# 移动互联网技术及应用

# 大作业报告

## 题目：LIFT 工具箱应用的设计与实现

类型：应用系统设计实现

| 姓名 | 班级 | 学号 |
| --- | --- | --- |
| 杜抒泽 | 2021211306 | 2021211110 |

**2024.06**

# 目录

1 项目概述 3
2 相关技术 4
2.1 概述 4
2.2 Swift 语言 4
2.3 SwiftUI 界面绘制 5
2.4 SwiftData 数据持久化 6
2.5 Core Motion 传感器交互 7
3 系统功能需求 10
3.1 概述 10
3.2 “关于”页面与工具箱主页面 11
3.3 国际化、本地化与暗黑模式适配 12
3.4 地动仪 16
3.5 行事历 19
3.6 生日提醒 24
3.7 二维码工具 28
4 系统设计与实现 32
4.1 概述与总体设计 32
4.2 系统分模块设计 33
4.3 模块分层设计 36
4.4 重要的具体实现细节 37
4.4.1 复杂的视图间导航 37
4.4.2 数据持久化的解决方案 38
4.4.3 二维码工具的相机调用 41
4.5 封装和使用运动传感器 45
5 系统可能的扩展 51
5.1 LIFT Watch App 51
5.2 接入 CloudKit 和 push Notification 52
5.3 扩展生日提醒功能 52
5.4 优化二维码工具 52
6 总结体会 53
附录 A 源代码 55

# 1. 项目概述

**LIFT**（Lightweight Integrated Featured Toolbox）是一个多功能、易于扩展的工具箱。**LIFT** 是一个 iOS 平台的原生应用，目前能够运行在 iPhone 与 iPad 上，且若干功能模块在经过简单适配后能够在 Apple Watch 上运行。

**LIFT** 包含了若干实用工具，具体的功能模块请参见§ 3 系统功能需求。**LIFT** 利用了移动设备的陀螺仪、加速度计等运动传感器，相机、图库等图像相关的硬件设备，剪贴板等系统级功能，并实现了数据持久化等移动应用中常用的高级功能，符合《移动互联网技术及应用》的课程主题。

目前，**LIFT** 的 .ipa 安装包体积超级小，仅有不到 400KB[1]，这得益于纯 Swift 原生实现，所有的运行时都为系统自带，无需额外的运行时库。**LIFT** 的资源占用也低到了可怕的程度。即使是启动传感器每隔 0.01 秒读取一次数据，其 CPU 占用率也保持在 1% 以下，内存占用也不会超过 40MB。

然而遗憾的是，由于我没有苹果开发者证书[2]，无法为该 .ipa 进行开发者签名，导致其只能在我自己的设备或越狱的设备上运行。如果您想要将其运行在您的设备上，请自行越狱或联系我。

**LIFT** 的 Swift 有效代码量（不包含空白行和注释）为 2000 行左右，实现了 4 个主要的工具模块，近 20 个页面，若干数据模型以及少量的标准库类扩展。重要的、大量使用传感器或数据模型的代码添加了注释（英文），而绘制界面相关的代码没有添加注释。

[1]比源码还小，我工程源码加上 icon 图像都接近 700KB 了。Apple 怎么做到的？
[2]99 美元每年，太贵了，真的买不起。

# 2. 相关技术

## 2.1 概述

在 LIFT 中，我使用以下的技术开发了一个 iOS 应用：

- XCode 15.2

  用于编写、编译、调试 iOS 应用，其用途与 Android Studio 类似。

- Swift 5.9.2
  - iOS / iPadOS 17.2 SDK，包含
  - SwiftUI：简单地绘制美观的界面
  - UIKit（含 Combine）：绘制复杂界面
  - SwiftData：数据持久化
  - Core Motion

    对设备陀螺仪、加速度计等若干传感器的封装，用于检测设备的运动状态。

  - PhotoKit、AVFoundation、UIPasteboard 等

    访问设备的图库（相册）、相机、剪贴板等功能。

- SF Symbols

  苹果提供的图标库与矢量图绘制工具。我用其绘制应用中的各种图标，以及应用的 icon。

## 2.2 Swift 语言

Swift 是一种静态类型语言，采用了现代化的编程范式，既能够进行面向对象编程，也有函数式编程的能力。Swift 编译器的 LLVM 后端不仅可以生成高效的机器码，还能够进行高级别的优化，使得 Swift 代码在运行时具

有优异的性能表现。Swift 语言的设计强调类型安全、内存安全以及避免使用指针，从而减少了开发中的错误和漏洞。

在底层，Swift 使用 ARC（自动引用计数）进行内存管理。ARC 是一种自动化的内存管理机制，可以在编译时插入适当的 retain 和 release 调用，从而在运行时自动管理对象的生命周期，避免了内存泄漏和悬垂指针的问题。

SwiftUI 和 SwiftData 是苹果生态系统中强大且高效的框架，它们通过现代化的设计和优化技术，为开发者提供了构建高性能、可维护的应用程序的强大工具。SwiftUI 的声明式语法和响应式编程模型，使得用户界面的开发变得更加简洁和高效，而 SwiftData 则通过简化的数据管理和持久化方案，提供了强大的数据处理能力。这些技术的结合，使得 iOS 应用的开发更加高效和便捷。

## 2.3 SwiftUI 界面绘制

SwiftUI 是苹果公司在 WWDC 2019 上发布的一个全新的声明式用户界面框架。SwiftUI 采用 declarative 的语法[3]，允许开发者以一种直观且简洁的方式来描述用户界面。SwiftUI 通过一种全新的方式，将视图和数据绑定在一起，当数据发生变化时，视图会自动更新。

SwiftUI 的核心特点之一是其声明式语法。开发者只需声明界面应该如何展示以及在特定状态下的变化，SwiftUI 会自动管理视图的更新和渲染。它的另一个重要特点是其高度集成的状态管理。通过 `@State`、`@Binding` 和 `@ObservedObject` 等属性包装器，开发者可以轻松地管理视图的状态和数据绑定。

[3]相比 Android 还在用 XML，遥遥领先。

在底层，SwiftUI 利用 Combine 框架来实现响应式编程。Combine 提供了一个统一的接口来处理异步事件流，通过 Publisher 和 Subscriber 进行数据流的传递和处理。SwiftUI 的视图会订阅状态的变化，当状态发生改变时，视图会自动刷新。

SwiftUI 的布局系统采用了一种称为 “Flexbox” 的算法，这种算法源于 Web 开发中的 CSS Flexbox 布局。Flexbox 布局能够自动适应各种屏幕尺寸和设备，从而大大简化了界面的布局工作。开发者可以使用 VStack、HStack 和 ZStack 等布局容器以及其他原语来创建复杂的界面布局。

为了提高性能，SwiftUI 利用了一些底层优化技术。SwiftUI 的视图是轻量级的结构体，它们被编译器高度优化，以减少内存占用和提高渲染效率。此外，SwiftUI 采用了一种称为 “视图重用” 的技术，当视图发生变化时，只会重新渲染需要更新的部分，而不是整个视图树。这种增量更新机制大大提高了界面的响应速度和流畅性。

## 2.4 SwiftData 数据持久化

SwiftData 是苹果在数据持久化方面提供的解决方案，它与 Core Data 紧密集成。Core Data 是一个功能强大的对象图管理和数据持久化框架，自推出以来，一直是 iOS 和 macOS 开发者管理应用数据的首选工具。SwiftData 通过简化 Core Data 的使用，使得数据管理和持久化变得更加便捷。

SwiftData 在底层使用 SQLite 作为默认的存储引擎，SQLite 是一种轻量级的关系型数据库，具有高效、稳定的特点。SwiftData 还支持其他存储方案，如 XML 和二进制文件，以满足不同的需求。

SwiftData 通过 `NSPersistentContainer` 来简化 Core Data 的初始化和配置。`NSPersistentContainer` 是一个高层次的封装，它封装了 Core Data

堆栈的所有组件，使得数据模型的创建和管理更加简单。开发者只需定义数据模型，然后通过 `NSPersistentContainer` 初始化数据堆栈，即可开始使用 Core Data 进行数据管理。

SwiftData 提供了丰富的 API 来进行数据的增删改查操作。开发者可以使用 `NSFetchRequest` 来查询数据，通过 `NSManagedObjectContext` 的 save() 方法来保存数据。为了提高查询性能，Core Data 提供了 `NSBatchUpdateRequest` 和 `NSBatchDeleteRequest`，这些批量操作能够在不加载对象的情况下直接操作持久化存储，从而提高了操作的效率。SwiftData 还支持数据关系的管理，包括一对一、一对多和多对多的关系。开发者可以在数据模型中定义关系，并通过 Core Data 自动管理这些关系的完整性和一致性。

为了提高性能，SwiftData 采用了多种优化技术。例如，Core Data 通过对象缓存和增量更新来减少内存占用和提高操作效率。当数据发生变化时，Core Data 只会更新受影响的对象，而不是重新加载整个数据集。此外，Core Data 提供了后台上下文，使得数据操作可以在后台线程中异步执行，从而避免阻塞主线程，提高应用的响应速度。

## 2.5 Core Motion 传感器交互

Core Motion 是苹果公司推出的一个强大框架，用于获取设备的运动传感器数据，包括加速度计、陀螺仪和磁力计。该框架广泛应用于健身追踪、AR 和导航等领域，提供了访问和处理这些传感器数据的统一接口。

Core Motion 依赖设备内的多种传感器来提供准确的运动数据。主要包括：

- **加速度计：** 测量设备沿 $x$、$y$ 和 $z$ 轴的线性加速度。加速度计数据可以用来检测设备的运动状态，如平移和晃动。
- **陀螺仪：** 测量设备的旋转速率，提供设备在三个维度上的角速度信息。陀螺仪数据用于检测设备的旋转运动。
- **磁力计：** 测量地磁场强度和方向，辅助设备确定其在三维空间中的方位。

Core Motion 采用了一种名为 IMU（惯性测量单元）的技术，将上述传感器的数据进行融合处理。IMU 通过将来自加速度计、陀螺仪和磁力计的数据进行综合分析，使用滤波算法来减少噪声和误差，从而提供更加精确和稳定的运动数据。滤波算法是数据融合的重要组成部分，卡尔曼滤波是最常用的一种。这种算法通过预测和校正两个步骤来处理传感器数据，减少由于噪声引起的误差，从而提供更为平滑和准确的运动数据。数据融合是 Core Motion 的核心，它通过复杂的算法将不同传感器的数据结合起来，提高数据的准确性和可靠性。

在底层，Core Motion 通过以下步骤处理传感器数据：

1. **数据采集：** 从加速度计、陀螺仪和磁力计等传感器实时采集数据。
2. **预处理：** 对原始数据进行初步处理，包括去除噪声和进行简单的平滑处理。
3. **数据融合：** 使用 IMU 技术和滤波算法将处理后的传感器数据进行融合，生成综合的运动数据。
4. **输出数据：** 将融合后的数据传递给应用程序使用，包括设备的加速度、旋转速率和姿态信息。

Core Motion 提供了高效的实时数据处理能力，能够在短时间内采集和处理大量传感器数据。这使得它在需要高频率数据更新的应用中表现出色，如 AR 应用和实时健身追踪应用。Core Motion 采用了优化的低功耗设计，

确保在持续获取传感器数据的情况下，尽量减少对设备电池的消耗。通过智能的数据采集和处理策略，Core Motion 能够在不影响应用性能的情况下延长设备的电池续航时间。为了支持持续的运动数据追踪，Core Motion 允许在应用后台运行时继续采集和处理传感器数据。这样，即使用户将应用切换到后台，仍然可以记录运动数据，如步数和活动量。Core Motion 提供了数据缓存和持久化功能，确保在设备重启或应用重新启动后，仍然可以访问之前采集的运动数据。这对于需要长时间数据追踪的应用非常重要，如健身和健康监测应用。

# 3. 系统功能需求

## 3.1 概述

LIFT 实现的功能：

- 实现了国际化与本地化，支持英文与中文两种语言[4]。
- 完全适配普通模式与夜间模式两种主题。

LIFT 目前包含了 4 个功能模块：

- 地动仪
  - 通过设备的传感器检测设备的震动状态。
  - 大量利用了移动设备的陀螺仪、加速度计等传感器数据。
- 行事历
  - 一个较复杂的事件记录工具，包含了数据模型之间的关系。
  - 是一个实现复杂界面的例子，实现了数据持久化。
  - 能够容易地实现跨设备同步功能。[5]
- 生日提醒
  - 一个简单的生日记录工具。
  - 实现了数据持久化。能够容易地实现跨设备同步功能。
- 二维码工具
  - 一个二维码扫描与生成工具。
  - 使用了移动设备的相机、图库、剪贴板。

[4]均未指定分类。也就是说，不为 “English (UK)” “中文（简体）” 或其他类似物，而是 “English” 和 “中文”。这样，其能适配更多的系统语言首选项。

[5]需要苹果开发者证书，99 美元每年。我写完了才知道需要这么多钱，不交钱连编译的权限都没有，忍痛删去了。

除了工具箱模块以外，还包含一个“关于”页面。以下给出所有页面和功能的图片和详细介绍。

截图大部分来自我自己的设备的实机截图，小部分在模拟器上。只展示 iPhone。iPad 的运行参见给出的视频。

## 3.2 “关于”页面与工具箱主页面

两个页面如下图所示：

图 1　LIFT 的“关于”页面

图 2　LIFT 的工具箱主页面

“关于”页面提供了应用的基本信息和开发者的相关资料。

该页面包含以下内容：

- **应用名称：** 应用的名称 **LIFT** 和简短描述。
- **版本号：** **LIFT** 其版本号（当前为 1.0.2）。
- **作者：** 开发者（我）的名字。
- **仓库链接：** 展示项目的 GitHub 仓库链接，用户可以点击链接访问项目的源代码（课程尚未结束，当前暂未公开，此处暂时指向我的 GitHub 主页）。
- **个人主页：** 我的博客链接。自己写的 App 里为什么不打个广告呢？

“关于”页面的设计简洁明了，用户可以一目了然地获取到应用和开发者的相关信息。

工具箱主页面展示了 **LIFT** 应用提供的各个工具，并通过图标和文字说明引导用户使用。主页面包含以下工具：

- **地动仪：** 用于测量设备的震动，帮助用户检测周围环境的地震情况。
- **行事历：** 帮助用户记录事件与重要日期，支持事件提醒和管理。
- **生日提醒：** 整理所有朋友的生日信息，并在生日当天提醒用户。
- **二维码工具：** 支持扫描和生成二维码，提供便捷的二维码管理功能。

每个工具项都配有图标和简短的功能描述，用户可以通过点击对应的工具项进入详细功能页面。

## 3.3 国际化、本地化与暗黑模式适配

**LIFT** 应用在设计和开发过程中，充分考虑到了国际化、本地化以及暗黑模式的适配，旨在为全球用户提供一致且优质的使用体验。

为了使 **LIFT** 能够服务于不同语言和文化背景的用户，我们实现了国际化和本地化支持。国际化（i18n）是指在应用中使用多语言支持的设计方式，

使得应用可以轻松添加新的语言。而本地化（l10n）则是指根据不同的语言和文化背景调整应用的内容和显示方式。例如，用户在系统设置中选择中文时，**LIFT** 应用会自动加载中文的本地化文件，显示中文界面。对于英文，应用也会自动切换到相应的语言显示。通过这种方式，我们确保不同语言的用户都能顺畅地使用 **LIFT**。

由于其他图片都在中文设置下，下面的几张图片展示英文设置下的 l10n 适配：

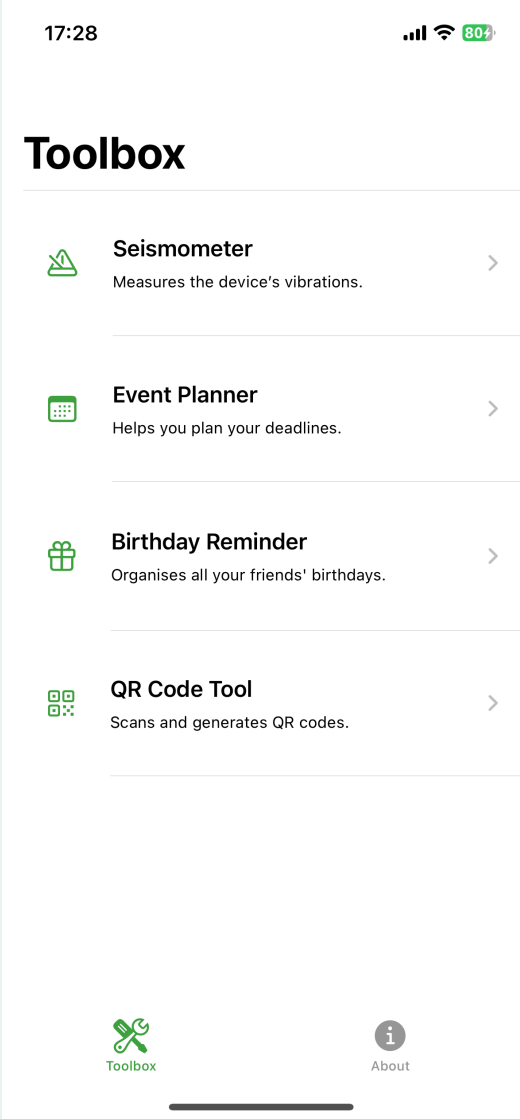


图 3 **LIFT** 主页面（英文）

图 4 行事历（英文）

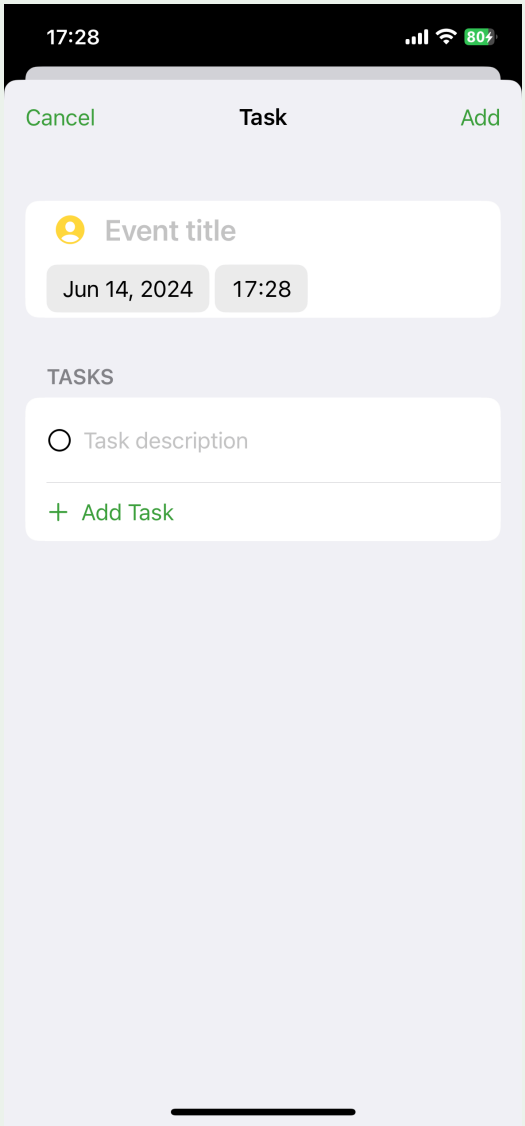


图 5　行事历添加新事件（英文）

暗黑模式（Dark Mode）已成为现代应用程序的重要特性之一，能够在低光环境下减少用户的眼睛疲劳，同时也可以节省设备的电量。**LIFT** 应用在设计时充分考虑了暗黑模式的适配，使得应用在明亮和暗黑两种模式下都能提供良好的视觉体验。

通过这种方式，无论用户选择了系统的明亮模式还是暗黑模式，**LIFT** 应用都能自动调整界面颜色，确保文本的可读性和整体界面的美观性。例如，

工具箱主页面和关于页面在暗黑模式下会自动切换到深色背景，并调整文本和图标的颜色对比度，以提供舒适的视觉效果。

下面的图片展示了 **LIFT** 应用在暗黑模式下的适配效果：

图 6 **LIFT** 主页面（暗黑）

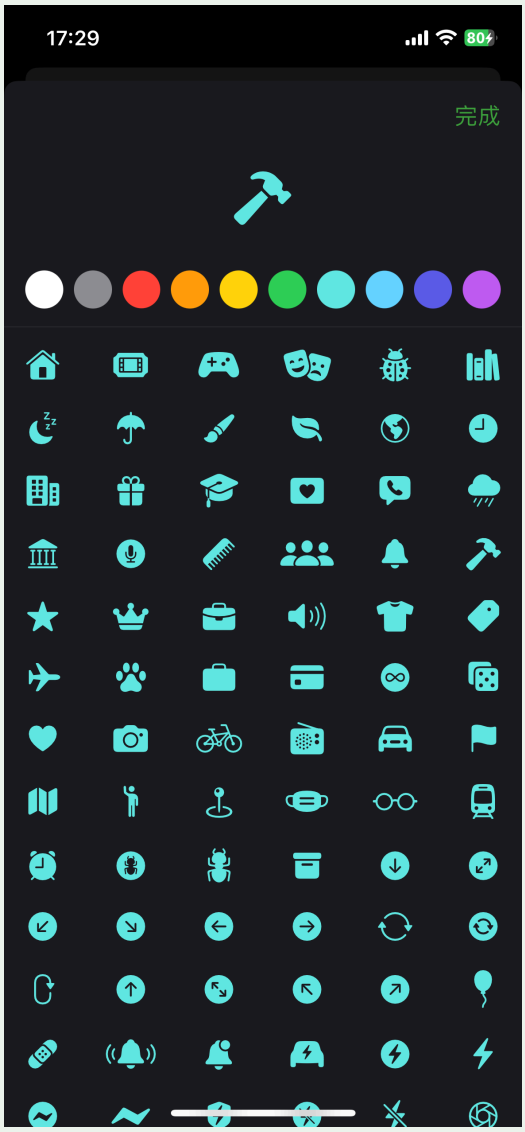


图 7 行事历事件修改图标（暗黑）

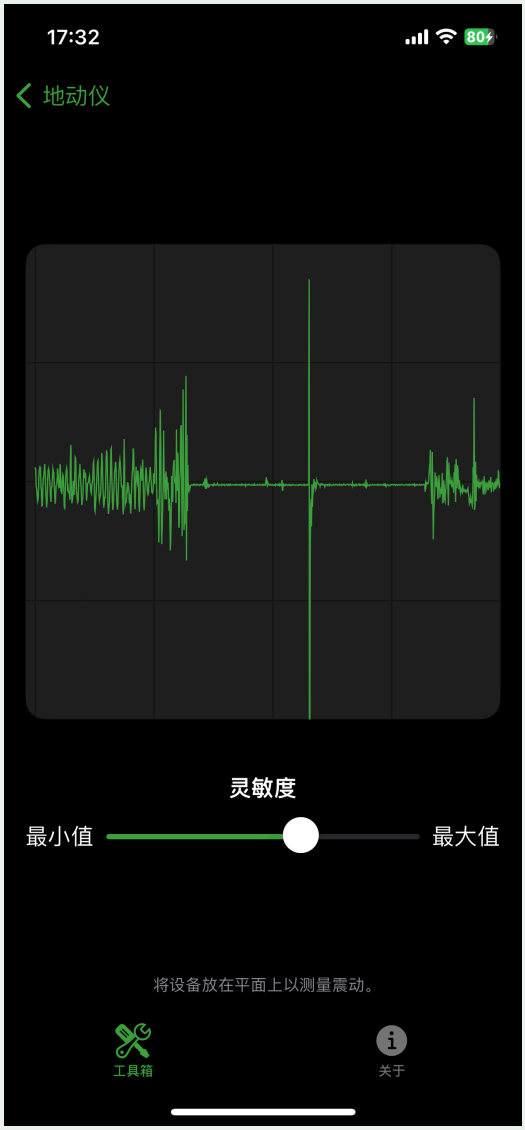


图 8　地动仪动态折线图（暗黑）

## 3.4 地动仪

LIFT 地动仪用于检测设备的震动状态。地动仪主要通过设备的陀螺仪和加速度计传感器来检测设备的运动状态，从而判断设备的震动情况，并以直观的方式展示给用户。地动仪共有三个页面，分别为主页面、指针表盘页面和动态图表页面。

地动仪主页面如下图所示：

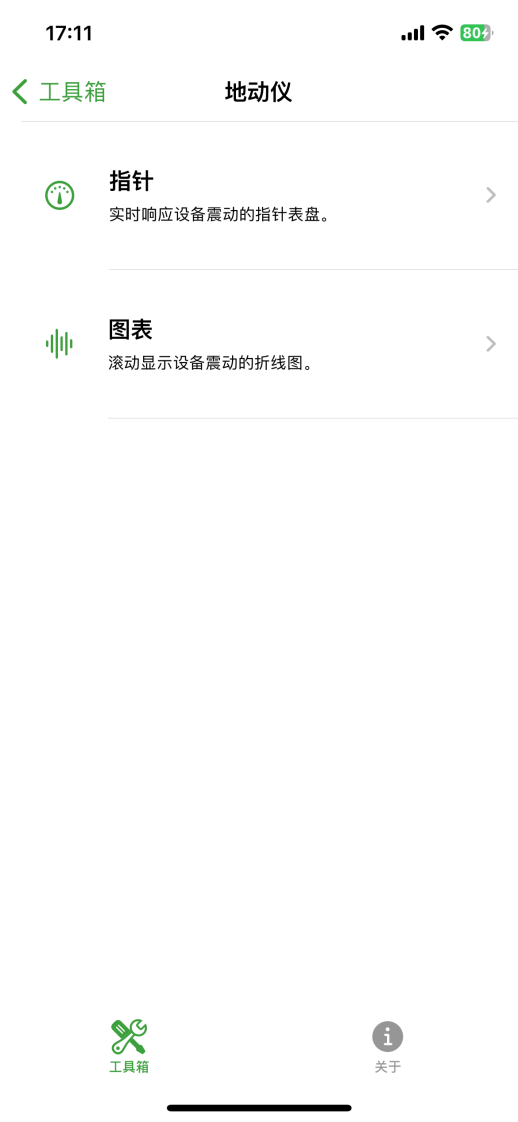


图 9　地动仪主页面

地动仪主页面是用户进入地动仪功能的入口。在这个页面上，用户可以选择不同的展示方式来查看震动数据。页面设计简洁明了，包含两个主要选项：

- **指针**：指针页面实时响应设备震动，展示一个指针表盘。
- **图表**：图表页面以动态折线图的形式展示设备震动的历史数据。

每个选项都配有图标和简短描述，帮助用户快速理解其功能和用途。通过点击对应的选项，用户可以进入详细的展示页面。

指针表盘页面和动态折线图页面如下图所示：

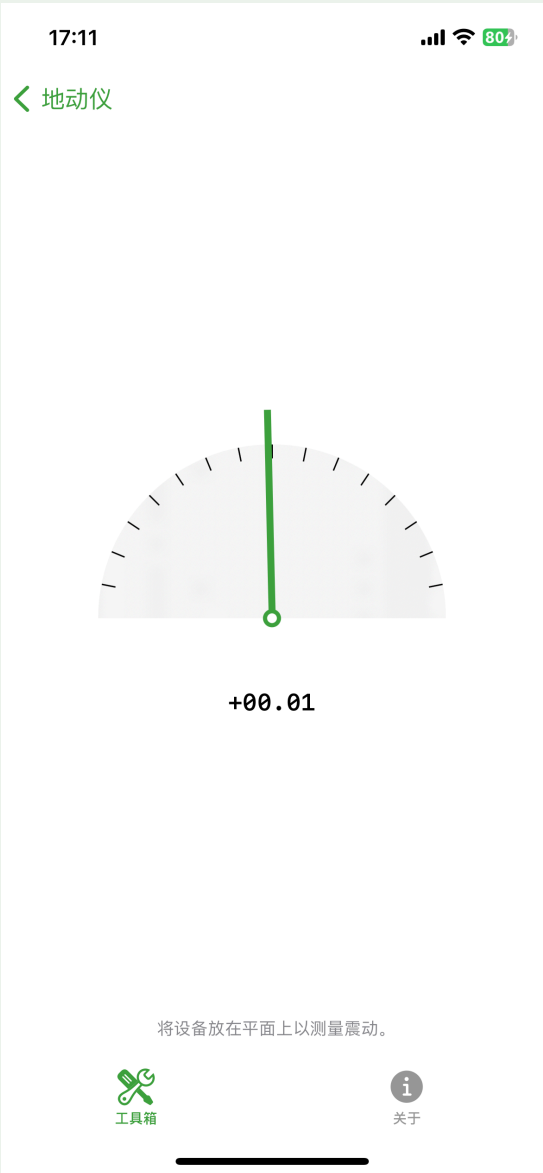


图 10　地动仪指针页面

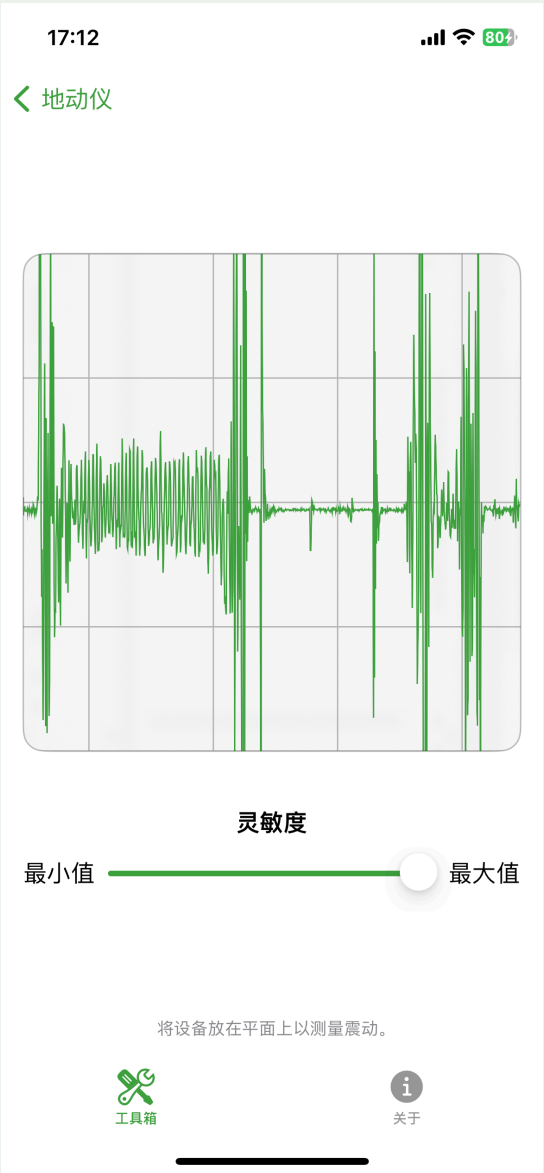


图 11　地动仪动态折线图页面

指针页面提供了一个实时响应设备震动的指针表盘。这个页面设计类似传统的地震仪，通过一个指针来指示当前的震动强度。指针会根据设备的震动

情况实时移动，每 0.01 秒刷新一次，用户可以通过观察指针的摆动情况来判断当前环境的震动强度。

图表页面以折线图的形式展示设备震动的历史数据。用户可以在图表中看到震动强度的变化趋势，从而了解一段时间内的震动情况。页面还提供了一个滑动条，用于调整灵敏度。用户可以通过滑动条来设置震动检测的灵敏度范围，从而获取更精确的数据。

图表页面的设计考虑到了数据的可视化效果，使用了绿色的折线和网格背景，使得数据展示更加直观易读。用户可以通过观察折线图上的波动情况，了解设备在不同时间点的震动强度变化。

两个页面底部都包含一段说明文字，提示用户将设备放在平面上以测量震动。这确保了测量的准确性和稳定性。此外，页面设计保持了与应用整体风格的一致性，使用了简洁的配色和布局，使得用户体验更加舒适。

通过这三个页面，地动仪功能为用户提供了丰富的震动检测和展示手段，帮助用户实时监测环境震动情况。设计上兼顾了美观和实用，确保用户在使用过程中能够获得良好的体验。

## 3.5 行事历

LIFT 行事历旨在帮助用户记录和管理各种事件和任务。行事历功能设计简洁实用，提供了多种视图和编辑方式，满足用户不同的需求。

进入行事历功能后，用户首先看到的是行事历主页面（事件列表页面），如下图所示：

图 12　事件列表

图 13　事件列表（空）

事件列表页面展示了用户创建的所有事件，以卡片形式排列。每个事件卡片显示事件名称、截止日期和关联任务的数量，帮助用户快速浏览和管理事件。事件按时间分类，近期事件优先显示，使得用户能够清晰地查看即将到来的重要事项。在这个页面，用户可以点击右上角的加号按钮添加新事件，也可以点击具体事件卡片查看详情和编辑任务。通过直观的界面设计，用户可以轻松地管理日程和任务，提高工作和生活效率。

当用户尚未添加任何事件时，事件列表页面会显示一个空状态提示。这个页面以简洁的图标和文字告知用户当前没有找到事件，并提示用户点击加号按钮开始添加新事件。这样的设计避免了空白页面带来的困惑，指引用户如何使用行事历功能。

当用户点击加号按钮时，会弹出一个添加事件的表单，如下图所示：

图 14　添加事件表单

用户被禁止直接拖动关闭表单，必须点击取消（什么都不做）或确认（添加事件）按钮。这样设计是为了避免用户误操作，确保用户在添加事件时能够专注填写内容。

除了点击添加以外，用户点击任何一个事件卡片，即可进入事件详情页面。再点击右上角的编辑按钮，即可进入事件编辑页面。事件详情和事件编辑页面如下图所示：

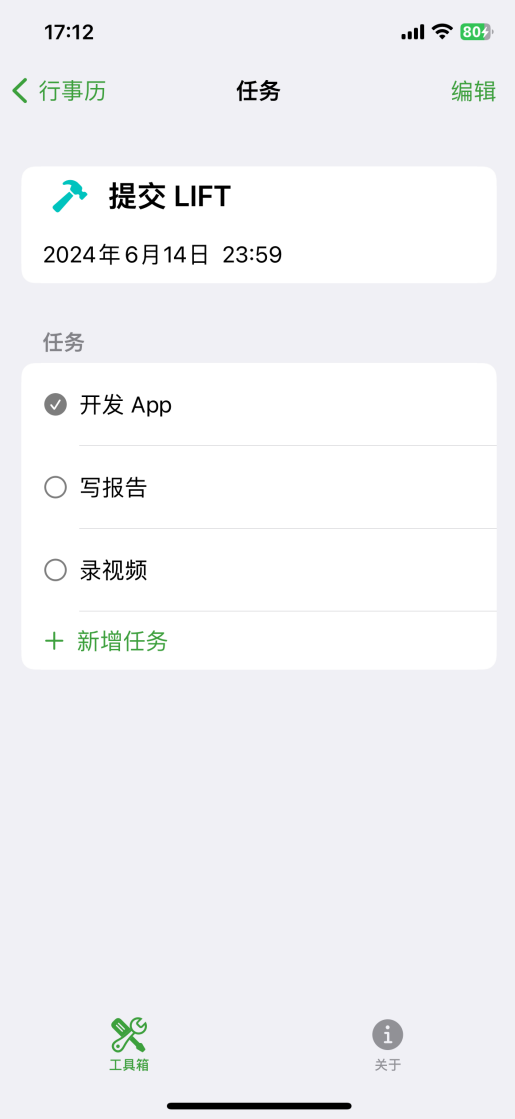


图 15　事件详情

图 16　事件编辑

事件任务编辑页面用于修改已创建的任务。在这个页面，用户可以修改任务的名称、截止日期和任务描述。通过滑动选择器，用户可以轻松地调整日期和时间，确保任务信息准确无误。

在编辑界面，用户点击图标可以修改事件图标，如下图：

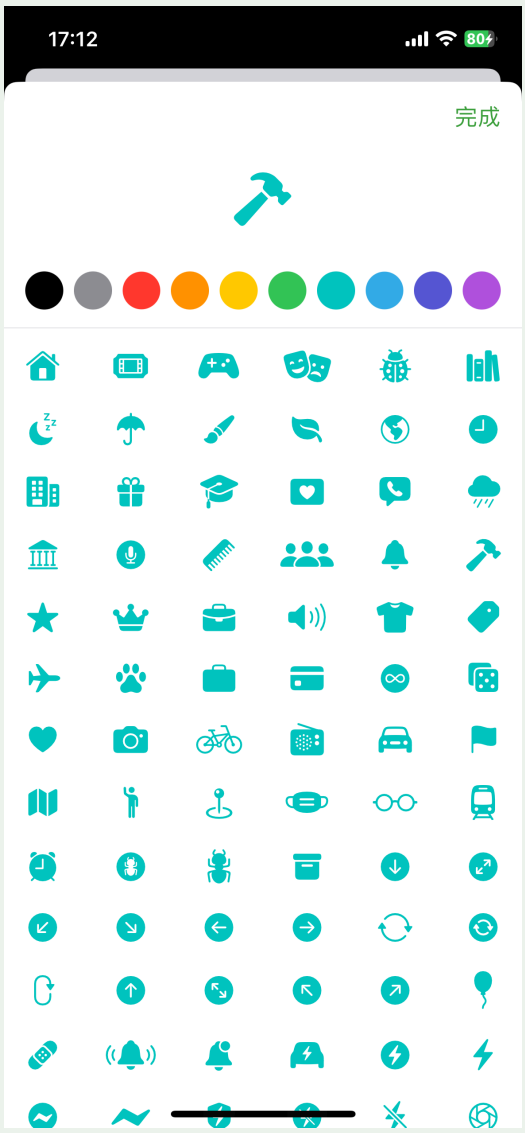


图 17　事件图标选择

图标修改页面提供了多种图标和颜色选项，用户可以为每个事件选择合适的图标和颜色，以便更好地区分和管理不同类型的事件。页面上展示了大量的图标和颜色，用户可以通过点击选择自己喜欢的样式。

这种个性化的设置增强了应用的可用性和用户体验，使得行事历功能不仅实用，而且美观。

通过这些设计，LIFT 的行事历功能为用户提供了全面的事件和任务管理解决方案，帮助用户有效地组织日程，提高生活和工作的效率。

## 3.6 生日提醒

LIFT 生日提醒是一项贴心的功能，旨在帮助用户记录和管理朋友的生日信息。它有很多扩展方案，例如可以增加提醒功能，也可扩展成一个通讯录。以下是各个界面的详细介绍。

进入生日提醒功能后，用户首先看到的是生日列表页面。

生日列表界面展示了用户记录的所有好友的生日信息。每个条目显示好友的名称和生日日期，按照日期顺序排列，方便用户查看近期的生日安排。在这个界面，用户可以点击右上角的加号按钮添加新的生日信息，或者点击已有条目查看和编辑详细信息。列表设计简洁直观，使用户能够快速找到所需的生日信息。对于今天生日的好友，列表会高亮显示，提醒用户及时发送祝福。

当用户尚未添加任何生日信息时，生日列表界面会显示一个空状态提示。页面中心展示了一个图标和提示文字，告知用户当前没有找到朋友的生日信息，并指引用户点击加号按钮开始添加。这样的设计不仅避免了空白页面带来的不便，还能有效引导用户如何开始使用该功能。

如下图所示：

图 18　生日列表

图 19　生日列表（空）

生日列表提供了搜索功能，如下图：

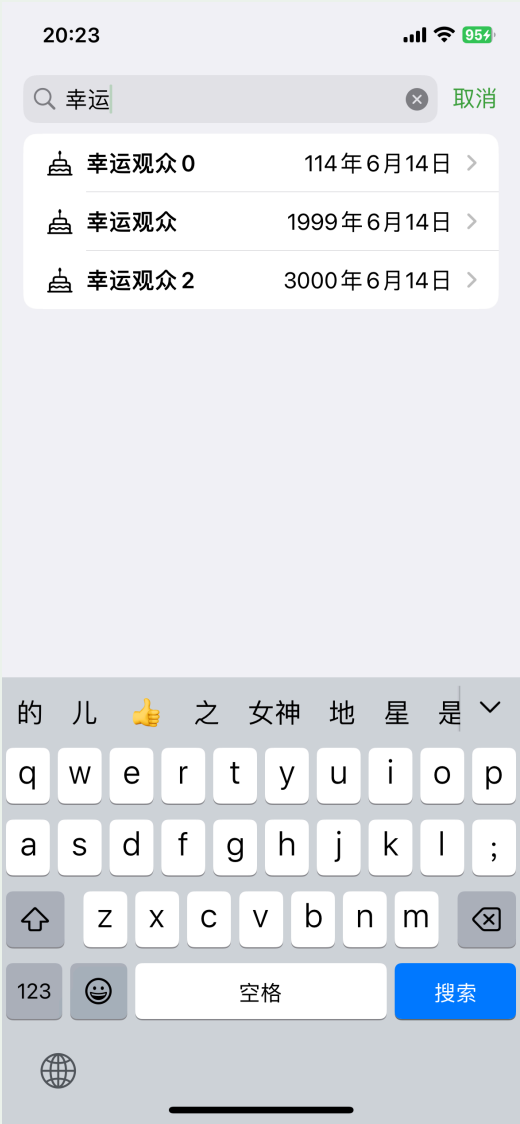


图 20　搜索示例 1

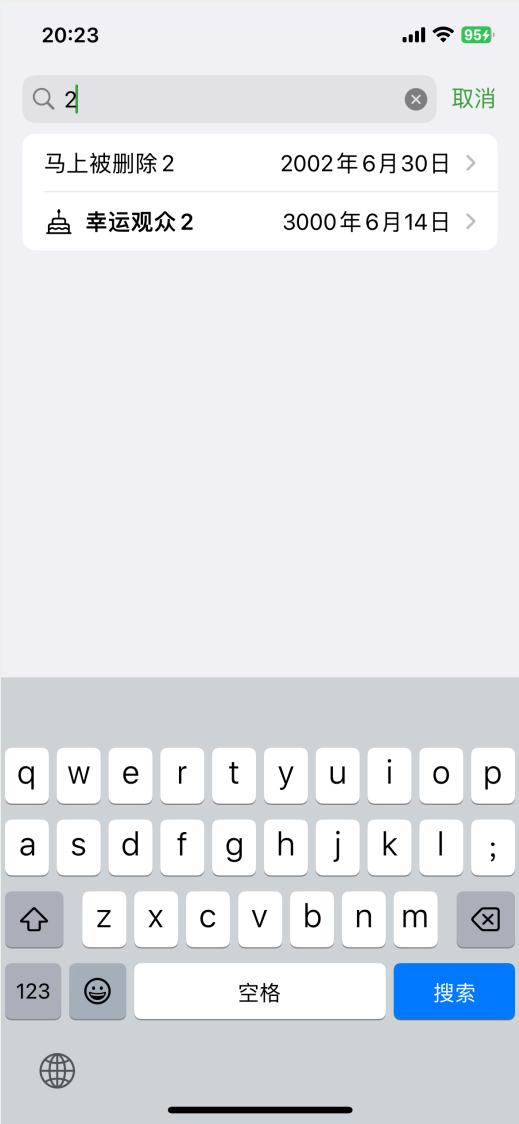


图 21　搜索示例 2

搜索界面提供了便捷的搜索功能，用户可以通过输入关键词快速查找特定好友的生日信息。搜索结果会实时显示在页面下方，用户可以点击相应条目进入详细信息界面。这一功能极大地方便了用户在生日列表中查找特定条目的需求，提升了用户体验。

点击任何一个条目可以进入好友详情页面，如下图所示：

图 22　好友详情

好友条目界面用于查看和编辑单个好友的生日信息。用户可以在这个界面修改好友的姓名和生日日期。日期选择器提供了直观的日期选择方式，使用户能够轻松准确地设置或更改生日信息。页面设计保持了一贯的简洁风格，确保用户在编辑过程中体验顺畅。

通过这些设计，**LIFT** 的生日提醒功能为用户提供了全面的生日管理解决方案。不仅能够帮助用户记录和管理朋友的生日信息，还能在生日当天进

行提醒，确保用户不会错过任何重要的生日庆祝。这一功能贴心实用，为用户的日常生活带来了极大的便利。

## 3.7 二维码工具

用户进入 LIFT 二维码工具后会进入选择器，包含两个选项：

- **生成器：** 从输入的文本生成二维码，可以保存到图库。
- **扫描器：** 扫描二维码获取内容，可以复制到剪贴板。

选择器如图：

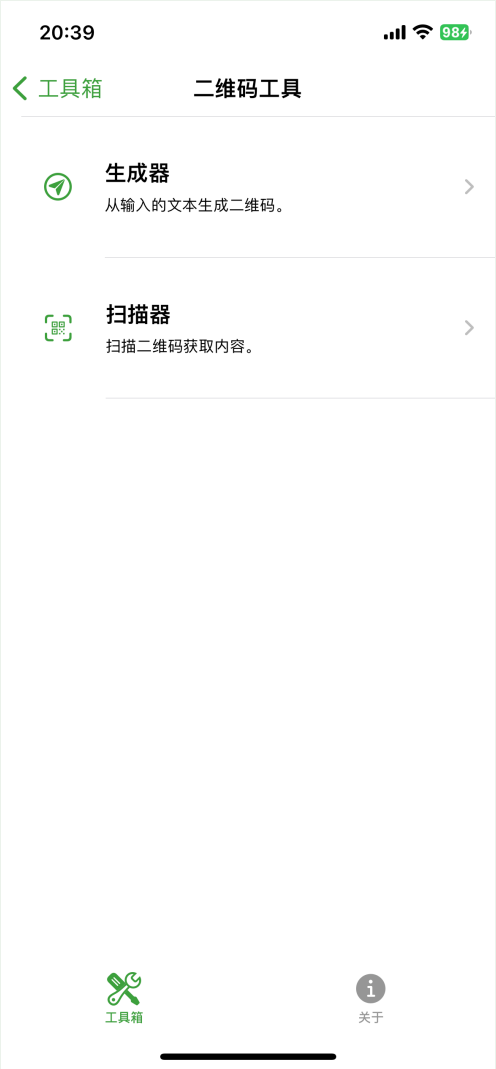


图 23　二维码工具选择器

点击生成器进入生成器：

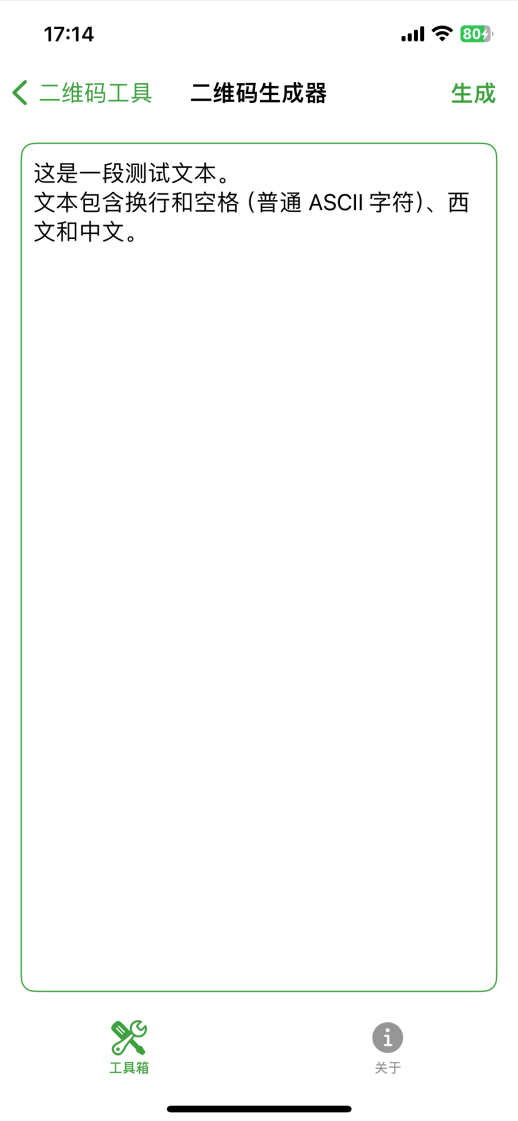


图 24　二维码生成器

有一个巨大的输入框。用户可以在输入框中输入文本，点击“生成”按钮，生成对应的二维码。

- **输入文本界面：**用户在输入框中输入需要生成二维码的文本。
- **生成的二维码：**点击生成按钮后，应用会展示生成的二维码。用户可以选择保存二维码。

- **二维码保存弹窗**：生成二维码后，用户可以选择保存二维码，应用会弹出已保存的提示。

生成的二维码和保存弹窗如图：

图 25　生成的二维码

图 26　二维码保存弹窗

除了生成器外，用户还可以使用扫描器。扫描器实际上调用了相机功能，因此没有单独的展示界面。

扫描器扫描到结果后，应用会显示二维码包含的文本内容，并提供复制功能。

- **扫描结果：**扫描二维码后，应用会显示二维码中包含的文本内容，用户可以点击复制按钮，将内容复制到剪贴板。
- **已复制弹窗：**点击复制按钮后，应用会弹出已复制的提示，确认用户操作成功。

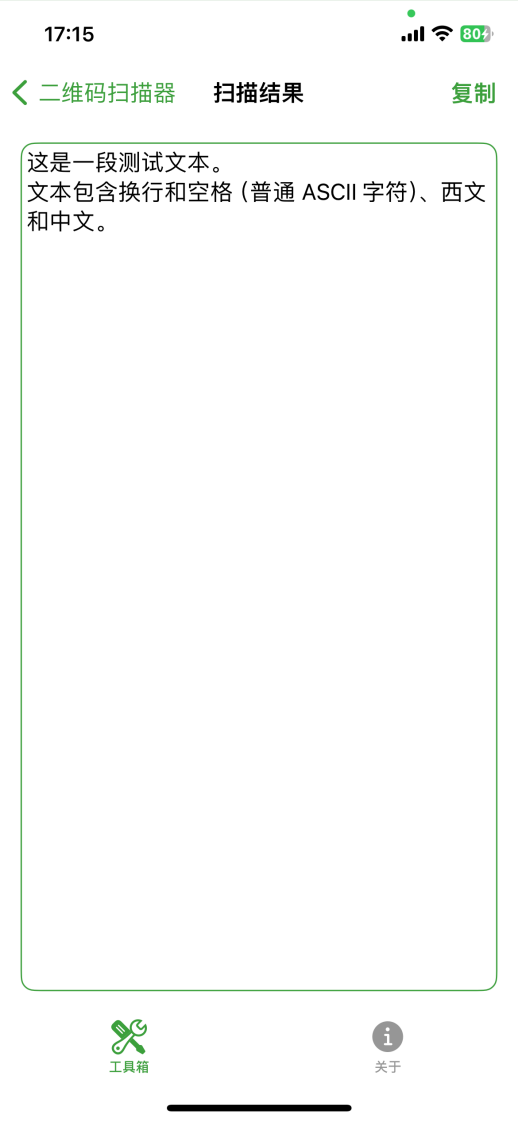


图 27　扫描结果

图 28　已复制弹窗

# 4. 系统设计与实现

## 4.1 概述与总体设计

项目本身的代码量还是比较大的。在代码根目录下运行 `tree` 可以看到以下目录结构:

```
.
├── App
│   ├── AboutView.swift
│   ├── ContentView.swift
│   ├── LIFTApp.swift
│   ├── SampleData.swift
│   └── ToolsView.swift
├── Assets.xcassets
│   ├── AccentColor.colorset
│   │   └── Contents.json
│   ├── AppIcon.appiconset
│   │   ├── 1024.png
│   │   └── Contents.json
│   ├── Contents.json
│   ├── icon.png
│   └── icon.svg
├── BirthdayReminder
│   ├── FilteredFriendsListView.swift
│   ├── Friend.swift
│   ├── FriendDetailView.swift
│   └── FriendsListView.swift
├── DDLPlanner
│   ├── Models
│   │   ├── ColorOptions.swift
│   │   ├── Event.swift
│   │   ├── Symbols.swift
│   │   └── Task.swift
│   ├── SFSymbolStyling.swift
│   └── Views
│       ├── EventEditView.swift
│       ├── EventListView.swift
│       ├── EventRowView.swift
│       ├── EventView.swift
│       ├── SymbolPickerView.swift
│       └── TaskRowView.swift
├── Info.plist
├── LIFT.entitlements
├── Localizable.xcstrings
├── Preview Content
```

```
│   └── Preview Assets.xcassets
│       └── Contents.json
├── QRCodeTool
│   ├── QRCodeGeneratorView.swift
│   ├── QRCodePickerView.swift
│   ├── QRCodeResultView.swift
│   ├── QRCodeScannerView.swift
│   └── QRCodeSheetView.swift
└── Seismometer
    ├── DoubleExtension.swift
    ├── MotionDetector.swift
    └── Views
        ├── GaugeBackgroundView.swift
        ├── GraphSeismometerView.swift
        ├── LineGraphView.swift
        ├── NeedleSeismometerView.swift
        └── SeismometerPickerView.swift
```

LIFT 由多个模块组成，每个模块负责实现特定的功能。这些模块包括：

- App：应用程序的核心模块，包含应用程序的主入口、主要视图以及关于页面。
- BirthdayReminder：生日提醒模块，管理和显示好友的生日信息。
- DDLPlanner：DDL 规划器模块，用于管理事件和任务。
- QRCodeTool：二维码工具模块，提供二维码生成和扫描功能。
- Seismometer：地震仪模块，用于检测和显示地震数据。

## 4.2 系统分模块设计

1. App 模块

   App 模块是 LIFT 应用程序的核心，包含应用的主入口文件、主视图以及其他视图。此外，该模块还包含开发用的 SwiftData 示例数据模型导入器。

   - LIFTApp.swift：应用程序的主入口，定义了应用的生命周期和全局环境。

- ContentView.swift：应用的主要视图，负责导航到不同的工具模块。
- AboutView.swift：显示应用的相关信息和开发者信息。
- SampleData.swift：提供示例数据用于开发和测试。
- ToolsView.swift：展示所有工具的列表视图。

2. BirthdayReminder 模块

BirthdayReminder 模块负责管理和显示好友的生日信息，用户可以添加、编辑和删除好友的生日。

- FilteredFriendsListView.swift：根据搜索条件过滤并显示好友列表。
- Friend.swift：定义好友的数据模型，包括姓名和生日信息。
- FriendDetailView.swift：显示和编辑好友详细信息的视图。
- FriendsListView.swift：显示所有好友的列表视图。

3. DDLPlanner 模块

DDLPlanner 模块用于管理事件和任务，用户可以添加、编辑和删除事件和任务，并对其进行分类和排序。

- Models 目录：包含数据模型文件。
  - ColorOptions.swift：定义事件和任务的颜色选项。
  - Event.swift：定义事件的数据模型。
  - Symbols.swift：定义符号的数据模型。
  - Task.swift：定义任务的数据模型。
- Views 目录：包含视图文件。
  - EventEditView.swift：编辑事件的视图。
  - EventListView.swift：显示事件列表的视图。
  - EventRowView.swift：显示单个事件的视图。

  - EventView.swift：显示事件详情的视图。
  - SymbolPickerView.swift：选择符号的视图。
  - TaskRowView.swift：显示单个任务的视图。
- SFSymbolStyling.swift：定义符号样式的辅助文件。

4. QRCodeTool 模块

QRCodeTool 模块提供二维码生成和扫描功能，用户可以输入文本生成二维码，或扫描二维码获取其内容。

- QRCodeGeneratorView.swift：二维码生成器视图，用户输入文本并生成二维码。
- QRCodePickerView.swift：用于选择二维码的视图。
- QRCodeResultView.swift：显示二维码扫描结果的视图。
- QRCodeScannerView.swift：二维码扫描器视图，调用相机进行二维码扫描。
- QRCodeSheetView.swift：显示二维码生成或扫描结果的视图。

5. Seismometer 模块

Seismometer 模块用于检测和显示震动数据，包含震动数据的检测和可视化组件。

- DoubleExtension.swift：Double 类型的扩展，提供地震数据计算的辅助方法。
- MotionDetector.swift：检测设备运动的核心逻辑。
- Views 目录：包含视图文件。
  - GaugeBackgroundView.swift：显示仪表背景的视图。
  - GraphSeismometerView.swift：显示地震波形图的视图。
  - LineGraphView.swift：通用的线形图视图。

- NeedleSeismometerView.swift：显示地震仪指针的视图。
- SeismometerPickerView.swift：选择不同地震仪样式的视图。

## 4.3 模块分层设计

- 视图层

  视图层负责呈现数据和处理用户交互。LIFT 应用程序中的每个模块都有自己独立的视图层，用于显示特定的功能界面。例如：

  - App **模块**
    - ContentView.swift：显示应用的主界面，导航到各个工具模块。
    - ToolsView.swift：显示所有工具的列表。
  - BirthdayReminder **模块**
    - FriendsListView.swift：显示所有好友的列表。
    - FriendDetailView.swift：显示和编辑好友详细信息的视图。
  - DDLPlanner **模块**
    - EventListView.swift：显示事件列表。
    - EventEditView.swift：编辑事件的视图。
  - QRCodeTool **模块**
    - QRCodeGeneratorView.swift：生成二维码的视图。
    - QRCodeScannerView.swift：扫描二维码的视图。
  - Seismometer **模块**
    - GraphSeismometerView.swift：显示地震波形图的视图。
    - NeedleSeismometerView.swift：显示地震仪指针的视图。

- 模型层

  模型层负责数据的定义和处理。每个模块都有自己的数据模型，用于存储和管理特定的功能数据。例如：

- `BirthdayReminder` **模块**
  - `Friend.swift`：定义好友的数据模型，包括姓名和生日信息。
- `DDLPlanner` **模块**
  - `Event.swift`：定义事件的数据模型。
  - `Task.swift`：定义任务的数据模型。
  - `ColorOptions.swift`：定义事件和任务的颜色选项。
  - `Symbols.swift`：定义符号的数据模型。

• 控制层

控制层负责处理业务逻辑和用户交互。**LIFT** 应用程序中的控制层包含在视图和模型之间，协调两者的交互。例如：

- `BirthdayReminder` **模块**
  - `FilteredFriendsListView.swift`：控制好友列表的过滤和显示。
  - `FriendDetailView.swift`：控制好友详细信息的显示和编辑。
- `DDLPlanner` **模块**
  - `EventEditView.swift`：控制事件的编辑和保存。
  - `EventListView.swift`：控制事件列表的显示和管理。
- `QRCodeTool` **模块**
  - `QRCodeGeneratorView.swift`：控制二维码的生成和显示。
  - `QRCodeScannerView.swift`：控制二维码的扫描和结果显示。
- `Seismometer` **模块**
  - `MotionDetector.swift`：控制地震数据的检测和处理。
  - `GraphSeismometerView.swift`：控制地震波形图的显示和更新。

## 4.4 重要的具体实现细节

### 4.4.1 复杂的视图间导航

SwiftUI 的 Naviagation 系列组件十分强大，开发者能够灵活、高效地控制应用程序的导航结构。`NavigationStack` 和 `NavigationSplitView` 是其核心组件。**LIFT** 中大量使用这两个组件以及 `NavigationLink`，实现了复杂的视图间导航。

`NavigationStack` 提供了一种堆栈式导航方法，允许在视图之间进行线性导航。它简化了在导航堆栈中推送和弹出视图的过程，并支持通过路径管理复杂的导航结构。`NavigationStack` 使得路径导航更为简单，特别适用于那些需要多个层级导航的应用程序。通过这种方式，开发者可以定义每个视图的路径，从而实现更复杂的导航逻辑和视图管理。

与此同时，`NavigationSplitView` 提供了分割视图导航的功能，适用于多面板的用户界面设计。这种导航方式在 iPad 和 Mac 应用程序中特别有用，因为它允许同时显示多个视图面板。`NavigationSplitView` 使得用户可以在主视图和详细视图之间无缝切换，从而提升了用户体验。

在 **LIFT** 中，几乎每个非底层页面都使用了 `NavigationStack` 进行了包装；而根视图在 iPad 上还使用条件编译，通过 `NavigationSplitView` 实现了包装。这种导航结构的设计使得 **LIFT** 应用程序的导航更加灵活和高效，用户可以方便地浏览和管理不同的视图。

### 4.4.2 数据持久化的解决方案

在移动应用中，数据持久化是一个关键的方面，确保用户的数据能够在应用程序重启或关闭后保持不变。下面结合 **LIFT** 的具体代码，讲解如何使用 SwiftData 进行数据持久化。

在 **LIFT** 中，数据模型类通过注解 `@Model` 来定义。以 `Event` 类为例，这个类实现了 `Identifiable` 和 `Hashable` 协议，保证每个事件对象都是可唯一

标识的，并且可以参与哈希运算。类中的各个属性如 id、symbol、colorRaw、title、tasks 和 date 定义了一个事件的基本信息。

```
// 定义事件模型类
@Model
class Event: Identifiable, Hashable {
  var id = UUID()       // 事件唯一标识符
  var symbol: String    // 事件符号
  var colorRaw: String  // 事件颜色（原始值）
  var title: String     // 事件标题
  var tasks: [Task]     // 关联任务列表，Task 也需要实现 @Model 注解（重要！）
  var date: Date        // 事件日期

  // 计算属性，根据原始颜色值获取 Color 对象
  var color: Color {
    ColorOptions(rawValue: colorRaw).map { $0.color } ?? .accentColor
  }

  // 自定义构造方法，@Model 注解生成的宏需要使用这种构造方法
  init(
    symbol: String,
    color colorOption: ColorOptions,
    title: String,
    tasks: [Task],
    date: Date
  ) {
    self.symbol = symbol
    self.colorRaw = colorOption.rawValue
    self.title = title
    self.tasks = tasks
    self.date = date
  }

  // 各种与数据持久化无关的方法和属性，省略
}
```

数据模型定义了应用程序中的核心数据结构，并使用 SwiftData 进行持久化存储。

在 LIFT 中，ModelContainer 用于管理应用程序的数据模型。LIFTApp 类通过一个共享的模型容器来管理所有的数据模型，并在应用程序的生命周期内持久化这些数据。

```
@main
struct LIFTApp: App {
  var sharedModelContainer: ModelContainer = {
    // 定义数据模型的 schema
    let schema = Schema([
      Friend.self,
      Event.self,
    ])

    // 配置模型容器，持久化存储
    let modelConfiguration = ModelConfiguration(
      schema: schema, isStoredInMemoryOnly: false
    )

    do {
      return try ModelContainer(
        for: schema, configurations: [modelConfiguration]
      )
    } catch {
      fatalError("Could not create ModelContainer: \(error)")
    }
  }()

  var body: some Scene {
    WindowGroup {
      ContentView()
    }
    .modelContainer(sharedModelContainer) // 将模型容器注入到视图中
  }
}
```

在这里，sharedModelContainer 定义了一个包含 Friend 和 Event 模型的容器，并且通过 ModelConfiguration 配置指定这些数据应被持久化存储而不是仅存储在内存中。

在 SwiftUI 中，视图和数据模型之间的绑定通过 @Query 注解实现。以 EventListView 为例，该视图通过查询 Event 模型来获取所有事件数据，并在列表中展示这些数据。

```
// 定义事件列表视图
struct EventListView: View {
```

```
  // 获取模型上下文
  @Environment(\.modelContext) private var context

  // 查询事件数据并按日期排序
  @Query(sort: \Event.date) private var events: [Event]

  // 状态变量，存储新创建的事件
  @State private var newEvent: Event?

  var body: some View {
    NavigationStack {
      // 显示事件列表，实现细节省略，其中大量使用 context 和 events，它们由 SwiftData 提供
    }
  }

  // 删除事件方法
  private func deleteEvents(at offsets: IndexSet) {
    for offset in offsets {
      context.delete(events[offset])
    }
  }
}
```

在这里，`@Query` 注解用于查询 `Event` 模型的数据，并按日期排序。视图通过 `ForEach` 循环遍历所有事件并根据日期进行分组显示。在工具栏中，添加了一个按钮，用于创建新事件并插入到数据模型容器中。

通过以上的代码示例，展示了如何在 SwiftUI 中使用 SwiftData 进行数据持久化。SwiftData 提供了强大的数据持久化和管理功能，使得开发者可以更方便地定义和操作数据模型。通过模型容器和数据查询，视图能够与数据模型紧密绑定，从而实现动态数据展示和管理。这样的设计不仅简化了数据持久化的实现，也提升了应用程序的性能和用户体验。

### 4.4.3 二维码工具的相机调用

`QRScannerView.swift` 代码旨在为 **LIFT** 提供二维码扫描功能。它使用 SwiftUI 进行用户界面设计，并使用 AVFoundation 访问摄像头

并执行二维码扫描。该实现分为两个主要部分：QRCodeScannerView 和 QRCodeScannerController。

QRCodeScannerView 是用户交互的主视图。它集成了二维码扫描器并处理到结果视图的导航。

```
struct QRCodeScannerView: View {
  // State variable to store the scanned QR code string
  @State private var scannedCode: String = ""
  // State variable to control the presentation of the result view
  @State private var isShowingResult: Bool = false

  var body: some View {
    // Navigation stack for navigation-based UI
    NavigationStack {
      // Custom controller for QR code scanning
      QRCodeScannerController(
        scannedCode: $scannedCode, isShowingResult: $isShowingResult
      )
      .edgesIgnoringSafeArea(.all)  // Extend the view to edges
      .navigationTitle("QR Code Scanner")  // Set the navigation title
      .navigationBarTitleDisplayMode(.inline)  // Display title inline
    }
    // Navigation destination to show the result view when a QR code is scanned
    .navigationDestination(isPresented: $isShowingResult) {
      QRCodeResultView(scannedCode: scannedCode)  // Pass scanned code to result
view
    }
  }
}
```

scannedCode 是一个状态变量，用于存储扫描到的二维码字符串。isShowingResult 是一个状态变量，用于控制是否显示结果视图。body 是视图的主体，它使用 NavigationStack 来管理导航。QRCodeScannerController 被嵌入其中，用于实际的二维码扫描功能。当 isShowingResult 为 true 时，会导航到 QRCodeResultView，显示扫描结果。

QRCodeScannerController 是遵循 UIViewControllerRepresentable 协议的控制器，用于在 SwiftUI 中集成 UIKit 组件。

```
struct QRCodeScannerController: UIViewControllerRepresentable {
  // Binding variables to pass data between SwiftUI and UIKit
  @Binding var scannedCode: String
  @Binding var isShowingResult: Bool

  // Coordinator class to handle delegate methods for QR code scanning
  class Coordinator: NSObject, AVCaptureMetadataOutputObjectsDelegate {
    var parent: QRCodeScannerController

    init(parent: QRCodeScannerController) {
      self.parent = parent
    }

    // Delegate method called when a metadata object is detected
    func metadataOutput(
      _ output: AVCaptureMetadataOutput,
      didOutput metadataObjects: [AVMetadataObject],
      from connection: AVCaptureConnection
    ) {
      if let metadataObject = metadataObjects.first {
        guard
          let readableObject = metadataObject
            as? AVMetadataMachineReadableCodeObject
        else { return }
        guard let stringValue = readableObject.stringValue else { return }
        AudioServicesPlaySystemSound(SystemSoundID(kSystemSoundID_Vibrate))  //
Vibrate on successful scan
        parent.scannedCode = stringValue  // Update scanned code
        parent.isShowingResult = true  // Show result view
      }
    }
  }

  // Create the coordinator instance
  func makeCoordinator() -> Coordinator {
    return Coordinator(parent: self)
  }

  // Method to create the UIViewController
  func makeUIViewController(context: Context) -> UIViewController {
    let viewController = UIViewController()
    let captureSession = AVCaptureSession()
    guard let videoCaptureDevice = AVCaptureDevice.default(for: .video) else {
      return viewController
    }
    let videoInput: AVCaptureDeviceInput

    do {
      videoInput = try AVCaptureDeviceInput(device: videoCaptureDevice)
```

```
    } catch {
      return viewController
    }

    // Add the video input to the capture session
    if captureSession.canAddInput(videoInput) {
      captureSession.addInput(videoInput)
    } else {
      return viewController
    }

    let metadataOutput = AVCaptureMetadataOutput()

    // Add the metadata output to the capture session
    if captureSession.canAddOutput(metadataOutput) {
      captureSession.addOutput(metadataOutput)

      metadataOutput.setMetadataObjectsDelegate(
        context.coordinator, queue: DispatchQueue.main)
      metadataOutput.metadataObjectTypes = [.qr]  // Set metadata type to QR
    } else {
      return viewController
    }

    // Create and configure the preview layer
    let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    previewLayer.frame = viewController.view.layer.bounds
    previewLayer.videoGravity = .resizeAspectFill
    viewController.view.layer.addSublayer(previewLayer)

    // Start the capture session on a background thread
    DispatchQueue.global(qos: .userInitiated).async {
      captureSession.startRunning()
    }

    return viewController
  }

  // Method to update the UIViewController (not used in this case)
  func updateUIViewController(
    _ uiViewController: UIViewController, context: Context
  ) {}
}
```

- `QRCodeScannerController` 使用 `UIViewControllerRepresentable` 协议，将 UIKit 的 `UIViewController` 集成到 SwiftUI 中。

- `Coordinator` 类实现了 `AVCaptureMetadataOutputObjectsDelegate` 协议，用于处理二维码扫描结果。
- `makeCoordinator` 方法创建并返回 `Coordinator` 实例。
- `makeUIViewController` 方法创建并配置 `UIViewController`，设置捕获会话和预览层，开始捕获会话。
- `updateUIViewController` 方法目前未使用，但可以根据需要更新视图控制器。

该代码实现了一个简单的二维码扫描功能，通过 `QRCodeScannerView` 进行用户交互，通过 `QRCodeScannerController` 实现二维码扫描逻辑。这种分离视图和控制器逻辑的方式使代码更清晰、可维护。

在 `info.plist` 中配置权限后，用户首次使用二维码扫描功能时，应用会请求用户授权访问相机。

## 4.5 封装和使用运动传感器

在现代移动设备中，运动检测已经成为许多应用程序的重要组成部分。iPhone SDK 提供了强大的 Core Motion 框架，使开发者能够轻松访问和处理设备的运动数据。Core Motion 提供了多种类来处理不同类型的传感器数据，包括加速度计、陀螺仪和设备运动传感器。`CMMotionManager` 类是 Core Motion 框架中的核心类之一，它用于管理和提供这些传感器的数据。我们可以使用 `CMMotionManager` 获取陀螺仪和设备运动传感器的数据。然而，在实际应用中，我们通常需要更复杂的功能，比如处理设备的姿态变化、计算运动的方向等。为了简化这些操作，我们可以封装一个类，统一管理和处理这些运动数据，这就是 `MotionDetector` 类的设计初衷。

```
class MotionDetector: ObservableObject {
  // Core Motion manager instance
  private let motionManager = CMMotionManager()
  // Set to hold cancellables for Combine
  private var cancellables: Set<AnyCancellable> = []
  // Time interval for motion data updates
  private var updateInterval: TimeInterval

  // Published properties for motion data
  @Published var pitch: Double = 0
  @Published var roll: Double = 0
  @Published var zAcceleration: Double = 0
  @Published var currentOrientation: DeviceOrientation = .landscapeLeft

  // Closure to be called on motion data update
  var onUpdate: (() -> Void) = {}

  // Initializer with update interval
  init(updateInterval: TimeInterval) {
    self.updateInterval = updateInterval
    setupOrientationObserver()
  }

  // Start motion updates
  func start() {
    if motionManager.isDeviceMotionAvailable {
      motionManager.startDeviceMotionUpdates()
      // Timer to periodically update motion data
      Timer.publish(every: updateInterval, on: .main, in: .common)
        .autoconnect()
        .sink { [weak self] _ in
          self?.updateMotionData()
        }
        .store(in: &cancellables)
    } else {
      print("Device motion not available")
    }
  }

  // Setup observer for device orientation changes
  private func setupOrientationObserver() {
    NotificationCenter.default.publisher(
      for: UIDevice.orientationDidChangeNotification
    )
    .compactMap { notification in
      (notification.object as? UIDevice)?.orientation
    }
    .map { DeviceOrientation($0) }
    .assign(to: \.currentOrientation, on: self)
```

```
    .store(in: &cancellables)
  }

  // Update motion data based on device motion
  func updateMotionData() {
    if let data = motionManager.deviceMotion {
      (roll, pitch) = currentOrientation.adjustedRollAndPitch(data.attitude)
      zAcceleration = data.userAcceleration.z
      onUpdate()
    }
  }

  // Stop motion updates and cancel subscriptions
  func stop() {
    motionManager.stopDeviceMotionUpdates()
    cancellables.forEach { $0.cancel() }
  }

  // Deinitializer to ensure stopping motion updates
  deinit {
    stop()
  }
}
```

`MotionDetector` 类实现了 `ObservableObject` 协议，使其能够与 SwiftUI 视图进行数据绑定。我们在类中声明了几个重要的属性和方法，以便管理和处理设备的运动数据。

首先，我们定义了一个 `CMMotionManager` 实例，用于管理设备的运动数据。我们还定义了一个 `Set<AnyCancellable>` 类型的属性 `cancellables`，用于存储 Combine 框架中的取消令牌。这样可以确保我们能够正确管理订阅的生命周期，避免内存泄漏。

接下来，我们定义了几个使用 `@Published` 属性包装器的属性：`pitch`、`roll`、`zAcceleration` 和 `currentOrientation`。这些属性会自动发布其值的变化，使得绑定到这些属性的 SwiftUI 视图能够自动更新。

在 MotionDetector 类的初始化方法中，我们设定了更新的时间间隔，并调用 setupOrientationObserver 方法来设置设备方向变化的观察者。在 setupOrientationObserver 方法中，我们使用 NotificationCenter 来监听设备方向的变化，并将其映射为 DeviceOrientation 枚举值。

start 方法用于启动设备的运动更新。我们首先检查设备是否支持设备运动传感器，如果支持，我们会启动设备运动更新，并使用 Combine 框架中的 Timer 类定期更新运动数据。在 updateMotionData 方法中，我们从 CMMotionManager 获取设备的运动数据，并更新 pitch、roll 和 zAcceleration 属性的值。

stop 方法用于停止设备的运动更新，并取消所有的订阅，以确保资源的正确释放。在类的析构函数 deinit 中，我们调用 stop 方法，以确保在对象销毁时停止运动更新。

为了方便启动 MotionDetector 实例，我们还定义了一个扩展：

```
extension MotionDetector {
  func started() → MotionDetector {
    start()
    return self
  }
}
```

这个扩展为 MotionDetector 添加了一个 started 方法，调用该方法会启动运动更新，并返回自身。这使得我们在创建 MotionDetector 实例时可以更加简洁地启动它。

最后，我们定义了一个 DeviceOrientation 枚举，用于表示设备的方向，并根据设备的方向调整 roll 和 pitch 的值：

```swift
enum DeviceOrientation {
  case portrait
  case portraitUpsideDown
  case landscapeLeft
  case landscapeRight
  case faceUp
  case faceDown
  case unknown

  init(_ uiDeviceOrientation: UIDeviceOrientation) {
    switch uiDeviceOrientation {
    case .portrait: self = .portrait
    case .portraitUpsideDown: self = .portraitUpsideDown
    case .landscapeLeft: self = .landscapeLeft
    case .landscapeRight: self = .landscapeRight
    case .faceUp: self = .faceUp
    case .faceDown: self = .faceDown
    case .unknown: self = .unknown
    @unknown default: self = .unknown
    }
  }

  // Adjust roll and pitch based on device orientation
  func adjustedRollAndPitch(_ attitude: CMAttitude) -> (
    roll: Double, pitch: Double
  ) {
    switch self {
    case .unknown, .faceUp, .faceDown:
      return (attitude.roll, -attitude.pitch)
    case .landscapeLeft:
      return (attitude.pitch, -attitude.roll)
    case .portrait:
      return (attitude.roll, attitude.pitch)
    case .portraitUpsideDown:
      return (-attitude.roll, -attitude.pitch)
    case .landscapeRight:
      return (-attitude.pitch, attitude.roll)
    }
  }
}
```

这个枚举通过一个初始化方法将 `UIDeviceOrientation` 转换为 `DeviceOrientation`，并提供一个 `adjustedRollAndPitch` 方法，根据设备的方向调整 `roll` 和 `pitch` 的值。

通过以上的封装，我们可以方便地使用 Core Motion 框架来检测和处理设备的运动数据。`MotionDetector` 类不仅简化了运动数据的获取和处理过程，还提供了与 SwiftUI 的无缝集成，使得运动数据可以直接绑定到视图中，实现动态更新。这样，**LIFT** 实现了检测设备震动数据并实时显示的功能，为用户提供了更加直观的体验。

# 5. 系统可能的扩展

系统的扩展方向非常多啊。这里挑几个说一说。

## 5.1 LIFT Watch App

由于 SwiftUI 的跨平台特性，我们可以很容易地将 LIFT 应用程序扩展到 Apple Watch 平台。我们可以将 LIFT 的部分功能移植到 Apple Watch 上，比如生日提醒、事件管理等。这样，用户可以在 Apple Watch 上随时随地使用 LIFT 的功能，提升了用户体验。

我进行了一些简单移植工作。这是 LIFT Watch App 的工具箱主页面：

图 29　LIFT Watch App Demo

可惜的是，由于移植需要的工作量较大，我还并未完成全部工作，LIFT 仍然只能在 iPhone 和 iPad 上运行。但是，这个方向非常有前景，可以进一步提升 LIFT 的用户体验。

## 5.2 接入 CloudKit 和 push Notification

CloudKit 可以用来存储用户的数据，实现数据的云端同步。push Notification 可以用来提醒用户生日、事件等。这两个功能是 LIFT 用户体验的重要组成部分。

这两个功能十分可惜。我没能实现它们的原因是我没有钱订阅每年 99 美元的开发者账号。接入 CloudKit 的准备工作我已经做好了，但是我没有权限编译。

实在是可惜啊。唉。

## 5.3 扩展生日提醒功能

生日提醒功能能做到的远不止于此。它实际上是通讯录的雏形。能想到的扩展方向有：

- 添加更多的键。比如地址、电话号码、邮箱等。
- 一旦有了电话号码作为键，就可以实现拨打电话的功能。
- 接入日历，与生日提醒结合，实现生日提醒的自动添加。

如果使劲动动脑，应该还能想到更多。

## 5.4 优化二维码工具

二维码工具也有很多优化的空间。比如：

- 允许用户自定义二维码的加密等级。
- 添加更多的二维码生成参数，如颜色。

# 6. 总结体会

这次开发 LIFT 工具箱的经历，让我在技术和上都经历了很大的成长。作为一名学生，能够独立完成这样一个多功能、易于扩展的 iOS / iPadOS 应用程序，对我来说是一种极大的挑战和成就感。

首先，在技术方面，我深入学习和运用了 Swift 语言及其相关框架，包括 SwiftUI、SwiftData、Core Motion 等。这些技术的结合，不仅让我对 iOS 开发有了更全面的理解，也让我在实践中掌握了很多书本上学不到的技巧。特别是 SwiftUI 的使用，让我深刻体会到声明式语法的简洁和高效。相比传统的 UIKit，SwiftUI 在界面绘制和数据绑定方面提供了极大的便利，使得开发效率显著提高。此外，通过 SwiftData 实现的数据持久化方案，也让我更好地理解了如何在移动应用中管理和存储数据。

在项目开发过程中，我遇到了不少的技术难题和 Bug。例如，在实现复杂视图间导航时，如何确保导航路径的正确性和数据传递的完整性，是一个让我头疼的问题。经过不断地调试和学习，我终于掌握了 `NavigationStack` 和 `NavigationSplitView`[6] 的使用方法，实现了灵活高效的视图导航。

在使用 SwiftData 时，我也磕了许多绊子。尽管 SwiftData 简化了数据持久化的很多操作，但在实际实现中，我发现数据模型的定义和管理并不是一帆风顺的。特别是在处理复杂数据关系和进行数据查询时，常常会遇到一些意想不到的问题。经过反复试验和调整，我最终找到了一些有效的解决方案，但这段经历也让我对数据持久化有了更深刻的理解。

[6]最好只在 iPad 上使用 `NavigationSplitView`！我在 iPhone 上使用它时遇到了一系列奇怪 bug，占用了我数个小时的调试时间。尽量使用条件编译在 iPhone 上改用 `NavigationStack`。

通过这次开发，我不仅提高了技术水平，还对移动互联网技术有了更深的认识。移动互联网技术的发展，使得各种应用变得更加丰富和多样化；通过这次实践，我深刻体会到，只有不断学习和掌握新技术，才能在快速发展的互联网时代立于不败之地。

# 附录 A 源代码

注意！只有文件名不以 View.swift 结尾的文件才有注释！以其结尾的文件是绘制前端的，注释了意义不大。

1. App/AboutView.swift

```swift
//
//  AboutView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

struct AboutView: View {
  var body: some View {
    VStack(spacing: 20) {
      HStack {
        Image(systemName: "figure.roll.runningpace")

        Text("LIFT")
      }
      .font(.largeTitle)
      .fontWeight(.heavy)
      .foregroundStyle(.tint)

      Text("Lightweight Integrated Featurized Toolbox")
        .foregroundStyle(.secondary)

      List {
        HStack {
          Text("Version")

          Spacer()

          Text("1.0.2")
            .foregroundStyle(.secondary)
        }

        HStack {
          Text("Author")

          Spacer()

          Text("fa_555")
```

```
            .foregroundStyle(.secondary)
        }

        HStack {
          Text("Repository (Temporarily unreleased)")

          Spacer()

          // make it a link
          Link("GitHub", destination: URL(string: "https://github.com/FA
555")!)
            .foregroundStyle(.link)
        }

        HStack {
          Text("Personal website")

          Spacer()

          Link(
            "blog.fa555.tech",
            destination: URL(string: "https://blog.fa555.tech")!
          )
          .foregroundStyle(.link)
        }
      }
      .listStyle(.plain)
      .frame(maxHeight: 200)
    }
    .padding()
  }
}

#Preview {
  AboutView()
}
```

2. App/ContentView.swift

```
//
//  ContentView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI
```

```
struct ContentView: View {
  var body: some View {
    TabView {
      ToolsView()
        .tabItem {
          Image(systemName: "wrench.and.screwdriver.fill")
          Text("Toolbox")
        }

      AboutView()
        .tabItem {
          Image(systemName: "info.circle")
          Text("About")
        }
    }
  }
}

#Preview {
  ContentView()
    .modelContainer(SampleData.shared.container)
}
```

3. App/LIFTApp.swift

```
//
//  LIFTApp.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftData
import SwiftUI

// Main entry point for the LIFT application
@main
struct LIFTApp: App {
  // Shared model container for the app
  var sharedModelContainer: ModelContainer = {
    // Define the schema including Friend and Event models
    let schema = Schema([
      Friend.self,
      Event.self,
    ])
    // Configure the model container with the schema and storage option
```

```
    let modelConfiguration = ModelConfiguration(
      schema: schema, isStoredInMemoryOnly: false
    )

    do {
      // Create and return the ModelContainer instance
      return try ModelContainer(
        for: schema, configurations: [modelConfiguration])
    } catch {
      // Handle errors during ModelContainer creation
      fatalError("Could not create ModelContainer: \(error)")
    }
  }()

  // Define the body of the application
  var body: some Scene {
    // Create the main window group for the app
    WindowGroup {
      ContentView()
    }
    // Attach the shared model container to the scene
    .modelContainer(sharedModelContainer)
  }
}
```

4. App/SampleData.swift

```
//
//  SampleData.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import Foundation
import SwiftData

// Singleton class to manage sample data for the application
@MainActor
class SampleData {
  // Shared instance of SampleData for use throughout the app
  static let shared = SampleData()

  // ModelContainer to manage data models
  let container: ModelContainer

  // Accessor for the main context of the ModelContainer
```

```
var context: ModelContext {
  container.mainContext
}

// Accessor for a sample Friend object
var friend: Friend {
  Friend.sampleData[0]
}

// Accessor for a sample Event object
var event: Event {
  Event.sampleData[2]
}

// Accessor for a sample Task object
var task: Task {
  Task(text: "Wait for full charge")
}

// Private initializer to set up the ModelContainer and insert sample data
private init() {
  // Define the schema including Friend and Event models
  let schema = Schema([
    Friend.self,
    Event.self,
  ])
  // Configure the model container with the schema
  let configuration = ModelConfiguration(schema: schema)

  do {
    // Create and assign the ModelContainer instance
    container = try ModelContainer(
      for: schema, configurations: [configuration]
    )

    // Insert sample data into the context
    insertSampleData()
  } catch {
    // Handle errors during ModelContainer creation
    fatalError("Failed to create ModelContainer: \(error)")
  }
}

// Method to insert sample data into the context
private func insertSampleData() {
  do {
    // Delete existing data in the context
    try context.delete(model: Friend.self)
    try context.delete(model: Event.self)
```

附录冗长，点此回到附录以前。

```
    } catch {
      // Print an error message if deletion fails
      print("Failed to delete existing Friends or Event: \(error)")
    }

    // Insert sample Friend data into the context
    for friend in Friend.sampleData {
      context.insert(friend)
    }

    // Insert sample Event data into the context
    for event in Event.sampleData {
      context.insert(event)
    }

    do {
      // Save the context to persist the sample data
      try context.save()
    } catch {
      // Print an error message if saving fails
      print("Failed to save sample data: \(error)")
    }
  }
}
```

5. App/ToolsView.swift

```
//
//  ToolsView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

struct ToolsInnerView: View {
  var body: some View {
    List {
      NavigationLink {
        SeismometerPickerView()
      } label: {
        HStack {
          Image(systemName: "metronome")
            .foregroundStyle(.tint)
            .padding()
            .font(.title2)
```

```
        VStack(alignment: .leading, spacing: 8) {
          Text("Seismometer")
            .font(.headline)

          Text("Measures the device's vibrations.")
            .font(.caption)
        }
        .padding(.trailing)
      }
    }
    .padding([.top, .bottom])

    NavigationLink {
      EventListView()
    } label: {
      HStack {
        Image(systemName: "calendar")
          .foregroundStyle(.tint)
          .padding()
          .font(.title2)

        VStack(alignment: .leading, spacing: 8) {
          Text("Event Planner")
            .font(.headline)

          Text("Helps you plan your deadlines.")
            .font(.caption)
        }
        .padding(.trailing)
      }
    }
    .padding([.top, .bottom])

    NavigationLink {
      FilteredFriendsListView()
    } label: {
      HStack {
        Image(systemName: "gift")
          .foregroundStyle(.tint)
          .padding()
          .font(.title2)

        VStack(alignment: .leading, spacing: 8) {
          Text("Birthday Reminder")
            .font(.headline)

          Text("Organises all your friends' birthdays.")
            .font(.caption)
```

```
          }
          .padding(.trailing)
        }
      }
      .padding([.top, .bottom])

      // A tool for scanning and generating QR codes.
      NavigationLink {
        QRCodePickerView()
      } label: {
        HStack {
          Image(systemName: "qrcode")
            .foregroundStyle(.tint)
            .padding()
            .font(.title2)

          VStack(alignment: .leading, spacing: 8) {
            Text("QR Code Tool")
              .font(.headline)

            Text("Scans and generates QR codes.")
              .font(.caption)
          }
          .padding(.trailing)
        }
      }
      .padding([.top, .bottom])
    }
    .listStyle(.plain)
    .navigationTitle(Text("Toolbox"))
  }
}

struct ToolsView: View {
  var body: some View {
    if UIDevice.current.userInterfaceIdiom == .pad {
      NavigationSplitView {
        ToolsInnerView()
      } detail: {
        Text("Select a tool to start.")
          .foregroundStyle(.secondary)
      }
    } else {
      NavigationStack {
        ToolsInnerView()
      }
    }
  }
}
```

附录冗长，点此回到附录以前。

```
#Preview {
  ToolsView()
    .modelContainer(SampleData.shared.container)
}
```

6. Assets.xcassets/AccentColor.colorset/Contents.json

```
{
  "colors" : [
    {
      "color" : {
        "color-space" : "srgb",
        "components" : {
          "alpha" : "1.000",
          "blue" : "0x40",
          "green" : "0xA0",
          "red" : "0x40"
        }
      },
      "idiom" : "universal"
    }
  ],
  "info" : {
    "author" : "xcode",
    "version" : 1
  }
}
```

7. Assets.xcassets/AppIcon.appiconset/Contents.json

```
{
  "images" : [
    {
      "filename" : "1024.png",
      "idiom" : "universal",
      "platform" : "ios",
      "size" : "1024x1024"
    }
  ],
  "info" : {
    "author" : "xcode",
    "version" : 1
  }
}
```

8. Assets.xcassets/Contents.json

```
{
  "info" : {
    "author" : "xcode",
    "version" : 1
  }
}
```

9. BirthdayReminder/FilteredFriendsListView.swift

```
//
//  FilteredFriendsListView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct FilteredFriendsListView: View {
  @State private var searchText = ""

  var body: some View {
    NavigationStack {
      FriendsListView(nameFilter: searchText)
        .searchable(text: $searchText)
    }
  }
}

#Preview {
  FilteredFriendsListView()
    .modelContainer(SampleData.shared.container)
}
```

10. BirthdayReminder/Friend.swift

```
//
//  Friend.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//
```

```
import Foundation
import SwiftData

// Model class representing a Friend entity
@Model
class Friend {
  // Sample data for testing and development
  static let sampleData = [
    Friend(
      name: "Panadol", birthday: Date(timeIntervalSince1970: 1_117_900_800)
    ),
    Friend(name: "Alice", birthday: Date(timeIntervalSince1970: 0)),
    Friend(name: "Bob", birthday: Date(timeIntervalSince1970:
1_000_000_000)),
    Friend(
      name: "Charlie", birthday: Date(timeIntervalSince1970: 2 *
1_000_000_000)
    ),
    Friend(
      name: "David", birthday: Date(timeIntervalSince1970: 3 * 1_000_000_000)
    ),
    Friend(
      name: "Eve", birthday: Date(timeIntervalSince1970: 4 * 1_000_000_000)
    ),
    Friend(name: "Frank", birthday: Date(timeIntervalSince1970:
1_118_600_000)),
    Friend(name: "Grace", birthday: .now),
  ]

  // Properties of the Friend model
  var name: String
  var birthday: Date

  // Initializer to create a Friend instance
  init(name: String, birthday: Date) {
    self.name = name
    self.birthday = birthday
  }

  // Computed property to check if today is the friend's birthday
  var isBirthdayToday: Bool {
    let calendar = Calendar.current
    let today = calendar.dateComponents([.month, .day], from: Date())
    let components = calendar.dateComponents([.month, .day], from: birthday)
    return today == components
  }
}
```

11. BirthdayReminder/FriendDetailView.swift

```swift
//
//  FriendDetailView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct FriendDetailView: View {
  @Bindable var friend: Friend
  let isNew: Bool
  @Environment(\.dismiss) private var dismiss
  @Environment(\.modelContext) private var context

  init(friend: Friend, isNew: Bool = false) {
    self.friend = friend
    self.isNew = isNew
  }

  var body: some View {
    Form {
      TextField("Name", text: $friend.name)

      DatePicker(
        "Birthday", selection: $friend.birthday, displayedComponents: [.date]
      )
    }
    .toolbar {
      if isNew {
        ToolbarItem(placement: .confirmationAction) {
          Button("Done") {
            dismiss()
          }
        }

        ToolbarItem(placement: .cancellationAction) {
          Button("Cancel") {
            context.delete(friend)
            dismiss()
          }
        }
      }
    }
    .navigationTitle(isNew ? "New Friend" : "Friend")
  }
}
```

```
#Preview {
  NavigationStack {
    FriendDetailView(friend: SampleData.shared.friend)
  }
  .modelContainer(SampleData.shared.container)
}

#Preview {
  NavigationStack {
    FriendDetailView(friend: SampleData.shared.friend, isNew: true)
  }
  .modelContainer(SampleData.shared.container)
}
```

12. BirthdayReminder/FriendsListView.swift

```
//
//  BirthdaysListView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftData
import SwiftUI

struct FriendsListView: View {
  @Query(sort: \Friend.birthday) private var friends: [Friend]
  @Environment(\.modelContext) private var context
  @State private var newFriend: Friend?

  init(nameFilter: String = "") {
    let predicate = #Predicate<Friend> { friend in
      return nameFilter.isEmpty
        || friend.name.localizedStandardContains(nameFilter)
    }

    _friends = Query(filter: predicate, sort: \Friend.birthday)
  }

  var body: some View {
    Group {
      if !friends.isEmpty {
        List {
          ForEach(friends) { friend in
            NavigationLink {
```

```
            FriendDetailView(friend: friend)
          } label: {
            HStack {
              if friend.isBirthdayToday {
                Image(systemName: "birthday.cake")
              }

              Text(friend.name)
                .bold(friend.isBirthdayToday)

              Spacer()

              Text(friend.birthday, style: .date)
            }
          }
        }
        .onDelete {
          deleteFriends(at: $0)
        }
      }
    } else {
      ContentUnavailableView {
        Label("No friends", systemImage: "person.3")

        Spacer()
          .frame(maxHeight: 20)

        Text("Tap + to start.")
          .font(.caption)
          .foregroundStyle(.secondary)
      }
      .background(Color(.systemGroupedBackground))
    }
  }
  .navigationTitle("Birthdays")
  .navigationBarTitleDisplayMode(.inline)
  .toolbar {
    #if !os(watchOS)
      ToolbarItem(placement: .topBarTrailing) {
        EditButton()
      }
    #endif

    ToolbarItem {
      Button(action: addFriend) {
        Label("Add Friend", systemImage: "plus")
      }
    }
  }
```

```
    .sheet(item: $newFriend) { friend in
      NavigationStack {
        FriendDetailView(friend: friend, isNew: true)
      }
      .interactiveDismissDisabled()
    }
  }

  private func addFriend() {
    withAnimation {
      let newFriend_ = Friend(name: "", birthday: .now)
      context.insert(newFriend_)
      DispatchQueue.main.async {
        newFriend = newFriend_
      }
    }
  }

  private func deleteFriends(at offsets: IndexSet) {
    for offset in offsets {
      context.delete(friends[offset])
    }
  }
}

#Preview("Empty list") {
  NavigationStack {
    FriendsListView()
  }
}

#Preview {
  NavigationStack {
    FriendsListView()
      .modelContainer(SampleData.shared.container)
  }
}
```

13. DDLPlanner/Models/ColorOptions.swift

```
//
//  ColorOptions.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//
```

```swift
import Foundation
import SwiftUI

// Enum representing various color options
enum ColorOptions: String, CaseIterable {
  case primary = "primary"
  case gray = "gray"
  case red = "red"
  case orange = "orange"
  case yellow = "yellow"
  case green = "green"
  case mint = "mint"
  case cyan = "cyan"
  case indigo = "indigo"
  case purple = "purple"

  // Default color option
  static let `default` = Self.primary

  // Random color option
  static var random: Self {
    return allCases.randomElement()!
  }

  // Computed property to get the corresponding SwiftUI Color
  var color: Color {
    switch self {
    case .primary:
      return .primary
    case .gray:
      return .gray
    case .red:
      return .red
    case .orange:
      return .orange
    case .yellow:
      return .yellow
    case .green:
      return .green
    case .mint:
      return .mint
    case .cyan:
      return .cyan
    case .indigo:
      return .indigo
    case .purple:
      return .purple
    }
```

附录冗长，点此回到附录以前。

```
  }
}
```

14. DDLPlanner/Models/Event.swift

```
//
//  Event.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import Foundation
import SwiftData
import SwiftUI

// Enum to categorize events based on their dates
enum EventCategory: String, Identifiable, CaseIterable {
  case withinWeek = "In 7 Days"
  case withinMonth = "In 30 Days"
  case future = "Future"
  case past = "Past"

  // Conforming to Identifiable protocol
  var id: String {
    rawValue
  }

  // Name property for the category
  var name: String {
    rawValue
  }
}

// Model class representing an Event
@Model
class Event: Identifiable, Hashable {
  // Sample data for testing and development
  static let sampleData: [Event] = [
    Event(
      symbol: "airplane", color: .indigo, title: "Travel to Paris",
      tasks: [
        Task(text: "Pack clothes"),
        Task(text: "Check in online"),
        Task(text: "Get to airport"),
      ],
      date: Date.now.daysOut(-3)
```

```
    ),
    Event(
      symbol: "archivebox.fill", color: .green, title: "Submit LIFT",
      tasks: [
        Task(text: "Write README"),
        Task(text: "Push to GitHub"),
        Task(text: "Submit to App Store"),
      ],
      date: Date.now.daysOut(1)
    ),
    Event(
      symbol: "arrow.down.left.circle.fill", color: .orange,
      title: "Charge iPhone",
      tasks: [
        Task(text: "Plug in charger"),
        Task(text: "Wait for full charge"),
      ],
      date: Date.now.daysOut(2)
    ),
    Event(
      symbol: "bell.fill", color: .red, title: "Meeting with John",
      tasks: [
        Task(text: "Prepare presentation"),
        Task(text: "Send meeting invite"),
      ],
      date: Date.now.daysOut(5)
    ),
    Event(
      symbol: "bicycle", color: .purple, title: "Bike to Work",
      tasks: [
        Task(text: "Check tire pressure"),
        Task(text: "Pack lunch"),
      ],
      date: Date.now.daysOut(7)
    ),
    Event(
      symbol: "bolt.fill", color: .green, title: "Fix Light Bulb",
      tasks: [
        Task(text: "Buy new light bulb"),
        Task(text: "Replace light bulb"),
      ],
      date: Date.now.daysOut(10)
    ),
    Event(
      symbol: "bandage.fill", color: .yellow, title: "Get Vaccinated",
      tasks: [
        Task(text: "Make appointment"),
        Task(text: "Bring ID"),
      ],
```

附录冗长，点此回到附录以前。

```
      date: Date.now.daysOut(14)
    ),
    Event(
      symbol: "balloon.fill", color: .indigo, title: "Birthday Party",
      tasks: [
        Task(text: "Buy gift"),
        Task(text: "RSVP"),
      ],
      date: Date.now.daysOut(30)
    ),
    Event(
      symbol: "book.fill", color: .green, title: "Read Book",
      tasks: [
        Task(text: "Buy book"),
        Task(text: "Read 10 pages"),
      ],
      date: Date.now.daysOut(45)
    ),
    Event(
      symbol: "briefcase.fill", color: .yellow, title: "Job Interview",
      tasks: [
        Task(text: "Research company"),
        Task(text: "Prepare questions"),
      ],
      date: Date.now.daysOut(60)
    ),
  ]

  // Properties of the Event model
  var id = UUID()
  var symbol: String
  var colorRaw: String
  var title: String
  var tasks: [Task]
  var date: Date

  // Computed property to get the corresponding SwiftUI Color
  var color: Color {
    ColorOptions(rawValue: colorRaw).map { $0.color } ?? .accentColor
  }

  // Initializer to create an empty Event instance
  init() {
    self.symbol = Symbols.random
    self.colorRaw = ColorOptions.random.rawValue
    self.title = ""
    self.tasks = [Task(text: "")]
    self.date = Date()
  }
```

```
// Initializer to create an Event instance with specified properties
init(
  symbol: String, color colorOption: ColorOptions, title: String,
  tasks: [Task], date: Date
) {
  self.symbol = symbol
  self.colorRaw = colorOption.rawValue
  self.title = title
  self.tasks = tasks
  self.date = date
}

// Computed property to get the count of remaining tasks
var remainingTaskCount: Int {
  tasks.filter { !$0.isCompleted }.count
}

// Computed property to check if all tasks are completed
var isCompleted: Bool {
  tasks.allSatisfy { $0.isCompleted }
}

// Computed property to check if the event date is in the past
var isPast: Bool {
  date < Date.now
}

// Computed property to check if the event is within a week
var isWithinWeek: Bool {
  Date.now < date && date < Date.now.daysOut(7)
}

// Computed property to check if the event is within a month
var isWithinMonth: Bool {
  Date.now.daysOut(7) ≤ date && date < Date.now.daysOut(30)
}

// Computed property to check if the event is in the future
var isFuture: Bool {
  Date.now.daysOut(30) ≤ date
}

// Computed property to get the event category
var category: EventCategory {
  if date < Date.now {
    return .past
  }
```

附录冗长，点此回到附录以前。

```
    if date < Date.now.daysOut(7) {
      return .withinWeek
    }

    if date < Date.now.daysOut(30) {
      return .withinMonth
    }

    return .future
  }

  // Conformance to Equatable protocol
  static func == (lhs: Event, rhs: Event) -> Bool {
    lhs.id == rhs.id
  }

  // Conformance to Hashable protocol
  func hash(into hasher: inout Hasher) {
    hasher.combine(id)
  }
}

// Extension to add a utility method to Date
extension Date {
  // Method to get a date offset by a number of days
  func daysOut(_ days: Int) -> Date {
    Calendar.autoupdatingCurrent.date(byAdding: .day, value: days, to: self)!
  }
}
```

15. DDLPlanner/Models/Symbols.swift

```
//
//  Symbols.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import Foundation

// Struct to manage a list of symbol names and provide random selections
struct Symbols {
  // Static property to get a random symbol from the list
  static var random = list.randomElement()!

  // Static method to get a list of random symbols with a specified size
```

```
static func randomList(size count: Int) → [String] {
  list.shuffled().prefix(count).map { $0 }
}

// Static array containing a list of symbol names
static let list = [
  "house.fill",
  "ticket.fill",
  "gamecontroller.fill",
  "theatermasks.fill",
  "ladybug.fill",
  "books.vertical.fill",
  "moon.zzz.fill",
  "umbrella.fill",
  "paintbrush.pointed.fill",
  "leaf.fill",
  "globe.americas.fill",
  "clock.fill",
  "building.2.fill",
  "gift.fill",
  "graduationcap.fill",
  "heart.rectangle.fill",
  "phone.bubble.left.fill",
  "cloud.rain.fill",
  "building.columns.fill",
  "mic.circle.fill",
  "comb.fill",
  "person.3.fill",
  "bell.fill",
  "hammer.fill",
  "star.fill",
  "crown.fill",
  "briefcase.fill",
  "speaker.wave.3.fill",
  "tshirt.fill",
  "tag.fill",
  "airplane",
  "pawprint.fill",
  "case.fill",
  "creditcard.fill",
  "infinity.circle.fill",
  "dice.fill",
  "heart.fill",
  "camera.fill",
  "bicycle",
  "radio.fill",
  "car.fill",
  "flag.fill",
  "map.fill",
```

```
    "figure.wave",
    "mappin.and.ellipse",
    "facemask.fill",
    "eyeglasses",
    "tram.fill",
    "alarm.fill",
    "ant.circle.fill",
    "ant.fill",
    "archivebox.fill",
    "arrow.down.circle.fill",
    "arrow.down.left.and.arrow.up.right.circle.fill",
    "arrow.down.left.circle.fill",
    "arrow.down.right.circle.fill",
    "arrow.left.circle.fill",
    "arrow.right.circle.fill",
    "arrow.triangle.2.circlepath",
    "arrow.triangle.2.circlepath.circle.fill",
    "arrow.triangle.capsulepath",
    "arrow.up.circle.fill",
    "arrow.up.left.and.arrow.down.right.circle.fill",
    "arrow.up.left.circle.fill",
    "arrow.up.right.circle.fill",
    "balloon.fill",
    "bandage.fill",
    "bell.and.waves.left.and.right.fill",
    "bell.badge.fill",
    "bolt.car.fill",
    "bolt.circle.fill",
    "bolt.fill",
    "bolt.horizontal.circle.fill",
    "bolt.horizontal.fill",
    "bolt.shield.fill",
    "bolt.slash.circle.fill",
    "bolt.slash.fill",
    "camera.aperture",
    "checkmark.circle.fill",
    "checkmark.square.fill",
    "chevron.compact.down",
    "chevron.compact.left",
    "chevron.compact.right",
    "chevron.compact.up",
    "chevron.down.circle.fill",
    "chevron.left.circle.fill",
    "chevron.right.circle.fill",
    "chevron.up.circle.fill",
    "clock.arrow.2.circlepath",
    "cloud.bolt.fill",
    "cloud.bolt.rain.fill",
    "cloud.drizzle.fill",
```

附录冗长，点此回到附录以前。

```
    "cloud.fill",
    "cloud.hail.fill",
    "cloud.heavyrain.fill",
    "cloud.moon.bolt.fill",
    "cloud.moon.fill",
    "cloud.moon.rain.fill",
    "cloud.sleet.fill",
    "cloud.snow.fill",
    "cloud.sun.bolt.fill",
    "cloud.sun.fill",
    "cloud.sun.rain.fill",
    "creditcard.and.123",
    "cup.and.saucer.fill",
    "doc.append.fill",
    "drop.fill",
    "drop.triangle.fill",
    "envelope.fill",
    "exclamationmark.circle.fill",
    "exclamationmark.octagon.fill",
    "exclamationmark.triangle.fill",
    "externaldrive.fill",
    "ferry.fill",
    "filemenu.and.cursorarrow",
    "film.fill",
    "flame.fill",
    "florinsign.circle.fill",
    "folder.badge.plus",
    "folder.fill",
    "globe.asia.australia.fill",
    "globe.badge.chevron.backward",
    "globe.europe.africa.fill",
    "guitars.fill",
    "hare.fill",
    "hifispeaker.fill",
    "house.and.flag.fill",
    "house.circle.fill",
    "house.lodge.fill",
    "internaldrive.fill",
    "key.fill",
    "keyboard.fill",
    "ladybug.circle.fill",
    "leaf.arrow.triangle.circlepath",
    "lightbulb.fill",
    "lizard.fill",
    "location.circle.fill",
    "location.fill",
    "location.north.circle.fill",
    "location.north.fill",
    "location.north.line.fill",
```

```
    "location.slash.fill",
    "location.square.fill",
    "lock.fill",
    "lock.shield.fill",
    "lungs.fill",
    "magnifyingglass",
    "magnifyingglass.circle.fill",
    "map.circle.fill",
    "mappin",
    "mappin.circle.fill",
    "mappin.slash",
    "megaphone.fill",
    "mic.fill",
    "mic.slash.fill",
    "moon.circle.fill",
    "moon.fill",
    "moon.stars.fill",
    "music.mic",
    "network",
    "paperclip.circle.fill",
    "person.2.fill",
    "person.and.arrow.left.and.arrow.right",
    "person.circle.fill",
    "person.crop.circle.badge.exclamationmark.fill",
    "person.crop.circle.fill",
    "person.crop.rectangle.fill",
    "person.crop.square.fill",
    "person.fill",
    "person.icloud.fill",
    "pills.fill",
    "printer.fill",
    "puzzlepiece.fill",
    "questionmark.circle.fill",
    "questionmark.square.fill",
    "smoke.fill",
    "sparkles",
    "stopwatch.fill",
    "sun.dust.fill",
    "sun.haze.fill",
    "sun.max.fill",
    "sun.min.fill",
    "sunrise.fill",
    "sunset.fill",
    "timer",
    "tortoise.fill",
    "trash.fill",
    "tray.fill",
    "tree.fill",
    "video.fill",
```

附录冗长，点此回到附录以前。

```
    "wind.snow",
    "xmark.circle.fill",
  ]
}
```

16. DDLPlanner/Models/Task.swift

```
//
//  Task.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import Foundation
import SwiftData

// Model class representing a Task entity
@Model
class Task: Identifiable, Hashable {
  // Unique identifier for the Task
  var id = UUID()
  // Timestamp when the Task was added
  var addTime = Date.now
  // Text description of the Task
  var text: String
  // Flag indicating if the Task is completed
  var isCompleted: Bool
  // Flag indicating if the Task is new
  var isNew: Bool

  // Initializer to create a Task instance
  init(text: String, isCompleted: Bool = false, isNew: Bool = false) {
    self.text = text
    self.isCompleted = isCompleted
    self.isNew = isNew
  }

  // Equatable protocol conformance to compare two Task instances
  static func == (lhs: Task, rhs: Task) -> Bool {
    lhs.id == rhs.id
  }

  // Hashable protocol conformance to hash the Task instance
  func hash(into hasher: inout Hasher) {
    hasher.combine(id)
```

```
  }
}
```

17. DDLPlanner/SFSymbolStyling.swift

```
//
//  SFSymbolStyling.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

// ViewModifier to apply specific styling to SF Symbols
struct SFSymbolStyling: ViewModifier {
  // Define the body method to specify the modifications
  func body(content: Content) → some View {
    content
      .imageScale(.large)  // Set the image scale to large
      .symbolRenderingMode(.monochrome)  // Render the symbol in monochrome
mode
  }
}

// Extension to make it easier to apply the SFSymbolStyling modifier
extension View {
  // Method to apply the SFSymbolStyling modifier to any view
  func sfSymbolStyling() → some View {
    modifier(SFSymbolStyling())
  }
}
```

18. DDLPlanner/Views/EventEditView.swift

```
//
//  EventEditView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftData
import SwiftUI
```

```
struct EventEditView: View {
  @Bindable var event: Event
  var isNew = false
  @Environment(\.dismiss) private var dismiss
  @Environment(\.modelContext) private var context

  @State private var isEditing = false

  var body: some View {
    NavigationStack {
      EventView(event: event, isEditing: isNew || isEditing)
        .toolbar {
          ToolbarItem(placement: .cancellationAction) {
            if isNew {
              Button("Cancel") {
                context.delete(event)
                dismiss()
              }
            }
          }

          ToolbarItem {
            Button {
              if isNew {
                dismiss()
              } else {
                isEditing.toggle()
              }
            } label: {
              Text(isNew ? "Add" : isEditing ? "Done" : "Edit")
            }
          }
        }
    }
  }
}

#Preview {
  EventEditView(event: SampleData.shared.event)
    .modelContainer(SampleData.shared.container)
}
```

19. DDLPlanner/Views/EventListView.swift

```
//
//  EventListView.swift
//  LIFT
```

```
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftData
import SwiftUI

struct EventListView: View {
  @Environment(\.modelContext) private var context
  @Query(sort: \Event.date) private var events: [Event]
  @State private var newEvent: Event?

  var body: some View {
    NavigationStack {
      Group {
        if !events.isEmpty {
          List {
            ForEach(EventCatetory.allCases) { category in
              let eventsInCategory =
                events
                .filter {
                  $0.category == category
                }
                .sorted {
                  $0.date < $1.date
                }

              if !eventsInCategory.isEmpty {
                Section {
                  ForEach(eventsInCategory) { event in
                    NavigationLink {
                      EventEditView(event: event)
                    } label: {
                      EventRowView(event: event)
                    }
                  }
                } header: {
                  let view = Text(NSLocalizedString(category.name, comment:
""))
                    .font(.callout)

                  if category == .past {
                    view.foregroundStyle(.secondary)
                  } else {
                    view.bold()
                  }

                  view
```

```
                }
              }
            }
            .onDelete {
              deleteEvents(at: $0)
            }
          }
        } else {
          ContentUnavailableView {
            Label("No events", systemImage: "calendar.badge.exclamationmark")

            Spacer()
              .frame(maxHeight: 20)

            Text("Tap + to start.")
              .font(.caption)
              .foregroundStyle(.secondary)
          }
          .background(Color(.systemGroupedBackground))
        }
      }
      .navigationTitle("Event Planner")
      .navigationBarTitleDisplayMode(.inline)
      .toolbar {
        ToolbarItem {
          Button {
            let newEvent_ = Event()
            context.insert(newEvent_)
            DispatchQueue.main.async {
              newEvent = newEvent_
            }
          } label: {
            Label("Add Event", systemImage: "plus")
          }
        }
      }
      .sheet(item: $newEvent) { event in
        NavigationStack {
          EventEditView(event: event, isNew: true)
        }
        .interactiveDismissDisabled()
      }
    }
  }

  private func deleteEvents(at offsets: IndexSet) {
    for offset in offsets {
      context.delete(events[offset])
    }
```

```
  }
}

#Preview("Empty list") {
  EventListView()
}

#Preview {
  EventListView()
    .modelContainer(SampleData.shared.container)
}
```

20. DDLPlanner/Views/EventRowView.swift

```
//
//  EventRowView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct EventRowView: View {
  let event: Event

  var body: some View {
    HStack {
      Image(systemName: event.symbol)
        .sfSymbolStyling()
        .foregroundStyle(event.color)
        .frame(width: 35, alignment: .center)
        .aspectRatio(contentMode: .fit)

      VStack(alignment: .leading, spacing: 5) {
        Text(event.title)
          .bold()

        Text(event.date.formatted(date: .abbreviated, time: .shortened))
          .font(.caption2)
          .foregroundStyle(.secondary)
      }

      if event.isCompleted {
        Spacer()

        Image(systemName: "checkmark")
```

```
            .sfSymbolStyling()
            .foregroundStyle(.secondary)
        }
      }
      .badge(event.remainingTaskCount)
    }
  }

  #Preview {
    EventRowView(event: SampleData.shared.event)
      .modelContainer(SampleData.shared.container)
  }
```

21. DDLPlanner/Views/EventView.swift

```
//
//  EventView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftData
import SwiftUI

struct EventView: View {
  @Bindable var event: Event
  let isEditing: Bool
  @State private var isPickingSymbol = false

  var body: some View {
    List {
      Section {
        if isEditing {
          HStack {
            Button {
              isPickingSymbol.toggle()
            } label: {
              Image(systemName: event.symbol)
                .sfSymbolStyling()
                .foregroundStyle(event.color)
                .opacity(0.75)
            }
            .buttonStyle(.plain)
            .padding(.horizontal, 5)

            TextField("Event title", text: $event.title)
```

```
            .font(.title2)
            .fontWeight(.semibold)
            .foregroundStyle(.gray)
        }

        DatePicker("Date", selection: $event.date)
          .labelsHidden()
      } else {
        HStack {
          Image(systemName: event.symbol)
            .sfSymbolStyling()
            .foregroundStyle(event.color)
            .padding(.horizontal, 5)

          Text(event.title)
            .font(.title2)
            .fontWeight(.semibold)
        }

        HStack {
          Text(event.date, style: .date)
          Text(event.date, style: .time)
        }
      }
    }
    .listRowSeparator(.hidden)

    Section {
      ForEach(
        $event.tasks.sorted { $lhs, $rhs in
          lhs.addTime < rhs.addTime
        }
      ) { $task in
        TaskRowView(task: $task, isEditing: isEditing)
      }
      .onDelete { indexSet in
        let sortedTasks = event.tasks.sorted { lhs, rhs in
          lhs.addTime < rhs.addTime
        }
        var indexSetToBeRemoved: IndexSet = []
        for index in indexSet {
          indexSetToBeRemoved.insert(
            event.tasks.firstIndex {
              $0.id == sortedTasks[index].id
            }!
          )
        }
        event.tasks.remove(atOffsets: indexSetToBeRemoved)
      }
```

```
        Button {
          event.tasks.append(Task(text: "", isNew: true))
        } label: {
          HStack {
            Image(systemName: "plus")
            Text("Add Task")
          }
        }
        .buttonStyle(.borderless)
      } header: {
        Text("Tasks")
          .bold()
          .font(.callout)
      }
    }
    .navigationTitle("Task")
    .navigationBarTitleDisplayMode(.inline)
    .sheet(isPresented: $isPickingSymbol) {
      SymbolPickerView(event: event)
    }
  }
}

#Preview {
  EventView(event: SampleData.shared.event, isEditing: false)
    .modelContainer(SampleData.shared.container)
}

#Preview {
  EventView(event: SampleData.shared.event, isEditing: true)
    .modelContainer(SampleData.shared.container)
}
```

22. DDLPlanner/Views/SymbolPickerView.swift

```
//
//  SymbolPickerView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct SymbolPickerView: View {
  @Bindable var event: Event
```

```
@State private var selectedColor: Color = ColorOptions.default.color
@Environment(\.dismiss) private var dismiss
@State private var symbolNames = Symbols.list
@State private var searchInput = ""

let columns = Array(repeating: GridItem(.flexible()), count: 6)

var body: some View {
  VStack {
    HStack {
      Spacer()

      Button {
        dismiss()
      } label: {
        Text("Done")
      }
      .padding([.top, .trailing])
    }

    Image(systemName: event.symbol)
      .font(.title)
      .imageScale(.large)
      .foregroundStyle(selectedColor)
      .frame(minHeight: 50, alignment: .center)
      .padding()

    HStack {
      ForEach(ColorOptions.allCases, id: \.self) { colorOption in
        let color = colorOption.color

        Button {
          selectedColor = color
          event.colorRaw = colorOption.rawValue
        } label: {
          Circle()
            .foregroundStyle(color)
        }
      }
    }
    .padding(.horizontal)
    .frame(height: 40)

    Divider()

    ScrollView {
      LazyVGrid(columns: columns) {
        ForEach(symbolNames, id: \.self) { symbolName in
          Button {
```

```
                event.symbol = symbolName
              } label: {
                Image(systemName: symbolName)
                  .sfSymbolStyling()
                  .foregroundStyle(selectedColor)
                  .padding(5)
              }
              .buttonStyle(.plain)
            }
          }
          .drawingGroup()
        }
      }
      .onAppear {
        selectedColor = event.color
      }
    }
  }

#Preview {
  SymbolPickerView(event: SampleData.shared.event)
    .modelContainer(SampleData.shared.container)
}
```

23. DDLPlanner/Views/TaskRowView.swift

```
//
//  TaskRowView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct TaskRowView: View {
  @Binding var task: Task
  var isEditing: Bool = false
  @FocusState private var isFocused: Bool

  var body: some View {
    HStack {
      if isEditing || task.isNew {
        Button {
          task.isCompleted.toggle()
        } label: {
          Image(
```

```
            systemName: task.isCompleted ? "checkmark.circle.fill" : "circle"
          )
        }
        .buttonStyle(.plain)

        TextField("Task description", text: $task.text)
          .focused($isFocused)
          .onChange(of: isFocused) { _, newValue in
            if !newValue {
              task.isNew = false
            }
          }
      } else {
        Image(systemName: task.isCompleted ? "checkmark.circle.fill" :
"circle")
          .foregroundStyle(.secondary)

        Text(task.text)
      }

      Spacer()
    }
    .padding(.vertical, 10)
    .task {
      if task.isNew {
        isFocused = true
      }
    }
  }
}

#Preview {
  TaskRowView(task: .constant(SampleData.shared.task))
    .modelContainer(SampleData.shared.container)
}
```

24. Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/
DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>UIBackgroundModes</key>
  <array/>
</dict>
</plist>
```

25. LIFT.entitlements

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/
DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>com.apple.security.app-sandbox</key>
  <true/>
  <key>com.apple.security.files.user-selected.read-only</key>
  <true/>
</dict>
</plist>
```

26. Localizable.xcstrings

```json
{
  "sourceLanguage" : "en",
  "strings" : {
    "%@" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "1.0.2" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "A graph that shows the device's vibration." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "滚动显示设备震动的折线图。"
          }
        }
```

```
      }
    },
    "A needle that responds to the device's vibration." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "实时响应设备震动的指针表盘。"
          }
        }
      }
    },
    "About" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "关于"
          }
        }
      }
    },
    "Add" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "新增"
          }
        }
      }
    },
    "Add Event" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "新增事件"
          }
        }
      }
    },
    "Add Friend" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "新增朋友"
```

附录冗长，点此回到附录以前。

```
          }
        }
      }
    },
    "Add Task" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "新增任务"
          }
        }
      }
    },
    "Author" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "作者"
          }
        }
      }
    },
    "Birthday" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "生日"
          }
        }
      }
    },
    "Birthday Reminder" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "生日提醒"
          }
        }
      }
    },
    "Birthdays" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
```

```
            "state" : "translated",
            "value" : "生日"
          }
        }
      }
    },
    "blog.fa555.tech" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "Cancel" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "取消"
          }
        }
      }
    },
    "Copied" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "已复制"
          }
        }
      }
    },
    "Copy" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "复制"
          }
        }
      }
    },
    "Date" : {
      "localizations" : {
```

```
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "日期"
          }
        }
      }
    },
    "Done" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "完成"
          }
        }
      }
    },
    "Edit" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "编辑"
          }
        }
      }
    },
    "Event Planner" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "行事历"
          }
        }
      }
    },
    "Event title" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "事件名"
          }
        }
      }
    },
```

附录冗长，点此回到附录以前。

```
    "fa_555" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "Friend" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "朋友"
          }
        }
      }
    },
    "Future" : {
      "extractionState" : "manual",
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "将来"
          }
        }
      }
    },
    "Generate" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "生成"
          }
        }
      }
    },
    "Generate a QR code from the input text." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "从输入的文本生成二维码。"
          }
```

附录冗长，点此回到附录以前。

```
        }
      }
    },
    "Generator" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "生成器"
          }
        }
      }
    },
    "GitHub" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "Graph" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "图表"
          }
        }
      }
    },
    "Helps you plan your deadlines." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "帮你记录事件与 DDL。"
          }
        }
      }
    },
    "If no QR code is shown, please check if the input text is too long and
try again." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
```

```
            "state" : "translated",
            "value" : "如果没有显示二维码，请检查输入文本是否过长并重试。"
          }
        }
      }
    },
    "In 7 Days" : {
      "extractionState" : "manual",
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "7 天内"
          }
        }
      }
    },
    "In 30 Days" : {
      "extractionState" : "manual",
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "30 天内"
          }
        }
      }
    },
    "LIFT" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
    "Lightweight Integrated Featurized Toolbox" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : ""
          }
        }
      }
    },
```

附录冗长，点此回到附录以前。

```
    "Max" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "高"
          }
        }
      }
    },
    "Measures the device's vibrations." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "测量设备的震动。"
          }
        }
      }
    },
    "Min" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "低"
          }
        }
      }
    },
    "Name" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "名字"
          }
        }
      }
    },
    "Needle" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "指针"
          }
        }
```

附录冗长，点此回到附录以前。

```
      }
    },
    "New Friend" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "新朋友"
          }
        }
      }
    },
    "No events" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "未找到事件"
          }
        }
      }
    },
    "No friends" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "未找到朋友"
          }
        }
      }
    },
    "No QR Code Generated" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "未生成二维码"
          }
        }
      }
    },
    "OK" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "好"
```

```
          }
        }
      }
    },
    "Organises all your friends' birthdays." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "整理所有朋友的生日。"
          }
        }
      }
    },
    "Past" : {
      "extractionState" : "manual",
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "过去"
          }
        }
      }
    },
    "Personal website" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "个人主页"
          }
        }
      }
    },
    "Pitch: %lf" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "俯仰角：%lf"
          }
        }
      }
    },
    "QR Code" : {
      "localizations" : {
        "zh" : {
```

```
          "stringUnit" : {
            "state" : "translated",
            "value" : "二维码"
          }
        }
      }
    },
    "QR Code Generator" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "二维码生成器"
          }
        }
      }
    },
    "QR Code Scanner" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "二维码扫描器"
          }
        }
      }
    },
    "QR Code Tool" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "二维码工具"
          }
        }
      }
    },
    "Repository (Temporarily unreleased)" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "仓库（暂未公开）"
          }
        }
      }
    },
    "Roll: %lf" : {
```

```
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "横滚角: %lf"
          }
        }
      }
    },
    "Save" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "保存"
          }
        }
      }
    },
    "Saved" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "已保存"
          }
        }
      }
    },
    "Scan a QR code to get the content." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "扫描二维码获取内容。"
          }
        }
      }
    },
    "Scan Result" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "扫描结果"
          }
        }
      }
```

```
    },
    "Scanner" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "扫描器"
          }
        }
      }
    },
    "Scans and generates QR codes." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "扫描和生成二维码。"
          }
        }
      }
    },
    "Seismometer" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "地动仪"
          }
        }
      }
    },
    "Select a tool to start." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "选择一个工具以开始。"
          }
        }
      }
    },
    "Sensitivity" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "灵敏度"
          }
```

```
        }
      }
    },
    "Set the device on a flat surface to record vibrations." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "将设备放在平面上以测量震动。"
          }
        }
      }
    },
    "Tap + to start." : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "轻点+以开始。"
          }
        }
      }
    },
    "Task" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "任务"
          }
        }
      }
    },
    "Task description" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "任务描述"
          }
        }
      }
    },
    "Tasks" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
```

```
            "value" : "任务"
          }
        }
      }
    },
    "Toolbox" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "工具箱"
          }
        }
      }
    },
    "Version" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "版本"
          }
        }
      }
    },
    "Z Acceleration: %lf" : {
      "localizations" : {
        "zh" : {
          "stringUnit" : {
            "state" : "translated",
            "value" : "Z 轴加速度: %lf"
          }
        }
      }
    }
  },
  "version" : "1.0"
}
```

27. Preview Content/Preview Assets.xcassets/Contents.json

```
{
  "info" : {
    "author" : "xcode",
    "version" : 1
  }
}
```

28. QRCodeTool/QRCodeGeneratorView.swift

```swift
//
//  QRCodeGeneratorView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import EFQRCode
import SwiftUI

struct QRCodeGeneratorView: View {
  @State private var inputText: String = ""
  @State private var isSheetPresented = false
  @State private var qrCodeImage: UIImage?

  var body: some View {
    NavigationStack {
      VStack {
        TextEditor(text: $inputText)
          .frame(maxHeight: .infinity)
          .padding(20)
          .overlay(
            RoundedRectangle(cornerRadius: 10)
              .stroke(.tint)
              .padding()
          )
      }
      .navigationTitle("QR Code Generator")
      .navigationBarTitleDisplayMode(.inline)
      .toolbar {
        ToolbarItem(placement: .confirmationAction) {
          Button("Generate") {
            generateQRCode()
          }
        }
      }
      .sheet(isPresented: $isSheetPresented) {
        QRCodeSheetView(qrCodeImage: qrCodeImage)
      }
    }
  }

  func generateQRCode() {
    if let cgImage = EFQRCode.generate(
      for: inputText,
```

```
      backgroundColor: UIColor.white.cgColor,
      foregroundColor: UIColor.black.cgColor
    ) {
      self.qrCodeImage = UIImage(cgImage: cgImage)
    } else {
      self.qrCodeImage = UIImage(systemName: "xmark.circle.fill")
    }

    self.isSheetPresented = true
  }
}

#Preview {
  QRCodeGeneratorView()
}
```

29. QRCodeTool/QRCodePickerView.swift

```
//
//  QRCodePickerView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct QRCodePickerView: View {
  var body: some View {
    NavigationStack {
      List {
        NavigationLink {
          QRCodeGeneratorView()
        } label: {
          HStack {
            Image(systemName: "paperplane.circle")
              .foregroundColor(Color.accentColor)
              .padding()
              .font(.title2)

            VStack(alignment: .leading, spacing: 8) {
              Text("Generator")
                .font(.headline)
              Text("Generate a QR code from the input text.")
                .font(.caption)
            }
            .padding(.trailing)
```

```
          }
        }
        .padding([.top, .bottom])

        NavigationLink {
          QRCodeScannerView()
        } label: {
          HStack {
            Image(systemName: "qrcode.viewfinder")
              .foregroundColor(Color.accentColor)
              .padding()
              .font(.title2)

            VStack(alignment: .leading, spacing: 8) {
              Text("Scanner")
                .font(.headline)
              Text("Scan a QR code to get the content.")
                .font(.caption)
            }
            .padding(.trailing)
          }
        }
        .padding([.top, .bottom])
      }
      .listStyle(.plain)
      .navigationBarTitle(Text("QR Code Tool"), displayMode: .inline)
    }
  }
}

#Preview {
  QRCodePickerView()
}
```

30. QRCodeTool/QRCodeResultView.swift

```
//
//  QRCodeResultView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI

struct QRCodeResultView: View {
  var scannedCode: String
```

```
  @State private var showAlert = false

  var body: some View {
    NavigationStack {
      GeometryReader { geometry in
        Text(scannedCode)
          .frame(
            minWidth: geometry.size.width - 40,
            maxHeight: .infinity,
            alignment: .topLeading
          )
          .padding(20)
          .overlay(
            RoundedRectangle(cornerRadius: 10)
              .stroke(.tint)
              .padding()
          )
          .navigationTitle("Scan Result")
          .navigationBarTitleDisplayMode(.inline)
          .toolbar {
            ToolbarItem(placement: .confirmationAction) {
              Button("Copy") {
                UIPasteboard.general.string = scannedCode
                showAlert = true
              }
              .alert("Copied", isPresented: $showAlert) {
                Button("OK") {
                  showAlert = false
                }
              }
            }
          }
      }
    }
  }
}

#Preview {
  QRCodeResultView(scannedCode: "Sample QR Code Content")
}
```

31. QRCodeTool/QRCodeScannerView.swift

```
//
//  QRScannerView.swift
//  LIFT
//
```

```
//  Created by 法伍 on 2024/6/13.
//

import AVFoundation
import SwiftUI

// View for QR code scanning
struct QRCodeScannerView: View {
  // State variable to store the scanned QR code string
  @State private var scannedCode: String = ""
  // State variable to control the presentation of the result view
  @State private var isShowingResult: Bool = false

  var body: some View {
    // Navigation stack for navigation-based UI
    NavigationStack {
      // Custom controller for QR code scanning
      QRCodeScannerController(
        scannedCode: $scannedCode, isShowingResult: $isShowingResult
      )
      .edgesIgnoringSafeArea(.all)  // Extend the view to edges
      .navigationTitle("QR Code Scanner")  // Set the navigation title
      .navigationBarTitleDisplayMode(.inline)  // Display title inline
    }
    // Navigation destination to show the result view when a QR code is
scanned
    .navigationDestination(isPresented: $isShowingResult) {
      QRCodeResultView(scannedCode: scannedCode)  // Pass scanned code to
result view
    }
  }
}

// Controller to handle QR code scanning using UIViewControllerRepresentable
struct QRCodeScannerController: UIViewControllerRepresentable {
  // Binding variables to pass data between SwiftUI and UIKit
  @Binding var scannedCode: String
  @Binding var isShowingResult: Bool

  // Coordinator class to handle delegate methods for QR code scanning
  class Coordinator: NSObject, AVCaptureMetadataOutputObjectsDelegate {
    var parent: QRCodeScannerController

    init(parent: QRCodeScannerController) {
      self.parent = parent
    }

    // Delegate method called when a metadata object is detected
```

```
    func metadataOutput(
      _ output: AVCaptureMetadataOutput,
      didOutput metadataObjects: [AVMetadataObject],
      from connection: AVCaptureConnection
    ) {
      if let metadataObject = metadataObjects.first {
        guard
          let readableObject = metadataObject
            as? AVMetadataMachineReadableCodeObject
        else { return }
        guard let stringValue = readableObject.stringValue else { return }

AudioServicesPlaySystemSound(SystemSoundID(kSystemSoundID_Vibrate))  //
Vibrate on successful scan
        parent.scannedCode = stringValue  // Update scanned code
        parent.isShowingResult = true  // Show result view
      }
    }
  }

  // Create the coordinator instance
  func makeCoordinator() -> Coordinator {
    return Coordinator(parent: self)
  }

  // Method to create the UIViewController
  func makeUIViewController(context: Context) -> UIViewController {
    let viewController = UIViewController()
    let captureSession = AVCaptureSession()
    guard let videoCaptureDevice = AVCaptureDevice.default(for: .video) else
{
      return viewController
    }
    let videoInput: AVCaptureDeviceInput

    do {
      videoInput = try AVCaptureDeviceInput(device: videoCaptureDevice)
    } catch {
      return viewController
    }

    // Add the video input to the capture session
    if captureSession.canAddInput(videoInput) {
      captureSession.addInput(videoInput)
    } else {
      return viewController
    }

    let metadataOutput = AVCaptureMetadataOutput()
```

```
    // Add the metadata output to the capture session
    if captureSession.canAddOutput(metadataOutput) {
      captureSession.addOutput(metadataOutput)

      metadataOutput.setMetadataObjectsDelegate(
        context.coordinator, queue: DispatchQueue.main)
      metadataOutput.metadataObjectTypes = [.qr]  // Set metadata type to QR
    } else {
      return viewController
    }

    // Create and configure the preview layer
    let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
    previewLayer.frame = viewController.view.layer.bounds
    previewLayer.videoGravity = .resizeAspectFill
    viewController.view.layer.addSublayer(previewLayer)

    // Start the capture session on a background thread
    DispatchQueue.global(qos: .userInitiated).async {
      captureSession.startRunning()
    }

    return viewController
  }

  // Method to update the UIViewController (not used in this case)
  func updateUIViewController(
    _ uiViewController: UIViewController, context: Context
  ) {}
}

// Preview provider for SwiftUI previews
#Preview {
  QRCodeScannerView()
}
```

32. QRCodeTool/QRCodeSheetView.swift

```
//
//  QRCodeSheetView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/13.
//

import SwiftUI
```

```
struct QRCodeSheetView: View {
  @State var qrCodeImage: UIImage?
  @State private var showAlert = false
  @Environment(\.dismiss) private var dismiss

  var body: some View {
    NavigationStack {
      VStack {
        Spacer()

        if let qrCodeImage = qrCodeImage {
          Image(uiImage: qrCodeImage)
            .resizable()
            .aspectRatio(contentMode: .fit)
            .padding()
        } else {
          Text("No QR Code Generated")
            .padding()
        }

        Spacer()

        Text(
          "If no QR code is shown, please check if the input text is too long
and try again."
        )
        .font(.caption)
        .foregroundStyle(.secondary)
      }
      .navigationBarTitle("QR Code", displayMode: .inline)
      .toolbar {
        ToolbarItem(placement: .cancellationAction) {
          Button("Cancel") {
            dismiss()
          }
        }

        ToolbarItem(placement: .confirmationAction) {
          Button("Save") {
            saveQRCode()
          }
          .alert("Saved", isPresented: $showAlert) {
            Button("OK") {
              dismiss()
            }
          }
        }
      }
```

```
      .padding()
    }
  }

  func saveQRCode() {
    guard let qrCodeImage = qrCodeImage else { return }
    UIImageWriteToSavedPhotosAlbum(qrCodeImage, nil, nil, nil)
    showAlert = true
//    DispatchQueue.main.asyncAfter(deadline: .now()) {
//      dismiss()
//    }
  }
}

#Preview {
  QRCodeSheetView(qrCodeImage: nil)
}
```

33. Seismometer/DoubleExtension.swift

```
//
//  DoubleExtension.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import Foundation

// Extension to add functionality to Double type
extension Double {

  // Method to format a Double as a fixed length string
  // with specified integer and fraction digits
  func describeAsFixedLengthString(
    integerDigits: Int = 2, fractionDigits: Int = 2
  ) → String {
    // Use the formatted method to create a string representation
    // with the desired format
    self.formatted(
      .number
        .sign(strategy: .always())  // Always show the sign (+/-)
        .precision(
          .integerAndFractionLength(
            integer: integerDigits,  // Number of integer digits
            fraction: fractionDigits  // Number of fraction digits
          )
```

```
        )
    )
  }
}
```

34. Seismometer/MotionDetector.swift

```swift
//
//  MotionDetector.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import Combine
import CoreMotion
import SwiftUI

// ObservableObject class for motion detection
class MotionDetector: ObservableObject {
  // Core Motion manager instance
  private let motionManager = CMMotionManager()
  // Set to hold cancellables for Combine
  private var cancellables: Set<AnyCancellable> = []
  // Time interval for motion data updates
  private var updateInterval: TimeInterval

  // Published properties for motion data
  @Published var pitch: Double = 0
  @Published var roll: Double = 0
  @Published var zAcceleration: Double = 0
  @Published var currentOrientation: DeviceOrientation = .landscapeLeft

  // Closure to be called on motion data update
  var onUpdate: (() -> Void) = {}

  // Initializer with update interval
  init(updateInterval: TimeInterval) {
    self.updateInterval = updateInterval
    setupOrientationObserver()
  }

  // Start motion updates
  func start() {
    if motionManager.isDeviceMotionAvailable {
      motionManager.startDeviceMotionUpdates()
      // Timer to periodically update motion data
```

```
      Timer.publish(every: updateInterval, on: .main, in: .common)
        .autoconnect()
        .sink { [weak self] _ in
          self?.updateMotionData()
        }
        .store(in: &cancellables)
    } else {
      print("Device motion not available")
    }
  }

  // Setup observer for device orientation changes
  private func setupOrientationObserver() {
    NotificationCenter.default.publisher(
      for: UIDevice.orientationDidChangeNotification
    )
    .compactMap { notification in
      (notification.object as? UIDevice)?.orientation
    }
    .map { DeviceOrientation($0) }
    .assign(to: \.currentOrientation, on: self)
    .store(in: &cancellables)
  }

  // Update motion data based on device motion
  func updateMotionData() {
    if let data = motionManager.deviceMotion {
      (roll, pitch) = currentOrientation.adjustedRollAndPitch(data.attitude)
      zAcceleration = data.userAcceleration.z
      onUpdate()
    }
  }

  // Stop motion updates and cancel subscriptions
  func stop() {
    motionManager.stopDeviceMotionUpdates()
    cancellables.forEach { $0.cancel() }
  }

  // Deinitializer to ensure stopping motion updates
  deinit {
    stop()
  }
}

// SwiftUI view to display motion data
struct MotionView: View {
  // ObservedObject to detect motion
  @ObservedObject var motionDetector = MotionDetector(updateInterval: 1.0)
```

```
    .started()

  var body: some View {
    VStack {
      Text("Pitch: \(motionDetector.pitch)")
      Text("Roll: \(motionDetector.roll)")
      Text("Z Acceleration: \(motionDetector.zAcceleration)")
    }
    .onAppear {
      motionDetector.start()
    }
    .onDisappear {
      motionDetector.stop()
    }
  }
}

#Preview {
  MotionView()
}

// Extension to start motion detector
extension MotionDetector {
  func started() -> MotionDetector {
    start()
    return self
  }
}

// Enum for device orientation with associated adjustments for roll and pitch
enum DeviceOrientation {
  case portrait
  case portraitUpsideDown
  case landscapeLeft
  case landscapeRight
  case faceUp
  case faceDown
  case unknown

  init(_ uiDeviceOrientation: UIDeviceOrientation) {
    switch uiDeviceOrientation {
    case .portrait: self = .portrait
    case .portraitUpsideDown: self = .portraitUpsideDown
    case .landscapeLeft: self = .landscapeLeft
    case .landscapeRight: self = .landscapeRight
    case .faceUp: self = .faceUp
    case .faceDown: self = .faceDown
    case .unknown: self = .unknown
    @unknown default: self = .unknown
```

附录冗长，点此回到附录以前。

```
    }
  }

  // Adjust roll and pitch based on device orientation
  func adjustedRollAndPitch(_ attitude: CMAttitude) -> (
    roll: Double, pitch: Double
  ) {
    switch self {
    case .unknown, .faceUp, .faceDown:
      return (attitude.roll, -attitude.pitch)
    case .landscapeLeft:
      return (attitude.pitch, -attitude.roll)
    case .portrait:
      return (attitude.roll, attitude.pitch)
    case .portraitUpsideDown:
      return (-attitude.roll, -attitude.pitch)
    case .landscapeRight:
      return (-attitude.pitch, attitude.roll)
    }
  }
}
```

35. Seismometer/Views/GaugeBackgroundView.swift

```
//
//  GaugeBackgroundView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

extension Angle: Identifiable {
  public var id: Double {
    radians
  }
}

struct GaugeBackgroundView: View {
  let width: Double
  let minAngle = Angle(degrees: -90)
  let maxAngle = Angle(degrees: 90)
  let tickCount = 17

  var tickLength: Double {
    width * 0.04
```

```
  }

  var gaugeTickAngles: [Angle] {
    let tickDegrees =
      (maxAngle.degrees - minAngle.degrees) / (Double(tickCount) - 1)
    var angles = [Angle]()

    for tick in 1..<tickCount - 1 {
      angles.append(Angle(degrees: 90 - (Double(tick) * tickDegrees)))
    }

    return angles
  }

  var body: some View {
    ZStack {
      Path { path in
        path.addArc(
          center: CGPoint(x: width / 2, y: width / 2),
          radius: width / 2,
          startAngle: Angle(degrees: minAngle.degrees + 90),
          endAngle: Angle(degrees: maxAngle.degrees + 90),
          clockwise: true
        )
      }
      .fill()
      .foregroundStyle(.thinMaterial)

      ForEach(gaugeTickAngles) { angle in
        Rectangle()
          .frame(width: 1, height: tickLength)
          .offset(y: -width / 2 + tickLength)
          .rotationEffect(angle, anchor: UnitPoint(x: 0.5, y: 1))
          .offset(y: width / 4 - tickLength / 2)
      }
    }
    .frame(width: width, height: width / 2)
  }
}

#Preview {
  GaugeBackgroundView(width: 300)
}
```

36. `Seismometer/Views/GraphSeismometerView.swift`

```

```

```swift
//
//  GraphSeismometerView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

struct GraphSeismometerView: View {
  private var detector = MotionDetector(updateInterval: 0.025)
  @State private var data = [Double]()
  let maxData = 1000

  @State private var sensitivity = 0.0
  let graphMaxValueMostSensitive = 0.01
  let graphMaxValueLeastSensitive = 1.0

  var graphMaxValue: Double {
    graphMaxValueMostSensitive + (1 - sensitivity)
      * (graphMaxValueLeastSensitive - graphMaxValueMostSensitive)
  }

  var graphMinValue: Double {
    -graphMaxValue
  }

  var body: some View {
    VStack {
      Spacer()

      LineGraph(
        data: data, maxData: maxData,
        minValue: graphMinValue,
        maxValue: graphMaxValue
      )
      .background(.thinMaterial)
      .cornerRadius(15)
      .padding()
      .aspectRatio(1, contentMode: .fit)
      .overlay(
        RoundedRectangle(cornerRadius: 15)
          .stroke(.black.opacity(0.25))
          .padding()
      )

      Spacer()
        .frame(height: 25)
```

```
      Text("Sensitivity")
        .font(.headline)

      Slider(
        value: $sensitivity, in: 0 ... 0.75,
        minimumValueLabel: Text("Min"),
        maximumValueLabel: Text("Max")
      ) {
        Text("Sensitivity")
      }
      .padding([.leading, .trailing])

      Spacer()

      Text("Set the device on a flat surface to record vibrations.")
        .foregroundStyle(.secondary)
        .font(.caption)
        .padding()
    }
    .onAppear {
      detector.start()

      detector.onUpdate = {
        data.append(-detector.zAcceleration)
        if data.count > maxData {
          data = Array(data.dropFirst())
        }
      }
    }
    .onDisappear {
      detector.stop()
    }
  }
}

#Preview {
  GraphSeismometerView()
}
```

37. Seismometer/Views/LineGraphView.swift

```
//
//  LineGraphView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
```

```swift
//

import SwiftUI

struct LineGraph: View {
  let data: [Double]
  let maxData: Int

  let minValue: Double
  let maxValue: Double

  let gridSpacing = 250

  @State private var timestep = 0

  func yGraphPosition(_ dataItem: Double, in size: CGSize) -> Double {
    let proportion = (dataItem - minValue) / (maxValue - minValue)

    let yValue: Double = size.height - proportion * size.height

    return yValue
  }

  func xGraphPosition(_ index: Int, in size: CGSize) -> Double {
    let increment = size.width / Double(maxData)
    let base = Double(maxData - data.count) * increment
    return base + Double(index) * increment
  }

  var body: some View {
    Canvas { context, size in
      var lines = Path()

      let increment = size.width / Double(maxData)

      let phase = -1 * timestep % gridSpacing
      var x = Double(phase)
      repeat {
        lines.move(to: CGPoint(x: x * increment, y: 0))
        lines.addLine(to: CGPoint(x: x * increment, y: size.height))
        x += Double(gridSpacing)
      } while x ≤ Double(maxData)

      var y = size.height / 2
      repeat {
        lines.move(to: CGPoint(x: 0, y: y))
        lines.addLine(to: CGPoint(x: size.width, y: y))
        y += increment * Double(gridSpacing)
      } while y ≤ size.height
```

```
      y = size.height / 2
      repeat {
        lines.move(to: CGPoint(x: 0, y: y))
        lines.addLine(to: CGPoint(x: size.width, y: y))
        y -= increment * Double(gridSpacing)
      } while y ≥ 0

      context.stroke(lines, with: .color(.black.opacity(0.25)))

      guard !data.isEmpty else { return }

      var path = Path()

      path.move(
        to: CGPoint(
          x: self.xGraphPosition(0, in: size),
          y: yGraphPosition(data[0], in: size)))

      for (index, dataPoint) in data.dropFirst().enumerated() {
        path.addLine(
          to: CGPoint(
            x: self.xGraphPosition(index, in: size),
            y: self.yGraphPosition(dataPoint, in: size)))
      }

      context.stroke(path, with: .color(.accentColor))
    }
    .onChange(of: data) { _, _ in
      timestep += 1
    }
  }
}
```

38. Seismometer/Views/NeedleSeismometerView.swift

```
//
//  NeedleSeismometerView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

struct NeedleSeismometerView: View {
  @ObservedObject private var detector = MotionDetector(updateInterval: 0.05)
```

```
let needleAnchor = UnitPoint(x: 0.5, y: 1)
let amplification = 2.0
var rotationAngle: Angle {
  Angle(radians: -detector.zAcceleration * amplification)
}

var body: some View {
  VStack {
    Spacer()

    ZStack(alignment: .bottom) {
      GaugeBackgroundView(width: 250)

      Rectangle()
        .foregroundColor(Color.accentColor)
        .frame(width: 5, height: 150)
        .rotationEffect(rotationAngle, anchor: needleAnchor)
        .overlay {
          VStack {
            Spacer()

            Circle()
              .stroke(lineWidth: 3)
              .fill()
              .frame(width: 10, height: 10)
              .foregroundStyle(.tint)
              .background(.white)
              .offset(x: 0, y: 5)
          }
        }
    }

    Spacer()
      .frame(height: 50)

    Text("\(detector.zAcceleration.describeAsFixedLengthString())")
      .font(.system(.body, design: .monospaced))
      .fontWeight(.semibold)

    Spacer()

    Text("Set the device on a flat surface to record vibrations.")
      .font(.caption)
      .foregroundStyle(.secondary)
      .padding()
  }
  .onAppear {
    detector.start()
```

```
    }
    .onDisappear {
      detector.stop()
    }
  }
}

#Preview {
  NeedleSeismometerView()
}
```

39. Seismometer/Views/SeismometerPickerView.swift

```
//
//  ModePickerView.swift
//  LIFT
//
//  Created by 法伍 on 2024/6/12.
//

import SwiftUI

struct SeismometerPickerView: View {
  var body: some View {
    NavigationStack {
      List {
        NavigationLink {
          NeedleSeismometerView()
        } label: {
          HStack {
            Image(systemName: "gauge")
              .foregroundColor(Color.accentColor)
              .padding()
              .font(.title2)

            VStack(alignment: .leading, spacing: 8) {
              Text("Needle")
                .font(.headline)
              Text("A needle that responds to the device's vibration.")
                .font(.caption)
            }
            .padding(.trailing)
          }
        }
        .padding([.top, .bottom])

        NavigationLink {
```

```
          GraphSeismometerView()
        } label: {
          HStack {
            Image(systemName: "waveform")
              .foregroundColor(Color.accentColor)
              .padding()
              .font(.title2)

            VStack(alignment: .leading, spacing: 8) {
              Text("Graph")
                .font(.headline)
              Text("A graph that shows the device's vibration.")
                .font(.caption)
            }
            .padding(.trailing)
          }
        }.padding([.top, .bottom])
      }
      .listStyle(.plain)
      .navigationBarTitle(Text("Seismometer"), displayMode: .inline)
    }
  }
}

#Preview {
  SeismometerPickerView()
}
```