以下是 **Mist Forest Valley 项目的代码结构** 和 **所有功能的简要总结**，包括代码内的名称，便于工作交接和理解。

**1. 项目目录结构总结**

mistforestvalley/

├── mistforestvalley-server/ # 后端服务目录

│ ├── routes/ # 后端 API 路由

│ │ ├── auth.js # 处理用户注册、登录、修改密码的路由

│ │ ├── admin.js # 管理员创建用户和重置密码的路由

│ │ ├── user.js # 用户个人信息、密码修改的路由

│ ├── db.js # 连接 SQLite 数据库的文件

│ ├── server.js # 启动 Node.js 服务器的文件

│ └── package.json # 后端项目的配置文件

├── mistforestvalley-web/ # 前端 Vue 项目目录

│ ├── src/ # 前端源代码

│ │ ├── assets/ # 存放静态资源（如图片等）

│ │ ├── components/ # 存放 Vue 组件

│ │ │ ├── Login.vue # 登录页面的 Vue 组件

│ │ ├── App.vue # Vue 应用的主组件

│ │ ├── main.js # Vue 应用的入口文件

│ ├── public/ # 静态资源（如 index.html）

│ ├── vite.config.js # Vite 配置文件

│ ├── package.json # 前端项目的配置文件

└── README.md # 项目的文档说明文件

**2. 主要功能总结**

**后端功能**

1. **用户注册** (/api/register)
   * 功能：处理新用户的注册请求。检查用户名是否已存在，并进行密码加密后存储。
   * 代码：auth.js
   * HTTP 方法：POST
   * 路由：/api/register
2. **用户登录** (/api/login)
   * 功能：验证用户的用户名和密码，生成 JWT 令牌并返回给前端。
   * 代码：auth.js
   * HTTP 方法：POST
   * 路由：/api/login
3. **密码修改** (/api/change-password)
   * 功能：处理用户修改密码的请求，确保用户输入的旧密码正确后，进行新密码的加密并更新。
   * 代码：auth.js
   * HTTP 方法：PATCH
   * 路由：/api/change-password
4. **管理员创建用户** (/api/admin/create-user)
   * 功能：管理员通过此接口创建新用户，并为其分配角色和初始密码。
   * 代码：admin.js
   * HTTP 方法：POST
   * 路由：/api/admin/create-user
5. **管理员重置密码** (/api/admin/reset-password)
   * 功能：管理员重置指定用户的密码，并可以为用户指定角色。
   * 代码：admin.js
   * HTTP 方法：POST
   * 路由：/api/admin/reset-password

**前端功能**

1. **登录界面**
   * 功能：用户输入用户名和密码进行登录，前端通过 fetch 请求后端登录接口 /api/login，成功后存储 JWT 令牌。
   * 代码：Login.vue
   * 主要函数：handleLogin
2. <template>
3. <div class="login-container">
4. <h2>用户登录</h2>
5. <input v-model="username" placeholder="用户名" />
6. <input v-model="password" type="password" placeholder="密码" />
7. <button @click="handleLogin">登录</button>
8. <p v-if="errorMessage" class="error">{{ errorMessage }}</p>
9. </div>
10. </template>
11. <script setup>
12. import { ref } from "vue";
13. const username = ref("");
14. const password = ref("");
15. const errorMessage = ref("");
16. const handleLogin = async () => {
17. try {
18. const response = await fetch("/api/login", { // Vite 代理会处理
19. method: "POST",
20. headers: {
21. "Content-Type": "application/json",
22. },
23. body: JSON.stringify({
24. username: username.value,
25. password: password.value,
26. }),
27. });
28. const data = await response.json();
29. if (response.ok) {
30. alert("登录成功！🎉");
31. console.log("Token:", data.token);
32. localStorage.setItem("token", data.token); // 存储 Token 以便后续请求使用
33. } else {
34. errorMessage.value = data.error || "登录失败";
35. }
36. } catch (error) {
37. errorMessage.value = "服务器连接失败";
38. }
39. };
40. </script>
41. <style>
42. .login-container {
43. display: flex;
44. flex-direction: column;
45. align-items: center;
46. gap: 10px;
47. margin-top: 50px;
48. }
49. .error {
50. color: red;
51. }
52. </style>

**跨域配置**

1. **CORS 配置**：
   * 使用 cors 中间件来解决跨域请求的问题，确保前端能够访问后端 API。
   * 代码：server.js
2. const cors = require("cors"); // 允许前端跨域请求
3. app.use(cors());

**JWT 令牌认证**

1. **JWT 生成与验证**：
   * 登录成功后，生成 JWT 令牌，并在后续的 API 请求中使用该令牌进行身份验证。
   * 代码：auth.js
   * 在每个需要身份验证的路由上，使用中间件 verifyToken 来验证 JWT 令牌。
2. // 在需要验证用户的 API 路由上使用 verifyToken 中间件
3. const verifyToken = require("../middlewares/authMiddleware");

**3. 后续计划功能**

1. **角色管理**：
   * 允许管理员为不同的用户分配角色（如 admin、farmer、npc、merfolk、junimo）。
   * 通过 admin.js 中的接口管理用户角色。
2. **用户信息展示**：
   * 登录后，展示用户的基本信息（如用户名、角色等），并允许用户修改个人信息。
3. **用户权限控制**：
   * 在后端 API 中根据用户的角色进行权限控制，确保不同角色的用户只能访问相应的资源。
4. **自动刷新 Token**：
   * 实现 Token 自动刷新机制，避免用户频繁登录。
5. **服务器部署与 HTTPS**：
   * 部署前后端到云服务器，并配置 SSL 证书，启用 HTTPS 安全连接。
   * 使用 **Let's Encrypt** 获取免费的 SSL 证书。
6. **性能优化与安全加固**：
   * 对前端和后端进行性能优化，使用缓存和压缩技术提升访问速度。
   * 加强后端安全性，防止 SQL 注入和跨站脚本攻击（XSS）。

以上是整个项目的简要总结，包含了已经实现的功能、代码结构、后续计划功能。你可以将这个总结复制到新的窗口继续工作，方便后续开发与管理。

下面是一份完整的项目与服务器配置总结文档，包含前后端、Nginx、Git 及其他相关配置信息，同时也包含了在后续调试时如何“询问足够信息后再开始推断错误原因”的说明。你可以复制到新窗口继续使用或分享给团队，作为参考文档。

**项目与服务器完整配置总结**

**一、项目整体概述**

本项目基于前后端分离架构：

* **前端**：使用 Vite + Vue（或 React）构建静态页面，生成的生产版本放在 dist/ 目录中。
* **后端**：使用 Node.js 与 Express 搭建 RESTful API，数据库采用 SQLite。
* **认证**：用户登录、注册及密码修改通过 Express API 实现，登录时返回 JWT 令牌。
* **版本管理**：使用 Git 进行前后端代码管理，并在服务器上通过 Git 拉取最新代码。
* **部署**：服务器使用 Ubuntu 22.04，Nginx 作为反向代理，配置 SSL、反向代理 API 请求至后端端口 5000。

**二、服务器与 Nginx 配置**

**1. 后端服务器（Express）**

* **目录**： /home/ubuntu/MistForestValley/mistforestvalley-server
* **主要文件**：
  + server.js  
    主要代码示例：
  + // include
  + const express = require('express');
  + const cors = require("cors");
  + const jwt = require("jsonwebtoken");
  + const app = express();
  + const userRoutes = require("./routes/user"); // 玩家 API
  + const authRoutes = require("./routes/auth"); // 用户 API
  + const adminRoutes = require("./routes/admin"); // 管理员 API
  + const SECRET\_KEY = "your\_secret\_key"; // 建议使用环境变量
  + app.use(cors());
  + app.use(express.json());
  + // 挂载用户相关 API（注意：authRoutes 中定义的 `/login`、`/register`、`/change-password` 等路由，挂载时会添加前缀 `/api`）
  + app.use("/api", authRoutes);
  + app.use("/api/user", userRoutes);
  + app.use("/api/admin", adminRoutes);
  + // 主页（简单返回欢迎信息）
  + app.get("/", (req, res) => {
  + res.send("欢迎来到雾森谷 : Mist Forest Valley！");
  + });
  + // 启动服务器：建议监听 0.0.0.0 以便让外部 Nginx 正确代理
  + app.listen(5000, "0.0.0.0", () => {
  + console.log("🌍 MistForestValley 后端运行在 http://127.0.0.1:5000");
  + });
* **路由文件**：
  + routes/auth.js 负责用户登录、注册、密码修改等。  
    示例（登录路由）：
  + const express = require("express");
  + const router = express.Router();
  + const bcrypt = require("bcrypt");
  + const jwt = require("jsonwebtoken");
  + const db = require("../db");
  + const SECRET\_KEY = "your\_secret\_key"; // 建议使用环境变量
  + // 用户登录
  + router.post("/login", (req, res) => {
  + const { username, password } = req.body;
  + if (!username || !password) {
  + return res.status(400).json({ error: "用户名和密码不能为空" });
  + }
  + db.get("SELECT \* FROM users WHERE username = ?", [username], (err, user) => {
  + if (err) return res.status(500).json({ error: "服务器错误" });
  + if (!user) return res.status(400).json({ error: "用户名或密码错误" });
  + bcrypt.compare(password, user.password, (err, isValid) => {
  + if (err) return res.status(500).json({ error: "服务器错误" });
  + if (!isValid) return res.status(400).json({ error: "用户名或密码错误" });
  + if (user.password\_changed === 0) {
  + return res.status(403).json({ error: "请修改密码", firstLogin: true });
  + }
  + const token = jwt.sign(
  + { userId: user.id, username: user.username, role: user.role },
  + SECRET\_KEY,
  + { expiresIn: "2h" }
  + );
  + res.json({
  + success: true,
  + message: "登录成功",
  + token,
  + userId: user.id,
  + role: user.role
  + });
  + });
  + });
  + });
  + module.exports = router;

**2. 前端服务器**

* **目录**： /home/ubuntu/MistForestValley/mistforestvalley-web
* **构建**：前端使用 Vite 构建，生成的生产文件在 dist/ 目录中。
* **Nginx 配置用于服务静态文件**。

**3. Nginx 配置**

* **配置文件位置**： /etc/nginx/sites-available/mistforestvalley
* **配置示例**：
* server {
* listen 80;
* server\_name mistforestvalley.club www.mistforestvalley.club;
* # API 代理设置：将 /api/ 的请求转发到后端 Express 服务器
* location /api/ {
* proxy\_pass http://127.0.0.1:5000;
* proxy\_set\_header Host $host;
* proxy\_set\_header X-Real-IP $remote\_addr;
* proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
* proxy\_set\_header X-Forwarded-Proto $scheme;
* }
* # 前端静态文件服务，指向构建输出的 dist 目录
* location / {
* root /home/ubuntu/MistForestValley/mistforestvalley-web/dist;
* index index.html;
* try\_files $uri /index.html;
* }
* }

**注意：**

* + proxy\_pass 指令中不带尾部斜杠（即写成 http://127.0.0.1:5000），这样 Nginx 会完整传递 /api/login 到后端，后端的挂载路径为 /api 后会匹配到 /login。
  + 如果遇到代理路径拼接问题，也可以尝试使用 rewrite 规则，但在本方案中我们使用完整传递的方式。
* **启用配置**：
  + 建立符号链接：
  + sudo ln -s /etc/nginx/sites-available/mistforestvalley /etc/nginx/sites-enabled/
  + 测试配置：
  + sudo nginx -t
  + 重启 Nginx：
  + sudo systemctl restart nginx

**4. 文件权限**

* 确保前端和后端文件目录权限正确，以免 Nginx 访问时出现 "Permission denied" 错误。
* sudo chown -R ubuntu:www-data /home/ubuntu/MistForestValley/mistforestvalley-web
* sudo chmod -R 755 /home/ubuntu/MistForestValley/mistforestvalley-web

**5. 进程管理**

* 使用 pm2 管理后端进程：
* sudo npm install -g pm2
* cd /home/ubuntu/MistForestValley/mistforestvalley-server
* pm2 start server.js --name mistforest-server
* pm2 save
* pm2 startup

**三、调试与问题排查指南**

在排查错误时，请务必遵循以下原则：

1. **询问足够信息再推断错误原因**：  
   在给出解决方案之前，首先确认以下信息：
   * 哪一步执行的结果与预期不符？（例如：Nginx 日志、后端返回结果、文件权限、进程状态等）
   * 是否有错误信息输出？请完整复制错误日志或截图。
   * 配置文件内容是否按照要求修改？请提供相关配置文件内容。
2. **逐步排查**：
   * 确认后端 API 是否正常运行（例如通过 curl -X POST http://127.0.0.1:5000/api/login）。
   * 确认 Nginx 配置是否正确、是否重启成功（使用 nginx -t 和 systemctl restart nginx）。
   * 检查 Nginx 日志（/var/log/nginx/access.log 和 /var/log/nginx/error.log）确认请求是否到达。
   * 检查前端代码中 API 请求地址是否正确（建议使用相对路径，如 fetch("/api/login")）。
3. **请在遇到问题时**：
   * 把相关的配置文件内容和错误日志完整复制出来。
   * 确认每个步骤执行后的状态，再继续排查下一步，避免过早推断错误原因。

**四、总结**

* **后端服务器**：Node.js + Express 提供 API 服务，挂载了 /api 路由（包含 /login、/register 等）。
* **前端服务器**：Vite 构建，生成的静态文件放在 dist/，由 Nginx 提供服务。
* **Nginx 代理**：将 /api/ 请求转发到 http://127.0.0.1:5000，文件服务目录设置为前端 dist/ 目录。
* **调试建议**：在每个调试环节，先询问并确认足够的信息，再开始分析错误原因。

这份文档包含了所有关键配置信息及调试步骤，请复制到新窗口使用。如果后续有任何问题，请先确认并提供足够的日志和配置信息，再开始进一步分析。

漏斗图

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。文本

AI 生成的内容可能不正确。文本

AI 生成的内容可能不正确。

电脑屏幕的截图

AI 生成的内容可能不正确。