Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота

магістра

з теми: «РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ “СТУДЕНТ” ДЛЯ СИСТЕМИ ANDROID»

Виконав: здобувач вищої освіти 2 курсу, групи KN1-M22
спеціальності 122 Комп’ютерні науки
Чернявський Артем Антонович

Керівник: **Пилипюк Т.М.,**
кандидат фізико-математичних наук, доцент, доцент кафедри комп’ютерних наук

Рецензент: **Шумиляк Л.М.,**
кандидат технічних наук, асистент кафедри програмного забезпечення комп'ютерних систем Чернівецького національного університету імені Юрія Федьковича

Кам’янець-Подільський – 2023

ЗМІСТ

ВСТУП 3

РОЗДІЛ 1. МОБІЛЬНІ ПРИСТРОЇ ЯК ІНСТРУМЕНТ ОРГАНІЗАЦІЇ НАВЧАННЯ ЗДОБУВАЧА ВИЩОЇ ОСВІТА 7

1.1. Дослідження платформ для створення мобільного додатку 7

1.2. Вибір інструментів створення мобільних додатків для платформи Android 12

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ 19

2.1. Технологія створення програмного забезпечення. Фреймворк Flutter 19

2.2. Вибір технології для зберігання користувацьких даних 22

РОЗДІЛ 3. ПОРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ 26

3.1. Постановка задачі 26

3.2. Функціонал додатку 29

3.3. Структура бази даних. Пов’язування елементів різних сервісів 33

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ СТВОРЕННЯ ДОДАТКУ 37

4.1. Підготовчий етап 37

4.2. Розробка системи аутентифікації користувачів. Ініціалізація бази даних 41

4.3. Розробка сервісу “Індивідуальний навчальний план” 48

4.4. Розробка сервісу “Залікова книжка студента” 63

4.5. Розробка сервісу “Графік освітнього процесу” 70

4.6. Опис користування мобільним додатком 76

ВИСНОВКИ 80

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 82

ДОДАТОК 84

ВСТУП

Актуальність дослідження. У сучасному світі такі аспекти суспільної діяльності як сфера освіти знаходяться у процесі постійної безперервної трансформації. Розвиток наукового прогресу змушує їх динамічно підлаштовуватись під сучасні тенденції. Сектору освіти потрібно постійно еволюціонувати для того, щоб відповідати сучасним суспільним вимогам. Закладам вищої освіти вкрай важливо йти в ногу з часом, адже від цього залежить їх здатність випускати фахівців, готових позитивно впливати на суспільство.

Розвиток сучасних комп'ютерних технологій сприяв значному спрощенню доступу до інформації. За останні роки процес отримання даних став більш прямолінійним та доступним. Особливо помітними ці зміни стали у сфері освіти, де програмні продукти та цифрові ресурси відіграють ключову роль. Заклади вищої освіти постійно використовують сучасні технологічні рішення для удосконалення освітнього процесу та полегшення роботи адміністративного та навчального характеру учасників освітнього процесу.

Університети прикладають багато зусиль, аби забезпечити здобувачів вищої освіти усією необхідною та актуальною інформацією, яка може бути важливою для їх продуктивного навчання. В тому числі для цього використовуються сучасні цифрові технології. Однак, заклади вищої освіти наразі знаходяться тільки в процесі переходу до повної цифровізації. Юридичні та законодавчі особливості, необхідність адаптації учасників освітнього процесу та інші причини можуть стати перепонами для досягнення максимального комфорту в організації навчання.

Саме з цих причин під час навчання у закладі вищої освіти, здобувачі вищої освіти можуть стикнутися з труднощами доступу до деякої паперової документації навчально-організаційного характеру. Часто існує проблема, коли ці документи доступні тільки у фізичному форматі та знаходяться в університеті. Деякі з них можуть надаватися здобувачу вищої освіти за його

запитом, а деякі тільки за необхідністю. Тим не менш, у них міститься інформація, яка може стати корисною для здобувача освіти в організації свого освітнього середовища, правильного розподілення часу на навчання, аналізу власної студентської успішності.

Прикладами таких документів можуть слугувати індивідуальні навчальні плани, залікові книжки студентів та графіки освітнього процесу. Це створює необхідність створення системи управління документацією, що передбачає використання електронних версій документів для забезпечення легшого доступу до них. Під час впровадження дистанційного та змішаного типів навчання у закладах вищої освіти використання програмних продуктів для доступу до навчально-організаційних документів набуло особливої актуальності.

Оскільки значна частина здобувачів вищої освіти постійно використовує мобільні пристрої, ефективність навчального процесу може бути значно підвищена шляхом впровадження спеціалізованого мобільного застосунку, який надавав би доступ до цих матеріалів у електронному вигляді.

Програмний комплекс “Студент” – це програмний продукт, який надає користувачу доступ до трьох сервісів: електронного індивідуального навчального плану, електронної залікової книжки студента та електронного графіка освітнього процесу. У нашому випадку є сенс розробки цих функцій у рамках одного мобільного додатку, у якому користувач може за допомогою пунктів меню обирати потрібний сервіс. У такого підходу є переваги:

- користувачу достатньо завантажити тільки один мобільний додаток замість декількох, що не тільки спростить та пришвидшить процес встановлення програмного комплексу, а й зменшить витрати пам’яті мобільного пристрою на зберігання застосунку;
- переключення з одного сервісу на інший забезпечить комфорт користувача і цим самим значно пришвидшить процес;

- розробник зможе використовувати спільні елементи реалізації програми, наприклад – використовувати спільну базу даних для різних сервісів, тим самим давши можливість пов’язати дані з різних сервісів між собою;
- реалізований функціонал спростить процес авторизації та реєстрації користувачів, оскільки можна використовувати спільний обліковий запис для доступу користувачів до функціоналу додатку.

Мета дипломної роботи – створення мобільного додатку для смартфонів та інших пристроїв, що працюють на базі операційної системи Andorid, який включає декілька різноманітних функцій для здобувача вищої освіти з метою більш комфортної організації його освітнього середовища в закладі вищої освіти. Результатом роботи є повноцінний мобільний застосунок, який вільно зможуть використовувати здобувачі вищої освіти за допомогою своїх Android-девайсів.

Для досягнення мети потрібно виконати **завдання**:

- проаналізувати ринок мобільних девайсів в Україні та обрати платформу для створення мобільного додатку;
- розглянути різноманіття інструментів розробки мобільних застосунків та обрати технології для створення застосунку;
- дослідити інструменти зберігання користувацьких даних та визначити відповідну до проєкту технологію;
- здійснити проєктування структури та функціоналу додатку;
- описати процес реалізації програмного продукту;
- продемонструвати процес взаємодії користувача з фінальним продуктом.

Предметом дослідження є створення програмного додатку для смартфонів та інших пристроїв, що працюють на базі платформи Android з використанням фреймворку Flutter, сервісу авторизації та управління користувачами Firebase Authentication, хмарної документно-орієнтованої системи керування базами даних Firebase Cloud Firestore.

Об’єктом дослідження є програмний додаток “Студент”, який включає в себе такі сервіси, як “Індивідуальний навчальний план студента”, “Залікова книжка студента” та “Графік освітнього процесу”.

Практичне значення одержаних результатів: результатом роботи є повноцінний мобільний додаток – програмний комплекс “Студент”, готовий до використання здобувачами вищої освіти.

Апробація результатів дослідження. Результати досліджень були оприлюднені на звітній науковій конференції здобувачів вищої освіти 1 листопада 2023 року та опубліковані у збірнику матеріалів наукової конференції здобувачів вищої освіти фізико-математичного факультету Кам’янець-Подільського національного університету імені Івана Огієнка [18]. Статтю «Програмний комплекс “Студент” для системи Android» за результатами дослідження опубліковано у Віснику Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 16 [19].

Структура роботи. Дипломна робота магістра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатку.

РОЗДІЛ 1. МОБІЛЬНІ ПРИСТРОЇ ЯК ІНСТРУМЕНТ ОРГАНІЗАЦІЇ НАВЧАННЯ ЗДОБУВАЧА ВИЩОЇ ОСВІТА

1.1. Дослідження платформ для створення мобільного додатку

Смартфони є надзвичайно популярними девайсами серед здобувачів вищої освіти. Щонайменше 92% української молоді використовують сучасні мобільні пристрої у повсякденному житті [1]. Вони відіграють важливу роль для організації освітнього процесу. Здобувач вищої освіти може використовувати їх для отримання потрібної йому інформації через мережу Інтернет завдяки можливостям свого телефону.

Смартфони стають все більш важливим інструментом в організації навчання у закладах вищої освіти, забезпечуючи більш сучасний та ефективний підхід до освіти. Мобільні пристрої використовуються для комунікації між студентами та викладачами, спрощуючи обмін інформацією та зворотний зв'язок. Здатність мобільних девайсів надавати швидкий доступ до потрібних даних в будь-який час і з будь-якого місця є їх ключовою перевагою, яку необхідно ефективно використовувати.

На сьогоднішній день майже 99% усіх мобільних пристроїв в Україні працюють на операційних системах Android або iOS. Згідно з даними за листопад 2023 року, платформа Android займає близько 69.38% українського ринку, тоді як iOS контролює 30.2%. Всі інші мобільні платформи разом займають лише 1.42% ринку [2].

На рисунку 1.1. подано популярність використання ОС для мобільних пристроїв в Україні.

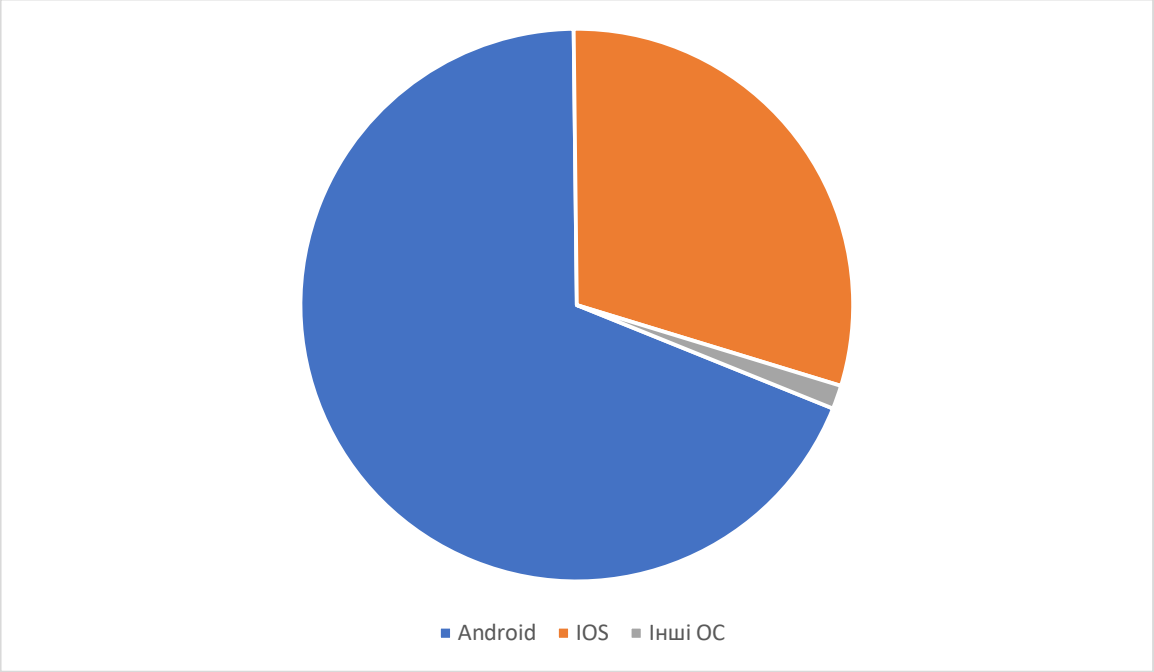

Рис. 1.1. Ринок ОС для мобільних пристроїв в Україні

Із рисунка 1.1 видно, що Android, розроблений Google, є найпопулярнішою операційною системою в Україні, оскільки охоплює приблизно 69.38% ринку. Ця перевага може бути обумовлена широким спектром виробників, які використовують цю систему, а також більш доступними цінами на пристрої, що працюють на Android. Ця платформа відома своєю гнучкістю, відкритістю і великим вибором програмного забезпечення.

З іншого боку, iOS, розроблена компанією Apple, контролює 30.2% українського ринку. Apple не дозволяє стороннім виробникам створювати мобільні пристрої на основі своєї операційної системи, таким чином єдині девайси, які працюють на платформі iOS виробляє сама компанія Apple. В основному це смартфони лінійки iPhone. Ця система відрізняється закритістю, високою якістю програмного забезпечення та зосередженням на конфіденційності та безпеці користувачів. Популярність iOS може бути зумовлена високою якістю продукції Apple та її статусом преміум-бренду.

У контексті цих даних, можна зробити висновок, що ринок мобільних операційних систем в Україні характеризується конкуренцією між двома

головними платформами – Android та iOS. Ця ситуація відображає глобальні тенденції та вказує на важливість цих платформ для розвитку мобільних технологій і програмного забезпечення.

Отже, враховуючи високу частку Android у ринковому сегменті, при розробці мобільного додатку варто зосередити зусилля на створенні додатків, що ефективно функціонують на цій платформі, але не ігнорувати і користувачів iOS. Тим не менш, популярність операційної системи – це не єдина річ, яку потрібно враховувати при виборі платформи для розробки власного мобільного додатку.

Для обгрунтування вибору платформи для розробки мобільного додатку здійснимо порівняння Android та iOS.

Розглянемо особливості розробки мобільних додатків для обох платформ. Обидві компанії, Google та Apple, надають розробникам власний набір інструментів для створення програмних продуктів для їх платформ [3].

Інтегроване середовище розробки (IDE) – це програмний пакет, який містить усі інструменти, необхідні для розроблення додатка, такі як редактор програмного коду, налагоджувач і засоби управління проектами.

Основним офіційним середовищем розробки для платформи Android є IDE, розроблене компанією Google – Android Studio. Це IDE дозволяє використовувати повний комплект для розробки програмного забезпечення Android (Android SDK). До нього входять таки ресурси для розробників як фреймворки, бібліотеки, налагоджувачі, емулятори та інші [3]. Android Studio забезпечує інтуїтивно зрозумілий інтерфейс та інтегроване середовище розробки, що спрощує процес створення та тестування додатків. Використання Android SDK в рамках цього середовища розробки дає програмістам доступ до останніх API та інструментів, що дозволяє створювати сучасні та функціональні додатки. Вбудовані налагоджувачі та інструменти для профілювання допомагають виявляти та усувати помилки, а також оптимізувати продуктивність додатків. Емулятори, що входять до

складу Android Studio, дозволяють тестувати додатки на різних пристроях та конфігураціях, не використовуючи фізичні пристрої.

В цей же час для розробки на iOS використовується XCode IDE. Це потужне інтегроване середовище розробки є ключовим інструментом для створення додатків не тільки для iOS, але й для macOS, watchOS та tvOS. Xcode надає розробникам комплексний набір інструментів, включаючи компілятори, бібліотеки, та інші ресурси, необхідні для ефективної розробки. Xcode включає Interface Builder, інструмент для візуального проектування інтерфейсів, який дозволяє розробникам легко створювати інтуїтивно зрозумілі, адаптивні інтерфейси для різних пристроїв Apple. Цей інструмент дозволяє перетягувати віджети для створення інтерфейсу, що істотно спрощує процес дизайну.

У підсумку, Xcode та Android Studio обидва надають потужні інструменти, необхідні для розробки мобільних додатків, але з різними особливостями та підходами, які відображають філософії та вимоги відповідних екосистем – Apple і Google.

Важливим аспектом вибору платформи для створення мобільного додатку є доступність інструментів для створення та можливість вільного розповсюдження готового програмного продукту серед користувачів. У цьому плані дві платформи мають великі відмінності. У порівнянні з платформою IOS, Android є відкрита для початку створення програмного забезпечення для будь-якого розробника та дає можливість використовувати додаток користувачами відразу після завершення процесу розробки, в той час як закритість платформи iOS є певною перешкодою як для розробника додатку, так і для вільного розповсюдження його серед користувачів, а також потребує певних фінансових витрат для створення програмного продукту.

Щоб створити Android-додаток, розробнику необхідно мати тільки персональний комп'ютер, який може працювати під керуванням будь-якої з сучасних операційних систем, таких як Windows, macOS чи Linux. Google, як розробник основної платформи Android, надає все необхідне програмне

забезпечення для створення додатків абсолютно безкоштовно, включно з інтегрованим середовищем розробки Android Studio та різноманітними бібліотеками та API для реалізації функціоналу додатка. В той самий час, Apple вимагає використання комп'ютерів Mac для розробки додатків під свої платформи, оскільки Xcode, основне середовище розробки для iOS, доступне тільки на macOS [4]. Це може стати серйозною перешкодою для розробників, які не мають доступу до обладнання Apple, оскільки продукція компанії зазвичай має вищу вартість порівняно з аналогічним обладнанням інших брендів. Такий підхід може обмежувати доступ до розробки на платформі iOS, особливо для індивідуальних розробників або невеликих стартапів з обмеженим бюджетом.

Коли розробка додатка завершена, розробник може зібрати його у вигляді програмного пакету (APK), який потім може бути встановлений на різних пристроях без необхідності витрачати додаткові кошти. Для більш широкого розповсюдження додатку, розробники часто вибирають публікацію своїх додатків у Google Play Store. Це не тільки забезпечує доступ до великої аудиторії потенційних користувачів по всьому світу, але також надає додатковий рівень довіри, оскільки додатки, розміщені на офіційному маркетплейсі, зазвичай сприймаються користувачами як більш надійні та безпечні. Для публікації додатку на Google Play Store, розробнику необхідно сплатити одноразовий внесок за реєстрацію облікового запису розробника, який становить 25 доларів США. Після подання додатку на Google Play, він проходить «ручну» перевірку адміністрацією магазину, що може зайняти деякий час. Цей процес має на меті переконатися, що додаток відповідає всім вимогам Google щодо якості та безпеки [5].

При розробці ж додатків на iOS такого ж вільного способу розповсюдження готового програмного продукту немає. Apple має більш закриту та контрольовану екосистему, що впливає на цей процес. Для розповсюдження iOS-додатків через App Store, розробники мають платити щорічний внесок за участь у програмі для розробників Apple, що становить

99 доларів США на рік. Також слід врахувати, що процес схвалення додатку в App Store може бути доволі тривалим та складним, існують строгі вимоги до дизайну та функціональності, а регулярні оновлення додатку також потребують повторного розгляду. При цьому, способу безкоштовно розповсюдити свій додаток серед користувачів, як це можна зробити з додатком на Android, не існує.

Після дослідження ринку операційних систем для мобільних девайсів, аналізу інструментів розробки та розповсюдження готових програмних продуктів, для мобільного додатку програмного комплексу “Студент” була обрана операційна система Android як основна платформа для розробки.

Причини такого вибору є такими:

- розповсюдженість платформи Android на ринку мобільних пристроїв;
- відсутність фінансових витрат на розробку завдяки доступності інструментів розробки для Android;
- можливість вільного розповсюдження готового програмного продукту серед користувачів.

Ці переваги роблять Android придатною платформою для розробки мобільного додатку, орієнтованого на здобувачів вищої освіти.

1.2. Вибір інструментів створення мобільних додатків для платформи Android

Мобільний додаток – це основний інструмент взаємодії користувача з функціоналом мобільного пристрою. Він розширює базові можливості девайсу, надаючи користувачам доступ до широкого спектру функцій і сервісів, які варіюються від простих утиліт до складних інтерактивних програм. Мобільні додатки можуть бути розроблені для різноманітних цілей, включаючи комунікацію, розваги, освіту, управління фінансами, здоров’я та

багато інших аспектів повсякденного життя. Приблизно 91% українських власників сенсорних смартфонів користується мобільними додатками [7].

За способом реалізації мобільні додатки можна розділити на такі основні категорії: нативні застосунки, крос-платформні застосунки, гібридні застосунки, та прогресивні веб-застосунки [8].

Розглянемо ці категорії.

Нативна розробка додатків для мобільних пристроїв є процесом створення програмного забезпечення спеціально для конкретної платформи або операційної системи. Вона дозволяє розробникам максимально використовувати можливості конкретної платформи, та забезпечувати високу продуктивність, швидкість та плавність роботи додатку. Додатки розробляються окремо для кожної операційної системи, Android або iOS, використовуючи мови програмування та інструменти, які є найбільш придатними для цих платформ.

Основними мовами програмування для створення мобільних додатків для Android є Kotlin та Java [9]. Java довгий час була основною мовою для розробки Android-додатків. Пізніше такий статус отримав Kotlin.

Kotlin, розроблений JetBrains, став основною мовою для розробки Android-додатків та є офіційно рекомендованим Google з 2019 року. Kotlin є відгалуженням Java, яке відрізняється тим, що програмний код компілюється у байт-код JVM. Це забезпечує плавну інтеграцію з існуючим кодом Java та його фреймворками. Синтаксис Kotlin сучасний і легкий для читання, що робить його привабливим як для новачків, так і для досвідчених розробників.

Тим не менш, Java залишається найпопулярнішою з двох офіційних мов для розробки Android-додатків. Це мова загального призначення, з корінням в C та C++, яка добре підходить для більшості завдань Android-розробки, включаючи написання компонентів Android-фреймворку та створення власних додатків. Java має велику спільноту розробників та багато доступних ресурсів для підтримки проєктів.

Для платформи iOS, основними мовами програмування є мови Swift та Objective-C [10].

Swift є основною мовою для нативної розробки iOS-додатків, розробленою Apple спеціально для своїх пропрієтарних платформ. Swift сучасна та легка у використанні, вона була розроблена для заміни старіших мов, таких як C, C++ та Objective-C. Swift швидко стає головною мовою розробників iOS. У 2023 році майже 5% розробників використовували Swift у порівнянні з приблизно 2%, які використовували Objective-C.

Хоча Swift швидко набирає популярності, Objective-C все ще використовується в нативній розробці iOS. Ця мова має довгу історію використання у розробці додатків Apple і раніше була основною мовою для iOS до початку використання Swift.

Можна виділити основні переваги та недоліки *нативної розробки* [11]:

Однією з ключових переваг нативної розробки є те, що додатки, створені таким чином, можуть безпосередньо взаємодіяти з апаратним забезпеченням пристрою, таким як камера, мікрофон, акселерометр і т.ін. У таких застосунків є повна інтеграція з операційною системою та екосистемою платформи. Нативні додатки можуть використовувати всі функції, які надає операційна система, такі як пуш-сповіщення, інтеграція з календарем, контактами та іншими системними додатками. Це дає розробникам більш глибокий контроль над функціоналом та дизайном додатку.

Другою основною перевагою є швидкість та стабільність готового програмного продукту. Вони запускаються, завантажуються та виконуються швидше за будь які інші та, як правило, мають менше помилок. Причиною цього є те, що код програми працює безпосередньо у самій операційній системі. Немає проміжного рівня програмного забезпечення (на відміну від веб-додатків), який може значно сповільнити виконання. Також нативні розробники мають доступ до інструментів та бібліотек для оптимізації продуктивності, таких як Profile GPU Rendering Tool для Android.

Головним же недоліком нативних додатків для мобільних платформ є складність та трудомісткість розробки, особливо коли є необхідність створювати окремі додатки для різних платформ. Компаніям або індивідуальним розробникам, які хочуть випустити свій додаток на обох платформах, необхідно вести розробку паралельно для кожної з них, що збільшує витрати часу та ресурсів. Крім того, це може призвести до складнощів у підтримці та оновленні додатків, оскільки кожна зміна чи додавання функціоналу вимагає внесення змін на кількох платформах одночасно.

Іншим видом мобільних додатків є *прогресивні веб-додатки* (Progressive Web Apps, або PWA). Це тип застосунків, які використовують веб-технології для створення користувацького досвіду, подібного до нативних додатків [12].

Після встановлення PWA виглядає як будь-який інший мобільний додаток: він має власний значок на головному екрані, панелі запуску програм, або в стартовому меню, з’являється під час пошуку програм на пристрої, відкривається в окремому вікні, повністю відокремленому від інтерфейсу веб-браузера. Прогресивний веб-додаток має доступ до високих рівнів інтеграції з операційною системою, наприклад, сервісом обробки URL-адреси. Також він може працювати в автономному режимі.

PWA розробляються виключно за допомогою веб-технологій, таких як HTML, CSS та JavaScript, що робить їх більш універсальними та легкими в розробці, та дозволяє уникнути потреби в окремих розробниках для різних платформ. Прогресивні веб-додатки забезпечують високий рівень сумісності з різними пристроями та операційними системами, оскільки вони використовують стандартні веб-технології. Вони не вимагають завантаження через магазини додатків, як Google Play або Apple App Store, а розповсюджуються безпосередньо через веб-сайт. Такі додатки зазвичай займають менше місця на пристрої користувача та споживають менше ресурсів, ніж нативні додатки.

Недоліками PWA є їх обмеження у доступі до певних апаратних можливостей пристрою, порівняно з нативними додатками; обмежена підтримка платформ, особливо більш старих, та низька швидкодія, в порівнянні з будь якими іншими типами мобільних додатків.

Гібридні мобільні додатки об'єднують переваги нативних та веб-додатків, дозволяючи розробникам створювати універсальні застосунки, які можуть працювати на різних операційних системах з використанням єдиної бази коду. Вони теж розробляються з використанням стандартних веб-технологій – HTML, CSS, та JavaScript, але на відміну від PWA, обгортаються в нативну оболонку. Ця оболонка дозволяє гібридному додатку взаємодіяти з функціями мобільної платформи і бути встановленим на пристрій як нативний додаток. Гібридні додатки можуть використовувати API мобільної платформи для доступу до камери, GPS та інших функцій пристрою, при цьому вони мають перевагу спільної бази коду для різних платформ, що полегшує розробку та підтримку.

Крос-платформна розробка – це ще один спосіб створення мобільних додатків для мобільних платформ. Я і у випадку з веб- та гібридними додатками, розробники можуть створювати свої застосунки з однієї кодової бази, та використовувати застосунки відразу на декількох різних платформах. Тим не менш, у них є суттєві відмінності [13].

Крос-платформні застосунки іноді називають “нативними крос-платформними додатками”, щоб підкреслити, що хоч ці додатки і випускаються для декількох операційних систем, вони мають більше спільного з нативними програмами, ніж веб-додатками. Хоча кодова база у цих додатків одна, кожна окрема версія для різних платформ компілюється окремо.

Це означає, що такі додатки не просто працюють на різних платформах, але й оптимізовані для кожної з них, забезпечуючи плавну інтеграцію з їхніми нативними можливостями та дизайном. Також це значить те, що під час розробки можна враховувати специфічні особливості та

обмеження кожної системи, такі як відмінності у дизайні інтерфейсу, взаємодії з апаратним забезпеченням та доступ до нативних API. Таким чином, кожна версія додатку не тільки працює на різних платформах, але й забезпечує оптимальний користувацький досвід на кожній з них.

На сьогоднішній день можна виділити два основних фреймоврки крос-платформної розробки, які конкурують між собою та боряться за лідерство на ринку [14]:

React Native, розроблений Facebook, є фреймворком для кросплатформеної розробки, який надає розробникам можливість створювати мобільні додатки, що майже рівні з нативним за швидкістю, використовуючи React та JavaScript. Цей фреймворк спрощує процес розробки додатків для iOS та Android, дозволяючи швидше впроваджувати їх на ринок. Хоча React Native залишається лідером у сфері кросплатформенної розробки, він стикається зі зростаючою конкуренцією з боку Flutter, який також звертає багато уваги розробників.

Flutter, розроблений Google, є відносно новим фреймворком для створення нативних інтерфейсів для мобільних, веб- та десктопних додатків з єдиної кодової бази. Він використовує мову програмування Dart і надає широкі можливості для створення гладких анімацій та високопродуктивних додатків. Однією з переваг Flutter є висока швидкість роботи готових додатків, яка досягається завдяки компіляції в машинний код. На сьогодні популярність цього фреймворку швидко зростає, що робить його актуальним інструментом для розробки мобільного додатку.

Недоліками крос-платформних додатків можна назвати такі: хоч такі додатки і працюють набагато швидше за веб-додатки, їх продуктивність все ще не досягає рівня нативних застосунків. Такі додатки можуть підтримувати не всі функції мобільних пристроїв, які є лише нативними, такі як складна графіка та анімація або 3D-ефекти. Коли Google і Apple додають нові функції на платформи Android і iOS, нативні рішення можуть негайно почати їх

використовувати. Але крос-платформним додаткам доводиться чекати, поки ці оновлення будуть адаптовані до обраного крос-платформного фреймворку.

Отже, у розділі 1 була досліджена роль мобільних пристроїв у освітньому процесі як важливих інструментів для здобувачів вищої освіти. Аналіз ринку операційних систем показав, що Android домінує на ринку мобільних платформ в Україні, що робить її хорошим вибором операційної системи для розробки мобільного додатку. Цей вибір також зумовлений доступністю ресурсів для розробки та більш простим процесом розповсюдження та публікації додатків порівняно з iOS.

Також розглянуто інструменти та технології для створення мобільних додатків, з особливим фокусом на крос-платформній розробці мобільних додатків. Аналіз підкреслює важливість мобільних технологій в освіті та виправдовує вибір конкретних технологій для розробки мобільного додатку.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

2.1. Технологія створення програмного забезпечення. Фреймворк Flutter

У результаті дослідження різних способів та інструментів створення програмного забезпечення для платформи Android, обрано фреймворк Flutter як основний комплект засобів розробки програмного комплексу “Студент”.

Flutter – це безкоштовний фреймворк від Google, створений для розробки крос-платформних нативних мобільних застосунків [15]. Запущений у 2017 році, цей фреймворк надає можливість розробляти додатки для iOS та Android, використовуючи одну базу коду та мову програмування. Ця особливість значно спрощує та прискорює розробку мобільних додатків. Незважаючи на свій відносно невеликий вік, Flutter швидко став популярним та ефективним інструментом для розробки. Flutter працює на основі іншої розробки Google – мові програмування Dart.

Використання Flutter для розробки додатків не лише залишається актуальним, а й має великі перспективи. Розробники підкреслюють, що динамічний розвиток Flutter, як на ринку, так і у сфері його функціональних можливостей, має потенціал стати майбутнім стандартом у крос-платформній розробці мобільних додатків. З моменту його запуску, швидкість інновацій та оновлень Flutter вже перевершила більшість провідних крос-платформних фреймворків у галузі розробки мобільних додатків.

Flutter дозволяє розробляти додатки, використовуючи єдину кодову базу. Додатки, створені за допомогою Flutter, можуть бути легко опубліковані та використані на різних платформах, включаючи Android, iOS, комп'ютери та веб. Це робить Flutter хорошим вибором для бізнесу, оскільки розробка міжплатформних додатків з Flutter забезпечує значну економію часу та ресурсів.

У Flutter віджет виступає як фундаментальний елемент для створення інтерфейсу користувача, використовуючи різноманітні будівельні блоки, такі як кнопки, відступи, шрифти, тощо. Ці віджети надають гнучкість для розробки різноманітних інтерфейсів, від базових до складних, використовуючи один і той же інструментарій. Візуалізація віджетів Flutter у вигляді дерева віджетів є зручною для розуміння та організації інтерфейсу розробником.

Функція гарячого перезавантаження (Hot Reload) у Flutter, яку високо оцінили професіонали у всьому світі, є однією з ключових переваг цього фреймворку в розробці додатків. Ця властивість дозволяє розробникам швидко перевіряти навіть найменші зміни в коді без необхідності повного перезапуску програми, що економить час розробника і підвищує ефективність розробки. З гарячим перезавантаженням у своєму розпорядженні розробники можуть експериментувати з кількома дизайнами без затримки та бачити, як вони впливають на програму, перш ніж зупинитися на остаточній концепції – і все це дозволить уникнути простоїв на етапах тестування або налагодження проєкту пізніше.

Важливість швидких програм і привабливого інтерфейсу не можна недооцінити. Якщо додаток для Android або iOS має високий час завантаження, ймовірно, люди просто видалять його. Однак із Flutter як опцією для створення мобільного застосунку ця ситуація значно відрізняється – скорочений час завантаження забезпечує швидший початок використання та зменшує ймовірність передчасної відмови користувачами. Завдяки використанню бібліотеки Skia Graphics Library від Google, Flutter забезпечує високу ефективність та плавність роботи програми. Крім того, використання спеціальних віджетів Flutter дозволяє розробникам легко створювати естетично привабливі та функціональні інтерфейси користувача, що підвищує загальну якість дизайну мобільної програми.

Flutter використовує мову програмування Dart, розроблену Google, яка має багато спільних рис з JavaScript або TypeScript. Однією з ключових

особливостей Dart є її підтримка реактивної моделі програмування для створення інтерфейсів користувача.

У реактивному програмуванні, замість того, щоб розробник вручну оновлював інтерфейс користувача після кожної зміни в коді, Flutter автоматично виконує ці оновлення. Це значно спрощує процес створення динамічних та адаптивних інтерфейсів, оскільки розробникам не потрібно турбуватися про ручне керування станом інтерфейсу. Завдяки цьому підходу, розробка стає швидшою, зручнішою та ефективнішою, що є великою перевагою для розробників, які прагнуть створювати високоякісні мобільні додатки.

Виходячи з дослідження фреймворку Flutter, причинами вибору саме цієї технології для розробки мобільного додатку стали:

- швидкість розробки та комфорт для розробника – мова Dart, яку використовує фреймворк Flutter, статистично є однією з улюблених мов серед програмістів завдяки своїй простоті, сучасним функціональним можливостям і підтримці об'єктно-орієнтованого програмування. Система віджетів у Flutter дозволяє легко створювати складні інтерфейси з повторним використанням компонентів, що суттєво прискорює процес розробки;
- використання вбудованих віджетів Material Design у Flutter дозволяє розробникам легко створювати візуально привабливі інтерфейси, які відповідають сучасним стандартам дизайну встановленим Google. Ці віджети не лише прискорюють процес розробки, але й забезпечують цілісний дизайн на різних пристроях та платформах;
- проста та швидка інтеграція іншого продукту Google – сервісу Firestore. Firestore є гнучкою, масштабованою базою даних. Вона дозволяє легко синхронізувати дані між користувачами в реальному часі, що є важливим для програмного комплексу "Студент". Flutter має вбудовану підтримку Firestore, що дозволяє легко і швидко інтегрувати хмарну базу даних без потреби у складному налаштуванні;

• можливість кросплатформового розвитку – за необхідністю з тієї ж кодової бази можна буде дуже легко створити версію додатка на інші платформи, такі як iOS, Windows або Web.

2.2. Вибір технології для зберігання користувацьких даних

Для мобільних застосунків, що оперують певною кількістю користувацьких даних, постає проблема збереження даних у організованому вигляді. Це важливо для забезпечення ефективності, швидкості доступу до даних та їхньої захищеності.

Вирішенням цієї проблеми є використання баз даних. База даних у контексті мобільних застосунків може бути визначена як сукупність структурованих даних, організованих таким чином, що вона дає можливість ефективно зберігати, модифікувати, зчитувати дані та керувати інформацією. Вона є критично важливою складовою архітектури мобільних додатків, оскільки дозволяє зберігати важливі дані, які потрібні для функціонування застосунку. База даних у мобільному застосунку забезпечує централізоване зберігання даних, що є важливим для багатьох функцій додатку.

Залежно від потреб застосунку, бази даних можуть бути реалізовані як у хмарі, так і локально на пристрої. Хмарні бази даних дозволяють легко масштабувати застосунки та забезпечувати доступ до даних з будь-якого місця, де є Інтернет. Однак, локальні бази даних, вбудовані безпосередньо у застосунок, дозволяють забезпечити доступ до даних навіть у відсутності Інтернет-з'єднання.

Існують певні категорії, до яких можна занести різні бази даних. Розглянемо їх.

- Локальні бази даних – бази даних, які зберігаються безпосередньо на мобільному пристрої. Вони забезпечують високу швидкість доступу до даних і можуть функціонувати в офлайн-режимі. Прикладами локальних баз даних є SQLite і Realm.

- Хмарні бази даних – бази даних, які розміщуються на віддалених серверах і доступні через Інтернет. Вони забезпечують масштабованість, централізоване зберігання даних та спрощують синхронізацію даних між різними пристроями. Прикладами є Firebase, MongoDB Atlas та Amazon DynamoDB.

- Синхронізовані бази даних – бази даних, які поєднують елементи локальних та хмарних баз даних, забезпечуючи зберігання даних як на мобільному пристрої, так і в хмарі. Вони автоматично синхронізують дані між локальними та хмарними версіями, що дозволяє користувачам мати доступ до актуальних даних незалежно від того, чи є у них Інтернет-з'єданння, чи ні. Couchbase Mobile та Firebase Realtime Database є прикладами таких систем.

- NoSQL бази даних – це гнучкі бази даних, які підтримують різні типи даних, не обмежуючись строгою схемою. Вони ідеально підходять для мобільних застосунків, які потребують великої гнучкості у зберіганні даних. До NoSQL баз даних належать MongoDB, Cassandra, CouchDB, та Firestore. Вони можуть використовуватися як в локальних, так і в хмарних рішеннях. Наприклад, MongoDB може бути використана як в якості основної хмарної бази даних, так і в якості вбудованої локальної бази даних в мобільному додатку.

Як технології зберігання користувацьких даних опишемо Firebase та Cloud Firestore.

У результаті дослідження різних способів та інструментів створення програмного забезпечення для платформи Android, обрано сервіс *Firebase Cloud Firestore* як хмарну базу даних для мобільного додатку "Студент".

Firestore – це NoSQL, документо-орієнтовна синхронізована база даних, що є одним із продуктів сервісу Firebase, розробленого Google [17]. Причини вибору цієї бази даних для мобільного додатку є такими:

- Cloud Firestore є синхронізованою базою даних, що зберігає інформацію в мережі Інтернет, та дозволяє проводити операції над нею через

мережу. Тим не менш, використання цієї бази даних у додатку не змушує користувача бути постійно підключеним до мережі Інтернет: зміни у додатку можна проводити навіть коли пристрій не має зв’язку з мережею, але коли він з’явиться, дані у хмарі синхронізуються та зміняться. Це підходить для задач додатку, що буде створений, та позитивно вплине на користувацький комфорт.

- Cloud Firestore є документо-орієнтовною базою даних, що означає, що вона зберігає дані у вигляді документів, які групуються в колекції. Кожен документ може містити різноманітні дані, такі як рядки, числа, об'єкти, масиви тощо. Ця структура надає велику гнучкість у організації та використанні даних. Це підходить для задач, які ми будемо ставити перед базою даних, які потребуватимуть високої гнучкості, категоризації елементів та можливості їх взаємо пов’язування.
- Ця база даних була створена і підтримується компанією Google, тією самою, якій підпорядковується основна технологія, що буде використана для створення самого застосунку – Flutter. Це означає, що саме ця база даних буде мати найкращу підтримку розробниками, в порівнянні з конкурентами, отже і технічних проблем та складнощів реалізації буде у мінімальній кількості.
- Google створив зручні інструменти для інтеграції цієї бази даних у Flutter додаток, такі як пакети Firebase та Cloud Firestore, а також утиліти для напів-автоматичної інсталяції, такі як Firebase CLI та плагіну для нього FlutterFire.
- Використання цих сервісів є безкоштовним для проєктів невеликого масштабу. Безкоштовно можна використовувати до 1 ГБ даних, та до 20000 запитів запису/читання у день. Цього цілком вистачить для проєкту для наших задач.

Отже, у розділі 2 розглянуто вибір технологій для розробки мобільного додатку "Студент". Детально проаналізовано різні підходи до мобільної розробки, інструменти та мови програмування, що використовуються для створення додатків на різних платформах. В результаті дослідження обрано

фреймворк Flutter як основний інструмент для розробки додатку та технології зберігання користувацьких даних Firebase та Cloud Firestore.

РОЗДІЛ 3. ПОРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Постановка задачі

Метою роботи є створення програмного комплексу «Студент» для платформи Android. Складовими програмного комплексу є три сервіси, які будуть доступні користувачу у рамках одного мобільного додатку. Функціями додатку є можливість здобувачу вищої освіти переглядати, змінювати та оновлювати інформацію документів навчально-організаційного характеру у електронному вигляді. Такими документами є:

- Індивідуальний навчальний план здобувача вищої освіти – документ, що визначає послідовність, форму і темп засвоєння здобувачем вищої освіти освітніх компонентів освітньої (освітньо-професійної чи освітньо-наукової) програми з метою реалізації його індивідуальної траєкторії та розробляється Університетом у взаємодії зі здобувачем вищої освіти за наявності необхідних для цього ресурсів.
- Залікова книжка студента – документ у якому містяться записи про складання студентом заліків, екзаменів, захист курсових і дипломних робіт (проєктів), навчальних та виробничих практик, атестацію здобувача вищої освіти.
- Графік освітнього процесу – це нормативний документ, який визначає календарні терміни семестрів, теоретичного навчання та практичної підготовки, семестрового контролю (екзаменаційних сесій), підготовки кваліфікаційних/дипломних робіт (проєктів), атестації здобувачів вищої освіти, канікул, самостійної роботи; науково-дослідницької роботи, оформлення та захисту дисертації.

Отже, завданням є створити програмний комплекс “Студент” – мобільний додаток, який надає користувачу доступ до трьох сервісів: електронний індивідуальний навчальний план, електронна залікова книжка студента та електронний графік освітнього процесу.

Мобільний додаток з таким функціоналом може використовуватись здобувачем вищої освіти протягом усього часу навчання у закладі вищої освіти, що може тривати роки. Отже, потрібно впевнитися, що дані, які будуть зберігатись у додатку, є надійно збереженими та захищеними. Необхідно впевнитися, що дані не будуть втрачені.

У зв’язку з цим, ми не маємо можливості покластися на внутрішню пам’ять телефона як на місце для зберігання даних. За роки мобільний пристрій може змінитись, загубитись, бути вкраденим, тощо. У такому випадку дані програми будуть втрачені. Крім того, внутрішня пам'ять телефону обмежена, і це іноді може стати проблемою при збереженні великих обсягів інформації.

Ефективним рішенням цієї проблеми є використання хмарних сервісів для зберігання баз даних. Вони не тільки дають достатній рівень захисту даних, а й дають можливість здобувачу вищої освіти зберегти всі користувацькі дані та знову їх використовувати навіть після заміни мобільного девайсу. Хмарні сервіси забезпечують високу доступність даних, оскільки вони можуть бути доступні з будь-якого пристрою, що має підключення до Інтернету.

При використанні хмарних баз даних з’являється необхідність надавати кожному окремому користувачу ті елементи бази даних, які призначені саме для нього. Це означає, що потрібно мати можливість розрізняти кожного окремого користувача додатку один від одного. Найкращим способом реалізації такого функціоналу є використання системи аутентифікації користувачів.

Аутентифікацію користувачів можна розділити на такі елементи як реєстрація та авторизація. Під час реєстрації користувача, він обирає та вводить свої ідентифікаційні дані, такі як логін (адреса електронної пошти), та пароль. Ці дані відомі тільки йому та будуть використовуватись у майбутньому. Коли реєстрація завершена, у базі даних з’являється сегмент, який буде використовуватись додатком для зберігання та відображення даних

конкретного користувача. У майбутньому, користувач завжди зможе отримати доступ до цих даних після проходження авторизації – вводу власного логіну та паролю, що використовувалися при реєстрації.

З такою реалізацією ми отримуємо надійний спосіб зберігання даних користувачів. Для доступу до даних здобувач освіти повинен тільки ввести дані авторизації, в той час як задача зберігання даних покладається на сервіс керування базами даних. Тепер дані не втратяться якщо виникнуть проблеми з мобільним пристроєм. Користувач може авторизуватись на будь-якому пристрої, на якому встановлений додаток, і отримати всі необхідні дані, що прив’язані до його аккаунту.

Як вже було досліджено, підходящою для наших цілей базою даних буде база даних Firestore. Отже, для створення мобільного додатку було вирішено використовувати сервіс авторизації Firebase Authentication разом з базою даних Firestore Database.

Після авторизації, користувач повинен мати можливість обрати сервіс, який він буде використовувати. Це може бути реалізовано через пункти меню на головному екрані додатку. Після вибору пункту меню відкривається новий екран з відповідним функціоналом.

Так як вищевказані документи навчально-організаційного характеру не зберігаються у електронному вигляді, здобувачу вищої освіти потрібно мати можливість самостійно вносити інформацію до них.

Індивідуальний навчальний план. Для реалізації цього документу в електронному вигляді, необхідно мати необхідний функціонал: створення нових освітніх компонентів (дисциплін, практик, індивідуальних завдань, наукових робіт, тощо.), виведення їх на екран, можливість редагувати та видаляти елементи. Згідно до складових індивідуального навчального плану, повинна бути можливість вводити такі пункти як: назва освітнього компоненту, терміни вивчення, форма підсумкового контролю, та навчальне навантаження. Навчальне навантаження включає у себе кількість навчальних годин виділених на навчальну роботу студента. У ньому показується кількість

годин на всю навчальну роботу разом, яку можна розділити на аудиторну роботу та самостійну роботу студента. В свою чергу, аудиторне навчальне навантаження складається з навчальних годин, виділених на лекційні заняття, на практичні чи семінарські заняття, на лабораторні заняття та курсові роботи. Окремо кожен освітній компонент повинен мати не тільки загальну кількість годин, виділених на нього, а й показувати це значення, переведене у кредити ЄКТС. Кредит ЄКТС, як правило, становить 30 навчальних годин.

Залікова книжка студента. Для реалізації цього документу у електронному вигляді потрібно мати такий функціонал: створення нових освітніх компонентів (дисциплін, практик, індивідуальних завдань, наукових робіт, тощо.), виведення їх на екран, можливість редагувати та видаляти елементи. Кожен компонент має такі дані: назва компоненту, термін проведення, форма підсумкового контролю, викладач, дата оцінювання, та оцінка.

Графік освітнього процесу. Для реалізації цього документу у електронному вигляді потрібно мати такий функціонал: створення нових освітніх компонентів, їх тип та точний термін реалізації (початок і кінець).

3.2. Функціонал додатку

Відповідно до постановки задачі, функціонал додатку має бути таким:

Екран авторизації, з полем для вводу логіну (електронної пошти) та полем для вводу паролю користувача. Кнопка “Увійти”, після натискання на яку перевіриться правильність введення логіну та паролю. У полі “пароль” текст паролю буде захованим. Якщо дані аутентифікації вірні, користувач потрапляє до головного меню додатку з усіма доступними сервісами. Якщо пароль був введений невірно, користувач отримає текст з помилкою. Цей екран буде екраном за замовчуванням для не авторизованого користувача. Щоб була можливість для нового користувача створити новий обліковий

запис, потрібна кнопка “Створити новий аккаунт”, яка перенаправить його на екран реєстрації.

Екран реєстрації, з полем вводу електронної пошти, полями для вводу паролю та його підтвердження. Дані перевіряються на коректність (чи введена адреса електронної пошти в правильному форматі, чи немає не підтримуваних базою даних символів введених у текстові поля). Також перевіряється чи однакові паролі у текстових полях вводу та підтвердження паролю. На екрані присутня кнопка “Зареєструватись”, що завершує процес створення облікового запису, та кнопка “Увійти”, щоб перейти назад до екрану авторизації. Після успішної реєстрації користувач потрапляє до головного екрану програми.

Основний екран, або головне меню, повинно містити пункти навігації по додатку. Вони будуть реалізовані за допомогою кнопок, які перенаправляють користувача до необхідного сервісу. Ці кнопки повинні бути відповідно підписані: “Індивідуальний навчальний план” “Залікова книжка студента” та “Графік освітнього процесу”. Для зручної роботи, у головному меню буде присутній елемент інтерфейсу типу “випадний список”, що дозволить обрати поточний семестр студента. Цей поточний семестр буде використовуватись у сервісах додатку як відкритий семестр за замовчуванням. Це зроблено тому, що найчастіше користувач буде використовувати дані саме поточного семестру навчання. На головному екрані також присутня кнопка виходу з облікового запису, що здійснює вихід з аккаунту користувача та повертає його до екранів аутентифікації.

Екран “Індивідуальний навчальний план” – екран має виконувати функції індивідуального навчального плану студента. На екрані присутній випадний список семестру, список освітніх компонентів, який ми виводимо на екран. В зв’язку з тим, що зазвичай кількість освітніх компонентів у одному семестрі достатньо велика, для економії місця у інтерфейсі вони будуть показані як список зі згорнутих плиток (tile). Плитка – це елемент інтерфейсу у дизайні мобільного додатку, який можна згортати та розгортати.

У згорнутому вигляді плитка показує мінімальний набір інформації, наприклад, тільки назву, а при розгортанні – показує всю інформацію про елемент. Таким чином, на кожний освітній елемент генерується одна плитка, що у згорнутому вигляді має такі елементи, як назва освітнього елементу, кнопка згортання/розгортання освітнього елементу, кнопка редагування та видалення освітнього елементу. Після натискання на плитку вона розгортається, звільняючи місце у списку під собою, та показуючи всю інформацію про освітній елемент. У такому вигляді доступна така інформація про нього, як назва, форма підсумкового контролю, та навчальне навантаження.

Над списком повинна знаходитись кнопка додавання нового освітнього елементу. Після натискання відкривається діалогове вікно, де запропоновано ввести такі дані про предмет, як його назву, години на лекційні заняття, години на практичні/семінарські заняття, години на лабораторні заняття, години на курсову роботу та години на індивідуальні завдання, а також форму підсумкового контролю, що репрезентується випадним списком з варіантами “Екзамен”, “Залік” та “Інше”. При додаванні елементу, сумуються всі аудиторні, та всі загальні години, та зберігаються як окремі значення. Так само рахується кількість навчальних кредитів ЄКТС. Нагорі – випадний список з вибором семестру, до якого буде доданий освітній елемент. Для комфорту користувача – обраним за замовчуванням буде той семестр, список дисциплін якого переглядав користувач коли натиснув кнопку “додати”. У діалоговому вікні присутні кнопки “Закрити” та “Зберегти”. При натисканні на останню новий освітній елемент буде доданий до списку.

При натисканні на кнопку “Редагувати” на плитці (репрезентується значком олівця), відкриється діалогове вікно, аналогічне діалоговому вікну при створенні нового освітнього елементу, але з уже заповненими полями з інформацією з об’єкту. При натисканні на кнопку “Видалити” (репрезентується значком смітника), відкривається інше діалогове вікно, де

користувач може вибрати, чи видалити тільки інформацію про елемент з індивідуального навчального плану, чи повністю видалили освітній елемент з усіх сервісів додатку.

Для більш зручного знаходження потрібних елементів у списку, на сторінці повинен бути функціонал сортування. Він буде виглядати як випадний список, після вибору елемента на якому список елементів пересортовується за відповідними параметрами. Способи сортування: “За алфавітом А-Я”, “За алфавітом Я-А”, “За аудиторними годинам”, “За індивідуальними годинами”, “За годинами всього”.

Екран “Залікова книжка студента” – на цьому екрані повинна міститися реалізація електронної залікової книжки. Ця сторінка буде мати багато спільних елементів з індивідуальним навчальним планом: випадне меню вибору семестру, інформацію про який користувач хоче дізнатись, кнопка додавання нового семестру, випадне меню з вибором сортування елементів у списку.

Елементи списку не містять так багато інформації, як у індивідуальному навчальному плані, тому є сенс виводити всю інформацію про оцінювання знань студента одразу. Список буде набором контейнерів з інформацією, які мають такі текстові дані як назва дисципліни, викладач, кількість годин та кредитів, виділених на дисципліну, дата проведення оцінювання та оцінка. Також на кожному елементі присутні кнопки “Редагувати” та “Видалити”, з аналогічним функціоналом до тих, що були у індивідуальному навчальному плані.

Для зручної роботи з цією інформацією, потрібно візуально розділяти елементи по формі підсумкового контролю. Кожен елемент має відповідний колір рамки контейнера, в залежності від типу – червоний для екзамену, жовтий для заліку, зелений для інших типів оцінювання. Під сортуванням можна буде відмічати прапорцем пункт “Групувати за формою підсумкового контролю”, що по черзі виведе елементи за категоріями “Екзамен”, “Залік”, “Інше”.

Екран "Графік освітнього процесу" має на меті перенести цей документ у електронний вигляд. Як для документа, що має на меті відмітити терміни і дати, має сенс скористатись календарем, щоб візуально представити цю інформацію користувачу. На календарі повинні відмічатись дати та терміни різних навчальних подій. При натисканні на дату – повинен виводитись список пов'язаних з цією датою елементів. В залежності від типу елементу, потрібно дати можливість їх розрізняти використовуючи відмітки різного кольору.

3.3. Структура бази даних. Пов'язування елементів різних сервісів

Після постановки задачі видно, що всі сервіси додатку – індивідуальний навчальний план, залікова книжка студента, та графік освітнього процесу мають багато спільних рис. У більшості випадків ці сервіси працюють з великою кількістю інформації, що може використовуватись у декількох сервісах одночасно. Цю інформацію можна привести до поняття "освітній елемент", до якого входять навчальні дисципліни, екзамени та заліки з них, навчальна та виробнича практика, їх захист, курсові та дипломні роботи, їх захист, тощо. Не кожен з них використовується у кожному з цих сервісів, тим не менш, повторюються вони досить часто. Наприклад, якщо навчальна дисципліна присутня в індивідуальному навчальному плані, ймовірніше за все з цієї дисципліни буде проведене оцінювання, термін якого буде вказаний у графіку навчального процесу, а його результати будуть внесені у залікову книжку.

Виходячи з цього можна дійти до висновку, що ефективним способом роботи з даними у мобільному додатку "Студент" буде таким, при якому коли користувач додає освітній елемент у один сервіс програмного комплексу, він потрапляє у один спільний список цих елементів у базі даних. Це дозволить пов'язати між собою інформацію про освітній елемент з різних сервісів між собою. У цього підходу є такі переваги:

- Користувачу не потрібно окремо додавати освітній елемент до кожного окремого сервісу додатку, що збереже його час на введення його назви, та дасть менше простору для помилки. При чому він зможе обирати, чи потрібно використовувати елемент з одного сервісу у інших, чи ні. Це збереже час на додання цього елементу.
- Програма зможе брати інформацію про освітній елемент, що спільна для декількох сервісів, щоб використовувати її всюди, а користувач зможе редагувати її з одного місця, і вона буде змінена у всіх сервісах. Прикладом цієї інформації буде форма підсумкового контролю.
- Це значно спростить структуру бази даних додатку. Використовувати структуру, де до одного елементу прив'язані усі його характеристики, буде практичним та ефективним способом роботи з базою даних.

Окремо від освітніх елементів потрібно позначити записи, які будуть вказуватись окремо лише у графіку освітнього процесу, та не використовуються у інших сервісах – наприклад, терміни атестації та канікул.

Розглянемо, як буде виглядати структура бази даних Firestore мобільного додатку.

Firestore є документо-орієнтовною базою даних, що означає, що вона працює за принципом колекція-документ-колекція. Це означає, що першим об’єктом бази даних є колекція – структура, у якої є своя унікальна назва, і до якої входить певний набір окремих документів. Кожний документ має власний ідентифікатор та поля з властивостями. Але окрім властивостей, кожний окремий документ може містити свій власний набір колекцій, у яку може входити ще набір документів, і так далі. Знаючи те, як працює база даних, створимо структуру бази даних для програмного комплексу “Студент”.

Основною колекцією додатку буде колекція “Студент” (“*student*” у базі даних). До цієї колекції ми вносимо дані про користувачів додатку. Коли новий обліковий запис користувача створений, у цій колекції створюється новий документ. Так як декілька користувачів з однаковою адресою

електронної пошти бути не може, ідентифікатором документів буде e-mail користувача. Це не тільки покращить структурний вигляд бази даних, а спростить звертання до елементів бази даних конкретного користувача у програмному коді.

У додатку використовується перемикання між навчальними семестрами для розділення навчальних елементів по часових рамках. Отже, є сенс розділити ці навчальні дисципліни по колекціям, що репрезентують навчальні семестри. В свою чергу уже в кожній колекції певного семестру буде набір документів освітніх елементів, з полями інформації у кожного. Отже, у документі користувача буде створена колекція “семестр” (“*semester*” у базі даних). У ній буде згенеровані документи семестрів – “семестр х” (“*Семестер х*” у базі даних), де х – це номер семестру. Кожен з цих документів містить 2 колекції: освітні елементи (“*Courses*” у базі даних), у якій розташовані всі освітні елементи, кожен у окремому документі зі своїми характеристиками. Та події (“*Events*” у базі даних), у які вносяться події, що відмічаються у графіку навчального процесу, кожен у окремому документі.

Розглянемо, як можна реалізувати пов’язування цих елементів у самому додатку для максимальної зручності роботи з ними.

Якщо користувач створює новий освітній елемент на сторінці індивідуального навчального плану, елемент списку з’явиться на сторінці залікової книжки, у вигляді контейнера з назвою освітнього елемента та кнопкою, яка пропонує заповнити інформацію залікової книжки про цей елемент. Не заповнені елементи завжди показані у кінці сторінки (після усіх заповнених) та сортуються і групуються окремо від них. При натисканні на кнопку “Додати інформацію” відкриється діалогове вікно, з уже введеною назвою та всіма необхідними полями. Після збереження вся інформація відображається у списку елементів залікової книжки.

Аналогічно, коли освітній елемент додається до залікової книжки студента, він з’являється у відокремленій категорії “Не заповнені” на сторінці

індивідуального навчального плану, що йде після усіх заповнених елементів, незалежно від сортування.

При додаванні дати зі сторінки залікової книжки студента, можна відмітити прапорець “Додати як подію”. Якщо вона відмічена, то на сторінці графіку освітнього процесу цей освітній елемент з’явиться як подія.

При створенні елементу з графіку освітнього процесу, він з’являється на двох інших сторінках у якості незаповненого елементу. Створені події окремо зберігаються тільки для сервісу графіку та не використовуються у інших сервісах.

Отже, розділ 3 зосереджений на проєктуванні мобільного додатку "Студент". У ньому описаний функціонал трьох основних сервісів: електронного індивідуального навчального плану, електронної залікової книжки студента та електронного графіка освітнього процесу. Розглянуто способи трансформації паперових навчально-організаційних документів у функціонал мобільного додатку. Описано, як саме потрібно розробити цей додаток для комфортної та ефективної роботи користувача з ним. Виконано підготовчий етап розробки у вигляді планування кожної окремої функції усіх сервісів додатку.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ СТВОРЕННЯ ДОДАТКУ

Для створення мобільного додатку обрано такі технології та програмне забезпечення:

Фреймворк: Flutter 3.13.4;

Мова програмування: Dart 3.1.2;

Інструменти розробника: Flutter DevTools 2.25.0;

Середовище розробки: Android Studio, версії Giraffe 2022.3.1 Patch 2;

Емулятор: Android Studio | Android Emulator 31.3.14;

Девайс емулятора: Pixel 6 Pro (Android 13);

Інші інструменти: Firebase CLI 12.5.4; FlutterFire 0.2.7

Бібліотеки Flutter:

- firebase_core: 2.16.0;
- firebase_auth: 4.10.0;
- cloud_firestore: 4.9.2;
- intl: 0.18.1;
- table_calendar: 3.0.9.

4.1. Підготовчий етап

Для початку розробки необхідно визначитися – використовувати персональний комп’ютер з однією з актуальних операційних систем. У даному випадку використовувався комп’ютер на базі операційної системи Windows 10.

Перший крок – встановлення середовища розробки Android Studio. Android Studio Community Edition можна безкоштовно завантажити з офіційного сайту. Редактор Android Studio є в комплекті з набором розробника Android SDK, тому завантажувати його не потрібно. Однак, для роботи з Flutter у цьому середовищі, потрібно також встановити набір інструментів розробки Dart SDK, щоб мати можливість писати програми

використовуючи цю мову. Після чого необхідно встановити набір інструментів розробки Flutter. На Windows встановлення усіх цих інструментів відбувається через запускний програмний файл-інсталятор. Після їх встановлення та вибору у налаштуваннях середовища розробки Android Studio, для нас відкривається можливість створити новий Flutter-проєкт.

Середовище Android Studio підтримує функціонал емуляторів Android девайсів. Після вибору пункту меню “Створити новий емулятор”, можна обрати модель телефону, що буде використовуватись як віртуальний мобільний пристрій, на якому можна буде бачити зміни, які ми вносимо до мобільного додатку.

Перше, що необхідно зробити – встановити пакети, що будуть використовуватись у додатку. Це можна зробити, використовуючи термінал. В інтерфейсі Android Studio термінал можна знайти як окрему вкладку внизу. При введенні наступної команди, оновлюються пакети, що використовуються разом зі стандартним Flutter проєктом:

```
flutter pub upgrade
```

А після цієї команди, всі пакети ініціалізуються в проєкт:

```
flutter pub get
```

Отже, встановлюємо наступні пакети:

firebase_core – основний пакет для роботи бази даних Firebase у Flutter проєкті. Функціонал Firebase не буде працювати без цього пакету;

firebase_auth – пакет аутентифікації Firebase Authentication. Використовується у додатку для створення сторінок входу та реєстрації. Ім’я користувача, що повертає цей пакет, використовується у базі даних;

cloud_firestore – пакет бази даних Firebase Cloud Firestore. Основний пакет, що використовується для усіх викликів з програми до бази даних. Дозволяє отримувати, додавати, видаляти та змінювати інформацію у базі даних;

table_calendar – пакет календаря для Flutter. Використовується на сторінці графіку навчальних занять для кращої візуалізації подій;

intl – пакет локалізації та форматування тексту. Використовується для декількох невеликих задач по виводу тексту в інтерфейс.

Для їх встановлення почергово вводимо у термінал команди:

```
flutter pub add firebase_core
flutter pub add firebase_auth
flutter pub add cloud_firestore
flutter pub add table_calendar
flutter pub add intl
flutter pub get
```

Наступний важливий крок – встановлення сервісів Firebase. Для початку роботи з Firebase необхідно перейти на офіційний сайт firebase.google.com та зареєструватись за допомогою власного Google аккаунту. Якщо все вірно, то ми потрапляємо до косолі Firebase – сайту для управління нашими базами даних Firebase. Тут можна натиснути на “Create New Project”, щоб створити новий проєкт. Після введення назви, ми готові створити базу даних та базу аутентифікації.

Для встановлення Firebase у Flutter проєкт Google створили окремий інструмент – Firebase CLI. Ця програма для терміналу дозволяє просто та швидко інтегрувати Firebase сервіси для проєктів написаних на різних технологіях. Вона може бути встановлена через файл-інсталятор, або за допомогою команди npm (Node Package Manager) через наступну команду:

```
npm install -g firebase-tools
```

Після встановлення переходимо до терміналу проєкту в середовищі Android Studio та вводимо команду для авторизації користувача:

```
firebase login
```

Тепер необхідно активувати плагін підтримки Firebase CLI у Flutter проєкті під назвою FlutterFire.

```
dart pub global activate flutterfire_cli
```

У терміналі запускаємо програму конфігурації Flutter проєкту:

```
flutterfire configure
```

Виконуючи інструкції, що відображаються в терміналі, можна прив’язати такі сервіси Firebase як Authentication та Cloud Firestore. Після підтвердження, у проєкті буде новий файл *firebase_options.dart*, у якому вказані основні налаштування Firebase для нашого проєкту, включаючи ID проєкту та API ключ. Файл налічує клас DefaultFirebaseOptions, який буде використовуватись для взаємодії проєкту і бази даних.

Останній крок на підготовчому етапі – ініціалізація цього класу в додатку. Після цього програма зможе звертатись до бази даних, знаючи конкретно, який Firebase проєкт для цього використовується. Таким чином, фрагмент коду основного файлу нашого додатку *main.dart* виглядатиме так:

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import "firebase_options.dart";

void main() async {
  WidgetsFlutterBinding.ensureInitialized(); //ініціалізуємо рушій Flutter
  await Firebase.initializeApp(
    //ініціалізуємо з'єднання між Flutter додатком та Firebase
```

```
  options: DefaultFirebaseOptions
      .currentPlatform, //визначає платформу, що використовується
  );
  runApp(const MyApp());}
```

4.2. Розробка системи аутентифікації користувачів. Ініціалізація бази даних

Як було досліджено раніше, для прив'язки кожного окремого користувача до його сегменту бази даних, необхідно створити систему аутентифікації користувача: авторизацію і реєстрацію. У файлах проєкту створюємо нові файли: *auth_page.dart*, *login_page.dart*, *register_page.dart*, та *login_or_register_page.dart*.

В основному файлі проєкту main.dart, потрібно створити перенаправлення на клас *AuthPage*. Це означає, що першочергово при запуску додаток направляє користувача на сторінку авторизації, на якій вже буде вирішуватись, куди направити користувача далі в залежності від умов – чи був користувач вже авторизований, чи ще ні. Також імпортуємо файл *auth_page.dart*, щоб *main.dart* зміг знайти його в проєкті. Фрагмент коду:

```
import 'pages/auth/auth_page.dart';

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: AuthPage(),
    );
  }
}
```

У файлі *auth_page.dart* необхідно перевірити, чи є користувач авторизованим, чи ні. Це можна зробити за допомогою функції

authStateChanges, з бібліотеки FirebaseAuth. Вона повертає потік даних вказаного користувача, який можна реалізувати завдяки побудовнику потоків Flutter – SreamBuilder. Цей віджет неодноразово використовується у проєкті, адже дозволяє працювати з інформацією, що потрапляє до програми у реальному часі. Такою інформацією є база даних Firebase. Отже, якщо програма знайшла дані поточного користувача у базі користувачів Firebase Authentication, і там вказано, що такий користувач є авторизованим у додатку, програма направить його на головну сторінку додатку з меню вибором функцій – *home_page*. Якщо ж виявиться, що користувач не авторизований, буде виконаний файл *login_or_register_page*, який контролює сторінки авторизації та реєстрації. Код файлу *auth.dart* виглядає так:

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import '../home_page.dart';
import 'login_or_register_page.dart';

class AuthPage extends StatelessWidget {
  const AuthPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder<User?>(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return HomePage();
          }
          return LoginOrRegisterPage();
        },
      ),
    );
  }
}
```

Файли *login_page.dart* та *register_page.dart* – відповідають за відображення сторінки авторизації та реєстрації відповідно. Необхідно надати можливість користувачу швидко переключатись з однієї сторінки на іншу. Для цього на сторінці реєстрації є посилання “Увійти”, що направляє користувача на сторінку авторизації та відповідно на сторінці входу є посилання “Створити новий аккаунт”, що направляє на сторінку реєстрації. Функція переключення на інший екран реалізована на сторінці *login_or_register_page.dart*. Фрагмент коду:

```
void toggleScreens() {
  setState(() {
    showLoginPage =
        !showLoginPage;
  });
}
@override
Widget build(BuildContext context) {
  if (showLoginPage) {
    return LoginPage(
      onTap: toggleScreens,
    );
  } else {
    return RegisterPage(onTap: toggleScreens);
  }
}
```

У цьому коді, змінна *showLoginPage* контролює те, який екран буде показаний на даний момент, і кожен раз, коли викликається функція *onTap* на сторінці реєстрації та авторизації, її значення змінюється на протилежне.

На сторінці login_page ми викликаємо функцію з бібліотеки FirebaseAuth – *signInWithEmailAndPassword*. Ми передаємо їй значення, що користувач вводить у текстові поля програми та намагається знайти підходящі дані для авторизації у базі. Якщо вони підходять, користувач успішно ввійде в аккаунт та потрапить до головної сторінки.

Фрагмент коду:

```
void signUserIn(BuildContext context) async {
  try {
```

```
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: emailController.text,
      password: passwordController.text,
    );
  } catch (error) {
    print(error);
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text("Помилка! Неправельний логін або пароль!"),
      ),
    );
  }
}
```

При реєстрації потрібно не тільки впевнитись, що пароль введений коректно, а й що паролі у текстових полях дійсно збігаються. Фрагмент коду:

```
void signUserUp(BuildContext context) async {
  try {
    if (passwordController.text == confirmPasswordController.text) {
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: emailController.text,
        password: passwordController.text,
      );
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Паролі не збігаються!"),
        ),
      );
    }
  } catch (error) {
    print(error);
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content:
            Text("Помилка реєстрації! Введіть коректні e-mail та пароль!"),
      ),
    );
  }
```

```
}
```

Таким чином у додатку реалізована аутентифікація. Після успішної авторизації, користувач потрапляє до головного меню застосунку.

Перед розробкою функціоналу додатку необхідно зробити ініціалізацію бази даних, створивши структуру файлів Firestore, яка нам потрібна. В першу чергу потрібно створити основну колекцію “студент”, у якій будуть сворюватись документи кожного користувача. У документів студента повинно бути поле поточного семестру (“*Current Semester*” у базі даних). Для реалізації цього, створимо новий файл *database_service.dart*. У ньому створюємо клас *DatabaseService*, у якому будуть знаходитись функції для роботи з базою даних Firestore, які можна викликати з різних точок програми. У цьому класі створюємо наступний метод:

```
static Future<void> createStudentDocument(var user) {
  //шлях до документу "student" - "%user%"
  final docRef = FirebaseFirestore.instance.collection("student").doc(user);
  final Map<String, dynamic> studentData = {
    'emailField': user,
    'currentSemesterField': "Семестер 1"
  };
  return docRef.set(studentData).then((_) {
    return createSemesterCollection(user);
  });
}
```

У цьому фрагменті коду ми спочатку використовуємо ключове слово static для того, щоб звертатись до цього методу з будь-якого файлу програми, не створюючи новий екземпляр класу. Цей метод приймає значення поточного користувача, щоб використати його логін (адресу електронної пошти) як назву документа у базі даних. Використовуючи метод FirebaseFirestore.instance.set із бібліотеки cloud_firestore ми можемо ставити значення потрібних полів документу, які ми передаємо як змінні типу

Map<String, dynamic>. Завдяки ключовому слову then ми робимо так, що після завершення виконання цього методу ми одразу викликаємо наступний метод. Метод *createSemesterCollection* виглядає таким чином:

```
static Future<void> createSemesterCollection(var user) {
 List<Future> tasks = [];
 for (int i = 1; i <= 10; i++) {
  tasks.add(FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection('semester')
      .doc('Семестер $i')
      .set({}));
 }
 return Future.wait(tasks);
}
```

У цьому фрагменті видно, як спочатку створюється колекція семестрів, а у ній у вигляді документів створюється список семестрів – за замовчуванням їх 10. Ініціалізація повинна проводитись при створенні облікового запису користувача, отже необхідно додати цей фрагмент коду до функції реєстрації:

```
    await DatabaseService.createStudentDocument(
      emailController.text);
```

Цей фрагмент коду викликає метод ініціалізації бази даних.

Таким чином, був створений функціонал аутентифікації у додаток. Інтерфейс сторінок авторизації та реєстрації подано на рисунку 4.1.

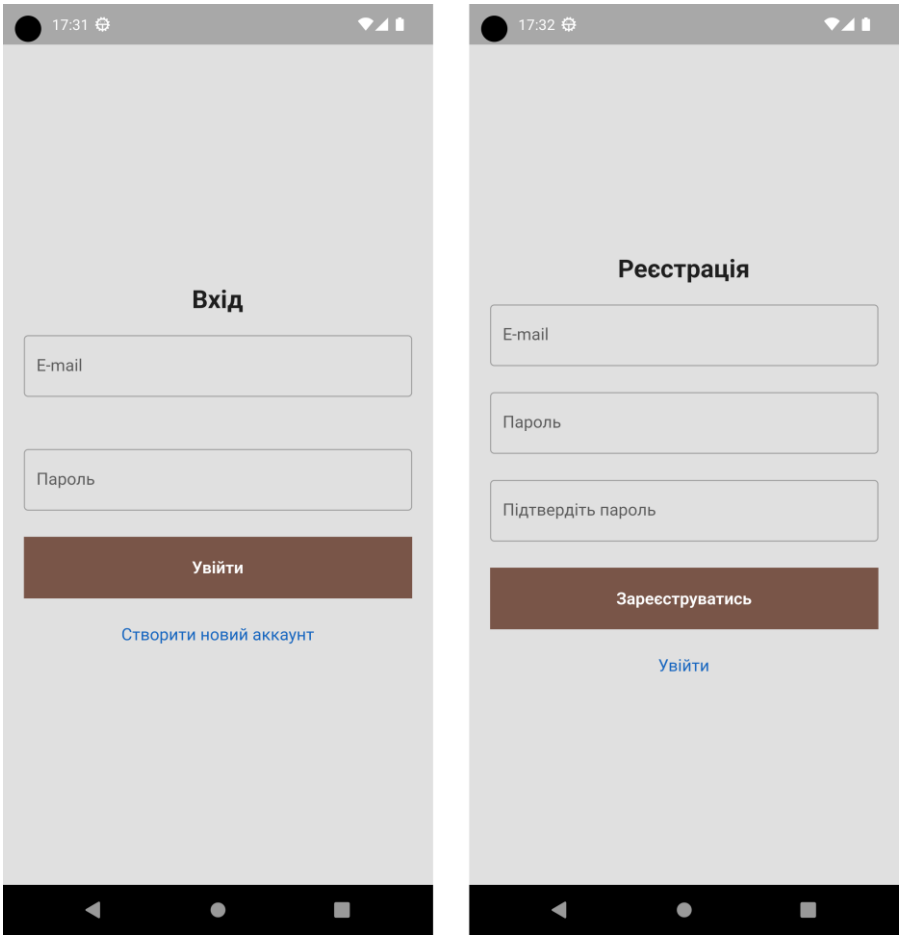

Рис. 4.1. Сторінки авторизації та реєстрації

4.3. Розробка сервісу “Індивідуальний навчальний план”

Для розробки сервісу “Індивідуальний навчальний план”, потрібно створити новий файл *courses_schedule_page.dart*. Це сторінка, на якій буде функціонал цього електронного документу.

В зв’язку з тим, що додаток буде окремо відображати інформацію про кожен семестр, має сенс зробити перемикання списку елементів в залежності від обраного семестру. Реалізуємо це у вигляді випадного меню зі списком семестрів, після зміни обраного значення на якому на екран виведеться список елементів для відповідного семестру.

У директорії програмних файлів додатку “lib”, створимо папку “*components*”, у якій будуть розміщені компоненти – окремі віджети – що будуть неодноразово використовуватись у додатку. Створюємо нові файли – *dropdownmenu_choose_semester.dart*, та *dropdownmenu_design.dart*. У першому розміщується реалізація випадного меню вибору семестру, у другому – дизайн меню. Клас *DropdownMenuChooseSemester* використовує віджет FutureBuilder, що дозволяє створити список елементів тільки тоді, коли він отримує всі необхідні дані або з мережі, або з інших функцій додатку. Доки інформація не була отримана, віджет буде відображатись як кільце завантаження. Для отримання списку семестрів користувача, створюємо нову функцію у класі *DatabaseService*, яка поверне список усіх документів колекції “Семестри”. Фрагмент коду:

```
static Future<List<String>> getSemesterList(var user) async {
  QuerySnapshot snapshot = await FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection('semester')
      .get();
  List<String> semesterList = snapshot.docs.map((doc) => doc.id).toList();
  return semesterList;
}
```

Як видно з цього фрагменту коду, необхідно спочатку асинхронізовано передати ім’я користувача (щоб функція знайшла саме його список семестрів), і тільки після цього теж асинхронізовано повернула дані. Отже, у класі випадного меню, присвоюємо значення локальної змінної до інформації з бази даних, використовуючи ключове слово late. Фрагмент коду:

```
late Future<List<String>> semesterList =
    DatabaseService.getSemesterList(user.email);
```

При завантаженні сторінки, вибраним значенням випадного меню повинно бути значення поточного семестру, обраного користувачем у головному меню додатку. Створюємо функцію отримання цього значення з бази даних у класі *DatabaseService*. Фрагмент коду:

```
static Future<String> getStudentField(var user, String field) async {
  DocumentSnapshot snapshot =
      await FirebaseFirestore.instance.collection('student').doc(user).get();
  if (snapshot.exists) {
    Map<String, dynamic> data = snapshot.data() as Map<String, dynamic>;
    return data[field];
  } else {
    return '';
  }
}
```

Важливо, щоб це значення викликалось тільки при першому завантажені цього меню, а далі залежало від значення поточного семестру на самій сторінці сервісу. Отже, для поточного значення семестру у випадному меню присвоюємо таке значення:

```
late Future<String> currentSemester = widget.initialSemester != null
    ? Future.value(widget.initialSemester)
    : DatabaseService.getStudentField(user.email, 'Current Semester');
```

Реалізацію випадного меню додатку можна побачити у наступному фрагменті коду:

```
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: semesterList,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Center(child: CircularProgressIndicator());
        } else if (snapshot.hasError) {
          return Center(child: Text('Помилка: ${snapshot.error}'));
        } else if (snapshot.hasData) {
          List<String>? dataList = snapshot.data;
          return FutureBuilder<String?>(
            future: currentSemester,
            builder: (BuildContext context, AsyncSnapshot<String?> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const Center(child: CircularProgressIndicator());
              } else if (snapshot.hasError) {
                return Center(child: Text('Помилка: ${snapshot.error}'));
              } else {
                return DropdownMenuDesign(
                  items: dataList!,
                  selectedItem: snapshot.data,
                  onChanged: (selectedItem) {
                    setState(() {
                      currentSemester = Future.value(selectedItem);
                    });
                    widget.onSelectedItemChanged(selectedItem!);
                  },
                );
              }
            },
          );
        }
        return const Center(child: CircularProgressIndicator());
      },
    );
  }
}
```

Коли цей віджет будується, він спочатку перевіряє стан Future, який називається *semesterList*. Якщо *semesterList* все ще завантажується, він показує індикатор завантаження (CircularProgressIndicator). Якщо виникає помилка, він відображає повідомлення про помилку. Коли дані стають доступними, він переходить до іншого FutureBuilder, вкладеного всередину, який працює з *currentSemester*, іншим Future. Цей внутрішній FutureBuilder показує індикатор завантаження та відображає помилку, якщо вона виникає, і, нарешті, коли дані стають доступними, він показує випадне меню (DropdownMenuDesign), заповнене елементами з *dataList* та вибраним елементом з *currentSemester*. Випадне меню дозволяє користувачеві вибрати елемент, який оновлює *currentSemester* і викликає зворотний виклик (*onSelectedItemChanged*) із вибраним елементом. Отже, тепер це меню можна використовувати на сторінці індивідуального навчального плану.

Наступним кроком буде виведення інформації на екран. Ефективним способом зчитати велику кількість інформації певного типу з бази даних буде створення класу освітнього елементу. Для цього потрібно створити новий файл *course_class.dart* з класом *Course*. У ньому будуть ініціалізовані параметри та реалізовані функції переводу вхідних значень у json формат, та назад з json формату. Це дозволить створити функції зчитування та занесення інформації у базу даних Firestroe, що може використовувати формат json для документів у базі даних. Це показано з частини програмного коду:

```
class Course {
  String? nameField;
  num? hoursLectionsField;
  num? hoursPracticesField;
    // продовження коду
  Course({
    this.nameField,
    this.hoursLectionsField,
    this.hoursPracticesField,
    // продовження коду
  }) {
```

```
  hoursLectionsField = hoursLectionsField ?? 0;
  hoursPracticesField = hoursPracticesField ?? 0;
  hoursLabsField = hoursLabsField ?? 0;
  // продовження коду
}
Map<String, dynamic> toJsonCourse() {
  return {
    'Name': nameField,
    'Hours Lections': hoursLectionsField,
    'Hours Practices': hoursPracticesField,
    // продовження коду
  };
}
Course.fromJsonCourse(Map<String, dynamic> json) {
  nameField = json['Name'];
  hoursLectionsField = json['Hours Lections'];
  // продовження коду
}
```

Використовуючи цей клас, можна створити методи для зчитування інформації з бази даних та занесення їх туди ж. Створюємо наступний метод для отримування даних з бази даних у класі *DatabaseService*:

```
static Future<List<Course>> getAllCourses(
    String userEmail, String semester) async {
  QuerySnapshot querySnapshot = await FirebaseFirestore.instance
      .collection("student")
      .doc(userEmail)
      .collection("semester")
      .doc(semester)
      .collection("Courses")
      .get();
  if (querySnapshot.docs.isNotEmpty) {
    return querySnapshot.docs
        .map((doc) =>
            Course.fromJsonCourse(doc.data() as Map<String, dynamic>))
        .toList();
  } else {
    return [];
```

```
  }
 }
```

Тепер цей метод можна використати для генерації списку дисциплін. Фрагмент коду:

```
 Future<List<Course>> fetchCourses(String semester) async {
  return await DatabaseService.getAllCourses(user.email!, semester);
 }

Future<List<Map<String, dynamic>>> generateCourses(String semester) async {
  List<Course> courses = await fetchCourses(semester);
  List<Course> filledCourses = [];
  List<Course> unfilledCourses = [];

  for (var course in courses) {
   if (course.isScheduleFilled == true) {
    filledCourses.add(course);
   } else {
    unfilledCourses.add(course);
   }
  }

  filledCourses = sortCourses(filledCourses);
  unfilledCourses = sortCourses(unfilledCourses);

  courses = [...filledCourses, ...unfilledCourses];
  expandedState = setExpandedState(courses);

  return courses.map((course) {
   return {
    'course': course,
    'isExpanded': !(course.isScheduleFilled ?? false),
   };
  }).toList();
 }
```

У цьому фрагменті коду проводиться збір даних з бази даних, після чого інформація трансформується у формат, який необхідний для

відображення даних на екрані. Також проводяться інші трансформації, такі як сортування даних, що здійснюється викликом функції *sortCourses* та проводиться розділення елементів на категорії – заповнені елементи та незаповнені елементи, що репрезентується викликом функції *setExpandedState*.

Отримавши інформацію з бази даних, маємо все, щоб відобразити цю інформацію на екрані. Для цього використовується FututreBuilder, що створить певний список елементів. Імплементацією списку елементів буде набір плиток, що можуть згортатись та розгортатись. У Flutter ці плитки репрезентуються віджетом Tile. Відповідно, списком таких плиток буде віджет ListTile. У результаті ми отримуємо віджет *_coursesListView*, який відповідає за відображення інформації на екрані. Цей віджет отримує як дані відформатований список елементів та їх значень і генерує відповідну до кількості елементів кількість плиток. Він же відповідає за дизайн згорнутої плитки: на плитці присутня кнопка згортання/розгортання, назва освітнього елементу, кнопка редагування та кнопка видалення елементу. При розгортанні плитки, віджет посилається на функцію *_courseDetails*, яка повертає дизайн заповненого та незаповненого елементу в залежності від параметрів.

У розгорнутому вигляді для заповнених освітніх елементів вказана основна інформація про освітній елемент згідно з постановкою задачі: назва, форма підсумкового контролю, лекційні години, практичні/семінарські години, лабораторні години, курсові години, аудиторні години, години на самостійну роботу, години усього, та кількість навчальних кредитів. У незаповнених елементів є лише кнопка, що пропонує заповнити інформацію про цей елемент у індивідуальному навчальному плані.

Отже, на сторінці присутній список елементів та меню вибору семестру. Для того, щоб реалізувати оновлення сторінки даними відповідного семестру, використовується метод *updateSelectedSemester*. Фрагмент коду:

```
void updateSelectedSemester(String selectedItem) {
  setState(() {
    selectedSemester = selectedItem;
    coursesFuture = generateCourses(selectedItem);
    coursesFuture.then((courses) {
      setState(() {
        expandedState = courses
            .map((course) => !(course['course'].isScheduleFilled ?? false))
            .toList();
      });
    });
  });
}
```

Він використовується з виклику самого випадного меню, щоб користуватись обраним поточним значенням як параметром.

Наступна функція, яка повинна бути на сторінці – це функція додання нового елементу. Створюємо новий файл *new_course_dialog.dart* – у ньому буде реалізована функція введення інформації користувачем та відправлення її у базу даних. Це буде діалогове вікно, яке буде відкриватись при натисканні кнопки “Додати новий елемент” та буде показувати поля для вводу інформації за постановкою задачі. На рисунку 4.2. можна побачити вигляд меню додавання освітнього елементу з набором текстових полів та підказками (hints) про те, яка інформація повинна вноситись. Вибір форми підсумкового контролю зроблено у вигляді випадного меню, з опціями “Екзамен”, “Залік” та “Інше”. Унизу присутні кнопки “Закрити” – закрити вікно без збереження змін, та “Зберегти” – зберегти дані у базі даних.

Рис. 4.2. Діалогове вікно "Додати новий освітній елемент"

Варто зазначити, що такі значення як "Усього аудиторних годин", "Усього годин" та "Кількість кредитів ЄКТС" не вказуються користувачем, а рахуються при доданні інформації у базу даних. За це відповідають наступні функції класу *Course*:

```
num calculateTotalHoursInClass() {
  return (hoursLectionsField ?? 0) +
      (hoursPracticesField ?? 0) +
      (hoursLabsField ?? 0) +
      (hoursCourseworkField ?? 0);
}

num calculateTotalHoursOverall() {
  return (hoursInClassTotalField ?? 0) + (hoursIndividualTotalField ?? 0);
```

```
}

num calculateTotalCredits() {
  if (hoursOverallTotalField == null) {
    return 0;
  } else {
    return num.parse((hoursOverallTotalField! / 30).toStringAsFixed(2));
  }
}
```

Тут, кількість аудиторних годин – це сума введених користувачем годин лекцій, практичних, лабораторних та курсових; сума усього – це кількість аудиторних годин плюс самостійна робота студента, а кількість кредитів – загальна кількість годин розділена на 30 (1 кредит ЄКТС = 30 годин).

Тепер, використовуючи клас *Course*, можна створити функцію для зміни полів освітніх елементів у базі даних. Фрагмент коду:

```
static Future<void> createOrUpdateCourse(
    var user, Course course, String semester) async {
  if (course.nameField!.trim().isEmpty) {
    throw Exception('Назва курсу не може бути порожньою!');
  }
  final docRef = FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection("semester")
      .doc(semester)
      .collection("Courses")
      .doc(course.nameField);
  return docRef.set(course.toJsonCourse());
}
```

З цього коду видно, що функція шукає по заданому шляху бази даних документи, назва яких збігається з назвою дисципліни яку передають до неї при виклику, та викликає функцію *toJsonCourse* з класу, яка спочатку трансформує дані у формат json, а потім передає їх у базу даних.

Таким чином, можна використати цей метод у діалоговому вікні, щоб зберегти інформацію про елемент, яку користувач вводить у текстові поля. Фрагмент цього коду виглядає так:

```
class _NewCourseDialogState extends State<NewCourseDialog> {
  final user = FirebaseAuth.instance.currentUser!;
  final nameFieldController = TextEditingController();
  final hoursLectionsFieldController = TextEditingController();
  final hoursPracticesFieldController = TextEditingController();
  //…

  TextButton _saveButton(BuildContext context) {
    return TextButton(
      child: const Text('Зберегти'),
      onPressed: () async {
        try {
          Course newCourse = Course(
            nameField: nameFieldController.text,
            hoursLectionsField:
                int.tryParse(hoursLectionsFieldController.text) ?? 0,
            hoursPracticesField:
                int.tryParse(hoursPracticesFieldController.text) ?? 0,
                //…

          await DatabaseService.createOrUpdateCourse(
              user.email, newCourse, selectedSemesterPage!);
```

З коду видно, що інформація яку вводить користувач у текстові поля, зберігається у змінних типу **Controller*, записується як об’єкт класу Course та

відправляється як параметр у базу функцію, з якої вона перетворюється у інформацію для бази даних.

Окрім функціоналу додавання елементу, це діалогове вікно також повинно відповідати за редагування уже готового елементу зі списку. Для цього додаємо булеву змінну *isEdit* до класу діалогового вікна, яка буде передаватися як параметр при виклику вікна з кнопки редагування. Це дозволить розділити функціонал класу в залежності від того, яка саме дія потрібна. В першу чергу потрібно зробити так, щоб при редагуванні текстові поля були заповнені значеннями цього елементу з бази даних. Ініціалізуємо їх таким чином:

```
@override
void initState() {
  super.initState();
  selectedSemesterPage = widget.currentSemester;
  if (widget.isEdit && widget.course != null) {
    nameFieldController.text = widget.course!.nameField ?? '';
    hoursLectionsFieldController.text =
        widget.course!.hoursLectionsField.toString();
    hoursPracticesFieldController.text =
        widget.course!.hoursPracticesField.toString();
//…
```

Отже, можна викликати меню редагування з кнопки редагування на кожному окремому елементі. У цьому меню можна редагувати значення, і після натискання кнопки “Зберегти”, вони зміняться і в базі даних. Вигляд вікна редагування можна побачити на рисунку 4.3.

Як відомо з постановки задачі, сторінка повинна показувати список елементів з ще не заповненими даними про індивідуальний навчальний план. Окремо створимо новий вид плитки для таких елементів. На ній буде лише кнопка “Додати інформацію”. Для виклику цього варіанту діалогового вікна редагування, потрібно створити новий параметр *isEditFilling*, який буде передаватись курсу при виклику з цієї кнопки.

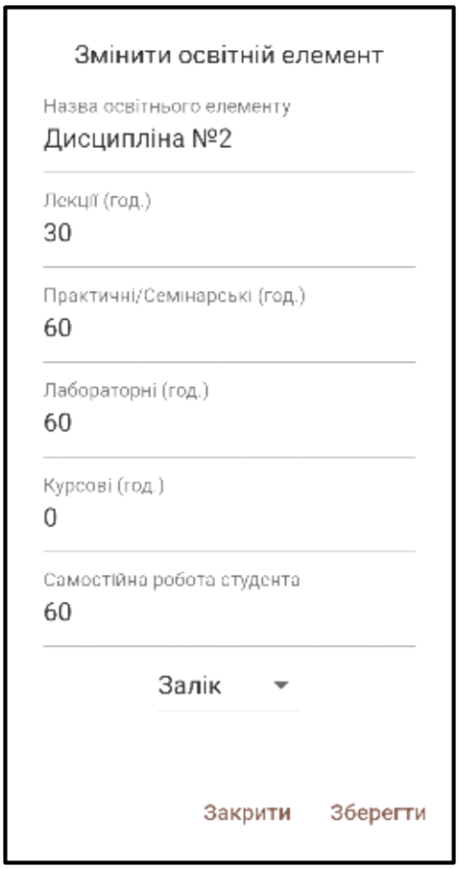

Рис. 4.3. Діалогове вікно "Змінити освітній елемент"

Для кожного елементу присутня функція видалення. При натисканні на кнопку видалення, відкривається нове діалогове вікно з пропозицією видалити інформацію про елемент зі сторінки індивідуального навчального плану, або повністю видалити елемент з бази даних. Для видалення елементу з бази даних, створимо нову функцію у класі *DatabaseService*. Фрагмент коду:

```
static Future<void> deleteCourse(var user, String courseName) async {
  try {
    var currentSemester = await getStudentField(user, 'Current Semester');
    await FirebaseFirestore.instance
        .collection("student")
        .doc(user)
        .collection("semester")
        .doc(currentSemester)
        .collection("Courses")
```

```
        .doc(courseName)
        .delete();
    } catch (e) {
      rethrow;
    }
  }
```

З коду видно, як здійснюється пошук документа з назвою, що відповідає назві освітнього елементу, та видаляємо його.

Останньою функцією цього сервісу є сортування елементів списку, яке реалізовано у вигляді випадного меню, після вибору типу сортування на якому змінюється порядок відображення освітніх елементів на екрані згідно певних параметрів. Опції сортування: “За алфавітом (А до Я)”, “За алфавітом (Я до А)”, “За аудиторними годинами”, “За індивідуальними годинами”, “За годинами загалом”. Створюємо випадне меню сортування з цими опціями. При виборі певного елементу меню, викликається відповідна функція сортування. Код функції сортування:

```
List<Course> sortCourses(List<Course> courses) {
  courses.sort((a, b) {
    if (a.isScheduleFilled == true && b.isScheduleFilled != true) {
      return -1;
    } else if (b.isScheduleFilled == true && a.isScheduleFilled != true) {
      return 1;
    } else {
      if (sortOption == SortOption.alphabeticalAsc) {
        return (a.nameField ?? "").compareTo(b.nameField ?? "");
      } else if (sortOption == SortOption.alphabeticalDesc) {
        return (b.nameField ?? "").compareTo(a.nameField ?? "");
      } else if (sortOption == SortOption.hoursInClassDesc) {
        return (b.hoursInClassTotalField?.toInt() ?? 0)
            .compareTo(a.hoursInClassTotalField?.toInt() ?? 0);
      } else if (sortOption == SortOption.hoursIndividualDesc) {
        return (b.hoursIndividualTotalField?.toInt() ?? 0)
```

```
            .compareTo(a.hoursIndividualTotalField?.toInt() ?? 0);
      } else if (sortOption == SortOption.hoursOverallDesc) {
        return (b.hoursOverallTotalField?.toInt() ?? 0)
            .compareTo(a.hoursOverallTotalField?.toInt() ?? 0);
      } else {
        return 0;
      }
    }
  });

  return courses;
}
```

Ця функція сортує список об'єктів *Course* на основі вказаних критеріїв. Спочатку вона віддає пріоритет *Course*, де *isScheduleFilled* == true (є істиним), та переміщує їх на початок списку. Потім вона сортує на основі значення *sortOption*. Якщо *sortOption* встановлено на *alphabeticalAsc*, вона сортує *Course* за алфавітом за їхнім полем *nameField*. Для *alphabeticalDesc* – сортування у зворотному порядку. Якщо опція сортування вибрана як сортування за годинами (*hoursInClassDesc*, *hoursIndividualDesc*, *hoursOverallDesc*), вона сортує *Course* на основі відповідних полів годин від більшого до меншого. У разі, якщо *sortOption* не відповідає жодній з цих умов, порядок *Course* залишається незмінним. Після сортування функція повертає змінений список *Course* (List< Course>). Він використовується методом *generateCourses* для розташування їх у необхідному порядку.

Таким чином, після розробки коду, ми отримали готовий сервіс “Індивідуальний навчальний план”, який приймає та відображає інформацію про навчальне навантаження та іншу інформацію про освітні елементи студента. На рисунку 4.4. показаний приклад інтерфейсу додатка на цій сторінці.

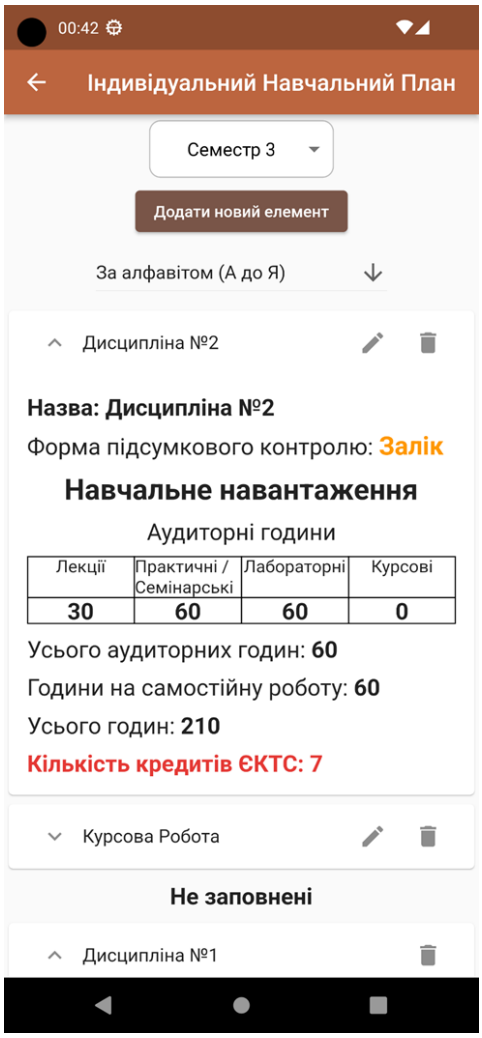

Рис. 4.4. Приклад роботи сервісу "Індивідуальний навчальний план"

4.4. Розробка сервісу "Залікова книжка студента"

Для розробки сервісу "Залікова книжка студента", потрібно створити новий файл *record_book_page.dart*. Це буде сторінка з функціоналом цього електронного документу.

Цей документ має багато спільних елементів з екраном індивідуального навчального плану: випадне меню семестрів, кнопка додавання нового навчального елементу, меню вибору сортування та список освітніх елементів. Основна відмінність від минулого сервісу – на цьому екрані список елементів буде відображатись не у вигляді плиток, які згортаються, а у вигляді контейнерів з інформацією.

Для сервісів індивідуального навчального плану та залікової книжки створимо систему, за якою при створенні елементу зі сторінки навчального плану, він з’явиться на сторінці залікової книжки як незаповнений елемент, який пропонується заповнити, і навпаки. Додамо відповідні технічні поля у клас Course, щоб значення заповнення елементів на сторінках запам’ятовувались та прив’язувались окремо до кожного елементу. Фрагмент коду:

```
class Course {
  //…
  bool? isScheduleFilled;
  bool? isRecordBookFilled;
  //…
Course({
  //…
    this.isScheduleFilled,
    this.isRecordBookFilled,
  //…
  }

  Map<String, dynamic> toJsonCourse() {
    return {
      //…
      '(app) isScheduleFilled': isScheduleFilled,
      '(app) isRecordBookFilled': isRecordBookFilled,
      //…
  }
  Course.fromJsonCourse(Map<String, dynamic> json) {
//…
    isScheduleFilled = json['(app) isScheduleFilled'];
    isRecordBookFilled = json['(app) isRecordBookFilled'];
//…
  }
}
```

Таким чином, ці значення можуть бути передані при виклику функції *new_course_dialog*, і в залежності від того, звідки це діалогове вікно було викликано, воно змінить ці технічні поля на потрібні, а сторінки сервісів використають їх для відображення даних. Щоб проконтролювати, з якого сервісу було викликано діалогове вікно, використовується ще одна змінна *isRecordBook*, від значення якої залежить, який функціонал у нього буде.

Отже, щоб зробити так, щоб при додаванні нового елементу значення наповненості у базі даних змінювалось коректно, маючи на увазі можливість редагування елементів, у класі діалогового вікна створюємо функції, які перевірятимуть чи потрібно змінювати значення змінних заповнення. Фрагмент коду:

```
bool getIsScheduleFilled() {
  if (widget.isEdit == true && widget.isEditFilling == false) {
    return widget.filledCourseSchedule;
  } else {
    if (widget.isEdit == false && widget.isRecordBook == false) {
      return true;
    } else {
      return widget.filledCourseSchedule;
    }
  }
}

bool getIsRecordBookFilled() {
  if (widget.isEdit == true && widget.isEditFilling == true) {
    if (widget.isRecordBook == true) {
      return true;
    } else {
      return widget.filledNewRecordBook;
    }
  } else {
    if (widget.isEdit == false && widget.isRecordBook == true) {
```

```
      return true;
    } else {
      return widget.filledNewRecordBook;
    }
   }
  }
```

У якості полів на цій сторінці використовуються такі як: назва елементу, форма підсумкового контролю, ім’я викладача, оцінка та дата оцінювання. При натисканні на копку вибору дати, відкривається діалогове меню Android з вибором дати та часу. Справа від цієї кнопки розташована кнопка очистки дати. Окрім цього на ній розміщена кнопка з прапорцем: якщо обрана дата, її можна поставити, та таким чином додати цей елемент у інший документ – графік освітнього процесу. Вигляд меню додання освітнього елементу для сторінки залікової книжки студента показано на рисунку 4.5.

Серед інших змін, в порівнянні з інтерфейсом індивідуального навчального плану, є присутність прапорця “Групувати за формою підсумкового контролю”, при обиранні якого по порядку відобразяться екзамени, заліки та інші, кожен зі своєю категорією. Групування також бере до уваги сортування елементів, що значить що для кожної окремої категорії сортування буде окремим (між собою).

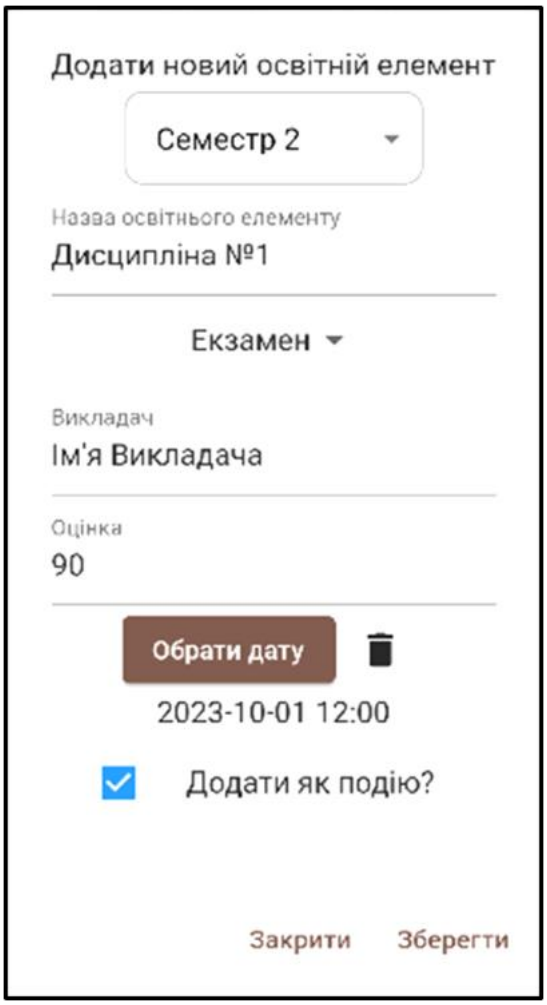

Рис. 4.5. Діалогове вікно "Додати новий освітній елемент" у сервісі "Залікова книжка студента"

Фрагмент коду:

```
List<Course> sortCourses(List<Course> courses) {
  //…
  // базове сортування…
 if (isGroupedByScoringType) {
  courses.sort((a, b) {
    int aValue = a.scoringTypeField == 'Екзамен'
        ? 1
        : a.scoringTypeField == 'Залік'
            ? 2
            : 3;
    int bValue = b.scoringTypeField == 'Екзамен'
        ? 1
        : b.scoringTypeField == 'Залік'
```

```
            ? 2
            : 3;
      return aValue.compareTo(bValue);
    });
  }
  courses.sort((a, b) {
    if (a.isRecordBookFilled == b.isRecordBookFilled) {
      return 0;
    } else if (a.isRecordBookFilled == true) {
      return -1;
    } else {
      return 1;
    }
  });

  return courses;
}
```

З коду видно, що якщо прапорець *isGroupedByScoringType* має значення true, ця функція додатково сортує курси на основі їхнього типу контролю, віддаючи пріоритет 'Екзамену', за ним 'Заліку', а потім іншим. Це досягається шляхом присвоєння числових значень кожному типу оцінювання та порівняння цих значень. Після чого курси групуються на основі того, чи є вони заповненими. Заповнені записи у журналі мають пріоритет.

Загалом, реалізація цього сервісу багато у чому збігається з попереднім, основні зміни у дизайні сторінки можна відслідкувати на рисунку 4.6.

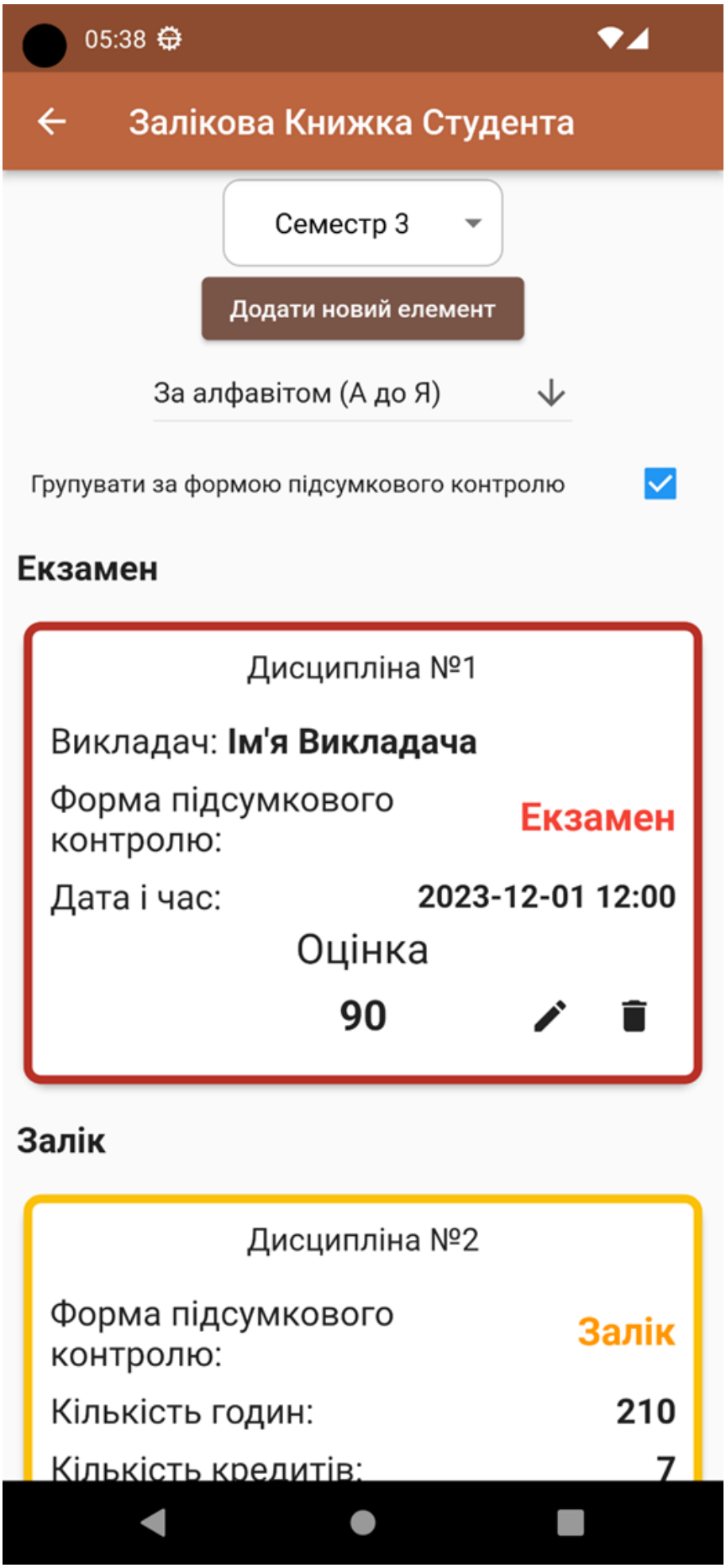

Рис. 4.6. Приклад роботи сервісу "Залікова Книжка Студента "

4.5. Розробка сервісу “Графік освітнього процесу”

Для розробки сервісу “Графік освітнього процесу”, потрібно створити новий файл *class_schedule_page.dart*. Це буде сторінка з функціоналом цього електронного документу.

Як зазначалося на етапі проєктування додатку, для реалізації функціоналу буде ефективним використати віджет календаря для відображення подій на екрані. У стандартну бібліотеку віджетів не входить віджет календаря, тому для вирішення цієї задачі використано сторонній віджет календаря TableCalendar.

TableCalendar – це популярний пакет для Flutter, що надає можливість додати функціональний календар на сторінку Flutter проєкту, який добре піддається користувацьким змінам. Він буде використаний на сторінці цього сервісу. Окремо для цього календаря створюємо файл my_calendar.dart. Згідно з документації цього пакету, найпростіша реалізація коду календаря виглядає таким чином:

```
Widget build(BuildContext context) {
  return TableCalendar(
    firstDay: DateTime.utc(2010, 10, 16),
    lastDay: DateTime.utc(2030, 3, 14),
    focusedDay: _focusedDay,
    calendarFormat: _calendarFormat,
    selectedDayPredicate: (day) {
      return isSameDay(_selectedDay, day);
    },
    onDaySelected: (selectedDay, focusedDay) {
      setState(() {
        _focusedDay = focusedDay;
        _selectedDay = selectedDay;
      });
    },
```

```
      onFormatChanged: (format) {
        if (_calendarFormat != format) {
          setState(() {
            _calendarFormat = format;
          });
        }
      },
      onPageChanged: (focusedDay) {
        _focusedDay = focusedDay;
      },
    );
  }
}
```

Цей код створює базовий календар із можливістю вибору дня, зміни формату календаря (місяць, два тижні або тиждень) та навігації між сторінками. Параметри *firstDay* та *lastDay* визначають діапазон дат календаря. *focusedDay* – це початковий відображуваний день. Функція *selectedDayPredicate* визначає, який день наразі вибраний. Зворотні виклики *onDaySelected*, *onFormatChanged* та *onPageChanged* дозволяють реагувати на взаємодію користувача.

В першу чергу необхідно перекласти календар українською. Для цього додаємо параметр локалізації, та змінюємо написи на кнопці вибору вигляду календаря:

```
//…
locale: 'uk_UA',
availableCalendarFormats: const {
  CalendarFormat.month: 'Місяць',
  CalendarFormat.twoWeeks: '2 Тижні',
  CalendarFormat.week: 'Тиждень',
},
//…
```

На календарі ми хочемо відображати такі елементи як “освітній елемент”, та “події”. На сторінці графіку освітнього процесу, створюємо дві

окремі кнопки з відповідним функціоналом. При натисканні на “Додати новий елемент”, відкривається діалогове вікно *new_course_dialog.dart*, що відображає такі поля для вводу: семестр, назва, тип, дата. Для виклику саме цих полів був створений новий параметр класу *NewCourseDialog* – *isClassSchedule*, що передається при виклику функції.

Щоб реалізувати створення нової події, потрібно створити аналогічні до *Course*: клас *Event* у файлі *event_class.dart*, функції *createOrUpdateEvent*, *getAllEvents* та *deleteEvent* у файлі *database_service*, та файл *new_event_dialog* для діалогового вікна додавання нової події.

На календарі ми хочемо відображати такі елементи як “освітній елемент”, та “події”. Якщо освітні елементи зазвичай означають, що у нього є якась одна дата (дата екзамену, закриття заліку, захисту курсової роботи, тощо), то у подій може бути як дата початку, так і дата кінця (канікули, сесія, атестація, тощо).

У діалоговому меню додавання події є такі поля як назва, тип події (за замовчуванням – “Інше”, поле для введення типу події, дата початку та кінця. Ці вікна продемонстровані на рисунку 4.7.

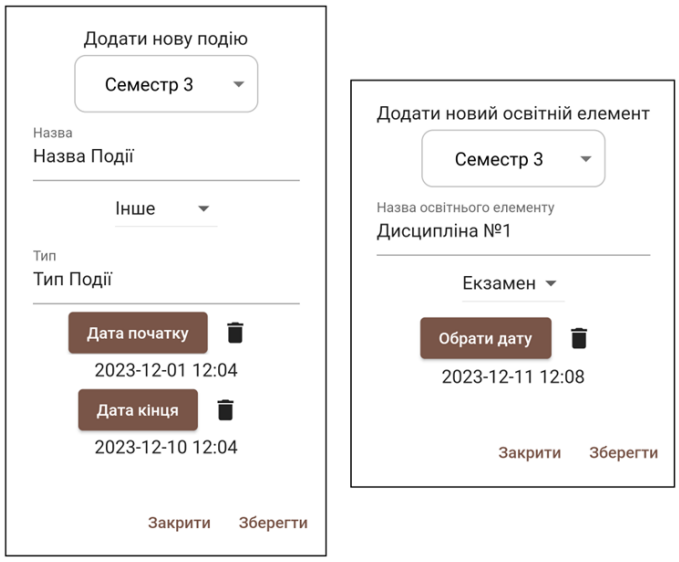

Рис. 4.7. Діалогові вікна “Додати нову подію” та “Додати новий освітній елемент” на сторінці графіку освітнього процесу

У календарі елементи відображаються кольором в залежності від типу елементів. Це реалізовано у наступному фрагменті коду:

```
Color getHighlightColor(DateTime date) {
 for (var course in widget.courses) {
  if (isSameDay(course.recordBookSelectedDateField!, date) &&
      course.isEvent == true) {
    switch (course.scoringTypeField) {
      case 'Екзамен':
        return Colors.red;
      case 'Залік':
        return Colors.yellow[800]!;
      default:
        return Colors.green;
    }
  }
 }
 for (var event in widget.events) {
  if (isSameDay(event.eventDateStart!, date) ||
      isSameDay(event.eventDateEnd!, date)) {
    switch (event.eventType) {
      case 'Екзамен':
        return Colors.red;
      case 'Залік':
        return Colors.yellow[800]!;
      default:
        return Colors.green;
    }
  }
 }
 return Colors.transparent;
}
```

Цей метод *getHighlightColor* перебирає список *courses* або *events*. Для кожної дати він перевіряє, чи дата відповідає даті освітнього елементу або події. Якщо знаходиться відповідність і курс позначений як подія (*isEvent* == true), або якщо дата відповідає початковій або кінцевій даті події, метод повертає колір залежно від типу елементу або події ('Екзамен', 'Залік' або інший), з червоним для 'Екзамену', темно-жовтим для 'Заліку' та зеленим для інших типів. Якщо для дати не знайдено відповідних курсів або подій, метод повертає прозорий колір, що вказує на відсутність особливого підсвічування цього дня.

Якщо ми додаємо подію з початковою та кінцевою датою, потрібно позначити дати між цією подією. Розглянемо відповідну реалізацію в наступному фрагменті коду:

```
List<EventSchedule> getEventsForDate(DateTime date) {
  List<EventSchedule> eventsForDate = [];
  for (var event in widget.events) {
    if (event.eventDateStart!.isBefore(date) &&
        event.eventDateEnd!.isAfter(date.add(const Duration(days: 1)))) {
      eventsForDate.add(event);
    }
  }
  return eventsForDate;
}


Widget buildDotsInBetween(BuildContext context, DateTime date, List events,
    Function getEventsForDate) {
  List<EventSchedule> eventsForDate = getEventsForDate(date);
  Set<String> eventTypes =
      eventsForDate.map((e) => e.eventType).whereType<String>().toSet();
  List<Color> dotColors = [];
  if (eventTypes.contains('Екзамен')) {
    dotColors.add(Colors.red);
  }
  if (eventTypes.contains('Залік')) {
    dotColors.add(Colors.yellow[800]!);
  }
  if (eventTypes.length > dotColors.length) {
    dotColors.add(Colors.green);
  }
  return Positioned(
    bottom: 1,
    child: Row(
      children: dotColors
          .map((color) => Container(
                margin: const EdgeInsets.symmetric(horizontal: 1),
                width: 5,
                height: 5,
                decoration: BoxDecoration(
```

```
                shape: BoxShape.circle,
                color: color,
              ),
            ))
        .toList(),
    ),
  );
}
```

Функція getEventsForDate приймає подію і перевіряє чи подія охоплює дану дату (тобто дата початку події є раніше, а дата закінчення – після зазначеної дати). Події, що відповідають цим критеріям, додаються до списку *eventsForDate*, який потім повертається і використовується у функції *buildDotsInBetween*.

Функція *buildDotsInBetween* відображає кольорові крапки під датами календаря, що знаходяться між початковою та кінцевою датами. Ці точки створюються як маленькі кольорові кружечки (Container з BoxShape.circle) і додаються до віджету Row. Потім використовується віджет Positioned для розміщення цього ряду точок у нижній частині комірки дати в календарі. Максимум показується по одній крапочці на кожен один тип елементу (тобто максимум – 3).

Окрім календаря, на сторінці графіку освітнього процесу у вигляді списку відображаються елементи та події, на кожній відображаються назва, тип, і дати. Функції у списку аналогічні до минулих сервісів – сортування, редагування, видалення.

При натисканні на дату на календарі, програма повинна показати список елементів у такому ж вигляді, як і на головній сторінці графіку, з різницею у тому, що відображаються лише ті елементи, які відбуваються в цю конкретну дату, та окремо ті, що тривають та включають цю дату. Для цього створено новий файл *calendar_dialog.dart*, у якому створено діалогове вікно, де відбувається реалізація цього функціоналу.

Таким чином створений сервіс “Графік освітнього процесу” для мобільного додатку. Приклад роботи цього сервісу подано на рисунку 4.8. З рисунку можна побачити, що на календарі виділені такі дати: темно-синім – обрана дата користувачем, або остання, на яку він натиснув; тускло-синя – це поточна дата, виділені червоним, жовтим та зеленим дати – це відмічені відповідно до їх типу дати початку та кінця; крапочками під датами – дати, які охоплюються певною подією (знаходяться між її початком та кінцем).

4.6. Опис користування мобільним додатком

Для коректної роботи додатку, девайс повинен відповідати таким вимогам:

- Android 5.0 (Lollipop) або вище. Це обмеження пов’язане з підтримкою бібліотекою Firebase тільки версій Android SDK вище 21, що відповідає версії Android 5.0;
- Щонайменше 1 GB оперативної пам’яті;
- Близько 30 MB вільного місця у пам’яті системи;
- Доступ до мережі Інтернет.

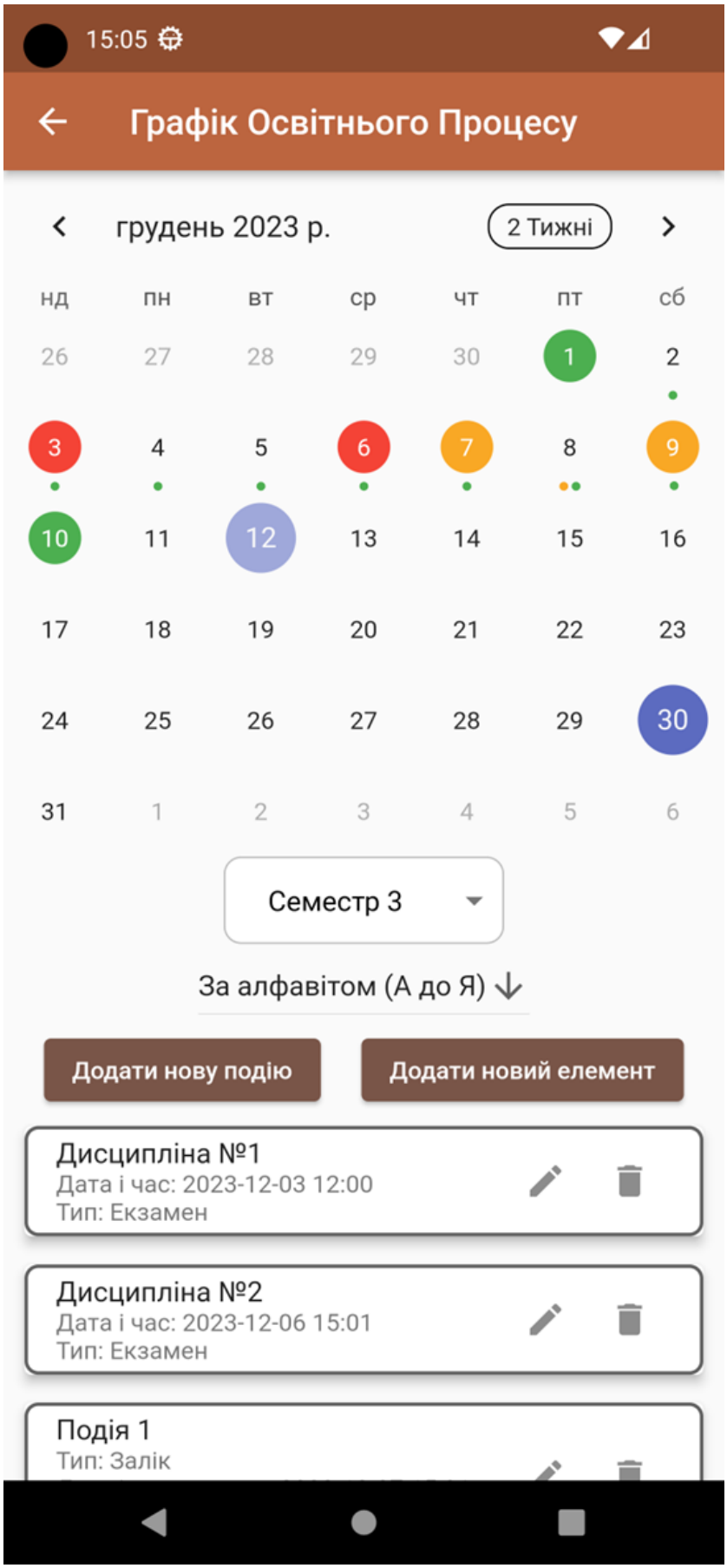

Рис. 4.8. Приклад роботи сервісу "Графік освітнього процесу"

Після завантаження та запуску додатку, користувач потрапляє на екран авторизації. Це перший екран додатку. Якщо у користувача немає облікового запису, йому потрібно натиснути на кнопку “Створити новий аккаунт”, після чого користувач потрапляє уже на сторінку реєстрації. Користувачу необхідно ввести коректну адресу електронної пошти у якості логіну користувача та придумати надійний пароль. Пароль вводиться у поле вводу паролю та підтвердження паролю, і якщо адреса електронної пошти правильна та паролі у полях збігаються, користувач потрапляє у головне меню додатку, зображене на рисунку 4.9.

Рис. 4.9. Головне меню додатку

У головному меню додатку знаходиться випадне меню вибору поточного навчального семестру. Якщо здобувач вищої освіти обере тут значення, цей обраний семестр буде використовуватись як семестр за замовчуванням у сервісах додатку.

Здобувач вищої освіти має можливість обрати один із трьох сервісів програмного комплексу: електронний індивідуальний навчальний план, електронну залікову книжку студента та електронний графік освітнього процесу. Для переходу до сервісу, користувач може натиснути на відповідний пункт головного меню додатку. У будь-який момент користувач може повернутись назад до головного меню, натиснувши на кнопку “Назад” на верхній панелі додатку, що доступна на всіх сторінках сервісів.

Додаток є готовим для використання здобувачем вищої освіти та виконує ті функції, які були визначені на етапі проєктування, а саме: зручний перегляд інформації з документів навчально-організаційного характеру, з можливостями сортування та групування, внесення нової інформації користувачем, редагування інформаційних елементів, їх видалення. На кожній сторінці сервісів можна обрати поточний семестр, інформацію для перегляду та виведення на екран.

Отже, у розділі 4 детально розглянуто усі етапи практичної реалізації коду мобільного додатку. Ретельно проаналізовано використані технології, інструменти розробки та особливості реалізації окремих компонентів додатку. Покроково описано, як і які програмні елементи, пакети та функції використовувалися для створення функціоналу кожного окремного сервісу. Зрозуміло описано програмну реалізацію мобільного застосунку та його використання на власному девайсі на платформі Android.

ВИСНОВКИ

У результаті дипломної роботи магістра створено повноцінний мобільний додаток – програмний комплекс “Студент”, що включає сервіси електронного індивідуального навчального плану здобувача вищої освіти, електронної залікової книжки та електронного графіка освітнього процесу. Створений додаток є готовим для використання здобувачами вищої освіти у закладах вищої освіти.

Проєктування та створення програмного додатку складалося з таких етапів:

- розглянуто актуальність та практичну користь мобільного додатку як інструменту для організації освітнього процесу здобувача вищої освіти;
- проаналізовано документи навчально-організаційного характеру, такі як індивідуальний навчальний план, залікова книжка та графік освітнього процесу як сервісів, що можуть бути використані у електронному вигляді шляхом трансформації їх у функції мобільного додатку;
- досліджено ринок мобільних операційних систем в Україні, розглянуто iOS та Android як платформи для розробки мобільних застосунків;
- здійснено аргументований вибір платформи Android для створення мобільного додатку;
- здійснено порівняння способів та інструментів розробки мобільних додатків для мобільних пристроїв;
- обрано фреймворк Flutter як набір інструментів для створення мобільного застосунку;
- проаналізовані способи збереження користувацької інформації у мобільному додатку;
- розглянутий сервіс Firebase Cloud Firestore як хмарна база даних для зберігання інформації для мобільного застосунку на Flutter.

У рамках виконання практичного завдання, пройдено такі етапи створення програмного додатку як: постановка задачі, планування функціоналу, проєктування програмної реалізації додатку та поступова покрокова розробка програмного коду.

Отже, результатом виконання завдання “Розробка програмного комплексу «Студент» для системи Android” є готовий програмний продукт – мобільний додаток “Студент”. Цей застосунок є вільним для розповсюдження та повністю готовим для використання здобувачами вищої освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Смартфони є у 55% українців, серед молоді – 92% (інфографіка). *УНІАН.* URL: https://www.unian.ua/economics/telecom/10500204-smartfoni-ye-u-55-ukrajinciv-sered-molodi-92-infografika.html.
2. Mobile Operating System Market Share Ukraine. *StatCounter*. URL: https://gs.statcounter.com/os-market-share/mobile/ukraine.
3. Toni Vujevic. Android vs. iOS app development: what's the difference? *DECODE*. URL: https://decode.agency/article/android-vs-ios-app-development/.
4. Subhasish Dutta. Android vs iOS App Development: What's the Major Difference? *Turing*. URL: https://www.turing.com/resources/mobile-app-development.
5. iOS App Development: How To Make Your First App. *LoginRadius*. URL: https://www.loginradius.com/blog/engineering/getting-started-with-ios-app-development/.
6. Abhinav Girdhar. How much does it cost to publish an app on the app store? *Appypie*. URL: https://www.appypie.com/faqs/how-much-does-it-cost-to-publish-an-app-on-the-app-store
7. Кількість користувачів смартфонів в Україні збільшилася до 85% — дослідження. *MediaSapiens*. URL: https://ms.detector.media/mediadoslidzhennya/post/21573/2018-08-03-kilkist-korystuvachiv-smartfoniv-v-ukraini-zbilshylasya-do-85-doslidzhennya/
8. Neeraj Sharma. Native, Web, Hybrid, or Cross-Platform – How to Choose the Right Mobile App for Your Business? *Kellon*. URL: https://www.kellton.com/kellton-tech-blog/how-to-choose-the-right-mobile-app-for-your-business
9. Yuliia Fedyk. Kotlin vs Java: Which One is Better for Android App Development. *Inveritasoft*. URL: https://inveritasoft.com/article-kotlin-vs-java:-which-one-is-better-for-android-app-development

10. Robert Johns. 5 Best iOS Programming Languages for App Development. *Hackr*. URL: https://hackr.io/blog/ios-programming-languages

11. Ante Baus. Pros and cons of native app development. *Decode Agency*. URL: https://decode.agency/article/native-app-development-pros-cons/

12. Progressive Web App Development. *Vilmate*. URL: https://vilmate.com/technologies/progressive-web-apps/

13. What is cross-platform mobile development? *Kotlin*. URL: https://kotlinlang.org/docs/cross-platform-mobile-development.htm

14. Yurii Luchaninov. Comparison of the Best Cross-Platform Mobile App Development Frameworks *Mobidev*. URL: https://mobidev.biz/blog/cross-platform-mobile-development-frameworks-comparison

15. Flutter Developer. Build more with Flutter. *Flutter.dev*. URL: https://flutter.dev/development

16. Mark Gamble. The Role of the Database in Mobile App Development. *The New Stack*. URL: https://thenewstack.io/the-role-of-the-database-in-mobile-app-development

17. Overview. *Firestore Documentation*. URL: https://cloud.google.com/firestore/docs

18. Чернявський А.А. Розробка програмного комплексу “Студент” для системи Android. Збірник матеріалів наукової конференції здобувачів вищої освіти фізико-математичного факультету Кам’янець-Подільського національного університету імені Івана Огієнка. 1 листопада 2023 року [Електронний ресурс]. Кам’янець-Подільський : Кам’янецьПодільський національний університет імені Івана Огієнка, 2023. С. 41-42.

19. Чернявський А.А. Програмний комплекс “Студент” для системи Android. Вісник Кам’янець-Подільського національного університету імені Івана Огієнка. Фізико-математичні науки. Випуск 16. Кам’янець-Подільський : Кам’янець-Подільський національний університет імені Івана Огієнка, 2023. С. 88-92.

ДОДАТОК

Програмний код мобільного додатку “Студент”

Вміст файлу main.dart

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      localizationsDelegates: const [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      supportedLocales: const [
        Locale('uk', ''),
      ],
      home: AuthPage(),
      theme: ThemeData(
        appBarTheme: AppBarTheme(
          color: Color(0xFFbc653f),
        ),
        elevatedButtonTheme: ElevatedButtonThemeData(
          style: ButtonStyle(
            backgroundColor: MaterialStateProperty.all(Colors.brown),
          ),
        ),
      ),
    );
  } }
```

Вміст файлу auth_page.dart

```
class AuthPage extends StatelessWidget {
  const AuthPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: StreamBuilder<User?>(
        stream: FirebaseAuth.instance.authStateChanges(),
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            return HomePage();
          }
          return const LoginOrRegisterPage();
        },
      ),
    );
  }
}
```

Вміст файлу login_page.dart

```
class LoginPage extends StatefulWidget {
  const LoginPage({super.key, required this.onTap});

  final Function()? onTap;

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final passwordController = TextEditingController();

  final emailController = TextEditingController();

  void signUserIn(BuildContext context) async {
    final scaffoldMessenger = ScaffoldMessenger.of(context);
    try {
```

```
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: emailController.text,
      password: passwordController.text,
    );
    checkDoesStudentDocumentExists();
  } on FirebaseAuthException catch (e) {
    String errorMessage;
    switch (e.code) {
      case 'user-not-found':
        errorMessage = 'Помилка! Не знайдено користувача з таким e-mail.';
        break;
      case 'wrong-password':
        errorMessage = 'Помилка! Неправельний логін або пароль!';
        break;
      default:
        errorMessage = 'Невідома помилка!';
    }
    scaffoldMessenger.showSnackBar(
      SnackBar(
        content: Text(errorMessage),
      ),
    );
  } catch (e) {
    scaffoldMessenger.showSnackBar(
      const SnackBar(
        content: Text('Невідома помилка!'),
      ),
    );
  }
}

void checkDoesStudentDocumentExists() async {
  var user = emailController.text;
  if (await DatabaseService.checkStudentDocument(user)) {
  } else {
    await DatabaseService.createStudentDocument(user);
  }
}

@override
Widget build(BuildContext context) {
```

```
    return Scaffold(
      backgroundColor: Colors.grey[300],
      body: SafeArea(
       child: Center(
        child: SingleChildScrollView(
         child: Column(
          children: [
           const Text("Вхід",
              style:
                 TextStyle(fontSize: 24, fontWeight: FontWeight.bold)),
           const SizedBox(height: 20),
           MyTextField(
            controller: emailController,
            hintText: "E-mail",
            obscureText: false,
           ),
           const SizedBox(height: 50),
           MyTextField(
              controller: passwordController,
              hintText: "Пароль",
              obscureText: true),
           const SizedBox(height: 25),
           MyButton(onTap: () => signUserIn(context), text: 'Увійти'),
           const SizedBox(height: 25),
           GestureDetector(
            onTap: widget.onTap,
            child: Text(
             'Створити новий аккаунт',
             style: TextStyle(color: Colors.blue[800], fontSize: 16),
            ),
           ),
          ],
         ),
        ),
       ),
      ));
  }
}
```

Вміст файлу register_page.dart

```
class RegisterPage extends StatefulWidget {
  const RegisterPage({super.key, required this.onTap});

  final Function()? onTap;

  @override
  State<RegisterPage> createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  final emailController = TextEditingController();
  final passwordController = TextEditingController();
  final confirmPasswordController = TextEditingController();

  void signUserUp(BuildContext context) async {
    final scaffoldMessenger = ScaffoldMessenger.of(context);
    try {
      if (passwordController.text == confirmPasswordController.text) {
        await FirebaseAuth.instance.createUserWithEmailAndPassword(
          email: emailController.text,
          password: passwordController.text,
        );
        await DatabaseService.createStudentDocument(emailController.text);
      } else {
        scaffoldMessenger.showSnackBar(
          const SnackBar(
            content: Text("Паролі не збігаються!"),
          ),
        );
      }
    } on FirebaseAuthException catch (e) {
      String errorMessage;
      switch (e.code) {
        case 'email-already-in-use':
          errorMessage = 'Помилка! Користувач вже існує!';
          break;
        case 'invalid-email':
          errorMessage = 'Помилка! Неправельний e-mail!';
          break;
```

```
      default:
        errorMessage = 'Невідома помилка!';
    }
    scaffoldMessenger.showSnackBar(
      SnackBar(
        content: Text(errorMessage),
      ),
    );
  } catch (e) {
    scaffoldMessenger.showSnackBar(
      const SnackBar(
        content: Text("Невідома помилка!"),
      ),
    );
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      backgroundColor: Colors.grey[300],
      body: SafeArea(
        child: Center(
          child: SingleChildScrollView(
            child: Column(
              children: [
                const Text("Реєстрація",
                    style:
                        TextStyle(fontSize: 24, fontWeight: FontWeight.bold)),
                const SizedBox(height: 20),
                MyTextField(
                  controller: emailController,
                  hintText: "E-mail",
                  obscureText: false,
                ),
                const SizedBox(height: 25),
                MyTextField(
                    controller: passwordController,
                    hintText: "Пароль",
                    obscureText: true),
                const SizedBox(height: 25),
```

```
            MyTextField(
                controller: confirmPasswordController,
                hintText: "Підтвердіть пароль",
                obscureText: true),
            const SizedBox(height: 25),
            MyButton(
              onTap: () => signUserUp(context),
              text: 'Зареєструватись',
            ),
            const SizedBox(height: 25),
            GestureDetector(
              onTap: widget.onTap,
              child: Text(
                'Увійти',
                style: TextStyle(color: Colors.blue[800], fontSize: 16),
        ), ), ], ), ), ), ) ); } }
```

Вміст файлу login_or_register_page.dart

```
class LoginPage extends StatefulWidget {
  const LoginPage({super.key, required this.onTap});

  final Function()? onTap;

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final passwordController = TextEditingController();

  final emailController = TextEditingController();

  void signUserIn(BuildContext context) async {
    final scaffoldMessenger = ScaffoldMessenger.of(context);
    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: emailController.text,
        password: passwordController.text,
      );
      checkDoesStudentDocumentExists();
```

```
  } on FirebaseAuthException catch (e) {
    String errorMessage;
    switch (e.code) {
      case 'user-not-found':
        errorMessage = 'Помилка! Не знайдено користувача з таким e-mail.';
        break;
      case 'wrong-password':
        errorMessage = 'Помилка! Неправельний логін або пароль!';
        break;
      default:
        errorMessage = 'Невідома помилка!';
    }
    scaffoldMessenger.showSnackBar(
      SnackBar(
        content: Text(errorMessage),
      ),
    );
  } catch (e) {
    scaffoldMessenger.showSnackBar(
      const SnackBar(
        content: Text('Невідома помилка!'),
      ),
    );
  }
}

void checkDoesStudentDocumentExists() async {
  var user = emailController.text;
  if (await DatabaseService.checkStudentDocument(user)) {
  } else {
    await DatabaseService.createStudentDocument(user);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      backgroundColor: Colors.grey[300],
      body: SafeArea(
        child: Center(
          child: SingleChildScrollView(
```

```
            child: Column(
              children: [
                const Text("Вхід",
                    style:
                        TextStyle(fontSize: 24, fontWeight: FontWeight.bold)),
                const SizedBox(height: 20),
                MyTextField(
                  controller: emailController,
                  hintText: "E-mail",
                  obscureText: false,
                ),
                const SizedBox(height: 50),
                MyTextField(
                    controller: passwordController,
                    hintText: "Пароль",
                    obscureText: true),
                const SizedBox(height: 25),
                MyButton(onTap: () => signUserIn(context), text: 'Увійти'),
                const SizedBox(height: 25),
                GestureDetector(
                  onTap: widget.onTap,
                  child: Text(
                    'Створити новий аккаунт',
                    style: TextStyle(color: Colors.blue[800], fontSize: 16),
          ) ), ], ), ), ), ) ); } }
```

Вміст файлу my_button.dart

```
import 'package:flutter/material.dart';

class MyButton extends StatelessWidget {
  final Function()? onTap;
  final String text;

  const MyButton({super.key, required this.onTap, required this.text});

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Container(
```

```
      padding: const EdgeInsets.all(20),
      margin: const EdgeInsets.symmetric(horizontal: 20),
      decoration: const BoxDecoration(color: Colors.brown),
      child: Center(
        child: Text(
          text,
          style: const TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.w600,
            fontSize: 16,
          ), ), ), ), ); } }
```

Вміст файлу my_textfield.dart

```
import 'package:flutter/material.dart';

class MyTextField extends StatelessWidget {
  final TextEditingController? controller;
  final String? hintText;
  final bool obscureText;

  const MyTextField(
      {super.key, this.controller, this.hintText, this.obscureText = false});

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 20),
      child: TextField(
        controller: controller,
        obscureText: obscureText,
        decoration: InputDecoration(
          enabledBorder: const OutlineInputBorder(
            borderSide: BorderSide(color: Colors.grey),
          ),
          focusedBorder: const OutlineInputBorder(
            borderSide: BorderSide(color: Colors.black),
          ),
          fillColor: Colors.white,
          hintText: hintText,
        ), ), ); } }
```

Вміст файлу my_datepicker.dart

```
import 'package:flutter/material.dart';

Future<DateTime> selectDate(BuildContext context) async {
  final DateTime? pickedDate = await showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime(2000),
    lastDate: DateTime(2100),
  );

  if (pickedDate != null) {
    final TimeOfDay? pickedTime = await showTimePicker(
      context: context,
      initialTime: TimeOfDay.now(),
    );

    if (pickedTime != null) {
      return DateTime(
        pickedDate.year,
        pickedDate.month,
        pickedDate.day,
        pickedTime.hour,
        pickedTime.minute,
      );
    }
  }

  return DateTime.now();
}
```

Вміст файлу dropdownmenu_design.dart

```
import 'package:flutter/material.dart';

class DropdownMenuDesign extends StatelessWidget {
  final List<String> items;
  final String? selectedItem;
```

```
final ValueChanged<String?>? onChanged;

const DropdownMenuDesign({
  Key? key,
  required this.items,
  this.selectedItem,
  this.onChanged,
}) : super(key: key);

@override
Widget build(BuildContext context) {
  return Container(
    width: 160,
    padding: const EdgeInsets.symmetric(),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(10.0),
      border: Border.all(color: Colors.grey[400]!),
    ),
    child: DropdownButtonHideUnderline(
      child: ButtonTheme(
        alignedDropdown: true,
        child: DropdownButton<String>(
          dropdownColor: Colors.white,
          isExpanded: true,
          items: items.map<DropdownMenuItem<String>>((String item) {
            return DropdownMenuItem<String>(
              value: item,
              child: SizedBox(
                height: 18,
                child: Align(
                  alignment: Alignment.centerLeft,
                  child: Text(
                    item,
                    style: const TextStyle(
                        fontSize: 16,
                        fontFamily: 'Sans-serif',
                        color: Colors.black),
                  ), ), ),  );
          }).toList(),
          onChanged: onChanged,
```

```
          value: selectedItem,
          icon: Icon(Icons.arrow_drop_down, color: Colors.grey[600]),
          padding: const EdgeInsets.symmetric(),
        ), ), ), ); } }
```

Вміст файлу dropdownmenu_user_semester.dart

```
class DropdownMenuUserSemester extends StatefulWidget {
  final Future<List<String>> listOfData;
  late Future<String> chosenValueInDatabase;
  final String? chosenField;

  DropdownMenuUserSemester(
      {super.key,
      required this.listOfData,
      required this.chosenValueInDatabase,
      required this.chosenField});

  @override
  State<DropdownMenuUserSemester> createState() =>
      _DropdownMenuUserSemesterState();
}

class _DropdownMenuUserSemesterState extends State<DropdownMenuUserSemester> {
  final user = FirebaseAuth.instance.currentUser!;

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: widget.listOfData,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const CircularProgressIndicator();
        } else if (snapshot.hasError) {
          return Text('Помилка: ${snapshot.error}');
        } else if (snapshot.hasData) {
          List<String>? dataList = snapshot.data;
          return FutureBuilder<String?>(
            future: widget.chosenValueInDatabase,
            builder: (BuildContext context, AsyncSnapshot<String?> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
```

```
            return const CircularProgressIndicator();
          } else if (snapshot.hasError) {
            return Text('Помилка: ${snapshot.error}');
          } else {
            return DropdownMenuDesign(
              items: dataList!,
              selectedItem: snapshot.data == "" ? null : snapshot.data,
              onChanged: (selectedItem) {
                DatabaseService.setStudentFields(
                    user.email, selectedItem!, widget.chosenField!);
                setState(() {
                  widget.chosenValueInDatabase = Future.value(selectedItem);
                } ); },); } }, ); }
      return const CircularProgressIndicator(); }, ); } }
```

Вміст файлу dropdownmenu_choose_semester.dart

```
class DropdownMenuChooseSemester extends StatefulWidget {
  final Function(String) onSelectedItemChanged;
  final String? initialSemester;

  const DropdownMenuChooseSemester({
    Key? key,
    required this.onSelectedItemChanged,
    this.initialSemester,
  }) : super(key: key);

  @override
  State<DropdownMenuChooseSemester> createState() =>
      _DropdownMenuChooseSemesterState();
}

class _DropdownMenuChooseSemesterState
    extends State<DropdownMenuChooseSemester> {
  final user = FirebaseAuth.instance.currentUser!;
  late Future<List<String>> semesterList =
      DatabaseService.getSemesterList(user.email);
  late Future<String> currentSemester = widget.initialSemester != null
      ? Future.value(widget.initialSemester)
      : DatabaseService.getStudentField(user.email, 'Current Semester');
```

```
@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: semesterList,
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      } else if (snapshot.hasError) {
        return Center(child: Text('Помилка: ${snapshot.error}'));
      } else if (snapshot.hasData) {
        List<String>? dataList = snapshot.data;
        return FutureBuilder<String?>(
          future: currentSemester,
          builder: (BuildContext context, AsyncSnapshot<String?> snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(child: CircularProgressIndicator());
            } else if (snapshot.hasError) {
              return Center(child: Text('Помилка: ${snapshot.error}'));
            } else {
              return DropdownMenuDesign(
                items: dataList!,
                selectedItem: snapshot.data,
                onChanged: (selectedItem) {
                  setState(() {
                    currentSemester = Future.value(selectedItem);
                  });
                  widget.onSelectedItemChanged(selectedItem!);
                }, ); }  }, ); }
      return const Center(child: CircularProgressIndicator()); }, ); } }
```

Вміст файлу course_class.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';

class Course {
  String? nameField;
  num? hoursLectionsField;
  num? hoursPracticesField;
  num? hoursLabsField;
  num? hoursCourseworkField;
```

```
num? hoursInClassTotalField;
num? hoursIndividualTotalField;
num? hoursOverallTotalField;
num? creditsOverallTotalField;
String? scoringTypeField;
String? recordBookTeacherField;
num? recordBookScoreField;
DateTime? recordBookSelectedDateField;
bool? isScheduleFilled;
bool? isRecordBookFilled;
bool? isClassScheduleOnly;
bool? isEvent;

Course({
  this.nameField,
  this.hoursLectionsField,
  this.hoursPracticesField,
  this.hoursLabsField,
  this.hoursCourseworkField,
  this.hoursIndividualTotalField,
  this.hoursOverallTotalField,
  this.scoringTypeField,
  this.recordBookTeacherField,
  this.recordBookScoreField,
  this.recordBookSelectedDateField,
  this.isScheduleFilled,
  this.isRecordBookFilled,
  this.isClassScheduleOnly,
  this.isEvent,
}) {
  hoursLectionsField = hoursLectionsField ?? 0;
  hoursPracticesField = hoursPracticesField ?? 0;
  hoursLabsField = hoursLabsField ?? 0;
  hoursCourseworkField = hoursCourseworkField ?? 0;
  hoursIndividualTotalField = hoursIndividualTotalField ?? 0;
  hoursOverallTotalField = hoursOverallTotalField ?? 0;
  hoursInClassTotalField = calculateTotalHoursInClass();
  hoursOverallTotalField = calculateTotalHoursOverall();
  creditsOverallTotalField = calculateTotalCredits();
  scoringTypeField = scoringTypeField ?? 'Залік';
  recordBookTeacherField = recordBookTeacherField ?? '';
```

```
  recordBookScoreField = recordBookScoreField ?? 0;
  recordBookSelectedDateField = recordBookSelectedDateField ??
      DateTime.fromMillisecondsSinceEpoch(978307200000, isUtc: true);
  isScheduleFilled = isScheduleFilled ?? false;
  isRecordBookFilled = isRecordBookFilled ?? false;
  isClassScheduleOnly = isClassScheduleOnly ?? false;
  isEvent = isEvent ?? false;
}

Map<String, dynamic> toJsonCourse() {
  return {
    'Name': nameField,
    'Hours Lections': hoursLectionsField,
    'Hours Practices': hoursPracticesField,
    'Hours Labs': hoursLabsField,
    'Hours Coursework': hoursCourseworkField,
    'Hours In Class Total': hoursInClassTotalField,
    'Hours Individual Total': hoursIndividualTotalField,
    'Hours Overall Total': hoursOverallTotalField,
    'Credits Overall Total': creditsOverallTotalField,
    'Scoring Type': scoringTypeField,
    '(Record Book) Teacher': recordBookTeacherField,
    '(Record Book) Score': recordBookScoreField,
    '(Record Book) Date': recordBookSelectedDateField,
    '(app) isScheduleFilled': isScheduleFilled,
    '(app) isRecordBookFilled': isRecordBookFilled,
    '(app) isClassScheduleOnly': isClassScheduleOnly,
    '(app) isEvent': isEvent,
  };
}

Course.fromJsonCourse(Map<String, dynamic> json) {
  nameField = json['Name'];
  hoursLectionsField = json['Hours Lections'];
  hoursPracticesField = json['Hours Practices'];
  hoursLabsField = json['Hours Labs'];
  hoursCourseworkField = json['Hours Coursework'];
  hoursInClassTotalField = json['Hours In Class Total'];
  hoursIndividualTotalField = json['Hours Individual Total'];
  hoursOverallTotalField = json['Hours Overall Total'];
  creditsOverallTotalField = json['Credits Overall Total'];
```

```
    scoringTypeField = json['Scoring Type'];
    recordBookTeacherField = json['(Record Book) Teacher'];
    recordBookScoreField = json['(Record Book) Score'];
    recordBookSelectedDateField = json['(Record Book) Date'] != null
        ? (json['(Record Book) Date'] as Timestamp).toDate()
        : null;
    isScheduleFilled = json['(app) isScheduleFilled'];
    isRecordBookFilled = json['(app) isRecordBookFilled'];
    isClassScheduleOnly = json['(app) isClassScheduleOnly'];
    isEvent = json['(app) isEvent'];
  }

  void recalculateTotals() {
    hoursInClassTotalField = calculateTotalHoursInClass();
    hoursOverallTotalField = calculateTotalHoursOverall();
    creditsOverallTotalField = calculateTotalCredits();
  }

  num calculateTotalHoursInClass() {
    return (hoursLectionsField ?? 0) +
        (hoursPracticesField ?? 0) +
        (hoursLabsField ?? 0) +
        (hoursCourseworkField ?? 0);
  }

  num calculateTotalHoursOverall() {
    return (hoursInClassTotalField ?? 0) + (hoursIndividualTotalField ?? 0);
  }

  num calculateTotalCredits() {
    if (hoursOverallTotalField == null) {
      return 0;
    } else {
      return num.parse((hoursOverallTotalField! / 30).toStringAsFixed(2));
    }
  }
}
```

Вміст файлу event_class.dart

```
import 'package:cloud_firestore/cloud_firestore.dart';
```

```
class EventSchedule {
  String? eventName;
  String? eventType;
  DateTime? eventDateStart;
  DateTime? eventDateEnd;

  EventSchedule({
    this.eventName,
    this.eventType,
    this.eventDateStart,
    this.eventDateEnd,
  }) {
    eventName = eventName ?? '';
    eventType = eventType ?? '';
    eventDateStart = eventDateStart ??
        DateTime.fromMillisecondsSinceEpoch(978307200000, isUtc: true);
    eventDateEnd = eventDateEnd ??
        DateTime.fromMillisecondsSinceEpoch(978307200000, isUtc: true);
  }

  Map<String, dynamic> toJsonEvent() {
    Map<String, dynamic> json = {
      'Event Name': eventName,
      'Event Type': eventType,
    };

    if (eventDateStart != null) {
      json['Event Date'] = eventDateStart;
    }

    if (eventDateEnd != null) {
      json['Event Date End'] = eventDateEnd;
    }

    return json;
  }

  EventSchedule.fromJsonEvent(Map<String, dynamic> json) {
    eventName = json['Event Name'];
    eventType = json['Event Type'];
```

```
    eventDateStart = json['Event Date'] != null
        ? (json['Event Date'] as Timestamp).toDate()
        : null;
    eventDateEnd = json['Event Date End'] != null
        ? (json['Event Date End'] as Timestamp).toDate()
        : null;
  }
}
```

Вміст файлу database_service.dart

```
class DatabaseService {
  static Future<void> createStudentDocument(var user) {
    final docRef = FirebaseFirestore.instance.collection("student").doc(user);
    final Map<String, dynamic> studentData = {
      'E-mail': user,
      'Current Semester': "Семестр 1"
    };
    return docRef.set(studentData).then((_) {
      return createSemesterCollection(user);
    });
  }

  static Future<void> createSemesterCollection(var user) {
    List<Future> tasks = [];
    for (int i = 1; i <= 8; i++) {
      tasks.add(FirebaseFirestore.instance
          .collection("student")
          .doc(user)
          .collection('semester')
          .doc('Семестр $i')
          .set({}));
    }
    return Future.wait(tasks);
  }

  static Future<void> setStudentFields(
      var user, String value, String field) async {
    final docRef = FirebaseFirestore.instance.collection('student').doc(user);
    DocumentSnapshot snapshot = await docRef.get();
    Map<String, dynamic> data = snapshot.data() as Map<String, dynamic>;
```

```
  data[field] = value;
  await docRef.set(data);
}

static Future<String> getStudentField(var user, String field) async {
  DocumentSnapshot snapshot =
      await FirebaseFirestore.instance.collection('student').doc(user).get();
  if (snapshot.exists) {
    Map<String, dynamic> data = snapshot.data() as Map<String, dynamic>;
    return data[field];
  } else {
    return '';
  }
}

static Future<bool> checkStudentDocument(var user) async {
  DocumentSnapshot snapshot =
      await FirebaseFirestore.instance.collection("student").doc(user).get();
  return snapshot.exists;
}

static Future<List<String>> getSemesterList(var user) async {
  QuerySnapshot snapshot = await FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection('semester')
      .get();
  List<String> semesterList = snapshot.docs.map((doc) => doc.id).toList();
  return semesterList;
}

static Future<List<Course>> getAllCourses(
    String userEmail, String semester) async {
  QuerySnapshot querySnapshot = await FirebaseFirestore.instance
      .collection("student")
      .doc(userEmail)
      .collection("semester")
      .doc(semester)
      .collection("Courses")
      .get();
  if (querySnapshot.docs.isNotEmpty) {
```

```
    return querySnapshot.docs
        .map((doc) =>
            Course.fromJsonCourse(doc.data() as Map<String, dynamic>))
        .toList();
  } else {
    return [];
  }
}

static Future<void> createOrUpdateCourse(
    var user, Course course, String semester) async {
  if (course.nameField!.trim().isEmpty) {
    throw Exception('Назва курсу не може бути порожньою!');
  }
  final docRef = FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection("semester")
      .doc(semester)
      .collection("Courses")
      .doc(course.nameField);
  return docRef.set(course.toJsonCourse());
}

static Future<void> deleteCourse(var user, String courseName) async {
  try {
    var currentSemester = await getStudentField(user, 'Current Semester');
    await FirebaseFirestore.instance
        .collection("student")
        .doc(user)
        .collection("semester")
        .doc(currentSemester)
        .collection("Courses")
        .doc(courseName)
        .delete();
  } catch (e) {
    rethrow;
  }
}

static Future<List<EventSchedule>> getAllEvents(
```

```
    String userEmail, String semester) async {
  QuerySnapshot querySnapshot = await FirebaseFirestore.instance
      .collection("student")
      .doc(userEmail)
      .collection("semester")
      .doc(semester)
      .collection("Events")
      .get();

  if (querySnapshot.docs.isNotEmpty) {
    return querySnapshot.docs
        .map((doc) =>
            EventSchedule.fromJsonEvent(doc.data() as Map<String, dynamic>))
        .toList();
  } else {
    return [];
  }
}

static Future<void> createOrUpdateEvent(
    var user, EventSchedule event, String semester) async {
  if (event.eventName!.trim().isEmpty) {
    throw Exception('Назва події не може бути порожньою!');
  }
  final docRef = FirebaseFirestore.instance
      .collection("student")
      .doc(user)
      .collection("semester")
      .doc(semester)
      .collection("Events")
      .doc(event.eventName);
  return docRef.set(event.toJsonEvent());
}

static Future<void> deleteEvent(var user, String eventName) async {
  try {
    var currentSemester = await getStudentField(user, 'Current Semester');
    await FirebaseFirestore.instance
        .collection("student")
        .doc(user)
        .collection("semester")
```

```
            .doc(currentSemester)
            .collection("Events")
            .doc(eventName)
            .delete();
      } catch (e) {
        rethrow;
      }
    }
  }
```

Вміст файлу courses_schedule_page.dart

```
class CoursesSchedulePage extends StatefulWidget {
  const CoursesSchedulePage({Key? key}) : super(key: key);

  @override
  State<CoursesSchedulePage> createState() => _CoursesSchedulePageState();
}

class _CoursesSchedulePageState extends State<CoursesSchedulePage> {
  final user = FirebaseAuth.instance.currentUser!;
  late Future<String> selectedSemesterOfUser =
      DatabaseService.getStudentField(user.email, 'Current Semester');
  late Future<List<String>> semesterList =
      DatabaseService.getSemesterList(user.email);

  SortOption sortOption = SortOption.alphabeticalAsc;
  String? selectedSemester;
  Future<List<Map<String, dynamic>>> coursesFuture = Future.value([]);
  List<bool> expandedState = [];

  @override
  void initState() {
    super.initState();
    selectedSemesterOfUser.then((value) {
      setState(() {
        selectedSemester = value;
        coursesFuture = generateCourses(selectedSemester!);
      });
    });
  }
```

```
void updateSelectedSemester(String selectedItem) {
  setState(() {
    selectedSemester = selectedItem;
    coursesFuture = generateCourses(selectedItem);
    coursesFuture.then((courses) {
      setState(() {
        expandedState = courses
            .map((course) => !(course['course'].isScheduleFilled ?? false))
            .toList();
      });
    });
  });
}

Widget _sortDropDownMenu() {
  return DropdownButton<SortOption>(
    value: sortOption,
    icon: const Icon(Icons.arrow_downward),
    onChanged: (SortOption? newValue) {
      setState(() {
        sortOption = newValue!;
        coursesFuture = generateCourses(selectedSemester!);
        coursesFuture.then((courses) {
          setState(() {
            expandedState = courses
                .map(
                    (course) => !(course['course'].isScheduleFilled ?? false))
                .toList();
          });
        });
      });
    },
    items: const <DropdownMenuItem<SortOption>>[
      DropdownMenuItem<SortOption>(
        value: SortOption.alphabeticalAsc,
        child: Text('За алфавітом (А до Я)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.alphabeticalDesc,
        child: Text('За алфавітом (Я до А)'),
```

```
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.hoursInClassDesc,
        child: Text('За аудиторними годинами'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.hoursIndividualDesc,
        child: Text('За індивідуальними годинами'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.hoursOverallDesc,
        child: Text('За годинами загалом'),
      ),
    ],
  );
}

Future<List<Course>> fetchCourses(String semester) async {
  return await DatabaseService.getAllCourses(user.email!, semester);
}

List<bool> setExpandedState(List<Course> courses) {
  return courses
      .map((course) => !(course.isScheduleFilled ?? false))
      .toList();
}

List<Course> sortCourses(List<Course> courses) {
  courses.sort((a, b) {
    if (a.isScheduleFilled == true && b.isScheduleFilled != true) {
      return -1;
    } else if (b.isScheduleFilled == true && a.isScheduleFilled != true) {
      return 1;
    } else {
      if (sortOption == SortOption.alphabeticalAsc) {
        return (a.nameField ?? "").compareTo(b.nameField ?? "");
      } else if (sortOption == SortOption.alphabeticalDesc) {
        return (b.nameField ?? "").compareTo(a.nameField ?? "");
      } else if (sortOption == SortOption.hoursInClassDesc) {
        return (b.hoursInClassTotalField?.toInt() ?? 0)
            .compareTo(a.hoursInClassTotalField?.toInt() ?? 0);
```

```
    } else if (sortOption == SortOption.hoursIndividualDesc) {
      return (b.hoursIndividualTotalField?.toInt() ?? 0)
          .compareTo(a.hoursIndividualTotalField?.toInt() ?? 0);
    } else if (sortOption == SortOption.hoursOverallDesc) {
      return (b.hoursOverallTotalField?.toInt() ?? 0)
          .compareTo(a.hoursOverallTotalField?.toInt() ?? 0);
    } else {
      return 0;
    }
   }
  });

  return courses;
}

Future<List<Map<String, dynamic>>> generateCourses(String semester) async {
  List<Course> courses = await fetchCourses(semester);
  List<Course> filledCourses = [];
  List<Course> unfilledCourses = [];

  for (var course in courses) {
    if (course.isScheduleFilled == true) {
      filledCourses.add(course);
    } else {
      unfilledCourses.add(course);
    }
  }

  filledCourses = sortCourses(filledCourses);
  unfilledCourses = sortCourses(unfilledCourses);

  courses = [...filledCourses, ...unfilledCourses];
  expandedState = setExpandedState(courses);

  return courses.map((course) {
    return {
      'course': course,
      'isExpanded': !(course.isScheduleFilled ?? false),
    };
  }).toList();
}
```

```
Widget _addNewCourseButton(BuildContext context) {
  return ElevatedButton(
    onPressed: () async {
      bool? result = await showDialog(
        context: context,
        builder: (BuildContext context) {
          return NewCourseDialog(
            currentSemester: selectedSemester,
          );
        },
      );
      if (result == true) {
        setState(() {
          coursesFuture = generateCourses(selectedSemester!);
        });
      }
    },
    child: const Text("Додати новий елемент"), ); }

void _showDeleteDialog(BuildContext context, Course course, String semester) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Видалити елемент?'),
        actions: <Widget>[
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Expanded(
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: ElevatedButton(
                    style: Theme.of(context).elevatedButtonTheme.style,
                    child: const Center(
                        child: Text('Видалити зі сторінки',
                            textAlign: TextAlign.center)),
                    onPressed: () async {
                      NavigatorState navigator = Navigator.of(context);
                      course.isScheduleFilled = false;
```

```
                await DatabaseService.createOrUpdateCourse(
                    user.email!, course, semester);
                setState(() {
                  coursesFuture = generateCourses(selectedSemester!);
                });
                navigator.pop();
              }, ), ),
          ),
          Expanded(
            child: Padding(
              padding: const EdgeInsets.all(8.0),
              child: ElevatedButton(
                style: Theme.of(context).elevatedButtonTheme.style,
                child: const Center(
                    child: Text('Повністю видалити елемент',
                        textAlign: TextAlign.center)),
                onPressed: () async {
                  Navigator.of(context).pop();
                  await DatabaseService.deleteCourse(
                      user.email!, course.nameField!);
                  setState(() {
                    coursesFuture = generateCourses(selectedSemester!);
                  }); }, ), ), ), ], ),
        TextButton(
          child: const Text(
            'Закрити',
            style: TextStyle(color: Colors.brown),
          ),
          onPressed: () {
            Navigator.of(context).pop();
        }, ), ], ); }, ); }

Widget _coursesListView(BuildContext context) {
  return FutureBuilder(
    future: coursesFuture,
    builder: (BuildContext context, AsyncSnapshot snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      } else if (snapshot.hasError) {
        return Text('Помилка: ${snapshot.error}');
      } else {
```

```
return ListView.builder(
  itemCount: snapshot.data.length,
  itemBuilder: (context, index) {
    Course course = snapshot.data[index]['course'];
    bool isNotFilledTitle = index > 0 &&
        snapshot.data[index - 1]['course'].isScheduleFilled == true &&
        course.isScheduleFilled == false;
    return Column(
      children: [
        if (isNotFilledTitle)
          const Padding(
            padding: EdgeInsets.all(8.0),
            child: Text(
              'Не заповнені',
              style: TextStyle(
                  fontSize: 20, fontWeight: FontWeight.bold),
            ),
          ),
        Card(
          child: Column(
            children: [
              ListTile(
                title: Row(
                  children: <Widget>[
                    IconButton(
                      icon: Icon(expandedState[index]
                          ? Icons.expand_less
                          : Icons.expand_more),
                      onPressed: () {
                        setState(() {
                          expandedState[index] =
                              !expandedState[index];
                        });
                      },
                    ),
                    Expanded(
                      child: Text(course.nameField ?? "",
                          style: const TextStyle(fontSize: 16)),
                    ),
                    IconButton(
                      icon: course.isScheduleFilled == true
```

```
                    ? const Icon(Icons.edit)
                    : Container(),
                onPressed: () async {
                  bool? result = await showDialog(
                    context: context,
                    builder: (BuildContext context) {
                      return NewCourseDialog(
                        isEdit: true,
                        isEditFilling: false,
                        course: course,
                        filledCourseSchedule: true,
                        currentSemester: selectedSemester,
                      );
                    },
                  );
                  if (result == true) {
                    setState(() {
                      coursesFuture =
                          generateCourses(selectedSemester!);
                    }); }  },  ),
              IconButton(
                icon: const Icon(Icons.delete),
                onPressed: () {
                  _showDeleteDialog(
                      context, course, selectedSemester!); }, ), ], ), ),
          if (expandedState[index])
            Padding(
              padding: const EdgeInsets.all(16.0),
              child: _courseDetails(course))
        ], ), ), ], ); }, ); } }, ); }

Widget _courseDetails(Course course) {
  if (!(course.isScheduleFilled ?? false)) {
    return _courseDetailsUnfilledCourse(course);
  } else {
    return _courseDetailsFilledCourse(course);
  }
}

Widget _courseDetailsFilledCourse(Course course) {
  Widget courseName = Row(
```

```
  children: [
    Expanded(
      child: Text(
        'Назва: ${course.nameField}',
        style: const TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 20,
        ),
        softWrap: true,
        overflow: TextOverflow.clip,
      ),
    ),
  ],
);
return Align(
  alignment: Alignment.centerLeft,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      courseName,
      const SizedBox(height: 10),
      Row(
        children: [
          Expanded(
            child: Text.rich(
              TextSpan(
                style: const TextStyle(fontSize: 20),
                children: <InlineSpan>[
                  const TextSpan(
                    text: 'Форма підсумкового контролю: ',
                  ),
                  TextSpan(
                    text: '${course.scoringTypeField}',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 22,
                      color: course.scoringTypeField == 'Екзамен'
                          ? Colors.red
                          : course.scoringTypeField == 'Залік'
                              ? Colors.orange
                              : Colors.green,
```

```
            ), ), ], ), ), ), ], ),
const SizedBox(height: 10),
const Center(
  child: Text(
    'Навчальне навантаження',
    style: TextStyle(fontSize: 25, fontWeight: FontWeight.bold),
  ),
),
const SizedBox(height: 10),
const Center(
    child: Text("Аудиторні години", style: TextStyle(fontSize: 20))),
const SizedBox(height: 10),
Table(
  border: TableBorder.all(color: Colors.black),
  children: [
    const TableRow(
      children: [
        Center(child: Text('Лекції', style: TextStyle(fontSize: 15))),
        Center(
            child: Text('Практичні / Семінарські',
                style: TextStyle(fontSize: 15))),
        Center(
            child:
                Text('Лабораторні', style: TextStyle(fontSize: 15))),
        Center(
            child: Text('Курсові', style: TextStyle(fontSize: 15))),
      ],
    ),
    TableRow(
      children: [
        Center(
            child: Text('${course.hoursLectionsField}',
                style: const TextStyle(
                    fontSize: 20, fontWeight: FontWeight.bold))),
        Center(
            child: Text('${course.hoursPracticesField}',
                style: const TextStyle(
                    fontSize: 20, fontWeight: FontWeight.bold))),
        Center(
            child: Text('${course.hoursLabsField}',
                style: const TextStyle(
```

```
                fontSize: 20, fontWeight: FontWeight.bold))),
        Center(
          child: Text('${course.hoursCourseworkField}',
            style: const TextStyle(
              fontSize: 20, fontWeight: FontWeight.bold))),
      ],
    ),
  ],
),
const SizedBox(height: 10),
Text.rich(
  TextSpan(
    style: const TextStyle(fontSize: 20),
    children: <InlineSpan>[
      const TextSpan(text: 'Усього аудиторних годин: '),
      TextSpan(
        text: '${course.hoursIndividualTotalField}',
        style: const TextStyle(fontWeight: FontWeight.bold)),
    ],
  ),
),
const SizedBox(height: 10),
Text.rich(
  TextSpan(
    style: const TextStyle(fontSize: 20),
    children: <InlineSpan>[
      const TextSpan(text: 'Години на самостійну роботу: '),
      TextSpan(
        text: '${course.hoursIndividualTotalField}',
        style: const TextStyle(fontWeight: FontWeight.bold)),
    ],
  ),
),
const SizedBox(height: 10),
Text.rich(
  TextSpan(
    style: const TextStyle(fontSize: 20),
    children: <InlineSpan>[
      const TextSpan(text: 'Усього годин: '),
      TextSpan(
        text: '${course.hoursOverallTotalField}',
```

```
                style: const TextStyle(fontWeight: FontWeight.bold)),
            ],
          ),
        ),
        const SizedBox(height: 10),
        Row(
          children: [
            Expanded(
              child: Text(
                'Кількість кредитів ЄКТС: ${course.creditsOverallTotalField != null &&
course.creditsOverallTotalField! % 1 == 0 ? course.creditsOverallTotalField!.toInt() :
course.creditsOverallTotalField}',
                style: TextStyle(
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                  color: Colors.red[600]),
      ), ), ], ), ], ), ); }

  Widget _courseDetailsUnfilledCourse(Course course) {
    return Align(
      alignment: Alignment.centerLeft,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          ElevatedButton(
            onPressed: () async {
              bool? result = await showDialog(
                context: context,
                builder: (BuildContext context) {
                  return NewCourseDialog(
                    isEdit: true,
                    isEditFilling: true,
                    course: course,
                    filledCourseSchedule: true,
                    filledNewRecordBook: course.isRecordBookFilled!,
                    currentSemester: selectedSemester,
                  );
                },
              );
              if (result == true) {
                setState(() {
```

```
            coursesFuture = generateCourses(selectedSemester!);
            coursesFuture.then((courses) {
              setState(() {
                expandedState = courses
                    .map((course) =>
                        !(course['course'].isScheduleFilled ?? false))
                    .toList();
          }); }); }); } },
          child: const Text('Додати інформацію'),
        ), ], ), ); }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Індивідуальний Навчальний План'),
      ),
      body: Center(
        child: Column(
          children: [
            const SizedBox(height: 5),
            DropdownMenuChooseSemester(
                onSelectedItemChanged: updateSelectedSemester),
            const SizedBox(height: 5),
            _addNewCourseButton(context),
            const SizedBox(height: 5),
            _sortDropDownMenu(),
            const SizedBox(height: 5),
            Expanded(child: _coursesListView(context)),
          ], ), ), ); } }

enum SortOption {
  alphabeticalAsc,
  alphabeticalDesc,
  hoursInClassDesc,
  hoursIndividualDesc,
  hoursOverallDesc,
}
```

Вміст файлу record_book_page.dart

```
class RecordBookPage extends StatefulWidget {
  const RecordBookPage({Key? key}) : super(key: key);

  @override
  State<RecordBookPage> createState() => _RecordBookPageState();
}

class _RecordBookPageState extends State<RecordBookPage> {
  final user = FirebaseAuth.instance.currentUser!;
  late Future<String> selectedSemester =
      DatabaseService.getStudentField(user.email, 'Current Semester');
  late Future<List<String>> semesterList =
      DatabaseService.getSemesterList(user.email);

  String? selectedSemesterPage;
  String? selectedCourse;
  Future<List<Map<String, dynamic>>> coursesFuture = Future.value([]);
  List<bool> expandedState = [];
  bool isSortedByDate = false;
  SortOption sortOption = SortOption.alphabeticalAsc;
  bool isGroupedByScoringType = false;

  @override
  void initState() {
    super.initState();
    selectedSemester.then((value) {
      setState(() {
        selectedSemesterPage = value;
        coursesFuture = generateCourses(selectedSemesterPage!);
      });
    });
  }

  void updateSelectedSemester(String selectedItem) {
    setState(() {
      selectedSemesterPage = selectedItem;
      coursesFuture = generateCourses(selectedItem);
    });
  }

  Future<List<Course>> fetchCourses(String semester) async {
```

```
  return await DatabaseService.getAllCourses(user.email!, semester);
}

List<bool> setExpandedState(List<Course> courses) {
  return List<bool>.filled(courses.length, false);
}

List<Course> sortCourses(List<Course> courses) {
  if (sortOption == SortOption.dateAsc) {
    courses.sort((a, b) => a.recordBookSelectedDateField!
        .compareTo(b.recordBookSelectedDateField!));
  } else if (sortOption == SortOption.dateDesc) {
    courses.sort((a, b) => b.recordBookSelectedDateField!
        .compareTo(a.recordBookSelectedDateField!));
  } else if (sortOption == SortOption.alphabeticalAsc) {
    courses.sort((a, b) => a.nameField!.compareTo(b.nameField!));
  } else if (sortOption == SortOption.alphabeticalDesc) {
    courses.sort((a, b) => b.nameField!.compareTo(a.nameField!));
  } else if (sortOption == SortOption.teacherAsc) {
    courses.sort((a, b) {
      if (a.recordBookTeacherField!.isEmpty &&
          b.recordBookTeacherField!.isEmpty) {
        return 0;
      } else if (a.recordBookTeacherField!.isEmpty) {
        return 1;
      } else if (b.recordBookTeacherField!.isEmpty) {
        return -1;
      } else {
        return a.recordBookTeacherField!.compareTo(b.recordBookTeacherField!);
      }
    });
  } else if (sortOption == SortOption.teacherDesc) {
    courses.sort((a, b) {
      if (a.recordBookTeacherField!.isEmpty &&
          b.recordBookTeacherField!.isEmpty) {
        return 0;
      } else if (a.recordBookTeacherField!.isEmpty) {
        return 1;
      } else if (b.recordBookTeacherField!.isEmpty) {
        return -1;
      } else {
```

```
        return b.recordBookTeacherField!.compareTo(a.recordBookTeacherField!);
      }
    });
  }

  if (isGroupedByScoringType) {
    courses.sort((a, b) {
      int aValue = a.scoringTypeField == 'Екзамен'
          ? 1
          : a.scoringTypeField == 'Залік'
              ? 2
              : 3;
      int bValue = b.scoringTypeField == 'Екзамен'
          ? 1
          : b.scoringTypeField == 'Залік'
              ? 2
              : 3;
      return aValue.compareTo(bValue);
    });
  }

  courses.sort((a, b) {
    if (a.isRecordBookFilled == b.isRecordBookFilled) {
      return 0;
    } else if (a.isRecordBookFilled == true) {
      return -1;
    } else {
      return 1;
    }
  });

  return courses;
}

Future<List<Map<String, dynamic>>> generateCourses(String semester) async {
  List<Course> courses = await fetchCourses(semester);
  expandedState = setExpandedState(courses);
  courses = sortCourses(courses);

  return courses.map((course) {
    return {
```

```
      'course': course,
      'isExpanded': false,
    };
  }).toList();
}

Widget _addScoresButton() {
  return ElevatedButton(
    child: const Text('Додати новий елемент'),
    onPressed: () async {
      bool? result = await showDialog(
        context: context,
        builder: (BuildContext context) {
          return NewCourseDialog(
            isRecordBook: true,
            currentSemester: selectedSemesterPage,
          );
        },
      );
      if (result == true) {
        setState(() {
          coursesFuture = generateCourses(selectedSemesterPage!);
        });
      }
    },
  );
}

Widget _sortDropDownMenu() {
  return DropdownButton<SortOption>(
    value: sortOption,
    icon: const Icon(Icons.arrow_downward),
    onChanged: (SortOption? newValue) {
      setState(() {
        sortOption = newValue!;
        coursesFuture = generateCourses(selectedSemesterPage!);
      });
    },
    items: const <DropdownMenuItem<SortOption>>[
      DropdownMenuItem<SortOption>(
        value: SortOption.alphabeticalAsc,
```

```
        child: Text('За алфавітом (А до Я)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.alphabeticalDesc,
        child: Text('За алфавітом (Я до А)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.dateAsc,
        child: Text('По даті (Старіші до новіших)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.dateDesc,
        child: Text('По даті (Новіші до старіших)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.teacherAsc,
        child: Text('За викладачем (А до Я)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.teacherDesc,
        child: Text('За викладачем (Я до Аcr)'),
      ),
    ],
  );
}

Widget _groupByScoringTypeCheckbox() {
  return CheckboxListTile(
    title: const Text('Групувати за формою підсумкового контролю',
        style: TextStyle(fontSize: 14)),
    value: isGroupedByScoringType,
    onChanged: (bool? value) {
      setState(() {
        isGroupedByScoringType = value!;
        coursesFuture = generateCourses(selectedSemesterPage!);
      });
    },
  );
}

Widget _categoryTitle(String title) {
```

```
  return Padding(
    padding: const EdgeInsets.all(8.0),
    child: Text(
      title,
      style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
    ),
  );
}

Widget _recordBookListView() {
  return FutureBuilder<List<Map<String, dynamic>>>(
    future: coursesFuture,
    builder: (BuildContext context,
        AsyncSnapshot<List<Map<String, dynamic>>> snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const CircularProgressIndicator();
      } else if (snapshot.hasError) {
        return Text('Помилка: ${snapshot.error}');
      } else {
        List<Map<String, dynamic>> examCoursesFilled = [];
        List<Map<String, dynamic>> scoringCoursesFilled = [];
        List<Map<String, dynamic>> otherCoursesFilled = [];
        List<Map<String, dynamic>> examCoursesNotFilled = [];
        List<Map<String, dynamic>> scoringCoursesNotFilled = [];
        List<Map<String, dynamic>> otherCoursesNotFilled = [];

        for (var courseMap in snapshot.data!) {
          Course course = courseMap['course'];
          if (course.isRecordBookFilled!) {
            if (course.scoringTypeField == 'Екзамен') {
              examCoursesFilled.add(courseMap);
            } else if (course.scoringTypeField == 'Залік') {
              scoringCoursesFilled.add(courseMap);
            } else {
              otherCoursesFilled.add(courseMap);
            }
          } else {
            if (course.scoringTypeField == 'Екзамен') {
              examCoursesNotFilled.add(courseMap);
            } else if (course.scoringTypeField == 'Залік') {
              scoringCoursesNotFilled.add(courseMap);
```

```
          } else {
            otherCoursesNotFilled.add(courseMap);
          }
        }
      }

      return Expanded(
        child: ListView(
          children: [
            if (isGroupedByScoringType && examCoursesFilled.isNotEmpty)
              ..._buildCategory('Екзамен', examCoursesFilled),
            if (isGroupedByScoringType && scoringCoursesFilled.isNotEmpty)
              ..._buildCategory('Залік', scoringCoursesFilled),
            if (isGroupedByScoringType && otherCoursesFilled.isNotEmpty)
              ..._buildCategory('Інше', otherCoursesFilled),
            if (isGroupedByScoringType && examCoursesNotFilled.isNotEmpty)
              ..._buildCategory('Екзамен', examCoursesNotFilled),
            if (isGroupedByScoringType &&
                scoringCoursesNotFilled.isNotEmpty)
              ..._buildCategory('Залік', scoringCoursesNotFilled),
            if (isGroupedByScoringType && otherCoursesNotFilled.isNotEmpty)
              ..._buildCategory('Інше', otherCoursesNotFilled),
            if (!isGroupedByScoringType)
              ...snapshot.data!
                  .map((courseMap) => _recordBookCell(courseMap['course']))
                  .toList(),
                  ], ), ); } }, ); }

List<Widget> _buildCategory(
    String title, List<Map<String, dynamic>> courses) {
  return [
    _categoryTitle(title),
    ...courses
        .map((courseMap) => _recordBookCell(courseMap['course']))
        .toList(),
  ];
}

Widget _recordBookCell(Course course) {
  Color backgroundColor = course.scoringTypeField == 'Екзамен'
      ? const Color.fromARGB(255, 184, 48, 38)
```

```
      : course.scoringTypeField == 'Залік'
          ? const Color.fromARGB(255, 250, 193, 8)
          : const Color.fromARGB(255, 60, 139, 63);
  return _buildCourseCell(course, backgroundColor);
}

Widget _buildCourseCell(Course course, Color borderColor) {
  return Padding(
    padding: const EdgeInsets.all(8.0),
    child: Card(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15.0),
      ),
      elevation: 3.0,
      child: Stack(
        children: [
          Container(
            padding: const EdgeInsets.all(10),
            decoration: BoxDecoration(
              border: Border.all(
                color: borderColor,
                width: 5.0,
              ),
              borderRadius: BorderRadius.circular(10),
            ),
            child: SingleChildScrollView(
              child: Column(
                children: [
                  Center(
                      child: Text(
                    course.nameField!,
                    style: const TextStyle(fontSize: 18),
                  )),
                  course.isRecordBookFilled ?? false
                      ? _buildFilledCourseDetails(course, borderColor)
                      : _buildEmptyCourseButton(course),
                ],
              ),
            ),
          ),
          if (!(course.isRecordBookFilled ?? false))
```

```
          Positioned(
            bottom: 0,
            right: 0,
            child: IconButton(
              icon: const Icon(Icons.delete),
              onPressed: () {
                showDeleteDialog(context, course, selectedSemesterPage!);
                  }, ), ), ], ), ), ); }

Widget _buildFilledCourseDetails(Course course, Color backgroundColor) {
  return Column(
    children: [
      const SizedBox(
        height: 10,
      ),
      if (course.recordBookTeacherField!.isNotEmpty)
        Column(
          children: [
            const SizedBox(
              height: 10,
            ),
            Row(
              children: [
                const Text(
                  "Викладач: ",
                  style: TextStyle(
                    fontSize: 20,
                  ),
                ),
                Expanded(
                  child: Text(
                    '${course.recordBookTeacherField}',
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 20,
                    ),
                    softWrap: true,
                    overflow: TextOverflow.clip,
                      ), ), ], ), ], ),
      const SizedBox(
        height: 10,
```

```
),
Row(
  children: [
    const Text('Форма підсумкового \nконтролю:',
        style: TextStyle(fontSize: 20)),
    Expanded(
      child: Text.rich(
        TextSpan(
          style: const TextStyle(fontSize: 20),
          children: <InlineSpan>[
            TextSpan(
              text: '${course.scoringTypeField}',
              style: TextStyle(
                fontWeight: FontWeight.bold,
                fontSize: 22,
                color: course.scoringTypeField == 'Екзамен'
                    ? Colors.red
                    : course.scoringTypeField == 'Залік'
                        ? Colors.orange
                        : Colors.green,
              ),
            ),
          ],
        ),
        softWrap: true,
        overflow: TextOverflow.clip,
        textAlign: TextAlign.end,
      ),
    ),
  ],
),
if (course.hoursOverallTotalField != null &&
    course.hoursOverallTotalField != 0)
  Column(
    children: [
      const SizedBox(
        height: 10,
      ),
      Row(
        children: [
          const Text('Кількість годин: ',
```

```
            style: TextStyle(fontSize: 20)),
        const Spacer(),
        Text(course.hoursOverallTotalField.toString(),
            textAlign: TextAlign.end,
            style: const TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ))
      ],
    ),
  ],
 ),
if (course.creditsOverallTotalField != null &&
    course.creditsOverallTotalField != 0)
  Column(
    children: [
      const SizedBox(
        height: 10,
      ),
      Row(
        children: [
          const Text('Кількість кредитів:',
              style: TextStyle(fontSize: 20)),
          const Spacer(),
          Text(
              course.creditsOverallTotalField != null &&
                      course.creditsOverallTotalField! % 1 == 0
                  ? course.creditsOverallTotalField!.toInt().toString()
                  : course.creditsOverallTotalField.toString(),
              textAlign: TextAlign.end,
              style: const TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
              ))
        ],
      ),
    ],
  ),
if (!course.recordBookSelectedDateField!.isAtSameMomentAs(
    DateTime.fromMillisecondsSinceEpoch(978307200000, isUtc: true)))
  Column(
```

```
          children: [
            const SizedBox(
              height: 10,
            ),
            Row(
              children: [
                const Text('Дата і час:', style: TextStyle(fontSize: 20)),
                const Spacer(),
                Text(
                  "${course.recordBookSelectedDateField?.year.toString().padLeft(4,'0')}-
${course.recordBookSelectedDateField?.month.toString().padLeft(2,0')}-
${course.recordBookSelectedDateField?.day.toString().padLeft(2,'0')}
${course.recordBookSelectedDateField?.hour.toString().padLeft(2,
'0')}:${course.recordBookSelectedDateField?.minute.toString().padLeft(2, '0')}",
                  textAlign: TextAlign.end,
                  style: const TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ))
              ],
            ),
          ],
        ),
        const SizedBox(
          height: 5,
        ),
        const Text('Оцінка', style: TextStyle(fontSize: 24)),
        Row(
          children: [
            Expanded(
              child: Stack(
                alignment: Alignment.center,
                children: [
                  Text(
                    course.recordBookScoreField == null ||
                            course.recordBookScoreField == 0
                        ? "-"
                        : course.recordBookScoreField.toString(),
                    textAlign: TextAlign.center,
                    style: const TextStyle(
                      fontSize: 24,
```

```
                fontWeight: FontWeight.bold,
              ),
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.end,
              children: [
                IconButton(
                  icon: const Icon(Icons.edit),
                  onPressed: () async {
                    bool? result = await showDialog(
                      context: context,
                      builder: (BuildContext context) {
                        return NewCourseDialog(
                          isEdit: true,
                          course: course,
                          isRecordBook: true,
                          filledNewRecordBook: true,
                          currentSemester: selectedSemesterPage,
                        );
                      },
                    );
                    if (result == true) {
                      setState(() {
                        coursesFuture =
                            generateCourses(selectedSemesterPage!);
                      });
                    }
                  },
                ),
                IconButton(
                  icon: const Icon(Icons.delete),
                  onPressed: () {
                    showDeleteDialog(
                        context, course, selectedSemesterPage!);
                  }, ), ], ), ], ), ), ], ), ], ); }


void showDeleteDialog(BuildContext context, Course course, String semester) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
```

```
return AlertDialog(
  title: const Text('Видалити елемент?'),
  actions: <Widget>[
    Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Expanded(
          child: Padding(
            padding: const EdgeInsets.all(8.0),
            child: Builder(
              builder: (context) => ElevatedButton(
                style: Theme.of(context).elevatedButtonTheme.style,
                child: const Center(
                    child: Text('Видалити зі сторінки',
                        textAlign: TextAlign.center)),
                onPressed: () async {
                  course.isRecordBookFilled = false;
                  await DatabaseService.createOrUpdateCourse(
                      user.email!, course, semester);
                  setState(() {
                    coursesFuture =
                        generateCourses(selectedSemesterPage!);
                  });
                  if (context.mounted) {
                    Navigator.of(context).pop();
              } }, ), ), ), ),
        Expanded(
          child: Padding(
            padding: const EdgeInsets.all(8.0),
            child: Builder(
              builder: (context) => ElevatedButton(
                style: Theme.of(context).elevatedButtonTheme.style,
                child: const Center(
                    child: Text('Повністтю видалити елемент',
                        textAlign: TextAlign.center)),
                onPressed: () async {
                  await DatabaseService.deleteCourse(
                      user.email!, course.nameField!);
                  setState(() {
                    coursesFuture =
                        generateCourses(selectedSemesterPage!);
```

```
                    });
                    if (context.mounted) {
                      Navigator.of(context).pop();
                    } }, ), ), ), ), ], ),
          TextButton(
            child:
                const Text('Закрити', style: TextStyle(color: Colors.brown)),
            onPressed: () {
              Navigator.of(context).pop();
        }, ), ], ); }, ); }

Widget _buildEmptyCourseButton(Course course) {
  return Container(
    padding: const EdgeInsets.all(10),
    child: ElevatedButton(
      onPressed: () async {
        bool? result = await showDialog(
          context: context,
          builder: (BuildContext context) {
            return NewCourseDialog(
              isEdit: true,
              isEditFilling: true,
              course: course,
              isRecordBook: true,
              filledNewRecordBook: true,
              filledCourseSchedule: course.isScheduleFilled ?? false,
              currentSemester: selectedSemesterPage,
            );
          },
        );
        if (result == true) {
          setState(() {
            coursesFuture = generateCourses(selectedSemesterPage!);
          });
        }
      },
      child: const Text('Додати інформацію'),
    ),
  );
}
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Залікова Книжка Студента'),
      ),
      body: Column(
        children: [
          const SizedBox(
            height: 5,
          ),
          Center(
              child: DropdownMenuChooseSemester(
                  onSelectedItemChanged: updateSelectedSemester)),
          _addScoresButton(),
          _sortDropDownMenu(),
          _groupByScoringTypeCheckbox(),
          _recordBookListView(),
        ],
      ),
    );
  }
}

enum SortOption {
  dateAsc,
  dateDesc,
  alphabeticalAsc,
  alphabeticalDesc,
  teacherAsc,
  teacherDesc
}
```

Вміст файлу my_calendar.dart

```
class MyCalendar extends StatefulWidget {
  final List<Course> courses;
  final List<EventSchedule> events;
  final String? selectedSemester;

  const MyCalendar(
```

```
      {super.key,
      required this.courses,
      required this.events,
      this.selectedSemester});

  @override
  MyCalendarState createState() => MyCalendarState();
}

class MyCalendarState extends State<MyCalendar> {
  CalendarFormat _calendarFormat = CalendarFormat.month;
  DateTime _focusedDay = DateTime.now();
  DateTime? _selectedDay;

  Color getHighlightColor(DateTime date) {
    for (var course in widget.courses) {
      if (isSameDay(course.recordBookSelectedDateField!, date) &&
          course.isEvent == true) {
        switch (course.scoringTypeField) {
          case 'Екзамен':
            return Colors.red;
          case 'Залік':
            return Colors.yellow[800]!;
          default:
            return Colors.green;
        }
      }
    }

    for (var event in widget.events) {
      if (isSameDay(event.eventDateStart!, date) ||
          isSameDay(event.eventDateEnd!, date)) {
        switch (event.eventType) {
          case 'Екзамен':
            return Colors.red;
          case 'Залік':
            return Colors.yellow[800]!;
          default:
            return Colors.green;
        }
      }
```

```
  }

  return Colors.transparent;
}

Widget buildDotsInBetween(BuildContext context, DateTime date, List events,
    Function getEventsForDate) {
  List<EventSchedule> eventsForDate = getEventsForDate(date);
  Set<String> eventTypes =
      eventsForDate.map((e) => e.eventType).whereType<String>().toSet();
  List<Color> dotColors = [];
  if (eventTypes.contains('Екзамен')) {
    dotColors.add(Colors.red);
  }
  if (eventTypes.contains('Залік')) {
    dotColors.add(Colors.yellow[800]!);
  }
  if (eventTypes.length > dotColors.length) {
    dotColors.add(Colors.green);
  }
  return Positioned(
    bottom: 1,
    child: Row(
      children: dotColors
          .map((color) => Container(
                margin: const EdgeInsets.symmetric(horizontal: 1),
                width: 5,
                height: 5,
                decoration: BoxDecoration(
                  shape: BoxShape.circle,
                  color: color,
                ),
              ))
          .toList(),
    ),
  );
}

List<EventSchedule> getEventsForDate(DateTime date) {
  List<EventSchedule> eventsForDate = [];
  for (var event in widget.events) {
```

```
    if (event.eventDateStart!.isBefore(date) &&
        event.eventDateEnd!.isAfter(date.add(const Duration(days: 1)))) {
      eventsForDate.add(event);
    }
  }
  return eventsForDate;
}

Widget? defaultBuilderFunction(
    BuildContext context, DateTime start, DateTime end) {
  Color highlightColor = getHighlightColor(start);

  if (highlightColor != Colors.transparent) {
    return Center(
      child: Container(
        width: 30,
        height: 30,
        decoration: BoxDecoration(
          color: highlightColor,
          shape: BoxShape.circle,
        ),
        alignment: Alignment.center,
        child: Text(
          '${start.day}',
          style: const TextStyle().copyWith(color: Colors.white),
        ),
      ),
    );
  }
  return Center(
    child: Container(
      width: 30,
      height: 30,
      alignment: Alignment.center,
      child: Text(
        '${start.day}',
      ),
    ),
  );
}
```

```
@override
Widget build(BuildContext context) {
  return TableCalendar(
    locale: 'uk_UA',
    firstDay: DateTime.utc(2000, 1, 1),
    lastDay: DateTime.utc(2030, 3, 14),
    focusedDay: _focusedDay,
    calendarFormat: _calendarFormat,
    selectedDayPredicate: (day) {
      return isSameDay(_selectedDay, day);
    },
    availableCalendarFormats: const {
      CalendarFormat.month: 'Місяць',
      CalendarFormat.twoWeeks: '2 Тижні',
      CalendarFormat.week: 'Тиждень',
    },
    onDaySelected: (selectedDay, focusedDay) {
      setState(() {
        _selectedDay = selectedDay;
        _focusedDay = focusedDay;
      });
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return CalendarDialog(
            selectedDate: selectedDay,
            courses: widget.courses,
            events: widget.events,
            selectedSemester: widget.selectedSemester,
          );
        },
      );
    },
    onFormatChanged: (format) {
      setState(() {
        _calendarFormat = format;
      });
    },
    onPageChanged: (focusedDay) {
      _focusedDay = focusedDay;
    },
```

```
      calendarBuilders: CalendarBuilders(
        markerBuilder: (context, date, events) =>
            buildDotsInBetween(context, date, events, getEventsForDate),
        defaultBuilder: defaultBuilderFunction,
      ),
    );
  }
}

class Event {
  final String title;

  Event(this.title);
}
```

Вміст файлу class_list_view_builder.dart

```
class ClassListView extends StatefulWidget {
  final List<dynamic> combinedList;
  final Function updateState;
  final User? user;
  final String? selectedSemester;

  const ClassListView({
    Key? key,
    required this.combinedList,
    required this.updateState,
    this.user,
    this.selectedSemester,
  }) : super(key: key);

  @override
  State<ClassListView> createState() => _ClassListViewState();
}

class _ClassListViewState extends State<ClassListView> {
  bool isDefaultDate(DateTime? date) {
    return date!.millisecondsSinceEpoch == 978307200000;
  }
```

```
void showDeleteDialog(BuildContext context, dynamic item) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Видалити елемент?'),
        actions: <Widget>[
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              if (item is Course)
                Expanded(
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Builder(
                      builder: (context) => ElevatedButton(
                        style: Theme.of(context).elevatedButtonTheme.style,
                        child: const Center(
                            child: Text('Видалити зі сторінки',
                                textAlign: TextAlign.center)),
                        onPressed: () async {
                          item.isEvent = false;
                          await DatabaseService.createOrUpdateCourse(
                              widget.user!.email!,
                              item,
                              widget.selectedSemester!);
                          widget.updateState();
                          if (context.mounted) {
                            Navigator.of(context).pop();
                          } }, ), ), ), ),
              Expanded(
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Builder(
                    builder: (context) => ElevatedButton(
                      style: Theme.of(context).elevatedButtonTheme.style,
                      child: const Center(
                          child: Text('Повністтю видалити елемент',
                              textAlign: TextAlign.center)),
                      onPressed: () async {
```

```
                    if (item is Course) {
                      await DatabaseService.deleteCourse(
                          widget.user!.email!, item.nameField!);
                    } else if (item is EventSchedule) {
                      await DatabaseService.deleteEvent(
                          widget.user!.email!, item.eventName!);
                    }
                    widget.updateState();
                    if (context.mounted) {
                      Navigator.of(context).pop();
                    } }, ), ), ), ), ], ),
            TextButton(
              child:
                  const Text('Закрити', style: TextStyle(color: Colors.brown)),
              onPressed: () {
                Navigator.of(context).pop();
              }, ), ], ); }, ); }

  Widget _buildListView(List<dynamic> combinedList) {
    return SingleChildScrollView(
      child: Column(
        children: widget.combinedList.map((item) {
          return _buildListItem(item);
        }).toList(),
      ),
    );
  }

  Widget _buildListItem(dynamic item) {
    if (item is Course) {
      String subtitle =
          'Дата і час: ${item.recordBookSelectedDateField?.year.toString().padLeft(4, '0')}-
${item.recordBookSelectedDateField?.month.toString().padLeft(2, '0')}-
${item.recordBookSelectedDateField?.day.toString().padLeft(2, '0')}
${item.recordBookSelectedDateField?.hour.toString().padLeft(2,
'0')}:${item.recordBookSelectedDateField?.minute.toString().padLeft(2, '0')}\nТип:
${item.scoringTypeField!}';
      return _buildListItemDesign(item.nameField!, subtitle, () {
        showDialog(
          context: context,
          builder: (BuildContext context) {
```

```
            return NewCourseDialog(
              isClassSchedule: true,
              isRecordBook: false,
              currentSemester: widget.selectedSemester,
              isEdit: true,
              course: item,
            );
          },
        );
      }, () {
        showDeleteDialog(context, item);
      }, item);
    } else if (item is EventSchedule) {
      String subtitle = 'Тип: ${item.eventType!}';
      if (!isDefaultDate(item.eventDateStart)) {
        subtitle +=
            '\nДата і час початку: ${item.eventDateStart?.year.toString().padLeft(4, '0')}-
${item.eventDateStart?.month.toString().padLeft(2, '0')}-${item.eventDateStart?.day.toString().padLeft(2,
'0')} ${item.eventDateStart?.hour.toString().padLeft(2,
'0')}:${item.eventDateStart?.minute.toString().padLeft(2, '0')}';
      }
      if (!isDefaultDate(item.eventDateEnd)) {
        subtitle +=
            '\nДата і час кінця: ${item.eventDateEnd?.year.toString().padLeft(4, '0')}-
${item.eventDateEnd?.month.toString().padLeft(2, '0')}-${item.eventDateEnd?.day.toString().padLeft(2,
'0')} ${item.eventDateEnd?.hour.toString().padLeft(2,
'0')}:${item.eventDateEnd?.minute.toString().padLeft(2, '0')}';
      }
      return _buildListItemDesign(item.eventName!, subtitle, () {
        showDialog(
          context: context,
          builder: (BuildContext context) {
            return NewEventDialog(
              isEdit: true,
              event: item,
              selectedSemester: widget.selectedSemester,
              onUpdate: widget.updateState,
            );
          },
        );
      }, () {
```

```
      showDeleteDialog(context, item);
    }, item);
  } else {
    throw Exception('Unknown type in combinedList');
  }
}

Widget _buildListItemDesign(String title, String subtitle, Function onEdit,
    Function onDelete, dynamic item) {
  BorderRadius borderRadius = BorderRadius.circular(8.0);
  return Padding(
    padding: const EdgeInsets.symmetric(vertical: 4.0, horizontal: 8.0),
    child: Card(
      elevation: 5.0,
      child: Container(
        decoration: BoxDecoration(
          border: Border.all(
            color: Colors.grey[700]!,
            width: 2,
          ),
          borderRadius: borderRadius,
        ),
        child: ListTile(
          title: Text(title),
          subtitle: Text(subtitle),
          trailing: Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              IconButton(
                icon: const Icon(Icons.edit),
                onPressed: () => onEdit(),
              ),
              IconButton(
                icon: const Icon(Icons.delete),
                onPressed: () => onDelete(),
              ), ], ), ), ), ), ); }

@override
Widget build(BuildContext context) {
  return _buildListView(widget.combinedList);
} }
```

Вміст файлу calendar_dialog.dart

```
class CalendarDialog extends StatelessWidget {
  final DateTime selectedDate;
  final List<Course> courses;
  final List<EventSchedule> events;
  final String? selectedSemester;

  const CalendarDialog({
    Key? key,
    required this.selectedDate,
    required this.courses,
    required this.events,
    this.selectedSemester,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    User? user = FirebaseAuth.instance.currentUser;

    List<Course> selectedCourses = courses.where((course) {
      return course.recordBookSelectedDateField!.day == selectedDate.day &&
          course.recordBookSelectedDateField!.month == selectedDate.month &&
          course.recordBookSelectedDateField!.year == selectedDate.year &&
          course.isEvent == true;
    }).toList();

    List<EventSchedule> selectedEvents = events.where((event) {
      return (event.eventDateStart!.day == selectedDate.day &&
              event.eventDateStart!.month == selectedDate.month &&
              event.eventDateStart!.year == selectedDate.year) ||
          (event.eventDateEnd!.day == selectedDate.day &&
              event.eventDateEnd!.month == selectedDate.month &&
              event.eventDateEnd!.year == selectedDate.year);
    }).toList();

    return AlertDialog(
      title: Text(
          'Події для дати: ${DateFormat('dd.MM.yyyy').format(selectedDate)}'),
      content: SizedBox(
        width: double.maxFinite,
```

```
        child: ListView(
          shrinkWrap: true,
          children: [
            ClassListView(
              combinedList: [...selectedCourses, ...selectedEvents],
              updateState: () {},
              user: user,
              selectedSemester: selectedSemester,
            ),
          ],
        ),
      ),
      actions: <Widget>[
        TextButton(
          child: const Text('Зберегти', style: TextStyle(color: Colors.brown)),
          onPressed: () {
            Navigator.of(context).pop();
          },
        ),
      ],
    );
  }
}
```

Вміст файлу new_event_dialog.dart

```
class NewEventDialog extends StatefulWidget {
  NewEventDialog({
    this.onUpdate,
    this.isEdit = false,
    this.event,
    this.selectedSemester,
    Key? key,
  }) : super(key: key);

  final Function? onUpdate;
  bool isEdit;
  final EventSchedule? event;
  final String? selectedSemester;

  @override
```

```
  State<NewEventDialog> createState() => _NewEventDialogState();
}

class _NewEventDialogState extends State<NewEventDialog> {
  final user = FirebaseAuth.instance.currentUser!;
  final eventNameController = TextEditingController();
  final eventTypeController = TextEditingController();
  String? selectedSemesterPage;
  DateTime? eventDateStart;
  DateTime? eventDateEnd;
  String? selectedScoringType = 'Інше';

  @override
  void initState() {
    super.initState();
    if (widget.isEdit && widget.event != null) {
      eventNameController.text = widget.event!.eventName ?? '';
      eventTypeController.text = widget.event!.eventType ?? '';
      eventDateStart = widget.event!.eventDateStart;
      eventDateEnd = widget.event!.eventDateEnd;
    }
    selectedSemesterPage = widget.selectedSemester;
  }

  TextButton _saveButton(BuildContext context) {
    return TextButton(
      child: const Text('Зберегти', style: TextStyle(color: Colors.brown)),
      onPressed: () async {
        try {
          String eventType;
          if (selectedScoringType == 'Інше') {
            eventType = eventTypeController.text.isEmpty
                ? 'Інше'
                : eventTypeController.text;
          } else {
            eventType = selectedScoringType!;
          }

          EventSchedule newEvent = EventSchedule(
            eventName: eventNameController.text,
            eventType: eventType,
```

```
        eventDateStart: eventDateStart,
        eventDateEnd: eventDateEnd,
      );
      await DatabaseService.createOrUpdateEvent(
        user.email,
        newEvent,
        selectedSemesterPage!,
      );
      if (widget.onUpdate != null) {
        widget.onUpdate!();
      }
      if (context.mounted) {
        Navigator.of(context).pop();
      }
    } catch (e) {
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: const Text('Помилка'),
            content: Text(e.toString()),
            actions: <Widget>[
              TextButton(
                child: const Text('Закрити'),
                onPressed: () {
                  Navigator.of(context).pop();
        }, ), ], ); }, ); } }, ); }

@override
Widget build(BuildContext context) {
  return AlertDialog(
    content: SingleChildScrollView(
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: <Widget>[
          Text(widget.isEdit ? 'Змінити подію' : 'Додати нову подію'),
          const SizedBox(height: 5),
          DropdownMenuChooseSemester(
            initialSemester: selectedSemesterPage,
            onSelectedItemChanged: (selectedItem) {
              setState(() {
```

```
      selectedSemesterPage = selectedItem;
    });
  },
),
TextField(
  controller: eventNameController,
  decoration: const InputDecoration(
    labelText: 'Назва',
  ),
),
DropdownButton<String>(
  value: selectedScoringType,
  onChanged: (String? newValue) {
    setState(() {
      selectedScoringType = newValue;
    });
  },
  items: <String>['Екзамен', 'Залік', 'Інше']
      .map<DropdownMenuItem<String>>((String value) {
    return DropdownMenuItem<String>(
      value: value,
      child: Text(value),
    );
  }).toList(),
),
TextField(
  controller: eventTypeController,
  decoration: const InputDecoration(
    labelText: 'Тип',
  ),
  enabled: selectedScoringType == 'Інше',
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    ElevatedButton(
      child: const Text('Дата початку'),
      onPressed: () async {
        eventDateStart = await selectDate(context);
        setState(() {});
      },
```

```
            ),
            IconButton(
              icon: const Icon(Icons.delete),
              onPressed: () {
                setState(() {
                  eventDateStart = null;
                });
              },
            ),
          ],
        ),
        Text(eventDateStart != null
            ? '${eventDateStart!.year.toString().padLeft(4, '0')}-
${eventDateStart!.month.toString().padLeft(2, '0')}-${eventDateStart!.day.toString().padLeft(2, '0')}
${eventDateStart!.hour.toString().padLeft(2, '0')}:${eventDateStart!.minute.toString().padLeft(2, '0')}'
            : ''),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              child: const Text('Дата кінця'),
              onPressed: () async {
                eventDateEnd = await selectDate(context);
                setState(() {});
              },
            ),
            IconButton(
              icon: const Icon(Icons.delete),
              onPressed: () {
                setState(() {
                  eventDateEnd = null;
                });
              },
            ),
          ],
        ),
        Text(eventDateEnd != null
            ? '${eventDateEnd!.year.toString().padLeft(4, '0')}-${eventDateEnd!.month.toString().padLeft(2,
'0')}-${eventDateEnd!.day.toString().padLeft(2, '0')} ${eventDateEnd!.hour.toString().padLeft(2,
'0')}:${eventDateEnd!.minute.toString().padLeft(2, '0')}'
            : ''),
```

```
          ],
        ),
      ),
      actions: <Widget>[
        TextButton(
          child: const Text('Закрити', style: TextStyle(color: Colors.brown)),
          onPressed: () {
            Navigator.of(context).pop();
          },
        ),
        _saveButton(context),
      ],
    );
  }
}
```

Вміст файлу class_schedule_page.dart

```
class ClassSchedulePage extends StatefulWidget {
  const ClassSchedulePage({Key? key}) : super(key: key);

  @override
  ClassSchedulePageState createState() => ClassSchedulePageState();
}

class ClassSchedulePageState extends State<ClassSchedulePage> {
  User? user = FirebaseAuth.instance.currentUser;
  Future<String>? selectedSemesterOfUser;
  String? selectedSemester;
  Future<List<Course>>? coursesFuture;
  Future<List<EventSchedule>>? eventsFuture;
  Future<List<dynamic>>? allDataFuture;
  SortOption sortOption = SortOption.byNameAtoZ;

  @override
  void initState() {
    super.initState();
    if (user != null) {
      selectedSemesterOfUser =
          DatabaseService.getStudentField(user!.email!, 'Current Semester');
```

```
    selectedSemesterOfUser?.then((value) => selectedSemester = value);
    allDataFuture = initializeData();
  }
}

void updateState() {
  setState(() {
    allDataFuture = initializeData();
  });
}

Widget _sortDropDownMenu() {
  return DropdownButton<SortOption>(
    value: sortOption,
    icon: const Icon(Icons.arrow_downward),
    onChanged: (SortOption? newValue) {
      setState(() {
        sortOption = newValue!;
        allDataFuture = initializeData();
      });
    },
    items: const <DropdownMenuItem<SortOption>>[
      DropdownMenuItem<SortOption>(
        value: SortOption.byNameAtoZ,
        child: Text('За алфавітом (А до Я)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.byNameZtoA,
        child: Text('За алфавітом (Я до А)'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.byStartDate,
        child: Text('По даті початку'),
      ),
      DropdownMenuItem<SortOption>(
        value: SortOption.byEndDate,
        child: Text('По даті кінця'),
      ),
    ],
  );
}
```

```
bool isDefaultDate(DateTime? date) {
  return date!.millisecondsSinceEpoch == 978307200000;
}

Future<List<dynamic>> initializeData() async {
  String selectedSemester = await selectedSemesterOfUser!;
  return Future.wait([
    fetchCourses(selectedSemester),
    fetchEvents(selectedSemester),
  ]);
}

Future<List<Course>> fetchCourses(String semester) async {
  if (user!.email == null) {
    throw Exception('User email is null');
  }
  List<Course> courses =
      await DatabaseService.getAllCourses(user!.email!, semester);
  courses.sort((a, b) {
    switch (sortOption) {
      case SortOption.byNameAtoZ:
        return a.nameField!.compareTo(b.nameField!);
      case SortOption.byNameZtoA:
        return b.nameField!.compareTo(a.nameField!);
      case SortOption.byStartDate:
      case SortOption.byEndDate:
        return a.recordBookSelectedDateField!
            .compareTo(b.recordBookSelectedDateField!);
    }
  });
  return courses;
}

Future<List<EventSchedule>> fetchEvents(String semester) async {
  if (user!.email == null) {
    throw Exception('User email is null');
  }
  List<EventSchedule> events =
      await DatabaseService.getAllEvents(user!.email!, semester);
  events.sort((a, b) {
```

```
    switch (sortOption) {
      case SortOption.byNameAtoZ:
        return a.eventName!.compareTo(b.eventName!);
      case SortOption.byNameZtoA:
        return b.eventName!.compareTo(a.eventName!);
      case SortOption.byStartDate:
        return a.eventDateStart!.compareTo(b.eventDateStart!);
      case SortOption.byEndDate:
        return a.eventDateEnd!.compareTo(b.eventDateEnd!);
    }
  });
  return events;
}

Widget _addNewEventButton(BuildContext context) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      ElevatedButton(
        onPressed: () {
          showDialog(
            context: context,
            builder: (BuildContext context) {
              return NewEventDialog(
                selectedSemester: selectedSemester,
                onUpdate: updateState,
              );
            },
          );
        },
        child: const Text('Додати нову подію'),
      ),
      ElevatedButton(
        onPressed: () async {
          final result = await showDialog(
            context: context,
            builder: (BuildContext context) {
              return NewCourseDialog(
                isClassSchedule: true,
                isRecordBook: false,
                filledNewRecordBook: false,
```

```
            filledCourseSchedule: false,
            currentSemester: selectedSemester,
          );
        },
      );
      if (result != null && result) {
        updateState();
      }
    },
    child: const Text('Додати новий елемент'),
  ),
 ],
 );
}

Widget _buildFutureBuilder() {
 return FutureBuilder<List<dynamic>>(
  future: allDataFuture,
  builder: (context, snapshot) {
   if (snapshot.connectionState == ConnectionState.waiting) {
    return const Center(child: CircularProgressIndicator());
   } else if (snapshot.hasError) {
    return Center(child: Text('Помилка: ${snapshot.error}'));
   } else {
    List<Course> courses = snapshot.data![0];
    List<EventSchedule> events = snapshot.data![1];
    return _buildBody(courses, events);
   }
  },
 );
}

Widget _buildBody(List<Course> courses, List<EventSchedule> events) {
 return SingleChildScrollView(
  child: Column(
   children: [
    _buildCalendar(courses, events),
    _buildDropdownMenu(),
    _sortDropDownMenu(),
    _addNewEventButton(context),
    ClassListView(
```

```
          combinedList: _combineLists(_filterCourses(courses), events),
          updateState: updateState,
          user: user,
          selectedSemester: selectedSemester,
        ),
      ],
    ),
  );
}

Widget _buildCalendar(List<Course> courses, List<EventSchedule> events) {
  return MyCalendar(
    courses: courses,
    events: events,
    selectedSemester: selectedSemester,
  );
}

Widget _buildDropdownMenu() {
  return DropdownMenuChooseSemester(
    initialSemester: selectedSemester,
    onSelectedItemChanged: (selectedItem) {
      setState(() {
        selectedSemester = selectedItem;
        allDataFuture = Future.wait([
          fetchCourses(selectedSemester!),
          fetchEvents(selectedSemester!),
        ]);
      });
    },
  );
}

List<Course> _filterCourses(List<Course> courses) {
  return courses.where((course) {
    return !isDefaultDate(course.recordBookSelectedDateField) &&
        course.isEvent == true;
  }).toList();
}

List<dynamic> _combineLists(
```

```
    List<Course> filteredCourses, List<EventSchedule> events) {
  return [...filteredCourses, ...events];
}

void showDeleteDialog(BuildContext context, dynamic item) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: const Text('Видалити елемент?'),
        actions: <Widget>[
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              if (item is Course)
                Expanded(
                  child: Padding(
                    padding: const EdgeInsets.all(8.0),
                    child: Builder(
                      builder: (context) => ElevatedButton(
                        style: Theme.of(context).elevatedButtonTheme.style,
                        child: const Center(
                            child: Text('Видалити зі сторінки',
                                textAlign: TextAlign.center)),
                        onPressed: () async {
                          item.isEvent = false;
                          await DatabaseService.createOrUpdateCourse(
                              user!.email!, item, selectedSemester!);
                          updateState();
                          if (context.mounted) {
                            Navigator.of(context).pop();
                          }
                        },
                      ),
                    ),
                  ),
                ),
              Expanded(
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Builder(
```

```
                builder: (context) => ElevatedButton(
                  style: Theme.of(context).elevatedButtonTheme.style,
                  child: const Center(
                    child: Text('Повністтю видалити елемент',
                      textAlign: TextAlign.center)),
                  onPressed: () async {
                   if (item is Course) {
                    await DatabaseService.deleteCourse(
                      user!.email!, item.nameField!);
                   } else if (item is EventSchedule) {
                    await DatabaseService.deleteEvent(
                      user!.email!, item.eventName!);
                   }
                   updateState();
                   if (context.mounted) {
                    Navigator.of(context).pop();
                   } }, ), ), ), ), ], ),
          TextButton(
            child:
              const Text('Закрити', style: TextStyle(color: Colors.brown)),
            onPressed: () {
             Navigator.of(context).pop();
            }, ), ], ); }, );  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Графік Освітнього Процесу'),
      ),
      body: _buildFutureBuilder(),
    );
  }
}

enum SortOption {
  byNameAtoZ,
  byNameZtoA,
  byStartDate,
  byEndDate,
}
```