КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теорії та технології програмування

Кваліфікаційна робота на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформатика»

спеціальності 122 «Комп'ютерні науки»
на тему:

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ ОСОБИСТИХ АКТИВНОСТЕЙ

Виконала студентка 4-го курсу
Савонік Оксана Миколаївна

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Омельчук Людмила Леонідівна

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«_____»_____ 2021 р,
протокол №_____
Завідувач кафедри
Нікітченко М.С. _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 47 сторінок, 48 ілюстрацій, 5 таблиць, 14 джерел посилань.

ANDROID, DART, FLUTTER, SQLITE, SHARED PREFERENCES.

Об’єктом роботи є процес планування особистих активностей у власному телефоні.

Предметом роботи є мультизадачний мобільний застосунок для планування, що поєднує в собі трекер звичок, список покупок та список щоденних справ.

Метою роботи є створення застосунку, який буде поєднувати усі необхідні компоненти для планування дня.

Методи розроблення: проектування бази даних, розробка програмного продукту на основі еволюційної методології.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Android Studio, мова програмування Dart, фреймворк Flutter, бібліотеки Flutter_Local_Notifications, sqflite, intl, Shared_Preferences, Provider.

Результати роботи: виконано загальний аналіз необхідних технологій, проаналізовано переваги та недоліки зазначених вище бібліотек, розроблено застосунок, який дозволяє планувати щоденні активності, реалізовано зручний та простий інтерфейс програми. Застосунок може допомогти користувачам планувати та контролювати свій день.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5

ВСТУП 6

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ 8

1.1 Огляд проблеми 8

1.2 Огляд наявних на ринку застосунків 8

1.2.1 Мобільний застосунок «Daily Planner» 8

1.2.2 Мобільний застосунок «Any.do» 11

1.2.3 Мобільний застосунок «Habit360» 12

1.2.4 Мобільний застосунок «План тижня» 13

1.3 Постановка задачі 15

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ 17

2.1 Мова програмування Dart 17

2.2 Фреймворк Flutter 18

2.3 База даних SQLite 19

РОЗДІЛ 3. АРХІТЕКТУРА ЗАСТОСУНКУ ТА ПРИНЦИПИ ЙОГО РОБОТИ 21

3.1 Планувальник завдань 21

3.2 Трекер звичок 24

3.3 Список покупок 26

3.4 Блок налаштувань 28

РОЗДІЛ 4. ІНСТРУКЦІЯ КОРИСТУВАЧА 30

ВИСНОВКИ 45

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 46

ДОДАТКИ.....48
Додаток А. Діаграма прецедентів.....48
Фрагмент 1.....48
Фрагмент 2.....49
Додаток Б. Діаграма класів.....50

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API – Application Programming Interface, прикладний програмний інтерфейс;

DOM – Document Object Model, інтерфейс, який дозволяє програмі отримати доступ до вмісту HTML;

HTTP – HyperText Transfer Protocol, протокол передачі гіпертексту;

HTML – HyperText Markup Language, мова розмітки гіпертексту;

IDE – Integrated Design Environment, інтегроване середовище розробки;

iOS – iPhone Operation System, операційна система iPhone;

JSON – JavaScript Object Notation, повідомлення об’єкта JavaScript;

SDK – Software Development Kit, набір засобів розробки;

SQL – Structured Query Language, структурована мова запитів;

БД – база даних;

СКБД – система керування базами даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному світі дуже важливо планувати свій день заздалегідь. Адже у всіх так багато справ, що просто неможливо все втримати у голові.

Звичайно, перше що приходить на думку це завести блокнот і вести свої справи там. Але у багатьох людей немає часу та і бажання купувати блокнот, носити його усюди з собою і просто вести його. Проте, мобільний телефон, є у кожного. Дослідження [1] показали, що середньостатистична людина близько 9 годин присвячує роботі зі смартфоном. Тому виникає можливість впорядковувати всі свої справи в телефоні, який завжди під рукою.

На ринку представлена достатньо велика кількість різноманітних мобільних застосунків присвячених плануванню дня, але вони зазвичай зосереджені на чомусь одному – чи то запис щоденних справ, чи формування списку покупок.

Актуальність роботи та підстави для її виконання. Якщо використовувати блокнот для планування активностей, то деяким людям може швидко набриднути щодня прописувати свої справи у блокноті, організовувати їх, привабливо оформляти, окрім цього, завжди існують ситуації, коли блокнота немає під рукою, що може призвести до втрати інтересу до планування.

Звичайно, існують застосунки, де можна записувати свої справи чи вести список покупок, але мати різні застосунки для планування свого життя не дуже раціонально і зручно.

Тому з’являється необхідність у створенні єдиного застосунку, який би поєднував у собі усі потрібні компоненти для планування щоденних активностей.

Мета і завдання роботи. Метою кваліфікаційної роботи є розробка програмного засобу для планування своїх активностей, який би поєднував у собі найнеобхідніші складові, такі як ведення щоденних справ, створення списку покупок, формування персональних звичок. Для досягнення цієї мети поставлено такі завдання:

1. Дослідити інформацію про існуючі застосунки планування.
2. Розробити технічне завдання до продукту.
3. Ознайомитись з теоретичними відомостями по необхідних для застосування технологіях.
4. Розробити інтерфейс та дизайн програмного продукту.
5. Розробити програму відповідно до складеного технічного завдання.

Об’єкт, методи й засоби розроблення. Об’єктом розроблення кваліфікаційної роботи є процес планування особистих активностей за допомогою мобільного застосунку.

В якості інструменту створення програмного засобу було обрано Android Studio – інтегроване середовище розробки (IDE) для платформи Android. В якості фреймворку використовувався Flutter – SDK для створення застосунків для платформ Android та iOS. Flutter використовує мову програмування Dart, яка позиціонується як заміна JavaScript.

Можливі сфери застосування. Результати кваліфікаційної роботи може використовувати кожен для планування та контролю щоденних активностей.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд проблеми

Більшість людей переслідує одна спільна проблема – нестача часу. Кожен хоча б раз у житті задумувався над тим, чому в добі лише двадцять чотири години. Ми постійно поспішаємо, а часу від того більше не стає. Вихід тут тільки один – раціональне планування дня.

Найголовніше – постійно бачити чіткий план дій на день. А так як усі люди дуже багато часу проводять у смартфонах [1], виникає можливість фіксувати свої справи там.

В кваліфікаційній роботі розроблено мобільний застосунок для планування персональних активностей, який поєднує в собі ведення щоденних справ, створення списку покупок, формування звичок. Таким чином користувачу непотрібно встановлювати безліч застосунків для планування, адже все буде зберігатися централізовано. Усі функції застосунку пов’язані між собою, що не дає користувачу проґавити жодну зі своїх справ.

1.2 Огляд наявних на ринку застосунків

Розглянемо представлені на ринку застосунки призначені для планування особистих активностей. Для дослідження було обрано та проаналізовано застосунки з різною функціональністю. Результати дослідження наведено нижче.

1.2.1 Мобільний застосунок «Daily Planner»

На торговому майданчику Play Market застосунок «Daily Planner» (див. рис. 1.2.1.1) [2] описано як щоденний планувальник, організатор цілей, плановий планувальник. На 21.03.2021 р. застосунок завантажили більше ста тисяч разів. Він представлений на ринку більше трьох років та середня оцінка застосунку складає 4.8.

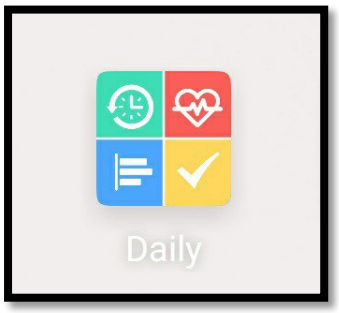


Рисунок 1.2.1.1. Логотип застосунку Daily Planner

Після користування застосунком було виявлено, що він призначений тільки для відстежування своїх звичок (див. рис. 1.2.1.2). Застосунок дозволяє додавати нові звички, але тільки назву звички, без будь-якої додаткової інформації, відстежувати, які звички були виконані сьогодні, ставити нагадування про виконання звичок.

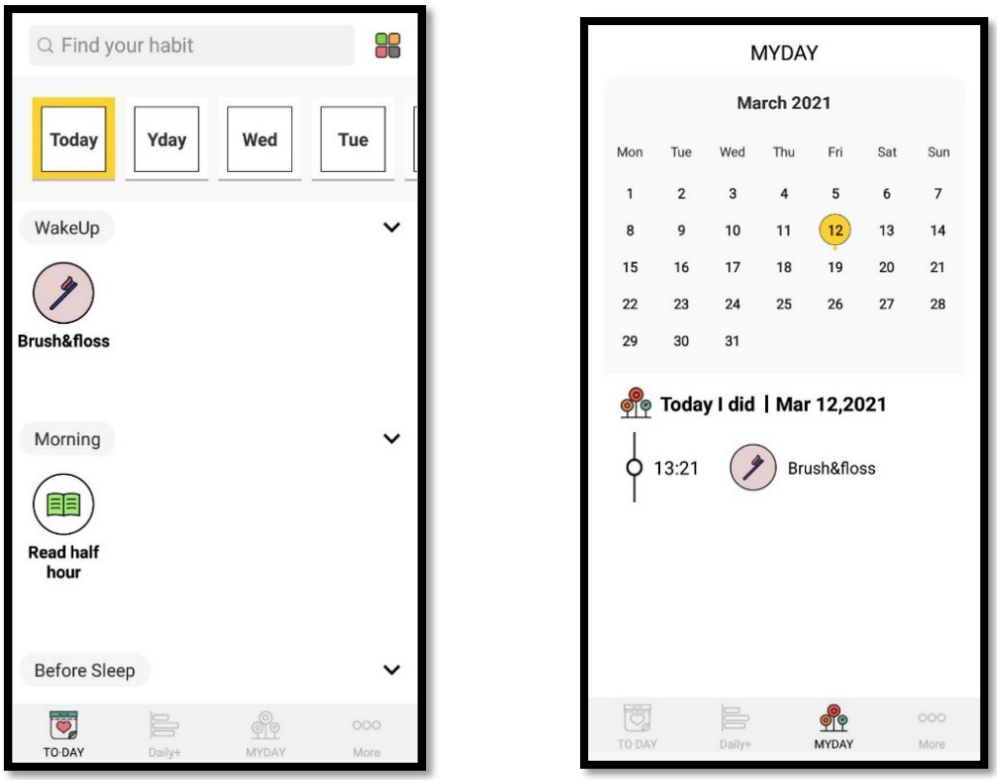


Рисунок 1.2.1.2. Інтерфейс застосунку Daily Planner

Застосунок і позиціонує себе як безкоштовний, проте кожні 5 хвилин з’являється екран з повідомленням про купівлю повної версії застосунку, що дуже заважає роботі з застосунком (див. рис. 1.2.1.3).

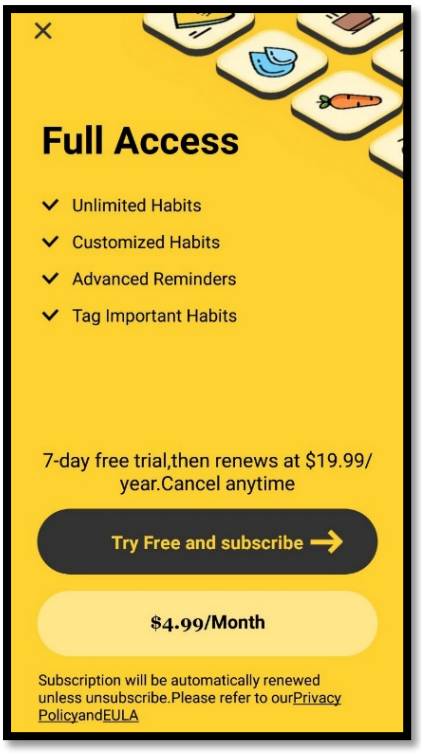


Рисунок 1.2.1.3. Реклама в застосунку Daily Planner

1.2.2 Мобільний застосунок «Any.do»

На торгівельному майданчику Play Market застосунок «Any.do» (див. рис. 1.2.2.1) [3] описано як планувальник списку завдань на день. Застосунок надає можливість спільного доступу до інформації, а також в описі зазначено режим фокусування. На 21.03.2021 застосунок завантажили більше десяти мільйонів разів. Він представлений на ринку більше трьох років, а його середня оцінка складає 4.4.

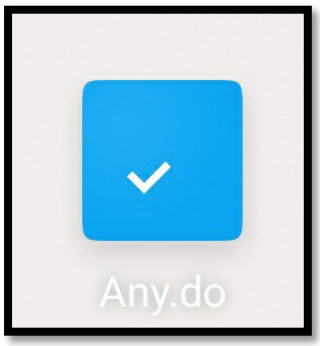


Рисунок 1.2.2.1. Логотип застосунку Any.do

Після користування застосунком було виявлено, що всі функції, що так приваблювали в цьому застосунку – режим фокусування, повторюванні завдання, кольорові позначки – доступні тільки у платній версії. В безкоштовній версії – це просто застосунок для запису своїх щоденних справ. Завдання можна ставити на день, 3 дні, тиждень, крім того, застосунок забезпечує синхронізацію з календарем, що покращує розуміння, які справи на який день заплановано (див. рис. 1.2.2.2).

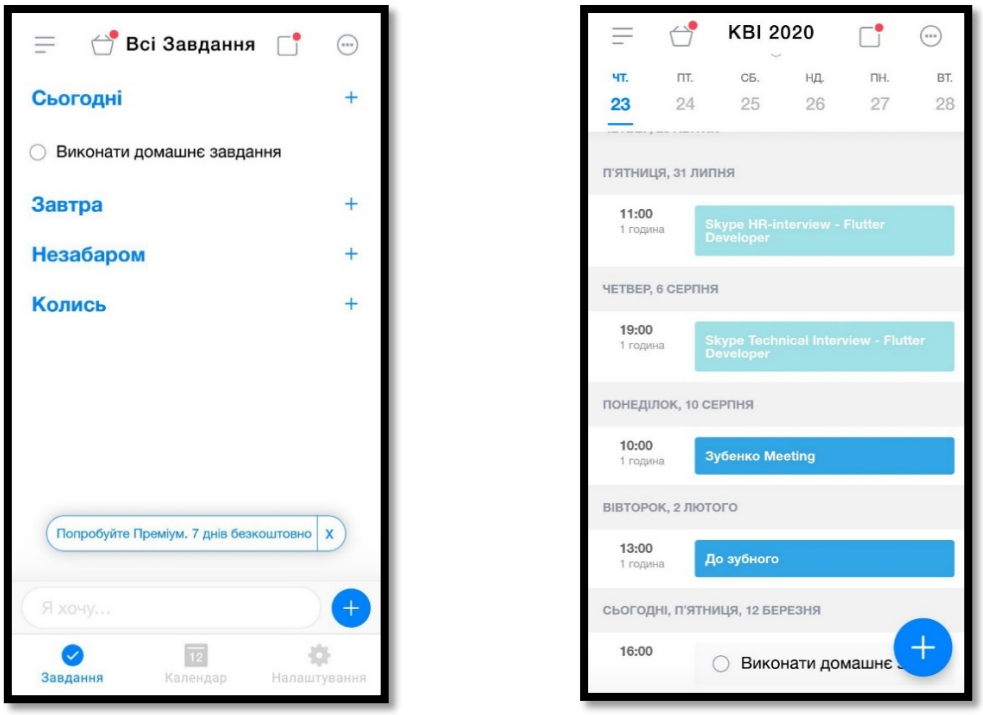


Рисунок 1.2.2.2. Інтерфейс застосунку Any.do

1.2.3 Мобільний застосунок «Habit360»

На торгівельному майданчику Play Market застосунок «Habit360» (див. рис. 1.2.3.1) [4] описано як трекер звичок та планувальник щоденної рутини. На 21.03.2021 р. застосунок завантажили більше десяти тисяч разів. Він представлений на ринку більше трьох років, а його середня оцінка складає 4.6.

Рисунок 1.2.3.1. Логотип застосунку Habit360

Застосунок «Habit360» досить зручний у користуванні. Проте він орієнтований лише на відстежування звичок – це єдина задача застосунку (див. рис. 1.2.3.2). Застосунок дозволяє додавати, видаляти звички, відслідковувати прогрес та надсилати нагадування про звички.

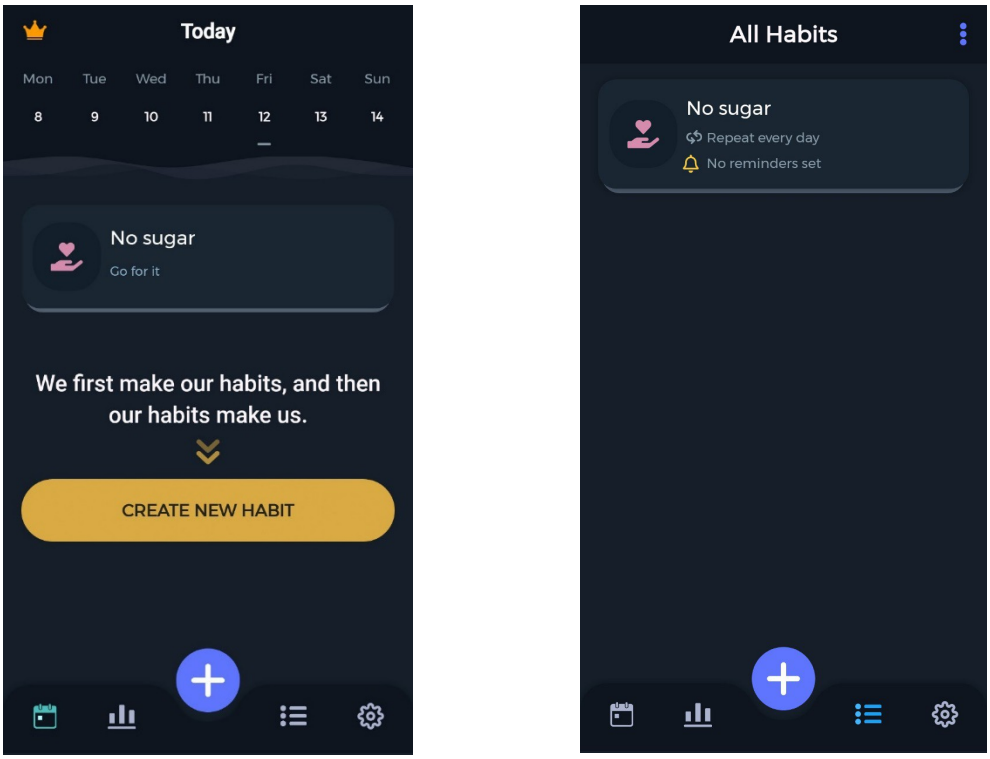


Рисунок 1.2.3.2. Інтерфейс застосунку Habit360

1.2.4 Мобільний застосунок «План тижня»

На торгівельному майданчику Play Market застосунок «План тижня» (див. рис. 1.2.4.1) [4] позиціонується як органайзер, щоденний планувальник. На 21.03.2021 р. застосунок завантажили більше одного мільйона разів. Він представлений на ринку більше трьох років, середня оцінка застосунку складає 4.1.

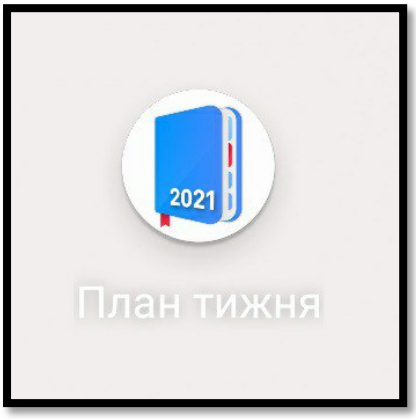


Рисунок 1.2.4.1. Логотип застосунку План тижня

Застосунок є аналогом блокноту, але в електронному вигляді. Як і у друкованому записнику, тут можна оформляти свої плани на день на умовній сторінці (див. рис. 1.2.4.2). В застосунку немає списків, окремих сторінок, усе виглядає як в блокноті, що є не дуже зручним у користуванні з телефону.

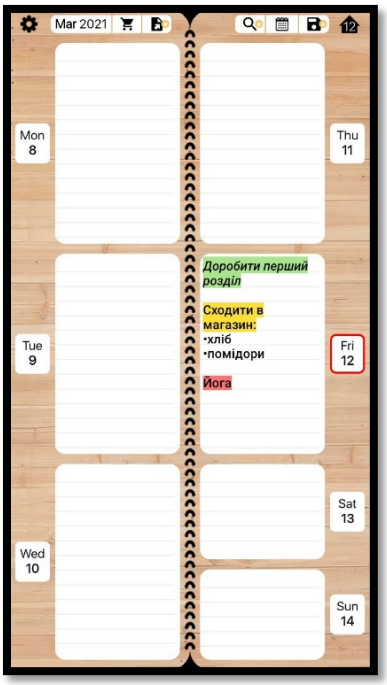


Рисунок 1.2.4.2. Інтерфейс застосунку План тижня

Порівняння вище розглянутих застосунків за найважливішими критеріями наведено в таблиці 1.2.1. В останньому рядку таблиці наведений застосунок, що є результатом кваліфікаційної роботи.

Таблиця 1.2.1. Порівняння застосунків

Назва застосунку	Безкоштовний	Необхідність надання персональної інформації	Зручність інтерфейсу	Функція списку справ	Функція списку покупок	Функція трекера звичок	Взаємозв’язок функцій між собою
Daily Planner	-	+	-	+	+	+	-
Any.do	-	+	+	+	+	-	+
Habit360	+	-	+	-	-	+	-
План тижня	+	-	-	+	+	+	-
PlannySmart	+	-	+	+	+	+	+

1.3 Постановка задачі

Після вивчення існуючих на ринку застосунків для планування дня, у них було виявлено такі недоліки:

- жоден з розглянутих застосунків не поєднує в собі усі компоненти особистих активностей, кожен застосунок виконує лише одну якусь функцію;
- більшість застосунків потребує платної преміум версії для повноцінного користування;
- деякі застосунки володіють зовсім незручним інтерфейсом;
- застосунки не продовжують роботу без надання персональних

даних.

З огляду на всі згадані вище чинники, до роботи були поставлені наступні вимоги:

- розробити мультизадачний застосунок, який буде поєднувати в собі трекер звичок, щоденний список справ, список покупок;
- застосунок повинен бути безкоштовним і забезпечувати користувачу усі наявні функції без додаткової плати;
- розробити зручний для користування інтерфейс;
- не запитувати персональні дані без нагальних на те причин.

Метою виконання кваліфікаційної роботи є створення єдиного мобільного застосунку для планування особистих активностей, який би поєднував усі необхідні компоненти та був зручний у користуванні.

Суть першого розділу полягала у зборі та аналізі наявної інформації про предметну область досліджуваного об’єкту, після чого, проведенні аналізу вже наявних рішень визначених проблем. Було здійснено огляд продуктів, які можуть задовольняти необхідним вимогами та виявлені їх недоліки. Результатом цього розділу є сформована постановка задачі з детальним списком вимог до отриманого застосунку.

РОЗДІЛ 2.
ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Застосунок, що розроблений в рамках кваліфікаційної роботи, призначений для забезпечення зручного та швидкого планування особистих активностей. Для реалізації застосунку була обрана мобільна операційна система Android [6], мова програмування Dart [7], фреймворк Flutter [8] та СКБД SQLite [9]. Детальніше про використанні технології описано у розділі нижче.

2.1 Мова програмування Dart

Dart – це оптимізована для клієнта мова програмування [10], яка призначена для розробки швидких застосунків на будь-якій платформі. Цю мову розробила компанія Google в жовтні 2011 року з метою запропонувати найпродуктивнішу мову програмування для мультиплатформної розробки у поєднанні з гнучкою платформою виконання для різноманітних фреймворків.

На відміну від багатьох мов, Dart був розроблений з метою зробити процес розробки максимально комфортним і швидким для розробників. Тож він постачається з досить широким набором вбудованих інструментів, таких як власний менеджер пакетів, різні компілятори / перекладачі, синтаксичний аналізатор та форматор. Також віртуальна машина Dart та збірка Just-in-Time роблять зміни коду негайно виконуваними.

Dart має великий набір основних бібліотек, які допомагають у виконанні багатьох повсякденних задач програмування:

- вбудовані типи, колекції та інші основні функції для кожної програми;

- такі типи колекцій як черги, зв’язані списки, хеш-карти, бінарні дерева;
- кодування та декодування для перетворення між різними представленнями даних, включаючи JSON;
- математичні константи і функції, а також генерація випадкових чисел;
- підтримка файлів, сокетів, HTTP та інших операцій введення-виведення;
- елементи HTML та інші ресурси для веб-застосунків, яким необхідно взаємодіяти з браузером і об’єктною моделлю документа (DOM).

Dart також формує основу Flutter. Dart надає мову і середовище виконання, яке використовується в застосунках Flutter, але Dart також підтримує багато основних задач розробника, такі як форматування, аналіз і тестування коду.

2.2 Фреймворк Flutter

Flutter – це безкоштовний і відкритий набір засобів розробки мобільного інтерфейсу для користувачів, створений компанією Google у травні 2017 року [11]. Flutter є основою Dart для створення крос-платформних додатків з одним кодом.

Flutter націлений на дві важливі речі:

- SDK: набір інструментів, який допомагає перетворити створений застосунок у нативний машинний код (код для iOS і Android).
- Фреймворк: колекція функціональних елементів користувальницького інтерфейсу (кнопки, текстові вводи, повзунки тощо), які можна персоналізувати під особисті

потреби.

Основні причини використовувати Flutter:

- Розробляти мобільний застосунок з Flutter дешевше, тому що не потрібно створювати і підтримувати два мобільних застосунки (один для iOS і Android).
- Можна легко застосовувати віджети, які надає Flutter і персоналізувати їх для створення оригінального користувальницького інтерфейсу.
- Не можливо помітити різницю між нативним застосунком і застосунком Flutter.
- Для створення продукту достатньо одного розробника.

2.3 База даних SQLite

SQLite – це вбудована кросплатформна система керування базами даних (СКБД), яка підтримує повний набір команд SQL [12]. Те, що вона вбудована означає, що база даних SQLite інтегрована з програмою, яка здійснює доступ до бази даних. Застосунки взаємодіють із базою даних SQLite, що читає та записує безпосередньо з файлів бази даних, що зберігаються на диску. В якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite.

SQLite є автономним засобом, що вимагає мінімальної підтримки з боку операційної системи або зовнішньої бібліотеки. Це робить SQLite придатним для використання в будь-якому середовищі, особливо на вбудованих пристроях, таких як iPhone, телефони Android, ігрові консолі, портативні медіаплеєри тощо.

Через безсерверну архітектуру не потрібно «встановлювати» SQLite перед його використанням. Немає жодного процесу сервера, який потрібно конфігурувати, запускати та зупиняти.

SQLite використовує динамічні типи для таблиць. Це означає, що можна зберігати будь-яке значення в будь-якому стовпці, незалежно від типу даних.

SQLite дозволяє єдиному підключенню до бази даних одночасно отримувати доступ до декількох файлів БД. Це приносить багато приємних функцій, таких як об’єднання таблиць у різних базах даних або копіювання даних за допомогою однієї команди.

РОЗДІЛ 3.
АРХІТЕКТУРА ЗАСТОСУНКУ ТА ПРИНЦИПИ ЙОГО РОБОТИ

На основі проведеного аналізу існуючих на ринку застосунків для планування активностей та виявлених у них переваг і недоліків у першому розділі, для покращення роботи власного застосунку була продумана концепція проекту та складена діаграма прецедентів застосунку (див. додаток А).

Застосунок складається з трьох великих частин: планувальник завдань, трекер звичок, список покупок. Ці компоненти взаємопов’язані між собою. Коли користувач додає продукт до списку покупок вперше за день, система запитує, чи потрібно додати до сьогоднішніх справ пункт «сходити за покупками» і, якщо відповідь позитивна, у списку справ автоматично з’являється такий пункт. Так само при створенні звички є можливість додати її в список завдань. Це означає, що задача «виконати назва_звички» буде з’являтися кожного дня в списку завдань на день, і при позначені її як виконаної, звичка буде автоматично позначатись зробленою за сьогоднішній день, і навпаки, якщо в розділі звичок позначити її зробленою, завдання в списку справ на сьогодні автоматично позначиться як виконане.

Також була розроблена система сповіщень і можливість обирати тему застосунку (світла чи темна), що описано в блоці налаштувань.

Усі компоненти проекту зображено в діаграмі класів у додатку Б.

3.1 Планувальник завдань

Компонент планувальник завдань було розроблено для ефективного

планування задач на день. Для зберігання даних в базі даних було створено дві таблиці – таблиця категорій завдань та таблиця самих завдань (див. рис. 3.1.1).

```
await database.execute("CREATE TABLE categories"
    "(id INTEGER PRIMARY KEY,"
    " name TEXT, description TEXT)");

await database.execute("CREATE TABLE todos"
    "(id INTEGER PRIMARY KEY, title TEXT,"
    " description TEXT, category TEXT,"
    " todoDate TEXT, isFinished INTEGER)");
```

Рисунок 3.1.1. Запити на створення таблиць в базі даних

Модель категорій знаходиться в класі Category і має такі поля, як id (первинний ідентифікатор), name (назва категорії), description (додаткова інформація про категорію). У таблиці 3.1.1 описано основні функції, що забезпечують взаємодію з базою даних. Усі функції зберігаються в класі CategoryService.

Таблиця 3.1.1. Основні функції взаємодії з базою даних для об’єкта категорії

Функція	**Її призначення**
saveCategory	забезпечує додавання в базу даних нової категорії
updateCategory	забезпечує оновлення категорії
readCategory	дозволяє отримати список усіх категорій з бази даних

readCategoryById	забезпечує отримання конкретної категорії за її ідентифікатором
deleteCategory	забезпечує видалення категорії з бази даних

Основний клас, в якому описуються всі форми взаємодії користувача з категоріями має назву CategoriesScreen, в ньому прописується дизайн сторінки категорій та увесь функціонал (див. додаток Б).

Модель завдань знаходиться в класі Todo і має такі поля, як id (первинний ідентифікатор), title (назва завдання), description (додаткова інформація по завданню), category (категорія, до якої відноситься завдання; може бути без категорії), todoDate (дата, коли потрібно виконати завдання), isFinished (булеве значення, яке зберігає 1, якщо завдання виконане і 0 в іншому випадку). У таблиці 3.1.2 описано функції, що забезпечують взаємодію з базою даних. Усі функції зберігаються в класі TodoService.

Таблиця 3.1.2. Основні функції взаємодії з базою даних для об’єкта завдань

Функція	**Її призначення**
saveTodo	додає нове завдання в базу даних
readTodos	повертає список усіх завдань з бази даних
readTododsByCategory	повертає список завдань конкретної категорії
readTodoById	повертає конкретне завдання за його ідентифікатором
updateTodo	оновлює завдання
deleteTodo	видаляє завдання з бази даних

Модель завдань має три сторінки взаємодії з користувачем – сьогоднішні завдання, завдання за категоріями, сторінка створення завдання. Розберемо кожну з них детальніше.

Сторінка сьогоднішніх завдань запрограмована показувати користувачу лише ті завдання, в полі todoDate яких зазначена сьогоднішня дата. На цій сторінці користувач може редагувати завдання, видаляти, помічати його як виконане, а також перейти на сторінку створення нового завдання. Додатково було розроблено можливість перенести завдання на сьогодні, якщо воно не було виконане вчора. Якщо користувач вирішує залишити завдання, виконується запит в базу даних на оновлення завдання, де оновлюється тільки дата виконання, а решта залишається без змін (див. рис. 3.1.2).

```
onPressed: ()async{
  _todo.id = task[0]['id'];
  _todo.title = task[0]['title'];
  _todo.todoDate = DateFormat('dd/MM/yyyy').format(todayDate);
  _todo.category = task[0]['category'];
  _todo.isFinished = task[0]['isFinished'];
  var result = await _todoService.updateTodo(_todo);
  if(result>0){
    Navigator.pop(context);
    getAllTodos();
  }
},
```

Рисунок 3.1.2. Функція оновлення завдання при перенесені його на сьогоднішній день

3.2 Трекер звичок

Компонент трекер звичок розроблено для підвищення ефективності користувача. Дані для цього компоненту зберігаються в базі даних у двох таблицях – таблиці звичок та таблиці подій, що можуть відбуватися зі звичками (виконання, пропуск, невиконання) (див. рис. 3.2.1).

```
await database.execute("CREATE TABLE habits"
    "(id INTEGER PRIMARY KEY AUTOINCREMENT,"
    " position INTEGER, title TEXT,"
    " twoDayRule INTEGER, cue TEXT, "
    "routine TEXT, reward TEXT,"
    " showReward INTEGER, advanced INTEGER");

await database.execute("CREATE TABLE events"
    "(id INTEGER, dateTime TEXT,"
    " dayType INTEGER,"
    " PRIMARY KEY(id, dateTime))");
```

Рисунок 3.2.1. Запити на створення таблиць в базі даних

Клас моделі звичок має назву Habit і складається з наступних полів:

- id – первинний ідентифікатор;
- position – позиція звички у списку (надає можливість переміщати звичку на екрані на будь-яку комфортну для вас позицію);
- title – назва звички;
- twoDayRule – булеве значення, яке позначає, чи користувач використовує правило 2 днів (це надає можливість не втратити результат, пропустивши один день виконання звички);
- makeTask – булеве значення, яке надає можливість додати виконання звички у щоденний список справ;
- cue – час, коли буде виконуватися звичка;
- routine – що саме потрібно повторювати;
- reward – винагорода за виконання рутини;
- showReward – булеве значення, яке позначає, чи показувати винагороду після виконання рутини;
- advanced – булеве значення, яке позначає, що використовуються розширенні налаштування опису

звичок, такі як час, рутина, винагорода.

У таблиці 3.2.1 наведено класи, що відповідають за роботу блоку трекер звичок, а також описано основні їх функції.

Таблиця 3.2.1. Опис класів блоку трекер звичок

Клас	**Опис**
HabitModel	описує функції взаємодії з базою даних, такі як додавання в базу даних, оновлення даних, видалення, читання з бази
HabitScreen	описує вигляд основної сторінки трекеру звичок, а саме вигляд верхньої панелі та кнопку додавання нової звички
CalendarColumn	описує вигляд основної сторінки, а саме відображення назв днів тижня, самих звичок і панелі з числами місяця на кожен день тижня
Habit	описує модель звичок, а також функції, що дозволяють змінювати стан звички (виконана, невиконана, пропущена)
CreateHabitPage	описує функціонал додавання нової звички
EditHabitPage	описує функціонал оновлення звички

3.3 Список покупок

Список покупок це невеликий компонент системи, який являє собою лише список продуктів, які необхідно купити. Тому було б не раціонально використовувати базу даних для зберігання усіх даних і навантажувати систему запитами в базу, цим самим знижуючи працездатність застосунку.

Через це було прийнято рішення використовувати для зберігання даних про список покупок бібліотеку SharedPreferences [13], яка дозволяє зберігати дані в постійному сховищі Android.

Було створено два списки – список усіх продуктів і список викреслених продуктів (див. рис. 3.3.1).

```
void _saveData() async {
  final prefs = await SharedPreferences.getInstance();
  prefs.setStringList("items", _items);
  prefs.setStringList("completedItems", _completedItems);
}
```

Рисунок 3.3.1. Приклад зберігання даних в SharedPreferences

Усі маніпуляції з даними відбуваються як зі звичайним списком. В таблиці 3.3.1 наведено функції та віджети, які використовуються в блоці список покупок.

Таблиця 3.3.1. Функції та віджети, що використовуються в блоці список покупок

Назва	**Опис**
loadData	завантажує з сховища SharedPreferences список усіх продуктів і список викреслених продуктів;
saveData	зберігає в сховище SharedPreferences список усіх продуктів і список викреслених продуктів;
addItem	отримує значення, що ввів користувач і зберігає його в список усіх продуктів;
completeItem	додає елемент в список викреслених продуктів і видаляє зі списку усіх продуктів;
uncompleteItem	додає елемент в список усіх продуктів і видаляє

	зі списку викреслених продуктів;
clearCompleted	очищає список викреслених продуктів;
buildList	віджет, що відображає усі списки;
buildListItem	віджет, що описує дизайн елемента списку усіх продуктів і викликає функцію completeItem;
buildCompletedListItem	віджет, що описує дизайн елемента списку викреслених продуктів і викликає функцію uncompleteItem;
addTodoFormDialog	описує процес додавання до списку сьогоднішніх справ завдання «сходити за покупками»;
build	віджет, який описує дизайн усієї сторінки компоненту список покупок і викликає необхідні функції.

Також у цій компоненті було реалізовано список підказок при вводі продукту. Він будується на уже введених користувачем продуктів і підказки починають видаватись при вводі уже першої літери продукту. Цей список можна очистити, натиснувши на відповідну кнопку інтерфейса.

3.4 Блок налаштувань

Блок налаштувань складається з теми та сповіщень. У застосунку було реалізовано дві теми – темна та світла. Для цього створювався окремий файл, де за допомогою класу ThemeData [14] було прописано усі кольори, шрифти, розміри тексту для обох тем (див. рис. 3.4.1).

```
ThemeData lightTheme = ThemeData(
    brightness: Brightness.light,
    backgroundColor: Colors.white,
    buttonColor: Colors.grey[700],
    textTheme: TextTheme(
        bodyText1: TextStyle(color: Color(0xFF4A4A4A)),
        bodyText2: TextStyle(color: Colors.black),
        headline1: TextStyle(color: Colors.black, fontSize: 18,
          fontWeight: FontWeight.bold,),
        headline2: TextStyle(
          color: Color(0xffa29aac),
          fontSize: 14,
          fontWeight: FontWeight.w600,
        ),
```

Рисунок 3.4.1. Стилі застосунку

Реалізовано щоденні сповіщення застосунку (див. рис. 3.4.2). В налаштуваннях їх можна ввімкнути або вимкнути. Також можна обрати час, коли ці сповіщення будуть надходити.

```
set setDailyNot(TimeOfDay value) {
  settingsData.setDailyNot = value;
  _prefs.then((SharedPreferences prefs) {
    var st = settingsData.toJson().toString();
    prefs.remove('planny_settings');
    prefs.setString('planny_settings', st);
  });
  _notificationCenter.setNotification(settingsData.getDailyNot);
  notifyListeners();
}
```

```
void setNotification(TimeOfDay time) async {
  await _dailyNotification(
      0, time, 'PlannySmart', 'Твої справи чекають!');
}
```

Рисунок 3.4.2. Функція встановлення сповіщень

РОЗДІЛ 4.
ІНСТРУКЦІЯ КОРИСТУВАЧА

Після завантаження застосунку відкривається головна сторінка (див. рис. 4.1), звідки можна перейти до сторінки завдань, сторінки звичок, списку покупок і налаштувань застосунку.

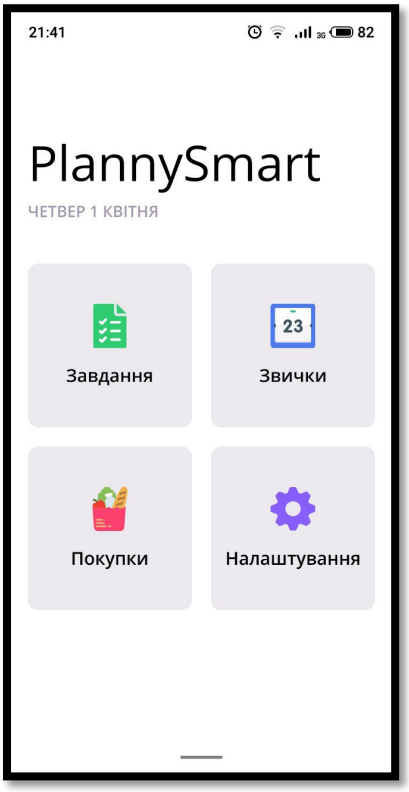


Рисунок 4.1. Головна сторінка

Натиснувши на блок «Завдання», перейдемо на сторінку завдань на сьогодні (див. рис. 4.2). Для того, щоб побачити усі сторінки блоку завдань, потрібно потягнути бокову панель або ж натиснути на меню в лівому верхньому кутку. Після цих маніпуляцій відкривається допоміжна бокова панель з навігацією (див. рис. 4.3).

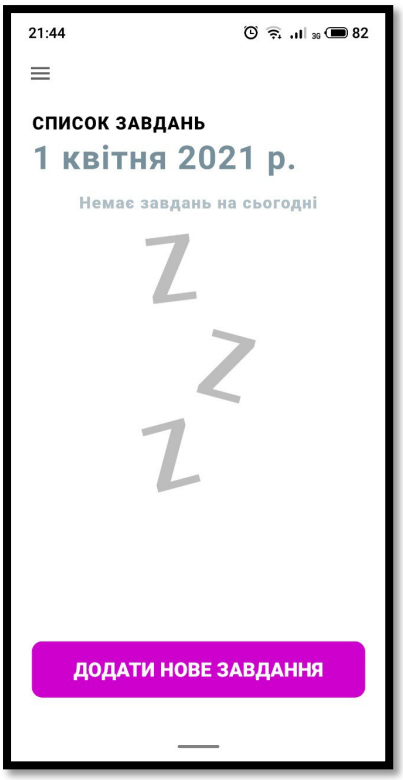


Рисунок 4.2 Сторінка сьогоднішніх завдань

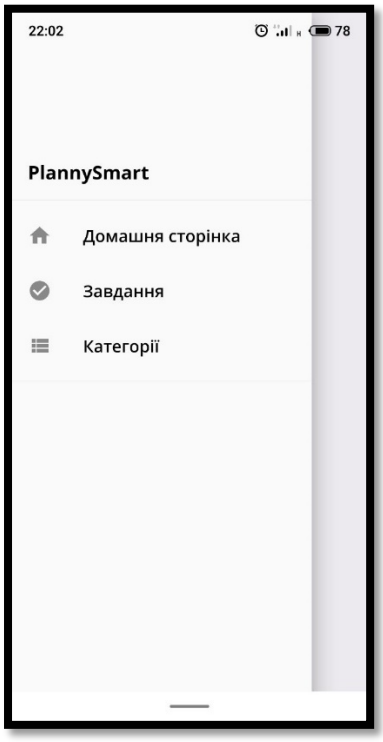


Рисунок 4.3 Бокова панель навігації

Звідси можна повернутись назад на головну сторінку, відкрити

сторінку категорій і завдань, а також після створення категорії, вона з’являється в боковій панелі навігації і, натиснувши на неї, можна перейти на сторінку завдань за конкретною категорією.

Для того, щоб відкрити сторінку категорій, натискаємо кнопку «Категорії» на боковій панелі навігації (див. рис. 4.4).

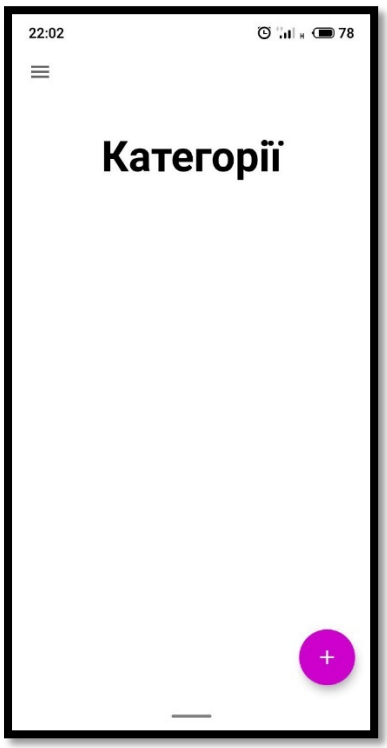


Рисунок 4.4. Сторінка категорій

Для того, щоб створити категорію, потрібно натиснути на плюс, після чого ввести назву категорії та натиснути кнопку «Зберегти» (див. рис. 4.5).

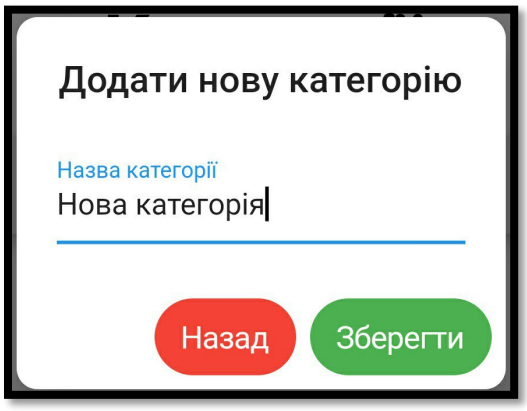


Рисунок 4.5. Створення категорії

Після цього нова категорія з’являється в списку категорій і в боковій панелі навігації (див. рис. 4.6). Тепер можна редагувати та видаляти категорію. Для цього просто необхідно натиснути на відповідні значки, після чого спливають відповідні діалоги. При редагуванні потрібно ввести нову назву і натиснути «Оновити» (див. рис. 4.7 а). Якщо ви хочете видалити категорію, потрібно натиснути кнопку «Видалити», інакше «Назад» (див. рис. 4.7 б).

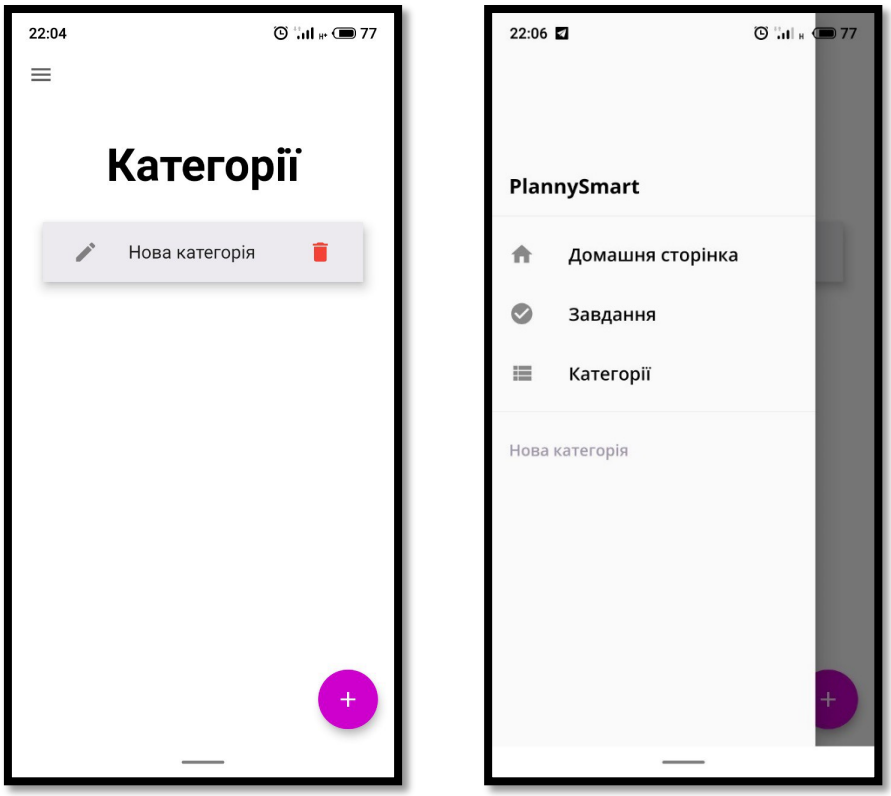
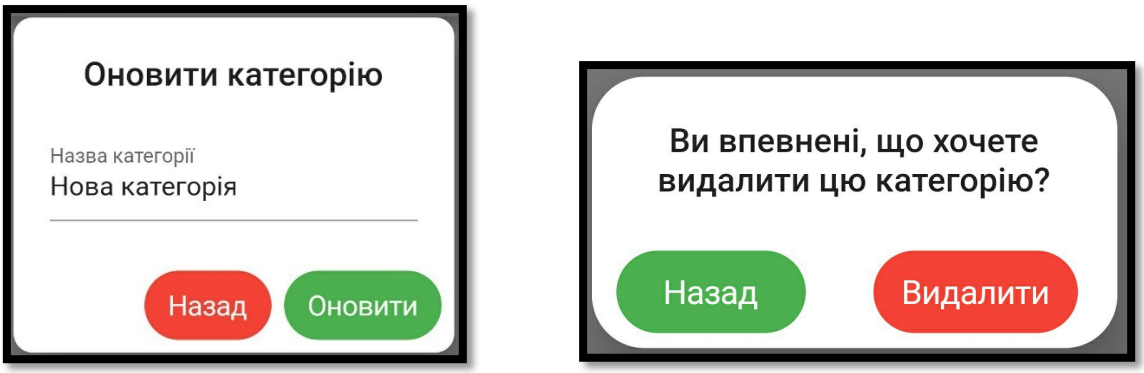


Рисунок 4.6. Нова категорія



а) б)

Рисунок 4.7. а) Редагування категорії; б) Видалення категорії

Повернемося на сторінку сьогоднішніх завдань і додамо перше на сьогодні завдання. Для цього натискаємо на кнопку «ДОДАТИ НОВЕ ЗАВДАННЯ», після чого відкривається сторінка створення завдання (див. рис. 4.8).

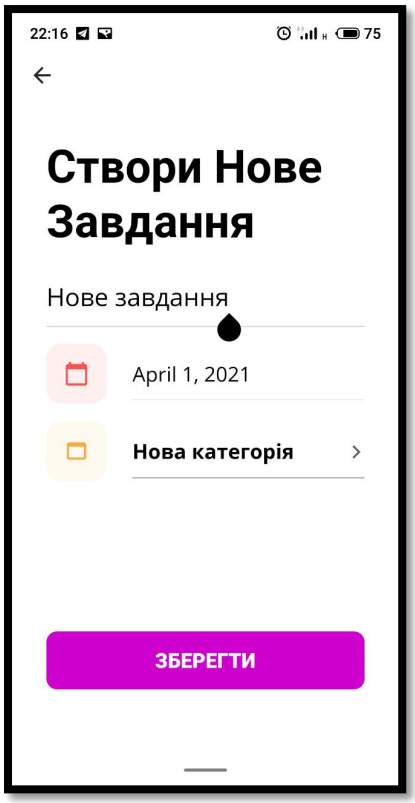


Рисунок 4.8. Сторінка створення завдання

Потрібно ввести назву завдання, обрати дату, коли потрібно його виконати та обрати категорію, до якої віднести це завдання. У прикладі обирається сьогоднішня на той момент дата – 01.04.2021, та щойно створена категорія. Поле категорій може залишатись порожнім, тоді завдання не буде відноситись до жодної з категорій. Після натиснення на кнопку «ЗБЕРЕГТИ», завдання додається до списку (див. рис. 4.9).

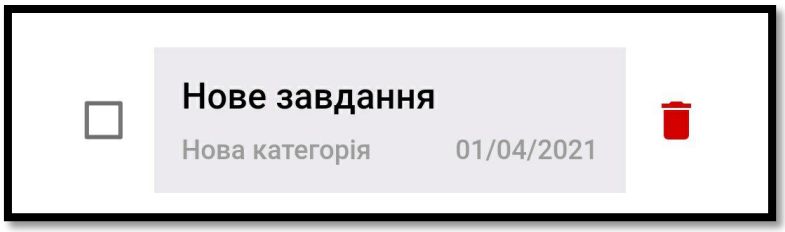


Рисунок 4.9. Нове завдання

Для того, щоб відредагувати завдання, потрібно натиснути на саме завдання. Тоді з’явиться спливаюче вікно (див. рис. 4.10), де можна змінити назву завдання, дату, категорію. Після внесених змін потрібно натиснути кнопку «Оновити».

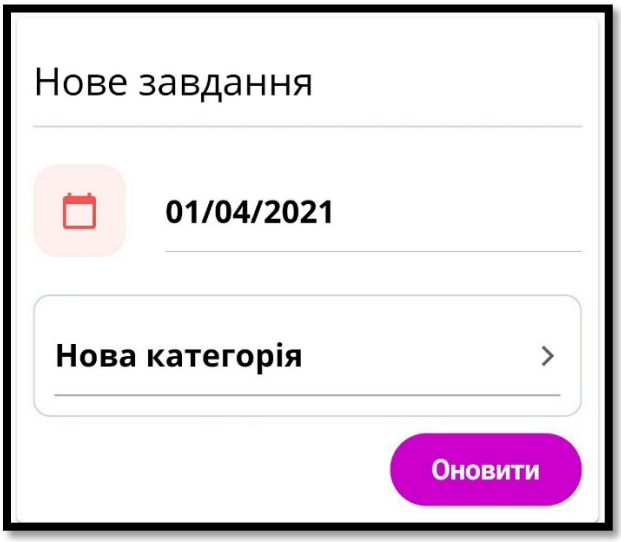


Рисунок 4.10. Редагування завдання

Для того, щоб видалити завдання, необхідно натиснути на значок сміттєвого баку, після чого з’явиться діалогове вікно (див. рис. 4.11). Натисніть «Видалити» для видалення, або «Назад», щоб повернутися до списку завдань.

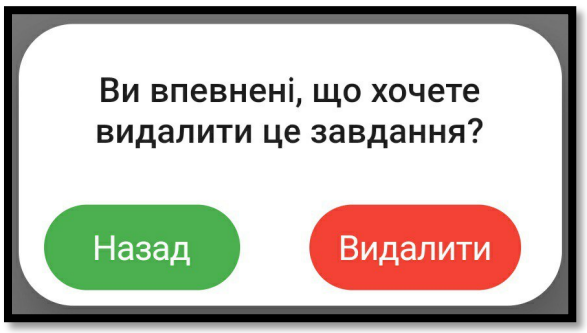


Рисунок 4.11. Видалення завдання

Для того, щоб відмітити завдання виконаним, потрібно поставити біля нього галочку (див. рис. 4.12).

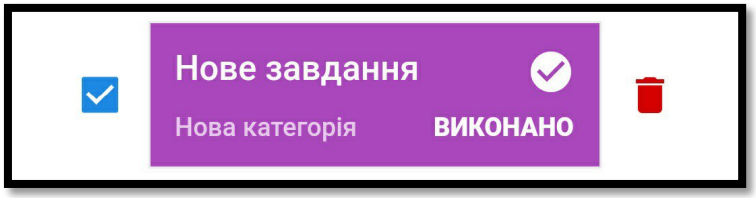


Рисунок 4.12. Виконане завдання

У випадку, якщо ви не виконали завдання учора і при цьому його не видалили, завдання переноситься на сьогодні, якщо ви дали на це згоду, інакше видаляється (див. рис. 4.13).

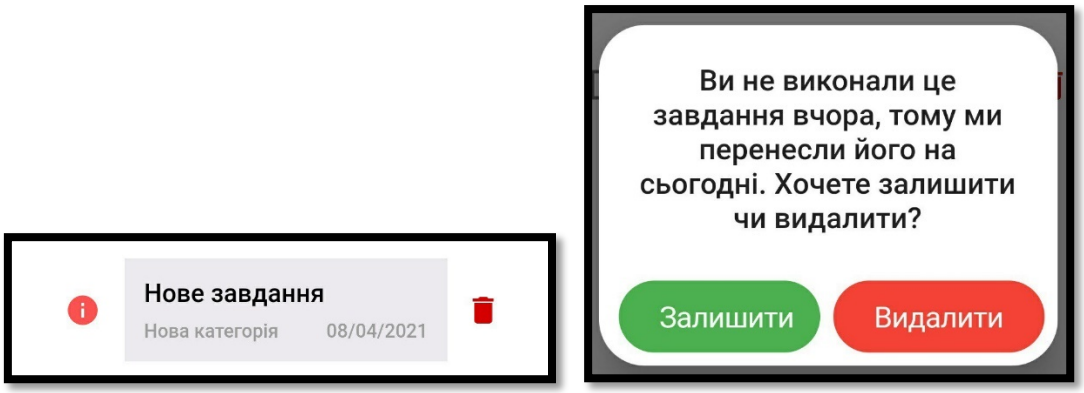


Рисунок 4.13. Перенесення завдання на сьогодні

Якщо відкрити сторінку «Нова категорія», побачимо список завдань для цієї категорії (див. рис. 4.14). Тут можна виконувати усі ті самі маніпуляції, що і зі списком сьогоднішніх справ.

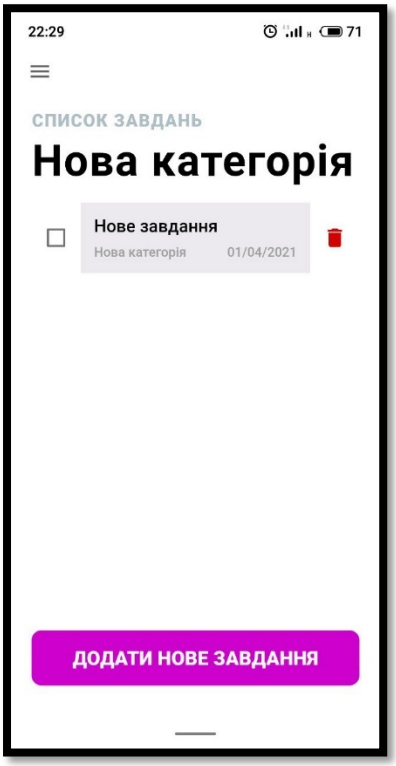


Рисунок 4.14. Сторінка «Нова категорія»

Повернемося на домашню сторінку і відкриємо блок «Покупки» (див. рис. 4.15). Для того, щоб додати товар до списку просто введіть його назву

і натисніть на плюс, після чого товар з’явиться на екрані. Для того, щоб викреслити товар зі списку, потрібно на нього натиснути, тоді він стане сірим і закресленим. Якщо у користувача виникне необхідність повернути цей товар, просто натисніть на нього знову. Для того, щоб очистити список від викреслених продуктів натисніть на іконку сміттєвого баку.

Рисунок 4.15. Список покупок

При введені користувачем товарів, їх назви зберігаються в список підказок, які показуються при вводі (див. рис. 4.16). Список підказок можна очистити, натиснувши на три крапки в правому верхньому кутку сторінки і вибравши «Очистити список підказок».

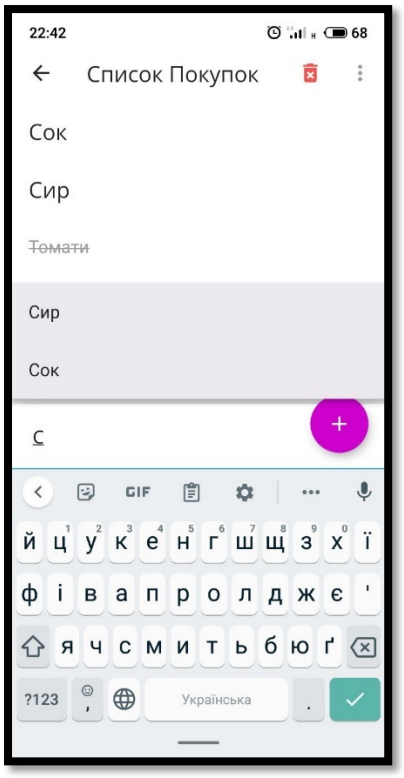


Рисунок 4.16. Підказки при вводі

Під час того, як створюється новий список покупок, з’являється діалогове вікно, яке пропонує додати завдання «Сходити за покупками» у список сьогоднішніх справ (див. рис. 4.17). Якщо натиснути «Так», завдання з’явиться у списку справ з сьогоднішньою датою, інакше нічого не відбувається.

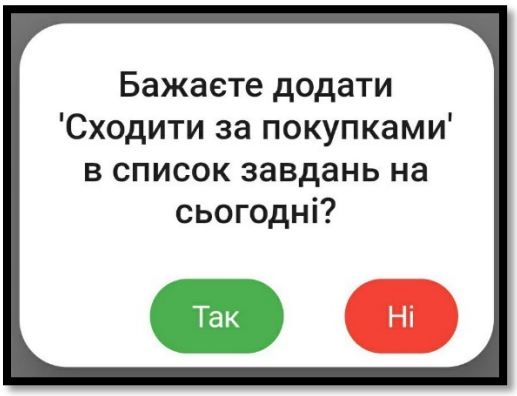


Рисунок 4.17. Створення завдання «Сходити за покупками»

Повернемося на домашню сторінку і відкриємо блок «Звички» (див. рис. 4.18). На цій сторінці відображається список звичок з віконцями на кожен день тижня у поточному місяці.

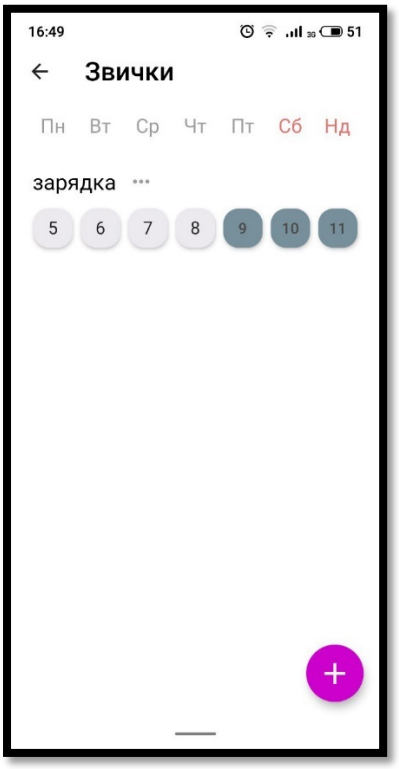


Рисунок 4.18. Сторінка «Звички»

Натиснувши на плюс відкривається сторінка створення нової звички (див. рис. 4.19). На сторінці обов’язково потрібно ввести назву звички. Є можливість використати правило двох днів – з цим правилом можна пропустити один день виконання звички і не втратити прогрес. Можна додати звичку в список щоденних завдань. Ця функція додає нове завдання «назва звички» кожного дня у ваш список справ і, якщо завдання помічається виконаним, звичка автоматично позначається виконаною сьогодні і в списку звичок, і навпаки. Також є можливість додати розширені налаштування для звички – час виконання, рутина, винагорода за виконання.

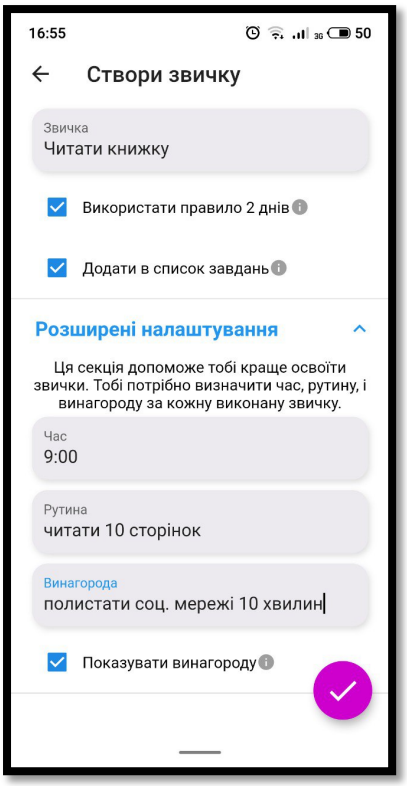


Рисунок 4.19. Створення звички

Для того, щоб відредагувати звичку чи видалити її, потрібно натиснути на три крапочки біля назви звички. Тоді відкриється сторінка редагування (див. рис. 4.20), де можна редагувати абсолютно усі поля, що відображались при створенні звички, а також можна видалити звичку, натиснувши на червоний сміттєвий бак в верхньому правому кутку.

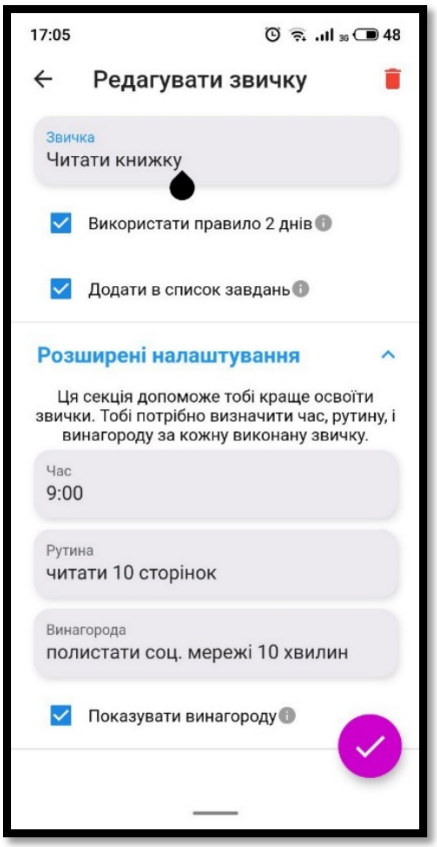


Рисунок 4.20. Редагування звички

Кожного дня звичку можна позначати виконаною, невиконаною або пропущеною (див. рис. 4.21).

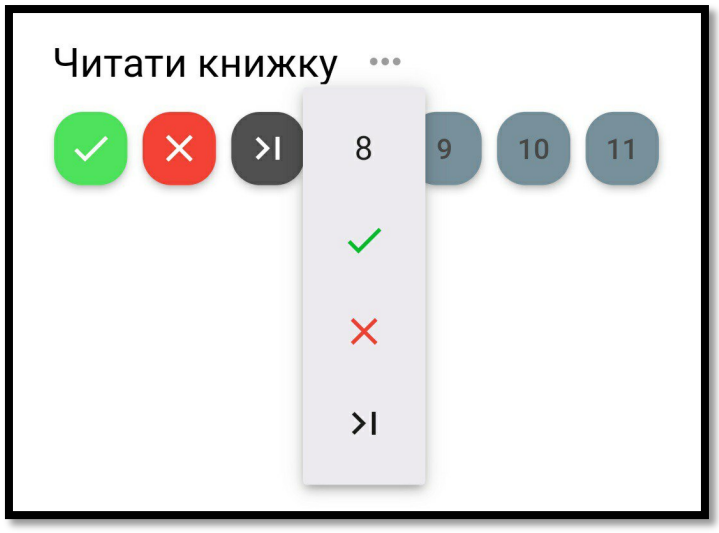


Рисунок 4.21. Стани звички

Тепер розглянемо блок «Налаштування» (див. рис. 4.22). Тут можна обрати тему застосунку та увімкнути чи вимкнути сповіщення, а також обрати о котрій годині буде надходити сповіщення.

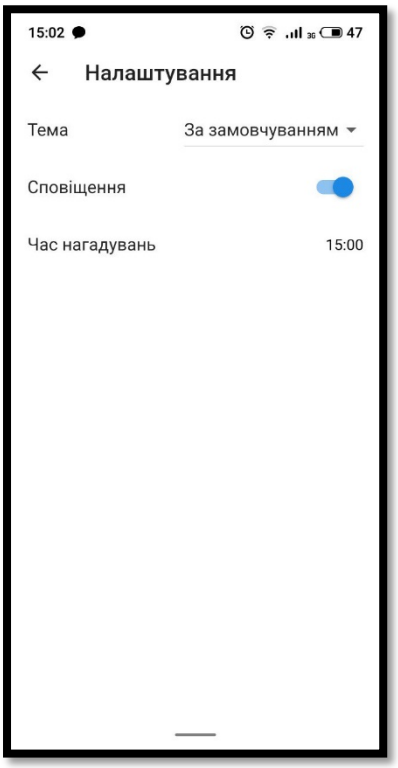
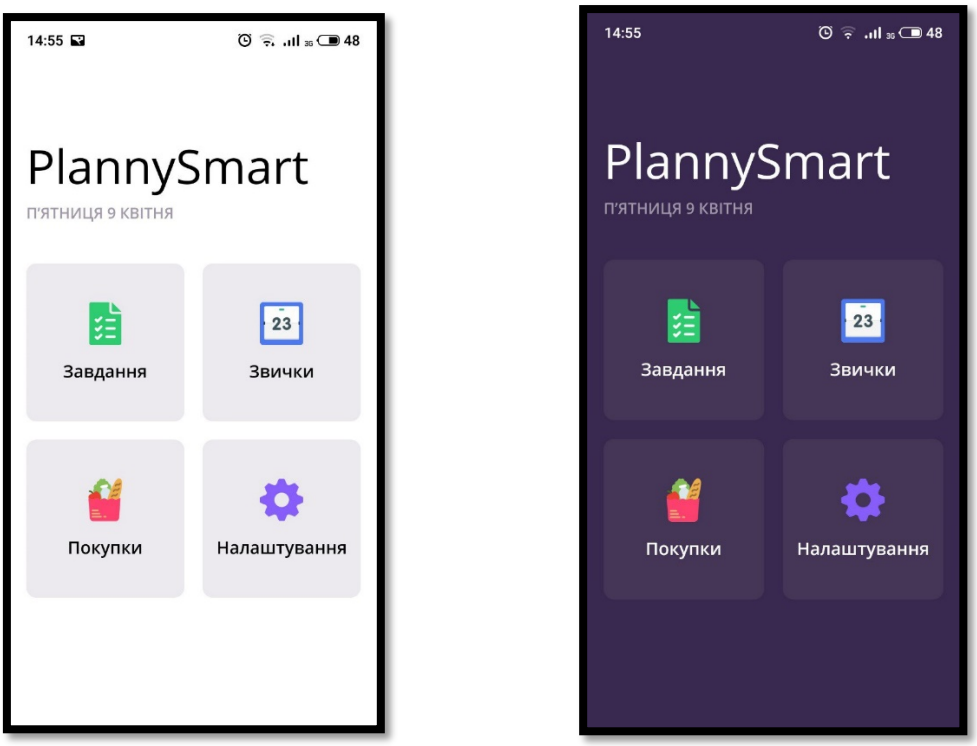


Рисунок 4.22. Сторінка налаштувань

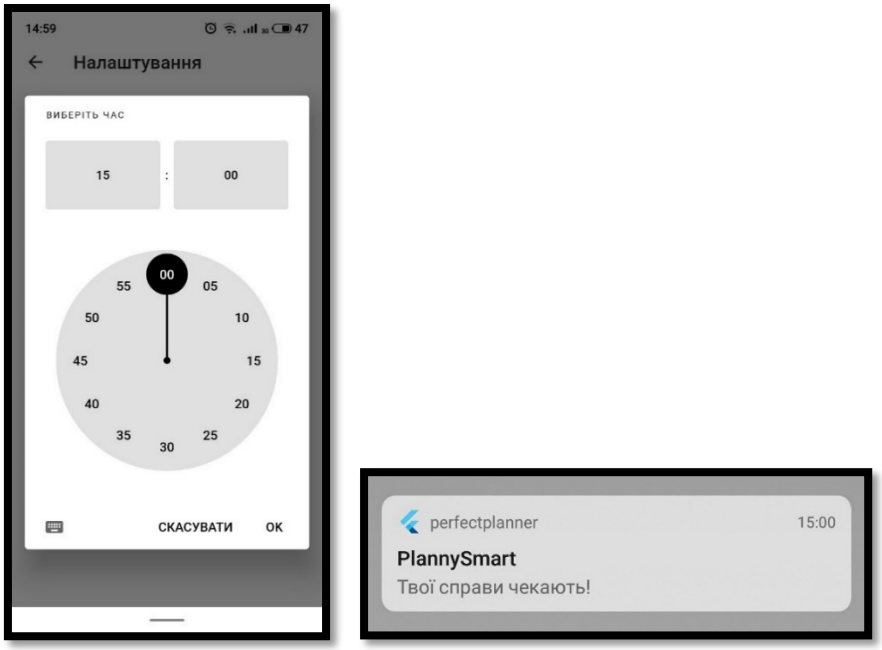
В застосунку є дві теми – темна та світла. Тема за замовчуванням – світла. На прикладі головної сторінки застосунку бачимо різницю (див. рис. 4.23 а, б).



а) б)

Рисунок 4.23. а) Світла тема; б) Темна тема

Сповіщення можна налаштовувати під себе – на зручний для вас час (див. рис. 4.24 а, б).



а) б)

Рисунок 4.24 а. Встановлення сповіщення, б. Вигляд сповіщення

ВИСНОВКИ

В ході виконання кваліфікаційної роботи розглянуто питання, пов’язані з необхідністю розробки застосунку для планування особистих активностей, а також описано процес розробки вищезгаданого застосунку.

Після аналізу предметної області даної системи та її компонентів визначено постановку задачі та основні вимоги, яких необхідно дотриматись в процесі реалізації.

Протягом виконання роботи було досліджено та закріплено знання за необхідними для роботи технологіями, а також покращено уміння в роботі з Dart та Flutter.

В процесі виконання кваліфікаційної роботи автором було досліджено інформацію про існуючі застосунки планування, розроблено технічне завдання до продукту та реалізовано мультизадачний застосунок, який поєднує в собі трекер звичок, щоденний список справ та список покупок. Було розроблено зручний та зрозумілий інтерфейс користування застосунком. Також користувачу надається можливість обрати більш комфортну для себе тему.

Після здійснення поставлених задач було досягнуто мети виконання кваліфікаційної роботи – розроблено застосунок для планування особистих активностей.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

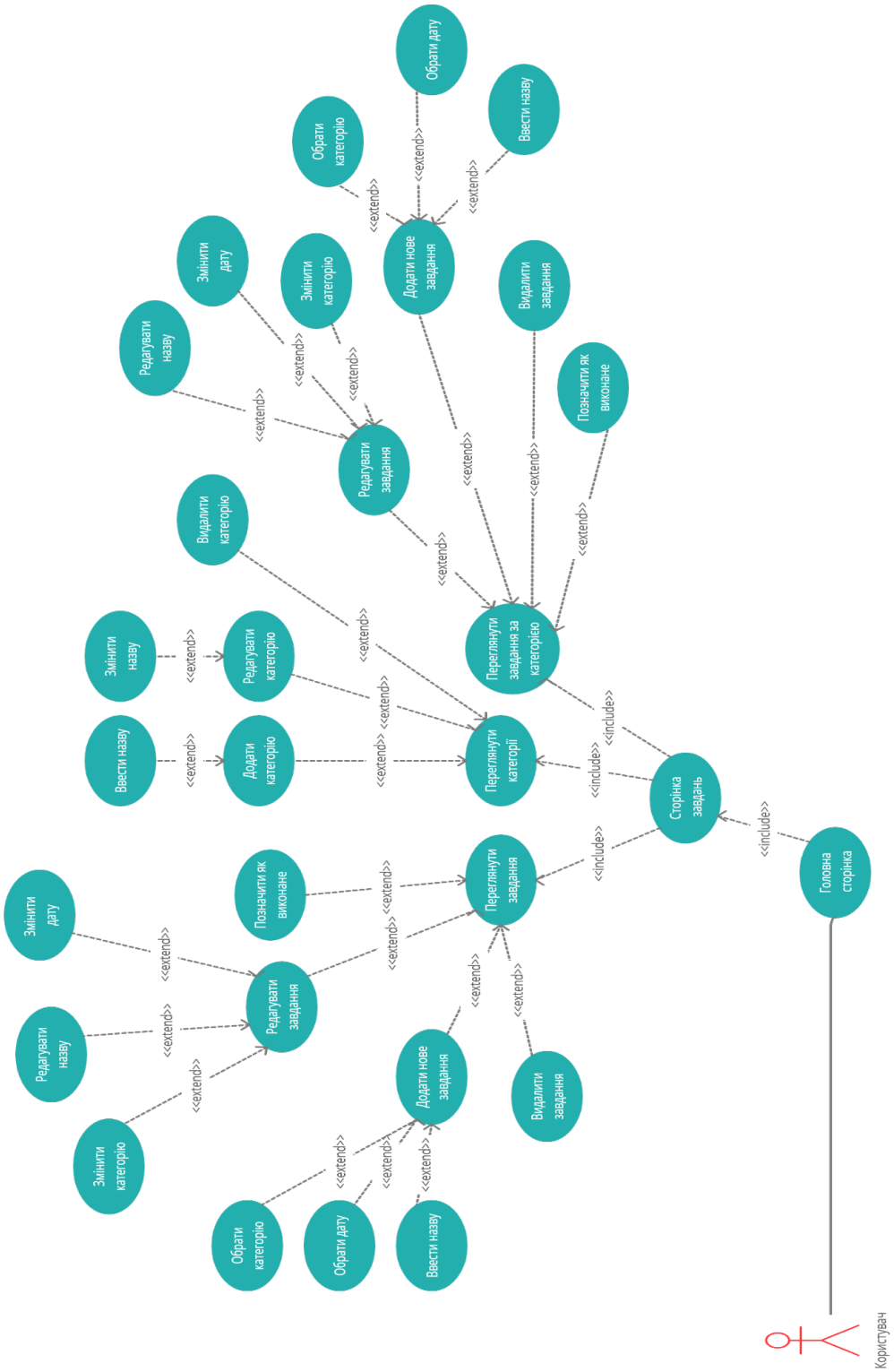
1. Дослідження британських вчених [Електронний ресурс] – Режим доступу до ресурсу: https://www.independent.co.uk/life-style/fashion/news/screen-time-average-lifetime-years-phone-laptop-tv-a9508751.html?fbclid=IwAR3V2OhSiNzA_GP_xUer8yET7t0zotpe9g6ICOt4qUPOWOwpDwecelYhXbM.
2. Застосунок Daily Planner [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.justorderly.dailyplanner.
3. Застосунок Any.do [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.anydo.
4. Застосунок Habit360 [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.maapps.habittracker.
5. Застосунок План тижня [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.weeklyplannerapp.weekplan.
6. Савонік О. М. Порівняльний аналіз мобільних операційних систем для розробників-початківців / О. М. Савонік, К. О. Скоробагатько. // Університет Григорія Сковороди в Переяславі. – 2021. – №36. – С. 122–124.
7. David K. Dart for Absolute Beginners / Kopec David., 2014.
8. Rap Payne. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps / Rap Payne., 2019.
9. Grant Allen. The Definitive Guide to SQLite / Grant Allen, Mike Owens., 2006.

10. Ivo Balbaert. Dart Cookbook / Ivo Balbaert., 2014.
11. Rap Payne. Beginning App Development with Flutter: Create Cross-Platform Mobile Apps / Rap Payne., 2019.
12. Санни Кумар Адитья. Android SQLite Essentials / Санни Кумар Адитья, Викаш Кумар Карн., 2014.
13. SharedPreferences [Електронний ресурс] – Режим доступу до ресурсу: https://www.fandroid.info/sharedpreferences-sohranenie-dannyh-v-postoyannoe-hranilishhe-android/.
14. ThemeData [Електронний ресурс] – Режим доступу до ресурсу: https://api.flutter.dev/flutter/material/ThemeData-class.html.

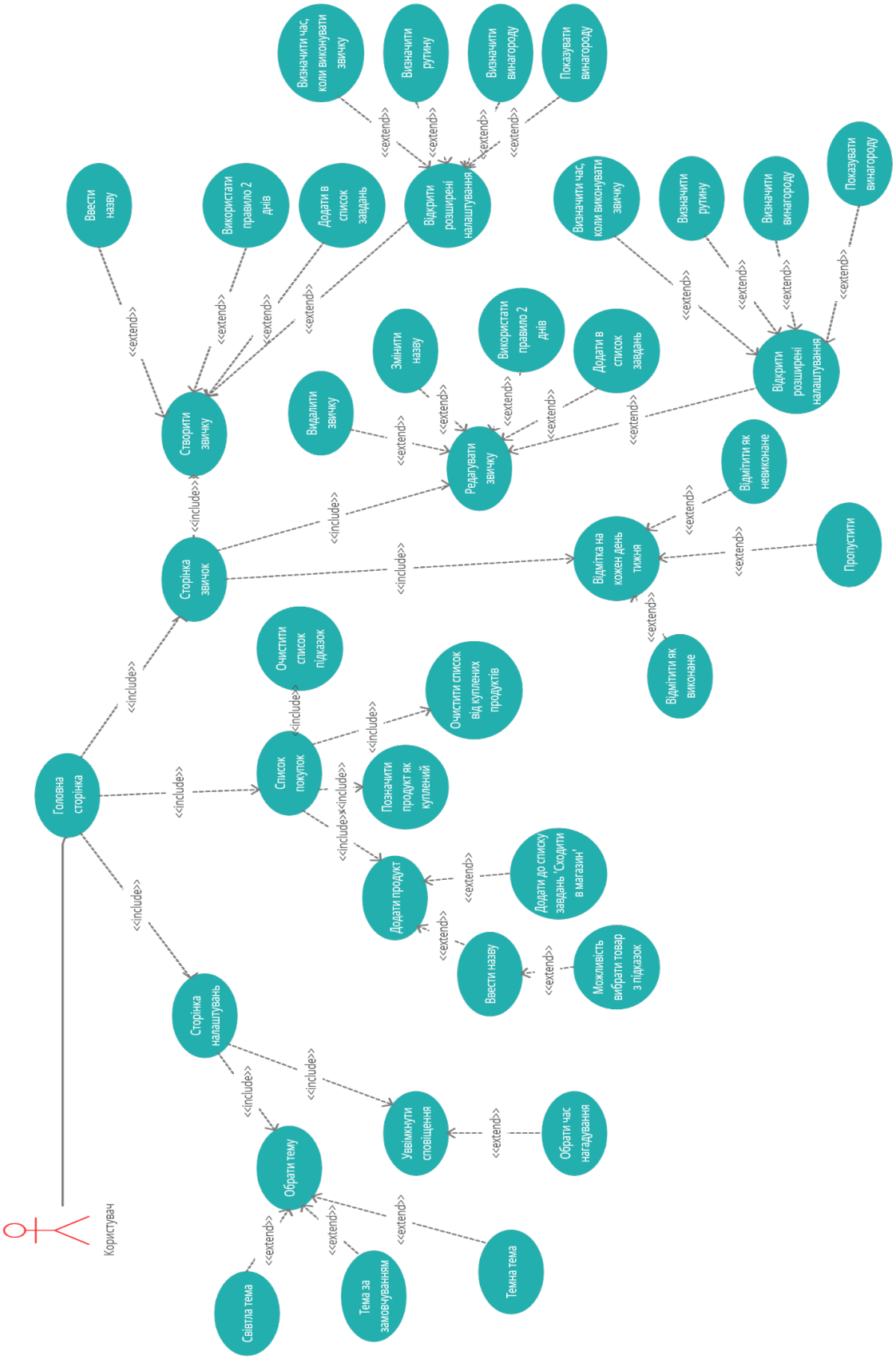
ДОДАТКИ

Додаток А. Діаграма прецедентів

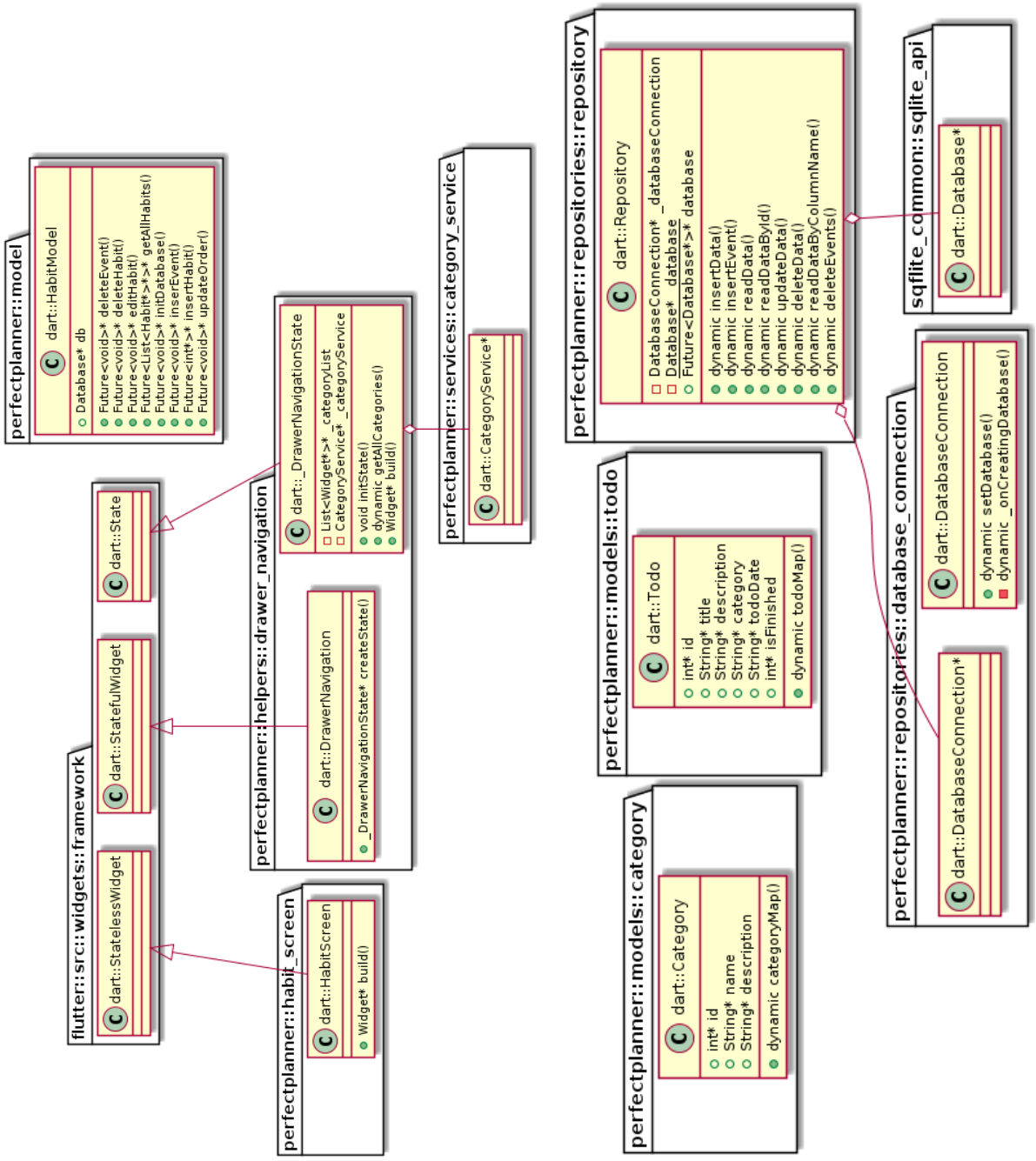
Фрагмент 1

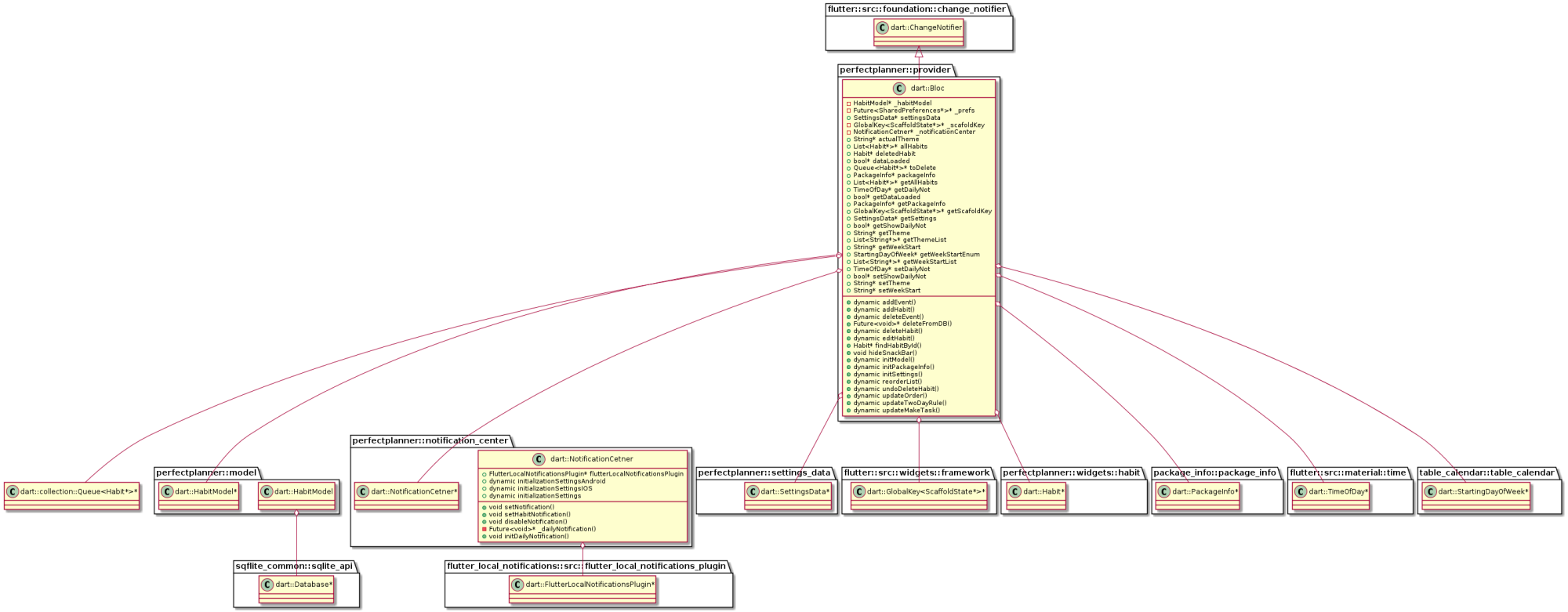


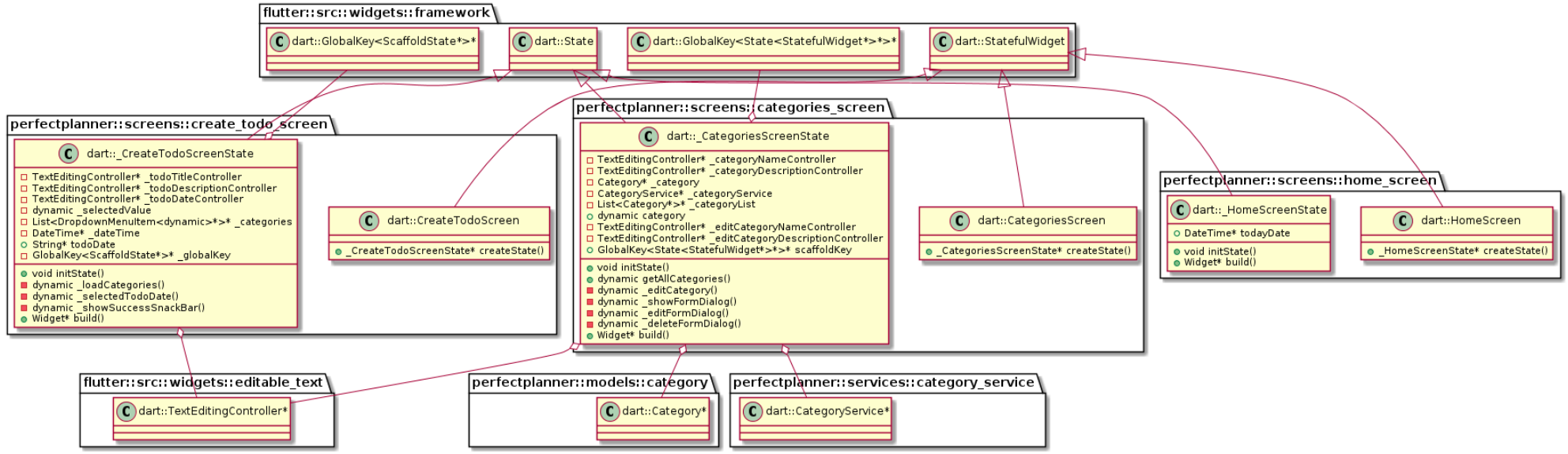
Фрагмент 2

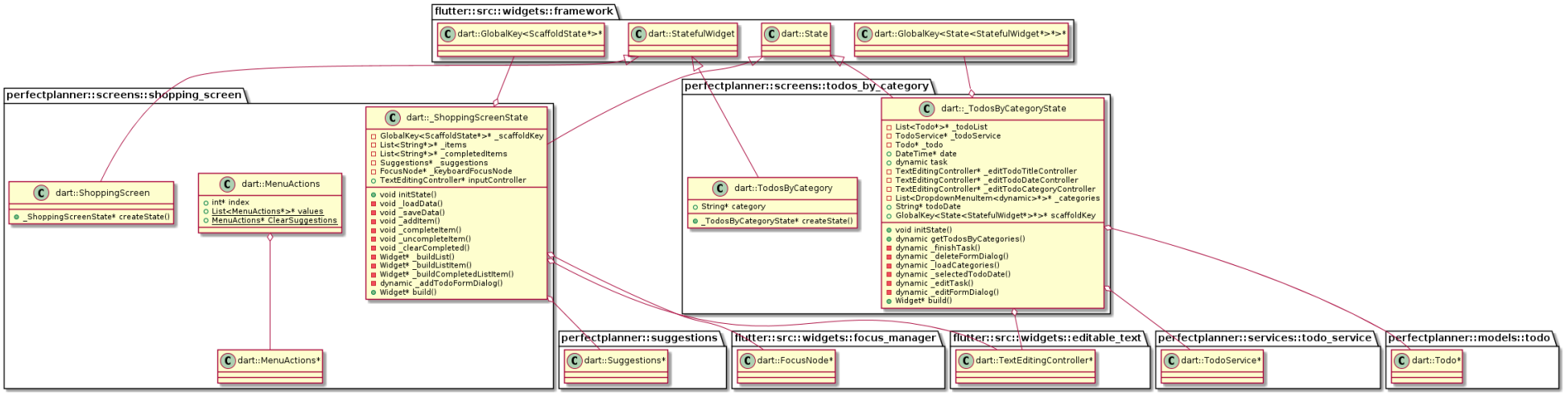

Користувач
Головна сторінка
Сторінка звичок
Створити звичку
Ввести назву
Використати правило 2 днів
Додати в список завдань
Редагувати звичку
Змінити назву
Видалити звичку
Відкрити розширені налаштування
Визначити час, коли виконувати звичку
Визначити рутину
Визначити винагороду
Показувати винагороду
Відмітка на кожен день тижня
Відмітити як невиконане
Пропустити
Відмітити як виконане
Список покупок
Очистити список підказок
Очистити список від куплених продуктів
Позначити продукт як куплений
Додати продукт
Додати до списку завдань 'Сходити в магазин'
Можливість вибрати товар з підказок
Сторінка налаштувань
Обрати тему
Світла тема
Темна тема
Тема за замовчуванням
Увімкнути сповіщення
Обрати час нагадування
<<include>>
<<extend>>

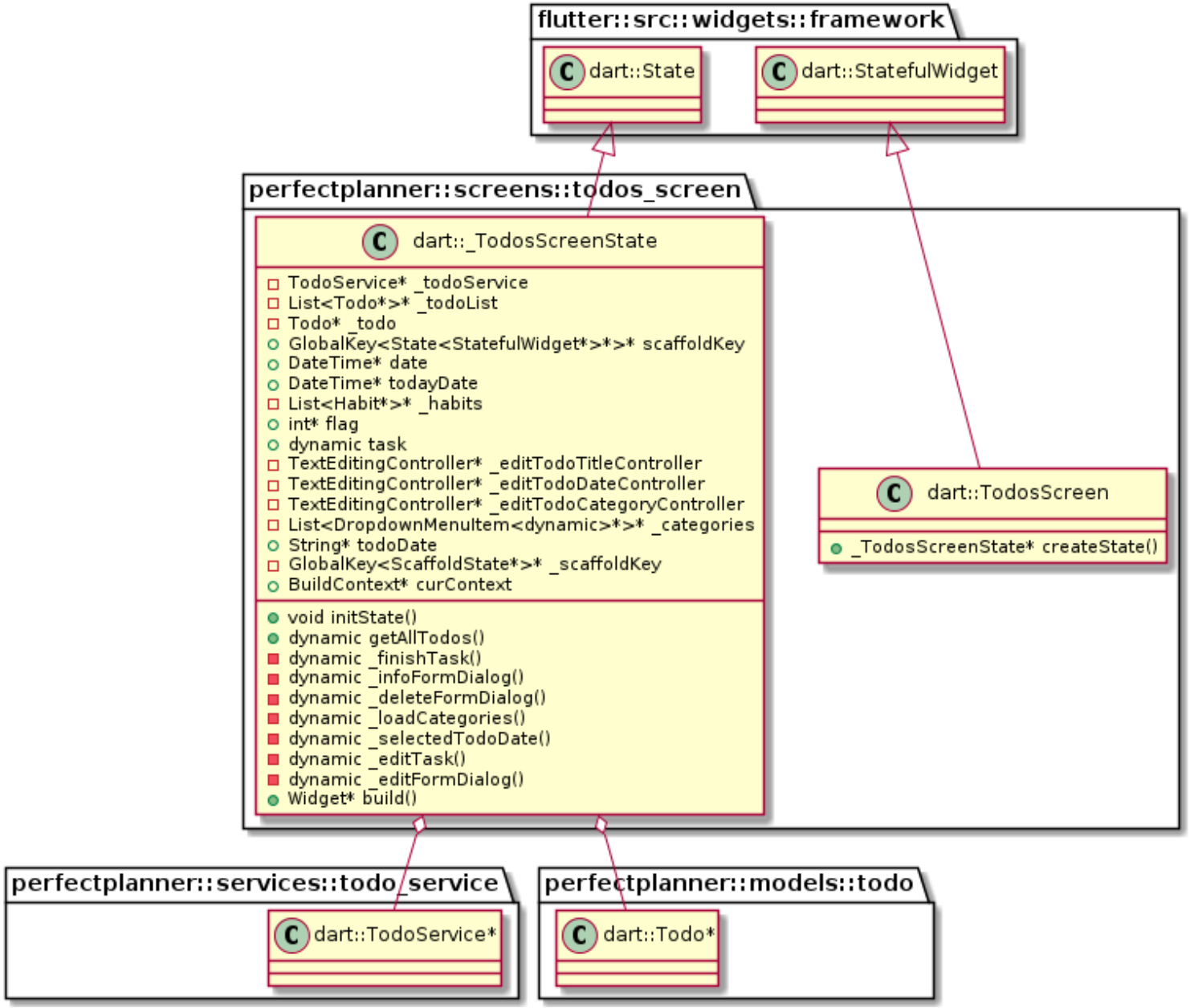
Додаток Б. Діаграма класів

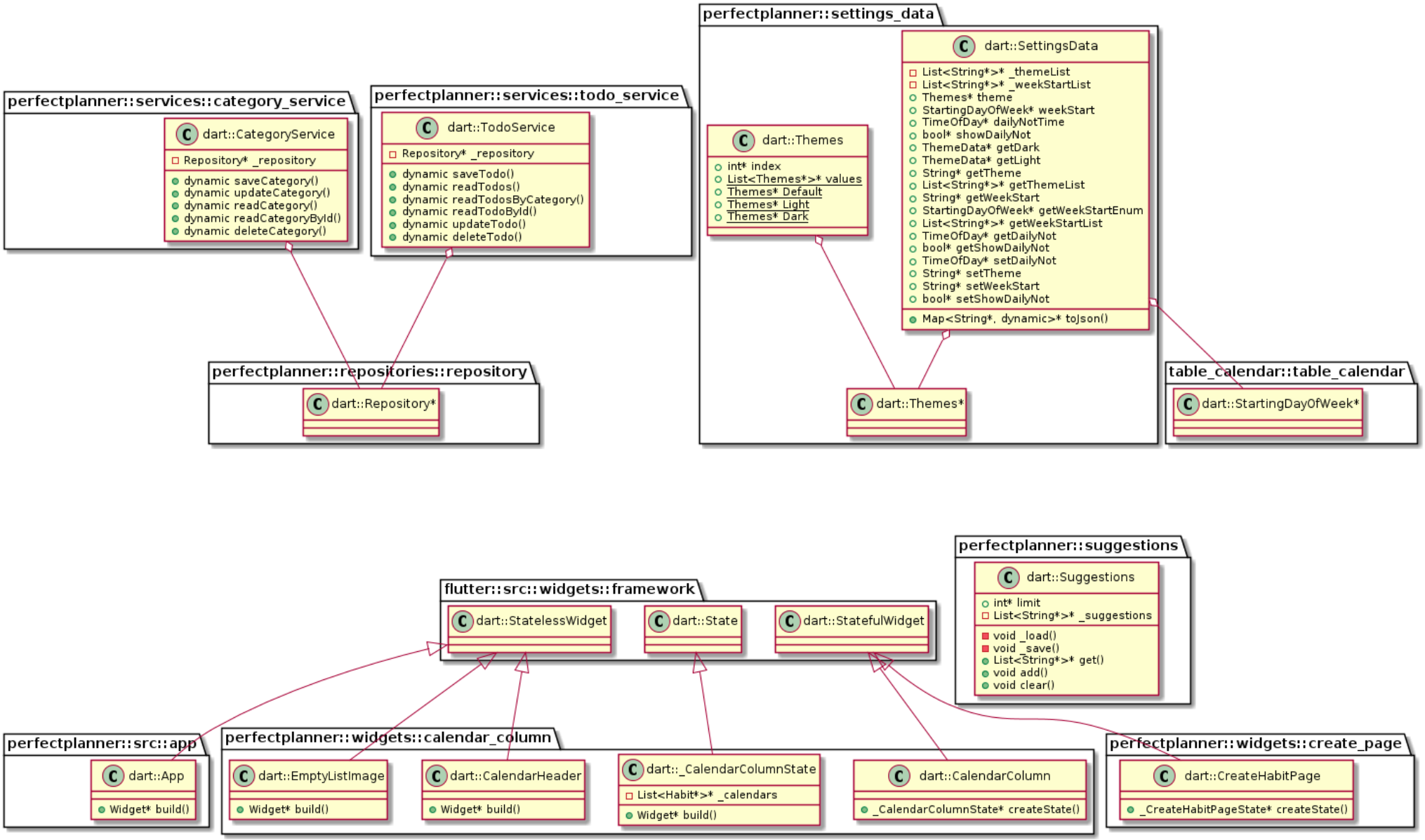


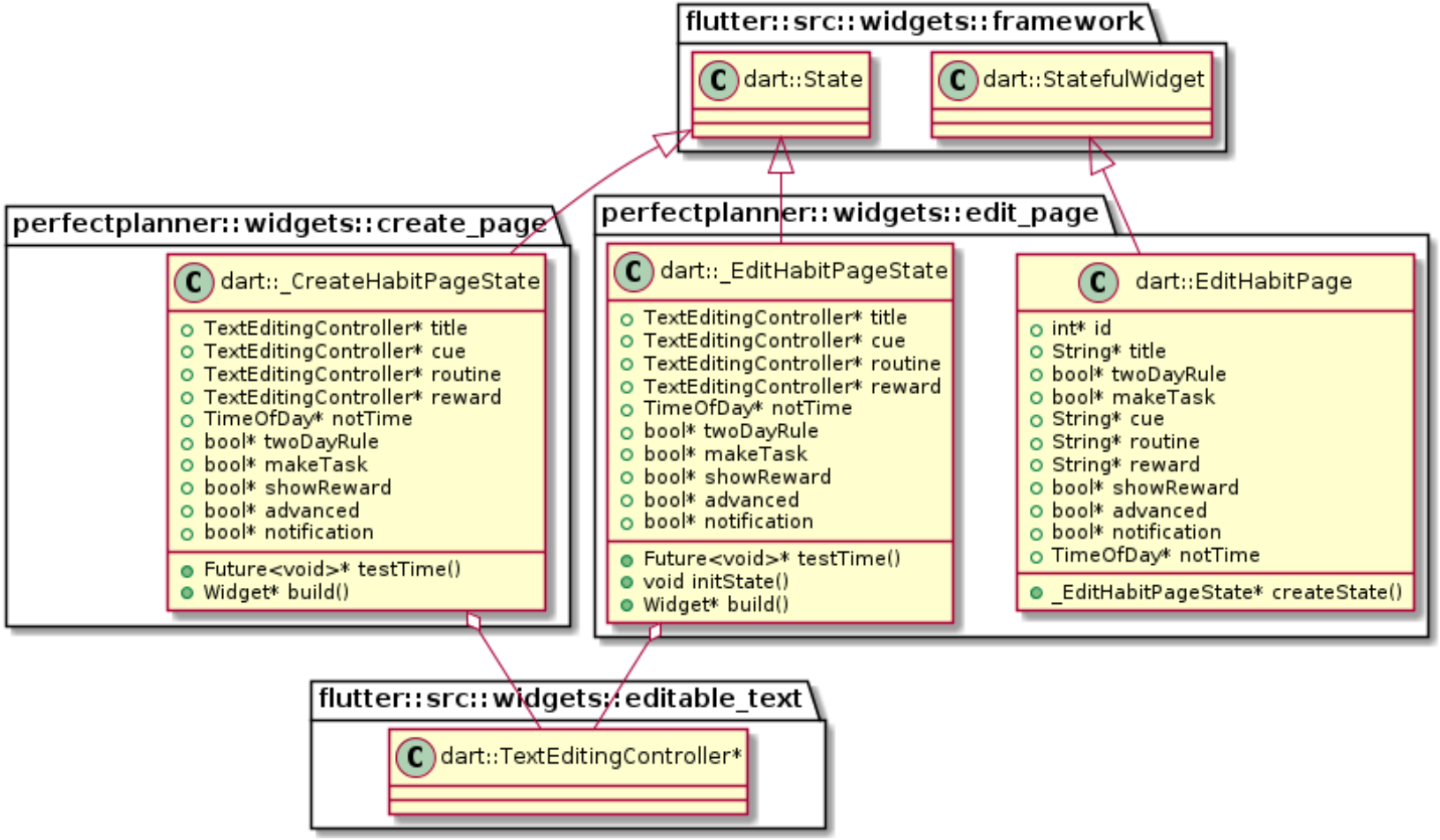

flutter::src::foundation::change_notifier
dart::ChangeNotifier
perfectplanner::provider
dart::Bloc
HabitModel* _habitModel
Future<SharedPreferences*>* _prefs
SettingsData* settingsData
GlobalKey<ScaffoldState*>* _scafoldKey
NotificationCetner* _notificationCenter
String* actualTheme
List<Habit*>* allHabits
Habit* deletedHabit
bool* dataLoaded
Queue<Habit*>* toDelete
PackageInfo* packageInfo
List<Habit*>* getAllHabits
TimeOfDay* getDailyNot
bool* getDataLoaded
PackageInfo* getPackageInfo
GlobalKey<ScaffoldState*>* getScafoldKey
SettingsData* getSettings
bool* getShowDailyNot
String* getTheme
List<String*>* getThemeList
String* getWeekStart
StartingDayOfWeek* getWeekStartEnum
List<String*>* getWeekStartList
TimeOfDay* setDailyNot
bool* setShowDailyNot
String* setTheme
String* setWeekStart
dynamic addEvent()
dynamic addHabit()
dynamic deleteEvent()
Future<void>* deleteFromDB()
dynamic deleteHabit()
dynamic editHabit()
Habit* findHabitById()
void hideSnackBar()
dynamic initModel()
dynamic initPackageInfo()
dynamic initSettings()
dynamic reorderList()
dynamic undoDeleteHabit()
dynamic updateOrder()
dynamic updateTwoDayRule()
dynamic updateMakeTask()
dart::collection::Queue<Habit*>*
perfectplanner::model
dart::HabitModel*
dart::HabitModel
perfectplanner::notification_center
dart::NotificationCetner*
dart::NotificationCetner
FlutterLocalNotificationsPlugin* flutterLocalNotificationsPlugin
dynamic initializationSettingsAndroid
dynamic initializationSettingsIOS
dynamic initializationSettings
void setNotification()
void setHabitNotification()
void disableNotification()
Future<void>* _dailyNotification()
void initDailyNotification()
perfectplanner::settings_data
dart::SettingsData*
flutter::src::widgets::framework
dart::GlobalKey<ScaffoldState*>*
perfectplanner::widgets::habit
dart::Habit*
package_info::package_info
dart::PackageInfo*
flutter::src::material::time
dart::TimeOfDay*
table_calendar::table_calendar
dart::StartingDayOfWeek*
sqflite_common::sqlite_api
dart::Database*
flutter_local_notifications::src::flutter_local_notifications_plugin
dart::FlutterLocalNotificationsPlugin*

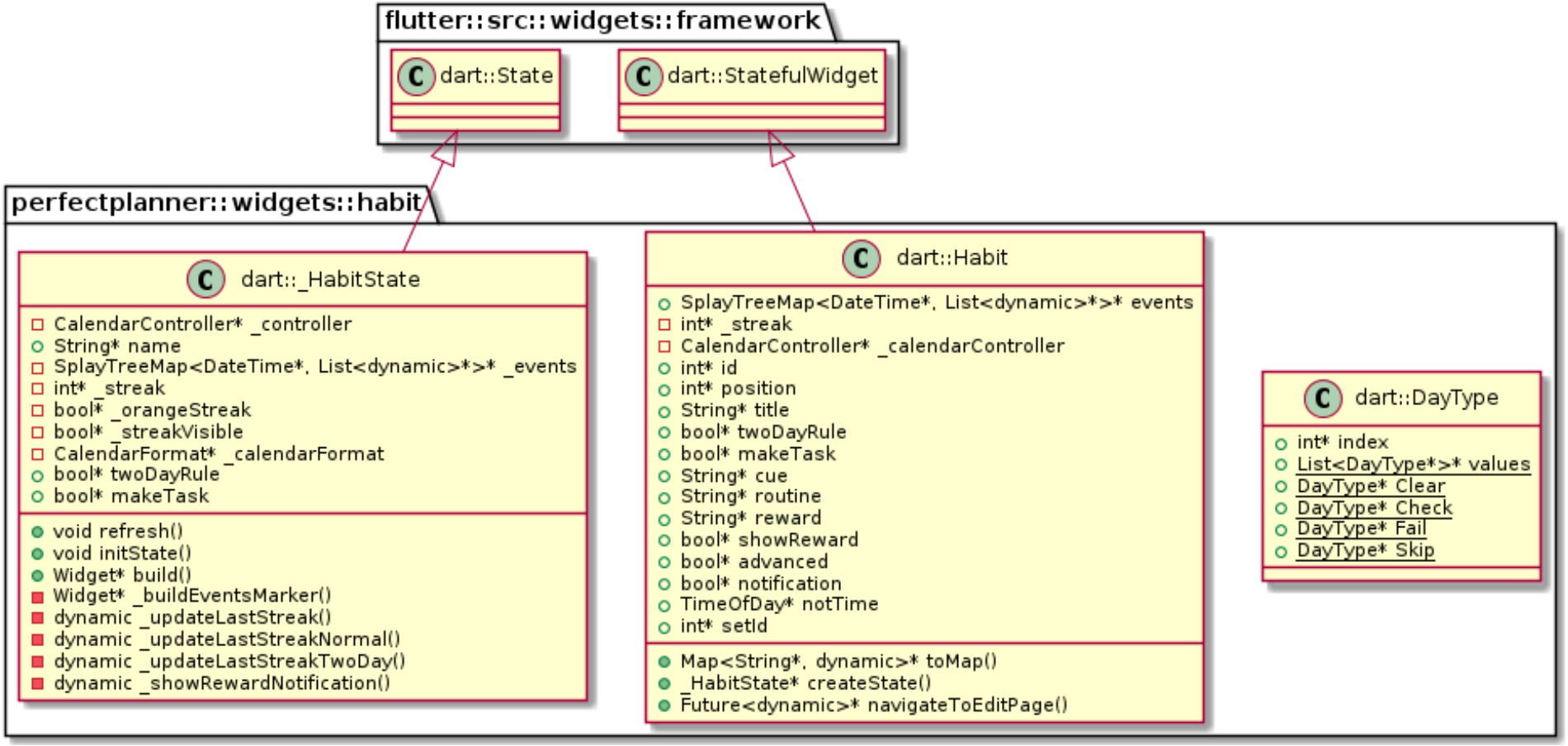

flutter::src::widgets::framework
dart::GlobalKey<ScaffoldState*>*
dart::State
dart::GlobalKey<State<StatefulWidget*>*>*
dart::StatefulWidget
perfectplanner::screens::create_todo_screen
dart::_CreateTodoScreenState
TextEditingController* _todoTitleController
TextEditingController* _todoDescriptionController
TextEditingController* _todoDateController
dynamic _selectedValue
List<DropdownMenuItem<dynamic>*>* _categories
DateTime* _dateTime
String* todoDate
GlobalKey<ScaffoldState*>* _globalKey
void initState()
dynamic _loadCategories()
dynamic _selectedTodoDate()
dynamic _showSuccessSnackBar()
Widget* build()
dart::CreateTodoScreen
_CreateTodoScreenState* createState()
perfectplanner::screens::categories_screen
dart::_CategoriesScreenState
TextEditingController* _categoryNameController
TextEditingController* _categoryDescriptionController
Category* _category
CategoryService* _categoryService
List<Category*>* _categoryList
dynamic category
TextEditingController* _editCategoryNameController
TextEditingController* _editCategoryDescriptionController
GlobalKey<State<StatefulWidget*>*>* scaffoldKey
void initState()
dynamic getAllCategories()
dynamic _editCategory()
dynamic _showFormDialog()
dynamic _editFormDialog()
dynamic _deleteFormDialog()
Widget* build()
dart::CategoriesScreen
_CategoriesScreenState* createState()
perfectplanner::screens::home_screen
dart::_HomeScreenState
DateTime* todayDate
void initState()
Widget* build()
dart::HomeScreen
_HomeScreenState* createState()
flutter::src::widgets::editable_text
dart::TextEditingController*
perfectplanner::models::category
dart::Category*
perfectplanner::services::category_service
dart::CategoryService*

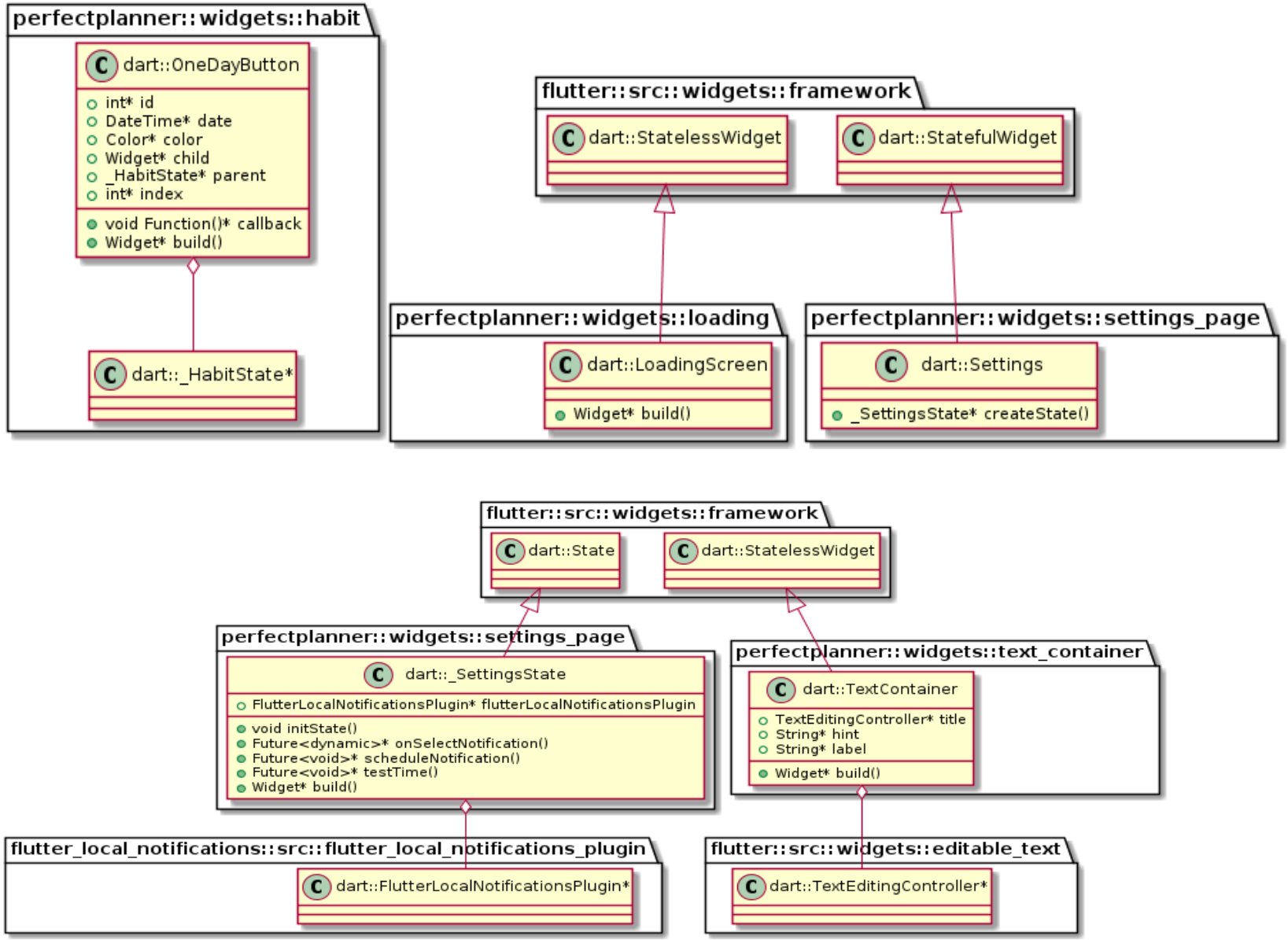

flutter::src::widgets::framework
dart::GlobalKey<ScaffoldState*>*
dart::StatefulWidget
dart::State
dart::GlobalKey<State<StatefulWidget*>*>*
perfectplanner::screens::shopping_screen
dart::ShoppingScreen
_ShoppingScreenState* createState()
dart::MenuActions
int* index
List<MenuActions*>* values
MenuActions* ClearSuggestions
dart::MenuActions*
dart::_ShoppingScreenState
GlobalKey<ScaffoldState*>* _scaffoldKey
List<String*>* _items
List<String*>* _completedItems
Suggestions* _suggestions
FocusNode* _KeyboardFocusNode
TextEditingController* inputController
void initState()
void _loadData()
void _saveData()
void _addItem()
void _completeItem()
void _uncompleteItem()
void _clearCompleted()
Widget* _buildList()
Widget* _buildListItem()
Widget* _buildCompletedListItem()
dynamic _addTodoFormDialog()
Widget* build()
perfectplanner::screens::todos_by_category
dart::TodosByCategory
String* category
_TodosByCategoryState* createState()
dart::_TodosByCategoryState
List<Todo*>* _todoList
TodoService* _todoService
Todo* _todo
DateTime* date
dynamic task
TextEditingController* _editTodoTitleController
TextEditingController* _editTodoDateController
TextEditingController* _editTodoCategoryController
List<DropdownMenuItem<dynamic>*>* _categories
String* todoDate
GlobalKey<State<StatefulWidget*>*>* scaffoldKey
void initState()
dynamic getTodosByCategories()
dynamic _finishTask()
dynamic _deleteFormDialog()
dynamic _loadCategories()
dynamic _selectedTodoDate()
dynamic _editTask()
dynamic _editFormDialog()
Widget* build()
perfectplanner::suggestions
dart::Suggestions*
flutter::src::widgets::focus_manager
dart::FocusNode*
flutter::src::widgets::editable_text
dart::TextEditingController*
perfectplanner::services::todo_service
dart::TodoService*
perfectplanner::models::todo
dart::Todo*


flutter::src::widgets::framework
dart::State
dart::StatefulWidget
perfectplanner::screens::todos_screen
dart::_TodosScreenState
TodoService* _todoService
List<Todo*>* _todoList
Todo* _todo
GlobalKey<State<StatefulWidget*>*>* scaffoldKey
DateTime* date
DateTime* todayDate
List<Habit*>* _habits
int* flag
dynamic task
TextEditingController* _editTodoTitleController
TextEditingController* _editTodoDateController
TextEditingController* _editTodoCategoryController
List<DropdownMenuItem<dynamic>*>* _categories
String* todoDate
GlobalKey<ScaffoldState*>* _scaffoldKey
BuildContext* curContext
void initState()
dynamic getAllTodos()
dynamic _finishTask()
dynamic _infoFormDialog()
dynamic _deleteFormDialog()
dynamic _loadCategories()
dynamic _selectedTodoDate()
dynamic _editTask()
dynamic _editFormDialog()
Widget* build()
dart::TodosScreen
_TodosScreenState* createState()
perfectplanner::services::todo_service
dart::TodoService*
perfectplanner::models::todo
dart::Todo*


perfectplanner::services::category_service
dart::CategoryService
Repository* _repository
dynamic saveCategory()
dynamic updateCategory()
dynamic readCategory()
dynamic readCategoryById()
dynamic deleteCategory()
perfectplanner::services::todo_service
dart::TodoService
Repository* _repository
dynamic saveTodo()
dynamic readTodos()
dynamic readTodosByCategory()
dynamic readTodoById()
dynamic updateTodo()
dynamic deleteTodo()
perfectplanner::settings_data
dart::SettingsData
List<String*>* _themeList
List<String*>* _weekStartList
Themes* theme
StartingDayOfWeek* weekStart
TimeOfDay* dailyNotTime
bool* showDailyNot
ThemeData* getDark
ThemeData* getLight
String* getTheme
List<String*>* getThemeList
String* getWeekStart
StartingDayOfWeek* getWeekStartEnum
List<String*>* getWeekStartList
TimeOfDay* getDailyNot
bool* getShowDailyNot
TimeOfDay* setDailyNot
String* setTheme
String* setWeekStart
bool* setShowDailyNot
Map<String*, dynamic>* toJson()
dart::Themes
int* index
List<Themes*>* values
Themes* Default
Themes* Light
Themes* Dark
dart::Themes*
perfectplanner::repositories::repository
dart::Repository*
table_calendar::table_calendar
dart::StartingDayOfWeek*
perfectplanner::suggestions
dart::Suggestions
int* limit
List<String*>* _suggestions
void _load()
void _save()
List<String*>* get()
void add()
void clear()
flutter::src::widgets::framework
dart::StatelessWidget
dart::State
dart::StatefulWidget
perfectplanner::src::app
dart::App
Widget* build()
perfectplanner::widgets::calendar_column
dart::EmptyListImage
Widget* build()
dart::CalendarHeader
Widget* build()
dart::_CalendarColumnState
List<Habit*>* _calendars
Widget* build()
dart::CalendarColumn
_CalendarColumnState* createState()
perfectplanner::widgets::create_page
dart::CreateHabitPage
_CreateHabitPageState* createState()


flutter::src::widgets::framework
dart::State
dart::StatefulWidget
perfectplanner::widgets::create_page
dart::_CreateHabitPageState
TextEditingController* title
TextEditingController* cue
TextEditingController* routine
TextEditingController* reward
TimeOfDay* notTime
bool* twoDayRule
bool* makeTask
bool* showReward
bool* advanced
bool* notification
Future<void>* testTime()
Widget* build()
perfectplanner::widgets::edit_page
dart::_EditHabitPageState
TextEditingController* title
TextEditingController* cue
TextEditingController* routine
TextEditingController* reward
TimeOfDay* notTime
bool* twoDayRule
bool* makeTask
bool* showReward
bool* advanced
bool* notification
Future<void>* testTime()
void initState()
Widget* build()
dart::EditHabitPage
int* id
String* title
bool* twoDayRule
bool* makeTask
String* cue
String* routine
String* reward
bool* showReward
bool* advanced
bool* notification
TimeOfDay* notTime
_EditHabitPageState* createState()
flutter::src::widgets::editable_text
dart::TextEditingController*


flutter::src::widgets::framework
dart::State
dart::StatefulWidget
perfectplanner::widgets::habit
dart::_HabitState
CalendarController* _controller
String* name
SplayTreeMap<DateTime*, List<dynamic>*>* _events
int* _streak
bool* _orangeStreak
bool* _streakVisible
CalendarFormat* _calendarFormat
bool* twoDayRule
bool* makeTask
void refresh()
void initState()
Widget* build()
Widget* _buildEventsMarker()
dynamic _updateLastStreak()
dynamic _updateLastStreakNormal()
dynamic _updateLastStreakTwoDay()
dynamic _showRewardNotification()
dart::Habit
SplayTreeMap<DateTime*, List<dynamic>*>* events
int* _streak
CalendarController* _calendarController
int* id
int* position
String* title
bool* twoDayRule
bool* makeTask
String* cue
String* routine
String* reward
bool* showReward
bool* advanced
bool* notification
TimeOfDay* notTime
int* setId
Map<String*, dynamic>* toMap()
_HabitState* createState()
Future<dynamic>* navigateToEditPage()
dart::DayType
int* index
List<DayType*>* values
DayType* Clear
DayType* Check
DayType* Fail
DayType* Skip


perfectplanner::widgets::habit
dart::OneDayButton
int* id
DateTime* date
Color* color
Widget* child
_HabitState* parent
int* index
void Function()* callback
Widget* build()
dart::_HabitState*
flutter::src::widgets::framework
dart::StatelessWidget
dart::StatefulWidget
perfectplanner::widgets::loading
dart::LoadingScreen
Widget* build()
perfectplanner::widgets::settings_page
dart::Settings
_SettingsState* createState()
flutter::src::widgets::framework
dart::State
dart::StatelessWidget
perfectplanner::widgets::settings_page
dart::_SettingsState
FlutterLocalNotificationsPlugin* flutterLocalNotificationsPlugin
void initState()
Future<dynamic>* onSelectNotification()
Future<void>* scheduleNotification()
Future<void>* testTime()
Widget* build()
perfectplanner::widgets::text_container
dart::TextContainer
TextEditingController* title
String* hint
String* label
Widget* build()
flutter_local_notifications::src::flutter_local_notifications_plugin
dart::FlutterLocalNotificationsPlugin*
flutter::src::widgets::editable_text
dart::TextEditingController*