УДК 004.738.1

С.М. ЗАХАРЧЕНКО, Т.І. ТРОЯНОВСЬКА, О.В. БОЙКО, В.С. РИБАЧЕНКО
Вінницький національний технічний університет

ЗАСТОСУВАННЯ ОДНОСТОРІНКОВИХ ВЕБ-ОРІЄНТОВАНИХ ІНТЕРФЕЙСІВ В СОЦІАЛЬНО ЗНАЧУЩИХ ПРОЕКТАХ

Робота присвячена аналізу та обґрунтуванню застосування односторінкових веб-орієнтованих інтерфейсів для соціально значущих проектів та служб, завданням яких є покращити рівень обслуговування громадян та пришвидшити реагування на суспільні проблеми. Пропонується застосування односторінкових веб-додатків для побудови взаємодії між громадянами і муніципальними службами. Застосовано дельфійський метод до побудови ефективного інтерфейсу подібних додатків та його реалізація.

Ключові слова: соціальні служби, веб-портали, веб-інтерфейс, дельфійський метод, односторінковий інтерфейс, веб-орієнтований моніторинг.

S.M. ZAKHARCHENKO, T.I. TROIANOVSKA, O.V. BOIKO, V.S. RYBACHENKO
Vinnytsia National Technical University

APPLICATION OF SINGLE-PAGE WEB-BASED INTERFACES IN SOCIAL PROJECTS

This article discussed social significant projects and services, provided to increase level of public maintenance and to speed up reacting to social problems, and demonstrates building of communication link between citizens and such services with web-oriented applications. Also, a living example of social web service is provided and an approach to its interface efficient construction.

Key words: social services, web-portals, web-based interface, the Delphi method, a single-page interface web-based monitoring.

Вступ

Бурхливий розвиток мобільних технологій в останнє десятиріччя справив значний вплив на структуру і природу мережі Інтернет. Почали виникати сервіси, спрямовані не на науково-технічну діяльність, а на соціальну. Виникли сайти і служби, які покликані забезпечувати комунікацію між людьми, організацію їх в спільноти, а також координацію їх спільних дій.

Наступним етапом в цьому процесі було виникнення поняття «електронної демократії», яка передбачала залучення широких верств громадян до управління державою як на макрорівні (участь в розробці та обговоренні законопроектів, тощо), так і на рівні вирішення місцевих проблем (координація територіальних громад і суспільних організацій для їх вирішення – наприклад, незаконних забудов, або ремонту інфраструктури).

Актуальність та мета статті

На сьогодні в державі накопичилась велика кількість соціальних проблем, перш за все інфраструктурних, таких як стан будинків чи доріг, як розглядається в даному прикладі. Проблеми великі, і мають загальнодержавний характер, причому дуже складно оцінити, які об'єкти потребують нагальної уваги, а які – ні. Вирішенням цих проблем, як правило, повинна займатись місцева влада, але на практиці цим займаються численні волонтерські організації, або ж окремі громадські активісти. Для налагодження ефективної роботи, а також встановлення контактів із владними структурами їм необхідний зручний і доступний інструмент комунікації, який містив би необхідну для роботи інформацію, надавав майданчик для обговорення та оцінки актуальності конкретно взятих випадків.

Метою статті є аналіз існуючої системи веб-орієнтованої комунікації між громадськими активістами та її вдосконалення для вирішення спеціальних задач засобами односторінкових веб-орієнтованих інтерфейсів.

Постановка задачі

1. Проаналізувати існуючі архітектурні рішення для веб-орієнтованих спеціалізованих порталів;
2. Обґрунтувати вибір односторінкової архітектури для створення порталу веб-комунікації між громадськими активістами;
3. Оцінка розгортання середовища та застосування фреймворків;
4. Розробити практичну реалізацію порталу веб-комунікації на основі односторінкової архітектури.

1. Аналіз архітектурних рішень для веб-орієнтованих спеціалізованих порталів

Одним із напрямків розвитку низового рівня електронної демократії є запропоноване Мауріціо Болоніньї поєднання соціальних груп за інтересами, із професійними експертними панелями та державними службами [1]. Його пропозиція представляла собою модифікацію «дельфійського метода» [2], розробленого в 1960-і роки для корпорації RAND, і призначеного для створення якомога точніших експертних оцінок.

Базовим принципом цього методу є уявлення, що велика кількість незалежних експертів, кожен із яких мінімально пов'язаний із іншими (краще, якщо взагалі не пов'язаний), краще оцінює ситуацію, ніж структурований колектив дослідників. Подібний підхід дозволяє виключити із роботи більшість суб'єктивних факторів командної роботи, як-то, політичні погляди, особисте несприйняття між різними працівниками, мовний бар'єр, тощо, а отже, утворена таким чином експертна панель буде репрезентувати максимально можливу кількість точок зору і дасть реальну картину ситуації, як і максимальну кількість варіантів рішень.

Організаційно «дельфійський метод» передбачав дворівневу структуру:

- **Група дослідників-експертів**. Кожен із них досліджує проблему самостійно, і надає свій аналіз і оцінку окремо, в письмовій формі;
- **Організаційна група**. Вона переглядає всі аналітичні записки, і зводить їх в єдиний звіт.

Діяльність «дельфійських груп» відбувається по ітеративному принципу. Спочатку збирається звіт по базовим питанням, з якого виокремлюються найменш висвітлені теми, які потім також аналізуються окремими експертами – і в результаті звіт коригується. Процес відбувається ітеративно, до досягнення узгодженості між усіма експертами, або буде підтверджено, що консенсусу досягти неможливо.

Подібна схема роботи дозволяє досягти максимального охоплення визначеної теми і продукує більш зважені висновки, ніж спеціально підібрані експертні групи.

Однак, через складну модель комунікації ця модель так і не набула широкого розповсюдження. Другого дихання вона набула тільки після появи електронних засобів зв'язку у реальному часі, таких як електронна пошта, що дозволяло інтенсифікувати обмін інформацією і перевести діяльність «дельфійських груп» на якісно новий рівень.

Яскравими прикладами організаційних структур на базі дельфійського методу є архітектури, запропоновані Біллом Гейтсом [3], Томом де Марко [4], а також Фредеріком Бруксом [5]. В свою чергу, вони реалізуються різними способами для різних задач – від корпоративного до соціального рівня.

В даній статті ефективне застосування дельфійського методу показано на прикладі соціального проекту, покликаного створити систему експертного моніторингу стану доріг, а також удосконалено організаційний процес – шляхом переходу від веб-інтерфейсів, орієнтованих на дискусію (тематичні групи в соціальних мережах, форуми тощо), які тяжіють до абстрактних обговорень проблематики взагалі, до проблемоцентричної архітектури, де процес моніторингу концентрується навколо конкретно існуючих випадків і таким чином має більшу практичність та ефективність.

2. Обґрунтування вибору односторінкової архітектури для створення порталу веб-комунікації між громадськими активістами

Основою комунікації, як було вказано вище, в дельфійському методі є механізм, який дозволяє експертам спілкуватись в реальному часі. Одним із таких механізмів є сайт, побудований по принципу «однієї сторінки», тобто, всі необхідні дані для введення, категоризації та аналізу даних сконцентровані на одній веб-сторінці, яка динамічно змінюється залежно від конкретного виду роботи. Такий інтерфейс також дозволяє перейти від моделі обговорення проблематики, коли комунікація будується навколо самого процесу, до проблемно-орієнтованої моделі, коли обговорення будується навколо конкретно взятого практичної проблеми, яка одночасно є і стартовою точкою обговорення, і предметом оцінювання актуальності. Таким чином, оцінка проблеми буде більш реалістичною, адже виставляється не лише на основі загального опису, а й на основі розгорнутого обговорення.

Односторінкові (single-page) інтерфейси будуються, як правило, за допомогою спеціальних клієнт-серверних оболонок (фреймворків), які працюють за принципом, показаним на рис. 1.

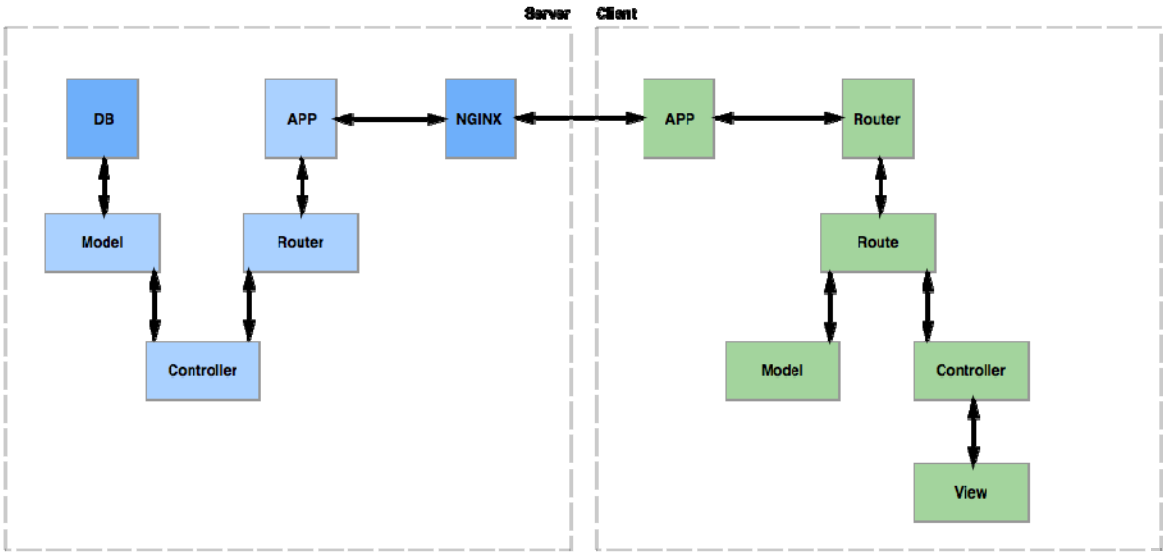

Рис. 1. Діаграма функціонування односторінкового інтерфейсу

На сьогодні існують наступні оболонки, які дозволяють будувати такі інтерфейси (табл. 1).

Виходячи із наведеної таблиці, для реалізації односторінкового інтерфейсу найбільше придатна EmberJS. Однак ця оболонка забезпечує користувацький інтерфейс, а для роботи безпосередньо із даними і перетворення «сирих» запитів на веб-запити необхідна серверна частина, яка слугуватиме передаточною ланкою між базою даних та веб-інтерфейсом.

Для цього на сервері необхідно розгорнути оболонку із неблокуючим (для забезпечення одночасного доступу багатьох користувачів), орієнтованим на події (для мінімізації мережевого трафіку) та

асинхронною моделлю запитів (для забезпечення роботи в реальному часі). На сьогодні існує низка таких оболонок (табл. 2).

Таблиця 1

Існуючі оболонки для односторінкового інтерфейсу

Назва	Архітектури	Короткий опис
AngularJS	Повністю клієнтська	Розрахована на потужну робочу станцію, всі шаблони компілюються безпосередньо на ній, дані зберігаються на сервері, але повністю перевантажуються клієнту
EmberJS	MVC, із розділенням на серверну і клієнтську	Розрахована на широкий діапазон пристроїв, дані зберігаються на сервері, шаблони компілюються на сервері, на клієнтській робочій станції тільки візуалізація
MeteorJS	Повністю серверна	Розрахована на розподілені системи, із потужним кластером і термінальними клієнтами, на робочій станції тільки відображення, вимагає широкого каналу для передачі даних

Таблиця 2

Оболонки для клієнтських робочих станцій

Назва	Архітектура	Короткий опис
Sling	Rhino	Базується на Java і найчастіше застосовується як додатковий пакет до розгорнутого JSP чи сервлетного движка.
Jaxer	SpiderMonkey	Базується на технології Apache, однак на сьогодні вже не оновлюється.
IO.js	V8	Найновіша платформа, створена на базі Node.js, яка швидко розвивається, але не є повноцінною.
Node.js	V8	Базується на движку V8 від GoogleChrome, на сьогодні є найбільш комплексним серверним движком для JavaScript.

Очевидно, що для обраної задачі найкраще підходить остання оболонка –Node.js. Відповідно, база даних також повинна підтримувати стандарт V8, тому обрано MongoDB. Додатковою перевагою цього вибору є її кросплатформеність, що дозволяє розгорнути систему на Linux, а отже, значно знизити витрати на розробку та розгортання проекту.

Таким чином, обрана для вирішення поставленої задачі архітектура виглядає наступним чином:

- Операційна система: LinuxUbuntu;
- Основна база даних: MongoDB;
- Серверний движок: Node.js;
- Клієнтський движок: EmberJS;
- Стильове оформлення: CSS (за допомогою препроцесора Less);
- Верстка сторінок: HTML (за допомогою шаблонів Handlebars з пакету EmberJS).

3. Розгортання середовища та застосування фреймворків

Оцінимо трудомісткість розгортання середовища та необхідного інструментарію для створення односторінкового інтерфейсу.

Основною проблемою на цьому етапі є конфігурація nginx. Оскільки обраним операційним середовищем є Linux Ubuntu, він уже присутній в базовій установці, і його можна встановити за допомогою простої команди apt-getinstallnginx. Оскільки на клієнтській частині використовується фреймворк EmberJS, в якості менеджера задач (TaskRunner) використовується Ember-cli. Ember-cli має також свої налаштування, які знаходяться в файлі .ember-cli. При збірці клієнтської частини Ember-cli використовує Broccoli, налаштування якого містяться в скрипті Brocfile.js.

Серверна частина програмного додатку моніторингу якості доріг створена за допомогою JavaScript на базі платформи NodeJS. В якості фреймворку був вибраний restify – фреймворк для створення API додатків, який був побудований з найпопулярнішого фреймворку для NodeJS – ExpressJS. ExpressJS потужний фреймворк, але використовуючи сучасні рішення – серверна частина повинна бути створена виключно у вигляді API рішення. Тобто, вона повинна віддавати та приймати дані JSON, проводити маніпуляції з базою даних та відповідати клієнтській частині. Зазвичай, це роблять всі серверні програмні додатки, але крім цього, вони ще створюють шаблон, заповнюють його даними, кешують і віддають клієнтській частині у вигляді HTML. Це створює велике навантаження на сервер, тому в даному програмному продукті за шаблони буде відповідати клієнт, адже сервер один – а клієнтів багато, відповідно, необхідно, по можливості,передати всю допустиму роботу на клієнтську частину.

Запустити серверну частину можна за допомогою команди nodecore/server. Далі при запиті будуть послідовно виконуватись методи, які були додані в стек методів за допомогою server.use, а потім методи контролеру по заданому шляху.

Контролери перевіряють вхідні дані запиту на наявність всіх необхідних полів і викликають необхідні методи з моделі. Модель безпосередньо робить запити в базу даних. Для роботи з базою даних MongoDB вибраний модуль для NodeJS під назвою mongoose. Він потребує створення моделі по схемі

документу і має стандартні методи пошуку, додавання, видалення, оновлення документу в базі даних. Модуль responser відповідає за створення JSON об'єкту моделі з даними із бази даних та викликає шаблон відповіді певного типу (для відповіді на запит отримання груп – шаблон груп).

Файл із налаштуваннями серверного програмного додатку, який знаходиться в директорії config, є типом JSON, і містить в собі дані авторизації для підключення до бази даних, налаштування серверного програмного забезпечення – такі як порт, хост і так далі. Трудомісткість встановлення середовища, таким чином, не відрізняється принципово від встановлення будь-якого програмного засобу.

Після завершення встановлення, необхідний створити файл налаштувань. Nginx налаштовується на проксування запитів, і є першою ланкою зі сторони серверу, який приймає та відповідає на запити. Якщо запит правильно сформований, він передає його серверному програмному додатку, який власне і реалізуватиме веб-інтерфейс.

На наступному етапі інформація передається в маршрутизатор (роутер, router). Він перевіряє існування шляху по отриманому запиту (в загальній формі – URI – UniformResourceIdentifier), і якщо немає – генерує помилку з кодом 404 (не знайдено). Якщо шлях знайдено, викликає метод відповідного контролеру, який в свою чергу, додатково робить перевірку даних і викликає відповідний метод моделі (наприклад, на вибірку даних). Після виконання методу, дані із бази даних повертаються в контролер який, за умови, що помилок не виникло, повертає результат виконання назад в nginx, і далі передаються клієнту.

Зі сторони клієнтської частини включається EmberJS, який приймає дані, і вносить зміни на відкриту веб-сторінку в режимі реального часу. Це рамочний алгоритм функціонування.

Всередині серверної частини процес відбувається шляхом послідовного виконання методів, які відповідають етапам процесу обробки даних: beforeModel; model; afterModel; setupController.

Шлях, який вказаний в запиті до сервера, представляє собою ідентифікатор задачі, яку необхідно виконати. Саме в методі model шлях трансформується в задачу, і формується зв'язок цієї задачі з відповідною моделлю. Цей метод повинен повернути в якості результату відповідь запиту до моделі. В свою чергу, модель створює запит до серверу з відповідним типом запиту та його URL і створює власне задачу, яка і представляє собою «конверт», в який буде поміщений результат. Якщо задача вирішена, але в процесі роботи виникла помилка, то задача буде відхилена і згенеровано статус Rejected, а якщо дані отримані,задача маркується як Resolved (вирішено).

Наступним етапом є метод setupController. Він у якості аргументу приймає контролер, який необхідний для роботи з моделлю, що була отримана на базі вибраних даних. Якщо методу setupController не було знайдено для даної задачі, викликається більш загальний контролер, вищий за ієрархією, задачею якого є підключення моделі в даний контролер.

Далі підключається шаблон візуалізації даних, який заповнюється даними з контролеру.

Графічно цей процес показано на рисунку 2.

Таким чином, при виконанні дій користувача, наприклад, натискання кнопки «Відправити на перевірку», створюється повідомлення. Повідомлення обробляється за наступною схемою:

- Знайти, чи зареєстрований вид дії (action) в контролері з назвою повідомлення;
- Якщо такого немає, знайти відповідний шлях (URI);
- Якщо такого немає, знайти його в батьківському URI. Наприклад, при url в /items/create – для ідентифікатора /create батьківським буде ідентифікатор /items;
- Якщо такого немає, батьківському URI вищого рівня, до кореневого каталогу;
- Якщо не знайдено – створити повідомлення про помилку.

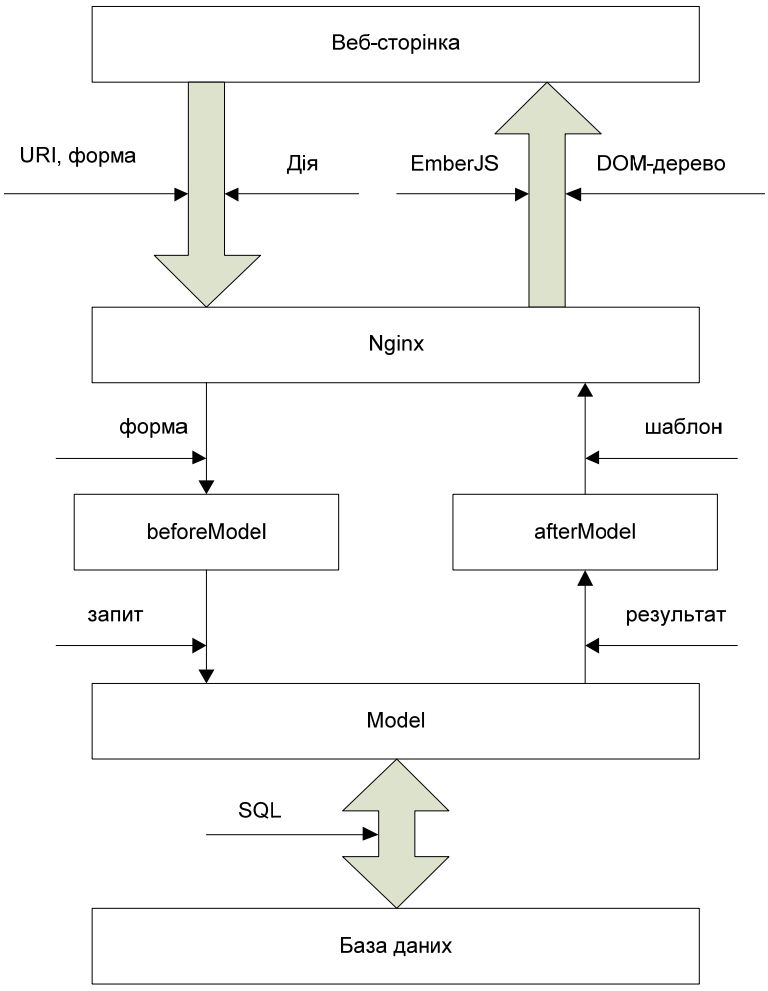

Рис. 2. Архітектура одно сторінкового інтерфейсу

Розглянемо цю схему на прикладі завантаження користувачем головної сторінки. При переході на веб-сайт продукту, перш за все, проходить завантаження програми, ініціалізаторів, маршрутизатора, а саме – URI routes/application.js з його контролером controllers/application.js та шаблоном templates/application.hbs і стилем styles/app.less. Перехоплення натискання клавіш клавіатури відбувається в процесі формування URI за допомогою робочих скорочень (shortcuts), які дозволяють швидко персоналізувати систему.

Авторизацію користувача проводить модуль ember-simple-auth. Він діє за допомогою домішок, тобто до тих URI які мають відображатись тільки для авторизованих користувачів додаються домішки (службові дані) модуля simple-auth. Також домішки додаються до URI Application для отримання методу logout (деавторизації), а також до URI, який відповідає за авторизацію користувача.

Відповідно, якщо користувач потрапляє на головну сторінку веб-сайту, спрацьовує URI (шлях) під назвою ApplicationIndex. В свою чергу, це простий шлях який перенаправляє користувача на URI group.atpage з параметром group_slug (назва групи) all – зарезервована назва групи, яка відображає всі дефекти доріг без фільтрування по групі. Перенаправлення здійснюється за допомогою методу transitionTo, який викликається в методі beforeModel.

Тепер, відповідно, відбувається перехід до URI GroupAtpage, який в свою чергу в методі model отримує дані про дефекти за фільтром який вказаний в url. Якщо користувач знаходиться на URL виду/all, то завантажуються всі дефекти без фільтру, а при url виду /group – завантажуються ті дефекти, які знаходяться в групі group. Крім того, після назви групи в URLl може бути також цифра, яка означає номер сторінки. Дефекти завантажуються з API серверної частини посторінково, по 10 дефектів на одну сторінку.

Таким чином, в рамках запропонованої нами моделі веб-сервіс представляє собою набір URI, кожен із яких повинен мати відповідну дію (задачу), яка, на основі переданих їй користувацьких даних, формує запит до моделі на сервері, і відображає повідомлення про успішне додавання нових даних (або про помилку) і оновлює сторінку, відкриту користувачем. Оновлення сторінки чи перехід на інші не вимагається, що значно зменшує витрати часу, а також збільшує ергономічність інтерфейсу за рахунок виключення зайвих дій по переходу на інші сторінки.

4. Реалізація порталу веб-комунікації на основі односторінкової архітектури

Перед реалізацією поставленої задачі, було проаналізовані існуючі проекти, та виявлено їх недоліки(табл. 3).

Таблиця 3

Існуючі проекти моніторингу стану доріг

Назва	Перелік недоліків
УкрЯма	- Відсутня функція пошуку - Інтерактивна карта містить інтерфейсні помилки
Контроль ремонту доріг	- Відсутнє фіксування пошкодженості дороги - Інтерактивна карта не фокусується
РосЯма	- Жорстка система аутентифікації - Слабка система захисту даних

Відповідно, було сформульовано критерії та функціональні можливості, яким повинна задовольняти розроблена система моніторингу якості доріг:

Додавання нової дорожньої проблеми, що включає в себе: а) фото дорожньої проблеми; б) місце дорожньої проблеми із координатам довготи і широти; в) група, до якої відноситься дорожня проблема. Це може бути як проблема, наприклад, автомобільної дороги або пішохідної.

- Короткий опис дорожньої проблеми;
- Відображення списку останніх доданих дорожніх проблем, кожна з якої має свій статус виконання – щойно додана, перевірена, опрацьовується, виконана або невиконана. Крім того, інформація повинна розбиватись на сторінки задля збереження трафіку користувача та зменшення навантаження на сервер;
- Фільтр списку дорожніх проблем по групам;
- Можливість роботи з програмним забезпеченням моніторингу якості доріг з мобільних пристроїв (смартфонів та планшетів) в повному обсязі функціоналу – переглядати, фільтрувати та додавати нові дорожні проблеми.

Відповідно будувалась реалізація різних підсистем сайту. Коротко опишемо їх і продемонструємо їх роботу.

Клієнтська частина програмного забезпечення моніторингу якості доріг складається з користувацької та адміністративної частин. Вони однакові, їх різниця складає в тому, що при доступі до адміністративної частини у користувача попросять дані авторизації (логін та пароль). Не надавши правильних даних авторизації користувачу буде заборонено доступ до адміністративних функцій.

Головна сторінка клієнтської частини програмного додатку зображена на рисунку 3. При відвідуванні веб-сайту з планшетних комп'ютерів та смартфонів, бокове меню зникає. Користувач може викликати його за допомогою спеціальної кнопки, розташованої в заголовку клієнтського додатку.

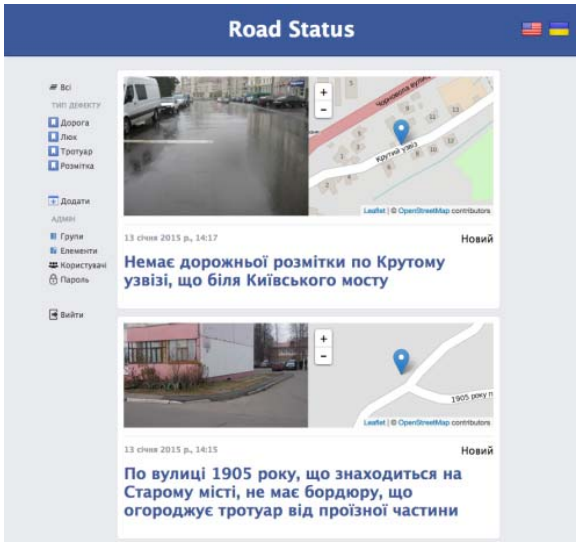

Рис. 3. Клієнтська частина програмного додатку

Дизайн клієнтської частини розроблено таким чином, щоб він адаптувався до пристрою користувача.

Додаток можна поділити на частини:

1) Заголовок клієнтського додатку.

2) Бокове меню клієнтського додатку.

3) Список дефектів, кожен з яких складається з типу дефекту, розташування, дати створення та короткого опису.

Вікно додавання дефекту зображене на рис. 4.

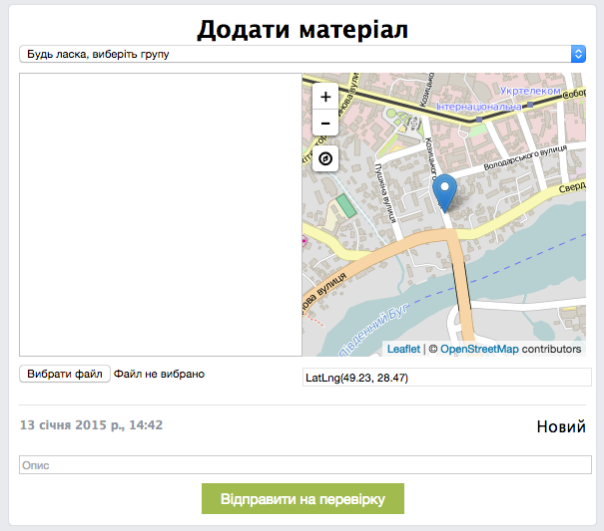

Рис. 4. Вікно додавання дефекту

Адміністративна частина створена в рамках клієнтської частини програмного додатку моніторингу якості доріг, і для доступу в неї потрібно пройти авторизацію (рис. 5).

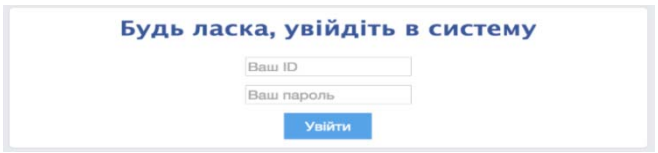

Рис. 5. Авторизація на сайті

Після успішної проходження авторизації в меню зліва з'являються додаткові пункти адміністрування програмного продукту – групи, елементи (дефекти), користувачі та зміна паролю (рис. 6).

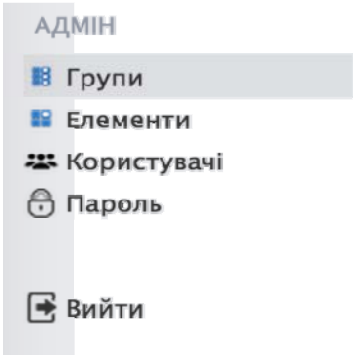

Рис. 6. Додаткові пункти адміністрування програмного продукту

Принцип роботи кожного пункту такий же як і в користувальницькій частині – є шляхи, контролери, компоненти, шаблони. Різниця лише в тому, що навігація до кожного з пунктів адміністративної частини потребує коректної авторизації адміністратору.

В розділі «Групи» редагуються типи дефектів та можуть бути додані нові типи дефектів.

В розділі «Елементи» знаходяться всі додані користувачами дефекти. Для зручності, спочатку потрібно вибрати групу дефектів, після чого будуть виведені всі додані до цієї групи дефекти, які можна редагувати.

Меню «Користувачі» дозволяє отримати список всіх користувачів, які додавали дефекти. Користувача можна підтвердити, заблокувати або видалити разом з усіма дефектами, що він додав.

Розділ «Пароль» адміністратор програмного продукту дозволяє змінити пароль для доступу до адміністративної частини.

В цілому, архітектура одно-сторінкового інтерфейсу забезпечує базове вирішення поставленої задачі, що дасть можливість побудувати повноцінну реалізацію дельфійського методу для контролю якості доріг.

Висновки

У даній роботі виконаний аналіз та проведено обґрунтування до застосування односторінкових веб-орієнтованих інтерфейсів для соціально значущих проектів та служб з метою оперативного реагування на соціальні проблеми громад. Створено прототип системи комунікації такої групи в реальному часі на базі профільного веб-сайту, організованого по принципу «односторінкового інтерфейсу» із застосуванням технологічних рішень на базі серверного та клієнтського JavaScriptcEngine.

В рамках проведеної роботи було проведено аналіз існуючих архітектурних рішень для веб-орієнтованих спеціалізованих порталів та обрано за основу дельфійський метод. В процесі розробки було удосконалено звичайну архітектуру комунікації і зведено її до односторінкового інтерфейсу. Це дозволяє значно спростити роботу із веб-сервісом, а отже, дозволяє залучення більш широкої експертної групи. Подібне спрощення також дозволяє мобілізувати веб-сервіси, адже за рахунок спрощеної односторінкової архітектури вона легко може бути перенесена на мобільні пристрої – модифікації підлягає тільки основна сторінка та шаблони.

Подібна система, як було виявлено, достатньо проста і дешева в реалізації, а відтак може бути застосована і для більш масштабних проектів.

Література

1. The Millennium Project. Futures Research Methodology : Amer Council for the United Nations / Jerome C. Glenn, Theodore J. Gordon. – New York, 2009. – 700 p.

2. Ковалев В.В. Анализ хозяйственной деятельности предприятия / В.В. Ковалев, О.Н. Волкова. – М. : ТК Велби, Изд-во Проспект, 2007. – 424 с.

3. Гейтс Б. Бизнес со скоростью мысли / Гейтс Б. – М. : Эксмо-Пресс, 2003. – 262 с.

4. DeMarco T. Peopleware: Productive Projects and Teams (3rd Edition) / Tom DeMarco, Timothy Lister. – NY : Addison-Wesley Professional, 2013. – 248 p.

5. Brooks F. Mythical Man-Month (2 edition) / Brooks F. – NY : Addison-Wesley Professional, 1995. – 322 p.

Рецензія/Peer review : 21.5.2016 р. Надрукована/Printed : 6.6.2016 р.
Рецензент: д.т.н., Лужецький В.А.