基本システム仕様書

2025/12/07

中塚季利人

履歴

日付	バージョン	作者	内容
2025/12/07	1.00	中塚	新規追加

目次

1. はじめに

本書は、ショートドラマ動画配信アプリ（Web + WebView アプリ）の基本設計を記述する。本システムは Cloudflare を基盤としたサーバーレス構成を採用し、高いパフォーマンスとスケーラビリティを持つ動画配信プラットフォームを実現する。

---

2. 開発の目的

- 新規俳優・スタッフの活躍を支援する「推し活プラットフォーム」の構築
- 独自コンテンツ（ショートドラマ）の配信
- ファンによる評価・レビュー・課金（コイン）を提供する総合アプリ

---

3. システムの用途

- スマホアプリ（WebView）および Web サイトで動画を視聴
- ユーザが作品・キャストを評価、コメント
- コイン購入および動画閲覧
- キャスト・スタッフが人気指標を把握
- 管理者が作品管理・承認作業・コメント承認などを行う

---

4. 対象となるユーザ

- 一般視聴者
- キャスト・スタッフ（出演者・制作関係者）
- 配信管理者

---

5. 基本設計方針

- フロント・バックエンドの明確な分離（**API**ファースト）
- **Cloudflare Workers API**
- **GraphQL** によるデータ取得
- **Next.js** による単一コードベースで **Web / WebView** を統一
- **Cloudflare Stream** で動画配信
- 認証方式（メール認証 **+ SMS 2**段階認証）
- 可観測性・運用性（ログ・エラー処理・レート制限）

---

6. システム設計方針

6-1. アーキテクチャ構成

```
スマホアプリ（Capacitor WebView）
Webブラウザ（Next.js）
        ↓
GraphQL Gateway（Apollo Server）
        ↓
Cloudflare Workers API
        ├ Cloudflare D1（DB）
        ├ Cloudflare R2（サムネ・画像）
        ├ Cloudflare Stream（動画）
        ├ Twilio or NTTドコモ SMS API
        └ Resend / SendGrid（メール）
```

6-2. データベース / ストレージ構成

- **D1（SQLiteベース）**
    - ユーザ
    - プロフィール
    - 作品
    - エピソード
    - レビュー
    - コメント
    - コインウォレット
    - コイン取引
    - 視聴履歴
    - ランキング
- **R2**
    - サムネイル画像
    - プロフィール画像
- **Cloudflare Stream**
    - 動画本体

6-3. 認証方式（重要）

① メール認証

- 新規登録時に認証メールを送信（6桁コード or Magic Link）
- Workers でメール送信 API を呼び出す（Resend/SendGrid）

② SMS 認証（2段階認証）

- 電話番号を登録
- Twilio などで 6桁 SMS コードを送信
- `two_factor_tokens` を D1 or KV に保持し、期限つき検証

③ トークン管理

- セキュアに扱うため **HttpOnly Cookie + JWT**（署名付き）を採用
- フロント・WebView 共通でログイン状態保持

④ アクセス制御

- ロール：
  - viewer
  - performer
  - admin
- GraphQL Resolver で role-based access control を実装

6-4 GraphQL 設計概要

主な**Query**

- `works`
- `work(id)`
- `episodes(workId)`
- `reviews(workId)`
- `search(keyword)`
- `ranking(type)`

- `viewerProfile`
- `me`（ログインユーザ情報）

主なMutation

- `createUser`
- `verifyEmail`
- `login`
- `enable2FA`
- `verifySMS`
- `purchaseCoin`
- `spendCoin`
- `submitReview`
- `submitComment`
- `adminApproveComment`
- `uploadThumbnail`
- `registerWork`
- `registerEpisode`

---

7. 実装方針

7-1. フロントエンド

- Next.js（App Router）

- Apollo Client
- UI: Tailwind CSS + shadcn/ui
- Capacitor で WebView ラップ
- 動画再生：Cloudflare Stream Player

7-2. バックエンド（**Workers**）

- TypeScript
- Hono or Miniflare + Apollo Server
- 認証は Middleware で全 GraphQL リクエストを検証
- Worker 内で Cloudflare API（Stream / R2 / D1）を呼び出す

7-3. セキュリティ

- CSRF対策：Cookie + SameSite 設定
- レート制限：Cloudflare Rules / Turnstile
- 画面録画防止：アプリ側（iOS/Android）実装

---

8. テスト方針

- 単体テスト：Jest（Resolver / Logic）
- E2Eテスト：Playwright（Web）
- 負荷テスト：k6
- 動作検証：iOS / Android WebView 動作確認

---

9. リリース後の更新

- ランキングロジックアップデート
- コメントNGワードAI（Phase 2）
- プッシュ通知（新作公開など）
- 検索強化（Meilisearch移行など）

---

10. 機能一覧

★ 基本機能

- ログイン / 会員登録（メール + パスワード）
- 2段階認証（SMS）
- チュートリアル（初回のみ）
- 検索機能
- ピックアップ動画 / おすすめ動画
- ランキング（日次更新）
- コイン購入 / コイン消費 / コイン換金
- MYページ（履歴 / お気に入り）
- プロフィール編集（承認制）
- コメント / 評価（承認制）

★ 作品・動画

- 作品情報表示
- エピソード再生

- サムネ登録
- 字幕なし（Cloudflare Stream に埋め込む）

★ 管理機能

- ユーザ管理
- コメント承認
- 作品・エピソード登録
- ピックアップ編集
- コイン換金管理

---

11. システム構成

```
Cloudflare
 │
 ├── Pages（Next.js）
 ├── Workers（GraphQL API）
 ├── R2（画像）
 ├── D1（データベース）
 ├── Stream（動画）
 ├── KV（2FAトークン等）
 └── Turnstile（Bot対策）
```

---

12. ハードウェア構成

- Cloudflare サーバーレス
- 物理サーバー選定不要

---

13. ソフトウェア構成

区分	使用技術
フロント	Next.js / TypeScript / Apollo Client
アプリ	Capacitor（iOS/Android）
API	Cloudflare Workers / Apollo Server / Hono
DB	Cloudflare D1
ストレージ	Cloudflare R2
動画配信	Cloudflare Stream
認証	JWT / Cookie / 2FA（SMS）
メール	Resend / SendGrid

---

14. 目標性能

- 初期 10,000 MAU
- 動画再生遅延：2秒以内
- API 応答速度：200ms以内
- 同時接続：5,000 を想定

---

15. 制限事項

- Cloudflare D1 はリレーショナルだが大規模運用に注意
- Worker は CPU時間に制限
- 大規模データ処理は Durable Objects や Queue を併用

---

16. 他システムとの比較

- Firebase などより SSR に強く SEO最適
- AWS より低コスト + エッジ高速
- Stream により動画配信が圧倒的に楽

---

17. 開発体制

下記は開発状況によって変動します。

- フロント：1名（中塚）
- API：1名（中塚）
- インフラ：1名（中塚）
- デザイン：1名（中塚の代理）
- QA：1名（中塚、兼任可）

---

18. 開発スケジュール

未定

---

19. 開発環境

- ローカル：Node.js / Miniflare / Wrangler
- CI/CD：GitHub Actions → Cloudflare Pages & Workers へデプロイ

---

20. 開発ターゲット

- iOS 15+
- Android 15+
- Web（Chrome, Safari, Edge）

---

21. 開発環境構成（バージョン）

ツール	バージョン例
Node.js	20.x
Next.js	14.x
TypeScript	5.x
Wrangler	最新
Cloudflare D1	beta
Apollo Client	3.x
Apollo Server（Workers対応）	v4
Capacitor	6.x

★補足

【 Stripe 決済およびコイン課金システム】

1. Stripe 決済概要
   本システムではアプリ内で利用するコインおよび動画の購入に関し、Stripe を利用したオンライン決済を採用する。Stripe は WebView からの安全な決済フローをサポートしており、Cloudflare Workers との親和性が高い。
   決済処理は全て「サーバー側（Workers側）」で完結し、クライアント側にクレジットカード情報を保持しない。

2. Stripe 導入目的
   ・決済の透明性と安全性を確保する

・PCI DSS 対応のためアプリ側でカード情報を一切保持しない
・会員登録者の年齢制限（18歳以上のカード決済）に対して Stripe 側で本人認証（3D セキュア）を実施
・アプリ内課金（IAP）を行わず Web 決済（外部決済）を採用することで手数料削減

3. Stripe 決済プロセス
下記のフローでコイン購入を実現する。

① ユーザーがアプリ内（WebView）から「コイン購入」を選択

② API（GraphQL → Workers）へ決済Intent作成リクエスト

③ Workers が Stripe API を呼び出し PaymentIntent を作成

④ クライアントに返却された client_secret を元に Stripe.js が支払い画面を表示

⑤ Stripe にてカード認証（3D Secure）

⑥ 決済成功後、Workers が Webhook を受信し、コインを付与

⑦ DB（D1）へ、取引履歴・残高反映・課金ログ保存

4. Stripe Webhook
決済完了後のコイン付与処理は Webhook により Workers 側で確実に処理される。
・イベント: checkout.session.completed
・処理内容:
- ユーザーID と注文データを照合
- 付与コイン数の計算
- D1 の user_wallet, coin_transactions に反映
- 重複反映防止のため idempotency key を保存

5. コインの仕様（Stripe 連携）
・コインは Stripe の商品として price_id を複数作成
・価格例: 120円, 320円, 480円, 1000円 など
・コイン残高は D1 の user_wallet にて管理
・取引履歴は coin_transactions に保存
・払い戻しは不可（特定商取引法に依存）
・PayPayポイント交換（換金）は運営が承認した場合にのみ Stripe とは無関係に処理（内部管理）

6. Stripe Customer 管理
ユーザーはアカウント作成時に Stripe Customer と紐づける。
Stripe Customer に以下を保存する。
・メールアドレス
・電話番号（SMS認証済みの場合）
・購入履歴
・決済トラブル防止のための Fraud メタデータ

7. Stripe と Cloudflare Workers の構成
   Workers 内に「stripeService」を配置し、以下を提供する。
   ・createPaymentIntent
   ・createCheckoutSession
   ・verifyWebhookSignature
   ・applyCoinToUser
   ・ログ出力（Cloudflare Logs / Sentry などに送信）