

愛知工業大学情報科学部情報科学科
コンピュータシステム専攻（メディア情報専攻）

令和7年度 卒業制作

メタバース向けアセット検索
Webアプリケーションの制作

PolySeek

2026年2月

制作者 X22004 安藤佑有

指導教員 水野慎士

目 次

| | |
|--------------------------------|-----------|
| 第1章 作品概要 | 1 |
| 1.1 サービス概要 | 1 |
| 1.2 対象ユーザ | 1 |
| 1.3 解決する課題 | 2 |
| 1.3.1 市場の急成長と検索の困難化 | 2 |
| 1.3.2 既存検索機能の限界 | 2 |
| 1.3.3 本作品によるアプローチ | 2 |
| 1.4 制作目的 | 3 |
| 第2章 システム設計 | 4 |
| 2.1 全体アーキテクチャ | 4 |
| 2.1.1 アーキテクチャ選定の背景 | 4 |
| 2.1.2 採用したアーキテクチャ | 4 |
| 2.2 技術スタック | 6 |
| 2.2.1 フロントエンド | 6 |
| 2.2.2 バックエンド | 6 |
| 2.2.3 インフラ・運用 | 6 |
| 2.3 タグシステムのデータモデル設計 | 7 |
| 2.3.1 設計の背景と方針 | 7 |
| 2.3.2 含意関係テーブル | 7 |
| 2.3.3 翻訳テーブル | 8 |
| 2.3.4 タグカテゴリ設計 | 8 |
| 2.3.5 バージョン管理設計 | 9 |
| 第3章 実装詳細 | 11 |
| 3.1 スクレイピング基盤 | 11 |
| 3.1.1 スクレイピングの必要性と課題 | 11 |
| 3.1.2 技術選定 | 11 |
| 3.1.3 レート制限の実装 | 11 |
| 3.1.4 レジューム機能 | 12 |
| 3.1.5 データ正規化 | 13 |
| 3.2 UI と検索機能 | 15 |
| 3.2.1 検索システム | 15 |
| 3.2.2 タグ編集インターフェース | 15 |
| 3.2.3 ユーザ機能 | 17 |
| 3.2.4 オンボーディング機能 | 17 |

| | |
|---------------------|-----------|
| 3.2.5 テストと CI/CD | 17 |
| 第4章 公開と運用 | 18 |
| 4.1 サービスリリース | 18 |
| 4.2 実装済み機能 | 18 |
| 4.2.1 認証・ユーザ機能 | 18 |
| 4.2.2 商品データ機能 | 18 |
| 4.2.3 タグシステム | 19 |
| 4.2.4 検索機能 | 19 |
| 4.2.5 管理機能 | 20 |
| 4.3 利用状況 | 22 |
| 4.3.1 現時点での実績 | 22 |
| 4.3.2 今後の成長に向けて | 22 |
| 4.4 運用対応 | 23 |
| 4.4.1 フィードバックに基づく改善 | 23 |
| 4.4.2 サーバ監視と安定運用 | 23 |
| 第5章 今後の予定 | 24 |
| 5.1 短期的なタスク | 24 |
| 5.1.1 含意関係モデルの活用 | 24 |
| 5.1.2 パフォーマンス最適化 | 24 |
| 5.1.3 プロモーション活動 | 24 |
| 5.2 中期的なタスク | 24 |
| 5.2.1 UI/UX の改善 | 25 |
| 5.2.2 多言語対応の基盤整備 | 25 |
| 5.3 継続的なタスク | 25 |
| 謝辞 | 26 |
| 参考文献 | 27 |

第1章 作品概要

1.1 サービス概要

本作品「PolySeek」は、BOOTH[1]で販売される3Dアバターや衣装等のデジタルアセットを対象とした検索Webサービスである。ユーザは商品URLを入力するだけで、タイトルや画像等の情報をデータベースに自動登録できる。

最大の特徴は、ユーザコミュニティによる自由な「タグ付け」機能である。「青いリボン」「レース生地」といった公式カテゴリにはない詳細な特徴での検索を可能にし、膨大な商品から理想のアセットを効率的に発見できる環境を提供する。



図 1.1: PolySeek トップページ

1.2 対象ユーザ

本作品の対象者はVRChatなどのメタバースプラットフォームで活動するクリエイターや一般ユーザである。特に、BOOTHで3Dアバター、衣装等のデジタルアセットを頻繁に購入する人々を主な対象とする。

1.3 解決する課題

1.3.1 市場の急成長と検索の困難化

VRChat 内で使用できるアバターやアセットの多くは、日本のクリエイタ向けマーケットプレイスであるBOOTH^[1]で取引されている。VRChat のユーザ増加に合わせて BOOTH の利用ユーザも増加し、2024 年度の 3D モデルカテゴリの注文件数は約 402 万件を超えており^[2]。これは前年比で 101% の注文件数の増加である。取引高に関しても 2024 年度は 58 億円、前年比 187% 増という記録的な増加が見られている。

BOOTH は VRChat と提携しており、VRChat カテゴリには多数の商品が出品されている。BOOTH を利用して商品を購入するユーザが増えるにつれ、BOOTH へ VRChat 向けアイテムを出品するクリエイタの数も増加している。

1.3.2 既存検索機能の限界

BOOTH の取引高が増える一方で、3D モデルカテゴリの商品数増加に伴い、ユーザが「欲しい」と思う商品を効率的に見つけ出すことが難しくなってきており、現在の BOOTH では商品に対してタグを付与することができるが、このタグは出品者のみが編集可能である。そのため、出品者によって商品に付けるタグの個数にはばらつきがあり、「フリル」「テーパード」といったデザインに関するファッショントピックのタグの活用が限定的であったり、「サイバーパンク」「ファンタジー」といった商品が属するジャンルのタグの定義が曖昧な状態となっている。

現在の BOOTH の検索機能はキーワード検索が中心であり、膨大な商品の中から目的のものを探し出すには多くの時間と労力がかかっている。例えば、検索欄に「VRChat」「こたつ」というキーワードを入力した場合、検索結果にはこたつそのもののアセットだけでなく、説明文に「こたつ」という単語が含まれている衣装やアセットも表示されてしまう。さらに、商品の並び順は初期状態で「人気順」となっているが、この人気順の基準が不明瞭であり、既に人気の商品だけが注目されやすいという問題もある。

1.3.3 本作品によるアプローチ

こうした課題を解決するためには、販売者が提供する公式情報だけに頼るのではなく、ユーザが持つ商品知識を検索に活用できる仕組みが必要である。本作品では、BOOTH の商品に対してアプリケーション内で誰でもタグを付与・削除できるシステムを構築し、ユーザコミュニティ主導のデータベースを作成することで、検索体験の向上を図る。

1.4 制作目的

本作品では、以下の三点を目的として開発を行った。

1. ユーザコミュニティによるタグ付けシステムの導入

より実用的で質の高いデジタルアセットのデータベースを構築する。

2. モダンなユーザインターフェースの提供

よく使う検索条件や機能に容易にアクセスできるようにすることで検索体験そのものを改善する。

3. 商品の一元管理機能

ユーザが関心を持った商品や購入した商品をプラットフォーム上で一元的に管理できる機能を提供する。

これらにより、情報収集やアセット管理の効率を高め、上述の課題解決を目指している。

第2章 システム設計

2.1 全体アーキテクチャ

2.1.1 アーキテクチャ選定の背景

本作品のアーキテクチャ設計にあたり、ホスティング方式の選定を行った。Vercel 等の PaaS (Platform as a Service) は、デプロイの簡便さやスケーラビリティというメリットがある。一方で、Serverless Functions の実行時間制限 (Vercel の場合、無料プランで 10 秒) があり、BOOTH からの商品情報スクレイピングのような長時間処理には制約がある。

本作品では Oracle Cloud Infrastructure (OCI) 上の VPS にセルフホスティングする構成を採用した。OCI の Always Free ティアでは、ARM64 アーキテクチャの VM を無料で利用できるため、学生プロジェクトとしてコスト面でも適切であった。

2.1.2 採用したアーキテクチャ

本アプリケーションは、フロントエンドとバックエンドを統合した Next.js 16 の App Router を基盤としている。Next.js の API Routes を活用することで、フロントエンドとバックエンドで型定義を共有でき、開発効率とコードの一貫性が向上する。Server Components によるサーバサイドレンダリングと、API Routes によるバックエンドロジックを一体化することで、開発効率とパフォーマンスの両立を実現している。

インフラ層では、スタードメインで DNS 管理を行い、nginx をリバースプロキシとして配置している。SSL/TLS 証明書は Let's Encrypt から自動取得し、HTTPS 通信を実現している。アプリケーションプロセスは PM2 で管理し、クラッシュ時の自動再起動やログ管理を行っている。データベースには PostgreSQL を採用し、Prisma ORM を介して型安全なデータアクセスを実現している。

デプロイは GitHub Actions を用いた CI/CD パイプラインで自動化しており、テスト実行後に SSH 経由で VPS へデプロイする構成としている。

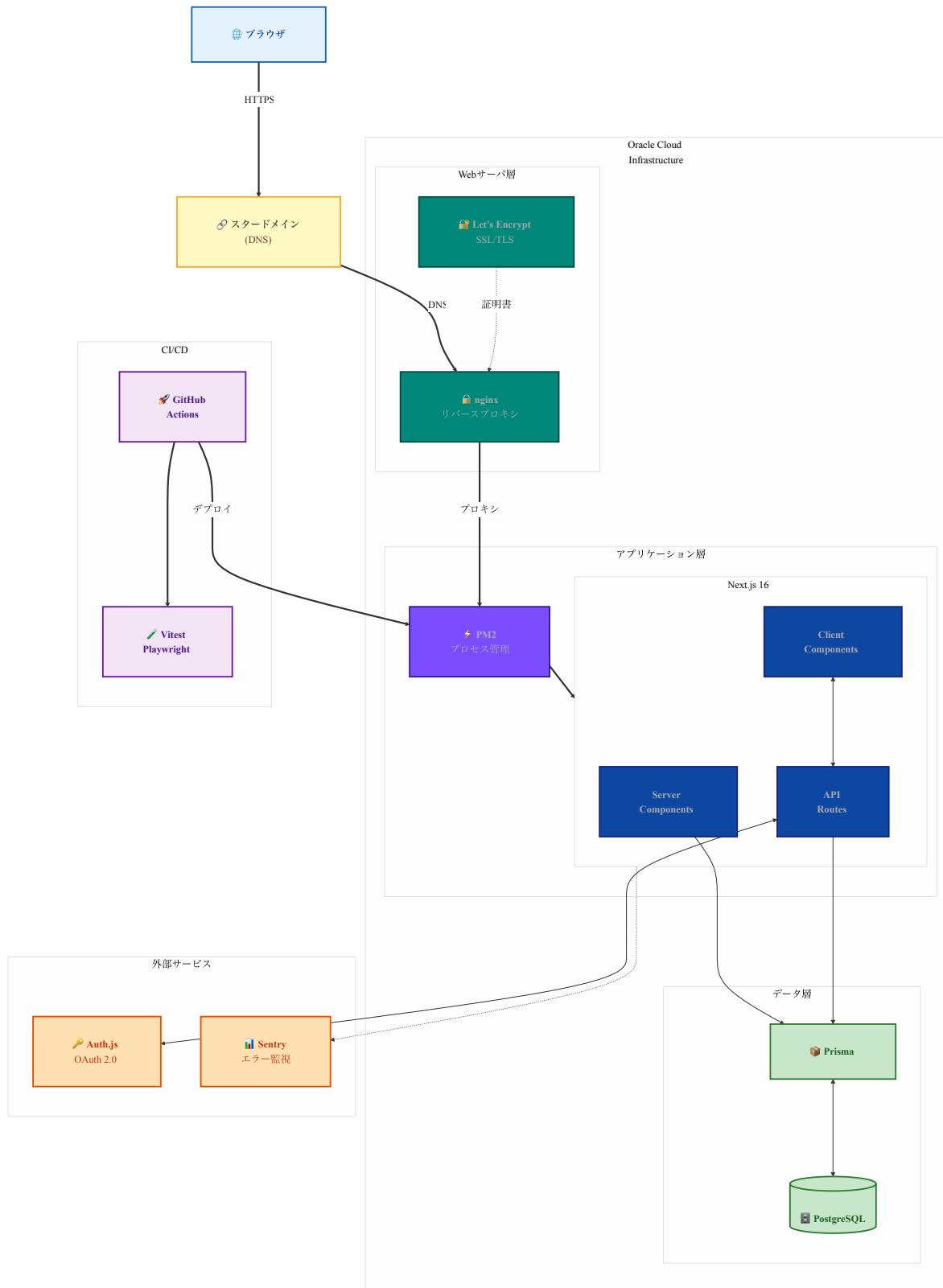


図 2.1: システム全体構成図

2.2 技術スタック

本アプリケーションの開発にあたっては、堅牢性と将来的な拡張性を確保するため、各レイヤーにおいて技術選定を行った。選定にあたっては、開発効率、型安全性、コミュニティの活発さ、ドキュメントの充実度を重視した。

2.2.1 フロントエンド

フロントエンド開発には TypeScript を採用した。静的型付けにより、コンパイル時のエラー検出や IDE による補完機能の恩恵を受けられる。本作品のようなタグシステムでは、タグオブジェクトの構造や API レスポンスの型を厳密に定義することで、ランタイムエラーの防止に貢献した。

UI フレームワークには Next.js 16 の App Router を採用した。Server Components によるレンダリング最適化や、ファイルベースルーティングによる直感的なページ構成が可能である点が特徴である。Vercel が開発元であることから、ドキュメントの充実度やアップデートの継続性についても信頼性が高い。

スタイリングには Tailwind CSS を採用した。ユーティリティファーストのアプローチにより、HTML とスタイルを同一ファイル内で管理でき、コンポーネントの可読性が向上する。アクセシビリティに配慮したコンポーネント基盤として Radix UI を併用し、キーボードナビゲーションやスクリーンリーダー対応を効率的に実装した。初回訪問ユーザ向けのオンボーディングツール機能には Driver.js を採用した。

2.2.2 バックエンド

バックエンドには Next.js API Routes を採用した。フロントエンドとバックエンドで TypeScript の型定義を共有できるため、API レスポンスの型不整合によるバグを防止でき、開発効率が向上する。

データベースには PostgreSQL を採用した。本作品のタグシステムでは、タグ間の含意関係や商品とタグの多対多関係など、リレーションナルな構造が多く存在する。外部キー制約や JOIN クエリを効率的に扱えるリレーションナルデータベースが適している。

ORM には Prisma を採用した。TypeScript との統合が優れており、スキーマ定義から自動生成される型によってデータベースアクセス時の型安全性が確保される。マイグレーション機能も充実しており、スキーマ変更を履歴管理しながら安全に適用できる。

認証システムには Auth.js（旧 NextAuth.js）を採用し、OAuth 2.0 ベースの認証を実現した。Google および Discord を OAuth プロバイダとして設定し、ユーザは既存のアカウントでログインできる。

2.2.3 インフラ・運用

インフラ構成については、図 2.1 で述べた通りである。

テスト環境については、ユニットテストに Vitest、E2E テストに Playwright を採用している。また、本番環境のエラー監視には Sentry を導入し、問題の早期発見と対応を可能としている。

2.3 タグシステムのデータモデル設計

2.3.1 設計の背景と方針

本作品の核心となるタグシステムの設計にあたり、既存のタグベース検索システムを調査した。画像共有サイトにおけるタグシステムでは、単純なタグ付けにとどまらず、タグ間の意味的関係（含意、エイリアス）を管理することで、検索の網羅性と精度を向上させている事例がある。

本作品でも同様のアプローチを採用し、以下の4つのデータモデルを設計した：含意関係テーブル、翻訳テーブル、タグカテゴリ、バージョン管理システムである。これらの設計により、単純なタグ検索を超えた、インテリジェントな検索体験の実現を目指した。

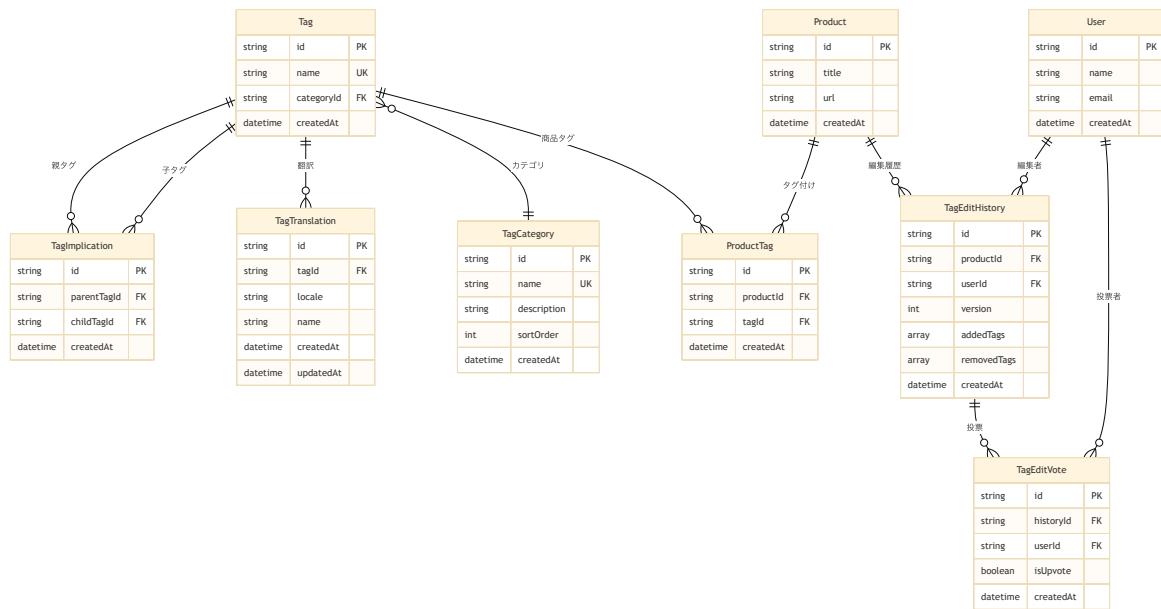


図 2.2: タグシステム ER 図

2.3.2 含意関係テーブル

タグ間の含意関係（親子関係）を表現するテーブルを設計した。これにより、「パークー」というタグを登録した際に、上位タグである「トップス」も自動登録されるようなインテリジェントなタグ付けが可能となる。

```

1 model TagImplication {
2     id             String      @id @default(cuid())
3     implyingTagId String      // 含意するタグ(例:「パンツ」) ID
4     impliedTagId  String      // 含意されるタグ(例:「衣服」または「下着」) ID
5     implyingTag    Tag        @relation("ImplyingTag", fields: [
6         implyingTagId])
7     impliedTag     Tag        @relation("ImpliedTag", fields: [
8         impliedTagId])
9     createdAt      DateTime   @default(now())
10
11    @@unique([implyingTagId, impliedTagId])
12 }
```

リスト 2.1: 含意関係モデル (Prisma Schema)

この設計により、ユーザが細かいタグを付けるだけで、検索時には上位カテゴリでもヒットするようになる。また、タグの一貫性が保たれることでデータベースの品質が向上し、タグ間の複雑な含意ネットワークも表現可能となる。

2.3.3 翻訳テーブル

海外展開を想定し、日本語タグに対する多言語翻訳を格納するテーブルを設計した。

```

1 model TagTranslation {
2     id             String      @id @default(cuid())
3     sourceTagId   String      // 元タグID
4     translatedTagId String    // 翻訳先タグID
5     sourceTag     Tag        @relation("SourceTag", fields: [
6         sourceTagId])
7     translatedTag Tag        @relation("TranslatedTag", fields: [
8         translatedTagId])
9     createdAt      DateTime   @default(now())
10
11    @@unique([sourceTagId, translatedTagId])
12 }
```

リスト 2.2: タグ翻訳モデル (Prisma Schema)

この設計では、単にタグに別言語の名称を持たせるのではなく、異なる言語バージョンのタグどうしをリレーションで結びつける形を採用した。これにより、各言語ごとに独立したタグ管理が可能となり、柔軟な翻訳マッピングが実現できる。

2.3.4 タグカテゴリ設計

タグを「衣服」「スタイル」「素材」などのカテゴリで分類することで、利用者が直感的かつ詳細な絞り込み検索ができるようにした。

```
1 model TagCategory {
2     id          String      @id @default(cuid())
3     name        String      @unique // カテゴリ名
4     color       String      // カテゴリの色例 (: '#FF0000')
5     tags        Tag []
6 }
7
8 model Tag {
9     id          String      @id @default(cuid())
10    name        String      @unique
11    tagCategoryId String?
12    tagCategory  TagCategory? @relation(fields: [tagCategoryId])
13    // ... 他のリレーション
14 }
```

リスト 2.3: タグカテゴリモデル (Prisma Schema)

2.3.5 バージョン管理設計

タグ編集の信頼性を担保するため、Git のようなバージョン管理システムをデータベース上で構築した。タグ編集のたびに追加・削除・維持されたタグの差分とバージョン番号を履歴テーブルに保存し、さらに編集へのコメントやスコア評価を記録可能な設計としている。

```
1 model TagEditHistory {
2     id          String      @id @default(cuid())
3     productId   String
4     product     Product    @relation(fields: [productId])
5     editorId   String
6     editor      User       @relation(fields: [editorId])
7     version     Int        // バージョン番号
8     addedTags  String[]   // 追加されたタグID
9     removedTags String[]  // 削除されたタグID
10    keptTags   String[]   // 維持されたタグID
11    comment    String?    // 編集コメント
12    score      Int        @default(0) // 編集結果に対するスコア
13    createdAt  DateTime   @default(now())
14
15    // 評価機能
16    votes      TagEditVote []
17 }
18
19 model TagEditVote {
20     id          String      @id @default(cuid())
21     historyId  String
22     history    TagEditHistory @relation(fields: [historyId])
23     userId     String
24     user       User        @relation(fields: [userId])
25     score      Int        // 評価スコア (例:) +1/-1
26     createdAt  DateTime   @default(now())
27
28     @@unique([historyId, userId])
29 }
```

リスト 2.4: タグ編集履歴モデル (Prisma Schema)

この設計により、誰がいつ、どのようにタグを編集したかという透明性が確保される。他のユーザは編集履歴に対して賛成・反対の投票を行うことができ、不正な編集があった場合にはロールバックが可能である。これらの仕組みにより、コミュニティ全体で情報品質を維持する体制が構築される。

第3章 実装詳細

3.1 スクレイピング基盤

3.1.1 スクレイピングの必要性と課題

本作品では、ユーザがBOOTHの商品URLを入力することで、商品情報をデータベースに登録できる機能を提供している。この機能を実現するためには、BOOTHの商品ページから必要な情報を自動的に抽出するスクレイピング処理が必要となる。

スクレイピング処理の実装にあたっては、いくつかの課題があった。まず、BOOTHはAPIを公開していないため、HTMLページを解析して情報を抽出する必要がある。また、大量のリクエストを送信すると対象サーバに負荷をかけるため、適切なレート制限が必要である。さらに、長時間の処理中に中断が発生した場合に備え、処理状態の保存と再開機能も求められた。

これらの課題に対応するため、Node.js上で動作する独自のスクレイピングツールを開発した。

3.1.2 技術選定

スクレイピング処理の実装にあたり、Puppeteer等のヘッドレスブラウザを使用する方式と、軽量なHTML解析ライブラリを使用する方式を比較検討した。

ヘッドレスブラウザを使用する場合、JavaScriptによって動的に生成されるコンテンツも取得できるメリットがある。しかし、ブラウザインスタンスの起動に時間がかかり、メモリ消費も大きい。BOOTHの商品ページは、主要な商品情報がHTMLに直接含まれているため、動的コンテンツの取得は必須ではないと判断した。

そのため、軽量なHTML解析ライブラリであるCheerioを採用した。CheerioはjQueryライクなAPIを提供し、DOM操作を直感的に行える点が特徴である。メモリ効率も良く、大量のページを処理する本作品の要件に適している。また、並行処理数の制限にはPromise対応のキュー制御ライブラリであるp-queueを使用し、対象サーバへの負荷を適切に管理している。

3.1.3 レート制限の実装

対象サーバへの負荷を考慮し、リクエスト間隔を制御するレート制限機能を実装した。

```
1 import PQueue from 'p-queue';
2
3 const queue = new PQueue({
4   concurrency: 1,           // 同時実行数
5   interval: 1000,           // インターバル (ミリ秒)
6   intervalCap: 1,           // インターバルあたりの最大実行数
7 });
8
9 async function scrapeProduct(url: string) {
10   return queue.add(async () => {
11     const response = await fetch(url);
12     const html = await response.text();
13     return parseProductHtml(html);
14   });
15 }
```

リスト 3.1: レート制限付きスクレイピングキュー

3.1.4 レジューム機能

大量の商品情報を収集する際、処理中断時に直前の状態から再開できるレジューム機能を実装した。これにより、ネットワークエラーやプロセス停止時にも、収集済みのデータを無駄にすることなく処理を継続できる。

```
1 interface ScrapeState {
2     lastProcessedId: string;
3     processedCount: number;
4     errorCount: number;
5 }
6
7 async function resumableScrape(stateFile: string) {
8     // 前回の状態を読み込み
9     const state = await loadState(stateFile);
10
11    // 未処理のアイテムから再開
12    const items = await getUnprocessedItems(state.lastProcessedId)
13        ;
14
15    for (const item of items) {
16        try {
17            await scrapeProduct(item.url);
18            state.lastProcessedId = item.id;
19            state.processedCount++;
20        } catch (error) {
21            state.errorCount++;
22        }
23        // 定期的に状態を保存
24        await saveState(stateFile, state);
25    }
26}
```

リスト 3.2: レジューム機能の実装例

3.1.5 データ正規化

スクレイピングで取得した生データを、データベースに格納可能な形式に正規化する処理を実装した。商品タイトル、価格、画像 URL、カテゴリなどの情報を抽出し、一貫したスキーマに変換する。

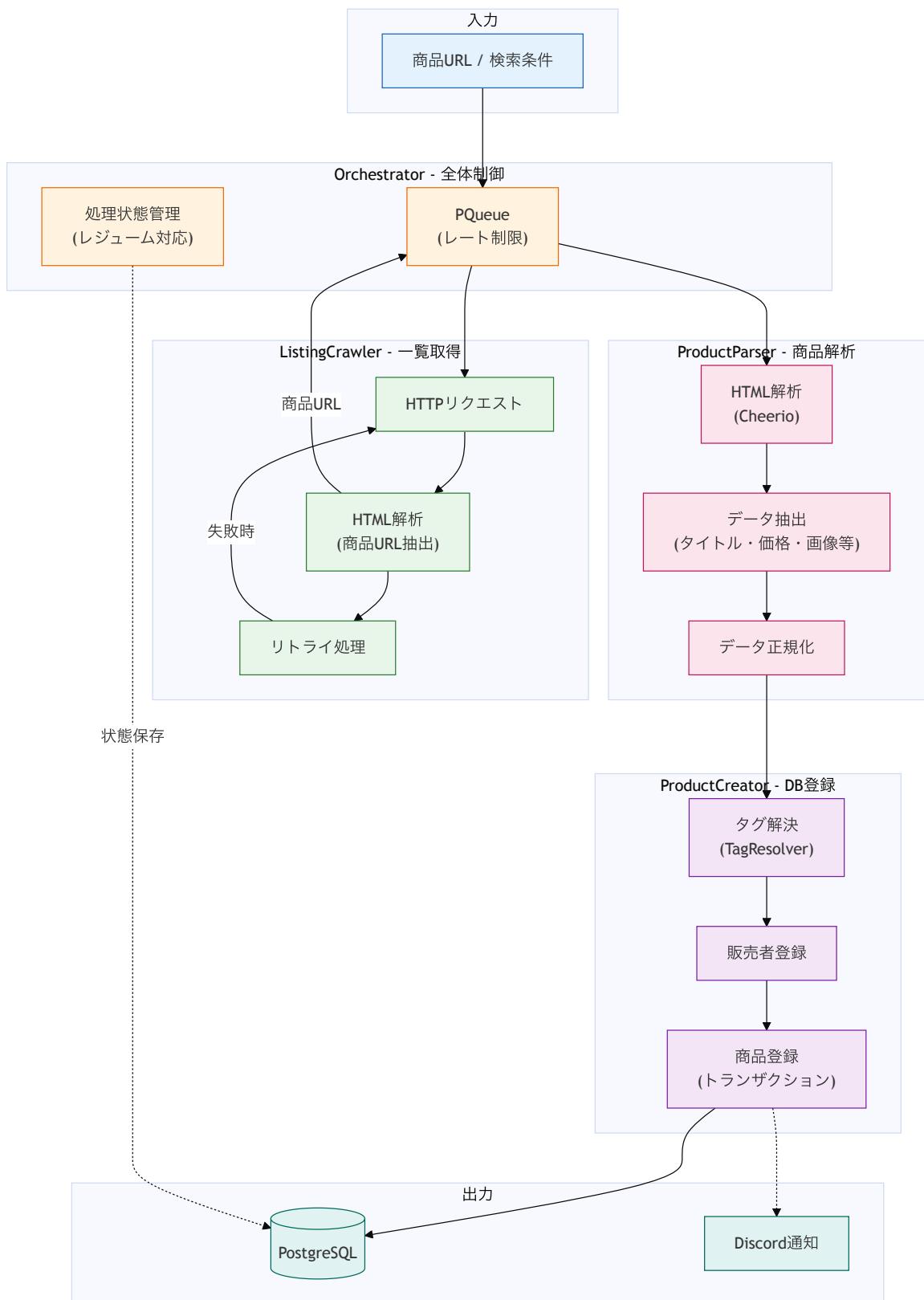


図 3.1: スクレイピング処理フロー

3.2 UI と検索機能

3.2.1 検索システム

検索機能は本作品の中核となる機能であり、ユーザが求める商品を効率的に発見できるよう、複数の検索手法を実装した。

既存のBOOTHの検索機能では、キーワードが商品タイトルや説明文に含まれているかどうかのみで検索が行われる。この方式では、「こたつ」を検索した際に、こたつ本体だけでなく「こたつに合う衣装」といった関連商品も表示されてしまう問題があった。

本作品では、この問題を解決するため、タグベースの検索システムを実装した。ユーザが付与したタグによる絞り込みでは、明確にタグ付けされた商品のみが検索結果に表示される。これにより、キーワード検索の曖昧さを排除し、目的の商品に効率的にたどり着くことが可能となった。

検索バーは、複数の条件を組み合わせるAND検索や、特定の要素を除外するマイナス検索にも対応し、高度なニーズにも応えられるように設計した。マイナス検索の実装にあたっては、検索クエリのペース処理を行い、「-」プレフィックスが付いたキーワードを除外条件として認識する仕組みを構築した。



図 3.2: 検索インターフェース

検索機能は、複数の検索手法を組み合わせて利用できるよう設計されている。キーワード検索では商品タイトルおよび説明文からの全文検索を行い、タグ検索ではユーザが付与したタグによる絞り込みが可能である。また、衣服やアクセサリー等のカテゴリ分類による検索や、価格帯を指定したフィルタリングにも対応している。さらに、これらの条件を組み合わせたAND検索や、特定条件を除外するマイナス検索機能により、高度な検索ニーズにも応えている。

3.2.2 タグ編集インターフェース

タグ編集機能は、本作品のコンセプトである「ユーザコミュニティ主導のデータベース構築」を実現するための重要な機能である。OAuth認証済みのユーザが自由にタグを編集できるインターフェースを実装した。

タグ編集UIの設計にあたっては、入力の効率性と誤操作の防止を両立させることを重視した。タグ入力欄では、オートコンプリート機能により既存のタグを候補として表示し、表記ゆれの防止と入力効率の向上を図っている。また、新しいタグを作成する際には、既存の類似タグを提示し、重複タグの作成を抑制する仕組みを導入した。

編集内容はすべてバージョン管理され、編集履歴として保存される。他のユーザは編集履歴を閲覧し、適切な編集に対しては賛成投票、不適切な編集に対しては反対投票を行うことができる。この投票機能により、コミュニティ全体でタグの品質を維持する体制を構築した。



図 3.3: タグ編集画面



図 3.4: タグ編集履歴と評価機能

3.2.3 ユーザ機能

第2章で述べた OAuth 2.0 認証を基盤として、ユーザ向け機能を実装した。ログイン後のユーザは、商品に対する「いいね」や「所有済み」のマーク機能を利用することができる。マイページでは、これらのマークを付けた商品をリスト形式で一覧管理できる。また、他のユーザのタグ編集履歴に対する投票機能も提供している。

3.2.4 オンボーディング機能

ユーザビリティ向上のため、Driver.js を用いたオンボーディングツール機能を実装した。初めてサービスを訪れたユーザでも主要機能を迷わず利用できるよう配慮している。

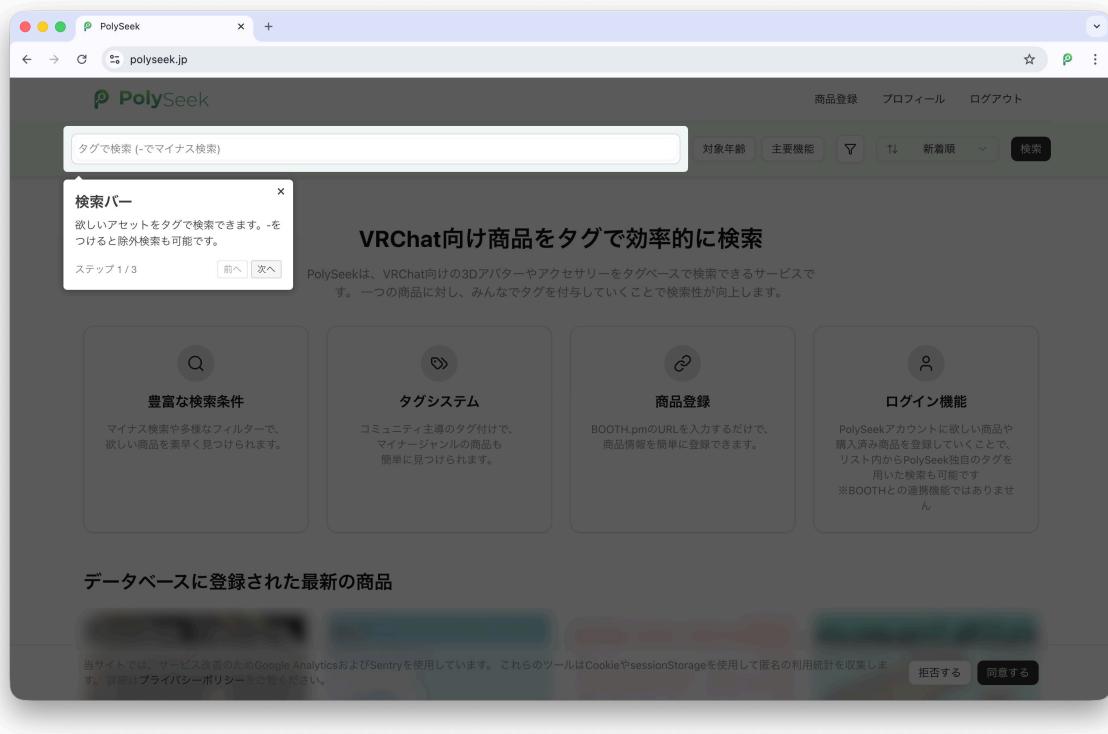


図 3.5: オンボーディングツール

3.2.5 テストと CI/CD

UI コンポーネントの品質を担保するため、第2章で述べた Vitest (ユニットテスト) と Playwright (E2E テスト) によるテスト環境を構築した。GitHub Actions によるテストの自動実行により、コード変更時に品質を維持しながら継続的なデプロイが可能となっている。

第4章 公開と運用

4.1 サービスリリース

2025年12月12日に本作品をリリースし [3]、以後X（旧Twitter）にて継続的な広報活動を行っている。また、Discordサーバを開設し、ユーザからのフィードバックを収集している。

4.2 実装済み機能

リリース時点で、認証・ユーザ機能、商品データ機能、タグシステム、検索機能、管理機能の5つの主要機能が動作している。以下、各機能について詳述する。

4.2.1 認証・ユーザ機能

第2章で述べたOAuth 2.0認証により、GoogleアカウントまたはDiscordアカウントによるログインを提供している。ログイン後のユーザには、プロフィール設定、「いいね」した商品や「所有済み」商品の一覧管理機能が利用可能となる。

4.2.2 商品データ機能

商品データの収集には、BOOTHからの情報スクレイピング機能を実装している。ユーザが商品URLを入力すると、システムは自動的に商品タイトル、価格、画像URL、説明文などの情報を取得し、データベースに登録する。既に登録済みの商品については、最新情報への自動更新も行われる。

商品詳細ページでは、取得した商品情報に加え、ユーザコミュニティによって付与されたタグや、他のユーザからの評価情報を確認できる。

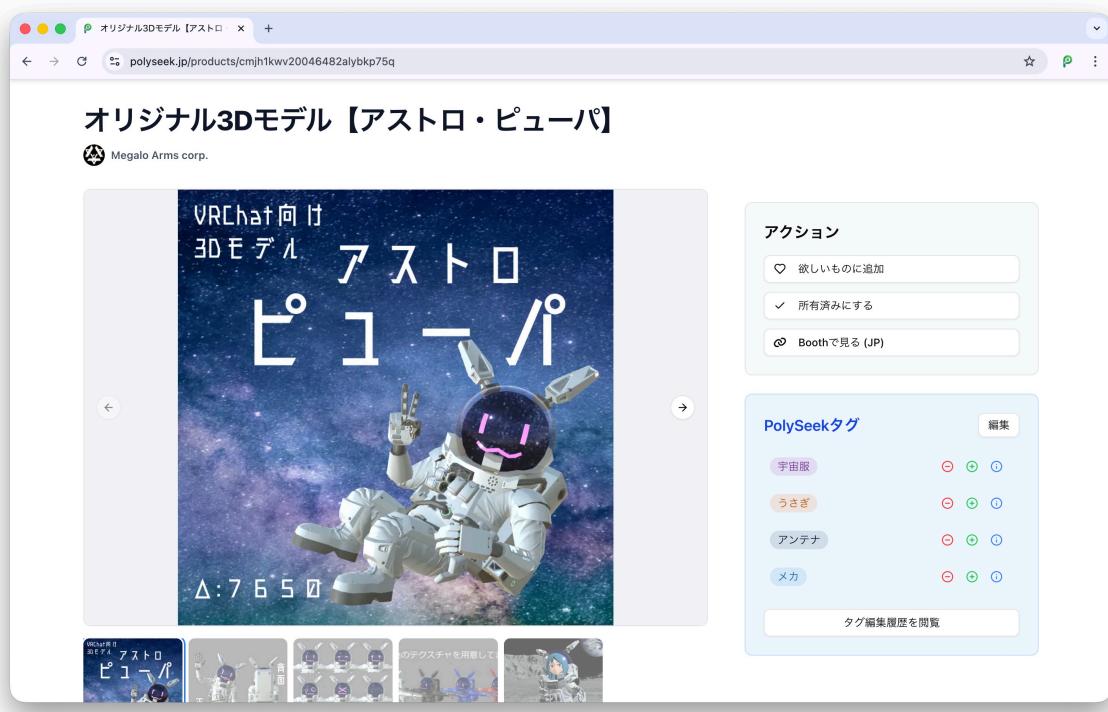


図 4.1: 商品詳細ページ

4.2.3 タグシステム

第2章で設計したタグシステムが稼働している。ログイン済みのユーザは、任意の商品に対してタグを自由に追加・削除でき、すべての編集操作はバージョン管理される。他のユーザは編集履歴を閲覧し、各編集に対して賛成または反対の評価を投票できる。

4.2.4 検索機能

第3章で述べた検索システムにより、キーワード検索、タグ・カテゴリによる絞り込み、価格帯フィルタリング、および複合条件検索が利用可能である。

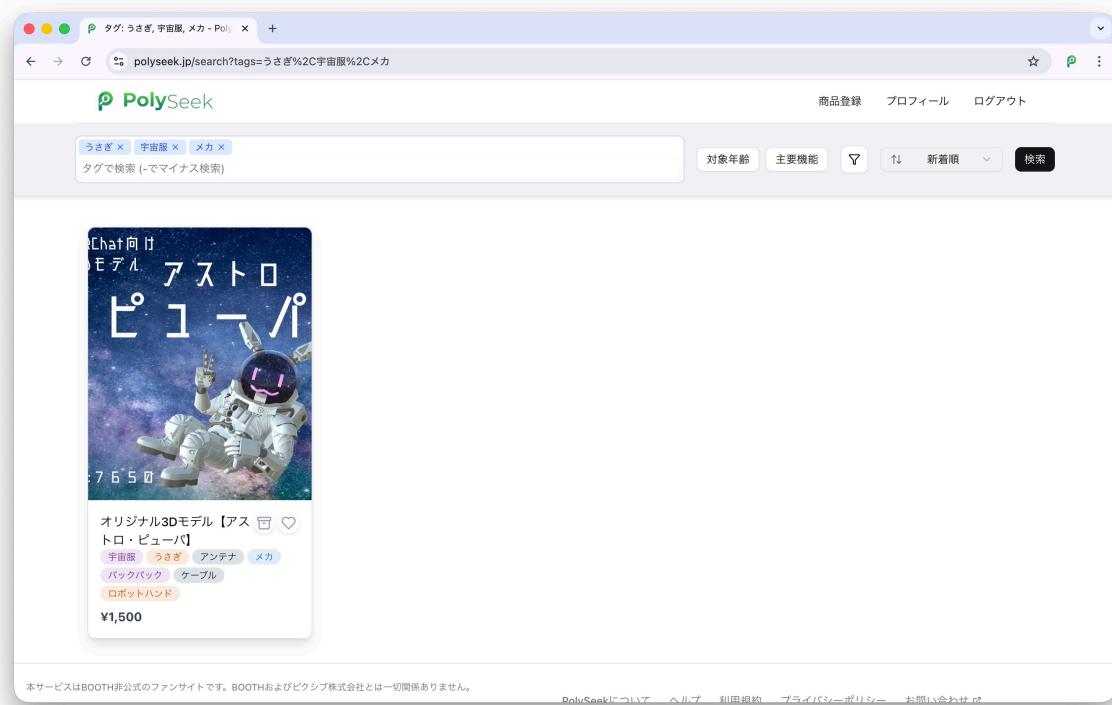


図 4.2: 検索結果ページ

4.2.5 管理機能

管理者向けには、タグ管理画面を提供している。この画面では、タグの統合や削除、カテゴリの設定などの操作が可能である。

The screenshot shows the PolySeek administrator interface with the URL polyseek.jp/admin. The main title is "管理者画面". Below it, the "タグ管理" tab is selected, showing the "タグ一覧" (Tag List) page. The page displays a table of tags with the following columns: 名前 (Name), カテゴリ (Category), 言語 (Language), エイリアス (Alias), 説明 (Description), 使用数 (Usage Count), and 操作 (Operations). The usage count column shows values such as 205446, 104111, 104111, 31372, 28531, 17170, 16016, 15985, 15851, and 14647. Each row has edit and delete buttons.

| 名前 | カテゴリ | 言語 | エイリアス | 説明 | 使用数 | 操作 |
|----------|------------|-----|-------|----|--------|----|
| 全年齢 | age_rating | 日本語 | いいえ | - | 205446 | |
| VRChat | メタ | 日本語 | いいえ | - | 104111 | |
| 3Dモデル | メタ | 日本語 | いいえ | - | 104111 | |
| 3D衣装 | メタ | 日本語 | いいえ | - | 31372 | |
| VRC想定モデル | メタ | 日本語 | いいえ | - | 28531 | |
| 3D装飾品 | メタ | 日本語 | いいえ | - | 17170 | |
| アクセサリー | 衣服 | 日本語 | いいえ | - | 16016 | |
| 衣装 | 衣服 | 日本語 | いいえ | - | 15985 | |
| 3Dテクスチャ | メタ | 日本語 | いいえ | - | 15851 | |
| 3D小道具 | メタ | 日本語 | いいえ | - | 14647 | |

図 4.3: 管理者向けタグ一覧画面

The screenshot shows the PolySeek administrator interface with the URL polyseek.jp/admin/reports. The main title is "管理者画面". Below it, the "通報管理" tab is selected, showing the "通報一覧" (Report List) page. The page displays a table of reports with the following columns: 日時 (Date), 対象 (Target), 理由 (Reason), 通報者 (Reporter), ステータス (Status), and アクション (Actions). There is one report listed:

| 日時 | 対象 | 理由 | 通報者 | ステータス | アクション |
|--------------------|------------|------|--|-------|-------|
| 2026/1/13 14:19:56 | 全年齢 TAG | test | admin polyseek polyseek.dev@gmail.com | 無視 | |

At the bottom of the page, there is a footer note: "本サービスはBOOTH非公式のファンサイトです。BOOTHおよびピクシブ株式会社とは一切関係ありません。" and copyright information: "© 2025 PolySeek".

図 4.4: タグ詳細編集画面

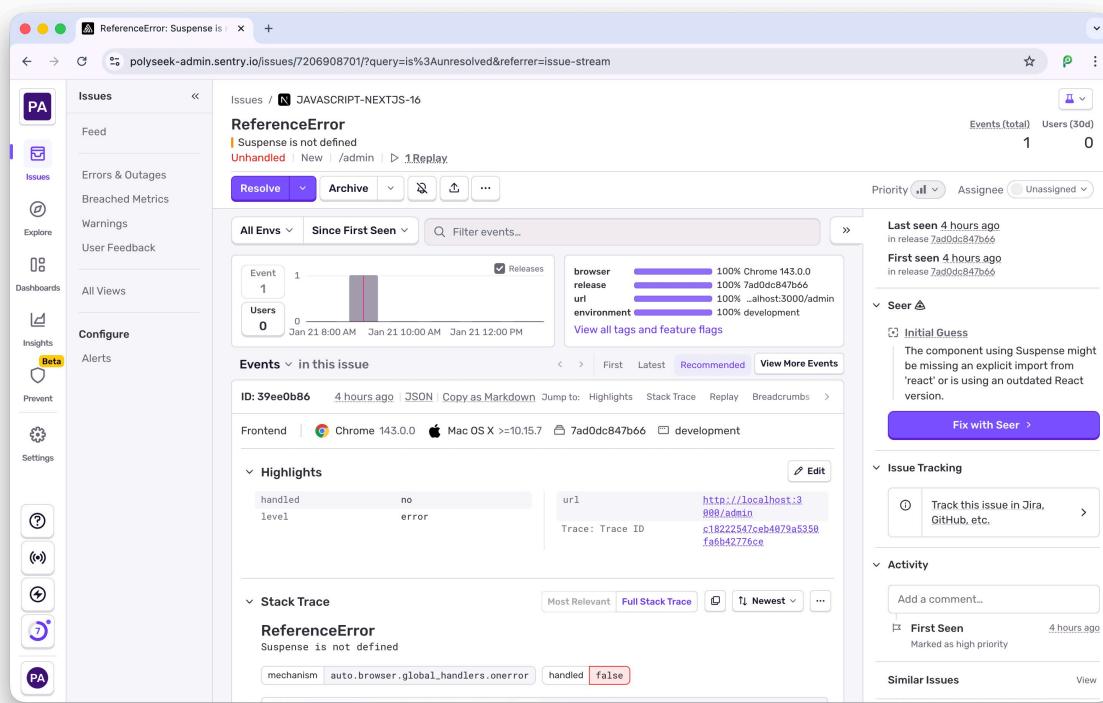


図 4.5: Sentry エラー監視ダッシュボード

4.3 利用状況

4.3.1 現時点での実績

2025年1月12日時点での利用状況について述べる。登録ユーザ数は9人であり、ユーザーによって作成されたタグ数は92個に達している。サービス公開から約1ヶ月という期間を考慮すると、タグデータベースの構築は順調に進んでいると評価できる。

本作品の価値は、ユーザコミュニティによって蓄積されるタグデータにある。現時点では少人数での運用であるが、92個のタグが作成されていることから、アクティブユーザ1人あたり平均10個以上のタグを作成している計算になる。これは、ユーザがタグ付け機能を積極的に活用していることを示している。

4.3.2 今後の成長に向けて

タグベースの検索システムは、タグデータが増えるほど検索の精度と網羅性が向上するネットワーク効果を持つ。現在はまだ初期段階であるが、ユーザ数とタグ数が増加することで、BOOTHの公式検索では実現できない詳細な検索が可能になると期待される。

4.4 運用対応

4.4.1 フィードバックに基づく改善

サービス公開後は、継続的な運用対応を行っている。Discord サーバを通じてユーザからのフィードバックを収集し、バグ報告や機能改善の要望に対応している。

これまでに対応した主な改善点として、検索結果の表示速度の改善や、タグ入力時のオートコンプリート機能の追加がある。これらの改善は、ユーザからの具体的なフィードバックに基づいて優先順位を決定し、実装を行った。

4.4.2 サーバ監視と安定運用

本番環境の安定運用のため、サーバの監視とパフォーマンス最適化にも取り組んでいる。第2章で述べた Sentry によるエラー監視や PM2 によるプロセス管理により、問題の早期発見とダウンタイムの最小化を実現している。

第5章 今後の予定

今後は、より快適にサービスを利用してもらうための機能実装およびプロモーションに注力する。本章では、短期的・中期的・継続的なタスクに分類して、今後の開発計画を述べる。

5.1 短期的なタスク

短期的には、既存の設計を活かした機能拡張と、サービスの認知度向上に取り組む。

5.1.1 含意関係モデルの活用

データベーススキーマで定義済みの含意関係モデルを活用し、よりインテリジェントなタグ登録機能の実装を目指す。具体的には、「パークー」というタグを登録した際に、上位タグである「トップス」も自動登録される機能を実装する。

この機能により、ユーザのタグ付け作業が簡略化されるとともに、検索時のヒット率が向上する。また、データベース内のタグに一貫性が生まれ、検索品質の向上が期待できる。

5.1.2 パフォーマンス最適化

ユーザ数およびデータ量の増加に備え、パフォーマンス最適化に取り組む。データベースエリの効率化やキャッシュ戦略の改善により、応答速度の維持を図る。また、画像配信についてはCDNの活用を検討し、ユーザ体験の向上を目指す。

5.1.3 プロモーション活動

サービスの認知度向上のため、SNSを活用したプロモーション活動を継続する。X（旧Twitter）での定期的な機能紹介や開発状況の報告を通じて、潜在的なユーザへのリーチを図る。また、VRChat関連コミュニティへの広報活動や、既存ユーザからのフィードバック収集も並行して進める。

5.2 中期的なタスク

中期的には、ユーザビリティの向上と国際化対応に注力する。

5.2.1 UI/UX の改善

ユーザフィードバックに基づき、インターフェースの継続的な改善を行う。特に、モバイル端末での利用体験の向上は優先度が高い。また、検索フィルターのUI改善やアクセシビリティの向上にも取り組み、より多くのユーザが快適に利用できる環境を整備する。

5.2.2 多言語対応の基盤整備

開発初期から海外展開を想定し、データベーススキーマにタグ翻訳モデルを用意している。この設計を活かし、日本語タグに対する英語翻訳の提案機能を実装する。また、ユーザの言語設定に応じたUI切り替え機能や、翻訳コミュニティの構築にも取り組み、国際的なユーザベースの拡大を目指す。

5.3 継続的なタスク

サービス運用においては、いくつかの対応を継続的に行う必要がある。商品出品者からの掲載取り下げ依頼への対応は、権利者の意向を尊重する上で重要である。また、ユーザからのフィードバックを反映した機能改善や、スパムや不正なタグ編集を行う悪質なユーザへの対応も欠かせない。

技術面では、セキュリティアップデートの適用やサーバ監視・障害対応を継続し、安定したサービス運用を維持する。

謝辞

本作品の制作にあたり、多くの方々にご支援とご協力をいただきました。

まず、指導教員である水野慎士先生には、制作の方向性から技術的なアドバイスまで、終始丁寧なご指導をいただきました。深く感謝申し上げます。

また、BOOTH商品ページの掲載を快諾いただきましたKAAS様、サンフィッシュくまの様、skkn様、いすい様、まえがみです。様に心より御礼申し上げます。

2026年2月
安藤佑有

参考文献

- [1] pixiv Inc. *BOOTH*. 2025. URL: <https://booth.pm/ja> (visited on 12/26/2025).
- [2] ピクシブ株式会社. *BOOTH 3D モデルカテゴリ取引白書 2025*. 2025. URL: <https://inside.pixiv.blog/2025/02/19/114500> (visited on 12/26/2025).
- [3] 安藤佑有. *PolySeek*. 2025. URL: <https://polyseek.jp> (visited on 12/26/2025).