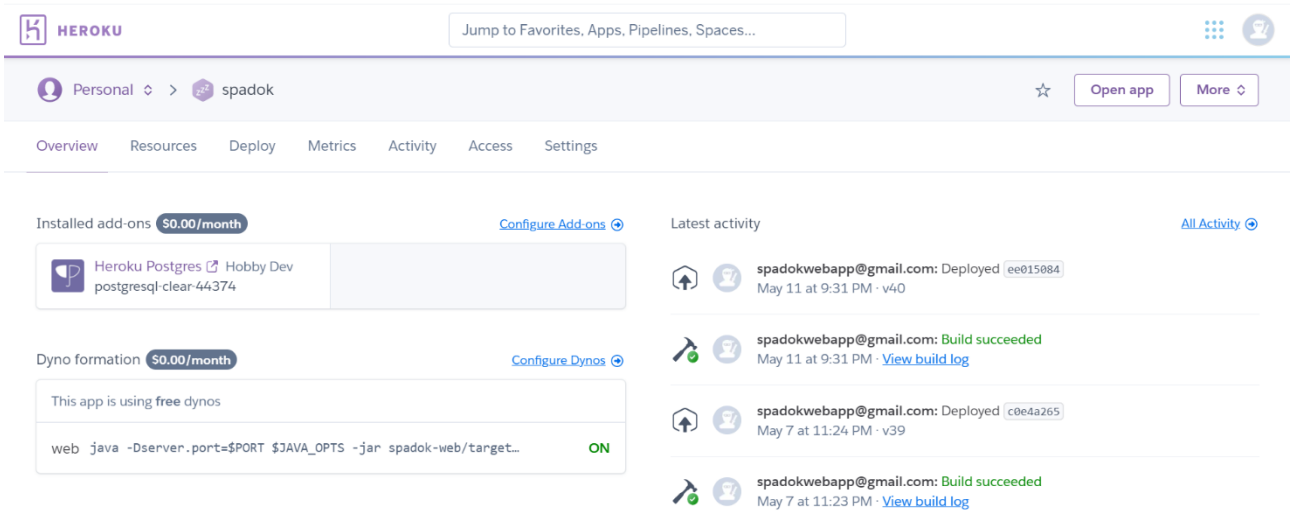

Рис. 2.14. Приклад проекту розташованого на Heroku

Amazon Web Services (AWS) - це найпоширеніша в світі хмарна платформа з широкими можливостями, що надає більше 200 повнофункціональних сервісів для центрів обробки даних по всій планеті. Мільйони клієнтів, в тому числі стартапи, які стали лідерами за швидкістю зростання, найбільші корпорації і передові урядові установи, використовують AWS для зниження витрат, підвищення гнучкості і прискореного впровадження інновацій. Використовується для зберігання фотографій і бази даних веб-сервісу (рис. 2.15.).

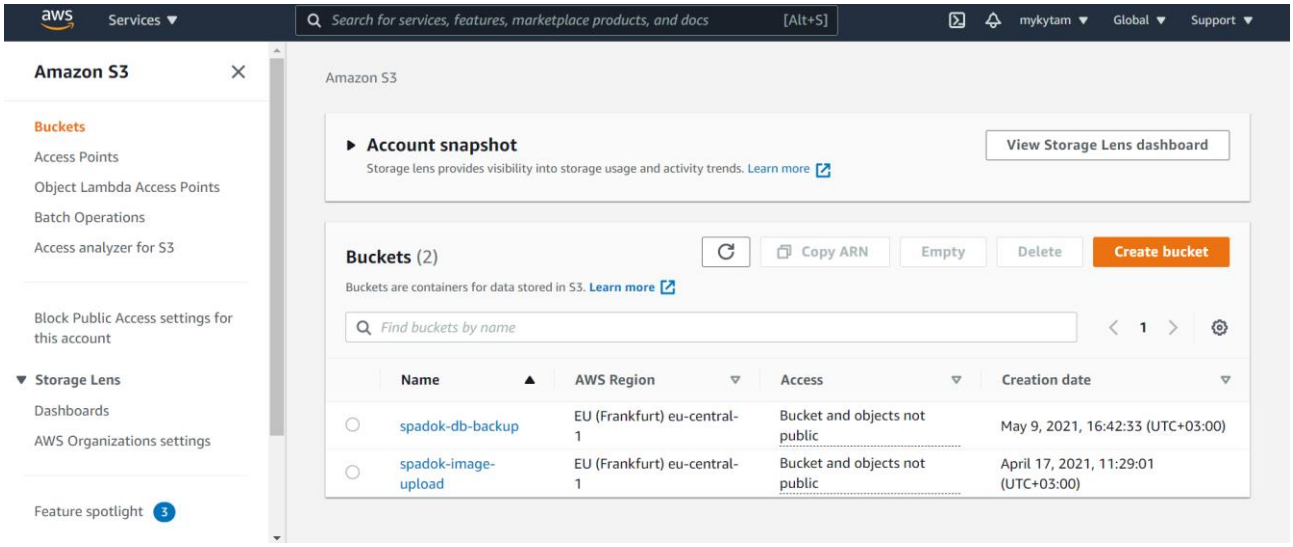

Рис. 2.15. Приклад S3 бакету розташованого AWS

2.6.3. Використана методологія розробки

Під час розробки даної кваліфікаційної роботи була використана гнучка методологія розробки Agile. Це ітеративний підхід до управління проектами та розробки ПЗ, що дозволяє командам прискорити доставку цінності клієнтам. Замість того щоб випускати весь продукт цілком, agile-команда виконує роботу в рамках невеликих, але зручних інкрементів. Вимоги, плани і результати постійно проходять перевірку на актуальність, завдяки чому команди можуть швидко реагувати на зміни.

Цей підхід був обраний через комплексність системи, що розробляється, тому розробка велася певними невеликими частинами. Agile сфокусована чотирьох ключових ідеях, на гнучкості та адаптивності підходу (рис. 2.16.):

– Організація ефективної взаємодії між людьми - базовий засіб досягнення цілей;

– Реально працюючий продукт є головною цінністю;

– Зміни, які можуть підвищити якість і конкурентоспроможність продукту, вітаються на будь-якому етапі розробки;

– Контрактна, технічна та інша регламентує документація вторинна по значущості щодо працюючого продукту і співпраці між учасниками проекту.

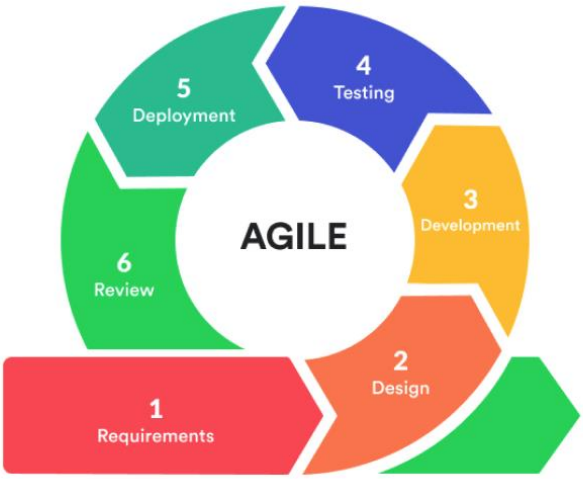

Рис. 2.16. Приклад циклу Agile

Веб-сервіс GitLab містить повний набір інструментів, які потрібні для Agile розробки.

Розробка системи велася так званими спринтами (рис. 2.17.). Спринт являє собою кінцевий проміжок часу, протягом якого робота повинна бути завершена, це може бути тиждень, кілька тижнів або, можливо, місяць або більше. При

розробці конкретно цієї кваліфікаційної роботи спринт мав тривалість один тиждень.

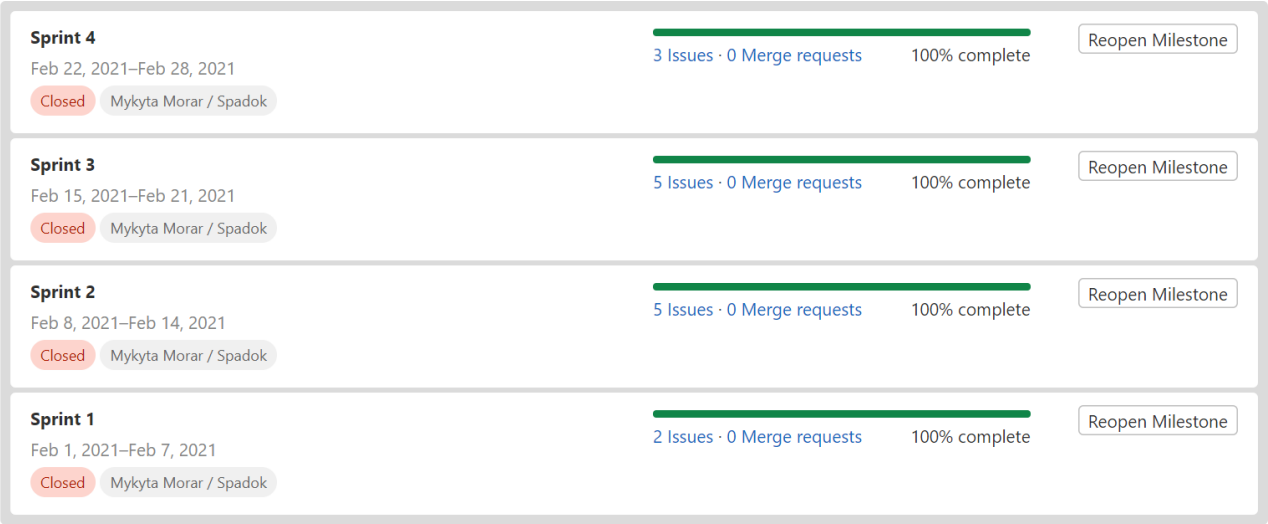

Рис. 2.17. Приклад спритів проекту

В Agile часто процес починається з user story (з англ. Історія користувача), яка охоплює одну функцію, яка потрібна для задоволення вимог (рис. 2.18.). У GitLab для цього служить issue в рамках проекту. Всього за час створення проекту було виконано близько 25 завдань.

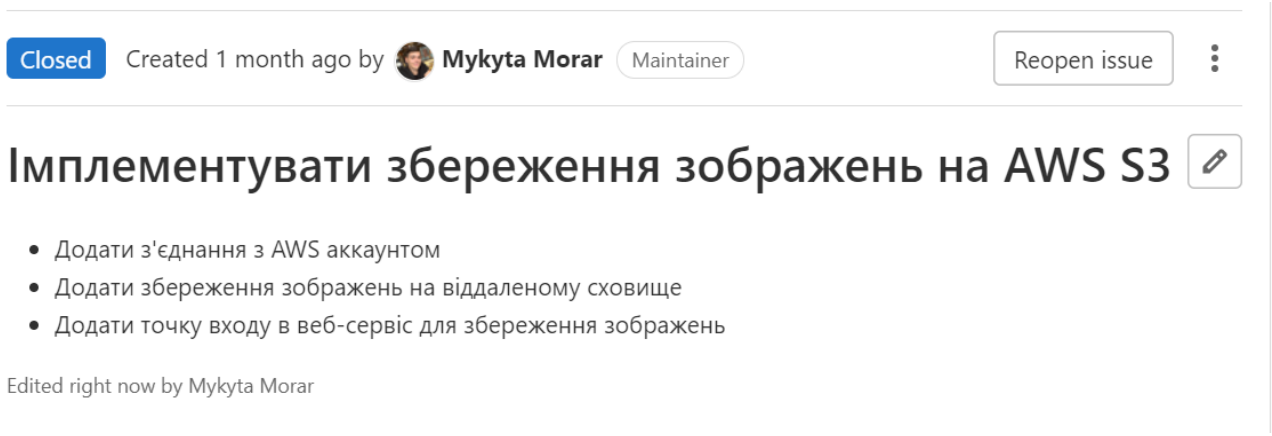

Рис. 2.18. Приклад однієї з user story

Всі активності, які були пов'язані з розробкою заносилися на "Issue Board" (з англ. Дошка завдань). Протягом спринту задачі проходять через різні етапи,

такі як To Do (готова до розробки), In Progress (ведеться розробка), Done (зроблено) и Closed (закрито) залежно від робочого процесу. У GitLab дошки (рис. 2.19.) дозволяють визначити етапи та дозволяють переміщувати задачі через них, це допомагає побачити стан спринту з точки зору робочого процесу.

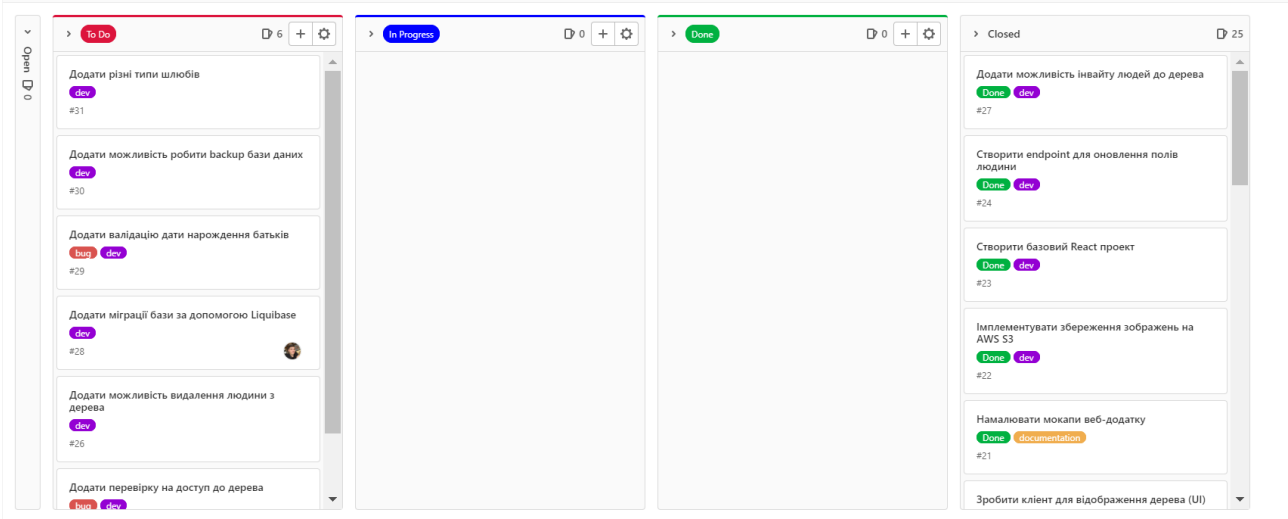

Рис. 2.19. Приклад дошки завдань

2.6.4. Виклик та завантаження програми

Для старту сервісу застосуються практика CI/CD – це комбінація безперервної інтеграції (continuous integration) і безперервного розгортання (continuous delivery або continuous deployment) програмного забезпечення в процесі розробки [22]. CI/CD об'єднує розробку і розгортання, прискорюючи процес складання, тестування і розгортання програми. Цей метод дає безліч плюсів. Метод забезпечує оперативність виведення нового функціоналу продукту. Як правило, це лічені години або дні. У той же час при класичному підході до розробки клієнтського софту це може зайняти тижні. Головні цілі CI / CD - звести до мінімуму помилки, прискорити складання і підвищити якість кінцевого продукту.

Безперервна інтеграція (CI) – це коли поновлення в код вносяться регулярно, до декількох разів на день. Інструменти CI, такі як GitLab, дозволяють створювати скрипти для автоматичного складання і тестування коду, після кожної завантаженої в репозиторій зміни. Спочатку перевірку проходить нова зміна окремо, після чого вноситься в код попередньої версії ПО, яке тестується вже цілком.

Безперервна доставка і Безперервне розгортання (CD) – після того, як зміни внесені і код протестований, ПО можна розгортати (деплоїти) – запускати його вже не на тестовому, а на робочих серверах. Безперервна доставка передбачає розгортання коду після кожної нової інтеграції. Ця операція проводиться вручну або автоматично. Безперервне розгортання – це вже повністю автоматичний процес, що проходить без контролю з боку команди.

У даній кваліфікаційній роботі використовуються обидві комбінації за допомогою CI/CD інструменту у GitLab (рис. 2.20. та рис. 2.21.). Після кожного нового коміту в якусь із гілок, налаштована безперервна інтеграція GitLab робить віддалено зборку проекту, що може відразу виявити, чи з'являться проблеми після додавання нової функціональності. Якщо зборка проекту пройшла успішно, то нові зміни автоматично розгортаються на віддаленому сервері (який надає PaaS платформа Heroku) за допомогою .jar файлу, який містить в собі нову версію проекту [21].

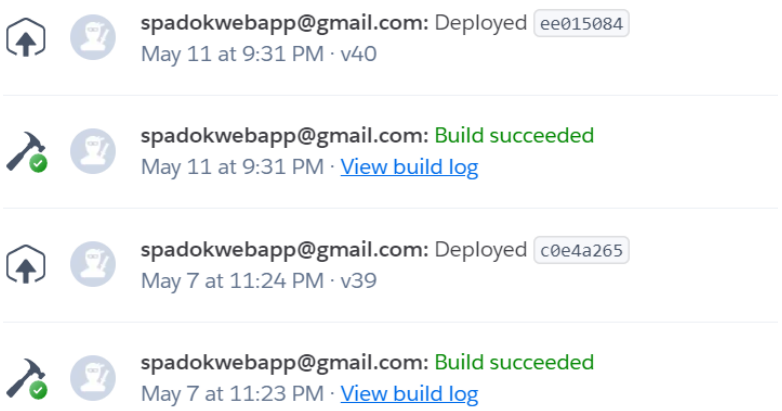

Рис. 2.20. Приклад безперервного розгортання на Heroku

На локальному комп'ютері виклик веб-сервісу відбувається за допомогою завантаження проекту збирачем проектів Maven, він запускає згенерований .jar файл, після чого можна перейти на порт (за стандартом 8080), де можна починати роботу з сервісом.

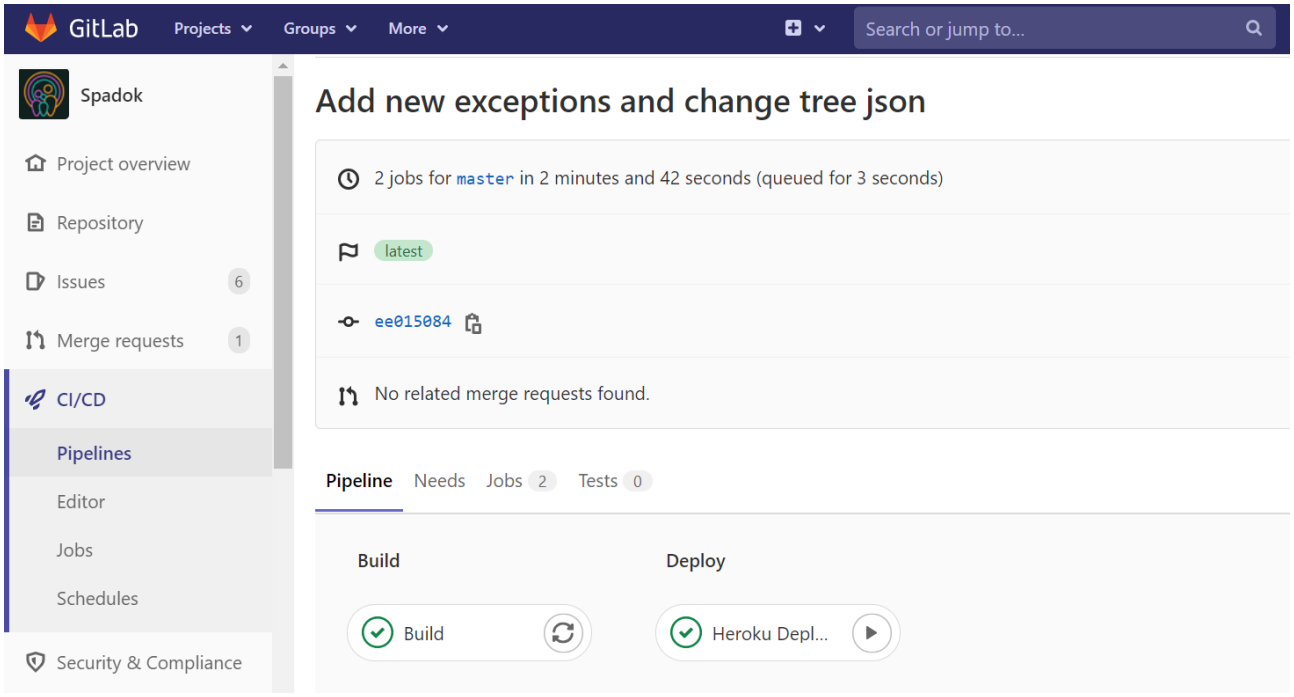

Рис. 2.21. Приклад безперервної інтеграції на GitLab

Клієнтська частина програми, яка складається з js і html файлів, не вимагає сервера для запуску проекту. Тому для запуску клієнтської частини можна просто перейти за певним URL, щоб відкрити сторінку клієнта (рис. 2.22.).

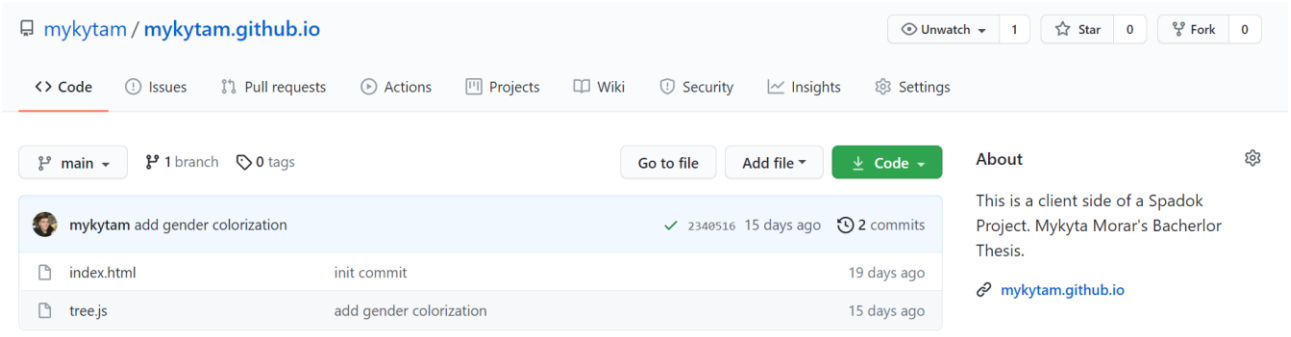

Рис. 2.22. Приклад репозиторію для GitHub Pages

Даний проект-клієнт розташовується на одному з найбільших веб-сервісів для спільної розробки програмного забезпечення GitHub, який надає механізм GitHub Pages для хостингу статичних веб-сторінок, що відмінно підходить для хостингу клієнтської частини кваліфікаційної роботи.

2.6.5. Опис інтерфейсу користувача

Так як фінальним продуктом кваліфікаційної роботи є веб-сервіс, це не передбачається клієнтську частину. Проблема в тому, що REST не є само описовим протоколом. Це означає, що клієнт повинен знати конкретну комбінацію URL, HTTP методу і формату відповіді для того, щоб почати взаємодію. У деяких випадках необхідно також знати також формат тіла запиту. Тому REST точки входу завжди повинні бути описані в одному конкретному документі, доступному для всіх інших розробників. Інструментом для візуалізації можливостей був обраний Swagger UI (рис. 2.23.).

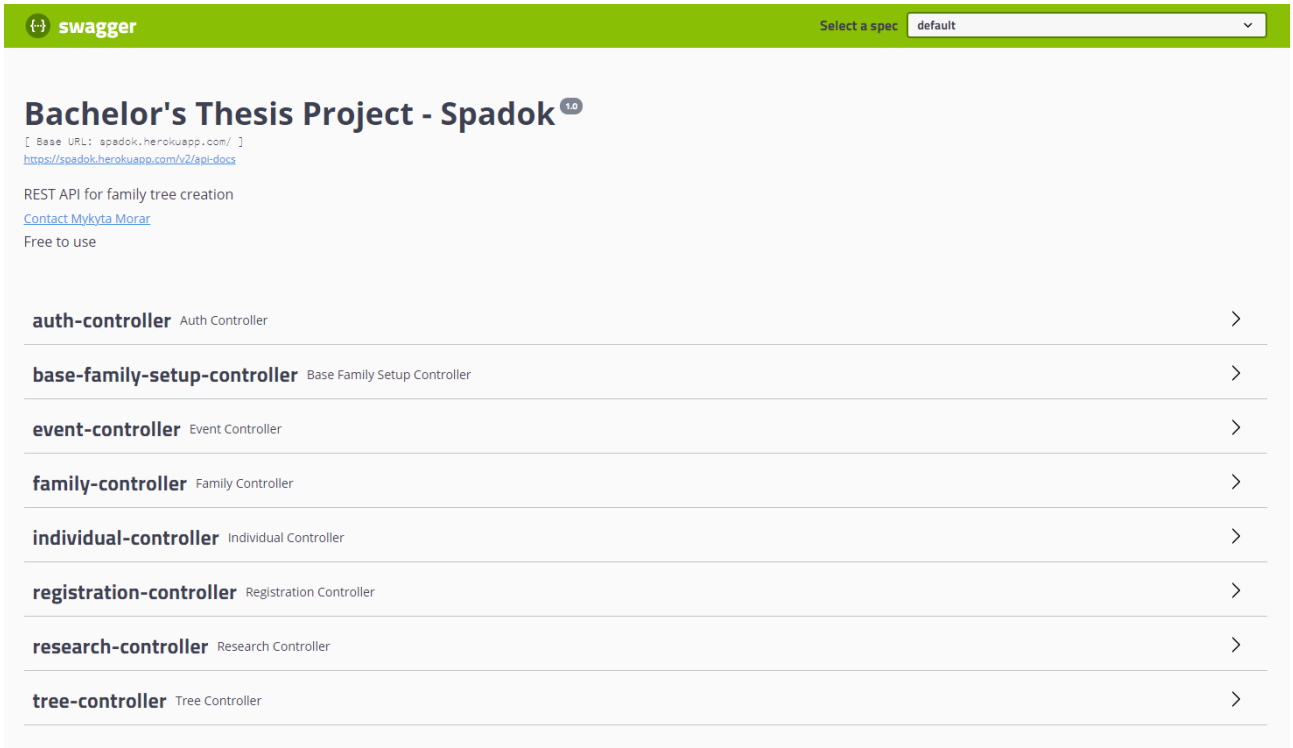

Рис. 2.23. Головна сторінка Swagger UI

По суті Swagger – це фреймворк для специфікації RESTful API. Його плюс полягає в тому, що він дає можливість не тільки інтерактивно переглядати специфікацію, а й відправляти запити на сервер. Після настройки фреймворку і опису коду, створюється html сторінка з документацією, яка генерується фреймворком і містить список з точок входу в веб-сервіс, потрібні тіла запитів для кожного запиту і багато іншого. На даній сторінці можна зробити відправку запиту на сервер і можна переглянути отримані дані. При вході на головну сторінку, користувач або програміст бачить повний список точок входу в додаток:

При розгортанні кожного з підзаголовків відкриваються доступні для користувачів точки входу в веб-сервіс, так звані «endpoints» (рис. 2.24.).

Рис. 2.24. Приклад доступних точок входу через контролер реєстрації

Поля для введення і відправлення запиту відкриваються при натисканні на точку входу (рис. 2.25.). Можливість редагування запиту відкривається після натискання кнопки «Try it out». Після заповнюється JSON файл і кнопкою «Execute» відправляється запит на сервер.

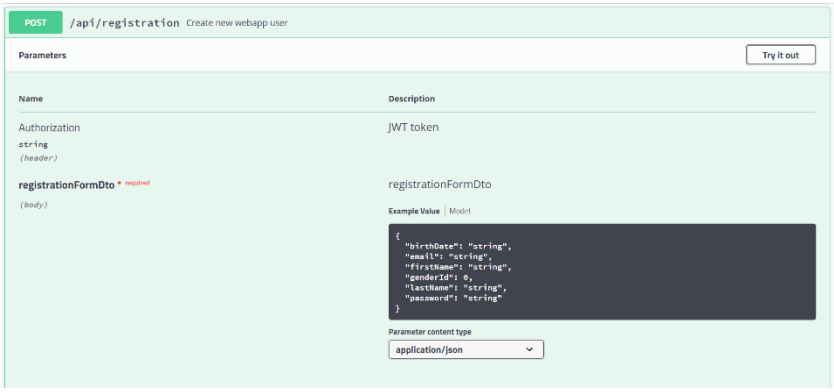

Рис. 2.25. Приклад форми запиту з полями

На кожен запит сервер посилає відповідь, вона може бути як позитивна (операція пройшла успішно), так і негативна (щось пішло не так, введені дані невірні). Кожна відповідь містить Response body, Response headers та Response Code (рис.2.26.).

– Response body - це тіло відповіді, в якому може зберігається корисна для користувача інформація (наприклад згенерований id нової моделі). Але тіло відповіді може бути і порожнім (як у випадку видалення якогось об'єкта).

– Reponse headers зберігають мета-дані про відповідь (дата, час, довга відповіді, тип з'єднання і так далі). Одне з "мета-полів" це authorization, який зберігає jwt токен, потрібний для авторизації та аутентифікації.

– Response Code - це HTTP код стану. Він являє собою ціле число з трьох десяткових цифр. Перша цифра вказує на клас стану.

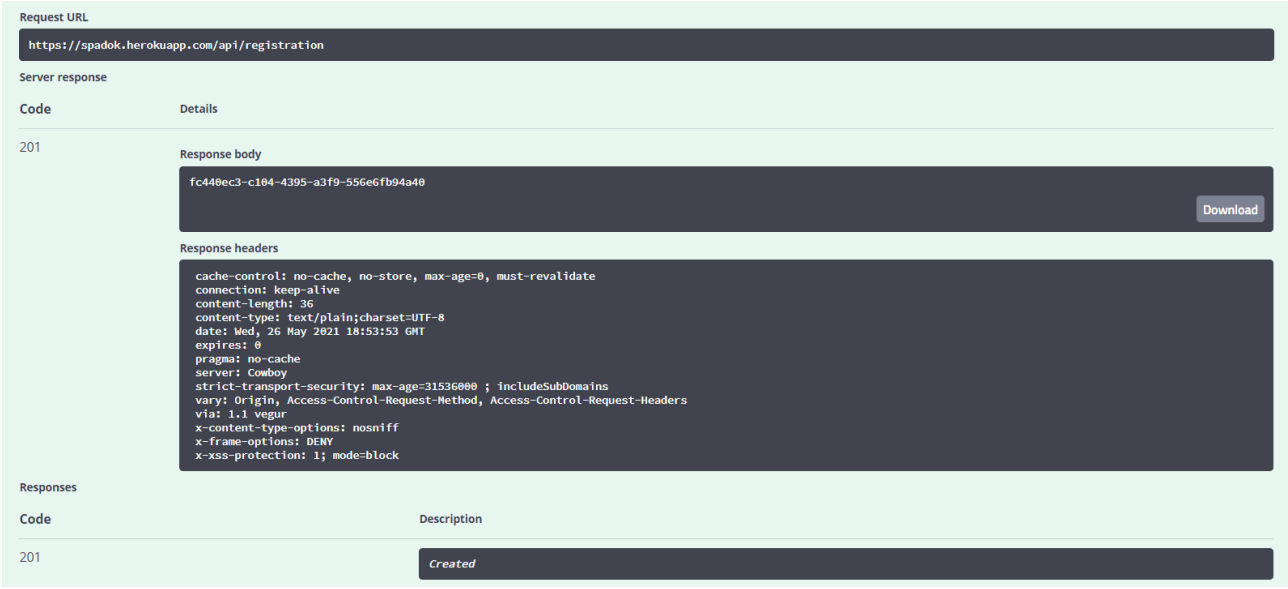

Рис. 2.26. Приклад отриманої відповіді з сервера

Для деяких точок входу в сервіс, таких як реєстрація або вхід за запрошенням, не потрібна перевірка на існуючого користувача. Але для багатьох endpoint потрібен JWT токен для здійснення запиту на сервер, його можна

отримати у запиті авторизації у веб-сервіс за допомогою раніше створених логіна і пароля. Токен знаходиться в headers відповіді в полі authorization. (рис. 2.27.)

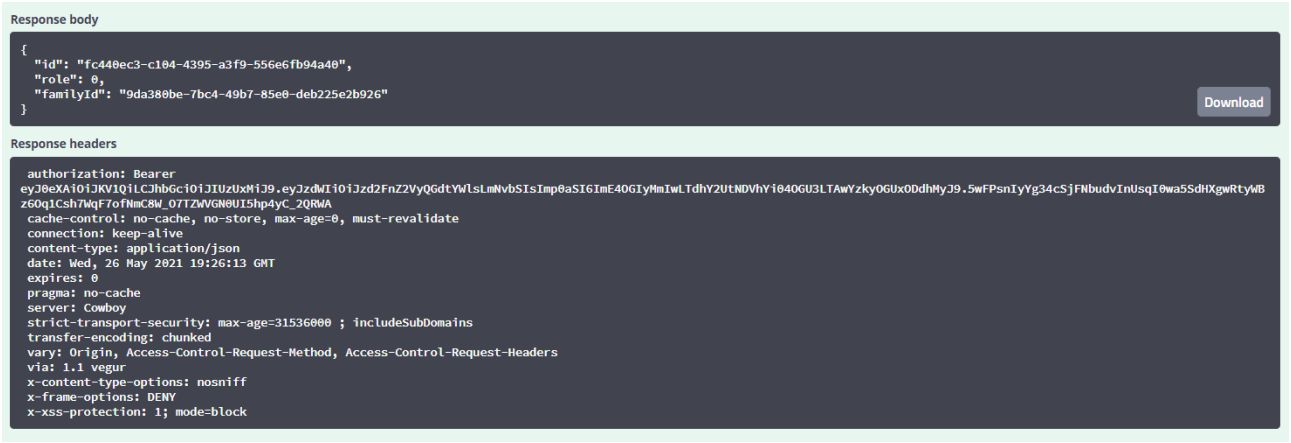

Рис. 2.27. Приклад відповіді на запит аутентифікації

Так само фреймворк Swagger UI робить зручним відправку запиту через автоматичну документованость коду, що дозволяє відразу відображати потрібні поля для відправки і зробити якісь поля обов'язковими (рис. 2.28.).

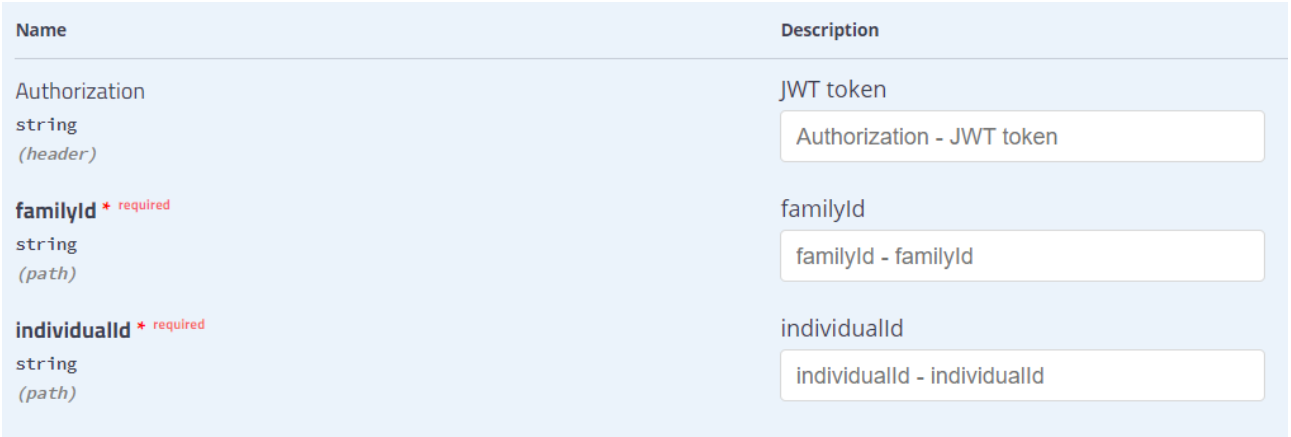

Рис. 2.28. Приклад полів для введення для створення запиту (з JWT токеном для аутентифікації)

Також в веб-сервісі присутня обробка нештатних ситуацій (рис. 2.29.). В такому випадку у відповіді на запит повертається JSON об'єкт, в якому міститися поля: status - це HTTP статус помилки; message - це зрозуміле повідомлення для

користувача API з якого можна зрозуміти, що викликало помилку; поле timestamp - це точна дата і час походження помилки. Ці поля будуть інформативні як і для клієнтів, так і для користувачів API і програмістів.

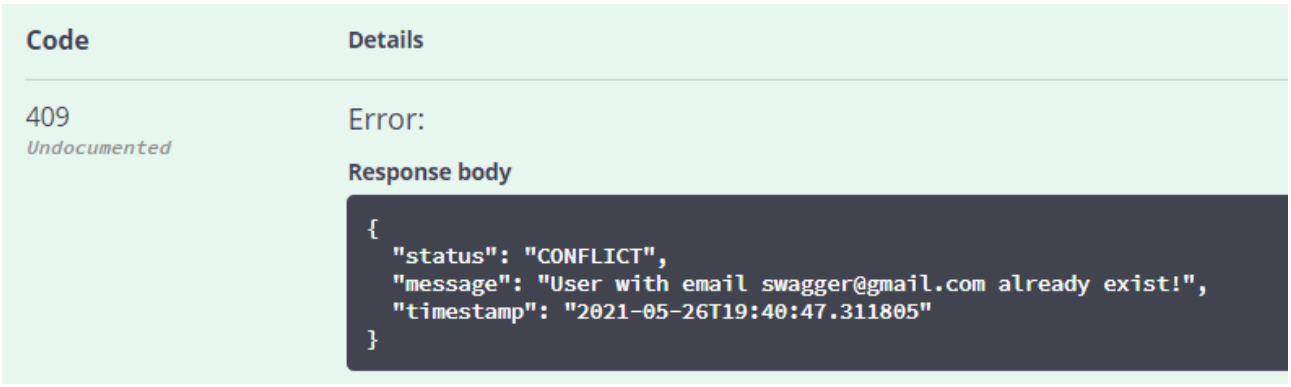

Рис. 2.29. Приклад відповіді на позаштатну ситуацію

Для основних сутностей надані кінцеві точки входу для дій: додавання, оновлення, отримання і видалення. У HTTP запитах це POST, PATCH, GET і DELETE відповідно. Також в веб-сервісі реалізовано пошук людей за системою сімейних дерев (рис. 2.30.)

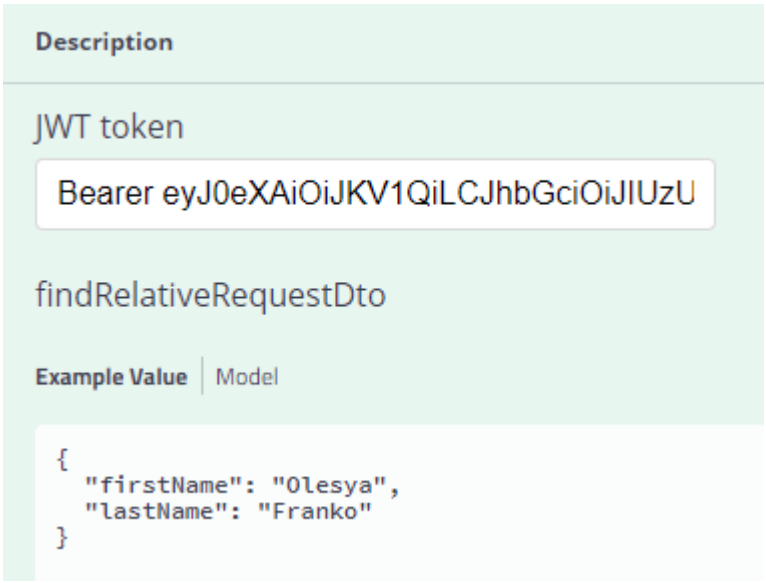

Рис. 2.30. Приклад запиту на сервер (пошук родича)

Для пошуку родича потрібно ввести його ім'я і прізвище. У разі якщо людина не знайдена, користувач отримає повідомлення з помилкою. У разі, якщо є збіг, один або більше, інформація про родичів буде описана у відповіді з сервера.

У відповіді (рис. 2.31.) знаходиться більш детальна інформація про шукану людину та адреса пошти адміністратора сімейного дерева, в якому знаходиться людина, за допомогою неї користувач може зв'язатися і уточнити інформацію про родича.

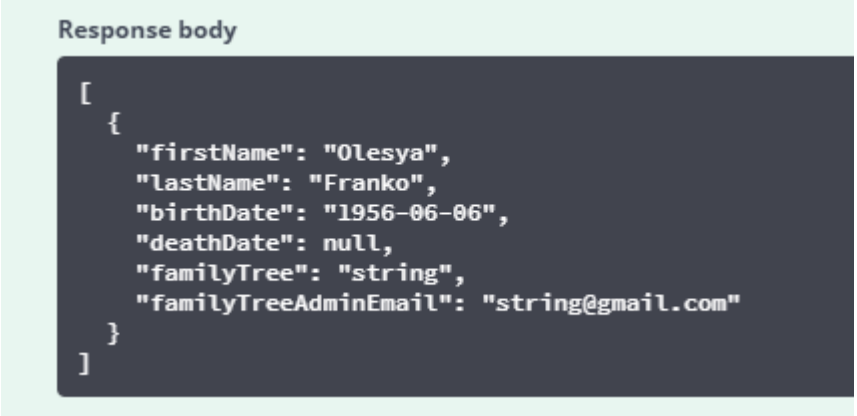

Рис. 2.31. Приклад відповіді з сервера на успішний пошук родича

Наступний список (рис. 2.32.) являє собою всі можливі точки входу в додатки, за допомогою який можна здійснювати дії з веб-сервісом і з якими повинні комунікувати клієнти. Список можна знайти за посиланням: https://spadok.herokuapp.com/swagger-ui.html.

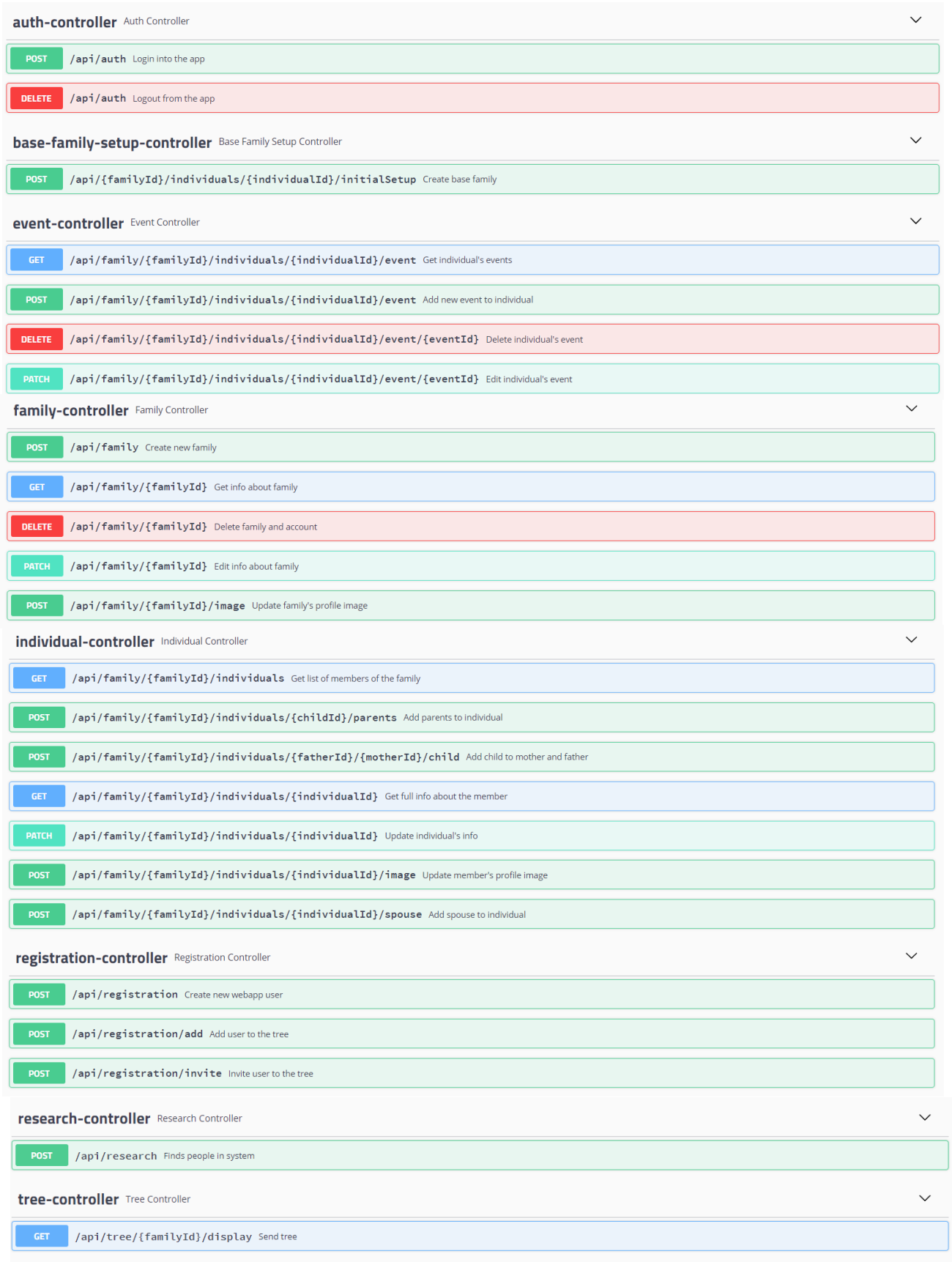

Рис. 2.32. Повний список всіх (24) точок входу в веб-сервіс

Додавання зображення користувачу відбувається так само через запит на сервер в формі запиту генерується фреймворком Swagger UI (рис. 2.33.):

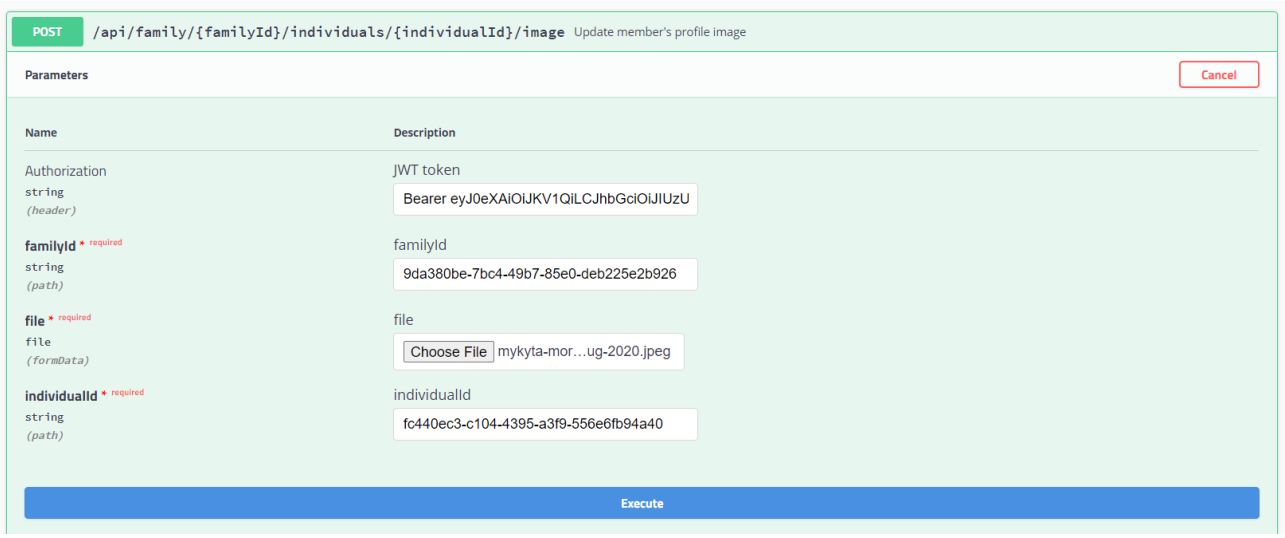

Рис. 2.33. Приклад форми для додавання зображення

При вдалому надсиланні зображення, у базу даних зберігається ідентифікатор зображення, а в сховищі S3 на Amazon Web Services додається новий файл з таким же ідентифікатором (рис. 2.34.).

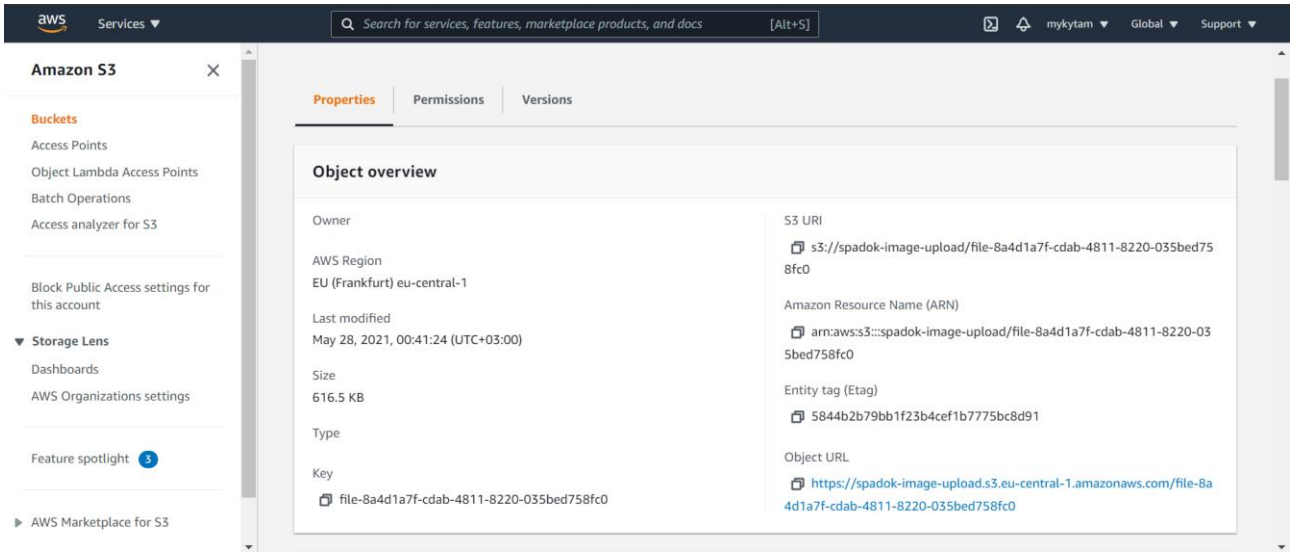

Рис. 2.34. Додане зображення яке зберігається на AWS S3

Крім візуального відображення можливостей веб-сервісу також був створений клієнт для відображення сімейних дерев. Головна сторінка виглядає наступним чином (рис. 2.35.) і запитує дані для входу:

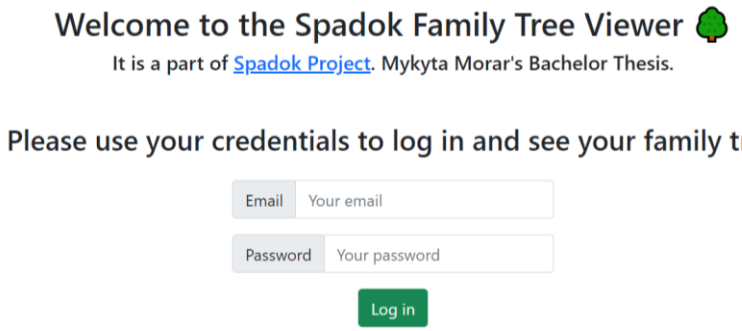

Рис. 2.35. Стартова сторінка клієнтської частини

Після введення даних користувачем відкривається базий вид сімейного дерева, яке відходить від користувача. У дереві є розбиття на жіночі і чоловіків по засобу кольорового відображення, рожевий - жінки, синій - чоловіки (рис. 2.36 та рис. 2.37).

Рис. 2.36. Базове відображення дерева

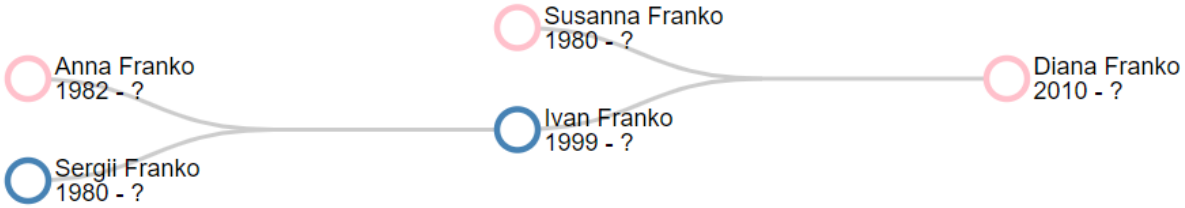

Рис. 2.37. Більш розгорнуте зображення дерева

Однією з додаткових особливостей відображення дерева є функція відображення дат народження і смерті, місць народження і смерті. Дану інформацію можна викликати при наведенні мишки на певну людину (рис. 2.38.).

Рис. 2.38. Відображення даних про користувача в дереві

Після більш детального заповнення даних про свою сім'ю, можна отримати досить велике дерево, що можна побачити на зображенні нижче (рис.2.39):

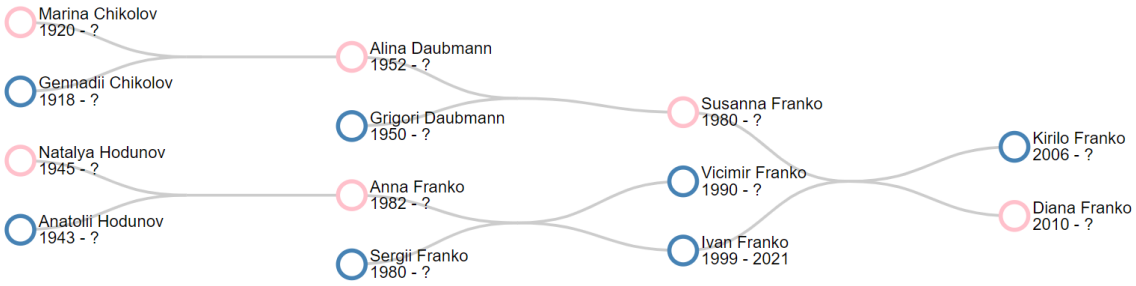

Рис. 2.39. Ще один приклад відображення родинного дерева

Клієнтську частину можна знайти за цим посиланням: https://mykytam.github.io/

2.6.6. Прогнозні припущення про розвиток об’єкту розробки

Основним напрямком розвитку об'єкта розробки є створення клієнтської частини проекту, яка буде взаємодіяти з веб-сервісом. Для цього були розроблені

мокапи (це макети, які дизайнери використовують для демонстрації своєї роботи), за допомогою яких подальша розробка може проходити більш зручно, так як вимоги до дизайна вже існують і є добре задокументований веб-сервіс (рис. 2.40.).

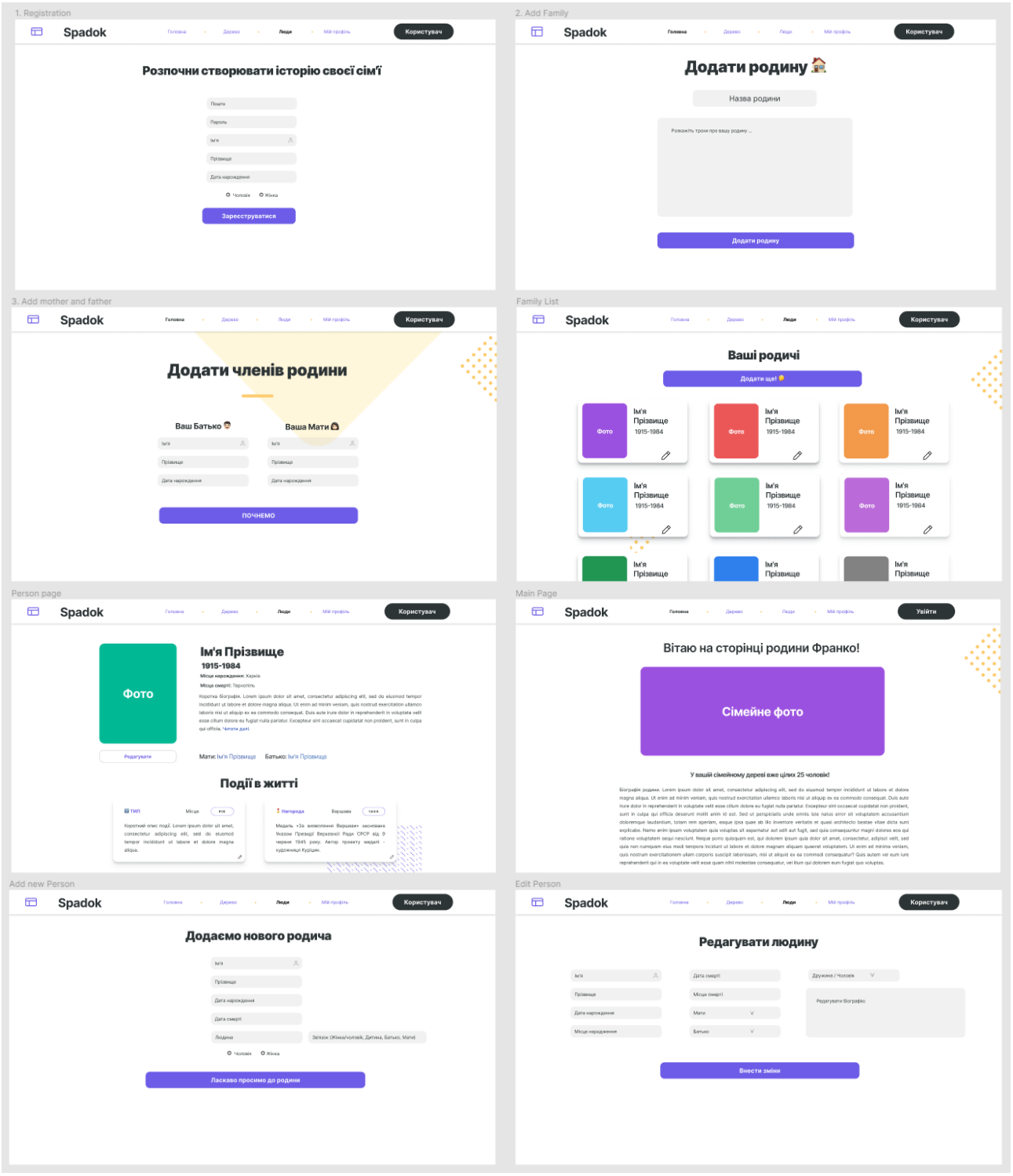

Рис. 2.40. Приклади сторінок реєстрації, додавання родичів і сім'ї, перегляд інформації про окремого родича і сім'ї, перегляд всіх родичів і т.д.

Крім додавання клієнтської частини, серверна частина також може бути поліпшена. З можливий поліпшень можна назвати:

- додавання різних типів шлюбів;
- додавання регулярного бекапу бази даних;
- додати пошук родичів на інших сайтах (таких як MyHeritage, FamilySearch та інші) через їх API;
- додавання загальної папки фотографій для користувачів для сімейного альбому;
- додавання міграцій бази даних через Liquibase;
- поліпшення генерації графічного виведення сімейного дерева.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 3853;

2. коефіцієнт складності програми – 1,3;

3. коефіцієнт корекції програми в ході її розробки – 0,07;

4. годинна заробітна плата програміста – 233 грн/год;

Середня годинна зарплата програміста була вирахувати виходячи з даних «Української спільноти програмістів (DOU)». Середньоукраїнська заробітна плата програміста, який пише програми на мові програмування Java і з досвідом роботи близько року дорівнює 1500 американських доларів в місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 27,34 грн, тому середня зарплата в гривнях дорівнює 41010 грн. При восьмигодинному робочому дні (176 годин в місяць в середнбому) середня зарплата за годину буде становитися 233 грн.

Джерела:

- Веб-сайт «Українська спільнота програмістів», URL: https://jobs.dou.ua/salaries/#period=dec2020&city=all&title=Software%20Engineer&language=Java&spec=&exp1=0&exp2=1
- Веб-сайт НБУ, URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,3;

7. вартість машино-години ЕОМ – 13,93 грн/год.

Звичайний комп'ютер споживає 180 Ватт / год, що дорівнює 0,18 кВт / год. Вартість одного кВт на годину від постачальника Yasno на стан середини 2021 року дорівнює 1,68 грн в незалежності від обсягу споживання. Отже вартість електроенергії споживаної комп'ютером за годину дорівнює 0,30 грн. Ціна довгострокової оренди комп'ютера дорівнює 2400 грн на місяць, з чого можна вирахувати, що ціна оренди на годину буде дорівнювати 13,63 грн. Остаточна вартість машино-години ЕОМ дорівнює 13,93 грн за годину.

Джерела:

- Веб-сайт постачальника електроенергії Yasno, URL: https://yasno.com.ua/b2c-tariffs
- Веб-сайт компанії орендаря комп'ютерів «ChipChip», URL: https://komputers.com.ua/arenda-kompyutera/

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин}, \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \text{ де} \qquad (3.2)$$

q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт корекції програми в ході її розробки.

$$Q = 3853 \cdot 1{,}3 \cdot (1 + 0{,}07) = 5359{,}52;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{5359{,}52 \cdot 1{,}2}{80 \cdot 1{,}3} = 61{,}84, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}; \qquad (3.4)$$

$$t_a = \frac{5359{,}52}{24 \cdot 1{,}3} = 171{,}77, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K}; \qquad (3.5)$$

$$t_n = \frac{5359,52}{25 \cdot 1,3} = 164,9, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5) \cdot K}; \qquad (3.6)$$

$$t_{\text{отл}} = \frac{5359,52}{5 \cdot 1,3} = 824,54, \text{ людино-годин,}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{\text{к}} = 1,2 \cdot t_{\text{отл}}; \qquad (3.7)$$

$$t_{\text{отл}}^{\text{к}} = 1,2 \cdot 824,54 = 1025,44, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \qquad (3.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{\partial \text{р}} = \frac{Q}{(15...20)\cdot K}; \qquad (3.9)$$

$$t_{\partial \text{р}} = \frac{5359,52}{17 \cdot 1,3} = 242,51, \text{ людино-годин.}$$

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}; \qquad (3.10)$$

$$t_{до} = 0{,}75 \cdot 242{,}51 = 181{,}88, \text{ людино-годин.}$$

$$t_{д} = 242{,}51 + 181{,}88 = 424{,}39, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 61,84 + 171,77 + 164,9 + 824,54 + 424,39 = 1697,44, людино-годин.

У результаті розрахував, що в загальній складності необхідно 1697,44 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 1697{,}44 \cdot 233 = 395503{,}52, \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 1697{,}44 \cdot 13{,}93 = 23645{,}33 \text{ грн.}$$

$$К_{ПО} = 395503{,}52 + 23645{,}33 = 419148{,}85 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k- число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = \frac{1697{,}44}{1 \cdot 176} = 9{,}6 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 1697,44 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 9,6 місяців при стандартному 40-годинному робочому тижні і 176-годинному

робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 419148,85 грн.

ВИСНОВКИ

В даній кваліфікаційній роботі був розроблений веб сервіс для створення сімейних дерев і вивчення родоводу.

Це програмне забезпечення призначене для надання можливості людям безкоштовно і без лімітів почати створення свого сімейного дерева і зайнятися вивченням родоводу по засобу заповнення даних про свою сім'ю і користуванням пошуку родичів по системі.

В результаті розробки веб-сервісу були реалізовані наступні можливості: реєстрація користувача, запрошення користувача зареєструватися в сервісі, реєстрація користувача на запрошення, авторизація та аутентифікація, вихід з сервісу, операції додавання / зміни / отримання / видалення сім'ї, додавання фотографій членам сім'ї та загальної фотографії для сім'ї, операції додавання / зміни / отримання / видалення подій в житті члена сім'ї, пошук родичів по веб-сервісу, додавання людей в сімейне дерево, оновлення / видалення інформації / біографії родичів, додавання зв'язків між родичами, відображення сімейного дерева в зручному графічному форматі з можливістю приховування частин дерева і переглядом базової інформації про людей.

Під час виконання даної кваліфікаційної роботи проекту були виконані наступні задачі:

– проаналізовано предметну область задачі;

– проведено порівняння з можливостей існуючих подібних сервісів;

– обрано раціональну структуру бази даних;

– написано програмний код веб-сервісу;

– розроблено зручний, захищений і добре задокументований API інтерфейс;

– розроблено пов'язаний з серверною частиною клієнт для генерації графічного відображення родинних дерев;

– проаналізовано подальший розвиток проекту.

Веб-сервіс реалізований на мові програмування Java з використанням фреймворку Spring. Використовувана база даних – PostgreSQL. Клієнтська частина була розроблена з використанням мови програмування JavaScript з використанням бібліотеки js_family_tree. Зберігання бази даних і файлів здійснюється на платформі хмарних обчислень Amazon Web Services і PaaS-платформі Heroku.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту (1697,44 людино-годин), проведений підрахунок вартості роботи по створенню програми (419148,85 грн) та розраховано час на його створення (9,6 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Коротенко Л. М., Шевцова О. С. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 "Комп'ютерні науки" галузі знань 12 Інформаційні технології. Нац. тех. ун-т. - Д : ДВНЗ НТУДП, 2021. - 65 с.
2. Вагонова О. Г., Волотковська Ю. А., Романюк Н.Н. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Програмне забезпечення”. Дніпропетровськ: Національний гірничий університет, 2013. – 11 с.
3. Арно Л. Проектування веб-API. Київ : ДМК Прес, 2019. – 440 с.
4. Галісеев Г. В. Програмування веб-застосувань (фронт-енд та бек-енд). Львів : Університет "Україна", 2019. – 113 с.
5. Фуско М. Сучасна мова Java. Лямбда-вирази, потоки і функціональне програмування. Київ : ДМК Прес, 2019. – 592 с.
6. Козміна Ю., Шефер К. Pro Spring 5: Поглиблений посібник із Spring Framework та його інструментів. Київ : Видавництво Діалектика, 2016. – 1120 с.
7. Калін М. Java Web Services: Up and Running. Харків : Наука і техніка, 2018. - 320 с.
8. Best genealogy sites 2021. URL: https://www.toptenreviews.com/best-genealogy-websites
9. Types of Family Trees or Pedigrees. URL: http://genj.sourceforge.net/wiki/en/manual/fam_tree_types

10. Family Tree Chart Types and Examples. URL: https://www.familytreemagazine.com/resources/family-tree-chart-types/

11. Building an Application with Spring Boot. URL: https://spring.io/guides/gs/spring-boot/

12. What is a REST API?. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api

13. JSON Web Token Introductio. URL: https://jwt.io/introduction

14. D3.js - Data-Driven Documents. URL: https://github.com/d3/d3/wiki

15. Best way to map the JPA and Hibernate ManyToMany relationship. URL: https://vladmihalcea.com/the-best-way-to-use-the-manytomany-annotation-with-jpa-and-hibernate/

16. Spring Data Access documentation. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#spring-data-tier

17. Spring MVC Web documentation: URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#spring-web

18. Guide to Spring Type Conversions | Baeldung. URL: https://www.baeldung.com/rest-with-spring-series

19. Guide to Spring Type Conversions. URL: https://www.baeldung.com/spring-type-conversions

20. PostgreSQL documentation. UUID Type. URL: https://www.postgresql.org/docs/9.1/datatype-uuid.html

21. Deploying with Git | Heroku Dev Center. URL: https://devcenter.heroku.com/articles/git

22. Keyword reference for the .gitlab-ci.yml file. URL: https://docs.gitlab.com/ee/ci/yaml/

23. Creating, configuring, and working with Amazon S3 buckets. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/creating-buckets-s3.html

24. Порівнюємо інструменти для CI/CD: Teamcity, Jenkins, Bitbucket та інші. URL: https://dou.ua/forums/topic/32724/

25. Building my Family Tree. URL: https://bigomega.medium.com/building-my-family-tree-ef0be1fba775

26. Graphs and geneaology. modelling the data. URL: https://simonthordal.github.io/neo4j/genealogy/2017/01/03/Graphs-and-Geneaology.-Modelling-the-Data/

27. Good Relationships: The Spring Data Neo4j Guide Book. URL: https://docs.spring.io/spring-data/neo4j/docs/4.1.7.RELEASE/reference/html/

28. Drawing Family Trees With JavaScript. URL: https://www.yworks.com/pages/drawing-family-trees-with-javascript

29. Beginner's D3.js Tutorial: Learn Data Visualization with JS. URL: https://www.educative.io/blog/d3js-tutorial

ДОДАТОК А

КОД ПРОГРАМИ

У додатку наведені приклади коду класів типу сервіс, контролер, репозиторій, домен, конвертер, виключення, dto і конфігурація. У додатку наведено приклади 26 файлів, загальна кількість файлів - 103, вони додаються на оптичному носії разом з кваліфікаційною роботою.

Приклад класів типу домен

BaseEntity.java

```
package com.spadok.persistence.domain;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.GenericGenerator;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;

import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;
import java.io.Serializable;
import java.time.LocalDateTime;

@Getter
@Setter
@MappedSuperclass
public abstract class BaseEntity implements Serializable {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(name = "UUID", strategy = "org.hibernate.id.UUIDGenerator")
    private String id;

    @CreatedDate
    private LocalDateTime createdDate;

    @LastModifiedDate
    private LocalDateTime lastModifiedDate;
}
```

EventType.java

```
package com.spadok.persistence.domain.enums;


public enum EventType {

    MARRIAGE (0, "Marriage"),
```

```
    ENGAGEMENT (1, "Engagement"),
    DIVORCE (2, "Divorce"),
    RESETTLEMENT (3, "Resettlement"),
    GRADUATION (4, "Graduation"),
    MILITARY_SERVICE (5,"Military Service"),
    REWARD (6, "Reward"),
    EMIGRATION (7, "Emigration");

    public final Integer id;
    public final String value;

    EventType(Integer id, String value) {
        this.id = id;
        this.value = value;
    }
}
```

Individual.java

```
package com.spadok.persistence.domain;
import com.spadok.persistence.domain.enums.Gender;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.experimental.Accessors;
import javax.persistence.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Accessors(chain = true)
@RequiredArgsConstructor
@Table(name = "individual")
public class Individual extends BaseEntity {
```

```
@Column(name = "first_name")
private String firstName;

@Column(name = "last_name")
private String lastName;

@Column(name = "birth_date")
private LocalDate birthDate;

@Column(name = "death_date")
private LocalDate deathDate;

@Column(name = "photoUrl")
private String photoUrl;

@Enumerated(EnumType.ORDINAL)
@Column(name = "gender_id")
private Gender gender;

@Column(name = "biography")
private String biography;

@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "place_of_birth_id")
private Place placeOfBirth;

@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "place_of_residence_id")
private Place placeOfResidence;

@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "place_of_death_id")
private Place placeOfDeath;

@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "online_user_id")
```

```
    private OnlineUser onlineUser;

    @OneToOne(mappedBy = "treeAdmin")
    private Family familyAdmin;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "family_id")
    private Family family;

    @OneToMany(mappedBy = "individual", cascade = CascadeType.ALL)
    private List<Event> events;

    @OneToMany(cascade=CascadeType.ALL)
    private List<Marriage> marriages = new ArrayList<>();

    public void addToEventList(Event event) {
        events.add(event);
        event.setIndividual(this);
    }

    public void addMarriage(Marriage marriage) {
        this.marriages.add(marriage);
    }
}
```

Place.java

```
package com.spadok.persistence.domain;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import javax.persistence.*;
import java.util.List;

@Getter
@Setter
@Entity
```

```
@Table(name = "place")
@RequiredArgsConstructor
public class Place extends BaseEntity {

    @Column(name = "place_name")
    private String placeName;

    @OneToMany(mappedBy = "place", cascade = CascadeType.PERSIST)
    private List<Event> events;

    @OneToMany(mappedBy = "placeOfBirth", cascade = CascadeType.PERSIST)
    private List<Individual> individualBirth;

    @OneToMany(mappedBy = "placeOfResidence", cascade = CascadeType.PERSIST)
    private List<Individual> individualResidence;

    @OneToMany(mappedBy = "placeOfDeath", cascade = CascadeType.PERSIST)
    private List<Individual> individualDeath;

    public void addToEventsList(Event event) {
        events.add(event);
        event.setPlace(this);
    }
}
```

Приклад класів типу репозиторій

IndividualRepository.java

```
package com.spadok.persistence.repository;

import com.spadok.persistence.domain.Family;
import com.spadok.persistence.domain.Individual;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
```

```
import java.util.List;

@Repository
public interface IndividualRepository extends PagingAndSortingRepository<Individual, String> {

    List<Individual> findIndividualsByFamily(Family family);
    Individual findIndividualById(String individualId);
    List<Individual> findAllByFirstNameAndLastName(String firstName, String lastName);
    @Query(value = "from Individual i where i.onlineUser.email= :onlineUserEmail")
    Individual findByOnlineUser(@Param("onlineUserEmail") String onlineUserEmail);
}
```

MarriageRepository.java

```
package com.spadok.persistence.repository;
import com.spadok.persistence.domain.Individual;
import com.spadok.persistence.domain.Marriage;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface MarriageRepository extends PagingAndSortingRepository<Marriage, String> {

   Marriage findByPersonId(String personId);
   Marriage findBySpouse(Individual spouse);
   List<Marriage> findAllBySpouse(Individual spouse);
   boolean existsBySpouse(Individual spouse);
   boolean existsByPersonId(String personId);
   Marriage findByPersonIdAndSpouse(String personId, Individual spouse);
   boolean existsByPersonIdAndSpouse(String personId, Individual spouse);
   Marriage findByChildrenContains(Individual child);
   boolean existsByChildrenContains(Individual child);
   @Query(value = "from Marriage m where m.spouse.family.id = :familyId")
```

```
    List<Marriage> findAllMarriagesByFamily(@Param("familyId") String familyId);
}
```

Приклад класів типу конвертер

CreateDtoToEventConverter.java

```
package com.spadok.service.converter.event;
import com.spadok.persistence.domain.Event;
import com.spadok.persistence.domain.enums.EventType;
import com.spadok.service.dto.event.EventCreateDto;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

@Component
public class CreateDtoToEventConverter implements Converter<EventCreateDto, Event> {

    @Override
    public Event convert(EventCreateDto eventCreateDto) {
        return new Event()
                .setEventDate(eventCreateDto.getEventDate())
                .setEventType(EventType.valueOf(eventCreateDto.getEventType().toUpperCase()))
                .setEventDescription(eventCreateDto.getEventDescription());
    }
}
```

FamilyToResponseDtoConverter.java

```
package com.spadok.service.converter.family;
import com.spadok.persistence.domain.Family;
import com.spadok.service.dto.family.FamilyResponseDto;
import org.springframework.core.convert.converter.Converter;
import org.springframework.stereotype.Component;

@Component
public class FamilyToResponseDtoConverter implements Converter<Family, FamilyResponseDto> {
    @Override
```

```
    public FamilyResponseDto convert(Family family) {
        return new FamilyResponseDto()
                .setFamilyName(family.getFamilyName())
                .setFamilyDescription(family.getFamilyDescription())
                .setPhotoUrl(family.getFamilyPhoto())
                .setAmountOfMembers(family.getMembersOfFamily().size());
    }
}}
```

Приклад класів типу сервіс

AuthService.java

```
package com.spadok.service.security;
import com.spadok.persistence.domain.Individual;
import com.spadok.persistence.domain.OnlineUser;
import com.spadok.persistence.repository.IndividualRepository;
import com.spadok.persistence.repository.OnlineUserRepository;
import com.spadok.service.dto.auth.LoginRequestDto;
import com.spadok.service.dto.auth.LoginResponseDto;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.stereotype.Service;

import static com.spadok.service.constants.Headers.AUTH_HEADER;
import static com.spadok.service.constants.Headers.AUTH_HEADER_PREFIX;
import static com.spadok.service.service.PasswordUtilService.hashPassword;

@Service
@RequiredArgsConstructor
public class AuthService {

    private final AuthenticationManager authenticationManager;
    private final OnlineUserRepository onlineUserRepository;
    private final IndividualRepository individualRepository;
    private final JwtTokenService jwtTokenService;
```

```
    private final TokenStore tokenStore;

    public ResponseEntity<LoginResponseDto> authenticate(LoginRequestDto dto) {
        String email = dto.getEmail();
        String password = dto.getPassword();
        OnlineUser onlineUser = onlineUserRepository.findByEmail(email);
        Individual individual = individualRepository.findByOnlineUser(onlineUser.getEmail());

        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(email, hashPassword(password,
onlineUser.getSalt())));

        String token = AUTH_HEADER_PREFIX + jwtTokenService.createToken(email);

        tokenStore.putToken(token);

        LoginResponseDto loginResponseDto = new LoginResponseDto(
                onlineUser.getIndividual().getId(),
                onlineUser.getRole().getValue(),
                individual.getFamily() != null ? individual.getFamily().getId() : null
        );

        return ResponseEntity.ok().header(AUTH_HEADER, token).body(loginResponseDto);
    }

    public void logout(String header) {
        tokenStore.removeToken(header);
    }
}
```

EventService.java

```
package com.spadok.service.service;

import com.spadok.persistence.domain.Event;
import com.spadok.persistence.domain.Place;
import com.spadok.persistence.domain.enums.EventType;
import com.spadok.persistence.repository.EventRepository;
import com.spadok.persistence.repository.IndividualRepository;
import com.spadok.persistence.repository.PlaceRepository;
```

```
import com.spadok.service.dto.event.EventCreateDto;
import com.spadok.service.dto.event.EventResponseDto;
import lombok.RequiredArgsConstructor;
import org.springframework.core.convert.ConversionService;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import javax.persistence.EntityNotFoundException;
import java.time.LocalDateTime;
import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
@RequiredArgsConstructor
public class EventService {

    private final IndividualRepository individualRepository;
    private final PlaceRepository placeRepository;
    private final EventRepository eventRepository;
    private final ConversionService conversionService;

    public String addEventToIndividual(String familyId, String individualId, EventCreateDto eventCreateDto) {
        var individual = individualRepository.findIndividualById(individualId);
        var event = conversionService.convert(eventCreateDto, Event.class);
        event.setCreatedDate(LocalDateTime.now());

        if (placeRepository.existsByPlaceName(eventCreateDto.getPlace())) {
            event.setPlace(placeRepository.findByPlaceName(eventCreateDto.getPlace()));
        } else {
            Place place = new Place();
            place.setPlaceName(eventCreateDto.getPlace());
            place.setCreatedDate(LocalDateTime.now());
            event.setPlace(place);
            placeRepository.save(place);
        }
```

```
    individual.addToEventList(event);
    eventRepository.save(event);
    individualRepository.save(individual);
    return event.getId();
}

public List<EventResponseDto> getEventsFromIndividual(String familyId, String individualId) {
    var individual = individualRepository.findIndividualById(individualId);
    return eventRepository.findAllByIndividual(individual)
            .stream()
            .map(event -> conversionService.convert(event, EventResponseDto.class))
            .collect(Collectors.toList());
}

public void editEvent(String familyId, EventCreateDto eventDto, String individualId, String eventId) {
    var individual = individualRepository.findIndividualById(individualId);
    var event = eventRepository.findById(eventId)
            .orElseThrow(() -> new EntityNotFoundException("Event with id " + eventId + " not found!"));;

    if (eventDto.getEventDescription() != null) {
        event.setEventDescription(eventDto.getEventDescription());
    }

    if (eventDto.getEventType() != null) {
        event.setEventType(EventType.valueOf(eventDto.getEventType().toUpperCase()));
    }

    if (eventDto.getEventDate() != null) {
        event.setEventDate(eventDto.getEventDate());
    }

    if (eventDto.getEventDescription() != null) {
        event.setEventDescription(eventDto.getEventDescription());
    }

    if (eventDto.getPlace() != null) {
```

```
            if (placeRepository.existsByPlaceName(eventDto.getPlace())) {
                event.setPlace(placeRepository.findByPlaceName(eventDto.getPlace()));
            } else {
                Place place = new Place();
                place.setPlaceName(eventDto.getPlace());
                place.setCreatedDate(LocalDateTime.now());
                event.setPlace(place);
                placeRepository.save(place);
            }
        }

        event.setLastModifiedDate(LocalDateTime.now());
        eventRepository.save(event);
    }

    public void deleteEvent(String familyId, String individualId, String eventId) {
        var event = eventRepository.findById(eventId)
                .orElseThrow(() -> new EntityNotFoundException("Event with id " + eventId + " not found!"));
        eventRepository.delete(event);
    }
}
```

Приклад класів типу DTO

InviteFormDto.java

```
package com.spadok.service.dto.auth;

import com.spadok.service.annotations.UniqueEmail;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.Accessors;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;

@Getter
@Setter
@Accessors(chain = true)
public class InviteFormDto {
```

```
    @UniqueEmail
    @NotBlank
    @Pattern(
            regexp = "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$",
            message = "Invalid email"
    )
    private String recipientEmail;

    @NotBlank
    private String familyId;

    @NotBlank
    private String individualId;
}
```

TreeDto.java

```
package com.spadok.service.dto.tree;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.Accessors;

import java.util.List;
import java.util.Map;

@Getter
@Setter
@Accessors(chain = true)
@JsonInclude(JsonInclude.Include.NON_NULL)
public class TreeDto {

    private String start;
    private Map<String, TreeEntity> persons;
    private Map<String, Union> unions;
    private List<List<String>> links;
```

}

Приклад класів типу конфігурація

AmazonConfig.java

```
package com.spadok.config;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AmazonConfig {

    @Value("${amazon.aws.access.key}")
    private String accessKey;

    @Value("${amazon.aws.secret.key}")
    private String secretKey;

    @Bean
    public AmazonS3 s3() {
        AWSCredentials awsCredentials = new BasicAWSCredentials(accessKey, secretKey);

        return AmazonS3ClientBuilder
                .standard()
                .withRegion("eu-central-1")
                .withCredentials(new AWSStaticCredentialsProvider(awsCredentials))
                .build();
    }
}
}
```

.gitlab-ci.yaml

```
stages:
  - build
  - deploy

image: bellsoft/liberica-openjdk-alpine:11.0.8

variables:
  MAVEN_OPTS: "-Dmaven.repo.local=./.m2/repository"
  HEROKU_API_KEY: "KEY-HERE"

cache:
  paths:
    - ./.m2/repository

Build:
  stage: build
  when: on_success
  before_script:
    - chmod +x mvnw
  script:
    - ./mvnw $MAVEN_OPTS install

Heroku Deploy:
  stage: deploy
  when: manual
  image: ruby:latest
  cache: {}
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl --provider=heroku --app=spadok --api-key=$HEROKU_API_KEY
  environment:
    name: staging
```

```
  url: https://spadok.herokuapp.com/
 only:
  - master
```

Procfile

```
web: java -Dserver.port=$PORT $JAVA_OPTS -jar spadok-web/target/spadok-web-0.0.1-SNAPSHOT-exec.jar --
```

Приклад класів типу контролер

AuthController.java

```
package com.spadok.controller;
import com.spadok.service.dto.auth.LoginRequestDto;
import com.spadok.service.dto.auth.LoginResponseDto;
import com.spadok.service.security.AuthService;
import io.swagger.annotations.ApiOperation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import static com.spadok.service.constants.Headers.AUTH_HEADER;

@RestController
@RequestMapping("api/auth")
@RequiredArgsConstructor
public class AuthController {

   private final AuthService authService;

   @ApiOperation(value = "Login into the app")
   @PostMapping
   public ResponseEntity<LoginResponseDto> login(@RequestBody LoginRequestDto loginRequestDto) {
      return authService.authenticate(loginRequestDto);
   }

   @ApiOperation(value = "Logout from the app")
   @DeleteMapping
   public void logout(@RequestHeader(AUTH_HEADER) String header) {
      authService.logout(header);
```

```
    }
}
```

EventController.java

```
package com.spadok.controller;

import com.spadok.service.dto.event.EventCreateDto;
import com.spadok.service.dto.event.EventResponseDto;
import com.spadok.service.service.EventService;
import io.swagger.annotations.ApiOperation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequiredArgsConstructor
@RequestMapping("api/family/{familyId}/individuals/{individualId}/event")
public class EventController {

    private final EventService eventService;

    @ApiOperation(value = "Add new event to individual")
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public String addEventToIndividual(@PathVariable String familyId,
                                       @PathVariable String individualId,
                                       @RequestBody EventCreateDto eventCreateDto) {
        return eventService.addEventToIndividual(familyId, individualId, eventCreateDto);
    }

    @ApiOperation(value = "Get individual's events")
    @GetMapping
    @ResponseStatus(HttpStatus.OK)
    public List<EventResponseDto> addEventToIndividual(@PathVariable String familyId,
                                                        @PathVariable String individualId) {
```

```
        return eventService.getEventsFromIndividual(familyId, individualId);
    }

    @ApiOperation(value = "Edit individual's event")
    @PatchMapping("{eventId}")
    @ResponseStatus(HttpStatus.OK)
    public void editEvent(@PathVariable String familyId,
                          @RequestBody EventCreateDto eventDto,
                          @PathVariable String individualId,
                          @PathVariable String eventId) {
        eventService.editEvent(familyId, eventDto, individualId, eventId);
    }

    @ApiOperation(value = "Delete individual's event")
    @DeleteMapping("{eventId}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteEvent(@PathVariable String familyId,
                            @PathVariable String individualId,
                            @PathVariable String eventId) {
        eventService.deleteEvent(familyId, individualId, eventId);
    }
}
```

TreeController.java

```
package com.spadok.controller;

import com.spadok.service.dto.tree.TreeData;
import com.spadok.service.service.TreeService;
import io.swagger.annotations.ApiOperation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequiredArgsConstructor
@RequestMapping("api/tree/{familyId}")
public class TreeController {
```

```
    private final TreeService treeService;

    @ApiOperation(value = "Send tree")
    @GetMapping("/display")
    @ResponseStatus(HttpStatus.OK)
    public TreeData getFamilyTree(@PathVariable String familyId) {
        return treeService.getFamilyTree(familyId);
    }
}
```

Приклад класів типу виключення

ApiError.java

```
package com.spadok.exceptions;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import org.springframework.http.HttpStatus;
import java.time.LocalDateTime;

@Data
@AllArgsConstructor
@Getter
public class ApiError {

    private HttpStatus status;
    private String message;
    private LocalDateTime timestamp;

    public ApiError(HttpStatus httpStatus, String message) {
        this(httpStatus, message, LocalDateTime.now());
    }
}
```

CustomRestExceptionHandler.java

```
package com.spadok.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.mail.MailSendException;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import javax.persistence.EntityExistsException;
import javax.persistence.EntityNotFoundException;
import java.time.LocalDateTime;

@ControllerAdvice
public class CustomRestExceptionHandler {

   @ExceptionHandler(EntityNotFoundException.class)
   public ResponseEntity<ApiError> entityNotFoundHandler(EntityNotFoundException ex) {
      ApiError apiError = new ApiError(HttpStatus.NOT_FOUND, ex.getMessage(), LocalDateTime.now());
      return new ResponseEntity<>(apiError, apiError.getStatus());
   }

   @ExceptionHandler(EntityExistsException.class)
   public ResponseEntity<ApiError> entityExistsHandler(EntityExistsException ex) {
      ApiError apiError = new ApiError(HttpStatus.CONFLICT, ex.getMessage(), LocalDateTime.now());
      return new ResponseEntity<>(apiError, apiError.getStatus());
   }

   @ExceptionHandler(MethodArgumentNotValidException.class)
   public ResponseEntity<ApiError> methodArgumentNotValidHandler(MethodArgumentNotValidException ex) {
      ValidationError validationError = new ValidationError(ex.getBindingResult().getFieldErrors());
      return new ResponseEntity<>(validationError, validationError.getStatus());
   }

   @ExceptionHandler(MailSendException.class)
```

```
    public ResponseEntity<ApiError> mailSendErrorHandler(MailSendException ex) {
        ApiError apiError = new ApiError(HttpStatus.SERVICE_UNAVAILABLE, ex.getMessage(),
LocalDateTime.now());
        return new ResponseEntity<>(apiError, apiError.getStatus());
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<ApiError> birthDateError(IllegalArgumentException ex) {
        ApiError apiError = new ApiError(HttpStatus.CONFLICT, ex.getMessage(), LocalDateTime.now());
        return new ResponseEntity<>(apiError, apiError.getStatus());
    }
}
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка веб-сервісу для створення родинних дерев та вивчення родоводу»

студента групи 122-17-3 Морара Микити Сергійовича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна_робота_Морар.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна_робота_Морар.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Морар.zip	Архів. Містить коди програми і скомпільовану програму.
Презентація	
Морар.pptx	Презентація кваліфікаційної роботи.