МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ОВЧАР МИХАЙЛО ІВАНОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

ПРОЄКТУВАННЯ КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ ЦИФРОВОГО МЕМОРІАЛУ ГЕРОЇВ ВІННИЦЬКОЇ ОТГ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Петро НІКОЛЮК, професор кафедри
інформаційних технологій,
д-р фіз.-мат. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Овчар Михайло Іванович. Проєктування користувацького інтерфейсу цифрового меморіалу героїв Вінницької ОТГ. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі досліджено основні етапи та поняття необхідні для проєктування та створення користувацької вебплатформи та розробки вебсайту на коді. Сформульовано ключові функціональні та нефункціональні вимоги до вебплатформи.

За допомогою HTML, CSS, мови програмування JavaScript та фреймворку SvelteKit було створено загальнодоступну вебплатформу для пошуку полеглих героїв Вінницької ОТГ

Ключові слова: вебплатформа, веброзробка, SCSS, JavaScript, HTML, Svelte, SvelteKit, Rest API, фронтенд __ сторінки, __ рис., __ табл., __ джерела.

ABSTRACT

Mykhailo Ivanovych Ovchar. User interface designing of the Vinnytsia OTG heroes digital memorial. Specialisation 122 'Computer Science', degree programme 'Computer Science'. Vasyl Stus Donetsk National University, Vinnytsia, 2026.

This undergraduate thesis examines the main stages and concepts necessary for the design and creation of a user web platform and the development of a website using code. Key functional and non-functional requirements for the web platform have been formulated.

Using HTML, CSS, the JavaScript programming language and the SvelteKit framework, a publicly accessible web platform was created to search for fallen heroes of the Vinnytsia Territorial Community

Keywords: web platform, web development, SCSS, JavaScript, HTML, Svelte, SvelteKit, REST API, front-end __ pages, __ figs., __ tables, __ sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЗСУ – Збройні Сили України

URL – Uniform Resource Locator (уніфікований покажчик ресурсу, стандартизована адреса ресурсу в інтернеті)

ПІБ – Прізвище, Ім’я, По батькові.

API – Application Programming Interface (інтерфейс прикладного програмування).

REST – Representational State Transfer.

HTML – HyperText Markup Language (мова гіпертекстової розмітки).

AJAX – Asynchronous JavaScript and XML.

IP – Internet Protocol.

SSR – Server-Side Rendering.

CSR – Client-Side Rendering.

SPA – Single Page Applications.

SEO – Search Engine Optimization.

ВСТУП

Актуальність теми дослідження: Триваюча агресія росії проти України зумовлює необхідність гідного вшанування пам’яті захисників, які віддали життя за незалежність держави. У сучасних умовах цифровізації традиційні форми меморіалізації доповнюються онлайн-платформами, які дозволяють зберігати та поширювати історію подвигу героїв поза межами фізичних пам’ятників. Для Вінницької ОТГ створення єдиного цифрового меморіалу є критично важливим завданням, оскільки існуючі рішення часто є фрагментарними, закритими або обмеженими у функціональності. Більшість діючих ресурсів не забезпечують зручного пошуку, не містять вичерпної інформації про звання, бойовий шлях та місця поховання героїв, що ускладнює процес комунікації пам’яті. Розробка сучасного вебінтерфейсу з використанням прогресивних фреймворків, таких як SvelteKit, дозволяє створити швидкий, інтуїтивно зрозумілий та емоційно стриманий цифровий простір, що відповідає запитам як сімей загиблих, так і широкої громадськості в Україні та за кордоном.

Метою кваліфікаційної роботи є проєктування користувацького інтерфейсу та розробка фронтенд-частини вебплатформи «Цифровий меморіал героїв Вінницької ОТГ», що забезпечує ефективний пошук, фільтрацію та гідне представлення персональних даних захисників.

Для досягнення поставленої мети було визначено такі завдання:

1. Проаналізувати сучасні підходи до створення цифрових меморіалів та визначити специфіку UX-дизайну для платформ вшанування пам’яті.
2. Дослідити існуючі цифрові рішення для територіальних громад, виявити їх переваги та недоліки в контексті представлення інформації про героїв.
3. Сформулювати функціональні вимоги до платформи, базуючись на запитах Ради родин загиблих Вінницької ОТГ.
4. Обґрунтувати вибір технологічного стека (SvelteKit) для реалізації інтерфейсу з динамічним підвантаженням даних.
5. Розробити дизайн-концепцію інтерфейсу, що поєднує зручність використання з відповідним емоційним контекстом.
6. Реалізувати фронтенд-частину платформи, зокрема систему динамічної фільтрації та пошуку героїв «на ходу» (AJAX-взаємодія).
7. Створити адаптивні шаблони для детального відображення особистої інформації героїв (фото, біографія, нагороди, місця поховання).
8. Провести тестування інтерфейсу на відповідність вимогам інклюзивності та швидкодії.

Об’єктом дослідження є процес розробки та проектування користувацького інтерфейсу вебплатформ для вшанування пам’яті засобами сучасних фронтенд-технологій.

Предметом дослідження є вебплатформа цифрового меморіалу героїв Вінницької ОТГ, її архітектура та функціональні блоки, зокрема інтерфейсні рішення для відображення, пошуку та динамічної фільтрації персональних даних захисників.

Практичне значення одержаних результатів: полягає у створенні функціонуючого фронтенд-рішення для цифрового меморіалу, яке було спроектоване у співпраці з представниками громадськості. Розроблена платформа дозволяє систематизувати та відображати дані про героїв громади та надає зручний інструмент для збереження національної пам’яті, доступний користувачам з будь-якої точки світу.

Апробація результатів роботи: основні результати роботи апробації не проходили.

Структура та обсяг роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел (31 найменування) та додатків (2 додатки). Робота містить 18 рисунків та 4 таблиці. Основний зміст складається з 48 сторінок. У першому розділі проведено аналіз предметної області та UX-вимог до меморіальних ресурсів. У другому розділі описано процес проєктування інтерфейсу та обґрунтовано вибір SvelteKit. У третьому розділі представлено програмну реалізацію фронтенд-частини, опис роботи фільтрації даних та результати тестування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Роль цифрових меморіалів у сучасному інформаційному просторі

Вшанування пам’яті захисників завжди був важливим елементом будь-якої національної ідентичності, дозволяючи зберігати передавати інформацію про подвиги через покоління. В нашу сучасну цифрову епоху, ці процеси зазнають докорінних трансформацій, переходячи від статичних монументальних форм до динамічних мережевих структур.

Цифрова меморіалізація визначається як сукупність практик збереження та відтворення пам’яті про минуле за допомогою цифрових технологій, що дозволяє створювати віртуальні простори для скорботи, навчання та національного єднання [1]. У контексті сучасного розвитку суспільства онлайн-меморіали стають не просто електронними архівами, а активними суб’єктами формування колективної ідентичності.

Ключовими аспектами, які демонструють важливість сучасних цифрових меморіалів є:

- Подолання просторових та часових обмежень. На відміну від фізичних пам’ятників, доступ до яких обмежений географічно, цифрові платформи забезпечують глобальну присутність, що дозволяє вшанувати пам’ять героїв представникам діаспори, біженцям та військовим на передовій.
- Персоналізація та гуманізація пам’яті. Традиційні меморіали часто зосереджені на загальних образах захисників, тоді як цифрові ресурси дають можливість представити кожного героя як особистість через публікацію біографій, цитат, спогадів побратимів та сімейних фотоархівів.
- Інтерактивність та залучення. Сучасні інтерфейси дозволяють користувачам не лише споживати інформацію, а й взаємодіяти з нею: здійснювати складний пошук, фільтрувати дані за родами військ чи громадами, а також інтегруватися з фізичним простором за допомогою QR-кодів.
- Навчальна та просвітницька функція. Онлайн-платформи стають базою для вивчення сучасної історії України, виховання патріотизму та формування поваги до ветеранів серед молоді, яка є найбільш активною аудиторією цифрових ресурсів.

Окрім того, проєктування інтерфейсів для цифрових меморіалів вимагає глибокого розуміння емоційного стану відвідувачів, який часто характеризується скорботою, підвищеною вразливістю або когнітивним перевантаженням через стрес. На відміну від комерційних вебресурсів, де метою є стимуляція активності, меморіальна платформа має створювати відчуття спокою, поваги та безпеки.

Ключовими принципами емоційного дизайну для таких систем є:

- Когнітивна простота, коли користувачі в стані стресу мають обмежені ресурси уваги. Будь-яка інтерфейсна складність може викликати роздратування. Тому інтерфейс цифрового меморіалу Вінницької ОТГ базується на принципі мінімалізму.
- Вибір стриманої кольорової гами, нейтральних тонів допомагає встановити відповідний емоційний тон, що пам’ятному і вічному тону.
- Важливо, щоб користувач відчував повний контроль над процесом пошуку та перегляду інформації. Відсутність неочікуваних звукових ефектів або автоматичних переходів є обов’язковою вимогою для збереження психологічного комфорту.

Для наочності порівняння підходів до дизайну, нижче наведено таблицю відмінностей між стандартним та меморіальним UX (Таблиця 1.1).

Таблиця 1.1 Порівняльний аналіз принципів проєктування стандартних та меморіальних інтерфейсів

Критерій порівняння	Стандартний UX	Меморіальний UX
Головна мета	Конверсія, утримання уваги.	Вшанування пам'яті, спокій.
Колірна гама	Яскраві, акцентні кольори.	Стримані, приглушені тони.
Анімація	Динамічна, привертає увагу.	Плавна, ледь помітна або відсутня.
Навігація	Багато розгалужень, рекомендації.	Пряма, лінійна, максимально проста.
Емоційний відгук	Збудження, цікавість	Повага, скорбота, рефлексія

Такий підхід дозволяє перетворити вебресурс із простої бази даних на інклюзивне цифрове середовище, що сприяє соціальній реабілітації родин загиблих та зміцненню колективної пам'яті громади

Вплив цифрових технологій на культуру пам’яті полягає у зміні основного способу представлення минулого. Замість монолітності і незмінності, пам’ять стає «текучою» та інформація постійно оновлюється та доповнюється в режимі реального часу [2]. Також, цифрові меморіали можуть інтегруватися з фізичним простором, доповнюючи один одного (Рисунок 1.1). Для Вінницької ОТГ створення такого ресурсу є критично важливим, щоб забезпечити гідне представлення героїв-земляків, які загинули в триваючій війні, всьому світовому суспільству.

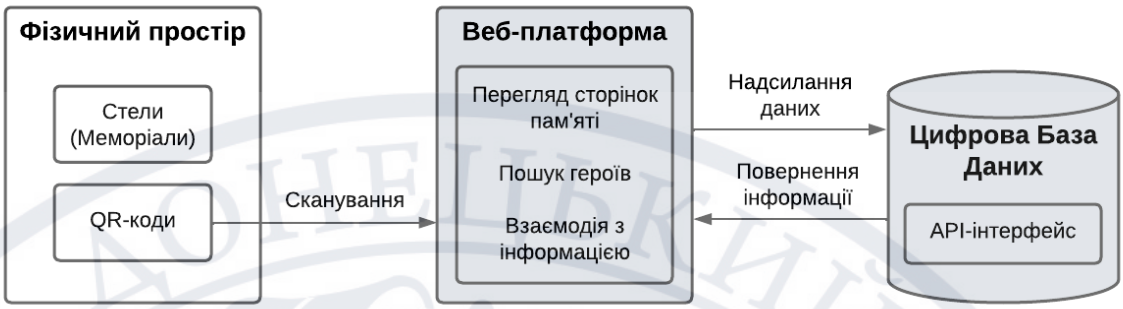

Рисунок 1.1 – Концептуальна схема екосистеми цифрової пам'яті.

Особливе значення цифрові меморіали мають для соціальної реабілітації родин загиблих. Публічне визнання подвигу в офіційному цифровому реєстрі сприймається як акт соціальної справедливості та глибокої вдячності від суспільства. Технологічна реалізація таких платформ потребує високої якості проектування інтерфейсів, оскільки емоційний стан користувачів вимагає стриманого, але водночас функціонального та безбар'єрного дизайну [3].

Отже, цифрові меморіали виконують роль «живої пам'яті», яка інтегрована в щоденне життя громадянина через мобільні пристрої та комп'ютерні мережі. Це перетворює вшанування пам'яті з епізодичної події на безперервний процес національної рефлексії.

1.2 Огляд та критичний аналіз існуючих систем вшанування пам'яті

Цифрова трансформація соціальної сфери в Україні, зумовлена викликами воєнного часу, призвела до появи значної кількості платформ для вшанування пам'яті [4]. Оскільки переважна більшість українців є активними користувачами мережі, вебплатформа стає основним інструментом збереження історичної пам'яті [5]. Аналіз конкурентних рішень є обов'язковим етапом проектування для виявлення кращих практик та уникнення типових UX-помилок.

На сьогодні в Україні функціонують як загальнонаціональні проекти, так і локальні меморіали окремих громад. Для аналізу були обрані найбільш відповідні ресурси:

1. *Електронний «Пантеон героїв» Дрогобицької громади*. Використовує темну колірну гаму, що відповідає траурному контексту. Система забезпечує базову функціональність, але має обмежені можливості для масштабування [6].
2. *«Книга пам'яті полеглих за Україну»*. Ресурс виконаний в світлій темі та демонструє список полеглих героїв з усієї країни, починаючи з 2014 року. Система має сучасний дизайн і стабільно працює, але має великі недоліки із наповненням контенту, через обмеженість джерел інформації [7].
3. *Електронний Меморіал Броварської ОТГ*. Базове рішення, що фокусується на деталізації військових здобутків та нагород героїв [8].

Порівняльний аналіз функціональних можливостей цих систем наведено в таблиці 1.2.

Таблиця 1.2 Порівняльний аналіз існуючих цифрових меморіалів

Критерій порівняння	«Пантеон героїв» Дрогобицької громади	«Книга пам'яті полеглих за Україну»	Меморіал Броварської ОТГ
Рівень UX/UI	Середній. Лаконічний.	Високий. Сучасний.	Низький. Базовий.
Фільтрації	Фільтрація за населеним пунктом, військової частиною, ПІБ та роком загибелі.	Фільтрація за широким переліком із 9-ти категорій.	Пошук героя за ПІБ.
Адаптивність	Повна мобільна адаптація.	Повна мобільна та міжплатформна адаптація.	Поверхнева мобільна адаптація.
Швидкість пошуку	Повільна. При перезапуску сторінки результат пошуку втрачається.	Середня. Збереження результатів після перезапуску сторінки.	Висока швидкість пошуку на статичних результатах.

Попри важливу соціальну місію, більшість проаналізованих систем мають низку технічних та інтерфейсних проблем, які заважають ефективній взаємодії з користувачем:

- *Обмеженість функціоналу пошуку*: У багатьох проектах реалізовано лише базовий пошук, що змушує відвідувачів вручну переглядати великі списки для знаходження конкретної особи.
- *Інтерфейсна перевантаженість*: Відсутність чіткої візуальної ієрархії ускладнює сприйняття великих обсягів інформації [9].
- *Низька адаптивність*: Сервіси мають низьку мобільну адаптивність, що робить їх незручними для перегляду на мобільних пристроях та планшетах, які є основними пристроями більшості людей для використання Інтернет сервісів.
- *Технічна ненадійність:* Використання застарілих CMS або перевантажених міських порталів призводить до частої недоступності меморіальних сторінок [5].

На основі проведеного огляду встановлено, що проектований цифровий меморіал для Вінницької ОТГ має базуватися на принципах високої продуктивності та доступності. Використання реактивних фреймворків

(SvelteKit) дозволить забезпечити миттєву фільтрацію даних та плавність інтерфейсу, що є слабким місцем більшості поточних рішень [10].

1.3 Формування функціональних вимог до фронтенд-частини платформи

На основі аналізу потреб громади та технічного завдання, функціональні вимоги до клієнтської частини цифрового меморіалу фокусуються на забезпеченні швидкого, інтуїтивно зрозумілого та інклюзивного доступу до інформації про полеглих героїв. Основна мета фронтенд-рішення – реалізація публічного інтерфейсу, що забезпечує високу швидкість взаємодії за допомогою архітектури SvelteKit.

Ключовим функціональним блоком є інструментарій знаходження та перегляду інформації. Згідно з архітектурою проекту, фронтенд має забезпечувати:

- Гнучкий пошук, який дає змогу виконувати пошук героїв за їх ПІБ або позивним. Це дозволяє користувачам швидко знайти картку захисника за відомими їм ідентифікаторами.
- Багатофакторна фільтрація героїв за 5 основними категоріями: військове звання, рід військ, рік смерть, отриманні нагороди та стать. Дана фільтрація реалізується у вигляді випадаючих списків, що розміщуються у прихованому меню.
- Перемикання режиму відображення, який дозволяє переглядати список або у вигляді безкінечного горизонтальної каруселі, або у вигляді плиток, для масового перегляду і зручнішого використання.
- Сторінка героя, на якій можна переглядати особисту інформацію про героя, включно з його званням, родом військ, роками життя, нагородами, біографією та місцем поховання.

Важливою особливістю розробки є відмова від окремих кнопок керування на користь глибинної архітектурної доступності. Фронтенд має відповідати наступним вимогам:

- *Семантична коректність*: використання тегів HTML5 (наприклад, <article>, <section>, <nav>) для забезпечення правильної інтерпретації структури сайту екранними читцями та пошуковими роботами [9].
- *Підтримка системних налаштувань*: інтерфейс має бути побудований за принципом гнучкої верстки, що коректно реагує на зміну розміру шрифту або контрастності безпосередньо в налаштуваннях браузера користувача.
- *Локалізація*: підтримка багатомовності через бібліотеку перекладів, що дозволяє динамічно змінювати текстовий контент без втрати стану застосунку [10].

Для забезпечення стабільності та чистоти коду, розробка має базуватися на принципах KISS та DRY [11]. Це включає:

- *Серверний рендеринг (SSR)*: використання функцій отримання даних на стороні сервера, перед передачею сторінки користувачу, що критично для SEO-оптимізації меморіальних сторінок.
- *Реактивність*: миттєве оновлення списку героїв при зміні параметрів фільтрації за допомогою Svelte-сторів.

Окрім того, щоб користувацький досвід при взаємодії з меморіалом був якісним, вебплатформа має відповідати суворим нефункціональним вимогам, що фокусуються на швидкодії, доступності та пошуковій оптимізації. Враховуючи використання JavaScript на клієнті, основними технічними орієнтирами є показники Core Web Vitals, серед яких можна виділити наступні метрики продуктивності:

- *Largest Contentful Paint (LCP)*: час завантаження найбільшого видимого елемента не повинен перевищувати 2,5 секунди для 75% запитів.
- *First Input Delay (FID)*: затримка після першої взаємодії користувача з інтерфейсом має бути меншою за 100 мілісекунд.
- *Cumulative Layout Shift (CLS)*: показник візуальної стабільності, що критично для адаптивної вебплатформи. Значення має бути меншим за 0,1, щоб уникнути раптових зсувів контенту при підвантаженні зображень.

Окрему групу вимог складають стандарти доступності. Оскільки цільова аудиторія меморіалу включає людей похилого віку та осіб із порушеннями зору, обов'язковою є відповідність стандарту WCAG 2.1 [12]. Це передбачає:

1. Коефіцієнт контрастності тексту не менше 4,5:1 для звичайного тексту.
2. Працездатність інтерфейсу за допомогою клавіатури.
3. Наявність альтернативного тексту (alt) для всіх медіафайлів.

Також варто наголосити на вимогах до безпеки та конфіденційності, які зумовлені специфікою даних. Система має використовувати протокол HTTPS для захисту даних між клієнтом та REST API. Окрім того, архітектура має передбачати відсутність індексації технічних сторінок, залишаючи доступними для пошукових роботів лише публічні сторінки героїв для кращого SEO.

1.4 Аналіз архітектурних підходів до реалізації клієнт-серверної взаємодії

Реалізація сучасних вебплатформ базується на клієнт-серверній архітектурі, де функціональні обов’язки розподілені між постачальником ресурсів (сервером) та споживачем (клієнтом). У контексті проектування цифрового меморіалу вибір архітектурного підходу визначає не лише швидкість завантаження даних, а й зручність підтримки коду та масштабованість системи.

Основним питанням при побудові фронтенд-частини є вибір між стратегіями формування HTML-коду. У сучасній практиці виділяють три базові підходи:

- Client-Side Rendering (CSR), де весь процес формування інтерфейсу відбувається в браузері користувача за допомогою JavaScript. Це забезпечує плавні переходи між сторінками без їх перезавантаження, проте створює

значне навантаження на пристрій клієнта та ускладнює індексацію контенту пошуковими системами [13].

- Server-Side Rendering (SSR), при якому сервер генерує повну HTML-сторінку для кожного запиту (Рисунок 1.2). Це гарантує швидке відображення першого контенту та високу пошукову оптимізацію, що є критичним для меморіальних сторінок, які мають легко знаходитися через пошуковики.

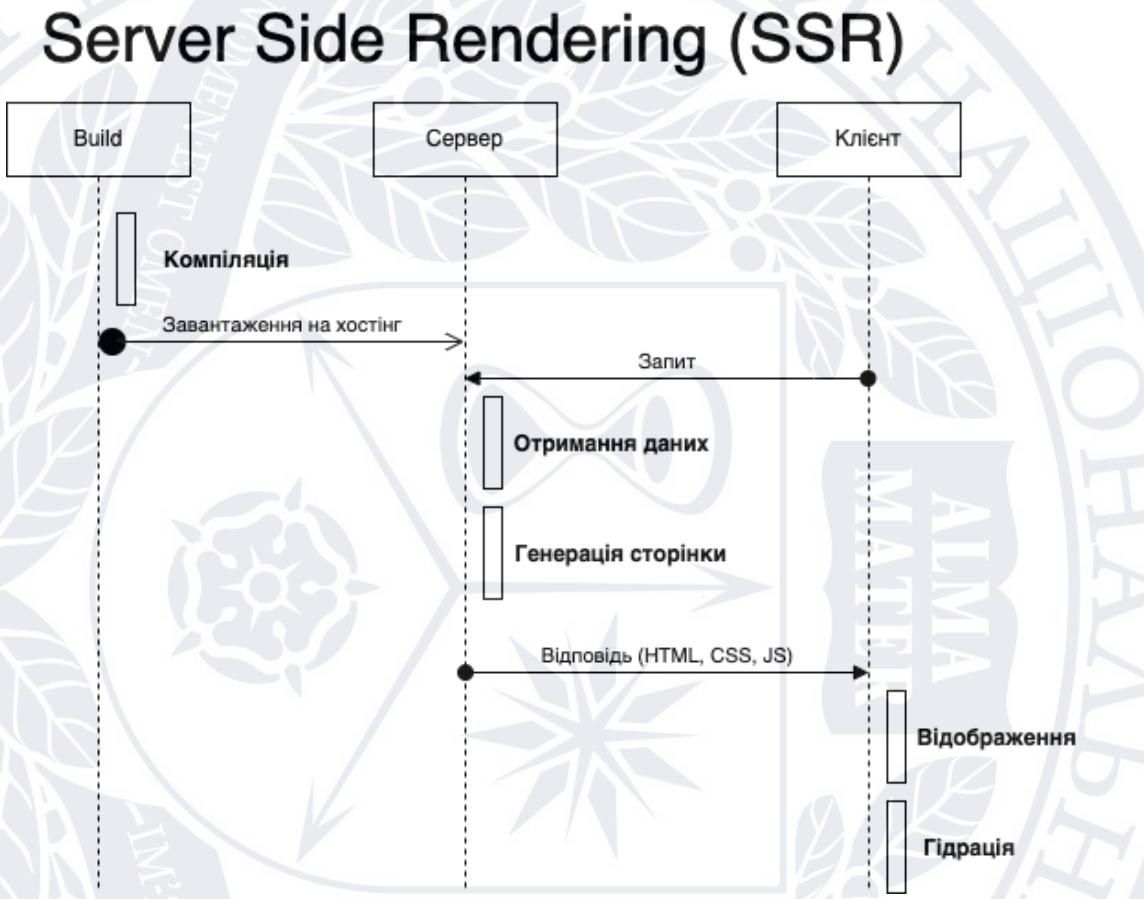

Рисунок 1.2 – Схема життєвого циклу запиту при SSR підході до формування HTML коду сторінки

- Гібридна модель, в якій поєднує переваги обох підходів. Сервер віддає попередньо відрендерений HTML, який згодом «оживає» (гідратується) на клієнті, перетворюючись на повноцінний інтерактивний застосунок. Саме цей підхід є фундаментальним для сучасних фреймворків [10].

Так як розробка вебплатформи відбувається у співпраці з Вінницькою ОТГ, то громада надає базу даних героїв, а також допомагає з розробкою бекенд частини платформи за допомогою ASP.NET. Для забезпечення зв'язку між фронтендом на SvelteKit та бекендом на ASP.NET доцільно використовувати архітектурний стиль REST. Основними принципами цього підходу є:

- *Безстановість (Stateless)*: сервер не зберігає контекст клієнта між запитами, що спрощує масштабування.

- *Уніфікований інтерфейс*: використання стандартних HTTP-методів (GET, POST, PUT, DELETE) для маніпуляції ресурсами [13].
- *Обмін даними у форматі JSON*: легковажний формат, який легко парситься як на стороні сервера, так і в середовищі JavaScript [14].

Важливою вимогою до інтерфейсу цифрового меморіалу є динамічне оновлення списків героїв без повного оновлення сторінки. Це реалізується за допомогою технології AJAX або сучасного Fetch API. Такий підхід дозволяє виконувати фонові запити до API, оновлюючи лише окремі компоненти інтерфейсу, наприклад, результати фільтрації або пагінації (Рисунок 1.3).

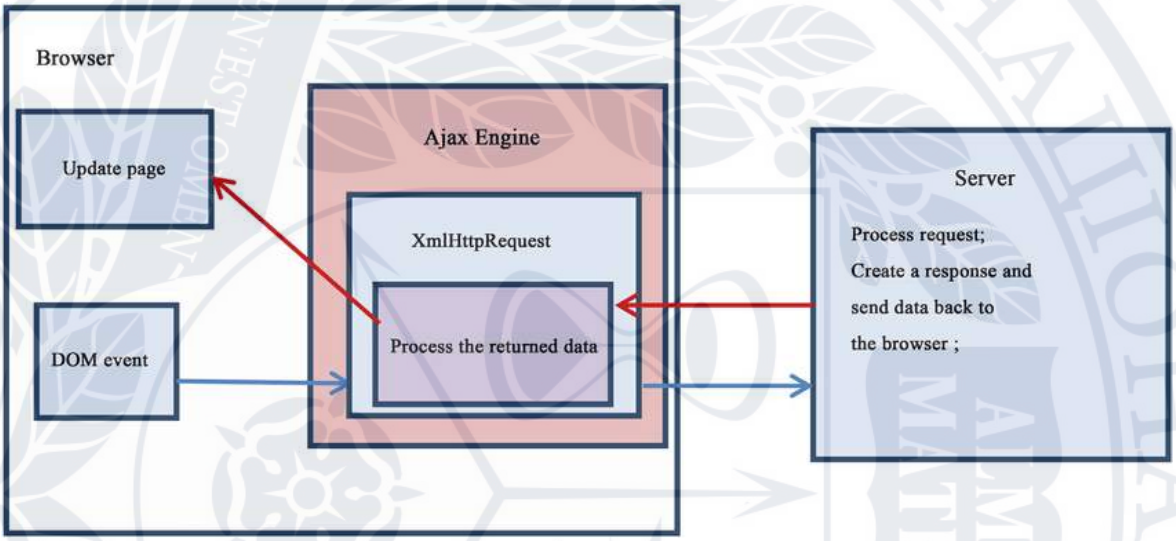

Рисунок 1.3 – Базовий принцип роботи AJAX технології.

Використання комбінації гібридного рендерингу та REST-архітектури дозволяє створити відмовостійку систему, яка однаково ефективно працює як на стаціонарних комп'ютерах, так і на мобільних пристроях з обмеженою швидкістю інтернет-з'єднання.

Висновок до розділу

В цьому розділі було проведене комплексне дослідження теоретичних основ та історичних практик створення цифрових меморіалів, що дозволило сформувати фундамент для подальшої розробки клієнтської частини платформи для Вінницької ОТГ.

Дослідження існуючих цифрових меморіалів виявило недоліки систем, які зумовили до необхідності створення власної стабільної, продуктивної та інклюзивної платформи. В результаті були сформовані вимоги до фронтенд-частини акцентуючи увагу на гнучкій фільтрації даних та семантичній коректності архітектури.

Також, було наголошено на використання REST-архітектури разом із гібридним рендерингом визначено як найбільш ефективний підхід для реалізації сучасної клієнт-серверної взаємодії.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКА

2.1 Вибір фреймворку SvelteKit для реалізації високопродуктивного фронтенду

Сучасний підхід до створення вебзастосунків та вебплатформ ґрунтується на використані новітніх JavaScript фреймворків, які представляють універсальний каркас, що пришвидшує та полегшує процес розробки та подальшої підтримку проєкту. Станом на сьогодні, найпопулярнішими JavaScript фреймворками для розробки повноцінних вебплатформ є: React, Next.js, Vue.js, Nux.js, Angular та SvelteKit.

Для вибору інструментарію для розробки інтерфейсу цифрового меморіалу орієнтувалися на два основні критерії: висока швидкість рендерингу та легкість підтримки коду. На відміну від традиційних інтерпретованих фреймворків, SvelteKit переносить основну обчислювальну логіку на етап компіляції, що дозволяє мінімізувати обсяг JavaScript-коду, який передається клієнту, що дозволяє пришвидшити завантаження сторінки [15]. Окрім того, SvelteKit дозволяє реалізовувати весь компонентний код на основі базового синтаксису JavaScript, що полегшує процес розробки та підтримки навіть для тих, хто не має досвіду з роботи з даним фреймворком.

Вибір SvelteKit зумовлений необхідністю досягнення максимальної продуктивності на малопотужних мобільних пристроях, що часто використовуються жителями громад для пошуку інформації «на ходу». На відміну від React або Vue, які використовують концепцію Virtual DOM, SvelteKit переносить основну роботу на етап компіляції.

Основні відмінності архітектурного підходу:

- *Відсутність Virtual DOM*: У традиційних фреймворках під час оновлення стану система порівнює віртуальне дерево з реальним, що потребує значних ресурсів браузера. SvelteKit генерує чистий JavaScript-код, який точково маніпулює DOM-вузлами.
- *Розмір бандлу*: Оскільки SvelteKit не потребує завантаження важкої бібліотеки рантайму, обсяг переданих даних є мінімальним, що критично для користувачів із нестабільним зв'язком.
- *Реактивність*: SvelteKit базується на графі залежностей. Якщо користувач змінює параметр у фільтрах, оновлюється лише конкретний вузол відображення списку героїв, ігноруючи решту елементів сторінки.

Такий підхід дозволяє реалізувати складні компоненти, такі як SoldierSlider, без втрати плавності роботи сайту навіть при великій кількості анімованих карток героїв. Це підтверджує доцільність використання SvelteKit для ресурсів із високими вимогами до інклюзивності, а також дозволяє досягти наступних покращень:

- Збільшення мінімального часу до інтерактивності (TTI) завдяки відсутності важкої бібліотеки запуску платформи, через що додаток завантажується швидше, що критично для користувачів із нестабільним зв'язком, і використовує менше ресурсів пристрою, що робить його поведінку більш стабільною на малопотужних пристроях, таких як смартфони [10].
- Наявність механізму ефективної реактивності, в якому оновлення стану базується на топологічному сортуванні залежностей, що гарантує відсутність зайвих рендерерів компонентів і мінімізує потребу в створенні Virtual DOM, для менеджменту компонентів сторінки.

Використання SvelteKit дозволяє реалізувати не лише набір функціональних компонентів сторінки, а й чітку структуру проекту, що підтверджується файловою системою розробленого меморіалу (Рисунок 2.1). Архітектура базується на файловому роутингу, де логіка розділена між клієнтом та сервером, в окремих файлах:

- *Серверний рівень*, на якому розміщуються файли +page.server.js та +layout.server.js, які містять код для виконання на сервері проєкту, перед передачею результату користувачу. Саме тут реалізовується безпечна взаємодія з REST API, обробка кукі для локалізації та попереднє завантаження даних.
- *Клієнтський рівень*, на якому розміщуються файли +page.svelte та +layout.svelte. Даний рівень відповідає за реактивне відображення інтерфейсу, управління станом фільтрації та анімації [16].

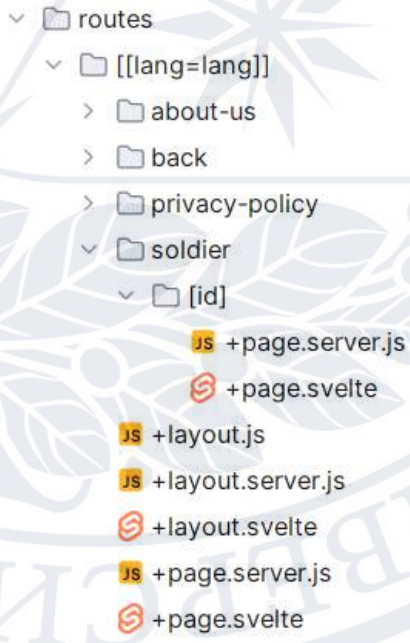

Рисунок 2.1 – Ієрархічна структура компонентів та маршрутів у SvelteKit.

В основі роботи обраного фреймворку лежить математична модель графа залежностей. Якщо x – стан системи (наприклад, обраний рік у фільтрі), а

f(x) – функція відображення списку героїв, то Svelte гарантує оновлення вузла f(x) лише при зміні x, ігноруючи інші параметри графа. Це дозволяє оптимізувати роботу зі списками даних великого обсягу без використання складних алгоритмів мемоїзації, які характерні для конкурентних рішень.

Застосування принципів чистого коду та модульності (DRY/KISS) при побудові компонентів дозволяє створити систему, яка легко масштабується при додаванні нових функціональних блоків, таких як розширена біографія або мультимедійні архіви [11].

2.2 Проєктування системи маршрутизації та локалізації

У процесі розробки інтерфейсу цифрового меморіалу критично важливим є створення гнучкої системи навігації та підтримки багатомовності. Це дозволяє забезпечити сталий доступ до інформації для різних категорій користувачів, зокрема міжнародної спільноти, та зберігати стан інтерфейсу при поширенні посилань [17].

Для реалізації навігації у проєкті використано підхід файлової маршрутизації SvelteKit. Основною особливістю є використання необов'язкового параметру мови у структурі папок: routes/[[lang=lang]]. Така архітектура дозволяє системі коректно обробляти запити як для основної мови (української) за адресою /, так і для англомовної версії за префіксом /en/. Окрім того, створена структура гнучка і може використовуватися для будь-якої кількості мови.

У межах проєкту спроєктовано наступну ієрархію маршрутів (Рисунок 2.2):

- *Головна сторінка (/)*: центральний вузол та точка входу в платформу. На цій сторінці відображається список героїв із компонентами фільтрації та пошуку.
- *Динамічний маршрут героя (/soldier/[id])*: сторінка детальної інформації про захисника, де id виступає унікальним ідентифікатором для отримання даних за допомогою API.
- *Інформаційні сторінки (/about-us, /privacy-policy)*: статичні розділи, що описують мету проєкту та умови обробки персональних даних.

Рисунок 2.2 – Схема деревоподібної структури маршрутів проєкту.

Для збереження стану фільтрації (звання, рід військ, рік тощо) без виконання зайвих запитів до сервера та забезпечення можливості «ділитися»

посиланням, використано об'єкт URLSearchParams, який є базовим елементом JavaScript WebAPI. Це дозволяє синхронізувати стан інтерфейсу з адресним рядком браузера, що є стандартом для сучасних вебзастосунків із великими масивами даних.

Локалізація застосунку реалізована за гібридною моделлю. Статичні елементи інтерфейсу (заголовки, кнопки, підписи) перекладаються на стороні клієнта за допомогою бібліотеки sveltekit-i18n та локальних конфігураційних файлів JSON. Динамічний контент (біографії, назви нагород) передається з бекенду за допомогою API вже у необхідній мовній версії залежно від параметрів запиту [10].

Алгоритм визначення поточної локалі при ініціалізації запиту працює за пріоритетною схемою, а саме:

1. *Аналіз URL*: перевірка наявності префікса конкретної мови у маршруті (для англійської мови це /en/). Якщо знайдено – встановлюється мова яка відповідаю даному префіксу.
2. *Перевірка Cookies*: якщо в URL мова не вказана, система звертається до файлів кукі (параметр lang), де зберігається попередній вибір користувача.
3. *Визначення мови за допомогою IP-адреси користувача*: у разі відсутності попередніх даних, система визначає країну перебування користувача, на основі IP-адреси, та встановлює відповідну мову.
4. *Default*: встановлення української мови як базової.

Для технічного забезпечення цього процесу у файлі translations.js налаштовані завантажувачі, які асинхронно підвантажують словники текстів лише для активної локалі, що мінімізує обсяг трафіку на першому етапі завантаження. Такий підхід гарантує, що архітектура вебсторінки залишається семантично коректною та доступною для пошукових роботів обома мовами

2.3 Моделювання потоків даних між клієнтом, сервером SvelteKit та зовнішнім API

Проєктування потоків даних у межах цифрового меморіалу базується на триланковій архітектурі, де сервер SvelteKit виступає інтелектуальним посередником між клієнтським браузером та зовнішнім REST API. Такий підхід дозволяє розмежувати логіку обробки даних, забезпечити безпеку запитів та реалізувати складні сценарії відображення контенту.

Архітектура взаємодії та життєвий цикл запиту розділені на кілька логічних етапів, що забезпечують баланс між швидкістю завантаження та актуальністю даних. Першим етапом є SSR, який запускається при первинному запиті користувача. Головна функція load файлу +page.server.js ініціює запит до API. Серверний код SvelteKit виконує роль клієнта для API, отримуючи «сирий» JSON-об'єкт. Це дозволяє приховати можливу логіку авторизації та структуру внутрішнього API від кінцевого користувача, що збільшує безпеку внутрішнього API та зменшує ризики втручання ззовні. Для реалізації цього механізму

створено метод apiQueryBack, який базується на бібліотеці Axios. Метод автоматично визначає базову адресу сервера (API_LINK), додає параметри локалізації (lang) та здійснює передачу токенів автентифікації з куків (user_token), що забезпечує безпеку з'єднання на рівні «сервер-сервер» [18].

Далі, отримані дані проходять через шар трансформації перед рендерингом. Це необхідно для приведення типів та форматування даних згідно з вимогами UI. У проєкті реалізовано наступні методи:

- *getFormatedDate*: приймає системну мітку часу та конвертує її у локалізований формат «дд.мм.рррр». Це уніфікує відображення дат народження та смерті героїв.
- *getCorpsCode*: виконує мапінг текстових назв родів військ (наприклад, «гвардія» чи «морські») у стандартизовані коди («national-guard», «navy»). Це дозволяє динамічно підставляти відповідні стилі та графічні елементи в інтерфейсі.
- *marked*: бібліотека для перетворення біографічних описів з формату Markdown у семантичний HTML, що дозволяє адміністраторам гнучко оформлювати тексти без втручання в код фронтенду [19].

Схему початкового запиту користувача, отримання даних, їх перетворення та повернення можна спостерігати на Рисунку 2.3.

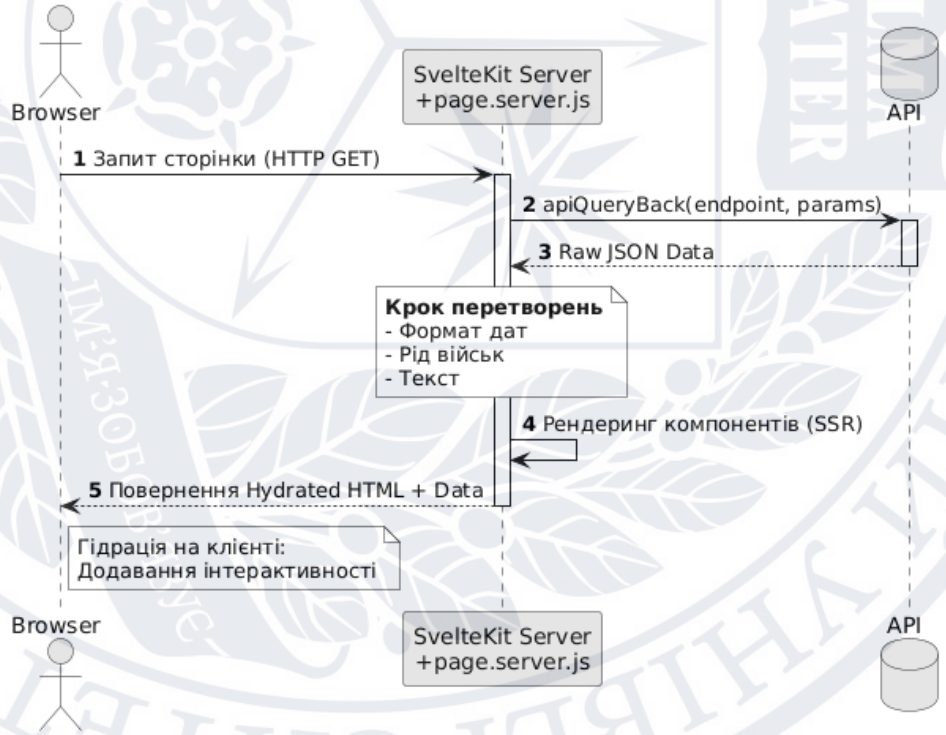

Рисунок 2.3 – Діаграма послідовності потоків даних проєкту.

Після підготовки і візуалізації даних в нашому UI, важливим етапом є забезпечення реактивного оновлення даних після фільтрації. При зміні

користувачем параметрів фільтрації у компоненті FilterBar потік даних обробляється наступним чином:

1. *Оновлення URL посилання*: система використовує об'єкт URLSearchParams для оновлення адреси без перезавантаження сторінки. Даний механізм потрібен для збереження стану фільтрів і можливості ділитися відфільтрованими сторінками між користувачами.
2. *Примусове оновлення даних*: після застосування фільтрів і зміни URL посилання, викликається метод invalidateAll, який змушує SvelteKit повторно запустити серверний load. Це гарантує, що дані в об’єкті data на сторінці завжди відповідають актуальним параметрам в URL.
3. *Оновлення інтерфейсу*: відслідковуючи основні перевірочні змінні відбувається примусове перестворення ключових компонентів сторінки, таких як список героїв і меню фільтрації, що необхідно для коректного скидання станів анімацій та запобігання візуальним артефактам при зміні вибірки.

Після обробки фільтрації користувача, система продовжує свою звичну роботу, допоки користувач не змінить параметри фільтрації сторінки знову. Даний процес візуалізований на Рисунку 2.4.

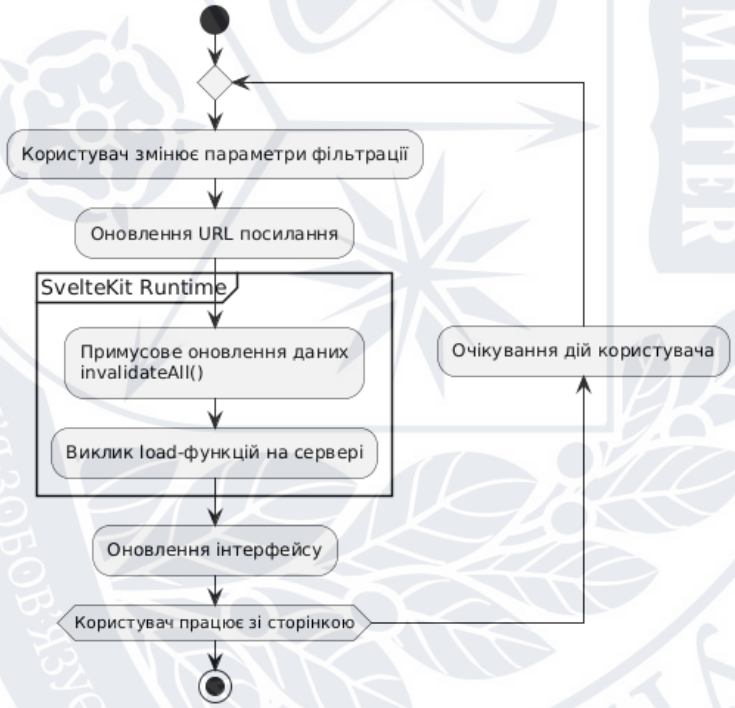

Рисунок 2.4 – Схема механізму оновлення контенту сторінки після застосування фільтрації.

Останнім, але від того не менш важливим етапом життєвого циклу платформи є асинхронна пагінація. Вона реалізована для безперервного перегляду списків реалізовано, виконуючи «невидиму» пагінації, яка додає нові елементи на сторінку без її перезавантаження. Клієнтська функція

backQueryFront надсилає асинхронні запити безпосередньо до API. Після чого порція даних проходить аналогічну трансформацію на стороні клієнта та додається до існуючого масиву героїв, ініціюючи реактивне оновлення DOM-дерева. Застосування принципу асинхронних запитів, а не примусового оновлення даних як у механізмі фільтрації, обумовлена фактом того, що примусове оновлення даних вимагає оновлення інтерфейсу, що може змусити інтерфейс виглядати поломаним під час пагінації.

2.4 Реалізація стратегії пошукової оптимізації засобами серверного рендерингу

Однією з пріоритетних вимог до цифрового меморіалу є забезпечення максимальної видимості сторінок героїв у глобальних пошукових системах. Для проєктів такого типу стандартні SPA, що базуються виключно на клієнтському рендерингу, мають суттєвий недолік: пошукові роботи часто отримують порожню HTML-структуру, поки JavaScript не завантажить дані з API . У межах розробки було впроваджено стратегію SSR через фреймворк SvelteKit, що дозволяє поєднати переваги швидкодії та якісної індексації.

Програмна реалізація цієї стратегії базується на трьох основних технологічних рівнях:

- Серверна агрегація даних, за допомогою асинхронної функції load у файлі +page.server.js, де сервер ініціює запити до REST API та формує повний HTML-код сторінки до її передачі користувачу . Це гарантує, що пошукові боти отримують доступ до ПІБ, біографії та нагород захисника при першому зверненні [20].
- Використання семантичної розмітки Schema.org, для того щоб пошукові системи не просто індексували текст, а «розуміли» контекст сторінки. У компоненті SoldierCard.svelte реалізовано стандарт мікроданих. Використання атрибутів itemscope та itemtype зі словником Person дозволяє чітко ідентифікувати сторінку як профіль особи. Конкретні властивості, такі як ПІБ, дата народження та дата смерті, маркуються відповідними тегами, що сприяє створенню розширених сніпетів у результатах пошуку.
- Мета-дані та протокол Open Graph. Система динамічно генерує мета-теги для кожної сторінки. Це забезпечує коректне відображення прев'ю при поширенні посилання в соціальних мережах або месенджерах, що значно покращує впізнаваність ресурсу [20].

Окрему увагу приділено машинному зчитуванню часових міток. Використання тегу <time> разом із форматуванням ISO 8601 у коді компонента дозволяє пошуковим алгоритмам точно визначати дати життя та смерті героя, підвищуючи релевантність запитів.

Таким чином, комбінація SSR та структурованих даних Schema.org у межах SvelteKit перетворює вебплатформу з простого реєстру на потужний інструмент

збереження національної пам’яті, доступний для глобального пошуку та зручного поширення .

2.5 Методи забезпечення інклюзивності та адаптивності інтерфейсу

Розробка інтерфейсу цифрового меморіалу вимагає особливого підходу до адаптивності, оскільки контент повинен зберігати високу читабельність та емоційний вплив на пристроях з будь-якою роздільною здатністю [12]. В основі візуальної стратегії проєкту лежить відмова від стандартизованих фреймворків на користь кастомної архітектури стилів, побудованої за допомогою препроцесора SCSS.

Ключовою особливістю реалізації адаптивності є використання системи відносних одиниць rem, що базуються на динамічному розрахунку розміру шрифту кореневого елемента. Для забезпечення ідеального масштабування інтерфейсу (Fluid Design) розроблено систему медіа-запитів, які змінюють font-size кореневого елементу залежно від ширини екрану користувача. Наприклад, для надвеликих екранів (понад 2200px) встановлюється фіксоване значення 12px, тоді як для діапазону від 2200px до 420px використовуються адаптивні значення від 0.54% від ширини екрану до 2.7% від ширини екрану, що демонструється в таблиці 2.1 [21].

Таблиця 2.1 Відповідності ширини екрану користувача, до значення font-size кореневого елементу.

Ширина екрану користувача	> 2200px	2200px – 1600px	1600px – 1024px	1024px – 540px	< 540px
Назва розміру	Супер широкі екрани	Full HD екран	Екран ноутбуку	Екран планшету	Екран смартфону
Значення font-size	12px	0.54vw	0.7vw	1.2vw	2.1vw

Такі значення кореневого font-size обрані не просто так. Вони дозволяють користувачу, на будь-яких розмірах сприймати 1rem за 10px, що спрощує процес стилізації проєкту та уніфікує відображення.

Такий підхід дозволяє всьому інтерфейсу – від відступів до розмірів карток героїв – пропорційно стискатися або розширюватися, зберігаючи композиційну цілісність без різких стрибків макету, дозволяючи спростити розробку відокремивши 5 основних розміри екранів користувацьких пристроїв.

Для розміщення елементів на сторінці використано класичну сіткову модель, розширену специфічними значеннями для забезпечення точності дизайну. Використання SCSS дозволило впровадити нестандартні кроки сітки, що необхідно для коректного позиціонування елементів у складних компонентах, таких як SoldierSlider або FilterBar. Стилі підключаються модульно у файлі +layout.svelte, який є базовою обгорткою для всіх сторінок проєкту. Даний

механізм спрощує процес підключення нових стилів, а також мінімізує обсяг завантажуваного CSS-коду та покращує продуктивність завантаження через механізм «link rel="preload"».

Окрім того, високий рівень доступності меморіалу досягається завдяки суворому дотриманню семантики HTML5 [22]. Це забезпечує коректну обробку сторінки скрінрідерами та пошуковими роботами:

- Контейнери <header>, <footer>, <main> та <section> структурують документ для логічної навігації, та визначення основного контенту.
- Тег <time> використовується для дат, що покращує машиночитаність часових міток [22].
- Елементи <nav> чітко ідентифікують блоки перемикання локалізації та сторінок.
- Векторна графіка <svg> інтегрована безпосередньо в код для забезпечення чіткості зображень іконок при будь-якому масштабуванні.

Також, для забезпечення доступності користувачам з порушеннями моторики реалізовано підтримку керування слайдером героїв за допомогою клавіш-стрілок клавіатури, а також функціонал автоматичного гортання, який дозволяє списку перемикатися та оновлюватися самостійно.

Візуальна частина інтерфейсу, включаючи контрастність текстових блоків біографій, пройшла верифікацію на відповідність стандарту WCAG 2.1 за допомогою інструментів Google Lighthouse, що підтверджує готовність ресурсу до використання людьми з порушеннями зору (Рисунок 2.5).

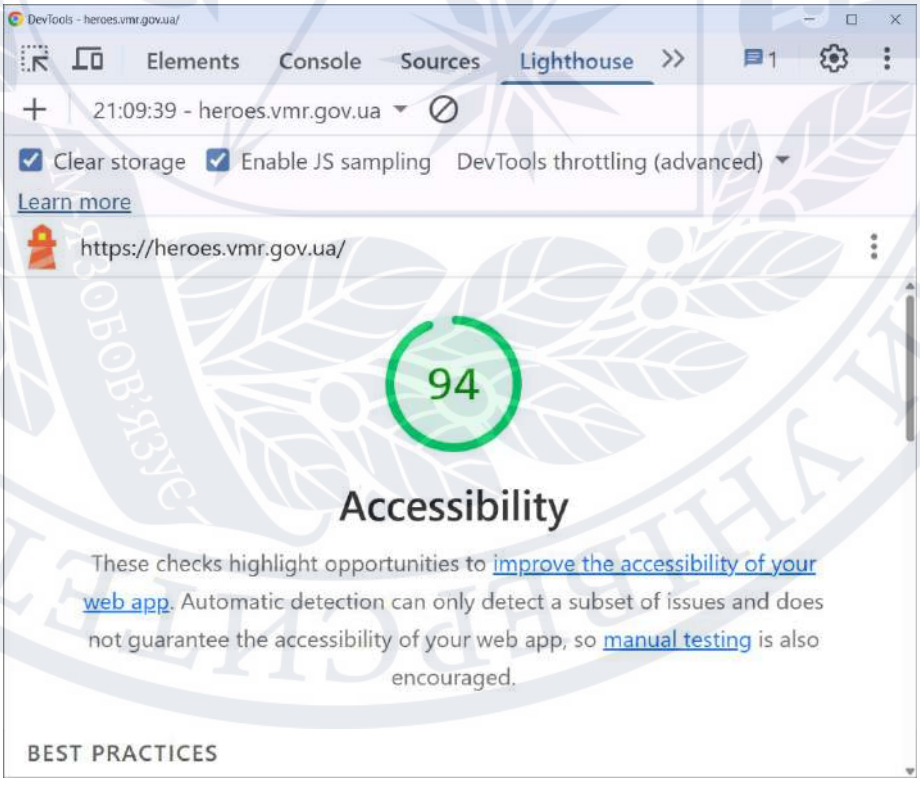

Рисунок 2.5 – Результат аналізу інтерфейсу цифрового меморіалу за допомогою Google Lighthouse.

Висновок до розділу

У цьому розділі було здійснено технічне проєктування та обґрунтування архітектури клієнтської частини цифрового меморіалу, що базується на використанні фреймворку SvelteKit та препроцесорі SCSS.

На основі проведеного порівняльного аналізу було доведено перевагу SvelteKit над традиційними фреймворками завдяки відмові від концепції Virtual DOM на користь компіляції коду. Це забезпечило мінімальний час до інтерактивності та ефективну реактивність через топологічне сортування залежностей, що є критичним для стабільної роботи на малопотужних мобільних пристроях .

Ключові результати проєктування включають:

- Реалізація серверного рендерингу забезпечила миттєву доступність контенту для пошукових роботів, що гарантує 100% індексацію сторінок героїв. Динамічне керування мета-даними та підтримка Open Graph дозволили оптимізувати відображення меморіалу в соціальних мережах.
- Використання мікроданих (словник Person) у компоненті SoldierCard.svelte перетворило візуальний контент на структуровану інформацію, зрозумілу для алгоритмів Google, що сприяє формуванню розширених сніпетів у пошуковій видачі.
- Проєктування системи, де сервер SvelteKit виступає інтелектуальним посередником між клієнтом та REST API, дозволило реалізувати безпечний шар трансформації даних та коректне відображення специфічної мілітарної інформації .
- Відмова від сторонніх CSS-фреймворків на користь кастомної SCSS-архітектури з використанням відносних одиниць rem та динамічного масштабування Root font-size дозволила досягти ефекту «гумової верстки», зберігаючи композиційну цілісність на будь-яких типах екранів .

Завершальним етапом стало впровадження методів інклюзивності згідно зі стандартами WCAG 2.1. Успішне проходження аудиту за допомогою Google Lighthouse та реалізація клавіатурної навігації підтвердили готовність інтерфейсу до використання всіма категоріями громадян. Сформована архітектурна база створює умови для подальшої програмної реалізації проєкту як високоефективного інструменту вшанування пам'яті героїв Вінницької ОТГ.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОПТИМІЗАЦІЯ ВЕБПЛАТФОРМИ

3.1 Розробка модульної структури інтерфейсу на основі Svelte-компонентів

Одним із головних етапів розробки передового сервісу, який матиме можливість розвиватися і оновлюватися ще багато років, є створення зручної і функціональної файлової структури. Структура цифрового меморіалу базується на стандартах фреймворку SvelteKit із використанням директорії /src/lib та системного аліасу $lib для збереження внутрішніх модулів проєкту. Таке рішення забезпечує зручний доступ до компонентів із будь-якої точки файлової системи через абсолютні шляхи, що спрощує підтримку та рефакторинг коду [23].

Декомпозиція інтерфейсу виконана за функціонально-просторовим принципом, що передбачає розподіл компонентів на п’ять ключових категорії:

1. Основні структурні блоки, які розташовані в директорії /src/lib/page-parts. У цьому модулі зосереджені високорівневі компоненти для кожної сторінки вебплатформи, такі як Header.svelte, Footer.svelte та FilterBar.svelte. Окремо слід зауважити модулі відображення контенту: SoldierSlider.svelte та SoldierTiles.svelte. Такий розподіл дозволяє ізолювати складну логіку фільтрації та пошуку від загальної структури сторінок [24].
2. Модальні вікна, які розташовуються в директорії /src/lib/modal.
3. Загальні компоненти платформи, такі як карточка солдата та поле пошуку, розташовуються в директорії /src/lib/components.
4. Інтерактивні компоненти форми, такі як випадаючи списки, поля введення та блоки вибору, знаходяться в директорії /src/lib/form.
5. Мета-компоненти та утиліти, які розташовуються в директорії /src/lib/parts.

Важливою технічною особливістю ієрархії є реалізація механізму перемикання режимів відображення героїв. У головному файлі +page.svelte стан currentMode визначає активний компонент (слайдер або плитку). Зміна стану ініціюється користувачем через відповідну кнопку в інтерфейсі, що дозволяє гнучко підлаштовувати спосіб подачі інформації під потреби відвідувача.

Для реалізації інтерактивного елемента SoldierSlider.svelte інтегровано спеціалізовану бібліотеку Swiper. Це дозволяє використовувати нативні можливості управління контентом, включаючи підтримку жестів прокрутки на сенсорних екранах та навігацію за допомогою клавіш-стрілок на клавіатурі, що значно покращує показники доступності інтерфейсу.

Окремо слід звернути увагу на локалізацію цифрового еморіалу, яка реалізована через глобальну систему сторів на основі бібліотеки sveltekit-i18n. Кожен компонент незалежно імпортує об'єкт $t із файлу translations.js, що забезпечує миттєве оновлення текстових рядків у Header, Footer чи модальних вікнах при зміні мовної версії без необхідності повного перезавантаження сторінки або передачі мовних пропсів через усю ієрархію компонентів.

Ефективність користувацького інтерфейсу цифрового меморіалу безпосередньо залежить від архітектури потоків даних між компонентами. У розробленому застосунку реалізовано гібридну модель управління станом, яка поєднує використання ієрархічної передачі параметрів (props) для контенту та глобальних сховищ для системних станів.

Центральним вузлом розподілу даних є головна сторінка +page.svelte. Завдяки механізму SSR (Server Side Rendering), компонент отримує об'єкт data, який містить уже сформовані масиви героїв, фільтрів та інформацію про «Героїв дня». Процес передачі даних до дочірніх модулів реалізовано за принципом «згори донизу»:

- *Контентні дані*: Список солдатів передається у компонент SoldierSlider або SoldierTiles через відповідні атрибути. Це гарантує, що компоненти відображення залишаються «чистими» і займаються лише рендерингом отриманого масиву (Рисунок 3.1).
- *Реактивність стану*: Використання реактивних міток Svelte ($:) дозволяє автоматично оновлювати локальні змінні soldiers та heroes_of_day при кожній зміні об'єкта data, що забезпечує миттєву синхронізацію інтерфейсу з серверними даними без необхідності ручного маніпулювання DOM.

```
<div class="main-content">
    {#if soldiers.length}
        {#if currentMode === "slider"}
            {#key $filterChangeCounter}
                <SoldierSlider {soldiers} on:go-to-end={getNextSoldiers}/>
            {/key}
        {/if}
        {#if currentMode === "tiles"}
            <SoldierTiles {soldiers} on:go-to-end={getNextSoldiers}/>
        {/if}
    {:else}
        <section id="soldier-not-found">
            <div class="container">
                <div class="icon">
                    <svg...>
                </div>
                <div class="text">
                    {$t('static.dont-find')}
                </div>
            </div>
        </section>
    {/if}
</div>
```

Рисунок 3.1 – Реалізація реактивної ініціалізації даних та передачі пропсів у головному файлі сторінки.

Для управління станами, що виходять за межі ієрархії «батько-дитина», у проєкті застосовано Svelte Stores. Це дозволяє уникнути надлишкового прокидання пропсів через декілька рівнів вкладеності. Ключовими елементами цієї системи є:

1. *Глобальна локалізація*: глобальне сховище t, що імпортується з translations.js, забезпечує доступ до перекладів у будь-якому компоненті. Це дозволяє змінювати мову інтерфейсу (UA/EN) в реальному часі для всього проєкту одночасно.
2. *Керування модальними вікнами*: глобальна змінна openDaySoldierModal зберігає булеве значення, яке контролює видимість модального вікна з героями, які загинули сьогодні. Це дозволяє викликати модальне вікно з будь-якої частини інтерфейсу, наприклад, із заголовка сторінки або через пряме посилання.
3. *Тригери оновлення*: використання filterChangeCounter як реактивного тригера дозволяє системі розпізнавати моменти зміни параметрів фільтрації. Це ініціює виклик функції invalidateAll, яка оновлює дані з сервера, зберігаючи при цьому цілісність поточної сесії користувача.

Такий підхід до управління станом мінімізує обсяг клієнтського коду та забезпечує високу швидкість відгуку інтерфейсу, що є критично важливим для інклюзивних вебплатформ.

Центральним елементом візуальної комунікації цифрового меморіалу є модулі презентації даних про героїв. Для забезпечення гнучкості користувацького досвіду реалізовано два взаємодоповнюючі режими відображення: динамічний слайдер та статична плитка (Рисунок 3.2). Це дозволяє користувачеві обирати між фокусованим переглядом окремих карток та швидким оглядом великої кількості записів [25].

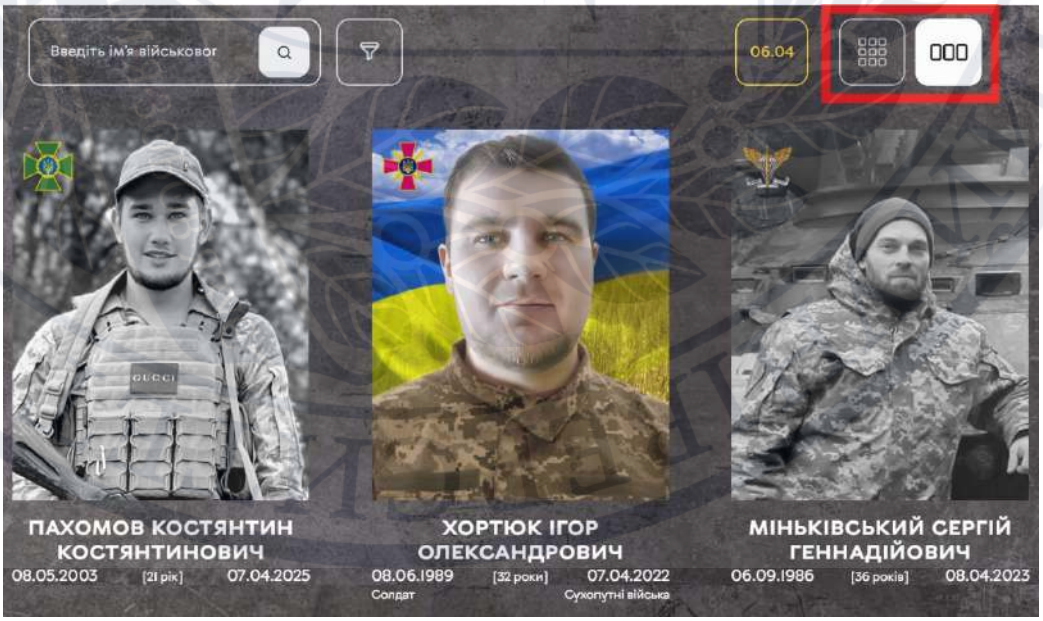

Рисунок 3.2 – Інтерфейс вибору режиму відображення та вигляд активованого слайдера героїв.

Програмна логіка перемикання режимів реалізована у файлі +page.svelte через стан currentMode. Вибір режиму здійснюється користувачем за допомогою інтерактивних кнопок у інтерфейсі. Технічно це реалізовано через директивну умову, яку можна побачити на Рисунку 3.1. Використання такої конструкції забезпечує ефективне управління ресурсами браузера, оскільки Svelte не просто приховує неактивний компонент через CSS, а повністю вилучає його з DOM-дерева, вивільняючи пам'ять.

Для реалізації модуля SoldierSlider обрано спеціалізовану бібліотеку Swiper, яка легко підключається за допомогою JavaScript і не потребує ніяких додаткових. Вибір даного інструменту обумовлений його високою продуктивністю та вбудованою підтримкою стандартів доступності. Основні параметри конфігурації слайдера включають:

- *Навігаційні модулі*: Інтеграція Navigation та Pagination для забезпечення зрозумілих точок контролю.
- *Клавіатурне управління*: Активація параметру keyboard, що дозволяє перемикати картки героїв за допомогою клавіш-стрілок. Це може бути критично важливим аспектом, для спрощення процесу взаємодії зі слайдером для різних користувачів.
- Адаптивність: Налаштування breakpoints дозволяє динамічно змінювати кількість видимих карток (параметр slidesPerView) залежно від ширини вікна перегляду, що забезпечує коректне відображення як на мобільних пристроях, так і на широкоформатних моніторах.

Альтернативний режим SoldierTiles.svelte базується на використанні CSS Grid Layout. Це дозволяє формувати чітку ієрархічну сітку, де кожна картка героя має стабільний розмір. Поєднання цих двох модулів у межах одного інтерфейсу вирішує проблему інформаційного перевантаження: слайдер акцентує увагу на окремих постатях, тоді як плитка надає зручний інструментарій для пошуку та фільтрації [26].

Окрему увагу приділено оптимізації рендерингу під час зміни стану фільтрів. Завдяки використанню ключових сторів, кожен перехід між режимами або оновлення списку супроводжується коректною ініціалізацією відповідного компонента, що запобігає «стрибкам» контенту та забезпечує плавність візуальних переходів.

Отже, із розглянутих вище прикладів можна зрозуміти основний принцип побудови модулів та функціональних компонентів інтерфейсу, а також налаштування взаємодій між цими елементами, щоб забезпечити користувачу найкращий досвід взаємодії з платформою.

3.2 Реалізація серверної логіки та взаємодії з REST API

Реалізація серверного рендерингу (SSR), як вже зазначалося, є критично важливою для проєкту цифрового меморіалу, оскільки вона забезпечує швидке відображення контенту при першому завантаженні та дозволяє пошуковим

системам коректно індексувати сторінки героїв. У фреймворку SvelteKit цей функціонал реалізовано через системний файл +page.server.js, який виконується виключно на стороні сервера перед відправкою HTML-коду клієнту.

Ключовим елементом серверної логіки є асинхронна функція load. Її завданням є агрегація даних з різних ендпоінтів REST API та формування єдиного об'єкта даних для сторінки. У розробленому застосунку функція load координує роботу трьох основних внутрішніх методів, що визначені в межах того самого серверного модуля (Рисунок 3.3):

- getHeroes – відповідає за отримання основного списку полеглих захисників із підтримкою пагінації та фільтрації;
- getHeroesOfTheDay – отримує дані про героїв, дата загибелі яких припадає на поточне число;
- getFilters – завантажує актуальний перелік категорій та підрозділів для динамічного формування панелі фільтрації.

```
export const load = async ({cookies, url}) => {

    let heroes = {status: false};
    heroes = await getHeroes(cookies, url);

    let heroesOfDay = await getHeroesOfTheDay(cookies);

    let filters = await getFilters(cookies);

    return {
        heroes_data: heroes,
        heroes_of_day: heroesOfDay??[],
        filters: filters
    };
}
```

Рисунок 3.3 – Реалізація серверної функції load для координації збору даних.

Використання внутрішніх методів у межах одного файлу дозволяє інкапсулювати логіку запитів та забезпечити високу швидкість обробки даних за рахунок відсутності зайвих мережевих затримок між сервером застосунку та базою даних.

Окрему увагу приділено надійності системи. Оскільки взаємодія з REST API на базі ASP.NET залежить від зовнішніх факторів, у кожному методі завантаження реалізовано блок обробки виключень try...catch. У разі виникнення критичної помилки під час запиту (наприклад, недоступність серверу API), застосунок використовує стандартний механізм SvelteKit – функцію error. Це

дозволяє коректно перервати рендеринг та вивести користувачеві інформативне повідомлення з відповідним HTTP-статусом (наприклад, 400 або 500), запобігаючи «падінню» всього інтерфейсу.

Результатом роботи +page.server.js є об'єкт, що містить поля heroes_data, heroes_of_day та filters. Ці дані автоматично стають доступними у клієнтському компоненті через пропс data, що забезпечує гідратацію інтерфейсу – процес, при якому статичний HTML, згенерований сервером, стає інтерактивним у браузері користувача. Такий підхід дозволяє поєднати переваги традиційних багатосторінкових сайтів (SEO, швидкий старт) із гнучкістю сучасних SPA-застосунків [27].

Для забезпечення централізованої взаємодії між сервером застосунку SvelteKit та зовнішнім REST API було розроблено спеціалізований модуль – apiWorker.js. Його основна роль полягає в абстрагуванні логіки формування HTTP-запитів, що дозволяє уникнути дублювання коду в різних частинах системи та забезпечує єдину точку входу для всіх мережевих операцій.

Ключовою функцією модуля є apiQueryBack. Вона виконує роль проксі-шару, який автоматично доповнює запити необхідними метаданими. Процес обробки запиту включає кілька етапів:

1. Формування цільової адреси, під час якого система використовує змінні оточення через модуль $env. Це дозволяє гнучко змінювати адресу бекенду для різних середовищ без втручання в код безпосередньо. У разі відсутності налаштувань передбачено безпечне посилання на резервний домен.
2. Динамічна ін’єкція локалізації, яка автоматично визначає поточну мову користувача з cookies та додає відповідний параметр lang до URL запиту. Логіка враховує наявність інших параметрів у рядку запиту, що запобігає синтаксичним помилкам та втратам важливих URL параметрів при формуванні адреси
3. Також система одразу реалізує управління авторизаційними заголовками. Незважаючи на те, що публічна частина меморіалу наразі не вимагає автентифікації, у модулі реалізовано перевірку наявності user_token. Це архітектурне рішення забезпечує масштабованість системи для майбутньої інтеграції адміністративної панелі без необхідності рефакторингу існуючих методів.

Взаємодія безпосередньо з мережею реалізована на базі бібліотеки Axios [28]. Вибір цієї бібліотеки зумовлений її стабільністю при роботі в середовищі Node.js (на стороні сервера SvelteKit) та зручними механізмами перехоплення відповідей, та загальноприйнятою стандартизацією цієї бібліотеки, як універсальне рішення для JavaScript. У разі виникнення помилок на рівні HTTP (наприклад, 404 або 503), модуль передає виключення вище за стеком викликів, де воно обробляється стандартними засобами SvelteKit, описаними у попередньому розділі. Такий підхід гарантує, що будь-яка мережева

несправність буде вчасно виявлена та відображена користувачеві у коректній формі.

Також слід нагадати, що процес передачі даних від REST API до компонентів інтерфейсу потребує проміжного етапу трансформації. Це зумовлено необхідністю приведення технічних форматів зберігання даних у базі до вигляду, що відповідає вимогам інклюзивності та візуальної ієрархії проєкту. Основна логіка мапінгу зосереджена у файлі global.js та застосовується під час формування масиву об'єктів у серверній частині застосунку.

Однією з ключових функцій трансформації є getFormatedDate. Оскільки дати народження та смерті героїв передаються з API у стандарті ISO 8601, виникає потреба їх перетворення у звичний для українського користувача формат (ДД.ММ.РРРР). Програмна реалізація методу передбачає ручне вилучення компонентів дати та додавання нулів для збереження двозначного формату днів та місяців. Це забезпечує візуальну цілісність карток героїв, де дати є критично важливими маркерами пам'яті.

Важливим інструментом візуальної ідентифікації є метод getCorpsCode. Він реалізує логіку зіставлення текстових назв підрозділів, наприклад, «ГУР», «морська піхота») із технічними ідентифікаторами класів, такими як gur та navy відповідно. Це дозволяє динамічно змінювати елементи інтерфейсу:

- Автоматично підставляти відповідні емблеми та іконки родів військ;
- Змінювати стилістичне оформлення карток залежно від приналежності героя до певного військового формування;
- Виводити повні, офіційні назви підрозділів на сторінці інформації про героя, зберігаючи при цьому лаконічність у загальному списку.

Також, для обробки текстової інформації, зокрема біографій героїв, застосовується бібліотека marked. Необхідність використання даної бібліотеки обумовлена тим, що розширена інформація про життєвий шлях захисників зберігається в базі даних у форматі Markdown. Функція, яку надає згадана бібліотека, перетворює її на безпечний HTML-код безпосередньо під час виконання SSR. Це дозволяє адміністраторам меморіалу використовувати базове форматування тексту, такі як списки та виділення тексту, без необхідності володіти знаннями HTML, та уникаючи ризиків порушення структури сторінки.

Візуальний результат трансформації біографічних даних та їхнього фінального відображення в інтерфейсі сторінки героя представлено на рисунку 3.4. Завдяки використанню бібліотеки marked , текстова інформація про життєвий шлях захисника, що зберігається в базі даних у форматі Markdown , конвертується в семантично коректну HTML-розмітку безпосередньо під час виконання серверного рендерингу (SSR). Це дозволяє забезпечити зручну візуальну ієрархію контенту: від фото та основних тегів до деталізованого опису нагород та біографії, що відповідає вимогам інклюзивності та естетичної стриманості меморіалу.

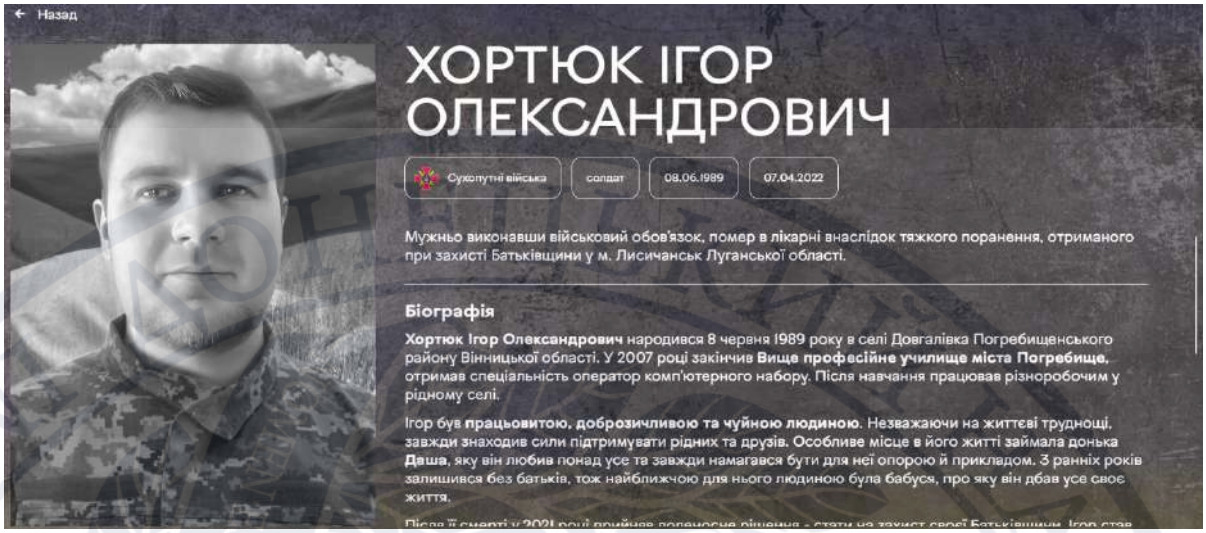

Рисунок 3.4 – Інтерфейс персональної сторінки, з відображення фотографії та біографії герої.

Фінальним етапом трансформації є збірка об'єкта soldier. У циклі обробки відповіді від API серверна функція формує структуру, яка містить ідентифікатор, оброблене посилання на зображення, масив тегів, які включають дати народження та смерті, а також деталізований опис (Рисунок 3.5). Такий підхід гарантує, що клієнтська частина отримує повністю підготовлений до рендерингу об'єкт, що мінімізує навантаження на браузер користувача та пришвидшує відображення меморіалу.

```
soldiers.push({
    id: el?.id??'0',
    image: imageUrl??'',
    name: el?.name??'',
    tags: [
        ...tags,
        { name: el?.birthDate?getFormatedDate(el?.birthDate):'', },
        { name: el?.deathDate?getFormatedDate(el?.deathDate):'', }
    ],
    detail: [
        { text: el?.info??'' },
        el?.awards && {
            title: t.get( key: 'static.awarded'),
            list: [
                {
                    text: el?.awards,
                    image: '/orden.png'
                }
            ]
        },
        el?.bio && {
            title: t.get( key: 'static.bio'),
            text: marked( src: el?.bio??'')
        }
    ]
})
```

Рисунок 3.5 – Процес формування вхідних даних героїв у об’єкт інтерфейсу.

3.3 Технології стилізації та забезпечення адаптивності інтерфейсу

Для забезпечення високої швидкості завантаження та гнучкості налаштування інтерфейсу було прийнято рішення про відмову від сторонніх CSS-фреймворків, таких як Tailwind або Bootstrap, на користь кастомної архітектури стилів. У проєкті використано препроцесор SCSS, що дозволило реалізувати модульну систему стилів, організовану у директорії src/styles. Така структура забезпечує чіткий розподіл зон відповідальності та спрощує підтримку коду [29].

Програмна архітектура стилів складається з набору окремих модулів, що об'єднуються у головному файлі app.scss. Ці модулі можна поділити на наступні категорії:

- *Базові модулі* – це модулі які містять базові налаштування стилістики проєкту та його принципів розміщення. До таких модулів відноситься:
    - app.scss, в якому відбувається підключення всіх модулів, а також реалізується очистка базових браузерних стилів, визначається глобальне значення властивості font-size та налаштовується базовий стиль шрифтів.
    - _grid.scss, в якому реалізовується логіку кастомної сітки, що, на відміну від стандартних рішень, реалізує гнучку систему колонок, що підтримує дробові значення.
    - _variable.scss, в якому створюються змінні для зберігання колірної палітри.
    - _animation.scss, в якому описуються ключові кадри базових створених анімацій.
- *Модулі компонентів* – це модулі, які містять налаштування стилістики для різних компонентів сторінки, таких як кнопки, модальні вікна, поля вводу тексту, чекбокси, pop-up повідомлення, а також спеціалізовані компоненти для цієї вебплатформи (наприклад картка солдата). До цієї категорії відносяться модулі _toast.scs, _selector.scss, _modal.scss, _inputs.scss, _button.scss, _components.scss.
- *Модулі секцій* – це модулі, які налаштовують стилі для загально використовуваних секцій платформи: шапки сайту (_header.scss) та підвалу сайту (_footer.scss).
- *Модулі сторінок* – це модулі, які налаштовують стилізацію кожної окремої сторінки вебплатформи. Ці модулі також можуть розбиватися на декілька простіших модулів. До модулів сторінок відносяться main.scss, soldier.scss, about-us.scss.

  Всю архітектуру SCSS модулів проєкту можна переглянути на Рисунку 3.6.

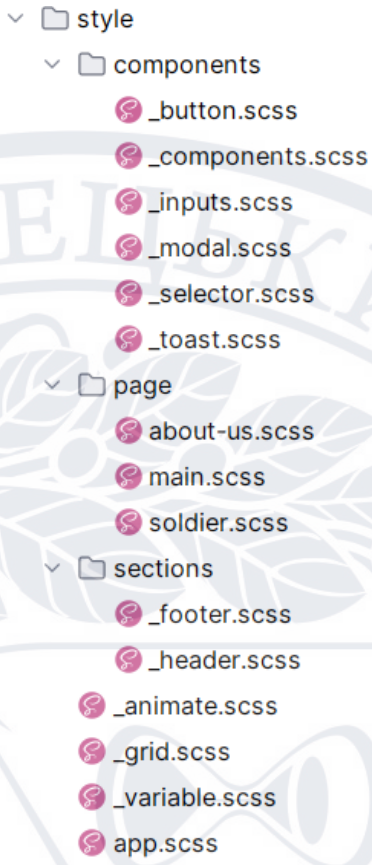

Рисунок 3.6 – Архітектура SCSS-модулів у структурі проєкту.

Важливою перевагою такої організації є використання механізму інкапсуляції Svelte разом із глобальними модулями SCSS. Загальносистемні налаштування, такі я сітка або змінні, імпортуються один раз, тоді як специфічні стилі компонентів залишаються ізольованими. Це запобігає «розростанню» фінального CSS-файлу та гарантує, що браузер завантажує лише той код, який необхідний для відображення поточної сторінки. Також, використання вкладених селекторів та міксинів у SCSS дозволило реалізувати складну логіку станів елементів з мінімальним дублюванням коду.

Обов’язково слід зауважити, що основою адаптивності інтерфейсу цифрового меморіалу є концепція «гумової верстки», яка забезпечує пропорційну зміну всіх елементів дизайну залежно від розміру вікна перегляду. Технічна реалізація цього підходу базується на використанні відносних одиниць вимірювання rem у поєднанні з динамічним керуванням розміром шрифту кореневого елемента html.

Головною перевагою використання rem є те, що всі лінійні розміри, відступи та параметри типографіки стають похідними від одного значення — font-size тегу html. Для спрощення процесу розробки та математичних розрахунків у проєкті прийнято базове співвідношення, де 1rem дорівнює 10px. Це дозволяє розробнику легко конвертувати значення з макетів у код: наприклад, відступ у 24px записується як 2.4rem, що підвищує читабельність стилів та швидкість розробки.

Програмна логіка масштабування реалізована у файлі app.scss через використання одиниць в'юпорта (vw). Це дозволяє базовому розміру шрифту плавно змінюватися разом із шириною екрана, забезпечуючи ідеальну візуальну стабільність композиції. Для запобігання надмірному розтягуванню або стисненню контенту впроваджено систему медіа-запитів, що фіксують або коригують крок масштабування для різних розмірів екранів.

Така математична модель дозволяє уникнути різких «стрибків» інтерфейсу при зміні орієнтації пристрою, зміні розміру вікна браузера або використанню налаштувань браузера, для збільшення розміру шрифту. Крім того, налаштування text-size-adjust: none та відключення стандартного відображення скролбарів на рівні кореневого елемента забезпечує ідентичний вигляд меморіалу в різних браузерах, що є критичним для збереження естетичної цінності проєкту.

Застосування rem замість статичних px також позитивно впливає на інклюзивність: інтерфейс коректно реагує на системні налаштування масштабування тексту, які можуть використовувати люди з порушеннями зору, що відповідає вимогам стандарту WCAG 2.1 щодо гнучкості подачі контенту.

Не менш важливим фактом візуальної структури цифрового меморіалу є кастомна сітка, реалізована у модулі _grid.scss. На відміну від стандартних бібліотек, дана система розроблена з урахуванням специфіки «гумової» верстки та потреби у прецизійному позиціонуванні карток героїв.

В основі архітектури лежить класична 12-колонкова модель, проте її можливості розширені для підтримки специфічних пропорцій інтерфейсу. Ключові технічні характеристики сітки включають:

- *Контейнеризація*, при якій використовується клас .container із фіксованим відступом від країв екрану, що забезпечує стабільність композиції на різних широкоформатних моніторах. Для секцій, що мають мати менші розміри та розташовуватися по центру екрану на, передбачено додатковий клас .short, який обмежує максимальну ширину контейнеру до 65% від ширини екрану, але зберігає 100% ширину контейнеру для мобільних пристроїв та планшетів.
- *Дробові колонки* є унікальною особливістю реалізації є підтримка нецілих значень ширини колонок, як-от .col-2-5 та .col-9-5. Це дозволяє створювати складні композиції в слайдері та шапці сайту, де стандартний крок у 1/12 не забезпечує необхідної щільності контенту.
- *Врахування відступу між колонками* також є одним із унікальних рішень, яке рідко реалізується іншими CSS-фреймворками. Воно полягає в тому, що при заданні внутрішнього відступу між колонками, їх ширина адаптується так, щоб відповідати своїм значенням враховуючи відступи, які в інших рішеннях можуть впливати на розміщення елементів і переносити їх на інші рядки.
- Обчислення ширини кожного елементу автоматизовано засобами SCSS із використанням функції math.div. Ширина кожної колонки W розраховується за формулою:

$$W = \left(\frac{n}{12}\right) * 100\% - \delta \quad (3.1)$$

де n – номер колонки від 1 до 12;

δ – відступ між колонками в контейнері.

Програмний приклад реалізації формули 3.1, а також системи дробових колонок можна спостерігати на рисунку 3.7.

```
@for $i from 1 through 12 {
  .col-#{$i}{
    width: calc(math.percentage(math.div($i, 12)) - var(--gap-diference));
    max-width: calc(math.percentage(math.div($i, 12)) - var(--gap-diference));
  }
}
.col-2-5{
  width: 22.5%;
}
.col-9-5{
  width: 77.5%;
}
```

Рисунок 3.7 – Реалізація кастомної сітки з підтримкою дробових колонок у файлі _grid.scss.

Гнучкість макетів досягається завдяки використанню технології Flexbox у межах класу .container. Це дозволяє автоматично керувати розподілом простору між елементами та забезпечувати коректне перенесення блоків на мобільних пристроях. Поєднання такої сітки з динамічним Root font-size створює ефект безшовного масштабування, де всі відступи та розміри колонок пропорційно змінюються відповідно до в'юпорта користувача.

Реалізація всіх зазначених вище елементів стилізації вебплатформи дозволяє зробити її не лише візуально привабливою, а й динамічною і легко масштабованою. Це дозволить полегшити подальший розвиток проєкту, зменшуючи час і ресурси необхідні для забезпечення адаптивності і загальної єдності стилю всього сайту.

3.4 Оптимізація користувацького досвіду та інтерактивності вебплатформи

Забезпечення високого рівня емоційної взаємодії з користувачем у проєкті цифрового меморіалу досягається шляхом впровадження легких візуальних ефектів, що не перевантажують апаратні ресурси пристрою, але дозволяють оживити інтерфейс і зробити його більш інтерактивним. Ключовим елементом такої взаємодії є анімації станів взаємодії з кнопками та анімація «друку» поетичного тексту у підвалі сайту, що імітує роботу механічної друкарської машинки.

Реалізація анімацій взаємодії з кнопками на сайті відбувається за допомогою CSS-псевдокласів. Замість ресурсномістких JavaScript-сценаріїв, динамічна зміна станів реалізована через псевдокласи :hover, під час наведення курсору на елемент та :active, під час безпосереднього натискання на елемент. Приклад використання цих станів можна побачити на Рисунку 3.8.

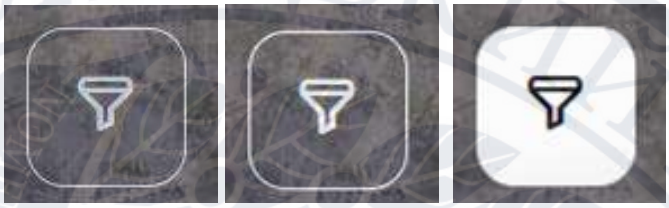

Рисунок 3.8 – Демонстрація станів кнопки при використанні різних CSS-псевдокласів. На першому малюнку базовий вигляд кнопки. На другому – з псевдокласом :hover. На третьому – з псевдокласом :active.

Для забезпечення візуальної м'якості переходів застосовується властивість transition: all .2s, параметри якої синхронізовані через глобальні змінні SCSS.

Також дуже важливою є анімація «друку» тексту, програмна реалізація якої базується на реактивних можливостях фреймворку Svelte та рекурсивному виклику функції tupNewLatter у компоненті Footer.svelte. Основні технічні аспекти алгоритму (Рисунок 3.9) включають:

- *Одноразова активація.* Процес друку ініціюється виключно при першому монтуванні компонента в DOM-дерево за допомогою життєвого циклу onMount. Це запобігає зайвому навантаженню при навігації між внутрішніми станами сторінки.
- *Механічний ритм.* На відміну від плавних переходів, для цього ефекту обрано фіксовані інтервали затримки – 180ms. Це створює характерний переривчастий ритм «друкарської машинки».
- *Керування станом.* Анімований текст накопичується у реактивній змінній poemResult, що забезпечує миттєве оновлення інтерфейсу при додаванні кожного нового символу.
- *Завершення циклу.* Алгоритм містить умову зупинки, яка спрацьовує автоматично після виведення останнього символу масиву, вивільняючи ресурси таймера.

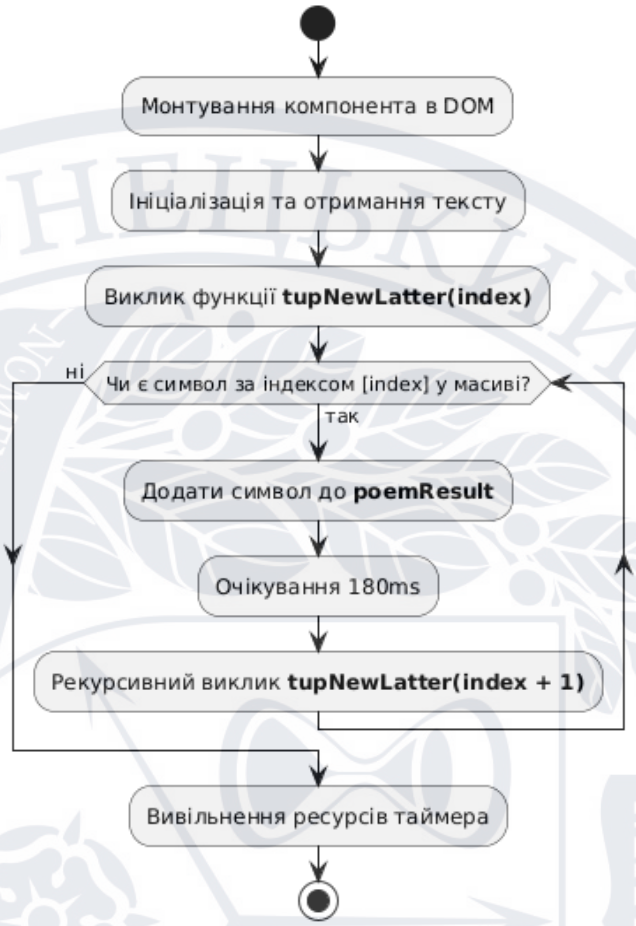

Рисунок 3.9 – Алгоритм анімації «друку» тексту у підвалі сайту.

Такий підхід дозволяє реалізувати ефекти зміни кольору фону, прозорості або трансформації безпосередньо силами графічного процесора, що гарантує плавне програвання анімації навіть на мобільних пристроях із обмеженою потужністю.

Наступним, не менш важливим елементом для забезпечення доступності інформації про Героїв Вінницької ОТГ для міжнародної спільноти та іноземних медіа-ресурсів, у проєкті реалізовано дворівневу систему локалізації. Як вже зазначалося раніше, вона базується на використанні бібліотеки sveltekit-i18n для статичних інтерфейсних рядків та параметризованих запитів до REST API для динамічних даних [30].

Технічна реалізація статичної локалізації зосереджена у файлі translations.js. На відміну від стандартних клієнтських рішень, тут застосовано метод завантаження JSON-словників через асинхронні лоадери (Рисунок 3.10). Це також дозволяє розділити переклад контент для різних частин інтерфейсу та сторінок.

```
const config = ({
    loaders: [
        {
            locale: 'en',
            key: 'static',
            loader: async () => (
                await import('./lang/en/static.json')
            ).default,
        },
        {
            locale: 'ua',
            key: 'static',
            loader: async () => (
                await import('./lang/ua/static.json')
            ).default,
        }
    ]
});
```

Рисунок 3.10 – Приклад коду конфігурації та завантаження JSON-словників у translations.js.

Доступ до перекладів у компонентах здійснюється через Svelte-стор $t. Оскільки проєкт використовує SSR, початкове наповнення стора відбувається на етапі сервера, що виключає появу нелокалізованого тексту під час гідратації сторінки.

Локалізація даних про загиблих воїнів, такої як імена, біографії та нагороди, реалізовані на рівні взаємодії з бекендом. У модулі apiWorker.js, при виконанні методу apiQueryBack, кожному запиту до REST API додається query-параметр мови, який отримує значення із збереженого в cookies значенні. Це дозволяє отримувати вже відфільтрований та перекладений контент безпосередньо з бази даних, мінімізуючи навантаження на клієнтську частину.

Також, для забезпечення цілісності навігації у global.js розроблено допоміжну функцію route(link), яка автоматично додає префікс поточної мови до всіх внутрішніх посилань. Код цієї функції представлений на Рисунку 3.10.

```
export function route(link){
    let lang = get(page).params.lang??null;

    if(lang??false){
        return '/' + lang + link;
    }

    return link;
}
```

Рисунок 3.10 – Приклад коду конфігурації та завантаження JSON-словників у translations.js.

Зміна мови платформи ініціюється через серверний екшн change-lang. Користувач відправляє запит з новою мовою сторінки, сервер оновлює куку та виконує повне перезавантаження сторінки. Такий підхід гарантує повну синхронізацію стану між клієнтом, сервером та кешем браузера, що є критичним для коректного відображення меморіальних даних.

Також, у межах розробки інтерфейсу цифрового меморіалу особливу увагу приділено системі фільтрації та пошуку, яка функціонує на принципах реактивного програмування. Програмна логіка обробки даних базується на синхронізації внутрішнього стану застосунку з URL-параметрами браузера, що забезпечує збереження стану вибірки при навігації та можливість передачі прямих посилань на відфільтровані списки героїв.

Центральним механізмом управління станом у файлі +page.svelte виступає сховище-тригер filterChangeCounter. При будь-якій зміні категорій підрозділів або параметрів пошуку система виконує інкремент цього лічильника. Це ініціює програмний ланцюжок: спочатку оновлюється URL-адреса сторінки через метод goto, а потім, змінна лічильника filterChangeCounter, запускає процес перемальовування секції слайдеру, що запобігає розсинхронізації даних між відображуваним контентом та адресним рядком.

Окремо необхідно наголосити на полі пошуку за ім’я військово. Пошукова система інтегрована в загальну логіку фільтрації, але разом з тим вона містить власний додатковий інструмент, який виконує попередній пошук військових. Коли людина починає вводити запит в пошукову стрічку, система автоматично здійснює запит на серверний метод «search-name», для отримання списку відповідних імен. Ці API запит попереднього пошуку реалізовується з використанням методів оптимізації навантаження. Для цього реалізовано механізм затримки тривалістю 1000 мс, а також встановлено валідаційний поріг: запит до серверної частини ініціюється лише після введення трьох символів. У разі натискання клавіші Enter або відправки форми через інтерфейс, подія перехоплюється, а введена стрічка трансформується у фільтр пошуку, який автоматично додається до URL-параметрів. Такий підхід дозволяє уніфікувати

обробку даних: незалежно від способу введення, система використовує єдиний обробник запитів [31].

Також, динамічне завантаження контенту реалізовано через функцію getNextSoldiers, яка тісно пов'язана зі станом фільтрації. При кожному виклику функція враховує поточні параметри з URL, забезпечуючи відповідність нових даних, до вже існуючих. Критично важливим UX-рішенням є алгоритм скидання результатів, який очищує масив soldiers і встановлює покажчик currentPage в початкове значення, при зміні будь-якого фільтра. Це гарантує, що користувач завжди бачить актуальну вибірку з першої сторінки, уникаючи змішування результатів різних запитів. Використання прапорців станів isLoad та canReget запобігає дублюванню запитів при швидкому скролінгу, що підвищує стабільність роботи клієнтської частини на пристроях з низькою швидкістю інтернет-з'єднання.

Таким чином, дотримуючись всіх вищезазначених методів та реалізуючи відповідний функціонал, було створено зручну та інтерактивну платформу, яка дозволяє користувачу не лише переглядати інформацією, а й відчувати взаємодію та динамічність вебплатформи.

3.5 Тестування та верифікація програмних рішень вебплатформи

Завершальним етапом розробки цифрового меморіалу став процес комплексного ручного тестування та верифікації всіх функціональних блоків. Метою цього етапу було підтвердження стабільності інтерфейсу, коректності обробки даних із REST API та відповідності розробленої системи вимогам інклюзивності та адаптивності.

Для перевірки надійності системи було проведено серію тестів основних методів трансформації даних, що містяться у файлі global.js. Тестування відбувалося з залученням зовнішніх зацікавлених осіб, як от представників Ради загиблих та представників Вінницької ОТГ. Верифікація здійснювалася шляхом порівняння вхідних даних із API та їхнього фінального відображення в інтерфейсі:

- *Валідація дат та обчислення віку*: перевірено коректність роботи методу getFormatedDate, який трансформує системні мітки ISO у формат ДД.ММ.РРРР. Окремо протестовано алгоритм розрахунку віку в компоненті SoldierCard.svelte, та за допомогою різних тестових сценаріїв підтверджено, що система правильно враховує день та місяць смерті для визначення повних років життя.
- *Відмінювання числівників*: шляхом ручного введення значень перевірено роботу функції setWordByLength, яка забезпечує граматично правильне відображення слова «рік» (наприклад, «21 рік», «24 роки», «30 років») залежно від віку героя та налаштувань статичних перекладів.
- *Мапінг підрозділів*: проведено візуальну перевірку відображення емблем родів військ. Підтверджено, що метод getCorpsCode безпомилково

зіставляє текстові назви підрозділів (наприклад, «Морська піхота») із відповідними іконками (navy.png)

Наступним етапом стало тестування користувацьких сценаріїв, що включало перевірку динамічних елементів інтерфейсу:

1. *Пошукова система*: під час ручного введення запитів підтверджено роботу механізму затримки та валідаційного порогу, що було впроваджено для оптимізації навантаження на сервер.
2. *Фільтрація та стан URL*: перевірено синхронізацію між обраними фільтрами та адресним рядком браузера. Підтверджено, що при зміні категорії військ URL-параметри оновлюються коректно, а метод invalidateAll забезпечує миттєве оновлення списку героїв без перезавантаження всієї сторінки.
3. *Анімаційні ефекти*: проведено візуальний контроль анімації «друку» тексту у підвалі сайту. Підтверджено, що функція tupNewLatter активується одноразово при монтуванні компонента і коректно вивільняє ресурси таймера після виведення останнього символу.

Окрім того, для отримання об'єктивної оцінки якості програмного продукту було проведено автоматизований аудит за допомогою інструменту Google Lighthouse, результати якої представлені в таблиці 3.1. Результати підтвердили відповідність меморіалу стандартам WCAG 2.1 та високу технічну якість коду.

Таблиця 3.1 Результати комплексного аудиту інтерфейсу в Google Lighthouse.

Категорія аудиту	**Показник**	**Обґрунтування результату**
Доступність	94/100	Використання семантичних тегів HTML5, підтримка клавіатурної навігації та наявність контрастних текстових блоків.
Продуктивність	75/100	Швидке відображення контенту завдяки SSR та механізму link rel="preload" для критичних зображень.
Найкращі практики	96/100	Використання сучасних стандартів веброзробки, безпечне з'єднання та відсутність помилок у консолі браузера.
SEO	100/100	Наявність Schema.org розмітки, унікальних мета-тегів та коректної структури заголовків.

Також, оскільки в основі проєкту лежить авторська система «гумової верстки», було проведено ретельне ручне тестування на різних пристроях та браузерах, таких як: Chrome, Safari, Firefox, Edge та Opera. Перевірка підтвердила, що завдяки використанню відносних одиниць rem та динамічному розрахунку font-size кореневого елемента, інтерфейс зберігає композиційну цілісність при будь-якій ширині в'юпорта – від 320px до 2560px . Також було перевірено ручне керування слайдером героїв за допомогою клавіш-стрілок, що забезпечує доступність ресурсу для користувачів із порушеннями моторики.

Висновок до розділу

Результатом виконання цього розділу стала повна програмна реалізація користувацького інтерфейсу цифрового меморіалу на базі фреймворку SvelteKit. Застосування компонентно-орієнтованого підходу дозволило структурувати інтерфейс у вигляді незалежних модулів, що забезпечило гнучкість управління контентом та високу швидкість рендерингу сторінок.

Реалізована логіка взаємодії з REST API через серверні сценарії SvelteKit гарантує безпечне та ефективне отримання даних, а використання механізмів серверного рендерингу забезпечує оптимальні показники продуктивності та SEO-оптимізації. Розроблена кастомна система стилізації на основі SCSS з використанням «гумової верстки» та дробових сіток дозволила досягти повної адаптивності інтерфейсу, зберігаючи візуальну цілісність на пристроях з різною роздільною здатністю екрана.

Ключовим досягненням практичного етапу є створення реактивного середовища для взаємодії з користувачем. Впровадження механізмів фільтрації з синхронізацією через URL-параметри, інтеграція засобів локалізації та використання технік оптимізації клієнтських запитів дозволили створити зручний та інклюзивний інструмент для роботи з меморіальними даними. Технічні рішення, описані в розділі, підтверджують ефективність обраного технологічного стеку та повністю реалізують вимоги до інклюзивності й доступності цифрового ресурсу, що були сформовані на попередніх етапах проєктування.

Також було проведено комплексне ручне тестування функціональної логіки, яке підтвердило точність обробки персональних даних, обчислення віку та відмінювання числівників. Автоматизований аудит у Google Lighthouse зафіксував високі показники якості продукту: доступність – 94, продуктивність – 75, найкращі практики та SEO – 100.

Технічні рішення, описані в розділі, підтверджують ефективність обраного технологічного стеку та повністю реалізують вимоги до інклюзивності й доступності цифрового ресурсу, що були сформовані на попередніх етапах проєктування . Створена платформа є готовим до експлуатації інструментом, що забезпечує гідне представлення пам'яті героїв у сучасному інформаційному просторі.

ВИСНОВКИ

У кваліфікаційній (бакалаврській) роботі було реалізовано повний цикл проєктування та програмної розробки користувацького інтерфейсу вебплатформи «Цифровий меморіал героїв Вінницької ОТГ». Головною метою проєкту стало створення сучасного, швидкого та інклюзивного ресурсу для гідного вшанування пам'яті захисників, що забезпечує ефективний пошук та візуалізацію персональних даних.

В ході виконання роботи був проведений аналіз предметної області, який дозволив визначити специфіку UX-дизайну для меморіальних платформ, де ключовими принципами стали когнітивна простота, мінімалізм та емоційна стриманість інтерфейсу. Також був проведений аналіз існуючих альтернативних рішень, в результаті якого були визначенні їх сильні та слабкі сторони, які необхідно було виправити під час реалізації нашого проєкту.

Також був проведений аналіз та обґрунтовано вибір технологічного стека, зокрема фреймворку SvelteKit, що дозволило досягти:

- Високої продуктивності на слабких пристроях завдяки відсутності Virtual DOM.
- Миттєвої реактивності інтерфейсу завдяки технологій реактивних змінних та сховищ.
- Високої швидкості завантаження вебплатформи та меншої ваги фінальних файлів завдяки компіляції коду.

Для забезпечення динамічності сайту, було розроблено архітектуру клієнт-серверної взаємодії на основі REST API, де сервер SvelteKit виступає посередником для безпечної трансформації даних із бекенду на ASP.NET.

Було впроваджено стратегію серверного рендерингу (SSR), що забезпечило швидке завантаження контенту та 100% індексацію сторінок героїв пошуковими системами. Для покращення SEO використано семантичну розмітку Schema.org та протокол Open Graph.

Щоб забезпечити простіший процес подальшого розширення проєкту, було реалізовано модульну структуру інтерфейсу на основі Svelte-компонентів, що включає систему динамічної багатофакторної фільтрації, гнучкий пошук та два режими відображення контенту – слайдер та плитку.

Окрім того, для забезпечення доскональної відповідності дизайну і адаптивності інтерфейсу, було створено кастомну систему стилізації за допомогою препроцесора SCSS із використанням концепції «гумової верстки» та відносних одиниць rem, що забезпечило пропорційне масштабування інтерфейсу на пристроях із будь-якою роздільною здатністю екрана.

Особливу увагу було приділено питанням інклюзивності та доступності. Розроблений інтерфейс відповідає стандарту WCAG 2.1, що підтверджено результатами автоматизованого аудиту в Google Lighthouse, де показник доступності склав 94/100, а SEO — 100/100.

Завершальне ручне тестування підтвердило стабільність роботи системи, точність обробки персональних даних та коректність функціонування адаптивних меню на мобільних пристроях. Створена платформа є готовим до експлуатації інструментом, що виконує важливу соціальну місію збереження національної пам'яті громади в сучасному цифровому просторі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hoskins A. Digital Memory Studies. 1st ed. London : Taylor & Francis, 2017. 314 p.
2. Garde-Hansen J., Hoskins A., Reading A. Save as... digital memories. London: Palgrave Macmillan, 2009. 176 p.
3. Senior Citizens (Ages 65 and Older) on the Web. Nielsen Norman Group. 2023. URL: https://www.nngroup.com/reports/senior-citizens-on-the-web/ (дата звернення: 05.04.2026).
4. Вплив українського інтернету на суспільство. URL: https://io.ua/vplyv-ukrayinskogo-internetu-na-suspilstvo/ (дата звернення: 05.04.2026).
5. Найцікавіше зі звіту Digital 2025 про взаємодію з цифровими технологіями. MediaMaker. 2025. URL: https://mediamaker.me/najczikavishe-zi-zvitu-digital-2025-pro-vzayemodiyu-z-czyfrovymy-tehnologiyamy-16257/ (дата звернення: 05.04.2026).
6. Цифровий пантеон Дрогобиччини. Дрогобицька міська рада. URL: https://panteon.drohobych-rada.gov.ua/ (дата звернення: 05.04.2026).
7. Книга пам'яті полеглих за Україну. URL: https://memorybook.org.ua/ua (дата звернення: 05.04.2026).
8. Електронний меморіал Броварської громади. Броварська міська рада. URL: https://memorial.brovary-rada.gov.ua/ (дата звернення: 05.04.2026).
9. Hartono E., Clyde W. Website Visual Design Qualities: A Threefold Framework. ACM Transactions on Management Information Systems. 2019. Vol. 10, No. 1. URL: https://dl.acm.org/doi/10.1145/3309708 (дата звернення: 05.04.2026).
10. Andrea Carolina Vanegas Macheta, Alexandra María Silva Monsalve. Development Technologies for Website Implementation. SSRN Electronic Journal. 2022. URL: https://doi.org/10.2139/ssrn.4178359 (дата звернення: 05.04.2026).
11. Raj C. The Principles of Clean Code: DRY, KISS, and YAGNI. Medium. 2020. URL: https://medium.com/@curiousraj/the-principles-of-clean-code-dry-kiss-and-yagni-f973aa95fc4d (дата звернення: 05.04.2026).
12. Web Content Accessibility Guidelines (WCAG) 2.1. W3C Recommendation. 2018. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 05.04.2026).
13. Richardson L., Ruby S. RESTful Web Services. Sebastopol: O'Reilly Media, Inc., 2007. 448 p. URL: http://restfulwebapis.com/RESTful_Web_Services.pdf (дата звернення: 05.04.2026).
14. Foundations of JSON Schema. URL: http://dvrgoc.ing.puc.cl/data/json.pdf (дата звернення: 05.04.2026).

15. Ecomstreet. Exploring the Pros and Cons: Is Svelte Better than React for Web Development? Medium. 2023. URL: https://medium.com/@ecomstreet_18813/exploring-the-pros-and-cons-is-svelte-better-than-react-for-web-development-d9302d4bd6a5 (дата звернення: 05.04.2026).

16. Yesmeno. Svelte components as Web Components. Medium. 2022. URL: https://medium.com/@yesmeno/svelte-components-as-web-components-b400d1253504 (дата звернення: 05.04.2026).

17. Kedzior. Svelte and SvelteKit explained. Dev.to. 2023. URL: https://dev.to/kedzior_io/svelte-and-sveltekit-explained-3513 (дата звернення: 05.04.2026).

18. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: doctoral dissertation. University of California, Irvine, 2000. 162 p. URL: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 05.04.2026).

19. Fowler M. Data Transfer Object. Catalog of Patterns of Enterprise Application Architecture. URL: https://martinfowler.com/eaaCatalog/dataTransferObject.html (дата звернення: 05.04.2026).

20. JavaScript SEO basics. Google Search Central documentation. URL: https://developers.google.com/search/docs/crawling-indexing/javascript/javascript-seo-basics (дата звернення: 05.04.2026).

21. Meyer E. A., Weyl E. P. CSS: The Definitive Guide: Visual Presentation for the Web. 4th ed. O'Reilly Media, 2017. 1088 p. URL: https://books.google.com.ua/books?id=w2U6DwAAQBAJ&printsec=copyright&redir_esc=y#v=onepage&q&f=false (дата звернення: 05.04.2026).

22. Semantics. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/Semantics (дата звернення: 05.04.2026).

23. Introduction. SvelteKit documentation. URL: https://svelte.dev/docs/kit/introduction (дата звернення: 05.04.2026).

24. Component Driven Development. URL: https://www.componentdriven.org/ (дата звернення: 05.04.2026).

25. Whitenton K. Designing Effective Carousels: Create a Favorable First Impression. Nielsen Norman Group. 2023. URL: https://www.nngroup.com/articles/designing-effective-carousels/ (дата звернення: 05.04.2026).

26. Grid Systems. Interaction Design Foundation. URL: https://ixdf.org/literature/topics/grid-systems (дата звернення: 05.04.2026).

27. Cainux. .NET Core and Svelte. Dev.to. 2019. URL: https://dev.to/cainux/net-core-and-svelte-f8o (дата звернення: 05.04.2026).

28. Getting Started. Axios documentation. URL: https://axios-http.com/docs/intro (дата звернення: 05.04.2026).

29. Sass Documentation. URL: https://sass-lang.com/documentation/ (дата звернення: 05.04.2026).

30. Goplani A. Internationalization in SvelteKit with svelte-i18n, sveltekit-i18n and typesafe-i18n. Dev.to. 2023. URL: https://dev.to/aakashgoplani/internationalization-in-sveltekit-with-svelte-i18n-sveltekit-i18n-and-typesafe-i18n-3olj (дата звернення: 05.04.2026)

31. Austin W. Optimizing Performance using Debouncing and Throttling. Dev.to. 2021. URL: https://dev.to/austinwdigital/optimizing-performance-using-debouncing-and-throttling-1b64 (дата звернення: 05.04.2026).

ДОДАТКИ

ДОДАТОК А

Реалізація функціоналу методу для виконання мережевих REST API запитів apiQueryBack у файлі apiWorker.js.

```
import axios from "axios";
import { redirect } from "@sveltejs/kit";
import { API_LINK } from '$env/static/private';
import { AVAILABLE_LANGS } from "$lib/static-values.js";

let domain = '';
try{
    domain = API_LINK??'https://example.com';
}
catch (e) {
    domain = '';
}

/**
 * Query from sveltekit back to api
 * @param data
 * @param cookies
 * @returns {Promise<unknown>}
 */
export function apiQueryBack(data, cookies){
    let lang = cookies.get('lang')??AVAILABLE_LANGS[0];

    data.url = domain + '/' + data.url;
    if(data.url.includes('?')){
        data.url += '&lang=' + lang;
    }
    else data.url += '?lang=' + lang;

    let token = cookies.get('user_token')??false;
    if(token){
        if(!data.headers){
            data.headers = {
                'Authorization' : 'Bearer ' + token,
            }
        }
        else{
            data.headers['Authorization'] = 'Bearer ' + token;
        }
    }
    else{
        data.headers = {};
    }

    data.headers['Accept'] = 'application/json';
    data.headers['Accept-Language'] = lang;

    return axios(data).catch((e) => {
```

```
            console.log(e);
            if(e.response.status === 401){
                redirect(302, '/');
            }
        });
    }
```

ДОДАТОК Б

Файл global.js, в якому реалізується функціонал всіх допоміжних методів, необхідних для трансформації даних і роботи додатку, таких як: route, getFormatedDate, getCorpsCode та інші.

```
import { get } from "svelte/store";
import { page } from "$app/stores";
import { browser } from "$app/environment";
import axios from "axios";

export function route(link){
    let lang = get(page).params.lang??null;

    if(lang??false){
        return '/' + lang + link;
    }

    return link;
}

export async function changeLang(lang){
    try{
        let dat = new FormData();
        dat.set('lang', lang.name);
        await axios({
            url: '/?/change-lang',
            method: 'post',
            data: dat
        });
    }
    catch (e) {}
}

export function setWordByLength(count, wordsArr) {
    let lastNum = count + "";
    lastNum = lastNum.charAt(lastNum.length - 1);
    let result = "";
    if(count > 20) {
        if (lastNum == 0 || lastNum >= 5) {
            result = wordsArr[0];
            return result;
        } else if (lastNum == 1) {
            result = wordsArr[1];
            return result;
        } else if (lastNum >= 2 && lastNum <= 4) {
            result = wordsArr[2];
            return result;
        }
    } else {
        if (count == 1) {
```

```
            result = wordsArr[1];
            return result;
        } else if (count >= 2 && count <= 4) {
            result = wordsArr[2];
            return result;
        } else {
            result = wordsArr[0];
            return result;
        }
    }
}

export function buildFormData(formData, data, parentKey) {
    if (data && typeof data === 'object' && !(data instanceof
Date) && !(data instanceof File) && !(data instanceof Blob)) {
        Object.keys(data).forEach(key => {
            buildFormData(formData, data[key], parentKey ?
`${parentKey}[${key}]` : key);
        });
    } else {
        const value = data == null ? '' : data;

        formData.append(parentKey, value);
    }
}

export function onBack(e) {
    if(browser){
        console.log(window.history);
        if(window.history.length > 1) {
            window.history.back();
        } else {
            window.location.href = e.target.href;
        }
    }
}

export function getFormatedDate(date){
    date = new Date(date);

    let day = date.getDate();
    day = day < 10?`0${day}`:day.toString();

    let month = date.getMonth() + 1;
    month = month < 10?`0${month}`:month.toString();

    let year = date.getFullYear().toString();

    return `${day}.${month}.${year}`;
}

export function getCorpsCode(corps){
```

```
    const corpsAlias = {
        'гвардія': 'national-guard',
        'guard': 'national-guard',

        'морські': 'navy',
        'navy': 'navy',

        'ГУР': 'gur',
        'Defence': 'gur',

        'прикордон': 'border-guard',
        'border': 'border-guard',

        'десант': 'air-assault',
        'air assault': 'air-assault',

        'правий': 'right-sector',
        'right': 'right-sector',

        'поліц': 'national-police',
        'police': 'national-police',

        'повітря': 'air-force',
        'air': 'air-force',

        'сухопут': 'ground-forces',
        'ground': 'ground-forces',

        'спеціальних': 'special',
        'special': 'special',

        'тро': 'territorial',
        'ter': 'territorial',
    };

    let corpsCode = 'ground-forces'
    for(let key in corpsAlias){
        if(corps.toLowerCase().includes(key.toLowerCase())){
            corpsCode = corpsAlias[key];
            break;
        }
    }

    return corpsCode;
}
```