软件配置与运维文档

1、配置管理

开发工具

- 本地浏览器
- IDE：Vscode
- 准备环境：Node.js 12+ ， npm
- 前端：Vue 3 + Vite框架（用node.js下载）
- 后端：使用java语言编写

配置运行

```
# 进入vue3-vite所在项目目录
# 安装依赖
npm install
# 运行
npm run dev
```

运行后可以直接启动前端，自动打开浏览器localhost:8080就可以访问了。

2、版本控制

我们使用Github作为我们的版本控制系统。

Git 是一种开源的分布式版本控制系统，它允许开发人员在本地存储库中进行版本控制，并支持多人协作和代码分支管理。GitHub 利用 Git 的功能，为开发人员提供了一个云端的代码托管和协作平台。通过 GitHub，我们可以创建代码仓库（Repositories），并在仓库中进行代码的提交、分支管理、代码合并等操作。GitHub 还提供了一系列的协作和协同工作功能，如问题追踪、代码审查、团队协作等。

基于 GitHub 进行版本控制的开发流程如下：

1. 创建代码仓库（Repository）：在 GitHub 上创建一个新的代码仓库，邀请小组成员成为合作者。
2. 克隆代码仓库：使用 Git 客户端工具将代码仓库克隆到本地开发环境中。可以使用以下命令克隆代码仓库：

   ```
   git clone <repository_url>
   ```

3. 创建分支（Branch）：根据开发任务或功能需求，创建一个新的分支。分支可以让开发人员在不影响主分支（如 master）的情况下进行独立的开发。

   ```
   git checkout <branch_name>
   ```

4. 开发和提交代码：在本地分支上进行代码开发，使用 Git 进行版本控制和提交。

```
git commit -m "Commit message"
```

5. 推送代码：将本地分支上的代码推送到 GitHub 服务器，以便团队其他成员进行协作或进行代码审查。可以使用下列命令，也可以直接在GitHub上选择本地上传。

```
git push origin <branch_name>
```

6. 提交 Pull Request：当开发完成并经过本地测试后，将代码分支推送到远程仓库，并在 GitHub 上创建一个 Pull Request。Pull Request 提供了一个方便的界面来讨论和审查代码，以及在主分支合并之前进行必要的修改和讨论。
7. 代码审查和合并：团队成员可以对 Pull Request 进行审查，并提供反馈和建议。一旦代码经过审查并被批准，可以将分支合并到主分支或其他指定分支中。

3、持续集成

持续集成（Continuous Integration，简称 CI）是一种软件开发实践，旨在通过频繁地集成和测试代码来改善软件开发过程的质量和效率。它的核心思想是将团队成员的代码变更集成到共享的主干代码库中，并自动进行构建、测试和部署等过程，以尽早发现和解决可能的问题。

持续集成步骤如下：

1. 设置源代码管理：将代码托管到版本控制系统中，确保每个开发人员都可以访问和提交代码。
2. 创建构建脚本：在代码仓库中添加构建脚本，使用npm脚本。构建脚本包含构建、编译和打包代码的步骤。
3. 编写自动化测试：编写适当的自动化测试，包括单元测试、集成测试和端到端测试等。确保测试覆盖代码的不同方面，并在构建过程中执行这些测试。
4. 设置构建步骤：在持续集成工具中配置构建步骤，包括检出代码、安装依赖项、运行构建脚本和执行自动化测试等。确保每个步骤都正确配置，并且能够成功执行。

4、部署和运维计划

部署和运维计划描述了软件的部署过程以及在生产环境中进行运维的计划和策略。三饭订餐系统的部署和运维计划包括以下内容：

1. 环境规划：
    - 硬件需求：确定所需的服务器、网络设备和POS终端等硬件资源。
    - 软件需求：确定所需的操作系统、数据库和订餐系统软件的版本要求。
2. 服务器配置：
    - 服务器设置：配置服务器的操作系统、网络设置和安全设置。
    - 服务器部署：安装所需的软件、库和依赖项，并进行必要的配置。
3. 数据库部署：
    - 数据库选择：选择适合的数据库系统，如MySQL。
    - 数据库配置：配置数据库的参数和权限，创建数据库和表结构。
    - 数据备份和恢复：制定定期备份数据库的计划，并确保备份的安全性和可恢复性。
4. 应用程序部署：
    - 代码获取：从版本控制系统（Github）中获取代码，并确保代码是最新的。
    - 编译和打包：根据项目的构建方式，执行编译和打包操作。

- 部署配置：配置应用程序的相关参数，如数据库连接、日志路径等。
- 应用程序启动：启动应用程序并确保其正常运行。

5. 监控和日志：
   - 系统监控：设置监控系统，监测服务器的性能指标（如CPU使用率、内存使用率等）。
   - 应用程序监控：监测应用程序的运行状态、请求量和响应时间等指标。
   - 日志管理：配置应用程序的日志记录，确保日志的完整性和安全性。
6. 安全管理：
   - 身份验证和访问控制：设置用户认证和权限管理，确保只有授权用户（即学校的师生）可以访问系统。
   - 数据加密：使用合适的加密算法保护用户敏感信息的安全性。
   - 漏洞修复：及时更新软件和补丁，修复已知的安全漏洞。
7. 运维计划：
   - 日常运维任务：列出每日、每周和每月的运维任务，如日志清理、数据库备份、系统巡检等。
   - 定期维护：规划系统定期维护的时间和内容，如系统升级、数据库优化等。
   - 紧急故障处理：制定故障响应计划，包括故障报警、紧急修复和恢复操作。