**创建React脚手架**

**全局安装**

可以通过以下命令全局安装Create React App：

npm install -g create-react-app

安装完成后，使用以下命令创建项目：

create-react-app 项目名称

然后进入项目目录并启动项目：

cd 项目名称

npm start

**使用npx安装**

npx是npm 5.2.0版本后新增的命令，用于简化工具包的使用。使用npx安装Create React App的命令如下：

npx create-react-app 项目名称

安装完成后，进入项目目录并启动项目：

cd 项目名称

npm start

**安装与使用**

要安装FastAPI脚手架，可以使用以下命令：

pip install fastapi-scaf

安装完成后，可以使用以下命令来查看帮助文档：

fastapi-scaf -h

**创建新项目**

要创建一个新项目，可以使用以下命令：

fastapi-scaf new <myproj>

**添加API**

进入项目根目录后，可以使用以下命令添加新的API：

fastapi-scaf add <myapi>

**运行项目**

进入项目根目录后，执行以下命令来安装依赖并运行项目：

pip install -r requirements.txt

uvicorn app.main:app --host=0.0.0.0 --port=8000 --log-level=info --log-config=./config/uvicorn\_logging.json --workers=5

**项目结构**

FastAPI脚手架生成的项目结构如下：

└── fastapi-scaf

├── app

│ ├── api

│ │ └── v1

│ ├── business

│ ├── datatype

│ ├── initializer

│ │ ├── conf

│ │ ├── db

│ │ ├── logger

│ │ └──

│ ├── middleware

│ ├── router

│ ├── utils

│ └── main.py

├── config

├── deploy

├── docs

├── log

├── .gitignore

├── LICENSE

├── README.md

└── requirements.txt

这种结构将项目分为API、业务逻辑、数据类型、初始化、中间件、路由和工具等模块，便于管理和维护[1](https://blog.csdn.net/atpuxiner/article/details/140845902" \t "_blank)[2](https://cloud.tencent.com/developer/article/1894327)。

登录数据库

**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjE5MjUwMTUwODQ5ODQzMDc3MTIiLCJwaG9uZSI6IjE4ODQ4MzgxNDAxIiwiZXhwIjoxNzUwMzg4OTQzfQ.3nL5xNPlUUnbOJCCys7F96MqH6ikVvuiiSy8zHpp9zU**

redis-server

* **Swagger UI**: <http://127.0.0.1:8000/docs>
* **ReDoc**: <http://127.0.0.1:8000/redoc>

1. 移除了轮询机制，改用 WebSocket 连接
2. 移除了加载状态（因为现在是实时推送）
3. 新消息会立即显示在列表顶部
4. 限制只显示最新的50条消息，防止内存占用过大

后端需要实现的 WebSocket 服务：

1. WebSocket 端点：ws://localhost:8000/ws/notifications
2. 消息格式（JSON）：

json

Apply to 02txt

{

  "accountName": "账户1",

  "transactionTime": "2024-01-20 10:30:45",

  "transactionHash": "0x1234...5678"

}

特点：

1. 实时性：消息会立即显示，没有延迟
2. 效率：不需要频繁的 HTTP 请求
3. 双向通信：如果需要，前端也可以发送消息到后端
4. 自动重连：如果连接断开，WebSocket 会自动尝试重连

如果你需要调整任何部分，比如：

1. WebSocket 的地址
2. 显示的消息数量限制
3. 消息的显示样式
4. 添加重连机制

请告诉我。

{

"accountName": "账户1",

"transactionTime": "2024-01-20 10:30:45",

"transactionHash": "0x1234...5678"

}

1. 根目录文件：

* runserver.py: 服务器启动文件，包含主程序入口和服务器配置
* requirements.txt: 项目依赖包列表
* README.md: 项目说明文档
* .gitignore: Git忽略文件配置
* LICENSE: 项目许可证文件

让我查看一下登录注册模块的具体文件：

Searched codebase for "login register auth user authentication" (backend/app/)

根据搜索结果，登录注册模块主要使用了以下文件：

1. 数据模型层 (app/datatype/user.py):

* User: 用户数据库模型，包含字段：
* id: 主键
* phone: 手机号（唯一索引）
* password: 密码
* jwt\_key: JWT密钥
* created\_at: 创建时间
* updated\_at: 更新时间
* UserLoginMdl: 登录请求模型
* UserTokenMdl: Token请求模型
* UserUpdateMdl: 用户更新模型

1. 业务逻辑层 (app/business/user.py):

* UserCreateBiz: 用户创建业务逻辑
* UserLoginBiz: 用户登录业务逻辑
* UserTokenBiz: Token生成业务逻辑
* UserUpdateBiz: 用户更新业务逻辑
* UserDeleteBiz: 用户删除业务逻辑

1. API接口层 (app/api/v1/user.py):

* /user (POST): 用户注册接口
* /user/login (POST): 用户登录接口

1. 认证中间件 (app/middleware/auth.py):

* JWTBearer: JWT认证中间件
* get\_current\_user: 获取当前用户信息的依赖函数

1. 工具类 (app/utils/auth.py):

* gen\_jwt: 生成JWT token
* verify\_jwt: 验证JWT token
* gen\_jwt\_key: 生成JWT密钥
* hash\_password: 密码加密
* verify\_password: 密码验证

登录注册流程：

1. 用户注册：

* 前端调用 /user POST接口
* 后端验证手机号格式
* 密码加密存储
* 生成JWT密钥
* 返回用户ID

1. 用户登录：

* 前端调用 /user/login POST接口
* 后端验证手机号和密码
* 生成新的JWT密钥和token
* 更新用户的JWT密钥
* 返回token给前端

1. 认证流程：

* 前端在请求头中携带token
* JWTBearer中间件验证token
* 验证通过后可以获取当前用户信息

安全特性：

* 使用bcrypt进行密码加密
* 使用JWT进行身份认证
* 每次登录生成新的JWT密钥
* 手机号格式验证
* 密码强度验证

需要我详细解释某个部分吗？

## [如何将文件上传到Git仓库](https://blog.csdn.net/qq_48592827/article/details/121221161)

[1](https://blog.csdn.net/qq_48592827/article/details/121221161)[2](https://blog.csdn.net/weixin_43682721/article/details/88570397)[3](https://cloud.tencent.com/developer/article/1504684)

要将本地文件上传到Git仓库，可以按照以下步骤进行操作：

**初始化本地仓库**

首先，打开你的项目文件夹，并在该文件夹中右键选择“Git Bash Here”以打开Git Bash窗口。然后，输入以下命令来初始化一个新的Git仓库：

git init

此命令将在项目文件夹中创建一个.git目录，用于跟踪和管理版本库[1](https://blog.csdn.net/qq_48592827/article/details/121221161" \t "https://cn.bing.com/_blank)。

**添加远程仓库**

接下来，复制你在Git平台（如GitHub或Gitee）上创建的仓库地址，并在Git Bash窗口中输入以下命令，将本地仓库与远程仓库关联：

git remote add origin 你的仓库地址

例如：

git remote add origin https://github.com/yourusername/yourrepository.git

**拉取远程仓库内容**

为了确保本地仓库与远程仓库同步，输入以下命令从远程仓库拉取内容：

git pull origin master

**添加文件到暂存区**

将本地项目的所有文件添加到Git的暂存区，输入以下命令：

git add

注意：命令中的.表示添加当前目录下的所有文件[2](https://blog.csdn.net/weixin_43682721/article/details/88570397" \t "https://cn.bing.com/_blank)。

**提交文件到本地仓库**

将暂存区的文件提交到本地仓库，输入以下命令：

git commit -m "提交项目"

其中，"提交项目"是提交说明，可以根据需要修改。

**推送文件到远程仓库**

最后，将本地仓库的内容推送到远程仓库，输入以下命令：

git push origin master

此命令将本地的master分支推送到远程仓库的master分支[3](https://cloud.tencent.com/developer/article/1504684" \t "https://cn.bing.com/_blank)。

完成以上步骤后，你可以在Git平台上刷新仓库页面，查看上传的文件。如果之后需要再次提交修改，只需重复执行git add .、git commit -m "提交说明"和git push origin master这三个步骤即可。

更新冲突解决方案

Cd [目标仓库副本目录]

Git stash 暂存本地·修改

更新 git pull

恢复本地修改 git stash pop

提交文件到缓存区 指定哪些被修改的文件作为该次的缓存

Cd [目标仓库副本记录]

查看被修改的文件列表 git status

提交文件到缓存区 git add [文件]

提交全部文件到缓存区 git add \*

查看具体文件修改内容 ：git diff [文件]

提交本地仓库

Git commot -m “更新日志”

提交远程仓库