**Київський національний торговельно-економічний університет**
**Кафедра інженерії програмного забезпечення та кібербезпеки**

# ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

**на тему:**

## «Проектування автоматизованої системи ведення електронних звернень та петицій»

Студента 2м курсу, 2з групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Мельниченка Мирослава Сергійовича

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Сашньова Мар'яна Василівна

Гарант освітньої програми доктор економічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Токар Володимир Володимирович

КИЇВ – 2021

**Київський національний торговельно-економічний університет**

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

**Затверджую**

Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«29» грудня 2020 р.

# Завдання

## на випускну кваліфікаційну роботу студентові

**Мельниченку Мирославу Сергійовичу**
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи**: «**Проектування автоматизованої системи обробки електронних петицій та звернень»
Затверджена наказом ректора від «28» грудня 2020 р. № 3923

2. Строк здачі студентом закінченої роботи 02.11.2021р.

3. Цільова установка та вихідні дані до роботи

*Мета роботи* — покращити практичну реалізацію інструментів електронної демократії, зокрема шляхом проєктування технічних рішень таких реалізацій — створити проєкт автоматизованої системи обробки електронних звернень та петицій.

*Об'єкт дослідження* — електронні звернення та петиції

*Предмет дослідження* — методи проєктування автоматизованої системи обробки електронних звернень та петицій

4. Консультанти роботи із зазначенням розділів, які консультують:

| Розділ | Консультант (прізвище, ініціали) | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| 1 | | | |
| 2 | | | |
| 3 | | | |

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1. Електронні звернення та петиції як інструменти електронної демократії

1.2. Дослідження існуючих автоматизованих систем обробки електронних звернень та петицій

1.3. Висновки до розділу 1

РОЗДІЛ 2 ОБГРУНТУВАННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЗАСОБІВ РЕАЛІЗАЦІ

2.1. Вибір фреймворку для реалізації клієнтської частини

2.2. Вибір фреймворку для реалізації серверної частини

2.3. Вибір бази даних

2.4. Висновки до розділу 2

РОЗДІЛ 3 ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБРОБКИ ЕЛЕКТРОННИХ ЗВЕРНЕНЬ ТА ПЕТИЦІЙ

3.1. Проєктування архітектури системи

3.2. Прототипування графічного інтерфейсу користувача

3.3. Документування програмного забезпечення

3.4. Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання роботи

| № | Назва етапів випускної кваліфікаційної роботи | Термін виконання етапів роботи | |
|---|---|---|---|
| | | за планом | фактично |
| 1 | 2 | 3 | 4 |
| 1 | Вибір теми випускного кваліфікаційного | 21.11.2020 | 21.11.2020 |
| 2 | Розробка та затвердження завдання на проект магістра | 29.12.2020 | 29.12.2020 |
| 3 | Вступ та перелік літературних джерел | 27.02.2021 | 27.02.2021 |
| 4 | Розробка технічного завдання | 20.03.2021 | 20.03.2021 |
| 5 | Розділ 1. Аналіз предметної області та існуючих рішень | 16.04.2021 | 16.04.2021 |
| 6 | Написання наукової статті | 22.05.2021 | 22.05.2021 |
| 7 | Розділ 2. Обґрунтування проєктних рішень та засобів реалізації | 24.05.2021 | 24.05.2021 |
| 8 | Розділ 3. Проєктування автоматизованої системи обробки електронних звернень та петицій | 21.06.2021 | 21.06.2021 |
| 9. | Розробка програми та методики тестування | 18.10.2021 | 18.10.2021 |
| 10 | Висновки та пропозиції | 01.11.2021 | 25.10.2021 |
| 11 | Здача випускного кваліфікаційного проекту на кафедру (перша перевірка) | 03.11.2021 | 03.11.2021 |
| 12 | Підготовка автореферату та презентації Доповіді | 03.11.2021 | 03.11.2021 |
| 13 | Попередній захист випускного кваліфікаційного проекту | 24.11.2021 | 24.11.2021 |
| 14 | Здача зброшурованої випускного кваліфікаційного проекту | 25.11.2021 | 25.11.2021 |
| 15 | Зовнішнє рецензування випускного кваліфікаційного проекту | 26.11.2021 | 26.11.2021 |
| 16 | Підготовка до публічного захисту випускного кваліфікаційного проекту | | |

1. Дата видачі завдання «29» грудня 2020 р.

2.Науковий керівник випускного кваліфікаційного проєкту Сашньова М. В.
(прізвище, ініціали, підпис)

3.Гарант освітньої програми Токар В.В.
(прізвище, ініціали, підпис)

4. Завдання прийняв до виконання студент Мельниченко М. С.
(прізвище, ініціали, підпис)

## АНОТАЦІЯ

Відповідно до мети робота присвячена проектуванню автоматизованої системи обробки електронних звернень та петицій. У рамках роботи досліджено теоретичні аспекти предметної області а також методи та засоби практичної реалізації.

В результаті порівняння інструментів практичної реалізації, було обрано Angular для створення клієнтської частини, Node.Js + Express для серверної частини, та MongoDB у якості бази даних.

Було визначено структуру архітектурного рішення та розроблено прототипи графічного інтерфейсу користувача.



**Ключові слова: електронна демократія, електронне звернення, електронна петиція, автоматизована система обробки**



## ABSTRACT

In accordance with the purpose, the work is devoted to the design of an automated system for processing electronic appeals and petitions. The theoretical aspects of the subject area as well as methods and means of practical implementation are investigated in the framework of the work.

As a result of comparing the tools of practical implementation, Angular was chosen to create the client part, Node.Js + Express for the server part, and MongoDB as the database.

The structure of the architectural solution was determined and prototypes of the graphical user interface were developed.



**Key words: e-democracy, e-appeal, e-petition, automated processing system**

**ЗМІСТ**



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | *КНТЕУ 121 02з-11.МР* | | | |
| *Зм.* | *Аркуш* | *№ докум* | *Підпис* | *Дата* | | | | |
| *Зав. кафедри* | | *Криворучко О.В.* | | *29.12.20* | *Проектування автоматизованої системи обробки електронних звернень та петицій* | *Стадія* | *Аркуш* | *Аркушів* |
| *Керівник* | | *Сашньова М. В.* | | *29.12.20* | | *З* | *2* | *40* |
| *Гарант* | | *Токар В.В.* | | *29.12.20* | | Факультет інформаційних технологій, 2 курс, 2м група | | |
| *Розробив.* | | *Мельниченко М. С.* | | *29.12.20* | *Зміст* | | | |

# ВСТУП

*Актуальність.* Стрімкий розвиток інформаційно-комунікаційних технологій призвів до бурхливого розвитку мережі Інтернет на початку 21 сторіччя. Пройшовши розвиток від невеликої мережі з обмеженої кількістю вузлів та вузьким колом користувачів до сьогодення, коли мережа налічує безліч вузлів, а кількість постійних користувачів обчислюється мільярдами, Інтернет беззаперечно став глобальним явищем, яке вплинуло не лише на окремих людей, а на суспільство в цілому, зробивши можливими зміни у суспільних відносинах та інститутах. Створюючись як система передачі інформації на випадок війни, врешті-решт Інтернет вийшов далеко за рамки цієї ідеї: крок за кроком напрямки застосування мережі збільшувались, вона стала інструментом та способом для комунікації, розваг, ведення бізнесу і, врешті-решт, черга дійшла до політико-правових процесів у суспільстві — у різних країнах почали здійснюватися реалізації ідеї електронної демократії, серед інструментів якої без перебільшення важливе місце займають електронні звернення та петиції. Хоча саму ідею електронної демократії важко назвати новою, її широке впровадження розпочалось відносно недавно, тому дослідження у напрямку проєктування автоматизованих систем обробки електронних звернень та петицій як інструменту електронної демократії, є досить актуальним, особливо для України, де на форумі, приуроченому 30-річчю держави, впровадження інструментів електронної демократії було названо одним із головних напрямків розвитку країни.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | КНТЕУ 121 06-07.МР | | |
| Зм. | Аркуш | № докум | Підпис | Дата | | | |
| Зав. кафедри | | Криворучко О.В. | | 27.02.21 | | | |
| Керівник | | Сашньова М. В. | | 27.02.21 | Проектування автоматизованої системи обробки електронних звернень та петицій | Стадія | Аркуш | Аркушів |
| Гарант | | Токар В.В. | | 27.02.21 | | В | 4 | 40 |
| Розробив. | | Мельниченко М. С. | | 27.02.21 | Вступ | Факультет інформаційних технологій, 2 курс, 2м група | | |

*Мета* роботи — покращити практичну реалізацію інструментів електронної демократії, зокрема шляхом проєктування технічних рішень таких реалізацій — створити проєкт автоматизованої системи обробки електронних звернень та петицій.

*Об'єкт* дослідження — електронні звернення та петиції.

*Предмет* дослідження — методи проєктування автоматизованої системи обробки електронних звернень та петицій.

*Завдання:*

1. дослідити електронні звернення та петиції як інструменти електронної демократії;
2. дослідити існуючі автоматизовані системи обробки електронних звернень та петицій;
3. обґрунтувати обрання фреймворку для клієнтської частини;
4. обґрунтувати обрання фреймворку для серверної частини;
5. обґрунтувати обрання бази даних;
6. обґрунтувати рішення проєктування системи;
7. розробити прототип графічного інтерфейсу;
8. задокументувати програмне забезпечення;

*Методи дослідження, використані у роботі:* методи галузі технологій програмування та загальнонаукові методи аналізу та синтезу.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

## 1.1. Електронні звернення та петиції як інструменти електронної демократії.

Поняття електронної демократії не нове для наукового співтовариства. Дослідженням питань, пов'язаних з детермінуванням поняття електронної демократії, структуризацією цього явища та особливостями його практичного впровадження присвячено багато робіт, як зарубіжних, так і вітчизняних авторів, однак ці роботи мають здебільшого політико-правовий характер, тому не є доцільним детально розглядати їх у рамках цієї роботи.

Для визначення поняття електронної демократії варто звернутися до чинного законодавства. Так, відповідно до Стратегії розвитку інформаційного суспільства в Україні, електронна демократія - форма суспільних відносин, за якої громадяни та організації залучаються до державотворення та державного управління, а також до місцевого самоуправління шляхом широкого застосування інформаційно-комунікаційних технологій [1].

Так, до переліку інструментів електронної демократії належать електронні звернення та електронні петиції. Для кращого розуміння предметної області необхідно детальніше розібратися з термінологічною складовою. Відповідно до чинного законодавства, електронні звернення це

письмове звернення, надіслане з використанням мережі Інтернет, засобів електронного зв'язку. Електронне звернення визначається як особлива форма колективного звернення громадян до Президента України, Верховної Ради України, Кабінету Міністрів України або органу місцевого самоврядування. Так, електронні звернення — це індивідуальні звернення осіб до суб'єктів владних повноважень, які за своєю суттю можуть бути пропозицією, клопотанням чи скаргою. На відміну від електронних звернень, електронні петиції — це звернення колективні, що за своїм змістом також можуть бути пропозицією, клопотанням або скаргою. Законодавством передбачено, що подати електронну петицію можна через офіційний вебсайт органу, якому вона адресована, або вебсайт громадського об'єднання, яке здійснює збір підписів на підтримку електронної петиції. Щодо електронних звернень окремі положення, присвячені засобам подання таких звернень, відсутні. Зазвичай такі звернення подаються на офіційну електронну пошту адресата звернення. Вищезазначені визначення понять та вимоги до електронних звернень та петицій регулюються чинним законодавством, зокрема Законом України «Про звернення громадян»[2].

Попри те, що електронна петиція є надзвичайно прогресивним інститутом у сучасності демократій, практика їх функціонування в Україні не є ефективною. З одного боку, Україна посідає 32 місце в ТОП-50 виконавців електронної участі у 2016 році за версією Огляду Організації Об'єднаних Націй щодо електронного уряду 2016 р. [3], але багато дослідників вказують на загрози та слабкості функціонування електронних петицій в Україні. Володимир Решота, наприклад, вважає що інструментом маніпулювання може стати існуюче регулювання електронних петицій і популяризація популістських ідей, які не мають нічого спільного з

верховенством права [4]. Анастасія Костянтинівська зазначає, що для того, щоб стати дійсно ефективними, електронні петиції мають бути регулюватися більш прозоро та відкрито, і громадяни мають здійснювати ефективний контроль за процесом звернення [5].

Електронна петиція як елемент електронної демократії сприяє: зміцненню участь, ініціативі та залученню громадян на національному, регіональному та місцевому рівнях суспільного життя; підвищенню прозорості демократичного процесу прийняття рішень та підзвітності демократичних інститутів; сприяє суспільній дискусії і увазі громадян до процесу прийняття рішень. Тому можна констатувати, що електронні петиції є одним з найважливіших інструментів електронної демократії.

## 1.2. Дослідження існуючих автоматизованих систем обробки електронних звернень та петицій

Для аналізу та порівняння існуючих рішень розглянемо наступні вебзастосунки:

1. Офіційне інтернет-представництво Президента України;
2. Сервіс «Електронні петиції» Верховної Ради України;
3. Сервіс «Електронні петиції» Київської міської ради.

Офіційне інтернет-представництво Президента України має сучасний інтерфейс.

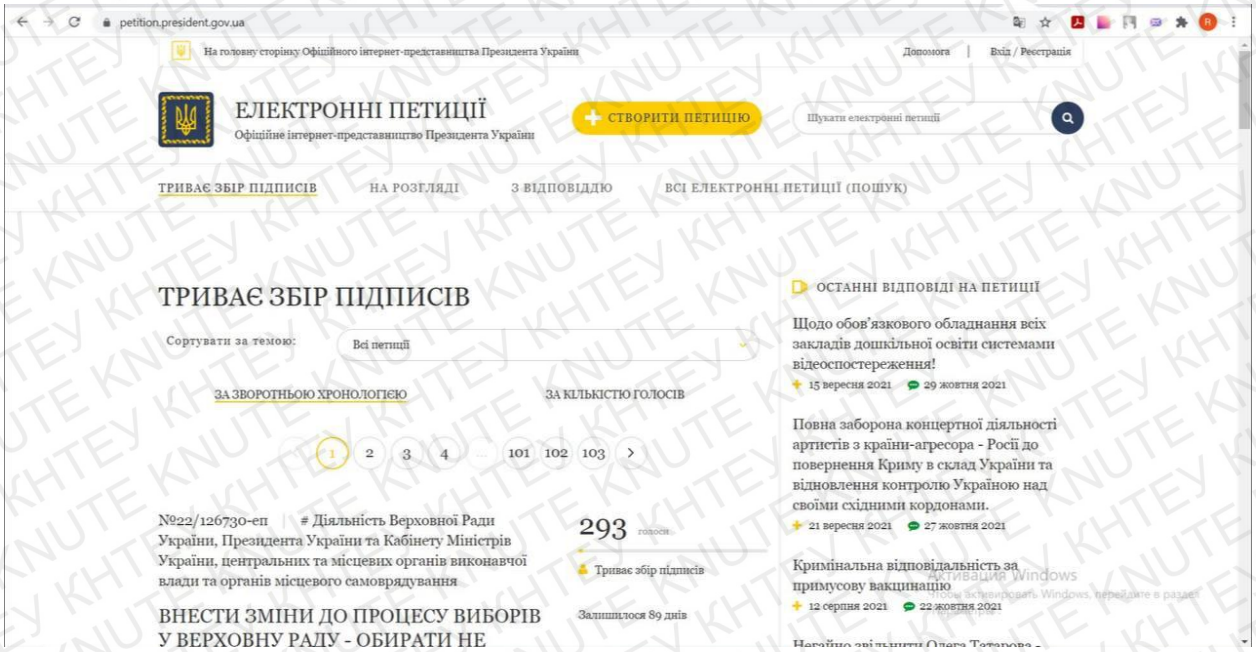


*Рис. 1.1. Інтерфейс інтернет-представництва Президента України*

Для створення або підписання існуючої петиції необхідно зареєструватись. Реєстрація відбувається за допомогою запропонованих рішень:

1. з використанням електронного цифрового підпису
2. з використанням Bank ID.

Інтерфейс створення петиції простий та зручний, необхідно ввести текст петиції та її назву, після чого підтвердити правильність введених даних натиском відповідної кнопки.

Для пошуку між усіх існуючих петицій існує спеціальний розділ «Всі електронні петиції». У цьому розділі можна відфільтрувати та відсортувати вже зареєстровані петиції. Фільтрація відбувається за наступними параметрами:

1. назва петиції;

2. тема петиції;
3. ПІБ автора петиції;
4. номер електронної петиції;

Відфільтровані петиції можна відсортувати за:

1. хронологією;
2. кількістю голосів (підписів);

Сервіс «Електронні петиції» Верховної Ради України має застарілий інтерфейс.

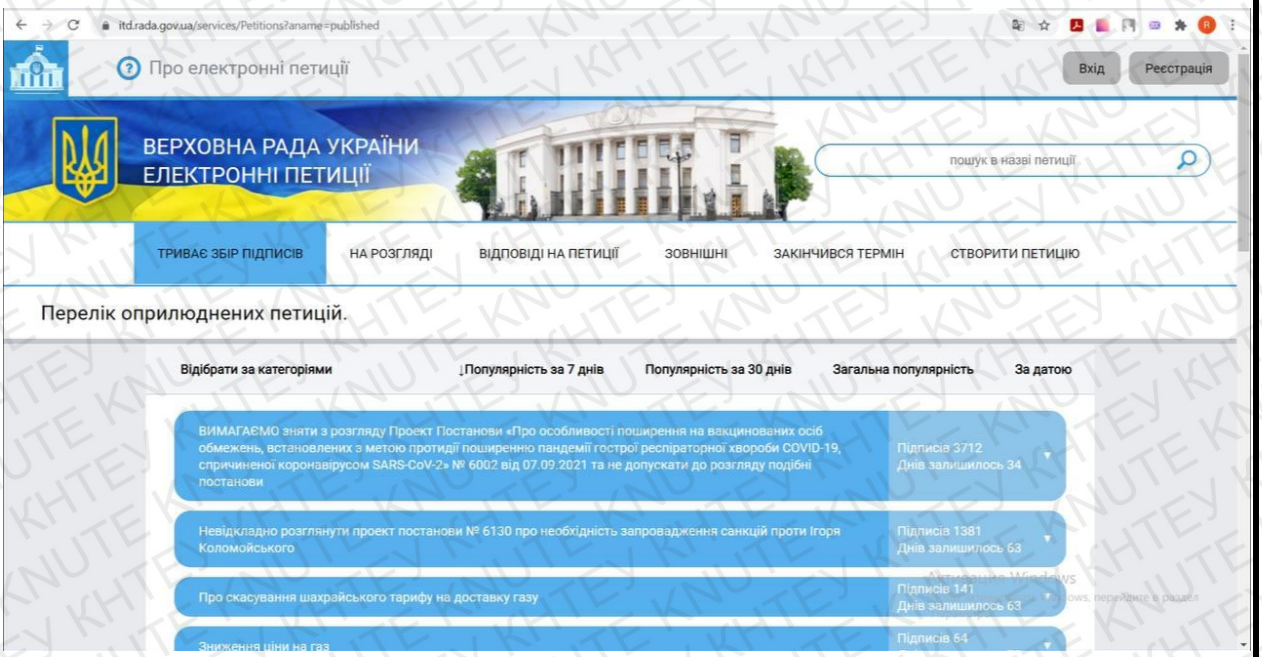


*Рис. 1.2. Інтерфейс сервісу «Електронні петиції» Верховної Ради України*

Для створення або підписання петицій необхідно авторизуватися. У випадку реєстрації сервіс потребує заповнення форми за наступними обов'язковими полями:

1. Прізвище;
2. Ім'я;
3. Електронна пошта;

4. Мобільний телефон;
5. Регіон;
6. Населений пункт;
7. Вулиця;
8. Будинок;

Після підтвердження реєстрації на вказаний номер мобільного телефону приходить SMS-сповіщення з кодом підтвердження реєстрації. Після вводу коду сервіс перенаправляє на сторінку з повідомленням про те, що для завершення реєстрації необхідно перейти за посиланням, надісланим на вказану адресу електронної пошти.

Для створення нової петиції необхідно заповнити форму з такими ж полями, як і у попередньому рішенні.

Пошук петицій не дуже зручний, реалізований у вигляді окремих розділів за статусом петицій, серед яких існують наступні статуси:

1. триває збір підписів;
2. на розгляді;
3. відповіді на петиції;
4. зовнішні;
5. закінчився термін;

Для кожного розділу реалізовано не однакові механізми фільтрації та сортування. Так, петиції зі статусом «триває збір підписів», «на розгляді» можна відфільтрувати за категоріями (темами петицій) та відсортувати за:

1. популярністю за 7 днів;
2. популярністю за 30 днів;
3. загальною популярністю;
4. за датою;

Петиції зі статусом «відповіді на петиції» та «закінчився термін» можна відфільтрувати за категоріями та відсортувати лише за датою. Для петицій зі статусом «зовнішні» фільтрація чи сортування не реалізоване. Окремо реалізований пошук петицій за назвою.

Сервіс «Електронні петиції» Київської міської ради має сучасний інтерфейс.

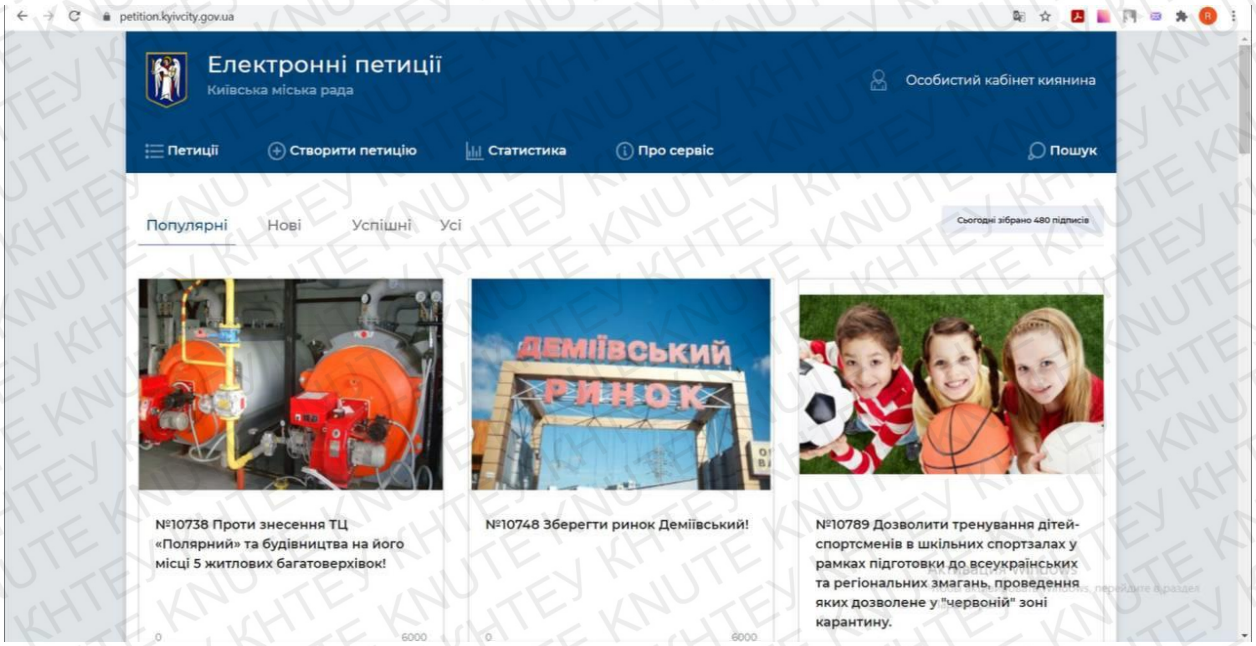


*Рис. 1.3. Інтерфейс сервісу «Електронні петиції» Київської міської ради*

Для подання нової петиції чи підписання існуючої необхідно пройти аутентифікацію через особистий кабінет киянина — Київ ID. Сервіс пропонує наступні засоби аутентифікації:

1. Bank ID;
2. Mobile ID;
3. ЕЦП;

Процес створення петиції не має суттєвих відмінностей від інших сервісів.

Однак пошук між наявних петицій реалізовано найбільш зручно для користувача. Так, зареєстровані на сервісі петиції можна фільтрувати за:

1. напрямком (темою) петиції;
2. автором;
3. статусом петиції;
4. кількістю підписів;
5. ключовими словами;
6. періодом подання петиції ( проміжок між двома датами);

Сортування відбувається декрементно за:

1. новизною;
2. популярністю;
3. успішністю;

Окремо реалізований пошук петицій за словами у назві.

## 1.3. Висновки до розділу 1

Електронні звернення та петиції є важливими інструментами електронної демократії. Хоча в Україні існує нормативно-правове регулювання цих інститутів, однак воно не охоплює технічні вимоги до практичної реалізації, що призводить до відсутності уніфікованого підходу проєктування та розробки технічних рішень: наявні рішення відрізняються не лише зручністю використання, а і за функціоналом. Хоча усі наявні рішення

представлені виключно у вигляді веб застосунків, вони охоплюють лише електронні петиції, у той час для електронних звернень подібні рішення відсутні — вони подаються у виглядів електронних листів. Тому у рамках цієї роботи вважається за доцільне створення єдиного проєктного рішення як для електронних петицій, так і для електронних звернень.

# РОЗДІЛ 2
# ОБГРУНТУВАННЯ ПРОЄКТНИХ РІШЕНЬ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ

## 2.1. Вибір фреймворку для реалізації клієнтської частини

Головна вимога до вебсайту – він повинен працювати швидко, якісно, надійно та мати зрозумілий інтерфейс.

Однак розробка якісного вебсайту залишається непростим завданням. Найчастіше це пов'язано з тим, що розробники використовують застарілі системи розробки, яких не можуть відмовитися, тому що вони є звичними та надійними. Технологія SPA дозволяє використовувати сучасні засоби розробки вебсайтів.

SPA (Single Page Application) — це вебдодаток або вебсайт, в якому використовується лише один HTML-документ як оболонка для всіх вебсторінок. Цей HTML-документ організовує взаємодію з користувачем через HTML-коди, що динамічно підвантажуються, CSS-стилі та JavaScript-код за допомогою технології AJAX. В результаті при оновленні даних вебсторінка не перезавантажується повністю, і, як наслідок, вебпрограми працюють швидше і стають зручнішими.

На даний момент кілька великих компаній надають різні засоби та технології в галузі SPA. До найбільших і популярних відносяться компанії Google і Facebook, які є розробниками таких SPA-продуктів, як Angular і React.

React - це JavaScript-бібліотека з відкритим вихідним кодом для розробки інтерфейсів користувача. Ця бібліотека розробляється та

підтримується компанією Facebook та спільнотою окремих розробників та корпорацій. Бібліотека React може бути успішно використана для розробки односторінкових та мобільних додатків.

Бібліотека React створена розробником програмного забезпечення Джорданом Валке і вперше була згадана у стрічці новин у 2011 році. Вихідний код був представлений у 2013 році на конференції JSConf US.

В основу розробки бібліотеки покладено компонентний підхід, що дозволяє створювати інкапсульовані компоненти, які мають власний стан, і об'єднувати ці компоненти в складні інтерфейси користувача. Оскільки логіка самих компонентів реалізована мовою програмування JavaScript, можна передавати дані по всьому додатку, не прив'язуючись до інтерфейсу DOM.

Однією з ключових особливостей бібліотеки є можливість використання препроцесора JSX, із синтаксисом, схожим з мовою розмітки HTML, яка перетворюється на код мови програмування JavaScript. Препроцесор JSX дозволяє описувати структуру інтерфейсу. Як правило, при використанні бібліотеки React для побудови компонентів використовують саме цей препроцесор, але можна використовувати так само і чистий JavaScript код.

До особливостей бібліотеки також належать можливість використання односпрямованої передачі. Властивості можуть передаватися від батьківських компонентів до дочірніх. Таким чином, компоненти отримують властивості як безлічі постійних значень, завдяки цьому виключається можливість змінювати властивості компонента при передачі даних, але залишається можливість зміни властивостей через функції зворотного виклику. Такий механізм називають «властивості донизу, події нагору».

Бібліотека React працює із використанням віртуального інтерфейсу DOM (Document Object Model). У пам'яті створюється кеш-структура, яка дозволяє обчислювати різницю попереднього та поточного стану

інтерфейсу. Таким чином, у розробника з'являється можливість працювати зі сторінкою, припущення, що вона повністю оновлюється, але які компоненти оновлювати, вирішує сама бібліотека. У свою чергу компоненти мають життєвий цикл, для кожного циклу реалізовані свої методи, які дозволяють розробникам запускати код на різних стадіях життєвого циклу компонента. Наведемо приклад:

• метод повиненComponentUpdate дозволяє скасувати перемальовку компонента, якщо вона не потрібна;

• метод componentDidMount викликається після першого відтворення компонента; зазвичай використовується для запиту даних із сервера;

• метод render – найважливіший метод життєвого циклу компонента; викликається щоразу за зміни даних для перемальовування даних у інтерфейсі.

Головною метою бібліотеки є надання високої швидкості доступу, простоти та масштабованості. Також вона дозволяє використовувати інші бібліотеки для розробки інтерфейсів, такі, як бібліотека для управління станом програми Redux або бібліотеки для маніпулювання даними GraphQl.

Angular — це платформа для розробки вебзастосунків, написана мовою програмування TypeScript. Розроблена та підтримується командою з Google, а також спільнотою розробників з різних компаній. Створювалася як нова версія фреймворку AngularJS, однак, нова версія була повністю переписана мовою програмування TypeScript і має іншу архітектуру, через що не має зворотної сумісності з попередньою версією.

Платформа не тільки надає інструменти, а й дизайни шаблону, що дозволяє створювати проєкт, що обслуговується і розширюється. Правильне дотримання архітектури програми, що надається під час створення, дозволяє уникнути плутанини у класах та методах.

Платформа Angular використовує розмітку HTML для визначення інтерфейсу користувача. Розмітка HTML є інтуїтивнішим і менш складним засобом, ніж процедурне визначення інтерфейсу в мові програмування JavaScript, що дозволяє залучити більше розробників.

Маніпуляція з об'єктами інтерфейсу DOM для представлення даних є частиною поведінки платформи. Код маніпулювання інтерфейсом DOM повинен бути всередині директив, а не у поданні. Це дуже спрощує роботу, зокрема через те, що платформа бачить подання як сторінку з кодом HTML із заповненими даними. Абстрагуючись від маніпуляцій інтерфейсу DOM, розробники можуть зосередитися на представленні інтерфейсу користувача без цих відволікаючих факторів.

У платформі Angular реалізовані контролери – прості функції, які мають лише одне завдання – маніпулювати областю дії. Наприклад, з'являється можливість використовувати платформу, щоб попередньо заповнити дані в області сервера або реалізувати перевірку бізнес-логіки. На відміну від інших середовищ, контролери не є об'єктами і не є нащадками інших об'єктів.

Модульне тестування, реалізоване у платформі, є її винятковою особливістю. Модульне тестування істотно перевершує традиційний спосіб, оскільки дозволяє створити окрему тестову сторінку, у якій перевіряє працездатність компонента.

Платформа Angular та бібліотека React є безкоштовними, і обидві користуються популярністю серед програмістів у галузі розробки вебсайтів та вебдодатків. Розглянемо переваги та недоліки кожної з платформ.

• При розробці вебсайту або вебдодатку найпотрібнішим властивістю інструмента є його самодостатність. Бібліотека React є основою для розробки інтерфейсу користувача, але потребує додаткових бібліотек, у той час як

платформа Angular є повноцінним фреймворком, який зазвичай не вимагає додаткових бібліотек.

• Поріг входження є головним фактором при виборі інструменту у розробників-початківців. Бібліотека React є досить простою у використанні: немає впровадження залежностей, немає класичних шаблонів, немає надто складних функцій. Бібліотека досить проста для освоєння, якщо розробник вже знайомий із мовою програмування JavaScript.

• Документація та підтримка також сильно впливає на вибір інструмента для розробників. Працюючи з бібліотекою React, ви повинні бути постійним учнем, оскільки структура часто оновлюється. У той час, як спільнота розробників намагається якнайшвидше просувати останню документацію, йти в ногу з усіма змінами не так просто. Платформа Angular викликає великий скептицизм, частково через непопулярність першої версії платформи Angular 1.0. Розробники звикли відхиляти фреймворк як надто складний, тому що на його вивчення витрачалося багато часу. Тим не менш, ця структура розроблена компанією Google, що говорить на користь вибору платформи Angular.

• Структура бібліотеки React надає розробникам свободу вибору. Немає єдиної правильної структури для програми, розробленої за допомогою бібліотеки React. У той же час, структура платформи Angular є фіксованою та складною, більш придатною для досвідчених розробників.

• Найбільшою перевагою платформи Angular є те, що, на відміну від бібліотеки React, він підтримує впровадження залежностей. Тому платформа Angular дозволяє мати різні цикли життя для різних компонентів.

• Бібліотека React використовує віртуальну об'єктну модель документів, яка дозволяє легко реалізовувати незначні зміни даних в одному елементі без поновлення структури всього дерева. У той час як платформа Angular

використовує реальну об'єктну модель документів, яка оновлює всю деревоподібну структуру, навіть коли зміни відбулися в одному елементі.

• Можна, мабуть, вважати, що основною ознакою будь-якого успішного інструменту є його продуктивність. Продуктивність бібліотеки React значно покращилась із запровадженням віртуальної об'єктної моделі документів. Оскільки всі віртуальні моделі легкі та побудовані на сервері, навантаження на браузер знижується. Платформа Angular працює гірше, особливо у разі складних та динамічних вебдодатків. На продуктивність додатків Angular негативно впливає двонаправлена прив'язка даних. Кожній прив'язці призначається спостерігач для відстеження змін, і кожен цикл триває доти, доки не будуть перевірені всі спостерігачі та пов'язані значення.

Розробка вебсайтів та вебдодатків за допомогою технології SPA на сьогоднішній день дуже популярна та стрімко розвивається, що мотивує розробників створювати та розвивати інструменти, які працюють на технології SPA. Бібліотека React та платформа Angular користуються великою популярністю серед розробників. Вони обидві гарні в області SPA, і кожна має відмінні риси. Але для розробників-початківців, на нашу думку, краща платформа Angular, оскільки, серед іншого, має менш низький поріг входження і більш зрозумілу документацію, а вбудоване використання TypeScript допоможе знизити кількість помилок під час розробки.

## 2.2. Вибір фреймворку для реалізації серверної частини

Node.js є серверною платформою, яка є частиною стека технологій, що охоплюють всі потреби веброзробки, і засновані на JavaScript. У Node.js використовується JavaScript-рушій V8, той же, що застосовується в браузері

Chrome і в інших браузерах, заснованих на Chromium. Внаслідок цього виявляється, що завдяки використанню Node.js код, призначений для виконання на сервері, можна писати на JavaScript. На базі платформи Node.js створено безліч фреймворків, включаючи такий популярний як Express.

Переваги:

1. Поява Node.js уможливила фуллстек-розробку вебпроєктів на JavaScript. В результаті в розпорядженні розробників серверних частин програм виявилися і сильні можливості JavaScript, і напрацювання екосистеми JS, бібліотеки, якими стало реально скористатися в серверному оточенні.
2. JavaScript-код, аналогічний за функціоналом, наприклад, кодом, написаним на C, виявляється компактнішим. Продуктивність JavaScript-коду при цьому досить висока для застосування його у проєктах, у яких важлива швидкість роботи коду.
3. Код клієнтських і серверних частин проєктів легше підтримувати в узгодженому стані, так як і там і там використовується та сама мова.
4. Один і той самий код можна спільно використовувати і для клієнта, і на сервері.
5. Завдяки існуванню модулів Node.js, які, по суті, є особливим чином оформленими фрагментами коду, розробники можуть зручно використовувати у своїх проєктах чужий код, а також власні напрацювання.
6. Платформа Node.js, і, відповідно, засновані на ній фреймворки, відрізняються невибагливістю ресурсів і масштабованістю. Саме тому Node.js це платформа, до якої часто вдаються ті, хто користується мікросервісними архітектурами.
7. Ця платформа добре підходить для розробки мікросервісів ще й через існування системи модулів Node.js, які можна уявити у вигляді будівельних блоків серверних програм.

8. У Node.js JavaScript код компілюється в машинний код, що дозволяє отримати набагато більшу продуктивність, ніж при інтерпретації коду. Спільнота JavaScript-
9. розробників бачить постійне покращення продуктивності Node.js за рахунок того, що Google постійно працює над удосконаленням V8.
10. Завдяки тому, що в Node.js є система введення-виводу, яка не блокує головний потік, ця платформа демонструє високу продуктивність. Достойна швидкість обробки запитів досягається завдяки використанню JavaScript-механізмів конкурентного виконання однопоточного коду.
11. Node.js - це опенсорсний проєкт, навколо якого зібралося величезне співтовариство розробників. Це означає, що той, хто зіткнеться з якоюсь проблемою, зможе досить швидко знайти її рішення.

Проєкт Spring Boot — це фреймворк для розробки бекенд-застосунків, заснований на Java, який, як і Node.js, використовується для розробки мікросервісів. Цей фреймворк спрощує створення додатків, заснованих на Spring, його можна уявити у вигляді інструмента створення самостійних Spring-додатків. Якщо ви плануєте в 2021 році перейти на Spring, то вам, безперечно, варто знати про те, чим вам у цій справі зможе допомогти Spring Boot.

Переваги:

1. Spring Boot дозволяє з мінімальними зусиллями створювати самостійні Spring-програми, полегшує процес їх конфігурування, спрощує роботу над ними. Подібні програми легко запускаються за допомогою команди java-jar.
2. Якщо в процесі створення Spring Boot-програми сталася помилка, вбудований аналізатор помилок допоможе впоратися з проблемою.

3. Spring Boot підтримує інтегровані сервери, як Tomcat і Jetty. Це означає, що тим, хто користується Spring Boot, не потрібно розгортати .war-файли на зовнішніх серверах.
4. Використання Spring Boot дозволяє полегшити конфігурацію Maven за рахунок наявності у системі початкових варіантів файлу pom.xml.
5. У можливості фреймворку входить автоматичне конфігурування Spring.
6. Spring Boot добре інтегрується з іншими фреймворками.
7. Фреймворк надає розробнику конфігурації, готові до продакшн-використання. Сюди входять, наприклад, метрики стану проєкту та зовнішні конфігурації.
8. При використанні Spring Boot немає потреби в застосуванні конфігурацій XML або засобів для генерування коду.
9. Застосування Spring Boot полегшує працю розробників за рахунок застосування принципу проєктування ПЗ, відомого як «Convention over Configuration».

Django – це опенсорсний бекенд-фреймворк, написаний на Python, що має наступні переваги:

1. Django дозволяє без особливих труднощів створювати динамічні вебпрограми з використанням Python. Цей фреймворк написаний на Python. У цьому полягає одна з головних переваг Django.
2. Фреймворк підтримує патерн проєктування MVC. Це допомагає розробникам у розділенні інтерфейсу користувача та бізнес-логіки Django-додатків.

3. Це швидкий фреймворк, який не перевантажений непотрібними можливостями. Я маю на увазі те, що використання Django дозволяє швидко вийти на працездатний проєкт.
4. Django не відноситься до мінімалістичних фреймворків, що широко використовуються для розробки мікросервісів. Він відрізняється потужністю, універсальністю та певною своєрідністю.
5. Автори цього фреймворку серйозно ставляться до безпеки. Тому вони пропонують розробникам, які використовують Django, відповідні інструменти. Я впевнений, що всі ви знаєте про те, як багато проблем існує в наші дні у сфері кібербезпеки. Тому надзвичайно важливими є питання захисту вебпроєктів. Django підтримує систему автентифікації користувачів, містить інструменти захисту від різних атак. Серед них засоби захисту від SQL-ін'єкцій, від міжсайтового скриптингу, від міжсайтової підробки запитів, від клікджекінгу.
6. Django проєкти відрізняються компактністю коду.
7. Розробники Django можуть моделювати базові класи. Це означає, що в їхньому розпорядженні завжди є ORM.
8. Django – це крос-платформний проєкт. Він добре працює на різних операційних системах. Крім того, він підтримує взаємодію з різними базами даних.
9. Це — фреймворк, програми, створені з використанням якого добре піддаються масштабуванню. Тому той, хто обирає Django, може бути впевненим, що зможе ефективно розвивати свій проєкт у міру його зростання.

10. Навколо Django сформувалася активна спільнота. Тому той, хто зіткнувся з якоюсь проблемою, зможе без особливих складнощів її вирішити.

Проаналізувавши вищевикладене, можна дійти висновку, що у рамках цієї роботи оптимальним є вибір фреймворку Express для Node.js.

## 2.3. Вибір бази даних

Враховуючи вибрані рішення для реалізації клієнтської та серверної частин, за доцільне буде обрати останній компонент системи так, утворити MEAN стек. До складу MEAN стеку входить документоорієнтована система управління базами даних MongoDB, яка забезпечує високу продуктивність і легку масштабованість. MongoDB не вимагає опису схем таблиць, завдяки чому можна легко зберігати об'єкти і таким чином швидше пристосовуватися до змін у вимогах.

MongoDB використовує JavaScript як мову запитів, дозволяючи використовувати його для отримання даних, а використання map-reduce дає можливість створювати програми з великими даними. У сукупності з підтримкою індексів, профільуванням запитів та повнотекстовим пошуком з підтримкою морфології дозволяє MongoDB швидко повернути необхідну інформацію відповідно до запиту. Висока стійкість до відмов та масштабованість бази даних MongoDB досягається за рахунок асинхронних реплікацій, набору реплік і розподілу даних на вузли.

## 2.4. Висновки до розділу 2

Обрані рішення реалізації призвели до вибору MEAN стеку для реалізації проєкту. Абревіатура MEAN є акронімом і складається з перших букв наступних слів:

(M) ongoDB – документована NoSQL база даних з відкритим вихідним кодом;

(E) xpressJS – гнучкий NodeJS вебфреймворк, що забезпечує широкий набір функцій для побудови одно- та багатосторінкових, гібридних веб додатків;

(A) ngularJS – JavaScript-фреймворком з відкритим вихідним кодом, розробляється Google, і призначений для розробки односторінкових додатків;

(N) odeJS – програмна платформа, побудована на Chrome V8 Javascript движку, що дозволяє легко будувати швидкі, масштабовані мережеві програми.

MEAN є Full Stack JavaScript фреймворком і дозволяє будувати високопродуктивні та гнучкі вебдодатки. MEAN Stack використовує одну мову програмування на всіх рівнях, що значно спрощує розробку.

# РОЗДІЛ 3
# ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБРОБКИ ЕЛЕКТРОННИХ ЗВЕРНЕНЬ ТА ПЕТИЦІЙ

## 3.1. Проєктування архітектури системи

Як правило, комп'ютери і програми, що входять до складу інформаційної системи, не є рівноправними. Деякі з них мають ресурси (файлова система, процесор, принтер, база даних і т.д.), інші мають можливість звертатися до цих ресурсів. Комп'ютер (або програму), що управляє ресурсом, називають сервером цього ресурсу (файл-сервер, сервер бази даних, обчислювальний сервер...). Клієнт і сервер будь-якого ресурсу можуть бути як одному комп'ютері, і різних комп'ютерах, пов'язаних мережею. [9, с.99].

У рамках багаторівневого представлення обчислювальних систем можна виділити три групи функцій, орієнтованих рішення різних подзадач:

1. функції введення та відображення даних (забезпечують взаємодію з користувачем);
2. прикладні функції, притаманні даної предметної області;
3. функції управління ресурсами (файловою системою, базою даних тощо)

Виконання цих функцій переважно забезпечується програмними засобами, які можна у вигляді взаємозалежних компонентів, де:

- компонент подання відповідає за інтерфейс користувача;

- прикладний компонент реалізує алгоритм розв'язання конкретного завдання;
- компонент керування ресурсом забезпечує доступ до необхідних ресурсів.

Автономна система (комп'ютер, не підключений до мережі) представляє всі ці компоненти як різних рівнях (ОС, службове ПЗ та утиліти, прикладне ПЗ), і лише на рівні додатків (не характерно для сучасних програм). Так само і мережа - вона представляє всі ці компоненти, але, загалом, розподілені між вузлами. Завдання зводиться до забезпечення взаємодії між цими компонентами [9, с.101].

Архітектура клієнт-сервер визначає загальні принципи організації взаємодії в мережі, де є сервери , вузли-постачальники деяких специфічних функцій (сервісів) і клієнти , споживачі цих функцій.

Практичні реалізації такої архітектури називаються клієнт-серверними технологіями . Кожна технологія визначає власні або використовує наявні правила взаємодії між клієнтом та сервером, які називаються протоколом обміну (протоколом взаємодії) [9, с.101] .

У будь-якій мережі (навіть одноранговій), побудованій на сучасних мережевих технологіях, є елементи клієнт-серверної взаємодії, найчастіше на основі дволанкової архітектури . Дволанковою вона називається через необхідність розподілу трьох базових компонентів між двома вузлами.

Дволанкова архітектура використовується в клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо та в повному обсязі, використовуючи лише власні ресурси. Тобто. сервер не викликає сторонні мережеві додатки і не звертається до сторонніх ресурсів для виконання будь-якої частини запиту.

Розташування компонентів за клієнта чи сервера визначає такі основні моделі їх взаємодії у межах дволанкової архітектури:

- сервер терміналів - розподілене подання даних;
- файл-сервер - доступ до віддаленої бази даних та файлових ресурсів;
- сервер БД - віддалене подання даних;
- сервер програм — віддалений додаток.

Історично першою з'явилася модель розподіленого уявлення даних (модель сервер терміналів). Вона реалізовувалася на універсальній ЕОМ (мейнфреймі), що виступала у ролі сервера, з підключеними до неї алфавітно-цифровими терміналами. Користувачі виконували введення даних з клавіатури терміналу, які потім передавалися на мейнфрейм і там виконувалася їхня обробка, включаючи формування «картинки» з результатами. Ця «картинка» і поверталася користувачеві на екран терміналу.

З появою персональних комп'ютерів та локальних мереж, була реалізована модель файлового сервера, що представляв доступ до файлових ресурсів, у т.ч і до віддаленої бази даних. У цьому випадку виділений вузол мережі є файловим сервером, на якому розміщено файли бази даних. На клієнтах виконуються додатки, в яких поєднані компонент подання та прикладний компонент (СУБД та прикладна програма), які використовують підключену віддалену базу як локальний файл. Протоколи обміну у своїй представляють набір низькорівневих викликів операцій файлової системи [9,с.100].

Така модель показала свою неефективність з огляду на те, що при активній роботі з таблицями БД виникає велике навантаження на мережу. Частковим рішенням є підтримка тиражування (реплікації) таблиць та запитів. У цьому випадку, наприклад, при зміні даних, оновлюється не вся таблиця, а лише її модифікована частина.

З появою спеціалізованих СУБД з'явилася можливість реалізації іншої моделі доступу до віддаленої бази даних моделі сервера баз даних. І тут ядро СУБД функціонує на сервері, прикладна програма клієнта, а протокол обміну забезпечується з допомогою мови SQL. Такий підхід у порівнянні з файловим сервером веде до зменшення завантаження мережі та уніфікації інтерфейсу клієнт-сервер. Однак, мережний трафік залишається досить високим, крім того, як і раніше, неможливо задовільне адміністрування додатків, оскільки в одній програмі поєднуються різні функції.

З розробкою та використанням лише на рівні серверів баз даних механізму збережених процедур виникла концепція активного сервера БД . У цьому випадку частина функцій прикладного компонента реалізовані у вигляді процедур, що зберігаються, що виконуються на стороні сервера. Решта прикладної логіки виконується на клієнтській стороні. Протокол взаємодії – відповідний діалект мови SQL.

Переваги такого підходу очевидні:

- можливе централізоване адміністрування прикладних функцій;
- зниження вартості володіння системою з допомогою оренди сервера , а не його купівлі;
- значне зниження мережного трафіку (бо передаються не SQL-запити, а виклики процедур, що зберігаються).

Основний недолік - обмеженість засобів розробки процедур, що зберігаються в порівнянні з мовами високого рівня.

Реалізація прикладного компонента за сервера представляє таку модель — сервер додатків. Перенесення функцій прикладного компонента на сервер знижує вимоги до конфігурації клієнтів та спрощує адміністрування, але становить підвищені вимоги до продуктивності, безпеки та надійності сервера.

В даний час намічається тенденція повернення до того, з чого починалася клієнт-серверна архітектура — централізації обчислень на основі моделі термінал-сервера. У сучасній реінкарнації термінали відрізняються від своїх алфавітно-цифрових предків тим, що маючи мінімум програмних та апаратних засобів, представляють мультимедійні можливості (в т.ч. графічний інтерфейс користувача ). Роботу терміналів забезпечує високопродуктивний сервер, куди винесено все, до віртуальних драйверів пристроїв, включаючи драйвери відеопідсистеми.

Ще одна тенденція в клієнт-серверних технологіях пов'язана з дедалі більшим використанням розподілених обчислень. Вони реалізуються на основі моделі сервера додатків, де мережний додаток розділений на дві та більше частин, кожна з яких може виконуватися на окремому комп'ютері. Виділені частини програми взаємодіють один з одним, обмінюючись повідомленнями в заздалегідь узгодженому форматі. У цьому випадку дволанкова клієнт-серверна архітектура стає триланковою[9, с. 102].

Зазвичай, третьою ланкою в триланковій архітектурі стає сервер додатків, тобто. компоненти розподіляються наступним чином:

1. Подання даних - на стороні клієнта.
2. Прикладний компонент - на виділеному сервері додатків (як варіант, що виконує функції проміжного ПЗ ).
3. Управління ресурсами - на сервері БД, який і представляє дані, що запитуються.

Проаналізувавши можливі варіанти архітектурних підходів, вирішено застосувати звичайну дволанкову клієнт-серверну архітектуру.

## 3.2. Прототипування графічного інтерфейсу користувача

Графічний інтерфейс користувача — це інтерфейс, що дозволяє користувачеві взаємодіяти з електронним пристроєм через графічні (візуальні елементи). Тобто усі звичні для користувачів текстові посилання, кнопки, форми, навігація тощо — це елементи графічного інтерфейсу користувача. Для того, щоб правильно побудувати систему, необхідно прототипувати її, визначити, які саме елементи вона має містити. Так, для побудови автоматизованої системи обробки електронних петицій та звернень, вона має містити наступні елементи:

1. кнопку реєстрації;
2. форму реєстрації;
3. кнопку аутентифікації;
4. форму аутентифікації;
5. навігацію між пошуком, створенням, підписанням петицій\звернень;
6. форму створення електронної петиції;
7. форму створення електронного звернення;
8. секцію пошуку петицій;
9. секцію з результатами пошуку;
10. особистий кабінет користувача тощо;

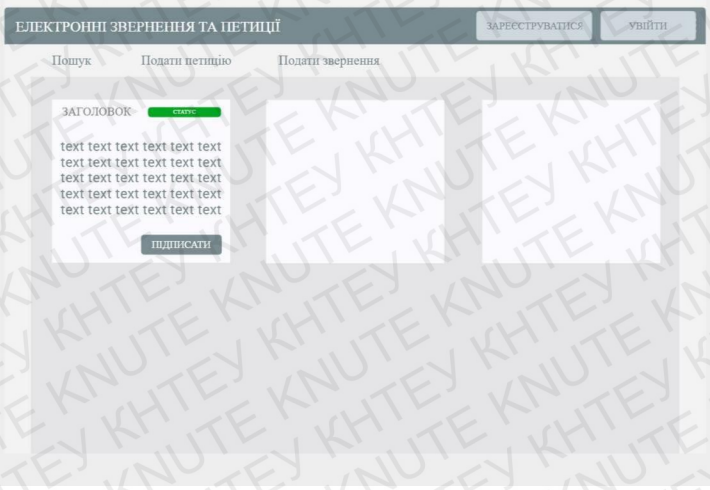


*Рис. 3.1. – Прототип головної сторінки*

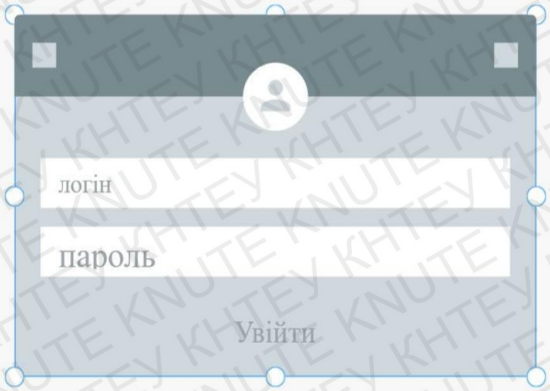


*Рис 3.2. Прототип форми аутентифікації*


*Рис. 3.3. Прототип форми створення звернення*

### 3.3. Документування програмного забезпечення

На початку роботи з автоматизованою системою користувач потрапляє на головний екран застосунку.

Для початку роботи з вебзастосунком, користувача має бути зареєстровано у системі. Для незареєстрованих користувачів доступ та

користування деякими частинами додатку є обмеженим. Для здійснення реєстрації потрібно заповнити наступні поля реєстраційної форми:

1. логін,
2. пароль
3. електронну адресу,
4. ім’я,
5. прізвище,
6. місто проживання,
7. номер паспорту,
8. номер телефону
9. дату народження

Якщо у користувача вже наявний акаунт в системі, він може здійснити вхід до системи. Тільки якщо користувач увійшов в систему він отримає доступ до повного функціонування системи без обмежень. Для цього необхідно ввести електронну пошту та пароль в поля форми входу.

У разі введеня помилкових даних, яких немає в системі, користувач отримає сповіщення про це.

Після аутентифікації у системі користувач може отримати доступ до створення нової  петиції. Необхідно перейти на сторінку створення петиції. Для створення петиції необхідно вписати у форму заголовок(тему), адресата петиції та викласти основний текст.

Якщо користувач заповнить форму не правильно, система повідомить його про це.

Користувач може  підписувати петиції інших користувачів, для цього необхідно обрати  будь-яку петицію та натиснути кнопку «Детальніше»

Після чого на екрані з’явиться вкладка з детальною інформацією про електронну петицію, ознайомившись з якою, користувач може її підтримати натиснувши кнопку «Підписати».

Також на сторінці петиції є можливість переглянути список користувачів, які підтримали дану петицію

Користувач може видалити створену петицію.

Якщо електронна петиція набирає необхідну кількість голосів, то вона надсилається до її адресата. У адресата з'являється дана петиція, натиснувши на неї, він потрапляє на сторінку цієї петиції. На ній є форма відповіді, після заповнення котрої адресат повинен натиснути кнопку "Відповісти" і відповідь буде розміщено у відповідній петиції.

## 3.4. Висновки до розділу 3

У цьому розділі було проаналізовано можливі варіанти архітектурних рішень, серед яких було обрано оптимальний. Було описано прототипування інтерфейсу та наведено окремі графічні прототипи. Також було здійснено документування програмного забезпечення.

# ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Метою цієї роботи було покращення практичної реалізації інструментів електронної демократії — створення проєкту автоматизованої системи обробки електронних петицій та звернень.

Для досягнення мети було:

1. досліджено електронні звернення та петиції як інструменти електронної демократії;
2. досліджено існуючі автоматизовані системи обробки електронних звернень та петицій;
3. обґрунтовано обрання фреймворку для клієнтської частини;
4. обґрунтовано обрання фреймворку для серверної частини;
5. обґрунтовано обрання бази даних;
6. обґрунтовано рішення проєктування системи;
7. розроблено прототип графічного інтерфейсу;
8. задокументувано програмне забезпечення;

Так, у рамках проекту було обрано і обгрунтовано застосування MEAN стеку, що включає в себе MongoDB, Express.js, Angular та Node.js, оскільки використання цього стеку не лише пришвидшує та полегшує розробку, але і має ряд переваг, що впливають на кінцевий продукт розробки: з допомогою стеку створюється додаток, оптимізований для розгортання у хмарних середовищах, спрощення розгортання завдяки вбудованому вебсерверу, полегшене керування великими обсягами даних з допомогою MongoDB.

Також ідея додати функціональну можливість подання електронних звернень до системи подання електронних петицій є новою для України, оскільки рішення зі схожим функціоналом відсутні, а звичайною практикою є подання електронних звернень за допомогою електронної пошти.

І хоча можна стверджувати, що мету роботи досягнуто, однак залишається перелік напрямків для майбутніх покращень:

1. створення зручної єдиної системи ідентифікації та аутентифікації користувачів;
2. створення механізму запобігання одночасного створення ідентичних або схожих за змістом петицій;
3. впровадження інтеграції з іншими вебзастосунками, соціальними мережами зокрема.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Стратегія розвитку інформаційного суспільства в Україні [Електронний ресурс] — Режим доступу: https://zakon.rada.gov.ua/laws/show/386-2013-%D1%80#n8
2. Закон України «Про звернення громадян» [Електронний ресурс] — Режим доступу: https://zakon.rada.gov.ua/laws/show/393/96-%D0%B2%D1%80#Text
3. “Top 50 Performers in E-participation in 2016.” United Nations E-Government Survey 2016 United Nations E-Government Survey, 2017, 197–217. doi: 10.18356/8e967554-en
4. Решота В. В. Електронна петиція як новий інструмент звернення громадян до органів публічної адміністрації / В. В. Решота // Науковий вісник Міжнародного гуманітарного університету. Серія : Юриспруденція. - 2015. - Вип. 15(1). - С. 91-94.
5. Константинівська А. К. Електронна петиція як ефективний інструмент політичної участі громадян / А. К. Константинівська // Epistemological studies in philosophy, social and political science. - 2018. - Vol. 1, Iss. 1-2. - С. 89-95.
6. Angular documentation [Electronic resource] — Access mode: https://angular.io/
7. NodeJs documentation [Electronic resource] — Access mode: https://nodejs.org/
8. Jay Sridhar. What Is JavaScript and How Does It Work? [Electronic resource] – Access mode: https://www.makeuseof.com/tag/what-is-javascript/

9. Проектування інформаційних систем: Загальні питання теорії проектування ІС (конспект лекцій) [Електронний ресурс] — Режим доступу: https://ela.kpi.ua/bitstream/123456789/33651/1/PIS_KL.pdf

# ДОДАТКИ

ДОДАТОК А

```
const express = require("express");
const app = (module.exports.app = express());
const hb = require("express-handlebars");
const db = require("./db");
const { hash, compare } = require("./bc");
const cookieSession = require("cookie-session");
const csurf = require("csurf");
const {
    requireLoggedOutUser,
    requireUnsignedPetition,
    requireSignedPetition,
    requireLoggedInUser,
} = require("./middleware");

const error = "Something went wrong, try again!";

let dataUrlsignature;
let validUrlUserHp;

app.engine("handlebars", hb());
app.set("view engine", "handlebars");

app.use(
    express.urlencoded({
        extended: false,
    })
);
```

```
app.use(
    cookieSession({
        secret: `pure being and pure nothing are the same.`,
        maxAge: 1000 * 60 * 60 * 24 * 14,
    })
);
app.use(csurf());

app.use(function (req, res, next) {
    res.set("x-frame-options", "DENY");
    res.locals.csrfToken = req.csrfToken();
    console.log("-------");
    console.log(`${req.method} request coming in on route ${req.url}`);
    next();
});

app.use(express.static("./public"));

app.get("/", (req, res) => {
    res.redirect("/register");
});

app.get("/register", requireLoggedOutUser, (req, res) => {
    res.render("register", {
        layout: "main",
    });
});

app.post("/register", requireLoggedOutUser, (req, res) => {
```

```
  const { firstName, lastName, email, password } = req.body;
  if ((firstName, lastName, email, password)) {
    hash(password)
      .then((hash) => {
        db.insertDetails(firstName, lastName, email, hash)
          .then((result) => {
            req.session.userId = result.rows[0].id;
            res.redirect("/profile");
          })
          .catch(() => {
            res.render("register", {
              error,
            });
          });
      })
      .catch((err) => {
        console.log("there is an error in hash", err);
      });
  } else {
    res.render("register", {
      error: "Please fill out all fields correctly!",
    });
  }
});

app.get("/login", requireLoggedOutUser, (req, res) => {
  res.render("login");
});

app.post("/login", requireLoggedOutUser, (req, res) => {
```

```
const { email, password } = req.body;
if ((email, password)) {
  db.getHashAndIdByEmail(email)
    .then((hash) => {
      compare(password, hash.rows[0].password)
        .then((result) => {
          if (result) {
            req.session.userId = hash.rows[0].id;
            db.checkIfSignatureByUserId(req.session.userId)
              .then((r) => {
                if (r.rows.length) {
                  req.session.sigId = r.rows[0].id;
                  res.redirect("/thanks");
                } else {
                  res.redirect("/petition");
                }
              })
              .catch((err) => {
                console.log(
                  "error in checkIfSignatureByUserId",
                  err
                );
                res.redirect("/petition");
              });
          } else {
            res.render("login", {
              error,
            });
          }
        })
```

```
            .catch((err) => {
              console.log("error in compare", err);
            });
        })
        .catch((err) => {
          console.log("error in getHashByEmail", err);
          res.render("login", {
            error,
          });
        });
    } else {
      res.render("register", {
        error: "Please fill out all fields correctly!",
      });
    }
});

app.get(
  "/petition",
  requireLoggedInUser,
  requireUnsignedPetition,
  (req, res) => {
    res.render("petition");
  }
);

app.post(
  "/petition",
  requireLoggedInUser,
  requireUnsignedPetition,
```

```
    (req, res) => {
        const { signature } = req.body;
        if (signature) {
            db.insertSignatureAndUserId(signature, req.session.userId)
                .then((result) => {
                    req.session.sigId = result.rows[0].id;
                    res.redirect("/thanks");
                })
                .catch((err) => {
                    console.log("error in SignatureAndUserId", err);
                    res.render("petition");
                });
        } else {
            res.render("petition", {
                layout: "main",
                noSignature:
                    "You still want to think about it? no problem take you time",
            });
        }
    }
);

app.get("/thanks", requireLoggedInUser, requireSignedPetition, (req, res) =>
{
    db.getSignature(req.session.sigId).then((result) => {
        dataUrlsignature = result.rows[0].signature;
        db.getTotalOfSigners().then(({ rows }) => {
            res.render("thanks", {
                layout: "main",
                rows,
```

```
                dataUrl: dataUrlsignature,
            });
        });
    });

    app.post("/thanks", requireLoggedInUser, requireSignedPetition, (req, res)
=> {
        db.deleteSignature(req.session.sigId)
        .then(() => {
            req.session.sigId = false;
            res.redirect("/petition");
        })
        .catch((err) => {
            console.log("error deleteSignature", err);
        });
    });

    app.get("/signers", requireLoggedInUser, requireSignedPetition, (req, res)
=> {
        db.getDataForSigners()
        .then(({ rows }) => {
            res.render("signers", {
                layout: "main",
                rows,
            });
        })
        .catch((err) => {
            console.log("error in db.getNames", err);
        });
```

```
});

app.get("/profile", requireLoggedInUser, requireLoggedInUser, (req, res) =>
{
  res.render("profile");
});

app.post("/profile", requireLoggedInUser, (req, res) => {
  let { age, city, url } = req.body;
  if (age == "") {
    age = null;
  }
  if (url.startsWith("https://") || url.startsWith("http://")) {
    validUrlUserHp = url;
    db.insertDataUserProfile(age, city, validUrlUserHp, req.session.userId)
      .then(() => {
        res.redirect("/petition");
      })
      .catch((err) => {
        console.log("error from insertDataUserProfile", err);
        res.render("profile", {
          layout: "main",
          error,
        });
      });
  } else {
    validUrlUserHp = "";
    db.insertDataUserProfile(age, city, validUrlUserHp, req.session.userId)
      .then(() => {
        res.redirect("/petition");
```

```
        })
        .catch((err) => {
            console.log("error from insertDataUserProfile", err);
            res.render("profile", {
                layout: "main",
                error: "Something went wrong, try again!",
            });
        });
    }
});

app.get("/profile/edit", requireLoggedInUser, (req, res) => {
    db.getProfileData(req.session.userId)
        .then(({ rows }) => {
            res.render("edit", {
                rows,
            });
        })
        .catch((err) => {
            console.log("error from getProfileData", err);
        });
});

app.post("/profile/edit", requireLoggedInUser, (req, res) => {
    let { firstName, lastName, email, password, age, city, url } = req.body;
    if (age == "") {
        age = null;
    }
    if (password) {
        hash(password).then((hash) => {
```

```
        db.updateUsersWithPassword(
          firstName,
          lastName,
          email,
          hash,
          req.session.userId
        )
          .then(() => {})
          .catch((err) => {
            console.log("error in updateUsersWithPassword", err);
            res.render("editError", {
              error,
            });
          });
      });

      db.updateUserProfiles(req.session.userId, age, city, url)
        .then(() => {
          res.redirect("/thanks");
        })
        .catch((err) => {
          console.log("error in updateUserProfiles", err);
          res.render("editError", {
            error,
          });
        });
    } else {
      db.updateUsersWithoutPassword(
        firstName,
        lastName,
```

```
        email,
        req.session.userId
      )
        .then(() => {})
        .catch((err) => {
          console.log("err pdateUsersWithoutPassword", err);
          res.render("editError", {
            error,
          });
        });
      db.updateUserProfiles(req.session.userId, age, city, url)
        .then(() => {
          res.redirect("/thanks");
        })
        .catch((err) => {
          console.log("error in updateUserProfiles", err);
          res.render("editError", {
            error,
          });
        });
    }
});

app.get(
  "/signers/:city",
  requireLoggedInUser,
  requireSignedPetition,
  (req, res) => {
    if (req.session.userId) {
      if (req.session.sigId) {
```

```
            const { city } = req.params;
            db.getSignersByCity(city)
              .then(({ rows }) => {
                res.render("signersByCity", {
                  city: city,
                  rows,
                });
              })
              .catch((err) => {
                console.log("error getSignersByCity", err);
              });
          } else {
            res.redirect("/petition");
          }
        } else {
          res.redirect("/register");
        }
      }
);
app.get("/logout", requireLoggedInUser, (req, res) => {
  req.session.userId = false;
  req.session.sigId = false;
  res.redirect("/register");
});

if (require.main == module) {
  app.listen(process.env.PORT || 8080, () =>
    console.log("Petitionserver listening")
  );
}
```