DuckStudy 软件配置与运维文档

目录

- DuckStudy 软件配置与运维文档
  - 目录
  - 1. 项目概述
  - 2. 配置管理
    - 2.1 环境配置
      - 环境变量
      - 不同环境配置文件
    - 2.2 配置文件管理
      - 前端配置
      - 后端配置
    - 2.3 第三方库与依赖
      - 后端依赖
      - 前端依赖
  - 3. 版本控制
    - 3.1 版本控制系统
    - 3.2 分支策略
    - 3.3 提交规范
    - 3.4 代码审查流程
  - 4. 持续集成与持续部署
    - 4.1 CI/CD 工具选型
    - 自动化测试
      - 前端测试
      - 后端测试
    - 代码质量检查
      - 前端代码质量检查
      - 后端代码质量检查
  - 5. 部署方案
    - 开发环境部署
    - 测试环境部署
    - 生产环境部署
  - 6. 运维计划
    - 监控系统
      - 应用性能监控
      - 系统监控
    - 日志管理
      - 日志收集
    - 备份恢复
      - 数据备份策略

        - 恢复流程
    - 安全策略
        - 安全实施标准
        - 定期安全审计
    - 扩展与性能优化
        - 性能优化策略
        - 水平扩展计划
- 7. 异常处理与问题排查
    - 常见问题排查流程
    - 紧急恢复流程
- 8. 文档维护
    - 文档更新策略
    - 变更日志
        - 项目版本历史
        - 主要功能迭代
        - 技术架构变更
        - 关键问题修复记录
        - 未来迭代计划

1. 项目概述

DuckStudy 是我们团队精心打造的综合学习平台，旨在为广大学生提供便捷的学习资源整合服务。我们开发的平台集成了多种实用功能，包括学习网站导航、课程评价、论坛交流、二手交易市场以及热门 GitHub 项目展示等。在设计过程中，我们特别注重用户体验，采用了现代化的界面设计和交互模式，确保用户能够流畅、舒适地使用我们的平台。

从技术角度来看，我们的技术栈选择了当前稳定且流行的组合：

- **前端**：我们采用原生 HTML5/CSS3 构建基础结构，使用 Vanilla JavaScript (ES6+) 实现交互功能，并结合 Bootstrap 5 框架保证跨设备的适配性。论坛和评价系统中，我们整合了 Quill 富文本编辑器，为用户提供更丰富的内容创作体验。
- **后端**：选择 Python 3.8+ 作为核心语言，基于 Flask 轻量级框架构建 API 服务，使用 BeautifulSoup4 进行数据抓取和处理，特别是用于获取 GitHub 的热门项目数据。
- **数据存储**：当前阶段采用 JSON 文件存储结构化数据，并通过文件系统管理用户上传的图片资源，这种方案在开发初期简化了部署流程，后续将考虑迁移至关系型或文档型数据库。

2. 配置管理

2.1 环境配置

DuckStudy 项目实施多环境配置管理策略，严格区分开发环境、测试环境和生产环境。

环境变量

环境变量通过 `.env` 文件进行集中管理，并借助 `python-dotenv` 库实现自动加载。系统定义的关键环境变量包括：

```
# Flask 应用配置
FLASK_APP=backend/app.py
FLASK_ENV=development  # 或 production
FLASK_DEBUG=1  # 开发环境设为1，生产环境设为0
FLASK_SECRET_KEY=your-secret-key

# GitHub API 配置
GITHUB_TOKEN=your-github-token

# 服务端口配置
PORT=5000
```

不同环境配置文件

系统配置采用分环境配置文件结构，集中存放于 `backend/config` 目录：

- `config_dev.py` - 开发环境配置文件
- `config_test.py` - 测试环境配置文件
- `config_prod.py` - 生产环境配置文件

配置文件的加载通过环境变量 `CONFIG_TYPE` 动态控制，实现环境隔离与配置一致性。

2.2 配置文件管理

前端配置

前端配置通过 `frontend/js/config.js` 文件管理，包括 API 路径、请求超时设置等：

```
// 创建前端配置文件示例
const CONFIG = {
  API_BASE_URL: 'http://localhost:5000',
  REQUEST_TIMEOUT: 30000,  // 毫秒
  IMAGE_PATH: '/images',
  DEFAULT_AVATAR: '/images/avatars/default.png',
  LOCAL_STORAGE_KEY: 'duckstudy_user'
};
```

后端配置

后端配置通过 `backend/config/config.py` 文件管理，设计一个基于环境变量的配置系统：

```python
# 配置类示例
class Config:
    """基础配置类"""
    SECRET_KEY = os.getenv('FLASK_SECRET_KEY', 'default-secret-key')
    BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    UPLOAD_FOLDER = os.path.join(BASE_DIR, '..', 'frontend', 'images')
    MAX_CONTENT_LENGTH = 16 * 1024 * 1024  # 16MB 文件上传限制
    JSON_SORT_KEYS = False  # 保持 JSON 响应的顺序

class DevelopmentConfig(Config):
    """开发环境配置"""
    DEBUG = True
    TESTING = False

class TestingConfig(Config):
    """测试环境配置"""
    DEBUG = False
    TESTING = True

class ProductionConfig(Config):
    """生产环境配置"""
    DEBUG = False
    TESTING = False

# 配置字典
config = {
    'development': DevelopmentConfig,
    'testing': TestingConfig,
    'production': ProductionConfig
}
```

2.3 第三方库与依赖

后端依赖

后端依赖通过 `backend/requirements.txt` 进行版本化管理，核心依赖组件包括：

```
flask==3.0.2
flask-cors==4.0.0
requests==2.31.0
python-dotenv==1.0.0
beautifulsoup4==4.12.2
werkzeug==3.0.1
uuid==1.30
quart==0.20.0
```

所有依赖均采用精确版本锁定策略，确保开发、测试和生产环境的一致性，消除依赖版本兼容性风险。

前端依赖

前端依赖通过 `package.json` 管理，主要包括开发依赖：

```
{
  "devDependencies": {
    "jest": "^29.7.0",
    "jest-environment-jsdom": "^29.7.0",
    "babel-jest": "^29.7.0",
    "@babel/core": "^7.23.9",
    "@babel/preset-env": "^7.23.9"
  }
}
```

前端生产依赖主要包括 Bootstrap 5 和 Quill 富文本编辑器等第三方库，存放在 `frontend/lib` 目录下。

3. 版本控制

3.1 版本控制系统

DuckStudy 项目使用 Git 作为版本控制系统，基本操作流程如下：

- 初始化仓库：`git init`
- 克隆仓库：`git clone [repository URL]`
- 添加文件：`git add [file]` 或 `git add .`
- 提交更改：`git commit -m "[commit message]"`
- 查看状态：`git status`
- 推送更改：`git push origin [branch name]`
- 拉取更改：`git pull origin [branch name]`

3.2 分支策略

采用 Git Flow 分支模型管理代码，主要分支如下：

- `main`：主分支，存储正式发布的代码
- `develop`：开发分支，存储最新开发代码
- `feature/*`：功能分支，用于开发新功能
- `release/*`：发布分支，用于准备发布
- `hotfix/*`：热修复分支，用于紧急修复生产环境问题
- `bugfix/*`：用于修复非紧急的问题

分支命名规范：

- 功能分支：`feature/功能名称`，例如 `feature/user-authentication`
- 发布分支：`release/版本号`，例如 `release/v1.0.0`
- 热修复分支：`hotfix/问题简述`，例如 `hotfix/login-issue`

3.3 提交规范

采用 Angular 提交规范，格式如下：

```
<类型>(<作用域>): <主题>

<正文>

<脚注>
```

类型包括：

- `feat`：新功能
- `fix`：修复 Bug
- `docs`：文档更新
- `style`：代码风格调整（不影响代码功能）
- `refactor`：代码重构
- `test`：测试相关
- `chore`：构建过程或辅助工具的变动

3.4 代码审查流程

1. 开发者完成功能开发，提交代码到功能分支
2. 创建合并请求（Pull Request）到开发分支
3. 指定至少一名代码审查员
4. 代码审查员检查代码质量、功能实现和测试覆盖率
5. 通过自动化测试和代码质量检查
6. 代码审查员批准合并请求
7. 合并功能分支到开发分支

4. 持续集成与持续部署

4.1 CI/CD 工具选型

本项目采用 GitHub Actions 作为持续集成和持续部署工具，基于其与 GitHub 仓库的原生集成能力及配置的高效性。

标准化 CI/CD 流程：

1. 代码提交触发自动化测试
2. 通过测试后进行代码质量检查
3. 构建应用
4. 自动部署到对应环境

自动化测试

DuckStudy 项目中配置了前端和后端的自动化测试。

前端测试

使用 Jest 进行前端测试，配置在 `jest.config.js` 中：

```
module.exports = {
  // 测试环境，使用jsdom模拟浏览器环境
  testEnvironment: "jsdom",

  // 测试文件匹配模式
  testMatch: [
    "**/tests/frontend/**/*.test.js"
  ],

  // 测试覆盖率收集目录
  collectCoverageFrom: [
    "frontend/js/**/*.js",
    "!frontend/js/lib/**/*.js", // 排除第三方库
  ],

  // 覆盖率报告配置
  coverageDirectory: "tests/coverage",
  coverageReporters: ["json", "lcov", "text", "clover", "html"],
  coverageThreshold: {
    global: {
      branches: 70,
      functions: 80,
      lines: 80,
      statements: 80
    }
  }
}
```

运行前端测试的命令：

```
npm test
npm run test:coverage  # 生成覆盖率报告
```

后端测试

使用 pytest 进行后端测试，配置在 `tests/conftest.py` 中：

```python
import pytest

@pytest.fixture
def app_client():
    """提供Flask测试客户端"""
    # 配置测试环境
    from backend.app import app
    app.config.update({
        'TESTING': True,
        'SERVER_NAME': 'localhost',
    })

    # 创建临时测试数据
    # ...

    # 提供测试客户端
    with app.test_client() as client:
        yield client

    # 清理测试数据
    # ...
```

运行后端测试的命令：

```
python -m pytest tests/backend
python -m pytest tests/backend --cov=backend  # 生成覆盖率报告
```

代码质量检查

前端代码质量检查

JavaScript 代码质量控制通过 ESLint 静态分析工具实现，标准配置文件 `.eslintrc.js`：

```
module.exports = {
  "env": {
    "browser": true,
    "es2021": true,
    "jest": true
  },
  "extends": "eslint:recommended",
  "parserOptions": {
    "ecmaVersion": 12,
    "sourceType": "module"
  },
  "rules": {
    "indent": ["error", 2],
    "linebreak-style": ["error", "unix"],
    "quotes": ["error", "single"],
    "semi": ["error", "always"]
  }
};
```

运行 ESLint 的命令：

```
npx eslint frontend/js --ext .js
```

后端代码质量检查

Python 代码质量控制采用 flake8 静态分析工具，标准化配置文件 `.flake8`：

```
[flake8]
max-line-length = 100
exclude = .git,__pycache__,build,dist
ignore = E203, W503
```

运行 flake8 的命令：

```
flake8 backend
```

5. 部署方案

开发环境部署

开发环境部署主要用于本地开发和测试，步骤如下：

1. 克隆代码库：

   ```
   git clone https://github.com/yourusername/DuckStudy.git
   cd DuckStudy
   ```

2. 创建 Python 虚拟环境：

```
python -m venv venv
# Windows
venv\Scripts\activate
# Linux/MacOS
source venv/bin/activate
```

3. 安装依赖：

```
pip install -r backend/requirements.txt
npm install  # 如果使用 npm 管理前端依赖
```

4. 配置环境变量：

```
# 创建 .env 文件
echo "FLASK_APP=backend/app.py" > backend/.env
echo "FLASK_ENV=development" >> backend/.env
echo "FLASK_DEBUG=1" >> backend/.env
echo "FLASK_SECRET_KEY=dev-secret-key" >> backend/.env
echo "GITHUB_TOKEN=your_token_here" >> backend/.env
```

5. 启动应用：

```
python backend/app.py
# 或使用 Flask CLI
flask run
```

测试环境部署

测试环境部署用于团队内部测试和验证功能，建议使用自动化部署：

1. 配置 GitHub Actions 工作流程：
   `.github/workflows/test-deploy.yml`：

```yaml
name: Test Environment Deployment

on:
  push:
    branches: [ develop ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.8'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r backend/requirements.txt
        pip install pytest pytest-cov

    - name: Run tests
      run: |
        python -m pytest tests/

  deploy:
    needs: test
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Deploy to test server
      uses: appleboy/ssh-action@master
      with:
        host: ${{ secrets.TEST_SERVER_HOST }}
        username: ${{ secrets.TEST_SERVER_USER }}
        key: ${{ secrets.TEST_SERVER_SSH_KEY }}
        script: |
          cd /path/to/test/deployment
          git pull origin develop
          source venv/bin/activate
          pip install -r backend/requirements.txt
          systemctl restart duckstudy-test
```

2. 配置测试服务器：

```
# 创建系统服务
sudo nano /etc/systemd/system/duckstudy-test.service

# 服务文件内容
[Unit]
Description=DuckStudy Test Environment
After=network.target

[Service]
User=appuser
WorkingDirectory=/path/to/test/deployment
Environment="PATH=/path/to/test/deployment/venv/bin"
Environment="FLASK_ENV=testing"
Environment="GITHUB_TOKEN=your_token_here"
ExecStart=/path/to/test/deployment/venv/bin/python backend/app.py
Restart=always

[Install]
WantedBy=multi-user.target
```

3. 启动服务：

```
sudo systemctl enable duckstudy-test
sudo systemctl start duckstudy-test
```

生产环境部署

生产环境部署建议使用更稳定的方式，如 Gunicorn + Nginx：

1. 配置 GitHub Actions 工作流程：
   `.github/workflows/prod-deploy.yml`：

```yaml
name: Production Deployment

on:
  push:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.8'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r backend/requirements.txt
        pip install pytest pytest-cov

    - name: Run tests
      run: |
        python -m pytest tests/

  deploy:
    needs: test
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Deploy to production server
      uses: appleboy/ssh-action@master
      with:
        host: ${{ secrets.PROD_SERVER_HOST }}
        username: ${{ secrets.PROD_SERVER_USER }}
        key: ${{ secrets.PROD_SERVER_SSH_KEY }}
        script: |
          cd /path/to/production/deployment
          git pull origin main
          source venv/bin/activate
          pip install -r backend/requirements.txt
          pip install gunicorn
          systemctl restart duckstudy-prod
```

2. 安装和配置 Gunicorn：

```
pip install gunicorn

# 创建 Gunicorn 启动脚本
nano /path/to/production/deployment/gunicorn_start.sh
```

脚本内容：

```
#!/bin/bash

NAME="duckstudy"
DIR=/path/to/production/deployment
USER=appuser
GROUP=appuser
WORKERS=3
BIND=unix:$DIR/gunicorn.sock
PYTHONPATH=$DIR
ENV_PATH=$DIR/venv/bin/

cd $DIR
source ${ENV_PATH}activate

export FLASK_APP=backend/app.py
export FLASK_ENV=production
export GITHUB_TOKEN=your_token_here

exec ${ENV_PATH}gunicorn backend.wsgi:app \
  --name $NAME \
  --workers $WORKERS \
  --user=$USER \
  --group=$GROUP \
  --bind=$BIND \
  --log-level=info \
  --log-file=$DIR/logs/gunicorn.log
```

3. 配置 Nginx：

```
server {
    listen 80;
    server_name yourdomain.com;

    access_log /var/log/nginx/duckstudy_access.log;
    error_log /var/log/nginx/duckstudy_error.log;

    location / {
        proxy_pass http://unix:/path/to/production/deployment/gunicorn.sock;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /static {
        alias /path/to/production/deployment/frontend;
    }
}
```

4. 配置系统服务：

```
sudo nano /etc/systemd/system/duckstudy-prod.service
```

服务文件内容：

```
[Unit]
Description=DuckStudy Production
After=network.target

[Service]
User=appuser
Group=appuser
WorkingDirectory=/path/to/production/deployment
ExecStart=/path/to/production/deployment/gunicorn_start.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

5. 启动服务：

```
sudo systemctl enable duckstudy-prod
sudo systemctl start duckstudy-prod
sudo systemctl status duckstudy-prod
```

6. 运维计划

监控系统

应用性能监控

应用性能监控基于 Prometheus 时序数据库和 Grafana 可视化平台构建：

1. Prometheus 服务器配置：

   ```
   # 创建配置文件
   mkdir -p /etc/prometheus
   nano /etc/prometheus/prometheus.yml
   ```

   标准监控配置：

   ```
   global:
     scrape_interval: 15s

   scrape_configs:
     - job_name: 'duckstudy'
       static_configs:
         - targets: ['localhost:5000']
   ```

2. Flask 应用监控集成：

   ```
   pip install prometheus-flask-exporter
   ```

   应用入口文件 `app.py` 中的监控代码：

   ```
   from prometheus_flask_exporter import PrometheusMetrics

   metrics = PrometheusMetrics(app)
   ```

3. Grafana 指标可视化实现

系统监控

系统资源监控通过 Node Exporter 实现，用于收集服务器的 CPU、内存、磁盘等系统指标，并集成至 Prometheus 监控平台。

日志管理

日志收集

日志管理系统采用 ELK 栈（Elasticsearch、Logstash、Kibana）架构，用于集中式日志收集、存储、分析和可视化：

1. 配置应用日志格式：

```
import logging
from logging.handlers import RotatingFileHandler
import os

def setup_logging(app):
    if not os.path.exists('logs'):
        os.mkdir('logs')
    file_handler = RotatingFileHandler('logs/duckstudy.log', maxBytes=10240, backupCount=10)
    file_handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]'
    ))
    file_handler.setLevel(logging.INFO)
    app.logger.addHandler(file_handler)
    app.logger.setLevel(logging.INFO)
    app.logger.info('DuckStudy startup')
```

2. 使用 Filebeat 收集日志并发送到 Elasticsearch

备份恢复

数据备份策略

针对 DuckStudy 基于文件的数据存储方式，设计备份策略：

1. 定时备份：

```
# 创建备份脚本
nano /path/to/backup.sh
```

脚本内容：

```
#!/bin/bash

TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BACKUP_DIR="/path/to/backups"
SOURCE_DIR="/path/to/production/deployment/frontend/data"

# 创建备份目录
mkdir -p "$BACKUP_DIR"

# 创建归档
tar -czf "$BACKUP_DIR/duckstudy_data_$TIMESTAMP.tar.gz" "$SOURCE_DIR"

# 保留最近30天的备份
find "$BACKUP_DIR" -type f -name "duckstudy_data_*.tar.gz" -mtime +30 -delete
```

2. 设置 Cron 任务：

```
# 每天凌晨3点执行备份
0 3 * * * /path/to/backup.sh >> /var/log/duckstudy_backup.log 2>&1
```

恢复流程

1. 选择需恢复的备份：

```
ls -l /path/to/backups
```

2. 恢复数据：

```
# 停止应用
sudo systemctl stop duckstudy-prod

# 解压备份文件到临时目录
mkdir -p /tmp/duckstudy_restore
tar -xzf /path/to/backups/duckstudy_data_20250615_030000.tar.gz -C /tmp/duckstudy_restore

# 复制数据文件
cp -r /tmp/duckstudy_restore/path/to/production/deployment/frontend/data/* /path/to/production/deploy

# 启动应用
sudo systemctl start duckstudy-prod
```

安全策略

安全实施标准

1. 传输层安全措施：
   - SSL/TLS加密配置：部署 Let's Encrypt 授权的证书
   - Nginx HTTPS 强制策略实施
2. 密码安全机制：

```
# 密码安全存储实现（bcrypt算法）
import bcrypt

def hash_password(password):
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password.encode('utf-8'), salt)
    return hashed.decode('utf-8')

def check_password(hashed, password):
    return bcrypt.checkpw(password.encode('utf-8'), hashed.encode('utf-8'))
```

3. CSRF（跨站请求伪造）防护机制：

```
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect(app)
```

4. XSS（跨站脚本）防御策略：
   - 输出内容HTML自动转义处理

- 内容安全策略（CSP）头实施

5. 文件上传安全控制：

```
# 文件上传类型与大小限制实现
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in {'png', 'jpg', 'jpeg', 'gif'}

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400
    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        return jsonify({'success': True, 'filename': filename})
    return jsonify({'error': 'File type not allowed'}), 400
```

定期安全审计

建立定期安全审计流程：

- 每季度进行一次安全扫描和代码审查
- 使用工具如 OWASP ZAP 进行安全漏洞扫描
- 审查和更新依赖库，修复已知漏洞

扩展与性能优化

性能优化策略

1. 前端性能优化方案：
   - 静态资源压缩与合并（CSS、JavaScript），减少网络传输开销
   - HTTP缓存策略实施，设置合理的Cache-Control响应头
   - 非关键资源延迟加载策略，提高首屏渲染速度
2. 后端性能优化架构：
   - 数据访问层缓存机制，减少重复计算和I/O操作
   - 内存缓存实现：

```python
# 应用级内存缓存实现
cache = {}

def get_cached_data(key, ttl=300):
    """获取缓存数据，ttl为过期时间（秒）"""
    if key in cache and time.time() - cache[key]['time'] < ttl:
        return cache[key]['data']
    return None

def set_cached_data(key, data):
    """设置缓存数据"""
    cache[key] = {'data': data, 'time': time.time()}
```

3. 数据库性能优化规范（数据库迁移后实施）：
    - 索引策略：针对高频查询字段建立适当索引
    - SQL查询优化：避免全表扫描，减少连接操作
    - 数据库连接池配置，避免频繁建立连接的开销

水平扩展计划

根据用户量增长制定的系统水平扩展策略：

1. 静态资源分发优化：
    - 实施 CDN（内容分发网络）架构，选用 Amazon CloudFront 或 Cloudflare
    - 重构前端资源路径，适应 CDN 分发模式
2. 负载均衡机制：
    - 部署应用集群，实现多实例分布式架构
    - 实施 Nginx 反向代理负载均衡或云服务提供商负载均衡服务
3. 数据存储架构优化：
    - 数据层迁移方案：从文件存储升级至关系型数据库（PostgreSQL）或文档型数据库（MongoDB）
    - 引入 Redis 缓存层，优化高频数据访问性能及会话状态管理

7. 异常处理与问题排查

常见问题排查流程

1. 检查应用日志：

```
tail -n 100 /path/to/production/deployment/logs/duckstudy.log
```

2. 检查系统日志：

```
journalctl -u duckstudy-prod -n 100
```

3. 检查 Nginx 日志：

```
tail -n 100 /var/log/nginx/duckstudy_error.log
```

4. 验证服务状态：

```
systemctl status duckstudy-prod
```

5. 进行健康检查：

```
curl -I http://localhost:5000/api/health
```

紧急恢复流程

1. 回滚版本：

```
cd /path/to/production/deployment
git reset --hard <previous_commit_hash>
systemctl restart duckstudy-prod
```

2. 检查并恢复最近备份：

```
# 参考备份恢复流程
```

8. 文档维护

文档更新策略

1. 在每次重大更改后更新本文档
2. 每季度审查一次文档内容
3. 维护变更日志
4. 在每次功能迭代后更新变更日志部分
5. 记录所有重要的技术栈变更和架构调整

变更日志

项目版本历史

- **v1.0.0 (2025-06-20)**：软件配置与运维文档初始版本创建
- **v0.9.0 (2025-06-18)**：完成测试配置和自动化测试框架搭建 (提交: 0212c91)
- **v0.8.5 (2025-06-17)**：新增添加课程功能、修复课程列表评分动态展示bug (提交: cef4c19)
- **v0.8.0 (2025-06-16)**：添加用户历史记录功能，优化系统性能 (提交: bb437ec)
- **v0.7.5 (2025-06-15)**：修改导航网页，优化用户体验 (提交: ce89515)
- **v0.7.0 (2025-06-11)**：实现用户收藏功能 (提交: 349debd)
- **v0.6.0 (2025-05-21)**：完善个人信息界面展示，修复头像更换功能，添加3D魔方小游戏 (提交: 6eea228)
- **v0.5.5 (2025-05-20)**：论坛与个人页面功能完善 (提交: 9e2e196)
- **v0.5.0 (2025-05-19)**：实现评论排序功能 (提交: 303f24a)
- **v0.4.0 (2025-05-16)**：系统功能完善与bug修复 (提交: a13800d)
- **v0.3.5 (2025-05-15)**：论坛功能完善，项目整体优化 (提交: 916b3be)
- **v0.3.0 (2025-05-08)**：完善课程评价系统 (提交: ab6eeaa)
- **v0.2.0 (2025-04-18)**：系统框架优化和功能扩展 (提交: edfa144)
- **v0.1.0 (2025-04-04)**：项目初始结构搭建 (提交: 2b46cab)

主要功能迭代

1. **课程模块**：
   - 2025-06-17：添加课程功能实现，用户可自行添加课程
   - 2025-05-08：完善课程评价系统，添加评分和评价标签
   - 2025-04-18：初始课程列表和详情页实现
2. **用户功能**：
   - 2025-06-16：添加历史记录功能，记录用户浏览痕迹
   - 2025-06-11：实现用户收藏功能，支持收藏课程和帖子
   - 2025-05-21：完善用户个人信息界面，修复头像更换功能
   - 2025-04-04：用户注册登录基本功能实现
3. **论坛功能**：
   - 2025-05-20：论坛页面完善，支持多种内容展示形式
   - 2025-05-19：评论排序功能实现，提高用户体验
   - 2025-05-15：论坛基础功能完善，支持富文本编辑
   - 2025-04-18：论坛基础结构设计与实现
4. **导航功能**：
   - 2025-06-15：优化导航页面设计，提高用户体验
   - 2025-04-04：基础导航功能实现
5. **其他功能**：
   - 2025-05-21：添加3D魔方小游戏，丰富平台功能
   - 2025-04-18：GitHub热门项目展示功能实现

技术架构变更

1. **前端**：
   - 2025-05-16：更新依赖，优化前端性能
   - 2025-05-15：整合Bootstrap与自定义CSS，提升界面美观度
   - 2025-04-04：初始前端架构搭建，采用HTML5/CSS3和原生JavaScript
2. **后端**：
   - 2025-06-18：完善测试框架，增强系统稳定性
   - 2025-05-16：优化API设计，提高响应速度
   - 2025-04-17：完善后端架构，实现基础REST API
3. **测试与部署**：
   - 2025-06-18：完成自动化测试配置
   - 2025-05-08：优化部署流程，提高开发效率
   - 2025-04-04：初始开发环境配置

关键问题修复记录

1. **功能性Bug修复**：
   - 2025-06-17：修复课程列表评分动态展示bug
   - 2025-05-21：修复用户头像无法正确更换问题
   - 2025-05-16：修复多处UI显示问题
   - 2025-05-16：修复用户注销后缓存未清理问题
2. **性能优化**：
   - 2025-06-16：优化系统性能，减少页面加载时间

- 2025-05-16：优化图片加载和处理流程
- 2025-04-18：优化API响应速度

3. **安全增强**：
   - 2025-05-16：增强密码安全性要求
   - 2025-04-17：改进用户身份验证机制

未来迭代计划

1. **v1.1.0 计划（2025年7月）**：
   - 数据存储优化：从JSON文件存储迁移到关系型数据库
   - 引入Redis缓存系统，提高高频访问数据的响应速度
   - 实现更完善的用户权限管理系统
2. **v1.2.0 计划（2025年8月）**：
   - 引入AI推荐系统，为用户推荐课程和学习资源
   - 改进移动端兼容性，提供更好的响应式设计
   - 添加多语言支持
3. **v2.0.0 计划（2025年第四季度）**：
   - 重构前端架构，采用现代前端框架（如React或Vue.js）
   - 实现微服务架构，拆分后端服务
   - 添加实时通知系统
   - 引入容器化部署方案，使用Docker和Kubernetes
4. **持续优化计划**：
   - 定期代码重构，保持代码质量
   - 扩展自动化测试覆盖率
   - 优化CI/CD流程，缩短部署时间
   - 增强安全性，定期进行安全审计