**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Основи Веб-програмування»**

**тема: «Додаток для календаря»**

***2025***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

**Кафедра інженерії програмного забезпечення в енергетиці**

***КУРСОВА РОБОТА***

**з дисципліни:**

**«Розробка програмного забезпечення мобільних пристроїв»**

**тема: Тестова документація проекту**

**«Додаток для календаря»**

***2024***

**НAЦIOНAЛЬНИЙ ТEXНIЧНИЙ УНIВEРCИТEТ УКРAЇНИ**

**«КИЇВCЬКИЙ ПOЛIТEXНIЧНИЙ IНCТИТУТ ІМЕНІ IГOРЯ CIКOРCЬКOГO»**

**Кафедра інженерії програмного забезпечення в енергетиці**

КУРСОВА РОБОТА

**з дисципліни**: « Основи Веб-програмування »

на тему: «Додаток для календаря»

Студента 2 курсу групи ТВ-31

напряму підготовки ***121 Програмна інженерія***

\_\_\_\_\_\_\_\_\_ Гогуля М. Д.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_д.т.н., доцент, Недашківський О. Л. \_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Член комісії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Посилання на **GitHub** репозиторій <https://github.com/Foresttm23/Calendar-App.git>

Київ – 2025 рік

**Національний технічний університет України**

**“Київський політехнічний інститут імені Ігоря Сікорського”**

Навчально-науковий інститут атомної та теплової енергетики \_

( повна назва )

Кафедра інженерії програмного забезпечення в енергетиці\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

( повна назва )

Напрям підготовки \_121\_\_\_\_ Програмна інженерія\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва)

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Гогулі Максиму Дмитровичу

(прізвище, ім’я, по батькові)

1. Тема роботи – «Додаток для календаря»

Керівник курсової роботи – Недашківський О.Л.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи: «26» грудня 2025 р.

3. Вихідні дані до проекту (роботи): мови програмування: Python(Flask), JS; Дизайн: HTML, CSS.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): розробити веб-додаток календаря.

5. Дата видачі завдання: 14 січня 2025

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |
| --- | --- | --- |
| Період  виконання | Назва етапу роботи | Максимальний бал |
| 1 тиждень | Розподілення на команди, отримання теми та варіанту на курсову роботу та формулювання основного завдання | - |
| 2 тиждень | Розділ 1. Створення діаграм компонентів, взаємодії та класів веб додатка | 5 |
| 3 тиждень | Розділ 2. Проектування структури меню з урахуванням потреб користувача | 5 |
| 4-5 тижні | Розділ 3. Розміщення та стилізація елементів інтерфейсу | 5 |
| 6 тиждень | Розділ 4. Проектування та реалізація моделей даних для збереження інформації | 5 |
| 7 тиждень | Розділ 5. Розробка механізмів отримання та оновлення даних | 5 |
| 8 тиждень | Розділ 6. Розробка бази даних для збереження та отримання інформації | 5 |
| 9-10 тижні | Розділ 7. Інтеграція сторонніх сервісів в систему | 5 |
| 11-12 тижні | Розділ 8. Створення серверної архітектури додатку | 5 |
| 13 тиждень | Розділ 9. Реалізація обробки запитів та взаємодії з базою даних на сервері | 5 |
| 14 тиждень | Розділ 10. Тестування та налаштування веб додатку на сервері | 5 |
| 15 тиждень | Оформлення записки до курсової роботи | 10 |
| 16 тиждень | Захист курсової роботи | 40 |

**Студент \_\_\_\_\_\_\_\_\_** Гогуля М. Д.

( підпис ) (прізвище та ініціали)

**Керівник курсової роботи \_\_\_\_\_\_\_\_\_** Недашківський О. Л.

( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

У курсовій роботі розроблено веб-додаток календаря, призначений для зручного планування подій та управління особистим часом користувача. Основна мета проекту - створити функціональний та зрозумілий інтерфейс для перегляду подій у різних режимах: день, тиждень, місяць, рік. Для реалізації серверної частини використано мову програмування Python та фреймворк Flask. Клієнтська частина розроблена з використанням HTML, CSS і JavaScript. В ході виконання проекту було зроблено авторизацію через Google-акаунт, інтеграцію з Google Calendar API та можливість додавання, редагування й видалення подій. Додаток забезпечує динамічне відображення подій у календарі та адаптивність до різних пристроїв.

Обсяг пояснювальної записки 132 аркуші, кількість ілюстрацій - 3, 2 додатки.

# ANNOTATION

In the course work, a calendar web application was developed, designed for convenient event planning and personal time management of the user. The main goal of the project is to create a functional and understandable interface for viewing events in different modes: day, week, month, year. The Python programming language and the Flask framework were used to implement the server part. The client part was developed using HTML, CSS and JavaScript. During the project, authorization was made via a Google account, integration with the Google Calendar API and the ability to add, edit and delete events. The application provides dynamic display of events in the calendar and adaptability to different devices.

The volume of the explanatory note is 132 sheets, the number of illustrations is 3, 2 appendices.

**ЗМІСТ**

[АНОТАЦІЯ 2](#_Toc199025119)

[ANNOTATION 2](#_Toc199025120)

[ВСТУП 4](#_Toc199025121)

[Розділ 1. Створення діаграм для опису системи. 5](#_Toc199025122)

[У Фронт-енді було зроблено: 9](#_Toc199025123)

[У Бек-енді було зроблено: 11](#_Toc199025124)

[Розділ 2. Проектування структури меню з урахуванням потреб користувача 13](#_Toc199025125)

[Розділ 3. Розміщення та стилізація елементів інтерфейсу 25](#_Toc199025126)

[Розділ 4. Проектування та реалізація моделей даних для збереження інформації. 27](#_Toc199025127)

[Розділ 5. Розробка механізмів отримання та оновлення даних. 29](#_Toc199025128)

[Розділ 6. Розробка бази даних для збереження та отримання інформації. 34](#_Toc199025129)

[6.1 PostgreSQL 34](#_Toc199025130)

[6.2 Redis 36](#_Toc199025131)

[Розділ 7. Інтеграція сторонніх сервісів. 37](#_Toc199025132)

[7.1 Google Calendar API 37](#_Toc199025133)

[7.2 Хостинг і база даних на Render 38](#_Toc199025134)

[Розділ 8. Створення серверної архітектури додатку. 39](#_Toc199025135)

[Розділ 9. Реалізація обробки запитів та взаємодії з базою даних на сервері. 42](#_Toc199025136)

[Розділ 10. Тестування та налаштування веб-додатку на сервері. 48](#_Toc199025137)

[Бек-енд 48](#_Toc199025138)

[Фронт-енд 52](#_Toc199025139)

[ВИСНОВКИ 57](#_Toc199025140)

[CПИCOК ВИКOРИCТAНИX ДЖEРEЛ 58](#_Toc199025141)

[ДОДАТОК 1 59](#_Toc199025142)

[ДОДАТОК 2 63](#_Toc199025143)

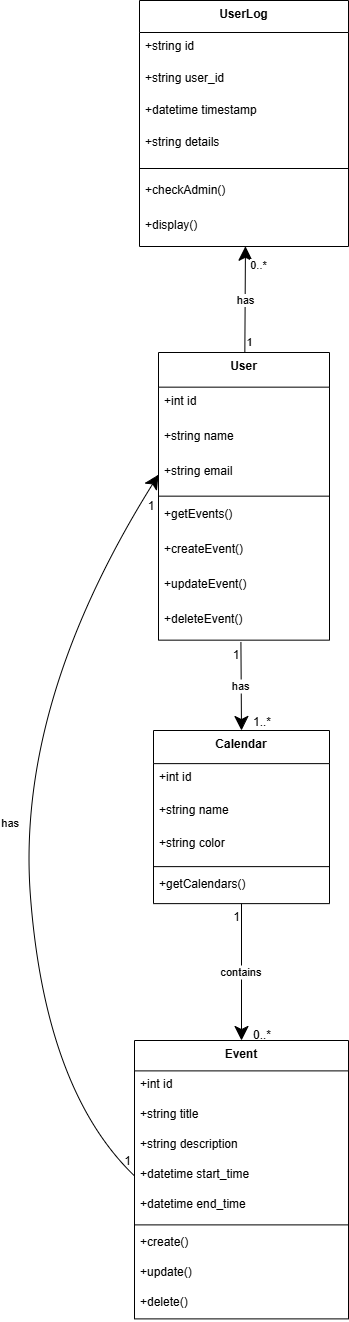
# ВСТУП

У сучасному світі головним ресурсом є час, тому правильне управління ним є ключовим елементом продуктивності. З метою оптимізації планування та організації повсякденного життя було створено веб-додаток календаря.

Основна функція додатку - допомогти користувачу ефективніше планувати свій час. Додатком зручно користуватися: він дозволяє створювати й редагувати події, а також інтегрується з сервісами Google, надсилаючи повідомлення про заплановані події. Модуль авторизації передбачає вхід через Google-акаунт, який, за потреби, можна створити безпосередньо у вікні входу. Це забезпечує як безпеку даних, так і зручність у використанні.

Актуальність цього додатку обумовлена постійною потребою в організації та ефективному управлінні часом у мінливому світі сьогодення. Інтеграція з онлайн-сервісами, такими як Google Calendar, а також розширені можливості планування роблять цей додаток корисним помічником для кожного, хто прагне жити продуктивно та організовано.

# Розділ 1. Створення діаграм для опису системи.

  
Рисунок 1.1 Діаграма класів.

**Діаграма класів – це** діаграма UML, що описує статичну структуру системи, її класи, атрибути, методи та зв’язки між класами.

В моїй діаграмі класів для календаря **(рис. 1.1),** є головний клас User, що відображає користувача системи, дані про нього, та його можливості (створення, редагування видалення подій у його календарях).

Залежним від класа User, є клас Calendar, він відображає Всі події, що були йому присвоєні класом User.

Відповідно клас Calendar містить в собі клас Event, що зберігає дані про окремі події та їх дані, наприклад заголовок та опис події, дату початку та кінця цієї події.

Останнім класом є клас UserLog, він зберігає в собі історію помилок користувача, час, коли це сталось та відповідно опис помилки, що виникла. Для зручності, можлива реалізація функції display(), що буде працювати тільки для розробника за допомогою checkAdmin().

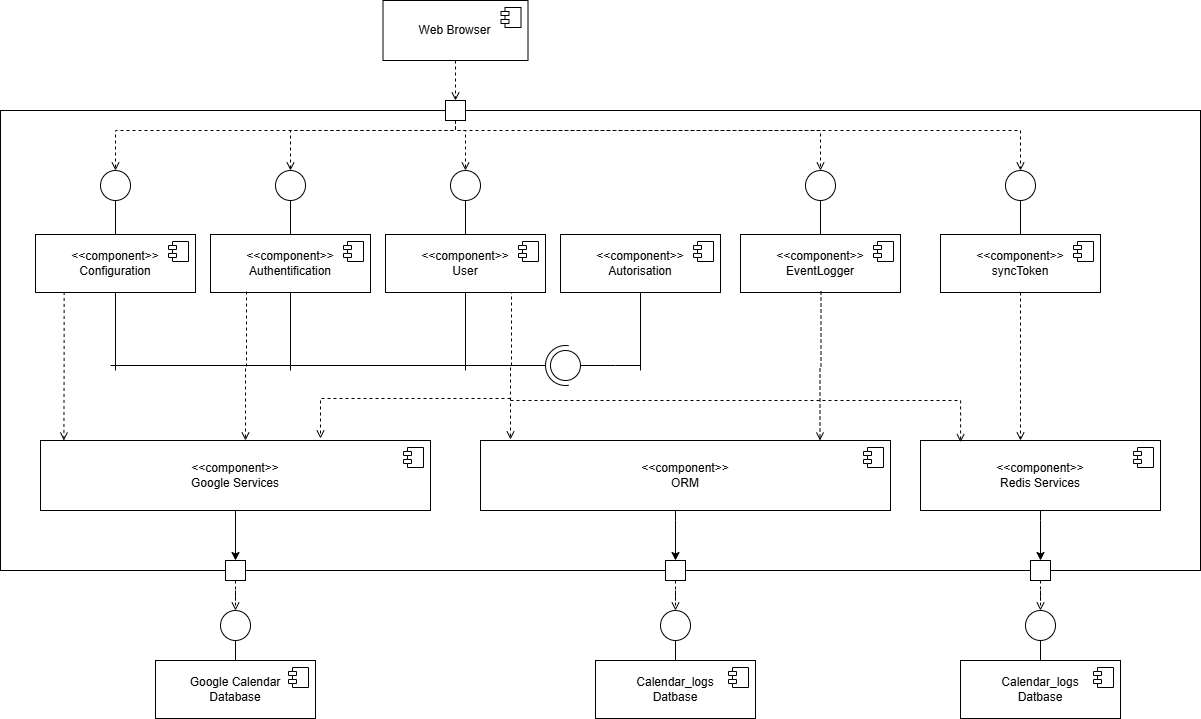
  
Рисунок 1.2 Діаграма Use Case.

**Діаграма Use Case – це** діаграма UML, що показує взаємодію користувача із системою.

В моїй діаграмі випадків використання для календаря **(рис. 1.2),** користувач має такі варіанти використання:

1. Зареєструватись в аккаунт
2. Переглянути календарі, обрати які календарі зображати, а які сховати.
3. Створити подію для певного календаря.
4. Видалити чи оновити подію в календарі.

Також після кожної успішної дії виводиться повідомлення користувачу про це.

  
Рисунок 1.3 Діаграма Компонентів.

**Діаграма компонентів – це** діаграма UML, що показує архітектуру системи, зосереджуючись на компонентах та їх взаємодії через інтерфейси.

В моїй діаграмі компонентів **(рис. 1.3)** є головний елемент системи,   
Web Browser, він пропонує такі інтерфейси:

* Configuration - налаштовує взаємодію системи
* Authentification – відповідає за автентифікацію користувача після авторизації
  + Autorisation – авторизує користувача в системі
* User – інформація про користувача, його імейл та дозвіл на доступ до його інформації.
* EventLogger – відповідає за збереження помилок користувача.
* SyncToken – відповідає за отримання тільки новий подій, а не всього списку.

Наступними важливими компонентами є Google Services, ORM та Redis Services, які забезпечують передачу інформації до відповідних баз даних.

Redis допомагає в збереженні тимчасових токенів користувача.

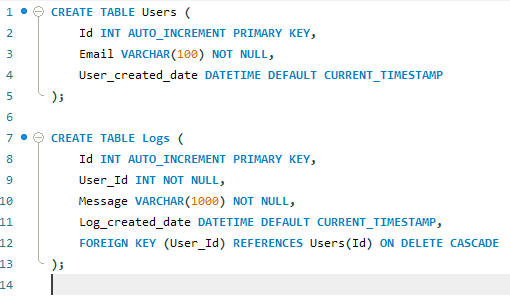
Google Services для збереження та оновлення подій.

Та власна ORM для логування помилок як системи, так і користувача.

Ці компоненти потребують доступу до інтерфейсу User, після чого:

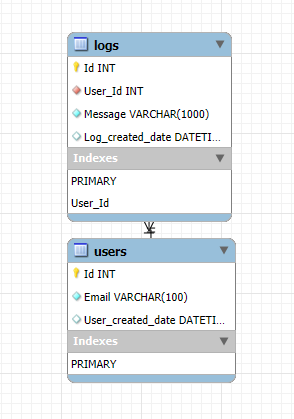
* Google Services передає дані в Google Calendar DB.
* ORM в Calendar\_logs DB.
* Redis в Redis DB

Для власної бази даних було розроблено таблиці, що допомагають відлагоджувати додаток:

  
Рисунок 1.4 MySQL код схеми Бази даних **(Calendar\_logs)**.

Таблиця Users **(рис 1.4)** Зберігає дані про користувача, а таблиця Logs зберігає відповідно дані логів до кожного користувача.

**На рисунку 1.5**  можна побачити Схему БД.

  
Рисунок 1.5 Схема Бази даних **(Calendar\_logs).**

В межах додатку було розроблено фронт-енд, частину з якою взаємодіє користувач та бек-енд, частину яка відповідає за зв’язок між сервером та інформацією, що отримує користувач.

## У Фронт-енді було зроблено:

1. Зручний інтерфейс з підібраними кольорами та розмірами, що гарно відображають додаток як на комп’ютерах (ширина ~1000 пікселів) та на телефонах (ширина ~350 пікселів). На телефонах також було вимкнено анімації, для зменшення навантаження.
2. Вся візуальна частина була зроблена з використанням стандартних **HTML** та **CSS.**
3. Відображення подій на розробленому власноруч календарі. Для цього було написано функції для розрахунку розмірів подія та ячейки, куди вона буде розміщуватись. Це було зроблено для таких 4 варіантів відображень:
   1. Рік, користувач може гортати сторінки років, перемкнутись на конкретний день, просто натиснувши на відповідний день, та може повернутись на поточний, якщо натисне на кнопочку «Сьогодні».
   2. Місяць, так само користувач може обрати бажаний день і перемикатись.
   3. Тиждень, тут користувач вже має доступ до подій та Українських вихідних днів, що були окремо підв’язані з Гугл календаря. Тут користувач також може:
      1. обрати події, що вже створена для її перегляду.
      2. Змінити існуючу подію
      3. Видалити подію
      4. Створити подію:
         1. Варіант 1: натиснути на «плюсик» зправа внизу та через меню створити подію.
         2. Варіант 2: натиснути в бажаному місці на календарі, й з’явиться вікно для створення події, дата початку буде, місце куди натиснув користувач, кінець через 30 хвилин.
   4. День, все від тиждня, але для одного дня. Функція побудови одна й та сама.
4. Можливість вибору календарів для відображення. Тобто користувач може обрати, події яких календарів необхідно зобразити на календарі, а які сховати.
5. Зліва вверху було зроблено відображеня поточного року та місяця.
6. Кнопку авторизації в аккаунт гугл.
7. Універсальні кнопки перемикання між датами (рік, місяць..).
8. Створення події в обраному користувачем зі списку календарі.
9. Вивід помилок по типу, помилка створення події, на головну сторінку календаря.
10. Підвантаження календаря, якщо дата наблизилась до нового року, та подальше кешування цих подій, щоб знову їх не отримувати.
11. Кешування всіх подій, всіх вихідних днів, обраних календарів та всіх календарів.

## У Бек-енді було зроблено:

1. Отримання подій з Гугл календаря за 1 рік, для цього було реалізовано:
   1. Отримання всіх календарів користувача.
   2. Отримання «syncToken» від Гугл календаря, або з бази даних «Redis» для завантаження тільки оновлень, а не всіх подій кожного разу.
   3. Збереження «syncToken» протягом 12 годин в Базі даних «Redis», після цього користувач автоматично отримає всі події за 1 рік.
   4. Підготовка даних, розбиття довгих подій на елементи для відображення в календарі, бо календарб генерується по днях.
   5. Повернення цих результатів зі статусом «SyncToken» (повне підвантаження чи часткове), для кожного календаря окремо.
2. Створено зовнішню базу даних та функцію для легкого запису
3. Додано логування всіх помилок з сервера в базу даних, на фронт-енді їх виводимо. Для цього було використано зовнішній сервіс Render, для надання послуг хостингу бази даних.
4. Отримання вихідних свят України.
5. Видалення всіх «SyncToken» з «Redis» після виходу с аккаунта, щоб в разі помилок, користувач міг просто перезайти в додаток.
6. Налаштовано АПІ ключі та налаштування «scopes» для доступу до подій календаря Гугл.
7. Налаштовано Логгер помилок в терміналі на ранніх стадіях додатка.
8. Була спроба реалізації вебхуків для календаря, але оскільки сенс від вебхука є тільки у випадку використання Веб-сокетів, ця реалізація не з’явилась у фінальній програмі.
9. Отримання даних користувача від Гугл після авторизації користувача.
10. Запущено додаток на онлайн веб сервісі Render (**Посилання**: [Calendar](https://coursework-web-jpy4.onrender.com/)).

# Розділ 2. Проектування структури меню з урахуванням потреб користувача

Загальний вигляд додатку на комп’ютері (Рис 2.1)

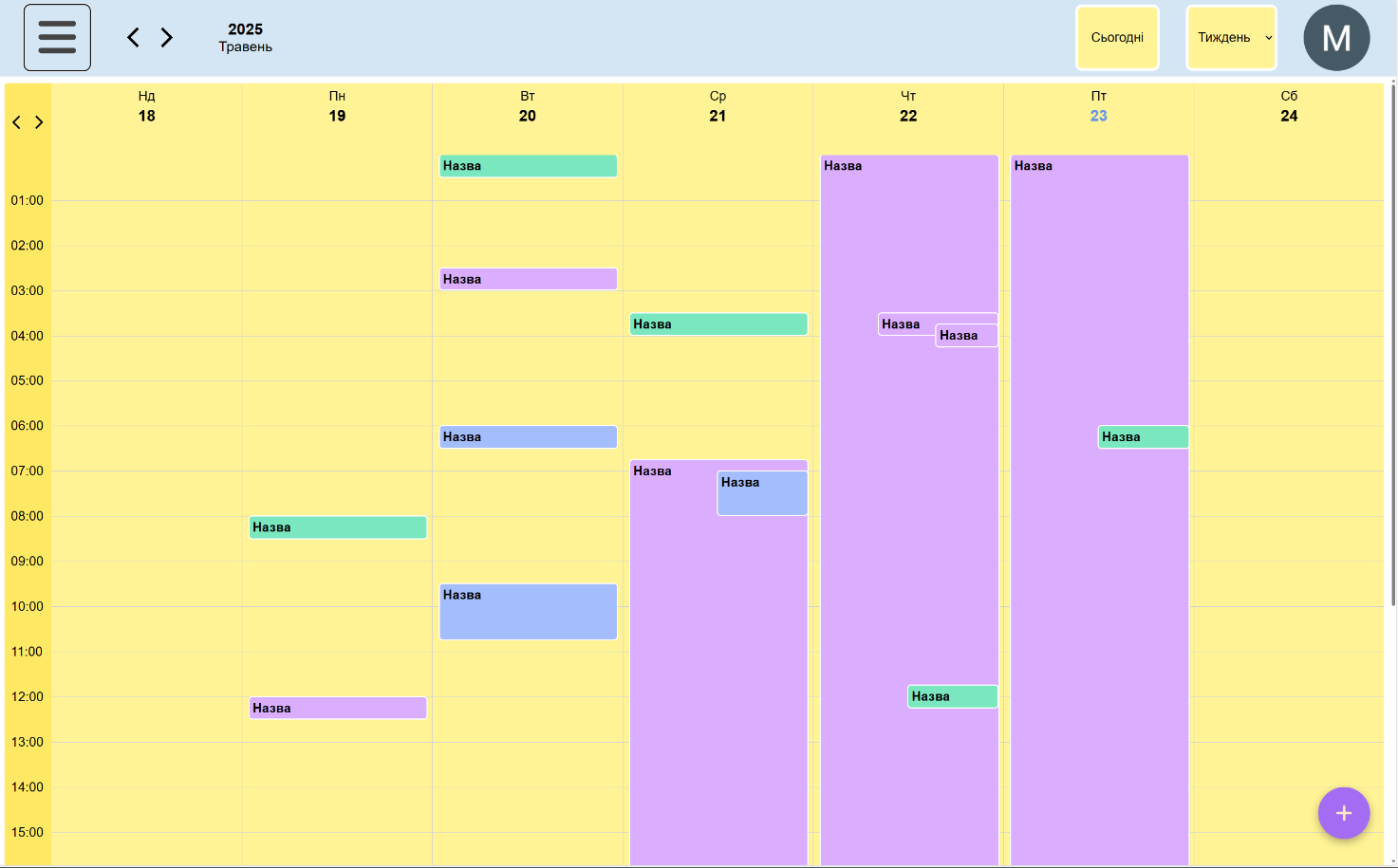


Рис 2.1 Загальний вигляд додатку на комп’ютері

Загальний вигляд додатку на Телефоні **(Рис 2.2)**

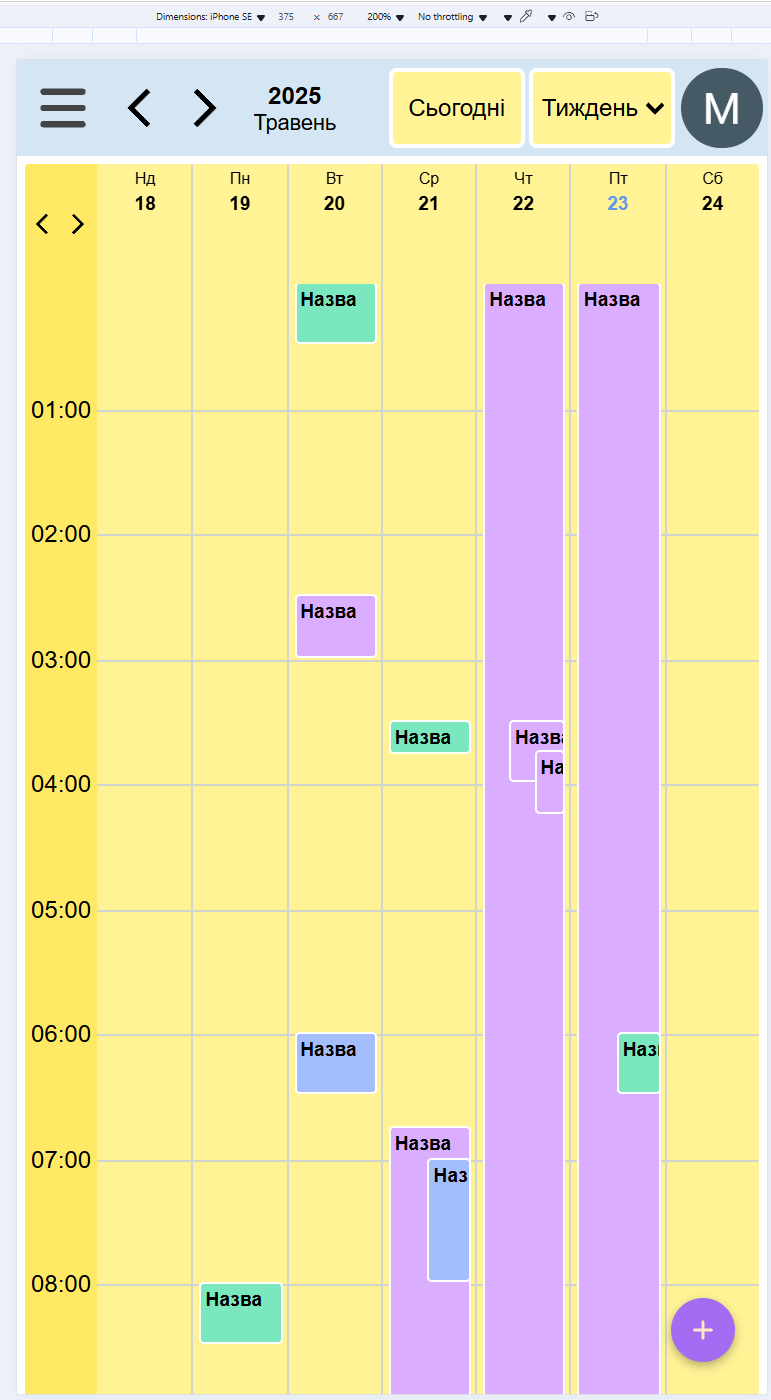


Рис 2.2 Загальний вигляд додатку на телефоні

**Додаток** складається з таких головних частин:

* **Верхня панель (Рис. 2.3)**
  + Меню календарів **(Рис. 2.4)**
    - Відкриває панель для доступу до календарів
      * Перегляд календарів
      * Вибір календаря для відображення
  + Стрілки для перемикання між датами в календарі.
  + Вивід поточної відображеної дати у форматі Рік, місяць
  + Кнопка «Сьогодні», що перемикає користувача на поточну дату.
  + Випадаючий список для вибору режиму відображення (рік, місяць..)
    - Для переключення між режимами
  + Профіль користувача, що заміняється логотипом Гугл, коли користувача не авторизований.

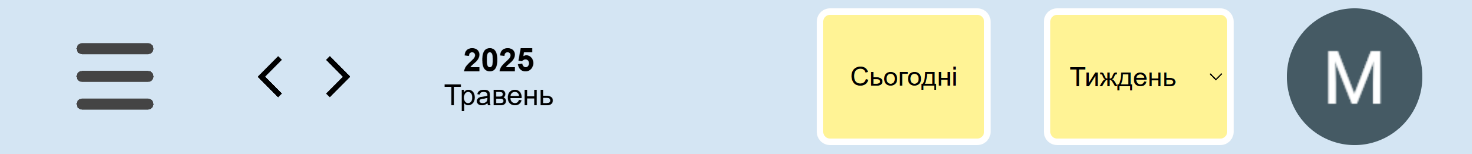


Рис 2.3 Верхня панель взаємодії

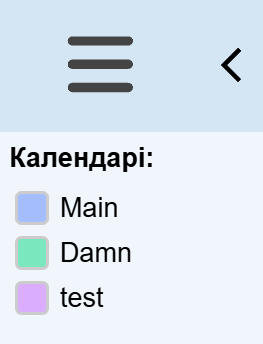


Рис 2.4 Відкрите меню

* **Календар (Рис. 2.5)**
  + Відображення календаря (рік, місяць...) **(Рис. 2.6 – 2.9)**
    - Перегляд подій користувача
      * При натисканні на обрану подію вилазить меню, де можна:
        + Видалити подію **(Рис. 2.10)**
        + Оновити подію **(Рис. 2.11)**
      * При натисканні на кнопку в правому нижньому куті **(Рис. 2.14)**, вилазить меню:
        + Створення подій **(Рис. 2.12)**
    - Перегляд Українських свят **(Рис. 2.13)**
    - Перегляд поточного дня **(має відмінний від інших колір)**

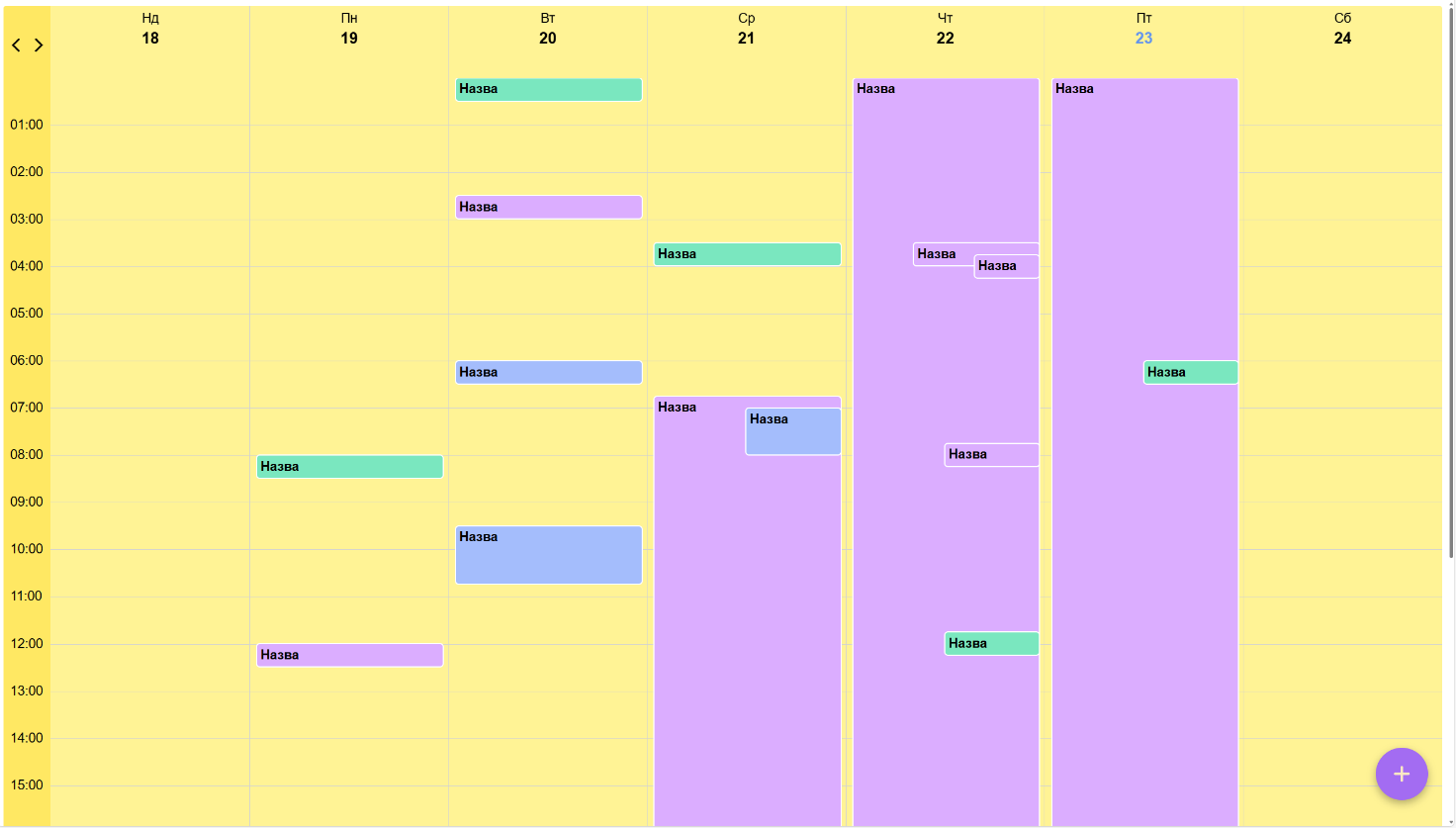


Рисунок 2.5 Календар

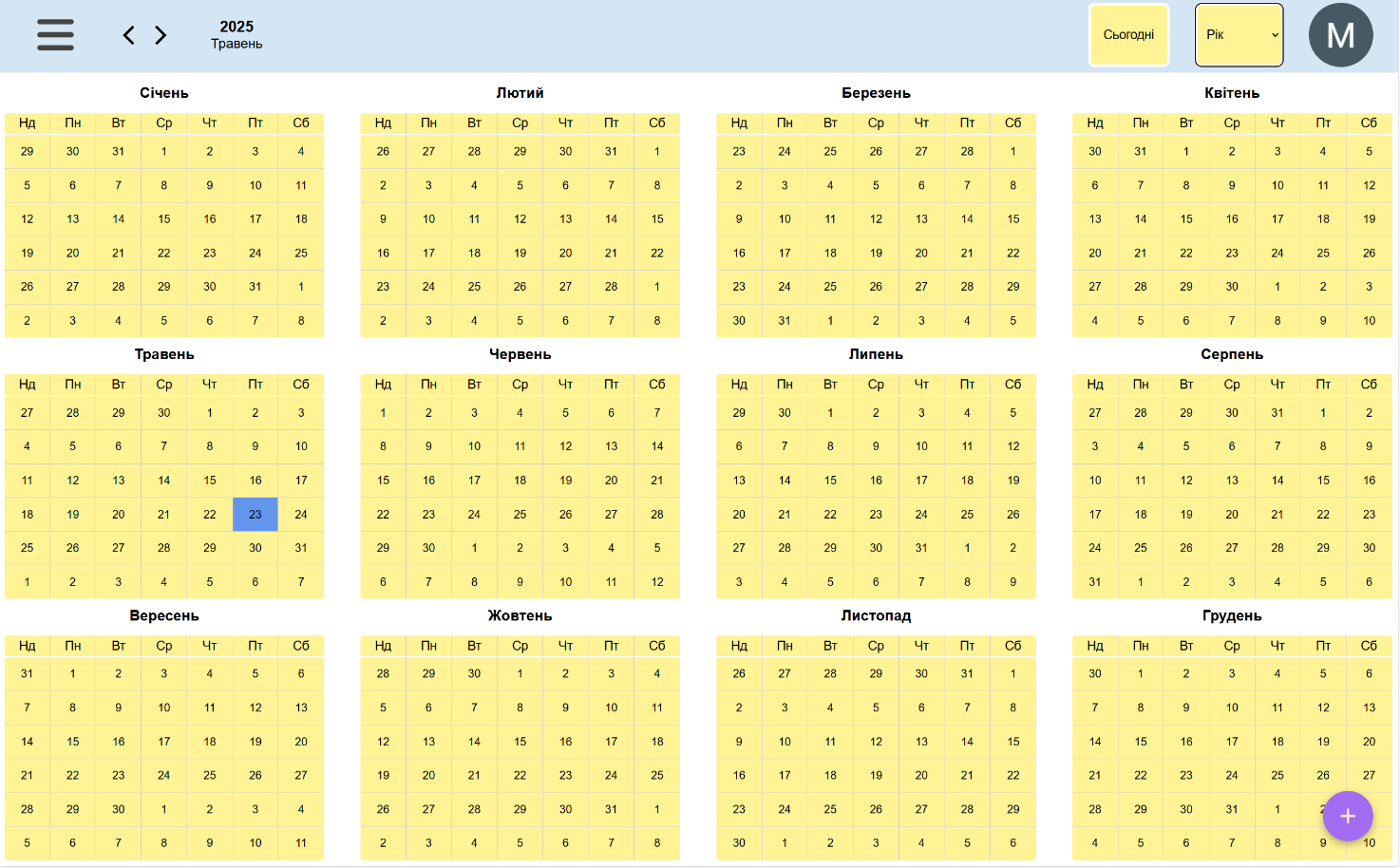


Рисунок 2.6 Додаток, відображення року **(кожна дата є активною кнопкою)**

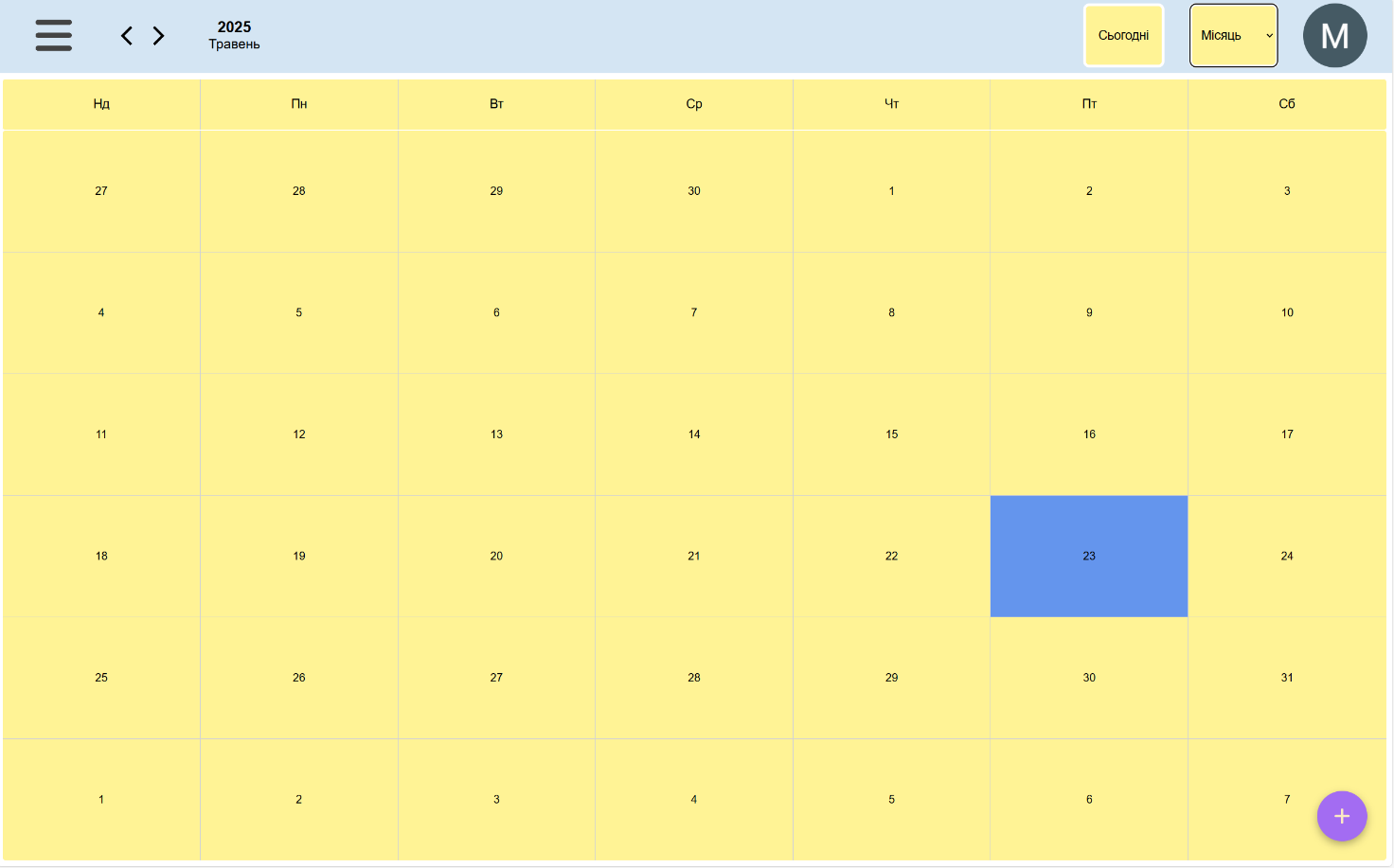


Рисунок 2.7 Додаток, відображення місяця **(кожна дата є активною кнопкою)**

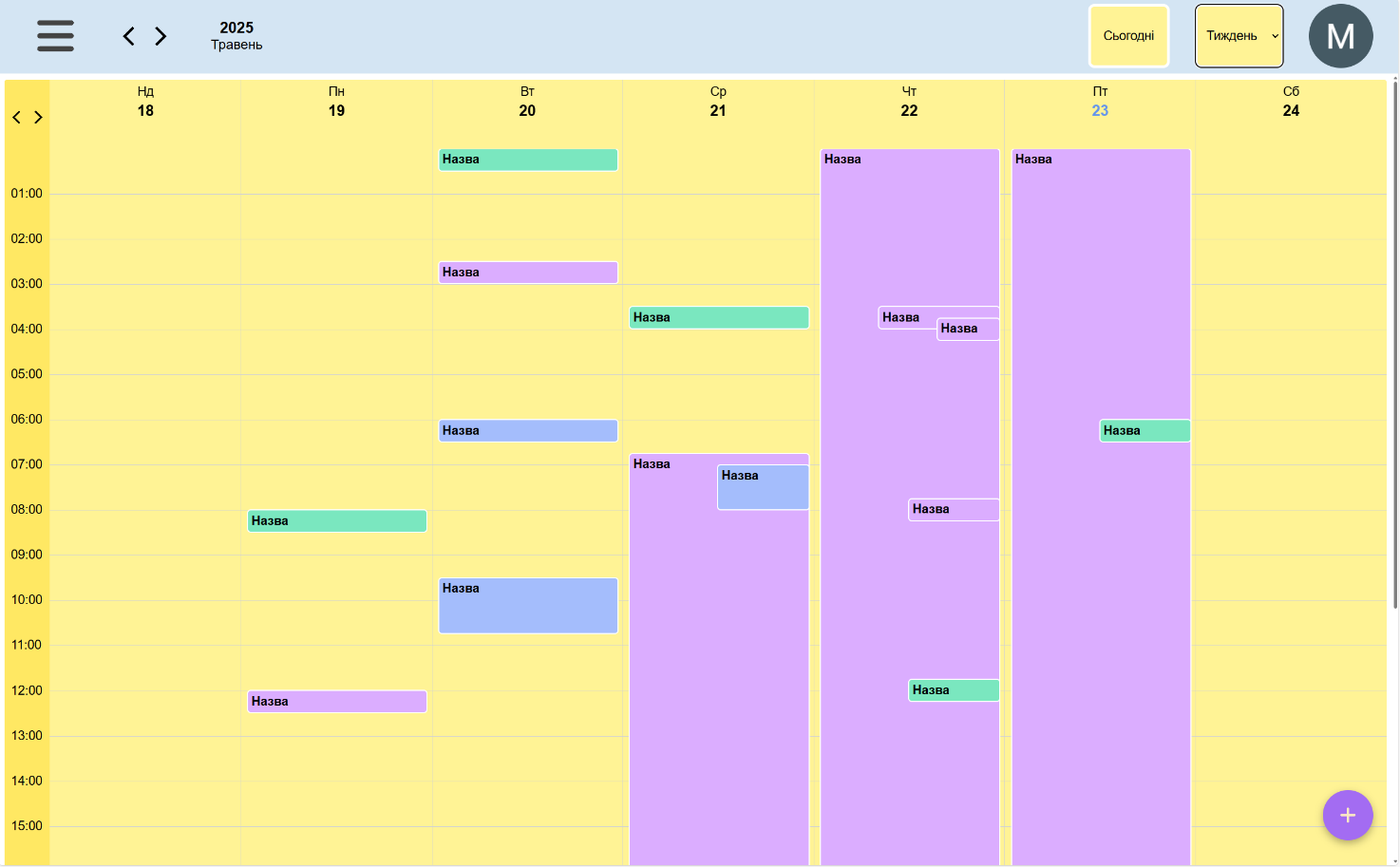


Рисунок 2.8 Додаток, відображення тиждня

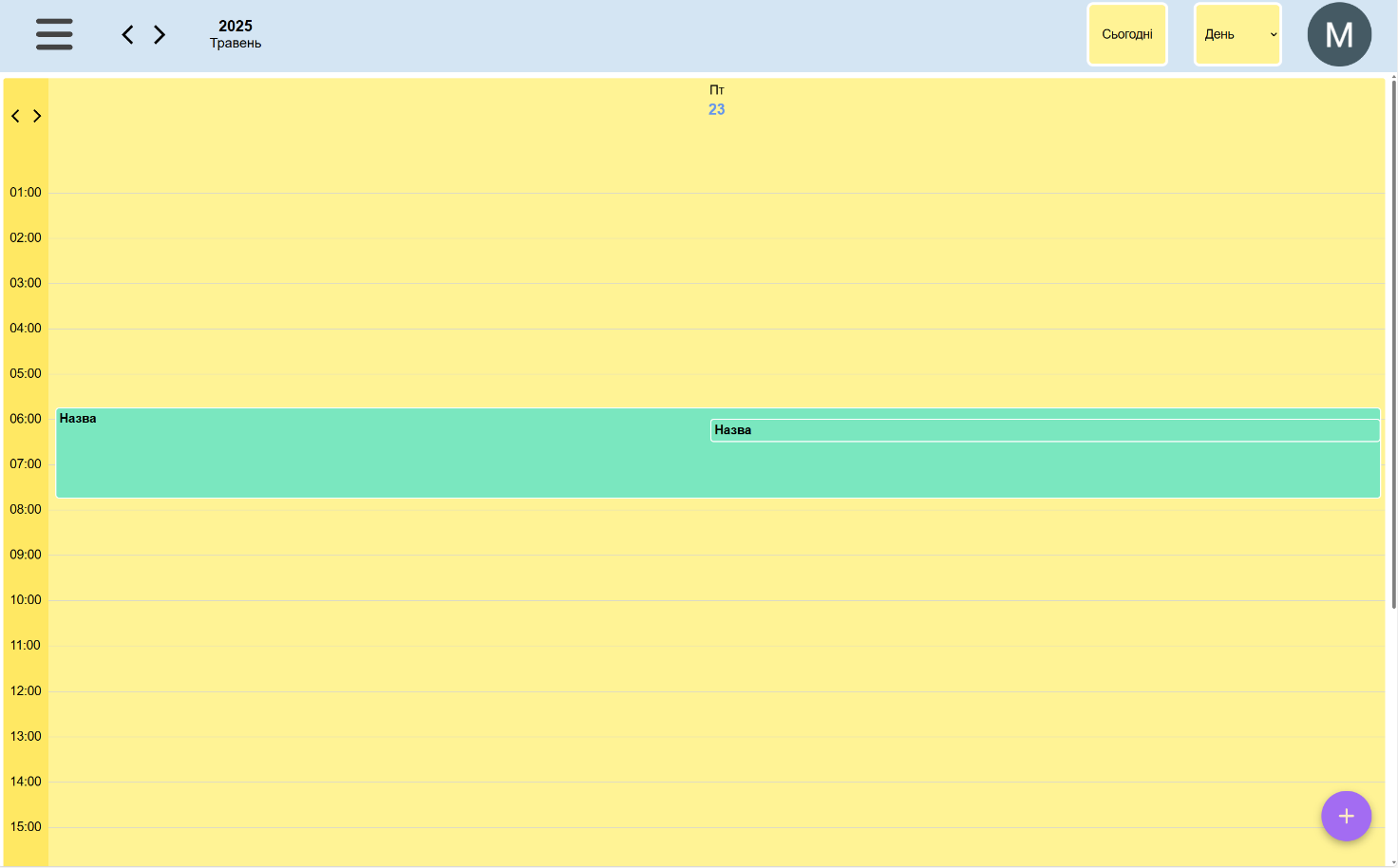


Рисунок 2.9 Додаток, відображення дня

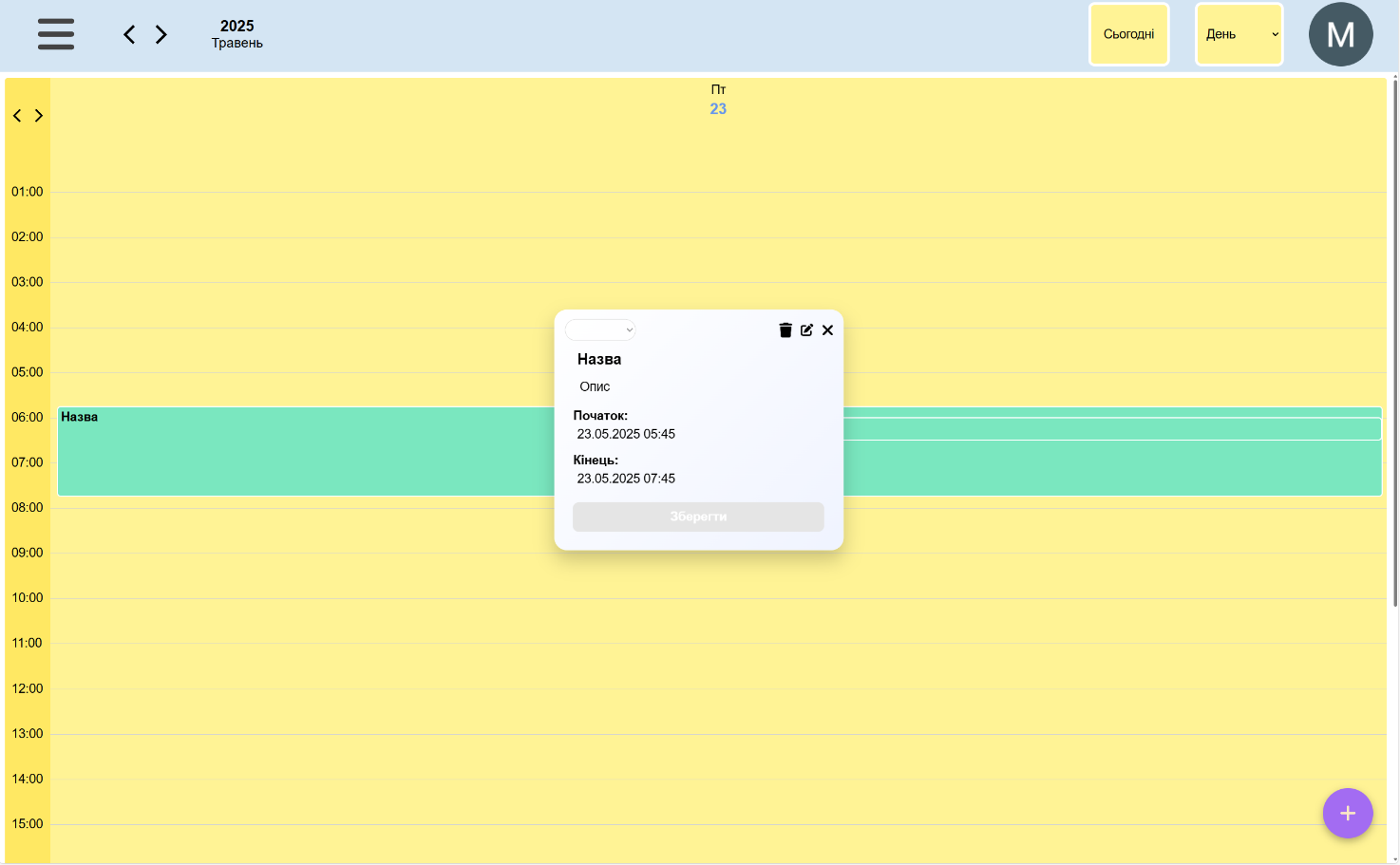


Рисунок 2.10 Видалення та оновлення подій (В залежності від обраної кнопки зверху)

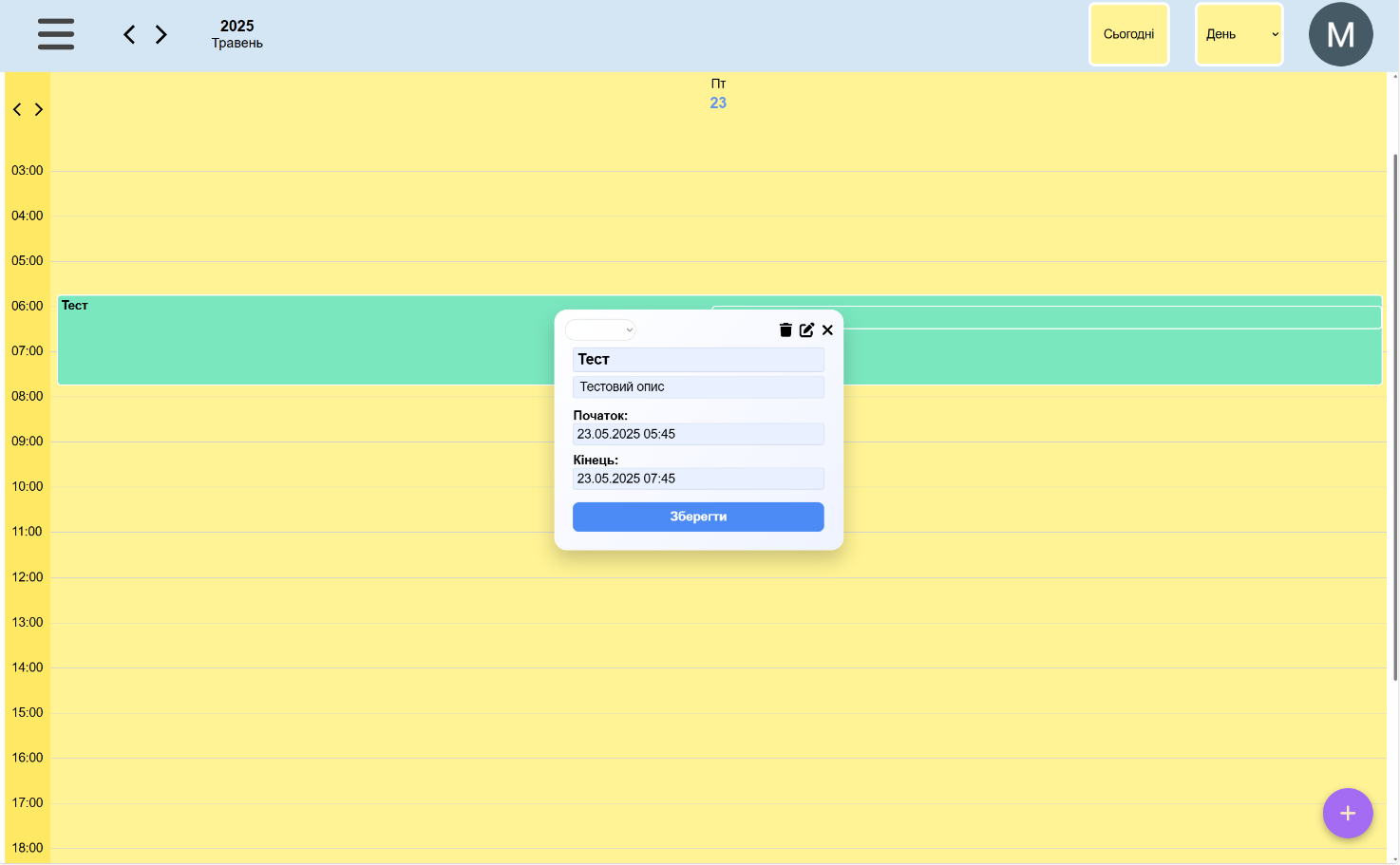


Рисунок 2.11 Оновлення подій **активна** (Текстові поля перетворюються на поля для вводу)

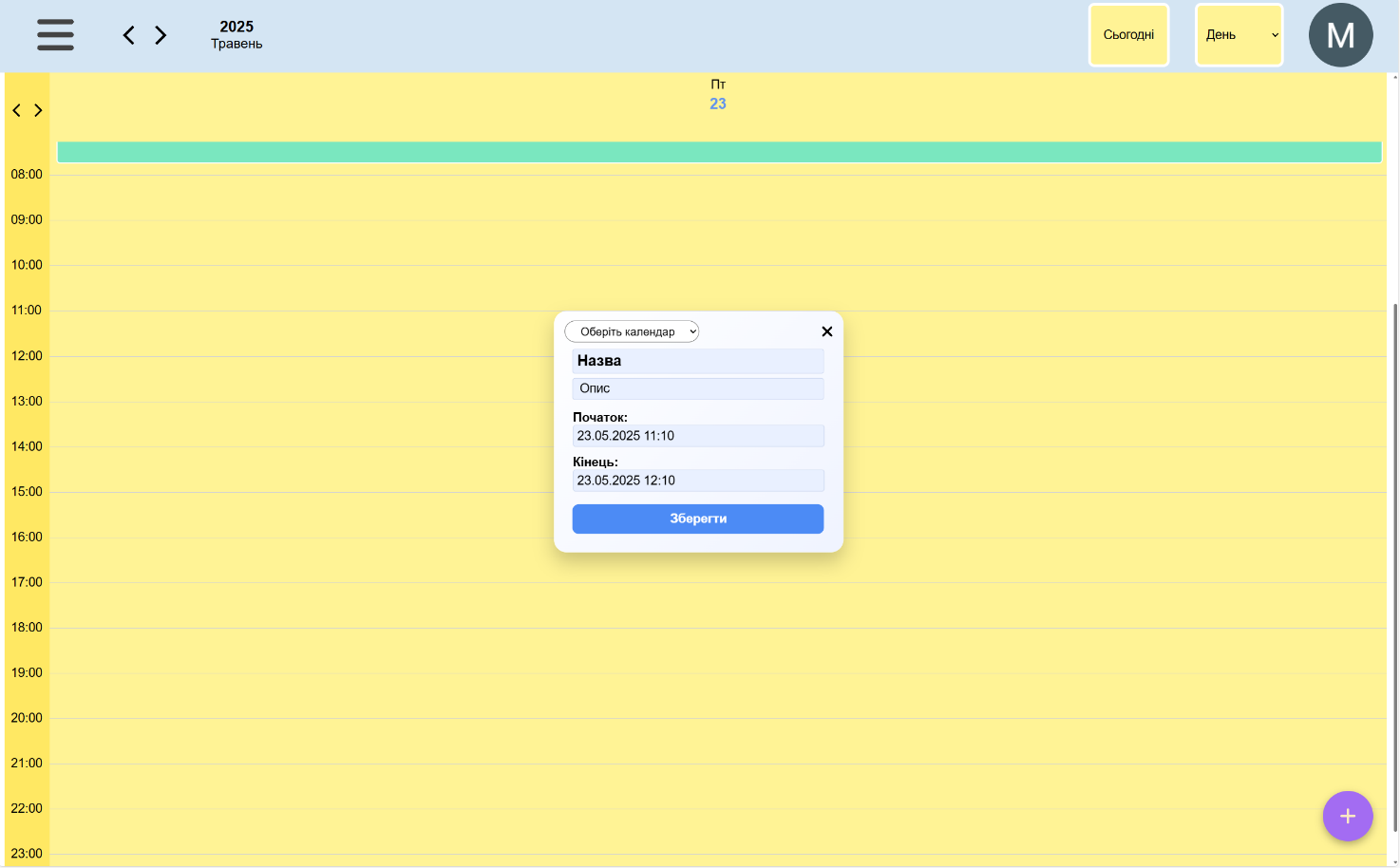


Рисунок 2.12 Створення подій (Вибір календаря для створення події)

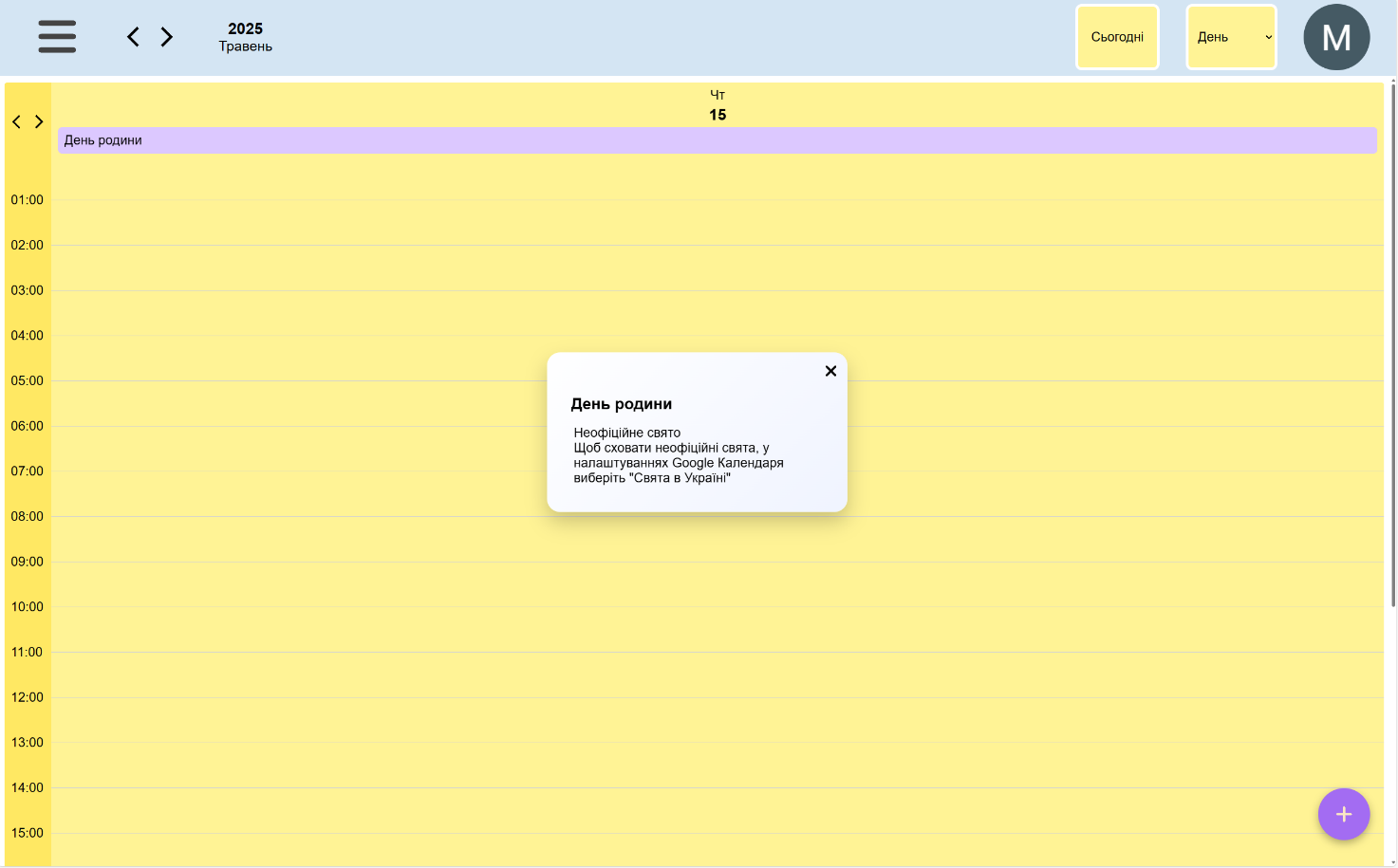


Рисунок 2.13 Перегляд українських свят

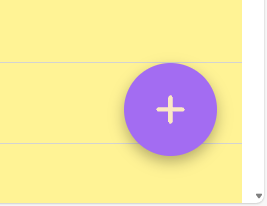


Рисунок 2.14 Кнопка створення подій (Правий нижній кут)

Повне відображення додатку з усіма відкритими вікнами **(Рис. 2.15 – 2.16)**

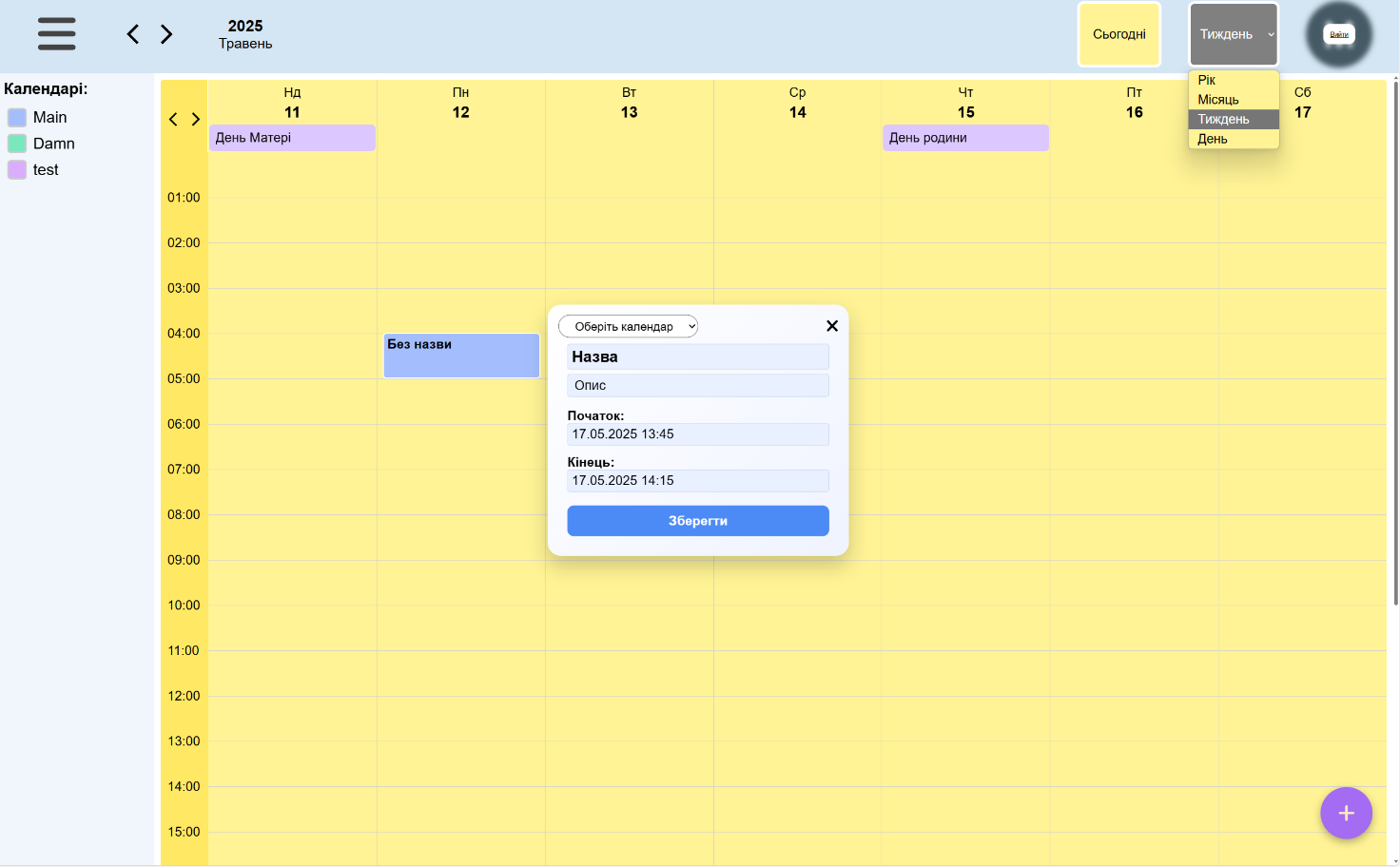


Рисунок 2.15 Додаток з усіма відкритими вікнами на комп’ютері

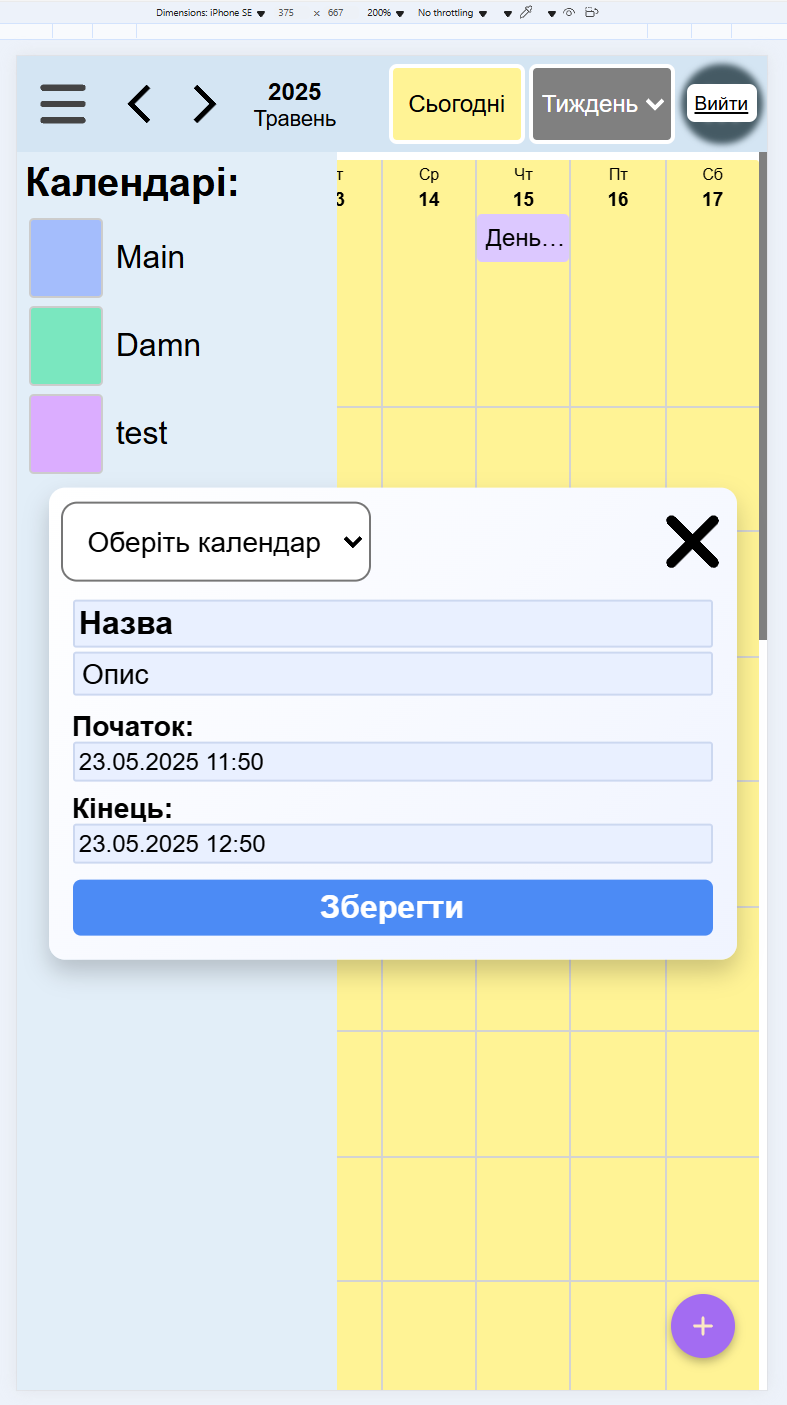


Рисунок 2.16 Додаток з усіма відкритими вікнами на Телефоні

Додаткові фото з відображенням додатку на телефоні (Рис. 2.17 – 2.18)

**Загалом інтерфейс користувача на телефоні відмінний від комп’ютера тільки розмірами елементів інтерфейсу.**

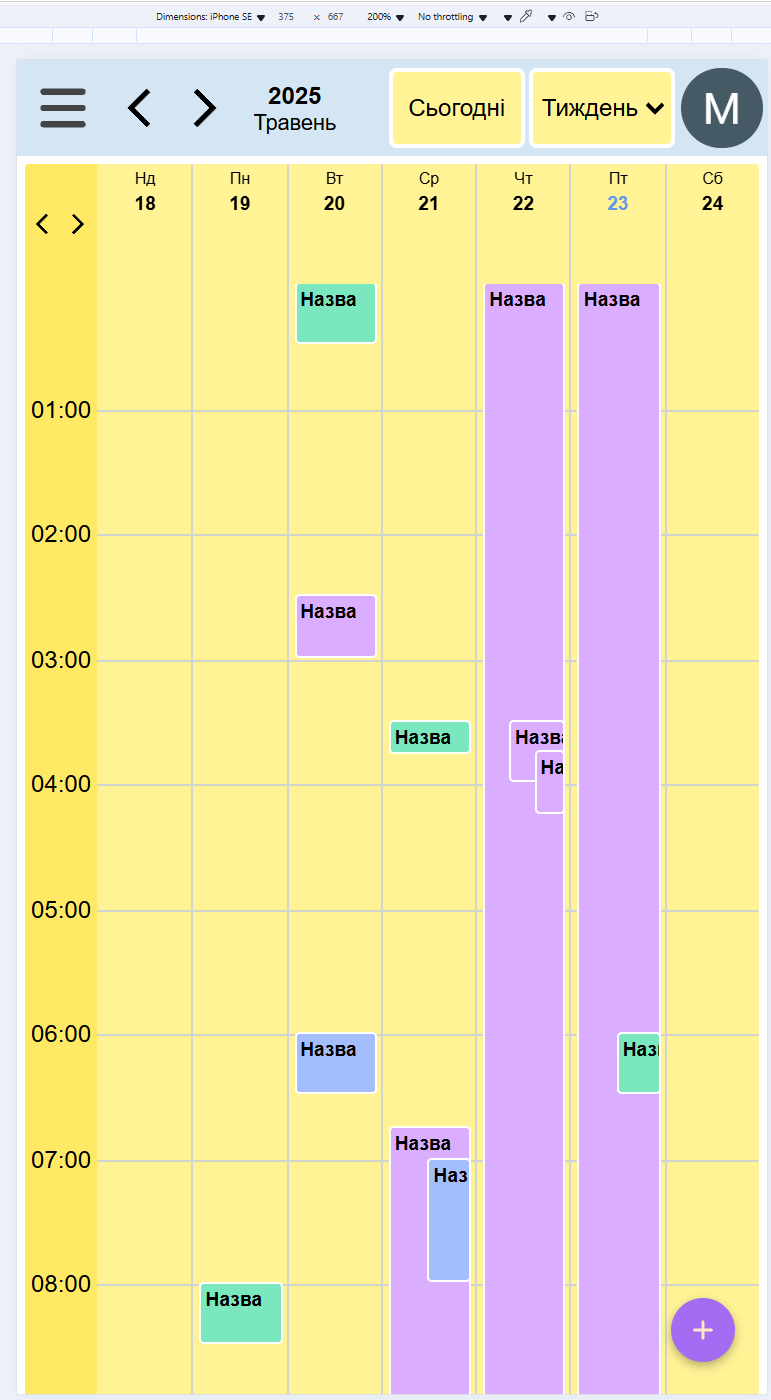


Рисунок 2.17 Відображення довгих подій на телефоні

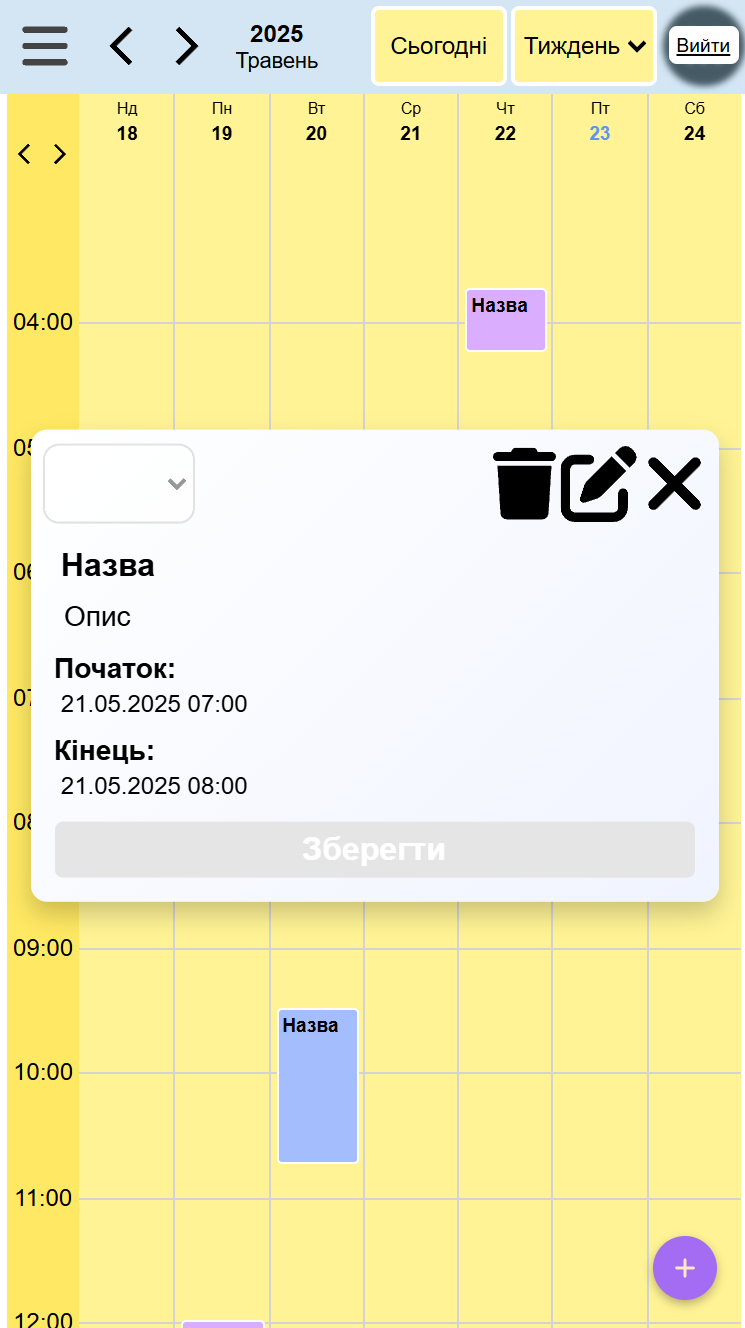


Рисунок 2.18 Перегляд подій на телефоні

# Розділ 3. Розміщення та стилізація елементів інтерфейсу

Загальний вигяд інтерфейсу обумовлений компактністю розмішення елементів інтерфейсу.

**На рис 3.1** можна побачити головні елементи інтерфейсу, перемикання дати, відображення та поточна дата.

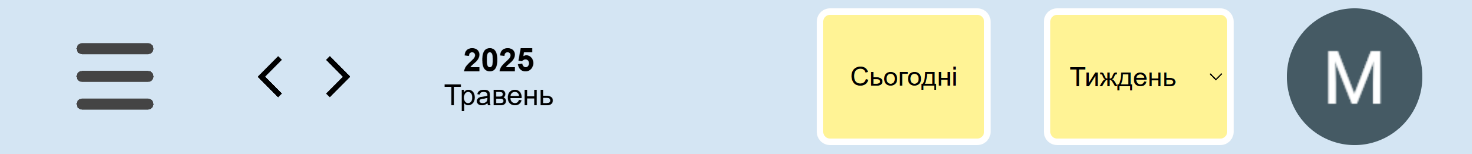


Рис 3.1 Верхня панель взаємодії

**На рис 3.2** можна побачити сам календар, зверїу відображається «липкий» елемент з датами та днем тиждня, а зліва час.

Також в календарі є полоси, що відображають розмежування дня та часу для зручності користування.

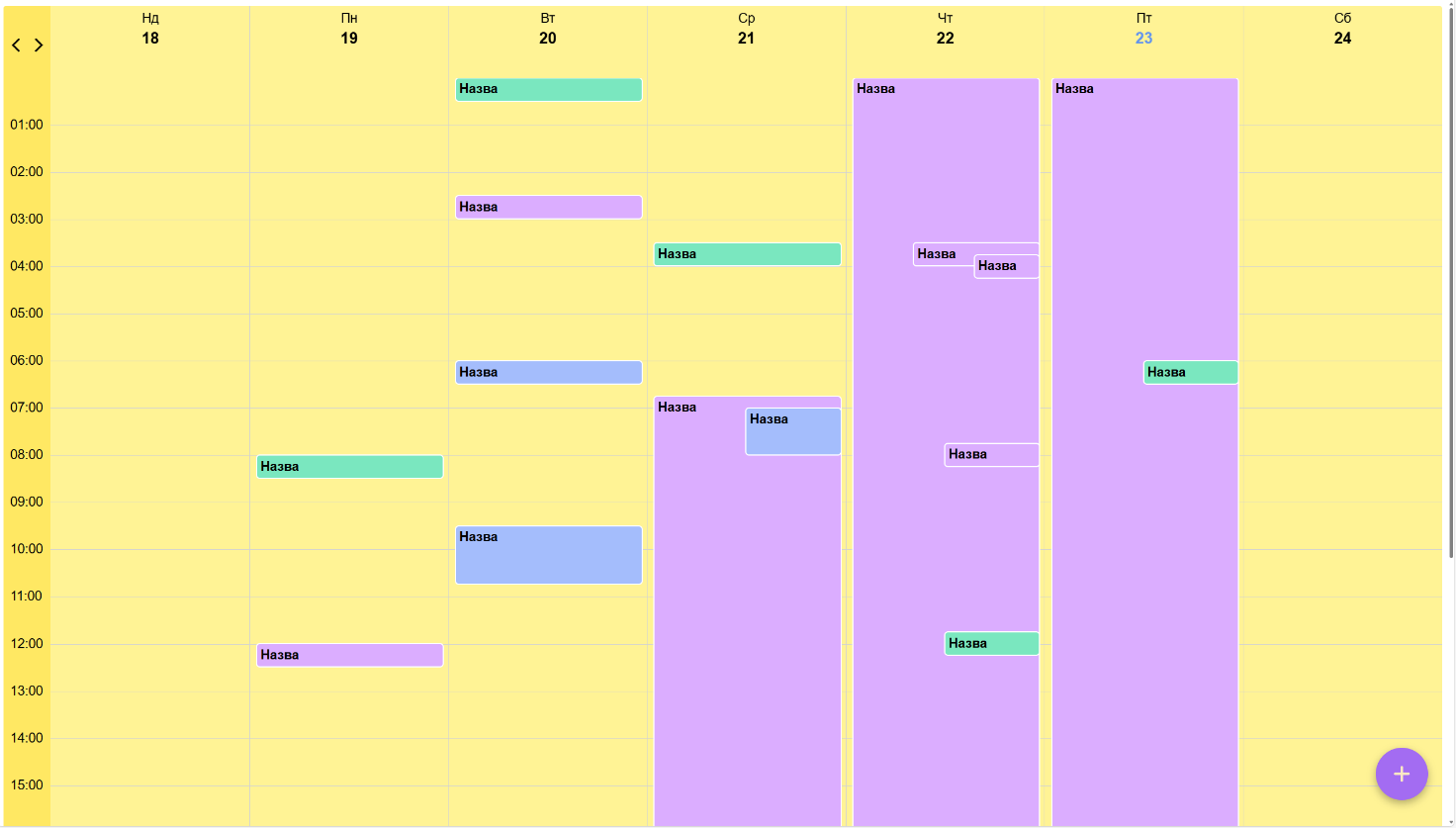


Рисунок 3.2 Календар

**На рис 3.3** можна побачити меню заповнення інформації про подію, це може бути, як створення так і оновлення події.

Тут є головні поля, що потрібні для користувача:

* Назва події, що буде відображатись на події
* Опис події
* Час початку та кінця події

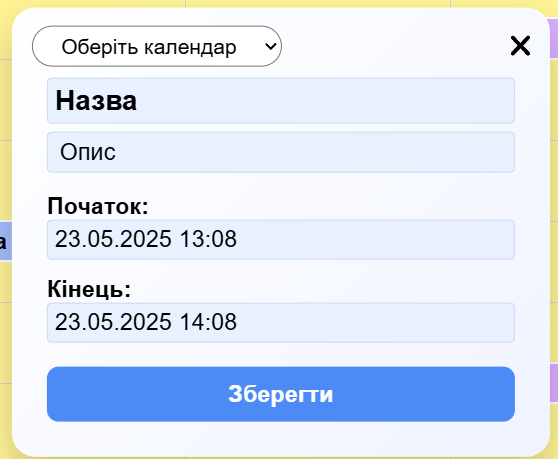


Рисунок 3.3 Календар

# Розділ 4. Проектування та реалізація моделей даних для збереження інформації.

В систему передбачено наступні сутності:

* Користувач
  + Ім’я
  + Пошта
  + Вміст календаря
  + Список календарів
* Календар
  + Назва
  + Опис
  + Колір
  + Власник
  + Вміст(події)
* Подія
  + Заголовок
  + Опис
  + Дата (початок, кінець)
  + Календар до якого належить
  + Місце початку\кінця відображення
  + Чи довга подія (більше 1 дня)
  + Статус події
* Учасник
  + Ім’я
  + Пошта

Модель даних «Користувач» зберігає в собі атрибути необхідні для створення, призначення подій у календарі. Користувач в «Календар» може створювати «Подія» в них. Користувач також може відправляти запрошення до свого Календарю або Події «Учаснику».

Залежності між моделями:

* Один користувач може створити багато календарів та подій
* Один користувач може призначити багато учасників до календаря та подій
* Один календар та подія можуть мати тільки одного власника
* Один календар та подія можуть мати багато учасників

# Розділ 5. Розробка механізмів отримання та оновлення даних.

1. Авторизація.

Після того, як користувач натискає кнопку «Іконка Гугл», коричтувача перенаправляє на сторінку авторизації «Гугл» за шляхом «/login/google», там він може створити аккаунт, або ж обрати зі збережених аккаунт.

Далі «Гугл» авторизація повертає результат у шляї «/login» про інформацію користувача. Бібіліотека «Flask Dance» для роботи з авторизацією обмінює ключ доступу авторизації «Гугл» на «Access token», в моєму випадку «Refresh token», бо я вказав це у відповідному полі доступу.

Й відповідно за цими даними я можу отримувати іншу інформацію про користувача, наприклад його фото профілю.

1. Отримання подій користувача.

При першому вході користувача в аккаунт, підвантажаться події поточного року користувача, для цього відбувається GET запит на сервер за шляхом /get\_event/<year>, де year це рік, що передає Фронт-енд.

У випадку, якщо користувач не авторизований в аккаунт сервер видасть помилку 401, та повернеться до користувача з відповідним повідомленням про це.

Але оскільки ця функція повинна виконуватись тільки, якщо користувач авторизований, то на Фронт-енді, вона просто не викличеться, якщо він не авторизований.

Першим кроком отримаємо календарі користувача, якщо все ок, далі сервер робить запит до «Гугл» з інформацією виду:

* 1. Дата початку отримання подій (year)
  2. Дата кінця отримання подій (year + 1)
  3. Показувати видалені події.

Якщо тут виникає помилка, то ми записуємо цю помилку в базу даних та повертаємо цю помилку користувачу, виводячи на екран.

Якщо все було успішно, то сервер отримує з відповіді:

* «syncToken», що дасть можливість при повторному запиті не отримувати всі події за рік, а тільки оновлення.
* Список подій
  + Цей список подій буде попередньо оброблено
    - Порахується розташування події на 24 часах, для коректного відображення.
    - Запишеться colorId, що відповідає календарям.
    - Запишеться calendarId, котрому відповідає конкретна подія.
    - Та призначиться статус події «confirmed», якщо ця подія існує та «canceled», якщо ця подія була видалена.

Після обробки подій всіх календарів, без помилок, ми повертаємо масив подій з масивом типу оновлення подій та код 200, «Full» або «Partial», якщо повне «Full» оновлення, то видаляємо всі події поточного року для конкретного календаря, та заповнюємо заново, якщо часткове «Partial», то змінюємо тільки ті події, що отримали з календаря.

Якщо список пустий, то повертаємо повідомлення з кодом 204 з пустим тілом та повідомляємо про це користувача.

Якщо сталась помилка з «syncToken» повертаємо код 410, записуємо інформацію про неї в базу даних, та просимо користувача перезайти в аккаунт.

Якщо сталась інша помилка, записуємо інформацію про неї в базу даних, та повертаємо помилку сервера 500 користувачу.

Протягом будь-якого з етапів у випадку виникнення помилки, її буде записано в базу даних а користувачу повідомлено про помилку.

1. Отримання свят.

Отримання свят також відбувається автоматично для кожного року, але не оновлюється й може бути виконано без авторизації користувача.

Для цього виконується запит GET за шляхом /get\_holidays/<year>, year рік, для якого будемо підвантажувати. Так само як і в отриманні подій, ми робимо запит до «Гугл», але тепер, замість інформації користувача, використовується створений мною Api ключ в Google Cloud Console. Далі ми робимо запит до українських свят, й обробляємо події так само, як і для подій.

Якщо в процесі цього не виникло помилок, то повертаємо список всіх подій. Якщо ж сталась помилка, то ми повертаємо код 500 з повідомленням про помилку на сервері.

1. Створення подій. Після того, як користувач зайде та заповнить меню створення подій на сайті, ця інформація передається на сервер за методом GET шляхом /create\_event/<calendar\_id>/<color\_id>. Тут ми отримуємо дані користувача з «Flask Dance» про користувача, та ставимо часовий формат України UTC+3 та робимо запит до «Гугл» про створення події.

Якщо успішно, то ми оброблюємо подію так само, як і при оттриманні подій, та повертаємо її на сайт для відображення користувачу, з кодом 201 та відповідним повідомленням, що подію було створено.

Якщо ж сталась помилка, нічого не змінюємо та повертаємо помилку 500 Помилка на сервері.

1. Оновлення подій.

Якщо користувач обрав подіб, перейшов в меню кореткні зміни, та відповідно підтведив їх, то робиться запит на сервер з методом PATCH для частвкового оновлення за шляхом /update\_event/<calendar\_id>/<color\_id>/<event\_id>, де calendar\_id це ідентифікатор календаря, до якого належить змінювана подія, color\_id це колір події відповідно до календаря, та event\_id відповідно самий ідентифікатор події.

На самому початку перевіряємо чи авториизований користувач, якщо так, то отримуємо іншу частину інформації від серверу про самі дані для оновлення події, щоб запобігти зайвих помилок протягом цього процесу, робиться запит до гугл календаря з переметрами calendar\_id та event\_id, щоб отримати всі заповнені поля події, далі, отримані зміни від Фронт-енду заміняються існуючим із запиту, та робиться новий запит для оновлення події, й відповідно попередня обробка події, та повернення її з кодом 200, якщо все успішно.

Якщо сталась помилка, то повертаємо користувачу помилку 500, помилка на сервері.

1. Видалення подій.

Якщо користувач в меню події обере іконку видалення, то виконається запит DELETE за шляхом /delete\_event/<calendar\_id>/<event\_id>, де calendar\_id це ідентифікатор календаря, до якого належить подія з ідентифікатором event\_id.

Далі робиться запит до «Гугл» про видалення цієї події, якщо успішно, то повертаємо статус 200 з повідмоленням про успішне видалення та видаляємо подію з календаря.

Якщо ж сталась помилка, то повертаємо повідомлення про помилку на сервері з кодом 500, та нічого не змінюємо.

1. Отримання календарів.

Виконується після авторизації користувача. Виконується запит GET за шляхом /list\_calendars.

Тут також перевіряється, чи авторизований користувач (401, або продовжуємо). Після перевірки, викликається функція для отримання календарів, це необхідно, бо ця функція також використовується і для отримання подій користувача.

Ця функція робить запит до «Гугл» про календарі користувача. Якщо все успішно, повертаємо календарі для відображення користувачу з кодом 200.

Якщо список календарів пустий, повертаємо статус 204 та відповідне повідомлення.

Якщо ж сталась помилка, то повертаємо код 500 помилка на сервері.

1. Вихід з аккаунту.

Якщо користувач натисне на свій профіль користувача, то з’явить поле, за надписом «Вийти», якщо користувач його натисне, буде виконано запит GET, за шляхом /logout, де спочатку відбувається повне видалення «syncToken» через запит про видалення до «Redis», якщо видалити не вийшло, логуємо це до БД, потім відповідно відбувається і очистка сессії користувача (вихід з аккаунту).

# Розділ 6. Розробка бази даних для збереження та отримання інформації.

## 6.1 PostgreSQL

1. Було обрано СКБД PostgreSQL, оскільки я вже був знайомим з MySQL, й PostgreSQL добре взаємодіє з фласк фреймворком за допомогою використанням бібліотеки SQLAlchemy. Крім того, сайт і база даних розміщуються на одному хостинговому сервісі «Render», тому PostgreSQL став ідеальним варіантом для мого проєкту.

Окрім цього, PostgreSQL має такі переваги:

* 1. Підтримує індексаціб пошуку, що оптимізує пошук по даних.
  2. Має високу надійність та захист даних, навіть після перезапуску серверу.
  3. Легко маштабується та дозволяє зручно зберігати та фільтрувати велику кількість логів.
  4. Підтримує заприти Агрегації, як JOIN та інші, що важливо для об’єднання таблиць за унікальними ключами.

1. Схема БД була згенерована автоматично в IDE і має вигляд **(рис 6.1):**

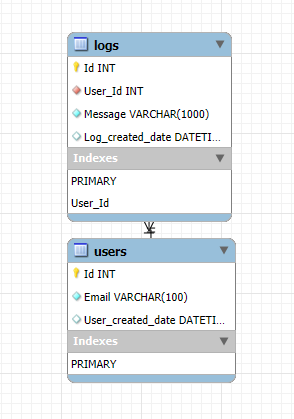


Рис. 6.1 Схема БД логів користувача

Та відповідно сторінка Логи користувачів за шляхом /admin та системи.

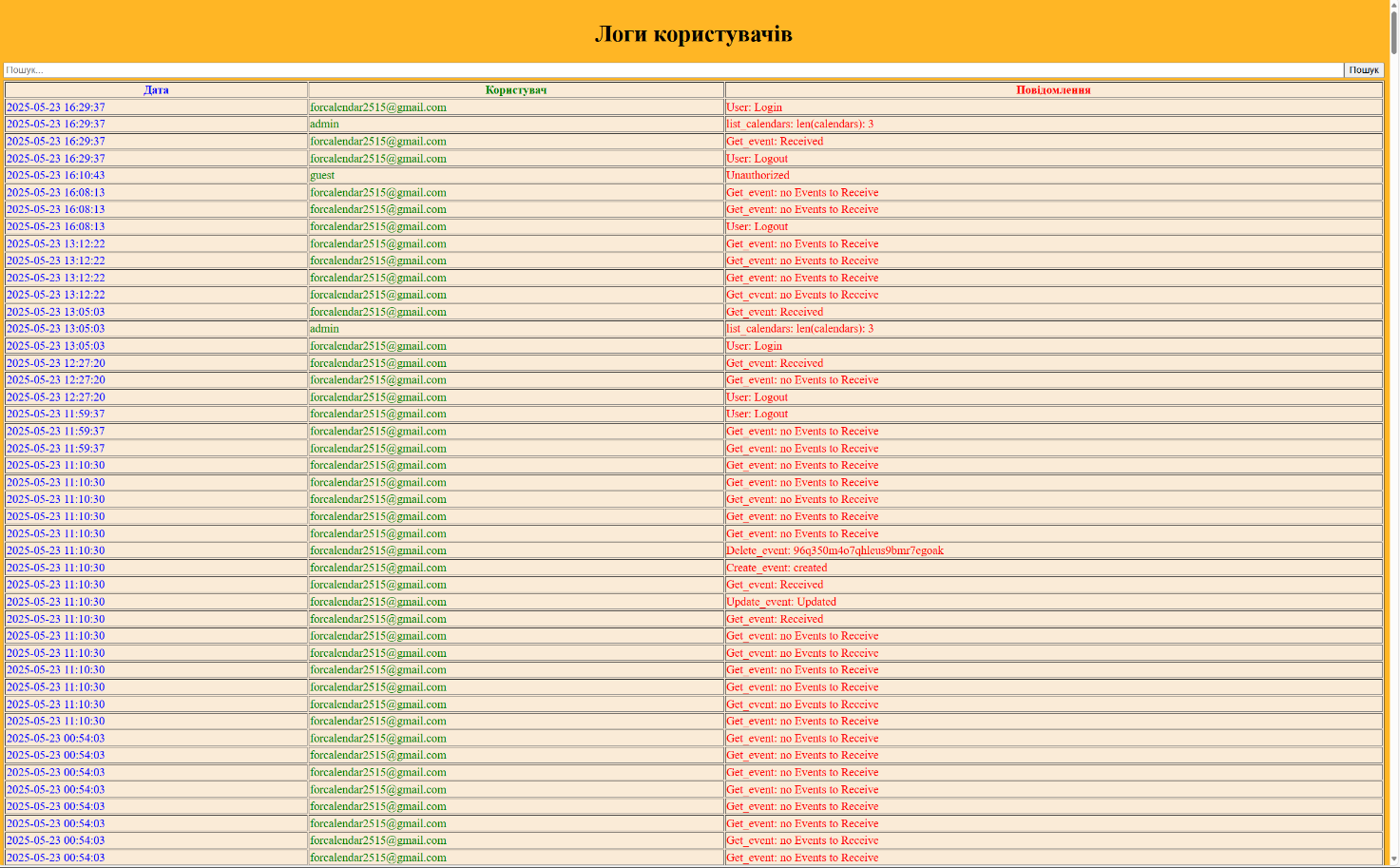


Рис. 6.2 Сторінка логів додатку

Ця схема БД була розроблена для логування помилок сервера та користувача, під час користування системою.

1. Було створено наступні індекси:
   1. Таблиця logs індекс з полем Id та зовнішній ключ User\_id, який використовується для зв’язування кожного запису з унікальним користувачем, що дозволяє зберігати довільну кількість логів для кожного користувача.
   2. Таблиця users та індекс з полем Id, як первинний ключ. Воно використовується для ефективного зв’язку з таблицею логів, що забезпечує швидкий доступ до логів, пов’язаних із конкретним користувачем.

## 6.2 Redis

Також було використано додаткову базу даних, для збереження тимчасових «syncToken» від «Гугл», для отримання змін.

1. Для цього було використано Redis, оскільки він швидко працює, та майже без затримок може зберігати та повертати дані.

Вона також має дуже прості налаштування типу ключ: значення, то ж не треба створювати таблиці, як в інших базаї даних. До переваг відносяться:

* 1. Працює в оперативній пам’яті, тому дуже швидкий.
  2. Підтримує встановлення часу життя, що ідеально підходить для тимчасових «syncToken».
  3. Має простий синтаксис, та високу продуктивність.

1. Redis використовувався для зберігання тимчасових «syncToken», отриманих від «Гугл» календаря, що дозволяє при наступному запиті отримувати тільки оновлені або нові події, без повторного сканування всього календаря.
2. Оскільки Redis це нереляційна БД, тож вона не використовує індекси, замість цього використовуєхться доступ у форматі ключ: значення, що значно спрощує доступ до інформації.

# Розділ 7. Інтеграція сторонніх сервісів.

У цьому проекті було реалізовано взаємодію зі сторонніми сервісами, що значно спростило розгортання вебзастосунку та його функціонал. Основними з них були Google Calendar API та хостинг Render.

## 7.1 Google Calendar API

Google Calendar API було використано у додатку для збереження та синхронізації подій користувача.

Авторизація користувача відбувається за допомогою Client ID та API Key, що можна згенерувати в Google Cloud Console.

Комунікація з сервісом відбувається за допомогою бібліотеки Flask-Dance та Google API Python Client.

Flask-Dance - використовувався для авторизації через OAuth, підтримки з’єднання та отримання даних про користувача.

Google API Python Client для виконання запитів до Google Calendar.

Для налаштування доступу до подій, необхідно налаштувати blueprint від Google, для авторизації додатка та його прав доступу. Для цього в полях **scope** було записано такі елементи прав доступу:

1. https://www.googleapis.com/auth/userinfo.email - для доступу до пошти користувача.
2. https://www.googleapis.com/auth/userinfo.profile - для отримання картинки профілю користувача.
3. <https://www.googleapis.com/auth/calendar> - для отримання повного доступу до створення/видалення подій та доступу до календарів користувача.

Далі для отримання подій користувача, необхідно спочатку зібрати необхідну інформацію Credentials про нього з session Flask-dance, де зберігаються його токени авторизації, та Client Id й Client Secret мого додатка, що я отримав в Google Cloud Console.

Після цього треба «збудувати сервіс» (build service), для доступу до Google API. Після чого можа робити запит до гугл з вимогами по подіям, календарям.

## 7.2 Хостинг і база даних на Render

Хостинг Render було обрано як хостинг-платформу для розгортання мого додатку Календаря та хостингу реляційної бази даних PostgreSql.

Сервіс Render також забеспечує автоматичне розгортування після кожного оновлення Git репозиторію додатка, що стало приємним бонусом.

Render надає простий інтерфейс для створення Веб на різних мовах програмування, а також безкоштовний хостинг БД PostgreSQL, що дозволило легко підключити базу даних до застосунку через SQLAlchemy.

Для розгортання мого додатку, я створив аккаунт та заповнив поля для створення нового Веб сервісу, після чого просто прикріпив свій Git репозиторій.

Для створення та підключення БД до мого додатку, я також заповнив поля з назвою та описом й потім отримав посилання для зовнішнього підключення, після чого за допомогою SQLAlchemy підключився до нього.

Render є зручним сервісом для невеликого або тестового розгортування, бо пропонує безкогтовний план з єдиним обмеженням, що через 15 хвилин неактивності, сервер переходить в «сплячий» режим.

# Розділ 8. Створення серверної архітектури додатку.

1. Для розробки серверної частини було використано монолітну архітектуру, оскільки вона ідеально підходить для невеликих та навіть середніх проектів.

Ця архітектура дозволяє об’єднати весь функціонал в одному додатку. Оскільки в мене відносно невеликий проект, то головними перевагами монолітною архітектури стали:

* 1. Швидке та просте розгортання – весь додаток запускається, як єдине ціле і не треба налаштовувати взаємодію між окремими сервісами.
  2. Простота розробки – вся логіка програми, обробка запитів, налаштування бази даних та інше, розташовано в одному кодовому середовищі.
  3. Менше фінансових витрат – хоч я і не витратив жодних коштів для розгортання свого додатку, підтримка монолітної архітектури загалом вимагає менше ресурсів і коштів, що особливо важливо для невеликих проектів з малим капіталом.

1. Опис головних модулів системи та їх відповідальність:
   1. App.py

Це головний файл проекту, що відповідає за старт і конфігурацію Flask додатку.

* + 1. Ініціалізує екземпляр додатку app.
    2. Підключає базу даних, логування, налаштовує Redis.
    3. Реєструє основні Blueprint проекту, модулі, що будуть працювати по маршрутам.
    4. Відповідає за підключення ресурсів (CSS, JS) через бібліотеку Flask-Assets для оптимізації завантаження фронтенду з декількох файлів.
  1. Events\_control.py

Це Blueprint, що відповідає за роботу з подіями Google Calendar.

* + 1. Реалізує маршрути для отримання, створення, оновлення та видалення подій.
    2. Використовує Redis для кешування «syncToken» ключа синхронізації, що дозволяє оптимізувати оновлення подій календаря.
    3. Логування помилок, та повернення результатів виконання запитів користувачу.
  1. Google\_auth.py

Blueprint, що реалізує авторизацію користувача через Google OAuth.

* + 1. Реалізує маршрути авторизації **login** та виходу з аккаунта **logout** користувача.
    2. Використовує Flask-Dance для зручної реалізації авторизації Google OAuth.
    3. При виході з аккаунту очищує кеш Redis, щоб видалити збережені токени синхронізації «syncToken» та сесії.
    4. Логування часу авторизації та виходу з аккаунту.
  1. Utils.py

Допоміжний модуль з функціями, для попередньої обробки данних подій.

* + 1. Функції для підготовки подій у формат, зручний для фронтенду.
    2. Функції отримання Credentials для побудови сервіса для запитів до Google Calendar API.
    3. Функція дял форматування дат і часу з перевіркою на помилки.
    4. Функція для отримання календарів користувача, необхідна як для запитів із фронт-енду так і для отримання подій користувач

# Розділ 9. Реалізація обробки запитів та взаємодії з базою даних на сервері.

1. GET /get\_events/<year>
   1. Даний запит відповідає за отримання данних подій календарів користувача за певний рік <year>, що передає клієнтська частина та данними аккаунту, що зберігаються на сервері через Flask-Dance. Запит захищеним авторизацією.
   2. Для отримання календарів користувача, було використано функцію get\_calendar\_list, що отримує дані користувача з Flask-Dance та будує сервіс, після чого виконується запит на цей сервіс Google для отримання списку календарів користувача. Сервіс має такий вигляд service = build("calendar", "v3", credentials=creds), де creds це дані отримані з Flask-Dance про користувача, а "calendar" та "v3" це звідки ми будемо брати ці дані (з календаря користувача). Й відповідно виконується запит service.calendarList().list().execute(), що повертає календарі користувача «сервісу».
   3. Після отримання календарів, ми робимо повторно запити для кожного календаря окремо, щоб отримати їх події виду service.events().list(calendarId=calendar\_id, \*\*params).execute(), де calendar\_id це ідентифікатор календаря, що ми отримали зі спику календарів, а \*\*params, це параметри запиту, що мають вид:
      1. params = { "timeMin": time\_min, "timeMax": time\_max, "singleEvents": True, "showDeleted": True, }, де
         1. timeMin це час початку, звідки починаємо отримувати події
         2. timeMax це час кінця
         3. singleEvents це параметр для отримання повторюваних подій, щоб отримати їх як окремі, не повязані.
         4. showDeleted це параметр для відображення видалених подій, щоб можна було коректно працювати із синхронізацією.
2. GET /get\_holidays/<year>
   1. Цей маршрут відповідає за отримання державних свят на переданий рік <year>. Запит не потребує авторизації користувача, оскільки використовує мій API-ключ доступу.
   2. Для доступу до публічного календаря свят використовується HOLIDAY\_CALENDAR, ідетифікатор якого зберігаєтсья у змінній середовища, а також API\_KEY, який був отриманий з Google Cloud Console для доступу до Google Calendar API без автентифікації користувача. Потім виконується URL запит на адресу виду:  
      url = (f"https://www.googleapis.com/calendar/v3/calendars/{calendar\_id}/events"  
      f"?key={api\_key}&timeMin={time\_min}&timeMax={time\_max}&singleEvents=true")  
      Де параметри:
      1. calendar\_id - ідентифікатор календаря свят.
      2. timeMin та timeMax – поочатку та кінця року (в моєму випадку)
      3. singleEvents=true - дозволяє розгорнути повторювані події в окремі події.
   3. Після відправки запиту за допомогою за url отримуємо відповідь у форматі json від Google Calendar, після чого отруємо вже самі події data.get("items", []), кожна подія попередньо оброблюється для клієнтською частини функцією event\_prepare\_data(calendar\_id, 99, event, HOURS), де:
      1. 99 – це умовний індекс кольору
      2. Event – це сама подія з API
      3. HOURS – це константа зі значення 24, як годин в добі
   4. Якщо все проходить успішно, повертається список оброблених подій (holidays) у форматі JSON з кодом 200. Якщо виникає помилка, він логуються через log\_action() в БД, а користувачу повертається повідомлення про помилку з кодом 500 Помилка на сервері.
3. POST /create\_event/<calendar\_id>/<color\_id>
   1. Цей маршрут відповідає за створення нової події у календарі користувача. Запит надсилається з клієнтської частини методом POST й перевіряє авторизацію користувача через Flask Dance. Змінні, що отримані через шлях являють собою calendar\_id – ідетифікатор календаря, до якого створюємо подію та color\_id – колір календаря до якого він належить.
   2. Авторизація користувача перевіряється через функцію check\_auth(). Якщо користувач не авторизований, повертається 401 та повідомлення, по це. В іншому випадку відбувається побудова сервірсу: service = build("calendar", "v3", credentials=creds) для роботи з Google Calendar.
   3. Далі дані нової події надходять у форматі:
      1. summary - назва події (або "Без назви", якщо не задано)
      2. description - опис події (або порожній рядок)
      3. start та end - це дата й час початку та завершення події у форматі ISO з часовою зоною "Europe/Kyiv"
   4. Потім подія створюється через виклик: service.events().insert(calendarId=calendar\_id, body=event\_body).execute() що додає подію до вказаного календаря користувача подію.
   5. Після створення подія обробляється функцією event\_prepare\_data (calendar\_id, color\_id, response, HOURS), де вона приймає бажану формат для клієнтської частини. color\_id передається як параметр, що використовується для відображення події певним кольором.
   6. Якщо все добре, повертається код 201 з повідомленням "Event created" і самою подією у форматі списку.
   7. У разі помилки, вона логується, а відповідь повертається з повідомленням "Create\_event; Server Error" і статусом 500 Помилка на сервері
4. PATCH /update\_event/<calendar\_id>/<color\_id>/<event\_id>
   1. Цей маршрут оновлює вже існуючу подію в календарі користувача. Потрібна авторизація користувача. Запит надсилається методом PATCH. Для оновлення вказується:
      1. calendar\_id - ідентифікатор календаря
      2. event\_id - ідентифікатор самої події
      3. color\_id - колір, який використовується на клієнті для візуалізації події.
   2. Перевірка авторизації виконується через check\_auth(). Якщо вона не пройдена - повертається відповідне повідомлення. Якщо все гаразд, облікові дані отримуються через get\_credentials(), і створюється сервіс service = build("calendar", "v3", credentials=creds).
   3. Із запиту зчитується JSON-обʼєкт data, який може містити деякі або всі з наступних полів:і
      1. summary - нова назва події
      2. description - новий опис події
      3. start\end - новий час початку\кінця
   4. Спочатку отримується повна версія існуючої події з Google Calendar через: service.events().get(calendarId=calendar\_id, eventId=event\_id).execute().   
      Це потрібно для запобігання передачі пустих полі з фронтенду.
   5. Оновлена подія надсилається до Google Calendar через: service.events().update(calendarId=calendar\_id, eventId=event\_id, body=event).execute()
   6. Відповідь updated\_event передається функції event\_prepare\_data(calendar\_id, color\_id, updated\_event, HOURS), щоб привести подію до зручного формату. Після чого логуються дії користувача та повертається відповідь.
   7. Якщо все добре, повертається відповідь 200 із повідомленням "Event updated" і оновленою подією.
   8. Якщо сталась помилка, вона логуються і повертається відповідь з повідомленням "Update\_event; Server Error" і статусом 500.
5. DELETE /delete\_event/<calendar\_id>/<event\_id>
   1. Цей маршрут призначений для видалення події з календаря. Він використовує метод DELETE, а на вхід приймає такі значення:
      1. calendar\_id - ідентифікатор календаря
      2. event\_id - ідентифікатор події, яку необхідно видалити
   2. Перевірка авторизації виконується через функцію check\_auth(). Якщо користувач не авторизований - повертається відповідне повідомлення. Якщо авторизація пройдена, облікові дані отримуються функцією get\_credentials().
   3. Далі створюється сервіс Google Calendar API: service = build("calendar", "v3", credentials=creds)
   4. Видалення події виконується за допомогою виклику: service.events().delete(calendarId=calendar\_id, eventId=event\_id).execute()
   5. Після успішного видалення події, дії користувача логуються через: log\_action(session["user\_info"]["user\_email"], f"Delete\_event: {event\_id}")
   6. Повертається відповідь із повідомленням "Event deleted" та event\_id й кодом 200.
   7. У разі помилки вона логується, і повертається відповідь з повідомленням "Delete\_event; Server Error" і статусом 500.

# Розділ 10. Тестування та налаштування веб-додатку на сервері.

## Бек-енд

1. Основний серверний код було покрито unit тестами на 100 %. Unit тести перевіряють, чи коректно працюють функції, чи повертають дані у необхідному форматі, та перевіряються всі варіації їх використання.

Utils.py це головний файл з усіма допоміжними функціями, й він був повністю покритий тестами. **(Рис. 10.1)**

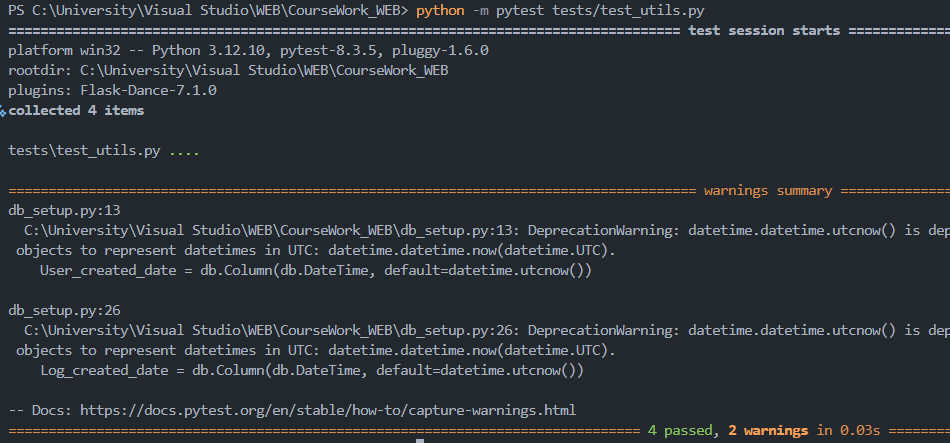


Рис. 10.1 Покриття unit тестами файлу Utils.py

1. Основний серверний код було покрито інтеграційними тестами на 100 % **(Рис.10.3).** Ці тести потрібні, щоб перевірити, що додаток працює коректно на кожному процессу розробки, та взаємодіють один між одним коректно. Зазвичай для цього використовують так звані «заглушки» «patch» сервіси, яким «примусово» призначають певні відповіді для імітації їх роботи. Також використовують «Mock», об’єкти, що імітують роботу справжнії сервісів.

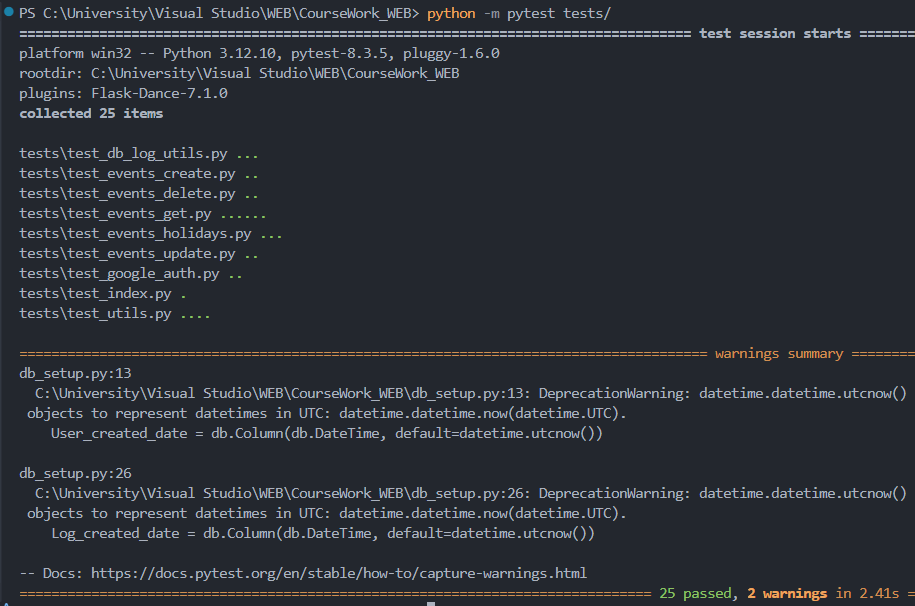


Рис. 10.3 Результат всіх інтеграційних тестів.

* 1. Перший файл test\_db\_log\_utils.py – логування, було покрито повністю, а саме такі сценарії:
     + 1. На початку було ініціалізовано тестову БД від Mysql Lite, для локального тестування.
     1. Успішне логування, повертаємо код 200. Для цього було зроблено тестову сессію користувача, та передано певні дані до БД для логування пост запитом.
     2. Хибне логування, код 400, коли надійшло недостатньо даних для логування. Просто логуємо, без повідомлення, як результат, недостатньо даних.
     3. Успішне логування користувача та перевірка, чи є такий лог на сторінці відображення логів /admin.
  2. Далі файл test\_events\_create – створення подій, повністю покрит тестами.
     1. Повідомлення 201, про успішне створення події. Для цього було зроблено мокування «Mock» необхідних сервісів, й потім передано у функцію, після чого логуємо повідомлення про це й перевіряємо, чи логування було успішне.
     2. Помилка сервера, код 500. Так само було зроблено «Mock» необхідних сервісів та викликаємо помилку у відповіді response. Й перевіряємо логування.
  3. Файл test\_events\_delete.py – видалення подій. Покрито на 100%.
     1. Успішне видалення код 200. Для цього зроблено «Mock» сервісів, й далі передаємо інформацію про подію до функції через запит resp = client.delete("/delete\_event/cal1/event123").
     2. Помилка 500. Якщо в процессі видалення сталась помилка, то має відбутись ця помилка. Для цього у функції було викликано помилку та перевірено, чи справді вона викликалась 500, та чи було залоговано про це повідомлення.
  4. Файл test\_events\_get.py – отримання подій користувача. 100%
     1. Успішне отримання подій без SyncToken 200. Для цього ми створюємо Mock відповідь Google API з nextSyncToken, що викликає збереження токена та повернення Full синхронізації. Після цього перевіряємо чи викликався Redis для збереження цього токена синхронізації.
     2. Успішне отримання подій з SyncToken 200. Для цього так само ми робимо запит, але цього разу ми робимо це з SyncToken, тому ми перевіряємо, чи змінився в нас тип підвантаження syncType на Partial для часткового оновлення кешу клієнта, й перевіряємо чи вірна подія передалась.
     3. Помилка в токені синхронізації SyncToken 410. Означає, шо час існування токену сплив, то це могло б призвести до помилки серверу, щоб цього запобігти, в додатку є перевірка помилок від Google Api, що і робить обробку цієї помилки. Для тестування цього, було використано токен синхронізації, й під час виклику оновлень, повертаємо статус 410, щоб викликати обробник і перевірити правильність обробки цієї помилки.
     4. Успішне виконання, але без отримання подій 204. Неважливо чи маємо ми токен синхронізації, якщо ми отримуємо успішне повідомлення від Google Api, і прицьому подій немає, то повертаємо статус 204. Для перевірки, під час запиту до Google Api, просто повертаємо пустий масив.
     5. Помилка серверу 500. Для перевірки цього викликаємо помилку під час запиту.
  5. Файл test\_events\_holidays.py – отримання вихідних подій. 100%
     1. Успішне отримання подій 200. Створюємо тестову відповідь Mock. Та викликаємо функцію, перевіряємо, чи отримали ми її.
     2. Успішне отримання без авторизації 200. Спеціально імітуємо, що користувач не авторизований, False викликом функції перевірки авторизації. Якщо статус 200, то ми можемо отримати події і без авторизації.
     3. Статус 500, помилка серверу. Під час запиту Mock викликаємо помилку, що призведе до логування. Перевіряємо, чи залогувалась ця помилка.
  6. Файл test\_events\_update.py – оновлення існуючих подій. 100%
     1. Успішне оновлення 200. Створюємо тестову подію, що оновимо, та подію, що її замінить. Далі створюємо Mock сервісів та виконуємо patch запит на оновлення. Після запиту імітуємо повідомлення від Google Api оновленною подією, та перевіряємо її поля.
     2. Помилка 500. Під час оновлення наш Mock сервір повертає помилку, що відповідно і зупиняє програму, перевіряємо це. В кінці логуємо та перевіряємо, чи справді помилка записана до БД.
  7. Файл test\_google\_auth.py – Вихід користувача з аккаунту.
     1. Успішний вихід користувача і SyncToken видалився 302. Код 302 повідомляє, про успішний вихід з аккаунту. Для перевірки записуємо користувача в сессію та створюємо тестові ключі зберігання SyncToken в Redis. Далі під час видалення замінюємо значення на успіх Mock та перевіряємо, чи вийшов користувач з аккаунту та чи викликалась функція видаленн ключів для збережених SyncToken/
     2. Успішний вихід користувача і SyncToken НЕ видалився 302. Для перевірки виходимо з тестового аккаунту та примусово призначаємо Mock значення, що ключі не видалились, й перевіряємо. Чи є відповідний запис про це в БД.

## Фронт-енд

Для тестування системи було використано наступні сценарії:

**Назва групи**: Перегляд Свят.

**Назва Сценарію**: Користувач не авторизований.

**Необхідні дії**: Користувач не увійшовши в аккаунт натискає на подію\свято в календарі.

**Очікуваний результат**: Користувачу відображається повідомлення про подію.

**Назва групи**: Перегляд Свят.

**Назва Сценарію**: Користувач авторизований, й не має права на зміну подій свят.

**Необхідні дії**: Користувач увійшовши в аккаунт натискає на подію\свято в календарі, й намагається її змінити.

**Очікуваний результат**: Користувачу відкриває подію й бачить інформацію про неї, але кнопки для редагування події \ свята немає.

**Назва групи**: Перегляд Подій.

**Назва Сценарію**: Користувач не авторизований намагається переглянути свої події календаря.

**Необхідні дії**: Користувач не увійшовши в аккаунт шукає подію створено ним в календарі Google, або коли він був авторизований.

**Очікуваний результат**: Користувач не знаходить подій, бо вони зникли після його виходу з аккаунту.

**Назва групи**: Перегляд Подій.

**Назва Сценарію**: Користувач авторизований і шукає свої події календаря.

**Необхідні дії**: Користувач увійшов в аккаунт і шукає подію створено ним в календарі Google, або створено нещодавно.

**Очікуваний результат**: Користувач бачить всі свої події, та може з ними взаємодіяти.

**Назва групи**: Переключення на СЬОГОДНІ.

**Назва Сценарію**: користувач авторизований \ не авторизований натискає на кнопку СЬОГОДНІ.

**Необхідні дії**: В правому верхньому куті користувач натискає кнопку переключення «СЬОГОДНІ».

**Очікуваний результат**: Календар відобразить поточний день у виді перегляду користувача. Якщо це був режим РІК, переключить на поточний рік, ТИЖДЕНЬ, переключить на поточний тиждень.

**Назва групи**: Переключення вигляду РІК, МІСЯЦЬ...

**Назва Сценарію**: користувач авторизований \ не авторизований натискає на меню вибору вигляду.

**Необхідні дії**: В правому верхньому куті користувач натискає кнопку на меню з одним із надписів РІК, МІСЯЦЬ, ТИЖДЕНЬ, ДЕНЬ

**Очікуваний результат**: Після вибору одного із запропонованих варіантів, календар відобразить сітку даного перегляду. Тобто при виборі РІК- відобразяться 12 місяців, МІСЯЦЬ – дні поточного місяця, Тиждень – 7 днів тижня починаючи від Неділі, День – поточний день.

**Назва групи**: Вибір меню Календарів.

**Назва Сценарію**: користувач НЕ авторизований натискає на кнопку меню календарів.

**Необхідні дії**: В лівому верхньому куті користувач натискає на «Бургер меню» для відображення календарів.

**Очікуваний результат**: Меню буде пустим з надписом Авторизуйтесь.

**Назва групи**: Вибір меню Календарів.

**Назва Сценарію**: користувач авторизований натискає на кнопку меню календарів.

**Необхідні дії**: В лівому верхньому куті користувач натискає на «Бургер меню» для відображення календарів.

**Очікуваний результат**: При відкриті меню буде мати список Календарів користувача й зможе змінювати їх відображення на календарі.

**Назва групи**: Меню календарів.

**Назва Сценарію**: користувач авторизований, натискає на меню календарів, і натискає на кнопку біля одного з календарів.

**Необхідні дії**: В лівому верхньому куті користувач натискає на «Бургер меню» для відображення календарів, після чого вибирає зі списку календар та натискає на кнопку біля нього.

**Очікуваний результат**: Після відкриття меню та натискання на одну з кнопок, відповідні події календаря зникнуть з відображення.

**Назва групи**: Створення\Оновлення подій.

**Назва Сценарію**: користувач НЕ авторизований, натискає на меню створення подій або на календар в пустому місці.

**Необхідні дії**: В правому нижньому куті користувач натискає на кнопку створення подій «+».

**Очікуваний результат**: Користувачу виводиться повідомлення на екран, що необхідно авторизуватись для цього.

**Назва групи**: Створення\Оновлення подій.

**Назва Сценарію**: користувач авторизований, натискає на меню створення подій або на календар в пустому місці.

**Необхідні дії**: В правому нижньому куті користувач натискає на кнопку створення подій «+».

**Очікуваний результат**: Користувачу відкривається меню створення подій.

**Назва групи**: Створення\Оновлення подій, хибні дані.

**Назва Сценарію**: користувач авторизований, в меню створення подій не обирає календар.

**Необхідні дії**: Користувач під час створення події не обирає календар, для якого він створює цю подію.

**Очікуваний результат**: Користувачу після натискання кнопки Зберегти виводиться повідомлення, що про необхідність вибору календаря зі списку.

**Назва групи**: Створення\Оновлення подій, хибні дані.

**Назва Сценарію**: користувач авторизований, в меню створення подій вводить не коректні дані для полів часу.

**Необхідні дії**: Користувач під час створення події вводить в неправильному форматі час Початку\Кінця події.

**Очікуваний результат**: Користувачу після натискання кнопки Зберегти виводиться повідомлення, про неправильний формат часу, та формат в якому його треба ввести.

**Назва групи**: Створення\Оновлення подій, хибні дані.

**Назва Сценарію**: користувач авторизований, в меню створення подій залигає пусті поля Опис події та Назва події.

**Необхідні дії**: Користувач під час створення події вводить в неправильному видаляє запропонований текст, та залишає його пустим.

**Очікуваний результат**: Користувачу після натискання кнопки Зберегти, виводиться повідомлення про успішне створення\оновлення події й він може знайти його в календарі.

**Назва групи**: Видалення подій, хибні дані.

**Назва Сценарію**: користувач авторизований, в меню події натискає на кнопку видалення.

**Необхідні дії**: Користувач обирає одну зі своїх подій на в правому верхньому куті меню обирає іконку «Сміттєвого контейнеру».

**Очікуваний результат**: Після натискання кнопки подію буде видалено, й користувач побачить відповідне повідомлення на екрані про це.

# ВИСНОВКИ

У ході виконання цієї курсової роботи було створено власний веб-додаток календаря, що може значно допомогти кожному, хто хоче краще організовувати свій час. Застосування мов програмування, таких як Python на фреймворці Flask й JavaScript, та інструментів дизайну HTML та CSS, дозволило реалізувати інтуїтивний та ефективний інтерфейс як для юзерів на телефонах так і на комп’ютерах.

Важливим аспектом є успішне вирішення завдань, пов'язаних із створенням, редагуванням, видаленням та переглядом подій. Додаток підтримує такі функції та може відображати як свята, так і події різної довжини.

Проект актуальний у сучасному світі, де календарі є невід'ємним елементом життя людей для організації власног часу та справ. Реалізація додатку відображає практичність і популярність календарів, надаючи зручний та ефективний інструмент для планування й організації часу.

Ось деякі конкретні досягнення, яких вдалося досягти в рамках цього проекту:

Простий, але ефективний інтерфейс: додаток має простий і зрозумілий інтерфейс, що легко відображається навіть на найпростіших девайсах, а також дозволяє користувачам легко створювати, редагувати, видаляти та переглядати власні події.

Оскільки додаток побудований на базі Google, то події з мого додатка календаря завжди будуть доступні й будуть приходити повідомлення про події незабаром. Ці нагадування дозволяють користувачам краще планувати свій час і уникати пропущених подій.

У цілому, розробка цього додатку була успішною. Додаток є інтуїтивним, функціональним і доступним для широкого кола користувачів.

# CПИCOК ВИКOРИCТAНИX ДЖEРEЛ

* + - 1. Calendars API. Google for Developers. URL: https://developers.google.com/workspace/calendar/api/v3/reference/calendars (date of access: 24.05.2025).
      2. Events response. Google for Developers. URL: https://developers.google.com/workspace/calendar/api/v3/reference/events (date of access: 24.05.2025).
      3. Flask and SQLAlchemy Integration. Flask-SQLAlchemy – Flask-SQLAlchemy Documentation (3.1.x). URL: https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/ (date of access: 24.05.2025).
      4. Flask-Assests. Flask-Assets – Flask-Assets 0.12 documentation. URL: https://flask-assets.readthedocs.io/en/latest/ (date of access: 24.05.2025).
      5. Flask-Dance. Flask-Dance – Flask Dance 7.1.0 documentation. URL: https://flask-dance.readthedocs.io/en/latest/ (date of access: 24.05.2025).
      6. Get events. Google for Developers. URL: https://developers.google.com/workspace/calendar/api/v3/reference/events/get#python (date of access: 24.05.2025).
      7. Google API Python setup. Google for Developers. URL: https://developers.google.com/workspace/calendar/api/quickstart/python (date of access: 24.05.2025).
      8. OAuth. GitHub. URL: https://github.com/googleapis/google-api-python-client/blob/main/docs/oauth.md (date of access: 24.05.2025).
      9. SQLAlchemy. SQLAlchemy Documentation – SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/20/orm/quickstart.html (date of access: 24.05.2025).
      10. Unittest mock documentation. Python documentation. URL: https://docs.python.org/3/library/unittest.mock.html (date of access: 24.05.2025).
      11. Unittest mock examples. Python documentation. URL: https://docs.python.org/3/library/unittest.mock-examples.html (date of access: 24.05.2025).

# ДОДАТОК 1

Діаграми

НТУУ «КПІ ім. І. Сікорського» ТЕФ ІАТЕ ТВ-31

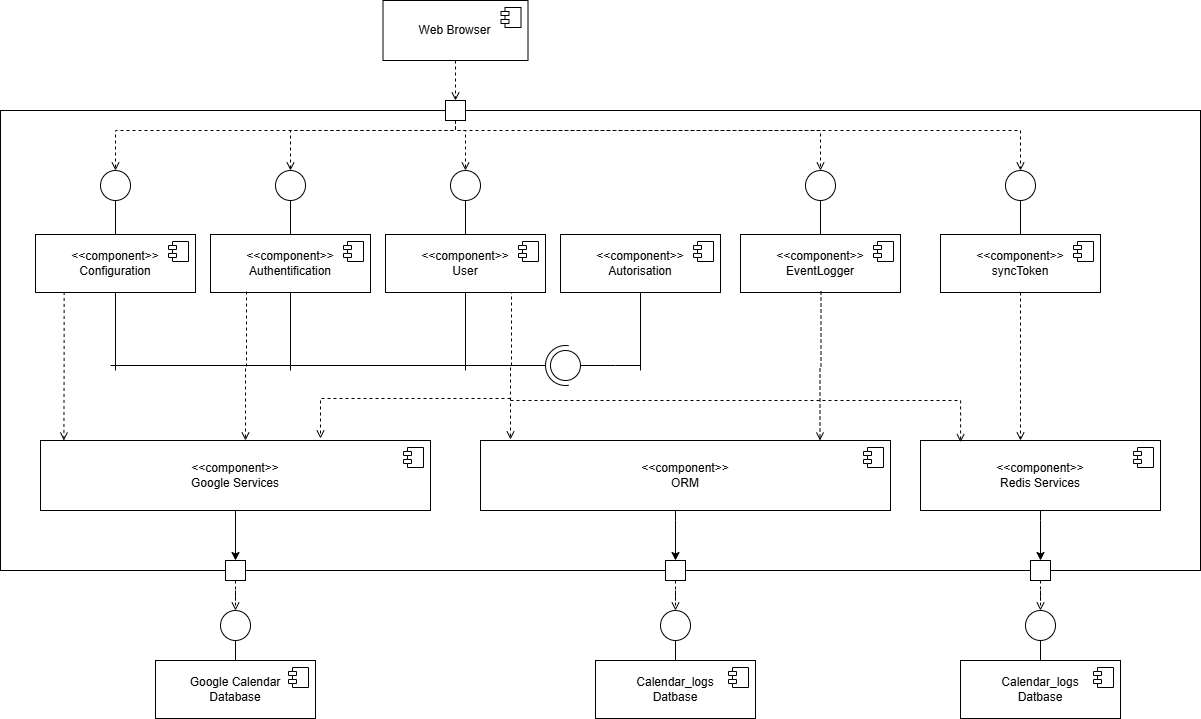
Листів 3

Київ – 2025

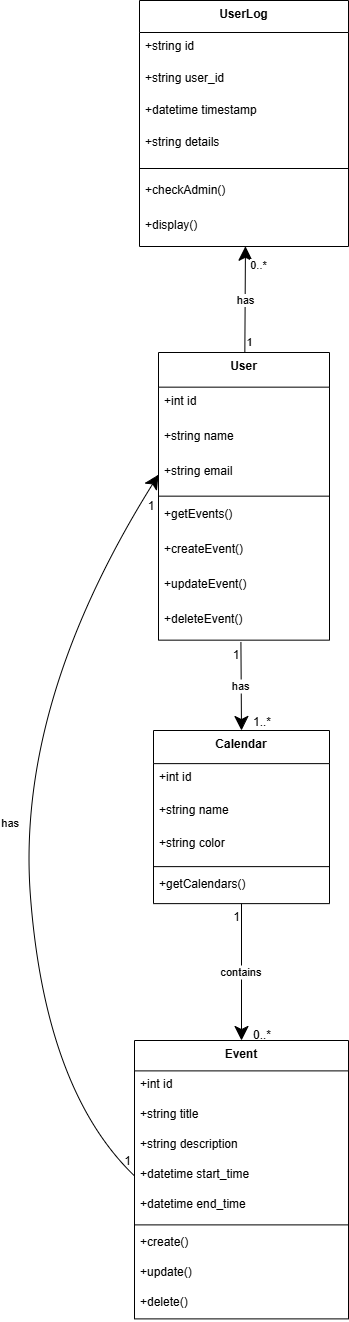
**Use case diagram**



**Components diagram**



**Class diagram**

****

# ДОДАТОК 2

Код програми

НТУУ «КПІ ім. І. Сікорського» ТЕФ ІАТЕ ТВ-31

Листів 67

Київ – 2025

PYTHON\FLASK

**App.py**

from flask import Flask, render\_template, session

from flask\_assets import Environment

import os

from assets import js, css

from google\_bp import google\_bp

from events\_control import events\_control

# from webhook import webhook\_control

from google\_auth import google\_auth\_control

# from ping import ping\_control

from logger\_setup import logger\_setup

from db\_setup import db\_control

from db\_log\_utils import db\_log\_control

from db\_init import db, init\_db

from dotenv import load\_dotenv

import os

load\_dotenv()

#

os.environ["OAUTHLIB\_INSECURE\_TRANSPORT"] = "1"

#

app = Flask(\_\_name\_\_)

app.secret\_key = os.getenv("SECRET\_KEY")

#

app.config["SQLALCHEMY\_DATABASE\_URI"] = os.getenv("POSTGRESQL")

app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = os.getenv(

"SQLALCHEMY\_TRACK\_MODIFICATIONS"

)

init\_db(app)

#

logger\_setup(app)

#

assets = Environment(app)

assets.register("js\_all", js)

assets.register("css\_all", css)

#

app.register\_blueprint(events\_control)

app.register\_blueprint(google\_bp, url\_prefix="/login")

# app.register\_blueprint(webhook\_control)

app.register\_blueprint(google\_auth\_control)

# app.register\_blueprint(ping\_control)

app.register\_blueprint(db\_control)

app.register\_blueprint(db\_log\_control)

with app.app\_context():

db.create\_all()

@app.route("/")

def index():

user = session.get("user\_info")

return render\_template("import.html", user=user)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True, port=5000)

**assets.py**

from flask\_assets import Bundle

js = Bundle(

"js/events/week\_day/create\_day/create\_day.js",

"js/events/week\_day/events\_day.js",

"js/events/week\_day/events\_week.js",

"js/view\_events\_menu/view\_events\_menu.js",

"js/events/year\_month/create\_month/create\_month.js",

"js/events/year\_month/events\_month.js",

"js/logs/logs.js",

"js/submenu\_container/submenu\_container.js",

"js/events/year\_month/events\_year.js",

"js/events/clicks/events\_clicks.js",

"js/events/utils/events\_modify\_utils.js",

"js/toast/toast.js",

"js/events/utils/events\_utils.js",

"js/events/utils/events\_view\_utils.js",

"js/events/DOMContentLoaded/DOMContentLoaded.js",

"js/const/const.js",

output="gen/packed.js",

)

css = Bundle(

"css/calendar/week\_day/create\_day/create\_day.css",

"css/calendar/week\_day/create\_day/generate\_holidays.css",

"css/calendar/week\_day/create\_day/lines\_container.css",

"css/view\_events\_menu/view\_events\_menu.css",

"css/toast/toast.css",

"css/calendar/week\_day/day.css",

"css/calendar/week\_day/week.css",

"css/create\_event\_button/create\_event\_button.css",

"css/calendar/view/calendar\_view.css",

"css/calendar/year\_month/create\_month/create\_month.css",

"css/calendar/year\_month/year.css",

"css/calendar/year\_month/month.css",

"css/config/config.css",

"css/submenu/submenu\_container.css",

"css/top/top\_container.css",

output="gen/packed.css",

)

**db\_init.py**

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

def init\_db(app):

db.init\_app(app)

**db\_log\_utils.py**

from flask import Blueprint, session, request, jsonify, render\_template, current\_app

from flask\_dance.contrib.google import google

from datetime import date, timedelta

from sqlalchemy import func

import json

from db\_setup import Log, User

from db\_init import db

db\_log\_control = Blueprint("db\_log\_control", \_\_name\_\_)

# ✅🗞️

def log\_action(email, message):

try:

user = User.query.filter\_by(Email=email).first()

if not user:

user = User(Email=email)

db.session.add(user)

db.session.commit()

log = Log(User\_Id=user.Id, Message=(message))

db.session.add(log)

db.session.commit()

return jsonify({"message": "Log\_action; OK"}), 200

except Exception as e:

current\_app.logger.info(f"Log\_action: Couldnt Write Logs; Error: {e}")

db.session.rollback()

return jsonify({"message": "Log\_action; Server Error"}), 500

# ✅🗞️

@db\_log\_control.route("/log", methods=["POST"])

def endpoint\_log\_action():

if not google.authorized or "user\_info" not in session:

user\_email = "guest"

else:

user\_email = session["user\_info"]["user\_email"]

data = request.get\_json()

message = data.get("message")

if not user\_email or not message:

log\_action("admin", "Endpoint\_log\_action: Emty Message or User Email")

return jsonify({"error": "Not Enough Data"}), 400

return log\_action(user\_email, message)

def get\_logs():

try:

logs = (

db.session.query(Log, User)

.join(User, Log.User\_Id == User.Id)

# .filter(func.date(Log.Log\_created\_date) == date.today()) # Якщо велике навантаження, то можна і по днях виводити

.order\_by(Log.Log\_created\_date.desc())

.all()

)

logs\_data = [

{

"email": user.Email,

"message": log.Message,

"date": (log.Log\_created\_date + timedelta(hours=3)).strftime(

"%Y-%m-%d %H:%M:%S"

),

}

for log, user in logs

]

return logs\_data

except Exception as e:

current\_app.logger.info(f"Get\_logs: Couldnt Recieve Logs; Error: {e}")

return []

# ✅🗞️

@db\_log\_control.route("/admin")

def view\_logs():

logs = get\_logs()

return render\_template("logs.html", logs=logs)

**db\_setup.py**

from flask import Blueprint, current\_app

from datetime import datetime, timedelta

from db\_init import db

db\_control = Blueprint("db\_control", \_\_name\_\_)

class User(db.Model):

\_\_tablename\_\_ = "Users"

Id = db.Column(db.Integer, primary\_key=True, autoincrement=True)

Email = db.Column(db.String(100), nullable=False)

User\_created\_date = db.Column(db.DateTime, default=datetime.utcnow())

Logs = db.relationship("Log", backref="user", cascade="all, delete", lazy=True)

class Log(db.Model):

\_\_tablename\_\_ = "Logs"

Id = db.Column(db.Integer, primary\_key=True, autoincrement=True)

User\_Id = db.Column(

db.Integer, db.ForeignKey("Users.Id", ondelete="CASCADE"), nullable=False

)

Message = db.Column(db.String(1000), nullable=False)

Log\_created\_date = db.Column(db.DateTime, default=datetime.utcnow())

**events\_control.py**

from flask import Blueprint, jsonify, request, redirect, url\_for, session, current\_app

from datetime import datetime

from flask\_dance.contrib.google import google

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import requests

from utils import event\_prepare\_data, get\_credentials, get\_calendar\_list

from redis\_setup import r

from db\_log\_utils import log\_action

import os

events\_control = Blueprint("events\_control", \_\_name\_\_)

WEBHOOK\_TTL\_SECONDS = 43200 # 43200 = 12 годин

HOURS = 24

# ✅🗞️

@events\_control.route("/get\_events/<year>")

def get\_event(year):

check = check\_auth()

# None якщо все ок

if check:

return check

year = int(year)

time\_min = f"{year}-01-01T00:00:00Z"

time\_max = f"{year + 1}-01-01T00:00:00Z"

#

user\_id = session["user\_info"]["user\_id"]

creds = get\_credentials()

service = build("calendar", "v3", credentials=creds)

all\_events = {}

syncType = {}

try:

calendars = get\_calendar\_list()

for calendar in calendars:

calendar\_id = calendar["id"]

# Свята окремо отримаємо

syncType[calendar\_id] = "Full"

key = f"User\_events\_syncToken:{user\_id}:{year}:{calendar\_id}"

params = {

"timeMin": time\_min,

"timeMax": time\_max,

"singleEvents": True,

"showDeleted": True,

}

# Якщо користувач активно користується, то ми отримуємо тільки зміни, щоб не навантажувати гугл апі та сервер,

# якщо користувач пішов, то ми отримуємо всі події, щоб впевнитись, що в нас актуальні дані

syncToken\_b = r.get(key)

if syncToken\_b:

syncToken = syncToken\_b.decode("utf-8")

# При першому виклику, отримуємо повну синхронізацію,

# при наступних в період WEBHOOK\_TTL\_SECONDS, отримуємо тільки зміни,

# якщо користувач не активний,

# то syncType = "Full" за замовчуванням, й отримуємо всі події по конкретному календарю

params = {"syncToken": syncToken, "showDeleted": True}

syncType[calendar\_id] = "Partial"

response = service.events().list(calendarId=calendar\_id, \*\*params).execute()

events = response.get("items", [])

all\_events[calendar\_id] = events

syncToken = response.get("nextSyncToken")

r.setex(key, WEBHOOK\_TTL\_SECONDS, syncToken)

except HttpError as e:

if e.resp.status == 410:

log\_action("admin", f"syncToken: Google API; Error: {e}")

return jsonify({"message": "syncToken:Error; Please Relogin"}), 410

log\_action("admin", f"Get\_event: Google API; Error: {e}")

return jsonify({"message": "Server Error"}), 500

# current\_app.logger.info(

# f'User: {session["user\_info"]["user\_email"]}; Get\_event: updatedMin param for get\_event()'

# )

#

result = []

color\_id = 0

for calendar\_id, events in all\_events.items():

color\_id += 1

for event in events:

result.extend(event\_prepare\_data(calendar\_id, color\_id, event, HOURS))

if result:

log\_action(

session["user\_info"]["user\_email"],

f"Get\_event: Received",

)

response\_data = { # Просто словник

"syncType": syncType,

"result": result,

}

return jsonify(response\_data), 200

else:

if len(result) == 0:

# current\_app.logger.info(

# f'User: {session["user\_info"]["user\_email"]}; Get\_event: no Events to Receive'

# )

log\_action(

session["user\_info"]["user\_email"],

f"Get\_event: no Events to Receive",

)

return "", 204

else:

log\_action(

session["user\_info"]["user\_email"],

f"Get\_event: Unknown Error; Result: {result}",

)

# Невідома помилка, повертаємо помилку 404

return jsonify({"message": "Unknown Error"}), 404

@events\_control.route("/get\_holidays/<year>")

def get\_holidays(year):

try:

year = int(year)

calendar\_id = os.getenv("HOLIDAY\_CALENDAR")

api\_key = os.getenv("API\_KEY")

time\_min = f"{year}-01-01T00:00:00Z"

time\_max = f"{year + 1}-01-01T00:00:00Z"

url = (

f"https://www.googleapis.com/calendar/v3/calendars/{calendar\_id}/events"

f"?key={api\_key}&timeMin={time\_min}&timeMax={time\_max}&singleEvents=true"

)

response = requests.get(url)

data = response.json()

events = data.get("items", [])

holidays = []

# import json

# print("Google API Response:", json.dumps(data, indent=2))

for event in events:

holidays.extend(event\_prepare\_data(calendar\_id, 99, event, HOURS))

return jsonify(holidays), 200

except Exception as e:

log\_action("admin", f"Get\_holidays: Error: {e}")

return jsonify({"message": "Get\_holidays; Server Error"}), 500

@events\_control.route("/create\_event/<calendar\_id>/<color\_id>", methods=["POST"])

def create\_event(calendar\_id, color\_id):

check = check\_auth()

# None якщо все ок

if check:

return check

try:

data = request.get\_json()

creds = get\_credentials()

service = build("calendar", "v3", credentials=creds)

event\_body = {

"summary": data.get("summary", "Без назви"),

"description": data.get("description", ""),

"start": {

"dateTime": data["start"],

"timeZone": "Europe/Kyiv",

},

"end": {

"dateTime": data["end"],

"timeZone": "Europe/Kyiv",

},

}

response = (

service.events().insert(calendarId=calendar\_id, body=event\_body).execute()

)

log\_action(session["user\_info"]["user\_email"], "Create\_event: created")

event = []

event.extend(

event\_prepare\_data(calendar\_id, color\_id, response, HOURS)

) # colorId взяти з джс

return jsonify({"message": "Event created", "event": event}), 201

except Exception as e:

log\_action(session["user\_info"]["user\_email"], f"Create\_event: Error: {e}")

return jsonify({"message": "Create\_event; Server Error"}), 500

@events\_control.route(

"/update\_event/<calendar\_id>/<color\_id>/<event\_id>", methods=["PATCH"]

)

def update\_event(calendar\_id, color\_id, event\_id):

check = check\_auth()

# None якщо все ок

if check:

return check

try:

data = request.get\_json()

creds = get\_credentials()

service = build("calendar", "v3", credentials=creds)

# якщо передати тільки частину, то воно може вказатись як порожнє, тому для безпечності

event = service.events().get(calendarId=calendar\_id, eventId=event\_id).execute()

if "summary" in data:

event["summary"] = data["summary"]

if "description" in data:

event["description"] = data["description"]

if "start" in data:

event["start"]["dateTime"] = data["start"]

if "end" in data:

event["end"]["dateTime"] = data["end"]

# Оскільки немає локалізації, то зайжди буде Україна

event["start"]["timeZone"] = "Europe/Kyiv"

event["end"]["timeZone"] = "Europe/Kyiv"

updated\_event = (

service.events()

.update(calendarId=calendar\_id, eventId=event\_id, body=event)

.execute()

)

event = []

event.extend(

event\_prepare\_data(calendar\_id, color\_id, updated\_event, HOURS)

) # colorId взяти з джс

log\_action(session["user\_info"]["user\_email"], f"Update\_event: Updated")

return jsonify({"message": "Event updated", "event": event}), 200

except Exception as e:

log\_action(

session["user\_info"]["user\_email"], f"Update\_event: Not Updated; Error: {e}"

)

return jsonify({"message": "Update\_event; Server Error"}), 500

@events\_control.route("/delete\_event/<calendar\_id>/<event\_id>", methods=["DELETE"])

def delete\_event(calendar\_id, event\_id):

check = check\_auth()

# None якщо все ок

if check:

return check

try:

creds = get\_credentials()

service = build("calendar", "v3", credentials=creds)

service.events().delete(calendarId=calendar\_id, eventId=event\_id).execute()

log\_action(session["user\_info"]["user\_email"], f"Delete\_event: {event\_id}")

return jsonify({"message": "Event deleted", "event\_id": event\_id}), 200

except Exception as e:

log\_action(session["user\_info"]["user\_email"], f"Delete\_event: Error: {e}")

return jsonify({"message": "Delete\_event; Server Error"}), 500

@events\_control.route("/list\_calendars")

def list\_calendars():

check = check\_auth()

# None якщо все ок

if check:

return check

try:

calendars = get\_calendar\_list()

if len(calendars) > 0:

log\_action("admin", f"list\_calendars: len(calendars): {len(calendars)}")

return (

jsonify({"message": "Calendars Received", "calendars": calendars}),

200,

)

return "", 204

except Exception as e:

log\_action("admin", f"List\_calendars: Error: {e}")

return jsonify({"message": "List\_calendars; Server Error"}), 500

def check\_auth():

if not google.authorized or "user\_info" not in session:

return jsonify({"message": "Unauthorized"}), 401

return None

**google\_auth.py**

from flask\_dance.contrib.google import google

from flask import redirect, url\_for, session, Blueprint, jsonify

# from webhook import webhook\_controller

from redis\_setup import r

from db\_log\_utils import log\_action

google\_auth\_control = Blueprint("google\_auth\_control", \_\_name\_\_)

# ✅🗞️

@google\_auth\_control.route("/login")

def user\_login():

resp = google.get("/oauth2/v2/userinfo")

assert resp.ok, resp.text

user\_info = resp.json()

session["user\_info"] = {

"name": user\_info["name"],

"picture": user\_info["picture"],

"user\_id": user\_info["id"],

"user\_email": user\_info["email"],

}

log\_action(

session["user\_info"]["user\_email"],

f"User: Login",

)

return redirect(url\_for("index"))

# ✅🗞️

@google\_auth\_control.route("/logout")

def user\_logout():

user\_id = session["user\_info"]["user\_id"]

key\_pattern = f"User\_events\_syncToken:{user\_id}:\*"

keys = list(r.scan\_iter(match=key\_pattern))

for key in keys:

# Якщо в користувача не завантажуються івенти, він може просто перезайти в аккаунт для вирішення проблеми

deleted = r.delete(key)

if deleted == 0:

log\_action(

session["user\_info"]["user\_email"],

f"User: Logout; User\_events\_syncToken: {key}; NOT deleted",

)

remaining\_keys = list(r.scan\_iter(match=key\_pattern))

if remaining\_keys:

log\_action(

session["user\_info"]["user\_email"],

f"Logout: Some keys not deleted: {remaining\_keys}",

)

log\_action(

session["user\_info"]["user\_email"],

f"User: Logout",

)

session.clear()

return redirect("/")

**google\_bp.py**

from flask\_dance.contrib.google import make\_google\_blueprint

import os

google\_bp = make\_google\_blueprint(

client\_id=os.getenv("CLIENT\_ID"),

client\_secret=os.getenv("CLIENT\_SECRET"),

redirect\_url="/login",

scope=[

"https://www.googleapis.com/auth/userinfo.email",

"https://www.googleapis.com/auth/userinfo.profile",

"https://www.googleapis.com/auth/calendar",

"openid",

],

reprompt\_consent=True,

offline=True,

)

**logger\_setup.py**

import logging

import sys

def logger\_setup(app):

root = logging.getLogger()

log\_level = logging.INFO

root.setLevel(log\_level)

handler = logging.StreamHandler(sys.stdout)

handler.setLevel(log\_level)

formatter = logging.Formatter(

"%(asctime)s - %(name)s - %(levelname)s - %(message)s"

)

handler.setFormatter(formatter)

app.logger.setLevel(log\_level)

**redis\_setup.py**

import redis

import os

r = redis.Redis(

host="coherent-sloth-18071.upstash.io",

port=6379,

password=os.getenv("REDIS\_PASSWORD"),

ssl=True,

)

**utils.py**

from dateutil import parser

from flask\_dance.contrib.google import google

from google.oauth2.credentials import Credentials

from googleapiclient.discovery import build

from datetime import datetime, timedelta, time

from google\_bp import google\_bp

from db\_log\_utils import log\_action

from redis\_setup import r

import os

def event\_prepare\_data(calendar\_id, color\_id, event, HOURS):

segments = []

start\_info = event.get("start", {})

end\_info = event.get("end", {})

event\_id = event.get("id")

summary = event.get("summary", "Без назви")

description = event.get("description", "")

if "holiday" in calendar\_id:

dt\_start\_holidays\_d = datetime.strptime(start\_info["date"], "%Y-%m-%d")

dt\_end\_holidays\_d = datetime.strptime(end\_info["date"], "%Y-%m-%d")

dt\_start\_holidays = dt\_start\_holidays\_d.date()

dt\_end\_holidays = dt\_end\_holidays\_d.date()

segments.append(

{

"calendarId": calendar\_id,

"event\_id": event\_id,

"summary": summary,

"description": description,

"eventStart": dt\_start\_holidays\_d.isoformat(),

"eventEnd": dt\_end\_holidays\_d.isoformat(),

"start": {

"year": dt\_start\_holidays.year,

"month": dt\_start\_holidays.month,

"day": dt\_start\_holidays.day,

},

"end": {

"year": dt\_end\_holidays.year,

"month": dt\_end\_holidays.month,

"day": dt\_end\_holidays.day,

},

}

)

return segments

status = event.get("status")

creator\_email = event.get("creator", {}).get("email", "Невідомо")

attendees = event.get("attendees", [])

attendee\_emails = [a.get("email", "Невідомо") for a in attendees]

dt\_start = (

parse\_datetime(start\_info.get("dateTime"))

if "dateTime" in start\_info

else datetime.strptime(start\_info.get("date"), "%Y-%m-%d")

)

dt\_end = (

parse\_datetime(end\_info.get("dateTime"))

if "dateTime" in end\_info

else datetime.strptime(end\_info.get("date"), "%Y-%m-%d")

)

dt\_end\_save = dt\_end

if dt\_start and dt\_end:

long\_event = False

if (dt\_end.date() - dt\_start.date()).days > 0:

long\_event = True

dt\_end -= timedelta(seconds=1)

current\_day = dt\_start.date()

last\_day = dt\_end.date()

while current\_day <= last\_day:

segment\_start = datetime.combine(current\_day, datetime.min.time())

segment\_end = datetime.combine(current\_day, datetime.max.time())

if current\_day == dt\_start.date():

segment\_start = dt\_start

if current\_day == dt\_end.date():

segment\_end = dt\_end

start\_minutes = segment\_start.hour \* 60 + segment\_start.minute

end\_minutes = segment\_end.hour \* 60 + segment\_end.minute

start\_position = start\_minutes / (HOURS \* 60)

end\_position = ((end\_minutes - start\_minutes) / 60) / HOURS

# log\_action(

# "admin",

# f"event\_prepare\_data: current\_day.day: {current\_day.day}; dt\_start: {dt\_start}; dt\_end: {dt\_end}; event\_id: {event\_id}; start\_position: {start\_position}; end\_position: {end\_position}",

# )

segments.append(

{

"calendarId": calendar\_id,

"summary": summary,

"description": description,

"event\_id": event\_id,

"colorId": color\_id,

"status": status,

# "creator\_email": creator\_email,

# "attendees": attendees,

# "attendee\_emails": attendee\_emails,

"eventStart": dt\_start.isoformat(),

"eventEnd": dt\_end\_save.isoformat(),

"start": {

"year": current\_day.year,

"month": current\_day.month,

"day": current\_day.day,

"hour": segment\_start.hour,

"minutes": segment\_start.minute,

},

"end": {

"year": current\_day.year,

"month": current\_day.month,

"day": current\_day.day,

"hour": segment\_end.hour,

"minutes": segment\_end.minute,

},

"startMinutes": start\_minutes,

"endMinutes": end\_minutes,

"startPosition": start\_position,

"endPosition": end\_position,

"longEvent": long\_event,

"overlaps": 0,

"column": 0,

}

)

current\_day += timedelta(days=1)

# log\_action("admin", f"event\_prepare\_data: segments: {segments};")

return segments

def parse\_datetime(dt\_str):

if not dt\_str:

return None

try:

return parser.parse(dt\_str)

except Exception as e:

log\_action("admin", f"Parse\_datetime: Parser Invalid Value; Error: {e}")

return None

# Фактично не потрібен, бо я і так пінгую сервер 😒😒😒😒😒😒😒😒😒

# def get\_user\_id(channel\_id):

# parts = channel\_id.split("=")

# if (

# len(parts) == 6

# and parts[0] == "user"

# and parts[2] == "calendar"

# and parts[4] == "time"

# ):

# user\_id = parts[1]

# calendar = parts[3]

# return user\_id, calendar

# else:

# log\_action("admin", f"Get\_user\_id: channel\_id Parts Invalid Lenght")

# return None, None

# Фактично не потрібен, бо я і так пінгую сервер 😒😒😒😒😒😒😒😒😒

def get\_credentials():

token = google.token["access\_token"]

refresh\_token = google.token.get("refresh\_token")

client\_id = google\_bp.client\_id

client\_secret = google\_bp.client\_secret

token\_uri = "https://oauth2.googleapis.com/token"

scope = google\_bp.scope

credentials = Credentials(

token=token,

refresh\_token=refresh\_token,

token\_uri=token\_uri,

client\_id=client\_id,

client\_secret=client\_secret,

scopes=scope,

)

return credentials

def get\_calendar\_list():

creds = get\_credentials()

service = build("calendar", "v3", credentials=creds)

try:

calendar\_list = service.calendarList().list().execute()

except Exception as e:

log\_action("admin", f"get\_calendar\_list: Error: {e}")

raise e

color\_id = 0

calendars = []

for calendar in calendar\_list.get("items", []):

if not "holiday" in calendar.get("id"):

color\_id += 1

calendars.append(

{

"id": calendar.get("id"),

"summary": calendar.get("summary"),

"timeZone": calendar.get("timeZone"),

"accessRole": calendar.get("accessRole"),

"colorId": color\_id,

}

)

return calendars

HTML

**Base.html**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Calendar</title>

{% assets "css\_all" %}

<link rel="stylesheet" href="{{ ASSET\_URL }}" />

{% endassets %}

<link

rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css"

/>

</head>

<body>

{% block content %}{% endblock %}

<!-- 🛑 🛑 -->

<script id="user\_json" type="application/json">

{{ user | tojson | safe }}

</script>

{% assets "js\_all" %}

<script src="{{ ASSET\_URL }}" defer></script>

{% endassets %}

<script src="https://accounts.google.com/gsi/client" async defer></script>

</body>

</html>

**calendar.html**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Calendar</title>

{% assets "css\_all" %}

<link rel="stylesheet" href="{{ ASSET\_URL }}" />

{% endassets %}

<link

rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.0/css/all.min.css"

/>

</head>

<body>

{% block content %}{% endblock %}

<!-- 🛑 🛑 -->

<script id="user\_json" type="application/json">

{{ user | tojson | safe }}

</script>

{% assets "js\_all" %}

<script src="{{ ASSET\_URL }}" defer></script>

{% endassets %}

<script src="https://accounts.google.com/gsi/client" async defer></script>

</body>

</html>

**create\_event\_button.html**

<div class="event-button" onclick="createEventsMenu()">

<svg

xmlns="http://www.w3.org/2000/svg"

viewBox="0 0 448 512"

class="event-icon"

>

<path

d="M256 80c0-17.7-14.3-32-32-32s-32 14.3-32 32v144H48c-17.7 0-32 14.3-32 32s14.3 32 32 32h144v144c0 17.7 14.3 32 32 32s32-14.3 32-32V288h144c17.7 0 32-14.3 32-32s-14.3-32-32-32H256V80z"

/>

</svg>

</div>

**import.html**

{% extends "base.html" %}

{% block content %}

{% include "create\_event\_button.html" %}

{% include "view\_events\_menu.html" %}

{% include "top\_container.html"%}

{% include "toast.html"%}

<div class="bottom-container">

{% include "submenu\_container.html" %}

{% include "calendar.html" %}

</div>

{%endblock %}

**logs.html**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Логи</title>

</head>

<body

style="

background-color: rgb(255, 182, 37);

display: flex;

flex-direction: column;

"

>

<h1 style="text-align: center">Логи користувачів</h1>

<div style="width: 100%; display: flex">

<input

type="text"

id="searchInput"

placeholder="Пошук..."

onkeyup="searchLogs()"

style="flex: 1"

/>

<button onclick="searchLogs()">Пошук</button> <datalist>b</datalist>

</div>

{% if logs %}

<table

border="1"

width="100%"

style="background-color: antiquewhite; margin-top: 0.2rem"

>

<thead>

<tr>

<th style="color: blue">Дата</th>

<th style="color: green">Користувач</th>

<th style="color: red">Повідомлення</th>

</tr>

</thead>

<tbody id="logsTable">

{% for log in logs[:1000] %}

<tr class="log-list">

<td style="color: blue">{{ log.date }}</td>

<td style="color: green">{{ log.email }}</td>

<td style="color: red">{{ log.message }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% else %}

<p>Немає логів для відображення.</p>

{% endif %}

</body>

</html>

**submenu\_container.html**

<div class="submenu\_container" id="submenu\_container">

<!-- 🛑MENU LIST🛑 -->

<div class="submenu-color">

<div id="calendarsSubmenu" class="submenu"></div>

</div>

</div>

**toast.html**

<div id="toast" class="hidden toast"></div>

**top\_container.html**

<div class="top\_container">

<div class="menu\_container">

<button class="menu\_button" id="menu\_button" onclick="toggleMenu()">

<svg

class="button\_icon"

xmlns="http://www.w3.org/2000/svg"

viewBox="0 0 448 512"

>

<path

d="M0 96C0 78.3 14.3 64 32 64l384 0c17.7 0 32 14.3 32 32s-14.3 32-32 32L32 128C14.3 128 0 113.7 0 96zM0 256c0-17.7 14.3-32 32-32l384 0c17.7 0 32 14.3 32 32s-14.3 32-32 32L32 288c-17.7 0-32-14.3-32-32zM448 416c0 17.7-14.3 32-32 32L32 448c-17.7 0-32-14.3-32-32s14.3-32 32-32l384 0c17.7 0 32 14.3 32 32z"

/>

</svg>

</button>

<div class="change-view-date-top-container">

<div class="arrows-button-container">

<button class="arrows-button-calendar" onclick="changeDateView(`prev`)">

<svg

class="arrows-calendar"

xmlns="http://www.w3.org/2000/svg"

width="24"

height="24"

viewBox="0 0 24 24"

>

<path

d="M15 19l-7-7 7-7"

stroke="currentColor"

stroke-width="2"

fill="none"

/>

</svg>

</button>

<button class="arrows-button-calendar" onclick="changeDateView(`next`)">

<svg

class="arrows-calendar"

xmlns="http://www.w3.org/2000/svg"

width="24"

height="24"

viewBox="0 0 24 24"

>

<path

d="M9 5l7 7-7 7"

stroke="currentColor"

stroke-width="2"

fill="none"

/>

</svg>

</button>

</div>

</div>

<div class="show-current-date-container">

<!-- Маска, записуємо наййдовший місяць, щоб не займати зайвий простір фіксованою шириною -->

<!-- Можна і фіксованою шириною, але при зміні висоти доведеться також змінювати і ширину -->

<div

class="show-current-date-month"

style="visibility: hidden; height: 0; overflow: hidden"

>

Вересень

</div>

<div id="showCurrentDateYear" class="show-current-date-year"></div>

<div id="showCurrentDateMonth" class="show-current-date-month"></div>

</div>

</div>

<!-- 🛑TOP LIST🛑 -->

<div class="app-info-container">

<button onclick="showView('today')">Сьогодні</button>

<div class="view-toggle">

<select id="viewSelect" onchange="changeView(event)">

<option selected class="hidden"></option>

<option value="year">Рік</option>

<option value="month">Місяць</option>

<option value="week">Тиждень</option>

<option value="day">День</option>

</select>

</div>

<!-- 🛑AUTH LIST🛑 -->

<div class="auth\_container">

{% if user %}

<img

src="{{ user.picture }}"

id="profile\_pic"

class="profile\_pic"

onclick="toggleProfileMenu()"

/>

<div id="profile\_menu" class="profile\_menu hidden">

<a id="googleLogout" href="/logout"> Вийти </a>

</div>

{% else %}

<div class="menu\_button">

<svg

class="button\_icon"

id="googleLogin"

xmlns="http://www.w3.org/2000/svg"

viewBox="0 0 488 512"

>

<path

d="M488 261.8C488 403.3 391.1 504 248 504 110.8 504 0 393.2 0 256S110.8 8 248 8c66.8 0 123 24.5 166.3 64.9l-67.5 64.9C258.5 52.6 94.3 116.6 94.3 256c0 86.5 69.1 156.6 153.7 156.6 98.2 0 135-70.4 140.8-106.9H248v-85.3h236.1c2.3 12.7 3.9 24.9 3.9 41.4z"

/>

</svg>

</div>

{% endif %}

</div>

</div>

</div>

**view\_events\_menu.html**

<div id="eventsViewEvent" class="event-view-menu hidden">

<div class="view-menu-top-buttons-container">

<div class="view-menu-choose-calendar">

<select id="viewMenuChooseCalendar" onchange=""></select>

</div>

<div class="event-icons">

<div id="deleteEvent" class="icon-view" onclick="deleteEventController()">

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 448 512">

<path

d="M135.2 17.7L128 32 32 32C14.3 32 0 46.3 0 64S14.3 96 32 96l384 0c17.7 0 32-14.3 32-32s-14.3-32-32-32l-96 0-7.2-14.3C307.4 6.8 296.3 0 284.2 0L163.8 0c-12.1 0-23.2 6.8-28.6 17.7zM416 128L32 128 53.2 467c1.6 25.3 22.6 45 47.9 45l245.8 0c25.3 0 46.3-19.7 47.9-45L416 128z"

/>

</svg>

</div>

<div id="editEvent" class="icon-view" onclick="editEventsMenu()">

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512">

<path

d="M471.6 21.7c-21.9-21.9-57.3-21.9-79.2 0L362.3 51.7l97.9 97.9 30.1-30.1c21.9-21.9 21.9-57.3 0-79.2L471.6 21.7zm-299.2 220c-6.1 6.1-10.8 13.6-13.5 21.9l-29.6 88.8c-2.9 8.6-.6 18.1 5.8 24.6s15.9 8.7 24.6 5.8l88.8-29.6c8.2-2.7 15.7-7.4 21.9-13.5L437.7 172.3 339.7 74.3 172.4 241.7zM96 64C43 64 0 107 0 160L0 416c0 53 43 96 96 96l256 0c53 0 96-43 96-96l0-96c0-17.7-14.3-32-32-32s-32 14.3-32 32l0 96c0 17.7-14.3 32-32 32L96 448c-17.7 0-32-14.3-32-32l0-256c0-17.7 14.3-32 32-32l96 0c17.7 0 32-14.3 32-32s-14.3-32-32-32L96 64z"

/>

</svg>

</div>

<div class="icon-view" onclick="closeEventMenu()">

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 384 512">

<path

d="M342.6 150.6c12.5-12.5 12.5-32.8 0-45.3s-32.8-12.5-45.3 0L192 210.7 86.6 105.4c-12.5-12.5-32.8-12.5-45.3 0s-12.5 32.8 0 45.3L146.7 256 41.4 361.4c-12.5 12.5-12.5 32.8 0 45.3s32.8 12.5 45.3 0L192 301.3 297.4 406.6c12.5 12.5 32.8 12.5 45.3 0s12.5-32.8 0-45.3L237.3 256 342.6 150.6z"

/>

</svg>

</div>

</div>

</div>

<div class="event-header">

<div class="event-view-summary" id="eventSummary"></div>

<div class="event-view-description" id="eventDescription">Без опису</div>

</div>

<div class="event-view-time-container" id="timeStart">

<strong>Початок:</strong>

<div class="event-view-time" id="eventStart">Час початку</div>

</div>

<div class="event-view-time-container" id="timeEnd">

<strong>Кінець:</strong>

<div class="event-view-time" id="eventEnd">Час завершення</div>

</div>

<div id="saveEvent" class="save-button inactive">Зберегти</div>

</div>

JS

**events\_clicks.js**

function outerClick(event) {

const slot = event.currentTarget;

const rect = slot.getBoundingClientRect();

const clickY = event.clientY - rect.top;

const slotHeight = rect.height;

const totalMinutes = Math.floor((clickY / slotHeight) \* 24 \* 60);

const some\_hour = Math.floor(totalMinutes / 60);

const some\_minute = Math.floor((totalMinutes % 60) / 15) \* 15;

const date = slot.dateData;

const startDate = new Date(

`${date}T${String(some\_hour).padStart(2, "0")}:${String(

some\_minute

).padStart(2, "0")}:00`

);

const endDate = new Date(startDate.getTime() + 30 \* 60 \* 1000);

const formattedStart = formatTime(null, startDate.toISOString());

const formattedEnd = formatTime(null, endDate.toISOString());

createEventsMenu();

const eventStart = document.getElementById("eventStart");

const eventEnd = document.getElementById("eventEnd");

eventStart.value = formattedStart;

eventEnd.value = formattedEnd;

}

function innerClick(e) {

e.stopPropagation();

currentEvent = e.currentTarget.eventData;

showEventDetails(currentEvent);

}

**DOMContentLoaded.js**

document.addEventListener("DOMContentLoaded", async () => {

const user\_json = document.getElementById("user\_json").textContent || "null";

user = JSON.parse(user\_json);

initCachedEvents(user);

initCachedHolidays(); // поки тестую

// localStorage.removeItem("cachedHolidays");

initCachedView();

initCachedDate();

initCachedCalendars();

// initSelectedCalendars();

// Оновлюємо сторінку для оновлення подій

await loadUserCalendars(user);

await populateCalendarsSubmenu();

showView(cachedView);

// Також завантажує вихідні, тому треба

await loadEvents();

const loginBtn = document.getElementById("googleLogin");

if (loginBtn) {

loginBtn.addEventListener("click", () => {

window.location.href = "/login/google";

});

}

if (user) {

// Якщо користувач не авторизовний, нема сенсу, кожен раз кидати запит

loadEventsSetInterval();

// Фактично не потрібен, бо я і так пінгую сервер 😒

// checkUserWebhook();

// checkUserWebhookSetInterval();

}

});

**events\_modify\_utils.js**

// Поки нічого не перевіряв, але має бути правильно

// Працює

async function createEvent(calendarId, colorId, eventData) {

try {

const response = await fetch(

`/create\_event/${encodeURIComponent(calendarId)}/${colorId}`,

{

method: "POST",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(eventData),

}

);

const result = await handleErrors(response);

showToast(result.message);

// Щоб не чекати на вебхук

loadEventsUpdate({

syncType: { [calendarId]: "Partial" },

result: result.event,

});

showView(cachedView);

// Оскільки вебхука вже немає, можна просто викликати loadEvents(), отримає тільки оновлення

// loadEvents();

} catch (error) {

showToast("!: " + error.message);

}

}

// Працює

async function updateEvent(calendarId, colorId, eventId, updatedData) {

try {

const response = await fetch(

`/update\_event/${encodeURIComponent(

calendarId

)}/${colorId}/${encodeURIComponent(eventId)}`,

{

method: "PATCH",

headers: {

"Content-Type": "application/json",

},

body: JSON.stringify(updatedData),

}

);

const result = await handleErrors(response);

showToast(result.message);

// Щоб не чекати на вебхук

loadEventsUpdate({

syncType: { [calendarId]: "Partial" },

result: result.event,

});

showView(cachedView);

// Оскільки вебхука вже немає, можна просто викликати loadEvents(), отримає тільки оновлення

// loadEvents();

} catch (error) {

showToast("!: " + error.message);

}

}

// Працює

async function deleteEvent(calendarId, eventId) {

try {

const response = await fetch(

`/delete\_event/${encodeURIComponent(calendarId)}/${encodeURIComponent(

eventId

)}`,

{

method: "DELETE",

}

);

const result = await handleErrors(response);

showToast(result.message);

// Щоб не чекати на вебхук

loadEventsUpdate({

syncType: { [calendarId]: "Partial" },

result: [

{ calendarId: calendarId, event\_id: eventId, status: "cancelled" },

],

});

showView(cachedView);

// Оскільки вебхука вже немає, можна просто викликати loadEvents(), отримає тільки оновлення

// loadEvents();

} catch (error) {

showToast("!: " + error.message);

}

}

// // Працює

// async function createEventTest() {

// const start = new Date();

// let end = new Date();

// end.setHours(end.getHours() + 1);

// await createEvent({

// start: start.toISOString(),

// end: end.toISOString(),

// summary: "Тестова подія",

// description: "Це опис",

// });

// }

// // Працює

// async function updateEventTest() {

// const start = new Date();

// let end = new Date();

// end.setHours(end.getHours() + 4);

// const eventId = "cuq20gbud5c1o2hknonrp3jitc";

// await updateEvent(eventId, {

// start: start.toISOString(),

// end: end.toISOString(),

// });

// }

// // Працює

// async function deleteEventTest() {

// const eventId = "sbgbiotqoslts1cmj0vts1s740";

// await deleteEvent(eventId);

// }

**events\_utils.js**

async function loadEvents(year = cachedDate.getFullYear()) {

// Оскільки кожен рік має хоча б одну подію, перевіряємо чи збігається отриманий рік з святами в кеші

const hasYear = cachedHolidays.some((obj) => obj.start.year === year);

// Отримує значення, тільки коли в нас змінився рік, на той, що не підвантажено

if (cachedHolidays.length === 0 || !hasYear) {

// Якщо в нас cachedHolidays пустий, то треба отримати вихідні

const holidayEvents = await loadHolidayEvents(year);

// Вже є обробка помилок у функції вище

if (holidayEvents.length != 0) {

showToast("Year Loaded");

cachedHolidays.push(...holidayEvents);

localStorage.setItem("cachedHolidays", JSON.stringify(cachedHolidays));

}

}

if (user) {

const userEvents = await loadUserEvents(year);

if (userEvents) {

// фактично дублюється зі ствренням подій, але він не буде так часто завантажуватись тому ОК

showToast("Events Updated");

loadEventsUpdate(userEvents);

}

}

showView(cachedView);

}

async function loadUserEvents(year) {

try {

const response = await fetch(`/get\_events/${year}`);

return await handleErrors(response);

} catch {}

}

async function loadHolidayEvents(year) {

try {

const response = await fetch(`/get\_holidays/${year}`);

return await handleErrors(response);

} catch {}

}

async function handleErrors(response) {

if (response.status === 204) {

showToast("No Items to Receive");

return null;

}

const data = await response.json();

if (response.status === 201) {

console.log(data.message);

} else if (!response.ok) {

logAction(data.message);

showToast("!: " + data.message);

throw new Error(data.message);

}

return data;

}

async function loadUserCalendars(user, update = false) {

if (user) {

if (cachedCalendars.length === 0 || update) {

try {

const response = await fetch("/list\_calendars");

const data = await handleErrors(response);

if (data === null) {

return;

}

cachedCalendars = data.calendars;

localStorage.setItem(

"cachedCalendars",

JSON.stringify(cachedCalendars)

);

} catch {}

}

} else {

cachedCalendars = [];

localStorage.setItem("cachedCalendars", JSON.stringify(cachedCalendars));

}

}

// Доробити

function getCalendarDayEvents(year, month, day) {

if (user) {

if (cachedEvents.length > 0) {

// Треба отримувати кешовані календарі, та відповідно обирати, що показувати, а що ні

const choosedCalendarEvents = getChoosedCalendarEvents();

const eventsForDay = eventsForDaySort(

choosedCalendarEvents,

year,

month,

day

);

return eventsForDay;

} else {

let message = `getCalendarDayEvents: cachedEvents Length is Empty;`;

console.log(message);

return {

message: "Error",

};

}

} else {

// Якщо юзер не в аккаунті

return {

message: "Error",

};

}

}

function getChoosedCalendarEvents() {

const choosedCalendarEvents = [];

cachedEvents.forEach((event) => {

if (selectedCalendars.includes(event.calendarId)) {

choosedCalendarEvents.push(event);

}

});

// console.log(choosedCalendarEvents);

return choosedCalendarEvents;

}

function eventsForDaySort(events, year, month, day) {

const segments = events.flat();

// console.log(segments);

result\_list = calendarSort(segments, year, month, day);

result\_list.sort(

(

{ startMinutes: aStart, endMinutes: aEnd },

{ startMinutes: bStart, endMinutes: bEnd }

) => {

if (aStart !== bStart) return aStart - bStart;

return bEnd - bStart - (aEnd - aStart); // сортування за тривалістю, якщо час початку однаковий

}

);

return result\_list;

}

function calendarSort(segments, year, month, day) {

const result\_list = segments.filter((segment) => {

return (

segment.start && // перевірка start для впевненості

segment.start.year === year &&

segment.start.month === month &&

segment.start.day === day

);

});

return result\_list;

}

function loadEventsUpdate(res\_data) {

console.log("loading Events");

const syncTypes = res\_data.syncType;

const data = res\_data.result;

if (!data || data.message === "Error") {

console.error("Отримано пусті або ж некоректні події: ", data);

return;

}

let updatedCachedEvents = [...cachedEvents];

try {

// Групуємо вхідні сегменти за event\_id

const incomingEventsGrouped = data.reduce((acc, segment) => {

const calendarId = segment.calendar\_id;

const eventId = segment.event\_id;

acc[eventId] = acc[eventId] || [];

acc[eventId].push(segment);

return acc;

}, {});

// Спочатку, якщо якась з подій має повну синхронізацію, видаляємо її старі екземпляри

const yearStart = cachedDate.getFullYear();

const yearEnd = yearStart + 1;

for (const calendarId in syncTypes) {

if (syncTypes[calendarId] === "Full") {

updatedCachedEvents = updatedCachedEvents.filter((event) => {

const start = new Date(event.eventStart);

const end = new Date(event.eventEnd);

return (

event.calendar\_id !== calendarId ||

end.getFullYear() <= yearStart ||

start.getFullYear() >= yearEnd

);

});

}

}

// Обробляємо кожну унікальну подію з вхідних даних

for (const eventId in incomingEventsGrouped) {

const newSegments = incomingEventsGrouped[eventId];

const status = newSegments[0]?.status;

const calendarId = newSegments[0]?.calendar\_id;

// Видаляємо старі версії цієї події

updatedCachedEvents = updatedCachedEvents.filter(

(cachedEv) => cachedEv.event\_id !== eventId

);

if (status !== "cancelled") {

updatedCachedEvents.push(...newSegments);

}

}

cachedEvents = updatedCachedEvents;

localStorage.setItem("cachedEvents", JSON.stringify(cachedEvents));

} catch (error) {

showToast("!: " + "Couldnt load Events Update");

logAction(`loadEventsUpdate: Error: ${error.message};`);

}

}

// async function pingUser() {

// Фактично не потрібен, бо я і так пінгую сервер 😒😒😒😒😒😒😒😒😒

// try {

// const response = await fetch("/ping", { method: "POST" });

// await handleErrors(response);

// await loadEvents();

// showView(cachedView);

// } catch (error) {

// console.error("!: " + error.message);

// }

// }

function loadEventsSetInterval() {

setInterval(async () => {

await loadEvents();

}, 300000); // 5 хвил

}

// Фактично не потрібен, бо я і так пінгую сервер 😒😒😒😒😒😒😒😒😒

// async function checkUserWebhook() {

// try {

// const response = await fetch("/check\_webhook\_expiration");

// // await handleErrors(response);

// } catch (error) {

// console.error("!: " + error.message);

// }

// }

// function checkUserWebhookSetInterval() {

// if (webhookIntervalId) {

// clearInterval(webhookIntervalId);

// }

// webhookIntervalId = setInterval(async () => {

// await checkUserWebhook();

// }, 10000);

// }

// Фактично не потрібен, бо я і так пінгую сервер 😒😒😒😒😒😒😒😒😒

async function logAction(message) {

try {

const response = await fetch("/log", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ message }),

});

await handleErrors(response);

} catch (error) {

console.error("!: " + error.message);

}

}

**events\_view\_utils.js**

function toggleMenu() {

const submenu = document.getElementById("submenu\_container");

submenu.classList.toggle("open");

}

function changeView(event) {

const viewId = event.target.value;

showView(viewId);

}

function showView(viewId) {

if (viewId === "today") {

changeDateView(viewId);

viewId = cachedView;

}

cachedView = viewId;

localStorage.setItem("cachedView", JSON.stringify(viewId));

updateShowCurrentDate();

const viewSelect = document.getElementById("viewSelect");

viewSelect.value = viewId;

const views = document.querySelectorAll(".calendar-view");

views.forEach((v) => v.classList.add("hidden"));

document.getElementById(viewId + "-view").classList.remove("hidden");

if (viewId === "year") {

renderYear();

} else if (viewId === "month") {

renderMonth();

} else if (viewId === "week") {

generateWeek();

} else if (viewId === "day") {

generateDay();

} else {

let message = `showView: viewId Invalid Value; viewId: ${viewId};`;

logAction(message);

}

}

function toggleProfileMenu() {

const profile\_logout = document.getElementById("profile\_menu");

profile\_logout.classList.toggle("hidden");

const profile\_pic = document.getElementById("profile\_pic");

profile\_pic.classList.toggle("profile\_pic\_blur");

}

function initCachedEvents(user) {

if (user) {

if (localStorage.getItem("cachedEvents") !== null) {

try {

const saved = localStorage.getItem("cachedEvents");

cachedEvents = JSON.parse(saved);

} catch (e) {

let message = `initCachedEvents: cachedEvents Invalid Value; Error: ${e};`;

logAction(message);

}

} else {

cachedEvents = [];

localStorage.setItem("cachedEvents", JSON.stringify(cachedEvents));

}

} else {

cachedEvents = [];

localStorage.removeItem("cachedEvents");

}

}

function initCachedHolidays() {

if (localStorage.getItem("cachedHolidays") !== null) {

try {

const saved = localStorage.getItem("cachedHolidays");

cachedHolidays = JSON.parse(saved);

} catch (e) {

let message = `initCachedHolidays: cachedHolidays Invalid Value; Error: ${e};`;

logAction(message);

}

} else {

cachedHolidays = [];

localStorage.setItem("cachedHolidays", JSON.stringify(cachedHolidays));

}

}

function initCachedCalendars() {

if (localStorage.getItem("cachedCalendars") !== null || !user) {

try {

const saved = localStorage.getItem("cachedCalendars");

cachedCalendars = JSON.parse(saved);

} catch (e) {

let message = `initCachedCalendars: cachedCalendars Invalid Value; Error: ${e};`;

logAction(message);

}

} else {

cachedCalendars = [];

localStorage.setItem("cachedCalendars", JSON.stringify(cachedCalendars));

}

}

function initCachedView() {

if (localStorage.getItem("cachedView") !== null) {

try {

const saved = localStorage.getItem("cachedView");

cachedView = JSON.parse(saved);

const viewSelect = document.getElementById("viewSelect");

viewSelect.value = cachedView;

} catch (e) {

let message = `initCachedView: cachedView Invalid Value; Error: ${e};`;

logAction(message);

}

} else {

localStorage.setItem("cachedView", JSON.stringify("day"));

}

showView(cachedView);

}

function initCachedDate() {

if (localStorage.getItem("cachedDate") !== null) {

try {

const saved = localStorage.getItem("cachedDate");

cachedDate = new Date(JSON.parse(saved));

} catch (e) {

let message = `initCachedDate: cachedDate Invalid Value; Error: ${e};`;

logAction(message);

}

} else {

cachedDate = new Date();

localStorage.setItem("cachedDate", JSON.stringify(cachedDate));

}

showView(cachedView);

}

async function changeDateView(changeType) {

tempDate = new Date(cachedDate);

if (changeType === "prev") {

if (cachedView === "day") {

cachedDate.setDate(cachedDate.getDate() - 1);

} else if (cachedView === "week") {

cachedDate.setDate(cachedDate.getDate() - 7);

} else if (cachedView === "month") {

cachedDate.setMonth(cachedDate.getMonth() - 1);

} else if (cachedView === "year") {

cachedDate.setFullYear(cachedDate.getFullYear() - 1);

}

} else if (changeType === "next") {

if (cachedView === "day") {

cachedDate.setDate(cachedDate.getDate() + 1);

} else if (cachedView === "week") {

cachedDate.setDate(cachedDate.getDate() + 7);

} else if (cachedView === "month") {

cachedDate.setMonth(cachedDate.getMonth() + 1);

} else if (cachedView === "year") {

cachedDate.setFullYear(cachedDate.getFullYear() + 1);

}

} else if (changeType === "today") {

cachedDate = new Date();

}

localStorage.setItem("cachedDate", JSON.stringify(cachedDate));

showView(cachedView);

if (cachedDate.getFullYear() != tempDate.getFullYear()) {

await loadEvents();

}

const prevYear = new Date(cachedDate);

prevYear.setDate(cachedDate.getDate() - 7);

const nextYear = new Date(cachedDate);

nextYear.setDate(cachedDate.getDate() + 7);

// Оскільки відображення подій відбувається в тижні(7днів), то в кінці року, можуть не підвантажитись події, бо частина року буде минула

if (prevYear.getFullYear() != cachedDate.getFullYear()) {

await loadEvents(prevYear.getFullYear());

} else if (nextYear.getFullYear() != cachedDate.getFullYear()) {

await loadEvents(nextYear.getFullYear());

}

}

function updateShowCurrentDate() {

const updateYear = document.getElementById("showCurrentDateYear");

updateYear.textContent = cachedDate.getFullYear();

const updateMonth = document.getElementById("showCurrentDateMonth");

const currentMonth = cachedDate.getMonth();

updateMonth.textContent = monthNames[currentMonth];

}

**create\_day.js**

function generateDayHours(viewId, year, month, day, grid, container, num = 0) {

const Hours = 24;

const events = getCalendarDayEvents(year, month + 1, day);

const slot\_time = document.createElement("div");

slot\_time.classList.add("hour-slot");

slot\_time.setAttribute("role", "button");

slot\_time.setAttribute("tabindex", "0");

const saveMonth = String(month + 1).padStart(2, "0");

const saveDay = String(day).padStart(2, "0");

slot\_time.dateData = `${year}-${saveMonth}-${saveDay}`;

slot\_time.addEventListener("click", outerClick);

if (events.message != "Error" && events.length > 0) {

const events\_container = document.createElement("div");

events\_container.classList.add("events-container");

events.forEach((event, i) => {

let column = 0;

while (

events.some(

(e, j) =>

j < i &&

e.column === column &&

event.startMinutes < e.endMinutes &&

event.endMinutes > e.startMinutes

)

) {

column++;

}

event.column = column;

const overlaps = events.filter(

(otherEvent, j) =>

j !== i &&

event.startMinutes < otherEvent.endMinutes &&

event.endMinutes > otherEvent.startMinutes

).length;

event.overlaps = overlaps;

});

for (let event of events) {

const slot\_event = document.createElement("div");

slot\_event.classList.add("event-slot");

if (event.longEvent === true) {

// slot\_event.style.backgroundColor = "rgb(255, 255, 255)"; // IDK

}

// 👌👌👌

const colorId = event.colorId;

const bgColor = eventColors[colorId] || "#cccccc";

slot\_event.style.backgroundColor = bgColor;

slot\_event.setAttribute("role", "button");

slot\_event.setAttribute("tabindex", "0");

slot\_event.eventData = event;

slot\_event.addEventListener("click", innerClick);

slot\_event.style.top = `calc(${event.startPosition \* 100}% - 0.06rem)`;

slot\_event.style.height = `calc(${event.endPosition \* 100}% - 0.17rem)`;

const slot\_event\_text = document.createElement("div");

slot\_event\_text.classList.add("slot-event-text");

slot\_event\_text.textContent = event.summary;

slot\_event.appendChild(slot\_event\_text);

if (event.overlaps > 0) {

const totalSlots = event.overlaps + 1;

const slotWidth = (100 - (totalSlots - 1)) / totalSlots;

slot\_event.style.left = `${event.column \* slotWidth}%`;

const temp = `${event.column \* slotWidth}%`;

slot\_event.style.width = `calc(${100}% - ${temp})`;

slot\_event.style.zIndex = event.column + 10; // Фонові лінії +10

} else {

slot\_event.style.width = "100%";

}

events\_container.appendChild(slot\_event);

}

slot\_time.appendChild(events\_container);

} else if (events.length === 0) {

console.log("No Events to Display");

}

if ((viewId === "week" && num == 0) || viewId === "day") {

const linesContainer = document.createElement("div");

linesContainer.classList.add("lines-container");

linesContainer.style.pointerEvents = "none";

const lineTextContainer = document.createElement("div");

lineTextContainer.classList.add("line-text-container");

for (let hour = 0; hour <= Hours; hour++) {

const formattedHour = hour.toString().padStart(2, "0") + ":00";

if (hour > 0 && hour < Hours) {

const hLine = document.createElement("div");

hLine.classList.add("horizontal-line");

hLine.style.top = `calc(${(hour / Hours) \* 100}%)`; // + 2rem

const lineText = document.createElement("div");

lineText.classList.add("horizontal-time");

lineText.style.top = `calc(${(hour / Hours) \* 100}%)`; // + 2rem

lineText.textContent = formattedHour;

lineTextContainer.appendChild(lineText);

linesContainer.appendChild(hLine);

}

}

if (viewId === "week") {

for (let i = 1; i < 7; i++) {

// 6 вертикальних ліній між 7 днями

const vLine = document.createElement("div");

vLine.classList.add("vertical-line");

vLine.style.left = `${(i / 7) \* 100}%`;

linesContainer.appendChild(vLine);

}

}

container.appendChild(linesContainer);

grid.appendChild(lineTextContainer);

}

container.appendChild(slot\_time);

grid.appendChild(container);

generateHolidays(year, month, day, num);

}

function generateHolidays(year, month, day, num) {

const holiday = calendarSort(cachedHolidays, year, month + 1, day);

if (holiday.length === 0) {

return;

}

const dayInfoContainer = document.getElementById(`day-info-container-${num}`);

const holidayEventContainer = document.createElement("div");

holidayEventContainer.classList.add("holiday-event-container");

holidayEventContainer.eventData = holiday[0];

holidayEventContainer.addEventListener("click", innerClick);

const holidayEventText = document.createElement("div");

holidayEventText.classList.add("holiday-event-text");

holidayEventText.textContent = holiday[0].summary;

holidayEventContainer.appendChild(holidayEventText);

dayInfoContainer.appendChild(holidayEventContainer);

}

const eventColors = {

1: "#a4bdfc",

2: "#7ae7bf",

3: "#dbadff",

4: "#ff887c",

5: "#fbd75b",

6: "#ffb878",

7: "#46d6db",

8: "#e1e1e1",

9: "#5484ed",

10: "#51b749",

11: "#dc2127",

};

**events\_day.js**

function generateDay() {

year = cachedDate.getFullYear();

month = cachedDate.getMonth();

day = cachedDate.getDate();

const today = new Date();

const dayGrid = document.getElementById("day-grid");

dayGrid.innerHTML = "";

const day\_container = document.createElement("div");

day\_container.classList.add("day-container");

const dayInfoContainer = document.createElement("div");

dayInfoContainer.classList.add("day-info-container");

// Схоже, що не встигало чи просто не очищувалось, тому призначив інший елемент

dayInfoContainer.setAttribute("id", `day-info-container-${-1}`);

const dayGridDateInfo = document.getElementById("dayGridDateInfo");

dayGridDateInfo.innerHTML = "";

const date = new Date(year, month, day);

const dayOfWeek = date.getDay();

const dayOfWeekInfo = document.createElement("div");

dayOfWeekInfo.classList.add("day-of-week-info");

dayOfWeekInfo.textContent = `${daysOfWeek[dayOfWeek]}`;

const dayNumInfo = document.createElement("div");

dayNumInfo.classList.add("day-num-info");

dayNumInfo.textContent = `${day}`;

changeCurrentDateColorDay(today, year, month, day, dayNumInfo);

dayInfoContainer.appendChild(dayOfWeekInfo);

dayInfoContainer.appendChild(dayNumInfo);

dayGridDateInfo.appendChild(dayInfoContainer);

generateDayHours("day", year, month, day, dayGrid, day\_container, -1);

}

**events\_week.js**

function generateWeek() {

const year = cachedDate.getFullYear();

const month = cachedDate.getMonth();

const day = cachedDate.getDate();

const today = new Date();

const weekGrid = document.getElementById("week-grid");

weekGrid.innerHTML = "";

const week\_container = document.createElement("div");

week\_container.classList.add("week-container");

const weekGridDateInfo = document.getElementById("weekGridDateInfo");

weekGridDateInfo.innerHTML = "";

const startDate = new Date(year, month, day);

const dayOfWeek = startDate.getDay();

const sundayDate = new Date(startDate);

// По суті віднімаємо день тижня (1-понеділок..) від дати, щоб отримати неділю

sundayDate.setDate(startDate.getDate() - dayOfWeek);

for (let i = 0; i < 7; i++) {

const currentDate = new Date(sundayDate);

currentDate.setDate(sundayDate.getDate() + i);

const currentDay = currentDate.getDate();

const dayInfoContainer = document.createElement("div");

dayInfoContainer.classList.add("day-info-container");

dayInfoContainer.setAttribute("id", `day-info-container-${i}`);

if (i != 6) {

dayInfoContainer.style.borderRight = "1px solid lightgrey"; // white

}

if (i === 0) {

dayInfoContainer.style.borderLeft = "1px solid transparent"; // просто зміщення, без відображення

}

const currDayOfWeek = currentDate.getDay();

const dayOfWeekInfo = document.createElement("div");

dayOfWeekInfo.classList.add("day-of-week-info");

dayOfWeekInfo.textContent = `${daysOfWeek[currDayOfWeek]}`;

const dayNumInfo = document.createElement("div");

dayNumInfo.classList.add("day-num-info");

dayNumInfo.textContent = `${currentDay}`;

changeCurrentDateColorDay(

today,

year,

month,

currentDate.getDate(),

dayNumInfo

);

dayInfoContainer.appendChild(dayOfWeekInfo);

dayInfoContainer.appendChild(dayNumInfo);

weekGridDateInfo.appendChild(dayInfoContainer);

generateDayHours(

"week",

currentDate.getFullYear(),

currentDate.getMonth(),

currentDate.getDate(),

weekGrid,

week\_container,

i

);

}

}

function changeCurrentDateColorDay(today, year, month, day, el) {

if (

year === today.getFullYear() &&

month === today.getMonth() &&

day === today.getDate()

) {

el.style.color = "cornflowerblue";

}

}

**create\_month.js**

function createMonth(

year,

month, // 0-індекс

dateView,

yearContainer = null,

dayClass = "month-day"

) {

const monthGridInfoContainer = document.createElement("div");

monthGridInfoContainer.setAttribute("id", "monthGridInfoContainer");

const monthDayEl = document.createElement("div");

monthDayEl.classList.add("month-day-grid");

if (yearContainer) {

monthGridInfoContainer.classList.add("year-grid-info-container");

// monthDayEl.style.marginBottom = "0";

yearContainer.appendChild(monthGridInfoContainer);

yearContainer.appendChild(monthDayEl);

} else {

const container = document.createElement("div");

container.classList.add("view-container");

monthGridInfoContainer.classList.add("month-grid-info-container");

container.appendChild(monthGridInfoContainer);

container.appendChild(monthDayEl);

dateView.appendChild(container);

}

//

const daysInWeek = 7;

const maxRowsDays = 6;

// Очистити старі елементи

monthDayEl.innerHTML = "";

const daysInMonth = new Date(year, month + 1, 0).getDate();

const firstDay = new Date(year, month, 1).getDay();

const startOffset = firstDay;

const prevMonthLastDay = new Date(year, month, 0).getDate();

const today = new Date();

// Дні з попереднього місяця

for (let i = startOffset - 1; i >= 0; i--) {

const day = prevMonthLastDay - i;

const prevEl = document.createElement("button");

prevEl.dataset.month = month - 1;

if (month - 1 < 0) {

prevEl.dataset.month = 12;

prevEl.dataset.year = year - 1;

}

prevEl.dataset.day = i;

monthButtonListener(prevEl);

prevEl.classList.add(dayClass);

prevEl.textContent = day;

prevEl.style.borderRight = "1px solid lightgrey";

prevEl.style.borderBottom = "1px solid lightgrey";

changeCurrentDateColorMonth(today, year, month, day, prevEl);

monthDayEl.appendChild(prevEl);

}

const lastRow = daysInWeek \* maxRowsDays - maxRowsDays;

// Поточний місяць

for (let i = 1; i <= daysInMonth; i++) {

const dayEl = document.createElement("button"); // button

dayEl.dataset.month = month;

dayEl.dataset.day = i;

monthButtonListener(dayEl);

dayEl.classList.add(dayClass);

dayEl.textContent = i;

const dayIndex = (startOffset + i - 1) % daysInWeek;

if (dayIndex !== 6) {

dayEl.style.borderRight = "1px solid lightgrey";

}

const currentLastRow = startOffset + i;

if (currentLastRow < lastRow) {

dayEl.style.borderBottom = "1px solid lightgrey";

}

changeCurrentDateColorMonth(today, year, month, i, dayEl);

monthDayEl.appendChild(dayEl);

}

// Дні з наступного місяця

const totalCells = daysInWeek \* maxRowsDays;

const currentCount = startOffset + daysInMonth;

const nextDays = totalCells - currentCount;

for (let i = 1; i <= nextDays; i++) {

const nextEl = document.createElement("button");

nextEl.dataset.month = month + 1;

nextEl.dataset.day = i;

if (month + 1 === 12) {

nextEl.dataset.month = 0;

nextEl.dataset.year = year + 1;

}

monthButtonListener(nextEl);

nextEl.classList.add(dayClass);

nextEl.textContent = i;

// Не ставити бордер для останнього елемента в ряді

if ((i + currentCount - 1) % daysInWeek !== 6) {

nextEl.style.borderRight = "1px solid lightgrey";

}

const nextLastRow = startOffset + daysInMonth + i;

if (nextLastRow < lastRow) {

nextEl.style.borderBottom = "1px solid lightgrey";

}

changeCurrentDateColorMonth(today, year, month, i, nextEl);

monthDayEl.appendChild(nextEl);

}

let gridInfoContainer = monthGridInfoContainer;

gridInfoContainer.innerHTML = "";

const gridDateInfo = document.createElement("div");

gridDateInfo.classList.add("grid-date-info");

for (let i = 0; i < 7; i++) {

const dayInfoMonthContainer = document.createElement("div");

dayInfoMonthContainer.classList.add("day-info-month-container");

if (i != 6) {

dayInfoMonthContainer.style.borderRight = "1px solid lightgrey"; // white

}

if (i === 6) {

dayInfoMonthContainer.style.borderRight = "1px solid transparent"; // зміщуємо вліво на 1 піксель

}

const dayOfWeekInfo = document.createElement("div");

dayOfWeekInfo.classList.add("day-of-week-info");

dayOfWeekInfo.textContent = daysOfWeek[i];

dayInfoMonthContainer.appendChild(dayOfWeekInfo);

gridDateInfo.appendChild(dayInfoMonthContainer);

}

gridInfoContainer.appendChild(gridDateInfo);

}

function monthButtonListener(el) {

el.addEventListener("click", () => {

const clickedMonth = el.dataset.month;

const clickedDay = el.dataset.day;

const clickedYear = el.dataset.year;

if (clickedYear) {

cachedDate.setYear(clickedYear);

}

cachedDate.setMonth(clickedMonth);

cachedDate.setDate(clickedDay);

localStorage.setItem("cachedDate", JSON.stringify(cachedDate));

showView("week");

// console.log(clickedDay);

});

}

function changeCurrentDateColorMonth(today, year, month, day, el) {

if (

year === today.getFullYear() &&

month === today.getMonth() &&

day === today.getDate()

) {

el.style.backgroundColor = "cornflowerblue";

}

}

**events\_month.js**

function renderMonth() {

const year = cachedDate.getFullYear();

const month = cachedDate.getMonth();

// const monthGrid = document.getElementById("month-grid");

const monthView = document.getElementById("month-view");

monthView.innerHTML = "";

createMonth(year, month, monthView);

}

**events\_year.js**

function renderYear() {

const year = cachedDate.getFullYear();

const yearView = document.getElementById("year-view");

yearView.innerHTML = "";

const yearGrid = document.createElement("div");

yearGrid.setAttribute("id", "yearGrid");

yearGrid.classList.add("year-grid");

yearView.appendChild(yearGrid);

for (let m = 0; m < 12; m++) {

const container = document.createElement("div");

container.classList.add("view-container");

const title = document.createElement("h3");

title.textContent = monthNames[m];

container.appendChild(title);

createMonth(year, m, yearView, container);

// container.appendChild(monthGrid);

yearGrid.appendChild(container);

}

}

**logs.js**

function searchLogs() {

const query = document

.getElementById("searchInput")

.value.toLowerCase()

.trim();

const keywords = query.split(/\s+/); // розділяє по пробілах

const logs = document.querySelectorAll(".log-list");

let shownCount = 0;

logs.forEach((log) => {

const date = log.querySelector("td:nth-child(1)").textContent.toLowerCase();

const email = log

.querySelector("td:nth-child(2)")

.textContent.toLowerCase();

const message = log

.querySelector("td:nth-child(3)")

.textContent.toLowerCase();

const combined = `${date} ${email} ${message}`;

const matchesAll = keywords.every((word) => combined.includes(word));

if (matchesAll && shownCount < 100) {

log.style.display = "";

shownCount++;

} else {

log.style.display = "none";

}

});

}

**submenu\_container.js**

async function populateCalendarsSubmenu() {

// Можна в ДОМ кинуть

const submenu = document.getElementById("calendarsSubmenu");

submenu.innerHTML = "";

const calendarsUserInfo = document.createElement("div");

if (user) calendarsUserInfo.textContent = "Календарі:";

else calendarsUserInfo.textContent = "Авторизуйтесь";

calendarsUserInfo.classList.add("calendars-user-info");

submenu.appendChild(calendarsUserInfo);

if (user) {

const allCalendarsIds = cachedCalendars.map((el) => el.id);

const savedSelected =

JSON.parse(localStorage.getItem("selectedCalendars")) || allCalendarsIds;

selectedCalendars = [...savedSelected];

cachedCalendars.forEach((calendar) => {

const calendarInfoContainer = document.createElement("div");

calendarInfoContainer.classList.add("calendar-info-container");

const checkbox = document.createElement("div");

checkbox.classList.add("calendar-checkbox-svg");

checkbox.dataset.id = calendar.id;

checkbox.dataset.color = calendar.colorId;

if (savedSelected.includes(calendar.id)) {

checkbox.classList.add("checked");

}

/////////

if (checkbox.classList.contains("checked")) {

checkbox.style.backgroundColor =

eventColors[calendar.colorId] || "#cccccc";

} else {

checkbox.style.backgroundColor = "transparent";

}

/////////

checkbox.addEventListener("click", () => {

checkbox.classList.toggle("checked");

/////////

if (checkbox.classList.contains("checked")) {

checkbox.style.backgroundColor =

eventColors[calendar.colorId] || "#cccccc";

} else {

checkbox.style.backgroundColor = "transparent";

}

/////////

const id = checkbox.dataset.id;

if (checkbox.classList.contains("checked")) {

if (!selectedCalendars.includes(id)) selectedCalendars.push(id);

} else {

selectedCalendars = selectedCalendars.filter((el) => el !== id);

}

// 💾 Зберігаємо

localStorage.setItem(

"selectedCalendars",

JSON.stringify(selectedCalendars)

);

console.log("✅ selectedCalendars:", selectedCalendars);

showView(cachedView);

});

const calendarSummary = document.createElement("div");

calendarSummary.classList.add("calendar-info-summary");

calendarSummary.textContent = calendar.summary.includes("@gmail.com")

? "Main"

: calendar.summary;

calendarInfoContainer.appendChild(checkbox);

calendarInfoContainer.appendChild(calendarSummary);

submenu.appendChild(calendarInfoContainer);

});

} else {

localStorage.removeItem("selectedCalendars");

}

}

**toast.js**

function showToast(message, duration = 3000) {

const toast = document.getElementById("toast");

toast.textContent = message;

toast.classList.remove("hidden");

toast.classList.add("toast-show");

// Сховати через duration

setTimeout(() => {

toast.classList.remove("toast-show");

setTimeout(() => toast.classList.add("hidden"), 300); // дочекатись завершення анімації

}, duration);

}

**view\_events\_menu.js**

function showEventDetails(event) {

if (!user && !event.calendarId.includes("holiday")) {

showToast("Авторизуйтесь");

return;

}

// Коли перемикаємось між подіями, скидуємо до звичайного вигляду

returnToStartView();

// console.log("event", event);

const { summary, description, eventStart, eventEnd } = event;

const setContent = (id, text) => {

const elem = document.getElementById(id);

if (!elem) return;

if (elem.tagName === "INPUT") {

elem.value = text;

} else {

elem.innerText = text;

}

};

if (event.calendarId.includes("holiday") || event === placeholderEvent) {

document.getElementById("editEvent").classList.add("hidden");

document.getElementById("deleteEvent").classList.add("hidden");

document.getElementById("viewMenuChooseCalendar").disabled = false;

populateViewMenuChooseCalendar(placeholderEvent.calendarId);

} else {

document.getElementById("editEvent").classList.remove("hidden");

document.getElementById("deleteEvent").classList.remove("hidden");

document.getElementById("viewMenuChooseCalendar").disabled = true;

populateViewMenuChooseCalendar("");

}

if (event.calendarId.includes("holiday")) {

document.getElementById("viewMenuChooseCalendar").classList.add("hidden");

document.getElementById("saveEvent").classList.add("hidden");

} else {

document

.getElementById("viewMenuChooseCalendar")

.classList.remove("hidden");

document.getElementById("saveEvent").classList.remove("hidden");

}

setContent("eventSummary", summary || "Без назви");

setContent("eventDescription", description || "Без опису");

if (!event.calendarId.includes("holiday")) {

document.getElementById("timeStart").classList.remove("hidden");

document.getElementById("timeEnd").classList.remove("hidden");

setContent("eventStart", formatTime(event, eventStart));

setContent("eventEnd", formatTime(event, eventEnd));

} else {

document.getElementById("timeStart").classList.add("hidden");

document.getElementById("timeEnd").classList.add("hidden");

}

document.getElementById("eventsViewEvent").classList.remove("hidden");

}

function formatTime(event, isoString) {

const date = new Date(isoString);

const day = String(date.getDate()).padStart(2, "0");

const month = String(date.getMonth() + 1).padStart(2, "0");

const year = date.getFullYear();

const hours = String(date.getHours()).padStart(2, "0");

const minutes = String(date.getMinutes()).padStart(2, "0");

return `${day}.${month}.${year} ${hours}:${minutes}`;

}

function closeEventMenu() {

const container = document.getElementById("eventsViewEvent");

returnToStartView();

container.classList.add("hidden");

}

function editEventsMenu(type = "update") {

const container = document.getElementById("eventsViewEvent");

const saveButton = document.getElementById("saveEvent");

if (type === "update") saveButton.onclick = () => updateEventController();

if (type === "create") saveButton.onclick = () => createEventController();

if (container.querySelector("input") && type === "create") {

return;

}

if (container.querySelector("input")) {

returnToStartView();

return;

}

const editableIds = [

"eventSummary",

"eventDescription",

"eventStart",

"eventEnd",

];

editableIds.forEach((id) => {

const elem = document.getElementById(id);

if (!elem) return;

const input = document.createElement("input");

input.type = "text";

input.value = elem.innerText;

input.id = elem.id;

input.className = elem.className;

input.dataset.originalTag = elem.tagName.toLowerCase();

elem.replaceWith(input);

});

saveButton.classList.remove("inactive");

saveButton.classList.add("active");

}

function returnToStartView() {

const container = document.getElementById("eventsViewEvent");

// Знайти всі input, які ми створили замість div

const inputs = container.querySelectorAll("input[data-original-tag]");

const saveButton = document.getElementById("saveEvent");

saveButton.classList.remove("active");

saveButton.classList.add("inactive");

inputs.forEach((input) => {

const originalTag = input.dataset.originalTag;

const newElem = document.createElement(originalTag);

newElem.id = input.id;

newElem.innerText = input.value;

newElem.className = input.className;

input.replaceWith(newElem);

});

}

// доробити

function deleteEventController() {

closeEventMenu();

deleteEvent(currentEvent.calendarId, currentEvent.event\_id);

}

function updateEventController() {

const eventStart = document.getElementById("eventStart").value;

const eventEnd = document.getElementById("eventEnd").value;

const eventSummary = document.getElementById("eventSummary").value;

const eventDescription = document.getElementById("eventDescription").value;

let eventStartIso, eventEndIso;

try {

eventStartIso = toISO(eventStart);

eventEndIso = toISO(eventEnd);

} catch (e) {

showToast(e.message);

return;

}

closeEventMenu();

updateEvent(

currentEvent.calendarId,

currentEvent.colorId,

currentEvent.event\_id,

{

summary: eventSummary,

description: eventDescription,

start: eventStartIso,

end: eventEndIso,

}

);

}

function toISO(str) {

const regex = /^(\d{2})\.(\d{2})\.(\d{4}) (\d{2}):(\d{2})$/;

const match = str.match(regex);

if (!match) {

throw new Error("Введіть дату у форматі: DD.MM.YYYY HH:mm");

}

const [\_, day, month, year, hours, minutes] = match;

const date = new Date(`${year}-${month}-${day}T${hours}:${minutes}:00`);

// Повертаємо у форматі ISO без часової зони

return date.toISOString();

}

function createEventsMenu(outer = false) {

showEventDetails(placeholderEvent);

editEventsMenu("create");

}

function populateViewMenuChooseCalendar(calendarId) {

const viewMenuChooseCalendar = document.getElementById(

"viewMenuChooseCalendar"

);

viewMenuChooseCalendar.innerHTML = "";

const placeholderOption = document.createElement("option");

placeholderOption.disabled = true;

placeholderOption.selected = true;

placeholderOption.hidden = true;

placeholderOption.textContent = calendarId;

viewMenuChooseCalendar.appendChild(placeholderOption);

cachedCalendars.forEach((el) => {

const option = document.createElement("option");

option.value = el.id;

if (el.id.includes("@gmail.com")) {

option.textContent = "Main";

} else {

option.textContent = el.summary;

}

option.setAttribute("colorId", el.colorId);

viewMenuChooseCalendar.appendChild(option);

});

}

function createEventController() {

const eventStart = document.getElementById("eventStart").value;

const eventEnd = document.getElementById("eventEnd").value;

const eventSummary = document.getElementById("eventSummary").value;

const eventDescription = document.getElementById("eventDescription").value;

const select = document.getElementById("viewMenuChooseCalendar");

const calendarId = select.value;

if (calendarId === placeholderEvent.calendarId) {

showToast("Оберіть Календар");

return;

}

const selectedOption = select.options[select.selectedIndex];

const colorId = selectedOption.getAttribute("colorId");

let eventStartIso, eventEndIso;

try {

eventStartIso = toISO(eventStart);

eventEndIso = toISO(eventEnd);

} catch (e) {

showToast(e.message);

return;

}

closeEventMenu();

createEvent(calendarId, colorId, {

summary: eventSummary,

description: eventDescription,

start: eventStartIso,

end: eventEndIso,

});

}

CSS

**calendar\_view.css**

.calendar {

height: 100%;

/\* overflow: auto; \*/

flex: 1;

display: flex;

flex-direction: column;

/\* 🛑2🛑 \*/

transition: margin-left 0.3s ease;

}

.view-toggle {

position: relative;

margin-right: var(--top-buttons-margin-sides);

height: 100%;

}

.view-toggle select {

height: 100%;

padding-right: 1rem;

padding-left: 0.5rem;

background-color: rgb(255, 243, 149);

border: 0.25rem solid #ffffff;

border-radius: 0.5rem;

font-size: 1rem;

font-weight: 500;

cursor: pointer;

transition: background-color 0.05s ease;

}

.view-toggle select:hover {

background-color: grey;

color: white;

}

.view-toggle select option {

color: black;

background-color: rgb(255, 243, 149);

}

.calendar-view {

position: relative;

display: flex;

flex-direction: column;

flex: 1;

/\* Займає решту простору \*/

background: white;

padding: 0.5rem 0.5rem;

}

.hidden {

display: none;

}

.grid-info-container {

position: sticky;

top: 0;

/\* Поки що для теста 🛑🛑🛑🛑🛑 \*/

height: 5.5rem;

display: flex;

overflow: hidden;

/\* border-bottom: 2px solid white; \*/

/\* margin-bottom: 0.5rem; \*/

border-top-right-radius: 0.2rem;

border-top-left-radius: 0.2rem;

z-index: 10000;

}

.year-grid-info-container,

.month-grid-info-container {

position: relative;

display: flex;

border-bottom: 2px solid white;

border-radius: 0.2rem;

overflow: hidden;

}

.year-grid-info-container {

height: 1.65rem;

}

.month-grid-info-container {

height: 4rem;

}

.change-view-date-calendar-container {

display: flex;

position: relative;

justify-content: center;

align-items: flex-start;

height: 6rem;

width: calc(2.6rem + 0.5rem + 0.5rem);

background-color: rgb(255, 232, 99);

z-index: 100;

}

.grid-date-info {

position: relative;

display: flex;

flex: 1;

background-color: rgb(255, 243, 149);

}

.arrows-button-container {

position: absolute;

top: 0;

bottom: 0;

left: 0;

right: 0;

display: flex;

align-items: center;

justify-content: center;

margin-left: 0.15rem;

margin-right: 0.15rem;

}

.arrows-button-calendar {

background: none;

border: none;

padding: 0;

border-radius: 50%;

display: flex;

flex: 1;

}

.arrows-button-calendar:hover {

background-color: grey;

color: white;

}

.arrows-calendar {

height: 100%;

width: 100%;

}

**create\_day.css**

.hour-slot {

position: relative;

display: grid;

flex: 1;

/\* 🛑1🛑 \*/

/\* background: #f18f96; \*/

font-weight: bold;

padding: 2rem;

/\* 🛑1🛑 \*/

text-align: left;

align-items: center;

/\* border-radius: 0.5rem; \*/

pointer-events: auto;

}

.events-container {

position: absolute;

top: 0;

bottom: 0;

left: 0.5rem;

right: 0.5rem;

}

.event-slot {

display: flex;

position: absolute;

overflow: hidden;

text-overflow: ellipsis;

border-radius: 0.3rem;

min-height: fit-content !important;

justify-content: left;

align-items: start;

border: white 0.1rem solid;

/\* rgb(132, 178, 244) \*/

background-color: lightgrey;

/\* rgb(212, 229, 243) \*/

z-index: 10;

user-select: none;

}

.slot-event-text {

margin: 0.25rem 0.25rem;

}

**generate\_holidays.css**

.holiday-event-container {

overflow: hidden;

display: flex;

width: 99%;

height: 2rem;

align-items: center;

margin-bottom: 0.25rem;

border-radius: 0.3rem;

background-color: rgb(220, 200, 255);

}

.holiday-event-text {

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

user-select: none;

font-size: 1rem;

font-weight: 500;

margin-left: 0.5rem;

}

**lines\_container.css**

.lines-container {

position: absolute;

left: 0;

right: 0;

top: 0; /\* 🛑1🛑 \*/

bottom: 0;

}

.horizontal-line {

position: absolute;

left: 0;

right: 0;

height: 0;

border-top: 1px solid lightgrey;

color: #000000;

z-index: 1;

}

.vertical-line {

position: absolute;

top: 0;

bottom: 0;

width: 0;

border-left: 1px solid lightgrey;

z-index: 1;

transform: none;

}

.line-text-container {

display: flex;

position: relative;

padding-right: 0.5rem;

padding-left: 0.5rem;

min-width: 2.6rem;

background-color: rgb(255, 232, 99);

z-index: 20;

}

.horizontal-time {

position: absolute;

padding: 0;

user-select: none;

/\* ✅✅✅✅ \*/

transform: translateY(-50%);

}

**day.css**

.day-grid {

flex: 1;

overflow: auto;

position: relative;

display: flex;

border-bottom-right-radius: 0.5rem;

border-bottom-left-radius: 0.5rem;

background-color: rgb(255, 243, 149);

min-height: 125vh;

}

.day-container {

flex: 1;

position: relative;

display: grid;

}

.day-info-container {

display: flex;

flex-direction: column;

overflow: hidden;

flex: 1;

align-items: center;

justify-content: flex-start;

/\* 0.5rem \*/

padding-top: 0.4rem;

height: 100%;

}

.day-of-week-info {

font-size: 1rem;

font-weight: 500;

}

.day-num-info {

font-size: 1.2rem;

font-weight: 600;

margin-top: 0.25rem;

margin-bottom: 0.25rem;

}

.day-info-month-container {

display: flex;

flex-direction: column;

flex: 1;

align-items: center;

justify-content: center;

height: 100%;

}

**week.css**

.week-grid {

flex: 1;

/\* overflow: auto; \*/

position: relative;

display: flex;

background-color: rgb(255, 243, 149);

border-bottom-right-radius: 0.5rem;

border-bottom-left-radius: 0.5rem;

min-height: 125vh;

}

.week-container {

flex: 1;

position: relative;

display: grid;

grid-template-columns: repeat(7, 1fr);

}

**create\_month.css**

.view-container {

display: flex;

flex-direction: column;

min-width: fit-content;

flex: 1;

}

.view-container h3 {

text-align: center;

font-weight: bold;

margin-top: 0;

margin-bottom: 0.9rem;

}

**month.css**

.month-day-grid {

flex: 1;

display: grid;

grid-template-columns: repeat(7, 1fr);

overflow: hidden;

border-radius: 0.2rem;

/\* border: 1px solid lightgrey; \*/

/\* margin-top: 1px; \*/

}

.day-name {

display: flex;

align-items: center;

justify-content: center;

font-weight: bold;

}

**year.css**

.year-grid {

flex: 1;

overflow: auto;

display: grid;

grid-template-columns: repeat(4, 1fr);

row-gap: 0.65rem;

column-gap: 2.9rem;

margin-top: 0.45rem;

}

.month-day {

display: flex;

flex: 1;

font-size: 0.9rem !important;

align-items: center;

justify-content: center;

background: none;

border: none;

font-size: 0.8rem;

font-weight: 500;

background: rgb(255, 243, 149);

/\* border: 1px solid lightgrey; \*/

}

.month-day:hover {

background-color: grey;

color: white;

}

**config.css**

/\* 🆕Main🆕 \*/

@media (max-width: 480px) {

html {

font-size: 8px;

}

.week-grid,

.day-grid {

min-height: 225vh;

}

:root {

--max-height: 6rem;

--top-buttons-width: 4rem;

--top-buttons-margin-sides: 0.3rem;

--padding-container: 0.75rem;

--margin-top-container: 0.2rem;

--submenu-container-width: 20rem;

--menu-hover-width: 11rem;

}

.show-current-date-container {

font-size: calc(var(--max-height) / 4) !important;

}

.view-toggle select {

padding: 0;

font-size: 1.5rem !important;

display: block;

}

/\* .submenu\_container {

position: absolute;

top: 6rem;

left: 0;

} \*/

.submenu\_container {

z-index: 9999999;

}

.calendars-user-info {

font-size: 2.5rem !important;

}

.calendar-info-summary {

font-size: 2rem !important;

}

.calendar-checkbox-svg {

height: 5rem !important;

width: 6.5rem !important;

}

.view-menu-choose-calendar select {

height: 5rem;

font-size: 1.75rem;

}

.event-view-summary {

font-size: 2rem;

}

.event-view-description {

font-size: 1.75rem;

}

.event-view-time-container {

font-size: 1.75rem;

}

.save-button {

font-size: 2rem;

}

.event-view-time {

font-size: 1.5rem;

}

.icon-view {

width: 4rem;

}

.event-view-menu {

padding-top: 7rem;

}

.event-view-menu {

width: 40rem;

z-index: 999999999999;

}

.toast {

font-size: 1.5rem;

}

.calendar-info-container {

gap: 0.75rem !important;

}

.submenu\_container.open {

width: 0 !important;

}

.submenu-color {

z-index: 9999999;

background: rgb(226, 238, 248) !important;

}

.change-view-date-top-container {

width: 8rem;

}

.show-current-date-container {

margin-left: 0;

}

.profile\_menu a {

font-size: 1.2rem;

}

.month-day {

font-size: 1.5rem;

}

.holiday-event-container {

padding: 0.5rem 0;

}

/\* .view-toggle select option {

font-size: 4rem !important;

} \*/

.grid-info-container {

height: 7.5rem;

}

.event-slot {

font-size: 1.2rem;

}

.app-info-container button {

/\* width: 8rem; \*/

font-size: 1.5rem !important;

}

.year-grid-info-container {

height: 2.5rem;

}

.view-container h3 {

margin-top: 0.5rem;

font-size: 2rem;

}

.top\_container {

gap: 0 !important;

}

.year-grid {

grid-template-columns: repeat(1, 1fr);

min-height: 525vh;

}

.horizontal-time {

font-size: 1.5rem;

}

.change-view-date-calendar-container {

width: calc(3.5rem + 0.5rem + 0.5rem) !important;

height: 100%;

}

.holiday-event-text {

font-size: 1.5rem;

}

.line-text-container {

min-width: 3.5rem !important;

display: flex;

align-items: center;

justify-content: center;

}

\*:hover {

transition: none !important;

animation: none !important;

}

.menu\_button:hover {

background: none !important;

}

\* {

transition: none !important;

animation: none !important;

}

}

\* {

scrollbar-width: thin; /\* auto | thin | none 🆕🆕🆕🆕🆕\*/

scrollbar-color: grey transparent;

}

@media (min-width: 1000px) {

html {

font-size: 18px;

}

:root {

--max-height: 6rem;

--top-buttons-height: 8rem;

--top-buttons-width: 4rem;

--top-buttons-margin-sides: 2rem;

--padding-container: 0.75rem;

--margin-top-container: 0.1rem;

--submenu-container-width: 12rem;

--menu-hover-width: 11rem;

}

}

/\* 🛑🛑 \*/

/\* @media (max-width: 600px) {

:root {

--max-height: 5rem;

--top-buttons-height: 4rem;

--top-buttons-width: 2rem;

--top-buttons-margin-sides: 1rem;

--padding-container: 0.375rem;

--margin-top-container: 0.05rem;

--menu-hover-width: 5.5rem;

}

} \*/

body {

display: flex;

flex-direction: column;

/\* grid-template-columns: repeat(2, 1fr); \*/

margin: 0;

padding: 0;

font-family: "Arial", sans-serif;

height: 100vh;

width: 100vw;

/\* 🛑🛑 \*/

}

**create\_event\_button.css**

.event-button {

position: fixed;

bottom: 2rem;

right: 2rem;

height: 4rem;

width: 4rem;

background-color: #a36cf2;

border-radius: 50%;

box-shadow: 0 0.25rem 0.625rem rgba(0, 0, 0, 0.3);

display: flex;

justify-content: center;

align-items: center;

cursor: pointer;

transition: all 0.3s ease;

z-index: 1000;

}

.event-button:hover {

background-color: #8b4de0;

transform: scale(1.1);

box-shadow: 0 0.375rem 0.875rem rgba(0, 0, 0, 0.4);

}

.event-icon {

width: 1.5rem;

height: 1.5rem;

fill: #fbe8c4;

}

**submenu\_container.css**

.bottom-container {

position: relative;

display: flex;

flex: 1;

overflow: auto;

}

.submenu\_container {

position: relative;

visibility: hidden;

overflow: hidden;

width: 0;

height: 100%;

transition: width 0.2s ease, visibility 0s linear 0.3s;

}

.submenu-color {

position: fixed;

height: 100%;

width: var(--submenu-container-width);

background: rgba(212, 229, 243, 0.35);

}

.submenu\_container.open {

position: relative;

width: var(--submenu-container-width);

visibility: visible;

opacity: 1;

display: flex;

flex-direction: row;

transition: width 0.2s ease;

}

.submenu {

display: flex;

flex: 1;

overflow: hidden;

flex-direction: column;

margin: 0.5rem 0.5rem;

transform-origin: left;

}

.calendars-user-info {

font-size: 1.2rem;

font-weight: 600;

margin-bottom: 0.5rem;

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

}

.calendar-info-container {

display: flex;

justify-content: start;

align-items: center;

gap: 0.5rem;

padding: 0.25rem;

}

.calendar-info-summary {

font-size: 1.2rem;

font-weight: 500;

width: 100%;

overflow: hidden;

text-overflow: ellipsis;

white-space: nowrap;

}

.calendar-checkbox-svg {

min-width: 1.5rem;

height: 1.5rem;

border: 0.15rem solid #ccc;

border-radius: 0.3rem;

cursor: pointer;

box-sizing: border-box;

transition: background-color 0.3s ease, border-color 0.3s ease;

}

/\* .calendar-checkbox-svg.checked {

background-color: var(--color-id);

border-color: var(--color-id);

} \*/

/\*

overflow: hidden;

text-overflow: ellipsis;

\*/

**toast.css**

.toast {

position: fixed;

top: 25%;

left: 50%;

transform: translate(-50%, -50%) scale(1);

overflow-wrap: break-word;

background-color: #ffffff;

color: rgb(0, 0, 0);

padding: 1rem 1.5rem;

border-radius: 0.5rem;

font-size: 1rem;

box-shadow: 0 0.25rem 0.75rem rgba(0, 0, 0, 0.2);

z-index: 9999;

opacity: 0;

transition: opacity 0.3s ease;

}

.toast-show {

opacity: 1;

}

.hidden {

display: none;

}

**top\_container.css**

.top\_container {

position: relative;

display: flex;

align-items: center;

justify-content: space-between;

gap: 3.5rem;

height: var(--max-height);

background-color: rgb(212, 229, 243);

}

.menu\_container {

display: flex;

margin-left: var(--top-buttons-margin-sides);

height: 85%;

}

.menu\_button {

position: relative;

padding: 0;

height: 100%;

aspect-ratio: 1;

border: none;

background-color: transparent;

display: flex;

align-items: center;

justify-content: center;

border-radius: 0.5rem;

transition: background-color 0.1s ease;

}

.menu\_button:hover {

background-color: rgba(0, 0, 0, 0.05);

}

.button\_icon {

width: 65%;

height: 65%;

fill: #444;

}

/\* 👌 \*/

.app-info-container {

margin-right: var(--top-buttons-margin-sides);

/\* left: calc(2 \* var(--top-buttons-margin-sides) + var(--top-buttons-width)); \*/

display: flex;

align-items: center;

height: 85%;

padding: 0;

}

.app-info-container button {

position: relative;

margin-right: var(--top-buttons-margin-sides);

display: flex;

align-items: center;

justify-content: center;

height: 100%;

padding-right: 1rem;

padding-left: 1rem;

background-color: rgb(255, 243, 149);

border: 0.25rem solid #ffffff;

border-radius: 0.5rem;

font-size: 1rem;

font-weight: 500;

cursor: pointer;

transition: background-color 0.05s ease;

}

.app-info-container button:hover {

background-color: grey;

color: white;

}

.auth\_container {

position: relative;

height: 100%;

display: flex;

align-items: center;

justify-content: center;

padding: 0;

user-select: none;

}

.profile\_pic {

height: 100%;

transition: box-shadow 0.2s ease;

border-radius: 50%;

cursor: pointer;

}

.profile\_pic:hover {

box-shadow: 0.1rem 0.1rem 0.4rem rgba(0, 0, 0, 0.4);

}

.profile\_pic\_blur {

filter: blur(0.2rem);

}

.profile\_menu {

position: absolute;

align-items: center;

font-size: 0.5rem;

}

.profile\_menu a {

background: white;

color: #000;

padding: 0.5rem;

border-radius: 0.5rem;

}

.hidden {

display: none;

}

/\* .google\_icon {

width: 100%;

height: 100%;

transition: filter 0.2s ease;

}

.google\_icon:hover {

filter: brightness(1.2) invert(1);

} \*/

.change-view-date-top-container {

position: relative;

display: flex;

margin-left: var(--top-buttons-margin-sides);

height: 100%;

aspect-ratio: 1;

}

.show-current-date-container {

position: relative;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

margin-left: var(--top-buttons-margin-sides);

/\* Якщо треба змінити розмір, міняємо і шрифт, можна зробити через var(--max-height), типу поділити на 5 і так далі \*/

font-size: calc(var(--max-height) / 5);

padding: 0 0.35rem;

/\* Якщо треба все ж таки фіксована висота, можна взяти якісь середню \*/

/\* width: 4rem; \*/

border-radius: 0.5rem;

/\* background-color: rgb(255, 243, 149);

border: 0.25rem solid #ffffff; \*/

}

.show-current-date-year {

font-size: 100%;

text-align: center;

font-weight: 600;

}

.show-current-date-month {

font-size: 90%;

text-align: center;

font-weight: 500;

}

**view\_events\_menu.css**

.view-menu-top-buttons-container {

position: absolute;

display: flex;

justify-content: space-between;

align-items: center;

top: 0.75rem;

left: 0.85rem;

right: 0.75rem;

}

.view-menu-choose-calendar {

position: relative;

height: 100%;

}

.view-menu-choose-calendar select {

padding: 0.3rem 1rem;

border-radius: 1rem;

font-size: 0.9rem;

}

.view-menu-choose-calendar select option {

background-color: white;

}

.inactive {

background-color: rgb(229, 229, 229);

user-select: none;

}

.save-button {

margin-top: 1rem;

padding: 0.6rem 1.2rem;

text-align: center;

color: white;

font-weight: bold;

border-radius: 0.5rem;

cursor: pointer;

transition: background-color 0.2s ease;

}

.active {

background-color: #4c8bf5;

}

.active:hover {

background-color: #3a6cd5;

}

.event-view-menu {

position: fixed;

top: 50%;

left: 50%;

transform: translate(-50%, -50%) scale(1);

background: linear-gradient(135deg, #ffffff, #f0f4ff);

/\* background: rgba(251, 249, 245, 0.99); \*/

padding: 1.5rem;

padding-top: 3rem;

border-radius: 1rem;

box-shadow: 0 0.75rem 1.5rem rgba(0, 0, 0, 0.2);

z-index: 1000;

width: 20rem;

animation: fadeIn 0.15s ease-out;

}

.event-view-menu input {

border-radius: 0.2rem;

background-color: #e9f0ff;

border: 1px solid #ccd8f0;

}

.event-view-menu input:hover {

background-color: #dbe7ff;

}

.event-view-menu input:focus {

outline: none;

border: 1px solid rgb(165, 165, 165);

}

.event-view-summary {

overflow-wrap: break-word;

font-size: 1.2rem;

font-weight: bold;

padding: 0.25rem 0.3rem;

background: none;

border: 1px solid transparent;

}

.event-view-description {

overflow-wrap: break-word;

margin-top: 0.35rem;

margin-bottom: 0.2rem;

font-size: 1rem;

padding: 0.25rem 0.5rem;

background: none;

border: 1px solid transparent;

}

.event-view-summary input {

font-size: 1.2rem;

margin-top: 0.5rem;

}

.event-view-description input {

font-size: 1rem;

}

.event-view-time-container {

display: flex;

flex-direction: column;

margin: 0.65rem 0;

font-size: 1rem;

}

.event-view-time {

overflow-wrap: break-word;

font-size: 1rem;

padding: 0.25rem 0.3rem;

background: none;

border: 1px solid transparent;

}

.event-header {

display: flex;

justify-content: center;

flex-direction: column;

}

.event-view-menu h3 {

margin: 0;

font-size: 1.3rem;

color: #333;

}

.event-icons {

position: relative;

height: 100%;

display: flex;

align-items: center;

justify-content: center;

gap: 0.65rem;

}

.icon-view {

width: 1rem;

display: flex;

align-items: center;

justify-content: center;

cursor: pointer;

transition: transform 0.2s ease;

user-select: none;

}

.icon-view:hover {

transform: scale(1.2);

}

.hidden {

display: none;

}

@keyframes fadeIn {

from {

opacity: 0;

transform: translate(-50%, -50%) scale(0.95);

}

to {

opacity: 1;

transform: translate(-50%, -50%) scale(1);

}

}