DOI 10.15276/imms.v14.no1-2.37
УДК 519.6:004.93

Informatics and Mathematical Methods in Simulation
Vol.14 (2024), No. 1-2, pp. 37-44

РОЗРОБКА ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ВЕБ-ДОДАТКІВ

Ю.Ю. Козіна[1], Б.І. Юхименко[2], О.В. Іщенко[3]

Національний університет «Одеська політехніка»,
1, Шевченка пр., Одеса, 65044, Україна
emails: yuliyakc21@gmail.com[1], biruteyu@gmail.com[2], alesya.ishchenko@gmail.com[3]

В умовах росту інформаційних технологій все більшу популярність і потрібність набувають додатки, що розробляються для використання їх на різних платформах – кросплатформенні додатки. Це обумовлено появою та розвитком всіляких пристроїв, на яких вони можуть функціонувати. Одним з напрямків, що швидко розвиваються в галузі багатоплатформеності в даний час, є розробка додатків, що працюють на різних операційних системах, таких як десктопні – Windows, Mac OS, мобільні – iOS, Android. Однак, існуючі методики їх розробки орієнтовані на створення додатків під конкретну операційну систему, що обмежує універсальність їх застосування. У роботі вирішено актуальне завдання розробки та тестування інформаційної системи кросплатформних веб-додатків, використовуючи зв'язку фреймворків. Так само, в роботі проведено аналіз новітніх методик та інструментів кросплатформної розробки та їх засобів автоматизованого тестування. У підсумку розроблено багатоплатформний програмний продукт, який реалізує архітектуру «клієнт-сервер» та працює на платформах ios, android, і так само в будь-якому web браузері. Розроблений програмний продукт дозволяє користувачам зберігати та отримувати доступ до своїх нотатків на різноманітних платформах та редагувати їх.
Ключові слова*:* тестування, кросплатформні додатки, клієнт-серверна архітектура.

Вступ. Існуючі методики розробки кросплатформних веб-додатків орієнтовані на створення програмного забезпечення під конкретну операційну систему, що обмежує універсальність їх застосування. Наприклад, додатки, розроблені під систему Windows не завжди будуть працювати на мобільних платформах. А з урахуванням зростаючого попиту на мобільні додатки, актуальність вирішення даного протиріччя зростає. Крім того, досягнення високої якості розробки кросплатформних додатків може бути забезпечено ефективною організацією процесу їх автоматизованого тестування. Задача дослідження підходів до розробки кросплатформних веб-додатків з високим ступенем переносимості коду є актуальною на даний час. Крім того, треба шукати шляхи, які дозволять гнучке нарощувати функціонал для клієнт-серверних задач.

Мета роботи. Вимоги практики розробки кросплатформних додатків, що функціонують на десктопних і мобільних платформах, а також організація процесу їх автоматизованого тестування обґрунтовують актуальність даної роботи. Метою роботи є розробка та тестування кросплатформної інформаційної системи (ІС), архітектура якої дозволить гнучке нарощувати функціонал для клієнт-серверних задач.

Основна частина. Основними критеріями до обґрунтування вибору підходу до розробки, інструментів та методології програмування при реалізації кросплатформних додатків обрано: високий відсоток переносимості коду; продуктивність; безкоштовність продуктів; швидкість розробки; якість кінцевого продукту.

Виберемо підхід, що використовує фреймворки та зробимо вибір фреймворків кросплатформної розробки, який ґрунтується на відповідності сильних та слабких сторін фреймворків наведеним критеріям. Обрано Titanium SDK[1]. Він є безкоштовний. Має високу переносимість коду, поступається тільки PhoneGap за цим параметром, проте виграє у нього в продуктивності. Швидкості розробки сприяє велика

кількість готових рішень «з коробки». Однак якість продукту може постраждати через те, що використовувана мова javascript асинхронна та в численних розгалуженнях програмісту легко помилитися. Додатково розглянемо ще такий додатковий побічний критерій, як популярність та її динаміка використання, для цього скористаємося сервісом «Google trends» – рис. 1. По порівняння було обрано фреймворки: Titanium, PhoneGap та Xamarin.

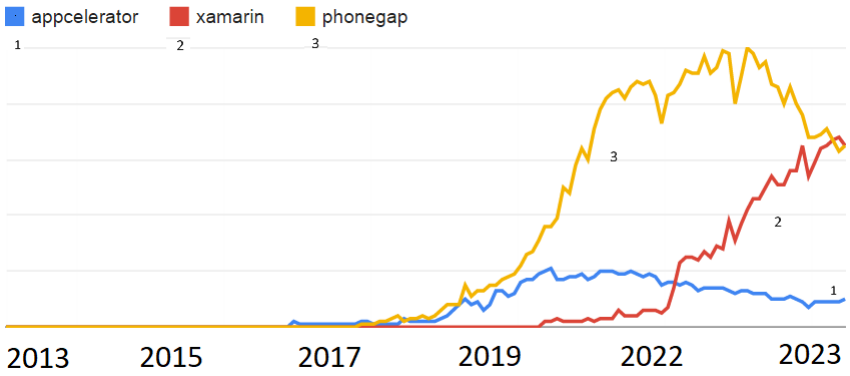


Рис. 1. Популярність та її динаміка використання фреймворків

Titanium SDK на графіку Google trends представляє «appcelerator» так як це ключове слово найбільш поширено для його пошуку. Як ми бачимо Titanium найменш популярний з розглянутих фреймворків, але його динаміка володіє деякою стабільністю. Це говорить нам про те, що ми стикаємося з проблемою слабкої підтримки фреймворку з боку співтовариства, але ми цілком можемо розраховувати, що цей фреймворк матиме підтримку з боку власника і не буде покинутий найближчим часом. Але так як нашим завданням є клієнт-серверне програмне забезпечення, то нам також потрібен інструментарій для створення серверної частини – веб-сервіс зі своїм API. Хоча Titanium SDK і дозволяє написання веб-додатків, але його функціонал призначений для самостійних додатків і написання веб-сервісу на ньому буде недоцільно. Тому виберемо потрібний інструментарій [2]. Результатом вибору буде – Node.js. Цей фреймворк повністю безкоштовний, виграє в продуктивності у аналогів за рахунок асинхронності. Швидкість розробки висока за рахунок декількох факторів:
– мова javascript. Одна мова для двох фреймворків дає незаперечний плюс в швидкості розробки, так як програмісту не потрібно буде переключатися з однієї мови на іншу;
– багата бібліотека готових рішень (модулів) npm і мінімум готових рішень у чистій збірці Node.js. Ці два фактори в комплексі дозволяють використовувати тільки ті модулі які потрібні і ніяких зайвих, це особливо корисно якщо враховувати, що нам потрібно створювати за допомогою цього інструменту не цілий веб-сайт, а тільки веб-сервіс і нам не потрібні всякі модулі та заготовки для веб-сайтів як у фреймворків конкурентів (не враховуючи Spring, він так само має високий рівень «модульності»).

При всіх перевагах, все ж таки залишається уразливість з якістю кінцевого продукту із за асинхронності як і у випадку з Titanium SDK. Для усунення вразливостей пов'язаних з помилками при написанні коду, будемо використовувати таку методологію програмування як BDD.

Розглянемо такі інструменти автоматизованого тестування, які одночасно працюють з Node.js і Titanium SDK:

1) Ti-mocha – це перенесена версія Mocha під Titanium SDK, іншого трохи менш популярного BDD фреймворка для javascript. На відміну від Jasmine, Mocha не має вбудованої бібліотеки тверджень і бібліотеки для роботи з заглушками, що швидше йде в плюс, тому що можна налаштувати його під себе. До нього можна

без проблем підключити такі бібліотеки як: should.js, assert.js, які дозволяють використовувати такі ключові слова як should (повинен) або як assert (очікується). А так само використовувати таку потужну бібліотеку для заглушок як Sinon. Ti mocha разом з бібліотекою тверджень should є головним претендентом для вибору як BDD фреймворку для Titanium.

2) Tishadow. Розробники цього продукту позиціонують його як повний набір інструментів для швидкої розробки додатків на Titanium SDK. Крім BDD фреймворку він включає в себе інструменти для швидкого розгортання додатків під всі платформи. BDD фреймворк заснований на реалізаціях Jasmine під Titanium і володіє всіма сильними і слабкими сторонами Jasmine. Але на відміну від Titanium-Jasmine цей проект постійно оновлюється.
3) Ticalabash – це перенесена версія фреймворку calabash для автоматизованого тестування для мобільних платформ. Calabash є ПЗ з відкритим вихідним кодом і широко використовується серед розробників. Основним плюсом calabash є те, що він використовує BDD фреймворк Cucumber, який дозволяє писати тести на мові максимально наближеної до природної мови. При всіх перевагах великою проблемою Ticalabash є його помилки і не стабільність.

Виходячи з аналізу розглянутих BDD фреймворків вибираємо Ti-mocha в зв'язці з should.js, і використовуємо таку ж зв'язку під Node.js.

Зауважимо що, так як ми використовуємо методологію розробки BDD, то специфікація додатка описується в тому ж стилі, що і тести, якби дублюючі їх. Так само слід розділяти поняття користувач і користувач програми. Під користувачем мається на увазі об'єкт в додатку, а користувач програми це людина яка використовує наше програмне забезпечення.

Перелічимо функціональні вимоги до системи:

1) Система повинна дозволяти, використовуючи зв'язку «пароль-ім'я користувача», створювати нового користувача.
2) Система повинна зберігати інформацію про користувача в базі даних.
3) Система повинна перевіряти при створенні користувача, що користувач додатки не помилився при введенні пароля, за допомогою підтвердження пароля, і повідомляти його в іншому випадку.
4) Система повинна дозволяти користувачеві додатки виконувати вхід в свій аккаунт допомогою правильно введеної комбінацією «пароль-ім'я користувача».
5) Система після входу користувача повинна відображати профіль користувача.
6) Система при повторному відкритті програми після його закриття повинна автоматично виконувати вхід, якщо це можливо, якщо ні, то пропонувати користувачеві додатки повторно ввести комбінацію «пароль-ім'я користувача».
7) Система повинна дозволяти користувачеві виконати вихід з усіх пристроїв.
8) Система повинна дозволяти користувачеві зберігати нотатки в своєму акаунті.
9) Система повинна дозволяти користувачеві видаляти та змінювати його нотатки.

Розглянемо нефункціональні вимоги.

Система повинна функціонувати на мобільних пристроях з ОС Android, Ios, а так само на інших платформах через web інтерфейс.

Всі дані системи повинні зберігатися в одному місці (на сервері).

Проектована інформаційна система буде створена на основі клієнт-серверної архітектури. Архітектура клієнт-сервер – обчислювальна або мережева архітектура, в якій завдання або мережева навантаження розподілені між постачальниками послуг, званими серверами, і замовниками послуг, званими клієнтами. Фізично клієнт і сервер – це програмне забезпечення. Зазвичай, вони взаємодіють через комп'ютерну мережу за допомогою мережевих протоколів і знаходяться на різних обчислювальних машинах. Сервера очікують від клієнтських програм запити і надають їм свої ресурси у вигляді даних. У нашому випадку роль сервера виконує додаток на Node.js, клієнти – додатки

що виконуються на різних платформах (мобільні та web) написані на Titanium SDK. Клієнти посилають запити на різні операції з об'єктом користувача або мікро повідомлень, сервер в свою чергу працює з базою даних і повертає потрібні дані клієнтам. Спілкування між клієнтами і сервером йде через протокол передачі гіпертексту HTTP (англ. HyperText Transfer Protocol).

Використовуються такі методи HTTP:

– OPTIONS. Надсилається клієнтами для отримання списку дозволених методів;
– GET. Використовується для отримання запитуваного вмісту;
– POST. Застосовується для передачі даних до серверу;
– DELETE. Запит на видалення якихось даних.

Використовуваний формат даних для передачі в обидві сторони – JSON.

Серверна частина це ядро всієї інформаційної системи. Практично весь функціонал реалізується на сервері, це і створення користувачів, і система аутентифікації, і система нотаток повідомлень, робота з базою даних. Для того щоб надати доступ до всіх цих функцій додаткам клієнтам, на сервері необхідно реалізувати механізм який буде розуміти HTTP запити клієнтів і повертати їм потрібні дані – REST API. У REST API в якості формату запитів до сервера використовується URI – аналог гіперпосилання в веб браузері. Сервер розпізнає URI і метод HTTP запиту і виконує відповідні дії, потім повертає дані клієнта. Для створення REST API сервера, ми будемо використовувати спеціально призначене розширення для Node.js під назвою Restify. Систему управління базою даних ми виберемо основну для більшості Node додатків – MongoDB, а так само розширення для Node для роботи з нею – Mongoose. Насамперед створимо структуру нашої бази даних. У нас є дві сутності: користувачі та нотатки. Користувач може мати багато нотаток, а нотатка тільки одного користувача. Для сутності користувача нам потрібні такі атрибути:

– Id. Первинний ключ;
– Username. Унікальне ім'я користувача;
– Password_hash. Хеш-сума пароля для аутентифікації;
– Token. Унікальний ключ використовується для аутентифікації.

Атрибути для сутності нотаток:

– Id;
– User_id. Зовнішній ключ для зв'язку з користувачем;
– Body. Текст нотатки;
– Title. Тема повідомлення;
– Created_at. Час створення повідомлення.

Схема бази даних в нотації UML зображена на рис. 2.

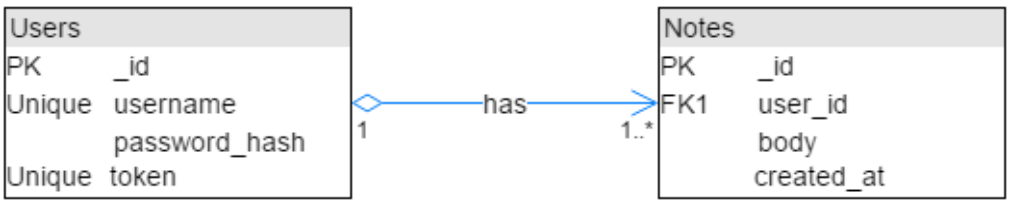


Рис.2. Схема БД в нотації UML

Наступним нашим кроком буде реалізація реєстрації (створення користувача). Для цього спочатку ініціалізуємо нашу базу даних і створимо модель користувача User (Додаток В), яка містить опис схеми моделі для її зв'язку з БД. Далі створимо в контролері користувача функцію «signup», яка відповідатиме за створення користувача. Функція «signup» отримує на вході такі аргументи: ім'я користувача; пароль.

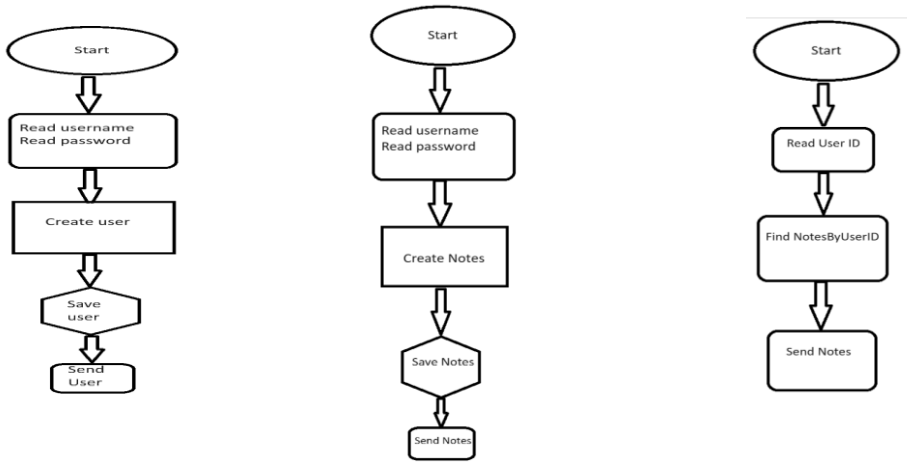
Далі функція обчислює хеш-суму пароля за допомогою алгоритму bcrypt [3], і генерує випадковий рядок, який буде використаний як секретний ключ при

аутентифікації. Далі функція створює модель з параметрами зазначеними вище, зберігає її в базу даних і повертає клієнту об'єкт користувача без поля з хеш-сумою пароля. Далі нам потрібно дати доступ до цієї функції клієнтам, для цього у файлі контролера маршрутів пропишемо маршрут POST HTTP запиту для URI: «/api/signup». Функція реєстрації на цьому закінчена. Блок схема даної функції зображена на рис 3a.
Наступним кроком буде реалізація системи аутентифікації. Для цього створимо функцію «signin» в контролері користувача. Вхідними аргументами цієї функції будуть ім'я користувача та пароль. Далі ми шукаємо відповідного користувача в базі даних і звіряємо хеш-суму пароля із запиту і хеш-суму пароля в базі даних, якщо вони збігаються, генеруємо новий секретний ключ в поле «token» і повертаємо об'єкт користувача клієнту, якщо ні, повертаємо користувачеві об'єкт з повідомленням про помилку. Прописуємо маршрут GET HTTP запиту для URI: «/api/signin».

Але кожен раз надсилати комбінацію «ім'я користувача-пароль» небезпечно і недоцільно, тому так само реалізуємо функцію аутентифікації по секретному ключу, який ми генерували раніше. Функція «signinByToken» повинна приймати на вході секретний ключ від клієнта, далі шукаємо його в базі даних і якщо він знайдений, повертаємо клієнту об'єкт користувача, якщо ні, повертаємо повідомлення про помилку. Прописуємо маршрут GET HTTP запиту для URI: «/api/user».

Останнім кроком системи аутентифікації буде реалізація функція виходу – logout. Функція записується за аналогією з попередніми функціями у файлі контролера користувача і виконує одну просту операцію, створює новий секретний ключ. Прописуємо маршрут GET HTTP запиту для URI: «/api/logout».

Далі реалізуємо систему нотаток, для цього створимо спочатку її модель. Потім створимо функції createNote і getNote і removeNote, які створюватимуть, отримуватимуть і видалятимуть нотатки. Доступ до цих функцій буде забезпечуватися через аутентифікацію по ключу доступу, тому в цих функціях об'єкт користувача буде завжди визначений. Функція createNote буде отримувати на вході текст нотатки. Далі функція буде створювати екземпляр моделі нотаток, і зберігати модель в БД. Блок схема даної функції зображена на рис3b.



a. b. c.

Рис. 3. Блок схеми функцій: «signup» (a), createNote (b), getNote (c)

Функція getNote отримує на вході id користувача і повертає клієнту всі пов'язані з поточним користувачем нотатки з БД. Блок схема даної функції зображена на рис.3c.

При розробці системи, нам требо генерувати унікальні ідентифікатори, які використовуються для аутентифікації користувачів.

Так як ідентифікатор, по суті, дає повний доступ до аккаунту користувача, до його генерації слід підійти серйозно, головна вимога до гарантії його унікальності, так як при співпаданні ідентифікаторів у різних користувачів, один з користувачів отримає

доступ до чужого аккаунту. Виходячи з цього ймовірність дворазового генерування одного і того ж ідентифікатора – виникнення колізії – повинна бути вкрай мала.

Так само ідентифікатор по можливості не повинен бути сильно довгим в цілях економії трафіку.

На ймовірність виникнення колізії впливають два фактори:

1) розмір ідентифікаційного простору – кількість можливих унікальних ідентифікаторів;

2) метод генерування ідентифікаторів – яким чином ідентифікатор вибирається із загального простору.

В ідеалі нам потрібно великий простір (щодо наших потреб), з якого випадково обираються рівномірно розподілені ідентифікатори. Тому для генерації випадкових рівномірно розподілених ідентифікаторів ми будемо конвертувати результат функції crypto.randomBytes(*N*), де *N* кількість повернених випадкових байт, у шістнадцятирічну строку. Ця функція, із пакету інструментів OpenSSL, є реалізацією криптографічно стійкого псевдовипадкового алгоритму який базується на «Вихрі Мерсена» [4]. Залишилося підрахувати скільки випадкових байт нас, влаштує, щоб ймовірність колізії була вкрай мала.

Представимо нашу задачу в наступному абстрактному вигляді: Дано n випадкових чисел з дискретного рівномірного розподілу з діапазоном $[1, H]$. Яка ймовірність $p(n, H)$, що, принаймні два числа збігаються?

Дана задача є узагальненням парадоксу днів народжень [5, 6]. Знайдемо зворотну ймовірність $\bar{p}(n, H)$ при $n < H$, таку що всі числа будуть різними:

$$\bar{p}(n, H) = 1 \times \left(1 - \frac{1}{H}\right) \times \left(1 - \frac{2}{H}\right) \times \ldots \times \left(1 - \frac{n-1}{H}\right). \quad (1)$$

Розкладання в ряд Тейлора експоненційної функції:

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots \quad (2)$$

дає наближення першого порядку для x при $x \ll 1$:

$$e^x \approx 1 + x. \quad (3)$$

Щоб застосувати це наближення до формули (1) покладемо $x = -a / H$. Таким чином, отримаємо:

$$e^{-\frac{a}{H}} \approx 1 - \frac{a}{H} \quad (4)$$

Отримана апроксимація:

$$p(n, H) \approx 1 - e^{\frac{-n(n-1)}{2H}} \approx 1 - \left(\frac{H-1)}{H}\right)^{\frac{n(n-1)}{2}}. \quad (5)$$

Тепер напишемо формулу для оберненої задачі:

$$n(p,H) \approx \sqrt{2H \times \ln\left(\frac{1}{1-p}\right)}. \quad (6)$$

Якщо повернуться до задачі ймовірності виникнення колізій ідентифікаторів, то H – буде позначати розмір ідентифікаційного простору, p –ймовірність колізії, n – кількість згенерованих ідентифікаторів. Розмір ідентифікаційного простору обчислюється таким чином:

$$H = 2^{8N}, \quad (7)$$

де N – довжина ідентифікатора в байтах.

Використовуючи формули (5) і (6), складемо таблицю ймовірностей колізій для визначення потрібної довжини ідентифікатора. Нижче наведено фрагмент цієї таблиці:

Таблиця 1

Таблиця ймовірностей колізій

N	$p(n,H)$		Розрахунок років			
	1E-12	0,10%	$Y_1(10^{-12})$	$Y_2(10^{-12})$	$Y_1(0.1\%)$	$Y_2(0.1\%)$
2	< 2	8	--	--	--	--
4	< 2	2073	--	--	--	--
8	4295	1,36E+08	1E-07	8E-03	0,004	258
12	2,81E+08	8,90E+12	9E-03	536	282,320	2E+07
18	4,72E+15	1,49E+20	149752	9E+09	5E+09	3E+14
22	3,09E+20	9,79E+24	1E+10	6E+14	3E+14	2E+19

Де, Y_1 – кількість років для досягнення заданої ймовірності колізії при тисячі генераціях в секунду (реальний проект), Y_2 – кількість років для досягнення заданої ймовірності колізії при однієї генерації в хвилину (навчальний проект).

Виходячи з перерахованих даних, для навчального проекту можна прийняти довжину ідентифікатора рівну 8 байтам, що гарантує, що за 258 років, ми досягнемо ймовірності колізії в 0,1%. Для реального проекту варто вибрати > 12 байт.

Відсоток переносимого коду обчислювався таким способом:

$$\textbf{ВПК} = \frac{\text{кількість рядків переносимого коду}}{\text{загальна кількість рядків коду}} \cdot 100\% = \frac{323}{341} \cdot 100\% = 94.7\% \quad (8)$$

Відсоток покриття коду тестами обчислювався таким способом:

$$\textbf{ВПКТ} = \frac{\text{кількість рядків коду покритих тестами}}{\text{загальна кількість рядків коду}} \cdot 100\% = \frac{307}{341} \cdot 100\% = 90\% \quad (9)$$

$$\frac{307}{341} \cdot 100\% = 90\%.$$

Висновки. Рооозв’язане актуальне завдання розробки та тестування інформаційної системи кросплатформних веб-додатків, використовуючи зв'язку фреймворків. Так само в роботі проведено аналіз новітніх методик та інструментів кросплатформної розробки та їх засобів автоматизованого тестування.

У підсумку розроблено багатоплатформний програмний продукт, який реалізує архітектуру «клієнт-сервер» та працює на платформах ios, android, і так само в будь-якому web браузері. Розроблений програмний продукт дозволяє користувачам зберігати та отримувати доступ до своїх нотатків на різноманітних платформах та редагувати їх.

При цьому досягнуто відсоток переносимості коду в 94.7%, що є відмінним результатом. Якби ми писали код окремо під 3 кожні платформи, то було б імовірно в 2,84 рази більше рядків коду, звідси час, витрачений на розробку, за рахунок використання цієї зв'язки фреймворків та архітектури, було скорочено в 2,84 рази.

Список літератури

1. Titanium Platform Overview. URL: https://titaniumsdk.com/guide/ Titanium_SDK/Titanium_SDK_Getting_Started/ Titanium_Platform_Overview.html
2. Spring Framework Advantages and Disadvantages. URL: http://www.aksindiblog.com/spring-framework-advantages-disadvantages.html
3. Bcrypt. URL: https://en.wikipedia.org/wiki/Bcrypt
4. Mersenne Twister Home Page. URL: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
5. Birthday problem . URL: https://en.wikipedia.org/wiki/Birthday_problem
6. Lundy M., Mees A. Convergence of an annealing algorithm. *Math. Programing.* 1996. V.34. P.111-124. URL: https://link.springer.com/article/10.1007/BF01582166

DEVELOPMENT AND TESTING OF WEB APPLICATIONS INFORMATION SYSTEM

Yu.Yu. Kozina[1], B.I. Yukhimenko[2], O.V. Ischenko[3]

National Odesa Polytechnic University,
1, Shevchenko Ave., Odesa, 65044, Ukraine
emails: yuliyakc21@gmail.com[1], biruteyu@gmail.com2, alesya.ishchenko@gmail.com[3]

In the conditions of the growth of information technologies, applications developed for usage on the different platforms – cross-platform applications – are gaining more and more popularity and need. This is due to the appearance and development of all kinds of devices on which they can function. One of the rapidly developing directions in the field of multi-platform is currently the development of applications that work on different operating systems, such as desktop – Windows, Mac OS, mobile – iOS, Android. However, the existing methods of their development are focused on creating applications for a specific operating system, which limits the universality of their application. The work solves the urgent task of developing and testing the information system of cross-platform web applications using the framework connection. Also, the work analyzes the latest methods and tools of cross-platform development and their means of automated testing. As a result, a multi-platform software product was developed, which implements the "client-server" architecture and works on the ios, android platforms, as well as in any web browser. The software product developed allows users to save and access their notes on various platforms and edit them.

Keywords: Testing, cross-platform applications, client-server architecture.