Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Програмний додаток аутентифікації та авторизації користувача»

Студента 4 курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ підпис студента — Фібрука Руслана Сергійовича

Науковий керівник кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Тищенко Дмитро Олександрович

Гарант освітньої програми кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Рзаєва Світлана Леонідівна

КИЇВ – 2023

Державний торговельно-економічний університет

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

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Фібруку Руслану Серігйовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту Система аутентифікації та авторизації користувача

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту є розробка та впровадження системи автентифікації та авторизації користувачів, яка забезпечить безперебійну та безпечну роботу користувачів при дотриманні найвищих стандартів безпеки. Система забезпечить механізми для перевірки особи користувачів та надання їм доступу до певних ресурсів на основі їхніх привілеїв та дозволів.

Об'єкт дослідження є програмний додаток, який забезпечує автентифікацію та авторизацію користувачів у веб-середовищі.

Предмет дослідження розробка є методи, технології та практики, пов'язані з автентифікацією та авторизацією користувачів у веб-розробці.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ЗАВДАННЯ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧА ВІДПОВІДНО ДО ЗАКОНУ

1.1 АНАЛІЗ ВИМОГ ДО АВТЕНТИФІКАЦІЇ В ІНФОРМАЦІЙНІЙ СИСТЕМІ

1.2 АНАЛІЗ НАЯВНИХ ТЕХНОЛОГІЙ АУТЕНТИФІКАЦІЇ

1.3 ТЕХНІЧНЕ ЗАВДАННЯ

1.4. ВИСНОВОК ДО РОЗДІЛУ 1

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОРИЗАЦІЇ

2.1 ОГЛЯД МОВ ПРОГРАМУВАННЯ

2.2 JAVASCRIPT ТА ЙОГО ФРЕЙМВОРКИ

2.3 ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ

2.4. ВИСНОВОК ДО РОЗДІЛУ 2

РОЗДІЛ 3. ТЕСТУВАННЯ ДОДАТКУ

3.1 ЗАГАЛЬНИЙ ПРОЦЕС ТЕСТУВАННЯ ДОДАТКІВ

3.2 ТЕСТУВАННЯ ГОТОВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.3. ВИСНОВОК ДО РОЗДІЛУ 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Аналіз завдання автентифікайії користувача відповідно до закону*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Розробка програмного забезпечення для авторизації*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Тестування додатку*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту Криворучко О.В
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Фібрук Р.С.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____Фібрука Р.С.__

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Рзаєва С.Л._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Розроблено програмне забезпечення для авторизації користувачів, що використовує мову програмування JavaScript. JavaScript є популярною та універсальною мовою, яка застосовується для веб-розробки як на стороні клієнта, так і на стороні сервера. Вона надає широкий набір бібліотек і фреймворків, що сприяють ефективній реалізації механізмів аутентифікації.

JavaScript є високорівневою інтерпретованою мовою програмування, яка дозволяє розробникам створювати інтерактивні веб-сторінки та динамічні веб-додатки. Вона підтримується всіма основними веб-браузерами, що робить її надійним вибором для кроссплатформенного розроблення. Крім того, JavaScript може використовуватись для розробки на стороні сервера за допомогою фреймворків, таких як Node.js.

Ключові слова: JS,ВЕБ ДОДАТОК ,АВТОРИЗАЦІЯ

ABSTRACT

We have developed user authorization software using the JavaScript programming language. JavaScript is a popular and versatile language used for web development on both the client and server sides. It provides a wide range of libraries and frameworks that contribute to the effective implementation of authentication mechanisms.

JavaScript is a high-level interpreted programming language that allows developers to create interactive web pages and dynamic web applications. It is supported by all major web browsers, making it a reliable choice for cross-platform development. In addition, JavaScript can be used for server-side development using frameworks such as Node.js.

Keywords: JS,WEB APPLICATION,AUTHORIZATION

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

XML (англ. Extensible Markup Language) – розширювана мова розмітки

					ДТЕУ 121 06-25.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		14.04.23	Програмний додаток аутентифікації та авторизації користувача	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		14.04.23		*ПС*	*2*	*63*
Гарант		Рзаєва С.Л.		14.04.23		*Факультет інформаційних технологій 4 курс, 6 група*		
Розробив		Фібрук Р. С.		14.04.23	*Перелік умовних скорочень*			

ЗМІСТ

ВСТУП

У сучасну цифрову епоху безпека та конфіденційність є першочерговими питаннями як для приватних осіб, так і для організацій. Зі зростанням значимості веб-додатків та онлайн-сервісів забезпечення автентифікації та авторизації користувачів стало критично важливим аспектом розробки програмного забезпечення. Здатність підтверджувати особу користувачів та контролювати їх доступ до конфіденційної інформації та функцій має вирішальне значення для запобігання несанкціонованому доступу та потенційним порушенням безпеки.

Ця дипломна робота спрямована на вирішення цих проблем шляхом розробки програмного додатку, який фокусується на автентифікації та авторизації користувачів, використовуючи мову програмування JavaScript. JavaScript, будучи універсальною і широко використовуваною мовою для веб-розробки, забезпечує надійну основу для створення безпечних і ефективних додатків, які можуть безперешкодно працювати на різних платформах.

Для досягнення цієї мети в проекті будуть використані різні стандартні методи автентифікації та авторизації, такі як автентифікація на основі паролів, автентифікація на основі токенів.

Також управління доступом. на основі ролей. Ці методи будуть реалізовані за допомогою фреймворків і бібліотек JavaScript, широко прийнятих у спільноті веб-розробників, таких як Node.js і Express.js.

Процес розробки складатиметься з кількох ключових етапів, включаючи збір вимог, проектування системи, реалізацію, тестування та розгортання. На кожному етапі ретельна увага приділятиметься найкращим практикам та методологіям безпеки, щоб забезпечити стійкість додатку до поширених вразливостей та загроз

Актуальність створеного додатку полягає в тому, що в сучасному цифровому світі безпека та конфіденційність користувачів є основними пріоритетами. Зі зростанням використання веб-додатків та онлайн-сервісів з'являється все більше потенційних загроз для безпеки інформації користувачів. Надійна автентифікація та авторизація є необхідними для запобігання несанкціонованому доступу до конфіденційних даних та виключення потенційних порушень безпеки. Отже, актуальність створеного додатку полягає у забезпеченні безпеки, захисту конфіденційної інформації та запобіганні несанкціонованому доступу до веб-додатків та сервісів, що є критичними аспектами в сучасній цифровій епосі.

Метою цього проекту є розробка та впровадження системи автентифікації та авторизації користувачів, яка забезпечить безперебійну та безпечну роботу користувачів при дотриманні найвищих стандартів безпеки. Система забезпечить механізми для перевірки особи користувачів та надання їм доступу до певних ресурсів на основі їхніх привілеїв та дозволів.

Завданням дослідження є створення зручного зрозумілого програмного додатку.

Об’єктом дослідження є програмний додаток, який забезпечує автентифікацію та авторизацію користувачів у веб-середовищі.

Предметом дослідження є методи, технології та практики, пов'язані з автентифікацією та авторизацією користувачів у веб-розробці.

РОЗДІЛ 1. АНАЛІЗ ЗАВДАННЯ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧА ВІДПОВІДНО ДО ЗАКОНУ

1.1 Аналіз вимог до автентифікації в інформаційній системі

З швидким зростанням мережевих систем і додатків, таких як електронна комерція, зростає попит на ефективну комп'ютерну безпеку. Більшість комп'ютерних систем захищені процесу ідентифікації користувача і аутентифікації. У той час як ідентифікація зазвичай являє собою непублічну інформацію, надану користувачами для самоідентифікації, і може бути відома системним адміністраторам та іншим користувачам системи, аутентифікація надає секретну, приватну інформацію, яка може підтвердити його особу. Існують різні підходи і техніки аутентифікації, від паролів до відкритих ключів.

Ще до появи комп'ютерів для аутентифікації людей використовувалися різні відмінні характеристики. Комп'ютерні системи застосували ці характеристики для аутентифікації користувача. Підходи до аутентифікації можна розділити на три типи відповідно до використовуваними ними відмінними характеристиками (Menkus, 1988), як показано на малюнку 1.1:

- Що знає користувач— аутентифікація на основі знань (наприклад, пароль, PIN-код, пароль доступу)
- Що є у користувача —аутентифікація на основі володіння (наприклад, карта пам'яті і токени смарт-карти)

- Хто такий користувач —аутентифікація на основі біометрії: фізіологічні (наприклад, відбиток пальця) або поведінкові (наприклад, динаміка клавіатури) характеристики

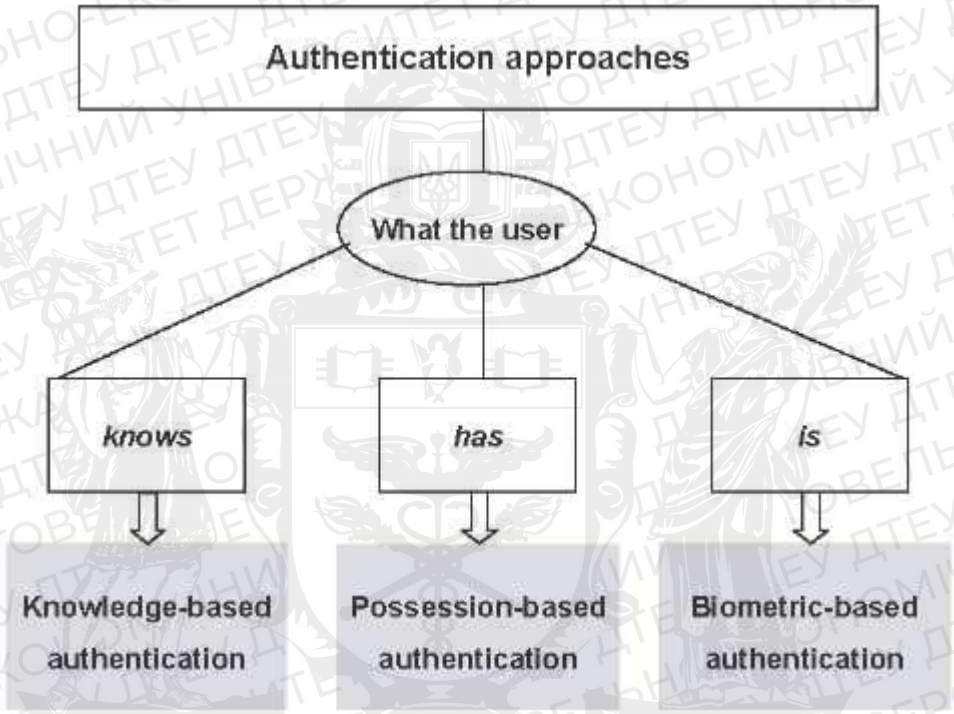

Рисунок 1.1- Класифікація методів аутентифікації

Оскільки всі ці типи аутентифікації мають переваги і недоліки, необхідно знайти компроміси між безпекою, простотою використання і простотою адміністрування. Типи аутентифікації можуть бути реалізовані окремо або в комбінації. Для посилення процесу аутентифікації рекомендується використовувати принаймні два типи. Кілька рівнів різних типів аутентифікації забезпечують значно більший захист.

Найбільш широко використовуваним типом аутентифікації є аутентифікація на основі знань.Приклади аутентифікації на основі знань включають паролі, фрази-паролі або пропозиції-паролі (Спектор і Гінзберг,

1994), графічні паролі (Торп і Ван Оршот, 2004; Виденбек, Уотерс, Биргет, Бродський і Мемон, 2005), особи-паролі (Бростофф і Сассе, 2000) та персональні ідентифікаційні номери (PIN-коди). Для перевірки і аутентифікації користувачів в незахищеною загальнодоступній мережі, такий як Інтернет, використовуються цифрові сертифікати і цифрові підписи. Вони надаються з використанням інфраструктури відкритих ключів (PKI), яка складається з пар відкритих і закритих криптографічних ключів (Adams & Lloyd, 1999).

Традиційною і, безумовно, найбільш широко використовуваною формою авторизації, заснованої на знаннях користувача, пароль (Zviran & Haga, 1993). Більшість комп'ютерних систем захищені ідентифікації користувача (наприклад, ім'я користувача або адресу електронної пошти користувача і пароля, як показано на малюнку 1.2

Рисунок1.2.-Аутентифікація за допомогою ідентифікації користувача і пароля

Пароль концептуально простий як для розробників системи, так і для кінцевих користувачів.Він складається з секретної серії символів в відповідності з деякими зумовленими правилами. Пара ідентифікатора

користувача і пароля діє як ідентифікація користувача та аутентифікація і служить для блокування несанкціонованого доступу до обчислювальних ресурсів. У більшості систем він може забезпечити ефективний захист при правильному використанні.

Однак відомо, що паролі схильні ряду підводних каменів з-за обмежень обробки інформації людиною (Sasse, Brostoff, & Weirich, 2001; Yan, Blackwell, Anderson, & Grant, 2005).По-перше, існує компроміс між запоминаемостью і безпекою. Паролі повинні бути важкими для вгадування і легкими для запам'ятовування. Той факт, що паролі, які важко вгадати і важко зламати, важко запам'ятати, що легко запам'ятовуються паролі легко вгадати і легко зламати, створює дилему для генерації паролів. Самий безпечний пароль - це випадкова рядок символів. Такі паролі важко вгадати іншим, але в той же час їх важко запам'ятати і, таким чином, вони змушують користувачів записувати їх, що погіршує їх секретність. Більш того, більшість користувачів мають кілька паролів для різних систем і додатків, що змушує їх запам'ятовувати кілька паролів. Щоб допомогти їм запам'ятати паролі, вони зазвичай вибирають значущі рядки, такі як імена, прізвиська або ініціали (Adams & Sasse, 1999), які легко запам'ятати, але також легко зламати. Вони також схильні дублювати свої паролі і, таким чином, викликають ефект доміно повторного використання пароля (Ives, Walsh, & Schneider, 2004); а саме, всі системи з однаковим паролем не більш безпечні, ніж сама слабка система, що використовує цей пароль.

Щоб підвищити безпеку паролів і захистити їх від атак за словником і методом перебору, політики паролів повинні бути реалізовані правила вибору і підтримки паролів (Smith, 2002). Основними правилами є:

- Паролі без словника та без імені.
- Досить довгі паролі зі змішаними типами символів.
- Старіння пароля і неможливість повторного використання.

- Складні паролі з використанням скорочень, рим і мнемонічних фраз, які важко вгадати і легко запам'ятати (Карстенс, Макколі-Белл, Мелоун і Демара, 2004; Ян та ін, 2005).
- Паролі не повинні передаватися разом і не повинні записуватися.
- Кількість невдалих спроб аутентифікації повинна бути обмежена системою.
- Паролі ніколи не повинні зберігатися відкритим текстом; вони повинні бути зашифровані або хэшированы.

Паролі, засновані на вищезгаданих правилах, більш ефективні, їх важче ідентифікувати і визначити з допомогою утиліт для злому. Щоб подолати проблему перехоплення паролів при аутентифікації через Інтернет, використовуються одноразові паролі. Одноразовий пароль може бути реалізований з допомогою смарт-карт — різновиди аутентифікації на основі володіння, обговорюваної нижче.

Паролі, використовувані в якості першого рівня аутентифікації, які дозволяють отримати доступ до ресурсів інформаційної системи через операційні системи, зазвичай називаються первинними паролями. Паролі, які використовуються в якості другого рівня аутентифікації, для подальшого контролю і захисту багаторівневого доступу до сегментів цих ресурсів, таким як конфіденційні програми або файли даних, зазвичай називаються вторинними паролями (Zviran & Haga, 1993).

При визначенні основних паролів виробник операційної системи використовує паролі, згенеровані системою, або паролі, згенеровані користувачем, з передбаченими правилами. Показано, що згенеровані користувачем паролі легше запам'ятати, але менш безпечні, ніж згенеровані системою паролі, оскільки їх можна легко вгадати (Lopez, Oppliger, & Pernul, 2004).

Щоб подолати труднощі із запам'ятовуванням паролів, був запропонований метод введення паролів у формі питань і відповідей (Haga & Zviran, 1991). Цей метод в основному використовується для вторинних паролів. Це включає в себе діалог між користувачем і системою, як показано на малюнку 3.

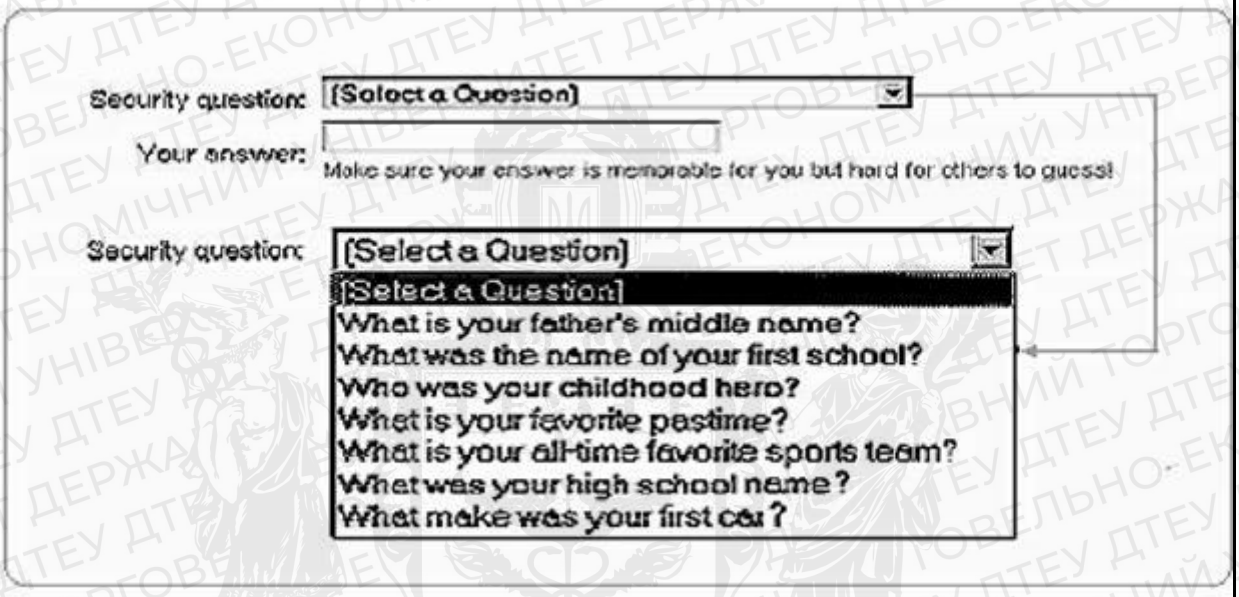

Рисунок 1.3- Приклад пароль в формі запитань та відповідей

У типовому сесії питань і відповідей користувачеві пропонується кілька

випадково вибраних коротких питань з набору питань, що зберігаються в його профілі в операційній системі. Доступ до системи або конкретного додатком надається тільки при збігу відповідей користувача з тими, які зберігаються в його / її профілі.

Існує два основних типи паролів типу "запитання-відповідь": когнітивні паролі і асоціативні паролі (Баннелл, Пі, Хендерсон, Нейпір і Кеннеді-Моффат, 1997; Хага і Звиран, 1991; Звиран і Хага, 1993). У когнітивних паролі користувач повинен надати системі відповіді на особисті питання, засновані на фактах або думках, такі як дівоче прізвище матері

користувача (на основі фактів) або улюблений тип музики користувача (на основі думок).

У разі асоціативних паролів користувач повинен надати системі набір словесних асоціацій, що складається з реплік, так і з унікальних пов'язаних з ними відповідей.

Аутентифікація на основі володіння, звана також аутентифікацією на основі токенів, заснована на тому, що є у користувача. Він використовує в основному фізичні об'єкти, якими володіє користувач, наприклад токени. Крім того факту, що пред'явлення дійсного сертифіката не доводить права власності, оскільки він міг бути вкрадений або дубльований якимось витонченим шахрайським способом (Svi-gals, 1994), існують проблеми адміністрування та незручності для користувачів, пов'язані з необхідністю їх носіння.

Токени зазвичай діляться на дві основні групи: токени пам'яті та смарт-токени.Маркери пам'яті зберігають інформацію, але не обробляють її. Найбільш поширеним типом токена пам'яті є магнітна картка, що використовується в основному для аутентифікації разом з механізмом аутентифікації, заснованим на знаннях, таких як PIN-код. Токени пам'яті недорогі у виробництві. Використання їх з PIN-кодами забезпечує значно більшу безпеку, ніж одні лише PIN-код або пароль.

На відміну від токенів пам'яті, інтелектуальні токени включають у себе одну або кілька вбудованих інтегральних схем, які дозволяють їм обробляти інформацію. Подібно токенам пам'яті, більшість інтелектуальних токенів використовуються для аутентифікації разом з механізмом аутентифікації на основі знань, таким як PIN-код. З різних типів смарт-токенів найбільш широко використовуються ті, в яких вбудований чіп, що містить мікропроцесор. Їх переносимість і криптографічні можливості призвели до їх широкого використання в багатьох додатках для віддаленої роботи та

електронної комерції (Juang, 2004; Ku & Chen, 2004; Wu & Chieu, 2003). Із-за своєї складності смарт-токени коштують дорожче, ніж токени пам'яті, але забезпечують більшу гнучкість і безпеку, і їх складніше підробити. Через їх високого рівня безпеки смарт-токени також використовуються в якості одноразових паролів для аутентифікації у відкритих мережах.

Аутентифікація на основі біометрії заснована на те, ким є користувач, а саме, автоматична ідентифікація з використанням певних анатомічних, фізіологічних або поведінкових особливостей і характеристик, пов'язаних з користувачем (Kim, 1995; Wayman, Jain, Maltoni, & Maio, 2004).

Біометрична аутентифікація заснована на тому факті, що певні фізіологічні або поведінкові характеристики надійно відрізняють однієї людини від іншого. Таким чином, можна встановити особу, засновану на те, ким є користувач, а не на тому, що користувач має або знає і пам'ятає. Біометрія включає в себе збір, так і порівняння цих характеристик. Біометричну систему можна розглядати як систему розпізнавання образів, що складається з трьох основних модулів: (1) модуля датчика, (2) модуля виділення ознак і (3) модуля зіставлення ознак. Особисті атрибути користувачів фіксуються та зберігаються у файлах посилань для порівняння з подальшою аутентифікацією, щоб визначити, чи існує збіг. Точність різних біометричних систем може бути оцінена шляхом вимірювання двох типів помилок (Matyas & Stapleton, 2000): (1) помилкове відхилення, тобто помилкове невідповідність (помилка типу I) і (2) помилкове прийняття, тобто помилкове збіг (помилка типу II). В біометричній системі, яка забезпечує високий рівень аутентифікації, частота цих двох помилок невелика.

Біометрична аутентифікація технічно складна і зазвичай дорога, оскільки вимагає спеціального обладнання.Хоча всі біометричні технології

по своїй суті схильні деякого рівня помилкового збігу або помилкового невідповідності, вони володіють високим рівнем безпеки. Незважаючи на їх високу безпеку, вони не мають високого рівня прийняття користувачами, оскільки сприймаються як нав'язливі і посягають на конфіденційність (Prabhakar, Pankanti, & Jain, 2003) за допомогою автоматизованих засобів. Вони також порушують етичні питання потенційного неправильного використання особистої біометрії, наприклад, для відстеження та моніторингу продуктивності (Alterman, 2003). Таким чином, вони не користуються популярністю і в основному використовуються в системах з дуже високим рівнем безпеки.

Поява біометричної аутентифікації вирішило проблеми, з якими стикалися традиційні методи перевірки, надавши найбільш ефективний і точний метод ідентифікації, має перевагу перед традиційними методами безпеки в тому, що його можна легко вкрасти або передати іншим користувачам. Біометричні системи також підвищують зручність користувачів, позбавляючи їх від необхідності визначати і запам'ятовувати паролі. Однак, незважаючи на зручність, цифрове сканування або шаблон уразливі для мережевого аналізу та після крадіжки більше не можуть бути використані (Ives et al., 2004).

Існують різні види біометрії (Matyas & Staple-ton, 2000), і вони зазвичай поділяються на дві основні категорії: фізіологічну і поведінкову біометрію. Фізіологічна біометрія, заснована на стабільних фізичних характеристиках користувача. Найбільш відомими з них є відбитки пальців, геометрії кисті, сканування райдужної оболонки, сітківки та особи. Відбитки пальців є найбільш широко використовуваної фізіологічною характеристикою в системах, які автоматично розпізнають особистість користувача (Ratha & Bolle, 2005; Wayman et al., 2004). Прикладом одного з його сучасних програм

є використання ідентифікації і аутентифікації на основі відбитків пальців для підтримки онлайн-екзаменів на основі веб-курсів (Auernheimer & Tsai, 2005).

Поведінкова біометрія, заснована на поведінкових атрибути користувачів, які є завченими рухами (Guven & Sogukpinar, 2003; O Gorman, 2003; Yu & Cho, 2004). Найбільш відомими з них є: динаміка натискання клавіш, динаміка підпису, динаміка миші і перевірка мови або голосу.

При виборі методу аутентифікації необхідно враховувати ряд факторів: ефективність, простота реалізації, зручність використання, а також відношення користувача і його прийняття (Фернелл, Доуленд, Иллингворт і Рейнольдс, 2000). У таблиці 1 показано ранжування трьох типів аутентифікації згідно з цими чотирма факторами.

Тип аутентифікації на основі знань є недорогим і простим у реалізації і зміні.На жаль, його легше всього скомпрометувати, і він менш безпечний, ніж токени або методи аутентифікації на основі біометрії, які по своїй суті більш безпечні. З іншого боку, токени і методи біометричної аутентифікації більш дороги в реалізації. Ставлення користувачів до аутентифікації на основі знань вельми позитивне, менш позитивне до аутентифікації на основі володіння і негативне до аутентифікації на основі біометрії (Deane, Barrelle, Henderson, & Mahar, 1995; Prabhakar et al., 2003). Оскільки аутентифікація на основі знань менш ефективна, ніж інші типи, рекомендується використовувати її при аутентифікації двох типів; наприклад, пароль і токен або пароль і натискання клавіші (Furnell, Papadopoulos, & Dowland, 2004; Yu & Cho, 2004).

Паролі забезпечують найбільш економічне рішення, вони перенесені в інші додатки, прості в розгортанні і масштабування для необмеженого числа користувачів. Вони інтегровані у багато операційні системи, і користувачі знайомі з ними.

Таблиця 1.1- Ранжування типів аутентифікації від 1 (низький) до 3 (високого)

		Чинник		
Тип аутентифікації	**Ефективність безпеки**	**Легкість реалізація**	**Простота використання**	**Відношення користувача та прийняття**
Заснований на знаннях	1	3	3	3
Заснований на володінні	2	2	2	2
Заснований на біометрії	3	1	1	1

Аутентифікація на основі володіння з використанням токенів забезпечує більш високий рівень безпеки, ніж аутентифікація на основі знань. Більшість проблем, пов'язаних з токенами, пов'язані з їх вартістю, адмініструванням, втратою і незадоволеністю користувачів. Через уразливості до крадіжки токен слід використовувати не окремо, а з іншим типом аутентифікації. У поєднанні з паролем це може забезпечити значні переваги, оскільки може зберігати або генерувати кілька паролів, і користувач повинен запам'ятати тільки один пароль, необхідний для доступу до токена.

Оскільки паролі порівняно недорогі, прості у використанні і привабливі для користувачів, вони, ймовірно, залишаться найбільш широко використовуваною формою аутентифікації в осяжному майбутньому. При належному управлінні вони можуть забезпечити ефективну безпеку. Необхідні подальші дослідження трьох типів аутентифікації, щоб удосконалити технології їх і зробити можливим розробку більш зручних та ефективних систем безпеки.

Комп'ютерні системи захищені трьома основними типами підходів до аутентифікації: (1) на основі знань, (2) на основі володіння і (3) на основі біометрії. Кожен з них має як переваги, так і недоліки. При виборі методу аутентифікації ми повинні враховувати компроміс між ефективністю безпеки, простотою впровадження, зручністю використання, а також ставленням користувача та його прийняттям. Щоб підсилити процес аутентифікації, рекомендується використовувати принаймні аутентифікацію двох типів.

1.2 Аналіз наявних технологій аутентифікації

З моменту їх розробки і впровадження обчислювальні системи були загальними пристроями, яким не вистачало якоїсь форми безпеки або конфіденційності створених і збережених даних. На початку 1960-х років Массачусетський технологічний інститут (MIT) розробив операційну систему з поділом часу, відому як Compatible Time-Sharing System (CTSS).

Ця система дозволяла кільком німим терміналів одночасно спільно використовувати ресурси одного централізованого комп'ютера. Це призвело до проблем із загальними файловими системами без властивої їм безпеки. Щоб створити безпечну файлову систему, в 1961 році Фернандо Корбато, співробітник Обчислювального центру Массачусетського технологічного інституту і засновник CTSS, вирішив цю проблему відсутності безпеки з допомогою паролів для аутентифікації користувачів до певних зберігаються даних і файлів. Однак Аллан Шерр, дослідник Массачусетського технологічного інституту, виявив, що серверні системи зберігають паролі у файлі майстер-паролів в легкодоступному місці, що дозволяє отримати доступ до будь-яких захищених паролем файлів. У 1970-х роках дослідник Bell Labs Роберт Морріс розробив метод захисту файлу майстер-пароля операційної системи Unix. Морріс використовував криптографічний метод,

відомий як "хеш-функція", яка робила пароль нечитабельним для людського ока, але не для комп'ютерної системи. Ця базова концепція незабаром була прийнята більшістю інших операційних систем.

Щоб отримати доступ до даних або сервісу, спочатку необхідно встановити справжність користувача за допомогою аутентифікації. Аутентифікація - це процес успішної автентифікації людини або пристрою [1]. Коли ми використовуємо банківську картку для здійснення покупки, ми аутентифицируемся, маючи карту і знаючи Особистий ідентифікаційний номер (PIN). Аутентифікація стала більш важливою з моменту широкого використання комп'ютерів. Уособлення користувача є критичною загрозою безпеки будь-якої комп'ютерної системи, і першим механізмом захисту від таких атак є аутентифікація користувача. Дані, які використовуються для підтвердження ідентифікації користувача, можуть бути підрозділені на три класи:

- Засновані на знаннях, які включають паролі, PIN-коди
- Засновані на володінні, які включають смарт-карти і токени
- Засновані на наслідуванні, такі як біометричні дані, які містять відбитки пальців і сканування сітківки

З часом, коли зловмисники розібралися, як використовувати алгоритми хешування "грубою силою", індустрія поліпшила хеш-функції і включила додаткові компоненти рандомізації. Наприклад, соління, щоб зробити хешований пароль унікальним. Створення Робертом Моррісом методів зберігання паролів на основі хеш-в 1970-х роках підвищило безпеку систем аутентифікації.

Інші криптографічні підходи, крім хешування, ефективні для аутентифікації. Однією з таких технологій є криптографія з відкритим ключем, або асиметрична криптографія. На початку 1970-х років було виявлено, що використовується асиметрична криптографія та відкриті /

закриті ключі. Хоча ці методи шифрування не були оприлюднені до 1990-х років, загальнодоступні дослідники самостійно виявили нові методи використання технології асиметричних ключів в кінці 1970-х років, що призвело до розробки широко використовуваного алгоритму асиметричного ключа RSA. В області аутентифікації цифрові сертифікати і підписи придбали вирішальне значення.

Дослідники і кіберзлочинці розробили нові способи використання паролів, оскільки все більше цифрових систем залежало від них в плані захисту. Як наслідок, галузь постійно прагне запровадити нові способи захисту процесу аутентифікації. Один з найбільших недоліків типової системи постійних паролів полягає в тому, що якщо зловмисник може присвоїти, вкрасти або підслухати чиї облікові дані, він може відтворити їх. Щоб протидіяти цьому, що, якщо пароль користувача змінювався кожен раз, коли він або вона входили в систему? Дослідники розробили стратегії, що дозволяють відрізняти людей від комп'ютерів в кінці 1990-х років. Ці методи, відомі як повністю автоматизований публічний тест Тюрінга для розрізнення комп'ютерів і людей (CAPTCHA). КАПЧІ не можуть бути використані для аутентифікації користувача, але їх можна використовувати для захисту від деяких атак автоматичної аутентифікації.

Настав час багатофакторної аутентифікації (MFA), яка все ще перебуває в стадії розробки, але отримала велике поширення в 2000-х роках. Паролі є найбільш використовуваною формою цифровий аутентифікації. Проте вони демонструють свій вік і неміч. Паролі є гарним механізмом аутентифікації при правильному використанні і дотриманні суворих правил безпеки. Проблема в тому, що більшість людей не дотримуються рекомендованих методів, і багато компаній, які обробляють паролі, також їм слідують. Незліченні витоку бази даних паролів сталися в результаті такого неправильного управління паролями за останні кілька десятиліть,

демонструючи, що самі по собі паролі нездатні захистити наші онлайн-ідентифікатори. MFA може вирішити цю проблему і допомогти її вирішити, але системи аутентифікації та альтернативні варіанти часто непомірно дороги або складні в реалізації. Сучасні мобільні телефони прокладають шлях до аутентифікації майбутнього. У 2010-х роках широка доступність смартфонів зробила технології біометрії і двофакторної аутентифікації (2FA) і MFA більш доступними для широкої публіки.

Завдяки своїй простоті і зручності у використанні SFA широко використовувалася, наприклад, з допомогою пароля (або PIN-коду) для підтвердження особи користувача. Паролі складаються з комбінації букв, цифр і спеціальних символів. Чим складніше комбінація з перерахованих вище, тим надійніше пароль і, отже, тим важче зловмиснику його виявити.

Середньостатестична людина володіє приблизно двадцятьма п'ятьма онлайн-акаунтами, але тільки у половини користувачів різні паролі в кожному обліковому записі. Реальність така, що одному користувачеві доводиться запам'ятовувати безліч паролів. Як наслідок, більшість людей віддають перевагу легкості, а не безпеки. Багато людей вибирають прості паролі, а не безпечні.

Надзвичайно прості паролі, які можуть включати ім'я користувача, дату народження і т. д., уразливі для фішингових атак. Паролі мають численні недоліки і самі по собі більше не ефективні для захисту даних, до яких здійснюється доступ і які передаються через Інтернет. Безпека облікового запису користувача може бути поставлене під загрозу, якщо пароль буде переданий іншим користувачам або виявлено. Неавторизований користувач може застосувати грубу силу у формі атак за словником чи методів соціальної інженерії для отримання доступу. Хакери можуть просто використовувати вільно доступні інструменти, які можуть бути

автоматизовані для підбору пароля користувача, пробуючи всі можливі комбінації, поки не знайдуть збіг.

З-за цілого ряду проблем з безпекою було виявлено, що SFA не може забезпечити ефективну безпеку. 2FA підвищує безпеку за рахунок об'єднання репрезентативних даних (комбінація імені користувача і пароля) з іншою формою ідентифікації, такий як фактор особистого володіння, який може включати захищений токен, використовує одноразовий пароль (OTP) [2]. 2FA може бути складений з трьох різних типів груп факторів, як показано на малюнку 1.4:

Фактор володіння — річ, яка є у користувача, наприклад, стільникові телефони

Фактор знання — річ, про яку відомо користувачеві, наприклад, пароль.

Біометричний фактор — факт про біометрії або поведінці користувача

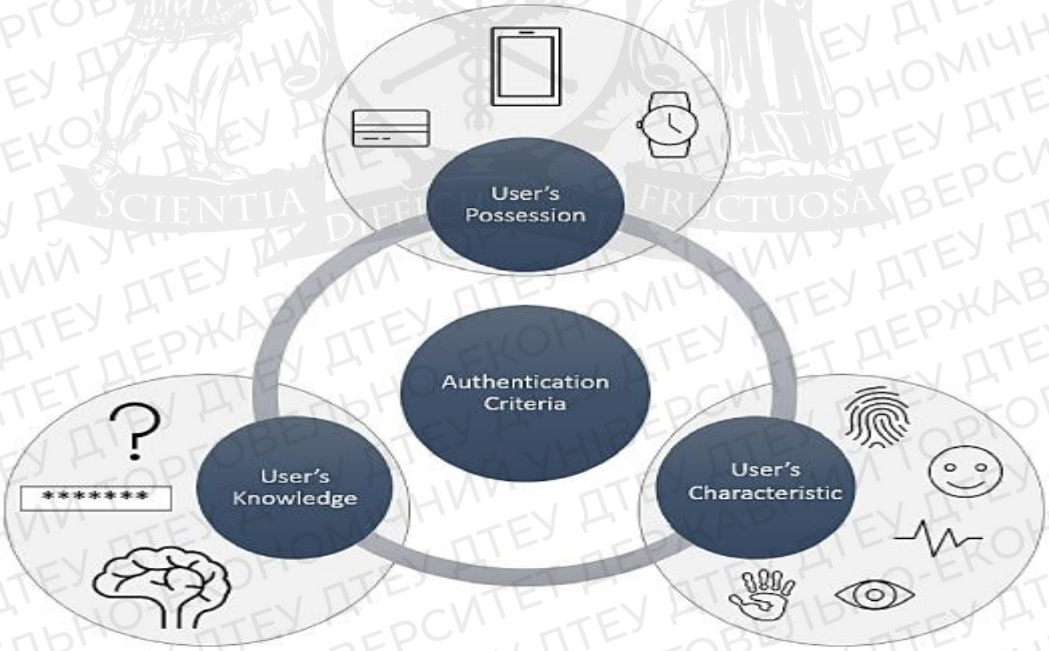

Рисунок 1.4-Критерії аутентифікації.

В даний час необхідно підвищувати рівень безпеки, оскільки атаки стають більш цілеспрямованими, а наслідки несанкціонованого доступу серйозними. Це особливо поширено для банківських платформ або платформ обробки персональних даних. Тепер вкрай важливо забезпечити більший контроль над перевіркою особистості людини, що намагається одержати доступ до цих систем. З урахуванням цих додаткових вимог немає сумнівів у тому, що пропонована захист значно вище, але в деяких випадках її все ще недостатньо. Це створює необхідність у створенні більшої кількості рівнів аутентифікації, які дозволять максимально підвищити безпеку.

Для вирішення цієї проблеми все більш поширеним стає MFA (див. малюнок 1.5). MFA зазвичай включає унікальні біологічні характеристики користувача, такі як відбиток пальця або сканування райдужної оболонки ока, оскільки вони часто відрізняються високою точністю при їх створенні та використанні [4]. Це спосіб запропонувати підвищений рівень безпеки для захисту комп'ютерного обладнання та інших життєво важливих служб від несанкціонованого доступу шляхом сполучення принаймні трьох типів облікових даних [5].

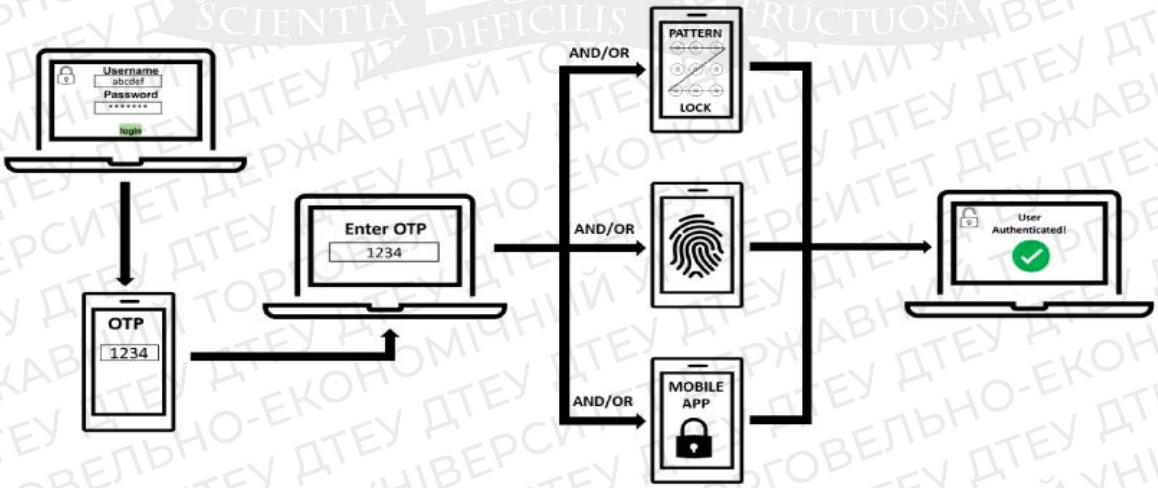

Рисунок 1.5-Багатофакторна аутентифікація.

Розглянемо повсякденну практику зняття готівки в Автоматичному касовому апараті (АТМ). Щоб отримати доступ до особистого кабінету і

вивести гроші, користувач повинен надати фізичний токен (банківську картку), який представляє фактор володіння, в той час як фактор знання представлений PIN-кодом. Цю систему можна було б легко зробити більш безпечною, додавши додатковий біометричний механізм.

Біометрія вносить свій внесок у MFA, комбінуючи фактори знань і володіння з біометричними чинниками для підвищення надійності ідентифікації, що ускладнює для суб'єкта погрози обман системи з допомогою уособлення. Оцінка багатьох біологічних особливостей, пов'язаних з ідентифікацією особистості індивіда, може значно поліпшити роботу системи MFA. Сканер відбитків пальців став найбільш часто використовуваним біометричним інтерфейсом з точки зору зручності використання. В основному це пов'язано з широким впровадженням виробниками смартфонів. Використання попередньо інтегрованих систем знижує вартість системи аутентифікації та спрощує її використання кінцевими користувачами. Одним з найбільш важливих елементів, які слід враховувати в сучасних системах аутентифікації, є компроміс між зручністю використання і безпекою. Підхід МЗС допускає широкий спектр ситуацій, у яких безпека має першорядне значення. Деякі з них докладно описані нижче [6]:

- Масові відкриті онлайн-курси (МООС), де важко відрізнити зареєстрованого користувача від користувача, який буде здавати іспит або робити домашнє завдання. З появою МООС в коледжах зараз, як ніколи, важливо надійно перевіряти особистості студентів. Оскільки набір факторів аутентифікації варіюється і може бути створено навіть в залежності від складності завдання, MFA є підходящим рішенням для підтвердження студентських посвідчень особи з-за його узгодженості і масштабованості.

- Банківські програми, такі як електронний грошовий переказ або онлайн-платежі, повинні бути захищені. MFA можна використовувати для швидкої перевірки авторизованих користувачів. Перекладені грошові кошти можуть бути визначальним чинником, так що для успішної ідентифікації користувачів необхідні зміни ідентифікаційних даних, відповідно, для менших сум грошей може бути обрана менш сувора факторна аутентифікація.
- Безпечний доступ до всіх видів електронних медичних записів може бути легко пов'язаний з Міністерством охорони здоров'я. Ця медична інформація є конфіденційною і приватної, і вона повинна бути захищена. Ідентифікуючи пристрій користувача, носій інформації і оточення, MFA може визначити метод автентифікації, що призводить до більш безпечної аутентифікації.

Використання біологічних елементів тягне за собою кілька недоліків, в основному з точки зору простоти використання, що надає важливе вплив на корисність системи MFA. З точки зору біометричної аутентифікації невідповідність між виміряним біометричним поданням і даними, записаними за первісною біометричної реєстрації, може бути проблематичним, особливо при використанні недорогого і неточного обладнання. Помилкові показники прийняття (FAR) і помилкові показники відхилення (FRR) - це проблеми, пов'язані з біометричною аутентифікацією. FAR і FRR надзвичайно важливі для роботи MFA, оскільки досягнення повної точності в рамках цих двох показників практично неможливо.

Варто відзначити, що в багатьох випадках інформація про фактичне місцезнаходження користувача також враховується при спробі аутентифікації. Ця процедура аутентифікації використовує розташування людини для полегшення ідентифікації. Цей критерій зазвичай використовує

системи Глобальної системи позиціонування (GPS), IP-адресу користувача або навіть ідентифікатор hive tower. Система використовує географічне розташування користувача і визначає, чи може бути виконана спроба входу в систему. Наприклад, якщо успішна спроба входу в систему була зроблена в одному географічному розташуванні, а через кілька хвилин зроблена ще одна спроба в зовсім іншому місці, спроба може бути відхилена, а обліковий запис заблокований для запобігання підозрілої поведінки [6].

1.3. Технічне завдання

1. Загальні відомості

1.1. Найменування системи

1.1.1. Повне найменування системи - SecureAccess Pro

1.1.2. Скорочене найменування системи - SecureAccess Pro

1.2. Планові терміни початку та закінчення робіт

Початок робіт - 01.01.2023

Закінчення робіт - 01.05.2023

1.3. Порядок оформлення і пред’явлення результатів робіт

Результати будуть представлені у вигляді програмного коду, текстової части з поясненням алгоритмів, принципів роботи. Результати будуть представлені для замовника в електронному форматі, додатково можуть бути представлені в текстувому форматі за запитом замовника.

Побудова системи для аутентифікації та авторизації користувача ділиться на розробку логіки програмного додатку та розробку інтерфейсу для користувача.

Виходячи з цього було сформульовано наступні завдання:

— Розробити зручний інтерфейс;

— Знайти безпечний URL-адрес засіб представлення претензій між двома сторонами;

— Забезпечити безпеку паролів під час автентифікації;

— Впровадити захищені канали зв’язку (наприклад, HTTPS) для шифрування конфіденційних даних.

— Забезпечити швидкію додатку

1.4 Потенційні користувачі системи

Система розраховано на старших людей які хочуть контролювати доступ до конфіденційної інформації та мати функцій для запобігання несанкціонованому доступу та потенційним порушенням безпеки шляхом підтвердження особи.

1.5 Практична значимість

Створений програмний продукт надає можливість збільшити безпеку та конфіденційність користувачів. Адже через збільшення використання веб-додатків та онлайн-сервісів з'являється все більше потенційних загроз для безпеки інформації користувачів. Надійна автентифікація та авторизація є необхідними для запобігання несанкціонованому доступу до конфіденційних даних та виключення потенційних порушень безпеки. Отже, практична знасиміть створеного додатку полягає у забезпеченні безпеки, захисту конфіденційної інформації та запобіганні несанкціонованому доступу до веб-додатків та сервісів, що є критичними аспектами в сучасній цифровій епосі.

1.4. Висновок до розділу 1

В результаті проведених досліджень було сформовано детальне технічне завдання, яке допоможе розробити якісний продукт – «Програмний додаток аутентифікації та авторизації користувача». Також було розглянуто важливість безпеки та конфіденційності в сучасній цифровій епосі. Було визначено, що автентифікація та авторизація користувачів є критично важливими аспектами розробки програмного забезпечення, що дозволяють

підтверджувати особу користувачів і контролювати їх доступ до конфіденційної інформації та функцій.

Ця робота має на меті вирішити ці проблеми шляхом розробки програмного додатку, який фокусується на безпеці автентифікації та авторизації користувачів з використанням мови програмування JavaScript. Для досягнення цієї мети будуть використані різні методи та фреймворки JavaScript, що дозволяють створювати безпечні та ефективні додатки.

Процес розробки додатку буде включати кілька етапів, включаючи збір вимог, проектування, реалізацію, тестування та розгортання. На кожному етапі будуть застосовуватись найкращі практики та методології безпеки для забезпечення безпеки додатку та захисту від вразливостей та загроз..

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОРИЗАЦІЇ

2.1 Огляд мов програмування

Розробка програмного забезпечення для авторизації вимагає вибору мови програмування, яка здатна ефективно обробляти автентифікацію користувача та протоколи безпеки. У цьому випадку JavaScript буде обраною мовою для завдання. JavaScript — це популярна та універсальна мова програмування, яка широко використовується для веб-розробки завдяки її здатності працювати як на стороні клієнта, так і на стороні сервера. Він надає багатий набір бібліотек і фреймворків, які роблять його придатним для реалізації механізмів автентифікації.

JavaScript — це інтерпретована мова програмування високого рівня, яка дозволяє розробникам створювати інтерактивні веб-сторінки та динамічні веб-додатки. Він підтримується всіма основними веб-браузерами, що робить його надійним вибором для створення кросплатформних програм. Універсальність JavaScript виходить за межі веб-браузера, оскільки його можна використовувати для розробки на стороні сервера за допомогою таких фреймворків, як Node.js.

Коли мова йде про впровадження систем авторизації, JavaScript пропонує кілька бібліотек і фреймворків, які спрощують процес. Серед популярних варіантів:

Passport.js: Passport.js є широко використовуваним проміжним програмним забезпеченням автентифікації для Node.js. Він надає повний

набір стратегій для запитів автентифікації, включаючи локальне ім'я користувача/пароль, дані для входу в соціальні мережі.

Веб-токени JSON (JWT): JWT — це компактний, безпечний для URL-адрес засіб представлення претензій між двома сторонами. Він зазвичай використовується для автентифікації та авторизації. JavaScript надає такі бібліотеки, як jsonwebtoken, для кодування та декодування JWT, що дозволяє розробникам безпечно передавати та перевіряти інформацію користувача.

bcrypt.js: Безпека паролів має вирішальне значення для систем автентифікації. bcrypt.js — це бібліотека JavaScript, яка забезпечує функції хешування та соління для безпечного зберігання та перевірки паролів. Він допомагає захистити дані користувачів, зберігаючи паролі в незмінному форматі, що значно ускладнює зловмисникам відновлення оригінальних паролів.

OAuth: OAuth — це відкритий стандарт авторизації, який дозволяє користувачам надавати доступ до своєї інформації на одному веб-сайті іншому веб-сайту, не відкриваючи своїх паролів. Бібліотеки JavaScript, такі як Passport.js і OAuth.js, спрощують реалізацію потоків автентифікації на основі OAuth, дозволяючи користувачам входити, використовуючи свої існуючі облікові записи на таких платформах, як Google, Facebook або Twitter.

Крім того, фреймворки JavaScript, такі як React і Angular, пропонують надійні інструменти для створення інтерфейсів користувача та обробки взаємодії на стороні клієнта, які можна інтегрувати з логікою автентифікації.

Розробляючи програмне забезпечення для авторизації, дуже важливо дотримуватися найкращих практик безпеки для захисту даних користувачів. Це включає впровадження захищених каналів зв'язку (наприклад, HTTPS),

шифрування конфіденційних даних, використання надійних методів автентифікації та застосування належного контролю доступу.

Python — це універсальна та широко використовувана мова програмування, яка може ефективно керувати розробкою програмного забезпечення для цілей авторизації. Python пропонує багату екосистему бібліотек і фреймворків, які полегшують впровадження систем автентифікації.

Однією з відомих бібліотек Python є Flask, легкий веб-фреймворк, який надає інструменти для створення веб-додатків. Flask добре інтегрується з різними бібліотеками автентифікації, такими як Flask-Login і Flask-Security, які пропонують такі функції, як керування користувачами, керування сеансами та хешування паролів. Ці бібліотеки спрощують реалізацію функцій автентифікації та авторизації.

Ще однією потужною бібліотекою Python є Django, надійна веб-платформа, яка включає вбудовані функції автентифікації та авторизації. Модуль автентифікації Django забезпечує безпечну автентифікацію користувача, керування паролями та обробку сеансу користувача. Він також підтримує різні методи автентифікації, включаючи ім’я користувача/пароль, вхід у соціальні мережі та автентифікацію на основі маркерів.

Python також пропонує бібліотеки для обробки веб-токенів JSON (JWT), наприклад PyJWT, що дозволяє розробникам безпечно кодувати та декодувати JWT. JWT зазвичай використовуються для автентифікації та авторизації, і Python надає необхідні інструменти для їх ефективної обробки.

Крім того, велика стандартна бібліотека Python включає модулі для криптографії, такі як hashlib і hmac, які можна використовувати для безпечного хешування паролів, шифрування та автентифікації повідомлень.

Іншою мовою програмування, яку варто розглянути для розробки програмного забезпечення для авторизації, є Java.

Java — це широко використовувана та надійна мова програмування, яка пропонує міцну основу для створення безпечних і масштабованих програм, включаючи системи автентифікації. Розгалужена екосистема та зрілі бібліотеки Java роблять її популярним вибором для розробки програмного забезпечення на рівні підприємства.

Java надає кілька фреймворків і бібліотек, які можуть полегшити реалізацію функцій автентифікації. Наприклад, Spring Security — це потужний і широко використовуваний фреймворк, який пропонує комплексні функції безпеки, включаючи модулі автентифікації та авторизації. Він забезпечує підтримку різних механізмів автентифікації, таких як ім'я користувача/пароль, OAuth і веб-токени JSON (JWT). Spring Security також добре інтегрується з іншими фреймворками Java, такими як Spring MVC і Spring Boot, що робить його придатним для створення веб-додатків.

Окрім Spring Security, Java пропонує такі бібліотеки, як Apache Shiro, які забезпечують гнучку та просту у використанні структуру безпеки для керування автентифікацією та авторизацією. Shiro підтримує кілька методів автентифікації, управління сеансами та контроль доступу.

Стандартна бібліотека Java також включає криптографічні API, які дозволяють розробникам впроваджувати безпечне хешування паролів, шифрування та цифрові підписи. Пакет javax.crypto надає класи та інтерфейси для шифрування та дешифрування, а пакет java.security пропонує функції для безпечної генерації ключів і керування ними.

Крім того, незалежність Java від платформи дозволяє запускати програми в різних операційних системах і середовищах. Ця гнучкість робить

Java придатною для розробки серверних компонентів, таких як сервери автентифікації або API, які можна інтегрувати в різні системи.

2.2 JavaScript та його фреймворки

Платформа веб-розробки - це абстракція, в якій програмне забезпечення, що забезпечує загальну функціональність, може бути вибірково змінено додатковим користувальницьким кодом. JavaScript Фреймворк - це фреймворк програми, написаний на JavaScript, де програмісти можуть маніпулювати функціями і використовувати їх для своєї зручності.

Один з найбільш потужних і ефективних фреймворків JavaScript, Angular - це фреймворк з відкритим вихідним кодом, який використовується для розробки односторінкового додатки (SPA). Він розширює HTML-код в додатку і інтерпретує атрибути для виконання прив'язки даних.

Характеристики

- Progressive Web Apps: Володіє можливостями сучасної веб-платформи для забезпечення роботи, подібної з додатком, з високою продуктивністю, в автономному режимі і установкою без яких-небудь дій.
- Робочий стіл: додатки, які встановлюються на робочий стіл Mac, Windows і Linux, можуть бути створені з використанням тих же методів Angular, що і веб, плюс можливість доступу до вбудованим API ОС.
- Генерація коду: Angular перетворює шаблони код, високо оптимізований для віртуальних машин JavaScript, що дає переваги рукописного коду.
- Поділ коду: З новим Component Router додатки angular завантажуються швидко, забезпечуючи автоматичне розділення коду.
- Шаблони: Створюйте подання користувальницького інтерфейсу з допомогою простого і потужного синтаксису шаблону.

- Angular CLI: інструменти командного рядка дозволяють швидко почати збірку, додати компоненти і тести, а потім миттєво розгорнути.
- Анімація: Створюйте високопродуктивні, складні хореографії і тимчасові рамки анімації з дуже невеликою кількістю коду за допомогою інтуїтивно зрозумілого API Angular.
- Доступність: Створюйте доступні програми з допомогою компонентів з підтримкою ARIA, керівництв розробника і вбудованою інфраструктури тестування.

Створений Facebook фреймворк React framework дуже швидко завоював популярність. Він використовується для розробки і експлуатації динамічного інтерфейсу веб-сторінок з високим вхідним трафіком. У ньому використовується віртуальний DOM, і, отже, інтеграція з будь-яким додатком стає більш простою.

Характеристики:

- Декларативний: Створює і динамічний інтерактивний користувальницький інтерфейс для веб-сайтів і мобільних додатків. Декларативні подання роблять код читабельним і простим у налагодженні.
- Віртуальний DOM: Для кожного об'єкта DOM існує відповідний "віртуальний об'єкт DOM". Він створює віртуальну копію вихідного DOM і є представленням об'єкта DOM.
- Обробка подій: В React створена повністю сумісна система подій об'єктної моделі W3C. Він також надає кросбраузерність інтерфейс для власного події.
- JSX: JSX - це синтаксис розмітки, який дуже нагадує HTML. JSX спрощує написання компонентів React, роблячи синтаксис майже ідентичним HTML, введеному у веб-сторінку.
- Продуктивність: React використовує односторонню прив'язку даних з архітектурою програми, званої Flux controls. ReactJS допомагає

оновлювати подання для користувача за допомогою Flux, керуючого робочим процесом додатки.

- React Native: React Native - це користувальницький рендерер для React; він використовує власні компоненти замість веб-компонентів, таких як React, в якості будівельних блоків.
- На основі компонентів: Все є компонентом веб-сторінки, розділений на невеликі компоненти для створення уявлення (або користувальницького інтерфейсу). Компоненти в ReactJS використовуються для визначення візуальних елементів і взаємодій у додатках.

Незважаючи на те, що цей фреймворк JavaScript був розроблений в 2016 році, він вже вийшов на ринок і довів свою цінність, пропонуючи безліч функцій. Режим подвійної інтеграції є однією з найбільш привабливих функцій для створення СПА-салону високого класу. Це надійна платформа для кроссплатформної розробки.

Характеристики:

- Vue.js Віртуальний DOM: використовує віртуальний DOM, клон основного елемента DOM.
- Прив'язка даних: Ця функція полегшує маніпулювання або присвоєння значень HTML-атрибутів.
- CSS-переходи та анімацію: в Vue є кілька методів для застосування переходу до елементів HTML, додавання, оновлення або видалення з DOM.
- Шаблон: Vue пропонує шаблони HTML, які пов'язують з DOM Vue.js дані примірника. Шаблони компілюються у віртуальні функції рендеринга DOM.
- Vue.js Складність: простіше з точки зору API і дизайну. Веб-розробник може створювати прості програми за один день.

Ember.js з'явився в 2015 році, і з тих пір він придбав популярність завдяки своїй широкій області застосування. Особливості Ember.js підтримує двосторонній прив'язку даних, що робить її надійною платформою для роботи зі складними користувацькими інтерфейсами. Популярні веб-сайти, такі як LinkedIn, Netflix і Nordstrom, використовують Ember.js для їх веб-сайтів.

Характеристики:

- Простота використання: Створюйте зручні веб-додатки, які прості в обслуговуванні.
- Особливості HTML і CSS: В основі програми лежать HTML і CSS моделі розробки.
- Инициализаторы примірники: Ember надає инициализаторы примірника для класів
- Ember.js Маршрути: пропонує маршрути, які є основними функціями, що використовуються для управління URL.
- Ember.js Налагодження: містить інструмент Ember Inspector для налагодження додатків Ember.
- Ember.js Створення шаблонів: використовує шаблони, які допомагають автоматично оновлювати модель при зміні вмісту додатків.

Meteor має кілька різних застосувань, що охоплюють значну частину розробки програмного забезпечення. Області застосування включають серверну розроблення, управління базою даних, бізнес-логіку і рендеринг інтерфейсу.

Характеристики:

- Meteor.js Екосистема розробки: являє собою изоморфную екосистему розробки з відкритим вихідним кодом (IDevE). Це полегшує створення веб-додатків реального часу з нуля, оскільки містить усі необхідні інтерфейсні і серверні компоненти..

- Ізоморфний код JavaScript: один і той же код може використовуватися у зовнішньому інтерфейсі, серверної частини, мобільних і веб-додатках. Це позбавляє розробників від необхідності встановлювати і налаштовувати різні менеджери модулів, бібліотеки, драйвери, API і багато іншого.
- Інтерфейсне рішення:Meteor пропонує інтерфейсну платформу розробки, Blaze.js , який пропонує деякі корисні функції. Він також інтегрується з популярними сучасними інтерфейсними фреймворками, такими Backbone.js для отримання кращих результатів.
- Meteor.js Інтеграція з базою даних: підтримує будь-яку базу даних з Node.js. Зазвичай використовується MongoDB.

Mithril - це клієнтський JavaScript фреймворк, який використовується в основному при розробці односторінкових додатків. Оскільки похідних функцій від базового класу не існує, реалізація фреймворку більш проста. Він невеликий (< 8 КБ gzip), швидкий і надає готові утиліти маршрутизації і XHR.

Характеристики:

- Розмір коду: Розмір Mithril становить близько 7,8 Кб в стислому вигляді і не залежить від інших бібліотек.
- Архітектура: Mithril не надає базових класів для розширення. При реалізації основних шаблонів MVC немає прихованих витрат на продуктивність.
- Small API: Mithril пропонує стислий API, і в результаті немає необхідності створювати функціональність для кожного сценарію.
- Декларативний: Mithril є декларативним і знижує складність коду.

Node.js - це серверна середовище виконання JavaScript, яка працює на різних платформах і має відкритий вихідний код. Платформа здатна керувати асинхронним введенням-виведенням з допомогою своєї архітектури,

керованої подіями. Він працює в середовищі виконання JavaScript і демонструє аналогічні властивості JAVA, такі як багатопоточність, упаковка і формування циклів.

Характеристики:

- Швидко: Бібліотека Node.js швидкий, коли справа доходить до виконання коду, оскільки побудований на движку JavaScript V8 Google Chrome.
- Асинхронний ввід-висновок, керований подіями: Всі API є асинхронними, що означає, що сервер не очікує повернення API з даними.
- Node.js Однопотоковий: поряд з циклированием подій слід однопотокового моделі.
- Node.js Висока масштабованість: використовує механізм подій, що дозволяє серверу відповідати неблокирующим чином, що робить його масштабованим.
- Відсутність буферизації: Коли справа доходить до завантаження аудіо - і відеофайлів, Node.js значно скорочує час обробки. Він не буферизує які-небудь дані, і додаток витягує їх порціями.
- Відкритий вихідний код: Будучи відкритим вихідним кодом, Node.js співтовариство запропонувало кілька дивовижних моделей, які можна використовувати для розширення можливостей Node.js додатка.

Polymer - це бібліотека JavaScript з відкритим вихідним кодом, розроблена Google, яка дозволяє створювати елементи веб-сайту, не стаючи занадто складною. Він також підтримує як односторонній, так і двосторонній прив'язку даних, що дає йому більш широку область застосування.

Характеристики:

- Полинаполнители: До складу Polymer входять полинаполнители для створення індивідуальних і багаторазових елементів.

- Можливість повторного використання: Polymer використовується для створення повторно використовуваних віджетів у веб-документах і додатках.
- Підходить для мобільних додатків: Polymer використовує Google material design для розробки мобільних додатків, забезпечуючи швидку і просту розробку мобільних додатків.
- Гнучкість: Користувальницькі елементи поширюються по мережі і дозволяють користувачам використовувати ці елементи за допомогою імпорту HTML.

Хоча фреймворк Aurelia і не так поширений, як колись, він корисний для розробки набагато більш надійних веб-сайтів. Цей фреймворк JS може розширювати HTML для кількох цілей, включаючи прив'язку даних. Його сучасна архітектура гарантує, що метою toll є одночасний переклад на стороні клієнта і на стороні сервера.

Характеристики:

- Компоненти: Компоненти є будівельними блоками фреймворку Aurelia і складаються з пар подання-модель JavaScript і уявлень HTML.
- Веб-стандарти: Це один із самих чистих сучасних фреймворків. Він повністю фокусується на веб-стандартах без непотрібних абстракцій.
- Розширюваний: платформа забезпечує простий спосіб інтеграції з іншими інструментами.
- Комерційна підтримка: Цей фреймворк пропонує комерційну та корпоративну підтримку.

Досить простий в освоєнні, Backbone використовується для створення односторінкових додатків. При розробці цього фреймворку використовується ідея про те, що всі серверні функції повинні виконуватися через API, що допомагає досягати складних функціональних можливостей з меншою кількістю коду.

Характеристики:

- Зручність: Для складання клієнтських веб-додатків надаються будівельні блоки, такі як моделі, подання, події, маршрутизатори і колекції.
- Проста у використанні бібліотека: Це проста бібліотека, яка допомагає розділити логіку бізнес - і користувальницького інтерфейсу.
- Безліч розширень: Backbone має безкоштовну бібліотеку з відкритим вихідним кодом і містить понад 100 доступних розширень.
- Залежності: Backbone має м'яку залежність від jQuery і жорстку залежність від Underscore.js .
- Організована розробка: Backbone дозволяє створювати клієнтські веб-додатки або мобільні додатки в добре структурованому та організованому форматі.

Коли справа доходить до веб-розробки, Javascript продовжує залишатися домінуючою мовою. Але всі різні фреймворки JavaScript мають своє власне застосування, а також переваги і недоліки. Так що майте це на увазі, коли думаєте про його виборі.

Уважно вивчіть вимоги вашого проекту, перш ніж вибирати фреймворк для вашого застосування, оскільки кожен фреймворк володіє унікальними функціями, які можуть вам знадобитися в процесі розробки. Крім функцій, також враховуйте криву навчання, складність, документацію щодо сумісності та підтримку спільноти.

2.3 Забезпечення захисту

Під час розробки веб-програми з функціями автентифікації та реєстрації в JavaScript забезпечення безпеки є надзвичайно важливим. JavaScript забезпечує різні заходи безпеки, які слід враховувати для захисту даних користувача та запобігання несанкціонованому доступу.

1) Безпечний зв'язок:

Реалізація безпечного зв'язку між клієнтом і сервером за допомогою HTTPS (HTTP Secure). Це шифрує дані, якими обмінюються браузер користувача та сервер, запобігаючи прослуховування та втручання.

2) Зберігання паролів:

Хешуйте та соліть паролі користувачів перед збереженням їх у базі даних. Використовуйте надійні алгоритми хешування, такі як bcrypt.js або scrypt, щоб захистити паролі від легкого злому. Salting додає додатковий рівень безпеки, вводячи унікальне значення для кожного пароля.

3) Автентифікація:

Впровадити надійний механізм автентифікації для перевірки ідентичності користувачів під час входу. Використовуйте методи автентифікації на основі сеансу або маркера. Автентифікація на основі сеансу передбачає створення та перевірку файлів cookie сеансу, а автентифікація на основі маркерів використовує маркери (наприклад, веб-токени JSON) для автентифікації запитів.

4) Перевірка введених даних:

Перевіряйте та дезінфікуйте введені користувачем дані як на стороні клієнта, так і на стороні сервера, щоб запобігти поширеним уразливостям безпеки, таким як впровадження SQL і атаки міжсайтових сценаріїв (XSS). Використовуйте такі бібліотеки, як DOMPurify, або запровадьте власну логіку перевірки введення.

5) Захист від підробки міжсайтових запитів (CSRF):

Впровадити заходи щодо запобігання атак CSRF. Створюйте та перевіряйте маркери CSRF для кожного автентифікованого сеансу користувача, щоб переконатися, що запити надходять із вашої програми, а не зі шкідливих джерел.

6) Обмеження тарифів і блокування облікових записів:

Впровадити механізми обмеження швидкості для захисту від атак грубою силою на облікові записи користувачів. Застосуйте блокування облікових записів після певної кількості невдалих спроб входу, щоб запобігти неавторизованому доступу.

7) Двофакторна автентифікація (2FA):

Розгляньте можливість застосування двофакторної автентифікації як додаткового рівня безпеки. Це може включати такі методи, як коди підтвердження на основі SMS, програми автентифікації або апаратні маркери.

8) Авторизація та контроль доступу:

Визначте належні правила контролю доступу, щоб обмежити доступ користувачів до певних ресурсів або дій. Впровадьте контроль доступу на основі ролей (RBAC) або інші механізми авторизації, щоб переконатися, що користувачі мають доступ лише до необхідних функцій.

9) Аудит безпеки та журналювання:

Реалізуйте механізми журналювання для реєстрації важливих подій безпеки та дій. Регулярно перевіряйте журнали на наявність будь-яких підозрілих дій і виконуйте аудит безпеки, щоб виявити потенційні вразливості.

10) Оновлюйте залежності:

Регулярно оновлюйте та виправляйте залежності та бібліотеки, які використовуються у вашій програмі, щоб усунути будь-які вразливості безпеки або слабкі місця, виявлені спільнотою.

2.4. Висновок до розділу 2

У цьому розділі було проведено огляд мов програмування для розробки програмного забезпечення для авторизації. Було вибрано мову JavaScript як оптимальний варіант для реалізації механізмів автентифікації.. JavaScript є популярною та універсальною мовою програмування, яка застосовується як на стороні клієнта, так і на стороні сервера. Вона підтримується всіма основними веб-браузерами і має багатий набір бібліотек і фреймворків, що полегшують розробку автентифікаційних механізмів. Також було проведено характеристику JavaScript фрейморків.

РОЗДІЛ 3. ТЕСТУВАННЯ ДОДАТКУ

3.1 Загальний процес тестування додатків

Для настільних додатків тестування повинно враховувати користувальницький інтерфейс, бізнес-логіку, бази даних, звіти, ролі і права, цілісність, зручність, функціональність, продуктивність, безпеку, сумісність обладнання та програмного забезпечення і потік даних.

Що стосується веб-додатків, тестувальники повинні надавати достатню значення продуктивності, завантаженні і безпеки застосування.

Іншими основними типами тестування, визначеними тестування веб-додатків, є функціональне тестування, кроссбраузерное тестування, UAT, бета-тестування, регрессионное тестування, тестування на сумісність, smoke-тестування, дослідницьке тестування, тестування сумісності і багатомовної підтримки та стрес-тестування.

Для мобільних додатків основними типами тестування, які слід виконувати, є тестування користувальницького інтерфейсу, тестування на основі правил, регрессионное тестування, функціональне тестування та тестування безпеки.

Отже, AUT (тестоване додаток) - це або настільне програмне забезпечення, веб-сайт, або мобільний додаток.

Добре відомий і обговорюваний аспект полягає в тому, що існує тільки 3 загальноприйняті методології тестування:

Чорний ящик#1) При тестуванні з використанням чорного ящика AUT перевіряється на відповідність його вимогам з урахуванням вхідних даних і очікуваних результатів, незалежно від того, як вхідні дані перетворюються у

вихідні дані. Тестувальники найменше стурбовані внутрішньою структурою або кодом, який реалізує бізнес-логіку програми.

Існує чотири основних методи розробки тестових прикладів для тестування "Чорного ящика":

- BVA (Аналіз граничних значень)
- EP (Поділ еквівалентності)
- Таблиці прийняття рішень
- Таблиці переходу станів (діаграми)

Тестування чорного ящика зазвичай використовується для функціонального, нефункціонального та регресійного тестування.

Білий ящик: Основною метою цієї методології є перевірка того, як бізнес-логіка програми реалізується кодом / програмою.

Тут тестується внутрішня структура програми, і для цього доступні наступні методи:

Покриття коду

Покриття шляху

Найбільш важливою концепцією у "Тестуванні програм" є функціональне тестування. Наша увага буде зосереджена на інструментах функціонального тестування.

Ось список деяких найбільш важливих і фундаментальних функцій, які надаються майже всіма інструментами "Функціонального тестування'.

Записуйте та відтворюйте

Параметризуйте Значення

Редактор сценаріїв

Запуск (тест або скрипт з режимами налагодження та оновлення)

Звіт про занедбаному сеансі

Різні постачальники надають специфічні функції, які роблять їх продукт унікальним у порівнянні з іншими продуктами конкурентів. Але

п'ять перерахованих вище функцій є найбільш поширеними і можуть бути знайдені практично у всіх інструментах функціонального тестування.

Планування завжди потрібно для будь-якої діяльності, і те ж саме стосується і тестування програмного забезпечення. Без належного плану завжди існує високий ризик відволіктися під час тестування. Якщо цей ризик стане фактом, то результати можуть бути жахливими.

5 Основних частин Хорошого Плану тестування:

#1) Сфера застосування

Огляд AUT

Функції (або галузі), що підлягають тестуванню.

Виключення (функції або області, не підлягають тестуванню) з відповідним обґрунтуванням.

Залежності (дій з тестування один від одного, якщо такі є).

#2) Цілі: В цьому розділі описуються цілі тестування, наприклад, перевірка виправлень помилок, додавання нових функцій або оновлення AUT і т. д.

#3) Фокус: В цьому розділі описується, який аспект програми буде включено до тестування, наприклад безпеку, функціональність, зручність використання, надійність, продуктивність або дієвість і т. д.

#4) Підхід: В цьому розділі описується, яка методологія тестування буде прийнята для яких областей AUT. Наприклад, у STP додатки ERP; розділ підходу може містити інформацію про те, що тестування чорного ящика буде підходом для розрахунку заробітної плати. Однак для звітів підхід буде полягати в тестуванні "сірого ящика".

5) Розклад: В цьому розділі описується, хто, що, де, коли і як буде робити на AUT. Розділ розкладу являє собою '4Ws 1H' STP. Зазвичай розклад складається у вигляді простої таблиці, але кожна організація може мати свій власний індивідуальний формат у відповідності зі своїми потребами.

3.2 Тестування готового програмного забезпечення

Щоб отримати доступ до програми потрібно зареєструватися або авторизуватися (Рис 3.1-Рис 3.2)

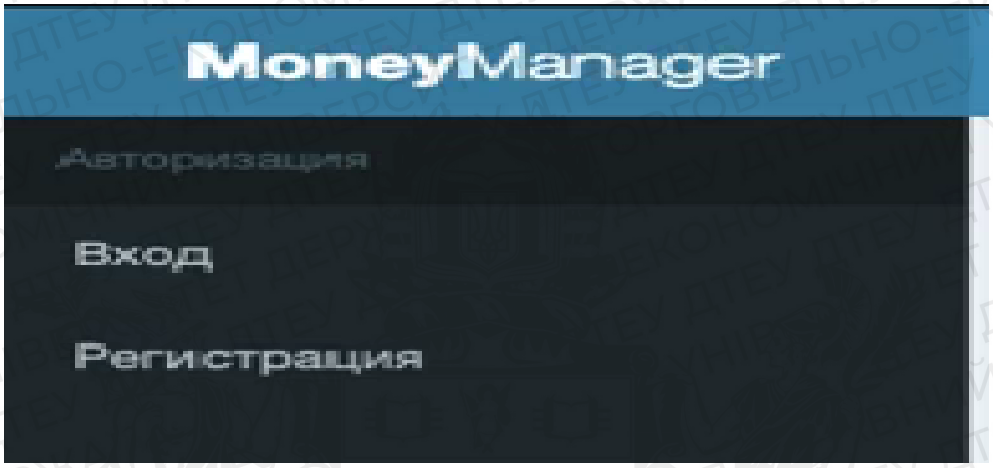

Рисунок 3.1-Вибір в програмі

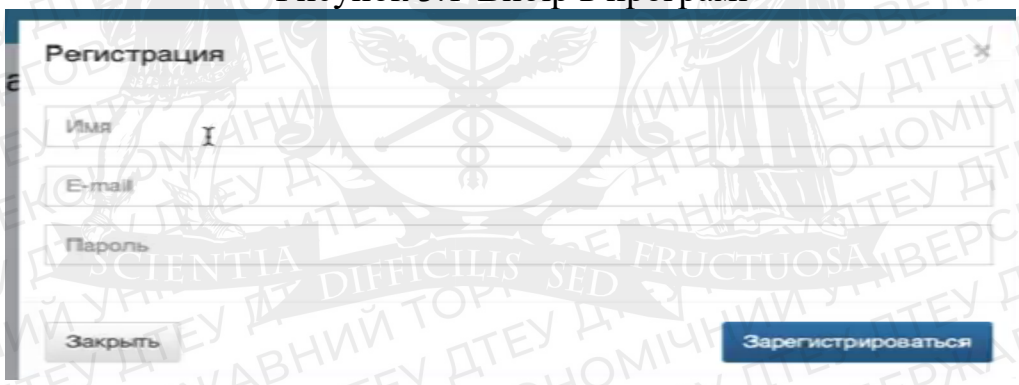

Рисунок 3.2-Реєстрація в програмі

Після того як авторизуємося в клієнті переходимо на головну сторінку програми

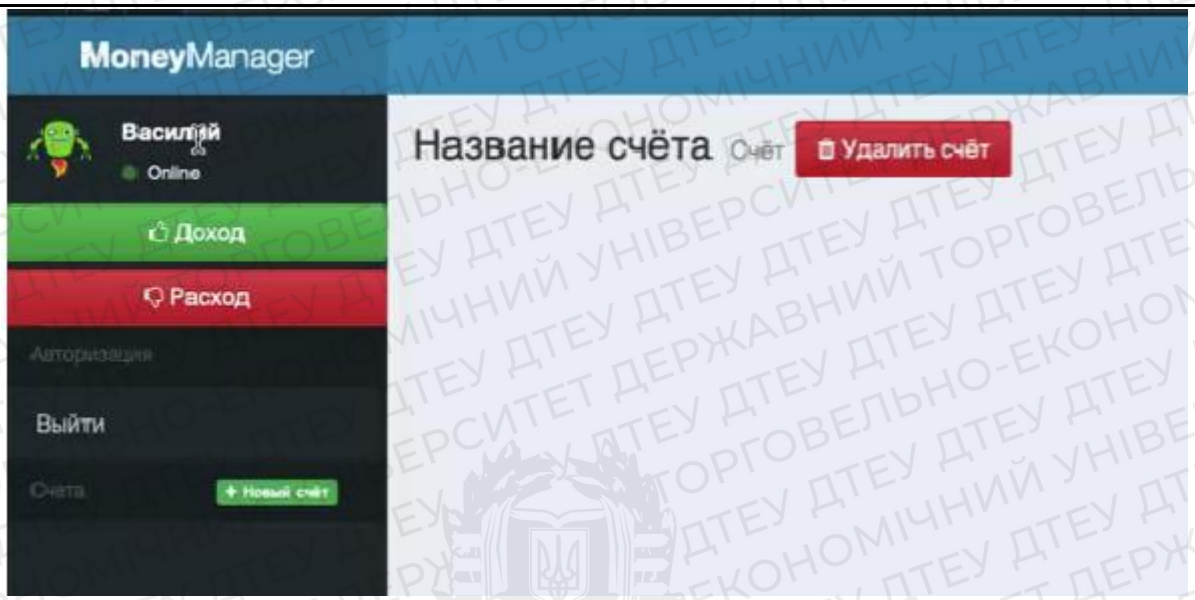

Рисунок 3.3-Головна сторінка додатку

Також для перевірки функцій додатку було написано тест кейси фрагменти якого вкажимо нище:

Тест 1:

```
describe("Функция createRequest", function() {
 const open = XMLHttpRequest.prototype.open,
  send = XMLHttpRequest.prototype.send;
 XMLHttpRequest.prototype.open = function( method, url, async ) {
  this.requestMethod = method;
  this.requestURL = url;
  open.call( this, method, url, async );
 };

 XMLHttpRequest.prototype.send = function( data ) {
  this.data = data;
  send.call( this, data );
 };

 it("Определена", function() {
  expect(createRequest).to.be.a('function');
```

```
it("Создаёт объект XMLHttpRequest", function() {
  const xhr = createRequest();
  expect(xhr).to.an.instanceof(XMLHttpRequest);
});

it("Передаёт параметр responseType", function() {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      responseType: 'json',
      method: 'POST'
    });
  expect(xhr.responseType).to.equal('json');
});

it("Передаёт параметр url", function() {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      method: 'POST'
    });

  expect(xhr.requestURL).to.equal(url);
});

it("Передаёт параметр method", function() {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
```

```
    xhr = createRequest({
      url,
      method: 'POST'
    });
    expect(xhr.requestMethod).to.equal('POST');
  });

  it("Вызывает callback и передаёт данные при успешном запросе", function( done ) {
    const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      responseType: 'json',
      method: 'GET',
      callback: (err, data) => {
        expect(data).to.be.a('object');
        done();
      }
    });
  });

  it("Передаёт FormData из свойства data (для методов кроме GET)", function() {
    const dummy = f => f,
      url = 'https://jsonplaceholder.typicode.com/todos/1',
      data =  {
        hello: 'world',
        iron: 'maiden'
```

```
    },
    xhr = createRequest({
     url,
     method: 'POST',
     data
    }),
    sentData = [...xhr.data].reduce(( target, [ key, value ]) => {
     target[ key ] = value;
     return target;
    }, {});
   expect(data).to.eql( sentData );
  });
});
```

Наданий код являє собою набір тестів, написаний на платформі Mocha framework, який перевіряє функціональність createRequest функції. Ось розбивка коду:

describeФункція використовується для визначення набору тестів під назвою "Функція createRequest". У ньому згруповано серія пов'язаних тестів.

XMLHttpRequest.prototype.openЦе досягаєтьсяmethod параметри, передані йому. urli Функція перезаписатиthis.requestMethod і this.requestURL, відповідно. Потім викликається вихідна open функція.

Дані зберігаються в `цьомуXMLHttpRequest.prototype.senddata параметр, переданий йому. Аналогічним чином, this.datai викликається вихідна send функція.

Bit тестовий приклад перевіряє, чи визначена createRequest функція і має вона тип 'function'.

функціяit тестовий приклад перевіряє, чи createRequestДругий XMLHttpRequest коли дзвонили

примірникit параметр правильно переданий bresponseType XMLHttpRequest Третій createRequest.

екземпляр, створенийit параметр переданий правильнourl в XMLHttpRequest Четвертий createRequest.

параметр переданий правильноit тестовий приклад перевіряє, чи method П'ятий XMLHttpRequest екземпляр, створений createRequest.

Шостий it тестовий приклад перевіряє асинхронне поведінку createRequest функції з допомогою done зворотного виклику.

об'єкт передається як правильноit тестовий приклад перевіряє, чи FormDataСьомий method це не "ОТРИМАТИ".

Кожен тестовий приклад перевіряє певний аспект createRequest функції, такої як передача параметрів і зворотний виклик, використовуючи твердження, що надаються бібліотекою Chai. Тести гарантують, що createRequest функція веде себе так, як очікувалося, і відповідає зазначеним вимогам.

Тест 2:

```
describe("Класс Entity", function() {

 it("Определён", function() {
  expect(Entity).to.be.a('function');
 });

 it("Создаёт экземпляр Entity", function() {
  expect(new Entity).to.be.an.instanceof(Entity);
 });

 it('Задано свойство URL', () => {
  expect(Entity.URL).to.be.a('string');
```

```
});

describe('Метод Entity.list()', () => {

 it('Возвращает XHR', () => {
  const xhr = Entity.list();

  expect(xhr).to.be.an.instanceof(XMLHttpRequest);
 });

 it('Создаёт GET-запрос', () => {
  const xhr = Entity.list();

  expect(xhr.requestMethod).to.be.equals( 'GET' );
 });

 it('Обращается по адресу, определённому в свойстве URL', () => {
  const xhr = Entity.list();

  expect(xhr.requestURL).to.be.equals( Entity.URL );
 });
});

describe('Метод Entity.create()', () => {

 it('Возвращает XHR', () => {
  const xhr = Entity.create();
```

```
    expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт POST-запрос', () => {
    const xhr = Entity.create();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = Entity.create();

    expect(xhr.requestURL).to.be.equals( Entity.URL );
  });
});

describe('Метод Entity.remove()', () => {

  it('Возвращает XHR', () => {
    const xhr = Entity.remove();

    expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт DELETE-запрос', () => {
    const xhr = Entity.remove();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
```

```
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = Entity.remove();

    expect(xhr.requestURL).to.be.equals( Entity.URL  + '/' );
  });
 });
});
```

Наданий код являє собою набір тестів, написаний на платформі Mocha framework, який перевіряє функціональність Entity класу

describeФункція використовується для визначення набору тестів під назвою "Клас Entity". У ньому згруповано серія пов'язаних тестів.

тестовий приклад перевіряє, чиітПерший Entity клас визначений і має тип 'function'.

повертає ключове словоітвикористовуючи Entity тестовий приклад перевіряє, чи створюєnewДругий Entity клас.

властивістьit тестовий приклад перевіряє, чи URLТретій Entity клас визначений і має тип 'string'.

блок для об'єкта методу `describe ВсерединіEntity.list()є ще три тестових прикладу.

Перший тестовий приклад перевіряє, Entity.list() чи повертає виклик примірник XMLHttpRequest.

Другий тестовий прикладXMLHttpRequest примірник - це 'GET'.

примірник дорівнює значеннюXMLHttpRequest ТретійURL властивість, визначене у Entity класі.

примірник, встановлює метод запитуdescribe повертає Entity.create()Ці тестові приклади гарантують, що Entity.list() опишіть блок. існує три тестових прикладу , аналогічних наведеним у Entity.create()XMLHttpRequestBсередині URL властивість, визначене у Entity класі.

наприклад, встановлюєdescribeповертає Entity.remove() Ці тестові приклади гарантуютьвиписуйте блоки. Entity.remove() блок для об'єкта методу `XMLHttpRequestBсередині URL властивість і '/'.

клас і його методи, такі якEntity КоженEntity клас веде себе так, як очікувалося, і відповідає зазначеним вимогам.

Важливо відзначити, що наданий код є набором тестів і не включає в себе реалізацію самого Entity класу.

Перевіривши графічну частину та провівши тестування через код відповідно до результатів робимо висновок що вийшов працюючий додаток та він виконує всі необхідні функції котрі задумав автор.

3.3. Висновок до розділу 3

В даному розділі описано загальний процес тестування додатків, зокрема для настільних, веб та мобільних додатків. Для настільних додатків тестування враховує такі аспекти, як користувальницький інтерфейс, бізнес-логіка, бази даних, звіти, ролі і права, цілісність, зручність, функціональність, продуктивність, безпека, сумісність обладнання та програмного забезпечення. Тестування веб-додатків включає попередні пункти плюс продуктивність та безпека застосування.

В розділі було також наведено різні типи тестування, такі як функціональне тестування, кроссбраузерне тестування, бета-тестування, регресійне тестування, тестування на сумісність, smoke-тестування, дослідницьке тестування, тестування сумісності і багатомовної підтримки, а також стрес-тестування.

Загальноприйнятими методологіями тестування є "Чорний ящик" та "Білий ящик".

При "Чорному ящику" тестування проводиться з фокусом на зовнішньому поведінці системи, без знання внутрішньої реалізації та структури коду. Тестувальник перевіряє відповідність вимогам, використовуючи вхідні дані та очікувані результати.

При "Білому ящику" тестування зосереджується на внутрішній структурі та реалізації системи. Тестувальник має знання про код та використовує його для розробки тестових сценаріїв та валідації логіки програми.

Отже, розділ 3 надає загальний огляд процесу тестування додатків і виділяє ключові аспекти і методи, які використовуються при тестуванні настільних, веб- та мобільних додатків

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Розробка програмного забезпечення для авторизації вимагає вибору мови програмування, здатного ефективно обробляти протоколи аутентифікації користувача та безпеки. У цьому випадку JavaScript буде виділеним мовою для виконання завдання. JavaScript - популярний і універсальний мову програмування, широко використовується для веб-розробки завдяки своїй здатності запускатися як на стороні клієнта, так і на стороні сервера. Він надає багатий набір бібліотек і фреймворків, які роблять його придатним для реалізації механізмів аутентифікації.

JavaScript - це високорівненвий інтерпретується в microsoft мова програмування, який дозволяє розробникам створювати інтерактивні веб-сторінки і динамічні веб-додатки. Він підтримується всіма основними веб-браузерами, що робить його надійним вибором для створення кроссплатформенних додатків. Універсальність JavaScript виходить за рамки веб-браузера, оскільки його можна використовувати для розробки на стороні сервера за допомогою таких фреймворків, як Node.js .

З швидким зростанням мережевих систем і додатків, таких як електронна комерція, зростає попит на ефективну комп'ютерну безпеку. Більшість комп'ютерних систем захищені процесу ідентифікації користувача і аутентифікації. У той час як ідентифікація зазвичай являє собою непублічну інформацію, надану користувачами для самоідентифікації, і може бути відома системним адміністраторам та іншим користувачам системи, аутентифікація надає секретну, приватну інформацію, яка може підтвердити його особу. Існують різні підходи і техніки аутентифікації, від паролів до відкритих ключів.

Ще до появи комп'ютерів для аутентифікації людей використовувалися різні відмінні характеристики. Комп'ютерні системи застосували ці характеристики для аутентифікації користувача. Підходи до аутентифікації можна розділити на три типи відповідно до використовуваними ними відмінними характеристиками (Menkus, 1988), як показано на малюнку 1.1:

• Що знає користувач— аутентифікація на основі знань (наприклад, пароль, PIN-код, пароль доступу)

• Що є у користувача —аутентифікація на основі володіння (наприклад, карта пам'яті і токени смарт-карти)

• Хто такий користувач —аутентифікація на основі біометрії: фізіологічні (наприклад, відбиток пальця) або поведінкові (наприклад, динаміка клавіатури) характеристики.

В ході проведення роботи було досліджено тему розроблено та протестовано додаток для авторизації в системі.

Пропозиції:

- Додати мультиплатформеність щоб додаток міг працювати на різних платформах, включаючи веб, мобільні пристрої і настільні комп'ютери, для забезпечення універсального доступу та використання.
- Додати управління правами доступу що дозволить точно налаштовувати права користувачів і контролювати їх доступ до різних ресурсів системи на основі їх ролей та привілеїв.
- Додати використання двофакторної аутентифікації: Додавання підтримки для двофакторної аутентифікації, такої як одноразові паролі або аутентифікація на основі біометричних даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1) "JavaScript: The Good Parts" by Douglas Crockford (2008)

2) "Eloquent JavaScript: A Modern Introduction to Programming" by Marijn Haverbeke (2018)

3) "JavaScript: The Definitive Guide" by David Flanagan (2020)

4) "Secure Coding in JavaScript and HTML: Protecting Web Sites and Applications" by David Johansen (2018)

5) "JavaScript Patterns" by Stoyan Stefanov (2010)

6) "Web Application Security: A Beginner's Guide" by Bryan Sullivan (2016)

7) "Professional JavaScript for Web Developers" by Nicholas (2019)

8) "JavaScript: Understanding the Weird Parts" by Anthony Alicea (2015)

9) "Secure JavaScript: A Guide to Web Application Security" by Daniel Santos (2012)

10) "JavaScript Security" by Y.E. Liang (2013)

11) "JavaScript Testing with Jasmine: JavaScript Behavior-Driven Development" by Evan Hahn (2014)

12) "The Principles of Object-Oriented JavaScript" by Nicholas C. Zakas (2007)

13) "Securing Ajax Applications: Ensuring the Safety of the Dynamic Web" by Christopher Wells (2014)

14) "Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications" by Ivan Ristic (2019)

15) "OAuth 2.0: Getting Started in Web-API Security" by Ryan Boyd (2019)

ДОДАТКИ

ДОДАТОК А

Лістинг додатку

```
"use strict";
/**
 * Класс User управляет авторизацией, выходом и
 * регистрацией пользователя из приложения
 * Имеет свойство URL, равное '/user'.
* */
class User {
 static URL = '/user';
 /**
  * Устанавливает текущего пользователя в
  * локальном хранилище.
  * */
 static setCurrent(user) {
  localStorage.user = JSON.stringify(user);
 }

 /**
  * Удаляет информацию об авторизованном
  * пользователе из локального хранилища.
  * */
 static unsetCurrent() {
  localStorage.removeItem("user");
 }

 /**
  * Возвращает текущего авторизованного пользователя
  * из локального хранилища
```

```
 * */
static current() {
 return JSON.parse(localStorage.getItem("user"));
}

/**
 * Получает информацию о текущем
 * авторизованном пользователе.
 * */
static fetch(data, callback = f => f) {
 const options = {
  url: this.URL + "/current",
  data: data,
  responseType: 'json',
  method: 'GET',
  callback: (err, response) => {
   if (response && response.user) {
    User.setCurrent(response.user);
   } else {
    User.unsetCurrent();
   }
   callback(err, response);
  }
 };
 return createRequest(options);
}

/**
 * Производит попытку авторизации.
 * После успешной авторизации необходимо
```

```
 * сохранить пользователя через метод
 * User.setCurrent.
 * */
static login(data, callback = f => f) {
 const options = {
  url: this.URL + "/login",
  data: data,
  responseType: 'json',
  method: 'POST',
  callback: (err, response) => {
   if (response && response.user) {
    User.setCurrent(response.user);
   }
   callback(err, response);
  }
 };
 return createRequest(options);
}

/**
 * Производит попытку регистрации пользователя.
 * После успешной авторизации необходимо
 * сохранить пользователя через метод
 * User.setCurrent.
 * */
static register(data, callback = f => f) {
 const options = {
  url: this.URL + '/register',
  data: data,
  responseType: 'json',
```

```
    method: 'POST',
    callback: (err, response) => {
      if (response && response.user) {
        User.setCurrent(response.user);
      }
      callback(err, response);
    }
  };
  return createRequest(options);
}

/**
 * Производит выход из приложения. После успешного
 * выхода необходимо вызвать метод User.unsetCurrent
 * */
static logout(data, callback = f => f) {
  const options = {
    url: this.URL + '/logout',
    data: data,
    responseType: 'json',
    method: 'POST',
    callback: (err, response) => {
      if (response && response.user) {
        User.unsetCurrent(response.user);
      }
      callback(err, response);
    }
  };
  return createRequest(options);
}
```

```
} "use strict";
/**
 * Класс Entity - базовый для взаимодействия с сервером.
 * Имеет свойство URL, равно пустой строке.
 * */
class Entity {
  static URL = "";
  /**
   * Запрашивает с сервера список данных.
   * Это могут быть счета или доходы/расходы
   * (в зависимости от того, что наследуется от Entity)
   * */
  static list(data, callback = f => f) {
    const options = {
      url: this.URL,
      method: 'GET',
      data: data,
      responseType: 'json',
      callback: callback,
    };
    return createRequest(options);
  }

  /**
   * Создаёт счёт или доход/расход с помощью запроса
   * на сервер. (в зависимости от того,
   * что наследуется от Entity)
   * */
  static create(data, callback = f => f) {
    const objectData = Object.assign({_method: 'PUT'}, data);
```

```
  const options = {
    url: this.URL,
    data: objectData,
    responseType: 'json',
    method: 'POST',
    callback: callback,
  };
  return createRequest(options);
}

/**
 * Получает информацию о счёте или доходе/расходе
 * (в зависимости от того, что наследуется от Entity)
 * */
static get(id = '', data, callback = f => f) {
  const options = {
    url: this.URL + "/" + id,
    data: data,
    responseType: 'json',
    method: 'GET',
    callback: callback,
  };
  return createRequest(options);
}

/**
 * Удаляет информацию о счёте или доходе/расходе
 * (в зависимости от того, что наследуется от Entity)
 * */
static remove(id = '', data, callback = f => f) {
```

```
    const objectData = Object.assign({id, _method: 'DELETE'}, data);
    const options = {
     url: this.URL + "/",
     data: objectData,
     responseType: 'json',
     method: "POST",
     callback: callback,
    };
    return createRequest(options);
  }
} "use strict";
/**
 * Основная функция для совершения запросов
 * на сервер.
 * */
const createRequest = (options = {}) => {
    let xhr = new XMLHttpRequest();
    let formData = new FormData();
    xhr.responseType = options.responseType;
    if ('headers' in options) {
      for (let header in options.headers) {
        xhr.setRequestHeader(header, options.headers[header]);
      }
    }
    if (options.method === "GET") {
      options.url += "?";
      for (let param in options.data) {
        options.url += param + "=" + options.data[param] + "&";
        formData.append(param, options.data[param]);
      }
```

```
    options.url = options.url.slice(0, -1);
  } else {
    for (let param in options.data) {
      formData.append(param, options.data[param]);
    }
  xhr.withCredentials = true;
  try {
    xhr.addEventListener("readystatechange", function () {
      if (this.readyState === 4 && this.status === 200) {
        options.callback(null, xhr.response);
      }
    });
  } catch (err) {
    options.callback(err);
  }
  xhr.open(options.method, options.url, true);

  try {
    if (options.method == "GET") {
      xhr.send();
    } else {
      xhr.send(formData);
    }

    return xhr;
  } catch (err) {
    options.callback(err);
  }
};
```

ДОДАТОК Б

Лістинг тестів

```
describe("Класс User", function() {

 it("Определён", function() {
  expect(User).to.be.a('function');
 });

 it("Создаёт экземпляр User", function() {
  expect(new User).to.be.an.instanceof(User);
 });

 it('Свойство URL имеет значение /user', () => {
  expect(User.URL).to.be.equals( '/user' );
 });

 describe('Метод User.login()', () => {

  it('Возвращает XHR', () => {
   const xhr = User.login();

   expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт POST-запрос', () => {
   const xhr = User.login();

   expect(xhr.requestMethod).to.be.equals( 'POST' );
  });
```

```
  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = User.login();

    expect(xhr.requestURL).to.be.equals(  User.URL + '/login' );
  });
});

describe('Метод User.register()', () => {

  it('Возвращает XHR', () => {
    const xhr = User.register();

    expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт POST-запрос', () => {
    const xhr = User.register();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = User.register();

    expect(xhr.requestURL).to.be.equals( User.URL + '/register' );
  });
});

describe('Метод User.logout()', () => {
```

```
  it('Возвращает XHR', () => {
    const xhr = User.logout();

    expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт POST-запрос', () => {
    const xhr = User.logout();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = User.logout();

    expect(xhr.requestURL).to.be.equals( User.URL + '/logout' );
  });
 });
}); describe("Класс Entity", function() {

 it("Определён", function() {
  expect(Entity).to.be.a('function');
 });

 it("Создаёт экземпляр Entity", function() {
  expect(new Entity).to.be.an.instanceof(Entity);
 });

 it('Задано свойство URL', () => {
  expect(Entity.URL).to.be.a('string');
```

```
});

describe('Метод Entity.list()', () => {

 it('Возвращает XHR', () => {
  const xhr = Entity.list();

  expect(xhr).to.be.an.instanceof(XMLHttpRequest);
 });

 it('Создаёт GET-запрос', () => {
  const xhr = Entity.list();

  expect(xhr.requestMethod).to.be.equals( 'GET' );
 });

 it('Обращается по адресу, определённому в свойстве URL', () => {
  const xhr = Entity.list();

  expect(xhr.requestURL).to.be.equals( Entity.URL );
 });
});

describe('Метод Entity.create()', () => {

 it('Возвращает XHR', () => {
  const xhr = Entity.create();

  expect(xhr).to.be.an.instanceof(XMLHttpRequest);
 });
```

```
  it('Создаёт POST-запрос', () => {
    const xhr = Entity.create();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = Entity.create();

    expect(xhr.requestURL).to.be.equals( Entity.URL );
  });
});

describe('Метод Entity.remove()', () => {

  it('Возвращает XHR', () => {
    const xhr = Entity.remove();

    expect(xhr).to.be.an.instanceof(XMLHttpRequest);
  });

  it('Создаёт DELETE-запрос', () => {
    const xhr = Entity.remove();

    expect(xhr.requestMethod).to.be.equals( 'POST' );
  });

  it('Обращается по адресу, определённому в свойстве URL', () => {
    const xhr = Entity.remove();
```

```
    expect(xhr.requestURL).to.be.equals( Entity.URL  + '/' );
   });
  });
}); describe("Функция createRequest", function() {
  const open = XMLHttpRequest.prototype.open,
   send = XMLHttpRequest.prototype.send;
  XMLHttpRequest.prototype.open = function( method, url, async ) {
   this.requestMethod = method;
   this.requestURL = url;
   open.call( this, method, url, async );
  };

  XMLHttpRequest.prototype.send = function( data ) {
   this.data = data;
   send.call( this, data );
  };

  it("Определена", function() {
   expect(createRequest).to.be.a('function');
  });
  it("Создаёт объект XMLHttpRequest", function() {
   const xhr = createRequest();
   expect(xhr).to.an.instanceof(XMLHttpRequest);
  });

  it("Передаёт параметр responseType", function() {
   const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
     url,
```

```
      responseType: 'json',
      method: 'POST'
    });
  expect(xhr.responseType).to.equal('json');
});

it("Передаёт параметр url", function() {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      method: 'POST'
    });
  expect(xhr.requestURL).to.equal(url);
});
it("Передаёт параметр method", function() {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      method: 'POST'
    });
  expect(xhr.requestMethod).to.equal('POST');
});

it("Вызывает callback и передаёт данные при успешном запросе",
function( done ) {
  const url = 'https://jsonplaceholder.typicode.com/todos/1',
    xhr = createRequest({
      url,
      responseType: 'json',
      method: 'GET',
```

```
      callback: (err, data) => {
       expect(data).to.be.a('object');
       done();
      }
     });
    });
    it("Передаёт FormData из свойства data (для методов кроме GET)",
function() {
     const dummy = f => f,
      url = 'https://jsonplaceholder.typicode.com/todos/1',
      data =  {
       hello: 'world',
       iron: 'maiden'
      },
      xhr = createRequest({
       url,
       method: 'POST',
       data
      }),
      sentData = [...xhr.data].reduce(( target, [ key, value ]) => {
       target[ key ] = value;
       return target;
      }, {});
     expect(data).to.eql( sentData );
    });
   });
```