Swift Plugins: ускоряем сборку проекта

Максим Гришутин, руководитель отдела iOS разработки

Ozon Tech 2023

Мой опыт

OZON tech

White Label App
для банков (ВТБ, ОТП,
Абсолют и еще +100)

ROUTE 256

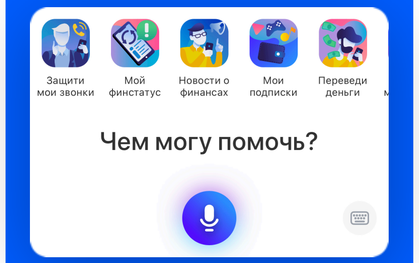
SwiftUI iOS
курсы
в Route 256

iOS App для
селлеров в Ozon

OZON tech

Публичные
выступления

OpenSource
Комьюнити



Помощник
в Тинькофф

Prefire

Генерация
Snapshot tests

Telegram

ozon tech

Prefire iOS

Выкладываю свои проекты
и мысли о современной iOS
разработке

prefire_ios

О чем мы поговорим

- Проблема скорости сборки
- Что такое **Swift Plugins**?
- Как работать с плагинами?
- Сравнение с **Build Phase Script**
- Результат использования плагинов

Проблема скорости сборки

Вводные для контекста

- Модуляризация (**Swift Package Manager**)
- Больше **300K** строк
- Отсутствие **Objective C**
- Множество помощников

Линтеры | Генераторы

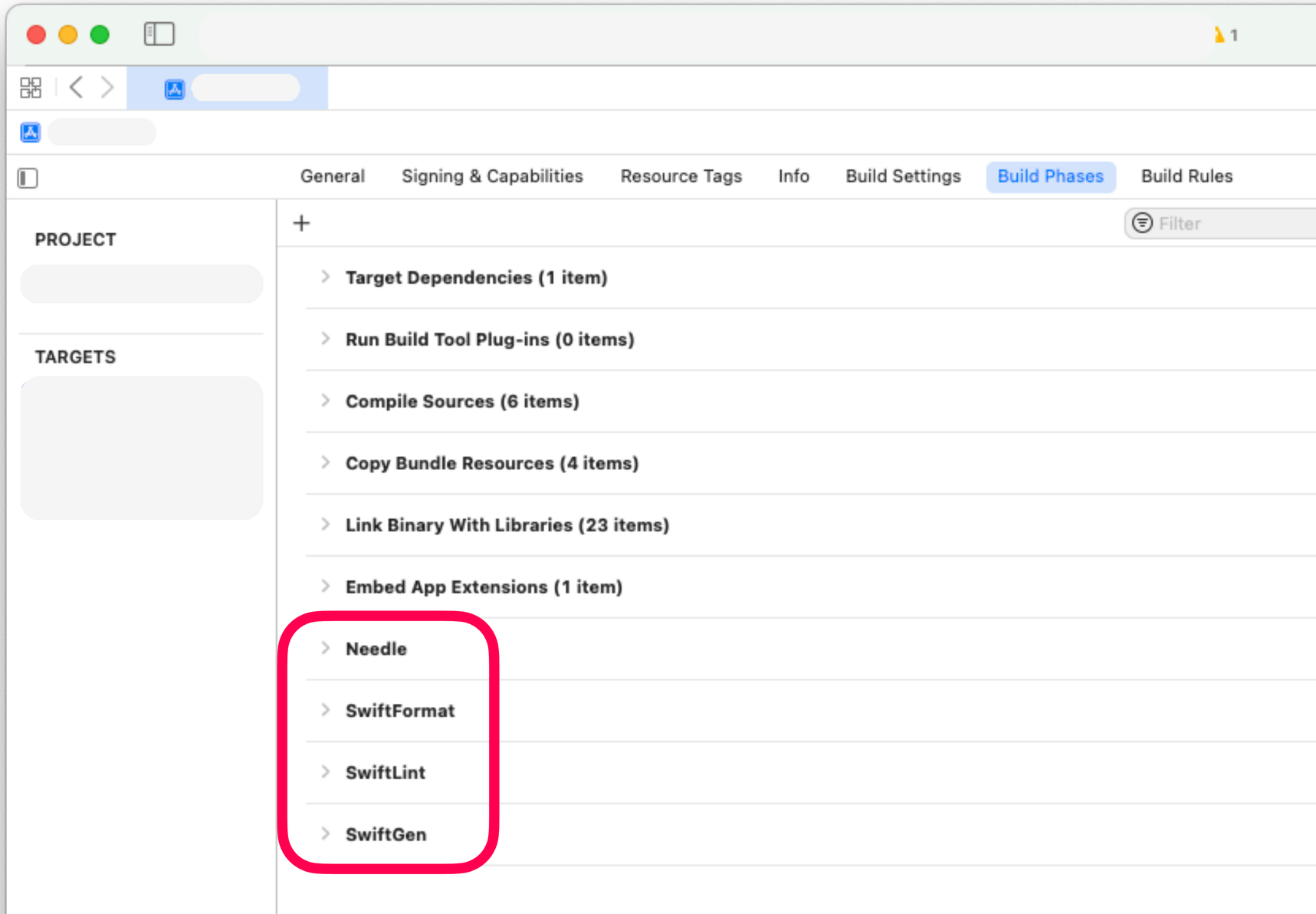
Форматтеры | Кастомные скрипты

Начнем с истории

Из чего состояла сборка

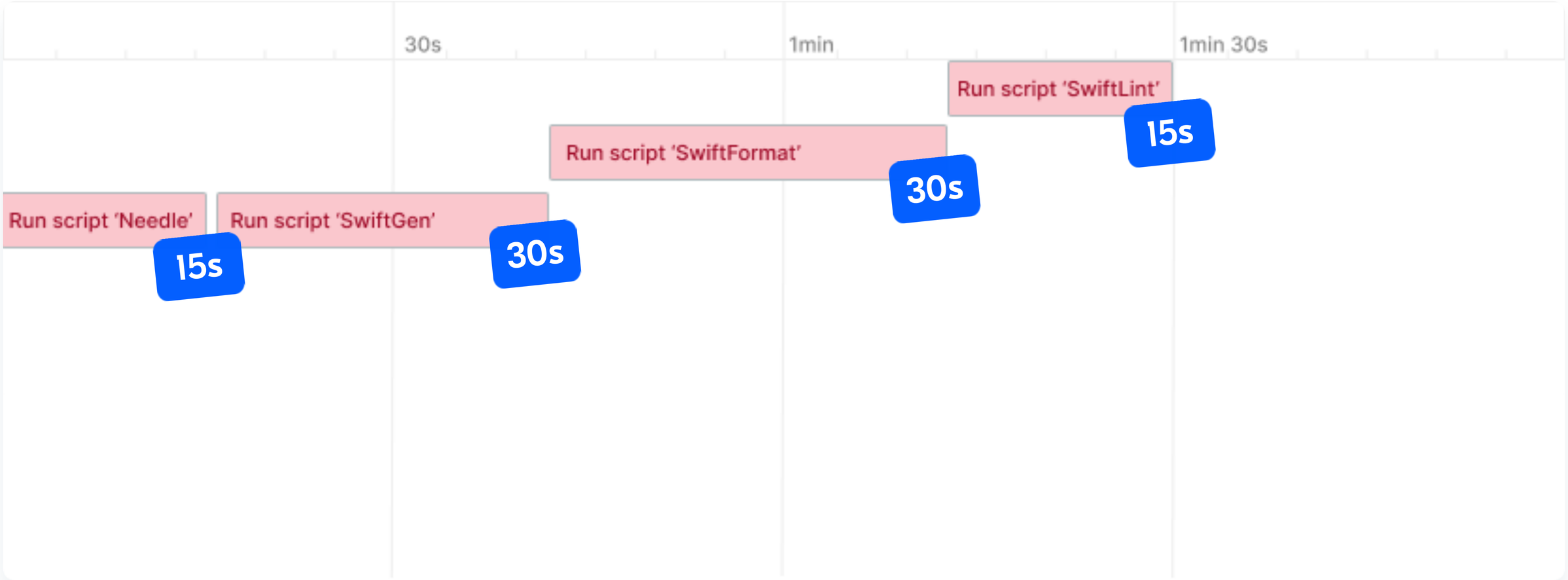
Build Phase Script

- Needle (DI)
- SwiftFormat
- SwiftLint
- SwiftGen



Как это выглядит

Build Timeline



Правило ☕

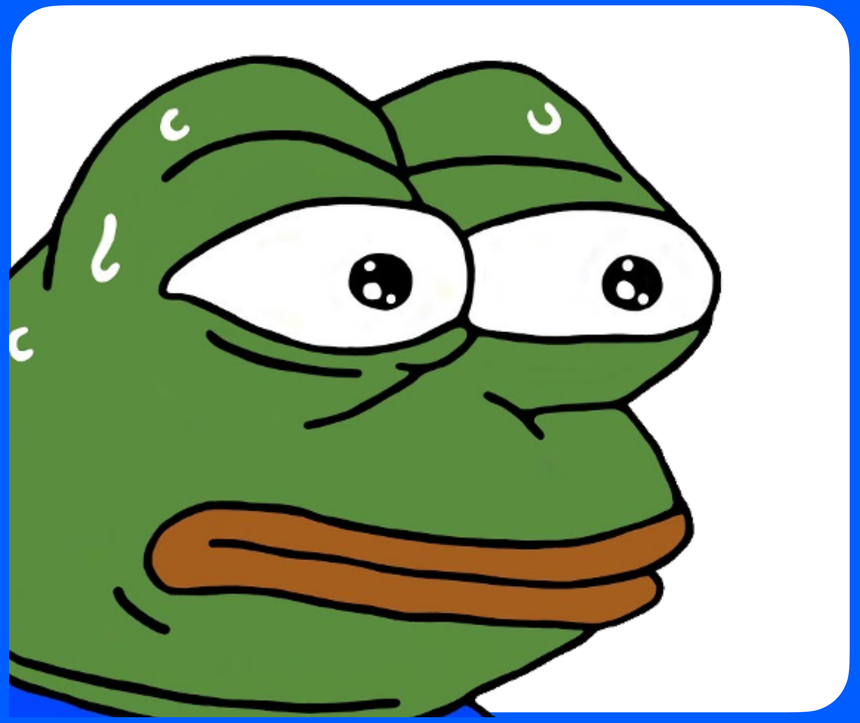
- Разработчик не ждет более **10-15с**
- Иначе он отвлекается за кофе ☕

```
if developer.wait() ⩾ 15.0 {
    print("I need coffee ☕")
} else {
    print("I have to keep working 😢")
}
```

Цель: ☕ 5s

Текущее ~1m 30s

В 18 раз медленнее



Когда это происходит

Почти худший случай

Первый билд проекта

1.

Пустой кеш

2.

Изменение корневого модуля

3.

Как можно оптимизировать?

OZON tech

Вынести все на CI

1.

Архитектурно решить проблему

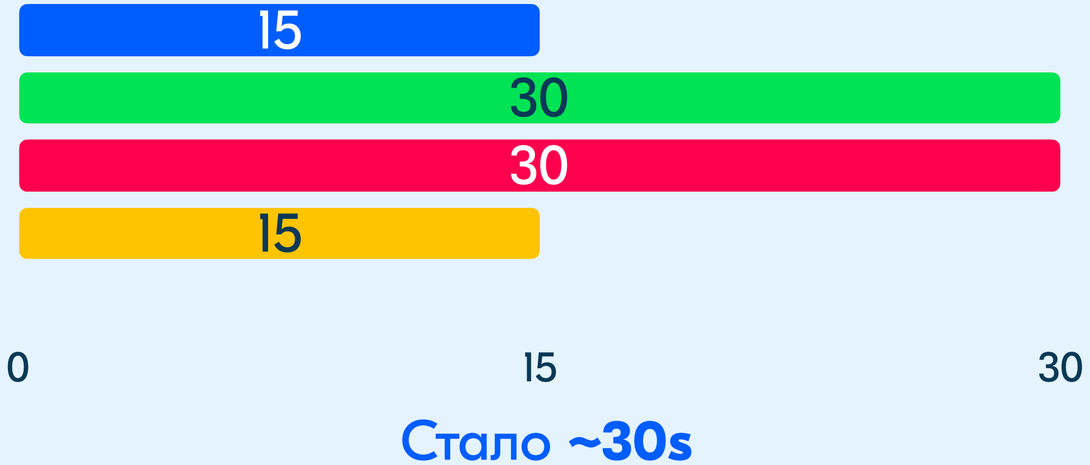
2.

Как можно оптимизировать?

В теории

Параллелизация

Было **~1m 30s**



Стало **~30s**

Цель: ☕ 5s

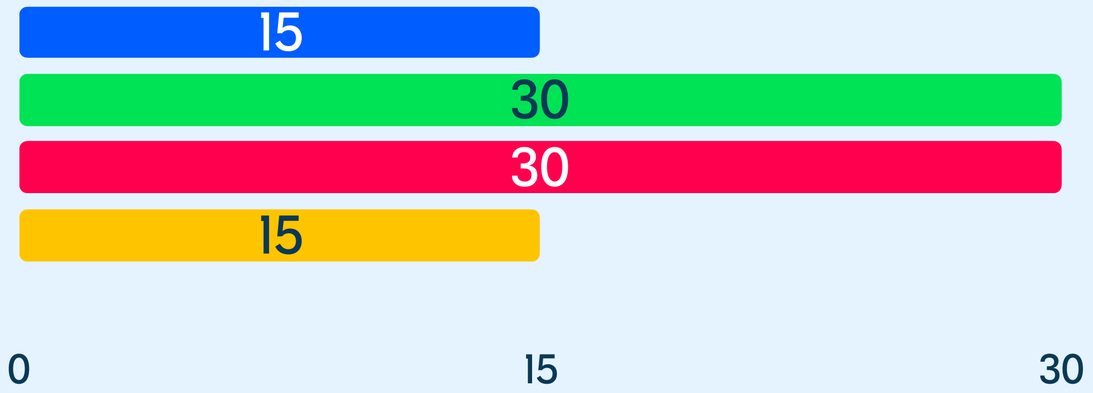
Текущее ~30s

В 6 раз медленнее

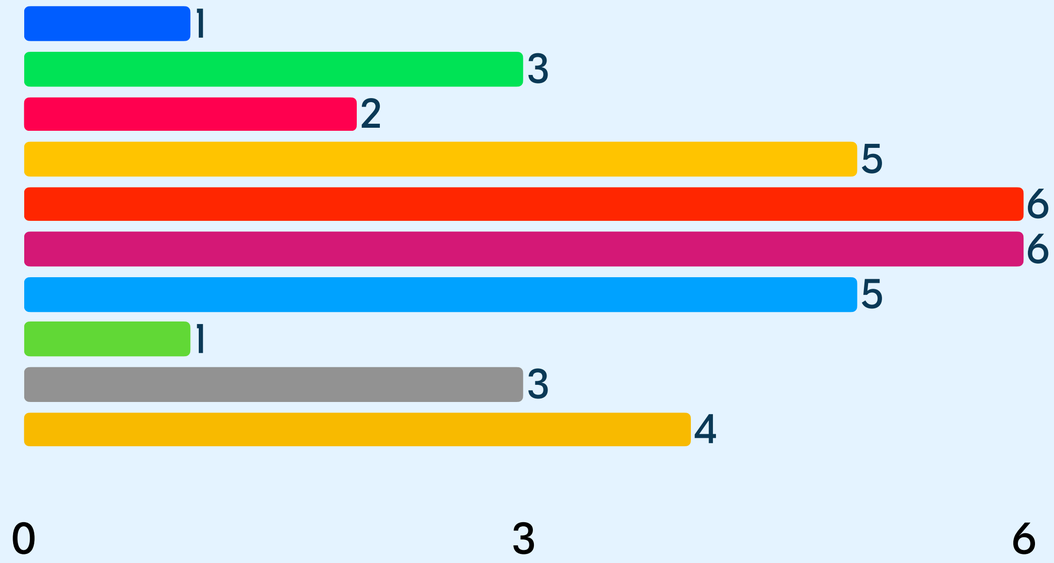
Как можно оптимизировать?

В теории

Отдельный вызов на модуль 10 модулей



Было ~30s



Стало ~6s

Цель: ☕ 5s

Текущее ~6s

Сойдёт

Как это сделать

Технологии

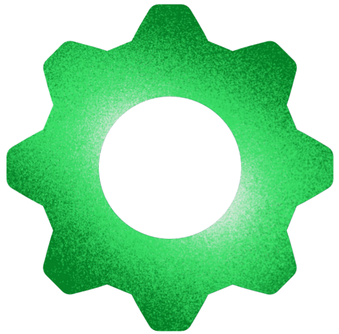
Build Phase Script + Bash

+ Боль 😢

1.

Swift Plugin

2.



Что такое **Swift Plugins**?

Swift Plugins

ozontech

- **Swift** скрипт
- Релиз в **2022**
- Нативный от **Apple**
- **API** от **SPM**

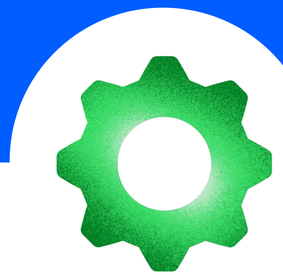
Xcode

Terminal

Swift Package Manager

База Swift Plugins

Swift скрипт



- Запускается **во время сборки**
- Запускается **вручную**

Типы плагинов

OZON tech

Command plugin

Запускается вручную

Build tool plugin

Запускается до сборки

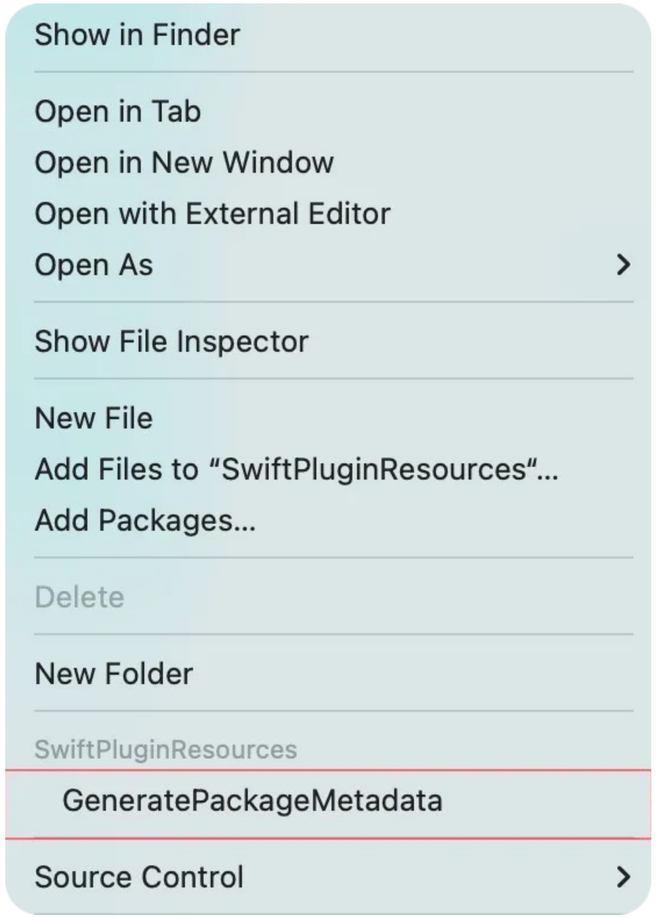
Как запускать плагины

Command plugin

Терминал

```
swift package
plugin Prefire
```

Xcode



Примеры плагинов

OZON tech

SwiftLint

Проверка стиля кода

SwiftGen

Генерация кода для ресурсов

Sourcery

Генератор кода по шаблону

Prefire

Генерация Snapshot тестов

Вынести код для переиспользования другими

Немного о проблемах

Проблемы

OZON tech

Новый инструмент

Есть баги

1.

Не хватает параметров для запуска

2.

BuildPlugin не может изменять файлы проекта

Только DeriveData

3.

Как работать с плагинами?

Как работать с плагинами?

- Находите/создаёте плагин
- Подключаете плагин

Репозиторий списка плагинов

awesome-swift-plugins

Создание плагина

- Выбираем/пишем скрипт
- Создаем репозиторий с **SPM**
- Создаем плагин

Выбираем/пишем скрипт

Категории

Форматирование 1.

Тестирование 2.

Генерация 3.

И еще много чего...

Выберем скрипт

- Command Plugin
- Build Tool Plug-in

Sourcery

Генератор кода по шаблону

Выберем скрипт

- Command Plugin
- **Build Tool Plug-in** ⭐

Sourcery

Генератор кода по шаблону

Создаем плагин

```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```

Создаем плагин

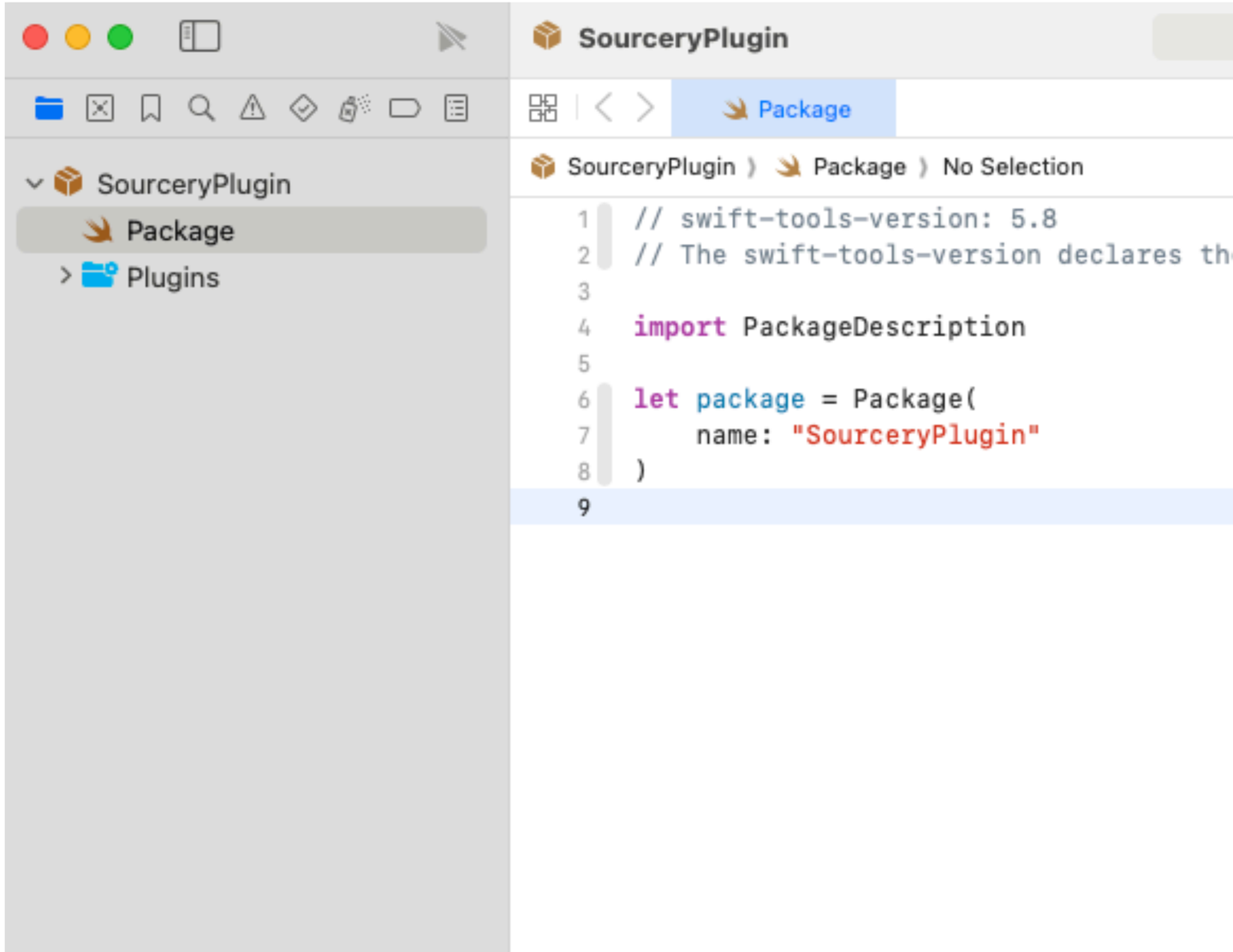
```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```

Создаем плагин

```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```

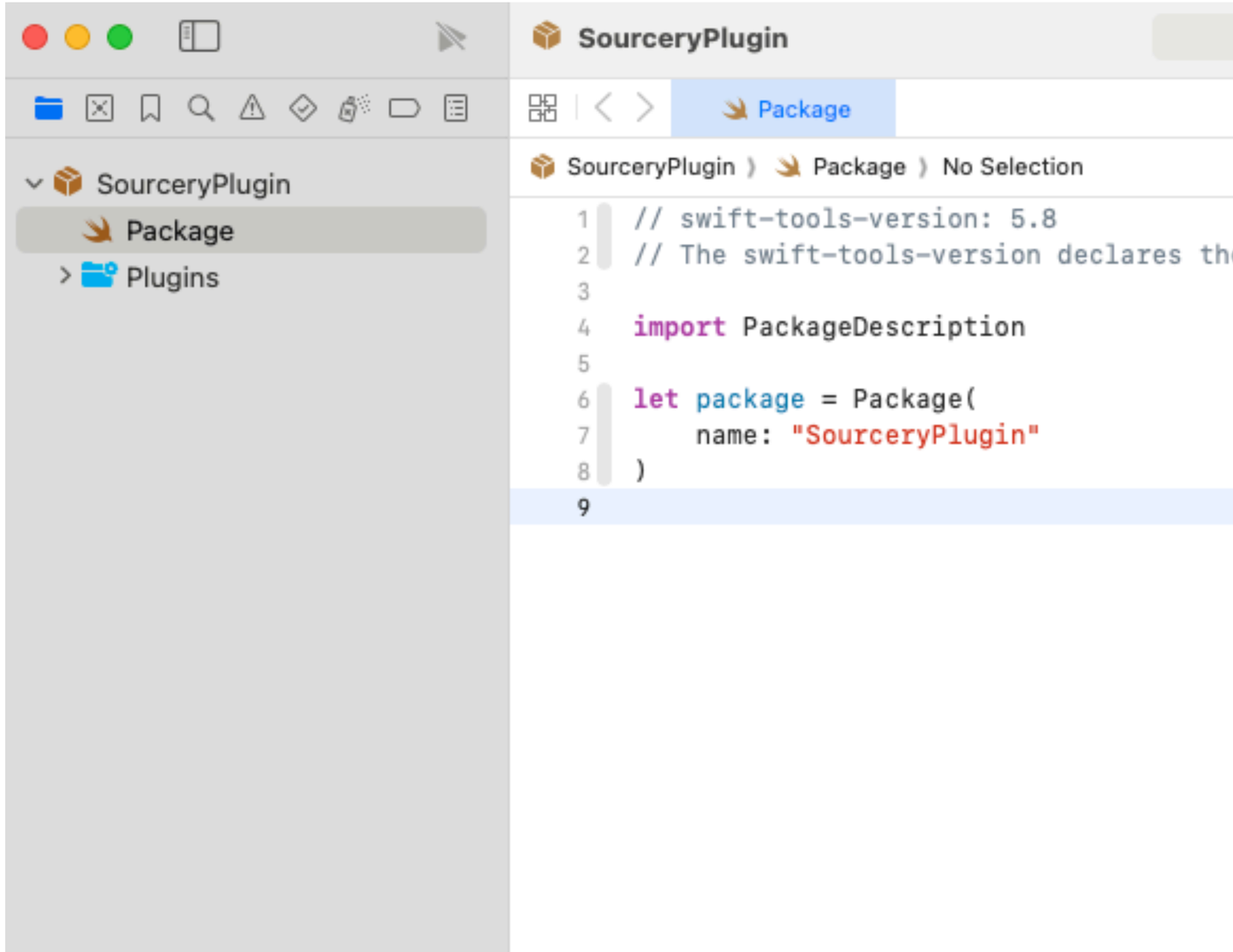
Создаем плагин

```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```



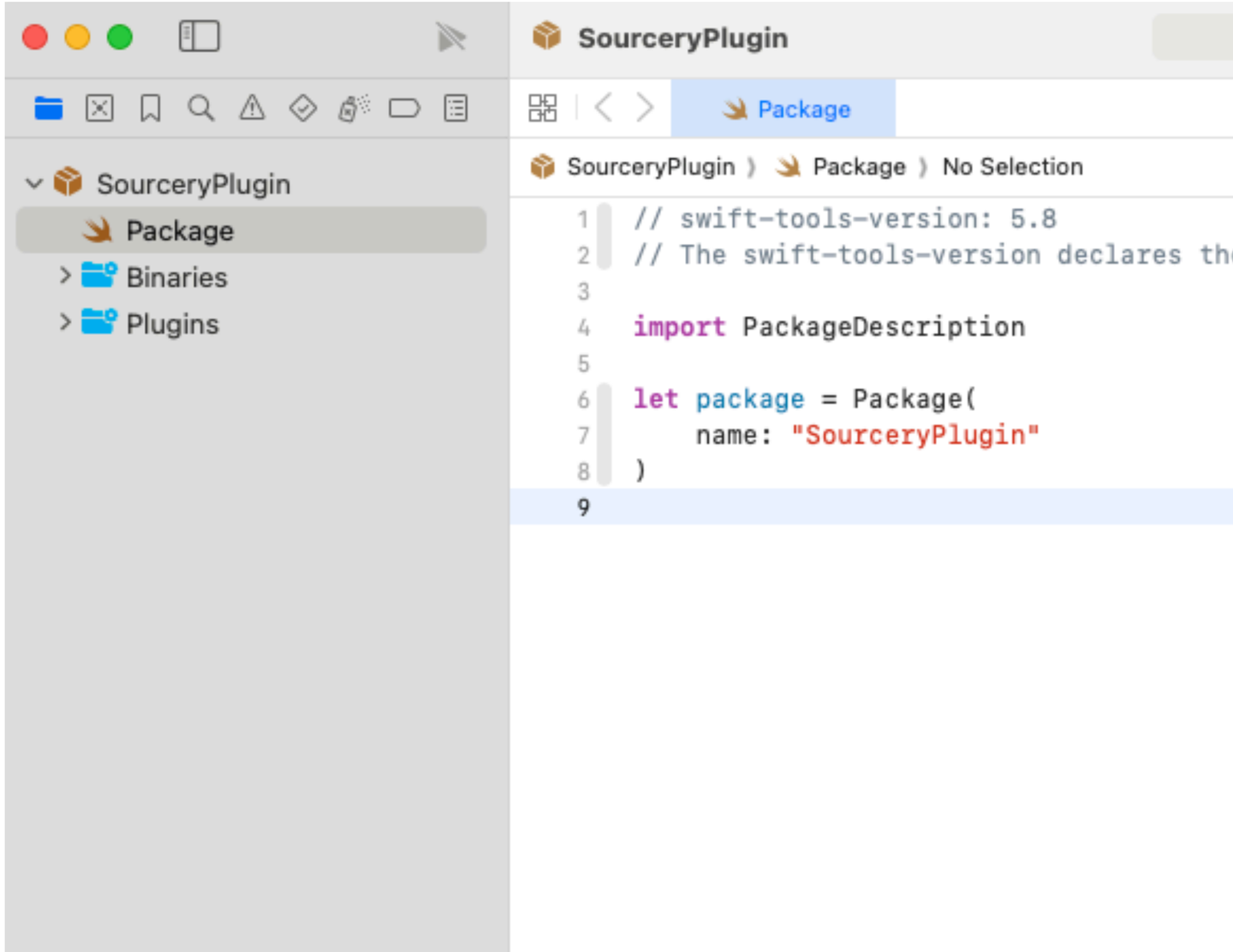
Создаем плагин

```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```



Создаем плагин

```
// Создаем и переходим в папку
mkdir SourceryPlugin; cd SourceryPlugin

// Создаем пустой пакет
swift package init --type empty

// Создаем папку для Sourcery
mkdir Binaries

// Создаем папку для Скрипта
mkdir Plugins
```



Создаем плагин

Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin"
)
```

Создаем плагин

Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin",
    platforms: [.iOS(.v13)],
    products: [
        .plugin(
            name: "SourceryPlugin",
            targets: ["SourceryPlugin"]
        )
    ],
    targets: [
        .plugin(
            name: "SourceryPlugin",
            capability: .buildTool(),
            dependencies: ["Sourcery"]
        ),
        .binaryTarget(
            name: "Sourcery",
            path: "Binaries/Sourcery.artifactbundle"
        )
    ]
)
```

Создаем плагин

Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin",
    platforms: [.iOS(.v13)],
    products: [
        .plugin(
            name: "SourceryPlugin",
            targets: ["SourceryPlugin"]
        )
    ],
    targets: [
        .plugin(
            name: "SourceryPlugin",
            capability: .buildTool(),
            dependencies: ["Sourcery"]
        ),
        .binaryTarget(
            name: "Sourcery",
            path: "Binaries/Sourcery.artifactbundle"
        )
    ]
)
```

Создаем плагин

Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin",
    platforms: [.iOS(.v13)],
    products: [
        .plugin(
            name: "SourceryPlugin",
            targets: ["SourceryPlugin"]
        )
    ],
    targets: [
        .plugin(
            name: "SourceryPlugin",
            capability: .buildTool(),
            dependencies: ["Sourcery"]
        ),
        .binaryTarget(
            name: "Sourcery",
            path: "Binaries/Sourcery.artifactbundle"
        )
    ]
)
```

Создаем плагин

Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin",
    platforms: [.iOS(.v13)],
    products: [
        .plugin(
            name: "SourceryPlugin",
            targets: ["SourceryPlugin"]
        )
    ],
    targets: [
        .plugin(
            name: "SourceryPlugin",
            capability: .buildTool(),
            dependencies: ["Sourcery"]
        ),
        .binaryTarget(
            name: "Sourcery",
            path: "Binaries/Sourcery.artifactbundle"
        )
    ]
)
```

Создаем плагин

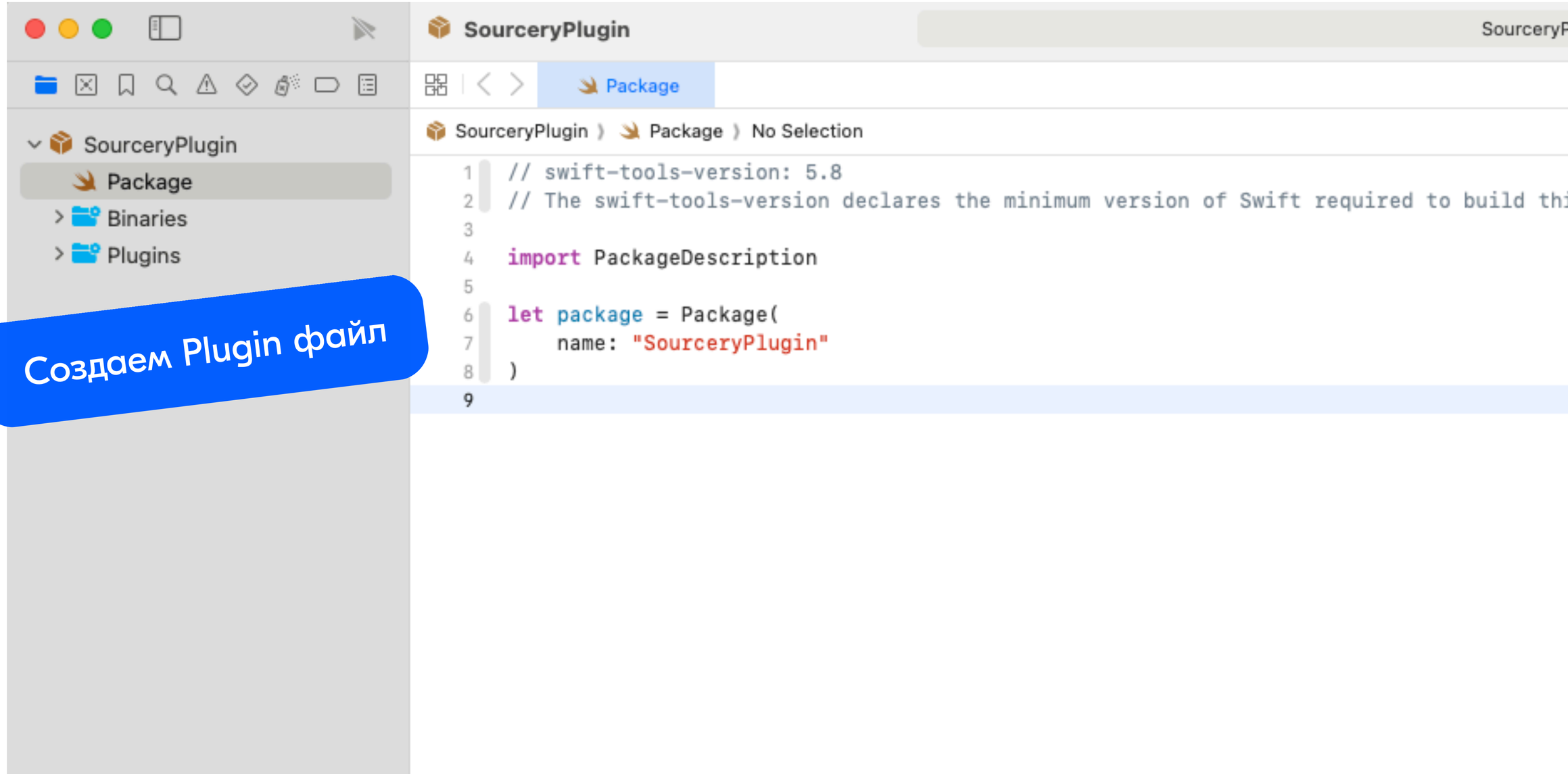
Package.swift

```
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "SourceryPlugin",
    platforms: [.iOS(.v13)],
    products: [
        .plugin(
            name: "SourceryPlugin",
            targets: ["SourceryPlugin"]
        )
    ],
    targets: [
        .plugin(
            name: "SourceryPlugin",
            capability: .buildTool(),
            dependencies: ["Sourcery"]
        ),
        .binaryTarget(
            name: "Sourcery",
            path: "Binaries/Sourcery.artifactbundle"
        )
    ]
)
```

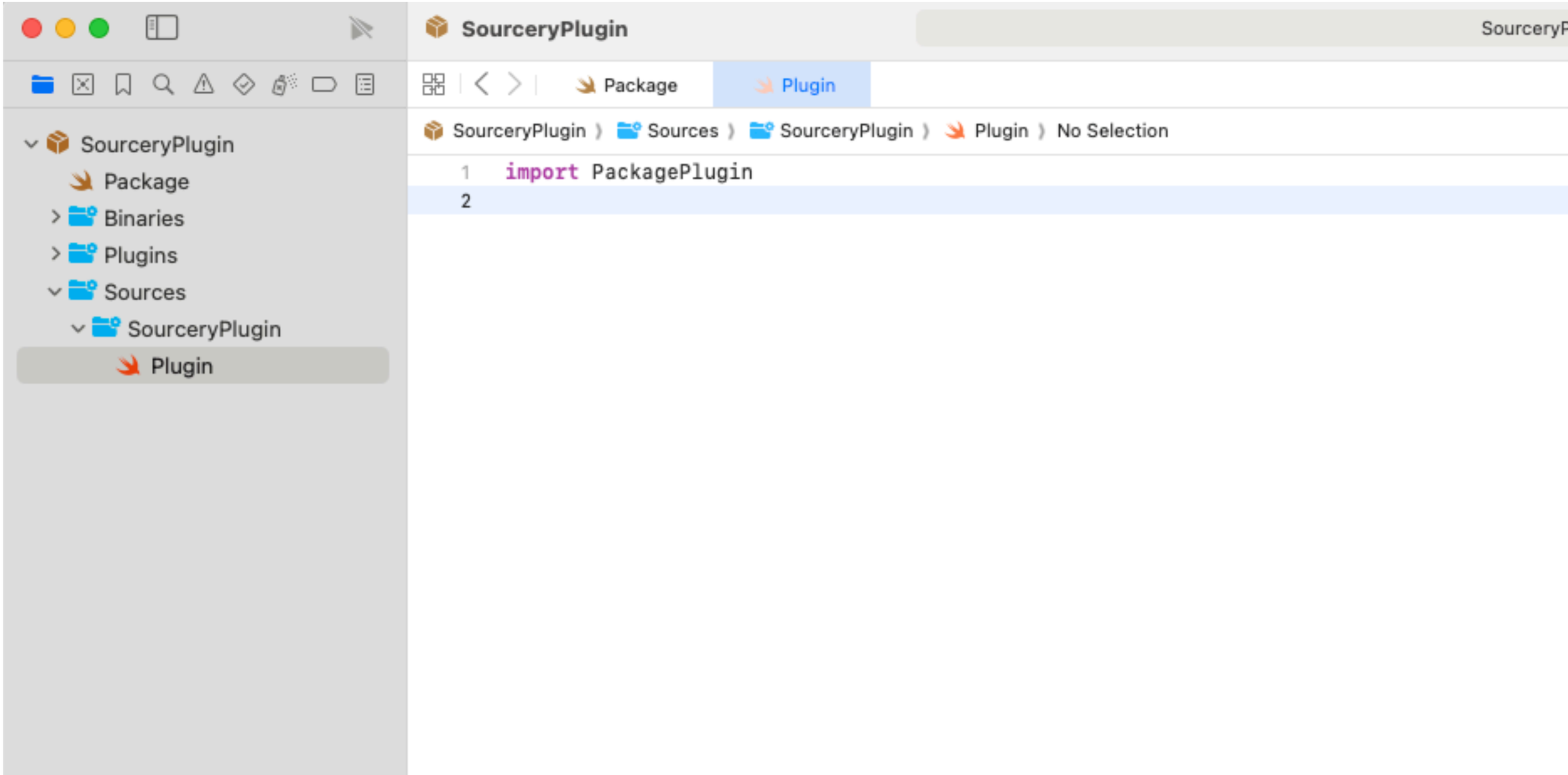
Пакет готов

Осталось добавить скрипт

Создаем плагин



Создаем плагин



Создаем плагин

Plugin.swift

```
import PackagePlugin
```

Создаем плагин

Plugin.swift

```
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
```

Создаем плагин

Plugin.swift

```
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")
```

Создаем плагин

ozon tech

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

Создаем плагин

ozon tech

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

Создаем плагин

Plugin.swift

```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

Создаем плагин

Plugin.swift

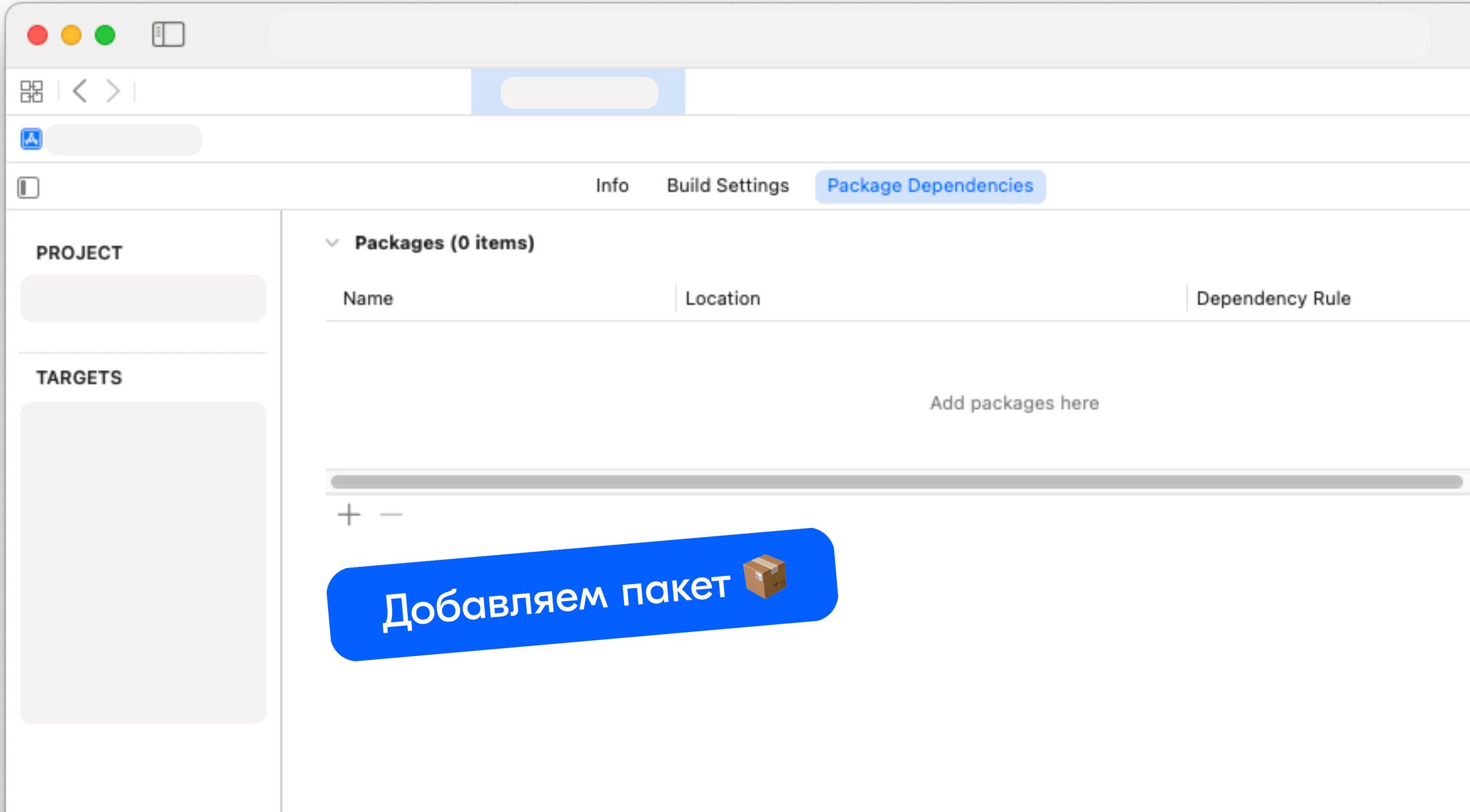
```swift
import PackagePlugin

@main
struct SourceryPlugin: BuildToolPlugin {
    func createBuildCommands(context: PluginContext, target: Target) throws -> [Command] {
        // Файл конфигурации для Sourcery
        let sourceryConfigFile = target.directory.appending("Configs/sourcery.yml")

        // Директория, где будут генерироваться файлы
        let generatedSourcesDirectory = context.pluginWorkDirectory

        // Бинарник Sourcery
        let executable = try context.tool(named: "Sourcery").path

        // Директория с шаблоном
        let templatesDirectory = executable.removingLastComponent().removingLastComponent().appending("templates")

        // Создаем файл в нужной директории
        try FileManager.default.createDirectory(atPath: generatedSourcesDirectory.string, withIntermediateDirectories: true)

        return [
            .prebuildCommand(
                displayName: "Running Sourcery",
                executable: executable,
                arguments: [ "--config", "\(sourceryConfigFile)"],
                environment: [
                    "TARGET_DIR": "\(target.directory)",
                    "DERIVED_DATA_DIR": "\(generatedSourcesDirectory)",
                    "TEMPLATES_DIR": "\(templatesDirectory)"
                ],
                outputFilesDirectory: generatedSourcesDirectory
            )
```

SourceryPlugin

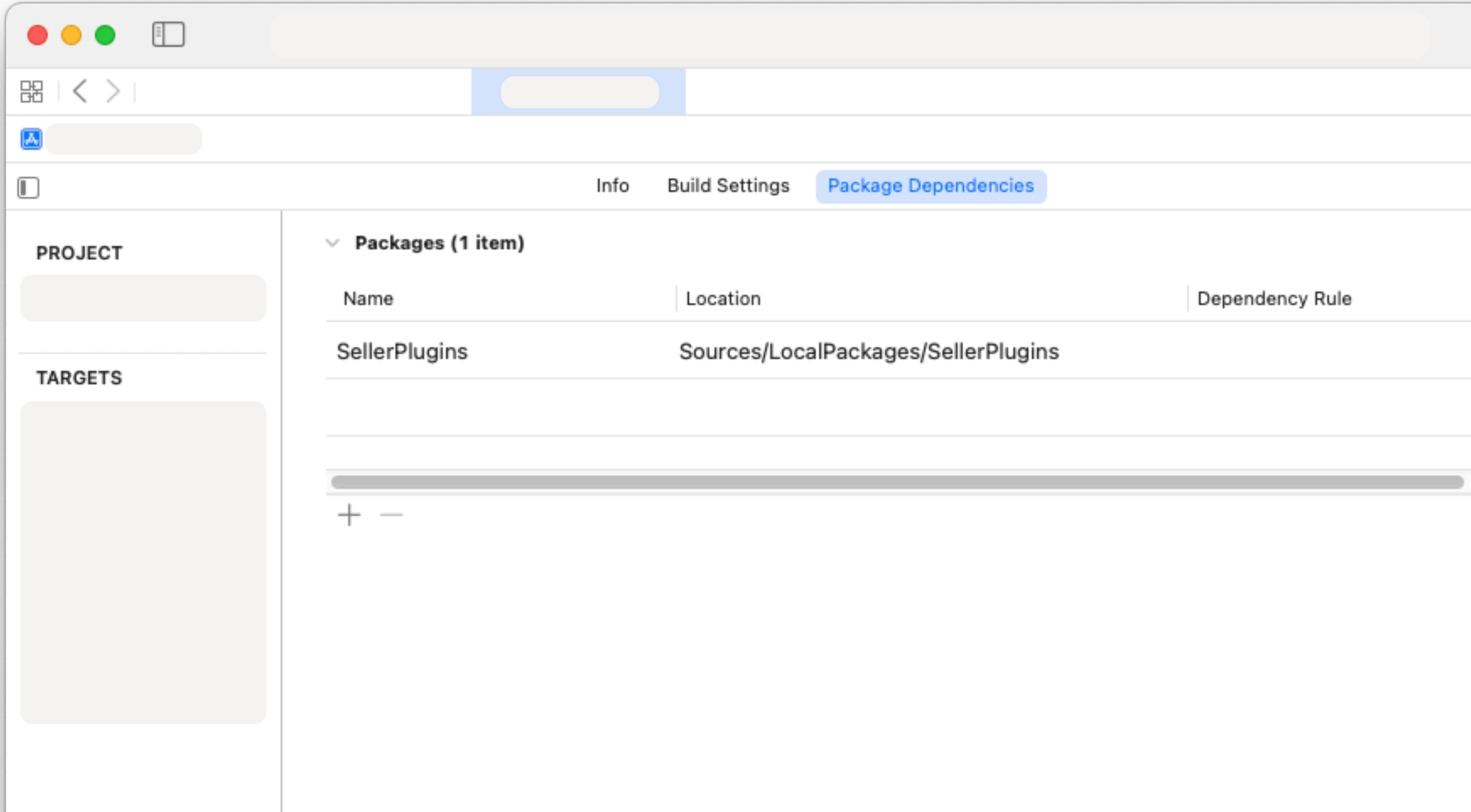
Github

Sourcery Plugin

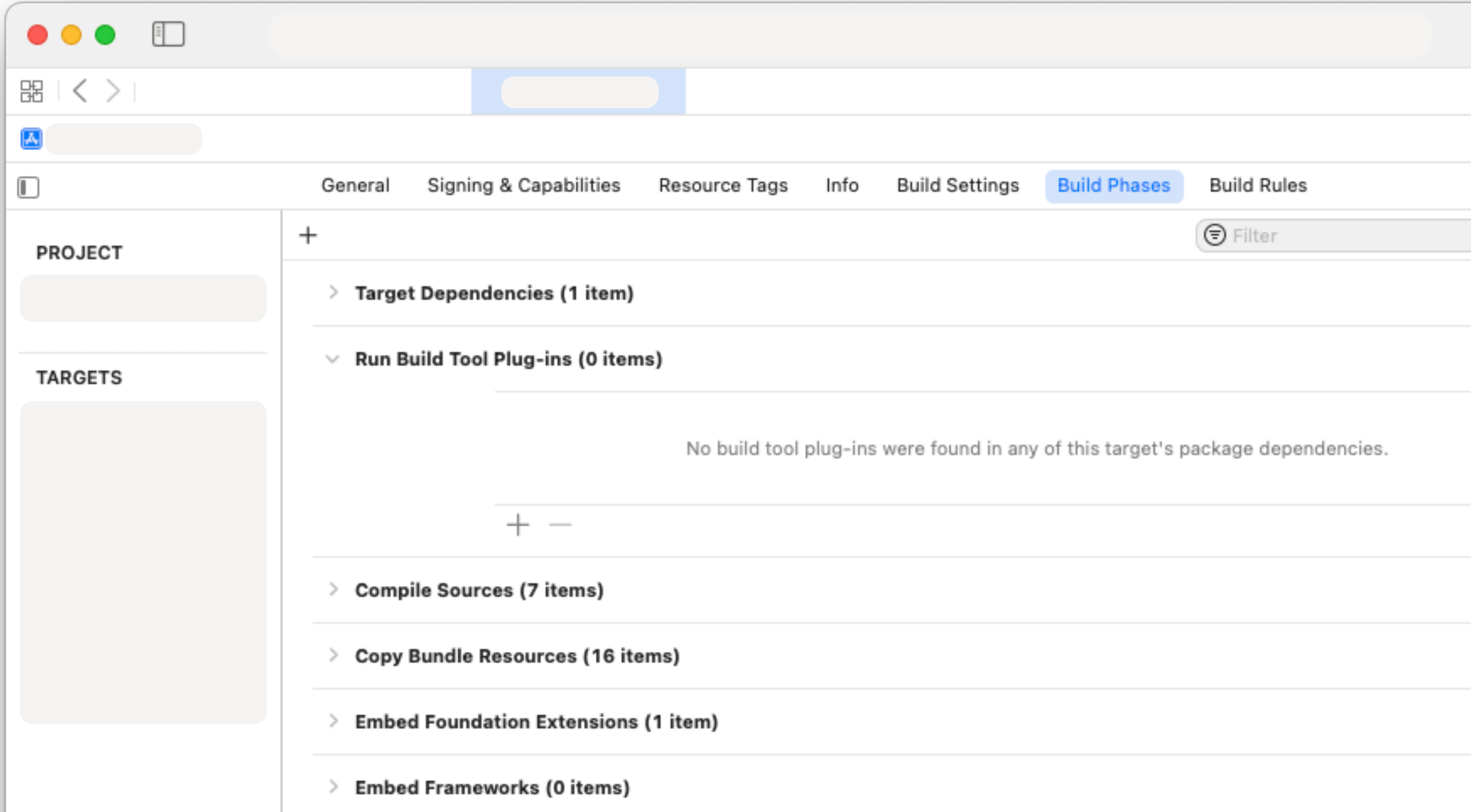
Подключаем


Info
Build Settings
Package Dependencies
PROJECT
TARGETS
Packages (0 items)
Name
Location
Dependency Rule
Add packages here
Добавляем пакет 📦

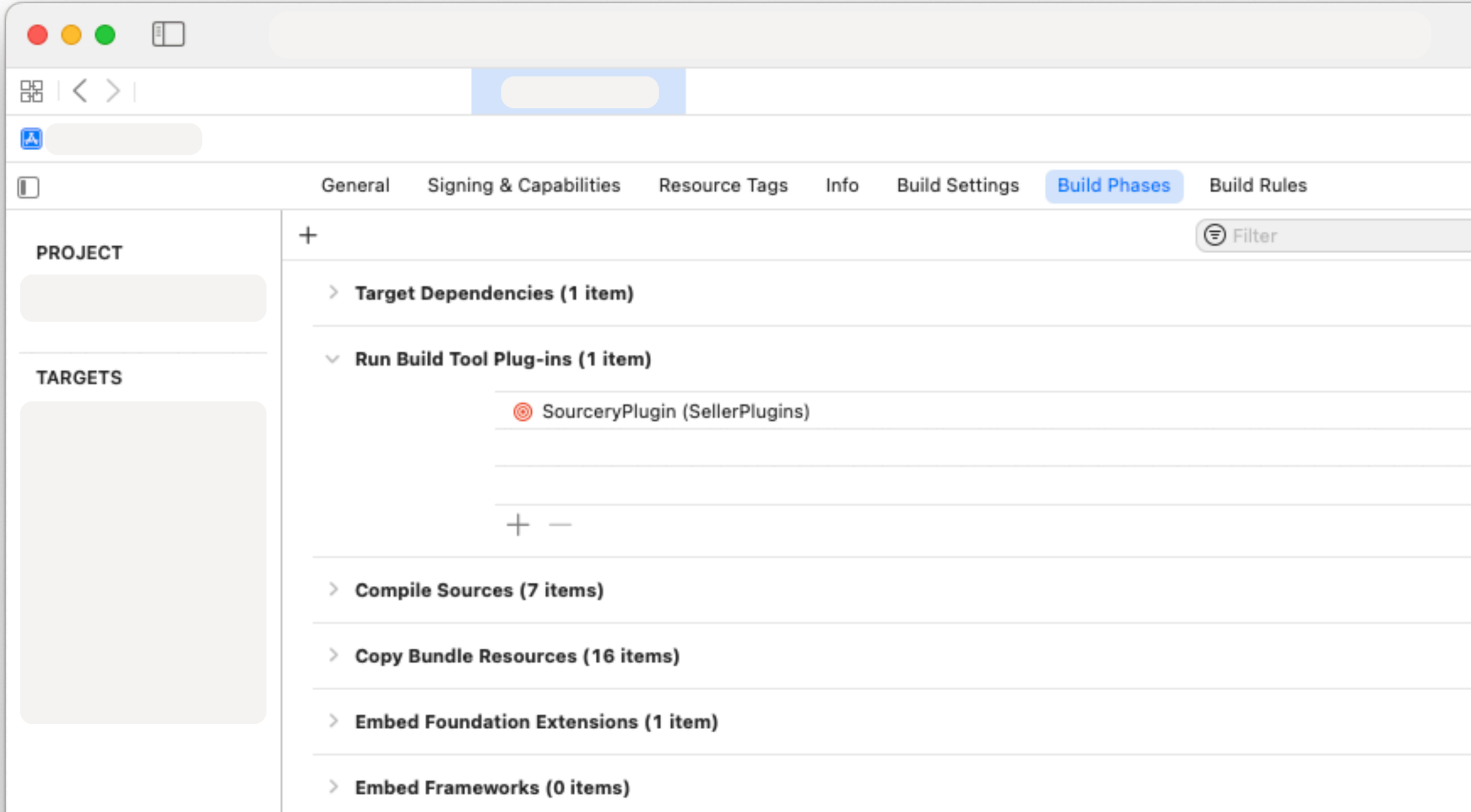
Подключаем


Info
Build Settings
Package Dependencies
PROJECT
TARGETS
Packages (1 item)
Name
Location
Dependency Rule
SellerPlugins
Sources/LocalPackages/SellerPlugins

Подключаем


General
Signing & Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
TARGETS
Filter
Target Dependencies (1 item)
Run Build Tool Plug-ins (0 items)
No build tool plug-ins were found in any of this target's package dependencies.
Compile Sources (7 items)
Copy Bundle Resources (16 items)
Embed Foundation Extensions (1 item)
Embed Frameworks (0 items)

Подключаем


General
Signing & Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
TARGETS
Filter
Target Dependencies (1 item)
Run Build Tool Plug-ins (1 item)
SourceryPlugin (SellerPlugins)
Compile Sources (7 items)
Copy Bundle Resources (16 items)
Embed Foundation Extensions (1 item)
Embed Frameworks (0 items)

Сравнение с **Build Phase Script**

Сравним

Swift Plugins	Build Phase Script
	
	
	
	

- Простота подключения
- Шаринг кода
- Поддержка комьюнити
- Скорость

Простота подключения

На примере SwiftLint

Swift Plugins

- Установка в **Xcode**

⚙️ Build Phase Script

- Установка через **Homebrew**

⚠️ Могут быть разные версии библиотек

Сравним

Swift Plugins	Build Phase Script
👍	👎

- Простота подключения
- Шаринг кода
- Поддержка комьюнити
- Скорость

Шаринг кода

OZON tech

Swift Plugins

- Как обычный проект

Github + SPM

Build Phase Script

- Нет специализированных инструментов

Сравним

Swift Plugins	Build Phase Script
👍	👎
👍	👎

- Простота подключения
- Шаринг кода
- Поддержка комьюнити
- Скорость

Поддержка комьюнити

OZON tech

Swift Plugins

- Недостаточно популярен

⚙️ Build Phase Script

- Все используют

Единственный до выхода Swift Plugins

Сравним

Swift Plugins	Build Phase Script
👍	👎
👍	👎
👎	👍

- Простота подключения
- Шаринг кода
- Поддержка комьюнити
- Скорость

Скорость

OZON tech

Swift Plugins

- Дает параллелизацию 👍
- Компилируется 👎

~0.3s

⚙️ Build Phase Script

- Не дает параллелизации 👎
- Не компилируется 👍

Сравним

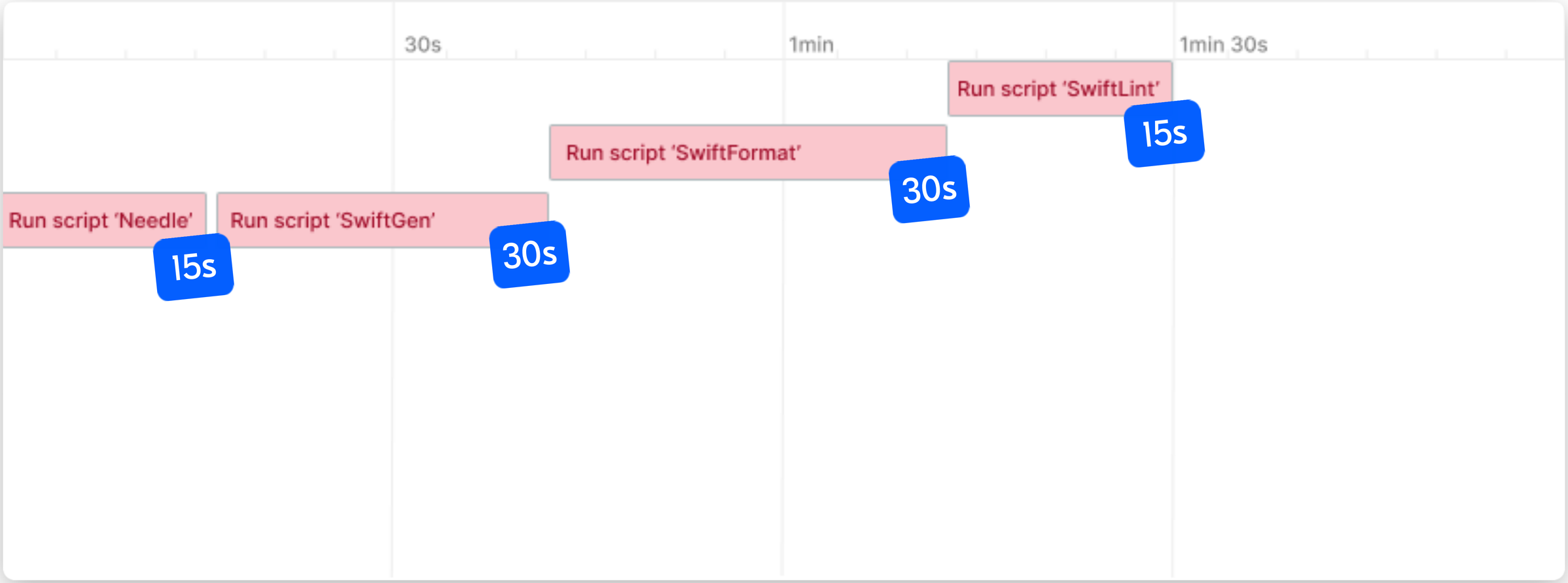
Swift Plugins	Build Phase Script
👍	👎
👍	👎
👎	👍
👍	👍

- Простота подключения
- Шаринг кода
- Поддержка комьюнити
- Скорость

Результат использования плагинов

Как это было

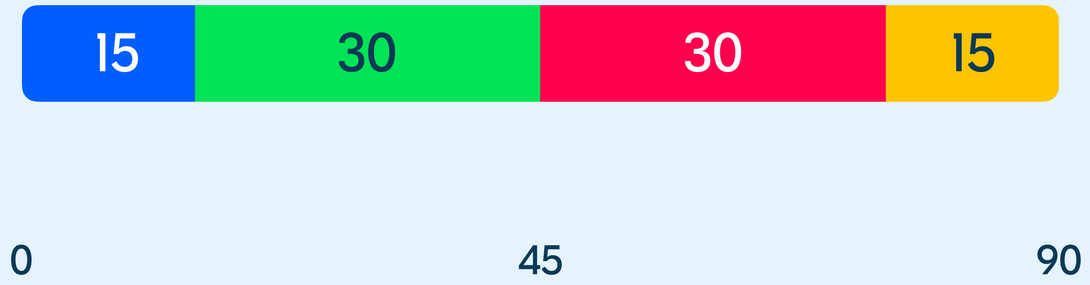
Build Timeline



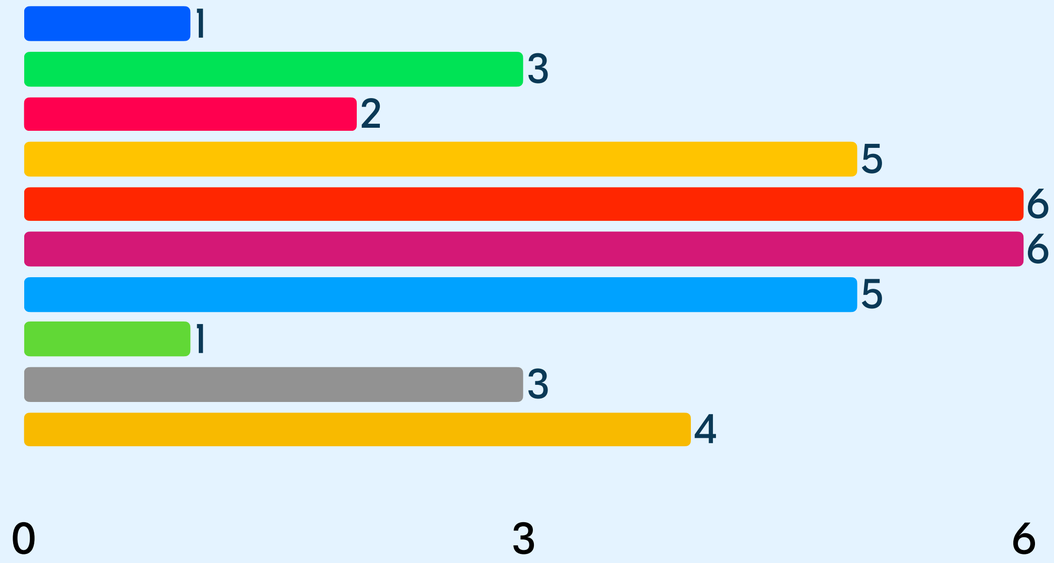
Как можно оптимизировать?

В теории

- Параллелизация
- Отдельный вызов на модуль
- Плагины



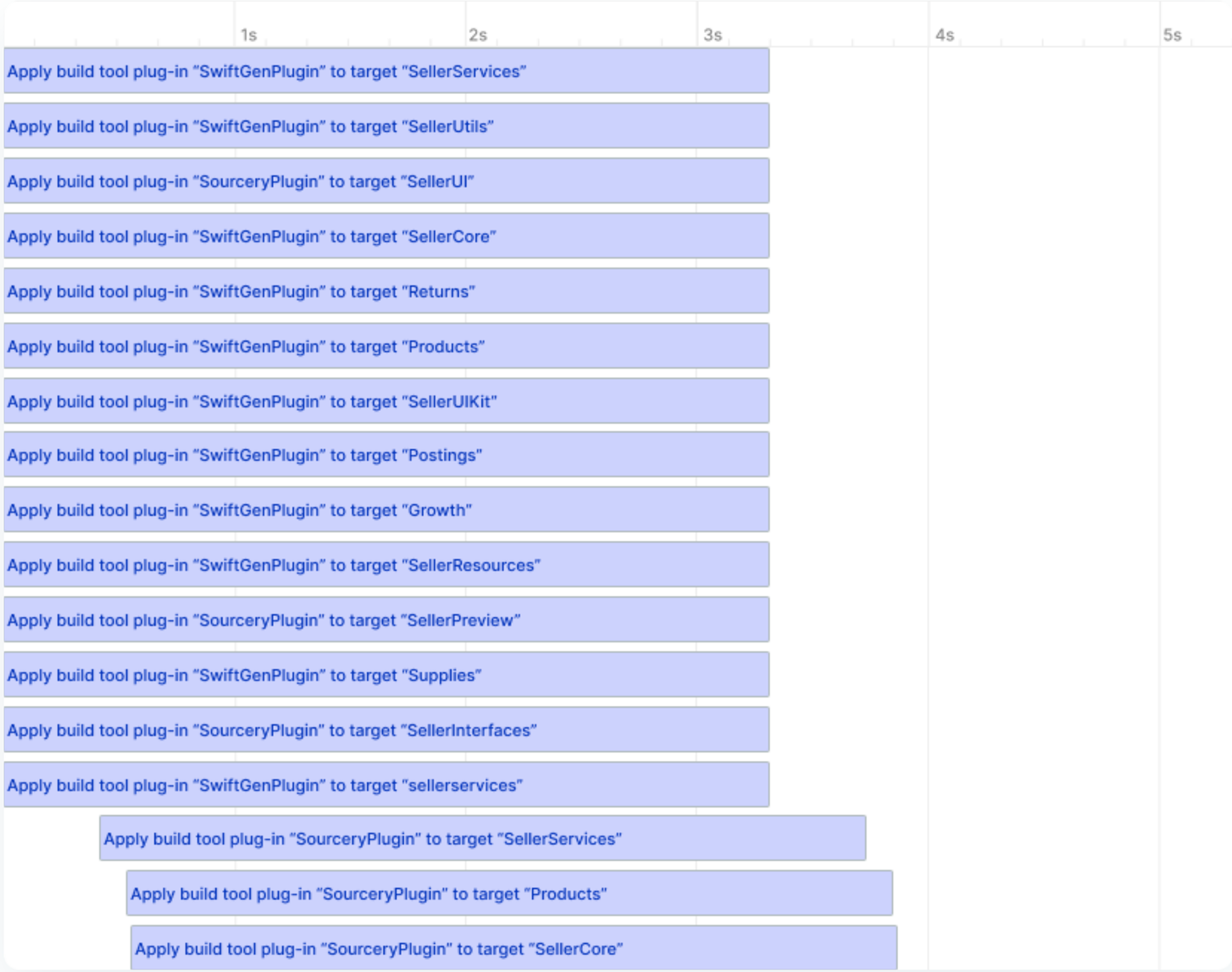
Было **~1m 30s**



Стало **~6s**

Получаем результат

- **Параллелизация**



ozon tech

Swift Plugins: Ускоряем сборку проекта

Максим Гришутин

prefire_ios

maxgrishutin