**Edivator**

**基于百度大小模型的团队协作智能编辑器**

**[技术流程文档]**

TABLE OF CONTENTS

[1. 概述 4](#_Toc172280809)

[1.1 目的 4](#_Toc172280810)

[1.2 范围 4](#_Toc172280811)

[1.3 文档结构 4](#_Toc172280812)

[2. 项目背景和目标 5](#_Toc172280813)

[2.1 项目背景 5](#_Toc172280814)

[2.2 项目目标 5](#_Toc172280815)

[3. 技术栈 5](#_Toc172280816)

[3.1 技术选型表格一览： 5](#_Toc172280817)

[3.2 重要技术介绍： 6](#_Toc172280818)

[4. 开发流程 7](#_Toc172280819)

[4.1 需求分析： 7](#_Toc172280820)

[4.2 系统设计： 7](#_Toc172280821)

[4.3 开发编码： 7](#_Toc172280822)

[4.4 测试验证： 8](#_Toc172280823)

[4.5 部署上线： 8](#_Toc172280824)

[5. 代码管理 8](#_Toc172280825)

[5.1 版本控制： 8](#_Toc172280826)

[5.2 代码规范： 9](#_Toc172280827)

[5.3 代码备份与恢复： 9](#_Toc172280828)

[6. 安全性措施 9](#_Toc172280829)

[6.1 架构安全性基本情况介绍 9](#_Toc172280830)

[6.2 外部常见攻击手段 9](#_Toc172280831)

[1. XSS 攻击 9](#_Toc172280832)

[2. 注入攻击 10](#_Toc172280833)

[3. CSRF攻击 10](#_Toc172280834)

[4. 其他攻击和漏洞 10](#_Toc172280835)

[6.3 攻击应对方式 10](#_Toc172280836)

[1. XSS攻击应对 10](#_Toc172280837)

[2.注入攻击应对 10](#_Toc172280838)

[3. CSRF攻击应对 11](#_Toc172280839)

[4. 安全开发实践 11](#_Toc172280840)

[7. 性能优化 11](#_Toc172280841)

[7.1 数据库优化 11](#_Toc172280842)

[7.2 前端性能优化 12](#_Toc172280843)

[7.3 服务器性能优化 12](#_Toc172280844)

[7.4 代码优化 12](#_Toc172280845)

[8. 测试流程 12](#_Toc172280846)

[8.1 单元测试 12](#_Toc172280847)

[8.2 集成测试 12](#_Toc172280848)

[8.3 用户验收测试： 13](#_Toc172280849)

[9. 总结 13](#_Toc172280850)

# 概述

本技术流程文档旨在详细描述在线文档编辑平台的开发过程中涉及的各项技术活动、步骤、方法和标准。该文档的目标是确保开发团队在整个项目生命周期中能够按照一致性和高效性的流程进行工作，从而实现系统的高质量交付和持续优化。

## 目的

本技术流程文档的主要目的是：

* 为开发团队提供清晰、详细的技术指导，确保项目按计划有序进行。
* 定义系统的技术架构和技术栈，明确各个阶段的开发流程和规范。
* 描述系统的环境配置、部署流程、代码管理和测试方法，确保系统的稳定性、安全性和高性能。
* 提供系统维护、性能优化、安全性措施等方面的技术细节，确保系统在上线后的可靠运行和持续改进。

## 范围

本技术流程文档覆盖了在线文档编辑平台的以下几个方面：

* **项目背景和目标**：概述项目的业务背景、目标和需求。
* **技术栈**：列出并解释项目使用的主要技术栈。
* **开发流程**：详细说明需求分析、设计、开发、测试、部署和维护等各阶段的具体步骤和方法。
* **代码管理**：介绍版本控制、代码提交、代码评审和合并的流程和规范。
* **安全性措施**：详细说明系统的安全性设计和防护措施。
* **性能优化**：描述系统的性能优化策略和方法。
* **测试流程**：提供详细的测试计划、测试用例和测试流程说明。
* **总结**：对描述文档的总结。

## 文档结构

本技术流程文档采用模块化的结构，每个章节分别介绍一个特定的技术主题。各章节之间相对独立，但也相互关联，形成一个完整的技术指导体系。文档的具体结构如下：

1. 概述
2. 项目背景和目标
3. 技术栈
4. 开发流程
5. 代码管理
6. 安全性措施
7. 性能优化
8. 测试流程
9. 总结

# 项目背景和目标

## 项目背景

在数字化时代，用户在日常学习和工作中常常需要编辑和处理大量的多模态信息。一个结合生成式 AI 能力的智能编辑器，能够处理图片、文本、表格、代码甚至音视频，并辅助用户进行内容创作，将极大地提升学习效率和用户体验。未来大型的 AI 原生应用将主要采用大小模型混合使用的方式，通过灵活选择和应用不同规模的模型，以适应各种复杂多变的应用场景，从而在性能和效率之间达到更好的平衡。

## 项目目标

本项目旨在设计和开发一个结合AI大模型与小模型的高智能度、体验流畅、功能全面、性能卓越的在线文档编辑器。项目要求基于百度飞桨AI Studio星河社区与百度智能云，充分利用AI Studio上的各类模型库与零代码模型产线，实现在线文档编辑器的开发和部署。通过利用百度提供的大小模型，项目将展示如何在不同场景下灵活应用这些模型，以达到最优的性能和效果。

# 技术栈

我们在此将此次开发用到的技术列举出来，并对其中的一些重要技术做出解释。

## 技术选型表格一览：

**表 3-1 技术选型**

|  |  |
| --- | --- |
| **模块** | **技术** |
| 后端应用框架 | Django |
| 前端应用框架 | Vue3 |
| 前端组件通信 | Pinia |
| 前端路由 | Vue-Router |
| 前端 HTTP 库 | Axios |
| 代码管理及版本控制 | Git |
| 开发工具 | Pycharm Vscode（后端） |
| Vscode（前端） |
| 数据库 | MySQL |
| 服务器部署 | Nginx |
| 前端脚本语言 | TypeScript |

## 重要技术介绍：

1. Django：

Django 是一个高级 Python Web 框架，旨在快速开发安全且可维护的 Web 应用。它提供了众多内置功能，如ORM（对象关系映射）、表单处理、用户认证和管理等，帮助开发者快速构建功能完备的 Web 应用。Django 的模块化设计和强大的社区支持，使其具备高扩展性和可靠性。通过使用 Django，我们可以确保系统具有良好的性能和安全性。

1. Vue3：

Vue3 是 Vue.js 的最新版本，具有显著的性能提升和更好的 TypeScript 支持。Vue3 引入了组合式 API，使代码组织更加灵活和可维护。其响应式设计和组件化架构，能够帮助开发者构建高效、灵活的用户界面。通过 Vue3，我们可以创建一个直观、用户友好的前端界面。

1. Pinia：

在前端组件通信方面，我们使用了 Pinia。Pinia 是 Vue.js 的一个状态管理库，用于在 Vue 应用中管理全局状态。Pinia 的 API 设计简洁，易于上手，并且对 TypeScript 有良好的支持，提供强类型检查。通过使用 Pinia，我们可以实现高效、可靠的状态管理，确保组件间的数据传递和同步。

1. Vue-Router：

前端路由管理由 Vue-Router 完成。Vue-Router 是 Vue.js 官方的路由管理器，支持动态路由匹配、嵌套路由和导航守卫。它可以帮助我们构建单页面应用（SPA），提供良好的用户体验和灵活的导航控制。

1. Axios：

Axios 是一个基于 Promise 的 HTTP 客户端，具有简洁的 API，易于进行 HTTP 请求操作。它支持并发请求、请求和响应拦截器，可以在请求发送前或响应接收后进行处理，如添加认证令牌或处理错误。通过使用 Axios，我们可以高效地与后端服务器进行数据交互。

1. Git：

Git 是一个分布式版本控制系统，支持创建和合并分支，使团队成员可以独立开发和测试新功能。它记录代码的每次变更，便于追踪和回滚到以前的版本。GitHub、GitLab 等平台进一步增强了 Git 的协作能力，通过代码共享和协同工作，提高团队开发效率。

1. IDE（PyCharm & VSCode）：

开发工具方面，后端开发使用 PyCharm 和 VSCode，前端开发使用 VSCode。PyCharm 是一款功能强大的 Python 开发工具，提供代码智能补全、调试、测试、项目管理等功能，适用于后端开发。VSCode 是一款轻量级且功能强大的代码编辑器，支持多种编程语言和插件扩展，适用于前端和后端开发。

1. MySQL：

MySQL 是一个广泛使用的开源关系型数据库管理系统，具有高效的查询性能和快速的数据处理能力。MySQL 提供数据备份、恢复和高可用性方案，确保数据的安全和可靠。它支持大规模数据存储和处理，能够满足不同规模的应用需求。

1. Nginx：

Nginx 是一个高性能的 HTTP 和反向代理服务器，广泛用于 Web 服务器和负载均衡器。Nginx 具有高并发处理能力，能够处理大量并发连接，适用于高流量的网站。它提供高效的静态文件服务，能够快速响应用户请求，通过负载均衡将请求分发到多个服务器，提高系统的可靠性和可用性。

1. TypeScript：

TypeScript 是 JavaScript 的超集，增加了静态类型定义，使代码在开发过程中能够进行类型检查，减少运行时错误。TypeScript 提供了面向对象编程的功能和强类型系统，增强了代码的可读性和可维护性。

# 开发流程

为了确保项目开发的顺利进行，我们制定了一套完整的开发流程，包括需求分析、系统设计、开发实施、测试验证、部署上线等各个环节。

## 需求分析：

1. 需求收集：团队成员从题目要求及赛题讲解处获取了最原始的需求
2. 需求分析：对收集到的需求进行整理和分析，明确系统的功能需求和非功能需求，并对一些不太明确的需求在讨论群里与出题方进行了确认。
3. 需求评审：团队成员最后进行探讨，确保所有需求明确、可行，并得到各个团队成员一致认可。

## 系统设计：

1. 系统架构设计：根据需求分析的结果，设计系统的整体架构，包括前端架构、后端架构和数据库架构。确定各模块之间的关系和接口。
2. 详细设计：详细设计各个模块的功能和实现方案，包括数据库设计、接口设计、业务逻辑设计等，描述各模块的详细实现方案。
3. 技术选型：根据系统需求和设计方案，选择合适的技术栈和工具，包括编程语言、框架、数据库、服务器等。确保所选技术能够满足系统的性能、安全性、可扩展性等要求。

## 开发编码：

1. 开发环境搭建：配置开发环境，包括安装所需的软件、工具和依赖包。搭建版本控制系统，确保代码的统一管理和协同开发。
2. 编码实现：根据详细设计文档，进行各模块的编码实现。编写单元测试，确保代码的正确性和稳定性。
3. 集成开发：将各模块进行集成，确保模块之间的接口和数据流畅通。进行集成测试，验证系统的整体功能和性能。

## 测试验证：

1. 测试计划：编写测试计划，确定测试范围、测试策略、测试方法和测试用例。确定测试环境和测试数据，确保测试的全面性和覆盖率。
2. 功能测试：进行功能测试，验证系统各模块的功能是否符合需求分析的结果。使用黑盒测试和白盒测试方法，发现和修复功能缺陷。
3. 性能测试：进行性能测试，验证系统在高并发和大数据量下的性能表现。进行负载测试、压力测试和容量测试，确保系统的性能和稳定性。

## 部署上线：

1. 系统部署：将系统部署到生产环境，进行上线前的最后检查。进行数据迁移和初始化，确保系统数据的完整性和一致性。
2. 上线验证：进行上线验证，邀请用户体验成型系统，确保系统在生产环境中的正常运行。验证系统的功能、性能和安全性，确保系统满足上线要求。

# 代码管理

在本项目中，我们采用了系统化的代码管理策略，以确保代码的质量、版本控制和协同开发。主要的代码管理工具是 Git，并结合 GitHub 进行协同开发和版本管理。以下是详细的代码管理流程和策略：

## 版本控制：

1. 使用 Git 进行版本控制：Git 是一个分布式版本控制系统，能够高效地管理代码的版本变更。 我们在项目开始时，初始化 Git 仓库，并将其托管在 GitHub 上，确保代码的安全性和可访问性。
2. 分支策略：

我们采用 Git Flow 分支模型，确保开发流程的有序进行。主要分支包括：

* main 分支：用于存放已经发布的生产环境代码。
* develop 分支：用于存放最新的开发代码，是下一个版本的主要分支。
* feature 分支：用于开发新功能，每个功能一个独立的分支，从 develop 分支创建，完成后合并回 develop 分支。
* release 分支：用于发布准备和最后的测试，从 develop 分支创建，准备发布时合并到 main 分支。
* hotfix 分支：用于紧急修复生产环境的严重问题，从 main 分支创建，修复完成后合并回 main 和 develop 分支。

1. 分支命名规范：

为确保分支的可读性和管理的便捷性，我们规定分支命名应当具有描述性和一致性。例如：

* feature/feature-name：新功能开发分支。
* release/release-version：发布分支。
* hotfix/hotfix-description：热修复分支。

## 代码规范：

1. 提交规范：

每次提交代码时，要求开发者编写清晰、简洁的提交说明，以描述本次提交的目的和修改内容。提交信息应遵循以下格式：

* 标题行：简要说明修改内容，不超过50个字符。
* 描述行：详细说明修改内容、修改原因和影响范围。

1. 代码提交频率：

开发者应当在完成一个相对独立的功能或修复后及时提交代码，避免一次性提交大量代码。确保每次提交的代码能够通过现有的单元测试和代码审查。

## 代码备份与恢复：

1. 代码备份：通过 GitHub 的仓库托管服务，确保代码的实时备份和多地存储。 定期将代码仓库的备份文件下载存档，确保在极端情况下仍能恢复代码。
2. 代码恢复：当发生代码丢失或破坏时，通过 Git 提供的版本控制和恢复功能，快速恢复到之前的稳定版本，确保项目的持续运行。

通过上述系统化的代码管理策略，我们能够确保项目开发过程中的代码质量、版本控制和协同开发的高效性和可靠性。

# 安全性措施

## 架构安全性基本情况介绍

系统的安全性措施是保护系统免受恶意访问和攻击的关键要素。其主要目标是确保敏感数据的机密性和完整性，同时防止未经授权的访问。安全架构的成功衡量标准是其应对已知和未知攻击方式的能力，以及是否具备可靠的安全策略和机制。

## 外部常见攻击手段

### XSS 攻击

跨站脚本攻击（XSS）是一种通过注入恶意脚本来攻击用户的技术。攻击者注入这些脚本，然后让用户的浏览器执行它们，从而实现攻击目标。XSS攻击分为两种主要类型：

反射型XSS：攻击者将恶意脚本注入到特定URL，诱使用户点击，从而触发攻击。恶意脚本通常用于窃取用户的Cookie等敏感信息。

存储型XSS：攻击者将恶意脚本存储在目标服务器上，用户浏览网页时，这些脚本会从服务器检索并执行，导致攻击。

### 注入攻击

注入攻击是黑客通过向应用程序注入恶意代码或命令来利用系统漏洞的方法。主要形式包括：

SQL注入：攻击者通过在输入中注入SQL命令来操纵数据库。这可能导致数据泄露、破坏数据库或获取非法访问。

OS注入：攻击者通过在输入中注入操作系统命令来控制服务器。这种攻击方式通常与弱点扫描和系统漏洞利用相结合。

### CSRF攻击

跨站点请求伪造（CSRF）攻击是一种攻击方式，攻击者通过利用受害者的权限以其名义执行操作，通常是在受害者不知情的情况下进行。这种攻击可能包括恶意操作，如更改密码、进行金融交易或发表帖子。

### 其他攻击和漏洞

HTML注释漏洞：开发人员在服务器端代码中使用HTML注释语法进行注释，但这些注释有时会在客户端浏览器中显示，为攻击者提供机会。

文件上传漏洞：文件上传功能可能存在漏洞，如果允许上传可执行的恶意程序，攻击者可能获取服务器控制权。

路径遍历攻击：攻击者试图通过使用相对路径来遍历系统目录，访问未授权的文件和目录。

## 攻击应对方式

### 1. XSS攻击应对

输入验证和过滤：实施输入验证是防止XSS攻击的首要任务。在前端和后端都要对用户输入的数据进行验证和过滤。过滤器可以删除或转义潜在的恶意脚本，如 <script> 标签或其他HTML特殊字符。这确保了用户提供的输入不会被解释为代码。

HTTPOnly 和 Secure 标志：将 Cookie 标记为 HTTPOnly 可以防止 JavaScript 访问它们，从而提高了用户会话的安全性。此外，设置 Secure 标志可以确保 Cookie 仅通过加密的HTTPS连接传输，防止敏感数据被窃取。

内容安全策略（CSP）：CSP是一个重要的安全策略，它允许您定义哪些内容可以执行，哪些不可以。通过配置CSP，可以减少XSS攻击的成功几率，因为它限制了允许加载的资源和域名。

### 2.注入攻击应对

参数化查询：使用参数化查询是防止SQL注入攻击的最佳方式。参数化查询将用户输入与SQL命令分开，确保用户输入不会被解释为SQL代码。现代的数据库访问框架和ORM（对象关系映射）工具通常支持参数化查询。

数据消毒：数据消毒是一种对用户输入数据进行处理的方法，以去除潜在的威胁。这包括过滤输入，删除或转义特殊字符，以确保用户提供的数据不会执行恶意操作。

### CSRF攻击应对

随机令牌：引入随机生成的令牌是防止CSRF攻击的一种常见方法。这个令牌将包括在每个请求中，服务器将验证令牌的合法性。这确保了攻击者无法伪造请求。

双重提交 Cookie：使用双重提交Cookie方法，服务器将在用户会话期间创建一个随机生成的令牌，将其存储在Cookie中，同时也在请求参数中包含这个令牌。服务器在接收请求时会比较这两个令牌，以确保请求的合法性。

系统使用 JSON Web Token (JWT) 进行身份验证。每次用户登录成功后，服务器会生成一个 JWT 并返回给客户端。客户端在每次请求时将 JWT 附加到请求头中，服务器通过校验 JWT 的有效性和完整性来确认用户身份，确保请求的安全性。 JWT 使用非对称加密算法签名，防止令牌被篡改，且令牌中包含了用户身份信息和权限，方便服务器进行快速验证。

Referer 检查：检查请求来源，验证其是否合法，以防止CSRF攻击。这可以通过检查HTTP请求头中的Referer域来实现。但要注意，有些代理服务器可能不发送Referer头，因此不应仅依赖这种方法。

### 安全开发实践

在应对外部攻击手段时，还需要采取以下安全开发实践：

安全编码：在编写代码时，采用最佳的安全编码实践，包括正确使用输入验证、不信任的数据处理、安全的密码存储等。

定期安全审查：对代码进行定期的安全审查和漏洞扫描，以及应用程序安全测试，可以帮助识别并纠正潜在的漏洞。

安全更新和补丁：及时安装安全更新和补丁，以保护系统免受已知漏洞的攻击。

以上安全措施和最佳实践有助于创建更安全的架构，减少系统受到外部攻击的风险，确保用户数据和隐私的安全性。随着安全威胁的不断演进，保持系统的安全性需要持续不断的努力和关注。

# 性能优化

为了确保系统在高负载条件下依然能够高效运行，我们计划实施一系列性能优化措施。这些措施覆盖了数据库优化、前端性能优化、服务器性能优化和代码优化等多个方面。

## 数据库优化

1. 索引优化：为数据库中的关键字段添加索引，特别是常用于查询、排序和连接的字段。通过优化索引，可以显著提高数据库查询性能。
2. 查询优化：优化数据库查询语句，避免全表扫描，尽量使用索引扫描、范围扫描等高效的查询方式。
3. 数据库分片：针对大数据量的应用场景，采用数据库分片技术，将数据分布到多个数据库实例中，提高并发处理能力和数据访问速度。

## 前端性能优化

1. 代码分割：采用代码分割技术，将前端代码按需加载，减少初始加载时间，提高页面响应速度。 使用 Webpack 等工具对代码进行打包和压缩，减少文件大小和请求次数。
2. 资源压缩与合并：对静态资源（如 CSS、JavaScript 和图片等）进行压缩和合并，减少文件大小和 HTTP 请求次数。 使用 gzip 或 Brotli 等压缩算法，将静态资源进行压缩后传输，提高传输效率。
3. 异步组件：使用 Vue.js 的异步组件功能，在需要时动态加载组件，减少初始加载时间，提高应用的性能。 异步组件能够按需加载，避免一次性加载所有组件，提高页面的加载速度和响应速度。
4. Vite 构建工具：采用 Vite 作为前端构建工具，以其快速的热模块替换（HMR）和高效的开发体验，提高开发效率。 Vite 使用 ES 模块进行构建，加快了项目的构建速度，减少了开发和生产环境下的编译时间。

## 服务器性能优化

1. 负载均衡：采用负载均衡技术，将用户请求分配到多个服务器实例中，避免单点压力过大，提高系统的并发处理能力。 配置 Nginx 或 HAProxy 等负载均衡器，均衡分发请求，确保服务器资源的高效利用。
2. 缓存机制：在服务器端使用缓存机制，对于频繁访问的数据和页面，进行缓存处理，减少数据库查询和计算的次数。 配置 CDN（内容分发网络），将静态资源分发到全球各地的 CDN 节点，提高资源的访问速度和用户体验。

## 代码优化

1. 算法优化：对关键算法和逻辑进行优化，选择高效的算法和数据结构，提升代码执行效率。使用时间复杂度和空间复杂度较低的算法，避免不必要的计算和资源消耗。

# 测试流程

为了确保系统的质量和稳定性，我们制定了一套完整的测试流程。该流程涵盖了从单元测试、集成测试到用户验收测试等多个阶段，确保系统在各个层面都能够正常运行，并满足预期的功能和性能要求。

## 单元测试

1. 定义测试用例：为每个功能模块编写详细的单元测试用例，确保所有核心功能和业务逻辑都得到充分覆盖。测试用例应包括正常操作、边界条件和异常情况，确保代码在各种情况下都能正常工作。
2. 代码覆盖率：定期检查测试覆盖率报告，找出未覆盖的代码部分，补充相应的测试用例。

## 集成测试

1. 集成测试计划：制定详细的集成测试计划，列出需要测试的系统组件和集成点，以及相应的测试用例和预期结果。 确保所有关键的系统集成点，包括数据库、第三方 API 和内部服务的集成都得到充分测试。
2. 测试环境：搭建与生产环境相似的测试环境，确保测试结果的可靠性和准确性。 使用 Docker、Kubernetes 等工具创建隔离的测试环境，方便进行多次测试和回归测试。

## 用户验收测试：

1. 用户验收测试计划：制定用户验收测试计划，列出需要测试的业务流程和功能模块，以及相应的测试用例和预期结果。确保所有关键的业务功能都得到充分测试，满足用户的需求和期望，体验让用户满意。
2. 测试用户选择：选择具有代表性的测试用户，进行用户验收测试，确保测试结果的可靠性和代表性。测试用户可以是实际的终端用户、客户代表或内部业务人员。
3. 验收测试执行：根据测试计划，执行用户验收测试，记录测试结果和用户反馈。对发现的问题进行修复和调整，确保系统满足用户的需求和预期。

# 总结

本功能架构文档详细描述了项目的设计目标、背景、关键功能模块和技术实现方案。通过对各个模块的详细设计和功能描述，我们明确了系统的整体架构和各个子系统之间的关系，为后续的开发和实施提供了清晰的指导。

在本项目中，我们采用了 Django 作为后端框架，Vue3 作为前端框架，结合 Pinia 进行前端组件通信，使用 Vue-Router 进行路由管理，并通过 Axios 进行前端 HTTP 请求。我们还使用了 Git 进行代码管理和版本控制，选用 MySQL 作为数据库，Nginx 作为服务器部署方案。这些技术的选型和应用，确保了系统的高效性、稳定性和可扩展性。

文档中还详细阐述了性能优化、安全性措施、测试流程和代码管理等内容，确保系统在各个方面都达到预期的性能指标和安全要求。通过性能优化措施，我们能够提升系统的响应速度和并发处理能力；通过安全性措施，我们能够保护用户数据的隐私和安全；通过完整的测试流程，我们能够确保系统的质量和稳定性。

总的来说，本功能架构文档为系统的设计和开发提供了全面的指导，确保了项目的顺利推进和高质量交付。在后续的开发过程中，我们将严格按照文档中的设计和要求进行实施，不断优化和完善系统，最终实现项目的目标和预期。通过精心设计和合理规划，我们有信心在项目上线后提供一个高效、稳定、安全的系统，满足用户的需求和期望。