技术选型文档

环境：MACOS、IOS 13.0+、Xcode12.0+

语言：Swift

框架：SwiftUI、ARKit、RealityKit、StoreKit、CoreData、SwiftyJSON、SDWebImageSwiftUI

一、前端框架

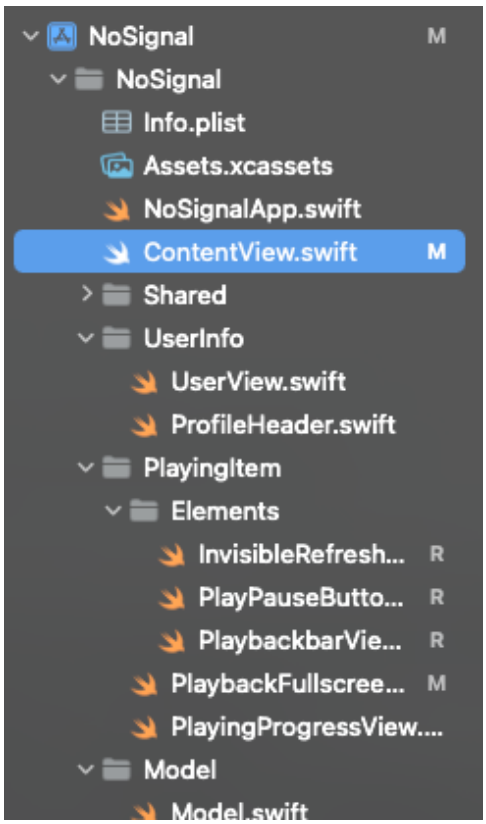
1.1 Swift && SwiftUI

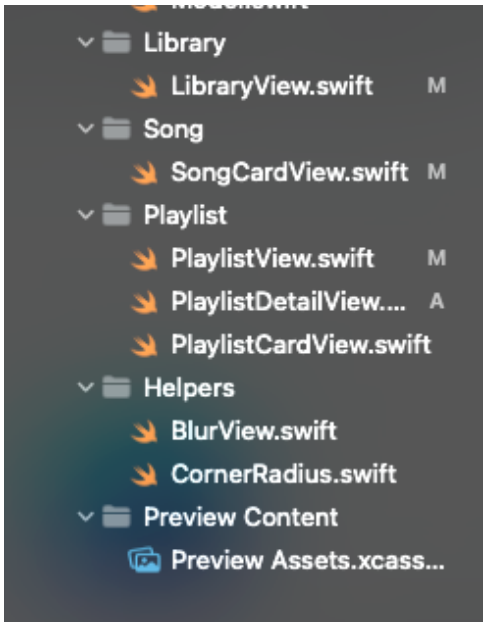
为什么用 Swift

> The powerful programming language that is also easy to learn.
>
> 一种强大但极易学习的编程语言
>
> Swift 是一种强大直观的编程语言，适用于 macOS、iOS、watchOS 和 Apple tvOS 等。编写 Swift 代码的过程充满了乐趣和互动。Swift 语法简洁，但表现力强，更包含了开发者喜爱的现代功能。Swift 代码从设计上保证安全，同时还能开发出运行快如闪电的软件。

Swift 开源且易学易用，语法简洁，标准库完备，同时还可以与 Objective- C 共存，支持更现代的面向对象语法、甚至函数式编程。具有垃圾回收、内存自动管理，尤其是使用更高性能的 Apple LLVM 编译器，使其更加高效。结合 StoryBoard 或 SwiftUI 可以快速上手 APP 开发。

总之，使用 Swift 的好处是说不尽的。





为什么用 SwiftUI

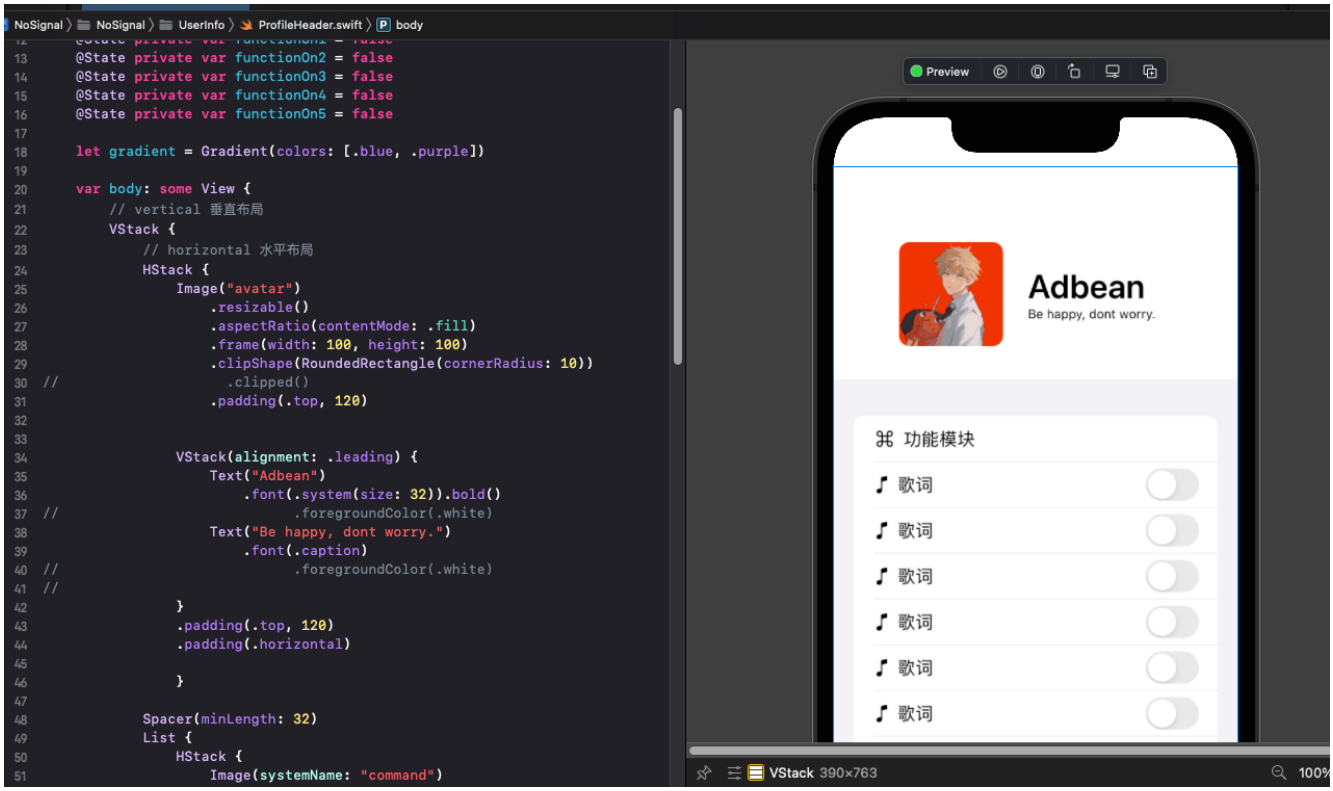
> SwiftUI is a user interface toolkit that lets us design apps in a **declarative way**.
>
> SwiftUI 是一种**描述式**的构建 UI 的方式。

项目一开始我们就决定弃用传统的 Objective C 纯代码布局，因为其存在无法实时渲染页面的明显缺点，界面的一点更改就需要重新编译整个项目和重新渲染，非常耗时。于是我们转向了 StoryBoard、SwiftUI 或者 Flutter，最终我们决定使用较为新颖的 SwiftUI，我们认为 StoryBoard

虽然容易上手并且易于布局，但由于每个组件和页面都需要使用不同的 Actions 来沟通不利于模块化修改，并且不同页面很难继承或者重用。

而 SwiftUI 虽然新颖但跨平台和移植性远不如 Flutter，但因为项目时间原因，Flutter 的上手难度要比 SwiftUI 略高些，所以经过几天的使用，我们最终选择了同样是纯代码布局、并且可以实时渲染检查页面更改，用更精简的代码便可以实现页面布局、动画和组件的 SwiftUI。

并且 SwiftUI 具有响应式布局的好处，利用 HStack、VStack、ZStack 可以快速对页面进行布局：



1.2 组件 && 框架

播放器组件

利用 MediaPlayer 控制音乐播放，MPMusicPlayerController 初始化播放器。

网络接口

使用 StoreKit 库，通过应用内购与 App Store 进行交互

StoreKit 库 中 Apple Music API 的 SKCloudServiceController 类，可以帮助同步用户的 Apple Music 实现读取歌单、歌曲列表等等，帮助用户为云服务执行设置，如Apple音乐订阅。

页面和动画组件

TabView、Group、withAnimation 实现动画、matchedGeometryEffect 和 onTapGesture 实现动作效果等等。

图片加载组件

SDWebImageSwiftUI 是基于 SwiftUI 和 SDWebImage 实现的图像加载框架，它实现了来自 SDWebimage 所有的功能，如异步图像加载，存储器/磁盘缓存，动画图像播放和表演。

该框架提供了不同的视图 View 结构，API 与 SwiftUI 框架指南匹配。如果熟悉 Image，您会发现易于使用 WebImage 和 AnimatedImage

Since SDWebImageSwiftUI is built on top of SDWebImage, it provide both the out-of-box features as well as advanced powerful features you may want in real world Apps. Check our Wiki when you need:

- Animated Image full-stack solution, with a balance of CPU && RAM
- Progressive image loading, with animation support
- Reusable download, never request a single URL twice
- URL Request / Response Modifier, provide custom HTTP Header
- Image Transformer, apply corner radius or CIFilter
- Multiple caches system, query from different source
- Multiple loaders system, load from different resource

数据存储组件

使用 CoreData 在单个设备上持久化或缓存数据，或使用 CloudKit 将数据同步到多个设备。

保存应用程序的永久数据以供离线使用，缓存临时数据，并在单个设备上为应用程序添加撤销功能。为了在一个 iCloud 帐户中跨多个设备同步数据，Core Data 自动将您的模式镜像到 CloudKit 容器。

通过Core Data的Data Model编辑器，您可以定义数据的类型和关系，并生成相应的类定义。Core Data可以在运行时管理对象实例，以提供以下特性

JSON 数据处理组件

使用 SwiftyJSON 使得在Swift中处理 JSON 数据变得更加容易，Swift 是一门类型严格的语言，虽然严格类型能在传输数据时避免一些错误，但在处理 JSON 等领域时，它却变得麻烦（隐式类型）。

而使用 SwiftyJSON 能够将 JSON 数据转换成字典，方便地提取数据。

1.3 ARKit、RealityKit

ARKit是苹果在2017年WWDC推出的AR开发平台。开发人员可以使用这套工具为iPhone和iPad创建增强现实应用程序。

基础的组件有：

`class ARSession (英文)` 用于控制增强现实体验的主要对象。

`class ARConfiguration (英文)` 这个对象用于定义给定时间在会话中启用的特定框架功能。

`class ARAnchor (英文)` 关注对象在物理环境中的位置和方向。

1.3.1 AR 演唱会

利用能够自动完成渲染处理的视图，打造功能全面的 AR 体验。

`class ARView (英文)` 这个视图用于通过 RealityKit 呈现增强现实体验。

`class ARSCNView (英文)` 这个视图让您可以利用 SceneKit 呈现增强现实体验。

`class ARSKView (英文)` 这个视图让您可以利用 SpriteKit 呈现 AR 体验。

1.3.2 AR 留言墙

通过显示锚定的文本，为 AR 体验添加注释。

在增强现实体验中为物体创建屏幕注释 (英文)利用在屏幕上给现实和虚拟物体贴上虚拟便签，为 AR 体验添加注释。

识别和标记任意对象 (英文)利用自定光学识别算法，创建用于跟踪相机视频流中所识别对象的锚点。

二、后端框架

2.1 歌曲数据来源

本项目的所有歌曲来源通过各大主流音乐平台的公开服务器进行搜索，使用我们部署的服务端作为中转，将歌曲文件传递到 App 应用。我们并没有自己的歌曲数据库，需要借助网络中的资源库来支撑我们的歌曲搜索功能。

- 项目中使用到的搜索歌曲 API，能够在 Github 上找到别人总结好的列表。其本质是模拟对应音乐平台的客户端，向音乐平台的公开服务器发送搜索请求，并得到响应。因此可以将这个功能看作使用公开服务器的一个微服务。

2.2 Mock 服务

由于开发过程中部分功能不能完全并行开发，使用 Mock 服务能够模拟服务端请求的返回值，提高客户端调试的效率。当服务端的相应功能完成后则可以直接进行无缝对接。

2.2.1 Postman

Postman 是一款用于接口测试的工具，它支持 http 协议的多种请求，可以对项目的 request 和 response 进行调试。同时 Postman 具有 Mock server 的功能，能够快速地部署 Mock 服务，自定义各个路径对应的 request 以及期望得到的 response。

2.2 数据持久化

除去音乐资源，我们还需要对用户账号信息进行持久化。

2.2.2 MySQL

MySQL 是主流的关系型数据库之一，被广泛应用于中小型 web 应用中实现数据存储的功能。目前我们的项目规模并不是很大，因此使用这款数据库能很好地满足在开发中所有数据存储需求。由于 MySQL 的 InnoDB 内核，在规模增大的情况下能够进行有效的并发控制，使得数据库性能不会有明显的下降。同时 InnoDB 还支持 commit、rollback 还有 recovery 等事务安全的功能，保证了数据的一致性。