ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web - застосунку для управління особистими фінансами з використанням Python, HTML та CSS»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро РОМАШКАН
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Дмитро РОМАШКАН

Керівник: Ігор АВЕРІЧЕВ
к.е.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
«_____»_____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ромашкану Дмитру Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка WEB - застосунку для управління особистими фінансами з використанням Python, HTML та CSS»

керівник кваліфікаційної роботи к.е.н., доцент кафедри ІПЗ Ігор АВЕРІЧЕВ

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. №36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, вимоги до програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі.
2. Вибір технології та інструментів для розробки проєкту.
3. Огляд та розробка WEB - застосунку.
4. Тестування WEB - застосунку.

5. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Діаграма робочого процесу.
    3. Діаграма прецедентів.
    4. Діаграма класів.
    5. Діаграма предметної галузі.
    6. Апробація результатів досліджень.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	28.02-07.03.24	
2	Аналіз існуючих застосунків для управління особистими фінансами	07.03-13.03.24	
3	Дослідження функціональних та нефункціональних вимог до застосунку	14.03-19.03.24	
4	Реалізація програмного застосунку	20.03-15.04.24	
6	Тестування програмного застосунку	16.04-19.04.24	
7	Оформлення роботи: вступ, висновки, реферат	20.04-03.05.24	
8	Розробка демонстраційних матеріалів	04.05-12.05.24	
9	Попередній захист роботи	13.05-31.05.24	

Здобувач вищої освіти _____ Дмитро РОМАШКАН
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор АВЕРІЧЕВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 52 стор., 3 табл., 17 рис., 17 джерел.

Мета роботи – поліпщення процесу управління особистими фінансами, за допомогою Python, HTML та CSS.

Об'єкт дослідження – процес управління особистих витрат користувача.

Предмет дослідження – WEB - застосунок для управління особистими фінансами та витратами.

Короткий зміст роботи: У роботі проведено аналіз існуючих застосунків для управління особистими фінансами. Розроблено функціональні та нефункціональні вимоги до застосунку. Спроектовано інтерфейс користувача. Розроблений веб-додаток для управління фінансами демонструє значний потенціал для допомоги користувачам у плануванні та контролі їх фінансів. Проведено модульне та ручне тестування застосунку.

КЛЮЧОВІ СЛОВА : УПРАВЛІННЯ ФІНАНСАМИ,ЗАСТОСУНОК МОВОЮ PYTHON,ТЕСТУВАННЯ ЗАСТОСУНКУ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД	-	База Даних
AMQP	-	Advanced Message Queuing Protocol
REST	-	Representational State Transfer
UI	-	Інтерфейс користувача
NUI	-	Природний інтерфейс користувача
GUI	-	Графічний інтерфейс користувача
EIP	-	Шаблони корпоративної інтеграції
VDI	-	Мережеве налаштування
TUI	-	Відчутний інтерфейс користувача

ВСТУП

Розвиток цифрових технологій та все більша доступність інтернету роблять розробку web-застосунків для управління фінансами актуальним та своєчасним завданням. На сучасному ринку існує чимало програмних рішень для управління особистими фінансами, кожна з яких адаптована до потреб конкретних користувачів із різними цілями.

У цьому контексті був розроблений web-застосунок для управління особистими фінансами. Використовуючи сучасні технології програмування та веб-розробки, цей веб-застосунок надає користувачам зручні інструменти для створення, редагування, видалення та відстеження, пропонуючи простий у використанні інструмент для контролю своїх доходів та витрат. Завдяки використанню мови програмування Python та фреймворку Django, додаток забезпечує надійність, швидкість та зручність в користуванні.

Основні завдання цього дипломного проекту включають:

Відстеження доходів і витрат. Web-застосунок забезпечує користувачам можливість детально контролювати їхні фінансові операції, включно із категоріями, датами та сумами, що сприятиме кращому розумінню їхньої фінансової картини і обґрунтованому управлінню бюджетом.

Бюджетування. Web-застосунок дозволить користувачам планувати та контролювати свій бюджет, встановлювати фінансові цілі та відслідковувати їх досягнення, отримуючи персоналізовані поради для ефективного управління коштами.

Аналіз фінансової ситуації. Web-застосунок надасть зручні інструменти для аналізу фінансового стану, такі як графіки, діаграми та звіти, що допоможуть користувачам зосередитися на важливих аспектах їх фінансового управління.

Нагадування та оповіщення. Web-застосунок буде включати функцію нагадувань про регулярні платежі, наближення термінів сплати та досягнення

фінансових цілей, що допоможе користувачам підтримувати фінансову дисципліну.

В результаті розробки веб-застосунку користувачі отримають інструмент, що забезпечує їм більшу свободу та незалежність у фінансових питаннях, сприяючи усвідомленому споживанню та оптимізації ресурсів. Під час розробки використані передові веб-технології та фреймворки для забезпечення високої надійності, безпеки та користувацької зручності додатку.

Розроблений додаток підвищить рівень фінансової грамотності та свідомості серед користувачів, сприяючи покращенню їх фінансового благополуччя і якості життя.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Поняття Web-застосунку

Мільйони компаній обмінюються інформацією в Інтернеті та взаємодіють зі своєю цільовою аудиторією. Це допомагає їм здійснювати швидкі та безпечні транзакції через Інтернет. Однак бізнес-цілей можна досягти, якщо підприємства зможуть зберігати всі ці дані для представлення якісних результатів кінцевим користувачам.

В індустрії розробки веб-додаток більше схожий на програму, яка використовує веб-браузер для зберігання та пошуку інформації для представлення інформації користувачам. Це дозволяє користувачеві взаємодіяти з компанією за допомогою онлайн-форм, електронних кошиків для покупок, CMS.

Наведемо приклади веб-додатків: онлайн-банкінг, онлайн-опитування, онлайн-форуми, онлайн-бронювання, кошик для покупок та інтерактивні ігри. Вивчати веб-розробку – це все одно, що мати занадто багато речей.

Розглянемо визначення веб-додаток. Це клієнт-серверна програма, де є браузер (клієнт) і веб-сервер. Логіка веб-додатку розподілена між сервером і клієнтом, є канал для обміну інформацією і сховище даних, розташоване локально або в хмарі. Веб-додатки з’явилися на етапі еволюції веб-сайтів і, дійсно, мають багато спільного. Факторами, які відрізняють веб-сайт від веб-додатку, є інтерактивність, інтеграція та автентифікація.

Сучасні веб-програми все ще використовують концепцію 3-рівневої архітектури, яка розділяє програми на рівень презентації, рівень програми та рівень даних.

Веб-програма (веб-програма) — це прикладна програма, яка зберігається на віддаленому сервері та доставляється через Інтернет через інтерфейс браузера. Веб-служби є веб-програмами за визначенням, і багато, хоча не всі, веб-сайти

містять веб-програми. Розробники розробляють веб-програми для широкого спектру використання та користувачів, від організації до окремої людини з багатьох причин. Веб-програми, які часто використовуються, включають веб-пошту, онлайн-калькулятори або магазини електронної комерції. Хоча користувачі можуть отримати доступ лише до деяких веб-програм за допомогою певного браузера, більшість із них доступні незалежно від браузера.

Веб-програми не потрібно завантажувати, оскільки доступ до них здійснюється через мережу. Користувачі можуть отримати доступ до веб-програми через веб-браузер, наприклад Google Chrome, Mozilla Firefox або Safari. Щоб веб-програма працювала, їй потрібен веб-сервер, сервер додатків і база даних. Веб-сервери керують запитами, які надходять від клієнта, тоді як сервер додатків виконує поставлене завдання. База даних зберігає будь-яку необхідну інформацію. Веб-програми зазвичай мають короткі цикли розробки та невеликі групи розробників. Розробники пишуть більшість веб-додатків на JavaScript, HTML5 або CSS. У програмуванні на стороні клієнта зазвичай використовуються ці мови, які допомагають створювати інтерфейс програми. Програмування на стороні сервера створює сценарії, які використовуватиме веб-програма. Такі мови, як Python, Java і Ruby, зазвичай використовуються в програмуванні на стороні сервера.

Переваги Веб-програми мають багато переваг. Деякі загальні переваги включають наступне: Кілька користувачів можуть отримати доступ до однієї версії програми. Користувачам не потрібно встановлювати додаток. Користувачі можуть отримати доступ до програми через різні платформи, такі як настільний комп’ютер, ноутбук або мобільний пристрій. Користувачі можуть отримати доступ до програми через кілька браузерів.

У секторі мобільних комп’ютерів веб-програми іноді протиставляються власним програмам, які розробники створюють спеціально для певної платформи чи пристрою та встановлюють на цьому пристрої. Власні програми зазвичай можуть використовувати спеціальне апаратне забезпечення пристрою, наприклад

GPS або камеру в мобільній власній програмі. Програми, які поєднують два підходи, іноді називають гібридними програмами. Гібридні програми працюють подібно до веб-програм, але встановлюються на пристрій так само, як рідна програма. Гібридні програми також можуть використовувати ресурси, що стосуються пристрою, за допомогою внутрішніх API. Завантажені рідні програми іноді можуть працювати в автономному режимі; однак гібридні програми не мають цієї функції. Гібридна програма, як правило, матиме подібні елементи навігації до веб-програми, оскільки вони в основному базуються на веб-програмах.

1.2 Порівняння існуючих рішень та аналогів

Функція / Програма	Mint mobile	YNAB	RDS Manager
Створення бюджетів	+	+	+
Відстеження витрат	+	+	+
Потрібна синхронізація з Credit Karma	+	+	-
Інтеграція з банками	+	+	+
Графічне відображення	+	+	+
Персоналізація інтерфейсу	Повна	Повна	Повна

Рис. 1.1 – аналіз аналогів

На ринку існує кілька популярних рішень для управління фінансами, зокрема Mint та YNAB, кожен з яких має свої унікальні можливості та переваги.

Mint є високоавтоматизованим додатком, який здатний імпортувати фінансові дані прямо з банківських рахунків і кредитних карт. Це робить Mint ідеальним для тих, хто цінує зручність і мінімальні зусилля при управлінні своїми фінансами. Додаток автоматично генерує звіти, графіки і надає поради щодо оптимізації витрат, що допомагає користувачам глибше зрозуміти свій фінансовий стан. Однак, варто зазначити, що Mint має деякі обмеження, зокрема у кастомізації бюджетів і потенційні неточності в автоматичній категоризації витрат, що може бути важливим для деяких користувачів.

З іншого боку, YNAB пропонує більш активний підхід до управління фінансами, виховуючи користувачів працювати з принципом "живи на місяць вперед". Цей додаток допомагає планувати кожну витрачену гривню та включає численні інструменти для освіти у сфері фінансів. YNAB ідеально підходить для тих, хто хоче глибоко зануритися в бюджетування та планування свого фінансового майбутнього, пропонуючи сильні можливості для підвищення фінансової грамотності. Проте, додаток має високу вартість підписки та може здатися складним для освоєння новачками через велику кількість функцій та опцій.

Вибір між цими додатками залежить від особистих потреб користувача та його вподобань у стилі управління фінансами. Важливо також враховувати зручність, доступність і користувацький досвід, який надає кожен з додатків. Mint та YNAB пропонують різні підходи до фінансового контролю, кожен з яких може стати цінним інструментом для покращення фінансової поведінки у повсякденном житті.

1.3 Формування вимог до розробленого проекту

Забезпечення успіху проекту "Веб-додаток для контролю фінансів" вимагає детального визначення функціональних вимог і специфікацій, які мають відповідати потребам користувачів і забезпечити ефективне управління їхніми фінансами. Основні аспекти, які необхідно врахувати при формуванні вимог, включають:

Реєстрація та аутентифікація користувачів: Забезпечення можливості для реєстрації нових та аутентифікації існуючих користувачів, гарантуючи при цьому високий рівень безпеки та приватності даних.

Управління бюджетом: Функціональність, яка дозволить користувачам створювати, керувати бюджетами, визначати фінансові цілі та розподіляти кошти між категоріями витрат.

Відстеження доходів та витрат: Можливість для користувачів вводити, категоризувати та моніторити свої доходи та витрати, з автоматичним визначенням категорій транзакцій і відображенням звітів про витрати і доступні кошти. Інтуїтивний інтерфейс: Легкий для розуміння та використання інтерфейс, який дозволяє користувачам без спеціальних технічних навичок ефективно взаємодіяти з усіма функціями додатку.

З урахуванням цих вимог, можна розробити функціональний та користувацьки- приємний веб-додаток для контролю фінансів, який відповідатиме основним потребам користувачів в управлінні їхніми фінансами. У майбутньому ці основи дозволяють розширювати та заповнювати додаток новими функціями для підвищення його ефективності та зручності використання.

2 ВИБІР ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ПРОЄКТУ

2.1 Сервіси авторизації користувачів

У сучасних WEB - застосунках сервіс авторизації користувачів є невід'ємною частиною, оскільки він відповідає за захист конфіденційної інформації та особистих даних користувачів. Ефективна система авторизації забезпечує, що доступ до функціоналу WEB – застосунку мають лише ті користувачі, які пройшли належну перевірку та ідентифікацію. Вона включає декілька ключових компонентів:

Реєстрація користувачів: одним з основних аспектів сервісу авторизації є процес реєстрації нових користувачів. Під час реєстрації користувачі вводять свої персональні дані. Це дозволяє створити унікальний обліковий запис, який буде використовуватись для подальшої ідентифікації та авторизації користувача.

Аутентифікація: після реєстрації, для доступу до свого облікового запису, користувач має ввести свої логін та пароль. Система авторизації перевіряє введені дані зі збереженими в базі даних і, у разі відповідності, надає доступ до додатку. Цей процес гарантує, що доступ має лише особа, яка має право використовувати обліковий запис.

Управління правами доступу: іншим важливим аспектом авторизації є управління правами доступу користувачів. Залежно від ролі користувача в системі (наприклад, адміністратор, менеджер, звичайний користувач), сервіс авторизації визначає рівень доступу до різних функцій додатку. Це дозволяє адміністраторам мати ширші можливості для управління системою, тоді як звичайні користувачі можуть мати обмежений доступ.

Захист даних: для забезпечення конфіденційності та безпеки даних користувачів, сервіс авторизації використовує різні методи захисту, такі як

шифрування паролів, хешування, а також токени доступу. Це знижує ризик несанкціонованого доступу до даних і забезпечує високий рівень захисту.

Відновлення пароля: сервіс авторизації також надає функцію відновлення забутого пароля. У разі втрати пароля користувач може ініціювати процес відновлення, який зазвичай включає підтвердження особи

2.2 Огляд додаткового інструментарію

HTML був вперше розроблений на початку 1990-х років Тімом Бернерсом-Лі, винахідником Всесвітньої павутини. Оригінальна версія HTML була дуже простою і допускала лише просте форматування тексту. З часом до HTML було додано нові функції, щоб зробити його більш потужним і гнучким. HTML 2.0 був випущений у 1995 році та представив багато нових функцій, включаючи таблиці, карти зображень та елементи керування формами. HTML 3.0 був випущений у 1997 році та додав підтримку таблиць стилів і фреймів. HTML 4.0 був випущений у 1998 році та представив ще більше функцій, включаючи підтримку мультимедійного вмісту. HTML 5 був вперше представлений у 2004 році, але лише у 2014 році він став офіційною рекомендацією Консорціуму Всесвітньої павутини (W3C). HTML 5 розроблений як більш гнучкий і потужний, ніж його попередники, з багатьма новими функціями, які полегшують створення динамічних та інтерактивних веб-сторінок.

Хто винайшов HTML? У 1980 році британський учений Тім Бернерс-Лі запропонував і зробив прототип системи для дослідників CERN, де він працював, для використання та обміну документами. У 1989 році Бернерс-Лі написав записку, в якій пропонував систему гіпертексту в Інтернеті. Він визначив HTML і написав програмне забезпечення для браузера та сервера наприкінці 1990 року.

Мова розмітки гіпертексту (HTML) — це набір символів або кодів розмітки, вставлених у файл, призначений для показу в Інтернеті. Розмітка повідомляє веб-браузерам, як відображати слова та зображення веб-сторінки. Кожна окрема частина коду розмітки (між символами «<» і «>») називається

елементом, хоча багато людей також називають його тегом. Деякі елементи постачаються парами, які вказують, коли певний ефект відображення повинен початися і коли він має закінчитися.

Мова розмітки гіпертексту (HTML) — це основна мова сценаріїв, яка використовується веб-браузерами для відтворення сторінок у Всесвітній павутині. HyperText дозволяє користувачеві натиснути посилання та бути перенаправленим на нову сторінку, на яку посилається це посилання. Ранні версії HTML були статичними (Web 1.0), тоді як нові ітерації мають велику динамічну гнучкість (Web 2.0, 3.0). Розмітка — це текст, який відображається між двома гострими дужками (тобто між «<» і «>»), а вміст — це все інше.

Розуміння HTML HyperText Markup Language — це комп'ютерна мова, яка полегшує створення веб-сайтів. Мова, яка має кодові слова та синтаксис, як і будь-яка інша мова, є відносно легкою для розуміння та, з часом, стає все більш потужною в тому, що вона дозволяє комусь створювати. HTML продовжує розвиватися, щоб відповідати запитам і вимогам Інтернету під виглядом Консорціуму Всесвітньої павутини, організації, яка розробляє та підтримує цю мову, наприклад, із переходом до Web 2.0. Гіпертекст — це метод, за допомогою якого користувачі Інтернету переміщуються в Інтернеті. Натискаючи спеціальний текст, який називається гіперпосиланнями, користувачі потрапляють на нові сторінки. Використання гіпер означає, що воно не є лінійним, тому користувачі можуть перейти в будь-яке місце в Інтернеті, просто натиснувши доступні посилання. Розмітка – це те, що теги HTML роблять із текстом усередині них; вони позначають його як певний вид тексту. Наприклад, текст розмітки може бути виділений жирним шрифтом або курсивом, щоб привернути особливу увагу до слова чи фрази.

Основи HTML За своєю суттю HTML — це серія коротких кодів, введених у текстовий файл. Це теги, які забезпечують можливості HTML. Текст зберігається як файл HTML і переглядається через веб-браузер. Браузер читає файл і переводить текст у видиму форму відповідно до вказівок кодів, які автор

використовував для написання того, що стає видимим рендерингом. Написання HTML вимагає правильного використання тегів для створення бачення автора.

Теги - це те, що відокремлює звичайний текст від HTML-коду. Теги – це слова в кутових дужках, які дозволяють графікам, зображенням і таблицям відображатися на веб-сторінці. Різні теги виконують різні функції. Найпростіші теги застосовують форматування до тексту. Оскільки веб-інтерфейси мають стати більш динамічними, можна використовувати каскадні таблиці стилів (CSS) і програми JavaScript. CSS робить веб-сторінки більш доступними, а JavaScript додає потужності базовому HTML.

HTML проти XML На відміну від HTML, Extensible Markup Language або XML дозволяє користувачам визначати власну розмітку. Наприклад, використовуючи XML, один користувач міг вибрати тег для позначення виноски, а інший користувач міг вибрати щось інше. Використовуючи HTML, для позначення певного типу інформації можна використовувати лише один заздалегідь визначений тег. XML-документи призначені для легкого читання, оскільки вони містять теги, визначені користувачем, і оскільки документи складаються лише з розмітки та вмісту.

Як можна використовувати HTML? Використання HTML включає: Розробка веб-сторінок Інтернет-навігація Функція зберігання в браузері Створення веб-документа Розробка гри Збагачення сайту

Суть Мова розмітки гіпертексту, або HTML, — це набір символів або кодів розмітки, вставлених у файл, призначений для показу в Інтернеті. Розмітка повідомляє веб-браузерам, як слова та зображення мають відображатися на веб-сторінці.

HTML, або мова гіпертекстової розмітки, є стандартною мовою розмітки, яка використовується для створення веб-сторінок. HTML забезпечує спосіб структурування вмісту та додавання йому семантичного значення. HTML існує з перших днів Інтернету, і протягом багатьох років він пройшов кілька версій.

HTML 5 — це остання версія мови з багатьма новими функціями, які роблять її потужним інструментом для веб-розробників.

HTML 5 має багато нових функцій, які роблять його потужним інструментом для веб-розробників. Деякі з ключових функцій HTML 5 включають: Покращена підтримка мультимедійного вмісту: HTML 5 забезпечує вбудовану підтримку відео- та аудіовмісту, що полегшує вбудовування цих типів медіа на веб-сторінки. Елемент Canvas: HTML 5 представляє елемент Canvas, який надає можливість створювати динамічну графіку та анімацію за допомогою JavaScript. Офлайн-веб-програми: HTML 5 забезпечує підтримку офлайн-веб-програм, дозволяючи доступ до веб-сторінок, навіть якщо користувач не підключений до Інтернету. Покращені форми: HTML 5 містить кілька нових елементів керування формами, зокрема засоби вибору дати, повзунки діапазону та засоби вибору кольорів. Покращена доступність: HTML 5 містить кілька нових функцій доступності, наприклад можливість додавати підписи та субтитри до відеовмісту. Семантична розмітка: HTML 5 представляє кілька нових семантичних елементів, таких як стаття, розділ і навігація, які забезпечують спосіб структурування вмісту та додання йому сенсу.

Використання HTML 5 для веб-розробки має багато переваг. Деякі з ключових переваг включають:

Покращена продуктивність: HTML 5 включає кілька нових функцій, які спрощують створення швидких і адаптивних веб-сторінок.

Легше розробляти та підтримувати: HTML 5 забезпечує простіший і послідовніший спосіб створення веб-сторінок, полегшуючи розробку та підтримку веб-додатків.

Сумісність із різними платформами: HTML 5 розроблено для роботи на всіх сучасних веб-переглядачах і пристроях, що робить його чудовим вибором для створення веб-програм, які працюють на різних платформах.

Покращена взаємодія з користувачем: HTML 5 надає кілька нових функцій, таких як підтримка мультимедійного вмісту та автономних веб-програм, які можуть покращити роботу веб-програм для користувача.

Використання HTML 5 для створення веб-сторінок відносно просте. Наведемо основні кроки для створення веб-сторінки HTML 5: Додайте декларацію типу документа: на початку свого HTML-документа додайте декларацію типу документа, щоб вказати, що ви використовуєте HTML 5. Додайте елемент HTML: Елемент HTML є кореневим елементом документа HTML. Він має охоплювати всі інші елементи вашого документа. Додайте елемент head: Елемент head містить інформацію про документ, як-от назву та будь-які метадані. Додайте елемент body: елемент body містить вміст документа, як текст, зображення та інші елементи. Використовуйте семантичну розмітку: використовуйте семантичні елементи, такі як стаття, розділ і навігація, щоб структурувати свій вміст і додати йому значення.

Щоб створювати форми, якими простіше користуватися та які доступніше потрібно використовувати елементи керування формами HTML 5, наприклад засоби вибору дати та повзунки діапазону.

Наведемо основні кроки:

Додайте мультимедійний вміст: використовуйте нові відео- та аудіоелементи, щоб вставляти мультимедійний вміст у свої веб-сторінки. Використовуйте елемент canvas: використовуйте елемент canvas для створення динамічної графіки та анімації за допомогою JavaScript.

Додайте підтримку офлайнових веб-програм: використовуйте нові функції офлайнових веб-програм, щоб створювати веб-програми, до яких можна отримати доступ, навіть якщо користувач не підключений до Інтернету.

HTML 5 — це потужний інструмент для веб-розробників, який надає багато нових функцій і переваг. Завдяки покращеній підтримці мультимедійного вмісту, кращим формам і покращеній доступності HTML 5 полегшує створення динамічних та інтерактивних веб-сторінок. Його покращена продуктивність,

спрощена розробка та обслуговування, а також крос-платформна сумісність роблять його чудовим вибором для створення веб-додатків, які працюють на різних платформах.

Розглянемо чому HTML називають мовою розмітки. Останнє оновлення створене 5 квітня 2024 р HTML розшифровується як HyperText Markup Language. HTML – це поширена мова, яка використовується для створення веб-сторінок. Він допомагає організувати та структурувати вміст веб-сторінки за допомогою спеціальних кодів, які називаються розміткою. Він складається з елементів або тегів, які визначають різні частини веб-сторінки. Документи HTML інтерпретуються веб-браузерами для відображення вмісту користувачам.

Яке призначення мови розмітки? Мови розмітки визначають структуру та подання тексту. Вони дозволяють користувачам коментувати текст з інструкціями щодо того, як його слід відображати. Мови розмітки забезпечують стандартизований спосіб форматування та стилізації документів. Мови розмітки полегшують доступність і оптимізацію пошукової системи, надаючи структурований вміст.

Наведемо значення мови розмітки в HTML. HTML розмітка відноситься до тегів, які використовуються для визначення елементів у документі. Ці теги вказують на те, як має бути відображено або структуровано вміст. Розмітка HTML включає теги для заголовків, абзаців, посилань, зображень тощо. Розмітка в HTML укладена в кутові дужки (< >) і зазвичай подається парами (відкриваючі та закриваючі теги). Розмітка HTML необхідна для створення добре структурованих і семантично значущих веб-сторінок.

HTML не є мовою програмування. HTML не може виконувати обчислення або виконувати алгоритми. Він не має таких функцій, як змінні, цикли чи умовні оператори. HTML в основному використовується для структурування та представлення вмісту, а не для реалізації логіки. На відміну від мов програмування, HTML зосереджується на визначенні компонування та організації

веб-документів. Через обмежену функціональність HTML вважається мовою розмітки, а не мовою програмування.

Переваги та недоліки використання HTML Переваги: Простий і легкий у вивченні, що робить його доступним для початківців. Підтримується всіма веб-браузерами, що забезпечує сумісність між платформами. Забезпечує семантичну розмітку, покращуючи доступність і SEO. Дозволяє інтегрувати такі мультимедійні елементи, як зображення, відео та аудіо. Легкий і швидкий для завантаження, покращуючи взаємодію з користувачем.

Наведемо недоліки: обмежений з точки зору інтерактивності та динамічної функціональності. Це може призвести до надмірності коду для складних макетів. Потрібні додаткові мови стилів і сценаріїв (CSS і JavaScript) для вдосконаленого дизайну та поведінки. Може не забезпечити достатній контроль над презентацією на всіх пристроях і розмірах екрана. Уразливий до ризиків безпеки, таких як міжсайтовий сценарій (XSS), якщо його не оброблено належним чином.

Веб-сторінка HTML із семантичними тегами Веб-сторінка використовує семантичні теги HTML для визначення своєї структури, покращуючи доступність і оптимізуючи пошукову систему. Семантичні теги, такі як <header>, <nav>, <main>, <section> і <footer>, надають ясність і сенс вмісту, підвищуючи його організацію та зручність використання.

Розглянемо навіщо потрібні семантичні елементи.

В HTML4 розробники використовували свої власні імена ідентифікаторів / класів для оформлення елементів: header, top, bottom, footer, menu, navigation, main, container, content, article, sidebar, topnav і т.д.

Це унеможливило для пошукових систем визначити правильний зміст вебсторінки.

З новими HTML5 елементами (<header> <footer> <nav> <section> <article>) це стало зробити набагато легше.

Семантика - це вивчення значень слів і фраз в мові.

Згідно W3C, семантичний Web: "дозволяє обмінюватися та спільно використовувати дані в додатках, підприємствах і спільнотах."

Семантичний елемент чітко описує його значення як для браузера, так і для розробника.

Приклади не семантичних елементів: та - нічого не говорять про свій зміст.

Приклади семантичних елементів: <form>, <table> та <article> - чітко вказують свій зміст.

Семантичні елементи HTML5 підтримуються у всіх сучасних браузерах.

Крім того, ви можете "навчити" старі браузери поводитись з "невідомими елементами".

Семантичні елементи також важливі для пошукових систем, які завдяки семантичним елементам "краще розуміють", який зміст знаходиться на вебсторінці.

Багато веб-сайтів містять HTML-код, наприклад, такий: , , , що вивзначає навігацію, верхній та нижній колонтитули.

HTML5 пропонує спеціальні семантичні елементи для визначення різних частин вебсторінки.

Нижче наведено алфавітний список деяких семантичних елементів в HTML5.

- <article> Визначає статтю.
- <aside> Визначає зміст, крім змісту сторінки.
- <details> Визначає додаткові деталі, які користувач може переглядати або приховувати.
- <figcaption> Визначає підпис для елемента <figure>.
- <figure> Визначає автономний зміст, наприклад, илюстрації, діаграми, фотографії, списки кодів і т.ін.
- <footer> Визначає нижній колонтитул для документа або розділу.

- <header> Визначає заголовок для документа або розділу.
- <main> Визначає основний зміст документа.
- <mark> Визначає помічений / виділений текст.
- <nav> Визначає навігаційні посилання.
- <section> Визначає розділ в документі.
- <summary> Визначає видимий заголовок для елемента <details>.
- <time> Визначає дату/час.

Елемент <footer> визначає нижній колонтитул для документа або розділу.

Елемент <footer> має містити інформацію про елемент, який його містить.

Елемент <footer> зазвичай містить:

інформація про авторство

інформація про авторські права

контактна інформація

мапа сайта

посилання повернення вгору

пов’язані документи

Може бути кілька елементів <footer> в одному документі. Наведемо приклад

```
<footer>
 <p>Posted by: Hege Refsnes</p>
 <p>Contact information: <a href="mailto:someone@example.com">
                                someone@example.com</a>.</p>
</footer>
```

Елемент <header> являє собою контейнер для вступного контенту або набору навігаційних посилань.

Елемент <header> зазвичай містить:

один або кілька елементів заголовку (<h1> - <h6>)

лого або іконку

інформацію про авторство

В одному HTML-документі може бути кілька елементів <header>.

Однак <header> неможна помістити в <footer>, <address> або інший елемент <header>.

Наступний приклад визначає заголовок статті:

<article>
<header>
<h1>What Does WWF Do?</h1>
WWF's mission:
</header>
WWF's mission is to stop the degradation of our planet's natural environment, and build a future in which humans live in harmony with nature.
</article>

Елемент<article> визначає незалежний, автономний зміст.

Елемент <section> визначає розділ в документі.

Отже, в Інтернеті є HTML-сторінки з елементами <section> які містять елементи <article> та елементи <article>, які містять елементи <section>.

Також є сторінки з елементами <section>, які містять елементи <section> та елементи <article>, які містять елементи <article>.

Елемент <article> визначає незалежний, автономний контент.

Стаття повинна мати смисл сама по собі, і має бути можливість читати її незалежно від іншої частини сайту.

Приклади того, де елемент <article> може використовуватись:

Повідомлення на форумі

Публікація в блозі

Газетна стаття

Наведемо приклад:

<article>
<h2>Чим займається WWF?</h2>
Місія WWF - зупинити деградацію природного середовища нашої планети

Земля,

і побудувати майбутнє, в якому люди будуть жити в гармонії з природою.

</p>

</article>

Елемент <section> визначає розділ у вебдокументі.

Згідно з документацією W3C по HTML5: "Розділ - це тематичне угруповання контенту, зазвичай з заголовком."

Домашню сторінку зазвичай можна розділити на розділи (секції) для ознайомлення, змісту і контактної інформації. Наведемо приклад:

```
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is .. </p>
</section>
```

Призначення підпису до малюнка - додати візуальне пояснення до зображення.

У HTML5 малюнок і підпис до нього можуть бути згруповані в елемент наведемо приклад <figure>:

```
<figure>
  <img src="pic_trulli.jpg" alt="Trulli">
  <figcaption>Fig1. - Trulli, Puglia, Italy.</figcaption>
</figure>
```

Елемент <img> визначає зображення, елемент <figcaption> визначає підпис до нього.

Розглянемо Каскадні таблиці стилів (CSS) — це мова, яка використовується для стилізації веб-сторінок, які містять елементи HTML, що визначає спосіб відображення елементів на веб-сторінках, зокрема макет, кольори, шрифти та інші властивості елементів на веб-сторінці. CSS працює, орієнтуючись на елементи HTML і застосовуючи правила стилю, щоб визначити, як вони мають

відображатися, включаючи такі властивості, як колір, розмір, макет і позиціонування.

CSS розшифровується як каскадні таблиці стилів. CSS економить багато роботи. Він може керувати макетом кількох веб-сторінок одночасно.

Що таке CSS? Каскадні таблиці стилів (CSS) використовуються для форматування макета веб-сторінки. За допомогою CSS ви можете керувати кольором, шрифтом, розміром тексту, інтервалом між елементами, розташуванням і компонуванням елементів, які фонові зображення або кольори фону використовувати, різними дисплеями для різних пристроїв і розмірами екрану, а також набагато більше!

Використання CSS CSS можна додати до документів HTML трьома способами: Inline - за допомогою атрибута style всередині елементів HTML Внутрішній – за допомогою елемента <style> в розділі <head> Зовнішній – за допомогою елемента <link> для посилання на зовнішній файл CSS Найпоширенішим способом додавання CSS є збереження стилів у зовнішніх файлах CSS. Однак у цьому підручнику ми використовуватимемо вбудовані та внутрішні стилі, тому що це легше продемонструвати, і вам легше спробувати самостійно. Вбудований CSS Вбудований CSS використовується для застосування унікального стилю до одного елемента HTML. Вбудований CSS використовує атрибут style елемента HTML. У наступному прикладі колір тексту елемента <h1> встановлюється синім, а колір тексту елемента <p> - червоним:

Наведемо приклад

```
<h1 style="color:blue;">A Blue Heading</h1>

<p style="color:red;">A red

paragraph.</p>Кольори, шрифти та

розміри CSS.
```

Властивість кольору CSS визначає колір тексту, який буде використовуватися. Властивість CSS font-family визначає шрифт, який буде

використовуватися. Властивість CSS font-size визначає розмір тексту, який буде використовуватися.

Границя CSS Властивість CSS border визначає рамку навколо елемента HTML. Порада. Ви можете визначити рамку майже для всіх елементів HTML.

приклад Використання властивості CSS border:

```
p {
border: 2px solid powderblue;
}
```

CSS Padding Властивість CSS padding визначає відступ (пробіл) між текстом і рамкою. приклад Використання властивостей рамки та відступу CSS:

```
p {
border: 2px solid powderblue;

padding: 30px;
}
```

Нижче наведено три типи CSS: Вбудований CSS Внутрішній або вбудований CSS Зовнішній CSS.

Вбудований CSS — це метод застосування стилю безпосередньо до окремих елементів HTML за допомогою атрибута «style» у тегу HTML, що дозволяє використовувати певний стиль для окремих елементів у документі HTML, замінюючи будь-які зовнішні чи внутрішні стилі.

Приклад вбудованого CSS: У цьому прикладі показано використання вбудованого CSS за допомогою документа HTML.

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Inline CSS</title>
</head>
<body>
    <p style="color:#009900;
            font-size:50px;
            font-style:italic;
            text-align:center;">
        GeeksForGeeks
    </p>
</body>
</html>
```

Пояснення прикладу вбудованого CSS. У наведеному прикладі виконано наступні кроки: використано Inline CSS безпосередньо до елемента абзацу. Змінює колір на зелений, розмір шрифту на 50 пікселів, стиль на курсив і вирівнювання по центру. Стиль перекриває зовнішній і внутрішній CSS.

Внутрішній або вбудований CSS: Внутрішній або вбудований CSS визначається в елементі <style> документа HTML. Він застосовує стилі до вказаних елементів HTML. Набір правил CSS має міститися у файлі HTML у розділі head, тобто CSS вбудовано в тег <style> у розділі head файлу HTML. Приклад внутрішнього або вбудованого CSS: У цьому прикладі показано використання внутрішнього CSS за допомогою документа HTML.

```
            A computer science portal for geeks
        </div>
    </div>
</body>
```

```
</html>
```

Пояснення прикладу внутрішнього або вбудованого CSS: У наведеному вище прикладі ми виконуємо ці кроки Ми використовуємо внутрішні стилі CSS, визначені в елементі <style> в розділі <head>. Правила CSS застосовуються до елементів із певними класами, як-от «main», «GFG» і «geeks». Клас «GFG» стилізує текст зеленим, великим шрифтом і жирним шрифтом. Клас «гіки» стилізує текст жирним шрифтом і меншим розміром шрифту.

Зовнішній CSS містить окремі файли CSS, які містять лише властивості стилю за допомогою атрибутів тегів (наприклад, клас, ідентифікатор, заголовок тощо). Властивість CSS записується в окремому файлі з розширенням .css і повинна бути пов’язана з HTML-документом за допомогою тегу посилання. Це означає, що для кожного елемента стиль можна встановити лише один раз і застосовуватиметься на всіх веб-сторінках. Приклад зовнішнього CSS: У цьому прикладі показано використання зовнішнього CSS за допомогою документа HTML. htmlCSS

```
<!DOCTYPE html>
<html>

<head>
    <title>External CSS</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div class="main">
        <div class="GFG">GeeksForGeeks</div>

        <div id="geeks">
```

```
            A computer science portal for geeks
        </div>
    </div>
</body>

</html>
```

Пояснення прикладу зовнішнього CSS. У наведеному прикладі використовуємо зовнішній файл CSS «style.css», у якому визначаємо колір тіла як синій. За допомогою класу .main текст вирівнюється по центру. Клас .GFG встановлює колір тексту на зелений, розмір шрифту на 50 пікселів і щільність шрифту на жирний. #geeks ID застосовує жирний стиль шрифту та розмір шрифту 20 пікселів.

Вбудований CSS має найвищий пріоритет, потім іде внутрішній/вбудований, а потім зовнішній CSS, який має найменший пріоритет. На одній сторінці можна визначити кілька таблиць стилів.

Для тегу HTML стилі можуть бути визначені в кількох типах стилів у наведеному нижче порядку. Оскільки Inline має найвищий пріоритет, будь-які стилі, визначені у внутрішніх і зовнішніх таблицях стилів, замінюються вбудованими стилями. Внутрішній або вбудований стоїть другим у списку пріоритетів і замінює стилі у зовнішній таблиці стилів. Зовнішні таблиці стилів мають найменший пріоритет. Якщо немає стилів, визначених ні у вбудованій, ні у внутрішній таблиці стилів, тоді для тегів HTML застосовуються зовнішні правила таблиці стилів.

CSS є основою веб-сторінок і використовується для розробки веб-сторінок шляхом оформлення веб-сайтів і веб-додатків. Ви можете вивчити CSS з нуля, дотримуючись цього підручника та прикладів CSS.

Розглянемо атрибут стилю HTML для вбудованого стилю Використовуйте елемент HTML <style>, щоб визначити внутрішній CSS Використовуйте елемент HTML <link> для посилання на зовнішній файл CSS Використовуйте елемент

HTML <head> для зберігання елементів <style> і <link> Використовуйте властивість кольору CSS для кольорів тексту Використовуйте властивість CSS font-family для текстових шрифтів Використовуйте властивість CSS font-size для розміру тексту Використовуйте властивість CSS border для меж Використовуйте властивість padding CSS для простору всередині рамки Використовуйте властивість поля CSS для простору за межами

2.3 Опис сервісу авторізації користувачів

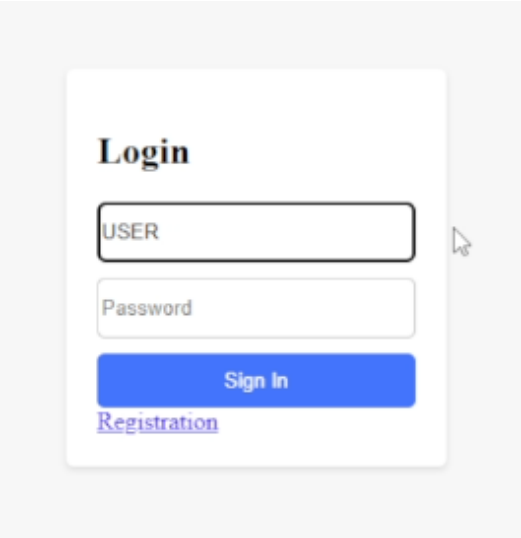


Рис. 2.1 – екран авторизації користувача

Дослідження та порівняння функціональностей та можливостей віддало первагу Firebase Authentication. Розглянемо основні підстави обрання Firebase Authentication:

Різноманітність методів аутентифікації: Firebase Authentication підтримує широкий спектр методів аутентифікації, включаючи вхід через електронну пошту та пароль, соціальні мережі (такі як Google, Facebook, та Twitter), номери телефонів та інші ідентифікатори, як Apple Sign-In. Це дає можливість забезпечити зручність користувачам, які мають різні переваги щодо методів входу в систему.

Легкість інтеграції: Firebase Authentication надає добре документований API та SDK для різних платформ, що дозволяє швидко інтегрувати сервіс у різноманітні додатки. Надані компоненти UI, такі як форми реєстрації, входу та відновлення паролю, можна легко кастомізувати і вбудувати у власні інтерфейси.

Високий рівень безпеки: Firebase Authentication використовує сучасні методи шифрування та забезпечення безпеки даних. Всі комунікації між клієнтом і сервером зашифровані, а паролі захищені через хешування, що забезпечує надійний захист від несанкціонованого доступу.

Управління користувачами та ролями: Firebase Authentication дозволяє ефективно керувати користувачами та їхніми ролями. Функціональність API дозволяє з легкістю створювати, оновлювати та видаляти користувачів, а такожуправляти їхніми дозволами, забезпечуючи точний контроль доступу до ресурсів додатку.

Інтеграція з іншими сервісами Firebase: Однією з основних переваг є вбудована інтеграція з іншими продуктами Firebase, такими як Firestore, Firebase Realtime Database, і Cloud Functions. Це відкриває широкі можливості для розширення функціональності додатку та використання зв'язаних сервісів для більш глибокої інтеграції.

Підтримка та документація: Firebase забезпечує розгорнуту документацію вирішуючи технічні проблеми та знаходити відповіді на запитання. Велика кількість прикладів коду та підтримка з боку Google гарантують високий рівень допомоги під час розробки.

Загалом, Firebase Authentication дозволяє максимально зосередитися на основному функціоналі додатку, знижуючи складність та час, необхідний для розробки систем авторизації.

2.4 Допоміжні інструменти використані для налаштування клієнтської частини

При розробці застосунку було обрано декілька допоміжних інструментів, які значно полегшили роботу з клієнтською частиною веб-застосунку. Зокрема використано React Router, що дозволило забезпечити високу якість коду та ефективність розробки.

Використання React дозволило мені отримати такі переваги:

Статична типізація: Вказівка конкретних типів для змінних, параметрів функцій, і повернення значень допомагає виявити помилки ще на етапі розробки, значно зменшуючи час на дебагінг.

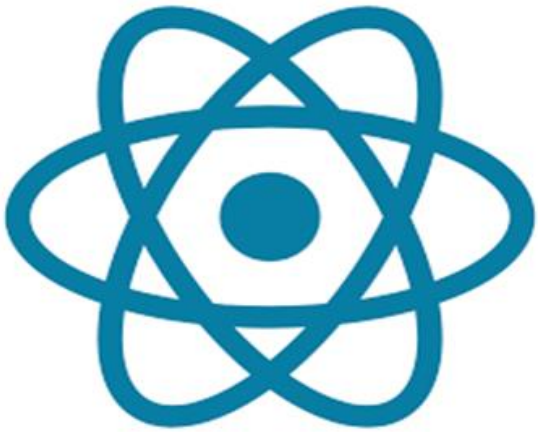


Рис. 2.2 – Позначка React

Підказки IDE: Використання розширених можливостей редакторів, таких як VS Code, допомагає отримувати підказки, автодоповнення коду та швидше виявляти потенційні проблеми завдяки інтеграції з TypeScript.

Краща організація коду: Застосування типізації та інтерфейсів сприяє кращій структуризації та організації коду, роблячи його легшим для читання та подальшої підтримки.

React Router — це стандартний вибір для маршрутизації в React-додатках, який надає такі можливості:

Декларативна маршрутизація: Визначення маршрутів через компоненти дозволяє інтуїтивно зрозуміти структуру URL-адрес та їх відповідність

компонентам.

Параметризовані маршрути: Можливість використовувати динамічні параметри в URL дозволяє розробляти гнучкіші та більш адаптивні веб-додатки.

Вкладена маршрутизація: Створення складної структури маршрутів всередині додатка полегшує управління багаторівневими інтерфейсами.

Для написання та тестування коду я вибрав Visual Studio Code, який є відкритим джерелом і повністю безкоштовним, на відміну від інших комерційних рішень. VS Code пропонує:

Інтеграцію з різними плагінами: Від Git до лінтерів та веб-серверів, що дозволяє значно розширити функціонал та адаптувати середовище під конкретні потреби проєкту.

Легка вага та висока продуктивність: Незважаючи на велику функціональність, редактор залишається швидким і не перевантажує систему навіть при великих проєктах.

Ці інструменти та технології забезпечили мені ефективні засоби для розробки, дозволяючи зосередитися на вирішенні задач бізнес-логіки, а не на вирішенні технічних проблем налаштування середовища.

2.5 Інструменти для налаштування серверної частини

При розробці серверної частини мого дипломного проєкту я вирішив використати мову програмування Python. Це популярний вибір серед розробників завдяки його читабельності, гнучкості та широкому спектру застосування. Python ідеально підходить для веб-розробки, наукових досліджень, скриптингу та багатьох інших задач.

Розглянемо основні переваги Python.

На рис. 2.3 наведено приклад коду Python.

```python
def calculate_expenses(expenses):
    """Функція для обрахунку суми витрат зі списку."""
    total = 0
    for expense in expenses:
        total += expense
    return total

# Список з прикладами витрат
expenses_list = [150, 230, 120, 50]

# Виклик функції та вивід результату
total_expenses = calculate_expenses(expenses_list)
print(f"Загальна сума витрат: {total_expenses} грн")
```

Рис. 2.3 – приклад коду Python

Модулі для виконання майже будь-яких завдань, від веб-сервісів до роботи з датами та системними операціями.

Спільнота та підтримка: маючи одну з найбільших спільнот розробників, Python надає величезну кількість відкритих бібліотек, фреймворків і інструментів, які значно спрощують розробку.

Інтерпретована мова: як інтерпретована мова, Python дозволяє швидку ітерацію та тестування, оскільки зміни можуть бути виконані та перевірені в реальному часі без необхідності компіляції коду. Flask та Django є двома основними фреймворками для веб-розробки на Python, кожен з яких має свої унікальні характеристики та переваги:

Flask є мікро-фреймворком, що надає основні інструменти для створення веб-додатків, але залишає багато архітектурних рішень на розсуд розробника. Це ідеально підходить для невеликих проектів або коли розробники хочуть мати більше контролю над компонентами додатка. На рис. 2.4 наведено приклад коду Flask.

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Привіт, світ! Ласкаво просимо до мого Flask додатку!'

if __name__ == '__main__':
    app.run(debug=True)
```

Рис. 2.4 – приклад коду Flask

На рис.2.5 наведено приклад коду Django.

```python
import sys
from django.conf import settings
from django.core.management import execute_from_command_line
from django.http import HttpResponse
from django.urls import path
from django.conf.urls import url

settings.configure(
    DEBUG=True,
    ROOT_URLCONF=sys.modules[__name__],
    SECRET_KEY='asecretkey',
)

def home(request):
    """ Функція вигляду, яка повертає HTTP-відповідь на запит """
    return HttpResponse("Привіт від Django!")

urlpatterns = [
    path('', home, name='home'),
]

if __name__ == '__main__':
    execute_from_command_line(sys.argv)
```

Рис. 2.5 – приклад коду Django

Django є більш великим фреймворком, який слідує філософії "все включено". Він надає багатий набір функціональностей "з коробки", таких як адміністративний інтерфейс, користувацькі аутентифікації та ORM (Object-

Relational Mapping). Django підходить для середніх та великих проектів, де важлива швидкість розробки та висока інтегрованість компонентів.

У контексті застосунку для управління особистими фінансами, де важлива гнучкість та швидкість розробки, було використано Flask. Flask дозволяє легко інтегрувати з Firebase Firestore через SDK, забезпечуючи гнучкість та простоту в обробці запитів та взаємодії з базою даних. Також, Flask підтримує розширення, які можуть бути додані для підтримки більш складних функцій, таких як маршрутизація, валідація форм, та робота з автентифікацією.

Основними елементами цієї системи є класи, які забезпечують роботу з користувачами, аутентифікацію та управлінням правами доступу.

На рисунку 2.6 представлено UML діаграма класів.

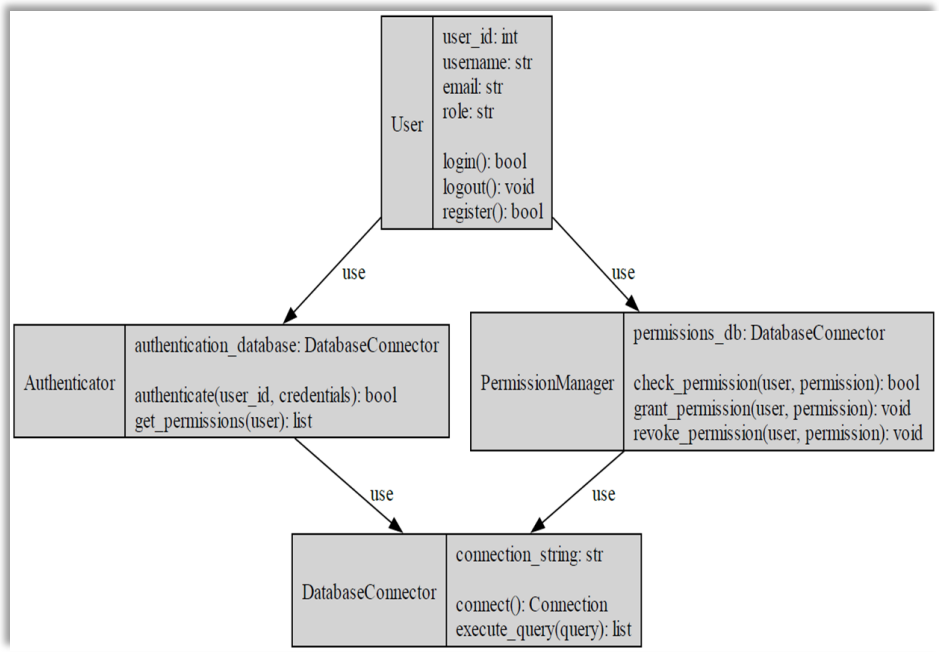


Рис. 2.6 Діаграма класів

На рис. 2.7 наведено діаграма робочого процесу web-застосунку для управління фінансами.

Рис. 2.7 Діаграма робочого процесу

3 ОГЛЯД ТА РОЗРОБКА WEB-ЗАСТОСУНКУ

Одним із підходів до розробки сервіс-орієнтованих веб-додатків є перетворення бізнес-моделей високого рівня на мову композиції, яка реалізує бізнес-процеси за допомогою веб-сервісів. У процесі розробки програмного забезпечення зазвичай використовуються об’єктно-орієнтований аналіз і дизайн, а також діаграми на основі UML.

Технологія Web Services базується на концепції сервіс-орієнтованих обчислень. Веб-сервіси — це стандарти, які об’єднують веб-програми шляхом підключення та спільного використання бізнес-процесів у мережі, де програми різних постачальників, мов і платформ спілкуються між собою та з клієнтами.

Веб-програми — це програми, доступ до яких здійснюється через веб-браузер через мережу та розроблені за допомогою мов, що підтримуються браузером (наприклад, HTML, JavaScript).

Виконання веб-програми залежить від веб-браузерів і включають багато програм, таких як роздрібні онлайн-продажі, онлайн-аукціони та веб-пошта. Веб-додатки потрібні в області взаємодії між компаніями через мережі, наприклад, для закордонних компаній, які передають проекти одна одній.

Прийняття інфраструктури веб-додатків може забезпечити життєво важливі процеси, такі як переказ коштів і оновлення інформації про ціни. Через складність сервісних систем аналіз кожного компонента та підсистеми стає більш складним.

У галузі веб-інженерії існує потреба в методологіях розробки веб-сервісів. Веб-сервіси надають інструмент для розробки та впровадження бізнес-процесів, наприклад, мови опису веб-сервісів (WSDL) і мови виконання бізнес-процесів (BPEL).

Сервіси обіцяють вийти за межі простого обміну інформаціє до концепції доступу, програмування та інтеграції служб додатків. Кінцевим результатом є те, що тоді легше створювати нові складені додатки, які використовують фрагменти

логіки додатків та/або дані, які знаходяться в існуючих системах. Це являє собою фундаментальну зміну соціально-економічної структури спільноти розробників програмного забезпечення, яка покращує ефективність і продуктивність розробки програмного забезпечення.

3.1 Огляд бази даних

Модель даних — це абстракція, що відображає структуру даних, визначає їх взаємозв'язки та правила, які регулюють доступ та маніпуляції з цими даними. Вона вказує, як організовані, зберігаються та обробляються дані в рамках інформаційної системи чи програмного забезпечення. Модель даних може бути візуалізована через схеми чи діаграми, які демонструють структуру даних і їхні зв'язки, окреслюючи типи даних, їх атрибути та методи взаємодії.

Моделі даних служать для проектування баз даних або програмних додатків і є фундаментом для розробки алгоритмів обробки даних. Існує багато різних типів моделей даних, включно з реляційною, ієрархічною, мережевою, об'єктно-орієнтованою моделями, а також моделями на базі XML та іншими. Вибір певної моделі даних залежить від специфіки завдання та вимог до системи, де вона буде використовуватися.

3.1.1 Модель User (користувач)

Кожному бізнесу для процвітання потрібні клієнти, незалежно від бізнес-моделі B2C або B2B. Визначення цільової аудиторії бізнесу є одним із перших кроків до пошуку клієнтів: вам потрібно знати, кого ви хочете охопити своїми продуктами чи послугами та як ваш бізнес має бути адаптований до ринку. Моделювання користувача є одним із методів для цього, і в цій статті ми заглибимося в такі деталі, як:

Розглянемо що таке моделювання користувачів та чому моделювання користувачів є важливим. Наведемо джерела даних і методи моделювання користувачів.

Представимо чотири реальних приклади моделювання користувачів.

Соціальні мережі та пошукові системи.

Управління та вдосконалення продуктів.

Цифровий маркетинг і рекламні технології.

Електронна комерція та роздрібна торгівля.

Будь-яка галузь може отримати вигоду від розуміння того, що спонукає клієнтів і з якими проблемами вони стикаються, тому давайте навчимося використовувати інноваційні технології для нашої користі.

Моделювання користувачів — це більш технічний підхід до визначення цільової аудиторії та адаптації систем до конкретних потреб аудиторії. Це підрозділ взаємодії людини з комп'ютером, який використовується для створення розуміння користувача.

Потрібно надавати релевантний вміст своїм потенційним клієнтам, щоб вони могли стати вашими справжніми клієнтами.

Джерела даних і методи моделювання користувачів. Щоб створити модель користувача для бізнесу, потрібно визначити джерела інформації, які знадобляться.

Наведемо алгоритм запитань про те, яких цілей потрібно прагнути та яка інформація буде корисною для досягнення цієї мети.

Існує багато підходів до моделювання користувачів, розглянемо три, які є найбільш актуальними джерелами інформації:

Відповідність відгуку. Потрібно дізнатися, що шукають люди, а також дізнатися, як вони реагують на запити та відгуки. Таким чином, можемо адаптувати свою систему таким чином, щоб найкращі результати (найбільш релевантні вашим користувачам) за результатами оцінювалися першими.

Моделювання користувача мобільного середовища. Оскільки все змінюється та стає більш мобільним, потрібно адаптувати систему до мобільного середовища, щоб побачити, чи змінюється поведінка користувача за різних обставин. Результати цього дослідження допоможуть вам створити кращий

мобільний інтерфейс користувача, а отже, спростити та оптимізувати шлях вашого користувача.

Демографічне моделювання користувачів. Це один з найбільш традиційних способів створення портрета потенційного користувача.

Статистика демографічних досліджень стосується освіти, релігії, віку, статі тощо. Наприклад, молода дівчина відреагує на оголошення інакше, ніж 60-річна жінка. Або люди активного віку (18 – 35 років), як правило, мають ширші соціальні зв'язки, ніж люди після 35 років. Знаючи ці речі, можемо тонко адаптувати користувацький досвід для відвідувачів.

Розглянемо що робити з інформацією, яку маємо. Це наступне рішення, яке потрібно прийняти. Для цього наведемо кілька підходів.

Статичне моделювання користувача. Якщо зібрана інформація, вона залишається статичною та незмінною. Ці типи наборів даних корисні для служб, які не потребують налаштування на льоту. Наприклад, це може бути блог-платформа, яка не потребує постійних налаштувань.

Динамічне моделювання користувача. При такому підході інформація оновлюється поступово. Дані можуть бути оновлені повністю або деякі їх частини, які більш схильні до змін. Механізми рекомендацій можуть використовувати цей тип підходу моделювання користувачів.

Високоадаптивне моделювання користувача. Деяким підприємствам потрібна екстремальна настройка на льоту, щоб показати своїм клієнтам найрелевантніші та найточніші дані. Наприклад, згадайте свій досвід покупок на Amazon або AliExpress. Коли ви відвідаєте сторінку продукту, ви побачите продукти, пов'язані з тим, що ви бачили.

Опис моделі користувача, яку отримуємо за допомогою сервісу для авторизації користувачів "Firebase Authentication" представлено в таблиці 3.

Таблиця 3.1

Опис моделі User

Назва поля	Тип	Опис
UID	String	Унікальний ідентифікатор користувача, який автоматично генерується Firebase при створенні нового облікового запису.
Email	String	Електронна адреса користувача, яка використовується як основний метод для входу та зв'язку.
Password	String	Хешований пароль користувача. Firebase забезпечує безпечне зберігання паролів, хоча самі паролі зберігаються в зашифрованому вигляді, а не як звичайний текст.
DisplayName	String	Ім'я, під яким користувач представляється у системі. Це необов'язкове поле.
PhotoURL	String	URL-адреса зображення профілю користувача.
PhoneNumber	String	Номер телефону користувача, який може використовуватися для двофакторної аутентифікації або відновлення доступу.
EmailVerified	Boolean	Булевий прапорець, який вказує, чи підтвердив користувач свою електронну адресу.
CreationTime	Timestamp	Час створення облікового запису, записаний як мітка часу.
LastLoginTime	Timestamp	Час останнього входу в систему, також записаний як мітка часу.

Стереотипне моделювання користувачів. Цей тип моделювання користувачів створюється з узагальненої версії статичних профілів, тобто ви

робите припущення щодо користувачів на основі більших фрагментів, об’єднаних спільними характеристиками.

Це протилежність високоадаптивному підходу. Стереотипи корисні, коли вам не потрібна особиста інформація або ви не маєте доступу до неї, наприклад, коли вам потрібно дотримуватися стандартів Загального регламенту захисту даних.

3.2 Модель User (користувач)

В таблиці 3.1 представлено опис моделі Budget, яка дозволяє користувачам створювати бюджети для різних категорій витрат, встановлюючи суму коштів, яку вони планують витратити протягом певного періоду.

Таблиця 3.2

Опис моделі Budget

Назва поля	Тип	Опис
BudgetID	String	Унікальний ідентифікатор бюджету, автоматично генерується Firebase.
UserID	String	Ідентифікатор користувача, якому належить бюджет.
Category	String	Категорія витрат, наприклад, "житло", "транспорт", "розваги".
Amount	Float	Сума, призначена для категорії на відповідний період.
StartDate	Timestamp	Дата початку періоду, на який розрахований бюджет.
EndDate	Timestamp	Дата закінчення періоду, на який розрахований бюджет.
Notes	String	Додаткові примітки або коментарі щодо бюджету.

3.3 Модель Expense (витрата)

В таблиці 3.2 представлено опис моделі Expense.

Таблиця 3.3

Модель Expense

Назва поля	Тип	Опис
ExpenseID	String	Унікальний ідентифікатор витрати, автоматично генерується Firebase.
UserID	String	Ідентифікатор користувача, який здійснив витрату.
Amount	Float	Сума витрати.
Category	String	Категорія витрати, наприклад, "їжа", "транспорт", "освіта".
Date	Timestamp	Дата і час, коли витрата була здійснена.
Description	String	Опис витрати, деталі транзакції або додаткові зауваження.
PaymentMethod	String	Спосіб оплати, наприклад, "готівка", "кредитна карта", "банківський переказ".

3.4 Огляд операцій з даними

В цьому підпункті буде досліджено та розглянуто операції з даниим у розробленому веб-додатку.

3.5 Створення бюджету

Функція створення бюджету в веб-додатку для управління особистими фінансами забезпечує користувачам інструменти для планування та контролю їх фінансів. Цей процес включає кілька важливих етапів:

Визначення фінансових цілей: користувачі починають із встановлення своїх фінансових цілей, таких як заощадження на пенсію, купівля житла, або фінансування освіти. Цілі визначаються в додатку і слугують основою для розробки бюджетних планів.

Аналіз доходів та витрат: додаток надає можливість вводити та автоматично імпортувати дані про доходи і витрати. Це може включати автоматичне отримання інформації з банківських рахунків, кредитних карток, а також введення вручну. Це дозволяє користувачам мати актуальну інформацію про свій фінансовий стан.

Установлення бюджетних категорій та лімітів: Користувачі можуть створювати категорії витрат, такі як "Житло", "Транспорт", "Розваги", та встановлювати ліміти витрат за кожною категорією. Додаток дозволяє встановлювати щомісячні, квартальні або річні бюджети.

Моніторинг та звіти: Додаток автоматично відслідковує витрати по категоріях та порівнює їх з встановленими бюджетними лімітами. Користувачі отримують звіти та сповіщення про стан виконання бюджету, що допомагає їм уникати перевитрат і реагувати на фінансові відхилення вчасно.

Інтеграція з моделями даних: функція бюджетування інтегрована з моделями User і Expense у базі даних. Це забезпечує синхронізацію даних між різними модулями додатку та відображення усієї інформації в одному інтерфейсі.

Функціонал створення бюджету в веб-додатку для контролю фінансів стає ключовим інструментом для користувачів, які прагнуть ефективно керувати своїми фінансами, планувати свої витрати, та досягати фінансових цілей. Завдяки гнучкості налаштувань і інтеграції з фінансовими сервісами, додаток допомагає забезпечувати точне та зручне управління бюджетом.

На рис. 3.1 представлено код, який повідомляє про створення бюджету.

```
from firebase_admin import firestore

db = firestore.client()

def create_budget(user_id, category, amount, start_date, end_date, notes):
    try:
        doc_ref = db.collection('budgets').add({
            'userId': user_id,
            'category': category,
            'amount': amount,
            'startDate': start_date,
            'endDate': end_date,
            'notes': notes
        })
        print(f"Бюджет успішно створено з ID: {doc_ref.id}")
    except Exception as e:
        print(f"Помилка при створенні бюджету: {str(e)}")
```

Рис. 3.1 Створення бюджету

firebase_admin використовується для взаємодії з Firebase Firestore.

db.collection('budgets').add() додає новий документ у колекцію "budgets".

Параметри, які передаються, включають ID користувача, категорію, суму, дати та примітки, що дозволяє фіксувати всі необхідні деталі бюджету у базі даних.

Користувачі вводять деталі, такі як категорія (наприклад, їжа, транспорт), сума бюджету, дати початку та завершення бюджетного періоду, і додаткові примітки, які можуть включати додаткову інформацію про бюджет або спеціальні інструкції.

Збереження в Firestore.

Функція використовує Firestore для збереження інформації в колекції "budgets". Firestore забезпечує швидке та надійне збереження даних, що дозволяє миттєво отримати доступ до збережених даних та забезпечити їх актуальність.

Обробка Помилок.

При роботі з зовнішніми системами, такими як база даних, важливо обробляти можливі помилки. Функція включає блок try-except для перехоплення винятків, які можуть виникнути під час додавання документа до Firestore, такі як мережеві помилки або проблеми з доступом до бази даних.

3.6 Створення витрат

Розглянемо введення Даних про Витрати.

Користувачі можуть вводити деталі своїх витрат, включаючи категорію витрат, суму, дату та додаткові коментарі.

Збереження в Firestore.

Вся інформація про витрати буде зберігатися в базі даних Firestore у відповідній колекції, забезпечуючи легкий доступ та управління.

Автоматичне Оновлення Бюджету.

Після додавання витрати система автоматично оновлює відповідний бюджет, зменшуючи доступний баланс за категорією витрат.

Обробка Помилок.

Функція включає обробку можливих помилок, що можуть виникнути під час введення або збереження даних.

На рис. 3.2 представлено код, який повідомляє про створення витрат.

```python
from firebase_admin import firestore
import datetime

db = firestore.client()

def create_expense(user_id, category, amount, date, notes):
    try:
        expense_ref = db.collection('expenses').add({
            'userId': user_id,
            'category': category,
            'amount': amount,
            'date': datetime.datetime.strptime(date, '%Y-%m-%d'),
            'notes': notes
        })
        print(f"Витрата успішно створена з ID: {expense_ref.id}")
        update_budget(user_id, category, amount)
    except Exception as e:
        print(f"Помилка при створенні витрати: {str(e)}")

def update_budget(user_id, category, amount):
    budget_ref = db.collection('budgets').where('userId', '==', user_id).where('category', '==', category).get()
    if budget_ref:
        for doc in budget_ref:
            new_amount = doc.to_dict()['amount'] - amount
            db.collection('budgets').document(doc.id).update({'amount': new_amount})
            print(f"Бюджет оновлено: новий баланс {new_amount}")
```

Рисунок 3.2 Створення витрат

Функція create_expense дозволяє створювати запис про витрати, заносячи дані до Firestore.

Функція update_budget оновлює бюджет, зменшуючи відповідну суму витрати з доступного бюджету в даній категорії.

Цей підхід забезпечує ефективне відстеження витрат та управління бюджетом, дозволяючи користувачам зберігати контроль над своїми фінансами.

3.7 Видалення бюджету

Функціонал видалення бюджету є важливим аспектом управління фінансами в веб-додатку. Це дозволяє користувачам управляти своїми бюджетами, видаляючи ті, які більше не актуальні або які були створені помилково.

Наведемо основні завдання функції:

Ідентифікація Бюджету.

Користувач повинен мати можливість вибрати бюджет, який він хоче видалити. Це може бути зроблено за допомогою унікального ідентифікатора бюджету.

Підтвердження Видалення.

Перед видаленням бюджету користувачам може бути запропоновано підтвердити своє рішення, щоб уникнути випадкового видалення важливих даних.

Видалення з Бази Даних.

Функція виконає запит до бази даних для видалення відповідного документу з колекції бюджетів.

Обробка Помилок.

Як і при створенні бюджету, важливо обробляти помилки, які можуть виникнути під час видалення бюджету, наприклад, помилки доступу до бази даних або відсутність документа з таким ідентифікатором. На рис. 3.3 представлено код, який повідомляє про видалення бюджету.

```python
from firebase_admin import firestore

db = firestore.client()

def delete_budget(budget_id):
    try:
        db.collection('budgets').document(budget_id).delete()
        print(f"Бюджет з ID: {budget_id} успішно видалений.")
    except Exception as e:
        print(f"Помилка при видаленні бюджету: {str(e)}")
```

Рис. 3.3 Видалення бюджету

Розглянемо складові коду.

Функція delete_budget дозволяє видалити бюджет за його унікальним ідентифікатором.

Використання методу delete() на документі з колекції 'budgets' видаляє цей документ з бази даних.

Обробка винятків допомагає забезпечити, що будь-які помилки під час видалення будуть адекватно оброблені, надаючи користувачу зворотній зв'язок про успіх операції або про проблеми, які виникли.

Ця функція є необхідною для гнучкого управління бюджетами, даючи користувачам можливість адаптувати свої фінансові плани згідно з змінюючимися обставинами або цілями.

3.8 Огляд інтерфейсу веб-додатку

Проведемо детальний аналіз інтерфейсу веб-додатку, створеного для ефективного керування особистими фінансами користувачів. Адекватний інтерфейс є критично важливим аспектом успішної роботи будь-якого додатку, оскільки він формує основу для взаємодії користувачів із системою.

Загальний огляд інтерфейсу.

Введемо в загальну концепцію дизайну, яка включає вибір кольорової палітри, шрифтів та стилів, що забезпечують зручне та привабливе візуальне середовище. Ці елементи сприяють зручності та естетиці додатку, роблячи його більш приємним для користувача.

Опис основних розділів та функцій.

Перейдемо до детального опису ключових сторінок і функцій додатку. Обговоримо:

Перегляд балансу рахунків:

Сторінка дозволяє користувачам швидко отримати огляд їхніх фінансових ресурсів.

Категоризація транзакцій: Надає можливість класифікації витрат за категоріями для кращого фінансового планування.

Створення та управління бюджетами: Ця функція допомагає користувачам планувати та контролювати їхні витрати відповідно до фінансових цілей.

Кожен елемент інтерфейсу буде супроводжуватись детальними інструкціями про його функції та способи взаємодії з користувачем. Включатимемо описи взаємодії з кнопками, полями для вводу, фільтрами, що забезпечують легке та інтуїтивно зрозуміле управління даними.

Представимо сторінки Web-застосунку на яки потрапляє користувач. На рис. 3.4 наведено вітальна сторінка додатку.

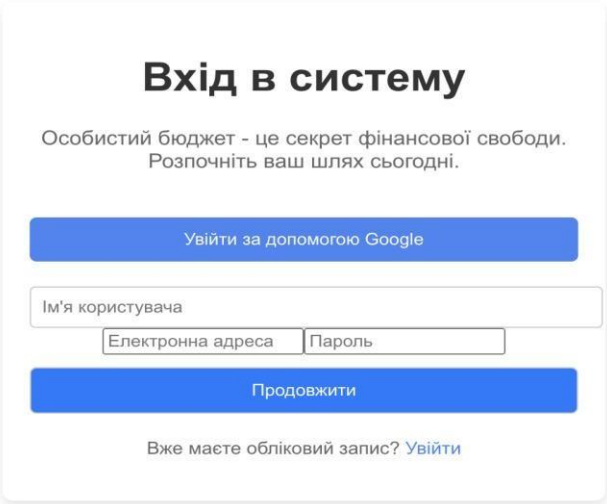


Рис. 3.4 Вітальна сторінка додатку

Додаткові екрани.

Опис додаткових екранів для перегляду та управління витратами, бюджетами, а також функцій, пов'язаних зі створенням нових витрат чи бюджетів та їх видаленням. Ці сторінки надають користувачам гнучкість у керуванні своїми фінансами відповідно до особистих потреб і цілей.

Основні елементи управління та навігації знаходяться на вітальній сторінці додатку. Огляд початкового екрану, вітальної сторінки, виступає як вхідна точка в додаток і містить основні елементи управління та навігації. На вітальній сторінки, користувач отримує актуальну інформацію про свій фінансовий стан, включаючи останні транзакції.

Наступна стартова сторінка веб-додатку наведена для контролю фінансів користувачам та надає можливість зареєструватися і почати управління своїми фінансами. Реєстрація користувачів відбувається наступними способами:

1. Метод облікового запису Google, який дозволяє використовувати існуючий акаунт Google для швидкого входу.

2. Класичний метод: введення електронної пошти для створення нового профілю в системі.

На рис. 3.5 наведено головна сторінка додатку.

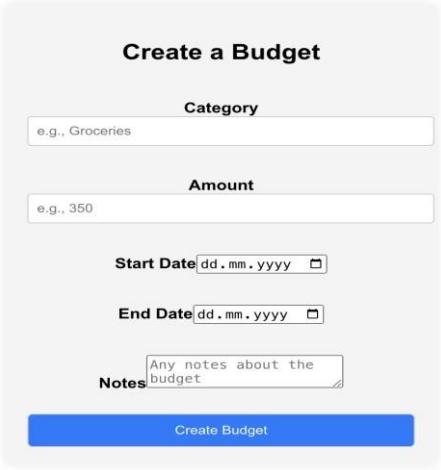


Рис. 3.5 Головна сторінка додатку

Сервіс надає можливість розширити список соціальних мереж для реєстрації, додаючи такі платформи як Facebook, Slack, GitHub, що дозволяє користувачам вибрати найзручніший для них спосіб авторизації. Після входу в систему, користувач отримує доступ до головної форми для створення бюджету. На цій сторінці користувач може легко планувати свої фінанси, вказуючи категорію витрат, суму, а також дати початку та закінчення бюджетного періоду. Також є можливість додавання приміток для деталізації бюджетного плану. Всі ці дані допоможуть користувачу ефективно керувати своїми коштами, встановлюючи чіткі фінансові рамки для кожної категорії витрат. На рис. 3.6 наведено головна сторінка після створення бюджету.

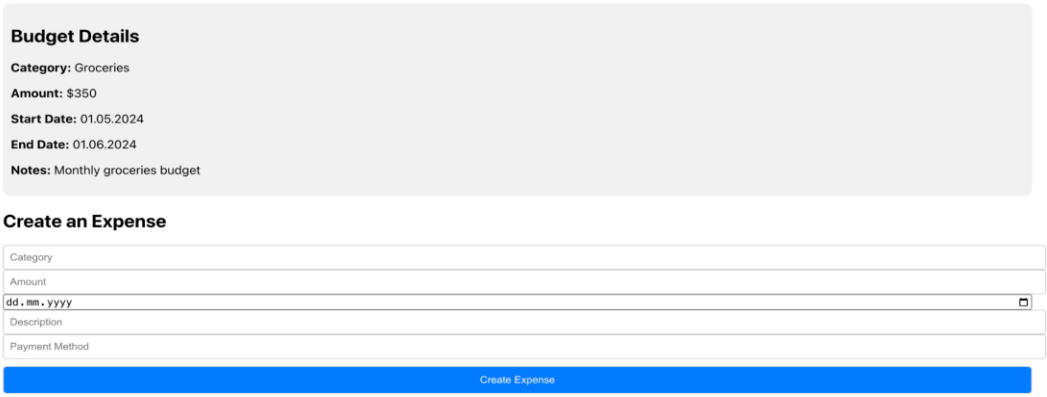


Рис. 3.6 Головна сторінка після створення бюджету

На рис. 3.7 наведено головна сторінка після створення витрат.

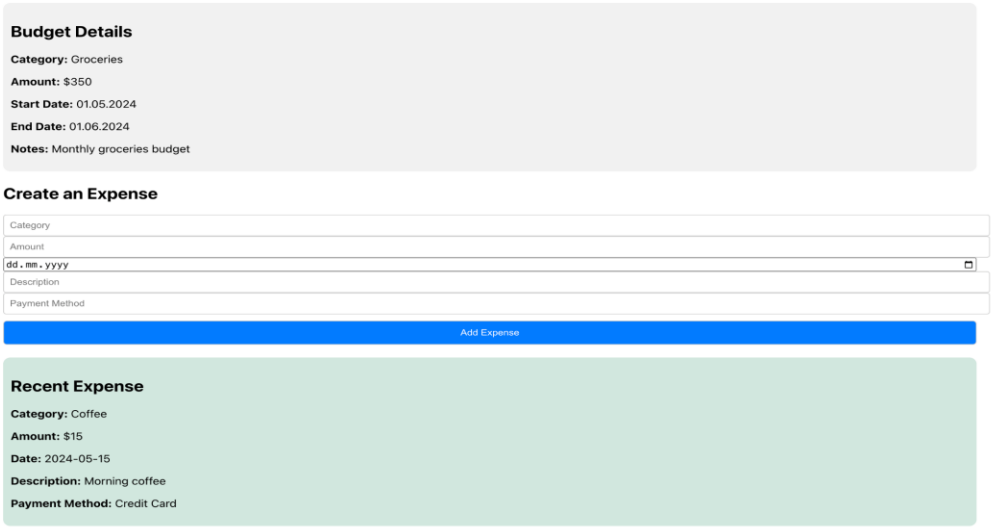


Рис. 3.7 Головна сторінка після створення витрат

Після успішного створення бюджету на сторінці з'являється нова форма для додавання витрат. У цій формі користувач може вказати категорію витрат, суму, дату, опис та спосіб оплати, щоб ефективно керувати своїми фінансами в межах заданого бюджету.

Для більшої наглядності і практичності, на сторінці було додано відображення прикладу уже створеної витрати, яка містить детальну інформацію про витрати на каву. Ця демонстрація допомагає користувачам зрозуміти, як вони можуть ефективно вводити та відслідковувати свої фінансові операції у веб-додатку. Отже, коли користувач вносить дані про нову витрату, він може відразу побачити приклад того, як ця інформація буде представлена і збережена у системі. На рис. 3.8 наведено вікно налаштувань облікового запису користувача.

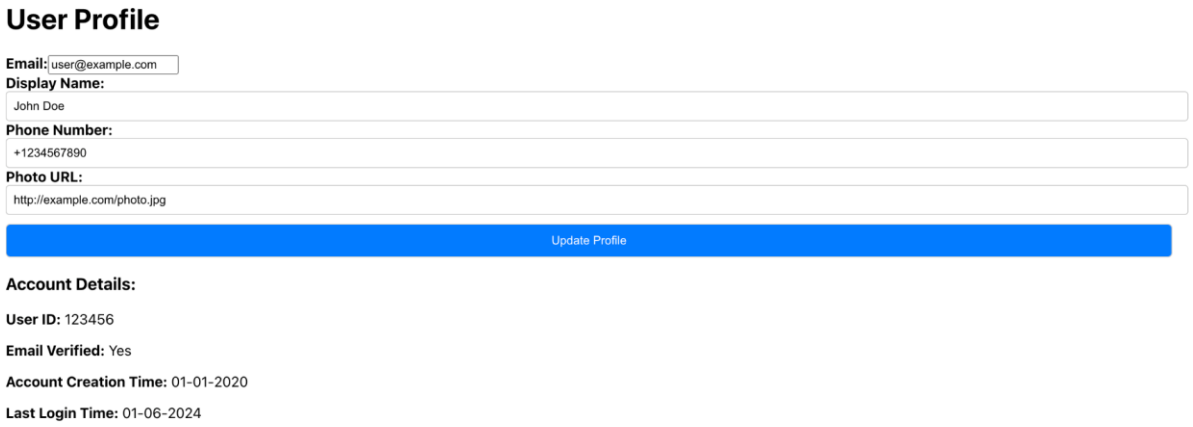


Рис. 3.8 Вікно налаштувань облікового запису користувача

Завдяки використанню функціональності додатку, користувачам надається можливість виконувати різноманітні операції, пов'язані з управлінням їхнім профілем, а саме:

Оновлення електронної адреси, що дозволяє змінювати основний засіб комунікації з додатком.

Зміна імені для відображення, яке використовується в усіх модулях системи для ідентифікації користувача.

Оновлення номера телефону, що може використовуватися для додаткової верифікації та відновлення доступу.

Модифікація URL фотографії профілю, що надає користувачам можливість персоналізувати свій візуальний образ в додатку. На рис. 3.9 наведено вікно зміни паролю.

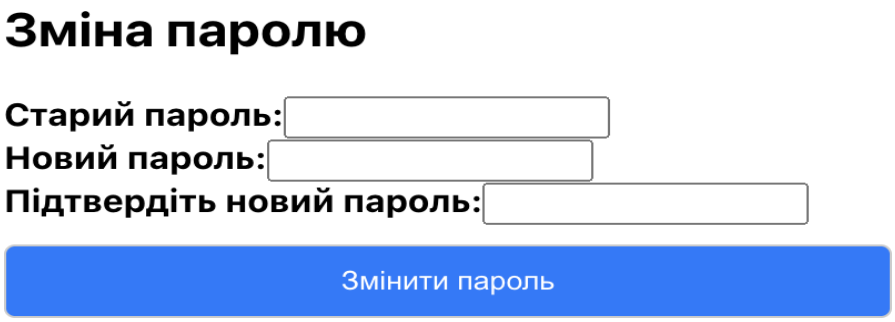


Рис. 3.9 Вікно зміни паролю

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Ручне тестування Web-застосунку для управління особистими фінансами

Для створення ручного тестування застосунок для управління особистими фінансами з використанням Python, HTML та CSS необхідно виконати наступні кроки:

Визначте основні функції, які повинні бути використані, такі як: створення бюджету, ведення витрат, реєстрація та аутентифікація, перегляд та аналіз фінансів, видання даних, зміна даних профілю.

Оцініть нефункціональні вимоги : Визначте вимоги до швидкості, інтуїтивно зрозумілого інтерфейсу та стабільності роботи.

Для тестування веб-інтерфейсу використовуйте Selenium WebDriver, який дозволяє автоматизувати тестування веб-сторінок. Напишіть код тестів : Напишіть код тестів на Python, використовуючи Selenium WebDriver та бібліотеки для тестування, наприклад pytest або unittest.

Тестування на навантаженість - протестуйте застосунок на навантаженість, щоб зрозуміти, як він працює під час високого навантаження.

Тестування на стабільність - протестуйте застосунок на стабільність, щоб налаштувати, як він працює під час різних ситуацій.

Тестування на надійність - протестуйте застосунок на надійність, щоб виконати програму, яка працює правильно під час різних ситуацій.

Аналіз результатів - проаналізуйте результати тестування, щоб зрозуміти, які функції працюють правильно, а які потребують корегування.

4.2 Тестування Web-застосунку за допомогою Python

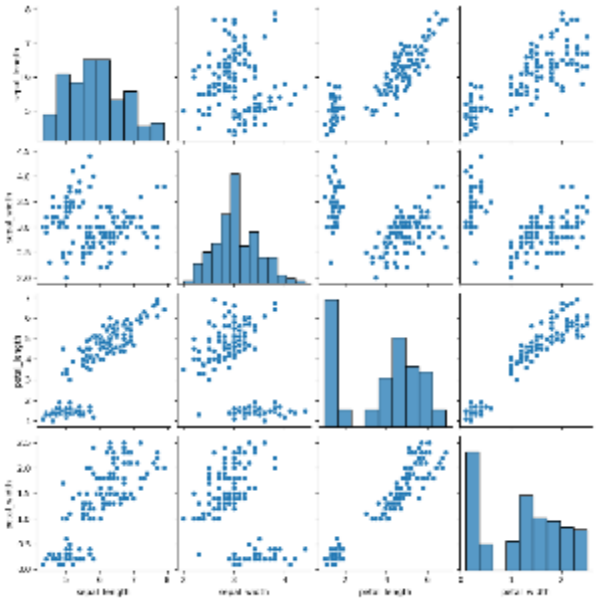


Рис. 4.1 Діаграма роботи програми

Нижче наведено приклад тестування коду на Python з використанням Selenium WebDriver:

Створіть інстанцію WebDriver driver = webdriver.Chrome()

Перейдіть на сторінку створення бюджету driver.get("http://example.com/budget-create")

Введіть дані для створення бюджету budget_name = "Test Budget" budget_category = "Test Category" budget_amount = "100.00" budget_currency = "USD" budget_period = "Monthly"

Введіть дані в поля driver.find_element_by_id("budget-name").send_keys(budget_name) driver.find_element_by_id("budget-category").send_keys(budget_category) driver.find_element_by_id("budget-amount").send_keys(budget_amount) driver.find_element_by_id("budget-currency").send_keys(budget_currency) driver.find_element_by_id("budget-period").send_keys(budget_period)

Створіть бюджет driver.find_element_by_id("create-budget").click()

Перейдіть на сторінку витрат driver.get("http://example.com/expenses")

Введіть дані для створення витрати expense_date = "2024-01-01" expense_category = "Test Category" expense_amount = "50.00" expense_method =

"Cash"

Введіть дані в поля driver.find_element_by_id("expense-date").send_keys(expense_date) driver.find_element_by_id("expense-category").send_keys(expense_category) driver.find_element_by_id("expense-amount").send_keys(expense_amount) driver.find_element_by_id("expense-method").send_keys(expense_method)

Створіть витрату driver.find_element_by_id("create-expense").click()

Перейдіть на сторінку фінансів driver.get("http://example.com/finance")

Перегляньте фінанси finance_data = driver.find_element_by_id("finance-data").text # Перевірте фінанси assert finance_data == "Test Budget: 100.00 USD, Test Category: 50.00 USD"

Закрийте інстанцію WebDriver driver.quit()

Цей код тестує створення бюджету та витрат, а також перегляд фінансів. Він використовує Selenium WebDriver для взаємодії з веб-інтерфейсом та очікує коректного результату.

ВИСНОВКИ

У рамках дипломної роботи було розроблено веб-застосунок для управління особистими фінансами, який дозволяє користувачам ефективно створювати, відстежувати та аналізувати свої бюджети та витрати. Робота охоплювала весь процес від вибору та обґрунтування використання технологій до реалізації конкретних функцій веб-додатку.

У процесі розробки було використано мову програмування Python та фреймворки React.js для клієнтської частини та Firebase для серверної частини та управління даними. Використання цих технологій забезпечило створення надійного та масштабованого веб-додатку.

Основні аспекти роботи включали інтеграцію з Firebase для аутентифікації користувачів і зберігання даних, реалізацію інтерфейсу користувача за допомогою React, що дозволило створити інтуїтивно зрозумілий та зручний дизайн. Також було розроблено кілька ключових функціональних можливостей, таких як створення та видалення бюджетів, реєстрація та вхід у систему, та введення та видалення витрат.

Додаток було тестовано з метою виявлення помилок та недоліків, що допомогло удосконалити його перед фінальним запуском. Тестування підтвердило функціональність всіх основних компонентів та забезпечило коректну роботу веб-додатку.

У підсумку, розроблений веб-додаток для управління фінансами демонструє значний потенціал для допомоги користувачам у плануванні та контролі їх фінансів.

Він забезпечує користувачам зручні інструменти для ведення фінансового обліку та може бути використаний як ефективний засіб для досягнення фінансової стабільності та цілей. У майбутньому можливе розширення функціональності веб-додатку та його адаптація до нових вимог ринку та потреб користувачів.

Робота пройшла апробацію на науково-технічних конференціях, за результатами апробації опубліковано тези доповідей:

1. Ромашкан Д.С., Аверічев І.М. Розробка WEB – застосунку для управління

особистими фінансами з використанням мови програмування Python. Сучасний стан та перспективи розвитку ІОТ : Матеріали V міжнародної всеукраїнської науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м.Київ. К.: ДУІКТ, 2024 С.43-45.

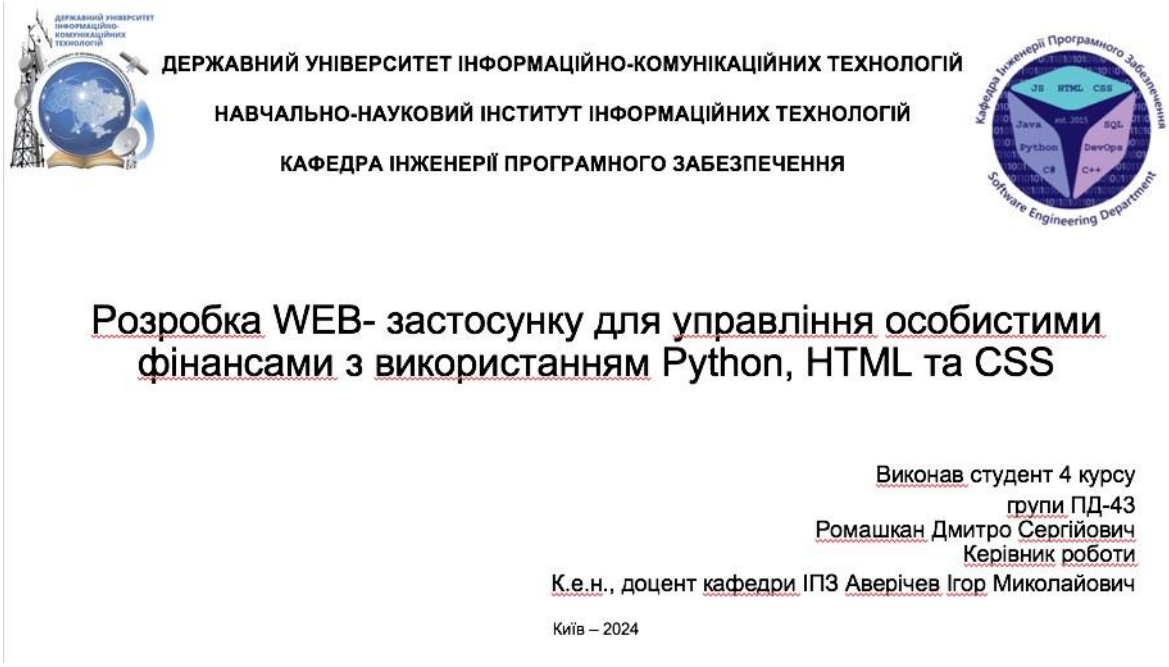
2. Ромашкан Д.С., Аверічев І.М. Класифікація вимог для управління особистими фінансами. Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м.Київ. К. : ДУІКТ, 2024. С.62-63.

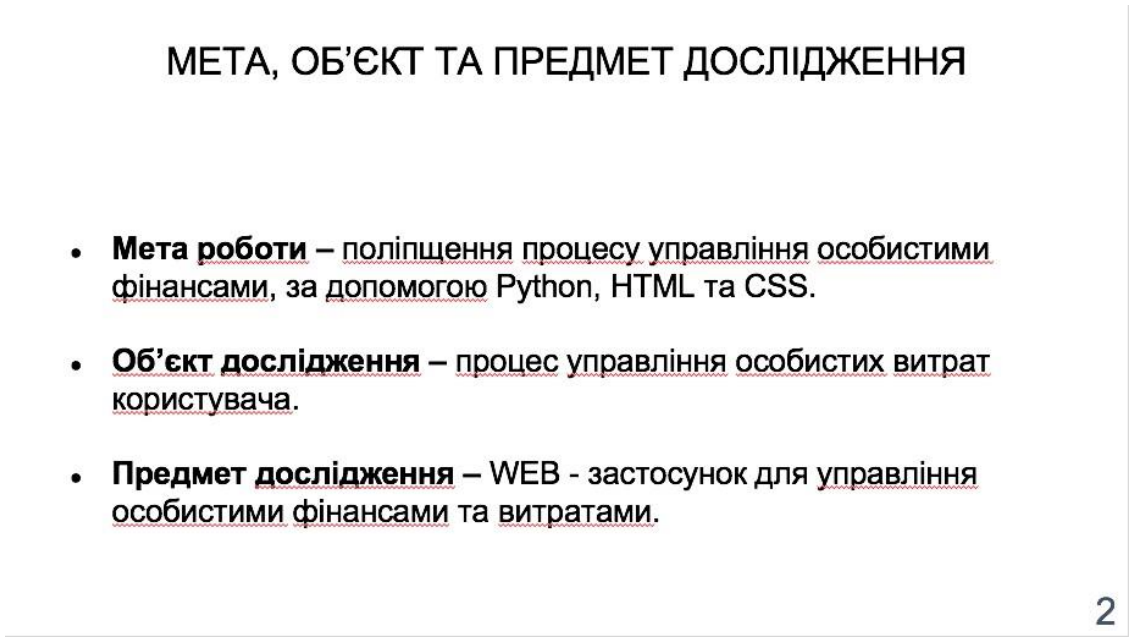
ПЕРЕЛІК ПОСИЛАНЬ

1. ReactJS [Електронний ресурс]. Доступно на: https://react.dev/
2. Getting started with React [Електронний ресурс]. Доступно на: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
3. TypeScript [Електронний ресурс]. Доступно на: https://www.typescriptlang.org/docs/
4. JavaScript [Електронний ресурс]. Доступно на: https://developer.mozilla.org/en-US/docs/Web/JavaScript
5. Firebase [Електронний ресурс]. Доступно на: https://firebase.google.com/docs
6. Firebase Storage JS SDK [Електронний ресурс]. Доступно на: https://firebase.google.com/docs/reference/js/v8/firebase.storage.Storage
7. Vite (Next generation frontend tooling) [Електронний ресурс]. Доступно на: https://vitejs.dev/guide/
8. HTML (HyperText Markup Language) [Електронний ресурс]. Доступно на: https://devdocs.io/html/
9. CSS (Cascading Style Sheets) [Електронний ресурс]. Доступно на: https://w3schoolsua.github.io/css/index.html#gsc.tab=0
10. Create React App (CRA) [Електронний ресурс]. Доступно на: https://create-react-app.dev/
11. Офіційна документація Python. Доступно на: https://docs.python.org/3/
12. Офіційна документація HTML. Доступно на: https://html.spec.whatwg.org/
13. Блог "Real Python". Доступно на: https://realpython.com/

14. Блог "CSS-Tricks". Доступно на: https://css-tricks.com/
15. Flask Web Development: Developing Web Applications with Python 1st Edition by Miguel Grinberg.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати можливості сучасних WEB - застосунків для розробки застосунку з управління особистими фінансами.
2. Визначити основні функціональні та нефункціональні вимоги до WEB - застосунку для управління особистими фінансами.
3. Розробити архітектуру для реалізації WEB - застосунку, включаючи використання Python, React.js, HTML, CSS та Firebase.
4. Спроектувати інтерфейс користувача, який є інтуїтивно зрозумілим та зручним для управління особистими фінансами, з використанням компонентної архітектури React.
5. Розробити WEB - застосунок для створення, відстеження, аналізу та видалення бюджетів , включаючи інтеграцію з Firebase для зберігання даних.
6. Провести модульне та інтеграційне тестування програмного забезпечення.

3

АНАЛІЗ АНАЛОГІВ

Функція / Програма	**Mint mobile**	**YNAB**	**RDS Manager**
Створення бюджетів	+	+	+
Відстеження витрат	+	+	+
Потрібна синхронізація з Credit Karma	+	+	-
Інтеграція з банками	+	+	+
Графічне відображення	+	+	+
Персоналізація інтерфейсу	Повна	Повна	Повна

4

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні:

1. Реєстрація та аутентифікація користувачів.
2. Додавання, редагування та видалення завдань .
3. Відображення списку завдань користувача через веб-інтерфейс.
4. Відображення зведених звітів про фінаносову активність.
5. Відправлення нагадувань про фінансові витрати.
6. Управління щоденними витратами з деталями про категорію, суму, дату та метод оплати.

Нефункціональні:

1. Швидкий відгук на запити користувача.
2. Захист персональних даних користувачів..
3. Інтуїтивно зрозумілий та зручний інтерфейс користувача різного віку та досвіду.

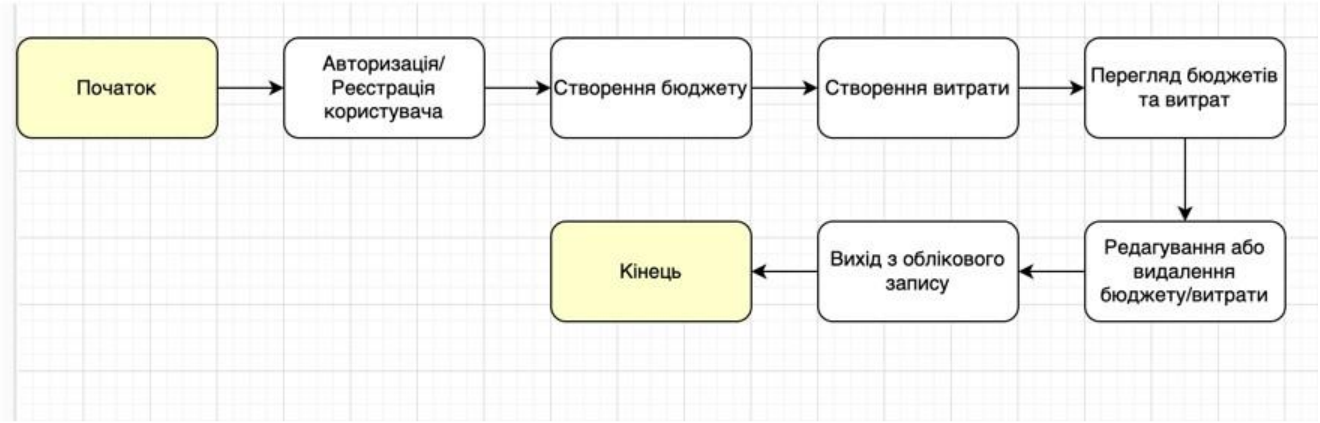
5

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

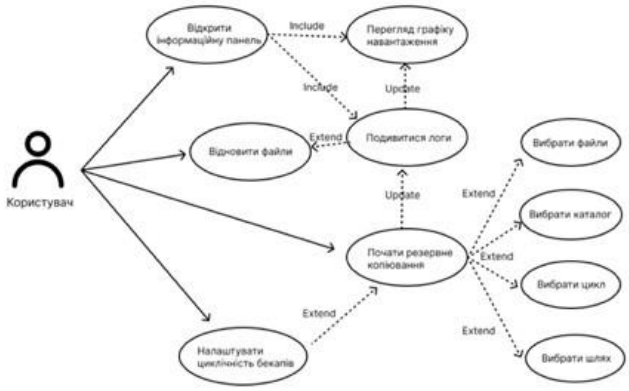
6

Діаграма робочого процесу WEB-застосунок для управління фінансами



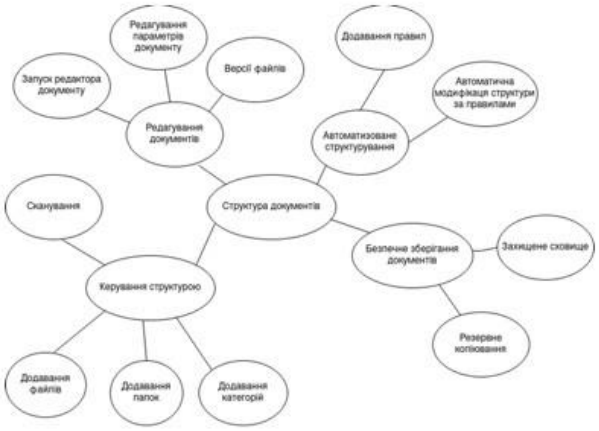
7

ДІАГРАМА ПРЕЦЕДЕНТІВ



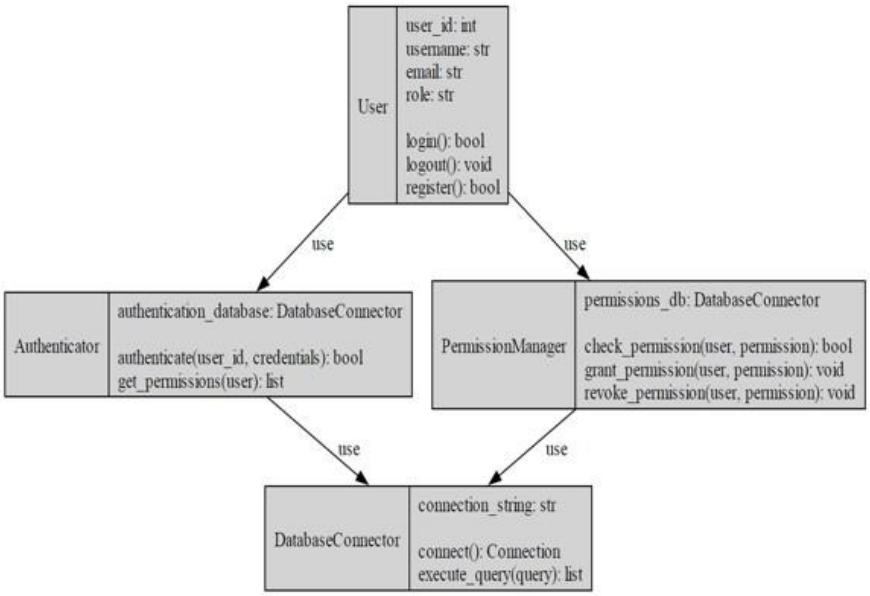
8

Діаграма предметної галузі



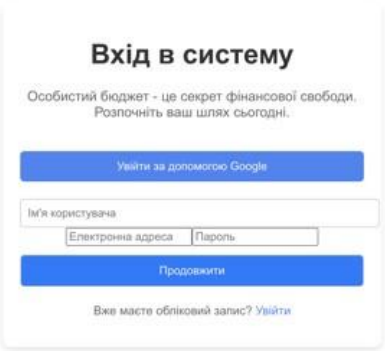
9

ДІАГРАМА КЛАСІВ



10

ЕКРАННІ ФОРМИ

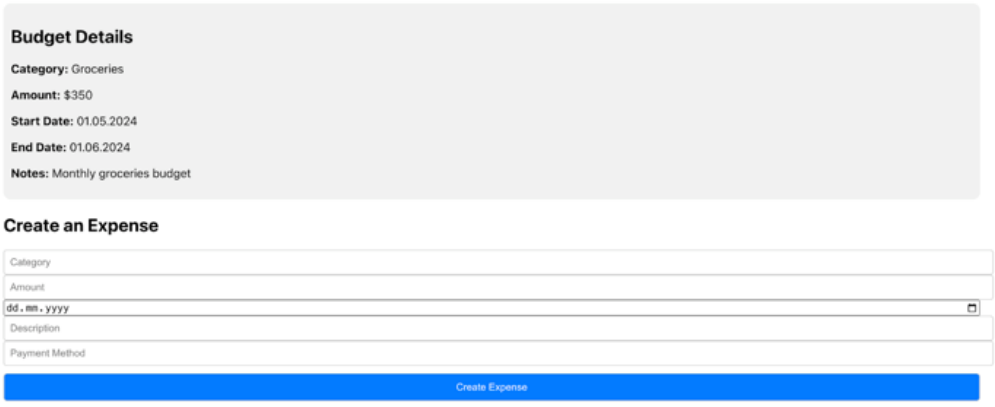


Екран входу до системи

Екран створення бюджету

11

ЕКРАННІ ФОРМИ



Екран після створення бюджету

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ромашкан Д.С., Аверічев І.М. Розробка WEB – застосунку для управління особистими фінансами з використанням мови програмування Python. Сучасний стан та перспективи розвитку IOT : Матеріали V міжнародної всеукраїнської науково-технічної конференції. Збірник тез. 18.04.2024, ДУІКТ, м.Київ. К.: ДУІКТ, 2024 С.43-45.

2. Ромашкан Д.С., Аверічев І.М. Класифікація вимог для управління особистими фінансами. Застосування програмного забезпечення в ІКТ: Матеріали всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м.Київ. К. : ДУІКТ, 2024. С.62-63.

13

ВИСНОВКИ

1. Проаналізовано можливості WEB-застосунку для управління особистими фінансами. Використано технології, які найкраще підходять для створення WEB-застосунку для контролю особистих фінансів, зокрема, HTML, CSS, Python, React.js та Firebase для клієнтської частини .
2. Визначено функціональні та нефункціональні вимоги до WEB- застосунку для управління особистими фінансами.
3. Спроектовано архітектуру WEB- застосунку. Описано основні компоненти системи, включно з модулями для управління бюджетами, витратами, а також звітами.
4. Розроблено користувацький інтерфейс. Веб-інтерфейс додатку був розроблений з урахуванням зручності та інтуїтивно зрозумілого навігаційного досвіду..
5. Проведено модульне та інтеграційне тестування програмного забезпечення.

14

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
from telegram import Update, Bot
from telegram.ext import Updater, CommandHandler, CallbackContext, MessageHandler,
Filters
import logging
import pymysql.cursors

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
                    level=logging.INFO)
logger = logging.getLogger(__name__)

class TaskManager:
    def __init__(self, connection):
        self.connection = connection

    def add_task(self, user_id, description, deadline):
        with self.connection.cursor() as cursor:
            sql = "INSERT INTO `tasks` (`user_id`, `description`, `deadline`) VALUES (%s, %s,
%s)"
            cursor.execute(sql, (user_id, description, deadline))
        self.connection.commit()
    def get_tasks(self, user_id):
        with self.connection.cursor() as cursor:
            sql = "SELECT `id`, `description`, `deadline` FROM `tasks` WHERE `user_id`=%s"
            cursor.execute(sql, (user_id,))
            result = cursor.fetchall()
            return result

    def edit_task(self, task_id, new_description, new_deadline):
        with self.connection.cursor() as cursor:
            sql = "UPDATE `tasks` SET `description`=%s, `deadline`=%s WHERE `id`=%s"
            cursor.execute(sql, (new_description, new_deadline, task_id))
        self.connection.commit()

    def delete_task(self, task_id):
        with self.connection.cursor() as cursor:
            sql = "DELETE FROM `tasks` WHERE `id`=%s"
            cursor.execute(sql, (task_id,))
        self.connection.commit()

class UserManager:
    def __init__(self, connection):
        self.connection = connection

    def register_user(self, user_id, name, email, role):
```

```
        with self.connection.cursor() as cursor:
            sql = "INSERT INTO `users` (`user_id`, `name`, `email`, `role`) VALUES (%s, %s,
%s, %s)"
            cursor.execute(sql, (user_id, name, email, role))
        self.connection.commit()

    def user_exists(self, user_id):
        with self.connection.cursor() as cursor:
            sql = "SELECT `id` FROM `users` WHERE `user_id`=%s"
            cursor.execute(sql, (user_id,))
            result = cursor.fetchone()
            return result is not None

connection = pymysql.connect(host='localhost',
                             user='*****',
                             password='*****',
                             database='bd_demo_pr',
                             cursorclass=pymysql.cursors.DictCursor)

task_manager = TaskManager(connection)
user_manager = UserManager(connection)

def start(update: Update, context: CallbackContext) -> None:
    update.message.reply_text('Привіт! Я бот, який допомагає управляти завданнями.
Введіть /register для реєстрації.')

def register(update: Update, context: CallbackContext) -> None:
    args = update.message.text.split(maxsplit=3)
    if len(args) >= 4:
        user_id = update.effective_user.id
        name = args[1]
        email = args[2]
        role = args[3]
        if user_manager.user_exists(user_id):
            update.message.reply_text('Ви вже зареєстровані.')
        else:
            user_manager.register_user(user_id, name, email, role)
            update.message.reply_text('Реєстрація успішна!')
    else:
        update.message.reply_text('Використання: /register <ім\'я> <електронна пошта>
<роль>')

def add_task(update: Update, context: CallbackContext) -> None:
    args = update.message.text.split(maxsplit=2)
    if len(args) >= 3:
        task_manager.add_task(update.effective_user.id, args[1], args[2])
        update.message.reply_text('Завдання додано!')
    else:
```

```
        update.message.reply_text('Використання: /add <опис> <дедлайн>')

def edit_task(update: Update, context: CallbackContext) -> None:
    args = update.message.text.split(maxsplit=3)
    if len(args) >= 4:
        task_id = int(args[1])
        new_description = args[2]
        new_deadline = args[3]
        task_manager.edit_task(task_id, new_description, new_deadline)
        update.message.reply_text('Завдання відредаговано!')
    else:
        update.message.reply_text('Використання: /edit <id завдання> <новий опис> <новий
дедлайн>')

def delete_task(update: Update, context: CallbackContext) -> None:
    args = update.message.text.split(maxsplit=1)
    if len(args) == 2:
        task_id = int(args[1])
        task_manager.delete_task(task_id)
        update.message.reply_text('Завдання видалено!')
    else:
        update.message.reply_text('Використання: /delete <id завдання>')

def show_tasks(update: Update, context: CallbackContext) -> None:
    tasks = task_manager.get_tasks(update.effective_user.id)
    if tasks:
        message = '\n'.join([f'ID: {task["id"]}, Опис: {task["description"]}, Дедлайн:
{task["deadline"]}' for task in tasks])
        update.message.reply_text(message)
    else:
        update.message.reply_text('У вас немає активних завдань.')

import datetime

class NotificationManager:
    def __init__(self, connection, bot):
        self.connection = connection
        self.bot = bot

    def send_reminders(self):
        now = datetime.datetime.now()
        with self.connection.cursor() as cursor:
            sql = "SELECT `user_id`, `description`, `deadline` FROM `tasks` WHERE `deadline`
> %s AND `deadline` <= %s"
            cursor.execute(sql, (now, now + datetime.timedelta(days=1)))
            tasks = cursor.fetchall()

            for task in tasks:
```

```
            user_id = task['user_id']
            description = task['description']
            deadline = task['deadline']
            message = f'Нагадування! Завдання: {description}, дедлайн: {deadline}'
            self.bot.send_message(chat_id=user_id, text=message)

def detailed_task(update: Update, context: CallbackContext) -> None:
    args = update.message.text.split(maxsplit=1)
    if len(args) == 2:
        task_id = int(args[1])
        with connection.cursor() as cursor:
            sql = "SELECT `description`, `deadline` FROM `tasks` WHERE `id`=%s"
            cursor.execute(sql, (task_id,))
            task = cursor.fetchone()
            if task:
                update.message.reply_text(f'Опис: {task["description"]}\nДедлайн:
{task["deadline"]}')
            else:
                update.message.reply_text('Завдання не знайдено.')
    else:
        update.message.reply_text('Використання: /detail <id завдання>')

def list_users(update: Update, context: CallbackContext) -> None:
    with connection.cursor() as cursor:
        sql = "SELECT `user_id`, `name`, `email`, `role` FROM `users`"
        cursor.execute(sql)
        users = cursor.fetchall()
        if users:
            message = '\n'.join([f'ID: {user["user_id"]}, Ім\'я: {user["name"]}, Електронна
пошта: {user["email"]}, Роль: {user["role"]}' for user in users])
            update.message.reply_text(message)
        else:
            update.message.reply_text('Жодного користувача не знайдено.')

def notify_users(bot: Bot, job_context):
    notification_manager = NotificationManager(connection, bot)
    notification_manager.send_reminders()

def main() -> None:
    bot_token = '*****'
    bot = Bot(token=bot_token)
    updater = Updater(bot=bot, use_context=True)

    dispatcher = updater.dispatcher
    dispatcher.add_handler(CommandHandler("start", start))
    dispatcher.add_handler(CommandHandler("register", register))
    dispatcher.add_handler(CommandHandler("add", add_task))
    dispatcher.add_handler(CommandHandler("edit", edit_task))
```

```
    dispatcher.add_handler(CommandHandler("delete", delete_task))
    dispatcher.add_handler(CommandHandler("tasks", show_tasks))
    dispatcher.add_handler(CommandHandler("detail", detailed_task))
    dispatcher.add_handler(CommandHandler("list_users", list_users))
    job_queue = updater.job_queue
    job_queue.run_daily(notify_users, time=datetime.time(hour=9, minute=0, second=0))

    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()
```