前后端项目开发任务要求文档

一、文档目的

明确前后端分离项目（Django + Vite）开发中，关于 “技术栈工作方式” 和 “开发流程” 的具体要求，帮助清晰区分两者的差异并落地执行。

二、技术栈工作方式：理解 “技术如何运行”

技术栈工作方式是指前端（Vite）、后端（Django）在不同环境（开发 / 生产）中 “客观运行的原理和逻辑”，是实现功能的底层支撑。需重点掌握以下内容：

2.1 开发环境（本地开发时）的工作逻辑

2.1.1 前端（Vite）的运行机制

npm run dev 的本质前端项目 package.json 的 scripts 字段通常配置"dev": "vite"，因此 npm run dev 实际执行 vite 命令，作用是启动 Vite 开发服务器（vite server）。

“前端部署在 vite server” 的含义 Vite server 是临时本地服务器（默认地址 http://localhost:5173），用于托管前端未打包源码（如.vue、.js 文件），核心作用包括：

支持 “热模块替换（HMR）”：修改代码无需手动刷新，自动更新视图；

解析 ES 模块：直接处理源码，无需提前打包，提升开发效率。

Vite proxy 代理的作用与配置开发时前端（http://localhost:5173）与后端（Django 默认 http://localhost:8000）因 “域名 / 端口不同” 存在跨域问题，proxy 通过 “请求转发” 解决：

前端/api 请求被 Vite server 转发到后端地址，后端响应后再由 Vite server 返回前端（视为 “同源”，规避跨域限制）。

示例配置（vite.config.js）：

javascript

运行

```
export default defineConfig({
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:8000', // 后端地址
        changeOrigin: true, // 允许跨域
        rewrite: (path) => path.replace(/^\/api/, '') // 移除/api 前缀（按需配置）
      }
    }
  }
})
```

开发环境前后端交互流程① 前端 npm run dev 启动 vite server，浏览器访问 http://localhost:5173；② 前端发起 fetch('/api/auth/login')请求；③ Vite server 将请求转发到 http://localhost:8000/auth/login（Django 接口）；④ Django 处理后返回数据，Vite server 转发给前端，前端渲染页面。

2.1.2 后端（Django）的运行机制

开发时通过 python manage.py runserver 启动 Django 开发服务器（默认 http://localhost:8000），直接提供 API 接口，无需复杂部署。

2.2 生产环境（上线运行时）的工作逻辑

2.2.1 前端的部署与运行

bundle 的生成执行 npm run build 时，Vite 对源码进行压缩、合并、转译，生成静态文件

（.js、.css、.html 等），即 “bundle”。该产物体积小、加载快，适合生产环境。
前端部署将 dist 目录（含 bundle）部署到静态服务器（如 Nginx、阿里云 OSS、GitHub Pages），用户浏览器通过静态服务器 URL（如 https://example.com）下载 bundle 并运行。

2.2.2 后端的部署与运行

Django 通过 Docker 打包为镜像，部署到云服务器（如阿里云 ECS），启动后对外提供 API 接口（如 https://api.example.com）。

2.2.3 生产环境前后端交互流程

① 用户访问 https://example.com，浏览器下载 bundle 并执行；② 前端 fetch('https://api.example.com/auth/login')请求后端；③ 后端处理后返回数据，前端渲染页面；④ 跨域处理：Django 配置 django-cors-headers，允许前端域名（如 CORS_ALLOWED_ORIGINS = ['https://example.com']）。

三、开发流程：明确 “人如何推进项目”

开发流程是 “主观执行的步骤”，即从项目初始化到上线的完整行动指南，需按阶段有序推进。

3.1 项目初始化阶段：搭建基础框架

目标：创建前后端项目骨架，确保开发环境可正常工作。

3.1.1 规划文件结构

后端（Django）目录示例：

plaintext

```
project_backend/
├── manage.py
├── project_core/  # 核心配置
│   ├── settings.py
│   ├── urls.py
│   └── ...
├── apps/  # 业务模块
│   ├── auth/  # 登录注册
│   └── ...
└── requirements.txt  # 依赖列表
```

前端（Vite）目录示例：

plaintext

```
project_frontend/
├── node_modules/
├── src/
│   ├── api/  # 接口请求
│   ├── pages/  # 页面组件
│   ├── utils/  # 工具函数
│   └── main.js
├── vite.config.js  # Vite 配置
└── package.json
```

3.1.2 搭建基础框架

后端：django-admin startproject project_core .创建项目，python manage.py startapp auth 创建业务模块；

前端：npm create vite@latest 创建项目（选 Vue/JavaScript），npm install 安装依赖。

3.1.3 验证开发环境

启动后端：python manage.py runserver，访问 http://localhost:8000 确认正常；

启动前端：npm run dev，访问 http://localhost:5173 确认正常；

验证代理：前端发起 fetch('/api/hello')，后端添加/hello 接口，确认请求可正常交互。

3.2 功能实现阶段：以 “登录注册” 为例

目标：按 “先学基础→再做验证→最后落地业务” 的流程开发，确保技术和业务可靠。

3.2.1 学习基础知识

数据库基础：理解 “表、字段、记录”，掌握 “CRUD”（增删改查）操作；

Django ORM：学习通过 Model 定义表、通过 Model.objects.create()/get()操作数据；

前后端交互：了解 HTTP 方法（GET/POST）、状态码（200/401）、JSON 格式。

3.2.2 技术验证（demo 测试）

在单独 demo 项目中验证核心技术（避免污染主项目）：

测试 Django ORM：创建 User 模型（username、password），通过 makemigrations/migrate 生成表，测试 “新增 / 查询用户”；

测试数据库连接：分别连接 SQLite（默认）和 MySQL（配置 settings.py 的 DATABASES），确保均可正常读写。

3.2.3 业务落地（主项目实现）

后端实现：

① 定义 User 模型（或用 Django 内置 AbstractUser）；

② 配置路由：path('api/auth/login', login_view), path('api/auth/register', register_view)；

③ 编写视图函数：

注册：接收 username/password，用 User.objects.create_user()创建（自动加密密码）；

登录：接收 username/password，用 authenticate() 验证，返回 token（推荐 djangorestframework-simplejwt）。

前端实现：

① 在 src/api/auth.js 封装登录 / 注册请求（用 fetch/axios）；

② 创建 Login.vue 页面，实现表单输入与提交逻辑；

③ 处理响应：登录成功存储 token 并跳转，失败显示错误。

3.2.4 主动求助

遇到技术问题（如 “Django auth 密码加密逻辑”“JWT 刷新机制”）时，及时确认最佳实践，避免漏洞。

3.3 项目收尾阶段：确保可维护性和可部署性

目标：规范代码管理，完善文档，为上线做准备。

3.3.1 版本控制

提交代码：git add . → git commit -m "feat: 完成登录注册功能（前端表单+后端接口）"；

推送远程：git push origin main，备份到 GitHub/GitLab。

3.3.2 文档编写

复杂功能在./docs 目录添加文档，内容包括：

接口设计：URL、请求方法、参数 / 响应示例；

实现思路：如 “登录用 JWT，token 有效期 2 小时，存储在 localStorage”；

技术难点：如 “解决 Vite proxy 路径匹配问题”。

3.3.3 部署准备

前端：npm run build 生成 bundle，用 serve dist 测试本地静态部署；

后端：编写 Dockerfile，配置生产环境 settings.py（关闭 DEBUG、配置 MySQL、设置 CORS）；

锁定依赖：前端 package.json、后端 requirements.txt 固定版本。

四、核心区分：技术栈工作方式 vs 开发流程

维度	技术栈工作方式	开发流程
本质	技术自身的运行原理（客观规律）	人推进项目的步骤（主观行动）
关注重点	“vite server 如何运行”“proxy 如何跨域” 等	“先搭框架还是先写接口” 等步骤顺序
作用	解释 “为什么能运行”	指导 “如何一步步做完”
举例	开发环境中 vite proxy 转发请求的链路	实现登录功能时 “先学 ORM 再写视图”

五、执行建议

开发前：先理解 “技术栈工作方式”（如 proxy 配置），避免调试困难；

开发中：按 “开发流程” 推进，每个子任务完成后确认，避免遗漏；

上线前：核对 “生产环境技术逻辑” 和 “部署准备”，确保服务正常运行。