МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему: «**ПРОЄКТУВАННЯ СИСТЕМИ ОНЛАЙН-ПЛАНУВАННЯ ОСОБИСТОГО ТА СІМЕЙНОГО ЧАСУ**»

здобувача Щетініної Марії Дмитрівни
_____
(ПІБ) *(Підпис)*

Науковий керівник:
д.т.н., проф.
_____Шевченко К. Л.

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти
завідувач кафедри:
к.е.н., доцент
_____Тішков Б.О.

Київ 2025

Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

ПОГОДЖЕНО:
Керівник проєктної групи(гарант)
освітньо-професійної програми
_____Помазун О.М.
"____" _____2025 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____Тішков Б.О.
"____" _____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувачки вищої освіти ***Щетініної Марії Дмитрівни***

очної (денної) форми навчання

на підготовку кваліфікаційної бакалаврської роботи
на тему: «*Проєктування системи онлайн планування особистого та сімейного часу»*

Тему затверджено наказом ректора Університету від « *7* » *березня 2025* р. № *466- ст.*

Кваліфікаційна бакалаврська робота виконується на матеріалах
аналізу існуючих рішень у сфері планування часу; порівняльного аналізу технологій та підходів до розробки веб-додатків

План кваліфікаційної бакалаврської роботи
Розділ І Аналіз теоретичних основ побудови систем он-лайн планування

Розділ ІІ Розробка та проєктування системи он-лайн планування

Розділ ІІІ Реалізація та тестування системи

Об'єкт дослідження: процеси планування та організації особистого та сімейного часу з використанням інформаційних технологій

Предмет дослідження: методи та засоби розробки веб-орієнтованої системи планування часу з інтеграцією зовнішніх сервісів та розширеним функціоналом

Мета кваліфікаційної бакалаврської роботи: полягає у розробці системи онлайн планування особистого та сімейного часу з інтеграцією сервісів Google та реалізацією додаткових функцій.

Конкретні завдання, які здобувачка повинна виконати для досягнення поставленої мети:

У розділі I Проаналізувати існуючі рішення у сфері планування особистого та сімейного планування часу. Визначити особливості інтеграції з сервісами Google (Google Account, Google Calendar). Визначити методології та принципи проектування інтерфейсів користувача для систем планування

У розділі II Розробити технічну архітектуру системи, описати взаємодію компонентів. Розробити моделі даних та структури бази даних. Розробити функціональну специфікацію системи. .

У розділі III Реалізувати процедури авторизації через Google та управління особистими/сімейними акаунтами. Інтегрувати систему з Google Calendar та реалізувати функціонал управління подіями. Розробити та реалізувати додаткові функції (нотатки, вішлісти з приватним/публічним доступом, списки покупок)

Завдання підготував
науковий керівник **_____ Шевченко Костянтин Леонідович**

«_» червня 2025 р.

Завдання одержав
здобувач **_____ Щетініна Марія Дмитрівна _____**

«_» червня 2025 р..

Відгук
про кваліфікаційну бакалаврську роботу
здобувачки навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми
«Комп'ютерні науки»
_____ ***Щетініної Марії Дмитрівни*** _____

на тему
«Проєктування системи онлайн планування особистого та сімейного часу»

1. Актуальність теми: враховуючі особливості сучасного динамічного життя та викликаною цим потребою ефективного управління особистим та сімейним часом, тема роботи є актуальною.
2. Позитивні риси кваліфікаційної бакалаврської роботи: в роботі проведено: ґрунтовний аналіз існуючих рішень у сфері планування часу; порівняльний аналіз технологій та підходів до розробки веб-додатків; моделювання архітектури системи та структури бази даних.
3. Наявність самостійних розробок автора: автором самостійно виконані всі поставлені в індивідуальному завданні задачі.
4. Цінність теоретичних висновків та практичних рекомендацій: розроблена система може бути використана для оптимізації процесів особистого та сімейного планування. Зручний інструментарій для управління подіями, збереження корисної інформації та відстеження повсякденних завдань сприяє зниженню стресу та підвищенню продуктивності.
5. Наявність недоліків: в роботі зустрічаються незначні відхилення у форматуванні тексту, суттєвих змістовних зауважень немає.
6. Загальна оцінка кваліфікаційної бакалаврської роботи та її допущення до захисту перед ЕК: кваліфікаційна бакалаврська робота на тему «Проєктування системи онлайн планування особистого та сімейного часу» відповідає встановленим вимогам та рекомендується до захисту, а її автор Щетініна Марія Дмитрівна заслуговує на присвоєння освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 122 «Комп’ютерні науки».

Науковий керівник
д.т.н., професор кафедри
інформаційних систем в економіці Шевченко К.Л.
(підпис)

“___”_____ 20__ р

Рецензія

на кваліфікаційну бакалаврську роботу здобувача вищої освіти

Щетініної Марії Дмитрівни

(прізвище, ім'я, по батькові)

тема: «**Проєктування системи онлайн-планування особистого та сімейного часу**»

Актуальність теми кваліфікаційної бакалаврської роботи і доцільність її розроблення. Тайм –менеджмент є актуальним напрямом наукових досліджень, оволодіння засадами якого є передумовою успіху людини в повсякденному та професійному житті. Цифрові формати більшості додатків довели свою ефективність. Тому проєктування системи онлайн-планування особистого та сімейного часу може бути вигідним проєктом та має комерційну значущість.

Якість проведеного дослідження Дослідження проведено на належному науково-методичному рівні. Автор проаналізував 30 сучасних наукових джерел, проаналізував існуючі рішення у сфері особистого та сімейного планування часу, врахував необхідність інтеграції з сервісами Google (Google Account, Google Calendar) та авторизацію через Google та управління особистими/сімейними акаунтами.

Позитивні риси кваліфікаційної бакалаврської роботи. Автору кваліфікаційної бакалаврської роботи вдалося визначити оптимальний набір функцій та використати сучасні підходи для забезпечення максимальної ефективності користування системою онлайн-планування особистого та сімейного часу. Інтеграція з сервісами Google, зокрема Google Account та Google Calendar, забезпечує надійну основу для авторизації користувачів та синхронізації даних календаря, водночас розширюючи функціональність системи через доступ до екосистеми Google, що має високу прикладну цінність. Інтерфейс системи спроєктовано за принципами User-Centered Design, що максимально спрощує для користувачів процес планування.

Автором спроєктовано технічну архітектуру системи онлайн-планування часу, що базується на патерні MVC. В її основі лежить мікрофреймворк Flask, який виконує роль контролера, система ORM SQLAlchemy для роботи з моделлю даних, та шаблонізатор Jinja2 для представлення (view). Цей підхід забезпечує модульність і легкість підтримки системи.

Зауваження. Не виявлено.

Практична значимість висновків і рекомендацій. Безпека даних користувача забезпечується завдяки автентифікації через Google, система не зберігає паролів користувачів. Профіль користувача включає його ім'я та електронну адресу, а також надається доступ до функцій системи: управління сімейним акаунтом, створення подій, нотаток, вішлістів та списків покупок. Управління сімейними акаунтами є унікальною особливістю системи, що дозволяє користувачам створювати сімейні групи та запрошувати до них інших користувачів для спільного планування та координації.

Загальна оцінка кваліфікаційної бакалаврської роботи та його допущення до захисту перед ЕК. Кваліфікаційну бакалаврську роботу допущено до захисту з оцінкою **«відмінно».**

Рецензент:

Рудакова Світлана Григорівна,
кандидат
технічних наук,
доцент, доцент кафедри
соціоекономіки та управління
персоналом
КНЕУ ім. В. Гетьмана

Підпис М. Д. Щетініної
ЗАСВІДЧУЮ
Начальник відділу кадрів
Київський національний економічний університет імені Вадима Гетьмана
Н. І. Терепецук
20___р.

АНОТАЦІЯ

Кваліфікаційна бакалаврська робота присвячена проектуванню та розробці системи онлайн планування особистого та сімейного часу з інтеграцією сервісів Google. У роботі проведено аналіз існуючих рішень у сфері планування часу, досліджено особливості інтеграції з Google Account та Google Calendar, вивчено методології проектування інтерфейсів користувача. Розроблена система включає авторизацію через Google, функціонал управління особистими та сімейними акаунтами, створення та синхронізацію подій між членами родини, а також додаткові функції – нотатки, вішлісти з можливістю налаштування приватності та списки покупок. Система реалізована на базі веб-фреймворку Flask з використанням SQLAlchemy для роботи з базою даних та OAuth 2.0 для інтеграції з сервісами Google. Розроблене рішення дозволяє ефективно організовувати особистий та сімейний час, координувати плани між членами родини та управляти повсякденними завдання

РЕФЕРАТ

Кваліфікаційна бакалаврська робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи – 72 сторінок. Робота містить 19 рисунків, 1 таблицю, список використаних джерел із 30 найменувань та 3 додатки.

У вступі обґрунтовано актуальність теми, визначено мету, завдання, об'єкт та предмет дослідження, а також розкрито практичне значення отриманих результатів та перелічено використані методи дослідження.

У першому розділі "Характеристика та аналіз предметної галузі" проведено аналіз існуючих рішень у сфері особистого та сімейного планування часу , досліджено особливості інтеграції з сервісами Google, зокрема Google Account та Google Calendar , та вивчено методології і принципи проектування інтерфейсів користувача для систем планування.

У другому розділі "Розробка вимог і моделювання інформаційної системи" розроблено та специфіковано вимоги до інформаційної системи , виконано постановку задачі та змодельовано архітектуру програмних компонентів і структуру даних за допомогою UML-діаграм.

У третьому розділі "Проектування та реалізація компонентів підсистеми" детально описано інформаційне, технічне та програмне забезпечення. Продемонстровано реалізацію ключових функцій: авторизації через Google та управління особистими/сімейними акаунтами , інтеграції з Google Calendar для створення та синхронізації подій , а також розробку додаткових функцій, таких як нотатки, вішлісти з налаштуванням приватності та списки покупок. Результати реалізації підтверджено скріншотами розробленої системи.

У висновках підсумовано основні результати кваліфікаційної роботи, визначено практичне значення розробленої системи та окреслено потенційні напрямки для її подальшого розвитку.

Ключові слова: СИСТЕМА ПЛАНУВАННЯ ЧАСУ, ВЕБ-РОЗРОБКА, GOOGLE CALENDAR, OAUTH 2.0, FLASK, SQLALCHEMY, СІМЕЙНИЙ АКАУНТ, СИНХРОНІЗАЦІЯ ПОДІЙ, ВІШЛІСТИ, СПИСКИ ПОКУПО

ЗМІСТ

ВСТУП

Актуальність теми дослідження обумовлена зростаючою потребою в ефективному управлінні особистим та сімейним часом в умовах сучасного динамічного життя. Цифровізація повсякденних процесів та поширення мобільних пристроїв створюють підґрунтя для впровадження інтелектуальних систем планування, що дозволяють оптимізувати розподіл часових ресурсів, координувати дії між членами родини та знижувати когнітивне навантаження, пов'язане з управлінням численними зобов'язаннями. Традиційні методи планування, такі як паперові календарі та нотатки, поступаються місцем цифровим рішенням, які забезпечують доступність інформації з будь-якого пристрою, підтримують автоматичну синхронізацію даних та надають додаткові функціональні можливості. Інтеграція з існуючими екосистемами, такими як Google, дозволяє створювати комплексні рішення, що органічно доповнюють звичний цифровий досвід користувачів та відповідають їхнім потребам у зручному та гнучкому плануванні.

Мета і завдання дослідження. Мета кваліфікаційної бакалаврської роботи полягає у розробці системи онлайн планування особистого та сімейного часу з інтеграцією сервісів Google та реалізацією додаткових функцій для підвищення ефективності організації повсякденного життя. Для досягнення поставленої мети визначено такі завдання: проаналізувати існуючі рішення у сфері особистого та сімейного планування часу; дослідити особливості інтеграції з сервісами Google (Google Account, Google Calendar); вивчити методології та принципи проектування інтерфейсів користувача для систем планування; розробити технічну архітектуру системи та опис взаємодії компонентів; спроектувати модель даних та структуру бази даних; розробити функціональну специфікацію системи; реалізувати авторизацію через Google та управління особистими/сімейними акаунтами; інтегрувати Google Calendar та реалізувати функціонал управління подіями; розробити та впровадити додаткові функції (нотатки, вішлісти з приватним/публічним доступом, списки покупок).

Об'єкт дослідження: процеси планування та організації особистого та сімейного часу з використанням інформаційних технологій.

Предмет дослідження: методи та засоби розробки веб-орієнтованої системи планування часу з інтеграцією зовнішніх сервісів та розширеним функціоналом.

Практичне значення роботи. Розроблена система може бути використана для оптимізації процесів особистого та сімейного планування, підвищення ефективності розподілу часу та покращення координації дій між членами родини. Система надає зручний інструментарій для управління подіями, зберігання корисної інформації та відстеження повсякденних завдань, що сприяє зниженню стресу та підвищенню продуктивності. Практичне значення також полягає в демонстрації методів інтеграції з API Google та реалізації механізмів авторизації через OAuth 2.0, що може бути використано при розробці інших веб-додатків з подібними вимогами.

Методи дослідження. У роботі використано комплекс методів, зокрема: аналіз існуючих рішень у сфері планування часу; порівняльний аналіз технологій та підходів до розробки веб-додатків; моделювання архітектури системи та структури бази даних; об'єктно-орієнтоване проектування при розробці програмних компонентів; експериментальний метод при тестуванні функціональності системи. Для розробки системи застосовано сучасні веб-технології, включаючи фреймворк Flask для серверної частини, SQLAlchemy для роботи з базою даних, OAuth 2.0 для автентифікації та Google Calendar API для управління подіями календаря.

РОЗДІЛ 1. ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1Характеристика предметної галузі та об'єкта дослідження

Інтеграція із сервісами Google представляє собою комплексний процес взаємодії між сторонніми додатками та екосистемою Google, що дозволяє розширити функціональність додатків шляхом використання існуючих даних користувача та технічних можливостей платформи Google. В контексті систем планування особистого та сімейного часу, інтеграція з Google Account та Google Calendar забезпечує суттєві переваги як для розробників, так і для кінцевих користувачів. Google Account функціонує як централізована система аутентифікації та авторизації, що дозволяє користувачам отримувати доступ до різноманітних сервісів за допомогою єдиного облікового запису, уникаючи необхідності створювати та запам'ятовувати окремі облікові дані для кожного сервісу. Google Calendar, у свою чергу, є потужним інструментом управління розкладом, що пропонує розширені можливості для створення, редагування та синхронізації подій, підтримує спільний доступ до календарів та інтегрується з широким спектром інших сервісів Google. Ці компоненти екосистеми Google становлять фундаментальну основу для розробки сучасних систем планування часу з розширеними можливостями інтеграції та синхронізації [6].

Технічна реалізація інтеграції з Google Account здійснюється за допомогою протоколу OAuth 2.0, який є стандартом індустрії для делегованої авторизації. OAuth 2.0 визначається як протокол авторизації, що дозволяє стороннім додаткам отримувати обмежений доступ до облікового запису користувача на сервері ресурсів (у даному випадку — Google) без необхідності передачі облікових даних між сервісами. У процесі OAuth-авторизації користувач перенаправляється на сторінку автентифікації Google, де вводить свої облікові дані та надає додатку дозвіл на доступ до конкретних ресурсів свого облікового запису. Після успішної автентифікації та авторизації, Google генерує токен

доступу (access token) та, опціонально, токен оновлення (refresh token), які додаток використовує для взаємодії з API Google від імені користувача. Така архітектура забезпечує високий рівень безпеки, оскільки облікові дані користувача не покидають межі довіреного домену Google, а додаток отримує лише обмежений доступ до конкретних ресурсів, який може бути відкликаний користувачем у будь-який момент через панель керування безпекою Google Account.

Інтеграція з Google Calendar здійснюється через Google Calendar API — програмний інтерфейс, що надає широкі можливості для взаємодії з календарями користувача. Calendar API підтримує повний спектр операцій CRUD (Create, Read, Update, Delete) для управління календарями, подіями, нагадуваннями та налаштуваннями доступу. Особливо значущою для систем сімейного планування є функціональність спільного доступу до календарів, яка дозволяє користувачам надавати різні рівні доступу (від перегляду до повного редагування) іншим користувачам Google. Calendar API також підтримує push-нотифікації, що дозволяють додаткам отримувати миттєві повідомлення про зміни в календарях без необхідності періодичного опитування серверів Google. Ця можливість є особливо корисною для забезпечення синхронності даних у режимі реального часу між різними користувачами сімейного календаря, що істотно підвищує ефективність координації планів та мінімізує ризики виникнення конфліктів у розкладі [7].

Розробка інтерфейсу взаємодії з Google Calendar API вимагає ретельного планування архітектури та оптимізації запитів для забезпечення ефективного використання ресурсів та дотримання лімітів використання API. Google Calendar API має квоти та обмеження на кількість запитів, які можуть бути виконані від імені одного користувача або проекту протягом певного періоду часу. Ці обмеження необхідно враховувати при проектуванні логіки синхронізації даних між системою планування та Google Calendar. Оптимальним підходом є використання інкрементальної синхронізації, що передбачає завантаження лише тих даних, які зазнали змін з моменту останньої синхронізації, а не повне

перезавантаження всіх даних календаря. Такий підхід можна реалізувати за допомогою механізму syncToken, що надається Calendar API, або шляхом відстеження часових міток останніх змін та використання їх для фільтрації запитів. Додатково, для мінімізації кількості необхідних запитів, варто використовувати пакетну обробку операцій, яка дозволяє виконувати кілька змін у календарі за допомогою одного запиту до API, що суттєво підвищує ефективність взаємодії з сервісами Google.

Забезпечення безпеки та конфіденційності даних користувачів є фундаментальним аспектом інтеграції з сервісами Google. Google надає розробникам потужні інструменти для управління доступом до даних користувачів через механізм областей доступу (scopes). Область доступу визначається як рівень дозволу, що надається додатку для взаємодії з конкретним типом даних або сервісом Google. У контексті інтеграції з Google Calendar, розробники можуть запитувати різні рівні доступу, від обмеженого доступу лише для читання даних календаря до повного доступу для створення та редагування подій у всіх календарях користувача. Відповідно до принципу мінімальних привілеїв, який є наріжним каменем безпечної розробки, система планування часу повинна запитувати лише ті області доступу, які необхідні для реалізації її функціоналу. Наприклад, якщо система не потребує доступу до електронної пошти користувача, вона не повинна запитувати відповідні дозволи. Такий підхід не лише підвищує безпеку системи, але й збільшує довіру користувачів, оскільки вони бачать, що додаток запитує лише обґрунтовані дозволи [8].

Специфіка інтеграції з Google Account для створення сімейних акаунтів потребує додаткового розгляду, оскільки Google не надає прямого API для управління сімейними групами. Натомість, розробникам необхідно реалізувати власну логіку управління сімейними зв'язками в рамках своєї системи, використовуючи Google Account лише як механізм автентифікації користувачів. Сімейний акаунт у системі планування часу може бути реалізований як окрема сутність, до якої можуть бути прив'язані кілька індивідуальних облікових записів

користувачів, автентифікованих через Google Account. Процес додавання члена родини до сімейного акаунту може включати генерацію унікального запрошення або коду, який новий користувач вводить після автентифікації через Google. Альтернативний підхід передбачає використання електронної пошти Google як ідентифікатора для надсилання запрошень та підтвердження приналежності до сімейної групи. У будь-якому випадку, система повинна забезпечувати чітке розмежування ролей та прав доступу між адміністратором сімейного акаунту (який може додавати або видаляти членів) та звичайними користувачами, що забезпечує належний рівень контролю та безпеки управління сімейним акаунтом.

Розробка механізму синхронізації подій між особистими та сімейними календарями вимагає ретельної архітектурної проробки для забезпечення консистентності даних та оптимального використання API Google Calendar. Центральним елементом такої архітектури може бути серверна компонента, що виступає посередником між клієнтськими додатками та Google Calendar API, координуючи процеси синхронізації та вирішуючи потенційні конфлікти. Коли користувач створює подію в сімейному календарі та призначає її для конкретних членів сім'ї, система повинна не лише додати цю подію до сімейного календаря в Google, але й забезпечити її коректне відображення в особистих календарях відповідних користувачів. Технічно це можна реалізувати кількома шляхами: через створення дубліката події в особистому календарі кожного користувача, через надання доступу до конкретної події сімейного календаря або через використання механізму спільних календарів Google. Кожен з цих підходів має свої переваги та обмеження, і вибір оптимального рішення залежить від конкретних вимог до системи та особливостей її використання. Незалежно від обраного підходу, система повинна забезпечувати двосторонню синхронізацію, щоб зміни, внесені в подію через Google Calendar (наприклад, через веб-інтерфейс або мобільний додаток Google), коректно відображалися в системі планування часу і навпаки [6].

Інтеграція додаткових функцій, таких як нотатки, вішлісти та списки покупок, з екосистемою Google представляє собою окремий технічний виклик, оскільки ці типи даних не мають прямих аналогів серед стандартних сервісів Google, доступних через публічні API. Для реалізації цих функцій може бути використано кілька підходів, зокрема: створення власної бази даних для зберігання цих типів даних з прив'язкою до облікових записів Google користувачів; використання Google Drive API для зберігання структурованих даних у форматі JSON або іншому форматі; адаптація Google Tasks API для зберігання списків завдань, що можуть виконувати роль вішлістів або списків покупок. Кожен з цих підходів має свої особливості щодо складності реалізації, можливостей офлайн-доступу, синхронізації між пристроями та інтеграції з іншими компонентами системи. При проектуванні механізмів управління доступом до цих типів даних (наприклад, для реалізації приватних та публічних вішлістів) необхідно розробити чітку модель прав доступу та забезпечити її послідовне застосування незалежно від способу зберігання даних. Така модель повинна враховувати різні сценарії використання, включаючи перегляд, редагування, додавання та видалення елементів, а також можливість зміни статусу доступу з приватного на публічний і навпаки [5].

1.2Аналіз літературних джерел та практичного досвіду використання ІС і технологій в предметній галузі

У сучасному динамічному світі ефективне управління часом стало невід'ємною частиною повсякденного життя. Особисте та сімейне планування часу розглядається як систематичний процес організації та розподілу часових ресурсів для досягнення конкретних цілей, оптимізації продуктивності та забезпечення балансу між професійними обов'язками та особистим життям. За останнє десятиліття спостерігається стрімке зростання ринку цифрових рішень для планування часу, що пропонують різноманітний функціонал для

задоволення потреб як окремих користувачів, так і сімейних груп. Ці інструменти еволюціонували від простих календарів та списків завдань до комплексних екосистем з інтегрованими функціями синхронізації, сповіщень та аналітики використання часу, демонструючи трансформацію підходів до тайм-менеджменту в цифрову епоху [1, с. 121].

Серед основних категорій існуючих рішень у сфері планування часу можна виділити: персональні календарі та органайзери (Google Calendar, Apple Calendar), спеціалізовані додатки для планування завдань (Todoist, Asana, Trello), комплексні системи сімейного планування (Cozi Family Organizer, FamilyWall) та інтегровані платформи продуктивності (Microsoft 365, Notion). Кожна з цих категорій має свої особливості та підходи до організації часу. Персональні календарі зосереджені на візуалізації розкладу в різних часових масштабах (день, тиждень, місяць) і зазвичай пропонують базовий функціонал для створення повторюваних подій, встановлення нагадувань та інтеграції з іншими сервісами. Додатки для планування завдань акцентують увагу на методологіях тайм-менеджменту, таких як GTD (Getting Things Done) або метод Pomodoro, і надають інструменти для пріоритизації, категоризації та відстеження виконання завдань, що суттєво підвищує персональну продуктивність.

Системи сімейного планування становлять особливий інтерес у контексті дослідження, оскільки вони спрямовані на вирішення специфічних проблем координації та синхронізації планів між кількома членами родини. Згідно з дослідженнями в галузі соціології та психології сім'ї, ефективна комунікація та координація планів є фундаментальними факторами, що впливають на якість сімейних відносин та загальне благополуччя родини. Сімейне планування часу визначається як колективний процес узгодження індивідуальних розкладів, розподілу відповідальності та спільного прийняття рішень щодо використання часових ресурсів сім'ї. Додатки на кшталт Cozi Family Organizer, FamilyWall та TimeTree пропонують спеціалізовані функції для спільного доступу до календарів, створення списків покупок, обміну нотатками та координації

домашніх обов'язків, що значно спрощує організацію повсякденного життя родини та сприяє формуванню здорових сімейних рутин [2, с. 47].

Інтеграція з екосистемами великих технологічних компаній, таких як Google, Apple та Microsoft, стала визначальною тенденцією в розвитку сучасних систем планування часу. Ця інтеграція забезпечує безперебійний обмін даними між різними пристроями та платформами, що є суттєвою перевагою в умовах мультиплатформенного використання. Google Calendar, наприклад, відзначається глибокою інтеграцією з іншими сервісами Google, включаючи Gmail, Google Meet та Google Tasks, що дозволяє автоматично створювати події на основі електронних листів, планувати відеоконференції та синхронізувати завдання. Подібним чином, Apple Calendar інтегрується з iCloud та іншими сервісами екосистеми Apple, забезпечуючи безшовний досвід користування на пристроях iOS та macOS. Ця екосистемна інтеграція розширює функціональні можливості систем планування та підвищує їх зручність використання, що є суттєвим фактором при виборі рішення для особистого чи сімейного планування.

Окрему увагу слід приділити аспектам безпеки та конфіденційності в існуючих системах планування часу, особливо в контексті сімейного використання. Баланс між зручністю спільного доступу до інформації та захистом особистих даних залишається складним завданням для розробників. Більшість сучасних рішень пропонують гнучкі налаштування приватності, що дозволяють користувачам контролювати видимість своїх подій, завдань та нотаток для інших членів сім'ї. Наприклад, системи на кшталт FamilyWall та Microsoft Family Safety дозволяють встановлювати різні рівні доступу для різних категорій інформації, таких як особисті зустрічі, медичні призначення або вішлісти подарунків. Такий підхід забезпечує необхідний рівень приватності в межах спільного сімейного простору, враховуючи індивідуальні потреби кожного члена родини та характер інформації, що підлягає спільному використанню [3, с. 84].

Адаптивність до різноманітних сценаріїв використання є одним з найбільш істотних викликів для розробників систем планування часу. Сучасні родини

демонструють значну варіативність у своїй структурі, розмірі, способах комунікації та потребах щодо планування. Деякі з існуючих рішень, як-от Picniic або 2houses, спеціалізуються на конкретних сценаріях, таких як планування для розлучених батьків або великих сімей з кількома поколіннями. Ці спеціалізовані додатки пропонують функціонал, адаптований до специфічних потреб цільової аудиторії, наприклад, інструменти для управління батьківськими обов'язками, відстеження витрат на дітей або координації догляду за літніми членами родини. Водночас, універсальні системи планування, такі як Google Calendar або Microsoft Outlook, прагнуть забезпечити гнучкість та масштабованість, які дозволяють адаптувати їх до різноманітних сценаріїв використання через розширюваність функціоналу та інтеграцію з іншими сервісами [4].

Інноваційні технології, такі як штучний інтелект та машинне навчання, поступово інтегруються в системи планування часу, відкриваючи нові можливості для персоналізації та оптимізації. Алгоритми ШІ аналізують патерни використання календаря, історію взаємодії з завданнями та контекстуальні дані для надання персоналізованих рекомендацій щодо планування, автоматичного категоризування подій та прогнозування часу, необхідного для виконання завдань. Наприклад, Google Calendar використовує технології машинного навчання для функції "Розумні пропозиції", яка автоматично пропонує оптимальний час для зустрічей на основі переваг учасників та їхньої доступності. Подібним чином, Microsoft Scheduler використовує ШІ для автоматизації процесу планування зустрічей через аналіз електронних листів та наявних подій у календарі. Ці технологічні інновації значно підвищують ефективність планування та зменшують когнітивне навантаження, пов'язане з управлінням часом, що є особливо цінним для сімей з напруженими розкладами.

Мобільність та кросплатформенність стали стандартом для сучасних систем планування часу, відображаючи тенденцію до все більш динамічного та розподіленого способу життя. Більшість провідних рішень пропонують синхронізовані веб-додатки, мобільні додатки для iOS та Android, а також інтеграцію з носимими пристроями, такими як смарт-годинники. Ця

багатоплатформенна доступність забезпечує неперервність планування незалежно від місцезнаходження користувача та використовуваного пристрою. Особливо варто відзначити розвиток мобільних додатків для планування, які пропонують не лише доступ до основних функцій, але й специфічні мобільні можливості, такі як геолокаційні нагадування (наприклад, нагадування про покупку, коли користувач знаходиться поблизу магазину) або інтеграцію з системами навігації для оптимізації маршрутів між запланованими подіями, що суттєво підвищує практичну цінність цих систем у контексті динамічного сімейного життя [1].

Інформаційна модель системи базується на семи основних сутностях та їх взаємозв'язках: Користувач як центральна сутність з унікальним ідентифікатором, електронною адресою та ім'ям; Сім'я з назвою, датою створення та посиланням на користувача-власника; Членство в сім'ї для реалізації багато-до-багатьох зв'язку між користувачами та сімейними групами з визначенням ролей; Нотатка з заголовком, текстовим вмістом та прив'язкою до власника; Вішліст з назвою, булевим індикатором приватності та посиланням на користувача; Елемент списку покупок з назвою товару, статусом придбання та ідентифікатором власника; Подія календаря як віртуальна сутність, що синхронізується з Google Calendar через API з локальним кешуванням метаданих для оптимізації продуктивності. Структура зв'язків включає зв'язки один-до-багатьох між користувачем та його особистими даними, багато-до-багатьох між користувачами та сімейними групами через проміжну таблицю членства, та інтеграційні зв'язки з зовнішніми сервісами Google через REST API взаємодію з автентифікацією на основі OAuth 2.0.

Незважаючи на значний прогрес у розвитку систем планування часу, сучасні рішення все ще мають певні обмеження та невирішені виклики. Одним з найбільш помітних обмежень є складність інтеграції між різними екосистемами та платформами. Хоча більшість систем підтримує базові стандарти обміну даними календаря (такі як iCalendar), повна функціональна сумісність між рішеннями від різних виробників залишається проблематичною. Це створює

значні труднощі для сімей, члени яких використовують різні пристрої та сервіси, і часто призводить до фрагментації даних планування. Іншим суттєвим викликом є баланс між автоматизацією та збереженням контролю користувача над процесом планування. Надмірна автоматизація може призвести до відчуття втрати контролю та знизити рівень довіри до системи, особливо в контексті сімейного планування, де емоційні та міжособистісні аспекти відіграють значну роль. Ці обмеження створюють простір для інновацій та вдосконалення майбутніх рішень у сфері особистого та сімейного планування часу [5].

У контексті проектування нової системи онлайн планування особистого та сімейного часу, аналіз існуючих рішень дозволяє визначити оптимальний набір функцій та підходів для забезпечення максимальної ефективності та зручності користування. Інтеграція з сервісами Google, зокрема Google Account та Google Calendar, забезпечує надійну основу для авторизації користувачів та синхронізації даних календаря, водночас розширюючи функціональність системи через доступ до екосистеми Google. Гнучка модель управління доступом, що дозволяє розмежовувати приватну та спільну інформацію, є необхідною для забезпечення балансу між прозорістю та приватністю в сімейному контексті. Додаткові функції, такі як нотатки, вішлісти та списки покупок, розширюють сферу застосування системи за межі базового планування подій, перетворюючи її на комплексний інструмент організації сімейного життя. Розробка інтуїтивно зрозумілого користувацького інтерфейсу, що забезпечує легкий доступ до всіх функцій системи, є фундаментальним фактором для забезпечення позитивного досвіду користування та підвищення рівня адаптації системи серед цільової аудиторії. Врахування цих аспектів дозволить створити конкурентоспроможне рішення, що ефективно задовольнятиме потреби сучасних сімей у плануванні та організації часу [3].

1.3 Методології та принципи проектування інтерфейсів користувача для систем планування

Проектування інтерфейсів користувача для систем планування є комплексним процесом, що знаходиться на перетині технологічних можливостей, психології сприйняття та принципів ефективної організації даних. Інтерфейс користувача в контексті систем планування визначається як сукупність візуальних, інтерактивних та структурних елементів, що забезпечують взаємодію користувача з функціоналом системи та представлення часових даних у зрозумілій та доступній формі. Фундаментальним завданням такого інтерфейсу є створення середовища, що максимально спрощує процеси створення, редагування, перегляду та аналізу планів, мінімізуючи когнітивне навантаження на користувача та забезпечуючи інтуїтивно зрозумілі механізми взаємодії. У сфері проектування інтерфейсів для систем планування сформувалися різноманітні методології, що відображають еволюцію підходів до взаємодії людини з комп'ютером та специфіку представлення часової інформації. Серед основних методологій варто виокремити User-Centered Design (UCD), що фокусується на глибокому розумінні потреб, обмежень та контексту використання системи кінцевими користувачами; Responsive Design, що забезпечує адаптивність інтерфейсу до різних пристроїв та розмірів екранів; та Material Design, що пропонує комплексний підхід до візуального оформлення та інтерактивності елементів інтерфейсу, спираючись на метафори фізичного світу та принципи природного руху [9].

User-Centered Design (UCD) є методологічним підходом, що ставить користувача та його потреби в центр процесу проектування. UCD передбачає активне залучення користувачів на всіх етапах розробки, від початкових досліджень та формування вимог до тестування прототипів та ітеративного вдосконалення дизайну. У контексті систем планування особистого та сімейного часу, застосування UCD дозволяє виявити специфічні патерни використання календарів та органайзерів різними категоріями користувачів, зрозуміти їхні

пріоритети щодо представлення часової інформації та визначити оптимальні механізми взаємодії з системою. Процес UCD зазвичай включає кілька основних етапів: дослідження контексту використання, що передбачає вивчення сценаріїв та умов, в яких користувачі планують свій час; специфікацію вимог на основі виявлених потреб та обмежень користувачів; створення дизайн-рішень, що відповідають цим вимогам; та оцінку розроблених рішень через юзабіліті-тестування та інші методи валідації. Особливу роль у дослідженні контексту використання систем планування відіграють методи етнографічного спостереження, глибинні інтерв'ю та дослідження щоденників використання, що дозволяють виявити неочевидні аспекти взаємодії користувачів з існуючими рішеннями та зрозуміти їхні неартикульовані потреби. Наприклад, дослідження можуть виявити, що батьки з малими дітьми часто планують свій розклад з урахуванням режиму сну та харчування дітей, і, відповідно, потребують специфічних інструментів для фіксації та візуалізації цих режимів у контексті загального сімейного розкладу [8].

Візуальна ієрархія та інформаційна архітектура є фундаментальними аспектами проектування інтерфейсів для систем планування, що визначають ефективність сприйняття та навігації користувачів у контексті часової інформації. Візуальна ієрархія визначається як систематичне розташування та оформлення елементів інтерфейсу таким чином, щоб відображати їхню відносну значущість та логічні взаємозв'язки, спрямовуючи увагу користувача в найбільш оптимальному порядку. В інтерфейсах календарів та планувальників візуальна ієрархія допомагає користувачам швидко ідентифікувати найсуттєвішу інформацію, таку як поточна дата, близькі події, конфлікти в розкладі або важливі нагадування. Інформаційна архітектура, у свою чергу, охоплює структурування та організацію контенту системи, визначення взаємозв'язків між різними типами даних та розробку навігаційних механізмів. У системах планування інформаційна архітектура повинна забезпечувати інтуїтивно зрозумілі шляхи доступу до інформації в різних часових масштабах (день, тиждень, місяць, рік) та категоріях (особисті події, сімейні події, нагадування,

завдання). Одним з ефективних підходів до проектування інформаційної архітектури для систем планування є принцип прогресивного розкриття (progressive disclosure), що передбачає поетапне представлення інформації залежно від її релевантності та контексту взаємодії користувача з системою. Наприклад, у режимі перегляду місяця система може відображати лише назви найсуттєвіших подій, тоді як при переході до перегляду дня — надавати детальну інформацію про кожну подію, включаючи час, місце, учасників та нотатки.

Принципи естетики та візуального дизайну відіграють істотну роль у формуванні користувацького досвіду систем планування, впливаючи не лише на суб'єктивне задоволення від взаємодії, але й на функціональну ефективність інтерфейсу. Естетика інтерфейсу — це сукупність візуальних характеристик, що формують цілісне емоційне та перцептивне враження користувача від системи. Дослідження в галузі HCI (Human-Computer Interaction) демонструють феномен "естетико-узабіліті ефекту", згідно з яким користувачі сприймають естетично привабливі інтерфейси як більш функціональні та зручні у використанні, навіть якщо об'єктивні показники юзабіліті залишаються незмінними. У контексті систем планування, де взаємодія з інтерфейсом є регулярною та тривалою, естетичні аспекти набувають особливої значущості для формування позитивного досвіду та підвищення лояльності користувачів. При проектуванні візуального стилю систем планування доцільно дотримуватися принципів узгодженості (використання єдиної системи кольорів, типографіки та іконографії), чіткості (забезпечення достатнього контрасту та читабельності елементів інтерфейсу) та візуальної економії (мінімізація надлишкових декоративних елементів). Колірна гама інтерфейсу повинна не лише відповідати естетичним уподобанням цільової аудиторії, але й виконувати функціональну роль, допомагаючи в диференціації різних типів подій, календарів або рівнів пріоритету. Наприклад, система може використовувати різні відтінки синього для особистих подій, зеленого для сімейних активностей та червоного для невідкладних нагадувань, що полегшує візуальну навігацію в насиченому інформацією просторі календаря [10].

Інтерактивність та відгук інтерфейсу є критичними аспектами проектування систем планування, що безпосередньо впливають на ефективність взаємодії користувача з системою. Інтерактивність у контексті інтерфейсів користувача визначається як здатність системи реагувати на дії користувача та забезпечувати контроль над процесами та даними. У системах планування часу інтерактивність повинна підтримувати основні сценарії використання, такі як створення та редагування подій, зміна масштабу перегляду календаря, фільтрація та пошук інформації, налаштування відображення та спільне редагування планів. Одним з ключових принципів проектування інтерактивних елементів є забезпечення їх доступності (affordance) — візуальних або контекстуальних підказок, що сигналізують про можливість взаємодії з елементом. Наприклад, візуальне відокремлення та характерне оформлення кнопки "Створити подію" підвищує її помітність та інтуїтивну зрозумілість для користувачів. Іншим суттєвим аспектом інтерактивності є забезпечення миттєвого та інформативного відгуку (feedback) на дії користувача, що створює відчуття контролю та зменшує невизначеність у процесі взаємодії. У контексті систем планування відгук може включати візуальні індикатори успішного створення або оновлення події, анімації, що відображають процеси синхронізації з хмарними службами, або повідомлення про потенційні конфлікти в розкладі. Особливу роль у забезпеченні ефективної інтерактивності відіграють жести та скорочення (shortcuts), що дозволяють досвідченим користувачам виконувати часто повторювані дії з мінімальною кількістю кроків, наприклад, створення нової події через подвійне натискання на комірку календаря або перетягування меж події для зміни її тривалості [7].

Адаптивність та інклюзивність є суттєвими принципами проектування сучасних інтерфейсів для систем планування, що забезпечують доступність та зручність використання для широкого спектру користувачів у різноманітних контекстах. Адаптивний дизайн (Responsive Design) — це підхід до проектування веб-інтерфейсів, що забезпечує оптимальне відображення та функціональність на різних пристроях та розмірах екранів через динамічну реорганізацію контенту

та елементів керування. У контексті систем планування адаптивність набуває особливої значущості, оскільки користувачі взаємодіють з календарями та планувальниками на різних пристроях — від настільних комп'ютерів з великими екранами до смартфонів з обмеженою площею відображення, і часто переходять між пристроями в межах одного сценарію використання. Адаптивний дизайн календаря повинен забезпечувати не лише пропорційне масштабування елементів інтерфейсу, але й переосмислення способів представлення інформації залежно від доступного простору та контексту використання. Наприклад, на мобільному пристрої система може фокусуватися на відображенні подій поточного дня з можливістю горизонтального прокручування для перегляду наступних днів, тоді як на десктопі — одночасно представляти повний тижневий або місячний огляд. Інклюзивний дизайн, у свою чергу, передбачає врахування потреб користувачів з різними фізичними та когнітивними можливостями, культурними особливостями та технологічними обмеженнями. У системах планування інклюзивність може проявлятися через підтримку високого контрасту та масштабування для користувачів з вадами зору, забезпечення повноцінного керування з клавіатури для користувачів з обмеженою моторикою, локалізацію інтерфейсу з урахуванням культурних особливостей сприйняття часу та дат, або оптимізацію продуктивності для пристроїв з обмеженими обчислювальними ресурсами [10].

У першому розділі було закладено теоретичну основу проєкту. Аналіз рішень показав, що існує ніша для системи, яка поєднує глибоку інтеграцію з сервісами Google та деякі сімейні функції, як вішлісти та спільні списки, з гнучкими налаштуваннями приватності.

Технічно проєкт є обґрунтованим — Google Account забезпечує безпечну авторизацію через OAuth 2.0, а Google Calendar API дозволяє реалізувати ключову функцію — синхронізацію подій між членами родини.

Також було визначено, що інтерфейс системи має проєктуватися за принципами User-Centered Design, бути інтуїтивно зрозумілим, щоб максимально спростити для користувачів процес планування. Отже, цей розділ обґрунтовує актуальність

розробки, підтверджує її технічну доцільність та визначає ключові технологічні й дизайнерські підходи, які будуть використані для проєктування та реалізації системи в наступних розділах.

РОЗДІЛ 2. РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз і специфікація вимог до інформаційної системи

Технічна архітектура системи онлайн планування особистого та сімейного часу представляє собою структуровану сукупність взаємопов'язаних програмних компонентів, що забезпечують функціонування всіх аспектів системи від авторизації користувачів до управління даними. В загальному розумінні, архітектура програмного забезпечення — це фундаментальна організація системи, втілена в її компонентах, їх взаємозв'язках між собою та з навколишнім середовищем, а також принципи, що визначають проектування та еволюцію системи. У контексті розроблюваної системи планування, архітектура базується на веб-орієнтованому підході з використанням патерну Model-View-Controller (MVC), що забезпечує чіткий розподіл відповідальності між компонентами та підвищує модульність системи. Основними компонентами архітектури є: серверна частина, реалізована за допомогою фреймворку Flask, система управління базами даних SQLite, інтеграційний модуль для взаємодії з API Google, клієнтська частина, представлена HTML-шаблонами з використанням шаблонізатора Jinja2, та допоміжні модулі для забезпечення авторизації та управління сесіями. Суттєвою особливістю архітектури є її орієнтація на інтеграцію з екосистемою Google, що визначає специфіку механізмів автентифікації та синхронізації даних [11, с. 18].

Для систематизації та наочного представлення вимог до системи було розроблено діаграму вимог (Рис. 2.1), що класифікує функціональні, нефункціональні та технічні аспекти розробки за категоріями та демонструє їх структурні взаємозв'язки в контексті загальної архітектури системи.

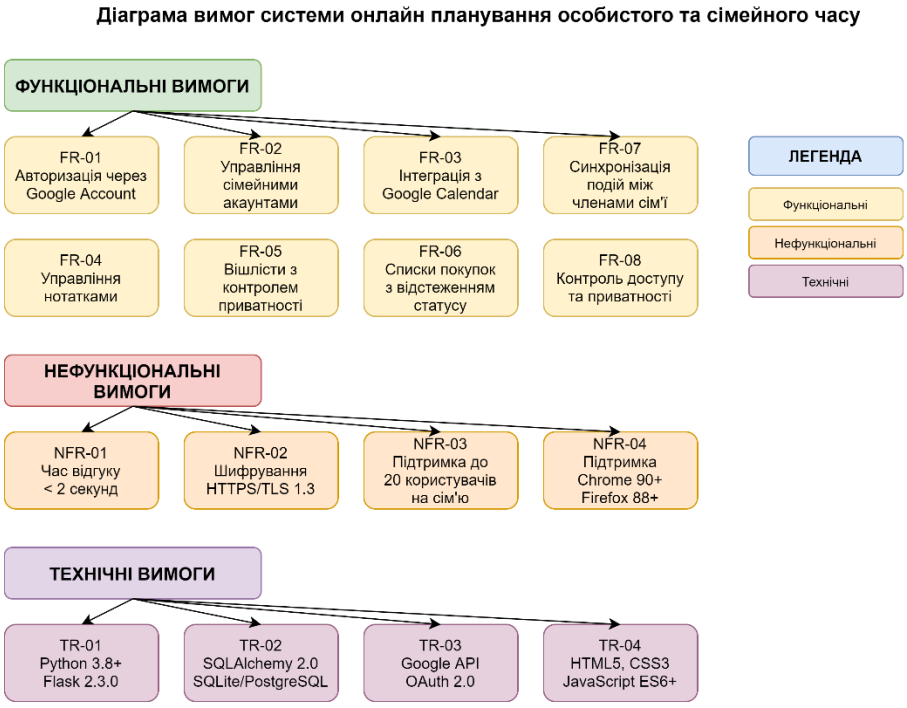


Рисунок 2.1 — Діаграма вимог системи онлайн планування особистого та сімейного часу

Представлена діаграма демонструє ієрархічну структуру вимог, де функціональні вимоги визначають основні можливості системи (авторизація, управління акаунтами, інтеграція з Google Calendar), нефункціональні вимоги встановлюють критерії якості та продуктивності (час відгуку, безпека, масштабованість), а технічні вимоги специфікують конкретні технології та стандарти реалізації (Python/Flask, SQLAlchemy, Google API, веб-технології).

Для детального аналізу функціональних можливостей системи та визначення взаємодії між різними категоріями користувачів було розроблено діаграму прецедентів (Рис. 2.2), що ідентифікує основних акторів системи та їх можливості взаємодії з функціональними компонентами.

Рисунок 2.2 — Діаграма прецедентів системи онлайн-планування особистого та сімейного часу

Представлена діаграма демонструє, що система має трьох основних акторів: звичайного Користувача з правами створення подій та управління особистими даними, Власника сім'ї з розширеними правами управління сімейною групою, та зовнішній сервіс Google Services для забезпечення автентифікації та календарної функціональності. Основні прецеденти включають авторизацію через Google Account, управління профілем, створення та синхронізацію подій календаря, роботу з нотатками, вішлістами та списками покупок, а також адміністративні функції створення сімей та запрошення нових членів.

Функціональна специфікація системи онлайн планування особистого та сімейного часу окреслює повний набір функціональних можливостей, що надаються користувачам, та описує поведінку системи за різних сценаріїв використання. Функціональна специфікація, як документ, що формально визначає вимоги до програмного забезпечення, слугує фундаментом для розробки та тестування системи, забезпечуючи однозначне розуміння очікуваної функціональності всіма учасниками процесу. Розроблювана система орієнтована на забезпечення п'яти основних функціональних компонентів: управління особистими та сімейними акаунтами, інтеграцію з Google Calendar для

планування подій, створення та управління нотатками, ведення вішлістів (списків бажань) та управління списками покупок. Кожен з цих компонентів має свої специфічні вимоги та обмеження, що визначають їх поведінку та взаємодію з іншими компонентами системи. Аналіз наданого коду демонструє, що система реалізована як веб-додаток на основі фреймворку Flask з використанням SQLAlchemy для роботи з базою даних та OAuth 2.0 для інтеграції з сервісами Google [19].

2.2 Постановка задачі

Характеристика задачі. Характеристика задачі проектування моделі даних полягає у створенні ефективної структури для зберігання та управління інформацією про користувачів, їх сімейні зв'язки, особисті записи та побутові списки в умовах багатокористувацького середовища. Специфіка задачі включає необхідність забезпечення одночасної роботи множини користувачів з можливістю розмежування особистих та спільних даних, підтримку сімейної кооперації через групові календарі, інтеграцію з Google Calendar API та реалізацію гнучких механізмів контролю приватності. Основні технічні обмеження системи передбачають максимальну кількість членів в одній сім'ї до 20 користувачів, розмір текстових полів до 5000 символів для нотаток та описів, час відгуку системи не більше 2 секунд для основних операцій та підтримку UTF-8 кодування для забезпечення багатомовності.

Вихідна інформація. Вихідна інформація системи представляє результати обробки даних у різних форматах: календарні дані у форматі JSON для взаємодії з Google Calendar API та HTML-розмітку для відображення у веб-інтерфейсі, особисті дані користувача, сімейну інформацію, структуровані списки нотаток, вішлісти та списки покупок.

Основні структурні одиниці вихідної інформації:

Для події в календарі: summary (назва), description (опис), start (об'єкт з датою/часом початку), end (об'єкт з датою/часом завершення), attendees (масив учасників).

Для сторінки профілю: ім'я та email користувача, перелік членів родини, списки особистих нотаток та вішлістів.

Для вішліста: назва, індикатор приватності (is_private) та посилання на власника.

Для списку покупок: назва товару та статус придбання (is_bought).

Вхідна інформація. Вхідна інформація надходить до системи з різних джерел: дані автентифікації OAuth 2.0 від Google, дані для створення календарних подій, користувацькі текстові дані (нотатки, вішлісти), елементи списків покупок, адміністративні дані для управління сімейними групами та системні параметри запитів.

Основні структурні одиниці вхідної інформації:

Дані автентифікації: токени доступу (access/refresh tokens) та базова профільна інформація (email, ім'я), що надходять від Google.

Дані для створення події: назва, опис, дата та час початку/завершення, що передаються з веб-форми.

Дані для створення нотатки: заголовок та текстовий вміст.

Дані для запрошення в родину: email-адреса користувача, якого запрошують.

2.3 Моделювання інформаційної системи

Структура системи моделюється з двох точок зору: архітектури програмних компонентів та структури даних.

Архітектура програмного забезпечення — це фундаментальна організація системи, втілена в її компонентах та їхніх взаємозв'язках. Розроблювана система базується на веб-орієнтованому підході з використанням патерну Model-View-

Функції контролера виконує серверна частина, реалізована на мікрофреймворку Flask. Вона обробляє HTTP-запити, взаємодіє з моделями

даних та повертає відповідні представлення. Маршрутизація реалізована через декоратори Flask (@app.route), які зв'язують URL-шляхи з функціями-обробниками.

Модель даних реалізована за допомогою ORM SQLAlchemy. Вона забезпечує структуроване зберігання та управління даними користувачів, сімей та інших сутностей системи.

Клієнтська частина, що відповідає за представлення, реалізована через HTML-шаблони з використанням шаблонізатора Jinja2.

Для забезпечення безпеки та управління сеансами користувачів використовуються сесії Flask та розширення Flask-Login з декоратором @login_required.

Структура даних системи та взаємозв'язки між сутностями представлені на UML-діаграмі класів (Рис. 2.3).

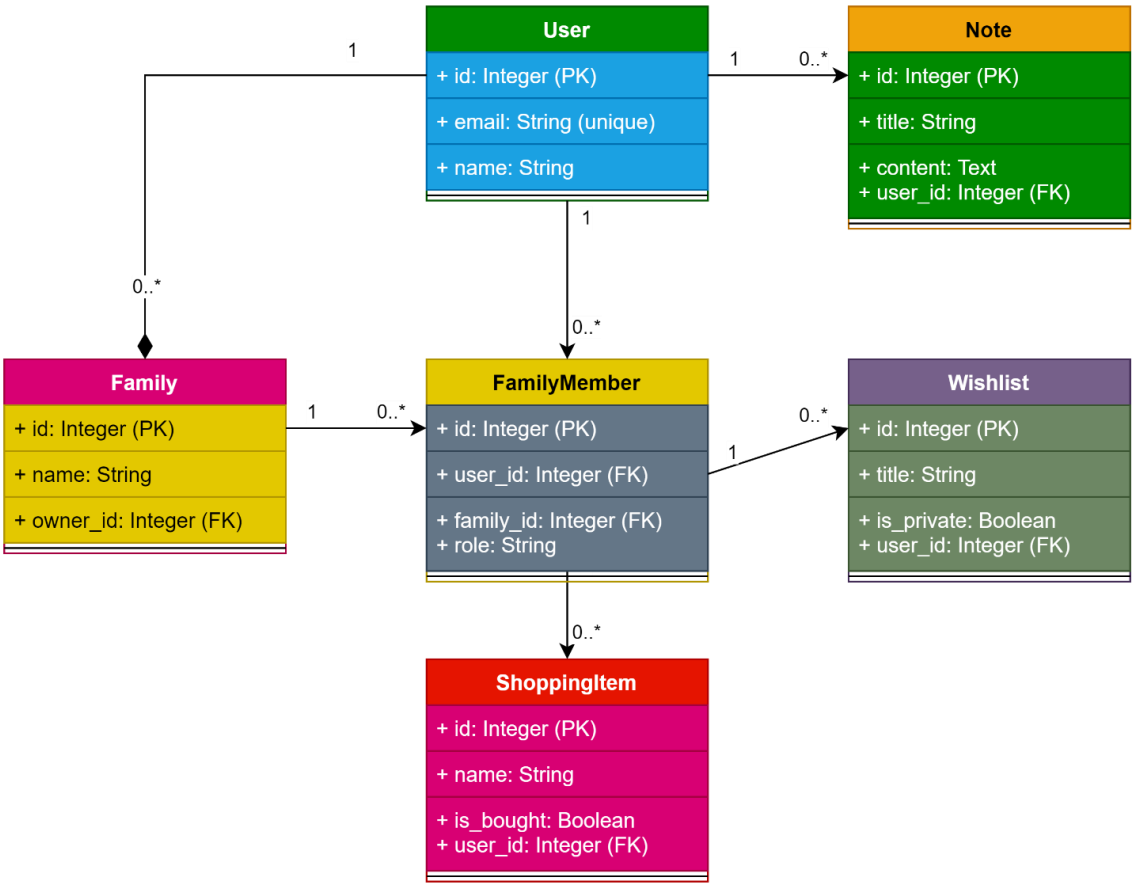


Рисунок 2.3 — UML-діаграма класів моделі даних системи онлайн планування особистого та сімейного часу

На представленій UML-діаграмі відображено шість основних класів моделей даних системи та зв'язки між ними. Основним типом зв'язків у системі є зв'язок "один-до-багатьох", що реалізується через зовнішні ключі у відповідних таблицях. Інтеграція з Google Calendar не потребує окремої моделі даних у базі даних системи, оскільки всі дані календаря зберігаються та синхронізуються безпосередньо через API Google. Натомість, система зберігає токени доступу та інші облікові дані для взаємодії з API в сесії користувача. Ці дані включають token (токен доступу), refresh_token (токен оновлення), token_uri (URI для оновлення токенів), client_id (ідентифікатор клієнта OAuth), client_secret (секрет клієнта OAuth) та scopes (області доступу, що були надані користувачем). Структура для зберігання цих даних створюється у функції callback() через код session["credentials"] = {...}. При створенні події в календарі, ці дані використовуються для ініціалізації об'єкта Credentials, який потім використовується для створення сервісу Google Calendar. Такий підхід до моделювання даних для інтеграції з зовнішніми сервісами є оптимальним, оскільки дозволяє уникнути дублювання даних та проблем з синхронізацією, а також забезпечує актуальність інформації календаря за рахунок прямої взаємодії з API Google [18].

Система управління базами даних SQLite використовується для реалізації сховища даних системи. SQLite є вбудованою реляційною базою даних, що не потребує окремого сервера та зберігає всі дані у єдиному файлі, що робить її ідеальним вибором для невеликих та середніх веб-додатків. У системі планування база даних SQLite ініціалізується через конфігурацію app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///users.db", де "users.db" — це шлях до файлу бази даних відносно кореневої директорії додатку. Створення таблиць у базі даних відбувається автоматично при першому запуску додатку через виклик db.create_all() у блоці if __name__ == "__main__". Ця функція аналізує всі класи моделей, зареєстровані в додатку, та створює відповідні таблиці з необхідними стовпцями та обмеженнями. Такий підхід до ініціалізації бази даних спрощує процес розгортання та тестування системи,

оскільки не потребує ручного створення таблиць або виконання міграцій. Варто відзначити, що для виробничого середовища або високонавантажених систем SQLite може бути замінено на більш потужні системи управління базами даних, такі як PostgreSQL або MySQL, без значних змін у коді, оскільки SQLAlchemy абстрагує взаємодію з конкретною СУБД.

Зв'язки між таблицями в базі даних є істотним аспектом проектування моделі даних, оскільки визначають взаємовідносини між різними сутностями та забезпечують цілісність даних. У розробленій системі використовуються різні типи зв'язків: "один-до-одного", "один-до-багатьох" та "багато-до-багатьох". Зв'язок "один-до-багатьох" є найбільш поширеним і використовується для пов'язування користувача з його даними, такими як нотатки, вішлісти та елементи списку покупок. Цей тип зв'язку реалізується через зовнішні ключі, що посилаються на первинний ключ користувача. Наприклад, атрибут user_id у моделі Note є зовнішнім ключем, що посилається на атрибут id у моделі User, що означає, що один користувач може мати багато нотаток, але кожна нотатка належить лише одному користувачу. Зв'язок "багато-до-багатьох" використовується для реалізації взаємовідносин між користувачами та сім'ями через проміжну таблицю FamilyMember. Ця структура дозволяє одному користувачу бути членом кількох сімей, і одній сім'ї включати кількох користувачів. Зв'язок "один-до-одного" явно не використовується в системі, але може бути реалізований при необхідності розширення функціональності, наприклад, для зв'язування користувача з його профілем або налаштуваннями. Правильне проектування зв'язків між таблицями є фундаментальним для забезпечення цілісності даних та ефективності запитів до бази даних [19].

Масштабованість та розширюваність моделі даних є суттєвими факторами, що визначають здатність системи адаптуватися до зростаючих обсягів даних та нових функціональних вимог. Розроблена модель даних забезпечує масштабованість через свою модульну структуру та чіткий розподіл відповідальності між різними сутностями. Кожна функціональна компонента системи (нотатки, вішлісти, списки покупок) представлена окремою моделлю

даних, що дозволяє незалежно розвивати та оптимізувати кожну компоненту без впливу на інші. Розширюваність моделі даних забезпечується через можливість додавання нових моделей та встановлення зв'язків з існуючими сутностями. Наприклад, для реалізації нової функціональності, такої як трекінг звичок або система нагадувань, можна створити нові моделі даних (Habit, Reminder) та пов'язати їх з існуючими моделями через відповідні зовнішні ключі. Такий підхід дозволяє поступово розвивати систему, додаючи нові функціональні можливості без необхідності повного перепроектування моделі даних. Для забезпечення масштабованості на рівні бази даних, система може бути мігрована з SQLite на більш потужні СУБД, такі як PostgreSQL або MySQL, що підтримують більші обсяги даних та більшу кількість одночасних з'єднань. Така міграція не потребує значних змін у коді, оскільки SQLAlchemy абстрагує взаємодію з конкретною СУБД [17].

У другому розділі було проведено комплексне проектування системи онлайн-планування. На основі аналізу було сформовано та детально специфіковано функціональні й нефункціональні вимоги до майбутнього додатку. Було виконано формальну постановку задачі, що включає опис характеристик системи, а також її вхідної та вихідної інформації.

За допомогою діаграм UML (діаграми прецедентів та класів) було змодельовано поведінку та статичну структуру системи. Також було обґрунтовано вибір архітектурного патерну MVC як основи для розробки, що забезпечує модульність та легкість підтримки програми. Таким чином, цей розділ створює повну проектну основу, що є фундаментом для практичної реалізації та тестування системи, описаних у наступному розділі.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ ПІДСИСТЕМИ

3.1 Інформаційне забезпечення

Інформаційне забезпечення системи є сукупністю методів та засобів для побудови інформаційної бази. Воно включає в себе організацію збору даних, системи їх класифікації та кодування, проектування форм документів та структуру інформаційних масивів.

Проектування моделі даних та структури бази даних є фундаментальним етапом у розробці системи онлайн планування особистого та сімейного часу, оскільки визначає способи зберігання, організації та взаємодії з інформацією в системі. Модель даних у контексті інформаційних систем представляє собою абстрактну модель, що організовує елементи даних та стандартизує їх взаємозв'язки, а також визначає, як дані можуть бути використані та збережені в рамках системи. Для розроблюваної системи планування було обрано реляційну модель даних, що є найбільш поширеним підходом до організації структурованих даних. Реляційна модель базується на концепції таблиць (відношень), що складаються з рядків (кортежів) та стовпців (атрибутів), та дозволяє встановлювати зв'язки між різними таблицями через механізми зовнішніх ключів. Для реалізації моделі даних у системі використовується об'єктно-реляційне відображення (ORM) через бібліотеку SQLAlchemy, що дозволяє абстрагуватися від прямої роботи з SQL та взаємодіяти з базою даних через об'єкти Python. Такий підхід суттєво спрощує розробку та підтримку системи, забезпечуючи прозорість та гнучкість при роботі з даними [15].

Центральною сутністю в моделі даних системи є клас User, що представляє користувача системи з його обліковими даними та профільною інформацією. У коді система визначає модель User з трьома основними атрибутами: id (первинний ключ, що унікально ідентифікує користувача), email (унікальний рядок для зберігання електронної адреси користувача, що слугує також для

інтеграції з Google Account) та name (рядок для зберігання імені користувача). Унікальність електронної адреси забезпечується через обмеження unique=True, що гарантує відсутність дублікатів у системі. Клас User також успадковує від класу UserMixin, що надається бібліотекою Flask-Login та забезпечує базову функціональність для авторизації користувачів, включаючи методи is_authenticated, is_active, is_anonymous та get_id. Ця модель є основою для реалізації функціональності авторизації та персоналізації в системі, оскільки всі дії та дані в системі асоціюються з конкретним користувачем. Варто відзначити, що система не зберігає паролі користувачів, оскільки автентифікація відбувається через OAuth 2.0 з використанням облікових записів Google, що підвищує безпеку системи та спрощує процес автентифікації для користувачів.

Для забезпечення функціональності сімейних акаунтів, модель даних включає дві взаємопов'язані сутності: Family та FamilyMember. Модель Family представляє сімейну групу з атрибутами id (первинний ключ), name (назва сім'ї) та owner_id (зовнішній ключ, що вказує на користувача-власника сім'ї). Зв'язок між таблицями User та Family реалізовано через механізм зовнішнього ключа, де атрибут owner_id таблиці Family посилається на атрибут id таблиці User через конструкцію db.ForeignKey("user.id"). Це гарантує референційну цілісність даних, тобто неможливість створення сім'ї без існуючого користувача-власника. Модель FamilyMember слугує для реалізації зв'язку багато-до-багатьох між користувачами та сім'ями, оскільки один користувач може бути членом кількох сімей, і одна сім'я може включати кількох користувачів. Ця модель містить атрибути id (первинний ключ), user_id (зовнішній ключ, що вказує на користувача), family_id (зовнішній ключ, що вказує на сім'ю) та role (рядок, що визначає роль користувача в сім'ї, наприклад, "owner" або "member"). Така структура дозволяє гнучко управляти складом сімей та ролями їх членів, що є необхідним для реалізації функціональності спільного планування та розділення доступу до даних між членами сім'ї [16].

Вішлісти (списки бажань) є більш складною функціональною сутністю, що включає механізми контролю приватності для регулювання доступу інших

членів сім'ї. Модель Wishlist містить атрибути id (первинний ключ), title (назва вішліста), is_private (булеве значення, що визначає статус приватності вішліста) та user_id (зовнішній ключ, що вказує на користувача-власника). Атрибут is_private має значення за замовчуванням True, що означає, що нові вішлісти створюються як приватні, якщо користувач не вказав інше. Цей атрибут використовується для реалізації механізму контролю доступу: приватні вішлісти видимі лише їх власнику, тоді як публічні (is_private=False) доступні для перегляду іншим членам сім'ї. У функції wishlists() цей механізм реалізовано через складний запит до бази даних, що об'єднує таблиці Wishlist, User та FamilyMember для отримання публічних вішлістів інших членів сім'ї поточного користувача. Запит включає умови Wishlist.user_id != current_user.id (вішліст не належить поточному користувачу), Wishlist.is_private == False (вішліст не є приватним) та FamilyMember.family_id == member.family_id (власник вішліста належить до тієї ж сім'ї, що й поточний користувач). Така структура даних та механізми запитів забезпечують гнучкий контроль приватності та доступу до вішлістів, що є істотним для функціональності сімейного планування [18].

Функціональність нотаток у системі забезпечується через модель Note, що представляє особисті записи користувача. Модель Note включає атрибути id (первинний ключ), title (рядок для зберігання заголовка нотатки), content (текстове поле для зберігання вмісту нотатки) та user_id (зовнішній ключ, що вказує на користувача-власника нотатки). Для зберігання вмісту нотатки використовується тип db.Text, що дозволяє зберігати довгі текстові дані без обмеження довжини, на відміну від типу db.String, який має обмеження. Зв'язок між таблицями User та Note реалізовано через зовнішній ключ user_id, що забезпечує асоціацію кожної нотатки з конкретним користувачем.

Цей зв'язок є типом "один-до-багатьох", оскільки один користувач може мати багато нотаток, але кожна нотатка належить лише одному користувачу. Такий підхід до моделювання даних дозволяє легко отримувати всі нотатки конкретного користувача через запит Note.query.filter_by(user_id=current_user.id).all(), що використовується у функції

notes() для відображення нотаток на сторінці користувача. Модель Note є прикладом сутності, що не потребує механізмів спільного доступу або синхронізації між користувачами, оскільки нотатки є особистими даними кожного користувача [17, с. 37].

Для реалізації функціональності списків покупок у системі використовується модель ShoppingItem, що представляє окремий елемент списку покупок. Модель включає атрибути id (первинний ключ), name (назва товару), is_bought (булеве значення, що вказує, чи придбано товар) та user_id (зовнішній ключ, що вказує на користувача-власника). Атрибут is_bought має значення за замовчуванням False, що означає, що нові елементи списку покупок створюються як "не придбані". Цей атрибут використовується для реалізації функціональності позначення товарів як придбаних/не придбаних без їх видалення зі списку. У функції toggle_shopping() стан елемента змінюється на протилежний через операцію item.is_bought = not item.is_bought. На рівні інтерфейсу, стан елемента відображається через різне форматування: придбані товари відображаються з перекресленим текстом, а не придбані — з іконкою кошика. Така модель даних дозволяє користувачам ефективно відстежувати стан своїх покупок та використовувати список як інструмент планування покупок. Варто відзначити, що, на відміну від вішлістів, модель ShoppingItem не включає механізми спільного доступу або контролю приватності, оскільки списки покупок розглядаються як особисті дані кожного користувача [15]. Для наочного представлення структури бази даних та взаємозв'язків між сутностями системи онлайн планування було розроблено UML-діаграму класів (Рис. 2.3).

3.2 Технічне забезпечення

Технічне забезпечення системи включає комплекс апаратних та програмних засобів, необхідних для функціонування системи в різних середовищах. Серверне обладнання має відповідати мінімальним вимогам:

процесор з 2 ядрами частотою 2.0 GHz, оперативна пам'ять 2 GB RAM, дисковий простір 10 GB SSD та мережеве з'єднання швидкістю 100 Mbps, при цьому рекомендовані характеристики включають 4-ядерний процесор 3.0 GHz, 8 GB RAM та 50 GB SSD для забезпечення оптимальної продуктивності. Клієнтські пристрої потребують будь-якого сучасного процесора, мінімум 2 GB RAM, 50 MB дискового простору для кешу браузера, екрану з роздільною здатністю від 1024x768 та інтернет-з'єднання швидкістю від 1 Mbps. Програмне забезпечення серверної частини базується на Python 3.8+ як основній мові розробки, Flask 2.3.0 як веб-фреймворку, SQLAlchemy 2.0 для роботи з базами даних, SQLite 3.36+ для розробки або PostgreSQL 13+ для виробничого середовища, та Google API Client Libraries для інтеграції з сервісами Google. Допоміжні програмні компоненти включають Nginx 1.20+ як веб-сервер та reverse proxy, Gunicorn 20.1+ як WSGI сервер для Python, та Redis 6.2+ для кешування сесій та управління чергами завдань. Клієнтське програмне забезпечення підтримує сучасні веб-браузери Chrome 90+, Firefox 88+, Safari 14+ та Edge 90+, використовує JavaScript ES6+ для інтерактивності інтерфейсу та HTML5/CSS3 для адаптивної верстки. Засоби розробки та тестування включають IDE Visual Studio Code або PyCharm, систему контролю версій Git, Docker для контейнеризації, pytest для модульного тестування та Selenium для інтеграційного тестування. Безпека системи забезпечується через HTTPS з TLS 1.3, OAuth 2.0 для автентифікації, CSRF захист через Flask-WTF, захист від SQL ін'єкцій через SQLAlchemy ORM та rate limiting для запобігання DDoS атакам, а також включає регулярні автоматичні оновлення безпеки операційної системи та Python пакетів.

3.3 Програмне забезпечення

Серверна частина системи, реалізована на основі мікрофреймворку Flask, відіграє центральну роль у взаємодії всіх компонентів системи. Flask було обрано як оптимальне рішення для створення веб-додатків середньої складності завдяки

його гнучкості, легкості у використанні та розширюваності. В рамках архітектури MVC, Flask виконує функції контролера, обробляючи HTTP-запити від клієнтів, взаємодіючи з моделями даних та повертаючи відповідні представлення. Маршрутизація запитів реалізована через декоратори Flask, які зв'язують URL-шляхи з відповідними функціями-обробниками. Аналіз коду показує, що система має розгалужену структуру маршрутів для обробки різних аспектів функціональності. Наприклад, для роботи з нотатками в системі передбачено маршрути /notes для відображення всіх нотаток користувача, $/notes/add$ для додавання нової нотатки та $/notes/delete/<int:note\_id>$ для видалення конкретної нотатки. Подібним чином організовано маршрути для інших функціональних компонентів: вішлістів, списків покупок та управління сімейними акаунтами. Для забезпечення безпеки та персоналізації взаємодії з користувачем, Flask використовує механізм сесій, що дозволяє зберігати стан автентифікації та інші дані між запитами. Також для підвищення безпеки та контролю доступу до функціональності системи використовується розширення Flask-Login, яке забезпечує механізми для автентифікації користувачів та обмеження доступу до захищених ресурсів через декоратор @login_required.

Модель даних системи, реалізована за допомогою ORM SQLAlchemy, забезпечує структуроване зберігання та управління даними користувачів, сімейних зв'язків та функціональних компонентів системи. Об'єктно-реляційне відображення (ORM) — це техніка програмування, яка дозволяє конвертувати дані між несумісними типовими системами в об'єктно-орієнтованих мовах програмування, створюючи віртуальну об'єктну базу даних, з якою можна працювати в рамках мови програмування. Аналіз коду демонструє, що в системі визначено шість основних класів моделей: User, Family, FamilyMember, Note, Wishlist та ShoppingItem. Клас User є центральним для моделі даних і представляє користувача системи з атрибутами id, email та name. Цей клас також успадковує від UserMixin класу Flask-Login, що надає базову функціональність для авторизації. Модель Family представляє сімейну групу з атрибутами id, name та owner_id, де останній є зовнішнім ключем, що вказує на користувача-власника

сім'ї. Для реалізації зв'язку між користувачами та сім'ями використовується проміжна модель FamilyMember, що містить атрибути user_id, family_id та role, де role визначає роль користувача в сім'ї (власник або звичайний член). Моделі Note, Wishlist та ShoppingItem представляють функціональні сутності системи, кожна з яких пов'язана з конкретним користувачем через зовнішній ключ user_id. Особливістю моделі Wishlist є атрибут is_private, що дозволяє користувачу вказати, чи повинен вішліст бути доступним для інших членів сім'ї [12].

Інтеграція з Google API є одним з найбільш суттєвих аспектів архітектури системи, що забезпечує функціональність автентифікації через Google Account та взаємодію з Google Calendar. Для реалізації цієї інтеграції використовується бібліотека google-auth-oauthlib, що забезпечує реалізацію протоколу OAuth 2.0 для автентифікації та авторизації користувачів. Аналіз коду показує, що процес OAuth-автентифікації реалізовано через функції login() та callback(). Функція login() створює об'єкт Flow з конфігурацією з файлу client_secret.json та запитує доступ до конкретних областей (scopes) API Google: "openid", "*https://www.googleapis.com/auth/userinfo.email*", "*https://www.googleapis.com/auth/userinfo.profile*" та "*https://www.googleapis.com/auth/calendar.events*".

Цей набір областей доступу дозволяє системі отримати інформацію про користувача та взаємодіяти з його календарем. Після ініціалізації OAuth-потоку, функція генерує URL для автентифікації з параметрами "prompt=consent" (для запиту явної згоди користувача), "$access\_type = offline$" (для отримання refresh_token) та "$include\_granted\_scopes = true$" (для включення раніше наданих дозволів). Згенерований URL повертається користувачу для перенаправлення на сторінку автентифікації Google. Функція callback() обробляє відповідь від Google після успішної автентифікації, отримуючи токени доступу та верифікуючи ідентифікаційні дані користувача через функцію id_token.verify_oauth2_token(). На основі отриманої інформації система знаходить або створює запис користувача в базі даних та ініціює сесію через Flask-Login [11].

Взаємодія з Google Calendar API, реалізована через бібліотеку googleapiclient, є центральним елементом функціональності планування в системі. Аналіз коду функції create_event() демонструє процес створення події в Google Calendar та її синхронізації між членами сім'ї. Функція перевіряє наявність облікових даних Google у сесії користувача та, за їх наявності, ініціалізує об'єкт Credentials з токеном доступу, refresh токеном та іншими параметрами з сесії. Цей об'єкт використовується для створення сервісу Google Calendar через функцію build("calendar", "v3", credentials=creds). Для забезпечення синхронізації подій між членами сім'ї, функція виконує запит до бази даних для знаходження сім'ї поточного користувача та отримання електронних адрес усіх її членів. Отримані адреси додаються до списку учасників події через код attendees = [{"email": email} for (email,) in members if email != current_user.email], де умова if email != current_user.email виключає поточного користувача зі списку учасників, оскільки він є організатором події. Дані про подію, включаючи назву, опис, час початку та завершення, а також список учасників, формуються у структуру JSON згідно з вимогами API Google Calendar. Варто відзначити реалізацію обробки часових даних: код перевіряє довжину рядка з часом та автоматично додає секунди, якщо вони відсутні, через умову if len(start) == 16: start += ":00". Після формування даних події, запит на її створення надсилається до API Google Calendar через метод service.events().insert(calendarId="primary", body=event, sendUpdates="all").execute(), де параметр sendUpdates="all" забезпечує автоматичне надсилання повідомлень про подію всім учасникам [13].

Механізм управління сімейними акаунтами є унікальною особливістю системи, що забезпечує функціональність спільного планування та координації між членами родини. Аналіз коду демонструє, що створення сімейного акаунту реалізовано через функцію family(), яка обробляє GET-запити для відображення форми створення сім'ї та POST-запити для збереження нової сім'ї в базі даних. При створенні сім'ї, користувач, який її створює, автоматично стає її власником через запис у таблиці FamilyMember з роллю "owner". Процес запрошення нових

членів до сім'ї реалізовано через функцію invite(), яка приймає електронну адресу запрошуваного користувача через форму на сторінці профілю. Функція перевіряє, чи зареєстрований користувач з такою адресою в системі, та чи є поточний користувач власником сім'ї. Якщо обидві умови виконуються, система створює новий запис у таблиці FamilyMember, пов'язуючи запрошеного користувача з сім'єю та призначаючи йому роль "member". Цей механізм забезпечує гнучке управління складом сім'ї та дозволяє контролювати, хто може бути її членом. На сторінці профілю користувач може бачити всіх членів своєї сім'ї через запит до бази даних, що об'єднує таблиці User та FamilyMember: $db.session.query(User).join(FamilyMember).filter(FamilyMember.family\_id = member.family\_id).all()$. Цей запит повертає об'єкти User для всіх користувачів, які є членами тієї ж сім'ї, що й поточний користувач.

Функціональність нотаток у системі представляє собою приклад базової компоненти управління особистими даними, що не потребує синхронізації з іншими членами сім'ї. Аналіз коду показує, що робота з нотатками реалізована через три основні функції: notes() для відображення всіх нотаток користувача, add_note() для створення нової нотатки та delete_note() для видалення існуючої нотатки. Функція notes() отримує всі нотатки поточного користувача через запит Note.query.filter_by(user_id=current_user.id).all() та передає їх у шаблон для відображення. Функція add_note() обробляє як GET-запити для відображення форми додавання нотатки, так і POST-запити для збереження нової нотатки. При обробці POST-запиту, функція отримує дані з форми (заголовок та зміст нотатки), створює новий об'єкт Note з цими даними та ідентифікатором поточного користувача, та зберігає його в базі даних. Функція delete_note() забезпечує видалення нотатки, але лише після перевірки, що вона належить поточному користувачу, через умову if note and note.user_id == current_user.id. Ця перевірка є частиною механізму безпеки, що запобігає несанкціонованому доступу до даних інших користувачів. Шаблон notes.html відображає нотатки користувача у вигляді списку з можливістю видалення кожної нотатки та додавання нових [14].

Функціональність вішлістів (списків бажань) демонструє складніший підхід до управління даними, що включає механізми контролю приватності та спільного доступу між членами сім'ї. Аналіз коду показує, що робота з вішлістами реалізована через функції wishlists(), add_wishlist() та delete_wishlist(). Особливістю функції wishlists() є її здатність відображати не лише вішлісти поточного користувача, але й публічні вішлісти інших членів його сім'ї. Цей функціонал реалізований через два запити до бази даних: перший отримує всі вішлісти поточного користувача через Wishlist.query.filter_by(user_id=current_user.id).all(), а другий — публічні вішлісти інших членів сім'ї через складний запит, що об'єднує таблиці Wishlist, User та FamilyMember. Цей запит використовує умови Wishlist.user_id != current_user.id (вішліст не належить поточному користувачу), Wishlist.is_private == False (вішліст не є приватним) та FamilyMember.family_id == member.family_id (власник вішліста належить до тієї ж сім'ї, що й поточний користувач). Функція add_wishlist() дозволяє користувачу створити новий вішліст та вказати його статус приватності через прапорець у формі. При збереженні вішліста, значення цього прапорця перетворюється на булеве значення через код is_private

=

$True\ if\ request.form.get("is\_private")\ else\ False$. Функція $delete\_wishlist()$, як і в випадку з нотатками, забезпечує видалення вішліста лише після перевірки, що він належить поточному користувачу. Шаблон wishlists.html відображає як власні вішлісти користувача з можливістю їх видалення, так і публічні вішлісти інших членів сім'ї лише для перегляду [12].

Функціональність списків покупок представляє собою компоненту з розширеними можливостями управління статусом елементів. Аналіз коду показує, що робота зі списками покупок реалізована через функції shopping(), add_shopping(), toggle_shopping() та delete_shopping(). Функція shopping() отримує всі елементи списку покупок поточного користувача та передає їх у шаблон для відображення. Функція add_shopping() створює новий елемент

списку покупок на основі даних з форми та зберігає його в базі даних. Особливістю функціональності списків покупок є можливість позначати елементи як куплені/не куплені без їх видалення зі списку. Ця функціональність реалізована через функцію toggle_shopping(), яка змінює значення атрибута is_bought елемента на протилежне через код

$$item.is_{bought} = not\ item.is_{bought}.$$

Функція delete_shopping() забезпечує повне видалення елемента зі списку покупок. Як і в інших функціях, що модифікують дані, toggle_shopping() та delete_shopping() включають перевірку, що елемент належить поточному користувачу, перед виконанням операції. Шаблон shopping.html відображає елементи списку покупок з можливістю позначати їх як куплені (через перекреслення тексту) та видаляти їх зі списку.

Клієнтська частина системи, реалізована через HTML-шаблони та шаблонізатор Jinja2, забезпечує інтерфейс взаємодії користувача з системою. Аналіз коду шаблонів показує, що система використовує базовий шаблон base.html, який містить загальну структуру HTML, метатеги, стилі та базову розмітку сторінки. Інші шаблони, такі як profile.html, notes.html, wishlists.html та shopping.html, розширюють базовий шаблон через директиву $\{\%\ extends\ "base.html"\ \%\}$ та замінюють блок контенту власним вмістом через конструкцію $\{\%\ block\ content\ \%\}\ldots\{\%\ endblock\ \%\}$. Така організація шаблонів дозволяє уникнути дублювання коду та забезпечує консистентність інтерфейсу. Шаблон profile.html відіграє роль центральної сторінки системи, що відображає інформацію про користувача, членів його сім'ї, а також надає доступ до основних функціональних компонентів: нотаток, вішлістів, списків покупок та створення подій. Шаблон містить умовні блоки, що відображають різний контент залежно від наявності даних, наприклад:

$$\{\%\ if\ family\_members\ \%\}\ldots\{\%\ else\ \%\}\ldots\{\%\ endif\ \%\}$$

для відображення списку членів сім'ї або повідомлення про відсутність сім'ї. Форми в шаблонах використовують метод POST для надсилання даних на сервер, наприклад, форма створення події в create_event.html надсилає дані на

маршрут /create_event. Для покращення користувацького досвіду, в системі використовуються CSS-стилі та HTML-класи, що забезпечують читабельність та зручність взаємодії з інтерфейсом [11].

Безпека та захист даних є невід'ємними аспектами архітектури системи, що реалізуються на різних рівнях: від автентифікації користувачів до контролю доступу до даних. Аналіз коду демонструє, що система використовує декілька механізмів для забезпечення безпеки: OAuth 2.0 для автентифікації через Google Account, сесії Flask для зберігання стану автентифікації, декоратор @login_required для обмеження доступу до захищених маршрутів та перевірки власності даних перед їх модифікацією. Особливу увагу приділено захисту від CSRF-атак через збереження та перевірку стану сесії під час OAuth-автентифікації. Це реалізовано через збереження параметра state у сесії перед перенаправленням на сторінку автентифікації Google: session["state"] = state, та його використання при створенні об'єкта Flow у функції callback(). Для захисту даних від несанкціонованого доступу, система включає перевірки власності перед виконанням операцій модифікації даних. Наприклад, функція delete_note() включає умову if note and note.user_id == current_user.id, що дозволяє видалити нотатку лише її власнику. Подібним чином, функція delete_wishlist() перевіряє, що вішліст належить поточному користувачу, перед його видаленням. Для захисту приватності даних між членами сім'ї, система використовує атрибут is_private моделі Wishlist та відповідні перевірки при формуванні списку доступних вішлістів. Це дозволяє користувачам контролювати, які з їхніх вішлістів будуть доступні для перегляду іншим членам сім'ї [14].

3.4 Результати реалізації інформаційної системи

Реалізація авторизації через Google в системі онлайн планування особистого та сімейного часу представляє собою фундаментальний компонент, що забезпечує безпечний та зручний механізм аутентифікації користувачів без

необхідності створення та зберігання локальних облікових даних. Авторизація через Google базується на протоколі OAuth 2.0, який є стандартом де-факто для реалізації делегованої авторизації. В контексті розробленої системи, OAuth 2.0 дозволяє отримати доступ до даних користувача в сервісах Google (профілю та календаря) після отримання явної згоди користувача, без необхідності передачі та зберігання його паролю. Для реалізації OAuth 2.0 в системі використовується бібліотека google-auth-oauthlib, що надає зручний інтерфейс для взаємодії з API Google. Процес авторизації починається з відображення користувачу вхідної сторінки системи (рис. 3.1), де представлено єдиний спосіб входу – через обліковий запис Google. Коли користувач натискає на відповідне посилання, система ініціює процес OAuth-автентифікації, перенаправляючи його на сторінку входу Google [22, с. 28].

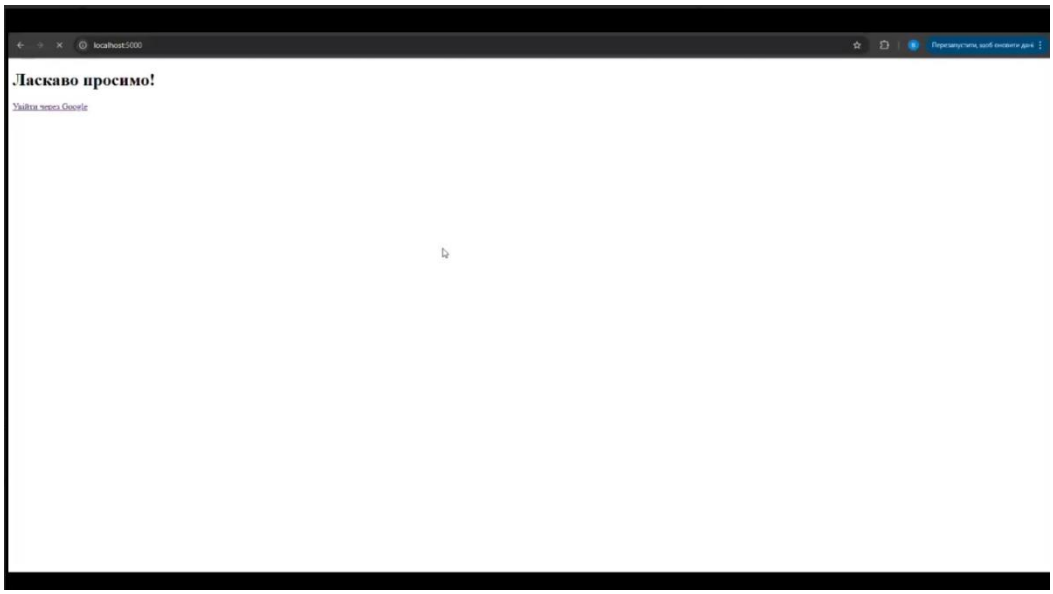


Рисунок 3.1- Спосіб входу через обліковий запис Google

Технічна реалізація процесу авторизації через Google в системі починається з маршруту "/login", що обробляє запити на вхід користувачів. У функції login(), викликається конструктор класу Flow з бібліотеки google-auth-oauthlib, якому передаються шлях до файлу client_secret.json з конфіденційними даними OAuth-клієнта, перелік необхідних областей доступу (scopes) та URL для зворотного виклику (callback). В системі запитуються наступні області доступу: "openid" для базової автентифікації, "*https://www.googleapis.com/auth/userinfo.email*" та "*https://www.googleapis.com/auth/*

userinfo.profile" для отримання інформації про користувача, та "*https://www.googleapis.com/auth/calendar.events*" для взаємодії з календарем. Після ініціалізації об'єкта Flow, генерується URL для автентифікації з параметрами "prompt=consent" (для запиту явної згоди користувача), "access_type=offline" (для отримання refresh_token) та "include_granted_scopes=true" (для включення раніше наданих дозволів). Згенерований URL повертається користувачу для перенаправлення на сторінку автентифікації Google, як показано на рис. 3.2, де користувач має обрати обліковий запис Google для входу в систему.

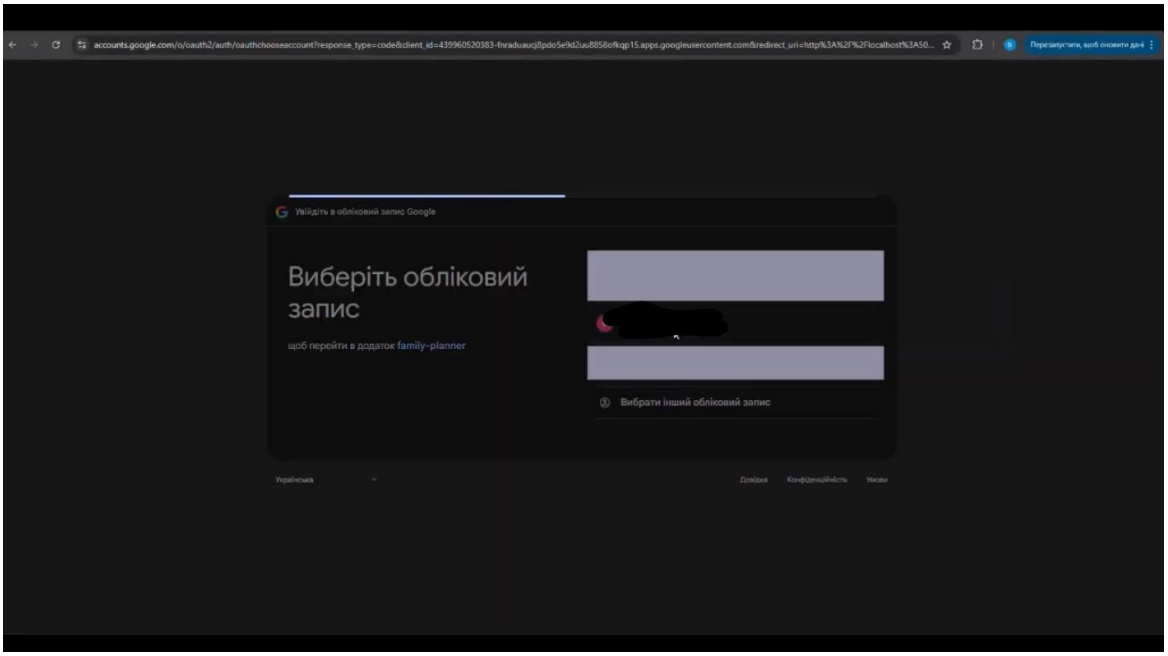


Рисунок 3.2-Обрання облікового запису Google для входу в систему

Після успішної автентифікації на стороні Google, користувач перенаправляється назад до системи за URL зворотного виклику з авторизаційним кодом. Цей процес обробляється функцією callback(), яка отримує авторизаційний код від Google та обмінює його на токени доступу. Ця функція створює новий об'єкт Flow, передаючи йому збережений стан сесії, що захищає від атак CSRF (Cross-Site Request Forgery), та викликає метод fetch_token() з URL-відповіддю від Google. В результаті система отримує токен доступу (access_token), що використовується для автентифікованих запитів до API Google, та, за наявності параметра "access_type=offline", токен оновлення

(refresh_token), що дозволяє отримувати нові токени доступу без повторної автентифікації користувача. Отримані токени зберігаються в сесії користувача для подальшого використання. Після отримання токенів, система верифікує ідентифікаційні дані користувача через функцію id_token.verify_oauth2_token(), що перевіряє підпис токена та його валідність. З отриманих даних система видобуває електронну адресу та ім'я користувача, які використовуються для пошуку або створення запису користувача в базі даних [23].

Управління особистими акаунтами в системі реалізовано через механізм сесій Flask та розширення Flask-Login, що забезпечує функціональність автентифікації та управління сеансами користувачів. Після успішної автентифікації через Google, функція callback() завершується викликом login_user(user), що ініціює сеанс користувача через Flask-Login та перенаправляє його на сторінку профілю. Flask-Login забезпечує збереження стану автентифікації між запитами, надаючи доступ до об'єкта поточного користувача через змінну current_user. Цей механізм дозволяє системі ідентифікувати користувача для всіх наступних запитів та обмежувати доступ до захищених ресурсів неавтентифікованим користувачам через декоратор @login_required. Особистий акаунт користувача представлений у базі даних записом у таблиці User з унікальним ідентифікатором, електронною адресою та іменем. Оскільки автентифікація відбувається через Google, система не зберігає паролів користувачів, що підвищує безпеку. На сторінці профілю користувача (рис. 3.3) відображається його ім'я та електронна адреса, а також надається доступ до функціональності системи: управління сімейним акаунтом, створення подій, нотаток, вішлістів та списків покупок.

Рисунок 3.3- Сторінка профілю користувача

Управління сімейними акаунтами є унікальною особливістю системи, що дозволяє користувачам створювати сімейні групи та запрошувати до них інших користувачів для спільного планування та координації. Створення сімейного акаунту реалізовано через функцію family(), що обробляє запити на форму створення сім'ї (GET) та збереження нової сім'ї (POST). При створенні сімейного акаунту, користувач вказує назву сім'ї, яка зберігається в таблиці Family разом з ідентифікатором користувача-власника. Після створення сімейного акаунту, система автоматично додає користувача-власника до таблиці FamilyMember з роллю "owner", що надає йому розширені права для управління сім'єю. Власник сім'ї може запрошувати інших користувачів приєднатися до сімейного акаунту через функцію invite(), що приймає електронну адресу запрошуваного користувача та перевіряє, чи існує користувач з такою адресою в системі. Якщо користувач знайдений, система додає його до таблиці FamilyMember з роллю "member" та сімейним ідентифікатором власника-запрошувача [24, с. 70].

Логіка запрошення користувачів до сімейного акаунту включає кілька рівнів перевірок для забезпечення безпеки та цілісності даних. По-перше, функція invite() перевіряє, чи є поточний користувач власником сімейного акаунту через запит до таблиці FamilyMember з умовами user_id=current_user.id та role="owner". Якщо користувач не є власником сімейного акаунту, система повертає повідомлення про помилку. По-друге, система перевіряє, чи існує

користувач з вказаною електронною адресою, через запит до таблиці User. Якщо користувач не знайдений, система повертає повідомлення про відсутність користувача. Нарешті, якщо всі перевірки пройдені успішно, система створює новий запис у таблиці FamilyMember, пов'язуючи запрошеного користувача з сімейним акаунтом власника, та перенаправляє власника на сторінку профілю. Цей процес забезпечує контрольоване та безпечне формування сімейних груп, де лише власник має право додавати нових членів.

Візуалізація членів сімейного акаунту реалізована на сторінці профілю користувача через запит до бази даних, що об'єднує таблиці User та FamilyMember. Функція profile() виконує запит db.session.query(User).join(FamilyMember).filter(FamilyMember.family_id == member.family_id).all(), що повертає всіх користувачів (об'єкти User), які є членами тієї ж сім'ї, що й поточний користувач. Результати запиту передаються у шаблон profile.html, де відображаються у вигляді списку з іменами та електронними адресами членів сім'ї. Для користувачів, які ще не є членами сімейного акаунту, система відображає повідомлення "Ви ще не в родині" та надає можливість створити сімейний акаунт через посилання "Створити сімейний акаунт", що веде на сторінку створення сім'ї. Для власників сімейного акаунту, система також надає можливість запрошувати нових членів через форму запрошення, що приймає електронну адресу запрошуваного користувача [25].

Безпека та захист даних є суттєвими аспектами реалізації механізмів авторизації та управління акаунтами. Система забезпечує кілька рівнів захисту: автентифікацію через OAuth 2.0 з використанням облікових записів Google, контроль доступу до захищених ресурсів через декоратор @login_required, захист від CSRF-атак через збереження та перевірку стану сесії під час OAuth-автентифікації, та перевірки прав доступу перед виконанням операцій управління сімейним акаунтом. Для захисту від CSRF-атак, система зберігає згенерований стан у сесії перед перенаправленням на сторінку автентифікації Google: session["state"] = state, та використовує його при створенні об'єкта Flow у функції callback(). Для контролю доступу до захищених ресурсів, усі

маршрути, що потребують автентифікації, захищені декоратором @login_required, який перенаправляє неавтентифікованих користувачів на сторінку входу. Для забезпечення безпеки даних сімейного акаунту, функція invite() включає перевірку, що лише власник сімейного акаунту може запрошувати нових членів [21].

Інтерфейс управління особистими та сімейними акаунтами розроблено з урахуванням принципів юзабіліті та інтуїтивної зрозумілості. Вхідна сторінка системи (рис. 3.1) представляє мінімалістичний інтерфейс з єдиним способом входу – через обліковий запис Google. Сторінка профілю користувача (рис. 3.3) є центральним елементом інтерфейсу, що надає доступ до всіх функціональних компонентів системи та відображає інформацію про користувача та його сімейний акаунт. Інтерфейс створення сімейного акаунту представлений формою з єдиним полем для введення назви сім'ї, що спрощує процес для користувача. Система також надає зрозумілі повідомлення про помилки у випадку невдалого запрошення користувача до сімейного акаунту, наприклад, "У вас немає сім'ї або ви не є власником" або "Користувача з таким email не знайдено". Такий підхід до проектування інтерфейсу забезпечує позитивний досвід користувача та інтуїтивне розуміння процесів авторизації та управління акаунтами.

Масштабованість та розширюваність механізмів авторизації та управління акаунтами забезпечуються через модульний підхід до організації коду та використання стандартних протоколів та бібліотек. Використання OAuth 2.0 для автентифікації дозволяє легко розширити систему для підтримки інших провайдерів ідентифікації, таких як Facebook, Twitter або Microsoft, без значних змін у архітектурі системи. Модель даних для управління сімейними акаунтами також спроектована з урахуванням можливих розширень, наприклад, для підтримки різних рівнів доступу в межах сімейного акаунту або для створення ієрархічних структур сімейних груп. База даних SQLite, що використовується в системі для розробки та тестування, може бути легко замінена на більш потужні СУБД, такі як PostgreSQL або MySQL, для підтримки більшої кількості

користувачів та сімейних акаунтів у виробничому середовищі. Це забезпечується через використання SQLAlchemy, що абстрагує взаємодію з конкретною СУБД [23].

Реалізація авторизації через Google та управління особистими/сімейними акаунтами в системі онлайн планування особистого та сімейного часу демонструє комплексний підхід до забезпечення безпечного та зручного доступу до функціональності системи. Використання OAuth 2.0 для автентифікації, Flask-Login для управління сеансами користувачів, та реляційної бази даних для зберігання даних про користувачів та сімейні акаунти, дозволяє системі ефективно вирішувати завдання ідентифікації користувачів та організації їх у сімейні групи. Візуальний інтерфейс системи, представлений на рис. 3.1 (вхідна сторінка), рис. 3.2 (сторінка вибору облікового запису Google) та рис. 3.3 (сторінка профілю користувача), забезпечує інтуїтивно зрозумілий процес авторизації та управління акаунтами. Технічна реалізація, описана в цьому підрозділі, демонструє практичне застосування принципів безпечної автентифікації та управління користувачами в контексті веб-додатку, що може бути використано як основа для розробки інших систем з подібними вимогами.

Функціональність управління особистими та сімейними акаунтами є основою системи, що забезпечує автентифікацію користувачів, створення сімейних груп та управління їх членами. Автентифікація користувачів реалізована через протокол OAuth 2.0 з використанням облікових записів Google, що дозволяє уникнути необхідності зберігання паролів користувачів та забезпечує високий рівень безпеки. Після автентифікації, користувач може створити сімейний акаунт через інтерфейс, представлений у шаблоні family.html, де він вказує назву сім'ї. При створенні сімейного акаунту, користувач автоматично стає його власником з роллю "owner", що надає йому розширені права для управління сім'єю. Власник сім'ї може запрошувати інших користувачів приєднатися до сімейного акаунту через функцію invite(), що приймає електронну адресу запрошуваного користувача. Система перевіряє, чи існує користувач з такою адресою, та, у разі успіху, додає його до сім'ї з роллю

"member". Функціональність перегляду членів сім'ї реалізована у шаблоні profile.html, що відображає список всіх членів сім'ї з їх іменами та електронними адресами [18]. Фрагмент коду, що демонструє логіку запрошення користувача до сім'ї, наведено нижче:

```
@app.route("/invite", methods=["POST"])
@login_required
def invite():
    email = request.form["invite_email"]
    invited_user = User.query.filter_by(email=email).first()
    # Знайти сім'ю поточного користувача
    family_member = FamilyMember.query.filter_by(user_id=current_user.id, role="owner").first()
    if not family_member:
        return "У вас немає сім'ї або ви не є власником", 400
    if invited_user:
        # Додаємо в сім'ю
        new_member = FamilyMember(user_id=invited_user.id, family_id=family_member.family_id, role="member")
        db.session.add(new_member)
        db.session.commit()
        return redirect(url_for("profile"))
    else:
        return "Користувача з таким email не знайдено", 404
```

Інтеграція з Google Calendar є центральним елементом функціональності планування в системі, що забезпечує можливість створення, редагування та синхронізації подій між членами сім'ї. Ця функціональність реалізована через використання Google Calendar API, доступ до якого отримується через OAuth 2.0 з областями доступу "https://www.googleapis.com/auth/calendar.events". Створення події реалізовано через функцію create_event(), що приймає дані з форми, представленої у шаблоні create_event.html, та створює відповідну подію

в Google Calendar. Формування даних події включає назву, опис, дату та час початку та завершення, а також список учасників, що автоматично включає всіх членів сім'ї користувача. При створенні події, система автоматично надсилає повідомлення про неї всім учасникам через параметр sendUpdates="all" в запиті до API. Аналіз коду показує, що система також обробляє специфічні випадки формату часу, автоматично додаючи секунди, якщо вони відсутні у введених даних. Функціональність перегляду подій не представлена у наданому коді, але вона може бути реалізована через додатковий запит до Google Calendar API для отримання подій поточного користувача та їх відображення у відповідному шаблоні [20].

Таблиця 3.1 - Основні параметри подій, що створюються в системі

Параметр	Тип даних	Опис
summary	Рядок	Назва події
description	Рядок	Опис події
start	Об'єкт	Дата та час початку події з часовим поясом
end	Об'єкт	Дата та час завершення події з часовим поясом
attendees	Масив	Список учасників події (електронні адреси)

Функціональність нотаток забезпечує користувачам можливість створення, перегляду та видалення особистих записів. Нотатки є індивідуальними для кожного користувача та не мають механізмів спільного доступу або синхронізації між членами сім'ї. Створення нової нотатки реалізовано через функцію add_note(), що приймає дані з форми, представленої у шаблоні add_note.html, та зберігає їх у базі даних, асоціюючи з поточним користувачем. Кожна нотатка має заголовок (title) та основний текст (content), що дозволяє структурувати інформацію. Перегляд всіх нотаток користувача реалізовано через функцію notes(), що отримує всі нотатки поточного користувача з бази даних та передає їх у шаблон notes.html для відображення. Видалення нотатки реалізовано через функцію delete_note(), що видаляє нотатку з бази даних після перевірки, що вона належить поточному користувачу, що забезпечує захист від несанкціонованого доступу до даних інших користувачів. Інтерфейс нотаток розроблено з урахуванням принципів юзабіліті, що дозволяє

користувачам легко створювати, переглядати та видаляти нотатки. Наприклад, шаблон notes.html відображає нотатки у вигляді списку з можливістю видалення кожної нотатки та додавання нових, а шаблон add_note.html надає зручну форму для введення заголовка та тексту нотатки [21].

Вішлісти (списки бажань) представляють собою функціональність з розширеними можливостями контролю приватності, що дозволяє користувачам створювати та управляти списками бажаних подарунків або покупок з можливістю надання доступу іншим членам сім'ї. Створення нового вішліста реалізовано через функцію add_wishlist(), що приймає дані з форми, представленої у шаблоні add_wishlist.html, та зберігає їх у базі даних. При створенні вішліста, користувач може вказати його статус приватності через прапорець у формі: приватні вішлісти видимі лише їх власнику, тоді як публічні доступні для перегляду іншим членам сім'ї. Перегляд вішлістів реалізовано через функцію wishlists(), що отримує всі вішлісти поточного користувача, а також публічні вішлісти інших членів його сім'ї, та передає їх у шаблон wishlists.html для відображення. Видалення вішліста реалізовано через функцію delete_wishlist(), що видаляє вішліст з бази даних після перевірки, що він належить поточному користувачу. Ця функціональність особливо корисна для планування подарунків та святкових заходів у сім'ї, оскільки дозволяє членам сім'ї бачити, що хочуть отримати інші, не розкриваючи при цьому свої приватні побажання. Аналіз коду демонструє, що функціональність вішлістів наразі обмежена базовими операціями створення, перегляду та видалення списків, без можливості додавання окремих елементів до списку або позначення їх як придбаних, що може бути розглянуто як напрямок для подальшого розвитку системи [19].

Функціональність списків покупок надає користувачам інструмент для планування та відстеження покупок з можливістю позначення елементів як придбаних без їх видалення зі списку. Ця функціональність реалізована через набір функцій: shopping() для відображення списку покупок, add_shopping() для додавання нового елемента до списку, toggle_shopping() для зміни статусу

елемента (придбаний/не придбаний) та delete_shopping() для видалення елемента зі списку. Кожен елемент списку покупок має назву (name) та статус (is_bought), що вказує, чи придбано товар. Інтерфейс списку покупок, представлений у шаблоні shopping.html, відображає елементи списку з можливістю позначення їх як придбаних (через перекреслення тексту) та видалення зі списку. Додавання нового елемента до списку реалізовано через форму, вбудовану безпосередньо в шаблон shopping.html, що спрощує процес управління списком. Ця функціональність є індивідуальною для кожного користувача і не має механізмів спільного доступу або синхронізації між членами сім'ї, що відповідає типовому сценарію використання списків покупок як особистого інструменту планування. Розглядаючи шаблон проектування MVC, модель даних ShoppingItem представляє інформацію про елементи списку покупок, контролер реалізований через відповідні функції Flask, а представлення — через шаблон shopping.html.

Інтеграція та взаємодія між різними функціональними компонентами є істотним аспектом специфікації системи, що забезпечує цілісний користувацький досвід. Центральним елементом інтеграції є сторінка профілю користувача, реалізована через функцію profile() та шаблон profile.html, що надає доступ до всіх основних функціональних компонентів системи. На сторінці профілю користувач може бачити інформацію про себе, членів своєї сім'ї, останні нотатки, вішлісти та мати швидкий доступ до створення нових подій та управління списком покупок. Така організація інтерфейсу забезпечує зручну навігацію та швидкий доступ до всіх функцій системи з єдиної точки входу. Аналіз коду демонструє, що система також забезпечує інтеграцію на рівні даних через зв'язки між різними моделями у базі даних. Наприклад, функція profile() виконує запити до бази даних для отримання членів сім'ї поточного користувача, його нотаток та вішлістів, а також публічних вішлістів інших членів сім'ї, що демонструє взаємозв'язок між різними функціональними компонентами. Інтеграція з Google Calendar забезпечує можливість створення подій, що автоматично включають всіх членів сім'ї як учасників, що демонструє взаємодію

між функціональністю управління сімейними акаунтами та плануванням подій [20].

Безпека та контроль доступу є наскрізними аспектами функціональної специфікації, що забезпечують захист даних користувачів та належний рівень приватності. Аналіз коду демонструє, що система реалізує кілька рівнів захисту: автентифікацію через OAuth 2.0 з використанням облікових записів Google, контроль доступу до захищених ресурсів через декоратор @login_required, перевірку прав власності на дані перед їх модифікацією та механізми контролю приватності для вішлістів. Для кожної функції, що модифікує дані (видалення нотатки, вішліста або елемента списку покупок, зміна статусу елемента списку покупок), система перевіряє, що дані належать поточному користувачу, перед виконанням операції. Наприклад, функція :

$delete\_note()$ включає умову $if\ note\ and\ note.user\_id\ ==\ current\_user.id$, що дозволяє видалити нотатку лише її власнику. Механізми контролю приватності для вішлістів реалізовані через атрибут is_private та відповідні перевірки при формуванні списку вішлістів, доступних для перегляду іншим членам сім'ї. Система також забезпечує захист від CSRF-атак через збереження та перевірку стану сесії під час OAuth-автентифікації. Ці механізми безпеки є невід'ємною частиною функціональної специфікації системи та забезпечують належний рівень захисту даних користувачів [21].

Далі розроблено реляційну модель даних, яка визначає структуру бази даних. Вона включає такі ключові сутності, як User, Family, Note, Wishlist та ShoppingItem. Для реалізації сімейного функціоналу створено проміжну таблицю FamilyMember, що встановлює зв'язок "багато-до-багатьох" між користувачами та сім'ями. Важливими елементами є спеціальні атрибути, як is_private у вішлістах та is_bought у списках покупок, що дозволяють реалізувати унікальний функціонал. При цьому дані календаря не зберігаються в системі, а синхронізуються напряму з Google API, щоб гарантувати їхню актуальність.

На завершення, функціональна специфікація формалізує поведінку системи. Вона передбачає безпечну автентифікацію користувачів через Google,

можливість створювати сімейні групи та запрошувати до них учасників. Головна функція — створення подій у Google Calendar, які автоматично стають спільними для всіх членів родини. Крім того, система пропонує інструменти для особистої організації: нотатки , списки покупок та вішлісти з гнучкими налаштуваннями приватності.

Інтеграція з Google Calendar представляє собою один із центральних елементів розробленої системи онлайн планування особистого та сімейного часу, що забезпечує створення, редагування та синхронізацію подій між членами родини. Google Calendar API — це програмний інтерфейс, що надає розробникам доступ до функціональності календаря Google, дозволяючи програмно взаємодіяти з даними користувацьких календарів [25]. У контексті розробленої системи, інтеграція з Google Calendar реалізована через бібліотеку googleapiclient, що надає зручний інтерфейс для взаємодії з API Google на мові Python. Процес інтеграції починається з отримання необхідних дозволів від користувача під час OAuth-автентифікації, де система запитує доступ до календаря користувача через область доступу "https://www.googleapis.com/auth/calendar.events". Цей дозвіл дозволяє системі створювати, редагувати та видаляти події в календарі користувача від його імені. Рис. 3.4 демонструє інтерфейс створення нової події в системі, що дозволяє користувачу ввести назву події, опис, дату та час початку та завершення. Після заповнення форми, ці дані передаються на сервер для створення події в Google Calendar.

Рисунок 3.4 -Інтерфейс створення нової події в системі

Технічна реалізація інтеграції з Google Calendar здійснюється через функцію create_event(), що обробляє як GET-запити для відображення форми створення події, так і POST-запити для збереження нової події в календарі. Ключовим елементом цієї функції є ініціалізація об'єкта Credentials з токенами доступу, збереженими в сесії користувача після успішної автентифікації. Цей об'єкт включає token (токен доступу), refresh_token (токен оновлення), token_uri (URI для оновлення токенів), client_id (ідентифікатор клієнта OAuth), client_secret (секрет клієнта OAuth) та scopes (області доступу). Об'єкт Credentials використовується для створення сервісу Google Calendar через функцію build("$calendar$","$v3$",$credentials = creds$), що надає доступ до API Calendar. Для підвищення надійності системи, функція create_event() включає перевірку наявності облікових даних Google у сесії користувача та перенаправляє на сторінку входу, якщо вони відсутні. Це забезпечує, що користувач завжди має валідні токени доступу перед спробою взаємодії з API Google Calendar. Формат даних для створення події відповідає вимогам API Google Calendar: структура JSON, що включає поля summary (назва події), description (опис), start (дата та час початку з часовим поясом), end (дата та час завершення з часовим поясом) та attendees (список учасників) [26].

Автоматична синхронізація подій між членами сім'ї є однією з ключових можливостей розробленої системи, що значно підвищує ефективність сімейного планування. Ця функціональність реалізована через автоматичне додавання всіх членів сім'ї як учасників події при її створенні. У функції create_event() система спочатку знаходить сім'ю поточного користувача через запит до таблиці FamilyMember:

$family\_member = FamilyMember.query.filter\_by(user\_id = current\_user.id).first()$. Якщо користувач є членом сім'ї, система отримує ідентифікатор сім'ї та використовує його для отримання електронних адрес всіх членів сім'ї через складний запит до бази даних:

$db.session.query(User.email).join(FamilyMember, User.id == FamilyMember.user\_id).filter(FamilyMember.family\_id == family\_id).all()$. Отримані адреси перетворюються на список об'єктів attendees для API Google Calendar з використанням генератора списків: attendees = [{"email": email} for (email,) in members if email != current_user.email], де умова if email != current_user.email виключає поточного користувача зі списку учасників, оскільки він є організатором події. Цей механізм забезпечує, що всі члени сім'ї автоматично отримують повідомлення про нову подію та бачать її у своїх календарях, що суттєво спрощує координацію планів між членами сім'ї.

Обробка часових даних є істотним аспектом функціональності управління подіями, що вимагає особливої уваги для забезпечення коректної роботи з API Google Calendar. У формі створення події, показаній на Рис. 3.5, користувач вводить дату та час початку та завершення події у форматі, що відповідає стандарту HTML5 для елементів input типу datetime-local: "$YYYY-MM-DDThh$". Однак, API Google Calendar очікує часові дані у форматі ISO 8601 з додаванням секунд: "$YYYY-MM-DDThh:mm$".

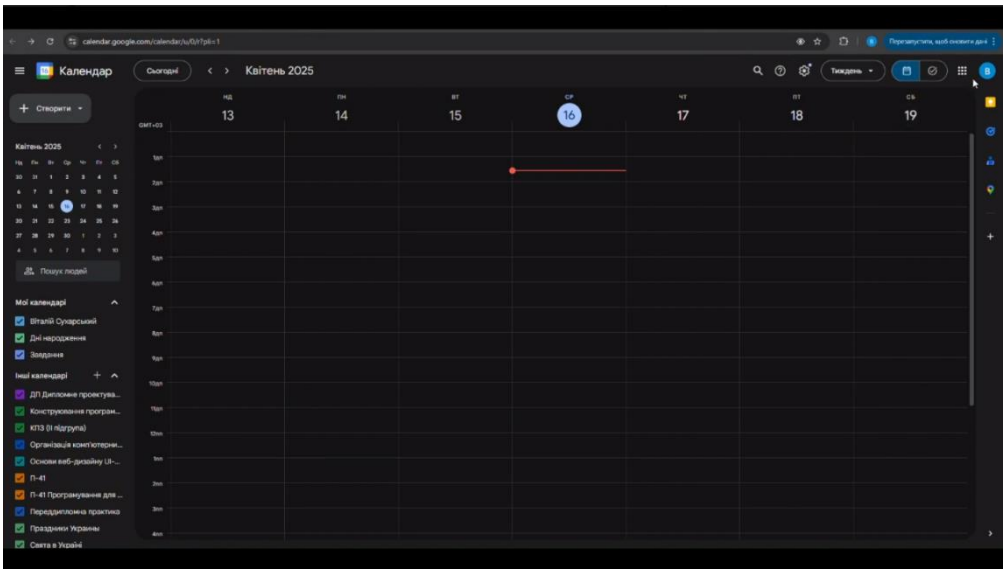


Рисунок 3.5-Створення події

Для забезпечення сумісності форматів, функція create_event() включає логіку обробки часових даних, що перевіряє довжину рядка з часом та автоматично додає секунди, якщо вони відсутні: if len(start) == 16: start += ":00". Крім того, система додає інформацію про часовий пояс до даних початку та завершення події через поле timeZone: "$start$": {"$dateTime$": $start$, "$timeZone$": "$Europe/Kyiv$"}. Це забезпечує коректне відображення часу події для всіх учасників, незалежно від їхнього місцезнаходження та налаштувань часового поясу. Рис. 3.5 демонструє результат створення події в Google Calendar, де видно, що подія успішно додана з коректними часовими даними [27, с. 57].

Відправлення повідомлень про події учасникам є автоматичною функцією Google Calendar, що активується параметром sendUpdates при створенні або оновленні події. У розробленій системі, функція create_event() використовує параметр $sendUpdates = "all"$ при виклику методу service.events().insert() для створення нової події. Це значення параметра вказує API Google Calendar надіслати повідомлення про подію всім учасникам, включаючи організатора, запрошених учасників та тих, хто має доступ до події через спільний календар. Повідомлення надсилаються на електронні адреси учасників, вказані у полі attendees, та містять інформацію про назву, опис, час та місце події, а також

посилання на подію в Google Calendar. Це дозволяє учасникам бути проінформованими про нові події та зміни в існуючих подіях, навіть якщо вони не використовують розроблену систему безпосередньо. Для користувачів, які мають налаштовані сповіщення в Google Calendar, система також генерує відповідні нагадування про події згідно з їхніми налаштуваннями, що додатково підвищує ефективність планування [22].

Управління перевіркою доступу до функціональності створення подій реалізовано через декоратор @login_required, що обмежує доступ до маршруту "/create_event" лише для автентифікованих користувачів. Цей механізм забезпечує, що лише користувачі, які успішно пройшли процес автентифікації через Google, можуть створювати події в календарі. Додатково, функція create_event() перевіряє наявність облікових даних Google у сесії користувача через умову if "credentials" not in session та перенаправляє на сторінку входу, якщо вони відсутні. Це гарантує, що користувач має валідні токени доступу для взаємодії з API Google Calendar перед спробою створення події. При формуванні списку учасників події, система також враховує сімейні зв'язки користувача, додаючи як учасників лише членів його сім'ї. Це реалізовано через запит до бази даних, що отримує лише тих користувачів, які належать до тієї ж сім'ї, що й поточний користувач. Такий підхід забезпечує приватність даних календаря та обмежує доступ до подій лише для членів відповідної сімейної групи [24].

Інтерфейс створення та перегляду подій розроблено з урахуванням принципів юзабіліті та інтуїтивної зрозумілості. Форма створення події, показана на Рис. 3.4, надає користувачу зрозумілі поля для введення назви події, опису, дати та часу початку та завершення. Для зручності введення часових даних, використовуються нативні елементи форми HTML5 типу datetime-local, що надають візуальний інтерфейс для вибору дати та часу з календаря. Після створення події, користувач перенаправляється на сторінку профілю, де може бачити список членів своєї сім'ї та мати доступ до інших функціональних компонентів системи. Для перегляду створених подій, користувач може використовувати інтерфейс Google Calendar, доступний за посиланням

календаря Google, як показано на Рис. 3.5. Таким чином, розроблена система інтегрується з існуючим інтерфейсом Google Calendar, що дозволяє користувачам використовувати звичні інструменти для перегляду та управління подіями, додатково розширюючи їх функціональністю автоматичної синхронізації між членами сім'ї [26].

Масштабованість та розширюваність функціональності управління подіями забезпечуються через модульний підхід до організації коду та використання стандартного API Google Calendar. Поточна реалізація зосереджена на базовій функціональності створення подій, але система легко може бути розширена для підтримки редагування та видалення подій, а також для додавання додаткових функцій, таких як повторювані події, нагадування або категоризація подій. Ці розширення можуть бути реалізовані через додаткові маршрути та функції, що використовують ті ж механізми авторизації та взаємодії з API Google Calendar, що й існуюча функціональність. Наприклад, для реалізації редагування події, можна створити маршрут "$/edit\_event/< event\_id >$", що отримує ідентифікатор події з URL та використовує метод service.events().get() для отримання даних події, метод service.events().update() для оновлення події після редагування користувачем. Використання Google Calendar API також забезпечує сумісність системи з різними клієнтами Google Calendar, включаючи веб-інтерфейс, мобільні додатки та сторонні клієнти, що підтримують синхронізацію з Google Calendar [27].

Розробка та впровадження додаткових функцій системи онлайн планування особистого та сімейного часу суттєво розширює її функціональні можливості, перетворюючи систему з простого календаря на комплексний інструмент організації повсякденного життя. Додаткові функції, такі як нотатки, вішлісти та списки покупок, дозволяють користувачам не лише планувати події та зустрічі, але й зберігати корисну інформацію, відстежувати бажані придбання та ефективно управляти повсякденними завданнями. Нотатки в контексті системи планування можна визначити як текстові записи користувача, що використовуються для збереження ідей, нагадувань, інструкцій або будь-якої

іншої інформації, яка не прив'язана до конкретної дати або часу. Вішлісти (від англ. wishlist — список бажань) представляють собою переліки бажаних придбань або подарунків, що можуть бути приватними (видимими лише для власника) або публічними (доступними для перегляду іншим членам сім'ї). Списки покупок є інструментом для планування та відстеження необхідних придбань, з можливістю позначення товарів як придбаних [28]. Розробка цих функцій вимагала комплексного підходу, що включав проектування моделей даних, реалізацію серверної логіки та створення інтуїтивно зрозумілого користувацького інтерфейсу, як демонструє скріншот форми створення нотатки на Рис. 3.6.

Рисунок 3.6- Запити для відображення форми додавання нотатки

Функціональність нотаток реалізована через набір маршрутів та функцій, що забезпечують основні операції створення, перегляду та видалення. Маршрут /notes обробляється функцією notes(), яка отримує всі нотатки поточного користувача з бази даних через запит $Note.query.filter\_by(user\_id = current\_user.id).all()$ та передає їх у шаблон $notes.html$ для відображення. Маршрут /notes/add обслуговується функцією add_note(), яка обробляє як GET-запити для відображення форми додавання нотатки (Рис. 3.6), так і POST-запити для збереження нової нотатки в базі даних.

Форма створення нотатки включає поля для введення назви та вмісту нотатки, а також кнопки для збереження або скасування операції. При обробці POST-запиту, функція add_note() отримує дані з форми через об'єкт request.form,

створює новий об'єкт класу Note з отриманими даними та ідентифікатором поточного користувача, та зберігає його в базі даних через операції db.session.add(note) та db.session.commit(). Маршрут /notes/delete/<int:note_id> обробляється функцією delete_note(), яка видаляє нотатку з вказаним ідентифікатором, але лише після перевірки, що вона належить поточному користувачу, через умову if note and note.user_id == current_user.id. Ця перевірка є частиною механізму безпеки, що запобігає несанкціонованому доступу до даних інших користувачів. Всі маршрути, пов'язані з нотатками, захищені декоратором @login_required, що гарантує доступ до цієї функціональності лише для автентифікованих користувачів [29].

Функціональність вішлістів включає механізми контролю приватності, що дозволяють користувачам вибирати, чи будуть їхні списки бажань видимими для інших членів сім'ї. Ця функціональність реалізована через атрибут is_private в моделі даних Wishlist, що визначає статус приватності кожного вішліста. При створенні нового вішліста через функцію add_wishlist(), користувач може вказати його статус приватності через прапорець у формі, що відображається за маршрутом /wishlists/add. Значення цього прапорця перетворюється на булеве значення через код is_private = True if request.form.get("is_private") else False та зберігається в базі даних разом з іншими даними вішліста. Функція wishlists(), що обробляє маршрут /wishlists, виконує два окремих запити до бази даних: перший отримує всі вішлісти поточного користувача незалежно від їх статусу приватності, а другий — публічні вішлісти інших членів його сім'ї. Другий запит є більш складним і об'єднує таблиці Wishlist, User та FamilyMember для отримання лише тих вішлістів, які належать іншим членам сім'ї поточного користувача та мають статус приватності is_private=False. Результати обох запитів передаються у шаблон wishlists.html для відображення у вигляді двох окремих списків: "Мої вішлісти" та "Публічні вішлісти членів родини". Функція delete_wishlist() забезпечує видалення вішліста з вказаним ідентифікатором, але лише після перевірки, що він належить поточному користувачу, аналогічно до функції видалення нотаток [28].

Функціональність списків покупок розширює можливості системи для планування повсякденних завдань, дозволяючи користувачам створювати та управляти списками необхідних придбань. Ця функціональність реалізована через набір маршрутів та функцій, що забезпечують операції додавання нових елементів, зміни їх статусу та видалення. Маршрут /shopping обробляється функцією shopping(), яка отримує всі елементи списку покупок поточного користувача та передає їх у шаблон shopping.html. Цей шаблон відображає список з можливістю позначення елементів як придбаних та видалення елементів зі списку. Додавання нового елемента до списку покупок реалізовано через маршрут /shopping/add та функцію add_shopping(), яка обробляє POST-запити з формою, вбудованою безпосередньо в шаблон shopping.html. Унікальною особливістю функціональності списків покупок є можливість позначати елементи як придбані без їх видалення зі списку. Ця функціональність реалізована через маршрут $/shopping/toggle/<int:item\_id>$ та функцію toggle_shopping(), яка змінює значення атрибута is_bought елемента на протилежне через операцію item.is_bought = not item.is_bought. На рівні інтерфейсу, елементи списку покупок відображаються по-різному залежно від їх статусу: придбані товари відображаються з перекресленим текстом та іконкою "☑", а не придбані — з іконкою кошика "🛒". Така візуальна диференціація допомагає користувачам швидко ідентифікувати статус кожного елемента списку.

Інтеграція додаткових функцій у загальний інтерфейс системи реалізована через сторінку профілю користувача, що слугує центральним хабом для доступу до всіх функціональних компонентів. На сторінці профілю, реалізованій через маршрут /profile та функцію profile(), користувач може бачити зведену інформацію про свої нотатки, вішлісти та мати швидкий доступ до списку покупок. Функція profile() виконує кілька запитів до бази даних для отримання всієї необхідної інформації: членів сім'ї поточного користувача, його нотаток, вішлістів та публічних вішлістів інших членів сім'ї. Результати цих запитів передаються у шаблон profile.html, який структурує інформацію у вигляді

окремих секцій з заголовками та надає посилання для переходу до повних версій відповідних функціональних компонентів. Такий підхід до організації інтерфейсу забезпечує зручну навігацію та швидкий доступ до всіх функцій системи з єдиної точки входу. На сторінці профілю також розміщені кнопки для створення нових нотаток, вішлістів та подій, що спрощує процес додавання нової інформації до системи. Додатково, сторінка профілю відображає інформацію про поточного користувача (ім'я та електронну адресу) та членів його сім'ї, що підкреслює соціальний аспект системи та її орієнтацію на сімейне використання [30].

Безпека та контроль доступу до даних додаткових функцій є наскрізними аспектами їх реалізації, що забезпечують захист особистої інформації користувачів та належний рівень приватності. Для кожної функції, що модифікує дані (видалення нотатки, вішліста або елемента списку покупок, зміна статусу елемента списку покупок), система включає перевірку, що дані належать поточному користувачу, перед виконанням операції. Наприклад, функція delete_note() включає умову if note and note.user_id == current_user.id, що дозволяє видалити нотатку лише її власнику. Аналогічні перевірки присутні у функціях delete_wishlist(), delete_shopping() та toggle_shopping(). Для вішлістів реалізовано додатковий рівень контролю доступу через атрибут is_private, що визначає, чи буде вішліст видимим для інших членів сім'ї. Публічні вішлісти (з is_private=False) доступні для перегляду всім членам сім'ї власника, тоді як приватні (з is_private=True) видимі лише для власника. Ця функціональність дозволяє користувачам контролювати видимість своїх списків бажань, що є особливо корисним при плануванні сюрпризів або подарунків для інших членів сім'ї. Всі маршрути, пов'язані з додатковими функціями, захищені декоратором @login_required, що гарантує доступ до цієї функціональності лише для автентифікованих користувачів та запобігає несанкціонованому доступу до даних.

Третій розділ кваліфікаційної бакалаврської роботи демонструє практичну реалізацію та функціональне тестування спроєктованої системи, перетворюючи теоретичні розробки на робочий продукт.

Реалізовано механізм авторизації через Google з використанням протоколу OAuth 2.0, що забезпечує безпечний вхід до системи. На основі цього створено систему управління особистими та сімейними акаунтами, де користувачі можуть формувати групи для спільного планування, запрошуючи інших учасників.

Також є інтеграція з Google Calendar. Розроблено функціонал, що при створенні події автоматично отримує з бази даних список членів родини та додає їх як учасників, забезпечуючи миттєву синхронізацію розкладу.

Крім цього, впроваджено додаткові функції, що розширюють можливості системи. Це нотатки для особистих записів , списки покупок з можливістю позначати придбані товари , та унікальна система вішлістів, що дозволяє створювати як приватні, так і видимі для родини списки бажань.

Наведені у розділі скріншоти візуально підтверджують працездатність кожного компонента.

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи було розроблено систему онлайн планування особистого та сімейного часу, що дозволяє користувачам ефективно організовувати свій розклад, зберігати корисну інформацію та координувати плани з членами родини. В рамках дослідження було проаналізовано існуючі рішення у сфері особистого та сімейного планування, вивчено особливості інтеграції з сервісами Google та методології проектування інтерфейсів користувача для систем планування. На основі отриманих знань було спроектовано технічну архітектуру системи, модель даних та функціональну специфікацію, що відповідають сучасним вимогам до веб-додатків та забезпечують оптимальний користувацький досвід. Розроблена система поєднує в собі функціональність календаря з додатковими інструментами для організації повсякденного життя, такими як нотатки, вішлісти та списки покупок, що суттєво розширює її можливості у порівнянні з традиційними календарями.

Технічна реалізація системи базується на використанні фреймворку Flask для серверної частини, SQLAlchemy для роботи з базою даних SQLite, та OAuth 2.0 для інтеграції з сервісами Google. Таке технологічне рішення дозволило створити гнучку та масштабовану архітектуру, що може бути легко розширена новими функціональними можливостями. Ключовою особливістю системи є інтеграція з Google Calendar, що забезпечує створення та синхронізацію подій між усіма членами родини автоматично. Це значно спрощує процес координації планів та підвищує ефективність сімейного планування. Авторизація через Google Account не лише забезпечує високий рівень безпеки, але й спрощує процес входу користувачів до системи, оскільки не потребує створення та запам'ятовування додаткових облікових даних.

Функціональність управління особистими та сімейними акаунтами реалізована таким чином, що дозволяє користувачам створювати сімейні групи та запрошувати до них інших користувачів системи. Це забезпечує формування

закритих сімейних спільнот, де члени родини можуть спільно планувати події, бачити публічні вішлісти один одного та ефективно координувати свої дії. Механізми контролю доступу та безпеки, впроваджені в системі, гарантують, що особисті дані користувачів залишаються захищеними, а доступ до функціональності системи обмежений лише для авторизованих користувачів. Для вішлістів реалізовано додатковий рівень приватності, що дозволяє користувачам вибирати, чи будуть їхні списки бажань видимими для інших членів сім'ї, що є особливо корисним при плануванні сюрпризів або подарунків.

Розроблений користувацький інтерфейс системи відповідає сучасним принципам юзабіліті та забезпечує інтуїтивно зрозумілу взаємодію з усіма функціональними компонентами. Сторінка профілю користувача слугує центральним хабом, де відображається зведена інформація про нотатки, вішлісти, членів сім'ї та надається швидкий доступ до всіх функцій системи. Форми для створення подій, нотаток та вішлістів розроблені з урахуванням потреб користувачів та забезпечують зручне введення даних. Інтерфейс списків покупок включає візуальну диференціацію елементів за їх статусом, що допомагає користувачам швидко ідентифікувати придбані та непридбані товари. Такий підхід до проектування інтерфейсу забезпечує позитивний досвід використання системи та мінімізує когнітивне навантаження на користувачів.

Результати кваліфікаційної бакалаврської роботи демонструють практичне застосування принципів веб-розробки, інтеграції з зовнішніми API та проектування баз даних для створення комплексної системи планування. Розроблена система може бути використана як основа для створення більш масштабних рішень у сфері особистого та сімейного планування або інтегрована з іншими сервісами для розширення її функціональності. Потенційними напрямками для подальшого розвитку системи можуть бути: впровадження мобільного додатку для забезпечення доступу до функціональності системи з мобільних пристроїв, розширення інтеграції з іншими сервісами Google, такими як Google Tasks або Google Keep, додавання функціональності повторюваних подій та нагадувань, впровадження аналітичних інструментів для відстеження

використання часу, та розширення можливостей категоризації та фільтрації даних. Загалом, розроблена система представляє собою функціональне та практичне рішення для оптимізації процесів особистого та сімейного планування, що відповідає сучасним технологічним стандартам та потребам користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.3.1 Нестеренко О. Б. Актуальні питання ефективного управління часом. International Forum: Problems and Scientific Solutions. CSIRO Publishing House ; Scientific Publishing Center "InterConf", 2021. С. 121-124. URL: https://er.knutd.edu.ua/bitstream/123456789/18931/3/P121-124.pdf (дата звернення: 15.04.2025).

1.3.2 Загороднюк О. В., Горбатюк М. І. Сучасні засоби планування часу менеджера. Редакційна колегія. 2021. С. 46-50. URL: https://mzedl.udau.edu.ua/assets/files/2021/21-st/zbirnik-1-ekciya-1.pdf#page=46 (дата звернення: 15.04.2025).

1.3.3 Данилевич Н., Давидюк Т. Управління робочим часом в сучасних умовах. III Всеукраїнська науково-практична інтернет-конференція «Управління». 2024. С. 84-87. URL: https://science.lpnu.ua/sites/default/files/attachments/2024/jun/34863/iiivnpikhrm.pdf#page=85 (дата звернення: 15.04.2025).

1.3.4 Берк Б. RESTful Java з JAX-RS 2.0. Київ : Вид-во Рідна мова, 2018. 394 с. URL: https://www.researchgate.net/publication/303121571_RESTful_Java_with_JAX-RS_20 (дата звернення: 15.04.2025).

1.3.5 Гречко Є. В. Розробка мобільного додатку для ефективного керування справами : кваліфікаційна робота бакалавра. Київ, 2024. URL: http://ds.knu.edu.ua/jspui/bitstream/123456789/6460/1/%D0%9A%D0%9C%D0%A0_%D0%93%D1%80%D0%B5%D1%87%D0%BA%D0%BE.pdf (дата звернення: 15.04.2025).

1.3.6 Мільков О. Г. Розробка кросплатформного мобільного додатку "Щоденник" : кваліфікаційна робота бакалавра. 2022. URL: https://ir.duan.edu.ua/items/cf2e1e85-1a51-4835-a334-ad634f163a8a (дата звернення: 15.04.2025).

1.3.7 Ульман Дж. Д., Відом Дж. Системи баз даних. Повний курс / пер. з англ. М. В. Назаров. Київ : Видавнича група BHV, 2018. 1088 с. URL: https://www.pearson.com/en-us/subject-catalog/p/database-systems-the-complete-book/P200000003037 (дата звернення: 15.04.2025).

1.3.8 Єршихін С. В. Комплексна система інтерактивного тайм-менеджменту : кваліфікаційна робота магістра. Київ, 2021. URL: https://ela.kpi.ua/server/api/core/bitstreams/3162e1f3-0f4d-4bc0-95d9-e23e9016cfd8/content (дата звернення: 15.04.2025).

1.3.9 Заказнюк Б. А. Використання Google Apps Script для розробки веб-застосунку календаря подій з інтеграцією Google Calendar : кваліфікаційна робота бакалавра. Тернопіль, 2024. URL: https://elartu.tntu.edu.ua/bitstream/lib/45738/1/2024_KRB_SN-41_Zakazniuk_B.A.pdf (дата звернення: 15.04.2025).

1.3.10 Google Calendar API: Official documentation. 2024. URL: https://developers.google.com/calendar/api/guides/overview (дата звернення: 15.04.2025).

1.3.11 Кільченко А. В., Климчук Д. М. Використання сервісу Google Календар для планування та організації науково-дослідної роботи в науковій установі. 2018. С. 18-23. URL: https://lib.iitta.gov.ua/id/eprint/711834/1/Kilchenko%20A.V._Klymchuk%20D.M._2018.pdf (дата звернення: 15.04.2025).

1.3.12 Мойсол Д. Д. Мобільний застосунок для тайм-менеджменту : кваліфікаційна робота бакалавра. Київ, 2024. URL: https://ela.kpi.ua/server/api/core/bitstreams/ed22b558-fa3f-40f6-8661-d707f3a0e51a/content (дата звернення: 15.04.2025).

1.3.13 Найдюк В. І., Черноволик Г. О. Розробка архітектури та алгоритмів сервісу синхронізації онлайн календарів : кваліфікаційна робота. Вінниця, 2024. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/43240/21304-75264-1-PB.pdf?sequence=1&isAllowed=y (дата звернення: 15.04.2025).

1.3.14 Рогаль П. В. Вебсервіс для обліку часу на основі Google Calendar : кваліфікаційна робота бакалавра. Київ, 2021. URL: https://ela.kpi.ua/server/api/core/bitstreams/f71b272f-38cc-463a-ad79-918f260dfc06/content (дата звернення: 15.04.2025).

1.3.15 Бавдик О. С. Розробка інформаційної система обліку, аналізу та прогнозування відвідування занять із застосуванням хмарних технологій : кваліфікаційна робота. Львів, 2024. URL: https://repository.lnup.edu.ua/jspui/bitstream/123456789/2385/1/Bavdyk.pdf (дата звернення: 15.04.2025).

1.3.16 Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol, CA : O'Reilly Media, 2022. 295 p. URL: https://www.oreilly.com/library/view/flask-web-development/9781491991725/ (дата звернення: 15.04.2025).

1.3.17 Закусило Т. М., Месюра В. І. Огляд сервісів, що реалізують організацію управлінням часу. Proceedings of the XII International scientific-practical conference «INTERNET-EDUCATION-SCIENCE»(IES-2020), Ukraine, Vinnytsia, 26-29 May 2020. Вінниця : ВНТУ, 2020. С. 36-39. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/30901/WORK-IES-2020-68-71.pdf?sequence=1&isAllowed=y (дата звернення: 15.04.2025).

1.3.18 Голубко А. І., Прищак М. Д. Тайм-менеджмент як дієвий інструмент ефективного використання часу в сучасних умовах : дис. Вінниця : ВНТУ, 2020. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/29693/9604.pdf?sequence=3 (дата звернення: 15.04.2025).

1.3.19 Мойсол Д. Д. Мобільний застосунок для тайм-менеджменту : кваліфікаційна робота бакалавра. Київ, 2024. URL: https://ela.kpi.ua/server/api/core/bitstreams/ed22b558-fa3f-40f6-8661-d707f3a0e51a/content (дата звернення: 15.04.2025).

1.3.20 Python Documentation. 2024. URL: https://docs.python.org/3/ (дата звернення: 15.04.2025).

1.3.21 Котельва М. А. Інформаційна система тайм-менеджменту користувачів : кваліфікаційна робота бакалавра. Суми, 2024. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/96910/1/Kotelva_bak_rob.pdf (дата звернення: 15.04.2025).

1.3.22 Колос М. М. Використання хмарного сервісу Google Calendar в освітньому процесі. Міжнародний науковий журнал «Інтернаука». 2023. С. 28-33. URL: https://d1wqtxts1xzle7.cloudfront.net/103428180/%D0%9F%D1%80%D0%BE%D1%81%D0%BA%D1%83%D1%80%D1%96%D0%B2%D1%81%D1%8C%D0%BA%D0%B8%D0%B8_%D0%BF%D0%BE%D0%B2%D1%96%D1%82_-libre.pdf?1686893186=&response-content-disposition=inline%3B+filename%3DTHE_PROSKURIV_DISTRICT_IN_THE_POST_REFOR.pdf&Expires=1744884319&Signature=RIZ8E5OXltMWg7FNHYWDXKKmRnjAjMVTx28aUDrxSx-r53pY-Juco93F3hS27B~rw0ivsHWzT5ea3ctDnfy71xu0aS8CNAHjaQc2UtVbh4ttDN6iR-Tm55GUlI4n29c2ok0YaJy7YOykrQE-EIOS-0k6Opk0cH6On480bT0-GFOx~mh4E5b9TDDSvkhXVnB7k3gnDyE21ntuSyUIqJOuPz4tYPH52LszyQ1m3TOjI8j41bTKpdkmNBq21rBb58pD2wXYsge~SVQrHKWi~iigsVNtvNspOXIyKfhgzGMGXGMDLR6C2eciRe95EVSZHBtZRrZZgh8G8-Zv3MxXQbFhYw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA#page=29 (дата звернення: 15.04.2025).

1.3.23 Ємельяненко Я. С. Вебзастосунок для планування роботи та концентрації : кваліфікаційна робота бакалавра. Київ, 2023. URL: https://ela.kpi.ua/server/api/core/bitstreams/b91ec28c-04e3-458f-b7d3-bfd046911e40/content (дата звернення: 15.04.2025).

1.3.24 Закусило Т. М. та ін. Розробка структури інформаційної технології організації особистого тайм-менеджменту. Вісник Вінницького політехнічного інституту. 2021. № 1. С. 70-76. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/31807/%D0%97%D0%B

0%D0%BA%D1%83%D1%81%D0%B8%D0%BB%D0%BE.pdf?sequence=1&isAllowed=y (дата звернення: 15.04.2025).

1.3.25 Романюк О. Розробка веб-застосування для планування дня : кваліфікаційна робота. Київ, 2022. URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/f29e349c-112e-4358-a2c9-570ec756e38a/content (дата звернення: 15.04.2025).

1.3.26 Іванов А. І. Вдосконалена система планування часу : магістерська дисертація. Запоріжжя, 2023. URL: https://eir.zp.edu.ua/server/api/core/bitstreams/3b531f59-12d5-4993-b3ea-366623ee3b7c/content (дата звернення: 15.04.2025).

1.3.27 Горелов А. М. Determining the set of the most useful functions for the system of planning and controling personal time. 2021. С. 57-57. URL: https://www.elibrary.ru/item.asp?id=46273466 (дата звернення: 15.04.2025).

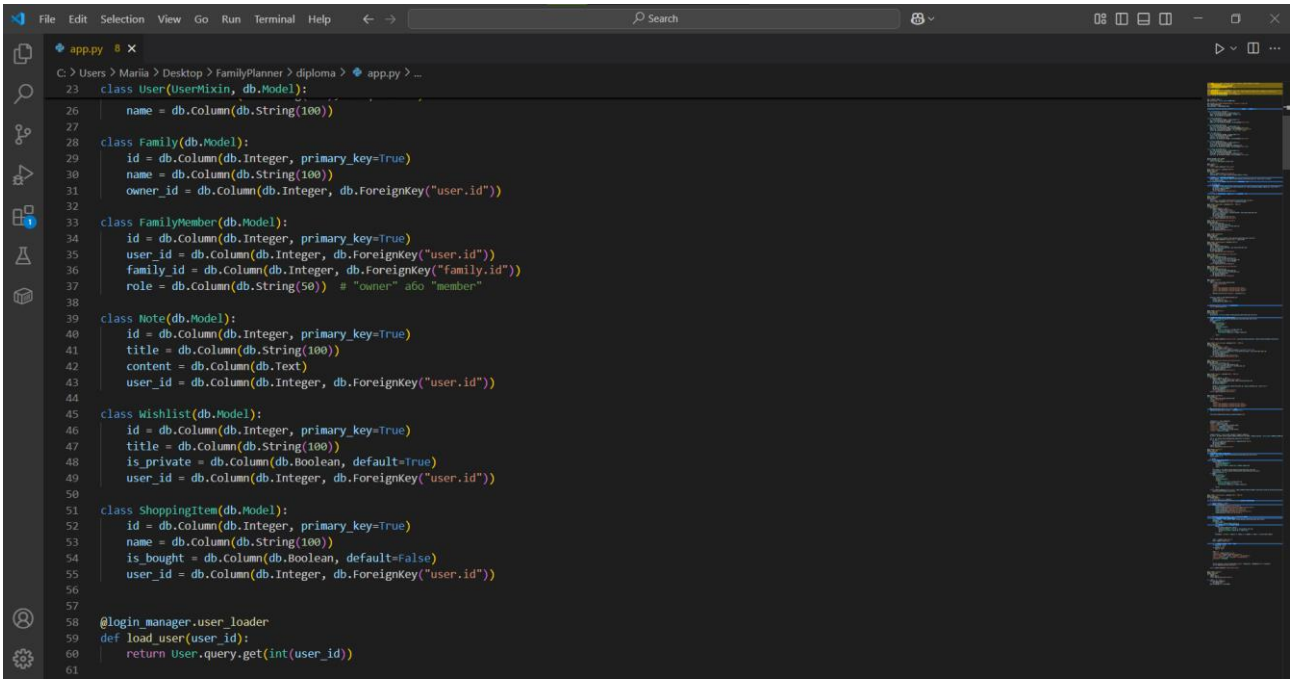
1.3.28 Дем'яник І. В., Романюк О. В. Вимоги до сучасних програмних застосунків для тайм-менеджменту : кваліфікаційна робота. Вінниця : ВНТУ, 2023. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/39294/17495.pdf?sequence=3&isAllowed=y (дата звернення: 15.04.2025).

1.3.29 Кучерявий В. Розробка веб-додатку з планування часу : кваліфікаційна робота. Київ, 2022. URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/a3105f26-40ff-4ea1-be37-bf22f5f26319/content (дата звернення: 15.04.2025).

1.3.30 Стамат В. М., Лазаренко А. В. Розвиток технологій тайм-менеджменту. 2024. URL: https://dspace.mnau.edu.ua/jspui/bitstream/123456789/19488/1/%D0%97%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA_%D0%9F%D0%94%D0%90%D0%A3_%D0%BE%D1%81%D1%96%D0%BD%D1%8C_2024_%20%281%29-1308-1310.pdf (дата звернення: 15.04.2025).

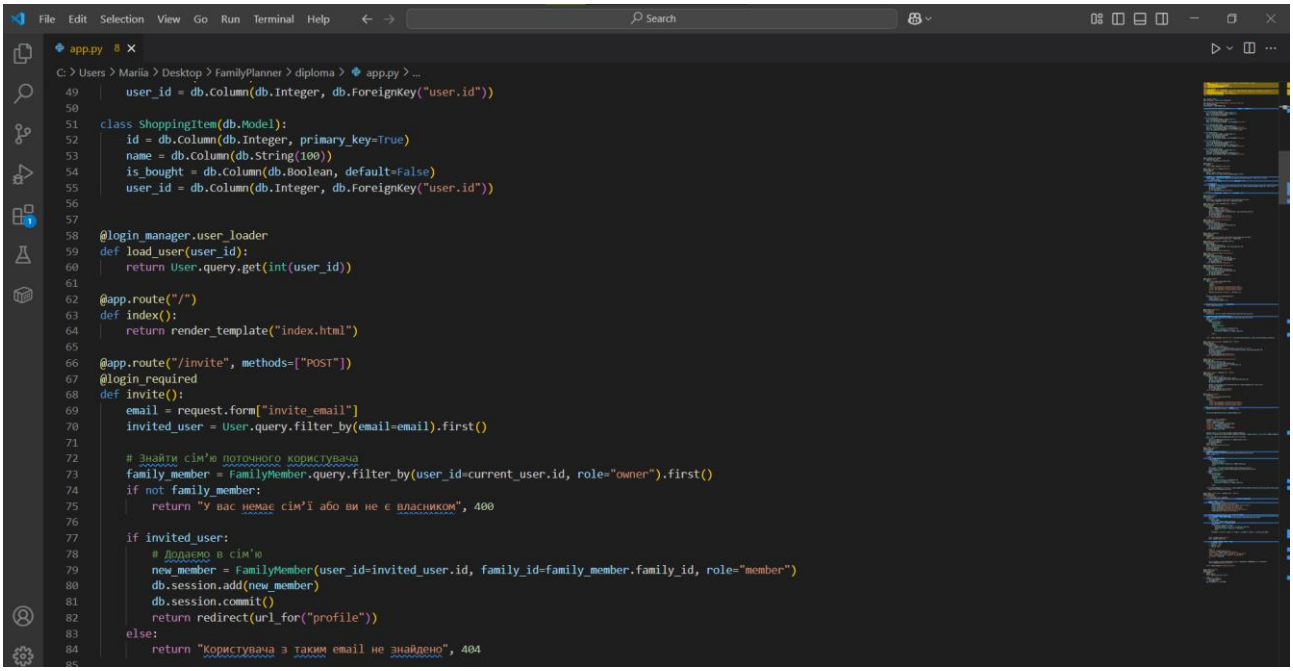
ДОДАТКИ

Додаток А «Фрагменти коду системи»



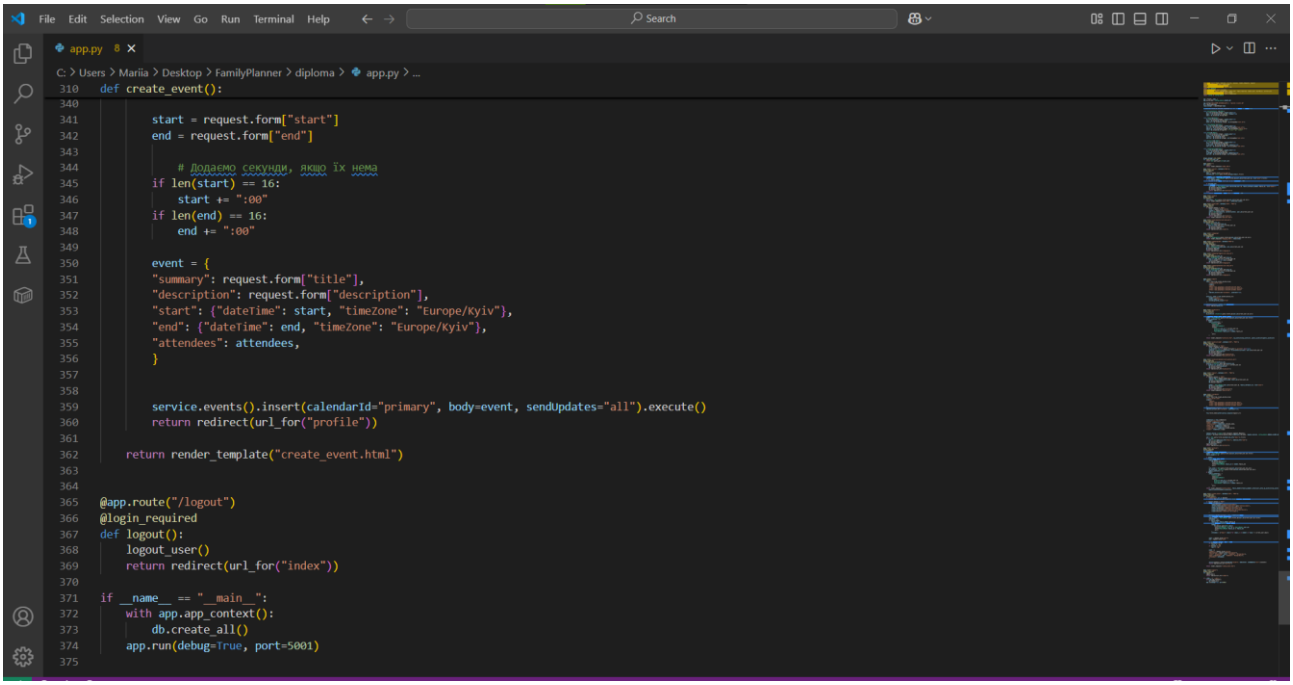
```python
class User(UserMixin, db.Model):
    name = db.Column(db.String(100))

class Family(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    owner_id = db.Column(db.Integer, db.ForeignKey("user.id"))

class FamilyMember(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))
    family_id = db.Column(db.Integer, db.ForeignKey("family.id"))
    role = db.Column(db.String(50))  # "owner" або "member"

class Note(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100))
    content = db.Column(db.Text)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))

class Wishlist(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100))
    is_private = db.Column(db.Boolean, default=True)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))

class ShoppingItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    is_bought = db.Column(db.Boolean, default=False)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Рисунок А.1 — Реалізація моделей даних (`User`, `Family`, `Wishlist` та ін.) за допомогою SQLAlchemy



```python
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))

class ShoppingItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    is_bought = db.Column(db.Boolean, default=False)
    user_id = db.Column(db.Integer, db.ForeignKey("user.id"))


@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/invite", methods=["POST"])
@login_required
def invite():
    email = request.form["invite_email"]
    invited_user = User.query.filter_by(email=email).first()

    # Знайти сім'ю поточного користувача
    family_member = FamilyMember.query.filter_by(user_id=current_user.id, role="owner").first()
    if not family_member:
        return "У вас немає сім'ї або ви не є власником", 400

    if invited_user:
        # Додаємо в сім'ю
        new_member = FamilyMember(user_id=invited_user.id, family_id=family_member.family_id, role="member")
        db.session.add(new_member)
        db.session.commit()
        return redirect(url_for("profile"))
    else:
        return "Користувача з таким email не знайдено", 404
```

Рисунок А.2 — Реалізація логіки запрошення користувача до сімейного акаунту



```python
def create_event():

        start = request.form["start"]
        end = request.form["end"]

            # Додаємо секунди, якщо їх нема
        if len(start) == 16:
            start += ":00"
        if len(end) == 16:
            end += ":00"

        event = {
        "summary": request.form["title"],
        "description": request.form["description"],
        "start": {"dateTime": start, "timeZone": "Europe/Kyiv"},
        "end": {"dateTime": end, "timeZone": "Europe/Kyiv"},
        "attendees": attendees,
        }


        service.events().insert(calendarId="primary", body=event, sendUpdates="all").execute()
        return redirect(url_for("profile"))

    return render_template("create_event.html")


@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for("index"))

if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True, port=5001)
```

Рисунок А.3 — Реалізація функції створення події та її синхронізації з Google Calendar

Додаток Б «Прототип системи»

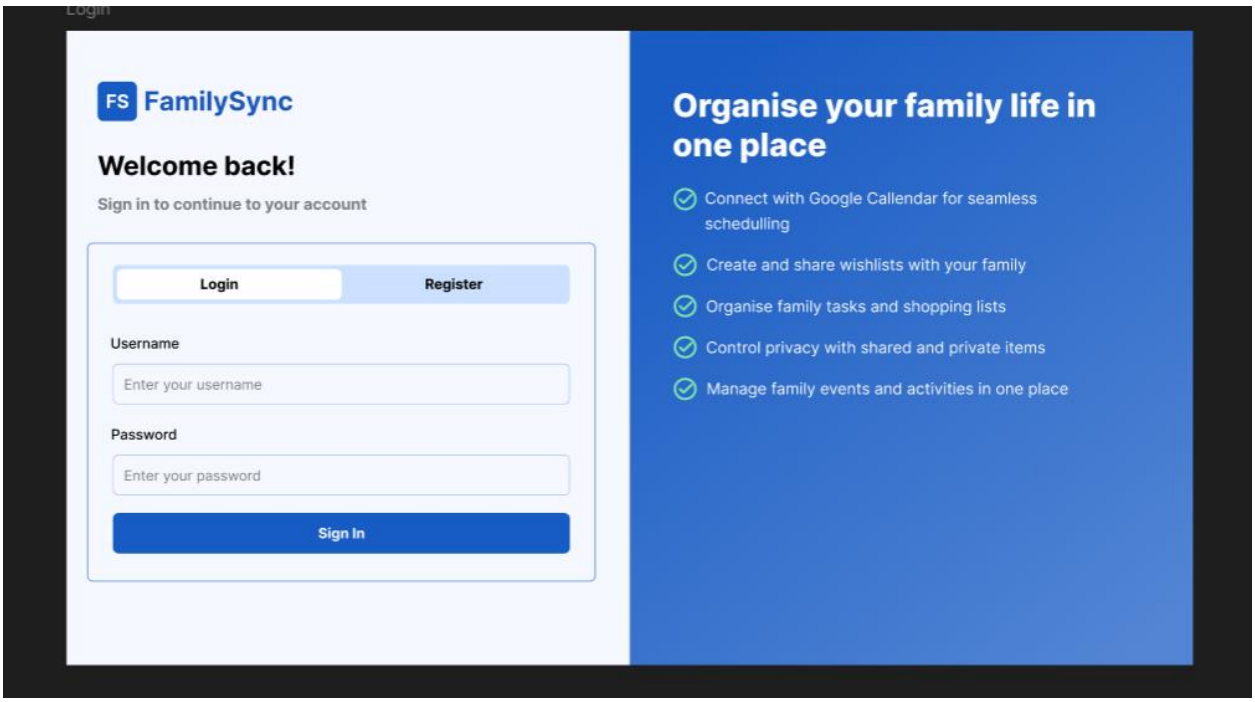


Рисунок Б.1 — Інтерфейс сторінки входу до системи

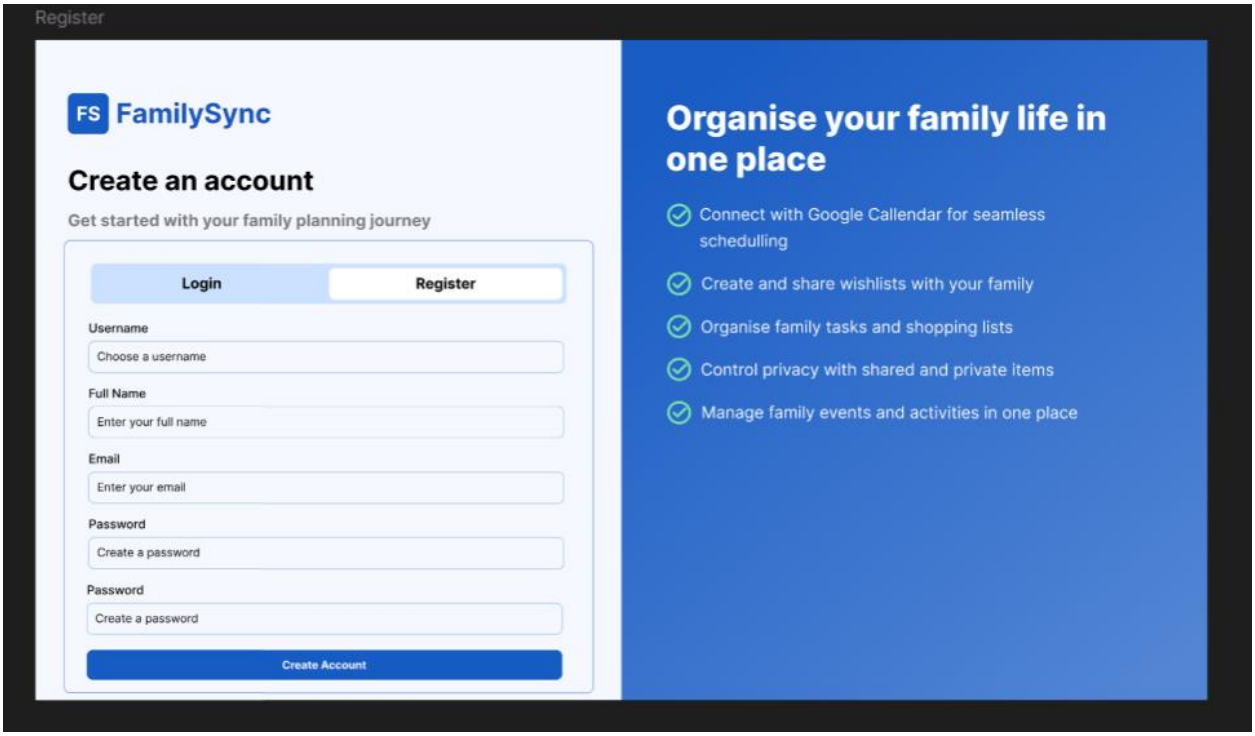


Рисунок Б.2 — Інтерфейс сторінки реєстрації нового користувача

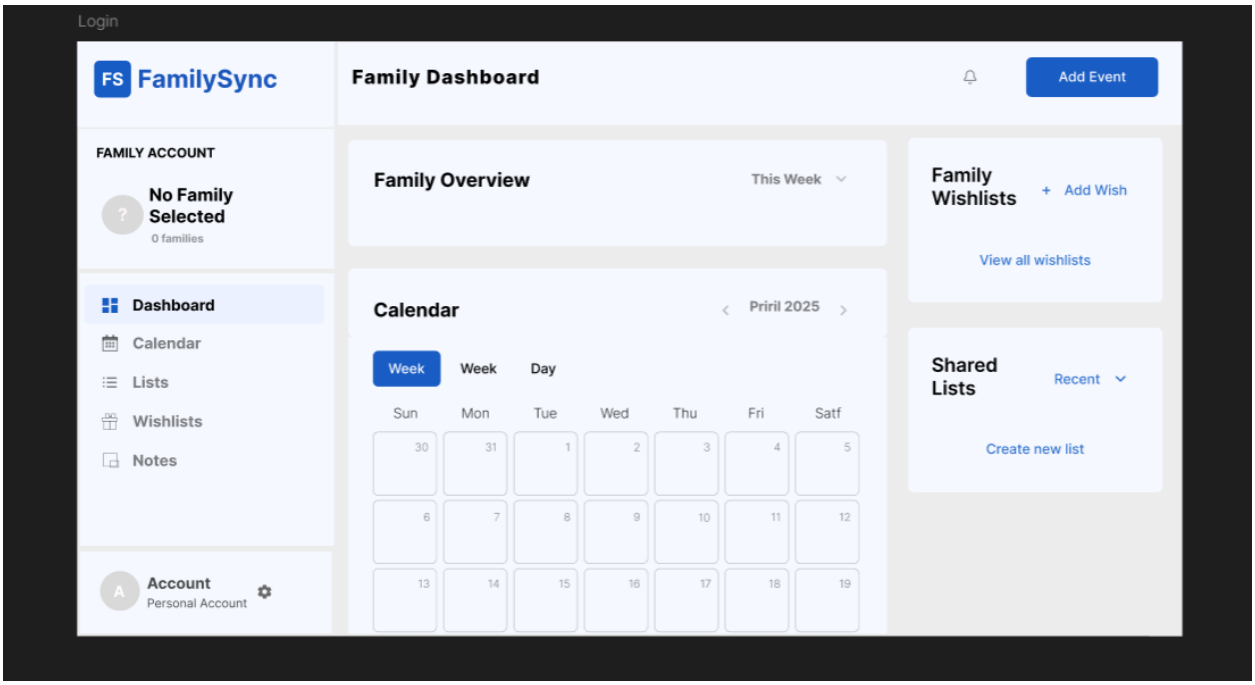


Рисунок Б.3 — Головна панель управління (Dashboard) системи

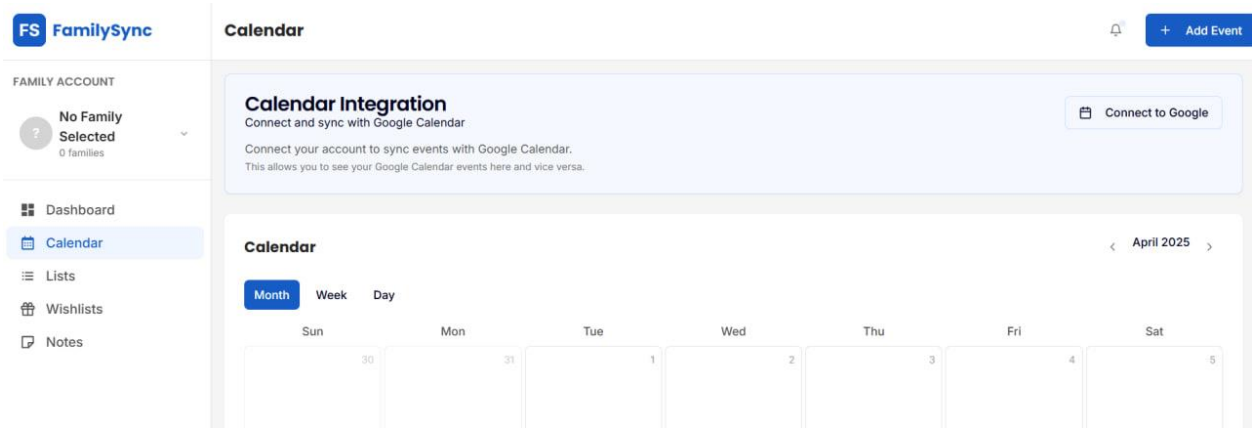


Рисунок Б.4 — Інтерфейс сторінки календаря та додавання подій

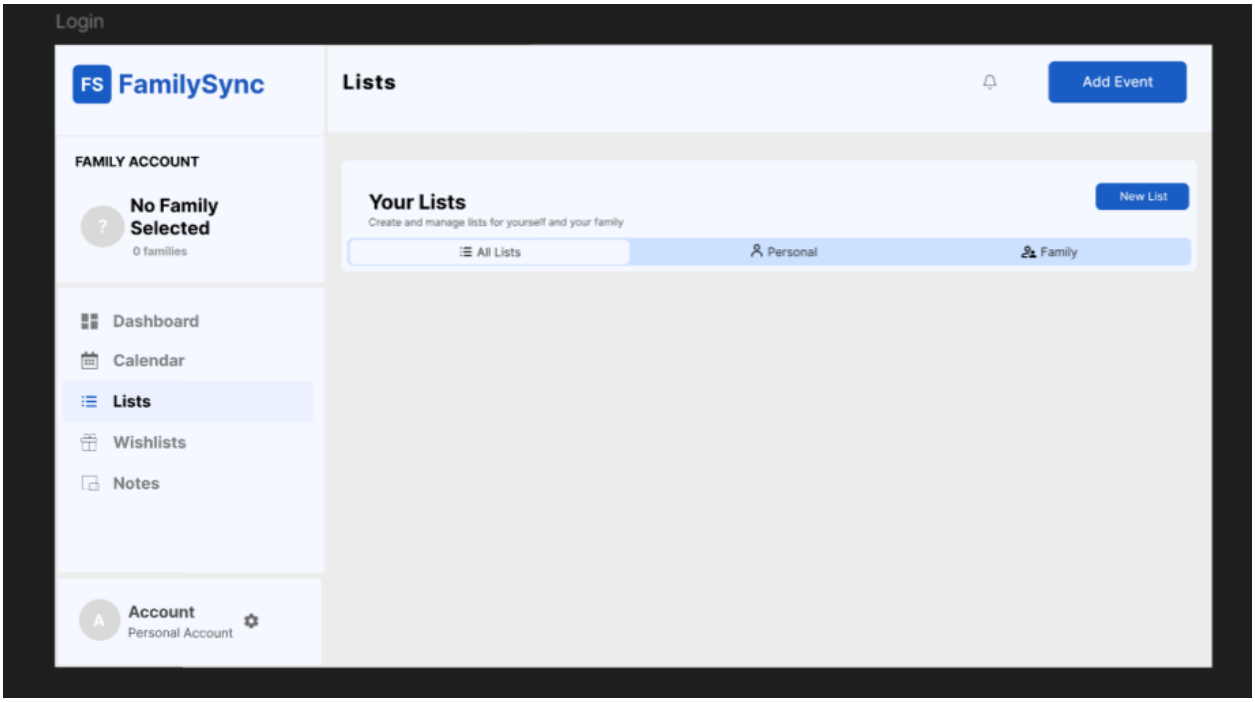


Рисунок Б.5 — Інтерфейс сторінки списків (Lists)

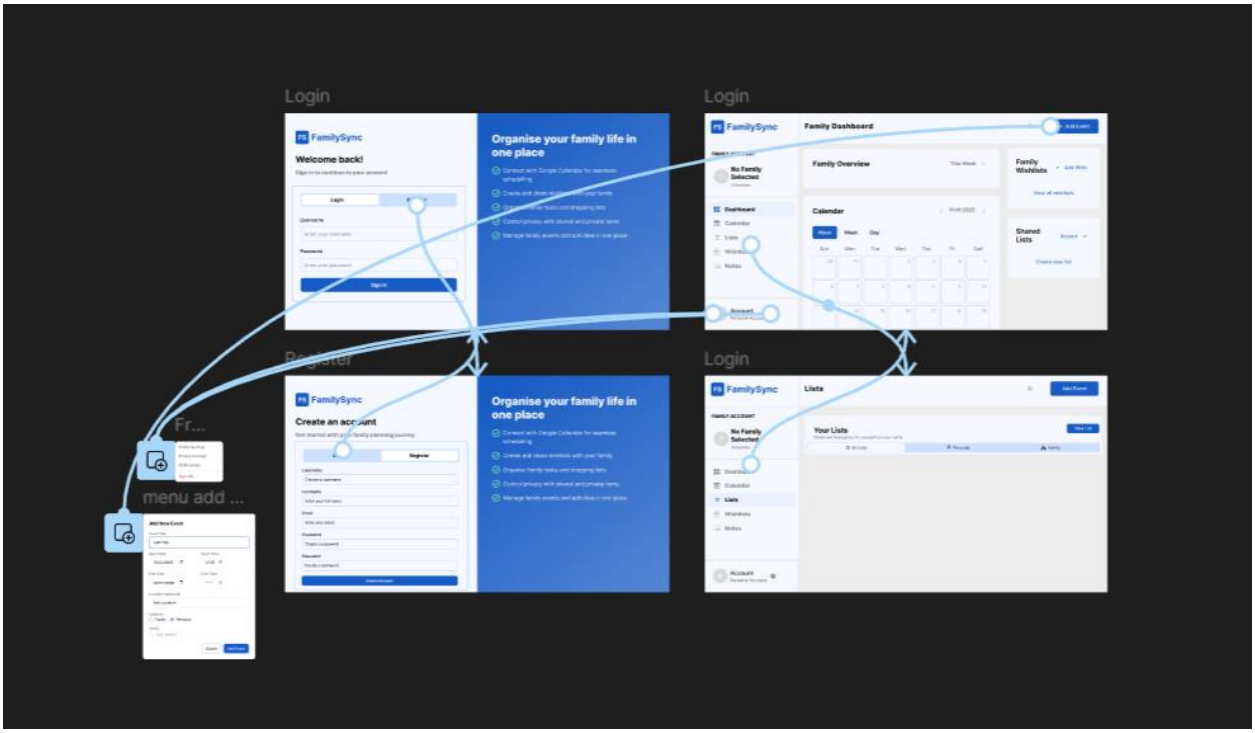


Рисунок Б.6 — Узагальнений вигляд зв'язків прототипу системи в застосунку Figma

Додаток В

StrikePlagiarism.com

Дата звіту 6/12/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU
Заголовок
ПРОЄКТУВАННЯ СИСТЕМИ ОНЛАЙН-ПЛАНУВАННЯ ОСОБИСТОГО ТА СІМЕЙНОГО ЧАСУ
Автор
Щетініна Марія ДмитрівнаШевченко Костянтин Леонідович
Науковий керівник / Експерт
підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.50% (1.50%) КП 1	0.00% (0.00%) КП 2	1.33% (1.33%) КЦ
25 Довжина фрази для коефіцієнта подібності 2	**14487** Кількість слів	**112648** Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		0
Інтервали		1
Мікропробіли		0
Білі знаки		0
Парафрази (SmartMarks)	a	14

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Розроблення інформаційної системи для торговельного підприємства [redacted] 6/3/2019 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	24 0.17 %
2	Розроблення інформаційної системи для торговельного підприємства [redacted] 6/3/2019 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	20 0.14 %