УДК 004.4`24 http://doi.org/10.15407/pp2023.01.011

М. А. Журба, І.В. Стеценко

АВТОМАТИЗАЦІЯ ПРОЦЕСУ СТВОРЕННЯ НОВИХ ЦІЛЕЙ У XCODE ПРОЄКТІ

На сьогоднішній день розробка застосунків під мобільні пристрої стає все більш популярною, але разом з тим більш вартісною. У деяких бізнес-моделях використовуються однотипні застосунки, створені на одній кодовій базі, із впровадженням специфічних відмінностей у дизайні або поведінці для кінцевого продукту. Один із підходів для розробки таких застосунків під iOS полягає у використанні окремих Xcode цілей для кожного продукту. Процес створення нової цілі вимагає проведення ряду рутинних операцій, що потребує витрат часу програміста, а написання коду потенційно схильне до помилок. У даній статті пропонується вдосконалити цей підхід шляхом використання скриптів для генерації проєктного файлу на основі заданих конфігурацій. Розроблено консольний скрипт для генерації проєктного файлу і проєкт для ілюстрації підходу.
Ключові слова: Xcode ціль, Xcode проєкт, мобільний застосунок, скрипт, генерація файлу, автоматизація

Вступ

Кожен бізнес прагне ефективно витрачати кошти на всіх етапах життєвого циклу продукту або сервісу. У випадку створення мобільних застосунків у бізнесі досить часто виникає потреба у схожих між собою застосунках із тих чи інших причин. Такими причинами можуть бути, зокрема, необхідність в окремих застосунках для різних регіонів, необхідність в окремих застосунках для дочірніх брендів, прагнення зайняти домінуючу позицію у певній ніші шляхом створення різних застосунків під окремими брендами, використання різних стратегій просування для різних застосунків. У будь-якому випадку створення нових застосунків з нуля та паралельна підтримка усієї множини застосунків не є ефективним рішенням. Не тільки тому, що вона вимагатиме великих витрат фінансів та часу, а ще й тому, що зростає ризик виникнення помилок. Може трапитись ситуація, за якої новий функціонал був реалізований тільки в одному із застосунків усієї множини через виникнення людських помилок.

Саме тому для таких розробок використовують підходи із перевикористанням коду і побудовою кінцевих продуктів на одній кодовій базі. У контексті iOS розробки таких підходів декілька: винесення спільного коду у бібліотеки і їх використання в окремих Xcode проєктах, використання окремих цілей для кожного з кінцевих продуктів у рамках одного Xcode проєкту [1], використання окремих схем для кожного з кінцевих продуктів у рамках одного Xcode проєкту [2].

У порівнянні з іншими, підхід із використанням окремих цілей є найбільш гнучким і таким, що мінімізує дублювання коду. Однак, є суттєвий недолік. Підхід вимагає додавання вручну нової цілі та її конфігурації, водночас для кожного файлу проєкту необхідно вказувати цілі, до яких він належить.

Метою даного наукового дослідження є автоматизація рутинних дій зі створення однотипних iOS застосунків за рахунок використання скрипта для генерування проєктного файлу на основі заданої конфігурації для цілей. Скрипт дозволить додавати нові цілі до проєкту за декілька секунд, а також мінімізувати шанс виникнення людських помилок. Розроблений невеликий застосунок використовується для ілюстрації оновленого підходу.

1. Огляд класичного підходу із використанням декількох цілей

Для розробки під iOS використовується середовище розробки Xcode. Після створення проєкту Xcode, автоматично створюються базові вихідні файли, ресурсні каталоги, а також ціль для продукту

(рис.1). Загалом ціль визначає окремий продукт, який буде створюватись в результаті виконання проєкту. Для цілі визначається список вихідних файлів для компіляції та список ресурсів, в яких може міститись інформація про використовувані зображення, кольори, шрифти тощо. Ці файли можуть належати одразу декільком цілям, а можуть просто знаходитись у проєкті і не належати жодній. Також у кожної цілі є ідентифікатор [3], за допомогою якого в подальшому App Store та iPhone відрізняє застосунки між собою. Схема визначає колекцію цілей для побудови та конфігурацію, яку слід використовувати під час побудови.

Таким чином, суть підходу полягає у створенні декількох цілей і встановлення різних ідентифікаторів для кожної з них. Для того, щоб створити нову ціль, необхідно виконати такі дії:

– перейти в Xcode проєкті до розділу з його налаштуваннями;

– у розділі цілей вибрати одну з існуючих цілей;

– обрати «дублювати», при цьому автоматично створиться нова ціль та нова схема для неї, новий файл `.plist` з базовими конфігураціями цілі. Всі вихідні та ресурсні файли, які належали до початкової цілі, тепер також належатимуть і до новоствореної;

– перейменувати всі новостворені цілі, схеми та файли відповідно до потреб проєкту (за замочуванням назви створюються шляхом додаванням Copy до первісних назв);

– у розділі цілей обрати нову ціль, перейти до вкладки General та змінити поле Display Name;

– у розділі Build Settings підрозділі Packaging оновити поле Info.plist та Product Bundle Identifier.

Навіть на цих етапах можуть виникнути деякі складнощі. Наприклад, де повинен бути файл Info.plist? Два файли з однаковим ім'ям не можуть перебувати в одній директорії, але розробнику зручно і звично, якщо файл називається саме Info.plist. Отож, необхідно також створити окремі директорії для зберігання файлів, специфічних для різних цілей. Але після зміни розташування файлу Info.plist доведеться знову оновити дані про його місцезнаходження у налаштуваннях цілі. Кожен із цих пунктів алгоритму розробник може пропустити або зробити його неправильно. До того ж, описаний підхід вимагає знань і розуміння того, як влаштоване середовище розробки Xcode, тобто доступний тільки досвідченому розробнику.

Навіть після налаштування цілей, їх потрібно підтримувати у належному стані,

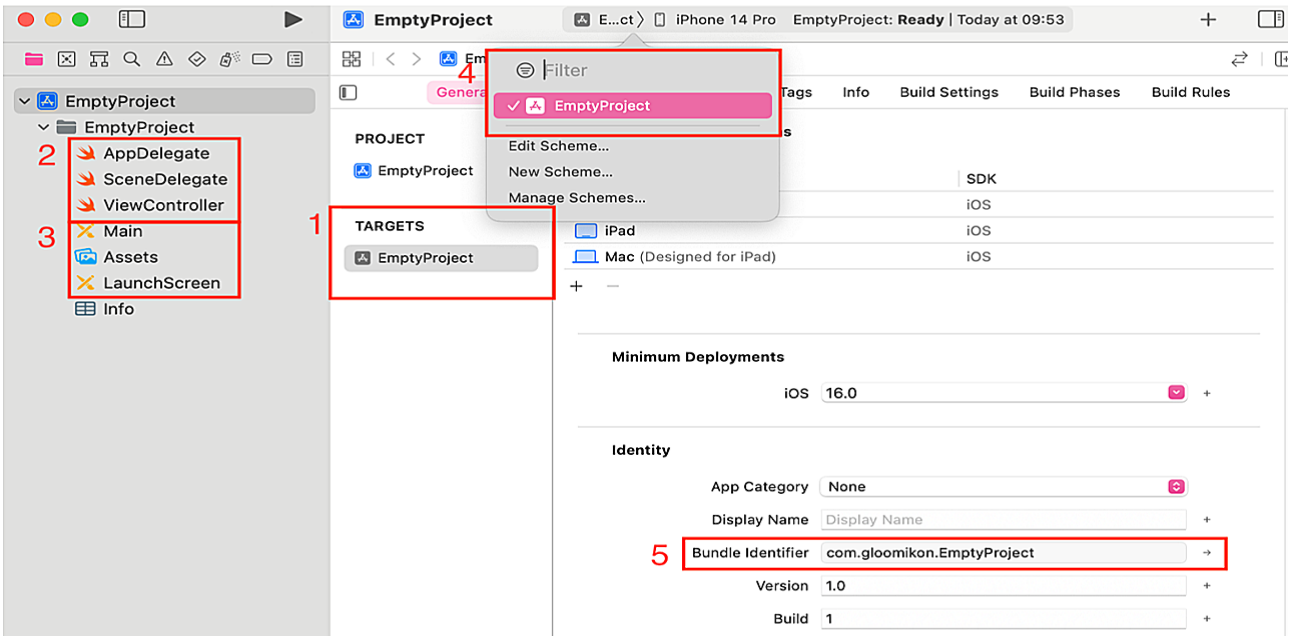

Рис. 1. Вікно новоствореного проєкту Xcode: 1 – Список цілей, 2 – Вихідні файли, 3 – Ресурсні файли, 4 – Список схем, 5 – Ідентифікатор цілі.

а саме, для кожного новоствореного файлу обирати, до яких цілей він належить. Оскільки вибір цілей вказується не в коді файлу, а через контроль прапорців інтерфейсу Xcode, то таку дію можна легко пропустити. Якщо прапорець не вибрати, то це буде означати, що файла не існує для цілі. Тобто, якщо певний файл створений у проєкті, це не означає, що ціль буде його бачити. Панель вибору прапорців показано на рисунку 2.

Отже, класичний підхід із використанням декількох цілей вимагає від розробника виконання ряду рутинних дій для створення нової цілі, які потенційно можуть бути не виконані або виконані неправильно. Список вихідних файлів та ресурсних каталогів створюваного застосунку визначається вказуванням відповідних прапорців цілей, що також є незручним. Тому необхідно знайти шляхи вирішення зазначених проблемних моментів.

2. Генерація проєктного файлу за допомогою утиліти для кодогенерації

Перша проблема, яка буде вирішуватися, – це встановлення посилань на вихідні файли та ресурсні каталоги для кожної нової цілі. Як вже було зазначено, у проєкті Xcode відповідність файлів та каталогів цілі встановлюється шляхом вказування відповідного прапорця. Але що саме відбувається, коли розробник вмикає або вимикає прапорець цілі для певного файлу? При створенні Xcode проєкту створюється файл із розширенням `.xcodeproj`. Насправді, це директорія, яка зберігає дані про проєкт. У ній знаходиться файл `.pbxproj` (Project Builder Xcode Project). Саме в цьому файлі містяться всі дані про конфігурації проєкту, його цілі, схеми. Коли розробник коригує прапорець, Xcode оновлює дані у файлі `.pbxproj`. Для розробника це незручно, адже файл має складний для сприйняття синтаксис (рис.3).

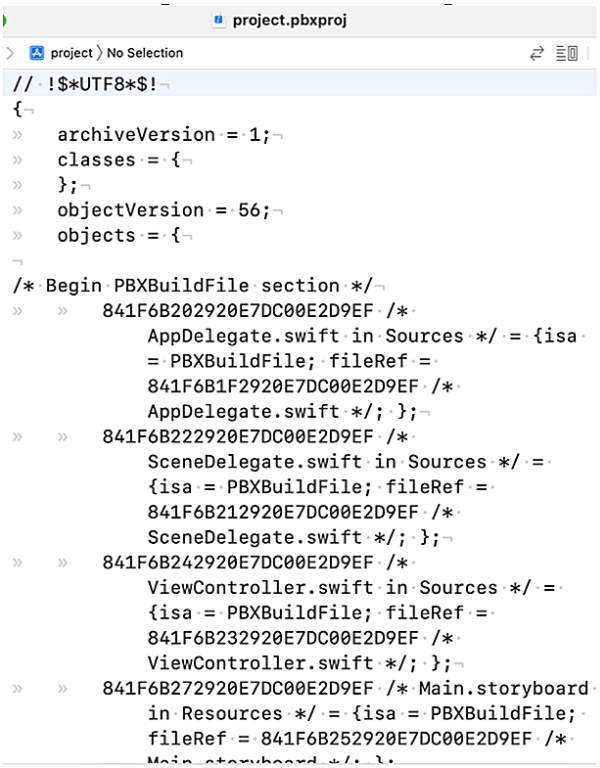

Рис. 3. Фрагмент файлу `.pbxproj`

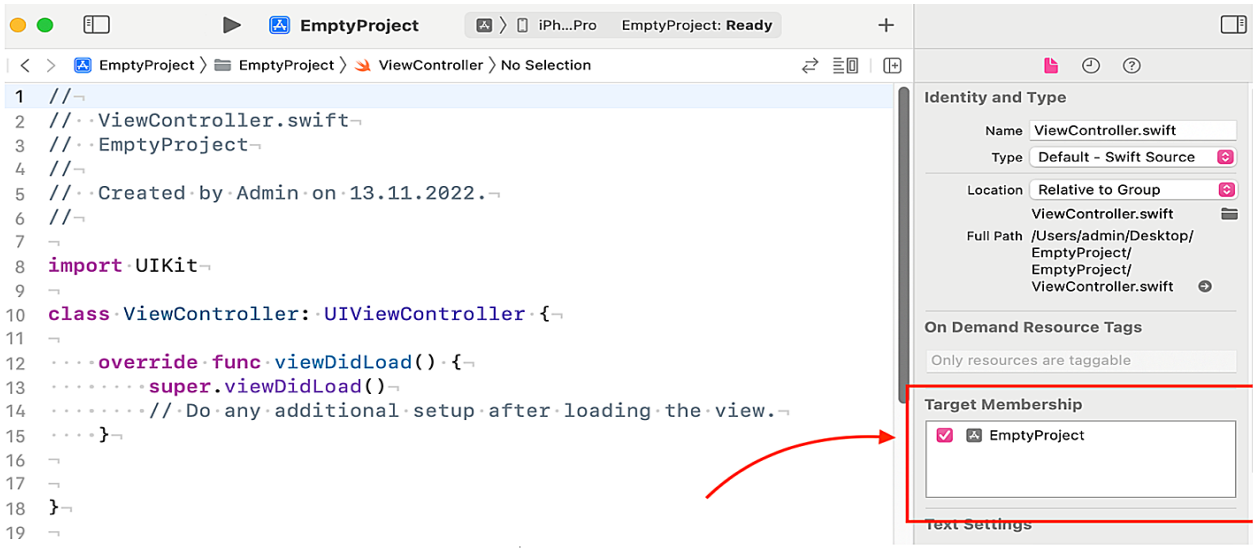

Рис. 2. Поле із налаштуванням приналежності файлу до цілі.

Саме тому ідея полягає у використанні утиліти XcodeGen [4], яка генерує директорію .xcodeproj на основі заданого конфігураційного файлу `project.yaml`. Цей підхід має наступні переваги:

– `.yaml` файл більш читабельний у порівнянні із класичним .pbxproj;

– для кожної цілі можна вказувати шляхи до директорій із вихідними файлами та ресурсними каталогами;

– зменшення випадків у роботі із системою контролю версій із конфліктами злиття у великих командах;

– створення нової цілі вимагатиме лише копіювання частини коду у файлі.

Алгоритм дій для того, щоб позбутися `.xcodeproj` і перейти до використання XcodeGen, може бути наступний:

– у репозиторії проєкту створити окремі директорії для кожної з цілей з їхніми назвами, а також каталог COMMON для спільних файлів;

– за допомогою утиліти BuildSettingExtractor [5] експортувати налаштування проєкту і цілей із .xcodeproj до конфігураційних файлів;

– видалити `.xcodeproj`;

– створити `project.yaml` та додати до нього такі поля:

– name – ім'я проєкту;

– configs – назви конфігів із вказанням успадкування release або debug;

– settings – налаштування проєкту;

– base – спільні налаштування для всіх конфігів, <ім'я конфігу> – налаштування, специфічні до конфігу;

– options – різні опції для XcodeGen;

– fileGroups – список файлів, які додані до проєкту, але не до цілей (у тому числі, можна додати файл project.yaml);

– settingGroups – список налаштувань для перевикористання (необхідно задати назву і потім вказати список налаштувань, пізніше сюди можна винести спільні налаштування для всіх цілей);

– targets – список цілей (для кожної цілі потрібно задати ім'я, тип, платформу, список налаштувань settings, список файлів цілі sources).

Приклад файлу `project.yaml` наведений у GitHub репозиторії [6].

Зрештою, потрібно лише використати консольну команду xcodegen generate, яка створить файл .xcodeproj.

Отже, за допомогою XcodeGen і конфігураційного файлу project.yaml можна вирішити проблему рутинного додавання файлів до цілей. Натомість можна лише вказати директорії з ними у полі sources. Також за допомогою конфігураційного файлу створення нової цілі відбувається простіше, швидше і знижується можливість виникнення помилок через зменшення загальної кількості операцій, які потрібно виконати розробнику.

Проте створення нової цілі відбувається все ще вручну. Далі буде описано, як можна автоматизувати цей процес.

3. Модифікація проєктного файлу за допомогою самописної утиліти командного рядку

У класичному підході з використанням декількох цілей для створення нової розробнику спочатку потрібно дублювати існуючу, потім змінити деякі поля і назви у новоствореній цілі. Тобто глобально, знаючи назву нової цілі, а також назву та параметри цілі, від якої буде зроблено дублювання, можна автоматизувати процес створення за допомогою написаного власноруч скрипту. Ця ідея і була закладена під час розробки утиліти командного рядку `igen`.

Команда `igen` приймає два параметри – шлях до файлу з конфігураціями цілей та шлях до project.yaml проєкту.

Перший файл має два поля:

– project – назва проєкту;

– targets – словник, в якому ключ – це назва цілі, а значення – це параметри цілі

Параметри можуть мати поля:

– steals – список існуючих папок, які потрібно скопіювати для нової цілі,

– inherit – назва цілі, від якої відбувається дублювання.

Утиліта працює за наступним алгоритмом:

– виконує парсинг файлу із конфігураціями цілей;

– виконує парсинг `project.yaml`, звідки дістаються дані про вже існуючі цілі;

– виконує створення моделей для нових цілей на основі вхідних даних;

– виконує кодогенерацію оновленого блоку для цілей файлу `project.yaml`;

– створює директорії COMMON і <назва_цілі> за допомогою виклику консольної команди `mkdir`;

– якщо у якоїсь цілі було вказано steals, виконує копіювання директорій за допомогою виклику консольної команди `cp -R`;

– оновлює файл `project.yaml` (оновлюється тільки блок цілей, ключі будуть відсортовані за алфавітним порядком);

– оновлює файл з конфігураціями цілей (видаляються усі параметри цілей, якщо такі є, залишаються тільки назви цілей).

Утиліта написана мовою Swift із використанням swift-argument-parser.

Повний код розробленого алгоритму доступний у GitHub репозиторії за посиланням [7].

4. Ілюстрація розробленого підходу

Для ілюстрації був розроблений невеликий iOS проєкт iNote для створення та редагування нотаток, що має дві цілі – Alpha і Beta. Тобто на одній кодовій базі створюється два застосунки.

Обидва застосунки мають однаковий каркас у контексті функціоналу, а саме: декілька екранів привітання, головний екран з нотатками, з якого можна перейти на екран створення нової нотатки або редагування вже існуючої. Однак, деталі реалізації – різні. Застосунок Alpha має три екрани привітання, а застосунок Beta – два. Тексти та зображення на цих екранах – різні.

Унікальність у зовнішньому вигляді досягається використанням однакових ключів для текстів та назв для кольорів й зображень, але наявності окремих ресурсних каталогів для кожної з цілей. Як можна побачити на рисунку 4, зображення, хоч і мають однакову назву, різні на вигляд.

Alpha відображає список нотаток у вигляді таблиці, а Beta – у вигляді колекції. Це реалізовано за допомогою створення двох контролерів відображення з унікальною логікою: один з них використовує елемент користувацького інтерфейсу UITableView, а інший – UICollectionView.

Кожен з контролерів знаходиться у файлі `NotesListViewController` і належить тільки до однієї з цілей. Зовнішній вигляд головного екрану обох застосунків зображено на рисунку 5.

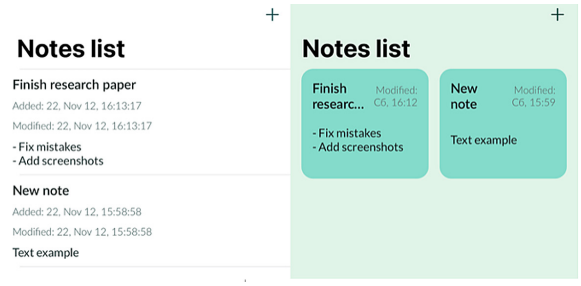

Рис. 5. Головний екран з нотатками кожного із застосунків

Водночас для того, щоб перевикористати спільну логіку, був створений файл та відповідний клас `BaseNotesListViewController`, від

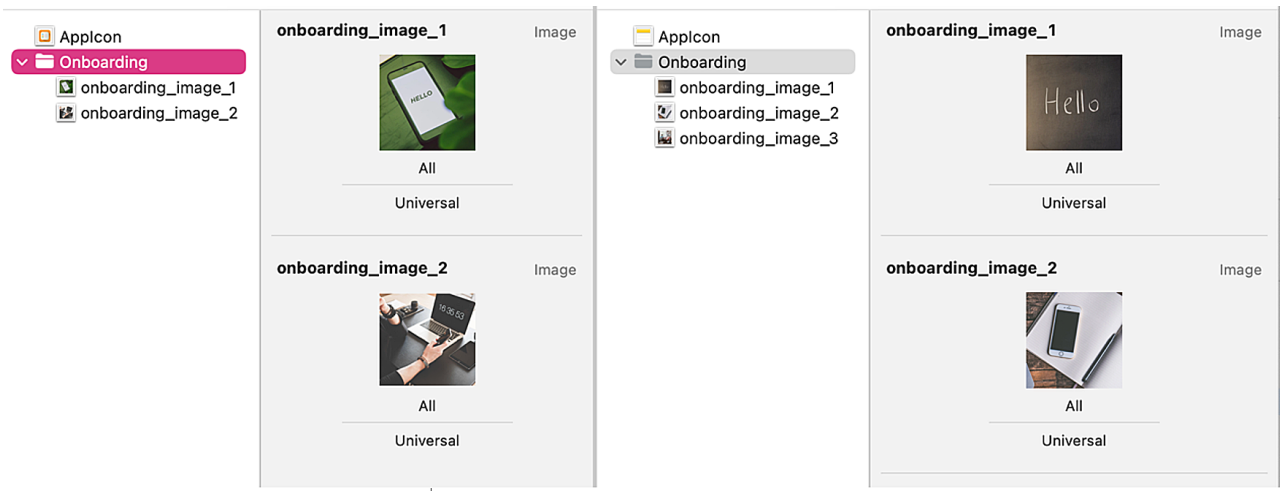

Рис. 4. Каталоги `Assets.xcassets` кожної з цілей.

якого успадковуються конкретні реалізації. Цей файл належить до обох цілей, як показано на рисунку 6.

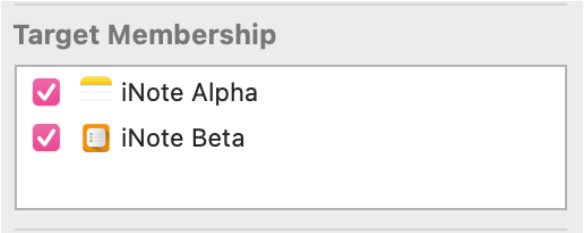

Рис. 6. Приналежність файлу `BaseNotesListViewController`

Відрізняється також і спосіб відображення екрану створення та/або редагування нотатків. Alpha виштовхує новий екран у стек навігації, використовуючи метод

```
pushViewController(
    _viewController:
UIViewController,
    animated: Bool),
```

а Beta – показує екран модально, використовуючи метод

```
present(
    _viewControllerToPresent:
UIViewController,
    animated flag: Bool,
    completion: (() -> Void)? = nil)
```

Для цього використовується такий же принцип. Клас `Coordinator`, що відповідає за навігацію, має шаблон із спільною логікою для створення модуля додавання та/або редагування нотаток. Цей шаблон знаходиться у файлі, який належить до обох цілей. Специфічна логіка знаходиться у різних файлах, кожен з яких належить тільки одній із цілей.

Описаний підхід до проєктування є використанням патерну «Шаблонний метод» [8], суть якого полягає у розбитті алгоритму на кроки (функції), виклику цих кроків у певному методі, а також реалізації тих методів, логіка яких буде однакова у базовому класі, а тих, логіка яких буде унікальна, – у підкласах.

На прикладі класу `Coordinator`, його метод `start(animated: Bool)` можна логічно розбити на два кроки: створення модулю

```
    func createModule() ->
UIViewController
```

та навігація до цього модулю

```
    func navigateToModule(with
viewController: UIViewController,
animated: Bool).
```

Створення модулю має однакову логіку. Для обох додатків це створення контролю відображення, класу для обробки бізнес-логіки, встановлення заголовку для панелі навігації. Тому реалізацію цього методу можна винести у базовий клас. Логіка власне відображення, як уже було зазначено раніше, відрізняється. Тому реалізація знаходиться у підкласах.

Структурно спільні для обох цілей файли знаходяться на диску у директорії COMMON, а окремі – у відповідних директоріях Alpha і Beta. Тобто кожна ціль буде мати файли з двох директорій – COMMON і <назва_цілі>. Інформація про це знаходиться у файлі `project.yaml` у полі sources кожної з цілей, що показано на рисунку 7.

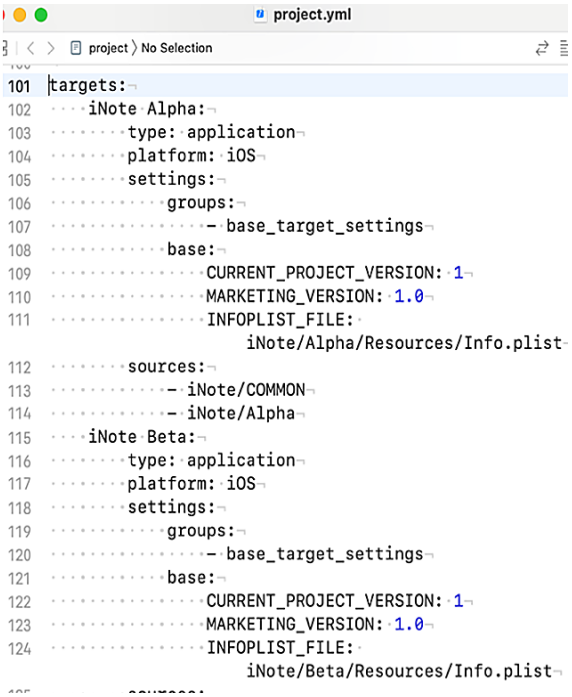

Рис. 7 Список цілей у файлі `project.yaml` для проєкту iNote

За необхідності створення третьої варіації застосунку, більше не потрібно вручну створювати ціль, налаштовувати її та розбиратися із файлами, і навіть не потрібно самостійно редагувати project.yaml.

Достатньо створити конфігураційний файл цілей для цього проєкту, наприклад, `iNote.yaml` із таким змістом:

```
project: iNote
targets:
Alpha:
Beta:
Gamma:
steals:
     – iNote/Alpha/Resources/
    – iNote/Beta/Modules/
       inherit: Alpha
```

Після цього запустити команду `./igen iNote.yaml project.yaml`.

У результаті виконання, скрипт самостійно оновить файли `iNote.yaml`, `project.yaml`, скопіює ресурсні файли із цілі Alpha, а вихідні файли – із цілі Beta. Після запуску xcode generate у проєкт буде додано нову ціль, повністю готову для використання. Згенеровані налаштування цілі Gamma показано на рисунку 8.

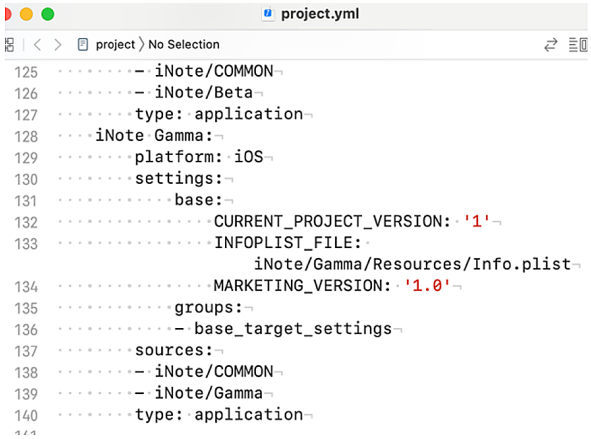

Рис. 8 Список цілей у файлі `project.yaml` після використання `igen`

Отож, використання скрипту `igen` та конфігураційного файлу дозволяє розробнику максимально швидко додавати нові цілі до існуючого проєкту.

Проєкт для ілюстрації розташований у GitHub репозиторії за посиланням [9].

Висновки

Удосконалено підхід із використанням декількох цілей у рамках одного Xcode проєкту шляхом використання XcodeGen для генерації проєктного файлу та самописної утиліти igen для модифікації конфігураційного файлу project.yaml.

Удосконалений підхід має ряд переваг. Використання XcodeGen дозволяє зберігати усі конфігурації проєкту в одному файлі у зрозумілому та сприйнятливому вигляді. Структура файлів та директорій у Xcode буде завжди синхронізовано із структурою на диску, зменшується можливість виникнення конфліктів злиття через одночасні модифікації файлу `.pbxproj`.

Використання утиліти `igen` уможливлює максимально швидке створення нових цілей на основі конфігураційного файлу, що дозволяє максимізувати використання людських ресурсів та грошей, пришвидшити час розробки, а також мінімізувати можливість виникнення людських помилок.

Розроблений проєкт Xcode для ілюстрації описаного підходу, який розміщено у відкритому доступі у GitHub репозиторії.

References

1. iOS : How to create two versions of your app with a single codebase [Online] – Available from: https://anutoshdatta.medium.com/ios-how-to-create-two-versions-of-your-app-with-a-single-codebase-eca0bb75b0fe, last accessed 2022/06/11.
2. 2 iOS Apps, 1 Codebase with XCode Scheme [Online] – Available from: https://testfairy.com/blog/2-ios-apps-1-codebase-with-xcode-scheme/, last accessed: 2022/06/11.
3. Bundle IDs [Online] – Available from: https://developer.apple.com/documentation/appstoreconnectapi/bundle_ids, last accessed: 2022/06/11.
4. XcodeGen [Online] – Available from: https://github.com/yonaskolb/XcodeGen, last accessed: 2022/06/11.
5. BuildSettingsExtractor [Online] – Available from: https://buildsettingextractor.com/, last accessed: 2022/06/11.
6. Example project.yaml [Online] – Available from: https://github.com/gloomikon/MultiTargetAutomatisation/blob/main/iNote/project.yml, last accessed: 2022/06/11.
7. igen [Online] – Available from: https://github.com/gloomikon/MultiTargetAutomatisation/tree/main/igen, last accessed: 2022/06/11.
8. Template Method [Online] – Available from: https://refactoring.guru/design-patterns/template-method, last accessed: 2022/06/11
9. MultiTargetAutomatisation Example Project [Online] – Available from: https://github.com/

gloomikon/MultiTargetAutomatisation, last accessed: 2022/06/11.

Одержано: 18.11.2022

***Про авторів*:**

Журба Микола Андрійович, магістрант другого року навчання Національного Технічного Університету України «КПІ імені Ігоря Сікорського».

Стеценко Інна Вячеславівна,
доктор технічних наук, професор,
професор кафедри інформатики та програмної інженерії НТУУ
“КПІ імені Ігоря Сікорського”.

Кількість наукових публікацій в українських виданнях – понад 100.
Кількість наукових публікацій в зарубіжних виданнях – 14. Індекс Хірша – 2. http://orcid.org/ 0000-0002-4601-0058

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
03056, м. Київ,
проспект Перемоги 37.
Тел.: (044) 236-9651
e-mail: gloomikon@gmail.com