吉首大学



码

客

之

家

合

作

规

范

文

档

|  |  |
| --- | --- |
| 成 员： | 舒镐、周鑫、于亮 |
| 班 级： | 20计科一班 |
| 学 号： | 2020404102、2020404101、2022200565 |
| 时 间： | 2023年6月21日 |

目录

[1. 团队协作 1](#_Toc1708104620)

[2. 项目管理 1](#_Toc552955244)

[2.1管理工具 1](#_Toc1357045339)

[2.2 开发方法 2](#_Toc1584681433)

[2.3 技术栈 2](#_Toc648654337)

[2.3.1 前端 2](#_Toc1306449787)

[2.3.2 后端 2](#_Toc1628763181)

[2.3.3 数据库 2](#_Toc648734758)

[3. 开发规范 2](#_Toc510601887)

[3.1文档规范 2](#_Toc341261397)

[3.1.1文档格式 2](#_Toc1798961889)

[3.1.2 文档分类 3](#_Toc730202310)

[3.2 团队协作规范 3](#_Toc1788665212)

[3.2.1 沟通与表达 3](#_Toc1620127378)

[3.2.2 任务分配和管理 3](#_Toc1535681733)

[3.2.3 代码审查和质量保证 3](#_Toc1744416885)

[3.3 代码格式规范 3](#_Toc967837351)

[3.3.1前端规范 3](#_Toc1401215879)

[3.3.2后端规范 4](#_Toc929605351)

[3.4 编码风格规范 4](#_Toc933602332)

[3.4.1 前端规范 4](#_Toc1538868942)

[3.4.2 后端规范 4](#_Toc1624747373)

[3.5 版本控制规范 4](#_Toc1874526406)

[3.5.1 代码提交规范 4](#_Toc1580204152)

[3.5.2 分支管理规范 5](#_Toc560920215)

1. 团队协作
   1. 团队成员  
       组长：舒镐

组员：于亮、周鑫

1. 项目管理

2.1管理工具

使用git分布式版本控制系统，协作开发和管理代码的变更。

2.2 开发方法

DevOps方法论

（1）快速交付：通过自动化和持续集成/持续交付（CI/CD）实践，加快软件的交付速度，减少发布周期，从而能够更快地满足用户需求；

（2）自动化：利用自动化工具和流程，减少人工操作，提高效率，并减少因人为错误而引入的问题；

（3）持续改进：通过不断的反馈、监控和数据分析，识别问题、瓶颈和改进机会，并在持续改进的基础上不断提高软件交付和运维过程。

（4）使用工具：Jenkins/Github Actions+Docker+Gitlab/Github/Gitee+Vercel

2.3 技术栈

2.3.1 前端

React

Next.js + Axios

2.3.2 后端

Java

SpringBoot + Mybatis

2.3.3 数据库

MySQL

1. 开发规范

3.1文档规范

3.1.1文档格式

|  |  |  |  |
| --- | --- | --- | --- |
| 内容种类 | 格式种类 | | |
| 字体 | 字号 | 间距 |
| 文档名 | 宋体（标题） | 二号 | 1.5倍间距 |
| 一级标题 | 宋体（标题） | 三号 | 1.5倍间距 |
| 二级标题 | 宋体（标题） | 四号 | 1.5倍间距 |
| 三级标题 | 宋体（标题） | 小四 | 1.5倍间距 |
| 正文 | 宋体（正文） | 小四 | 1.5倍间距 |

3.1.2 文档分类

|  |  |
| --- | --- |
| Object-Oriented Analysis | 需求分析文档 |
| Object-Oriented Design | 概要设计文档 |
| 详细设计文档 |

3.2 团队协作规范

3.2.1 沟通与表达

（1）使用明确、准确的语言进行沟通，避免模棱两可或含糊不清的表达；

（2）尊重他人意见，积极倾听和参与讨论，避免打断他人。

3.2.2 任务分配和管理

（1）将任务分配清晰明确，并确保每个团队成员了解其职责和截止日期；

（2）使用项目管理工具来跟踪任务的进展和状态，并保持更新。

3.2.3 代码审查和质量保证

（1）进行代码审查，以确保代码质量和一致性，提供有益的反馈和建议；

（2）遵循编码规范和最佳实践，如良好的命名约定、注释和代码结构等。

3.3 代码格式规范

3.3.1前端规范

eslint + prettier + editorconfig

3.3.2后端规范

editorconfig

3.4 编码风格规范

3.4.1 前端规范

参考阿里巴巴前端规约及配套工具

3.4.2 后端规范

参考阿里巴巴java开发手册（嵩山版）

3.5 版本控制规范

3.5.1 代码提交规范

（1）每次提交都应提供有意义的提交信息，描述提交的目的和更改的内容；

（2）提交信息应该简明扼要，遵循一致的格式和风格，具体请参考Angular的代码提交规范。样例如下表所示：

|  |  |  |
| --- | --- | --- |
| 序号 | 目的关键词 | 含义以及所更改内容 |
| 1 | feat | 新增功能 |
| 2 | fix | 修复bug |
| 3 | docs | 文档提交 |
| 4 | style | 代码格式 |
| 5 | refactor | 代码重构 |
| 6 | perf | 性能优化 |
| 7 | test | 增加测试 |
| 8 | chore | 其他变动 |
| 9 | revert | 代码回退 |
| 10 | build | 打包构建 |

3.5.2 分支管理规范

（1）主分支（main branch）

1. 主分支的命名应当为main或master；

2. 主分支存放最稳定的可发布版本；

3. 禁止直接在主分支上进行开发，只允许与发布版本的合并。

（2）开发分支（develop branch）

1. 开发分支是主要的集成分支，用于团队成员的日常开发工作；

2. 所有的特性分支、bug修复分支等都应该从开发分支派生出来。

（3）特性分支（feature branch）

1. 每个新功能应该在独立的特性分支上进行开发；

2. 特性分支应该从基准分支拉取（主分支/开发分支），并且用描述性名称命名，例如feature/add-login-function；

3. 在特性分支上进行功能开发、测试和调试；

4. 完成分支后，将特性分支合并回主分支。

（4）bug修复分支（fix branch）

1. 修复bug应该在独立的bug修复分支上进行；
2. 修复分支的命名应该清晰、具有描述性，例如fix/login-crash；
3. 修复完成后，bug修复分支应该合并回开发分支
4. beta分支（beta branch）
5. 在beta分支上进行预发布测试，以验证软件在真实环境中的稳定性和功能。这包括内部团队成员和一部分外部用户的参与；
6. 鼓励预发布测试期间的用户提供反馈、报告问题和提出改进建议。这有助于发现潜在问题并改善用户体验；
7. 根据用户反馈和测试结果，在beta分支上修复已发现的问题和缺陷。修复后的代码应提交到beta分支；
8. 经过多轮测试和修复后，如果beta分支被认为足够稳定和可靠，可以将其合并回主分支，并基于主分支进行正式的发布。
9. 发布分支（release branch）
10. 发布分支用于准备发布新版本；
11. 在发布分支上进行最后的测试、版本号更新和文档生成等工作；
12. 发布分支可能会从开发分支派生出来，或者从特定的功能分支而来；
13. 发布完成后，发布分支应该合并回开发分支和主分支。