**概要设计说明书（High-Level Design）**

Edge Saved – Article Bookmarking App

**文档版本：**v0.1

**作者：**（示例）Yan Huang

**最后更新：**2025-12-24

**读者：**后端工程师 / 全栈工程师

# 0. 执行摘要（Executive Summary）

Edge Saved 是一个基于 Cloudflare Workers 的 Edge SSR 示例项目，用于演示在无数据库、无 REST API 的前提下，如何在 Edge Runtime 环境中安全地实现用户级状态（文章收藏）的持久化与首屏渲染。

本设计的核心目标包括：

* 在 SSR 首屏阶段准确渲染用户收藏状态
* 防止客户端篡改用户状态数据
* 避免在 Edge 场景中过度引入复杂基础设施
* 在可扩展性与实现复杂度之间取得合理平衡

本文档重点关注系统设计动机、关键架构决策以及方案取舍；具体代码结构与运行方式请参考项目 **README.md**。

# 1. 项目说明

## 1.1 背景

在内容类应用中（新闻、博客、资讯平台），**“文章收藏”是一个高频但容易被低估的功能**。

在 Edge / SSR 场景下，该功能面临以下挑战：

* 用户状态（是否已收藏）需要在 **首屏 SSR 阶段即正确渲染**
* 不能依赖客户端本地状态（如 localStorage）
* 需要防止客户端篡改收藏状态
* Edge 环境下不适合引入数据库或复杂持久化系统

因此，本项目设计并实现一个 **基于 Cloudflare Workers 的 SSR 收藏功能示例**，用于验证在 Edge Runtime 中，如何使用 **签名 Cookie** 安全地持久化用户状态。

## 1.2 目标

本项目的目标包括：

* **服务端渲染（SSR）**
  + 首次页面加载即展示正确的收藏状态
  + 收藏数量与按钮状态由服务器计算并渲染
* **安全的状态持久化**
  + 使用 Cookie 存储收藏状态
  + 通过 HMAC-SHA256 对 Cookie 内容进行签名，防止篡改
* **简洁清晰的交互模型**
  + 使用 HTTP POST / Action 完成收藏切换
  + 不引入前后端分离或复杂 REST API
* **符合 Edge Runtime 约束**
  + 不使用 Node.js API
  + 使用 Web Crypto API（crypto.subtle）

## 1.3 非目标

为保持示例简洁，本项目 **不覆盖以下内容**：

* 不接入数据库（D1 / KV / 外部 DB）
* 不实现用户登录或鉴权系统
* 不实现复杂 UI 或样式系统
* 不考虑跨设备同步收藏状态

# 2. 需求澄清与约束

## 2.1 功能性需求

**文章列表页（GET /）**

* 展示固定数量的文章（硬编码）
* 显示每篇文章的收藏 / 未收藏状态
* 页面顶部展示当前收藏文章数量

**已收藏页（GET /saved）**

* 仅展示已收藏文章
* 若无收藏，显示空状态提示并提供返回入口

**收藏切换（POST /toggle）**

* 接收 articleId
* 根据当前 Cookie 状态进行收藏 / 取消收藏
* 更新 Cookie 并返回页面响应

## 2.2 非功能性需求

**安全性**

* 客户端不得伪造或篡改收藏列表
* 签名校验失败时必须安全降级

**缓存正确性**

* 所有依赖用户 Cookie 的 SSR 响应必须禁止共享缓存
* 防止用户间状态串读

**可维护性**

* 收藏逻辑、签名逻辑职责清晰
* 易于后续扩展更多 Cookie 状态

# 3. 思考过程

核心问题可以归纳为三点：

1. **如何在 SSR 阶段获得可靠的用户收藏状态**
2. **如何在无数据库的前提下安全地持久化状态**
3. **如何在 Edge Runtime 中实现加密签名**

针对以上问题，本设计选择：

* 使用 **Cookie 作为唯一状态存储**
* 使用 **HMAC-SHA256 签名** 确保完整性
* 通过 **React Router v7 loader / action** 实现完整的 SSR + 状态变更闭环

该方案在复杂度、安全性与 Edge 适配性之间取得平衡。

# 4. 方案对比与选型（Design Alternatives & Trade-offs）

在实现 Edge Saved 文章收藏功能时，核心问题在于：  
 **如何在 Edge SSR 环境中安全、可靠地保存用户的收藏状态，并在首屏渲染阶段即可使用。**

围绕这一问题，这里主要评估了以下几种方案。

## 4.1 方案一：纯客户端状态（LocalStorage / Client State）

方案描述

* 收藏状态仅保存在浏览器 localStorage 或前端内存中
* 页面初始 SSR 不包含收藏状态
* 客户端 hydrate 后再读取本地状态并更新 UI

优点

* 实现最简单
* 无服务端状态、无加密逻辑
* 不涉及 Cookie 或安全问题

缺点

* ❌ **不满足 SSR 要求**
  + 首屏无法渲染正确的收藏状态
* ❌ 页面刷新前后 UI 会闪动（hydration mismatch）
* ❌ 收藏状态完全由客户端控制，易被篡改
* ❌ 与 Edge SSR 的设计目标不符

## 4.2 方案二：服务端数据库 / KV 存储

方案描述

* 将收藏状态存储在：
  + 数据库（如 Postgres）
  + Cloudflare KV / D1
* SSR Loader 从 DB/KV 中读取用户收藏状态

优点

* 状态可靠、可扩展
* 支持跨设备同步
* 数据可持久化、可分析

缺点

* ❌ 明显超出题目范围（题目明确不要求 DB）
* ❌ 增加部署、运维、权限复杂度
* ❌ 对一个 Demo 级项目属于 **过度设计**
* ❌ Edge 冷启动与外部存储访问会引入额外延迟

结论

* 技术上可行，但**不符合本次 Interview Task 的约束与预期**
* 作为未来扩展方向保留，但不选用

## 4.3 方案三：未签名的普通 Cookie

方案描述

* 使用 Cookie 保存收藏文章 ID 列表
* SSR Loader 直接读取 Cookie 并渲染页面
* 不对 Cookie 内容进行签名或校验

优点

* 实现简单
* 满足 SSR 首屏渲染要求
* 无需额外存储系统

缺点

* ❌ Cookie 内容可被客户端任意篡改
* ❌ 无法区分合法状态与伪造状态
* ❌ 不符合安全性要求
* ❌ 题目明确要求“防止篡改”

结论

* 在功能层面可行
* 但**不满足安全要求**
* 不可作为最终方案

## 4.4 方案四：签名 Cookie（HMAC）【最终选用】

方案描述

* 使用 Cookie 存储收藏状态 Payload
* 使用 HMAC-SHA256 对 Payload 进行签名
* SSR Loader 在服务端校验签名后再使用数据
* 校验失败时安全降级（视为无收藏）

优点

* ✅ 满足 SSR 首屏渲染要求
* ✅ 无需数据库或外部存储
* ✅ 防止客户端篡改状态
* ✅ 非常适合 Edge Runtime（无状态）
* ✅ 实现复杂度可控，逻辑清晰

缺点

* Cookie 大小有限（需控制 payload）
* 状态仅限单设备 / 单浏览器
* 密钥轮换需要额外设计（可扩展）

在综合评估多种可选方案后，本项目选择使用 Signed Cookie + Edge SSR 的实现方式，

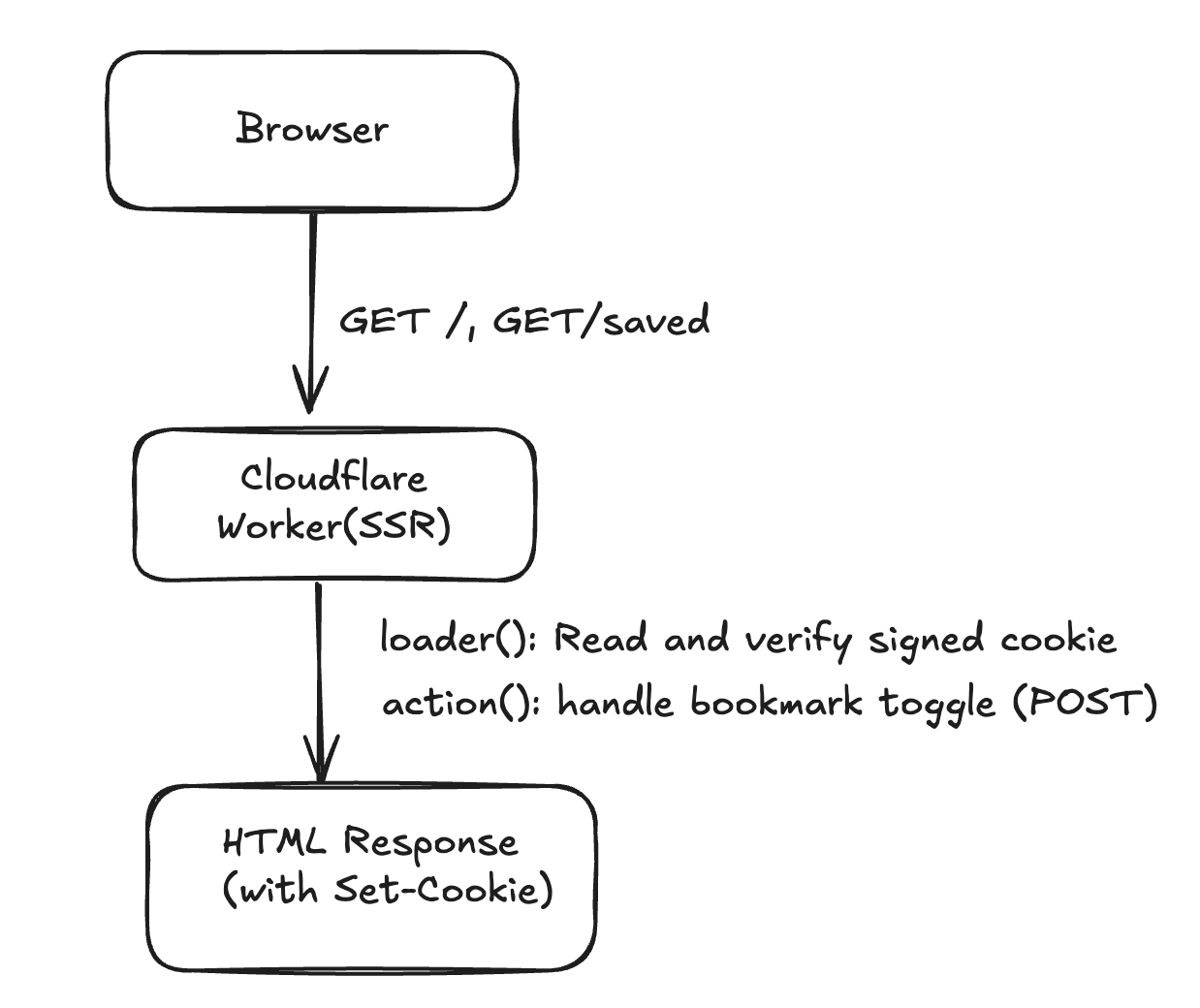
主要基于以下考量：

* 复杂度控制：避免为示例项目引入数据库或额外状态存储系统
* **安全性要求：用户收藏状态需要防止客户端篡改**
* SSR 一致性：首屏即渲染正确的用户状态
* Edge 适配性：方案需完全兼容 Cloudflare Workers Runtime
* 面试场景适配：突出架构取舍能力而非堆叠技术组件

在当前约束条件下，该方案在可维护性、安全性与实现成本之间取得了最佳平衡。

# 5. 总体架构设计

## 5.1 架构概览



整体架构采用典型的 **Edge SSR** 请求模型：

1. 浏览器向应用发起请求（**GET** / 或 **GET** /**saved**）

2. 请求直接到达 **Cloudflare Worker（SSR 入口）**

3. **Worker** 在处理请求时执行对应路由的 **loader** 或 **action**

4. **loader** 负责读取并校验 **Signed Cookie**，以**恢复用户收藏状态**

5. **action** 负责**处理收藏 / 取消收藏请求**，并生成新的签名 **Cookie**

6. **Worker** 返回 **SSR HTML** 响应，并在必要时通过 **Set-Cookie** 更新客户端状态

通过将用户状态处理集中在 Worker 的 SSR 阶段，系统避免了客户端状态不一致与首屏闪烁问题。

## 5.2 核心组件职责

**React Router Loader**

* 解析请求 Cookie
* 校验签名
* 计算收藏状态
* 返回 SSR 数据

**React Router Action**

* 接收 POST 请求
* 执行收藏 toggle
* 重新签名并写回 Cookie

**Cookie Utils**

* 统一处理：
  + payload 编码
  + HMAC 签名
  + 校验与反序列化

# 6. 接口设计

## 6.1 收藏切换接口

POST /toggle

请求参数（Form / Body）

字段 说明

articleId 文章唯一ID

**返回行为**

* 默认：HTTP Redirect 回来源页面
* 可选：Fetcher 无刷新更新

## 6.2 Cookie 与签名设计

### 6.2.1 Cookie Payload 结构

{

"ids": ["a1", "a3"],

"iat": 1700000000

}

* ids：已收藏文章 ID 列表
* iat：签发时间（用于调试与扩展）

### 6.2.2 签名机制

* 算法：HMAC-SHA256
* 密钥：APP\_SECRET（环境变量）
* 格式：

base64(payload).base64(signature)

### 6.2.3 校验失败策略

* 若解析或签名校验失败：
  + 视为无有效收藏
  + 可选择清空 Cookie
  + 页面正常渲染，不抛致命错误

## 6.3 SSR 与缓存控制

### 6.3.1 SSR 行为

* 所有页面均通过 Loader 在服务器渲染
* 客户端仅负责 hydration

### 6.3.2 HTTP Header 策略

Cache-Control: private, no-store

原因：

* 页面内容依赖用户 Cookie
* 禁止边缘或共享缓存复用

## 6.4 错误处理设计

### 6.4.1 Error Boundary

* 捕获 loader / action 中的异常
* 显示友好的错误提示

### 6.4.2 覆盖的错误场景

* Cookie 签名校验失败
* 非法 articleId
* 未预期的运行时异常

# 7. 扩展性考虑

## 7.1 当前方案的限制

* Cookie 大小限制了可存储的收藏数量
* 收藏状态仅在当前浏览器中生效，不支持跨设备同步
* Cookie 签名密钥轮换机制未在本示例中实现
* 不适用于需要强一致或复杂查询的业务场景

## 7.2 潜在的演进方向

* 使用 Cookie 仅作为 Session 标识，状态存储迁移至 KV 或数据库
* 引入版本号与密钥轮换机制以提升安全性
* 支持异步交互与局部刷新以改善用户体验
* 在业务复杂度提升后引入更完整的权限与风控体系