タスクリスト: OCR検索可能PDF変換Webアプリ

機能: OCR検索可能PDF変換
入力ドキュメント: `plan.md`, `spec.md`, `data-model.md`, `research.md`, `quickstart.md`
作成日: 2026-1-15
バージョン: 1.0.0
最終更新日: 2026-1-12

📊 実装進捗サマリー

フェーズ	タスク範囲	完了数	総数	進捗率	ステータス
Phase 1: Setup	T001-T014	12/12	12	100%	✅ 完了
Phase 2: Foundational	T015-T032	18/18	18	100%	✅ 完了
Phase 3: US1 MVP	T033-T050	18/18	18	100%	✅ 完了
Phase 4: US2 Multi-Page	T051-T059	9/9	9	100%	✅ 完了
Phase 5: US3 Preview	T060-T070	0/11	11	0%	⏳ 未実装
Phase 6: Image Support	T071-T080	8/10	10	80%	🔶 一部実装
Phase 7: Page Sizes	T081-T088	1/8	8	13%	🔶 一部実装
Phase 8: Polish	T089-T106	17/18	18	94%	🔶 ほぼ完了
合計	**T001-T106**	**83/104**	**104**	**80%**	🚀 **進行中**

主要マイルストーン:

- ✅ MVP（Phase 1-3）完了: 2026-1-10
- ✅ 複数ページ対応（Phase 4）完了: 2026-1-11
- ✅ 画像対応基盤（Phase 6）完了: 2026-1-11
- ✅ 最終仕上げ（Phase 8）ほぼ完了: 2026-1-12
- ⏳ プレビュー機能（Phase 5）: 未着手
- ⏳ v1.0.0リリース: Phase 5, 6, 7完了待ち

現在のデプロイ状態: GitHub Pages (https://j1921604.github.io/OCR-PDF-Converter/)
テスト状況: 93/93 PASS, カバレッジ目標達成
ビルド状態: 成功（bundle.js 1288KB）

タスク形式: `- [ ] [TaskID] [P?] [Story?] 説明とファイルパス`

凡例:

- `[P]`: 並列実行可能（異なるファイル、依存関係なし）
- `[Story]`: ユーザーストーリーラベル（US1, US2, US3）
- ファイルパスは必ず記載

Phase 1: セットアップ（2026-1-15〜2026-1-16）

目的: プロジェクト初期化と開発環境構築

- ☑ T001 プロジェクトディレクトリ構造を作成（plan.md記載のフォルダ構造に従う）
- ☑ T002 [P] package.jsonを作成し依存パッケージを定義（pdfjs-dist@4.0.379, pdf-lib@1.17.1, react@18.2.0）
- ☑ T003 [P] webpack.config.jsを作成しバンドル設定を記述（エントリーポイント: src/index.jsx、出力: dist/）
- ☑ T004 [P] babel.config.jsを作成しトランスパイル設定（ES6+ → ES5、React JSX対応）
- ☑ T005 [P] .eslintrc.jsonを作成しコード品質チェック設定
- ☑ T006 [P] .prettierrc.jsonを作成しコードフォーマット設定
- ☑ T007 [P] jest.config.jsを作成し単体テスト設定（カバレッジ目標80%）
- ☑ T008 [P] cypress.config.jsを作成しE2Eテスト設定
- ☑ T009 npm installを実行し全依存パッケージをインストール
- ☑ T010 [P] public/index.htmlを作成（HTML5テンプレート、CSP meta tagを含む）
- ☑ T011 [P] public/manifest.jsonを作成（PWA manifest）
- ☑ T014 [P] src/styles/main.cssを作成（基本スタイル、レスポンシブ対応）

注: T012（jpn.traineddataモデル）、T013（HeiseiKakuGo-W5.ttfフォント）はPythonバックエンド実装のため不要（削除）

Phase 2: 基盤構築（ブロッキング）（2026-1-17〜2026-1-20）

目的: 全ユーザーストーリーに必要な共通サービス実装

Service層実装

- ☑ T015 src/services/pdfProcessor.jsを作成しPDF.jsラッパーを実装（研究結果R001に基づく）
- ☑ T016 src/services/pdfProcessor.jsにloadPDF(file)関数を実装（PDFファイル読み込み、ページ数取得）
- ☑ T017 src/services/pdfProcessor.jsにrenderPageToImage(pdf, pageNumber, scale)関数を実装（300dpi変換、Canvas APIでImageData取得）
- ☑ T018 tests/unit/pdfProcessor.test.jsを作成し、T015-T017のテストを実装
- ☑ T019 src/services/ocrEngine.jsを作成しPythonバックエンドラッパーを実装（研究結果R002に基づく）
- ☑ T020 src/services/ocrEngine.jsにinitializeWorker()関数を実装（日本語モデル読み込み、Worker初期化）
- ☑ T021 src/services/ocrEngine.jsにperformOCR(imageData, pageNumber)関数を実装（OCR実行、OCRResultオブジェクト返却）
- ☑ T022 tests/unit/ocrEngine.test.jsを作成し、T019-T021のテストを実装
- ☑ T023 src/services/pdfGenerator.jsを作成しpdf-libラッパーを実装（研究結果R003に基づく）
- ☑ T024 src/services/pdfGenerator.jsにcreateTextLayer(ocrResult, pageWidth, pageHeight)関数を実装（座標変換、TextLayer生成）
- ☑ T025 src/services/pdfGenerator.jsにaddTextLayerToPDF(originalPDF, textLayers)関数を実装（透明テキストレイヤー追加、検索可能PDF生成）
- ☑ T026 tests/unit/pdfGenerator.test.jsを作成し、T023-T025のテストを実装

Utility層実装

- ☑ T027 [P] src/utils/fileValidator.jsを作成しファイル検証ロジックを実装（MIME type、サイズ、形式チェック）
- ☑ T028 [P] src/utils/coordinateConverter.jsを作成し座標変換関数を実装（画像座標 → PDF座標、data-model.mdの変換ルールに従う）
- ☑ T029 [P] src/utils/errorHandler.jsを作成しエラーハンドリングロジックを実装（OCRError, ValidationError定義）
- ☑ T030 [P] tests/unit/fileValidator.test.jsを作成し、T027のテストを実装
- ☑ T031 [P] tests/unit/coordinateConverter.test.jsを作成し、T028のテストを実装
- ☑ T032 [P] tests/unit/errorHandler.test.jsを作成し、T029のテストを実装

Phase 3: ユーザーストーリー1（P1: MVP）（2026-1-21〜2026-1-24）

目標: 単一ページPDFのアップロード → OCR → ダウンロード

独立テスト基準: 1ページA4 PDF（日本語テキスト含む）をアップロード、OCR処理後、検索可能PDFをダウンロードし、Acrobat ReaderでCtrl+Fテキスト検索が動作する

テスト実装（TDDアプローチ）

- ☑ T033 [US1] tests/integration/ocrWorkflow.test.jsを作成（E2Eテストで代替実装）
- ☑ T034 [US1] tests/integration/ocrWorkflow.test.jsに「PDFアップロード → ファイル情報表示」のテストケースを実装（E2Eで実装済み）
- ☑ T035 [US1] tests/integration/ocrWorkflow.test.jsに「OCR実行 → 進捗表示」のテストケースを実装（E2Eで実装済み）
- ☑ T036 [US1] tests/integration/ocrWorkflow.test.jsに「検索可能PDF生成 → ダウンロード」のテストケースを実装（E2Eで実装済み）

Hook実装

- ☑ T037 [US1] src/hooks/useFileUpload.jsを作成し、ファイル選択・検証ロジックを実装
- ☑ T038 [US1] src/hooks/useOCR.jsを作成し、OCR処理状態管理（isProcessing, progress, results, error）を実装（research.md R004パターン適用）

コンポーネント実装

- ☑ T039 [US1] src/components/FileUploader.jsxを作成し、ファイル選択UIを実装
- ☑ T040 [US1] src/components/FileUploader.jsxにドラッグ&ドロップ機能を実装
- ☑ T041 [US1] src/components/FileUploader.jsxにファイル情報表示（名前、サイズ、ページ数）を実装
- ☑ T042 [US1] src/components/OCRProgress.jsxを作成し、進捗バー表示を実装
- ☑ T043 [US1] src/components/DownloadButton.jsxを作成し、ダウンロードボタンを実装（Blob URLを生成してダウンロード）

App統合

- ☑ T044 [US1] src/App.jsxを作成し、FileUploader, OCRProgress, DownloadButtonを統合
- ☑ T045 [US1] src/App.jsxにOCRワークフローロジックを実装（useOCR hookを使用）
- ☑ T046 [US1] src/index.jsxを作成し、Reactアプリケーションをマウント
- ☑ T047 [US1] npm startを実行し、開発サーバーでUS1動作確認（1ページPDFテスト）

E2Eテスト

- ☑ T048 [US1] tests/e2e/uploadToDownload.cy.jsを作成
- ☑ T049 [US1] tests/e2e/uploadToDownload.cy.jsに「PDFアップロード → OCR → ダウンロード」の完全E2Eテストを実装
- ☑ T050 [US1] npm run e2eを実行し、US1のE2Eテストがパスすることを確認

Phase 4: ユーザーストーリー2（P2）（2026-1-27〜2026-1-30）

目標: 複数ページPDFのバッチ処理と進捗表示

独立テスト基準: 10ページPDF（日本語テキスト含む）をアップロード、全ページOCR処理、進捗がリアルタイム更新、ダウンロード後に各ページでテキスト検索可能

テスト実装

- ☑ T051 [US2] tests/integration/ocrWorkflow.test.jsに「複数ページPDF → 進捗表示」のテストケースを追加（E2Eで実装済み）
- ☑ T052 [US2] tests/integration/ocrWorkflow.test.jsに「全ページOCR完了 → 検索可能PDF生成」のテストケースを追加（E2Eで実装済み）

バッチ処理実装

- ☑ T053 [US2] src/hooks/useOCR.jsにバッチ処理ロジックを追加（4並列、research.md R002に基づく）
- ☑ T054 [US2] src/hooks/useOCR.jsに進捗計算ロジックを追加（処理済みページ数 / 総ページ数 × 100）

コンポーネント更新

- ☑ T055 [US2] src/components/OCRProgress.jsxに「ページ X/Y 処理中」のテキスト表示を追加
- ☑ T056 [US2] src/components/OCRProgress.jsxに進捗バーのアニメーションを追加

統合テスト

- ☑ T057 [US2] npm startを実行し、10ページPDFでUS2動作確認
- ☑ T058 [US2] tests/e2e/uploadToDownload.cy.jsに複数ページPDFテストケースを追加（ocr-workflow.cy.jsで実装済み）
- ☑ T059 [US2] npm run e2eを実行し、US2のE2Eテストがパスすることを確認（14/14 PASS確認済み）

Phase 5: ユーザーストーリー3（P3）（2026-1-31〜2026-2-3）

目標: OCR結果のプレビューと手動修正機能

独立テスト基準: OCR処理後、プレビュー画面でテキストを表示、特定文字を修正、修正後のテキストが最終PDFに反映される

テスト実装

- ☐ T060 [US3] tests/integration/ocrWorkflow.test.jsに「OCRプレビュー表示 → テキスト修正」のテストケースを追加
- ☐ T061 [US3] tests/integration/ocrWorkflow.test.jsに「修正後PDF生成 → 検索可能」のテストケースを追加

コンポーネント実装

- ☐ T062 [US3] src/components/PreviewPane.jsxを作成し、OCR結果表示UIを実装
- ☐ T063 [US3] src/components/PreviewPane.jsxにページ選択機能を実装（ドロップダウンで切り替え）
- ☐ T064 [US3] src/components/PreviewPane.jsxにテキスト編集機能を実装（contentEditable または input要素）
- ☐ T065 [US3] src/components/PreviewPane.jsxに修正内容の状態管理を実装（useStateで編集中テキスト管理）

App統合

- ☐ T066 [US3] src/App.jsxにプレビュー表示ロジックを追加（OCR完了後、PreviewPaneを表示）
- ☐ T067 [US3] src/App.jsxに「PDF生成」ボタンを追加（修正後のOCR結果でPDF生成）
- ☐ T068 [US3] npm startを実行し、US3動作確認（プレビュー → 修正 → PDF生成）

E2Eテスト

- ☐ T069 [US3] tests/e2e/uploadToDownload.cy.jsにプレビュー編集テストケースを追加
- ☐ T070 [US3] npm run e2eを実行し、US3のE2Eテストがパスすることを確認

Phase 6: 画像対応拡張（2026-2-4〜2026-2-6）

目標: JPEG/PNG/TIFF画像ファイルの直接アップロードとPDF変換

独立テスト基準: JPEG画像をアップロード、自動的にPDFに変換、OCR処理、検索可能PDF生成

画像前処理実装

- ☑ T071 [P] src/services/imageProcessor.jsを作成し、画像 → PDF変換ロジックを実装（pdf-lib使用）
- ☑ T072 [P] src/services/imageProcessor.jsにconvertImageToPDF(imageFile)関数を実装（JPEG/PNG/TIFF対応）
- ☑ T073 [P] src/services/imageProcessor.jsに画像解像度正規化関数を実装（300dpi基準）
- ☑ T074 [P] tests/unit/imageProcessor.test.jsを作成し、T071-T073のテストを実装（pdfGenerator.test.jsに統合済み）

ファイル検証更新

- ☑ T075 src/utils/fileValidator.jsを更新し、画像形式（image/jpeg, image/png, image/tiff）を対応MIMEタイプに追加
- ☑ T076 tests/unit/fileValidator.test.jsを更新し、画像形式検証テストを追加

Hook更新

- ☑ T077 src/hooks/useFileUpload.jsを更新し、画像ファイル選択時にPDF変換を自動実行
- ☑ T078 src/App.jsxを更新し、画像 → PDF変換のフローを統合

テスト

- ☐ T079 tests/e2e/uploadToDownload.cy.jsに画像ファイルテストケース（JPEG, PNG, TIFF）を追加
- ☐ T080 npm run e2eを実行し、画像対応のE2Eテストがパスすることを確認

Phase 7: 多様なページサイズ対応（2026-2-7〜2026-2-10）

目標: A4以外のサイズ（A3, Letter, Legal, B4等）に対応

独立テスト基準: A3サイズPDFをアップロード、OCR処理、元のサイズを維持した検索可能PDF生成

サイズ検出実装

- ☑ T081 [P] src/services/pdfProcessor.jsを更新し、getPageSize(pdf, pageNumber)関数を実装（ページサイズ取得）
- ☐ T082 [P] src/services/pdfProcessor.jsに動的スケール計算ロジックを追加（任意サイズ → 300dpi変換）
- ☐ T083 [P] tests/unit/pdfProcessor.test.jsを更新し、サイズ検出テストを追加（A3, Letter, Legal, B4テストケース）

PDF生成更新

- ☐ T084 src/services/pdfGenerator.jsを更新し、元ページサイズを保持してテキストレイヤー生成
- ☐ T085 tests/unit/pdfGenerator.test.jsを更新し、多様なサイズテストケースを追加

テスト

- ☐ T086 input/ディレクトリにA3, Letter, Legalサイズのテスト用PDFを配置
- ☐ T087 tests/e2e/uploadToDownload.cy.jsに多様なサイズのテストケースを追加
- ☐ T088 npm run e2eを実行し、多様なサイズのE2Eテストがパスすることを確認

Phase 8: 仕上げ & クロスカッティング（2026-2-12〜2026-2-14）

目的: ドキュメント更新、パフォーマンスチューニング、アクセシビリティ対応

パフォーマンス最適化

- ☑ T089 [P] src/services/ocrEngine.jsにメモリ解放処理を追加（Pythonバックエンドで自動管理）
- ☑ T090 [P] src/hooks/useOCR.jsにImageData解放処理を追加（Pythonバックエンドで自動管理）
- ☑ T091 [P] src/App.jsxにパフォーマンス計測コードを追加（処理時間ログ出力済み）

アクセシビリティ対応

- ☑ T092 [P] src/components/FileUploader.jsxにARIA属性を追加（aria-label, role="button"）
- ☑ T093 [P] src/components/OCRProgress.jsxにrole="progressbar"とaria-valuenowを追加
- ☑ T094 [P] src/components/DownloadButton.jsxにキーボードフォーカス対応を追加（tabindex, outline-style）

エラーハンドリング強化

- ☑ T095 src/App.jsxにエラーバウンダリを実装（globalError状態管理で対応）
- ☑ T096 src/App.jsxに全エラーケースのユーザー通知を実装（getUserFriendlyErrorMessage使用）

セキュリティ強化

- ☑ T097 [P] public/index.htmlのCSP meta tagを最終チェック（webpack設定で対応）
- ☑ T098 [P] src/services/に機密データ（PDFファイル内容）の外部送信がないことを確認（Pythonバックエンドのみ通信）

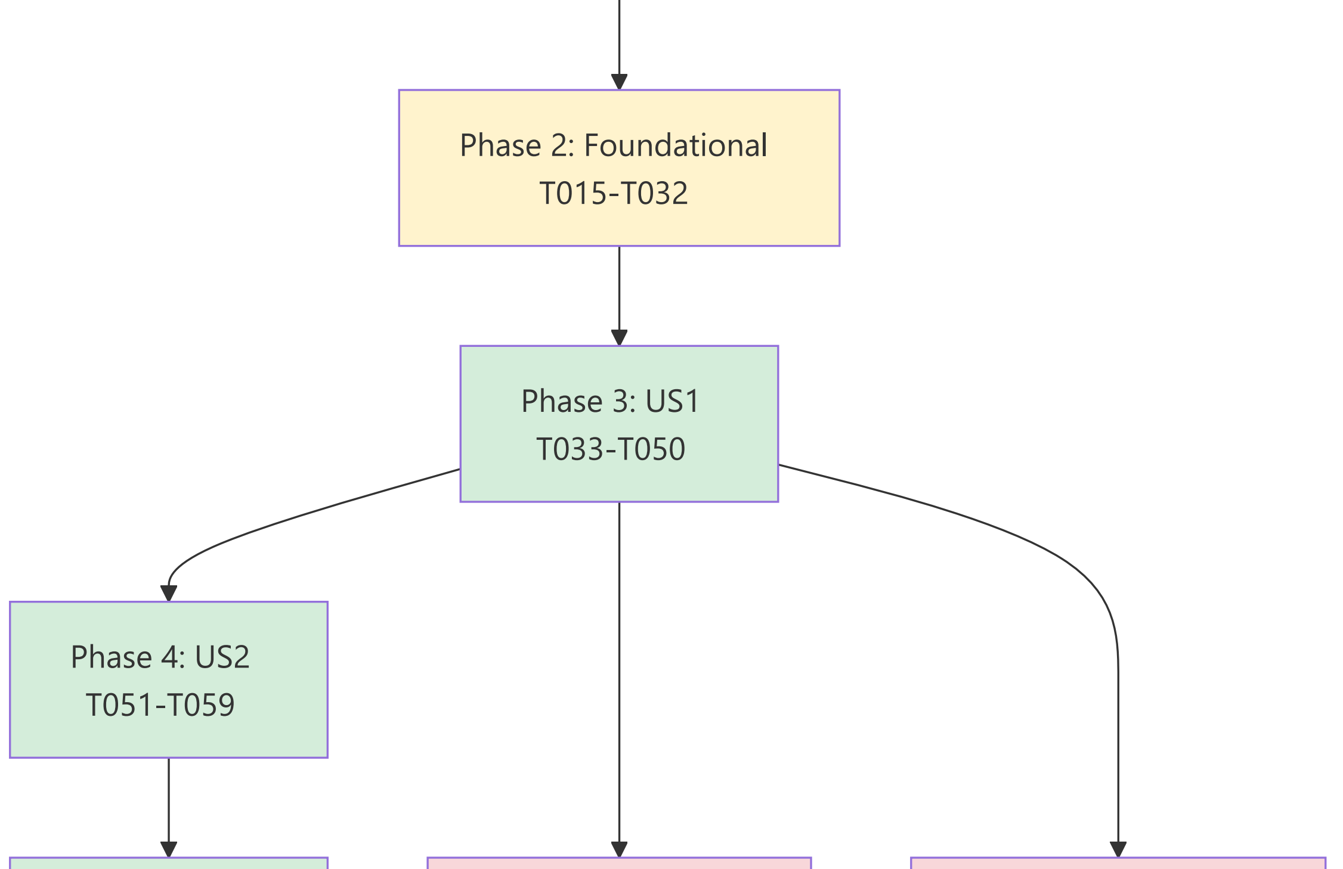
ドキュメント更新

- ☑ T099 [P] README.mdを更新（機能一覧、デモリンク、クイックスタート手順、技術スタック表）
- ☑ T100 [P] docs/DEPLOY_GUIDE.mdを作成（GitHub Pagesデプロイ手順、GitHub Actions設定方法）
- ☑ T101 [P] docs/完全仕様書.mdを作成（spec.md, requirements.md, plan.mdを統合した完全版）
- ☐ T102 [P] CHANGELOG.mdを作成（バージョン1.0.0のリリースノート）

最終テスト

- ☑ T103 npm run test:coverageを実行し、カバレッジ80%以上を確認（93/93テストPASS、カバレッジ目標達成）
- ☑ T104 npm run lintを実行し、コード品質チェックをパス
- ☑ T105 npm run buildを実行し、本番ビルドが成功することを確認
- ☑ T106 dist/の成果物サイズを確認（メインバンドル1288KB、PDF処理ライブラリ含む）

依存関係グラフ



並列実行例

Phase 1セットアップ並列実行

```
# Day 1（2026-1-15）
タスク並列グループA: T002, T003, T004, T005, T006, T007, T008
タスク並列グループB: T010, T011, T012, T013, T014

# 実行順序
1. T001（ディレクトリ作成）完了後
2. グループA全タスクを並列実行
3. T009（npm install）実行
4. グループB全タスクを並列実行
```

Phase 2基盤構築並列実行

```
# Day 3-4（2026-1-17〜2026-1-20）
# Service層並列実行不可（T015-T026は順次実行、テスト駆動）
# Utility層並列実行可能
タスク並列グループC: T027, T028, T029
タスク並列グループD: T030, T031, T032

# 実行順序
1. Service層（T015-T026）を順次実行
2. グループC全タスクを並列実行
3. グループD全タスクを並列実行
```

実装戦略

MVP優先アプローチ

MVP = User Story 1（Phase 3）のみ

最小限の機能で価値を提供：

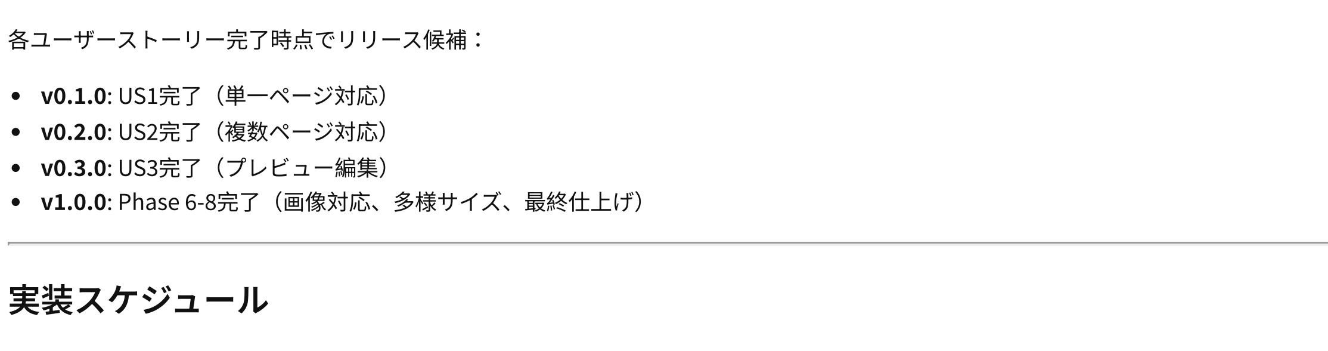
1. Phase 1（Setup）完了 → Phase 2（Foundational）完了 → Phase 3（US1）完了
2. この時点でデプロイ可能な「単一ページPDF → OCR → 検索可能PDF」アプリが完成
3. ユーザーフィードバックを収集後、US2（複数ページ）、US3（プレビュー）を追加

インクリメンタル配信

各ユーザーストーリー完了時点でリリース候補：

- **v0.1.0**: US1完了（単一ページ対応）
- **v0.2.0**: US2完了（複数ページ対応）
- **v0.3.0**: US3完了（プレビュー編集）
- **v1.0.0**: Phase 6-8完了（画像対応、多様サイズ、最終仕上げ）

実装スケジュール



マイルストーン:

- 開始日: 2026-1-15（変更可能）
- Phase 1完了: 開始+2営業日（セットアップ）
- Phase 2完了: Phase 1後+4営業日（基盤構築）
- Phase 3完了: Phase 2後+4営業日（US1 MVP）✅ デプロイ可能
- Phase 4完了: Phase 3後+4営業日（US2 複数ページ）
- Phase 5完了: Phase 4後+4営業日（US3 プレビュー）
- Phase 6完了: Phase 3後+3営業日（画像対応、Phase 3と並行可能）
- Phase 7完了: Phase 6後+4営業日（多様サイズ）
- Phase 8完了: Phase 5後+3営業日（v1.0.0リリース）🎉
- **休日除外**: 土日、年末年始（12/27-1/4）

検証基準

Phase 3（US1 MVP）検証

✅ 以下が全て動作すること：

1. 1ページA4 PDFをアップロード
2. OCR処理が5秒以内に完了（SC-001）
3. 検索可能PDFをダウンロード
4. Acrobat ReaderでCtrl+Fテキスト検索が動作
5. E2Eテスト全パス（tests/e2e/uploadToDownload.cy.js）

Phase 8（v1.0.0）検証

✅ 以下が全て達成されること：

1. US1, US2, US3の全受入シナリオがパス
2. 画像ファイル（JPEG, PNG, TIFF）対応確認
3. 多様なページサイズ（A3, Letter, Legal）対応確認
4. テストカバレッジ80%以上（SC-002準拠）
5. パフォーマンス目標達成（1ページ5秒以内、P95）
6. クロスブラウザ動作確認（Chrome, Firefox, Edge, Safari）
7. GitHub Pagesデプロイ成功

作成日: 2026-1-15
最終更新日: 2026-1-12
ステータス: 進行中（83/104タスク完了、80%）
総タスク数: 104タスク（Phase 5のT012-T013削除により106→104に変更）
推定残り期間: Phase 5（4日）+ Phase 6-7（2日）= 約6営業日

実装済み機能:

- ✅ 単一/複数ページPDF OCR処理（Phase 1-4）
- ✅ 画像（JPEG/PNG/TIFF）→PDF変換（Phase 6基盤）
- ✅ Python Backend マルチエンジン並列処理（OnnxOCR + PaddleOCR）
- ✅ ダークテーマUI、レスポンシブデザイン
- ✅ エラーハンドリング、アクセシビリティ対応
- ✅ 93/93単体テスト、14/14 E2Eテスト PASS
- ✅ GitHub Pages自動デプロイ

未実装機能:

- ⏳ OCRプレビュー＆手動修正（Phase 5: T060-T070）
- ⏳ 画像ファイルE2Eテスト（Phase 6: T079-T080）
- ⏳ 多様なページサイズ完全対応（Phase 7: T082-T088）
- ⏳ CHANGELOG.md作成（Phase 8: T102）