**Web应用开发手册（文字叙述版）**

**技术栈**: Django + Vue3 + PostgreSQL + DRF + Pinia

**1. 需求分析与设计**

**1.1 需求确认**

* **用户角色**：明确系统角色（如用户、管理员）及其权限范围。
* **功能清单**：用用户故事描述核心功能（例如：“用户可发布文章，并设置公开/私有状态”）。
* **非功能需求**：包括响应时间（如API需在500ms内返回）、数据安全性（如密码加密存储）、浏览器兼容性（支持Chrome/Firefox/Safari）。

**1.2 数据设计**

* **数据库模型**：
  + 使用Django的ORM定义模型，主表需包含基础字段（如ID、创建时间、更新时间）。
  + 关联关系使用外键（ForeignKey）或多对多字段（ManyToManyField），并明确on\_delete规则（如级联删除或置空）。
  + PostgreSQL优化：对频繁查询的字段建立索引，利用JSONField存储动态结构数据。
* **API数据结构**：所有接口返回统一格式，包含状态码（status）、提示信息（message）和业务数据（data）。

**2. 后端开发（Django）**

**2.1 模型与序列化器**

* **模型定义**：
  + 每个模型对应一张数据库表，字段类型需匹配业务需求（如CharField用于短文本，TextField用于长内容）。
  + 添加Meta类配置排序规则（如默认按创建时间倒序）。
* **序列化器**：
  + 使用DRF的ModelSerializer自动生成字段验证逻辑。
  + 自定义验证方法（如检查标题是否重复）需在序列化器中实现。

**2.2 视图与路由**

* **视图集**：
  + 继承ModelViewSet快速生成CRUD接口，覆写get\_queryset方法实现数据过滤（如仅返回当前用户的文章）。
  + 自定义动作（如“发布文章”）通过@action装饰器扩展，需指定HTTP方法（POST/PUT）和是否针对单个对象。
* **URL路由**：
  + 使用DRF的DefaultRouter自动生成RESTful风格路由（如/api/v1/posts/）。
  + 版本控制：URL路径中嵌入版本号（如/v1/），便于后续迭代兼容。

**2.3 安全与优化**

* **认证**：采用JWT（JSON Web Token），需配置登录接口返回access\_token和refresh\_token。
* **性能**：
  + 使用select\_related或prefetch\_related优化关联查询。
  + 高频接口添加缓存（如Redis存储热门文章列表）。

**3. 前端开发（Vue3）**

**3.1 项目结构**

* **组件化开发**：
  + 页面组件（如PostList.vue）放在views/目录，可复用组件（如Button.vue）放在components/。
  + 状态管理使用Pinia，按模块划分Store（如postStore管理文章数据）。
* **API交互**：
  + 封装Axios实例，统一设置基地址（BaseURL）和请求拦截器（如自动添加Token）。
  + 错误处理：拦截401错误跳转登录页，500错误显示友好提示。

**3.2 页面逻辑**

* **数据加载**：页面挂载时（onMounted）调用Pinia Action获取数据，加载状态用v-if或骨架屏优化体验。
* **表单提交**：
  + 使用v-model绑定表单数据，提交前调用Vuelidate进行客户端验证。
  + 提交成功后重置表单或跳转结果页。

**4. API规范**

**4.1 通用规则**

* **HTTP方法**：
  + GET：获取资源，POST：创建资源，PUT/PATCH：更新资源，DELETE：删除资源。
* **状态码**：
  + 200：成功，400：参数错误，401：未授权，404：资源不存在，500：服务器错误。

**4.2 端点示例**

* **文章管理**：
  + 获取分页列表：GET /api/v1/posts/?page=1&size=10
  + 创建文章：POST /api/v1/posts/（需JSON请求体）
  + 发布文章：POST /api/v1/posts/{id}/publish/

**5. 测试与部署**

**5.1 测试策略**

* **后端测试**：
  + 单元测试：验证模型方法和工具函数。
  + 接口测试：模拟HTTP请求，覆盖成功、失败、边界用例。
* **前端测试**：
  + 组件测试：验证Props和事件触发。
  + E2E测试：模拟用户操作流程（如登录→发布文章）。

**5.2 部署流程**

* **后端**：
  + 使用Gunicorn作为WSGI服务器，Nginx反向代理静态文件和API请求。
  + 环境变量（如数据库密码）通过.env文件管理。
* **前端**：
  + Vite构建生产环境代码，部署至Nginx或对象存储（如AWS S3）。

**6. 维护与监控**

* **日志**：Django日志按级别（INFO/ERROR）记录到文件，前端错误上报至Sentry。
* **监控**：Prometheus采集接口响应时间，Grafana可视化展示。
* **文档**：使用Swagger生成API文档，维护更新日志（CHANGELOG.md）。

**附：关键注意事项**

1. **代码评审**：合并请求（Merge Request）需至少一人审核通过。
2. **备份策略**：数据库每日全量备份，保留最近7天。
3. **依赖管理**：固定Python和Node.js依赖版本（如requirements.txt和package-lock.json）。

# Web应用开发手册

**技术栈**: Django 4.x + Vue3 + PostgreSQL + DRF + Pinia

## 1. 需求分析与设计

### 1.1 需求确认

* 明确用户角色（如普通用户、管理员）及其权限边界。
* 使用**用户故事**描述功能（例：作为用户，我可以发布文章，并设置可见性）。
* 文档化：需求文档需包含功能列表、非功能需求（性能、安全性）。

### 1.2 数据模型设计

* **数据库设计**：
  + 使用Django ORM定义模型，优先考虑CharField、ForeignKey、DateTimeField等。
  + PostgreSQL特性：利用ArrayField、JSONField存储非结构化数据。
* **API响应格式**：
* Json{  
   "status": "success/error",  
   "message": "操作描述",  
   "data": {} // 实际数据  
  }

## 2. 后端开发（Django）

### 2.1 模型与序列化器

* **模型示例**：
* Pythonclass Post(models.Model):  
   title = models.CharField(max\_length=200)  
   content = models.TextField()  
   author = models.ForeignKey(User, on\_delete=models.CASCADE)  
   created\_at = models.DateTimeField(auto\_now\_add=True)
* **序列化器**：
* Pythonclass PostSerializer(serializers.ModelSerializer):  
   class Meta:  
   model = Post  
   fields = '\_\_all\_\_'

### 2.2 视图与路由

* **视图集**（推荐ModelViewSet）：
* Pythonclass PostViewSet(viewsets.ModelViewSet):  
   queryset = Post.objects.all()  
   serializer\_class = PostSerializer
* **URL配置**：
* Pythonrouter = DefaultRouter()  
  router.register(r'posts', PostViewSet)  
  urlpatterns = [path('api/v1/', include(router.urls))]

### 2.3 自定义操作

* 使用@action扩展端点：
* Python@action(detail=True, methods=['post'])  
  def publish(self, request, pk=None):  
   post = self.get\_object()  
   post.is\_published = True  
   post.save()  
   return Response({"status": "success"})

## 3. 前端开发（Vue3）

### 3.1 项目结构

src/  
├── components/ // 可复用组件  
├── stores/ // Pinia状态管理  
├── views/ // 页面级组件  
├── api/ // 封装Axios请求  
└── router/ // 路由配置

### 3.2 API请求封装

JavaScript// api/post.js  
import axios from 'axios';  
  
export const getPosts = () => axios.get('/api/v1/posts/');  
export const createPost = (data) => axios.post('/api/v1/posts/', data);

### 3.3 状态管理（Pinia）

JavaScript// stores/postStore.js  
import { defineStore } from 'pinia';  
  
export const usePostStore = defineStore('post', {  
 state: () => ({ posts: [] }),  
 actions: {  
 async fetchPosts() {  
 this.posts = await getPosts();  
 }  
 }  
});

## 4. API规范

| 操作 | 方法 | 端点 | 说明 |
| --- | --- | --- | --- |
| 获取列表 | GET | /api/v1/posts/ | 支持分页?page=1 |
| 创建 | POST | /api/v1/posts/ | 需认证 |
| 更新 | PUT | /api/v1/posts/{id}/ | 全量更新 |
| 删除 | DELETE | /api/v1/posts/{id}/ | 返回204 No Content |
| 自定义动作 | POST | /api/v1/posts/{id}/publish/ | 如发布文章 |

## 5. 测试与部署

### 5.1 测试

* **后端**: Django单元测试（TestCase）+ DRF API测试（APITestCase）。
* **前端**: Jest（单元测试）+ Cypress（E2E测试）。

### 5.2 部署

* **后端**: Docker + Nginx + Gunicorn。
* **前端**: Vite构建，部署到Nginx或CDN。
* **数据库**: PostgreSQL配置连接池（如pgbouncer）。

## 6. 安全与监控

* **认证**: JWT（使用djangorestframework-simplejwt）。
* **日志**: Django日志记录到文件，前端错误上报（Sentry）。
* **监控**: Prometheus + Grafana监控接口性能。

**附：开发工具推荐**

* 版本控制: Git + GitHub/GitLab
* 接口调试: Postman/Insomnia
* 文档: Swagger + Markdown

**备注**: 本手册为精简版，实际开发中需根据项目需求调整细节。建议配合代码评审和CI/CD流程（如GitHub Actions）确保质量。

这份手册可直接打印或共享给团队成员，确保开发流程标准化。需要进一步扩展某部分时可随时补充！