Kindness Companion (善意伴侣) 应用 v3.0 (API 优先)

一个旨在通过技术传递善意、鼓励用户实践并反思日常善行的桌面应用程序。本项目旨在参与“代码有温度 ——用软件守护每一份善意”竞赛。**此版本优先采用云端 API 实现 AI 功能，以确保最佳的可移植性和打包便利性。**

🎯 项目愿景与核心理念

我们相信科技可以成为放大善意的工具。本应用致力于：

- **鼓励实践:** 提供易于参与的善行挑战，将善意融入日常。
- **促进反思:** 引导用户记录和思考善行带来的感受与影响。
- **提供陪伴:** 通过 AI 伙伴（调用 API）给予情感支持和积极反馈。
- **可视化进步:** 让用户看到自己善行积累带来的积极变化。
- **尊重隐私:** 将用户数据安全和隐私保护放在首位，明确告知 API 使用情况。

✨ 核心特性 (MVP - 最小可行产品)

MVP 目标: 快速验证核心概念，提供稳定流畅的基础体验，集成基础 API 实现 AI 特色功能。

1. 💖 **情感化设计与用户体验 (UX Focus)**
   - **描述:** 采用温暖、简洁的视觉风格，提供流畅直观的交互流程。
   - **实现思路:** PySide6 + QSS 样式 + Lottie 动画。
2. 🎯 **善行挑战浏览与订阅**
   - **描述:** 浏览、订阅和追踪预设的善行挑战。
   - **实现思路:** 本地 SQLite 数据库 (`challenges` , `user_challenges` 表)。
3. ✍️ **每日打卡与反思记录**
   - **描述:** 记录挑战完成情况和个人反思。
   - **实现思路:** 本地 SQLite 数据库 (`checkins` 表)。
4. 🏆 **基础进度与成就徽章**
   - **描述:** 可视化展示统计数据和解锁成就徽章。
   - **实现思路:** 本地 SQLite 数据库 (`achievements` , `user_achievements` 表) + 后端逻辑计算。
5. ⏰ **基础提醒功能**
   - **描述:** 设置每日固定时间提醒。
   - **实现思路:** APScheduler + 本地通知库 (如 `plyer`) + SQLite 存储设置。

6. 🔒 **本地优先与隐私保护 (核心数据)**
   - **描述:** 核心用户数据（挑战、打卡、反思、进度）默认存储在本地 SQLite。**AI 功能所需数据将发送至第三方 API 处理。**
   - **实现思路:** `sqlite3` 模块管理本地数据。**明确的隐私政策和用户同意流程是必需的。**
7. 🤖 **AI 电子宠物 (API 驱动)**
   - **描述:** 提供一个 AI 伙伴，通过调用**云端对话 API** (如 OpenAI GPT, Google Gemini) 和**情感分析 API**，根据用户的打卡行为和反思内容进行互动和鼓励。
   - **实现思路:**
     - **交互:** 用户输入或打卡事件触发 API 调用。
     - **技术:** `frontend/pet_ui.py` 展示界面。`ai_core/pet_handler.py` 负责调用外部 API，处理请求和响应。需要安全的 API Key 管理 (`config.py`)。
     - **API 选择:** 根据预算、性能和功能需求选择合适的 API 服务。
8. 📊 **AI 善举报告 (API 驱动)**
   - **描述:** 定期（如每周）通过调用**云端文本生成 API** (如 GPT) 生成简单的个性化文本总结报告。
   - **实现思路:**
     - **数据聚合:** 从本地 SQLite 查询相关数据。
     - **报告生成:** 将聚合数据和提示词发送给文本生成 API。`ai_core/report_generator.py` 负责 API 调用。
     - **可视化 (可选):** 仍可使用 Matplotlib 在本地生成图表嵌入报告（图片），或考虑使用 API 生成图表（如果 API 支持）。

🚀 未来增强特性 (Post-MVP - API 驱动)

这些功能将在核心体验稳定后，继续通过**调用云端 API** 来探索和添加：

- 🎯 **AI 个性化推荐 (API 驱动):**
  - **描述:** 调用**推荐系统 API** 或结合**文本分析/向量数据库 API**，根据用户历史行为和偏好推荐挑战。
  - **实现思路:** `ai_core/recommendation_engine.py` 封装 API 调用逻辑。可能需要将部分用户行为数据（匿名化处理后）发送给 API。
- 🎮 **AI 优化激励机制 (API 驱动):**
  - **描述:** 调用**机器学习 API** 或 **分析服务**，分析用户行为模式，为动态调整游戏化元素提供建议或自动化（需谨慎设计）。
  - **实现思路:** `ai_core/gamification_optimizer.py` 封装 API 调用。需要仔细考虑数据隐私和伦理问题。
- 🤝 **匿名善意墙 (后端 API):**
  - **描述:** 用户可选分享匿名感想至公共墙（只读）。

- **实现思路:** 需要一个简单的**自建后端 API** (Flask/FastAPI) 和云数据库 (Firebase/Supabase) 来处理分享和读取。**与 AI 功能的第三方 API 分开。**

🛠️ 技术栈 (API 优先)

- **语言:** Python 3.10+
- **GUI:** PySide6 (LGPLv3 许可)
- **本地数据库:** SQLite3
- **任务调度:** APScheduler
- **HTTP 请求:** `requests` (用于调用所有 API)
- **AI API 客户端库 (可选):** `openai`, `google-generativeai` 等 (根据选择的 API 服务)
- **本地通知:** `plyer` (跨平台) 或特定平台库
- **打包:** PyInstaller / cx_Freeze
- **测试:** pytest, pytest-qt
- **[可选] 自建后端 API (社区功能):** Flask / FastAPI
- **[可选] 云数据库 (社区功能):** Firebase Firestore / Supabase

注意: 不再直接依赖 `TensorFlow`, `PyTorch`, `Hugging Face Transformers`, `scikit-learn`, `Surprise` 等重型本地 AI 库。

🔧 环境配置与运行

```
# 1. 克隆仓库
git clone <your-repo-url>
cd kindness_companion_app

# 2. 创建并激活 Conda 环境 (推荐)
conda create -n kindness_companion python=3.10
conda activate kindness_companion
# 或者使用 venv
# python -m venv venv
# source venv/bin/activate  # Linux/macOS
# .\venv\Scripts\activate  # Windows

# 3. 安装依赖 (requirements.txt 应只包含轻量级库)
pip install -r requirements.txt

# 4. 配置 API 密钥 (!!! 关键步骤 !!!)
# - 创建 config.py 文件 (可基于 config_template.py)。
# - 在 config.py 中安全地配置您需要使用的所有第三方 AI API 密钥。
# - 强烈建议使用环境变量或密钥管理服务来加载密钥，而不是直接写入 config.py。
# - !! 确保 config.py 在 .gitignore 中，切勿提交到 Git !!
```

```
# 5. 初始化数据库（如果需要）
# python init_db.py

# 6. 运行应用
python main.py
```

📂 项目结构 (API 优先)

```
kindness_companion_app/
├── frontend/                   # 前端 UI 模块（PySide6）
│   └── ... (同前)
├── backend/                    # 本地后端逻辑（数据库交互，本地业务逻辑）
│   ├── __init__.py
│   ├── database_manager.py     # SQLite 管理
│   ├── challenge_manager.py    # 挑战数据管理
│   ├── progress_tracker.py     # 打卡、成就逻辑（本地部分）
│   ├── reminder_scheduler.py   # 本地提醒调度
│   └── utils.py                # 本地工具函数
├── ai_core/                    # AI 核心功能模块（API 客户端封装）
│   ├── __init__.py
│   ├── api_client.py           # 封装通用的 API 请求逻辑（含错误处理，重试）
│   ├── pet_handler.py          # 调用对话/情感 API 实现宠物交互
│   ├── report_generator.py     # 调用文本生成 API 生成报告
│   ├── recommendation_engine.py # [未来] 调用推荐相关 API
│   ├── gamification_optimizer.py # [未来] 调用分析/优化相关 API
│   └── utils.py                # AI 相关工具函数（如数据格式化）
├── api/                        # [可选] 自建轻量级后端 API（用于社区功能）
│   └── ... (同前)
├── resources/                  # 应用资源（图标，图片，样式，动画）
│   └── ... (同前)
├── tests/                      # 测试代码
│   ├── test_backend/
│   ├── test_frontend/
│   ├── test_ai_core/           # 测试 API 封装逻辑（可能需要 mock API 调用）
│   └── test_api/               # [可选] 测试自建 API
├── main.py                     # 应用启动入口
├── requirements.txt            # Python 依赖列表（轻量级）
├── config_template.py          # API 密钥配置模板
├── init_db.py                  # [可选] 数据库初始化脚本
├── LICENSE                     # 开源许可证文件
```

```
├── README.md                    # 项目说明文档（本文件）
└── .gitignore                   # Git 忽略配置（确保包含 config.py 等）
```

🧪 测试策略

- **单元测试:** 测试 `backend` 逻辑、`ai_core` 中的 API 请求格式化和响应解析逻辑（可使用 `unittest.mock` 模拟 API 调用）。
- **GUI 测试:** 使用 `pytest-qt` 测试前端交互。
- **集成测试:** 测试 UI 操作触发 `ai_core` 调用并正确更新 UI 的流程。
- **API 集成测试 (手动/小范围):** 在开发环境中，使用真实的 API Key 测试少量调用，确保与实际 API 的集成正常。
- **手动测试:** 在目标平台进行全面的手动测试。

🤝 贡献指南

我们欢迎各种形式的贡献！

1. **报告 Bug:** 发现问题？请在 GitHub Issues 中详细描述问题、复现步骤和您的环境。
2. **提出建议:** 有改进功能的想法？欢迎在 GitHub Issues 中创建 "enhancement" 标签的 issue。
3. **贡献代码:**
   - Fork 本仓库。
   - 基于 `main` 分支创建您的特性分支 (`git checkout -b feature/your-amazing-feature`)。
   - 进行修改，确保代码风格一致 (可考虑引入 Black, Flake8)。
   - 为您的修改添加必要的测试。
   - 确保所有测试通过。
   - 提交您的更改 (`git commit -m 'feat: Add some amazing feature'`) - 遵循 Conventional Commits 规范更佳。
   - 将您的分支推送到 GitHub (`git push origin feature/your-amazing-feature`)。
   - 创建 Pull Request 到 `main` 分支，清晰描述您的改动。

📜 开源许可

本项目采用 **LGPLv3** 许可证。这意味着您可以自由使用、修改和分发此软件（或其衍生品），但如果您分发包含此代码的软件，您需要以 LGPLv3 或兼容的许可证开源您的修改部分，并允许用户替换库。详情请见 `LICENSE` 文件。PySide6 本身也使用 LGPLv3 等许可。

⚠️ 伦理与隐私声明 (API 优先版)

我们高度重视用户的信任和数据安全。由于本项目现在依赖第三方 API 实现 AI 功能，请特别注意：

- **数据本地优先 (非 AI 数据):** 核心用户数据（挑战、打卡、进度、设置等非 AI 直接处理的数据）默认存储在用户本地 SQLite 数据库。
- **第三方 API 数据处理:**
  - **明确告知:** 在用户首次使用任何需要调用 AI API 的功能前，必须 **清晰、显著地告知** 用户：
    - 哪些数据（例如，反思文本、使用模式）会被发送给第三方服务商（例如 OpenAI, Google）。
    - 数据传输的目的（例如，生成宠物对话、生成报告）。
    - 指向第三方服务商隐私政策的链接。
  - **用户同意: 必须获得用户的显式同意 (Opt-in)** 后，才能将用户数据发送给第三方 API。应提供方便的设置选项，允许用户随时撤销同意或禁用相关 AI 功能。
- **数据最小化:** 仅发送 API 完成任务所必需的最少量数据。
- **匿名化考虑:** 尽可能在发送前对数据进行匿名化处理，但需注意某些 AI 功能（如个性化）可能依赖部分非匿名信息。
- **API Key 安全:** 严格保护 API Key，不在代码库中存储明文 Key。使用配置文件、环境变量或安全的密钥管理方案。
- **责任边界:** 应用开发者对本地数据的安全负责，但对于数据发送到第三方 API 后的处理，需依赖第三方服务商的安全和隐私实践。用户需被告知这一点。
- **AI 使用透明:** 清晰标示应用中使用了 AI 的地方及其局限性。

📦 打包与分发 (macOS 使用 PyInstaller)

将 Python 应用打包成用户可以直接运行的可执行文件，可以极大地方便分发。以下是在 macOS 上使用 PyInstaller 的基本步骤：

1. **安装 PyInstaller:**

   ```
   pip install pyinstaller
   ```

2. **准备打包:**
   - 确保你的应用可以在当前环境中通过 `python main.py` 正常运行。
   - 确保 `requirements.txt` 是最新的，并且只包含必要的库。移除开发或测试专用的库可以减小打包体积。
   - **处理资源文件:** 图标、图片、样式表、Lottie 动画等非代码文件需要被包含进最终的包里。PyInstaller 通过 `.spec` 文件或命令行参数 `--add-data` 来处理。
3. **生成 `.spec` 文件 (推荐):**
   `.spec` 文件提供了比命令行参数更灵活、更可控的打包配置。

```
pyi-makespec --windowed --name KindnessCcompanion main.py
```

- `--windowed`：创建一个没有终端窗口的 GUI 应用（macOS 上会生成 `.app` 包）。
- `--name`：指定生成的可执行文件和 `.app` 包的名称。

4. **编辑 `.spec` 文件：**
   打开生成的 `KindnessCcompanion.spec` 文件，进行修改：

```
# -*- mode: python ; coding: utf-8 -*-

block_cipher = None

a = Analysis(['main.py'],
             pathex=['/path/to/your/project/kindness_companion_app'], # 确保包含项目根目录
             binaries=[],
             datas=[
                 ('resources', 'resources'), # 将整个 resources 文件夹复制到包内的 resources 目录
                 # 如果有其他需要包含的文件或目录，继续添加
                 # ('path/to/local/file', 'destination_folder_in_app')
             ],
             hiddenimports=[
                 'PySide6.QtCore',
                 'PySide6.QtGui',
                 'PySide6.QtWidgets',
                 'PySide6.QtNetwork', # 如果用到网络功能
                 'PySide6.QtSvg',     # 如果用到 SVG 图标
                 # 根据你的代码实际使用的 PySide6 模块添加
                 'pkg_resources.py2_warn', # 有时需要处理 setuptools 警告
                 'apscheduler.schedulers.blocking', # 如果用到 APScheduler
                 'requests',
                 # 添加你依赖的其他库中可能未被自动发现的隐藏导入
             ],
             hookspath=[],
             runtime_hooks=[],
             excludes=[],
             win_no_prefer_redirects=False,
             win_private_assemblies=False,
             cipher=block_cipher,
             noarchive=False)

pyz = PYZ(a.pure, a.zipped_data,
             cipher=block_cipher)
```

```
exe = EXE(pyz,
          a.scripts,
          [],
          exclude_binaries=True,
          name='KindnessCompanion', # 可执行文件名
          debug=False,
          bootloader_ignore_signals=False,
          strip=False,
          upx=True, # 可以尝试使用 UPX 压缩，但有时会引起问题
          runtime_tmpdir=None,
          console=False, # 对于 --windowed 应用，这里是 False
          icon='resources/icons/app_icon.icns') # 指定 macOS 应用图标
(.icns 文件)

coll = COLLECT(exe,
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=False,
               upx=True,
               upx_exclude=[],
               name='KindnessCompanion') # 最终生成的文件夹名称 (如果不用 --
onefile)

app = BUNDLE(coll,
             name='KindnessCompanion.app', # macOS 应用包名称
             icon='resources/icons/app_icon.icns', # 再次指定图标
             bundle_identifier='com.yourcompany.kindnesscompanion', # 应用
的唯一标识符
             info_plist={ # 可以添加更多 macOS 特定的 Info.plist 信息
                 'NSPrincipalClass': 'NSApplication',
                 'NSAppleScriptEnabled': False,
                 'CFBundleDisplayName': 'Kindness Companion', # 应用显示名
称
                 'LSMinimumSystemVersion': '10.15', # 指定最低 macOS 版本
             })
```

- **`pathex`**：确保包含你的项目源代码根目录。
- **`datas`**：这是关键！使用 `('source_path', 'destination_in_bundle')` 格式添加所有资源文件。`('resources', 'resources')` 会将项目中的 `resources` 文件夹及其内容复制到打包后的 `.app` 包内的 `Contents/Resources/resources` 目录下。你的代码需要能正确找到这些资源（通常使用相对路径或 PyInstaller 提供的 `sys._MEIPASS` 路径）。
- **`hiddenimports`**：PyInstaller 可能无法自动检测到所有需要的库，特别是像 PySide6 这样复杂的库。需要手动添加可能缺失的子模块。

- `icon`: 指定 `.icns` 格式的 macOS 应用图标文件。
- `BUNDLE`: 这是生成 macOS `.app` 包的关键部分。`bundle_identifier` 是必需的，用于 macOS 识别应用。`info_plist` 可以定制更多应用信息。

5. **执行打包:**
   在终端中，使用 `.spec` 文件进行打包：

```
pyinstaller KindnessCompanion.spec
```

   或者，如果不想编辑 `.spec` 文件（不推荐用于复杂项目），可以尝试命令行：

```
# 示例：单目录打包（推荐调试）
pyinstaller --windowed --name KindnessCcompanion \
            --add-data "resources:resources" \
            --hidden-import PySide6.QtCore --hidden-import PySide6.QtGui
--hidden-import PySide6.QtWidgets \
            --icon "resources/icons/app_icon.icns" \
            main.py

# 示例：单文件打包（分发时更方便，但启动稍慢，可能遇到问题）
# pyinstaller --onefile --windowed --name KindnessCompanion \
#             --add-data "resources:resources" \
#             --hidden-import PySide6.QtCore --hidden-import
PySide6.QtGui --hidden-import PySide6.QtWidgets \
#             --icon "resources/icons/app_icon.icns" \
#             main.py
```

6. **查找结果:**
   打包成功后，你会在项目目录下找到一个 `dist` 文件夹。里面会包含 `KindnessCompanion.app` (如果使用了 `--windowed`) 或一个包含可执行文件和依赖的文件夹。
7. **测试:**
   将 `dist` 文件夹（或里面的 `.app` 包）复制到 **另一台干净的 macOS 机器**（或虚拟机）上进行测试，确保所有功能正常，资源文件能被正确加载。
8. **代码签名与公证 (分发给他人):**
   如果你想将应用分发给其他 macOS 用户，他们可能会遇到“无法打开来自身份不明开发者的应用”的警告。为了解决这个问题，你需要：
   - 拥有一个 Apple Developer Program 账号。
   - 使用你的开发者证书对 `.app` 包进行**代码签名 (Code Signing)**。
   - 将签名的应用提交给 Apple 进行**公证 (Notarization)**。
     这是一个相对复杂的过程，需要使用 `codesign` 和 `notarytool` 等命令行工具。
     PyInstaller 本身不直接处理这个。

重要提示:

- **打包是一个迭代过程:** 很少能一次成功，需要根据错误信息不断调整 `.spec` 文件中的 `hiddenimports` 和 `datas`。
- **虚拟环境:** 强烈建议在干净的虚拟环境中进行打包，只安装必要的依赖，避免包含不必要的库。
- **PySide6 打包:** PySide6 的打包有时比较棘手，确保使用的 PyInstaller 和 PySide6 版本兼容。查阅最新的 PyInstaller 和 PySide6 文档或社区讨论可能会有帮助。
- **路径问题:** 代码中访问资源文件的方式需要特别注意。使用相对路径，或者结合 `sys.frozen` 和 `sys._MEIPASS` 来判断是在开发环境运行还是在打包后的环境运行，从而构建正确的资源路径。

联系方式

- **GitHub:** nju-zym (请通过 GitHub Issues 联系)
- **电子邮件:** 241880200@smail.nju.edu.cn (备用联系方式)