Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка десктопного планувальника розкладу із використанням технологій C++ QT та Google API

Виконав(ла): студент(ка) 4 курсу групи СП-42
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Подедвірний С.І.
(підпис) (прізвище та ініціали)

Керівник Цуприк Г. Б.
(підпис) (прізвище та ініціали)

Нормоконтроль
(підпис) (прізвище та ініціали)

Завідувач кафедри
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 78, рисунків 15, додатків 4, презентація.

Ключові слова: C++, десктопний додаток, календар, керування часом, планувальник, Qt, розклад.

У роботі розроблено десктопну програму-планувальник розкладу з інтеграцією хмарного сервісу Google Calendar. Програмне забезпечення забезпечує авторизацію через Google-акаунт, перегляд, створення, редагування та синхронізацію подій із календарем користувача.

Об’єктом дослідження є процес розробки інтерактивної десктопної системи для календарного планування.

Метою роботи є створення зручного інструменту для планування подій з можливістю інтеграції з Google Calendar API.

Для реалізації використано мову програмування C++, фреймворк Qt, систему збирання CMake, а також протокол авторизації OAuth 2.0. Було виконано аналіз предметної області, побудовано UML-діаграми, розроблено графічний інтерфейс і реалізовано основний функціонал.

Методи дослідження включають: аналіз функціоналу Google API, моделювання системи, тестування роботи інтерфейсу та логіки програми.

Наукова новизна полягає у створенні повністю автономного клієнтського застосунку, що працює з Google Calendar без проміжного сервера.

Результати можуть бути використані як в особистому, так і в навчальному середовищі. У перспективі передбачено розвиток системи – додавання сповіщень, підтримки кількох користувачів та мобільної версії.

ABSTRACT

Bachelor's qualification work. Ivan Pulyuy Ternopil National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025. 77 pages, 15 figures, 4 appendices, presentation.

Keywords: C++, desktop application, calendar, time management, planner, Qt, schedule.

The work develops a desktop schedule planner program with integration of the Google Calendar cloud service. The software provides authorization via a Google account, viewing, creating, editing and synchronization of events with the user's calendar.

The object of the study is the process of developing an interactive desktop system for calendar planning.

The purpose of the work is to create a convenient tool for event planning with the ability to integrate with the Google Calendar API.

The implementation uses the C++ programming language, the Qt framework, the CMake build system, and the OAuth 2.0 authorization protocol. The subject area was analyzed, UML diagrams were constructed, a graphical interface was developed, and the main functionality was implemented.

The research methods include: analysis of Google API functionality, system modeling, testing of the interface and program logic.

The scientific novelty is the creation of a completely autonomous client application that works with Google Calendar without an intermediate server.

The results can be used both in a personal and educational environment. In the future, the system is planned to be developed - adding notifications, supporting multiple users, and a mobile version.

ЗМІСТ

ВСТУП

У сучасному світі управління часом є важливим аспектом для досягнення особистих і професійних цілей. Зростання кількості завдань та подій вимагає ефективних інструментів для організації та планування. Використання цифрових технологій для управління розкладом стає все більш популярним, оскільки вони дозволяють зберігати, синхронізувати та швидко отримувати доступ до важливої інформації.

Особливий інтерес викликає розробка десктопних планувальників, які забезпечують потужний функціонал у поєднанні з інтеграцією хмарних сервісів, таких як Google Calendar. Це дозволяє користувачам зручно синхронізувати свої події між різними пристроями та отримувати переваги від доступу до актуальних даних у будь-який час і в будь-якому місці. Такий підхід також зменшує залежність від одного пристрою, адже усі події зберігаються в хмарі та доступні навіть у разі втрати локальних даних.

Інтеграція з Google Calendar відкриває нові можливості для створення ефективного інструменту планування, який відповідає сучасним вимогам користувачів. Десктопний додаток, створений із використанням C++ та Qt, пропонує багатий функціонал, інтуїтивний інтерфейс і кросплатформеність, що робить його актуальним для широкої аудиторії. Крім того, використання мови C++ дозволяє досягти високої швидкодії додатка, а Qt забезпечує зручне створення графічного інтерфейсу та підтримку різних операційних систем.

Актуальність цієї роботи зумовлена потребою у вдосконаленні існуючих рішень для планування часу. Створення інтегрованого планувальника розкладу дозволить забезпечити користувачів зручним інструментом для управління своїм розкладом із можливістю роботи як у локальному, так і хмарному режимах. Із розвитком дистанційної роботи, навчання та проєктної діяльності, потреба в надійних засобах організації часу зростає з кожним роком.

Розробка цього програмного забезпечення включає аналіз вимог, проектування архітектури системи, розробку функціоналу та тестування. Особлива увага приділяється інтеграції з Google Calendar API для забезпечення синхронізації подій, автоматизації планування та спрощення взаємодії користувачів із програмою. Така синхронізація також забезпечує збереження історії подій, можливість спільного використання календарів та доступ до даних з будь-якого пристрою.

У рамках даної роботи заплановано реалізацію ключових можливостей планувальника, серед яких створення, редагування та видалення подій, інтеграція з хмарними сервісами, зокрема Google Calendar, для синхронізації даних, а також візуалізація розкладу за допомогою зручного інтерфейсу. Для досягнення цих цілей обрано технології C++ та Qt, що забезпечують високу продуктивність, кросплатформенність і гнучкість у реалізації функціоналу. Використання CMake спрощує процес складання проєкту на різних системах і полегшує його підтримку.

Додаток також матиме розширені можливості, такі як налаштування нагадувань, групування подій за категоріями, пошук у розкладі, а також підтримку кількох облікових записів. Окрему увагу буде приділено розробці користувацького інтерфейсу, орієнтованого на інтуїтивність і зручність, щоб користувачі могли швидко освоїти роботу з додатком. Передбачено адаптивність до роздільної здатності екрана, мінімізацію кількості кліків для базових дій, а також відображення подій у зручному вигляді — за днями, тижнями, місяцями.

Процес розробки починається з аналізу потреб користувачів та визначення технічних вимог. На основі цих даних проектується архітектура додатку, яка включає модулі для роботи з подіями, синхронізації з Google Calendar API, а також забезпечення безпеки даних. Зокрема, будуть впроваджені механізми шифрування для захисту особистої інформації користувачів та безпечного обміну даними між додатком і хмарним сервісом. Реалізація авторизації відбуватиметься через протокол OAuth 2.0 з використанням access token, що відповідає сучасним стандартам безпеки.

Економічна ефективність розробки такого планувальника зумовлена постійним попитом на інструменти для управління часом серед різних категорій користувачів — від студентів до підприємців. Завдяки можливостям інтеграції з популярними сервісами, додаток зможе зайняти свою нішу на ринку, пропонуючи користувачам унікальне поєднання локального і хмарного функціоналу. Оскільки більшість аналогів є вебзастосунками або мають обмежену локальну функціональність, запропоноване рішення вигідно вирізнятиметься на їх фоні.

Таким чином, результати роботи над проектом дозволять створити конкурентоспроможний продукт, що відповідатиме сучасним стандартам розробки програмного забезпечення та задовольнятиме потреби користувачів у плануванні свого часу. Це сприятиме підвищенню ефективності управління особистими і професійними завданнями, забезпечуючи високу надійність і зручність використання додатка.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Аналіз предметної області

У сучасному динамічному світі організація особистого часу, професійних завдань і спільних зустрічей є критично важливою для ефективної діяльності як окремих осіб, так і цілих команд. В умовах постійного зростання кількості обов’язків, дедлайнів та інформаційних потоків, люди все частіше звертаються до інструментів цифрового планування, які дозволяють створювати події, встановлювати нагадування, вести розклади зустрічей, а також швидко переглядати та коригувати графік.

Одним із найбільш популярних рішень у цій сфері є Google Calendar — хмарний календарний сервіс, який дозволяє користувачам створювати події, запрошувати учасників, синхронізувати дані між різними пристроями та отримувати нагадування через електронну пошту або сповіщення. Проте Google Calendar у своєму стандартному вигляді в основному орієнтований на веб- або мобільне використання. Це створює певний дискомфорт для користувачів, які віддають перевагу настільним (десктопним) програмам із локальним управлінням та розширеними функціями кастомізації.

У сфері настільних планувальників існує низка програм, проте багато з них або не підтримують інтеграцію з Google Calendar, або мають застарілий інтерфейс, обмежену кросплатформеність, або закритий вихідний код, що унеможливлює гнучку модифікацію системи під конкретні вимоги.

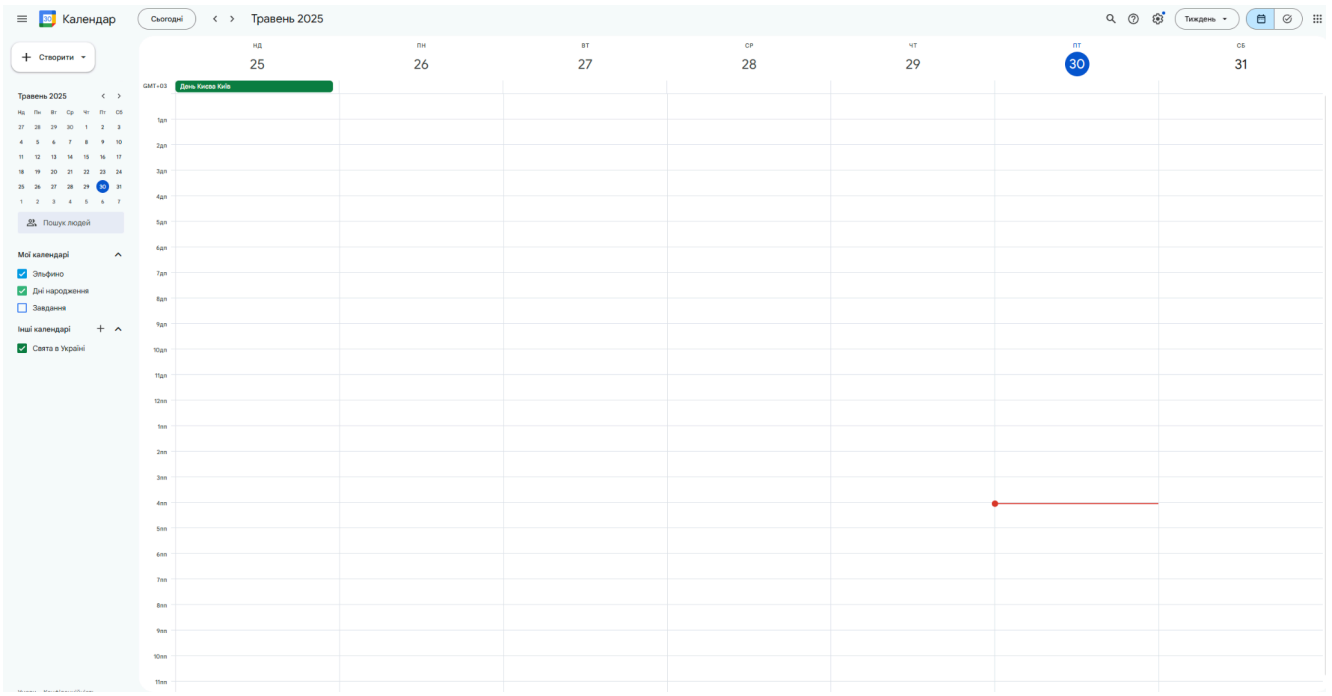


Рис. 1.1 – Зовнішній вигляд Google Calendar

З огляду на це виникає актуальна потреба в розробці десктопного додатку, який поєднує в собі такі характеристики:

1. Підтримка сучасного, інтуїтивно зрозумілого інтерфейсу.
2. Гнучка архітектура, що дозволяє масштабування.
3. Підтримка авторизації та безпечної взаємодії з Google calendar.
4. Можливість повноцінного керування подіями (створення, редагування, видалення).
5. Можливість локального зберігання подій для офлайн-режиму.
6. Кросплатформеність – підтримка Windows, Linux, macOS.
7. Використання відкритих технологій і мов програмування.

Особливої уваги заслуговує вибір Qt як основного інструменту для побудови інтерфейсу користувача. Qt є потужним кросплатформним фреймворком, який забезпечує широкі можливості для створення графічних інтерфейсів, взаємодії з мережею, обробки подій і зберігання даних. Крім того, Qt має хорошу інтеграцію з C++, що дозволяє реалізувати як продуктивну бізнес-логіку, так і зручний інтерфейс.

Іншою ключовою складовою є Google Calendar API, що дозволяє безпосередньо взаємодіяти з даними календаря за допомогою офіційного REST API. Це надає можливість:

1. Отримувати перелік календарів користувача.
2. Читати події з календаря.
3. Створювати нові події.
4. Оновлювати існуючі події.
5. Видалити події.

Доступ до Google Calendar API забезпечується через протокол OAuth 2.0, який гарантує безпечну авторизацію без передачі паролів.

Таким чином, предметна область охоплює цифрове планування подій, інтеграцію з хмарними сервісами, розробку зручного десктопного інтерфейсу та забезпечення безпеки взаємодії з персональними даними. Успішна реалізація програмної системи у цій предметній області дозволить створити сучасний інструмент для ефективного управління подіями з використанням вже знайомої інфраструктури Google.

1.1.2 Постановка задачі

Розвиток цифрових технологій призвів до того, що все більше людей покладаються на електронні засоби для планування свого часу, організації зустрічей, контролю дедлайнів і синхронізації різноманітних подій у своєму житті. Одним із найпопулярніших рішень для таких завдань є Google Calendar — хмарний сервіс, який дозволяє користувачам створювати, редагувати, переглядати події та організовувати спільні зустрічі. Однак, попри свою зручність і багатофункціональність, Google Calendar не має офіційного повноцінного десктопного клієнта, що особливо важливо для користувачів, які працюють за

персональними комп’ютерами або ноутбуками та віддають перевагу окремим додаткам замість роботи у браузері.

У зв’язку з цим виникає потреба у створенні настільного застосунку-планувальника, який дозволив би зручно керувати календарем подій, використовуючи локальний графічний інтерфейс, водночас залишаючись синхронізованим із хмарним середовищем Google. Такий додаток має бути побудований на основі кросплатформного фреймворку Qt, що забезпечить можливість роботи на різних операційних системах, зокрема Windows і Linux.

Основне завдання полягає у розробці програмного забезпечення, яке забезпечуватиме авторизацію користувача за допомогою протоколу OAuth 2.0, взаємодію з Google Calendar API, завантаження та відображення подій, а також їх створення, редагування та видалення. Окрім цього, система повинна забезпечити збереження даних користувача, підтримку роботи в офлайн-режимі з подальшою синхронізацією змін після підключення до Інтернету, та реалізувати систему нагадувань про майбутні події через локальні сповіщення.

Архітектура застосунку повинна бути модульною та гнучкою, щоб у майбутньому можливо було без суттєвих змін додавати нову функціональність, наприклад підтримку інших календарних сервісів або персоналізацію зовнішнього вигляду інтерфейсу. Успішною реалізацією проєкту вважатиметься працездатна система, яка дозволяє користувачу проходити автентифікацію, переглядати свій розклад подій, керувати ними через зручний інтерфейс та синхронізувати зміни з Google Calendar без втрати даних.

1.1.3 Пошук актантів та варіантів використання

На цьому етапі розробки важливо проаналізувати основних учасників взаємодії з програмною системою (так званих актантів), а також визначити типові сценарії їхньої діяльності у контексті роботи з десктопним планувальником. Це

дозволить чітко окреслити функціональні межі майбутньої програми, зрозуміти її користувацькі потреби та забезпечити створення зручного, інтуїтивно зрозумілого інтерфейсу.

Головним актантом у даній системі виступає користувач — фізична особа, яка встановлює програму на свій персональний комп’ютер і використовує її для організації власного календаря. Користувач повинен мати можливість проходити авторизацію через обліковий запис Google, отримувати доступ до своїх календарів, переглядати та редагувати події, а також отримувати сповіщення про майбутні завдання. Для нього важливою є зручність у користуванні, швидка взаємодія з інтерфейсом та стабільність синхронізації з хмарною службою.

Другим, опосередкованим актантом виступає Google Calendar API — зовнішній сервіс, з яким система встановлює зв’язок через мережу Інтернет. У цьому випадку взаємодія є строго визначеною згідно з технічною специфікацією API. Цей актант надає можливість доступу до календарів, здійснення операцій з подіями (читання, створення, редагування, видалення), а також забезпечує захищену передачу даних.

У межах взаємодії користувача з програмною системою можна виокремити кілька основних варіантів використання (сценаріїв), які описують типову поведінку. Найбільш базовим є сценарій первинного входу до системи, коли користувач уперше запускає програму, проходить автентифікацію через OAuth 2.0 і надає дозвіл на доступ до своїх календарів. Після цього користувач може переглядати наявні події у вигляді календаря, фільтрувати їх за датою або назвою, створювати нові події, редагувати їх або видаляти, якщо більше не потребує.

Ще одним варіантом використання є робота в умовах обмеженого або повністю відсутнього доступу до мережі. У цьому випадку програма повинна забезпечити локальне збереження змін, які вносяться користувачем, і після відновлення підключення до Інтернету — автоматично синхронізувати дані з хмарним сервісом. Таким чином, навіть у нестабільних мережевих умовах користувач не втрачає можливості працювати з календарем.

Також окремої уваги заслуговує сценарій нагадувань — ситуація, коли програма повинна сповіщати користувача про майбутні події згідно з їхніми налаштуваннями. Це можуть бути як спливаючі вікна на екрані, так і системні повідомлення, які з’являються у системному треї. Для користувача це є важливим елементом, що дозволяє не пропустити важливу зустріч або завдання.

Узагальнюючи, аналіз актантів та варіантів використання дозволяє структурувати вимоги до системи, глибше зрозуміти очікування користувача, а також забезпечити правильне планування архітектури програмного продукту. Саме на основі виявлених сценаріїв у подальшому формулюються вимоги до функціоналу, інтерфейсу та взаємодії із зовнішніми сервісами..

1.1.4 Опис ключових варіантів використання

Опис ключових варіантів використання (англ. use cases) є одним із найважливіших етапів аналізу вимог до програмної системи. Він дозволяє не лише виявити функціональні потреби користувача, але й структурувати логіку взаємодії між користувачем і програмою у вигляді послідовних дій. У цьому контексті застосунок-планувальник подій повинен реалізовувати низку критичних варіантів використання, які забезпечують його основну функціональність.

Першим та базовим варіантом використання є авторизація користувача через Google. Після запуску застосунку користувач має пройти процес автентифікації, що передбачає відкриття вебсторінки Google з відповідним запитом на доступ до даних календаря. Після успішного підтвердження дозволів користувач повертається до застосунку, де система отримує авторизаційний токен та оновлює інтерфейс відповідно до наявних календарів. У разі повторного запуску застосунок повинен розпізнавати збережені облікові дані і, за наявності дійсного токену, автоматично здійснювати вхід без повторного втручання користувача.

Наступним важливим варіантом є завантаження подій з Google Calendar. Після входу система ініціює запит до Google Calendar API з метою отримання списку подій за вибраний часовий проміжок. Ці події відображаються у вигляді зручного графічного представлення — списку, таблиці або візуального календаря з поділом за днями, тижнями чи місяцями. Користувач може переглядати деталі кожної події, такі як назва, опис, дата, час, список учасників та інші параметри.

Ще одним ключовим варіантом є створення нової події. При натисканні відповідної кнопки у графічному інтерфейсі відкривається форма, в якій користувач заповнює інформацію про подію: назву, опис, дату і час початку й завершення, а також за потреби — додає місце проведення, колір, повторюваність чи учасників. Після підтвердження система надсилає відповідний POST-запит до API Google Calendar, що дозволяє створити нову подію у вибраному календарі.

Редагування події реалізується подібно: користувач обирає наявну подію, вносить зміни до її параметрів, після чого система надсилає PUT-запит для оновлення цієї події на стороні Google. Аналогічно, видалення події ініціюється через інтерфейс, після чого система виконує DELETE-запит до API.

Важливим варіантом використання є синхронізація подій. У випадку, коли користувач працює без з’єднання з мережею, усі зміни зберігаються локально у тимчасовому сховищі. Після відновлення з’єднання система повинна автоматично виконати синхронізацію — надіслати зміни до Google Calendar та оновити локальний кеш відповідно до актуального стану хмарного сервера. Така функціональність критично важлива для забезпечення безперервної роботи користувача.

Окремим варіантом використання є система нагадувань. Користувач може налаштувати сповіщення, які будуть з’являтись за певний час до початку події. Ці сповіщення можуть реалізовуватись через спливаючі вікна у програмі або інтегруватись із системними повідомленнями операційної системи. Своєчасне інформування про заплановані події підвищує ефективність користувача і робить роботу з календарем більш надійною.

Рис. 1.2 – Діаграма варіантів використання системи.

Усі ці варіанти використання формують ядро функціональності застосунку. Саме на основі таких сценаріїв взаємодії надалі проектується архітектура, інтерфейс користувача та програмна логіка всієї системи. Їх реалізація гарантує відповідність застосунку вимогам користувача і забезпечує його реальну цінність як інструменту цифрового планування.

1.1.5 Моделювання словника системи

У процесі аналізу програмної системи важливо чітко визначити ключові поняття, терміни та об’єкти, які використовуються в рамках проєкту. Це дозволяє сформувати єдину термінологічну базу, що спрощує подальше проєктування,

розробку та тестування програмного забезпечення. Такий словник слугує своєрідною моделлю предметної області, яка містить як технічні, так і прикладні поняття.

Нижче подано основні терміни, що становлять ядро словника системи «Десктопний планувальник розкладу на C++ з використанням Qt та Google Calendar API»:

1. Користувач — фізична особа, яка працює з програмою через графічний інтерфейс. Користувач має обліковий запис Google, що дозволяє йому синхронізувати події через Google Calendar API.
2. Подія (Event) — одиниця календарного планування. Містить інформацію про дату, час початку та завершення, назву, опис, місце, а також, за потреби, список запрошених учасників.
3. Календар (Calendar) — набір подій, згрупованих за тематиками або призначенням. Один користувач може мати декілька календарів (особистий, робочий, публічний тощо).
4. Авторизація (Authorization) — процес входу користувача до програми із підтвердженням прав доступу до свого облікового запису Google за допомогою протоколу OAuth 2.0.
5. Токен доступу (Access Token) — цифровий ключ, що видається після авторизації і дозволяє тимчасово взаємодіяти з Google Calendar API без повторного входу.
6. Google Calendar API — зовнішній вебсервіс, що надає програмний інтерфейс для створення, редагування, перегляду та видалення календарних подій у хмарному обліковому записі Google.
7. Qt Framework — кросплатформна бібліотека для створення графічного інтерфейсу, що використовується для реалізації вікон, кнопок, календарів та інших компонентів UI.

Словник системи є динамічним: у процесі розробки можуть з’являтися нові поняття або змінюватися формулювання існуючих. Проте на етапі аналізу цей

перелік термінів створює базу для уніфікованої комунікації між учасниками розробки та забезпечує узгоджене бачення системи.

1.2. Проектування програмної системи

Проєктування програмної системи є ключовим етапом у процесі створення програмного забезпечення, оскільки саме на цьому етапі визначається структура майбутньої програми, її архітектура, логіка взаємодії компонентів, організація зберігання даних та моделі обробки інформації. Ретельне проєктування дозволяє підвищити масштабованість, розширюваність і підтримуваність застосунку, а також зменшити ймовірність помилок у подальшій розробці.

У цьому розділі розглянуто вибір підходу до розробки програмного продукту, побудову логічної схеми бази даних, моделювання класів системи за допомогою UML-нотації, а також загальну архітектуру застосунку, що визначає взаємодію між його основними модулями.

1.2.1 Вибір процесу розробки

Вибір процесу розробки програмного забезпечення є критично важливим рішенням, що визначає ефективність побудови програмної системи, взаємодію між розробником та іншими учасниками проєкту, гнучкість до змін і контроль за якістю програмного продукту.

Для реалізації десктопного планувальника розкладу на основі C++ із використанням фреймворку Qt і API Google Calendar було обрано ітеративно-інкрементний підхід із елементами методології Agile. Такий підхід

забезпечує поступове створення системи шляхом поділу розробки на окремі етапи (ітерації), у межах яких реалізується та тестується певна частина функціоналу.

Цей підхід було обрано з таких причин:

1. Гнучкість. У процесі реалізації системи можливі зміни вимог, особливо під час інтеграції з зовнішніми API або уточнення побажань користувача. Ітеративна модель дозволяє швидко адаптуватися до таких змін без необхідності повного перероблення системи.
2. Можливість раннього тестування. Після кожної ітерації створюється робоча версія системи, яку можна протестувати та оцінити. Це дозволяє вчасно виявити помилки, підвищити якість коду та уникнути накопичення технічного боргу.
3. Пріоритезація функціоналу. Розробка поділяється на логічні блоки: авторизація, відображення подій, створення подій, синхронізація тощо. Найважливіші модулі реалізуються першими, що дозволяє отримати MVP (мінімально життєздатний продукт) на ранніх етапах.
4. Контрольованість розробки. Кожен етап супроводжується плануванням, веденням завдань, аналізом ризиків, рефакторингом і проміжним тестуванням. Це дозволяє системно контролювати прогрес і швидко реагувати на проблеми.

Загалом, обрана модель розробки сприяє створенню якісного, масштабованого й адаптивного застосунку, що відповідає сучасним вимогам розробки користувацького програмного забезпечення з інтеграцією хмарних сервісів.

1.2.2 Побудова схеми бази даних

У рамках реалізації настільного застосунку-планувальника подій була обрана система керування базами даних SQLite. Це вбудована, легка та

кросплатформна СКБД, що не потребує окремого сервера та ідеально підходить для інтеграції з застосунками на базі Qt, зокрема через модуль QSqlDatabase. Вона забезпечує просте локальне збереження даних, зберігаючи всю базу у вигляді одного файлу на диску. Такий вибір дозволяє організувати як кешування подій, отриманих із Google Calendar, так і збереження змін, зроблених користувачем у режимі офлайн.

База даних виконує одразу кілька функцій: з одного боку, вона зберігає останній стан синхронізованих подій, що дозволяє користувачеві переглядати розклад навіть за відсутності підключення до мережі; з іншого — фіксує зміни, які були внесені локально до моменту відновлення з’єднання, з можливістю подальшої синхронізації з хмарним сервісом Google Calendar.

Основою структури бази даних є таблиця Event, у якій зберігаються всі основні характеристики подій — такі як назва, опис, час початку й завершення, місце проведення тощо. Кожна подія має унікальний локальний ідентифікатор, а також, у разі синхронізації з Google, глобальний gcal_id. Додаткову інформацію про сповіщення містить пов’язана таблиця Reminder, яка має зовнішній ключ до Event та описує тип і час нагадування.

Кожна подія також належить до певного календаря. Дані про календарі зберігаються у таблиці Calendar, що містить їх назви, кольорове оформлення, прапорці видимості, а також глобальні ідентифікатори, які надаються API Google. Зв’язок між подією та календарем реалізується через зовнішній ключ calendar_id.

Для забезпечення синхронізації змін, внесених офлайн, створено таблицю SyncQueue, яка фіксує всі операції додавання, оновлення або видалення подій. Кожен запис містить тип операції, JSON-представлення даних, що були змінені, та часову позначку. Це дозволяє системі точно визначити порядок дій і синхронізувати їх після відновлення доступу до Інтернету.

Для забезпечення безперервної авторизації застосунок зберігає access- і refresh-токени у таблиці AuthToken. Термін дії токенів зазначається окремим полем expires_at. Для безпеки токени зберігаються в зашифрованому вигляді, а

ключі шифрування інтегруються з вбудованим сховищем облікових даних операційної системи.

У базі також передбачено каскадне видалення пов’язаних записів, наприклад: при видаленні події автоматично видаляються всі її нагадування. Водночас реалізовані індекси за полями часу початку і завершення події, а також за ідентифікаторами Google, що забезпечує ефективний пошук і фільтрацію під час роботи застосунку.

Загальна структура бази даних є логічно нормалізованою, що забезпечує можливість розширення функціоналу. У майбутньому до системи можуть бути додані таблиці для зберігання вкладень або списку учасників подій без необхідності суттєвих змін у вже наявних таблицях. Отже, така схема бази даних дозволяє гнучко реалізувати всі необхідні функції, забезпечуючи надійне зберігання даних, можливість роботи офлайн, синхронізацію зі стороннім API та захист конфіденційної інформації користувача.

1.2.3 Побудова UML-діаграми класів

UML-діаграма класів є важливим інструментом для формального опису об'єктно-орієнтованої структури програмної системи. Вона дозволяє візуалізувати основні сутності, які будуть реалізовані в коді, їхні атрибути, методи, а також взаємозв’язки між класами. Побудова такої діаграми ще до початку реалізації дає змогу краще продумати логіку програми, забезпечити її масштабованість і полегшити подальше тестування та супровід.

У випадку розробки десктопного планувальника подій основними об'єктами, які реалізовані у вигляді класів, є подія, календар, нагадування, користувач, менеджер синхронізації, менеджер бази даних, а також інтерфейсні компоненти, що взаємодіють із цими класами. Основна бізнес-логіка зосереджена в таких

класах, як Event, Reminder, Calendar, SyncManager, DatabaseManager і GoogleApiClient.

Клас Event описує структуру події та включає атрибути, такі як заголовок, опис, час початку і завершення, місце проведення та ідентифікатори. Він також може містити список нагадувань (Reminder), які зберігаються як залежні об'єкти. Клас Calendar містить інформацію про конкретний календар (назва, колір, глобальний ID), до якого належать події. Для взаємодії з API Google використовується клас GoogleApiClient, що відповідає за авторизацію користувача, запити до API та обробку відповідей.

Компонент SyncManager реалізує логіку синхронізації змін між локальною базою даних та хмарним сервісом. Він взаємодіє з DatabaseManager, який безпосередньо відповідає за з’єднання з SQLite-базою та виконання CRUD-операцій.

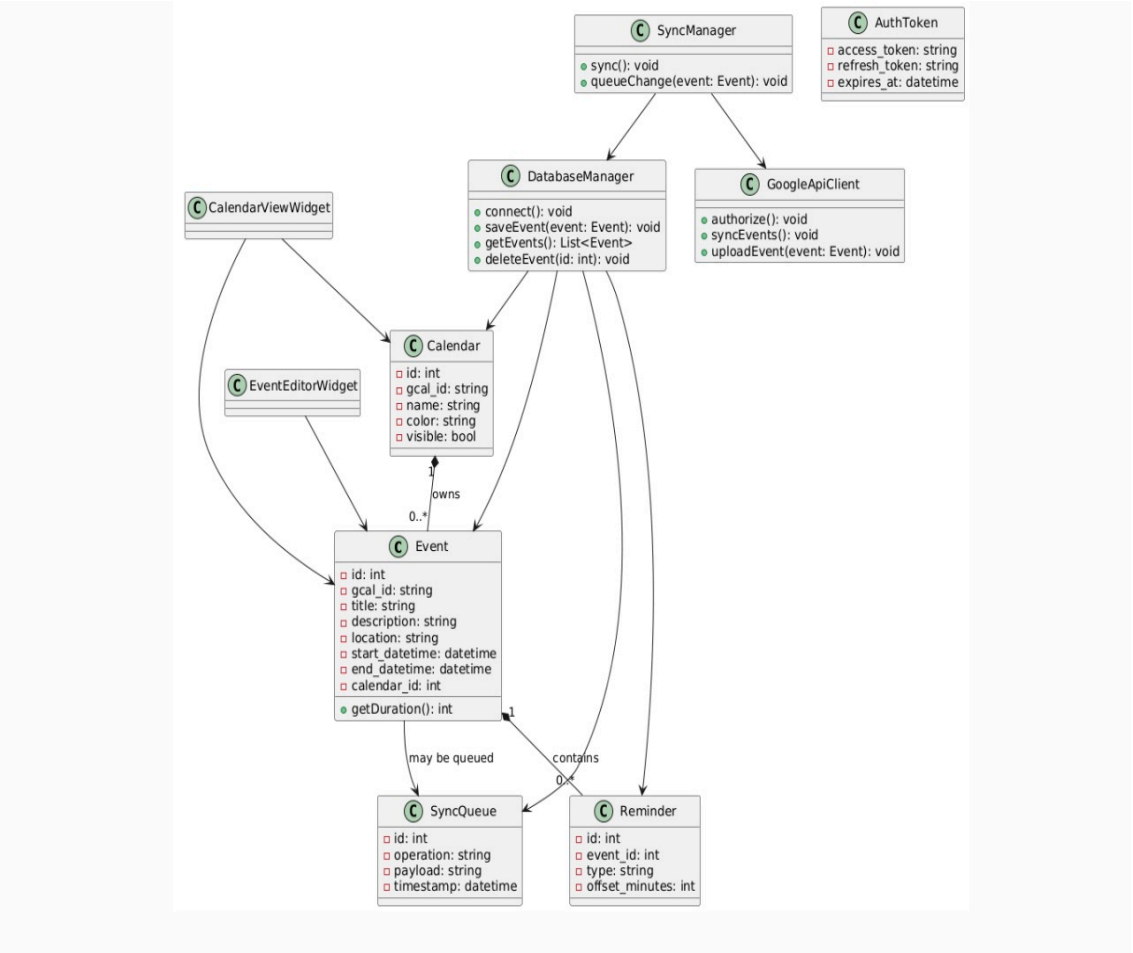


Рис. 1.3 – Діаграма класів системи.

З точки зору архітектури застосунку, інтерфейс користувача (створений за допомогою Qt) містить окремі компоненти для перегляду календаря, створення й редагування подій, авторизації та перегляду списку нагадувань. Ці елементи використовують відповідні класи даних як моделі, дотримуючись принципу MVC (Model-View-Controller).

Діаграма класів відображає зв’язки між цими сутностями: агрегацію (наприклад, подія складається з кількох нагадувань), залежність (використання одного класу іншим) або наслідування, якщо певні компоненти реалізують спільні інтерфейси чи базові класи. Це дозволяє зрозуміти загальну структуру застосунку, визначити точки розширення та передбачити вплив змін у класах на решту системи.

1.2.4 Моделювання архітектури системи

Архітектура програмної системи визначає загальні принципи побудови її структурних компонентів, способи їхньої взаємодії, рівні абстракції та відповідальність кожного модуля. У випадку розробки настільного планувальника подій з інтеграцією Google Calendar архітектура орієнтована на модульність, підтримку офлайн-режиму, зручність користування та надійність синхронізації.

Для проєктування архітектури було обрано багаторівневий підхід із чітким поділом відповідальності між логікою представлення (Presentation Layer), логікою взаємодії з даними (Data Layer), локальним зберіганням (Persistence Layer) та зовнішніми сервісами (Integration Layer). Кожен із цих рівнів реалізовано окремими класами або модулями, які комунікують між собою через чітко визначені інтерфейси.

Презентаційний рівень представлений графічним інтерфейсом користувача, побудованим на базі бібліотеки Qt. Основними його елементами є віджет перегляду календаря, віджет створення та редагування подій, форма авторизації, а

також повідомлення та сповіщення. Вся взаємодія користувача з програмою відбувається через ці компоненти. Вони передають команди в бізнес-логіку через сигнали й слоти, дотримуючись принципу Model-View-Controller (MVC).

Бізнес-логіка реалізується у вигляді окремих класів-моделей, що оперують об’єктами на кшталт подій, календарів або нагадувань. За обробку змін у цих моделях відповідає менеджер синхронізації (SyncManager), який відстежує операції створення, редагування та видалення, зберігає їх у локальній черзі змін, а при наявності мережі — викликає відповідні методи API-клієнта. Також бізнес-логіка включає клас планування нагадувань, який періодично перевіряє, чи потрібно сповістити користувача про події.

Доступ до локального сховища реалізований у вигляді окремого модуля DatabaseManager, який використовує SQLite для зберігання об'єктів. Він містить методи для збереження, вибірки, оновлення та видалення записів. Для авторизації, отримання списку календарів та подій використовується окремий клас GoogleApiClient, який працює через HTTP-запити з авторизацією OAuth 2.0. Отримані від Google Calendar дані перетворюються на локальні об'єкти і передаються до бази даних або напряму у віджети інтерфейсу.

Уся архітектура спроєктована таким чином, щоб забезпечити незалежність між модулями, простоту тестування та можливість розширення системи. Наприклад, при необхідності можна додати інший тип календарного сервісу або синхронізуватися через інший API без зміни решти логіки програми. Окрім того, архітектура підтримує автономний режим роботи — коли застосунок тимчасово не має доступу до Інтернету, усі зміни зберігаються локально та пізніше синхронізуються автоматично.

У результаті спроєктована архітектура поєднує простоту реалізації з гнучкістю й масштабованістю, забезпечує стійкість до мережевих збоїв і дозволяє зручно реалізувати сучасний інтерфейс користувача.

Загалом, така модульна архітектура дозволяє легко реалізувати функціональність розширення та адаптації системи під нові вимоги. Наприклад, у майбутньому можлива інтеграція з іншими календарними сервісами (Microsoft

Outlook, Apple Calendar), додавання підтримки спільного редагування подій або автоматичне оновлення через фонові служби. Завдяки чіткому поділу логіки між рівнями система залишається підтримуваною, тестованою та зручною для командної розробки.

1.3. Конструювання програмної системи

Конструювання програмної системи є одним із найважливіших етапів життєвого циклу програмного забезпечення, що передбачає реалізацію технічних рішень, спроєктованих на попередньому етапі. Основна мета цього розділу — детально описати обґрунтування вибору технологій, мови програмування, середовища розробки, бази даних, а також представити реалізацію основних компонентів системи.

У цьому розділі розглядається, чому саме було обрано мову C++ у поєднанні з фреймворком Qt, які переваги дає використання конкретного середовища розробки, як реалізовано зберігання інформації про події та календарі за допомогою локальної бази даних SQLite, а також як побудовано основні класи, що реалізують функціональність взаємодії з Google Calendar API, обробки подій та керування нагадуваннями.

1.3.1 Вибір мови та середовища розробки

Під час створення програмної системи було обрано мову програмування C++ у поєднанні з фреймворком Qt. Такий вибір зумовлений кількома факторами: по-перше, C++ забезпечує високу продуктивність, контроль над ресурсами та стабільність, що особливо важливо для настільних застосунків, які мають

працювати швидко та ефективно на різних операційних системах. По-друге, Qt надає широкий спектр засобів для створення кросплатформеного графічного інтерфейсу користувача, підтримує роботу з мережевими протоколами, локальними базами даних та має зручну систему сигналів і слотів, що значно спрощує побудову взаємодії між компонентами.

Розробка здійснювалася в середовищі Qt Creator — офіційному IDE для Qt. Воно має зручний графічний редактор форм, розширене автодоповнення, інтегровані засоби для побудови, налагодження та профілювання проєкту. Збірка застосунку виконується за допомогою CMake, що дозволяє точно контролювати процес компіляції, структуру проєкту та залежності.

Крім того, вибір Qt забезпечує легку адаптацію застосунку під різні платформи, включаючи Windows, Linux і macOS, без необхідності змінювати основну логіку або переписувати інтерфейс. Це значно полегшує тестування та розгортання програми на різних типах комп’ютерів. Підтримка роботи з бібліотекою QtNetwork дозволяє реалізувати взаємодію з Google Calendar API через HTTPS-запити, а QtSql забезпечує доступ до локальної бази даних SQLite, що використовується для зберігання подій, календарів та нагадувань.

Таким чином, поєднання C++, Qt і Qt Creator забезпечує ефективне середовище для розробки, яке поєднує продуктивність, зручність у створенні графічного інтерфейсу та достатню гнучкість для розширення функціональності в майбутньому.

1.3.2 Вибір СУБД та опис її фізичної моделі

Одним із ключових завдань при розробці настільного застосунку є організація зберігання даних. Для даної системи було обрано використання локальної реляційної бази даних SQLite, яка ідеально підходить для десктопних застосунків. Основними критеріями вибору стали її легковаговість, відсутність

потреби в окремому сервері, повна інтеграція з Qt через модуль QtSql, а також надійність та висока швидкість обробки запитів.

База даних використовується для зберігання всієї ключової інформації: подій, календарів, нагадувань, а також історії змін, що підлягають синхронізації з Google Calendar. Кожна таблиця має чітко визначену структуру та зв’язки з іншими таблицями, що дозволяє підтримувати цілісність даних.

Фізична модель бази даних представлена такими основними таблицями:

1. events — зберігає інформацію про події: назва, опис, час початку і завершення, місце, пов’язаний календар, а також зовнішній ідентифікатор з Google Calendar.
2. calendars — містить дані про підключені календарі користувача, включаючи їхні назви, кольори та статус відображення.
3. reminders — включає налаштування нагадувань для кожної події: тип (сповіщення або email), час до події.
4. sync_queue — слугує для накопичення змін у локальній базі, які необхідно синхронізувати з Google Calendar (створення, оновлення, видалення).
5. auth_tokens — таблиця, що зберігає токени авторизації для доступу до Google API.

Зв’язки між таблицями реалізовані за допомогою зовнішніх ключів. Наприклад, таблиця events має зовнішній ключ до таблиці calendars, а таблиця reminders — до events. Це дозволяє реалізувати каскадне видалення та підтримувати узгодженість даних при зміні або видаленні подій чи календарів.

Вся логіка взаємодії з базою даних реалізована у вигляді класу DatabaseManager, який інкапсулює SQL-запити та забезпечує зручні методи доступу до даних (отримання подій за діапазоном дат, збереження нової події, оновлення, видалення тощо). Такий підхід не лише підвищує надійність системи, а й забезпечує гнучкість при можливій зміні структури бази або переході до іншої СУБД у майбутньому.

Таким чином, використання SQLite як СУБД забезпечує необхідний баланс між простотою реалізації, функціональністю та продуктивністю, що є оптимальним вибором для автономної десктопної системи з можливістю синхронізації з хмарними сервісами.

1.3.3 Зовнішні бібліотеки та залежності

У процесі розробки програмної системи було використано низку зовнішніх бібліотек та модулів, які забезпечили інтеграцію з веб-сервісами, обробку мережевих запитів, авторизацію користувача та побудову графічного інтерфейсу.

Основною бібліотекою, на базі якої створено інтерфейс користувача, є Qt — кросплатформений фреймворк для C++, який надає розширені можливості для створення віконних застосунків. Для даної системи було задіяно наступні модулі Qt:

1. QtWidgets — для побудови графічного інтерфейсу;
2. QtNetwork — для здійснення HTTP-запитів до API Google;
3. QtCore — для роботи з датами, часом, сигналами/слотами та об'єктами Qt;
4. QtOAuth — для реалізації авторизації користувача через протокол OAuth 2.0.

Для реалізації авторизації за допомогою Google було використано клас QOAuth2AuthorizationCodeFlow, що дозволяє отримати токен доступу до Google Calendar API, необхідний для взаємодії з подіями календаря. Мережева взаємодія реалізується через QNetworkAccessManager, який забезпечує виконання HTTP-запитів до серверу Google API.

Також активно використовувався JSON-формат для обміну даними з API Google. Для цього застосовувались класи QJsonObject, QJsonDocument та QJsonArray з модуля QtCore, які дозволяють зручно формувати тіла запитів і парсити відповіді.

Для побудови та організації проєкту використовувалась система автоматизованої збірки CMake. Файл CMakeLists.txt містить інструкції для компіляції проєкту з урахуванням залежностей від модулів Qt, а також забезпечує правильну генерацію Make-файлів або проектів під середовища розробки (наприклад, Qt Creator).

1.3.4 Реалізація основних класів та методів

Графічний інтерфейс користувача є однією з ключових складових програмної системи, адже саме через нього здійснюється вся взаємодія користувача з програмою. Для розробки інтерфейсу було використано фреймворк Qt, який забезпечує гнучкі засоби створення вікон, кнопок, полів вводу та інших компонентів, необхідних для побудови зручного інтерфейсу. Робота над графічною частиною виконувалась із використанням візуального редактора Qt Designer, що дало змогу швидко компонувати елементи інтерфейсу та зосередитися на їх функціональності.

У головному вікні програми реалізовано основні елементи взаємодії, серед яких – кнопка авторизації через обліковий запис Google, таблиця для відображення подій з календаря, поля для введення назви події та часу її початку, а також кнопки для додавання, редагування та видалення подій. Кожна взаємодія з інтерфейсом обробляється за допомогою сигналів і слотів, що є типовим механізмом у середовищі Qt. Наприклад, після натискання кнопки додавання події активується відповідна функція, яка формує запит до API Google Calendar та додає нову подію. Аналогічно обробляються операції видалення чи редагування існуючих подій. Після кожної дії таблиця оновлюється автоматично, а поля вводу очищуються, що дозволяє ефективно працювати з даними без необхідності ручного оновлення інтерфейсу.

Інтерфейс програми є простим, зрозумілим і не потребує попередньої підготовки користувача. Завдяки використанню фреймворку Qt він виглядає однаково на різних операційних системах, забезпечуючи стабільну та зручну роботу. Перед авторизацією користувачу доступне лише головне вікно з кнопкою входу в обліковий запис Google (вставити скріншот 1 — головне вікно до авторизації). Після успішної авторизації з’являється доступ до функцій керування подіями, і таблиця заповнюється подіями, отриманими з Google Календаря (вставити скріншот 2 — інтерфейс після авторизації). У вікні також відображається приклад введення нової події з вказанням назви та часу початку (вставити скріншот 3 — введення нової події), після чого подія додається до таблиці (вставити скріншот 4 — оновлена таблиця подій).

Таким чином, інтерфейс користувача в програмній системі є максимально зручним, функціональним та орієнтованим на просту й швидку роботу з календарними подіями.

1.3.5 Реалізація основних класів та методів

Процес авторизації реалізується за допомогою QOAuth2Authorization та локального веб-сервера, що очікує на відповідь від Google. Після успішного отримання токена доступу застосунок може виконувати запити до API календаря.

```
oauth->setAuthorizationUrl(QUrl("https://accounts.google.com/o/oauth2/auth"));
oauth->setAccessTokenUrl(QUrl("https://oauth2.googleapis.com/token"));
oauth->setClientIdentifier("246209458791-fcob0je8feponk14q8cknjv2ismuivm3.apps.googleusercontent.com");
oauth->setClientIdentifierSharedKey("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");
oauth->setScope("https://www.googleapis.com/auth/calendar");

auto replyHandler = new QOAuthHttpServerReplyHandler(8080, this);
oauth->setReplyHandler(replyHandler);

connect(oauth, &QOAuth2AuthorizationCodeFlow::authorizeWithBrowser,
        [](const QUrl &url) { QDesktopServices::openUrl(url); });

connect(oauth, &QOAuth2AuthorizationCodeFlow::granted,
        this, &MainWindow::onAuthorized);

connect(ui->authorizeButton, &QPushButton::clicked,
        this, &MainWindow::onAuthorizeButtonClicked);

connect(ui->addEventButton, &QPushButton::clicked,
        this, &MainWindow::onAddEventButtonClicked);

connect(ui->deleteEventButton, &QPushButton::clicked,
        this, &MainWindow::onDeleteEventButtonClicked);

connect(ui->updateEventButton, &QPushButton::clicked,
        this, &MainWindow::onUpdateEventButtonClicked);

connect(ui->eventTable, &QTableWidget::cellClicked,
        this, &MainWindow::onEventTableClicked);
```

Рис. 1.3.3.1 – Процес авторизації.

Основним компонентом програмної системи є головне вікно, реалізоване у класі MainWindow, який успадковується від QMainWindow. Саме цей клас відповідає за графічний інтерфейс користувача, обробку подій інтерфейсу та взаємодію із зовнішнім сервісом Google Calendar за допомогою OAuth2-автентифікації та REST-запитів.

Метод onAuthorizeButtonClicked() викликає процедуру авторизації через відкриття браузера за посиланням від Google. Після завершення авторизації метод onAuthorized() формує запит до API Google Calendar, обробляє відповідь, парсить отримані події з JSON і формує список об'єктів CalendarEvent.

```
void MainWindow::onAuthorized()
{
    QUrl url("https://www.googleapis.com/calendar/v3/calendars/primary/events");
    QUrlQuery query;
    query.addQueryItem("maxResults", "20");
    query.addQueryItem("orderBy", "startTime");
    query.addQueryItem("singleEvents", "true");
    query.addQueryItem("timeMin", QDateTime::currentDateTimeUtc().toString(Qt::ISODate));
    url.setQuery(query);

    QNetworkRequest request(url);
    request.setRawHeader("Authorization", "Bearer " + oauth->token().toUtf8());

    QNetworkReply* reply = networkManager->get(request);

    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        if (reply->error() != QNetworkReply::NoError) {
            qDebug() << "Error fetching events:" << reply->errorString();
            reply->deleteLater();
            return;
        }

        QByteArray response = reply->readAll();
        QJsonDocument doc = QJsonDocument::fromJson(response);
        QJsonArray items = doc.object()["items"].toArray();

        eventListWithIds.clear();
        QList<CalendarEvent> events;

        for (const QJsonValue& val : items) {
            QJsonObject obj = val.toObject();
            QString id = obj["id"].toString();
            QString summary = obj["summary"].toString("(No title)");
            QString startTime = obj["start"].toObject().contains("dateTime")
                                    ? obj["start"].toObject()["dateTime"].toString()
                                    : obj["start"].toObject()["date"].toString();

            eventListWithIds.append(qMakePair(id, CalendarEvent{summary, startTime}));
            events.append({summary, startTime});
        }
```

Рис. 1.3.3.2 – Процес роботи з API.

Отриманий список подій оновлюється на інтерфейсі за допомогою методу updateEventTable(), який заповнює таблицю вікна відповідними даними. Це забезпечує зручне візуальне представлення майбутніх подій користувача.

```
void MainWindow::updateEventTable(const QList<CalendarEvent> &events)
{
    ui->eventTable->setRowCount(events.size());
    for (int i = 0; i < events.size(); ++i) {
        ui->eventTable->setItem(i, 0, new QTableWidgetItem(events[i].startTime));
        ui->eventTable->setItem(i, 1, new QTableWidgetItem(events[i].summary));
    }
}
```

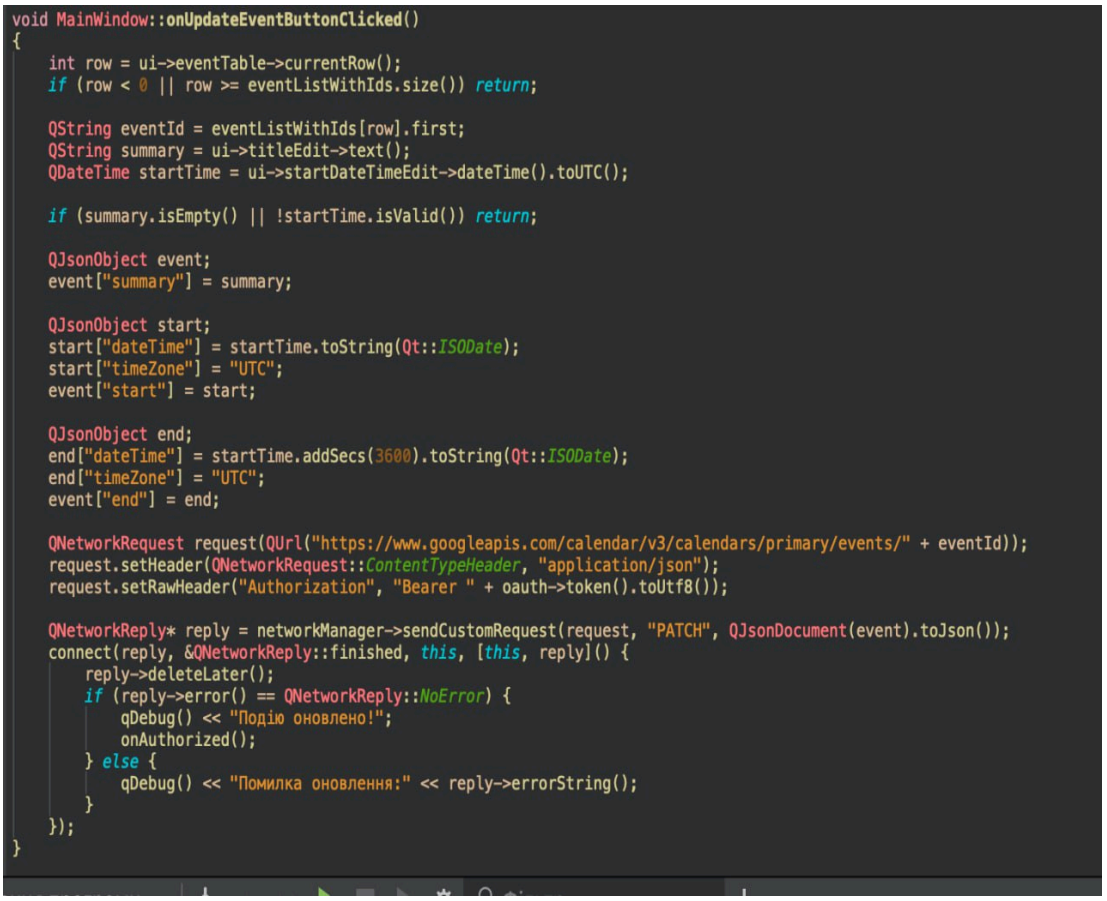
Рис. 1.3.3.3 – Оновлення календаря.

Користувач може створити нову подію за допомогою інтерфейсу, заповнивши форму, після чого викликається метод onAddEventButtonClicked(), що формує JSON-об’єкт події з датою, часом та назвою і відправляє POST-запит на сервер Google.

```
void MainWindow::onAddEventButtonClicked()
{
    QString summary = ui->titleEdit->text();
    QDateTime startTime = ui->startDateTimeEdit->dateTime().toUTC();

    if (summary.isEmpty() || !startTime.isValid()) return;

    QJsonObject event;
    event["summary"] = summary;

    QJsonObject start;
    start["dateTime"] = startTime.toString(Qt::ISODate);
    start["timeZone"] = "UTC";
    event["start"] = start;

    QJsonObject end;
    end["dateTime"] = startTime.addSecs(3600).toString(Qt::ISODate);
    end["timeZone"] = "UTC";
    event["end"] = end;

    QNetworkRequest request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/events"));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
    request.setRawHeader("Authorization", "Bearer " + oauth->token().toUtf8());

    QNetworkReply* reply = networkManager->post(request, QJsonDocument(event).toJson());

    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        reply->deleteLater();
        if (reply->error() == QNetworkReply::NoError) {
            qDebug() << "Подія додана!";
            onAuthorized();
        } else {
            qDebug() << "Помилка при додаванні події:" << reply->errorString();
        }
    });
}
```

Рис. 1.3.3.4 – Кнопка добавляння події.

Оновлення події реалізовано аналогічно, але через HTTP PATCH-запит. Метод onUpdateEventButtonClicked() дозволяє редагувати обрану подію на основі її ідентифікатора.



```
void MainWindow::onUpdateEventButtonClicked()
{
    int row = ui->eventTable->currentRow();
    if (row < 0 || row >= eventListWithIds.size()) return;

    QString eventId = eventListWithIds[row].first;
    QString summary = ui->titleEdit->text();
    QDateTime startTime = ui->startDateTimeEdit->dateTime().toUTC();

    if (summary.isEmpty() || !startTime.isValid()) return;

    QJsonObject event;
    event["summary"] = summary;

    QJsonObject start;
    start["dateTime"] = startTime.toString(Qt::ISODate);
    start["timeZone"] = "UTC";
    event["start"] = start;

    QJsonObject end;
    end["dateTime"] = startTime.addSecs(3600).toString(Qt::ISODate);
    end["timeZone"] = "UTC";
    event["end"] = end;

    QNetworkRequest request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/events/" + eventId));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
    request.setRawHeader("Authorization", "Bearer " + oauth->token().toUtf8());

    QNetworkReply* reply = networkManager->sendCustomRequest(request, "PATCH", QJsonDocument(event).toJson());
    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        reply->deleteLater();
        if (reply->error() == QNetworkReply::NoError) {
            qDebug() << "Подію оновлено!";
            onAuthorized();
        } else {
            qDebug() << "Помилка оновлення:" << reply->errorString();
        }
    });
}
```

Рис. 1.3.3.4 – Кнопка оновлення події.

Для видалення події використовується метод onDeleteEventButtonClicked(), який надсилає DELETE-запит до API Google Calendar. Після успішного видалення таблиця оновлюється автоматично.

Клас також містить допоміжні методи, зокрема onEventTableClicked(), що обробляє вибір рядка таблиці й заповнює форму редагування, забезпечуючи інтерактивність інтерфейсу.

Таким чином, реалізація класу MainWindow охоплює всю необхідну логіку для інтерактивної роботи з подіями та забезпечує повноцінний зв’язок між локальним інтерфейсом користувача та віддаленим хмарним API.

```
void MainWindow::onDeleteEventButtonClicked()
{
    int row = ui->eventTable->currentRow();
    if (row < 0 || row >= eventListWithIds.size()) return;

    QString eventId = eventListWithIds[row].first;

    QNetworkRequest request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/events/" + eventId));
    request.setRawHeader("Authorization", "Bearer " + oauth->token().toUtf8());

    QNetworkReply* reply = networkManager->deleteResource(request);
    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        reply->deleteLater();
        if (reply->error() == QNetworkReply::NoError) {
            qDebug() << "Подію видалено!";
            onAuthorized();
        } else {
            qDebug() << "Помилка видалення:" << reply->errorString();
        }
    });
}
```

Рис. 1.3.3.5 – Кнопка видалення події.

1.4. Використання програмної системи

Після завершення етапів розробки та тестування програмної системи, важливо визначити, як саме кінцеві користувачі можуть взаємодіяти з нею в реальних умовах. У цьому розділі розглядаються основні аспекти використання розробленого застосунку, зокрема процес його розгортання, вимоги до середовища, типові сценарії використання, а також засоби перевірки функціональної коректності роботи програмного забезпечення.

1.4.1 Розгортання програмної системи та системні вимоги

Для забезпечення коректної роботи розробленого десктопного планувальника подій необхідно підготувати робоче середовище, яке відповідає певним технічним та програмним вимогам.

Розгортання системи передбачає встановлення необхідних компонентів, зокрема Qt Framework (версія 5.15 або новіша), а також залежностей, пов’язаних із

роботою з мережевими запитами та OAuth2-автентифікацією. Середовище розробки Qt Creator використовується для побудови, компіляції та налагодження проєкту, а також для створення інтерфейсу користувача.

Кінцевий користувач може завантажити зібраний виконуваний файл разом із необхідними бібліотеками. Для цього застосовується утиліта windeployqt, яка автоматично додає всі потрібні файли Qt до директорії збірки. Надалі користувач може запускати програму без необхідності встановлення додаткових компонентів. Підключення до Google Calendar відбувається безпосередньо через інтернет, тому доступ до мережі є обов’язковим.

У випадку Linux-систем може бути необхідне встановлення додаткових бібліотек залежно від дистрибутива. Також потрібен браузер за замовчуванням для відкриття сторінки авторизації Google.

1.4.2 Розгортання програмної системи та системні вимоги

Програмна система призначена для взаємодії з Google Календарем, тому її основна функціональність полягає у зчитуванні, додаванні, редагуванні та видаленні подій користувача. Нижче наведено основні типові сценарії використання системи, що реалізуються через графічний інтерфейс.

Після запуску застосунку користувач бачить головне вікно, де відображено інтерфейс без завантажених подій, оскільки авторизація ще не пройдена. У цьому стані доступна лише кнопка для авторизації.

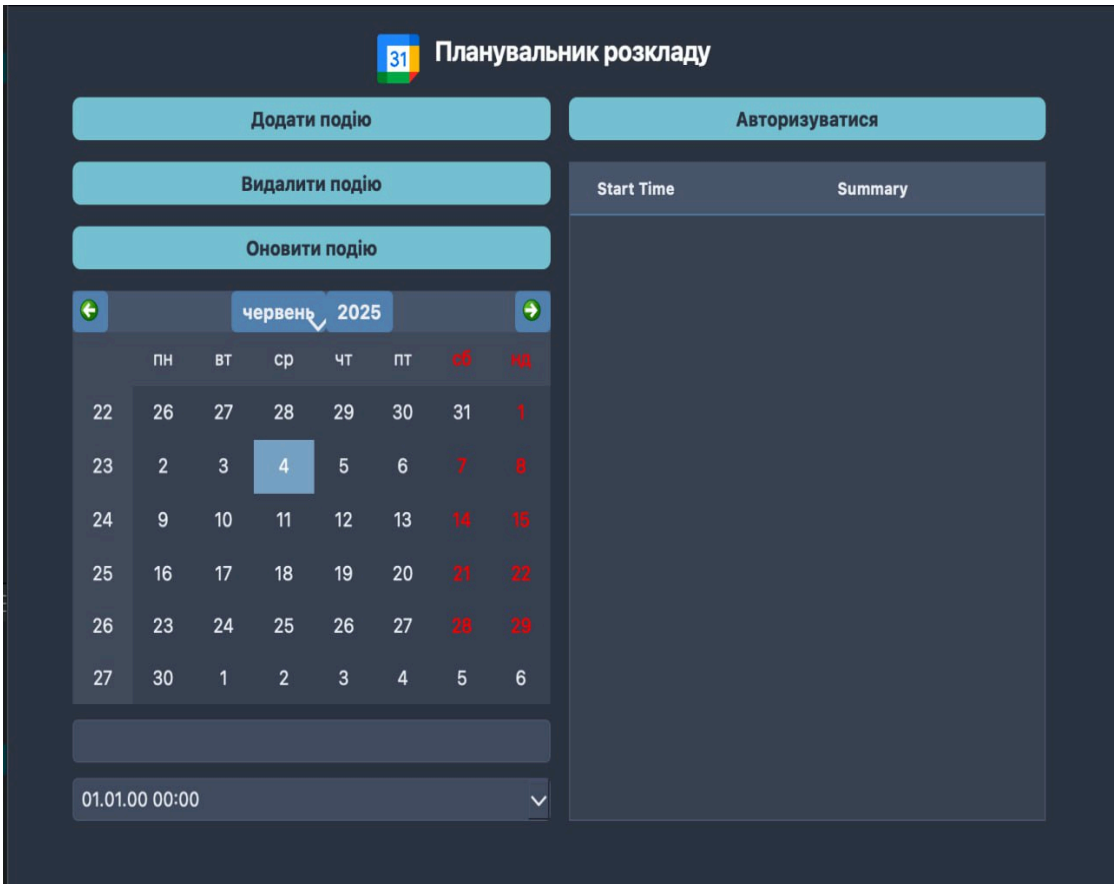


Рис. 1.4.2.1 – Інтерфейс планувальника розкладу.

Після натискання кнопки авторизації відкривається сторінка Google, де користувач підтверджує надання дозволу додатку на доступ до календаря.

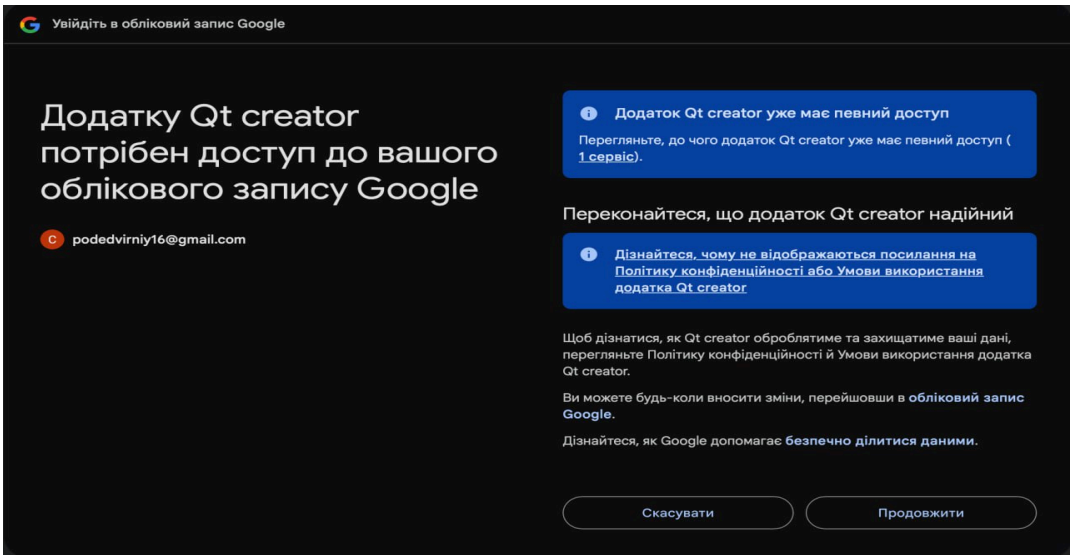


Рис. 1.4.2.2 – Авторизація через Google.

Після успішної авторизації система автоматично надсилає запит до Google Calendar API, отримує список найближчих подій (до 20) та відображає їх у вигляді таблиці в головному вікні. Також активуються кнопки для додавання, редагування та видалення подій.

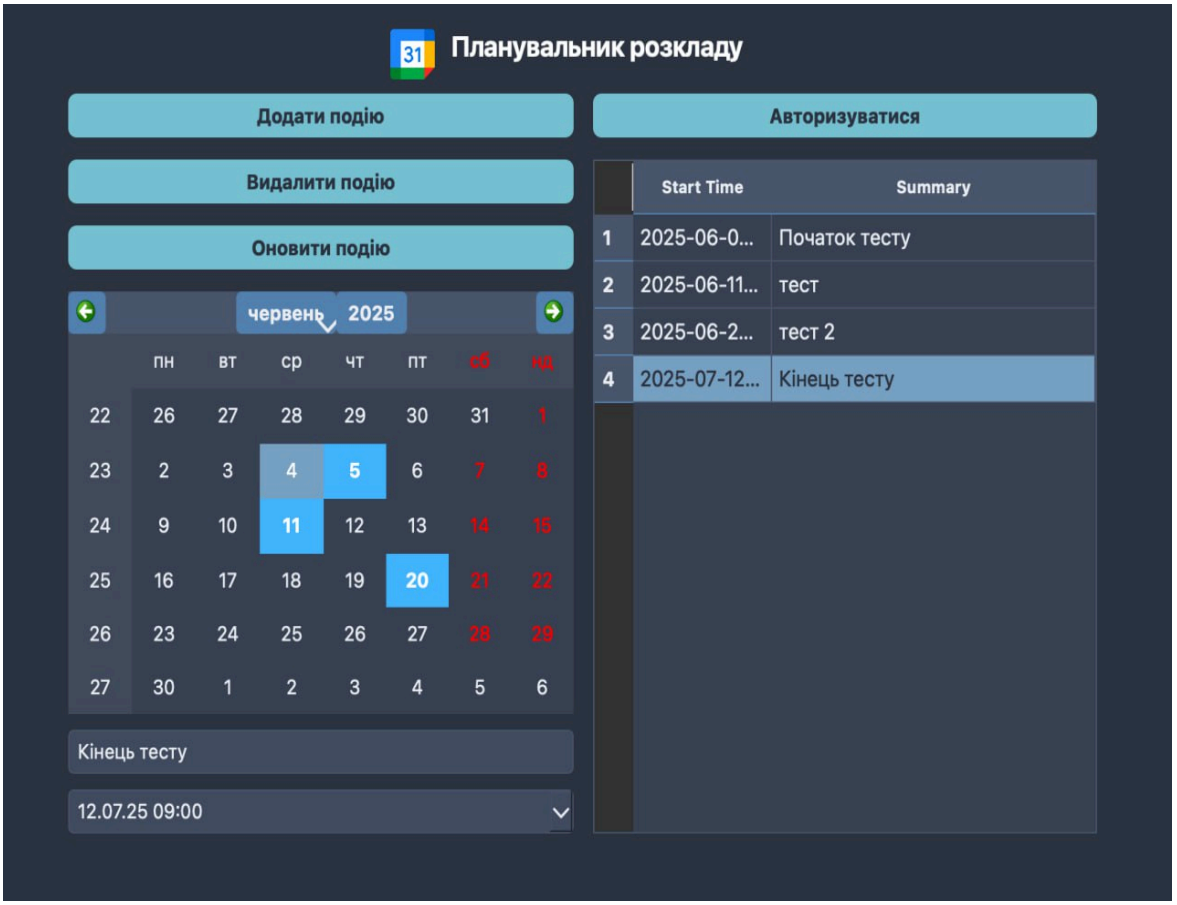


Рис. 1.4.2.2 – Авторизація через Google.

Користувач може додати нову подію, ввівши її назву у відповідне поле та вибравши дату і час початку. Після цього достатньо натиснути кнопку «Додати подію». Подія буде збережена в обліковому записі Google і одразу з’явиться у таблиці.

Для редагування події користувач вибирає її у списку, вносить зміни до полів і натискає кнопку «Оновити подію». Зміни автоматично синхронізуються з Google Календарем. Аналогічно, подію можна видалити, натиснувши відповідну кнопку після її вибору зі списку.

Таким чином, програмна система забезпечує інтуїтивно зрозумілу взаємодію користувача з календарем Google, охоплюючи повний цикл керування подіями без потреби відкривати браузер або сторонні інтерфейси.

1.4.3 Верифікація програмної системи

Верифікація програмної системи полягає у перевірці відповідності її функціональних можливостей поставленим вимогам та очікуванням користувачів. У процесі розробки було проведено низку тестувань, спрямованих на забезпечення стабільності роботи, коректної взаємодії з API Google Calendar та правильного відображення інформації в інтерфейсі користувача.

Перевірка функціональності проводилася вручну, відповідно до сценаріїв, що охоплюють ключові операції з подіями — авторизацію, завантаження, створення, оновлення та видалення подій. Усі ці дії були протестовані у типовому середовищі на Windows 10 із попередньо встановленим Qt середовищем розробки. Після успішної авторизації користувача, застосунок стабільно отримує доступ до календаря, завантажує список подій та забезпечує керування ними через графічний інтерфейс.

Зокрема, було підтверджено:

1. Коректність обробки відповіді від Google Calendar API;
2. Відповідність форматів дат та часу у подіях;
3. Стабільність додавання та оновлення подій із автоматичним оновленням таблиці;
4. Правильне збереження змін і видалення подій.

Таким чином, верифікація показала, що програмна система виконує всі функції, передбачені технічним завданням, і є придатною до використання кінцевими користувачами.

2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 План тестування

Тестування є ключовим етапом у життєвому циклі програмного забезпечення, що дозволяє переконатися в коректності роботи програмної системи відповідно до заданих вимог. Метою тестування є виявлення можливих помилок, перевірка надійності, продуктивності та зручності використання розробленого застосунку. У процесі тестування особлива увага приділяється взаємодії з Google Calendar API, обробці введених даних та стабільності інтерфейсу користувача.

План тестування визначає стратегію перевірки програмної системи, включаючи цілі тестування, типи тестів, критерії приймання, а також середовище та інструменти, які будуть використовуватись у процесі.

Основна мета тестування — забезпечити впевненість у тому, що система працює стабільно, коректно обробляє введення користувача та взаємодіє з віддаленими сервісами згідно з очікуваннями. Тестування охоплює як функціональні, так і нефункціональні вимоги.

У межах даного проєкту було заплановано виконання наступних типів тестування:

1. Модульне тестування, що охоплює окремі функції обробки подій (створення, оновлення, видалення);
2. Інтеграційне тестування взаємодії між компонентами інтерфейсу та Google Calendar API;
3. Тестування користувацького інтерфейсу, спрямоване на перевірку логіки керування подіями через графічну оболонку;
4. Регресійне тестування, яке виконується після внесення змін до коду, щоб впевнитися, що вже реалізовані функції залишаються працездатними.

Тестування проводиться в операційній системі Windows 10/11 з попередньо встановленим Qt 6, використовуючи реальний обліковий запис Google для

авторизації. Усі результати тестів фіксуються вручну та аналізуються для прийняття рішення про готовність системи до розгортання.

2.2 Розробка тестів

Розробка тестів передбачає створення конкретних тест-кейсів, які дозволяють перевірити функціональність усіх основних можливостей програмної системи. Для цього було сформульовано набір сценаріїв, кожен з яких перевіряє певну функцію: авторизацію, додавання, оновлення, видалення та відображення подій календаря.

Тестування проводилось вручну шляхом взаємодії з інтерфейсом користувача. Основна увага приділялась тому, як система поводиться за нормальних умов, а також у випадках некоректного введення чи мережевих помилок.

Таблиця 1 – тест-кейси

№	Назва тесту	Вхідні дані	Очікуваний результат	Статус
1	Авторизація через Google	Дійсний обліковий запис Google	Отримано токен доступу, події завантажено.	Пройдено
2	Додавання нової події	Назва: "Зустріч", час: 12:00 UTC	Подію додано до календаря та відображено в таблиці.	Пройдено
3	Видалення події	Обрана подія	Подію видалено з календаря та таблиці.	Пройдено
4	Оновлення події	Нова назва та час	Подію оновлено в Google Calendar.	Пройдено

Продовження таблиці 1

5	Відображення подій після авторизації	-	Завантажено та відображено останні події.	Пройдено
6	Обробка порожніх полів	Назва: (порожня), час: задано	Подію не додано, виведено попередження.	Пройдено
7	Авторизація скасована користувачем	Вихід із браузера	Доступ не надано, відображено повідомлення	Пройдено
8	Спроба дій без авторизації	Додати/видалити подію	Дії заблоковані або не виконуються	Пройдено

Для забезпечення надійності програмної системи було розроблено декілька юніт-тестів з використанням модуля Qt Test. Основною метою тестування є перевірка правильності виконання базових функцій програми, таких як відображення подій, додавання нових подій, робота з введенням користувача та оновленням інтерфейсу.

```cpp
#include <QtTest>
#include "mainwindow.h"

class TestMainWindow : public QObject {
    Q_OBJECT

private slots:
    void testUpdateEventTable();
};

void TestMainWindow::testUpdateEventTable() {
    MainWindow w;
    QList<CalendarEvent> testEvents = {
        {"Тестова подія", "2025-06-01T12:00:00Z"}
    };
    w.updateEventTable(testEvents);

    QCOMPARE(w.findChild<QTableWidget*>("eventTable")->rowCount(), 1);
    QCOMPARE(w.findChild<QTableWidget*>("eventTable")->item(0, 1)->text(), QString("Тестова подія"));
}

QTEST_MAIN(TestMainWindow)
#include "test_mainwindow.moc"
```

Рис. 2.2.1 – Тестування методу updateEventTable.

Інший важливий аспект — перевірка, чи слот onAddEventButtonClicked() працює правильно, зокрема якщо в поля введення вносяться некоректні або неповні дані.

```
void TestMainWindow::testAddEventValidation() {
    MainWindow w;
    w.show();


    w.findChild<QLineEdit*>("titleEdit")->setText("");
    w.findChild<QDateTimeEdit*>("startDateTimeEdit")->setDateTime(QDateTime::currentDateTime());

    QSignalSpy spy(w.findChild<QPushButton*>("addEventButton"), &QPushButton::clicked);
    QTest::mouseClick(w.findChild<QPushButton*>("addEventButton"), Qt::LeftButton);


    QVERIFY(spy.count() == 1);
}

```

Рис. 2.2.2 – Тестування методу onAddEventButtonClicked.

Окрім базових тестів функцій, було реалізовано тест для перевірки роботи інтерфейсу після кліку на таблицю подій:

```
void TestMainWindow::testTableClickFillFields() {
    MainWindow w;
    QList<CalendarEvent> testEvents = {
        {"Зустріч", "2025-06-03T14:00:00Z"}
    };
    w.updateEventTable(testEvents);
    w.show();

    QTableWidget* table = w.findChild<QTableWidget*>("eventTable");
    QTest::mouseClick(table->viewport(), Qt::LeftButton, Qt::NoModifier, table->visualItemRect(table->item(0, 0)).center());

    QCOMPARE(w.findChild<QLineEdit*>("titleEdit")->text(), QString("Зустріч"));
}

```

Рис. 2.2.2 – Тест для перевірки роботи інтерфейсу

2.3 Тестування продуктивності

Тестування продуктивності є важливим етапом оцінювання якості програмної системи, оскільки дозволяє виявити можливі вузькі місця у швидкодії,

зручності користування та адаптивності до різних середовищ. У рамках цього етапу було перевірено ключові аспекти роботи програми, пов’язані з її швидкодією та стабільністю.

Перш за все, було проведено вимірювання часу авторизації користувача через Google OAuth 2.0. У середньому, при стабільному підключенні до Інтернету, процес авторизації займає від 2 до 5 секунд. Основний вплив на швидкість має не програма, а зовнішні чинники, такі як швидкість мережі та відповідь серверів Google.

Далі було протестовано швидкість завантаження подій з Google Календаря після успішної авторизації. Програма виконує запит до API, отримує дані та відображає їх у таблиці. У випадку, коли календар містить 10–50 подій, завантаження займає в середньому 1–2 секунди. Якщо кількість подій більша, час зростає, однак залишається в межах прийнятного (до 3 секунд при 100+ подіях).

Оцінено також чутливість інтерфейсу до дій користувача. Після натискання кнопок (наприклад, додати або видалити подію) реакція програми є миттєвою — зміни відображаються у таблиці вже через декілька мілісекунд після виконання відповідної дії. Завдяки використанню механізму сигналів і слотів у Qt, інтерфейс залишається чутливим навіть при виконанні фонових операцій.

Програму також протестовано на кількох машинах з різними характеристиками. На сучасних комп’ютерах із процесором Intel Core i5/i7 і 8–16 ГБ оперативної пам’яті жодних затримок або підвисань не виявлено. На старіших системах із 4 ГБ оперативної пам’яті та слабшим процесором програма працює стабільно, хоча дещо помітне сповільнення в часі рендерингу таблиці при великій кількості подій.

Загалом, результати тестування продуктивності свідчать про високу швидкодію системи, стабільну роботу в реальному часі та хорошу адаптивність до різних конфігурацій апаратного забезпечення.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Застосування ризик орієнтованого підходу для побудови імовірних структурно-логічних моделей виникнення та розвитку надзвичайних ситуацій

Ризик-орієнтований підхід є ключовим елементом сучасної системи забезпечення безпеки життєдіяльності, оскільки він дозволяє перейти від реактивної до проактивної моделі реагування на надзвичайні ситуації (НС). Його суть полягає у визначенні, кількісній та якісній оцінці ризиків, прогнозуванні можливих сценаріїв розвитку подій, а також у прийнятті управлінських рішень на основі моделювання ймовірності виникнення і наслідків НС, [18, с. 35–36].

Ризик — кількісна оцінка небезпеки, це відношення числа тих чи інших небажаних реалізованих наслідків (n), до максимально можливої їх кількості (N) за конкретний період часу: $R = n/N$, тобто це частота реалізації небезпечностей. Він є супутником будь-якої діяльності людини, [18, с. 36].

Зазначене співвідношення дозволяє обчислювати як індивідуальний, так і груповий або загальний ризик: «Наведена формула дозволяє розрахувати розміри індивідуального, групового та загального ризику. При оцінці загального ризику величина N визначає максимальну кількість усіх подій, а при оцінці групового ризику — максимальну кількість подій у конкретній соціальній групі…», [18, с. 36].

Застосування цього підходу охоплює кілька важливих етапів:

1. Ідентифікація небезпек — виявлення чинників, що можуть призвести до НС, з урахуванням внутрішніх і зовнішніх джерел небезпеки (техногенні, природні, соціальні), [18, с. 33–34].
2. Аналіз ризиків — визначення ймовірності виникнення певної ситуації та її потенційних наслідків, тобто поєднання інтенсивності впливу і частоти його реалізації, [18, с. 35].
3. Побудова структурно-логічних моделей — графічне або алгоритмічне відображення розвитку подій, що дозволяє простежити

причинно-наслідкові зв’язки між різними елементами системи та їх вплив на масштаб наслідків, [18, с. 36].

Особливу цінність ризик-орієнтовані моделі мають у прогнозуванні надзвичайних ситуацій. Прийняття управлінських рішень базується на використанні матриць оцінки ризику, які поєднують частоту подій і категорії небезпеки. Це дозволяє класифікувати ризики як неприйнятні, небажані, припустимі з перевіркою та припустимі без перевірки, [19, с. 44]. За ступенем припустимості ризик поділяється на знехтуваний, прийнятний, граничнодопустимий і надмірний. Прийнятний ризик розглядається як компроміс між рівнем безпеки й можливостями його досягнення, враховуючи техніко-економічні та соціальні аспекти, [19, с. 42].

Концепція прийнятного ризику, що базується на досягненні прийнятного рівня безпеки, забезпечує баланс між витратами на підвищення безпеки технічних систем і соціально-економічними наслідками. Вона враховує рівень життя, соціально-економічне становище та розвиток науки і техніки , [19, с. 43, с. 44].

Аналіз небезпек починають з попереднього дослідження, яке дозволяє ідентифікувати джерела небезпек. Потім, при необхідності, дослідження можуть бути поглиблені і виконаний детальний якісний аналіз. Серед методів використовуються: аналіз дерева помилок, аналіз дерева подій, причинно-наслідковий аналіз, [19. с. 51].

Таким чином, ризик-орієнтований підхід сприяє створенню ефективних механізмів для запобігання та мінімізації наслідків надзвичайних ситуацій, забезпечуючи інтеграцію наукових методів моделювання ризиків у практичне управління безпекою життєдіяльності.

3.2 Заходи щодо забезпечення безпеки при проведенні дослідних робіт.

Проведення дослідних робіт пов’язане з низкою виробничих ризиків, що можуть виникнути через використання технічного обладнання, електроприладів, випромінювань, хімічних речовин тощо. Для збереження життя і здоров’я учасників досліджень необхідно впроваджувати комплекс заходів, спрямованих на створення безпечних умов праці.

Одним із ключових елементів організації безпечного дослідного процесу є проведення інструктажів з охорони праці: За характером і часом проведення інструктажі з питань охорони праці поділяються на вступний, первинний, повторний, позаплановий та цільовий.

Вступний інструктаж проводиться:

1. З усіма працівниками, яких приймають на постійну або тимчасову роботу.
2. З учнями та студентами, які прибули на підприємство для проходження виробничої практики , [20, с. 44–45].

Це особливо важливо в умовах, коли до виконання робіт залучаються студенти або стажери, які не мають достатнього практичного досвіду.

У випадках, коли технічні або організаційні заходи не забезпечують повної безпеки, застосовуються засоби індивідуального захисту (ЗІЗ):

«Засіб індивідуального захисту (ЗІЗ) — це засіб захисту, що надягається на тіло працівника або його частину, або використовується під час праці. ЗІЗ застосовують тоді, коли безпека робіт не може бути забезпечена конструкцією та розміщенням устаткування, організацією виробничих процесів…» , [20, с. 137-138].

Це можуть бути: захисні окуляри, рукавиці, халати, маски, респіратори, каски тощо.

При використанні джерел небезпечного випромінювання, наприклад лазерів, обов’язковими є колективні та індивідуальні заходи безпеки. Зокрема:

1. Огороджування зон можливого поширення лазерного випромінювання (прямого, розсіяного, відбитого).
2. Екранування променя лазера
3. Встановлення на лазерній установці блокувальних засобів та сигналізації початку та закінчення роботи лазера

До засобів індивідуального захисту від лазерного випромінювання належать захисні окуляри із світлофільтрами, маски, щитки, халати, рукавички, [20, с. 137].

Приміщення, у яких проводяться дослідні роботи, повинні відповідати вимогам нормативів щодо мікроклімату, освітлення, вентиляції та санітарної безпеки. Наприклад: Підлоги виробничих приміщень повинні бути зносостійкими, теплими, неслизькими, щільними, легко очищуватись, а в деяких цехах та дільницях — волого-, кислото- та вогнестійкими. [20, с. 145] Об'єм виробничого приміщення на одного працівника згідно з санітарними нормами повинен складати не менше 15 м³, а площа приміщення — не менше 4,5 м², [20, с. 144].

На завершення слід наголосити, що кожен учасник досліджень повинен бути ознайомлений із конкретною інструкцією з охорони праці, яка передбачає порядок дій до, під час та після завершення робіт, а також у разі виникнення аварійних ситуаці. Кожній інструкції з охорони праці присвоюється назва та скорочене позначення… Вона повинна містити такі розділи: загальні положення; вимоги безпеки перед початком роботи; під час виконання роботи; після закінчення роботи; в аварійних ситуаціях, [20, с. 30].

Таким чином, комплекс заходів безпеки при проведенні дослідних робіт охоплює: інструктажі, використання ЗІЗ, дотримання технічних вимог до приміщень, спеціальні засоби захисту при роботі з джерелами небезпеки, а також чітку регламентацію дій за інструкціями. Впровадження цих заходів є запорукою збереження здоров’я і працездатності працівників наукової сфери.

ВИСНОВКИ

У процесі виконання дипломної роботи було реалізовано повнофункціональну десктопну програмну систему для перегляду, додавання, редагування та видалення подій з особистого календаря користувача за допомогою інтеграції з Google Calendar API. Створене програмне забезпечення дозволяє користувачеві ефективно керувати своїм розкладом, не вдаючись до використання браузера чи сторонніх вебінтерфейсів, що значно спрощує процес планування та робить його більш зручним у повсякденному житті.

У першому розділі було проведено детальний аналіз предметної області, що дозволило глибше зрозуміти сучасні вимоги до систем планування. Було визначено основну мету та завдання проєкту, сформульовано функціональні та нефункціональні вимоги до програмного забезпечення, обґрунтовано вибір технічних засобів і методів реалізації. Побудовані UML-діаграми, включаючи діаграму класів та загальну архітектуру системи, дали змогу структуровано підійти до проектування майбутнього застосунку та забезпечити логічну цілісність його компонентів.

У другому розділі розглянуто процес реалізації системи. Як основну мову програмування обрано C++, завдяки її високій продуктивності, широким можливостям та гнучкості. Для створення графічного інтерфейсу застосовано фреймворк Qt, який дозволив реалізувати інтуїтивно зрозумілий, кросплатформений та адаптивний інтерфейс користувача. Важливою частиною розробки стала реалізація взаємодії з Google Calendar API. Для цього було використано компоненти QOAuth2AuthorizationCodeFlow для безпечної авторизації користувача та QNetworkAccessManager для надсилання запитів і обробки відповідей від хмарного сервісу. Архітектура додатка побудована на основі принципів об'єктно-орієнтованого програмування, що дозволяє легко масштабувати систему в майбутньому та додавати нові функціональні можливості.

У розділі тестування було розроблено план перевірки функціональності системи та створено низку тестових сценаріїв, які охоплюють основні дії користувача: авторизацію, перегляд, створення, редагування та видалення подій. Тестування проводилось на різних конфігураціях персональних комп’ютерів, що дозволило оцінити стабільність роботи програмного забезпечення в різних умовах. Отримані результати підтвердили відповідність системи технічним вимогам, її працездатність і стійкість до типових помилок користувача.

Окрему увагу було приділено зручності використання системи кінцевим користувачем. Графічний інтерфейс реалізовано в простій та доступній формі, що забезпечує швидку навігацію та мінімальну кількість дій для виконання основних операцій. Завдяки цьому навіть користувачі без глибоких технічних знань зможуть ефективно користуватися додатком. Інтерфейс дозволяє легко переглядати розклад, додавати події, редагувати їх або видаляти, забезпечуючи при цьому повну синхронізацію з обліковим записом Google.

Таким чином, мету дипломної роботи досягнуто повністю. Було створено зручний і надійний програмний засіб для організації особистого часу з використанням сучасних технологій. Система має практичну цінність і може бути використана широким колом користувачів для щоденного планування подій. Вона є хорошою основою для подальшого розвитку, зокрема впровадження підтримки кількох календарів, офлайн-режиму, синхронізації з іншими хмарними сервісами, використання системи нагадувань, а також створення мобільної версії програми. У майбутньому проєкт може бути доповнений аналітичними функціями — статистикою подій, візуалізацією завантаженості по днях і автоматичними рекомендаціями щодо планування часу. Завдяки розширенню функціональності, така система зможе перетворитися на комплексний інструмент управління особистим часом для широкої аудиторії користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Qt Documentation. [Електронний ресурс]. – Режим доступу: https://doc.qt.io

2. Google Calendar API Overview. [Електронний ресурс]. – Режим доступу: https://developers.google.com/calendar

3. OAuth 2.0 for Client-side Web Applications. [Електронний ресурс]. – Режим доступу: https://developers.google.com/identity/protocols/oauth2

4. C++ Language Reference. [Електронний ресурс]. – Режим доступу: https://en.cppreference.com

5. SQLite Documentation. [Електронний ресурс]. – Режим доступу: https://www.sqlite.org/docs.html

6. CMake Documentation. [Електронний ресурс]. – Режим доступу: https://cmake.org/documentation

7. REST API Design Guide. [Електронний ресурс]. – Режим доступу: https://restfulapi.net

8. JSON Format Standard. [Електронний ресурс]. – Режим доступу: https://www.json.org/json-en.html

9. Програмування з використанням Qt: офіційний посібник. [Електронний ресурс]. – Режим доступу: https://wiki.qt.io/Main

10. Google Cloud Identity Platform Documentation. [Електронний ресурс]. – Режим доступу: https://cloud.google.com/identity-platform/docs

11. Qt Network Authorization Module. [Електронний ресурс]. – Режим доступу: https://doc.qt.io/qt-6/qtnetworkauth-index.html

12. Signals and Slots in Qt. [Електронний ресурс]. – Режим доступу: https://doc.qt.io/qt-6/signalsandslots.html

13. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion Mykhaylo Petryk, Vitalii Chyzh, Halyna

Tsupryk, Oksana Petryk, ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23- 25, 2024, Ternopil, Ukraine, Opole, Poland, 2024, pp. 486-493 (Scopus) https://ceur-ws.org/Vol-3896/

14. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90. (фахове видання України)

15. Яворська, ЄБ, Кінаш, РВ, Цуприк, ГБ, Николайчук ВІ Проблеми виявлення біосигналів у медичних інформаційних системах/ЄБ Яворська, Кінаш РВ, ГБ Цуприк, ВІ Николайчук//IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині»(ІСМ–2021)[Текст]: зб. наук. пр.–Харків: Нац. аерокосм. ун-т ім. МЄ Жуковського «Харків. авіац. ін-т», 2021.–25-26 с.

16. Остапчук О., Цуприк Г. Технічні особливості взаємодії між клієнтом та сервером у реальному часі // Матеріали X науково-технічної конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя. – ТНТУ, 2022. – С. 124. – [Електронний ресурс]. – Режим доступу: https://elartu.tntu.edu.ua/handle/lib/40231

17. Доскач Н., Цуприк Г. Розробка системи забезпечення безпечного обміну інформацією з використанням технологій веб програмування // Матеріали X науково-технічної конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя. – ТНТУ, 2022. – С. 114. [Електронний ресурс]. – Режим доступу: https://elartu.tntu.edu.ua/handle/lib/40231

18. Атаманчук П. С., Мендерецький В. В., Панчук О. П., Чорна О. Г. Безпека життєдіяльності : навчальний посібник. — Київ : Центр учбової літератури, 2011. — 276 с.

19. Желібо Є.П., Заверуха Н.М., Зацарний В.В. Безпека життєдіяльності : навч. посіб. / за ред. Є.П. Желібо. – 6-е вид. – К. : Каравела, 2008. – 344 с.

20. Жидецький В. Ц. Основи охорони праці : підручник. — Львів : Афіша, 2004. — 350 с.

21. Типове положення про порядок проведення навчання та перевірки знань з питань охорони праці: НПАОП 0.00-4.12-05 / затв. наказом Держнаглядохоронпраці України від 26.01.2005 р. № 15; із змінами згідно з наказом Держпраці України від 30.01.2017 р. № 140. – Офіц. вид. – Київ: Держпраці України, 2017. – 44 с.

22. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

Додаток А: Структура проєкту

Лістинг коду файлу mainwindow.h :

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QOAuth2AuthorizationCodeFlow>
#include <QNetworkAccessManager>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

struct CalendarEvent {
    QString summary;
    QString startTime;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void onAuthorizeButtonClicked();
    void onAuthorized();
    void onAddEventButtonClicked();
    void onDeleteEventButtonClicked();
    void onUpdateEventButtonClicked();
    void onEventTableClicked();
    void updateCalendarWithEvents();
```

```
    void onCalendarDateClicked(const QDate &date);


private:
    void updateEventTable(const QList<CalendarEvent> &events);
    void clearInputs();

    Ui::MainWindow *ui;
    QOAuth2AuthorizationCodeFlow *oauth;
    QNetworkAccessManager *networkManager;

    QList<QPair<QString, CalendarEvent>> eventListWithIds;
};

#endif // MAINWINDOW_H
```

Лістинг коду файлу mainwindow.cpp :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QUrl>
#include <QUrlQuery>
#include <QDesktopServices>
#include <QOAuthHttpServerReplyHandler>
#include <QNetworkReply>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QTableWidgetItem>
#include <QDateTime>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , oauth(new QOAuth2AuthorizationCodeFlow(this))
    , networkManager(new QNetworkAccessManager(this))
{
    ui->setupUi(this);
```

```
        // Заголовок + логотип
        QPixmap logo(":/img/logo.png"); // шлях у ресурсі
                logo   =   logo.scaled(32,   32,   Qt::KeepAspectRatio,
Qt::SmoothTransformation); // за потреби

        QString titleHtml = "<table><tr>"
                            "<td><img src=\":/img/logo.png\" width='32'
height='32'></td>"
                               "<td style='color:white; font-size:18pt;
font-weight:bold; padding-left:10px;'>Планувальник розкладу</td>"
                            "</tr></table>";

        ui->titleLabel->setText(titleHtml);


oauth->setAuthorizationUrl(QUrl("https://accounts.google.com/o/oauth2/aut
h"));

oauth->setAccessTokenUrl(QUrl("https://oauth2.googleapis.com/token"));

oauth->setClientIdentifier("246209458791-fcob0je8feponk14q8cknjv2ismuivm3
.apps.googleusercontent.com");

oauth->setClientIdentifierSharedKey("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"
);
        oauth->setScope("https://www.googleapis.com/auth/calendar");

           auto  replyHandler  =  new  QOAuthHttpServerReplyHandler(8080,
this);
        oauth->setReplyHandler(replyHandler);

                                                        connect(oauth,
&QOAuth2AuthorizationCodeFlow::authorizeWithBrowser,
                    [](const QUrl &url) { QDesktopServices::openUrl(url);
});

        connect(oauth, &QOAuth2AuthorizationCodeFlow::granted,
                this, &MainWindow::onAuthorized);
```

```
        connect(ui->authorizeButton, &QPushButton::clicked,
                this, &MainWindow::onAuthorizeButtonClicked);

        connect(ui->addEventButton, &QPushButton::clicked,
                this, &MainWindow::onAddEventButtonClicked);

        connect(ui->deleteEventButton, &QPushButton::clicked,
                this, &MainWindow::onDeleteEventButtonClicked);

        connect(ui->updateEventButton, &QPushButton::clicked,
                this, &MainWindow::onUpdateEventButtonClicked);

        connect(ui->eventTable, &QTableWidget::cellClicked,
                this, &MainWindow::onEventTableClicked);

        ui->eventTable->setColumnCount(2);
             ui->eventTable->setHorizontalHeaderLabels(QStringList()  <<
"Start Time" << "Summary");
        ui->eventTable->horizontalHeader()->setStretchLastSection(true);
        QString style = R"(
    /* === Загальне оформлення === */
    QMainWindow {
        background-color: #2E3440;
    }

    /* === Текстові елементи === */
    QLabel, QCheckBox, QRadioButton {
        color: #D8DEE9;
        font-size: 14px;
    }

    /* === Поля вводу === */
    QLineEdit, QDateTimeEdit, QTextEdit {
        background-color: #434C5E;
        color: #ECEFF4;
        border: 1px solid #4C566A;
        border-radius: 4px;
        padding: 4px;
    }
```

```
/* === Календар === */
QCalendarWidget QAbstractItemView {
    selection-background-color: #81A1C1;
    background-color: #3B4252;
    color: #ECEFF4;
}
QCalendarWidget QWidget#qt_calendar_navigationbar {
    background-color: #4C566A;
    color: #ECEFF4;
}
QCalendarWidget QToolButton {
    background-color: #5E81AC;
    color: #ECEFF4;
    border-radius: 4px;
    padding: 4px;
}
QCalendarWidget QToolButton:hover {
    background-color: #81A1C1;
}

/* === Таблиця подій === */
QTableWidget {
    background-color: #3B4252;
    alternate-background-color: #434C5E;
    color: #ECEFF4;
    gridline-color: #4C566A;
    border: 1px solid #4C566A;
}
QHeaderView::section {
    background-color: #4C566A;
    color: white;
    padding: 4px;
    border: none;
}

/* === Кнопки === */
QPushButton {
    background-color: #88C0D0;
```

```
    color: #2E3440;
    border-radius: 6px;
    font-weight: bold;
    padding: 6px 12px;
}
QPushButton:hover {
    background-color: #81A1C1;
}
QPushButton:pressed {
    background-color: #5E81AC;
}

/* === ComboBox === */
QComboBox {
    background-color: #434C5E;
    color: #ECEFF4;
    border: 1px solid #4C566A;
    border-radius: 4px;
    padding: 4px;
}
QComboBox QAbstractItemView {
    background-color: #3B4252;
    color: #ECEFF4;
    selection-background-color: #81A1C1;
}

/* === TabWidget === */
QTabWidget::pane {
    border: 1px solid #4C566A;
    background-color: #3B4252;
}
QTabBar::tab {
    background: #4C566A;
    color: #ECEFF4;
    padding: 6px;
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
}
QTabBar::tab:selected {
```

```
    background: #5E81AC;
}

/* === GroupBox === */
QGroupBox {
    color: #D8DEE9;
    border: 1px solid #5E81AC;
    border-radius: 4px;
    margin-top: 10px;
}
QGroupBox::title {
    subcontrol-origin: margin;
    left: 10px;
    padding: 0 4px;
}

/* === Scrollbars === */
QScrollBar:vertical, QScrollBar:horizontal {
    background: #3B4252;
    border: none;
    width: 10px;
    margin: 2px;
}
QScrollBar::handle {
    background: #5E81AC;
    border-radius: 4px;
}
QScrollBar::handle:hover {
    background: #81A1C1;
}

/* === QDialog === */
QDialog {
    background-color: #2E3440;
    color: #D8DEE9;
}

/* === ToolTip === */
QToolTip {
```

```
    background-color: #4C566A;
    color: #ECEFF4;
    border: 1px solid #81A1C1;
    padding: 4px;
    border-radius: 4px;
}
QTableWidget {
    background-color: #3B4252;
    alternate-background-color: #434C5E;
    color: #ECEFF4;
    gridline-color: #4C566A;
    border: 1px solid #4C566A;
    selection-background-color: #81A1C1;
    selection-color: #2E3440;
    font-size: 14px;
}

/* Вирівняння вмісту клітинок */
QTableWidget::item {
    padding: 6px;
}

/* Hover ефект */
QTableWidget::item:hover {
    background-color: #5E81AC;
    color: white;
}

/* === Заголовки колонок === */
QHeaderView::section {
    background-color: #4C566A;
    color: #ECEFF4;
    padding: 6px;
    font-weight: bold;
    border: none;
    border-bottom: 1px solid #5E81AC;
}
```

```
    )";



        this->setStyleSheet(style);
    }

    MainWindow::~MainWindow()
    {
        delete ui;
    }

    void MainWindow::onAuthorizeButtonClicked()
    {
        oauth->grant();
    }

    void MainWindow::onAuthorized()
    {
                                                                    QUrl
url("https://www.googleapis.com/calendar/v3/calendars/primary/events");
        QUrlQuery query;
        query.addQueryItem("maxResults", "20");
        query.addQueryItem("orderBy", "startTime");
        query.addQueryItem("singleEvents", "true");
                                            query.addQueryItem("timeMin",
QDateTime::currentDateTimeUtc().toString(Qt::ISODate));
        url.setQuery(query);

        QNetworkRequest request(url);
                 request.setRawHeader("Authorization",   "Bearer   "   +
oauth->token().toUtf8());

        QNetworkReply* reply = networkManager->get(request);

        connect(reply, &QNetworkReply::finished, this, [this, reply]() {
            if (reply->error() != QNetworkReply::NoError) {
                            qDebug()  <<  "Error  fetching  events:"  <<
reply->errorString();
```

```
                reply->deleteLater();
                return;
            }

            QByteArray response = reply->readAll();
            QJsonDocument doc = QJsonDocument::fromJson(response);
            QJsonArray items = doc.object()["items"].toArray();

            eventListWithIds.clear();
            QList<CalendarEvent> events;

            for (const QJsonValue& val : items) {
                QJsonObject obj = val.toObject();
                QString id = obj["id"].toString();
                QString summary = obj["summary"].toString("(No title)");
                                                    QString    startTime    =
obj["start"].toObject().contains("dateTime")
                                                                            ?
obj["start"].toObject()["dateTime"].toString()
                                                                            :
obj["start"].toObject()["date"].toString();

                                        eventListWithIds.append(qMakePair(id,
CalendarEvent{summary, startTime}));
                events.append({summary, startTime});
            }

            updateEventTable(events);

            // Додаємо оновлення календаря після завантаження подій
            updateCalendarWithEvents();

            reply->deleteLater();
        });
    }

    void     MainWindow::updateEventTable(const     QList<CalendarEvent>
&events)
    {
```

```
        ui->eventTable->setRowCount(events.size());
        for (int i = 0; i < events.size(); ++i) {
                                    ui->eventTable->setItem(i,    0,    new
QTableWidgetItem(events[i].startTime));
                                    ui->eventTable->setItem(i,    1,    new
QTableWidgetItem(events[i].summary));
        }
    }

    void MainWindow::onAddEventButtonClicked()
    {
        QString summary = ui->titleEdit->text();
        QDateTime startTime = ui->startDateTimeEdit->dateTime().toUTC();

        if (summary.isEmpty() || !startTime.isValid()) return;

        QJsonObject event;
        event["summary"] = summary;

        QJsonObject start;
        start["dateTime"] = startTime.toString(Qt::ISODate);
        start["timeZone"] = "UTC";
        event["start"] = start;

        QJsonObject end;
        end["dateTime"] = startTime.addSecs(3600).toString(Qt::ISODate);
        end["timeZone"] = "UTC";
        event["end"] = end;

                                                    QNetworkRequest
request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/ev
ents"));
                    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json");
                  request.setRawHeader("Authorization",   "Bearer   "   +
oauth->token().toUtf8());

                QNetworkReply*   reply   =   networkManager->post(request,
QJsonDocument(event).toJson());
```

```
        connect(reply, &QNetworkReply::finished, this, [this, reply]() {
            reply->deleteLater();
            if (reply->error() == QNetworkReply::NoError) {
                qDebug() << "Подія додана!";
                onAuthorized();
            } else {
                        qDebug() << "Помилка при додаванні події:" <<
reply->errorString();
            }
        });
    }

    void MainWindow::onDeleteEventButtonClicked()
    {
        int row = ui->eventTable->currentRow();
        if (row < 0 || row >= eventListWithIds.size()) return;

        QString eventId = eventListWithIds[row].first;

                                                    QNetworkRequest
request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/ev
ents/" + eventId));
                request.setRawHeader("Authorization",   "Bearer   "   +
oauth->token().toUtf8());

        QNetworkReply* reply = networkManager->deleteResource(request);
        connect(reply, &QNetworkReply::finished, this, [this, reply]() {
            reply->deleteLater();
            if (reply->error() == QNetworkReply::NoError) {
                qDebug() << "Подію видалено!";
                onAuthorized();
            } else {
                              qDebug()   <<   "Помилка   видалення:"   <<
reply->errorString();
            }
        });
    }
```

```
void MainWindow::onUpdateEventButtonClicked()
{
    int row = ui->eventTable->currentRow();
    if (row < 0 || row >= eventListWithIds.size()) return;

    QString eventId = eventListWithIds[row].first;
    QString summary = ui->titleEdit->text();
    QDateTime startTime = ui->startDateTimeEdit->dateTime().toUTC();

    if (summary.isEmpty() || !startTime.isValid()) return;

    QJsonObject event;
    event["summary"] = summary;

    QJsonObject start;
    start["dateTime"] = startTime.toString(Qt::ISODate);
    start["timeZone"] = "UTC";
    event["start"] = start;

    QJsonObject end;
    end["dateTime"] = startTime.addSecs(3600).toString(Qt::ISODate);
    end["timeZone"] = "UTC";
    event["end"] = end;

                                                    QNetworkRequest
request(QUrl("https://www.googleapis.com/calendar/v3/calendars/primary/ev
ents/" + eventId));
                    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json");
                request.setRawHeader("Authorization",   "Bearer   "   +
oauth->token().toUtf8());

                                        QNetworkReply*        reply        =
networkManager->sendCustomRequest(request,                        "PATCH",
QJsonDocument(event).toJson());
    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        reply->deleteLater();
        if (reply->error() == QNetworkReply::NoError) {
            qDebug() << "Подію оновлено!";
```

```
                onAuthorized();
            } else {
                                qDebug()  <<  "Помилка  оновлення:"  <<
reply->errorString();
            }
        });
    }

    void MainWindow::onEventTableClicked()
    {
        int row = ui->eventTable->currentRow();
        if (row < 0 || row >= eventListWithIds.size()) return;

        CalendarEvent event = eventListWithIds[row].second;
        ui->titleEdit->setText(event.summary);

ui->startDateTimeEdit->setDateTime(QDateTime::fromString(event.startTime,
Qt::ISODate));
    }
    void MainWindow::updateCalendarWithEvents()
    {
                            ui->calendarWidget->setDateTextFormat(QDate(),
QTextCharFormat());

        QTextCharFormat format;
        format.setBackground(QColor(100, 180, 255));  // ніжно-голубий
        format.setForeground(Qt::white);
        format.setFontWeight(QFont::Bold);
        format.setToolTip("Подія в цей день");

        for (const auto &event : eventListWithIds) {
             QDate date = QDateTime::fromString(event.second.startTime,
Qt::ISODate).date();
            if (date.isValid()) {
                qDebug() << "Highlighting date:" << date.toString();
                ui->calendarWidget->setDateTextFormat(date, format);
            }
        }
    }
```

```
void MainWindow::onCalendarDateClicked(const QDate &date)
{
    QList<CalendarEvent> eventsForDate;

    for (const auto &event : eventListWithIds) {
                                            QDate     eventDate     =
QDateTime::fromString(event.second.startTime, Qt::ISODate).date();
        if (eventDate == date) {
            eventsForDate.append(event.second);
        }
    }

    updateEventTable(eventsForDate);
}
```

Додаток Б: Ілюстрації варіантів використання системи

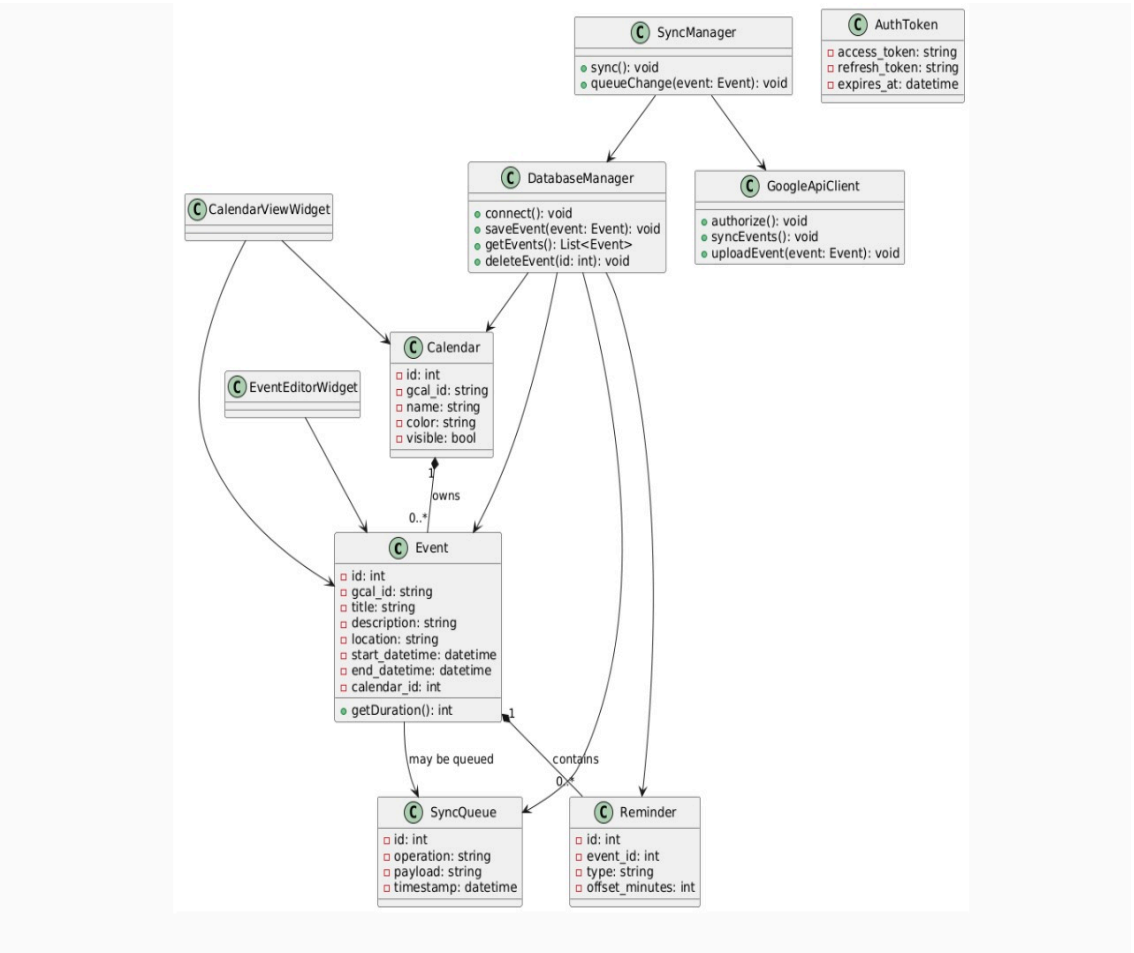


Рис. 1.3 – Діаграма класів системи.

Рис. 1.2 – Діаграма варіантів використання системи.

Додаток В: Публікація у науковій конференції

Zboriv, Ukraine, October 02-04, 2024. CEUR Workshop Proceedings, 2024, 3842, pp. 147-156.
2. Raspberry Pi Pico W. Arduino.ua. https://shorturl.at/CoKac (дата звернення 10.05.2025)
3. RPP-316 PD20W+QC22.5W. https://shorturl.at/aQxYF (дата звернення 11.05.2025)
4. Brown M., 2011. Power Sources and Supplies: World Class Designs. Newnes, 368 с.
5. Акумулятори 18650. https://shorturl.at/cmNGb (дата звернення 13.05.2025)

УДК 004.41

С.І. Подедвірний, Г.Б. Цуприк, к.т.н., доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ІНТЕГРАЦІЯ ДЕСКТОПНОГО РОЗКЛАДУ З GOOGLE CALENDAR ЗА ДОПОМОГОЮ C++/QT ТА GOOGLE API

S.I. Podedvirniy, H.B. Tsupryk, Ph.D., Assoc. Prof.

INTEGRATION OF DESKTOP WALLOW WITH GOOGLE CALENDAR USING C++/QT AND GOOGLE API

У сучасному цифровому середовищі персональне планування та управління часом набувають все більшого значення як у професійному, так і в особистому контексті. Хоча існує велика кількість вебсервісів для планування подій, багато користувачів виявляють потребу у функціональному десктопному застосунку, який би поєднував автономну роботу з можливістю хмарної синхронізації. Саме така концепція стала основою для створення програмного засобу — десктопного планувальника розкладу, розробленого із використанням кросплатформного фреймворку Qt (мова програмування C++) та інтегрованого з Google Calendar API.

Метою роботи стало створення зручного інтерфейсу для управління подіями, який дозволяє створювати, редагувати та видаляти події локально, а також здійснювати двосторонню синхронізацію з Google Calendar. Це дозволяє користувачеві мати доступ до своїх подій з будь-якого пристрою, водночас маючи стабільний інструмент для роботи в офлайн-режимі. Реалізація проєкту базується на поєднанні кількох ключових технологічних компонентів: Qt як інструменту для створення графічного інтерфейсу користувача, бібліотек для мережевої взаємодії (зокрема QNetworkAccessManager), та Google Calendar API, що дозволяє здійснювати запити до сервера за протоколом REST.

Особливу увагу було приділено реалізації процедури аутентифікації за допомогою протоколу OAuth 2.0. Цей етап передбачає відкриття сторінки авторизації в браузері, отримання коду доступу та подальше використання маркера доступу (access token) для виконання запитів до API. Таким чином, забезпечується безпечний обмін даними без потреби зберігати облікові дані користувача в системі. Комунікація із сервером Google Calendar здійснюється за допомогою HTTP-запитів (GET, POST, PATCH, DELETE), а результати обробляються у форматі JSON із використанням вбудованих можливостей Qt для роботи з цим форматом (класи QJsonDocument, QJsonObject тощо).

Архітектура застосунку реалізована за принципами Model-View-Controller (MVC), де логіка обробки подій відокремлена від відображення інтерфейсу. Це забезпечує зручність масштабування та супроводу коду. Дані календаря зберігаються у вигляді локального кешу з використанням формату JSON, що дозволяє забезпечити роботу в офлайн-режимі та синхронізувати зміни при відновленні підключення до Інтернету.

Інтерфейс користувача спроєктований відповідно до принципів простоти та ефективності. Основна панель дозволяє переглядати події за днем або тижнем, забезпечуючи зручну навігацію за допомогою календаря та візуалізації у вигляді таймлайнів або таблиці. Для кожної події можна задавати назву, опис, час початку та завершення, а також вибирати категорію, яка відображається різними кольорами. Передбачено інтерактивну навігацію,

гарячі клавіші, сповіщення про майбутні події, а також підтримку кількох облікових записів Google одночасно.

У процесі реалізації виникло кілька викликів, серед яких — правильна обробка токенів доступу та їх оновлення, обмеження на частоту запитів до API (rate limits), а також забезпечення стабільної роботи при нестабільному з'єднанні. Усі ці аспекти були враховані шляхом реалізації механізмів повторної спроби (retry), логування помилок і зручного інформування користувача про стан синхронізації.

Запропоноване рішення має переваги перед традиційними веб календарями, адже забезпечує повну функціональність навіть без доступу до Інтернету, має швидкодіюче нативне виконання, а також гнучкий інтерфейс, який можна адаптувати під конкретні потреби користувача. У порівнянні з існуючими аналогами, такими як Google Calendar Web, Microsoft Outlook чи Thunderbird Calendar, даний застосунок поєднує простоту десктопного інтерфейсу з перевагами хмарної інтеграції.

У перспективі розробка може бути доповнена синхронізацією з іншими сервісами календарів, а також інтеграцією з локальними базами даних (наприклад, SQLite) для зберігання історії змін та підтримки командної роботи. Можливе також впровадження штучного інтелекту для автоматичного розподілу подій або рекомендацій щодо оптимізації розкладу.

Таким чином, у роботі було створено повноцінний десктопний інструмент планування часу, який поєднує зручність локального використання з перевагами хмарної синхронізації. Результати свідчать про ефективність інтеграції C++/Qt із сучасними веб сервісами на прикладі Google Calendar API, що відкриває широкі перспективи для створення універсальних засобів персонального тайм-менеджменту, які відповідають вимогам сучасного користувача.

Література

1. Google Calendar API Overview | Calendar API | Google Developers. Google Developers. URL: https://developers.google.com/calendar (дата звернення: 19.05.2025).

2. RFC 6749: The OAuth 2.0 Authorization Framework / D. Hardt. IETF, 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 19.05.2025).

3. Qt Documentation | Application Development with Qt | Qt Company. URL: https://doc.qt.io/qt-6/ (дата звернення: 19.05.2025).

4. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

5. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

6. Яворська, Є.Б., Кінаш, Р.В., Цуприк, Г.Б., Николайчук В.І. Проблеми виявлення біосигналів у медичних інформаційних системах / Є.Б. Яворська, Кінаш Р.В., Г.Б. Цуприк, В.І. Николайчук // IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині» (ІСМ–2021) [Текст] : зб. наук. пр. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2021. – 25-26 с.

Додаток Г: Диск з роботою