**分 类 号：TP311.5**

**学校代码：11460**

**学 号：19130104**

**南京晓庄学院本科生毕业设计**

**基于Next.js的个人博客系统的设计与实现**

**Design and Implementation of a Personal Blogging System Based on Next.js**

**所属学院： 信息工程学院**

**所属专业： 计算机科学与技术**

**学生姓名： 顾佳凯**

**指导教师： 王小正**

**职 称： 讲师**

**研究起止日期：二○二二年十一月至二○二三年五月**

**二○二三年五月**

**学位论文独创性声明**

本人郑重声明：

1.坚持以“求实、创新”的科学精神从事研究工作。

2.本论文是我本人在导师指导下进行的研究工作和取得的成果。

3.本论文中除引文外，所有实验、数据和有关材料均是真实的。

4.本论文中除引文和致谢的内容外，不含其他人或者其它机构经发表或撰写过的研究成果。

5.其他同志对本研究所做的贡献均已在论文中作了声明并表达了谢意。

作者签名：

日   期：

指导教师签名：

日   期：

【摘要】

本文旨在设计并实现一个个人博客系统，为用户提供便捷、高效、现代化且成本低廉的内容创作与分享平台。采用 Next.js 作为前端框架，利用 Tailwind CSS 对页面样式进行设计，采用Supabase作为后端，以提供动态数据，渲染到页面展示，最后依托 Vercel 平台完成 DPS 开发流程，从而为用户提供稳定且易用的个人博客解决方案。

为确保博客系统具有高性能和优秀的用户体验，首先研究了 Next.js 的核心技术，如服务器端渲染（SSR）、静态生成（SSG），确保博客系统在性能、SEO 和用户体验方面具有优势。接下来采用 Tailwind CSS 框架快速构建美观且响应式的界面设计，满足各种设备和浏览器的兼容性需求。然后与supabase后端数据库结合，实现动态数据的页面渲染。同时，介绍了如何将系统部署到 Vercel 平台，利用其自动化的持续集成和部署（CI/CD）流程，实现博客系统的高可用性和可扩展性。

本文详细论述了个人博客系统的功能需求，包括文章撰写与发布、标签管理、评论与回复功能、利用 ChatGPT 自动生成文章摘要等。展示了整个开发过程，为开发者提供完整的技术实现参考。最后，通过对系统的测试和验证，证实了所设计实现的个人博客系统具备良好的性能和稳定性，满足现代个人博客需求。

【关键词】：个人博客; Next.js; Tailwind CSS; 持续集成与部署;

**【Abstract】**

The purpose of this paper is to design and implement a personal blog system that provides users with a convenient, efficient, modern and cost-effective platform for content creation and sharing. Next.js is used as the front-end framework, Tailwind CSS is used to style the pages, Supabase is used as the back-end to provide dynamic data to be rendered to the page display, and finally the Vercel platform is used to complete the DPS development process, thus providing a stable and easy-to-use personal blogging solution for users.

To ensure the blog system has high performance and excellent user experience, we first studied the core technologies of Next.js, such as server rendering (SSR) and static generation (SSG), to ensure the blog system has advantages in terms of performance, SEO and user experience. Next, Tailwind CSS framework was used to quickly build a beautiful and responsive interface design that meets the compatibility needs of various devices and browsers. Then it is combined with the supabase backend database to achieve dynamic data page rendering. It also describes how to deploy the system to the Vercel platform, using its automated continuous integration and deployment (CI/CD) process to achieve high availability and scalability of the blogging system.

This article discusses in detail the functional requirements of a personal blog system, including article writing and publishing, tag management, comment and reply functions, and automatic article summary generation using ChatGPT. The whole development process is demonstrated to provide developers with a complete technical implementation reference. Finally, through testing and verification of the system, it is confirmed that the designed and implemented personal blogging system has good performance and stability to meet the modern personal blogging requirements.

【Key words】: Personal blog; Next.js; Tailwind CSS; Continuous integration and deployment;

目录

[1. 绪论 1](#_Toc134102440)

[1.1. 背景与意义 1](#_Toc134102441)

[1.2. 研究现状 1](#_Toc134102442)

[1.3. 研究内容 4](#_Toc134102443)

[1.4. 本章小结 5](#_Toc134102444)

[2. 项目分析 6](#_Toc134102445)

[2.1. 可行性分析 6](#_Toc134102446)

[2.1.1. 技术可行性 6](#_Toc134102447)

[2.1.2. 经济可行性 6](#_Toc134102448)

[2.1.3. 操作可行性 6](#_Toc134102449)

[2.1.4. 时间可行性 6](#_Toc134102450)

[2.2. 需求分析 7](#_Toc134102451)

[2.2.1. 数据需求分析 7](#_Toc134102452)

[2.2.2. 性能需求分析 7](#_Toc134102453)

[2.3. 技术路线分析 7](#_Toc134102454)

[2.3.1. 开发模式分析 7](#_Toc134102455)

[2.3.2. 前端技术路线分析 8](#_Toc134102456)

[2.3.3. 后端技术路线分析 8](#_Toc134102457)

[2.4. 本章小结 9](#_Toc134102458)

[3. 设计方案 10](#_Toc134102459)

[3.1. 架构设计 10](#_Toc134102460)

[3.1.1. 系统总体架构设计 10](#_Toc134102461)

[3.1.2. 系统前端架构设计 11](#_Toc134102462)

[3.1.3. 系统后端架构设计 11](#_Toc134102463)

[3.2. 业务模型设计 12](#_Toc134102464)

[3.2.1. 前台业务流程设计 12](#_Toc134102465)

[3.2.2. 前台功能模块设计 14](#_Toc134102466)

[3.2.3. 后台业务流程设计 15](#_Toc134102467)

[3.2.4. 后台功能模块设计 15](#_Toc134102468)

[3.3. 数据库设计 17](#_Toc134102469)

[3.3.1. 数据库概念模型设计 17](#_Toc134102470)

[3.3.2. 数据库表结构设计 20](#_Toc134102471)

[3.4. 本章小结 22](#_Toc134102472)

[4. 系统实现 23](#_Toc134102473)

[4.1. ChatGPT自动总结文章摘要模块 23](#_Toc134102474)

[4.2. 评论模块实现 26](#_Toc134102475)

[4.2.1. Comment组件 26](#_Toc134102476)

[4.2.2. CommentForm组件 28](#_Toc134102477)

[4.2.3. CommentList组件 30](#_Toc134102478)

[4.3. 文章目录模块实现 32](#_Toc134102479)

[4.4. 本章小结 34](#_Toc134102480)

[5. 系统测试与部署 35](#_Toc134102481)

[5.1. 系统测试 35](#_Toc134102482)

[5.1.1. 功能模块测试 35](#_Toc134102483)

[5.1.2. 性能综合测试 37](#_Toc134102484)

[5.1.3. 网络安全测试 42](#_Toc134102485)

[5.1.4. 兼容性测试 43](#_Toc134102486)

[5.2. 系统部署 46](#_Toc134102487)

[**5.3.** **本章小结** 47](#_Toc134102488)

[6. 结论 48](#_Toc134102489)

[7. 参考文献 49](#_Toc134102490)

[8. 致谢 51](#_Toc134102491)

[9. 附录 52](#_Toc134102492)

# 

# 1.绪论

## 背景与意义

“在微信当道、移动互联网成为主流的时代，博客似乎已经死了。[1]”这是国人朱海涛在其一篇题名为《博客已死？移动互联网时代博客价值》文章中的开场白。此话不假，自从微信公众号出世以来，简中网的网站便开始慢慢死去，目前这一趋势仍然在进行中。

2018年网易博客宣布关闭，它的关停是博客时代正在远去的另一个例证。[2]

格格不入的博客在现代真的已经沦落到一无是处的境地了嘛？答案显然是否定的。

“博客有一个独特的特点，当你的文章质量足够高，有其独特价值的时候，你的博客会有不少回头客，这样的用户黏度是超过搜索量的。”好的博客会慢慢积累出一群忠诚的读者，因为笔者的文字引起了他们的共鸣。相较于信息时代爆炸的无用信息，好的博客简直就是一股清流，让你能更好地认识世界。

国内有学者提出“从博客到微信，是网络社交的窄化”的观点[3]，他认为随着微信的盛行，网络社交越来越缺乏对话和沟通的价值。确实，“微博和微信只是一盒快餐，而博客是一坛美酒，愈久弥香。[1]”因此，在2022年，博客依旧有其存在的必要和价值。

从零设计并实现一个个人博客系统的意义如下：

对个人而言，可以促进我的编程学习并能让我进一步坚持博客的初心—认真写作、认真思考，而不是把博客变为了玩具[4]，走向捣鼓插件、折腾配置的极端；另一个好处是个人能够成为内容的主人，而不是受制于平台，实现博客的安全[5]与自由[6]。

对社区而言，我将自己开发的个人博客系统开源出去，没准会得到别人的青睐，进而促进开源社区的繁荣。

## 研究现状

博客系统主要分为两大类，一类是动态博客框架，如大名鼎鼎的Wordpress[7]；另一类是静态博客框架[8]，如基于Go语言的Hugo博客。双方各自有其优点和缺点，对比维度概括起来，无外乎以下5个方面：

1. 用户友好程度
2. 可扩展性
3. 安全
4. 速度
5. 成本

全球有近40%的网站采用Wordpress这一动态博客框架，高使用率可以彰显出其对用户友好。它凭借着丰富的扩展和主题，几乎能满足普通用户的任何想象，因此5个方面中的1）和2）Wordpress可以得满分。

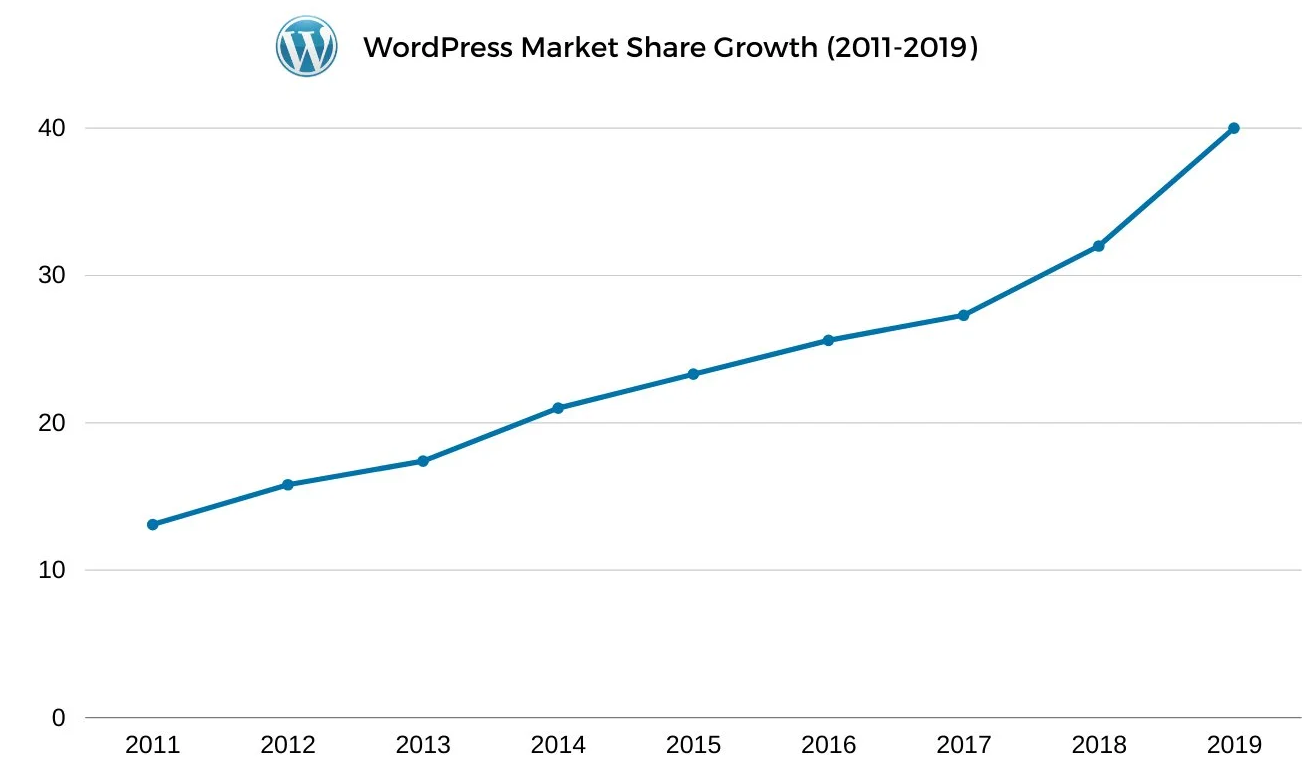


图1-1 Wordpress市场占比增长图

另外三个方面是Wordpress的缺点。①Wordpress会时不时爆出漏洞或托管主机会有遭黑客DDOS攻击的风险，因此安全性方面无法保证。②其速度方面稍逊于静态博客框架。③它需要有虚拟主机来托管，因此会耗费一些成本。

静态博客框架的优点是Wordpress的缺点，它的缺点反而是Wordpress的优点。以Hugo为例，Hugo生成站点的速度极其迅速，官网打出“The world’s fastest framework for building websites”的标语并不是浪得虚名。Hugo的快得益于Go语言性能的优良。

国外有位开发作者为此专门进行了一个实验来论证Hugo框架的快[9]，从实验数据来看，Hugo构建10000个页面只需10秒钟，相较于其他静态博客框架而言，快了不少。

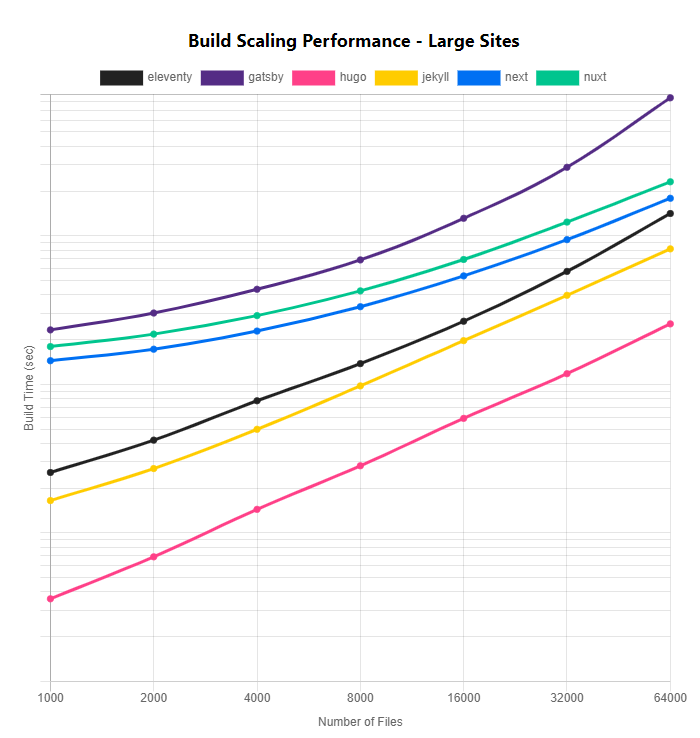


图1-2 静态博客框架实验数据

静态站点的成本可以忽略，因为市面上有很多免费的部署解决方案，如GitHub Pages、Vercel、Netlify等。在安全方面，静态站点不必担心被DDOS，因为它是纯静态页面。虽然静态博客实现动态交互效果可以通过第三方服务内嵌来得以实现，但这种策略与“从零开始进行个人博客系统的设计与实现理念”相悖，我个人更倾向于使用技术实现第三方服务类似的功能，以达到个人博客系统的自主可控。

随着 Web 技术的发展，前端框架和技术也在不断演进。从早期的 jQuery 到如今的 React、Vue 和 Angular 等现代前端框架，都在不断提高开发效率、优化用户体验。其中，Next.js 作为一种基于 React[10] 的服务端渲染（SSR[11]）框架，为博客系统的开发提供了更加优秀的性能和丰富的功能。Next.js的服务器端渲染（SSR[11]）和静态站点生成（SSG）功能使得集齐动态博客系统和静态博客系统的优点成为可能。

在博客系统的研究中，页面样式设计是一个重要的方面。随着响应式设计理念的普及，越来越多的博客系统开始采用自适应的布局和设计，以满足不同设备和浏览器的兼容性需求。Tailwind CSS 作为一种实用主义的 CSS 框架，为博客系统的样式设计提供了更加简洁、高效的解决方案。

随着云计算技术的发展，博客系统的部署和持续集成变得越来越简便。Vercel 平台作为一个支持多种前端框架和静态站点生成器的部署服务，为博客系统的部署提供了便捷的自动化流程，同时支持持续集成和部署（CI/CD[12]）功能，以提高开发效率和系统可用性。与此同时，其免费计划良心，对于个人博客系统的托管而言，绰绰有余。

当去年11月月底ChatGPT横空出世的那天起，就注定未来的各行各业会受到其波及。今年3月1日，ChatGPT宣布开放成本低廉的gpt-3.5-turbo API[13]，从那一刻开始，就注定ChatGPT+将成为未来的新趋势。

截至目前为止（2023.4.13），内容管理系统都陆陆续续地集成了chatgpt，如Wordpress的插件库中便已出现了ChatGPT相关的插件。因此我的个人博客系统也有必要集成ChatGPT，以实现个人博客系统的现代化。我的愿景是利用ChatGPT来自动生成文章摘要，帮助读者快速过滤信息。

综上所述，个人博客系统的研究现状表明，该领域仍在不断探索与发展，以满足不断变化的技术和用户需求。

## 研究内容

为了跟上时代的发展和满足用户的需求，本文的研究将着重在以下几个方面：

1. 利用 Next.js 和 Tailwind CSS 构建高性能、响应式的个人博客系统，提供优秀的用户体验。
2. 利用Supabase后端服务与前端页面进行联动，实现动态数据渲染至前端页面。

3） 将个人博客系统部署到 Vercel 平台，利用其自动化的持续集成和部署（CI/CD）流程，提高系统的可用性和可扩展性。

4） 研究并实现评论与回复功能、利用 ChatGPT 自动生成文章摘要等高级功能，提升博客系统的功能丰富性。

通过以上研究方向，本文旨在设计并实现一个现代化、高性能且功能丰富的个人博客系统，为用户提供便捷、高效的内容创作与分享平台。

## 本章小结

本章节首先描述了个人博客系统的研究背景与意义，接着综合分析了目前该领域的研究现状，最后结合研究现状提出了本文研究的重心，为下文展开对个人博客系统的具体描述埋下了伏笔。

# 项目分析

## 可行性分析

本节将从技术、经济、操作和时间四个方面分析项目的可行性。

### 技术可行性

通过使用成熟的技术栈，如Next.js、Tailwind CSS、Supabase和Vercel，项目在技术实现上是完全可行的。这些技术已经在许多应用和项目中得到广泛验证，具有良好的社区支持和丰富的文档。此外，ChatGPT的集成可以为项目增添现代化的功能。

### 经济可行性

本项目的开发成本主要在于开发者的时间投入。由于采用的技术栈和部署服务在免费计划下已经满足个人博客的需求，因此在经济方面具有可行性。此外，将项目开源可以为开发者带来知名度和其他潜在的收益。

### 操作可行性

项目采用的技术栈易于学习和上手，且具有良好的开发者体验。通过Next.js和Tailwind CSS的组合，可以方便地实现响应式和高性能的前端设计。Supabase作为后端服务提供了简单易用的API，便于与前端进行数据交互。Vercel平台的部署和持续集成则简化了项目的发布和更新过程。因此，项目在操作层面具有可行性。

### 时间可行性

通过使用成熟的技术栈和自动化部署流程，项目的开发周期可以得到有效缩短。另外，项目可以采用迭代的开发方式，逐步完善功能，以便在较短的时间内实现初始版本的发布。因此，在时间上，项目具有可行性。

综上所述，本项目从技术、经济、操作和时间四个方面具有可行性，为后续的开发奠定了基础。

## 需求分析

本节将从用例图、数据需求和性能需求三个方面对项目进行详细的需求分析，以确保项目的可靠性和实用性。

### 数据需求分析

1. 文章数据：包括标题、发布时间、标签、摘要、正文等。
2. 评论数据：包括评论者昵称、评论者邮箱、评论者网站（可选）、评论内容等。

为保证数据安全和可靠性，项目采用Supabase作为后端服务提供商，它提供了基于PostgreSQL的强大数据库功能，用于存储评论数据。

文章数据则交由Next.js来静态生成网页。

### 性能需求分析

为确保项目的性能和响应速度，需求分析如下：

1. 页面加载速度：为提高用户体验，页面加载速度应尽量优化，例如使用代码拆分等技术。
2. 响应式设计：网站需要适应各种设备和屏幕尺寸，采用Tailwind CSS实现响应式设计。
3. 服务器性能：使用Vercel平台自动扩展功能，确保服务器负载在高访问量时不影响性能。

通过以上需求分析，可以为项目的实施提供清晰的指导，为项目的顺利开展奠定基础。

## 技术路线分析

本节将从开发模式、前端技术和后端技术三个方面对项目的技术路线进行分析，以确保项目的技术选型符合需求。

### 开发模式分析

项目采用敏捷开发模式，以迅速响应需求变化和快速交付为核心目标。敏捷开发主要包括以下几个要点：

1. 分阶段交付：将项目分为若干个阶段，每个阶段分别实现一定的功能，并在GitHub的Project项目管理中进行标注。
2. 持续集成：采用自动化构建、测试和部署工具，确保项目的可持续发展。
3. 团队协作：项目成员（我）采用协作工具，如GitHub，进行任务分配、代码提交、问题反馈等。

### 前端技术路线分析

项目的前端技术选型主要包括以下几个方面：

1. 开发框架：采用Next.js框架，凭借其组件化思想和高性能特点，有助于提高开发效率和产品质量。
2. UI框架：使用Tailwind CSS作为UI框架，提供了丰富的样式和响应式设计，以满足不同设备的需求。
3. 路由管理：采用Next.js的普通路由和动态路由进行页面导航和路由控制。

### 后端技术路线分析

项目的后端技术选型主要包括以下几个方面：

1. 数据库：使用PostgreSQL数据库，具备高性能、可扩展性和安全性特点，满足项目的数据存储需求。
2. 后端服务：采用Supabase作为后端服务提供商，提供了一系列可扩展的API和数据库功能，简化后端开发。
3. 部署和扩展：使用Vercel平台进行项目部署，提供自动扩展功能，确保项目在高访问量下的稳定性。

通过以上技术路线分析，项目的技术选型旨在满足性能、可维护性、安全性等方面的要求，为项目的成功实施提供坚实的技术基础。

## 本章小结

本章对项目进行了全面的需求和技术分析。首先，通过四个方面的可行性分析（技术、经济、操作和时间），确认了项目的合理性和可行性。接下来，通过需求分析，深入挖掘了项目的用例图、数据需求和性能需求，为项目实施提供了明确的指导。最后，从技术路线的角度，分析了项目的开发模式、前端技术和后端技术选型，确保项目采用了最合适的技术方案。

综合以上分析，项目在需求和技术方面具备了明确的目标和实施基础，为后续的设计和开发工作奠定了基础。

# 设计方案

## 架构设计

在本项目中，我们采用了JAMstack[14]架构作为整体的技术框架。JAMstack（JavaScript、APIs、Markup）架构充分利用现代化前端技术，通过将前后端分离、利用APIs实现数据交互，提高了网站性能、安全性和易维护性。

### 系统总体架构设计

项目采用的JAMstack架构主要包括：前端展示层、后端服务层和数据存储层。前端展示层负责页面渲染和与用户的交互；后端服务层通过APIs提供数据和业务处理；数据存储层负责存储和管理各类数据。这三层之间的通信主要依赖于APIs，实现了良好的解耦和可扩展性。

前端展示层主要由Next.js和Tailwind CSS来把控，后端服务层包含Supabase的服务端以及第三方服务，如ChatGPT API。数据存储层由Supabase的服务端来提供支持。

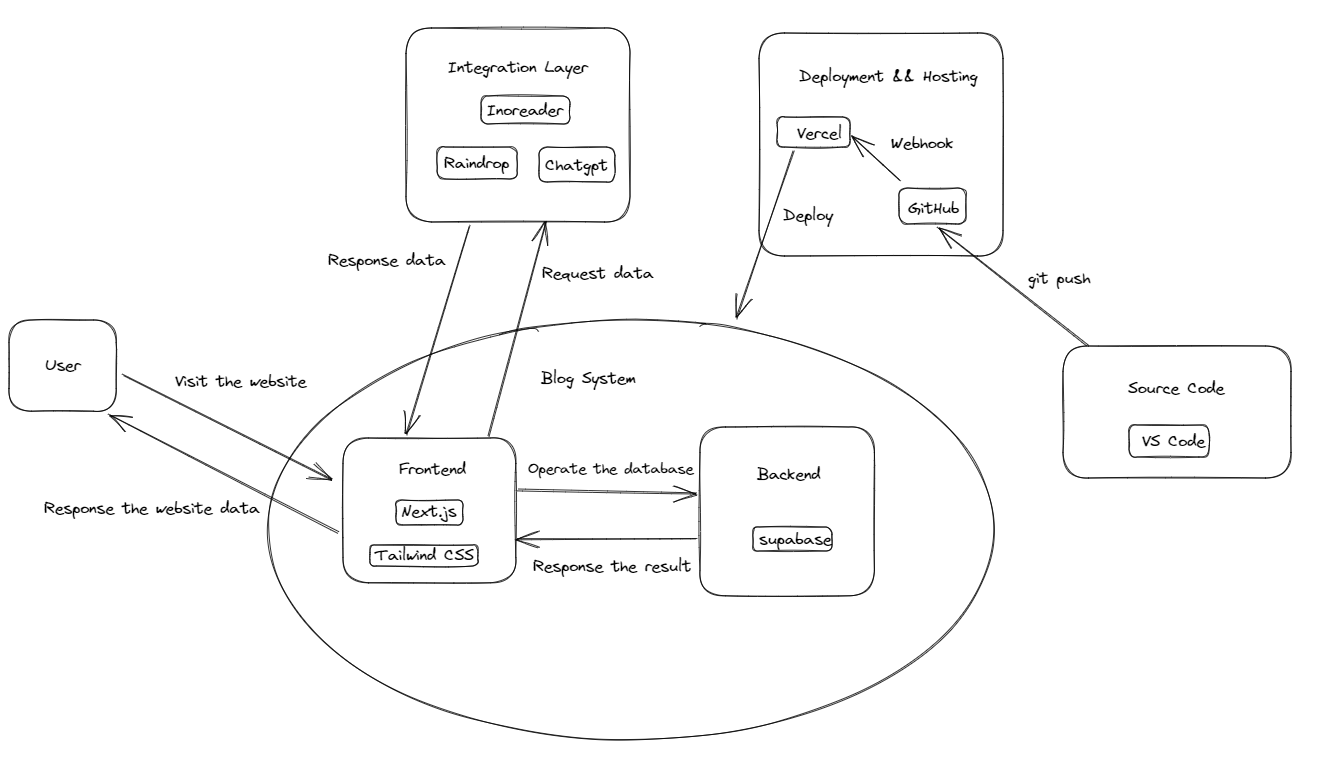


图3-1系统总体架构图

### 系统前端架构设计

前端采用Next.js框架和Tailwind CSS。Next.js是一款基于React的服务端渲染框架，提供了高效的页面渲染和优化功能。Tailwind CSS是一个实用性优先的CSS框架，便于快速构建响应式界面。结合Next.js和Tailwind CSS，前端能够实现高性能、可维护和易扩展的用户界面。

前端架构主要分为3个模块，文章模块、功能特色模块以及第三方模块。

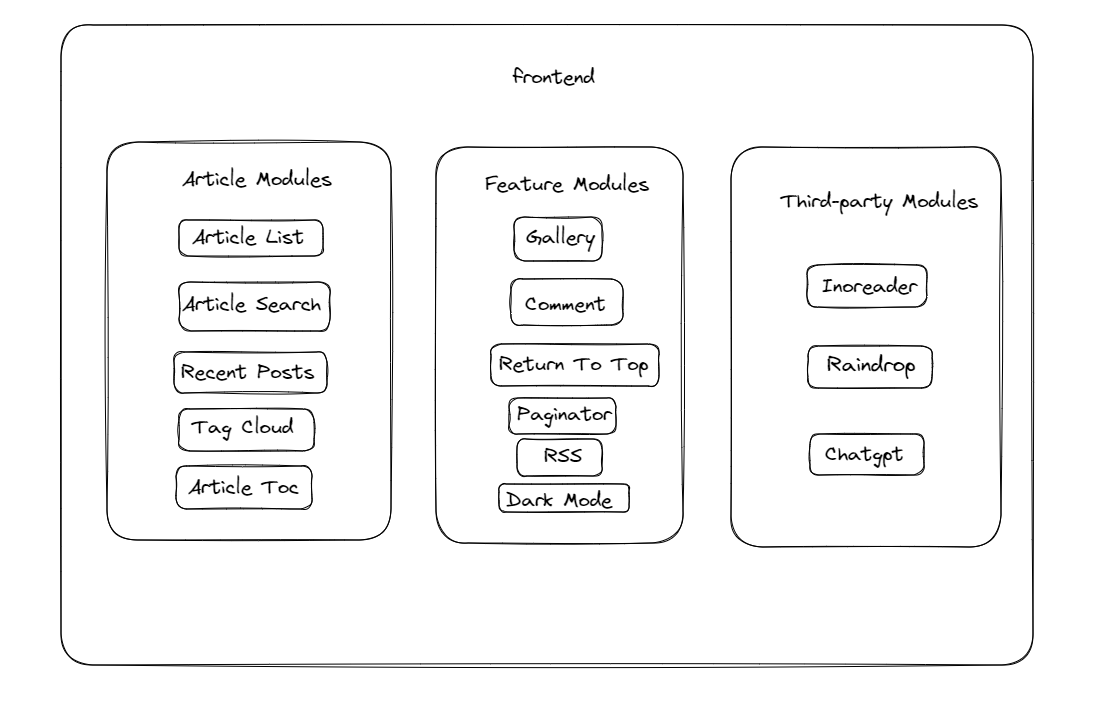


图3-2系统前端架构设计

### 系统后端架构设计

后端选用Supabase作为服务提供商，它是一个开源的Firebase替代品，提供实时数据库、认证和APIs。Supabase支持多种数据库和存储服务，简化了后端开发流程，使开发者能够专注于业务逻辑。

Supabase使用PostgreSQL作为其主要数据库，PostgreSQL是一种功能强大的开源对象关系数据库系统。数据存储中包含一张comment表格和一个comment\_emails视图[15]。

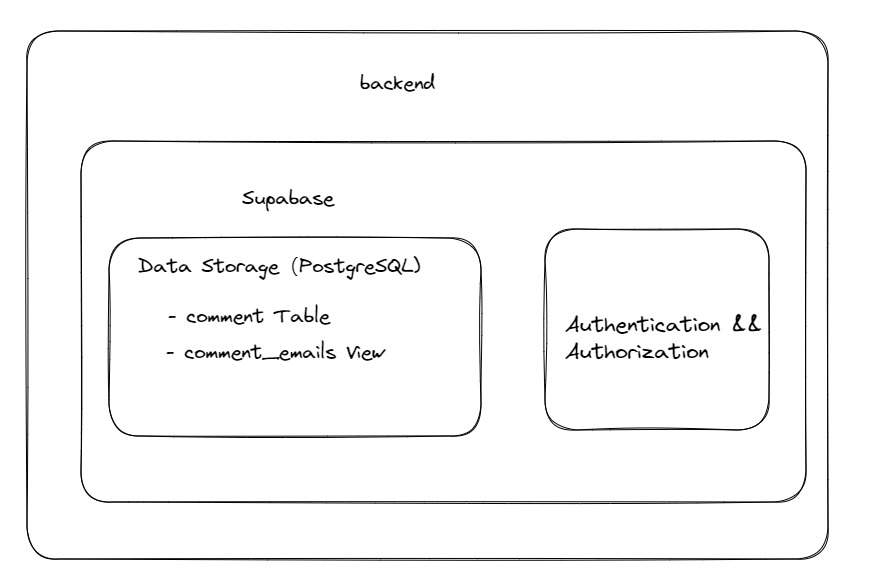


图3-3系统后端架构设计

通过以上架构设计，项目确保了高性能、安全性和可扩展性，为实现项目目标提供了坚实的技术基础。

## 业务模型设计

本项目的业务模式设计主要分为前台和后台两部分，各自包含了业务流程设计和功能模块设计。

### 前台业务流程设计

* 用户访问首页，浏览文章列表
* 用户使用搜索功能，根据关键词查找文章
* 用户查看近期文章、Inoreader广播、Raindrop书签分享及标签云
* 用户点击标签，筛选具有该标签的文章
* 用户查看文章目录（如有）
* 用户在文章页面查看ChatGPT生成的摘要
* 用户在评论区进行评论和回复操作
* 用户访问画廊，查看图片资源
* 用户使用RSS订阅文章
* 用户点击返回顶部按钮，快速回到页面顶部

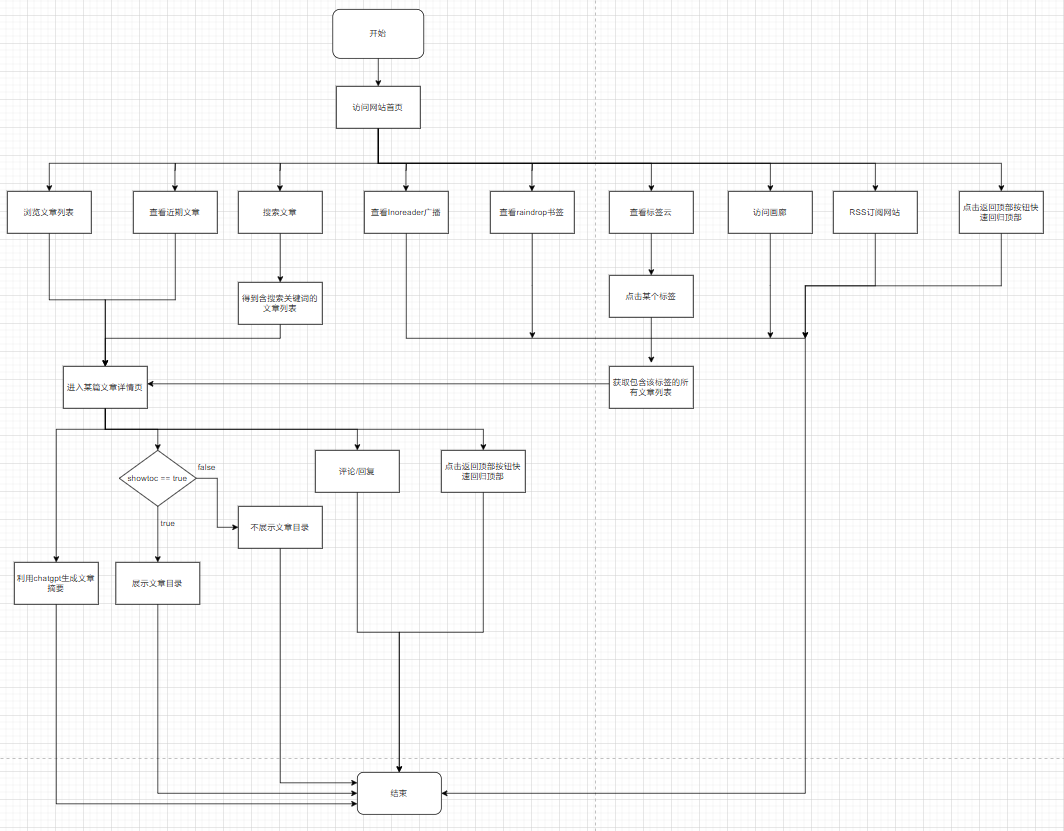


图3-4前台业务流程图

### 前台功能模块设计

* 文章列表模块
* 文章搜索模块
* 近期文章模块
* Inoreader广播模块
* Raindrop书签分享模块
* 标签云模块
* 返回顶部模块
* 文章目录模块
* ChatGPT总结文章摘要模块
* 文章列表分页器模块
* 评论模块
* 画廊模块
* RSS模块

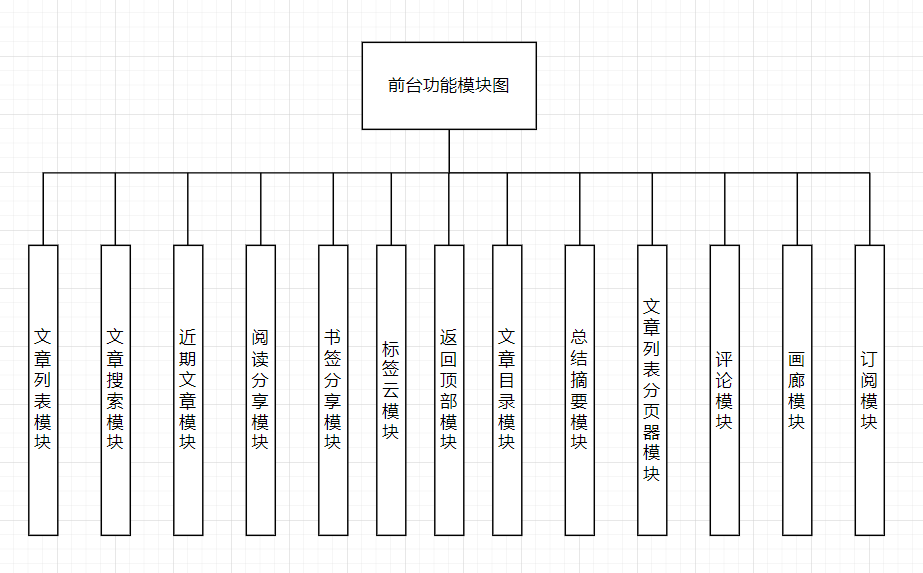


图3-5 前台功能模块图

### 后台业务流程设计

* 管理员在posts文件夹下新建md文件，编写文章
* 管理员为md文件添加frontmatter（标题、标签、时间、摘要、短链接等）
* 管理员将新文章推送到GitHub，触发Vercel自动部署

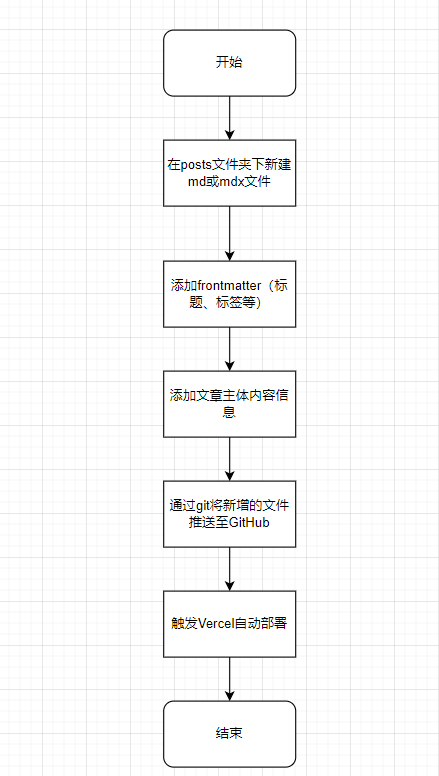


图3-6后台业务流程图

### 后台功能模块设计

* 文章创建模块
* frontmatter管理模块
* GitHub集成模块
* Vercel部署模块
* 评论管理模块（Supabase）

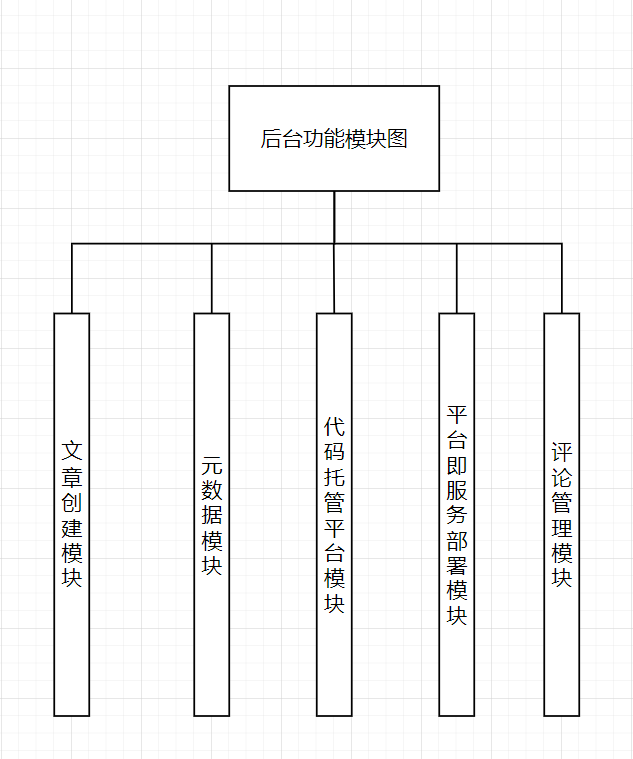


图3-7后台功能模块

通过以上业务模型设计，构建了一个功能丰富、易于使用的个人博客系统，满足用户在浏览、搜索、查看摘要、评论、订阅等方面的需求，同时展示了作者的近期文章、Inoreader广播和Raindrop书签分享。整体架构保证了系统的可拓展性和易维护性，为用户带来良好的阅读体验。

## 数据库设计

### 数据库概念模型设计

采用E-R图来表示实体与实体之间的关系。个人博客涉及的实体有作者、访客、文章、评论。其中实体之间的关系如下：

* 作者和文章：一个作者可以写多篇文章，而一篇文章只能有一个作者。因此，作者（Author）和文章（Article）之间是一对多（1:N）的关系。
* 文章和评论：一篇文章可以有多个评论，而一个评论只能属于一篇文章。因此，文章（Article）和评论（Comment）之间是一对多（1:N）的关系。
* 访客和评论：一个访客可以发表多个评论，而一个评论只能属于一个访客。因此，访客（Visitor）和评论（Comment）之间是一对多（1:N）的关系。
* 作者和评论：一个作者可以发表多个评论，而一个评论只能属于一个作者。因此，作者（Author）和评论（Comment）之间是一对多（1:N）的关系。
* 评论和父评论（如有）：一个评论可以回复一个父评论，一个父评论只能回复一个评论（子评论）。因此，评论（Comment）和父评论（Comment）之间存在一对一（1:1）的关系。

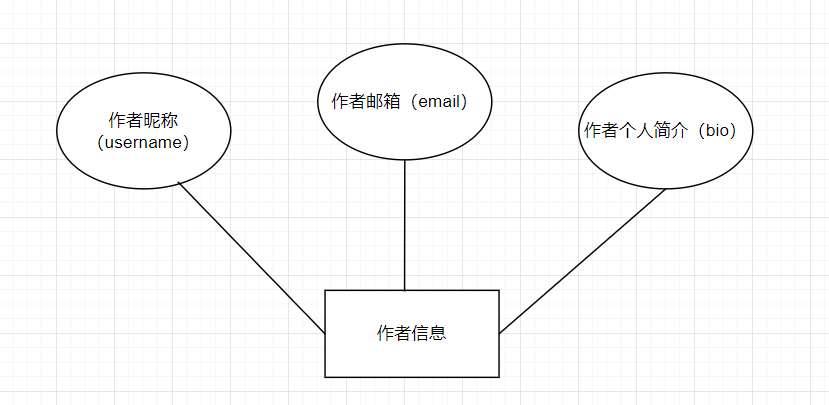


图3-8 作者信息实体图

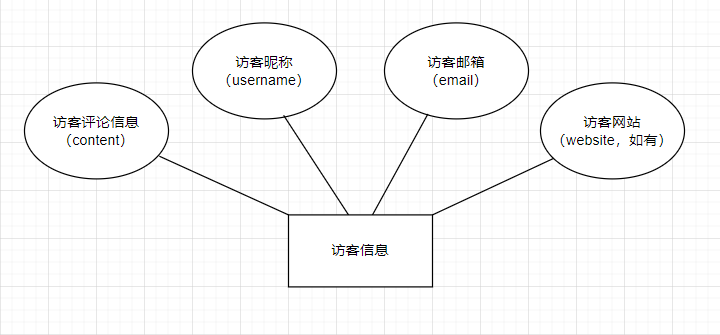


图3-9 访客信息实体图

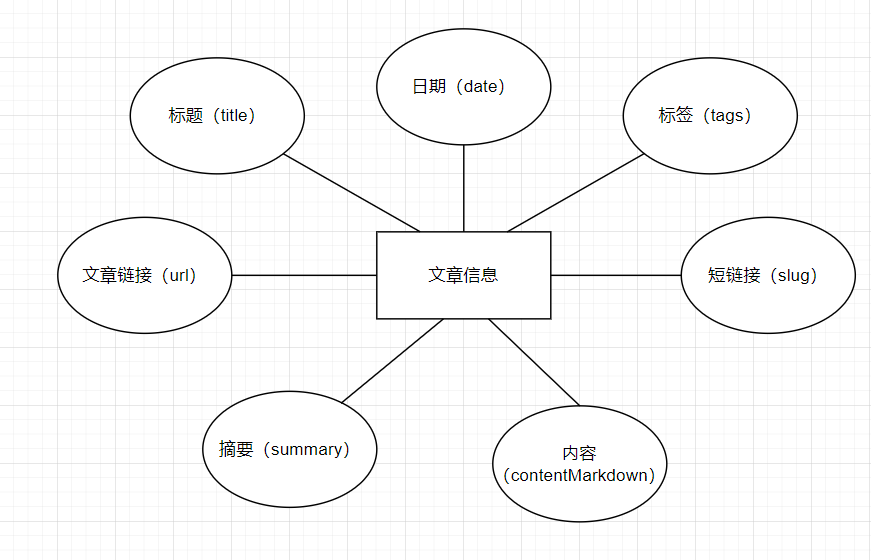


图3-10 文章信息实体图

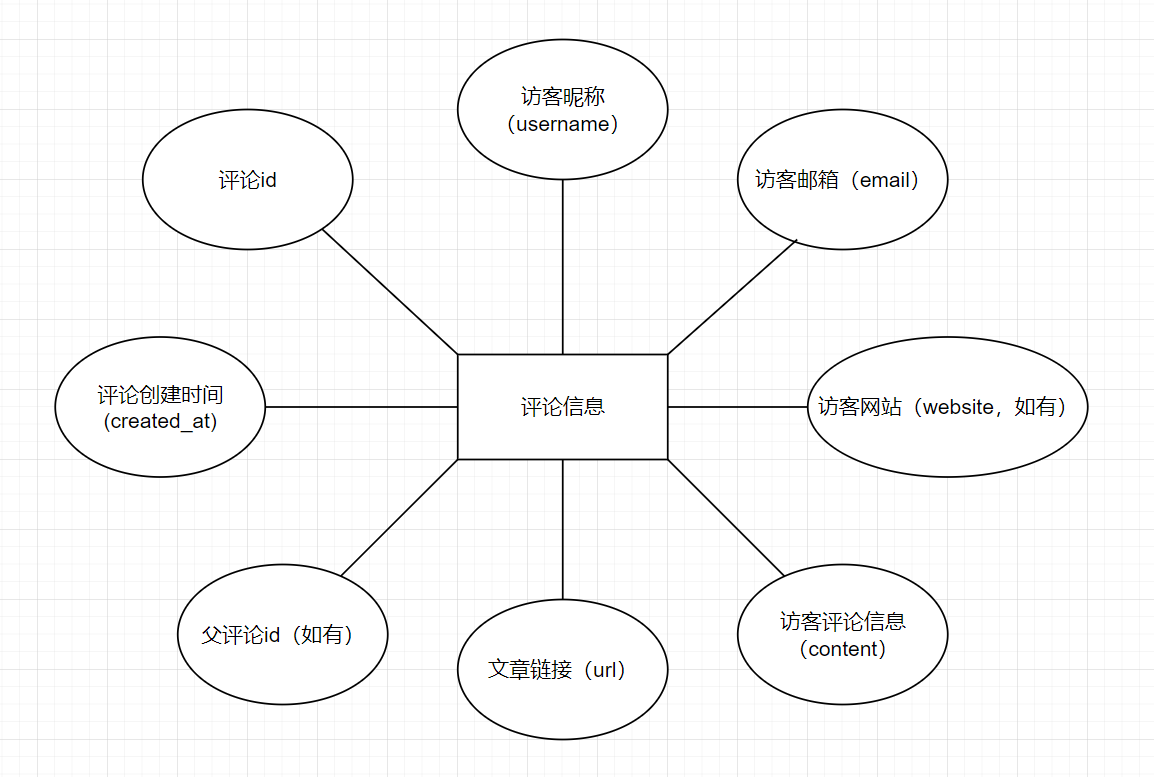


图3-11 评论信息实体图

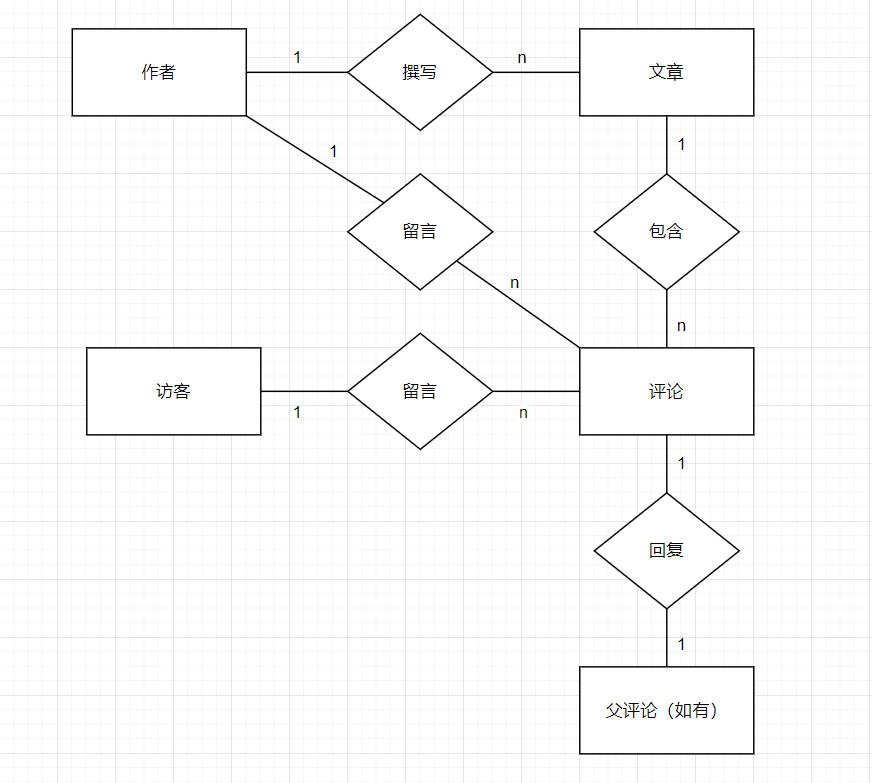


图3-12 总E-R图

### 数据库表结构设计

本项目采用Supabase作为数据库系统，主要用于存储和管理评论信息。在Supabase中创建了一个名为comments的数据表和一个名为comment\_emails的视图。comments表用于存放用户提交的评论数据，而comment\_emails视图则用于查询每个评论的父评论的邮箱地址。

表3-13 comments表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | 字段类型 | 字段描述 | 长度 | 是否允许为空 | 默认值 |
| id | bigint | 唯一标识每条评论，主键 | - | 否 | 自动生成，identity |
| username | varchar(20) | 评论者的用户名 | 20 | 否 | 无 |
| email | varchar(50) | 评论者的电子邮箱地址，用于发送回复通知 | 50 | 否 | 无 |
| website | varchar(30) | 评论者的个人网址，用于互相访问学习，促进博客文化的传播 | 30 | 是 | 无 |
| content | text | 记录评论内容 | - | 否 | 无 |
| url | varchar(255) | 与评论相关联的文章URL | 255 | 否 | 无 |
| created\_at | timestamp with time zone | 记录评论创建时间 | - | 否 | timezone('utc'::text, now()) |
| parent\_comment\_id | bigint | 记录父评论ID，用于表示评论回复关系（如评论是对另一个评论的回复） | - | 是 | 无 |

表3-14 comment\_emails视图

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名称 | 字段类型 | 字段描述 | 长度 | 是否允许为空 | 默认值 |
| id | bigint | 与comments表中的id关联，表示每条评论的唯一标识符 | - | 否 | 无 |
| emails | text[] | 记录每条评论的所有父评论的电子邮箱地址 | - | 否 | 无 |

在项目中，comments表用于存储和管理用户提交的评论信息，而comment\_emails视图则用于查询评论的父评论邮箱地址，以便在有新回复时发送通知。数据库设计部分旨在满足项目的评论功能需求。

## 本章小结

在本章中，我们详细讨论了基于Next.js、Tailwind CSS、Supabase和Vercel的个人博客系统的设计方案。首先，我们阐述了系统的总体架构设计，涉及前端架构、后端架构以及技术栈选型。接着，我们分析了业务模型设计，包括前台业务流程、前台功能模块、后台业务流程和后台功能模块。最后，我们对数据库进行了概念设计和表结构设计，介绍了comments表和comment\_emails视图的字段及其作用。本章通过全面深入地分析，为构建一个高效、易用的个人博客系统奠定了基础。

# 系统实现

## ChatGPT自动总结文章摘要模块

在个人博客系统中，我们实现了一个ChatGPT自动总结文章摘要的功能模块。该模块的核心功能是利用OpenAI的gpt-3.5-turbo模型，通过与API交互，自动生成文章摘要。这一功能不仅提高了用户阅读效率，还为博客增添了独特的亮点。以下是该模块的具体实现过程。

const fetchSummary = () => {

        setIsFetching(true)

        const truncatedContentMarkdown = *contentMarkdown*.slice(0, 1000)

        fetch('/api/chatgpt', {

            method: 'POST',

            headers: {

                'Content-Type': 'application/json',

            },

            body: JSON.stringify({

                message: `using Chinese to summary this article. The article content is: ${truncatedContentMarkdown}.

                Please summary this article within 100 chinese words.`,

            }),

        })

            .then((*response*) => *response*.json())

            .then(*data* => {

                setSummary(`${*data*.gptResponse}`)

                setIsFetching(false)

                setShowCopyButton(true)

            })

            .catch(*error* => {

                console.error('Error fetching summary:', *error*)

                setIsFetching(false)

            })

    }

图4-1 ChatGPT Summary模块部分代码

首先，在前端我们创建了一个名为ChatGPTSummary的React组件。该组件负责显示文章摘要的生成结果，并提供一个按钮供用户触发生成摘要的操作。当用户点击按钮后，组件会向api路由发送请求，api路由再向API地址发送请求，接着再依次响应数据，最终将数据渲染到页面上。

import { Configuration, OpenAIApi } from 'openai'

*// Create an instance of the OpenAI API*

const configuration = new Configuration({

    apiKey: process.env.OPENAI\_API\_KEY,

    organization: process.env.OPENAI\_ORG\_ID,

})

const openai = new OpenAIApi(configuration)

*// Create a completion*

export default async function handler(*req*, *res*) {

    if (*req*.method === 'POST') {

        const { message } = *req*.body

        const completion = await openai.createChatCompletion({

            model: 'gpt-3.5-turbo',

            messages: [{

                role: 'user',

                content: message,

            }],

        })

        const gptResponse = completion.data.choices[0].message.content

*res*.status(200).json({ gptResponse })

    }

}

图4-2 chatgpt api路由

在api路由中，我们创建了一个名为api/chatgpt.js的api路由。该路由用于处理前端发送的文章摘要生成请求。当该路由接收到请求后，它会将文章内容和生成摘要的指令一同发送给OpenAI的API。在这个过程中，我们使用了openai库，并通过配置对象Configuration和OpenAIApi实例来实现与OpenAI API的通信。

这种API路由的方式有以下好处：

1. 保证API不被盗用：通过在Next.js项目中创建API路由，可以在后端处理与ChatGPT API的通信，而不需要在前端暴露您的API密钥。这样可以有效防止API密钥被他人盗用。
2. 绕过地区限制：由于Vercel的服务器位于美国，因此它可以成功访问ChatGPT API。通过这种方式，可以绕过OpenAI对于中国地区的限制，并顺利使用ChatGPT API。

与API的通信过程中，我们将prompt提示内容设置为*using Chinese to summary this article. The article content is: ${contentMarkdown}. Please summary this article within 100 chinese words.*，以便在限定的字数范围内生成中文摘要。API接收到指令后，将返回生成的摘要结果。

在获取到摘要结果后，后端将结果返回给前端。前端的ChatGPTSummary组件接收到结果后，将摘要展示给用户。用户还可以选择复制该摘要，方便分享至社交网络，与他人分享自己的思考。

值得一提的是，由于我演示站点的域名托管在Cloudflare，并开启了Cloudflare的小云朵。因此访问流程如下图所示：

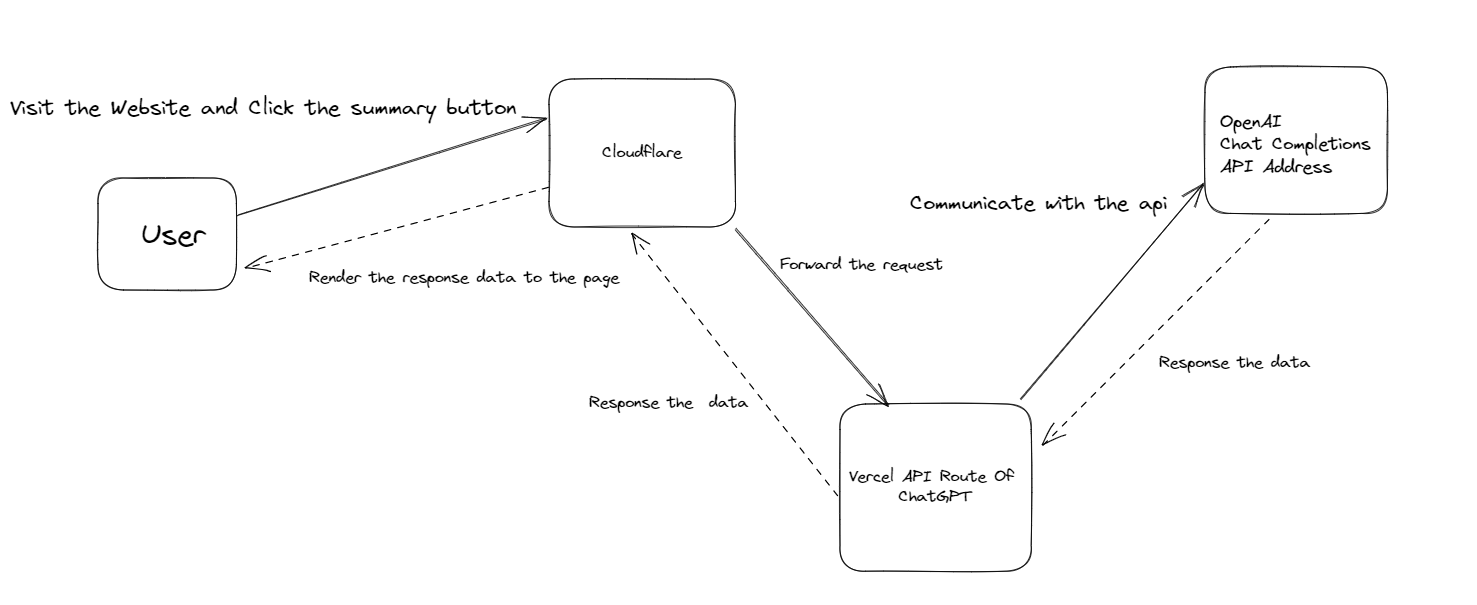


图4-3 ChatGPT自动总结文章摘要模块访问流程

通过这个模块的实现，我们提供了一个便捷的方式，让用户快速了解文章的主要内容。同时，该模块的实现过程也展示了如何利用现代自然语言处理技术（如OpenAI的gpt-3.5-turbo模型）为博客系统增添有趣的功能。

## 评论模块实现

在本项目中，评论模块的实现涉及到了三个主要的组件：Comment, CommentForm, 和 CommentList。在这一节中，我们将重点讨论这些组件中代表项目工作量和难度、具有一定创新性或研究性的功能。

### Comment组件

import { useState } from 'react'

import CommentForm from 'components/CommentForm'

import CommentList from 'components/CommentList'

*// 评论组件*

export default function Comment() {

const [quote, setQuote] = useState('')

const [updateList, setUpdateList] = useState(false)

const [parentCommentId, setParentCommentId] = useState(null)

const quoteComment = (*comment*,*commentId*) => {

setQuote(`<blockquote>${*comment*.username}的发言:<br/>${*comment*.content}</blockquote><br/>`)

    setParentCommentId(*commentId*)

}

return (

<>

    <h2 className="text-3xl font-bold mt-8 mb-4">留言</h2>

<CommentList quoteComment={quoteComment} updateList={updateList} />

        <h2 className="text-3xl font-bold mt-8 mb-4">我要发表看法</h2>

            <CommentForm quote={quote} setQuote={setQuote} setUpdateList={setUpdateList} parentCommentId={parentCommentId} setParentCommentId={setParentCommentId} />

   </>

)

}

图4-4 Comment组件代码

Comment 组件是评论模块的主要入口点，负责管理和组织 CommentForm 和 CommentList 两个子组件。该组件主要利用 useState[16] React Hooks 以及传递给子组件的回调函数来实现与其他组件的交互。

一个具有创新性的功能是实现引用评论功能。用户可以通过点击 CommentList 中的 "Quote" 按钮，将所选评论的内容插入到 CommentForm 中。为实现这一功能，我们在 Comment 组件中定义了一个名为 quoteComment 的回调函数，负责将被引用评论的信息传递给 CommentForm 组件。

### CommentForm组件

const handleSubmit = async (*e*) => {

*e*.preventDefault()

setButtonText('Submitting...')

const formData = new FormData(formRef.current)

const username = formData.get('username')

const email = formData.get('email')

const website = formData.get('website')

const content = formData.get('content')

const token = formData.get('cf-turnstile-response')

if (!validator.validate(email)) {

setEmailError('邮箱地址无效，请填写正确的邮箱地址。')

setButtonText('Submit')

return

} else {

setEmailError('')

}

const res = await fetch('/api/comInsert', {

  method: 'POST',

        body: JSON.stringify({ username, email, website, content, token,parent\_comment\_id: *parentCommentId* }),

        headers: {

        'Content-Type': 'application/json'

        }

  })

if (res.status === 200) {

alert('你的留言已被接受')

        formRef.current.reset()

turnstileRef.current.reset()

      setQuote('')

       setUpdateList((*prev*) => !*prev*)

        setParentCommentId(null)

       setButtonText('Submit')

        } else if (res.status === 403) {

alert('请完成人机验证')

           turnstileRef.current.reset()

          setButtonText('Submit')

        } else {

alert('留言出现了一些错误，请稍后再试')

          turnstileRef.current.reset()

          setButtonText('Submit')

}

}

图4-5 CommentForm组件部分代码

CommentForm 组件负责处理用户提交的评论表单，并向后端发送请求以将评论数据存储到数据库中。其中一个具有研究性的功能是使用 Cloudflare Turnstile 作为人机验证系统，以提高评论模块的安全性。通过在组件中引入 Turnstile 组件并在表单提交时验证 token，我们可以确保评论提交者是真实用户而非自动化脚本。

验证成功后，服务器会处理评论数据，并将其插入到评论数据库中。在插入过程中，api/comInsert.js会调用getEmailsFromParentComments函数获取所有父评论的电子邮件地址。然后，服务器会向这些电子邮件发送通知，告知他们收到了新的回复。

const data = await result.json()

    if (!data.success) {

        return *res*.status(403).json({ error: 'Invalid token' })

      } else {

        const { data: insertedData, error } = await supabase

        .from('comments')

        .insert([

        { username: username, email: email, website: website, content: content, url: cleanUrl, parent\_comment\_id: parent\_comment\_id },])

        .select()

        if (error) {

          console.error('Error inserting comment:', error)

          return *res*.status(500).json({ error: error.message })

        }

*// 插入成功后，获取相关父评论的email*

        const parentCommentEmails = await getEmailsFromParentComments(insertedData[0].id)

*// 发送邮件通知给所有相关父评论的作者*

        for (const parentCommentEmail of parentCommentEmails) {

          try {

                await transporter.sendMail({

              from: process.env.EMAIL\_USERNAME, *// 发件人地址*

              to: parentCommentEmail, *// 收件人地址*

              subject: `New reply to your comment in ${process.env.NEXT\_PUBLIC\_SITE\_TITLE}`, *// 主题*

              text: `${username} replied to your comment: ${processedContent}. Please visit ${cleanUrl} to view it.`, *// 纯文本内容*

              html: `<p>${username} replied to your comment: ${processedContent}. <br/> Please visit <a href="${cleanUrl}">${cleanUrl}</a> to view it.</p>`, *// HTML内容*

                })

          } catch (err) {

                console.error('Error sending email:', err)

          }

        }

*res*.status(200).json(insertedData)

      }

图4-6 comInsert.js部分代码

此外，我们在 CommentForm 中实现了邮箱验证功能，通过 validator 库检查用户输入的电子邮件地址是否有效。这有助于确保用户提交的评论信息更加完整和准确。

### CommentList组件

CommentList 组件负责从后端获取评论数据并呈现在页面上。我们使用了api/comSelect.js接口，该接口从评论数据库中提取与当前页面相关的评论，并按时间顺序排列。通过这种方式，用户可以看到所有其他用户在当前页面上发表的评论。

import supabase from 'utils/supabase'

const fetchCommentsFromDB = async (*url*) => {

    const { data, error } = await supabase

        .from('comments')

        .select('\*')

        .eq('url', *url*)

        .order('created\_at', { ascending: true })

    if (error) {

        throw new Error(error.message)

    }

    return data

}

export default async function handler(*req*, *res*) {

    try {

        const comments = await fetchCommentsFromDB(*req*.headers.referer)

*res*.status(200).json(comments)

    } catch (error) {

*res*.status(500).json({ error: error.message })

    }

}

图4-7 comSelect.js代码

const generateIdenticon = (*username*) => {

    const hash = CryptoJS.MD5(*username*).toString()

    const data = new Identicon(hash, { size: 64, format: 'svg' }).toString()

    return `data:image/svg+xml;base64,${data}`

}

图4-8 CommentList组件部分代码

为提高用户体验，我们实现了一个独特的功能，即为每个用户生成一个基于用户名的不重复的头像。通过使用 Identicon.js 库和 CryptoJS 库，我们可以根据用户名生成一个唯一的 MD5 哈希值，并将其转换为一个 SVG 格式的头像。这样，可以为用户呈现一个具有个性化视觉效果的头像。

综上所述，评论模块实现了一些具有创新性和研究性的功能，包括引用评论、人机验证、邮箱验证和基于用户名的头像生成。这些功能共同提高了评论模块的用户体验和安全性。

## 文章目录模块实现

文章目录模块的主要目的是为用户提供一个便捷的导航工具，使用户能够快速定位到文章中的特定部分。本模块的实现过程包括以下三个关键功能：

import { remark } from 'remark'

import { visit } from 'unist-util-visit'

import slug from 'remark-slug'

export default function parseHeading(*contentMarkdown*) {

    const headings = []

    remark()

        .use(slug)

        .use(() => (*tree*) => {

            visit(tree, 'heading', (*node*) => {

*// 只解析 h2 和 h3*

                if (node.depth === 2 || node.depth === 3) {

                    const id = node.data.id

                    const heading = {

                        depth: node.depth,

                        value: node.children[0].value,

                        id,

                    }

                    headings.push(heading)

                }

            })

        })

        .processSync(contentMarkdown)

    return headings

}

图4-9 lib/parseHeading.js代码

1）解析 Markdown 文档的标题层次结构 为了实现这一功能，我们引入了 remark 库以及 unist-util-visit 库。首先，我们使用 remark 对输入的 Markdown 文档进行解析，生成一个抽象语法树 (AST)。接着，使用 unist-util-visit 遍历该 AST，提取出文档中的所有标题节点（包括 h2 和 h3 级别的标题）。在解析过程中，还会利用 remark-slug 插件为每个标题生成一个唯一的 ID，以便后续实现锚点跳转。

const [expanded, setExpanded] = useState(true)

const headings = parseHeading(*contentMarkdown*)

const toggleSidebar = () => {

    setExpanded(!expanded)

}

const bulletStyle = (*depth*) => {

    if (*depth* === 2) {

        return '●'

    }

    if (*depth* === 3) {

        return '○'

    }

    return ''

}

图4-10 ArticleToc组件部分代码

2）构建可折叠的文章目录 在解析 Markdown 文档并提取出所有标题节点之后，我们将构建一个可折叠的文章目录组件。这个组件的状态由 expanded 变量控制，初始状态为展开。我们使用 useState Hook 管理 expanded 的状态。当用户点击目录标题时，通过 toggleSidebar 函数实现目录的展开与折叠。此外，还可以根据标题的层级显示不同的列表样式，以便用户更好地区分标题之间的层次关系。

3）实现标题锚点跳转 为了实现标题锚点跳转，我们需要为每个标题节点生成一个具有唯一 ID 的锚点链接。在解析标题节点时，我们已经使用 remark-slug 插件为每个标题生成了一个唯一的 ID。接下来，我们只需在生成的目录列表中为每个标题创建一个锚点链接，链接的 href 属性值为对应标题的 ID。当用户点击目录中的标题时，页面将自动滚动到对应的标题位置。

通过以上三个关键功能的实现，我们成功地为用户提供了一个便捷、易于使用的文章目录模块。该模块能够帮助用户快速定位到文章的特定部分，提高阅读效率。同时，本模块的实现过程中也充分展示了项目的工作量、难度以及一定程度的创新性和研究性。

## 本章小结

本章节介绍了本个人博客系统的3个特色功能的代码实现。

# 系统测试与部署

## 系统测试

### 功能模块测试

表5-1 系统功能测试表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试编号 | 功能模块 | 测试目的 | 测试步骤 | 预期结果 | 测试结果 |
| T01 | 查看文章列表 | 验证用户能正确查看文章列表 | 1. 打开博客主页  2. 浏览文章列表 | 应显示所有文章的标题、摘要、发布日期信息 | 测试通过 |
| T02 | 搜索文章 | 验证用户能通过关键词搜索文章 | 1. 在搜索框输入关键词  2. 按下回车键 | 应显示与搜索关键词相关的文章列表 | 测试通过 |
| T03 | 浏览近期文章 | 验证用户能查看最近发布的文章 | 1. 打开博客主页  2. 浏览近期文章栏目 | 应显示最近发布的5篇文章 | 测试通过 |
| T04 | 浏览阅读分享 | 验证用户能查看阅读分享栏目 | 1. 打开博客主页  2. 浏览阅读分享栏目 | 应显示最近推送的5条阅读分享条目 | 测试通过 |
| T05 | 浏览书签分享 | 验证用户能查看书签分享栏目 | 1. 打开博客主页  2. 浏览书签分享 | 应显示最近收藏的5条书签条目 | 测试通过 |
| T06 | 根据标签筛选文章 | 验证用户能通过标签筛选文章 | 1. 选择一个标签  2. 查看筛选结果 | 应显示与所选标签相关的文章列表 | 测试通过 |
| T07 | 返回页面顶部 | 验证用户能快速返回页面顶部 | 1. 滚动至页面据顶部有一段距离的位置  2. 点击“返回顶部”按钮 | 页面应平滑滚动至顶部 | 测试通过 |
| T08 | 文章分页功能 | 验证文章列表分页功能正常工作 | 1. 访问文章列表页面  2. 点击分页按钮 | 应显示对应页数的文章列表 | 测试通过 |
| T09 | 查看文章目录 | 验证用户能查看含showtoc为true的文章页目录 | 1. 打开一篇包含目录的文章2. 查看文章目录 | 应显示文章的目录结构 | 测试通过 |
| T10 | 使用ChatGPT总结博客文章 | 验证ChatGPT能为文章生成摘要 | 1. 打开一篇文章  2. 点击“总结摘要”按钮 | 应显示文章摘要 | 测试通过 |
| T11 | 评论文章和回复评论 | 验证用户能评论文章和回复评论，并发送邮件提醒 | 1. 打开一篇文章  2. 在评论区输入评论  3. 提交评论  4. 回复已有评论  5. 提交回复 | 评论和回复应显示在文章下方，相关用户收到邮件提醒 | 测试通过 |
| T12 | 画廊展示 | 验证用户能查看画廊 | 1. 点击画廊栏目  2. 查看画廊 | 应显示public/photos下的所有图片 | 测试通过 |
| T13 | 提供RSS订阅 | 验证用户能订阅博客的RSS源 | 1. 找到RSS订阅链接  2. 使用RSS阅读器订阅博客 | 应在RSS阅读器中显示博客文章更新 | 测试通过 |
| T14 | 支持夜间模式 | 验证用户能在博客中切换夜间模式 | 1. 打开博客首页  2. 切换至夜间模式  3. 浏览页面 | 页面应呈现夜间模式的配色和样式 | 测试通过 |
| T15 | 支持多语言 | 验证用户能在博客中切换语言 | 1. 打开博客首页  2. 切换至另一种语言 | 页面栏目标题应呈现所选语言的内容 | 测试通过 |
| T16-1 | 文章撰写 | 验证博主撰写的文章能被正确渲染 | 1. 在posts文件夹下新建一个文件  2. 编写正确的元数据  3. 添加文章内容 | 新创建的md文件成功渲染 | 测试通过 |
| T16-2 | 文章撰写 | 验证博主撰写的文章能被正确渲染 | 1. 在posts文件夹下新建一个文件  2. 元数据编写有误  3. 添加文章内容 | 新创建的md文件无法被成功渲染，页面报错 | 测试不通过 |

在本次测试中，我们对个人博客系统的各个功能模块进行了详细的检验。通过逐一验证这些功能，我们确保了系统的稳定性和可靠性。只要按照规定的格式编写文章的 frontmatter，并确保其内容无误，博客系统将能够完美实现所有预期功能。这也展示了系统在设计和实现过程中的高度灵活性和可扩展性。

### 性能综合测试

这一部分我们利用 Lighthouse[17]、Google PageSpeed Insights、Vercel Analysis对个人博客系统进行测试，以确保网站的加载速度、交互性能等方面达到优秀的水平。

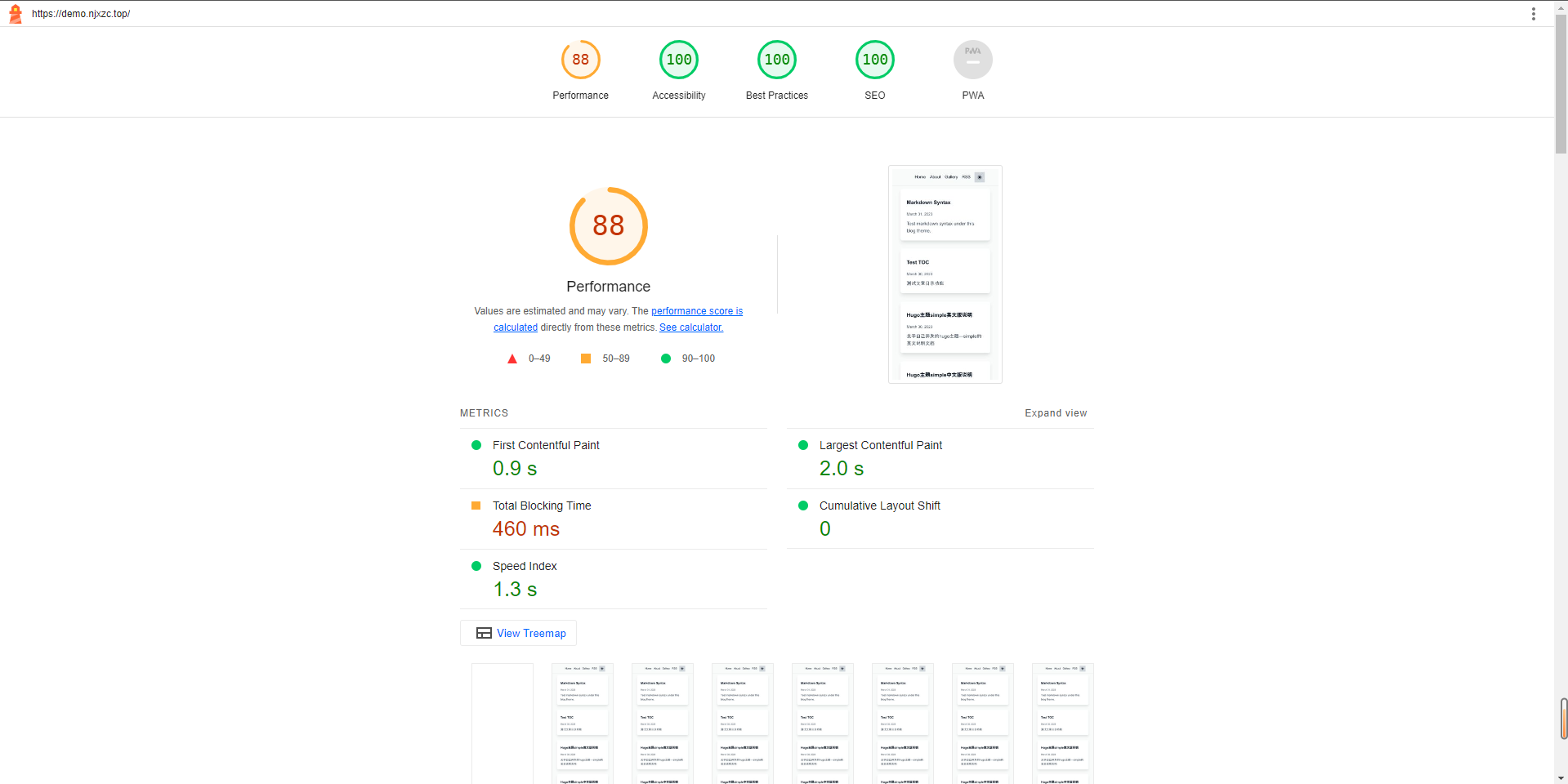


图5-1 Lighthouse评分

经过Lighthouse的测试，我们的个人博客系统在性能方面的得分为88分。虽然这个得分相对较高，但仍有改进的空间。经分析，性能得分主要受到JavaScript加载时间的影响。

在可访问性、最佳实践和搜索引擎优化方面，我们的个人博客系统均获得了满分，这表明系统在这三个方面的表现已经达到了很高的水平。这些高分得益于我们在开发过程中严格遵循相关的最佳实践和指南，确保系统为用户提供了良好的使用体验和无障碍访问。

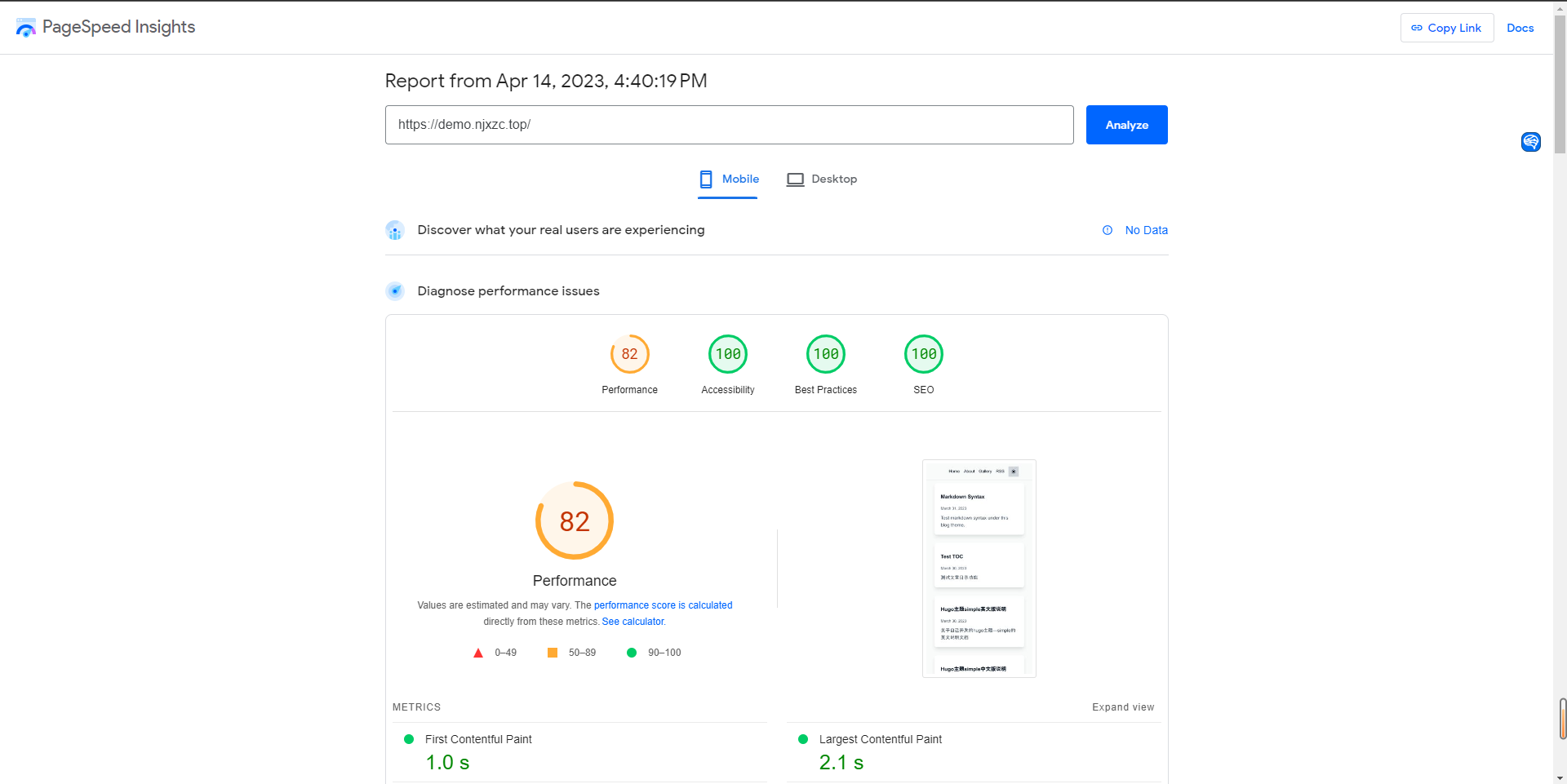


图5-2 Google PageSpeed Insights Mobile评分

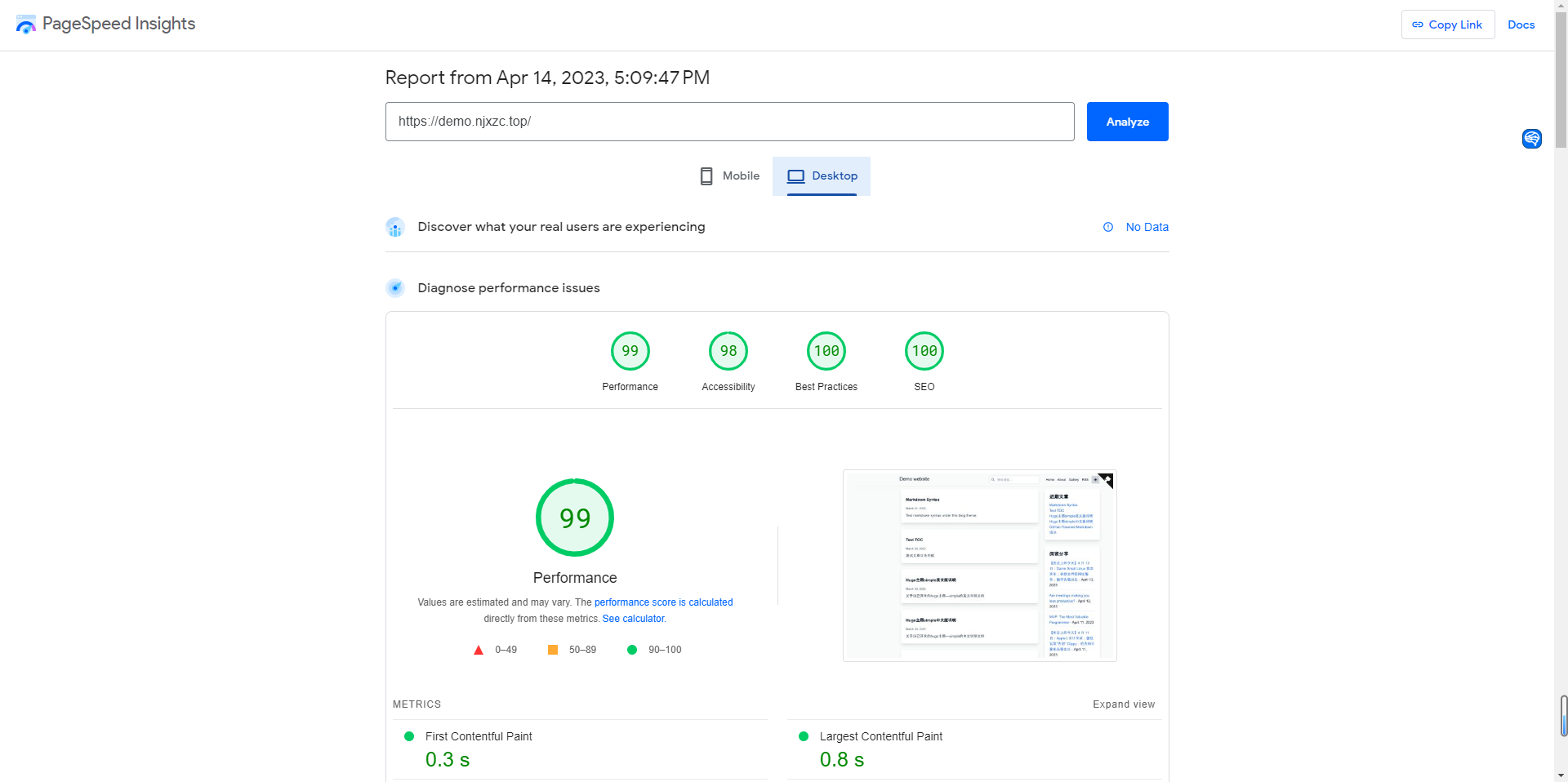


图5-3 Google PageSpeed Insights Desktop评分

经过PageSpeed Insights的评估，我们的个人博客系统在移动端的得分分别为：性能82分，可访问性100分，最佳实践100分和搜索引擎优化100分。其中，性能得分受到了JavaScript加载时间的影响。

在桌面端，我们的个人博客系统获得了如下评分：性能99分，可访问性98分，最佳实践100分和搜索引擎优化100分。性能得分同样受到JavaScript加载时间的影响，可访问性评分稍低的原因是背景和前景颜色之间的对比度不足。

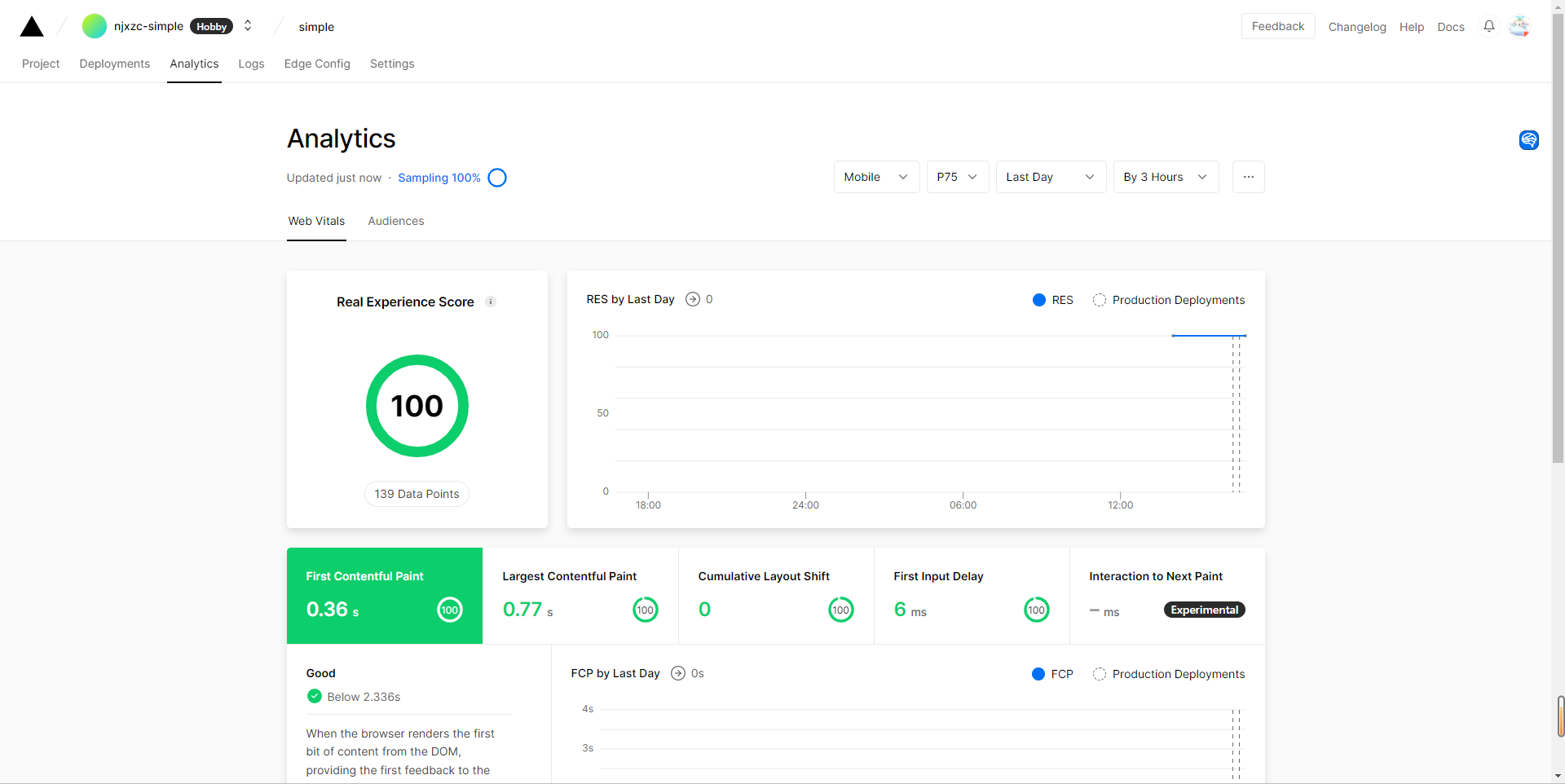


图5-4 Vercel Analytics移动端评分

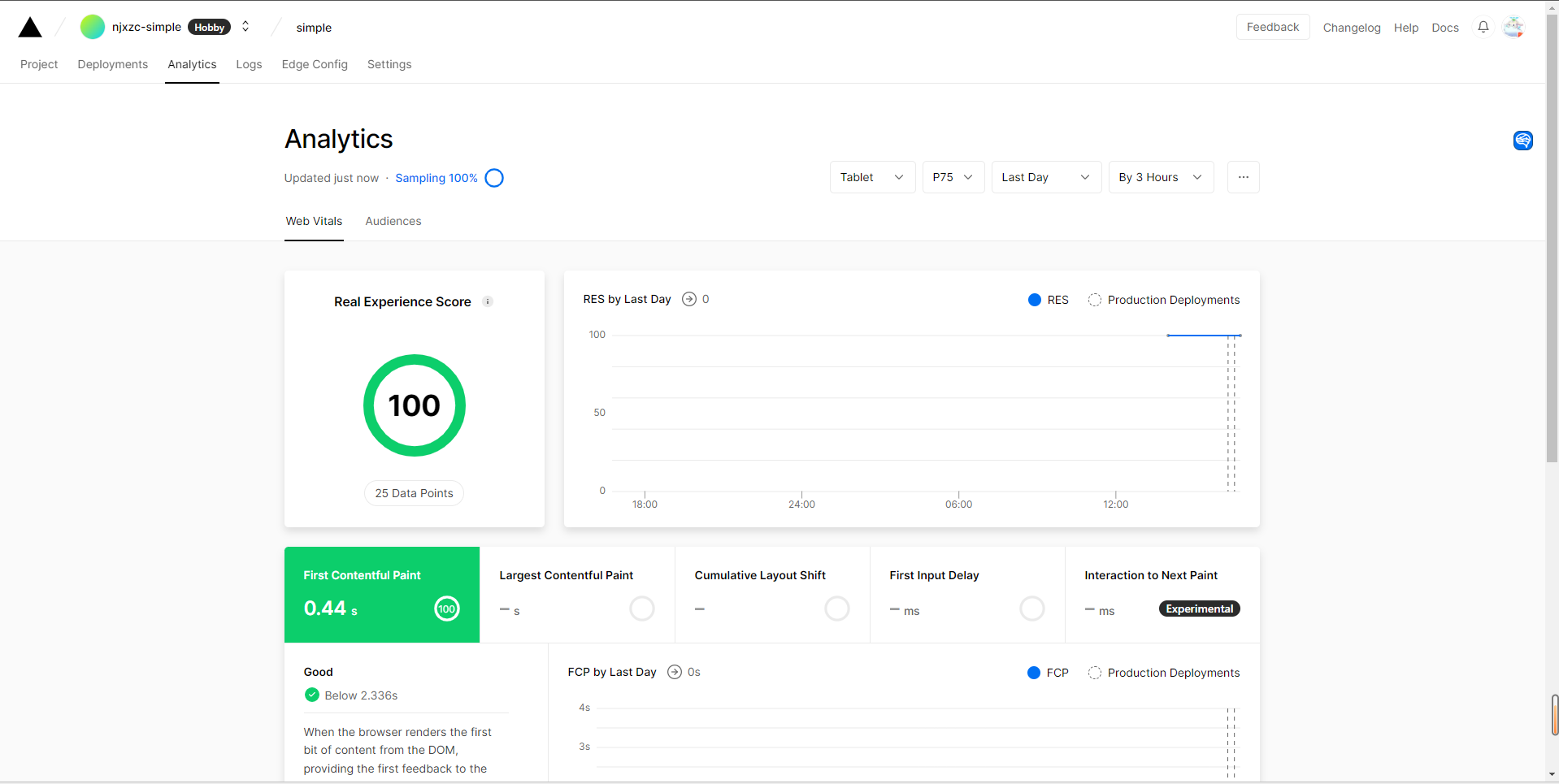


图5-5 Vercel Analytics平板端评分

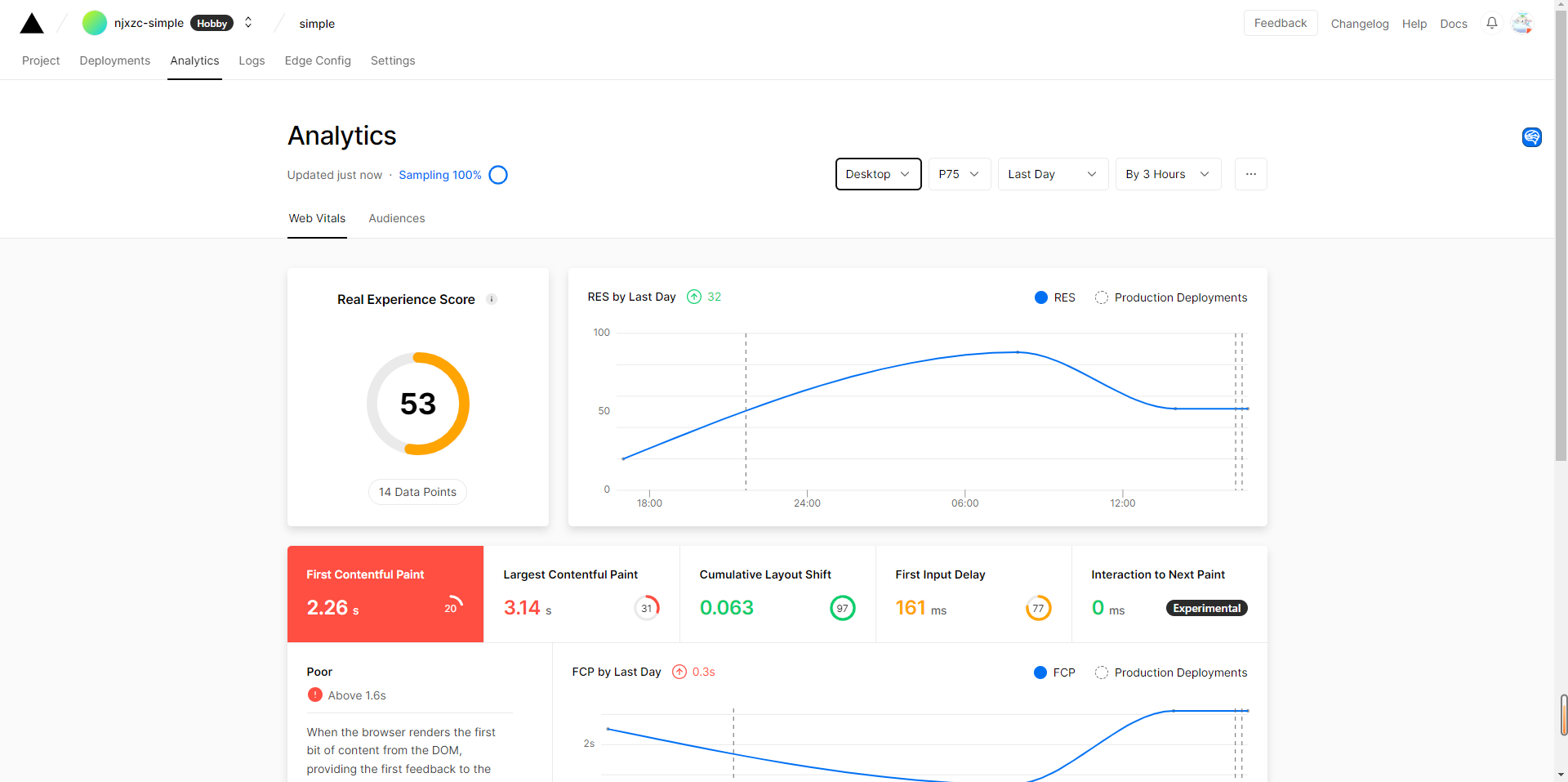


图5-6 Vercel Analytics电脑端评分

根据Vercel Analytics的评估结果，我们的个人博客系统在移动和平板设备上的实际体验得分（Real Experience Score）均达到了满分。然而，在桌面设备上，我们的博客系统仅获得了53分的评价。这主要是由于以下几个方面的性能表现较差：首屏内容绘制（First Contentful Paint，FCP）耗时2.26秒，最大内容绘制（Largest Contentful Paint，LCP）耗时3.14秒，以及首次输入延迟（First Input Delay，FID）耗时161毫秒。

通过对比和分析这些评估工具的结果，我们可以发现博客系统在可访问性、最佳实践和SEO方面表现优秀，但在性能方面仍有待提升。为了提高系统性能，我们可以考虑采取一些优化措施，如优化资源加载策略、减少不必要的渲染、使用代码拆分等技术。在后续开发过程中，我们将继续关注性能指标并采取相应措施，以进一步提升整体用户体验。

### 网络安全测试



图5-6 评论区提交script脚本

尝试在评论区的评论框中提交script脚本。

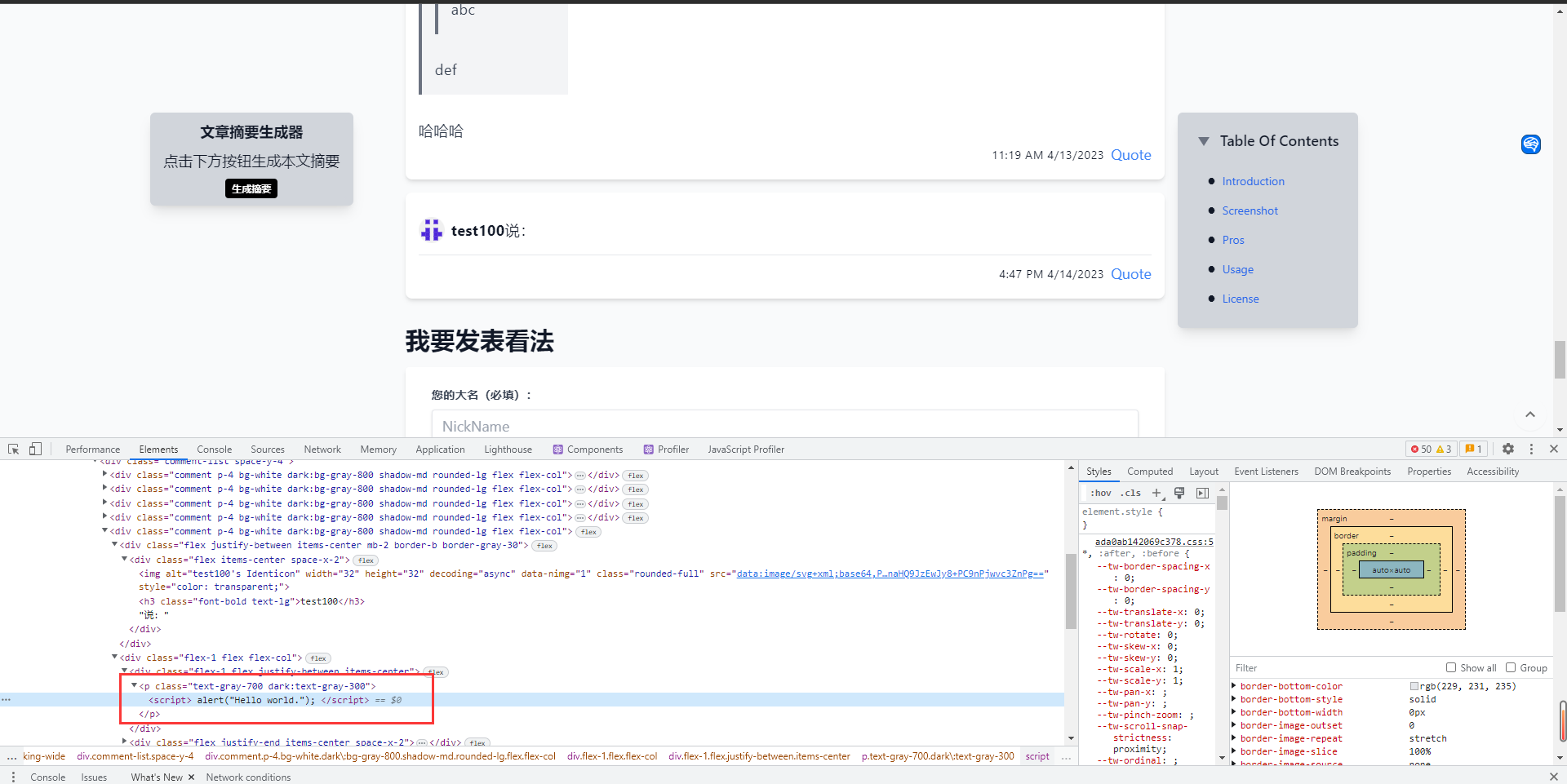


图5-7 script脚本呈现在页面源码上，但未执行

可以发现script标签在网页源码中有呈现，但是页面并没用出现弹窗，究其原因，在于 React 内部使用 innerHTML 将原始 HTML 字符串注入或添加到 DOM 中[18]。当通过 innerHTML[19] 设置时，浏览器不会执行 HTML 字符串中的脚本标签。以防止XSS跨站脚本攻击。

### 兼容性测试

使用 BrowserStack 等浏览器兼容性测试平台对博客系统进行兼容性测试，确保博客在不同浏览器和操作系统上正常运行。这次我采用的工具为LambdaTest，由于我通过了GitHub的学生认证，因此BrowserStack和Lambdatest这两款付费产品我都可以免费使用里面的付费功能一年。

LambdaTest 是一个在线浏览器兼容性测试平台，提供实时测试、自动化测试和可视化测试等功能。它允许你在多种设备、浏览器版本和操作系统上测试你的应用程序，以确保在各种环境下都能提供良好的用户体验。

由于设备的数量太多，因此我们着重挑选一些具有特色的设备，作为典型代表来测试网站的兼容性情况。

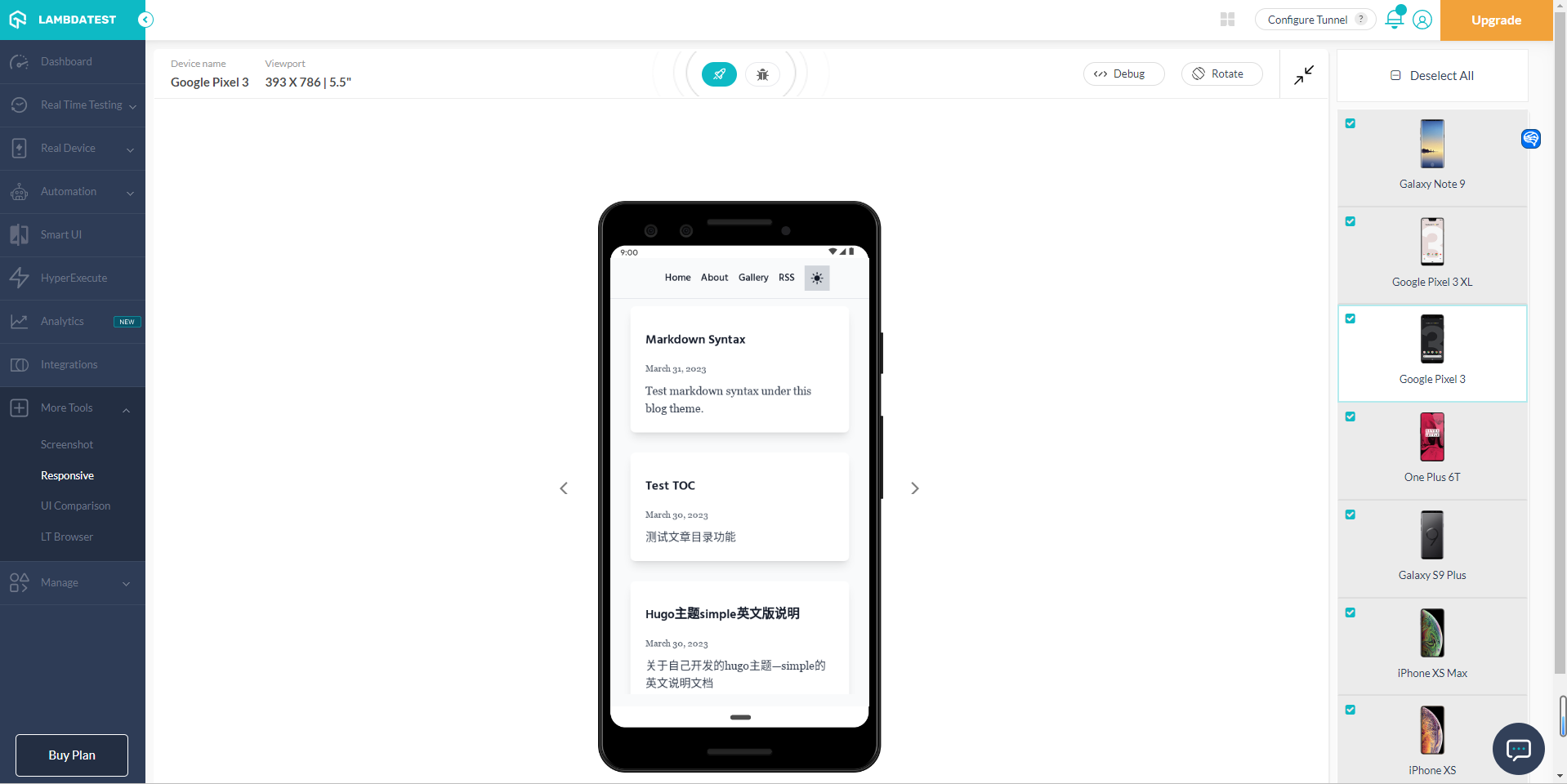


图5-8 Andriod手机测试效果

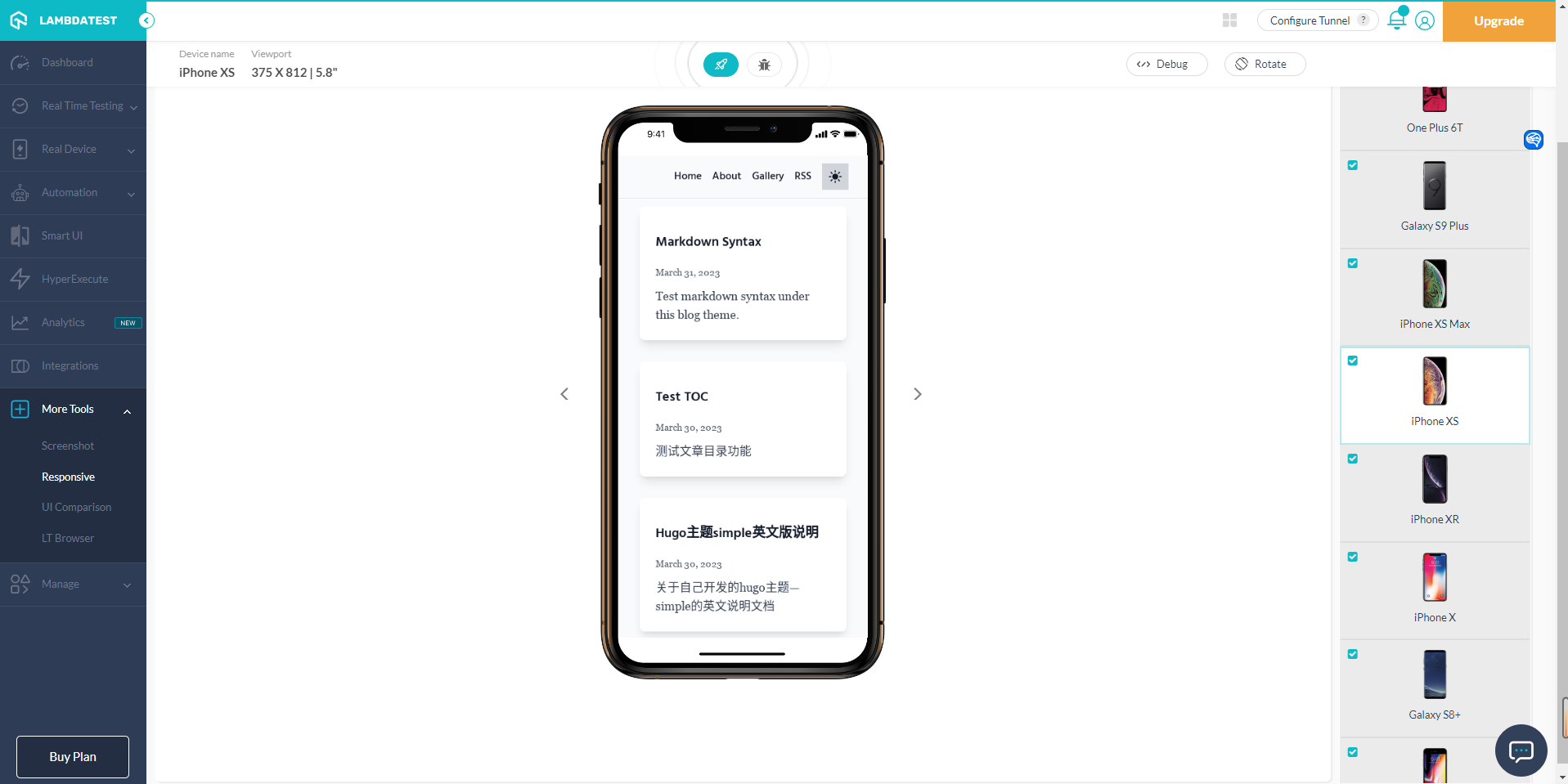


图5-9 iPhone手机测试效果

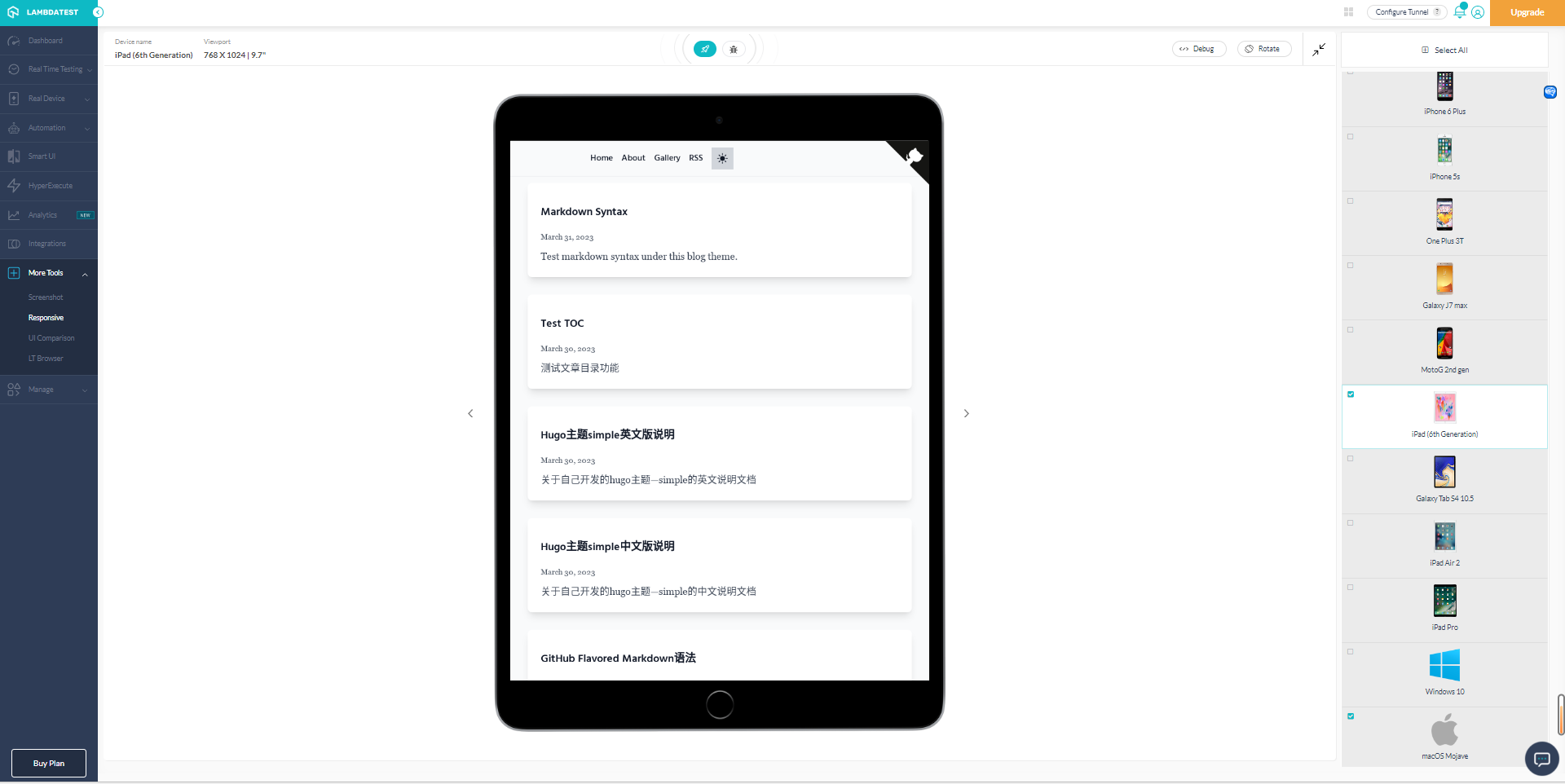


图5-10 iPad测试效果

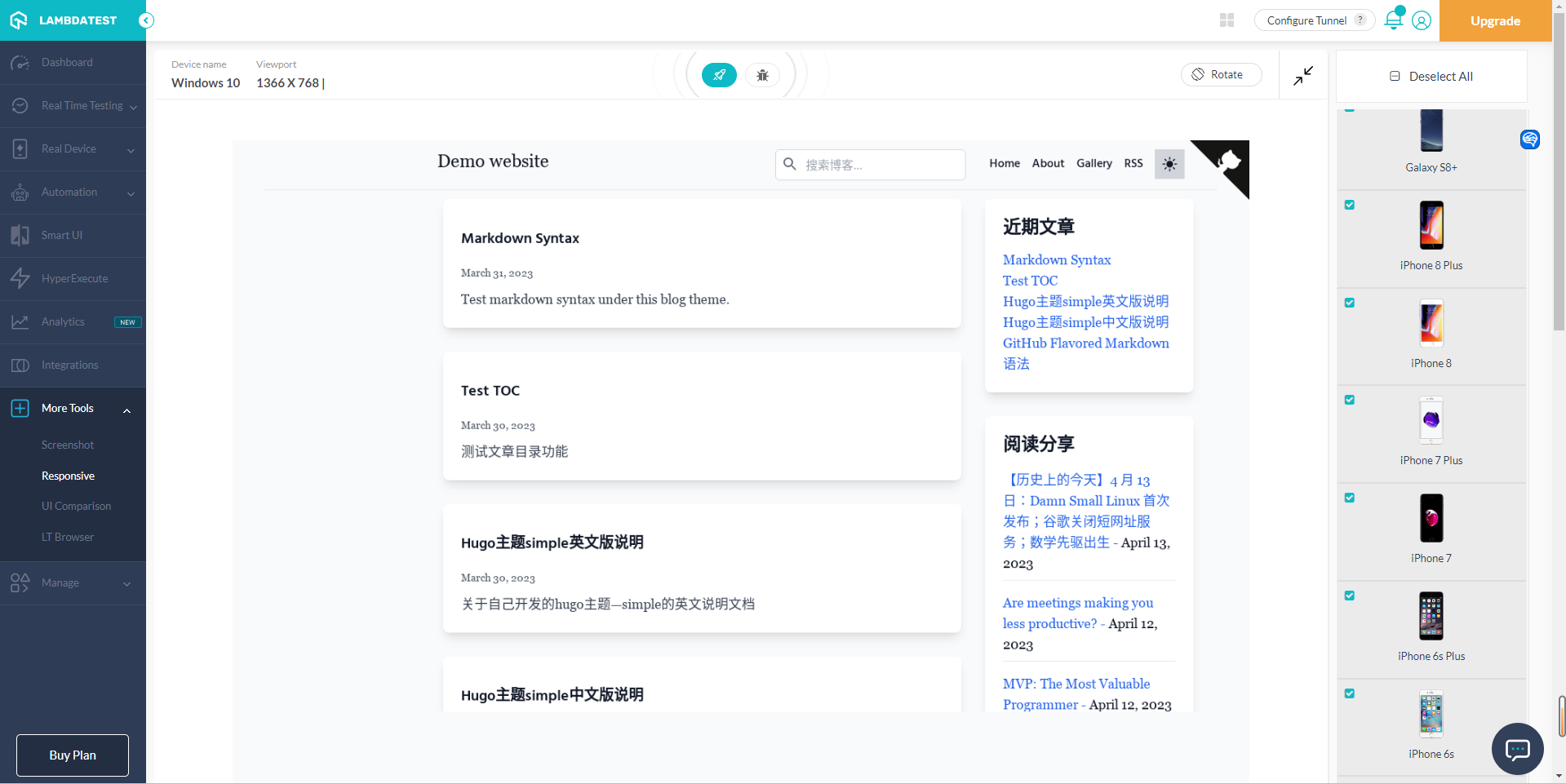


图5-11 Windows电脑测试效果

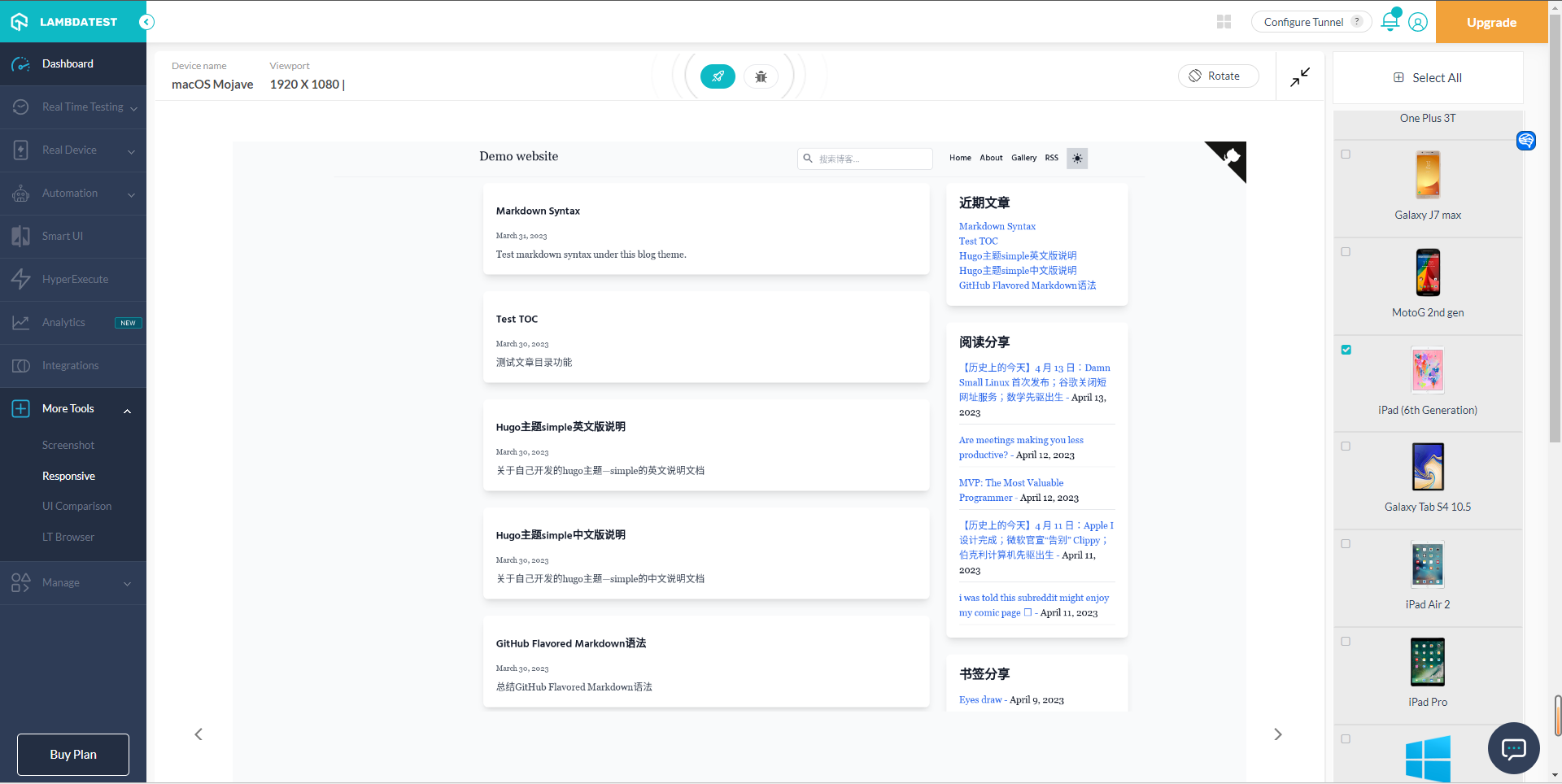


图5-12 Mac OS测试效果

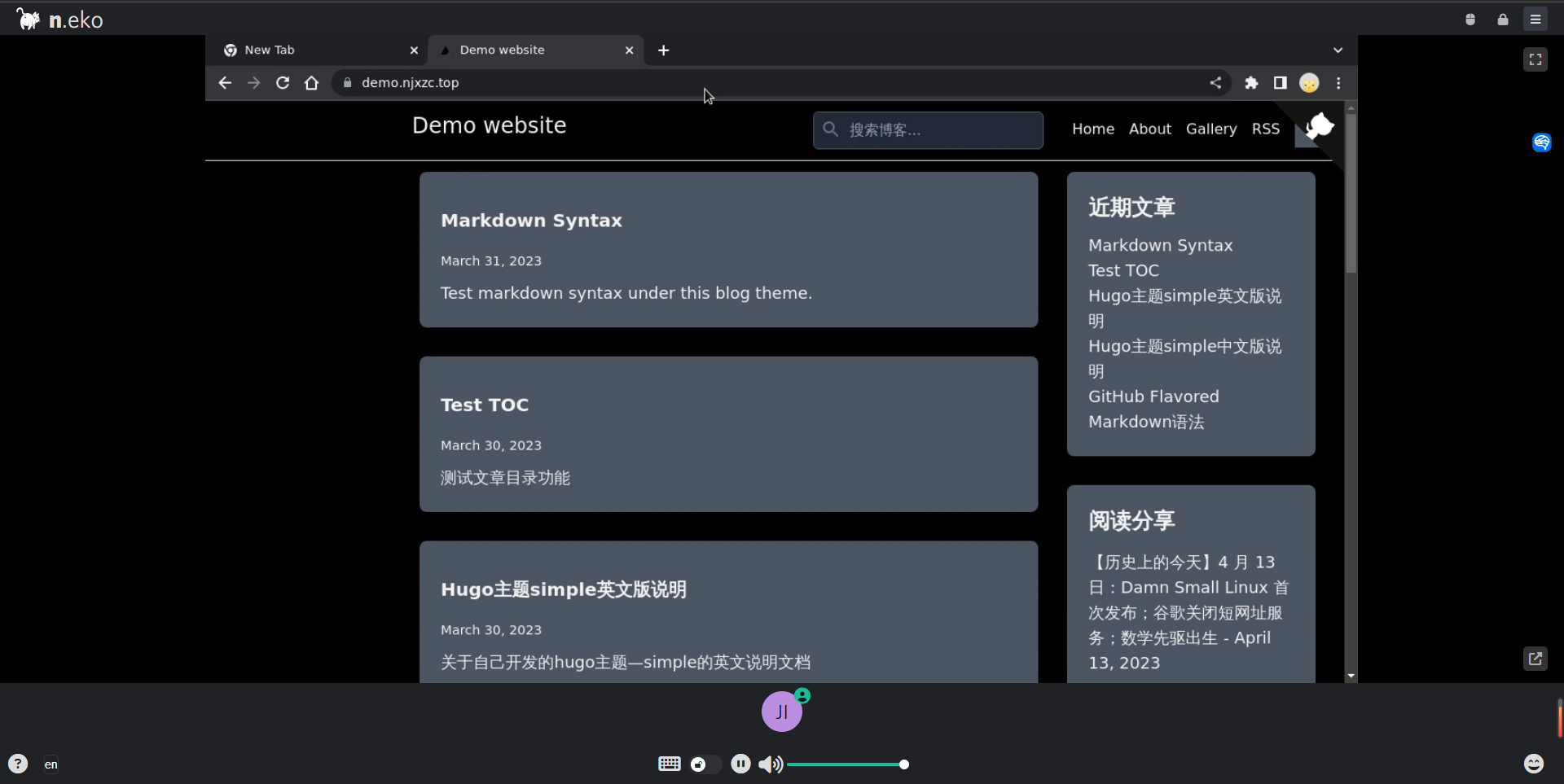


图5-13 Linux测试效果

我们在Android、iPhone、iPad、Windows、Mac和Linux设备上对系统进行了测试，并得到了令人满意的结果。

首先，我们通过采用响应式布局策略，确保了博客系统在不同屏幕尺寸和分辨率的设备上均能呈现出优美的界面。这种布局可以根据设备的屏幕尺寸自动调整，从而保证了在手机、平板和桌面设备上的视觉效果和交互体验。

其次，我们对博客系统在各种操作系统上的运行情况进行了检查，包括Windows、Mac和Linux。通过测试，我们发现系统在这些操作系统上均能正常运行，并且性能稳定。这意味着无论用户使用何种操作系统，都可以流畅地访问和使用我们的博客系统。

总之，经过全面的兼容性测试，我们的个人博客系统在各种设备和平台上表现出色，确保了广泛的用户群体可以无障碍地访问和使用。在后续的开发过程中，我们将继续关注兼容性问题，并随着技术和市场需求的变化，持续优化和改进博客系统，以满足更多用户的需求。

## 系统部署

在毕设的系统部署阶段，我们采用了现代化的开发工具和流程，以确保项目的可维护性和快速迭代。首先，我们使用Git进行版本控制，以便于对代码进行有效的管理和协作。通过将代码托管在GitHub上，我们可以充分利用其协作功能，同时确保项目的安全性和可靠性。

为了实现自动化部署和持续集成/持续部署（CI/CD）流程，我们选择了Vercel作为项目的部署平台。Vercel能够自动识别Next.js项目，并为其提供一键部署服务。当我们将项目导入Vercel时，平台会自动分析项目结构，识别出Next.js应用程序，并执行相应的构建和部署过程。

在项目开发过程中，每当我们向GitHub仓库推送代码更新时，Vercel会自动触发重新构建和部署过程，确保部署的系统始终与代码仓库中的最新版本保持一致。这种自动化的CI/CD流程使得我们可以快速迭代开发，更容易地发布新功能和修复错误。

通过利用Git、GitHub和Vercel等现代化开发工具和流程，我们的毕设项目在部署和维护方面表现出了高效和稳定性，从而保证了系统的可靠性和易用性。

* 1. **本章小结**

本章节从测试和部署两个方面对个人博客系统进行了全面评估。通过对性能、可访问性、网络安全和兼容性等关键领域的测试，确保了系统功能的正常可行性，并通过Git管理和Vercel自动化部署流程提高了系统的可维护性。这些测试和部署过程增强了项目的可行性和实用性。

# 结论

本次毕设项目采用Next.js框架、Tailwind CSS样式库、Supabase数据库以及Vercel部署平台，从零开始设计并实现了一个具有现代化特点的Web应用。在本次毕设中，我们成功实现了以下功能和成果：

1. 集成ChatGPT自动总结文章摘要：通过调用OpenAI的GPT模型，我们实现了自动为文章生成摘要的功能，使得用户在浏览文章列表时能够快速了解每篇文章的主要内容。
2. 评论区Cloudflare验证：为了确保评论系统的安全性，我们采用了Cloudflare的验证机制。在用户提交评论时，我们进行了一系列验证操作，有效地防止了垃圾评论和恶意攻击。
3. 响应式布局：利用Tailwind CSS样式库，我们实现了一个优雅的响应式布局，确保在不同设备和屏幕尺寸下的用户都能获得良好的使用体验。
4. 优化的性能和SEO：使用Next.js框架，我们实现了服务端渲染（SSR），从而提高了网页加载速度和搜索引擎优化（SEO）效果。

然而，在本次毕设过程中，我们也发现了一些不足之处：

1. 缺乏测试：由于时间和技能限制，本项目未能充分应用测试框架（如Jest[20]）进行单元测试和集成测试。这可能会导致未知的潜在问题和不稳定性。
2. 未使用TypeScript：尽管TypeScript能够提高代码的可读性和健壮性，但在本项目中我们仅使用了JavaScript。这意味着我们未能充分利用静态类型检查带来的优势。
3. 性能并未达到极致：一款现代化的个人博客系统的各个维度，在Lighthouse等测评中并未拿到满分，这意味着，性能方面有待提高。

尽管存在以上不足，但本次毕设为未来重构和优化提供了良好的基础。我将继续学习和探索，以期在未来的项目中更好地应用测试框架、TypeScript等技术，从而进一步提高应用的质量和稳定性。总之，本次毕设取得了一定的成果，为今后的学习和实践奠定了基础。

# 参考文献

[1] 朱海涛. 博客已死？移动互联网时代博客的价值[J]. 月光博客, 2015(01): 4287.

[2] 张旭. 博客关了,数据还好吗?[J]. 中国报业, 2018(17): 106.

[3] 杜运年. 从博客到微信:网络社交的窄化[J]. 新媒体研究, 2017, 3(02): 19-20.

[4] Ulysses. 博客是件玩具[J]. uselessulysses博客, 2021(01): 01.

[5] 博客时代落幕:你的青春和数据还好吗[J]. 中国报业, 2018(17): 106.

[6] 翁婷婷. 浅谈博客的个体化表达[J]. 西部广播电视, 2017(01): 28.

[7] Davis CJ, Heilemann M. WordPress[J]. Blog Design Solutions, 2006: 171-213.

[8] Dhillon V. Static Site Generators[J]. Creating Blogs with Jekyll, 2016: 21-33.

[9] Sean CD. Comparing Static Site Generator Build Times[J]. css-tricks, 2020(01): 01.

[10] So P. React[J]. Decoupled Drupal in Practice: Architect and Implement Decoupled Drupal Architectures Across the Stack, 2018: 313-334.

[11] Sun Y. Server-Side Rendering[J]. Practical Application Development with AppRun: Building Reliable, High-Performance Web Apps Using Elm-Inspired Architecture, Event Pub-Sub, and Components, 2019: 191-217.

[12] Loubser N. Hosting and CI/CD[J]. Software Engineering for Absolute Beginners: Your Guide to Creating Software Products, 2021: 313-324.

[13] openai. Introducing ChatGPT and Whisper APIs[J]. OpenAI, 2023: 1.

[14] Zammetti F. What Is JAMstack All About?[J]. Practical JAMstack: Blazing Fast, Simple, and Secure Web Development, the Modern Way, 2020: 1-17.

[15] Foster EC, Godbole SV. SQL Views and System Security[J]. Database Systems: A Pragmatic Approach, 2014: 259-278.

[16] Subramanian V. React State[J]. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node, 2017: 55-68.

[17] Klein A, zur Oven-Krockhaus I, Pastowski S. Tools zur Analyse des Internetauftritts kleiner und mittelständischer Unternehmen in der Tourismusbranche am Beispiel von Google Lighthouse[J]. Innovatives und digitales Marketing in der Praxis: Insights, Strategien und Impulse für Unternehmen, 2023: 115-138.

[18] CodingShower. How to Execute Rendered Script Tags with dangerouslySetInnerHTML in React? - CodingShower[J]. 2022.

[19] w3c organization. innerhtml[J]. W3C, 2008, 3(02): 0610.

[20] Thakkar M. Unit Testing Using Jest[J]. Building React Apps with Server-Side Rendering: Use React, Redux, and Next to Build Full Server-Side Rendering Applications, 2020: 153-174.

# 致谢

在本次毕设过程中，我深感荣幸能够在王小正老师的指导下独立完成这个项目。在此，我要向王老师表示由衷的敬意和感激。在论文撰写和课题研究过程中，王老师给予了我宝贵的建议和鼓励。他的专业素养和严谨的治学态度让我受益匪浅，这些都将成为我未来学习和工作的宝贵财富。

同时，我要感谢互联网上无私分享知识的人们。正是他们的慷慨奉献，让我有机会接触到丰富的学术资源，为我提供了解决问题的方法和灵感。这些在线资源对我的成长起到了关键性的作用，使我能够顺利地完成这个项目。

此外，我还要感谢我的家人和朋友们。在这段时间里，他们的支持和鼓励一直陪伴着我，给我带来了前进的动力。在遇到困难和挫折时，他们始终信任我、鼓励我，让我能够坚定信心，继续前行。

最后，我要感谢母校和所有为我提供帮助的老师和同学们。正是在这个充满智慧和激情的环境中，我学到了许多宝贵的知识和技能，为今后的人生道路打下了坚实的基础。

在此，我再次向所有给予我帮助和支持的人们表示衷心的感谢。我会继续努力，不负众望，为实现更高的目标而奋斗。

# 附录

1. 毕设项目GitHub仓库地址：<https://github.com/simple-is-awesome/simple>
2. 毕设项目说明文档地址：<https://docs.njxzc.top>
3. 毕设项目演示站点地址：<https://demo.njxzc.top>