改訂内容

バックエンド主導×AI駆動フロントエンド刷新講座 － カリキュラム改訂分析

はじめに

本文書は、カリキュラム2.0（Version 3.0-2025-05-18）から カリキュラム4.0（Version 4.0）への変更点と、改訂の背景・目的を分析したものです。主要な変更点と、それぞれの変更の意図を整理して説明します。

1. 主要な変更点とその目的

学生らは１年生のときに、HTML/CSS、PHP、PHPでHTML/CSS出力、Word/Excel、企画基礎などを勉強している。２年になったら、WEBアプリやSQL（MySQL）も学び始めているが２ヶ月しか経っていない。WEBアプリは、古典的なサーバーサイドが動的にフロントエンドを出力するタイプのものを学んでおり、フロントエンド・API・バックエンドの３層構造は学んでいない。学力は偏差値40くらいで、英語は英単語はまだなんとかなるが英文は読めない。興味は、ゲーム、アニメ、漫画、アイドルである学生が大半。将来は、プログラマをめざしている。

学校としては、小規模なプログラムでよいので、プログラムをコントロール可能であるという感触をみんなに持たせたい。３層構造だと、HTML/CSS、Javascript/DOM、API/JSON、PHP+SQLと、いろんな技術を串刺しで学ぶ必要がある。なので、これらの役割分担と実装法の根幹を学ばせたい。そのため、それ以外の技術は極力排除し、やむを得ず使う場合は中身の原理は置いておいて、道具として使うと言うふうにしたい。ただ、例外は設計に関する知識とスキルで、その部分は、生成AIでアシストするという方針である。

1.1 Render.comでのデプロイ体験への変更

変更内容:

- 新バージョンでは「**デプロイ体験**」のプラットフォームが変更され、Render.comを使用した公開工程が含まれています
- Day5のカリキュラムにRender.comへのデプロイ演習が組み込まれています
- 評価項目に「Render.comへのデプロイ成功と動作確認」（10%）が追加されています

変更目的:

- Vercelは多機能である分認知負荷が高いこと、バックエンドがPostgresのためMySQLで学んだ学生に認知負荷が高い可能性がある
- Reder.comについては「SupabaseとRender.comについて」を参照のこと
- 学生に「Web上で公開する」という達成感を与えることで、モチベーション向上を図る
- 作ったものが実際にインターネット上で動くという実践的な体験を提供する
- より就職・実務に近い形でのスキル習得を実現する

1.2 時間構成の変更

変更内容:

- 旧: 1日2コマ×各3時間（講30/実120/休30）
- 新: 1日4コマ×各90分（講義15分/演習60分/休憩15分）

変更目的:

- より細かく区切ることで、集中力の継続と疲労軽減を図る
- 各単元ごとの目標を明確化し、小さな成功体験を積み重ねられるようにする
- 講義と演習のサイクルを短くすることで、理解から実践へのフィードバックを効率化する

1.3 技術的な簡略化

変更内容:

- PUT/DELETEメソッドが削除され、GET/POSTに絞られている
- 「Supabase」から「ローカルMySQL」へのDB変更、デプロイ時はJSONファイル対応も許容
- 「docker-compose: nginx+php-fpm+supabase」が「Docker体験(任意・おまじない)」に簡略化

変更目的:

- 学習内容を基本に絞り、技術的なハードルを下げることで、全員が確実に到達できるようにする
- インフラの複雑さを減らし、学生がつまずくポイントを減らす
- Dockerの深い理解を目指すのではなく、「体験」レベルに留めることで、本質的な学習内容に集中できるようにする

1.4 Day4の復習日追加

変更内容:

- Day4全体が「復習Day」として新設され、Day1-3の内容を別テーマで再実践する構成に変更

変更目的:

- 学んだ内容を別のコンテキストで応用することで、知識の定着を図る
- 最初のテーマで理解が不十分だった学生に再チャレンジの機会を提供する
- デプロイ前に学習内容を総復習することで、デプロイ時の成功率を高める

1.5 ディレクトリ構造の簡略化

変更内容:

- APIディレクトリ構造が簡略化されている（handlers/などのサブディレクトリ削除）
- 複雑なディレクトリ構造を避け、フラットな構成になっている

変更目的:

- 初学者にとってわかりやすい、シンプルなディレクトリ構造にすることで学習負荷を軽減
- 基本を押さえた上で、より高度な設計パターンは将来的に学ぶ方針に変更

1.6 より詳細な日程計画

変更内容:

- 各日の時間帯が細分化され、より詳細なタイムテーブルになっている
- 各コマごとの具体的な成果物が明記されている

変更目的:

- 講師・TA・学生全員が進捗を把握しやすくする
- 各時間帯で何をすべきかが明確になり、講座運営の効率化を図る
- 達成すべき具体的な成果物を明示することで、学生のゴールを明確にする

2. 教育的アプローチの変更

2.1 「できた！」体験の重視

変更内容:

- 「学生が「できた！」「Webで動いた！」という強い達成感を得られるよう」という記述の追加
- 技術的な難易度を下げ、確実に完成させられる内容に変更

変更目的:

- 挫折せずに完走できることを最優先し、自己効力感を高める
- 成功体験を通じて、継続的な学習意欲を引き出す
- 難易度の高い内容よりも、基本をしっかり習得することに重点を置く

2.2 「おまじない」アプローチの明示的採用

変更内容:

- 「Docker（おまじないレベル）」「CORS設定（おまじない）」など、理解よりも体験を重視した表現の追加

変更目的:

- 複雑な概念を一度に全て理解させるのではなく、まずは「動く体験」を優先
- 将来的な理解のための「種まき」として、現段階では詳細な理解を求めない
- 学生の心理的負担を軽減し、「難しいことは後回し」にできる安心感を提供

3. 実務へのつながりの強化

3.1 実際のWebに公開する体験の追加

変更内容:

- Render.comを使ったデプロイ体験の追加
- 「作成したアプリケーションをWebに公開する体験」の追加

変更目的:

- 学校内だけでなく、実際のインターネット上でアプリケーションが動く体験を提供
- ポートフォリオとして残せる成果物を作り、就職活動にも役立てられるようにする
- 自分の作品を他者に見せられる喜びを体験させる

3.2 学習環境の柔軟性向上

変更内容:

- 「ローカルDB(MySQL)での開発体験と、デプロイ時のDB代替案（JSONファイル等）も提示し、柔軟性を持たせます。」という方針の追加

変更目的:

- 実際の開発現場でも起こりうる「開発環境と本番環境の違い」に対応する経験を提供

- 環境差異に適応する柔軟なエンジニアリング思考を育成
- 様々な制約の中でも代替案を考える創造力を養成

4. まとめ

カリキュラム2.0から4.0への改訂は、以下の方針に基づいています：

1. **完遂重視**：技術的難易度を適切に調整し、全学生が講座を完走できることを最優先
2. **達成感の向上**：Render.comへのデプロイという具体的なゴールを設定し、Webで公開する喜びを体験
3. **基礎の徹底**：発展的な内容より基礎的な内容を確実に身につける方針への転換
4. **反復学習**：Day4の復習日設定による知識の定着と理解深化
5. **柔軟な対応力**：環境差異への対応など、実務で求められる適応力の基礎を養成

これらの変更は、学生の挫折を防ぎ、IT開発の基礎をしっかりと身につけさせることを目的としています。「動くものを作る」という体験を通じて、継続的な学習意欲を喚起し、将来的により高度な内容へと発展させていくための土台を構築するカリキュラムへと進化しています。

カリキュラム4.0

バックエンド主導×AI駆動フロントエンド刷新講座（Render.comデプロイ対応版）

Version 4.0-2025-05-20

0. ドキュメントメタ

- **タイトル:** v0でUI要件を生成し、純PHP API を組み立て、Render.comにデプロイする5日間集中講座
- **対象:** HTML/CSS/JS & PHP基礎を修了した学習者(専門・大学2年~)
- **形式:** 1日4コマ×各90分（講義15分/演習60分/休憩15分）

1. 目標と到達スキル

カテゴリ	**ゴール**
要件定義	v0 Docs を使い、エンドユーザ向け要件書+画面リストを自動生成できる
UI設計	v0 Draw / First Draft でラフ→ ハイファイ HTML/CSSを出力し `public/` に配置できる。
フロント実装	`app.js` から `Workspace()` でAPI を呼び出し、DOMへ描画する基礎（GET/POST）を理解する。
API 開発	"素のPHP" でフロントコントローラ/ルーターを実装し、基本的なGET/POSTリクエスト（簡単なDB操作含む）を処理できる。DBはローカルMySQLを基本とし、デプロイ時はJSONファイル等での代替も可。
フルスタック連携	静的UIとPHP API を CORS設定（おまじないレベル）で統合し、Postman & ブラウザで動作検証できる。
デプロイ体験	**作成した静的フロントエンドとPHP Render.com にデプロイし、Web上で公開・動作確認できる。**

カテゴリ	ゴール
開発体験	簡単なDockerコマンド（おまじないレベル）で、ローカル開発環境（MySQL等）を起動・停止できることを体験する。

2. ディレクトリスキャフォルド

project-root/

├── public/ ← 静的フロント（v0 生成、Render.com Static Siteとしてデプロイ）

│ ├── index.html

│ ├── styles.css （CSS変数・デザイントークンはシンプルな利用に留める）

│ └── app.js （基本的なDOM操作、Fetch APIによるGET/POST）

├── api/ ← 純PHP API（Render.com Web Serviceとしてデプロイ）

│ ├── .htaccess ← （Render.comでは不要な場合が多い、参考程度）

│ ├── index.php ← フロントコントローラ兼ルーター（シンプルなGET/POST処理）

│ ├── db.php ← （DB接続処理、ローカルMySQL用。デプロイ時はJSONファイルアクセス等に変更も検討）

│ └── items.php ← （API処理ハンドラ、例: mangas.php, games.php）

└── docker-compose.yml （ローカル開発用MySQL起動のため、おまじないとして提供）

3. 5日間タイムテーブル(ワーク 約75%)

Day	コマ	時間帯	内容（講義15分/演習60分）	成果物・チェックリスト
1	1	09:00-10:30	●講座キックオフ&3Wヒアリング	

Day	コマ	時間帯	内容（講義15分/演習60分）	成果物・チェックリスト
	2	10:45-12:15	●v0 Docs で要件書自動生成演習	要件書(md)（非機能要件は簡略化）
	3	13:15-14:45	●v0 Draw で低忠実度ワイヤー作成演習	画面遷移図、ワイヤーフレーム（主要3画面程度）
	4	15:00-16:30	●ディレクトリ雛形作成 & live-server で静的ページ表示（シンプルなヘッダー/メイン/フッター）	public/index.html 雛形
2	1	09:00-10:30	●v0 First Draft → ハイファイ HTML/CSS 出力と確認	ハイファイHTML/CSS（v0出力）
	2	10:45-12:15	●HTML/CSS微調整、アクセシビリティ基礎解説（alt属性、基本的なセマンティックタグ中心）	調整済みHTML/CSS
	3	13:15-14:45	●styles.cssトークン整理・CSS変数導入演習（主要な色・フォント程度）	整理されたCSS（テーマ切り替えなし）
	4	15:00-16:30	●app.jsで**ハードコードJSONデータ**を用いた基本的なDOM操作（テンプレート	JavaScriptによる基本的なDOM操作コード（JSONデータはJS内に記述、検索・

Day	コマ	時間帯	内容（講義15分/演習60分）	成果物・チェックリスト
			リテラルで要素表示）	フィルタリングなし）
3	1	09:00-10:30	●PHP基礎復習（配列、連想配列、JSONエンコード/デコード）、シンプルなルーティング概念	
	2	10:45-12:15	●PHPルーター（api/index.php）実装（GETリクエスト処理、**単純なDB SELECT文またはJSONファイル読込**）	GETリクエストを処理するPHPコード（Controller/Model構造はシンプルに）
	3	13:15-14:45	●Postmanを使ったAPIテスト（GET、ステータスコードと主要キー確認程度）	PostmanでのGETリクエストテスト成功
	4	15:00-16:30	●app.js から Fetch API で GETリクエスト送信、CORS設定（おまじない）、シンプルなDOM表示とエラー確認	フロントからのAPIデータ取得・表示（GET、then().catch()中心、高度なUI/UXなし）
4	1	09:00-10:30	**[復習Day1-2]** 新規テーマでのUI設計（v0 Docs, Draw、ゲーム一覧・登録フォーム程度）	新テーマの簡易要件定義書、ワイヤーフレーム

Day	コマ	時間帯	内容（講義15分/演習60分）	成果物・チェックリスト
	2	10:45-12:15	[復習Day1-2] 新規テーマでのUI実装（v0 First Draft、上記2画面）	新テーマのハイファイHTML/CSS
	3	13:15-14:45	[復習Day3] 新規テーマでのAPI実装（PHP GET一覧 / PHP POST登録、単純なDB INSERTまたはJSON追記）	新テーマのGET/POST API
	4	15:00-16:30	[復習Day3] フロントからのAPI連携（Fetch GET一覧表示 / Fetch POSTフォーム送信）	新テーマのフロントからのGET/POST連携（成功/失敗は alert やコンソール表示）
5	1	09:00-10:30	●Day1-4重要ポイント再確認、Render.comアカウント作成とデプロイ手順概要解説	
	2-3	10:45-14:45	●Render.comへのデプロイ演習（静的サイト & PHP Webサービス）（詳細手順書とTAサポート必須）	デプロイされたWebアプリケーションURL
	4	15:00-16:30	●成果物発表会（各自3-5分LT、デプロイアプリデモ）、全体F	成果物デモ、KPT等での振り返り

Day	コマ	時間帯	内容（講義15分/演習60分）	成果物・チェックリスト
			B、Q&A、クロージング	

4. ワーク必修タスク（Render.comデプロイ対応）

1. **要件定義・UI設計：** v0 Docs, Draw, First Draft を用いて、指定されたテーマのUIを設計し、HTML/CSSを生成・調整する。
2. **API実装(GET/POST)：** PHPで基本的なGETリクエスト（一覧取得、詳細取得）とPOSTリクエスト（新規作成）を処理するAPIエンドポイントを作成する。DB操作は基本的なSELECT/INSERTに限定（ローカルMySQL、デプロイ時はJSONファイル等での代替も可）。
3. **フロントエンド連携(GET/POST)：** JavaScript（Fetch API）を用いて、作成したAPIからデータを取得しHTML上に表示(GET)、フォームからデータをAPIに送信(POST)する。
4. **復習課題：** Day4で、Day1〜Day3の主要な流れ（UI設計→API実装(GET/POST)→フロント連携）を別テーマで一通り実践する。
5. **Render.comへのデプロイ：** 作成した静的フロントエンドとPHPをRender.comにデプロイし、Web上で動作させる。
6. **Docker体験(任意・おまじない)：** 提供された `docker-compose.yml` を使って、ローカルでMySQL環境等が起動・停止することを体験する（詳細な仕組みの理解は問わない）。

5. 成果物&評価

配点	成果物	評価観点
30%	UI/UX（v0生成物 + 調整）	v0生成コードの適切な利用、基本的なHTML/CSSの理解と調整、アクセシビリティへの基本的な配慮
30%	API（PHP GET/POSTリクエスト処理）	ルーティングの基本、リクエストデータの取得、JSON形式での基本的なレスポンス返却、基本的なDB操作(SELECT/INSERT)またはファイル操作

配点	成果物	評価観点
20%	フロントエンド JS（Fetch GET/POST, DOM表示）	Fetch APIの基本的な使い方（GET/POST）、取得データのDOMへの表示、フォームデータの送信、基本的なエラーハンドリング意識
10%	**Render.comへのデプロイ成功と動作確認**	指示通りのデプロイ、Web上での基本的な動作確認
10%	発表・取り組み姿勢	Day4復習課題への取り組み、最終発表での説明の分かりやすさ（デプロイアプリのデモ）、質疑応答
（加点）	Docker体験、より高度なAPI機能への挑戦、DB設計・SQLの工夫など	

6. 使用ツール習熟度

ツール	用途	教員習熟	受講生習熟目安
v0	設計(AI UI 生成)	★★★	★（主要機能の操作）
Cursor / VSCode 等	静的フロント・PHP API編集	★★★	★（基本的なコード編集）
PHP 内蔵サーバー（php -S）	ローカルAPI 簡易起動	★★	★（コマンド実行）
Postman	API テスト（GET/POST）	★★	★（GET/POSTリクエスト送信、簡単なBody設定）
Render.com	**静的サイト & Webサービスデプロイ**	★★	**★（手順書に沿ったデプロイ操作）**
Docker Desktop（任意）	ローカルDB等コンテナ実行体験	★★	（コマンド実行体験のみ）

ツール	用途	教員習熟	受講生習熟目安
MySQL（ローカルDocker）	ローカルDB（基本的なSQL）	★★	★（SELECT/INSERT程度、またはJSONファイルで代替）

7. スタッフロール

役割	人数	ミッション
講師	1	v0 使用法 & PHP API基礎指導、Render.comデプロイ解説、全体ファシリテーション
TA	1-2	v0操作/HTML/CSS/PHP基礎/演習サポート、**デプロイ作業サポート**

8. まとめ（Render.comデプロイ対応）

この5日間講座は「要件→画面→コード→**公開**」Render.comで効率的に体験しつつ、素のPHPによる基本的なAPI構築(GET/POST)とフロントエンド連携の基礎をハンズオンで学びます。静的フロント×シンプルAPIのパターンを通じて、

- UI / API 分離の基本的な考え方
- フロントコントローラとルーティングの初歩
- JavaScriptによる非同期通信(GET/POST)とDOM操作の基礎
- **作成したアプリケーションをWebに公開する体験**

を段階的に習得します。学生が「できた！」「Webで動いた！」という強い達成感を得られるよう、AIツールとPaaSを活用し、扱う技術要素の難易度を調整したカリキュラムです。ローカルDB(MySQL)での開発体験と、デプロイ時のDB代替案（JSONファイル等）も提示し、柔軟性を持たせます。

9. 講座修了後に学生が出来ること

項目	具体スキル
AI活用設計	・ v0 Docs を使った基本的な要件定義書の生成 ・ v0 Draw での簡易ワイヤーフレ

項目	具体スキル
	ーム作成 · v0 First Draft による HTML/CSSコード生成と基本的な調整
フロントエンド開発	· 基本的な HTML/CSS レイアウト実装 · JavaScript での基本的なDOM操作 · Fetch APIを使った基本的なAPI呼び出し（GET/POST）と表示 · 簡単なエラーのコンソール確認
バックエンド開発	· PHPによるシンプルなフロントコントローラとルーティングの実装（GET/POSTリクエスト） · PHPでの基本的なデータ処理とJSONレスポンスの作成（DB操作は簡単なSELECT/INSERT、またはファイル操作） · Postmanを使った基本的なAPIテスト(GET/POST)
デプロイ・公開	**· Render.comを利用して、作成した静的フロントエンドとPHP APIをWeb上にデプロイし、公開できる。**
開発体験・その他	· PHP内蔵サーバーを使ったローカル開発 · 簡単なDockerコマンドによるローカルDB環境起動・停止体験 · フロントエンドとバックエンドの基本的な役割分担の理解

カリ構造

システム集中講座 構造分析

1. 全体の学習フロー

```
graph TD

    A[Day1: 要件定義・UI設計] --> B[Day2: UI実装・DOM操作基礎]

    B --> C[Day3: API実装・フロント連携]

    C --> D[Day4: 復習課題]

    D --> E[Day5: デプロイ・発表]
```

2. 日次・コマ別詳細分析

Day 1: 要件定義・UI設計

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
1	・講座の全体像を理解する ・3Wヒアリングの手法を習得する	・3Wヒアリング ・要件定義の基本概念	・基本的なWebアプリケーションの概念 ・要件定義の基礎知識	・3Wシート ・講座の全体像の理解
2	・v0 Docsを使った要件定義書の作成方法を習得する	・v0 Docs ・要件定義書の構造	・基本的な要件定義の知識 ・Markdownの基礎	・要件定義書(md) ・機能要件リスト
3	・v0 Drawを使ったワイヤーフレームの作成方法を習得する	・v0 Draw ・ワイヤーフレームの基本概念	・UI/UXの基礎知識 ・画面遷移の概念	・画面遷移図 ・ワイヤーフレーム（主要3画面）

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
4	・プロジェクトの基本構造を理解する ・開発環境のセットアップを行う	・ディレクトリ構造 ・live-server ・基本的なHTML構造	・HTML/CSSの基礎 ・ファイルシステムの基本操作	・public/index.html雛形 ・開発環境の準備完了

Day 2: UI実装・DOM操作基礎

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
1	・v0 First Draftを使ったHTML/CSSの生成方法を習得する	・v0 First Draft ・HTML/CSSの基本構造	・HTML/CSSの基礎知識 ・Day1のワイヤーフレーム	・ハイファイHTML/CSS（v0出力）
2	・HTML/CSSの調整方法を習得する ・アクセシビリティの基礎を理解する	・セマンティックHTML ・アクセシビリティ基礎 ・CSS調整手法	・HTML/CSSの基礎知識 ・Day2-1の生成コード	・調整済みHTML/CSS ・アクセシブルな構造
3	・CSSの体系的な管理方法を習得する	・CSS変数 ・デザイントークン ・CSSの構造化	・CSSの基礎知識 ・Day2-2の調整済みコード	・整理されたCSS ・テーマ管理の基礎
4	・JavaScriptによる基本的なDOM操作を習得する	・DOM操作の基礎 ・テンプレートリテラル ・JSONデータの扱い	・JavaScriptの基礎 ・Day2-3のHTML/CSS	・DOM操作コード ・JSONデータ表示

Day 3: API実装・フロント連携

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
1	・PHPの基礎を復習する ・ルーティングの概念を理解する	・PHP配列/連想配列 ・JSONエンコード/デコード ・ルーティング基礎	・PHPの基礎知識 ・JSONの基本概念	・PHP基礎の理解 ・ルーティングの概念理解
2	・PHPによるAPIの実装方法を習得する	・PHPルーター ・GETリクエスト処理 ・DB操作/ファイル操作	・PHPの基礎知識 ・Day3-1のルーティング概念	・GETリクエスト処理コード ・データ取得処理
3	・APIのテスト方法を習得する	・Postman ・APIテストの基礎 ・ステータスコード	・HTTPの基礎知識 ・Day3-2のAPI実装	・Postmanテスト結果 ・API動作確認
4	・フロントエンドとAPIの連携方法を習得する	・Fetch API ・CORS ・エラーハンドリング	・JavaScriptの基礎 ・Day3-3のAPIテスト結果	・フロント-API連携コード ・データ表示実装

Day 4: 復習課題

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
1	・新規テーマでの要件定義・UI設計を実践する	・v0 Docs/Drawの応用 ・要件定義の実践	・Day1の知識 ・v0の基本操作	・新テーマ要件定義書 ・ワイヤーフレーム

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
2	・新規テーマでのUI実装を実践する	・v0 First Draftの応用 ・HTML/CSS実装	・Day2の知識 ・v0の基本操作	・新テーマHTML/CSS ・UI実装
3	・新規テーマでのAPI実装を実践する	・PHP API実装の応用 ・POSTリクエスト処理	・Day3の知識 ・PHPの基礎	・新テーマAPI実装 ・GET/POST処理
4	・新規テーマでのフロント-API連携を実践する	・Fetch APIの応用 ・フォーム送信処理	・Day3の知識 ・JavaScriptの基礎	・新テーマフロント-API連携 ・フォーム実装

Day 5: デプロイ・発表

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
1	・講座の重要ポイントを再確認する ・デプロイの基本概念を理解する	・Render.comの基本概念 ・デプロイの流れ	・Day1-4の知識 ・Gitの基礎	・重要ポイントの理解 ・デプロイの準備
2	・Render.comへのデプロイ方法を習得する	・Render.com ・静的サイトデプロイ ・Webサービスデプロイ	・Day1-4の知識 ・Gitの基礎	・デプロイされたWebアプリ ・動作確認
3	・成果物の発表方法を習得する	・プレゼンテーション	・Day1-4の知識 ・発表の基礎	・デプロイされたWebアプリ

コマ	学習目標	導入する新技術/概念	前提知識	成果物イメージ
				・成果物デモ
4	・振り返りを行う	・KPT振り返り	・Day1-4の知識 ・発表の基礎	・振り返りシート

3. 概念の積み上げ関係

1. 要件定義・UI設計の流れ

3Wヒアリング → 要件定義書(v0 Docs) → ワイヤーフレーム(v0 Draw) → 開発環境構築

2. UI実装の流れ

HTML/CSS生成(v0 First Draft) → アクセシビリティ対応 → CSS整理 → DOM操作基礎

3. API実装の流れ

PHP基礎 → ルーティング実装 → APIテスト → フロント連携

4. 復習課題の流れ

新規テーマ要件定義 → UI実装 → API実装 → フロント連携

5. デプロイ・発表の流れ

重要ポイント確認 → デプロイ実践 → 成果物発表・振り返り

4. 技術スタックの関係性

フロントエンド

├── HTML/CSS (v0生成 + 調整)

└── JavaScript

　　├── DOM操作

　　└── Fetch API (GET/POST)

バックエンド

```
├── PHP
│   ├── ルーティング
│   ├── リクエスト処理
│   └── データ操作
└── データストア
    ├── MySQL（ローカル）
    └── JSONファイル（デプロイ時）
開発・デプロイ
├── v0（AIツール）
│   ├── Docs
│   ├── Draw
│   └── First Draft
├── 開発環境
│   ├── VS Code
│   ├── live-server
│   └── PHP内蔵サーバー
└── デプロイ
    └── Render.com
        ├── 静的サイト
        └── Webサービス
```

演習骨子

Day 1: 要件定義・UI設計

コマ1：講座キックオフと3Wヒアリング

・目標：

- 講座の全体像を理解する
- 3Wヒアリングの手法を習得する
- ユーザーストーリーの形式を理解し、機能要件を記述できる
- チーム開発の基本を理解する
- Render.comでのデプロイを意識したシンプルなプロジェクト構成の概要を理解する

・説明パート1（10-15分）：

- 講座の全体像と最終目標の説明
  - 5日間で作るもの：アニメ/ゲーム情報管理アプリ
  - 最終成果物：Render.comで公開されるWebアプリ
- チーム開発の基本（2人+AIの役割分担）
- 3Wヒアリングの基本概念
  - What（何を作るか）
  - Why（なぜ作るか）
  - Who（誰のためか）
- Render.comデプロイを意識したプロジェクト構成の概要（publicフォルダとapiフォルダ）

・演習1（30分）：

- チーム内での自己紹介（5分）
- 3Wシートの作成（25分）
  - ステップ1（5分）：個人でアイデア出し（具体的なペルソナ2つ以上、主要機能5つ以上）
  - ステップ2（10分）：チームでアイデア統合（既存サービスでは解決できていない課題の特定）
  - ステップ3（10分）：3Wシートの完成

・説明パート2（10-15分）：

- v0の基本概念と使い方
  - v0 Docs：要件定義書生成
  - v0 Draw：ワイヤーフレーム作成

        - v0 First Draft：HTML/CSS生成
    - ユーザーストーリーの基本と書き方
    - 次のコマの準備

・演習2（30分）：

    - ユーザーストーリーの作成（15分）
        - ステップ1（5分）：3Wシートを基に主要機能に対応するユーザーストーリーを5つ以上作成
        - ステップ2（10分）：各ユーザーストーリーに必要な画面要素を洗い出す（v0 Drawでのワイヤーフレーム作成を意識し、画面数は3画面程度に絞る）
    - 3Wシートのブラッシュアップ（15分）
        - ステップ1（5分）：v0 Docsで要件を洗い出し
        - ステップ2（10分）：チームで要件を整理・統合

・完成イメージ：

    - 3Wシート（マークダウン形式）
        - アプリの概要
        - 目的と背景
        - ターゲットユーザー
    - ユーザーストーリー（マークダウン形式）
    - v0の基本操作の理解
    - チーム開発の準備完了

コマ2：要件定義書の作成

・目標：

    - v0 Docsを使った要件定義書の作成方法を習得する
    - 機能要件の整理方法を理解する
    - 非機能要件を簡略化し、主要なものに絞る方針を理解する
    - 作成する要件定義が、Render.comでのデプロイを意識したpublic/apiフォルダ構成に繋がることを意識する

・説明パート1（10-15分）：

    - 要件定義書の基本構造
        - 機能要件と非機能要件（非機能要件は主要なものに絞り簡略化）
        - ユーザーストーリーの書き方
        - 画面一覧の整理方法
    - v0 Docsの詳細な使い方

    - プロンプトの書き方（3Wシート、ユーザーストーリーの活用）
    - 出力結果の調整方法
    - 非機能要件の簡略化と記述方法

・演習1（30分）：

- 3Wシートをもとにv0 Docsで要件定義書の初稿を生成する（15分）
    - ステップ1（5分）：v0 Docsで新規プロジェクト作成とプロンプト準備（3Wシート、ユーザーストーリーを組み込む）
    - ステップ2（10分）：プロンプト実行と生成された初稿の確認、AIとの対話による不足情報の補完
- 生成された機能要件の確認と整理（15分）
    - ステップ1（5分）：v0 Docsが出力した機能要件リストを確認
    - ステップ2（10分）：チームで機能要件を整理・分類（重複削除、粒度調整など）

・説明パート2（10-15分）：

- 画面一覧の整理方法
    - 画面遷移図の基本
    - 画面の優先順位付け
- v0 Docsでの画面一覧生成方法と編集
- 要件定義書の編集と改善ポイント（優先度付け、非機能要件の確認）

・演習2（30分）：

- 生成された要件定義書を編集・改善する（15分）
    - ステップ1（5分）：機能要件に優先度（A/B/C）を付与
    - ステップ2（10分）：非機能要件の確認と簡略化、画面一覧の調整
- 要件定義書の完成（15分）
    - ステップ1（5分）：v0 Docsで最終調整と全体レビュー
    - ステップ2（10分）：マークダウン形式で保存し、チームで共有

・完成イメージ：

- 要件定義書（マークダウン形式）
    - 機能要件リスト（優先度付き）
    - ユーザーストーリー（コマ1の成果物を元にv0が清書・追記したものを想定）
    - 画面一覧
    - 画面遷移図（v0 Docsでテキストベースで生成、または簡易的な図）
    - 非機能要件（簡略版）

コマ3：ワイヤーフレーム作成

・目標：

- v0 Drawを使ったワイヤーフレームの作成方法を習得する
- 画面遷移の概念を理解する
- 主要3画面（トップ、一覧、詳細）に絞ったワイヤーフレームと画面遷移図の作成方法を習得する
- 作成したUIデザインが、最終的にpublicフォルダ内のフロントエンド画面の設計図となることを理解する

・説明パート1（10-15分）：

- ワイヤーフレームの基本概念
    - 低忠実度と高忠実度
    - 画面要素の配置ルール
    - アクセシビリティの考慮点
- v0 Drawの詳細な使い方
    - 基本的な描画ツール
    - コンポーネントの使い方
    - 画面遷移の表現方法
    - 主要3画面（トップ、一覧、詳細）に絞った作成のポイント

・演習1（30分）：

- 主要3画面のワイヤーフレーム作成（トップ、一覧、詳細）（30分）
    - ステップ1（5分）：各画面の目的と主要要素を再確認（要件定義書参照）
    - ステップ2（20分）：v0 Drawで各画面のワイヤーフレームを生成・調整（プロンプト例を参考に、publicフォルダの設計図であることを意識）
    - ステップ3（5分）：生成されたワイヤーフレームの保存

・説明パート2（10-15分）：

- 画面遷移図の作成方法
    - 遷移の表現方法
    - 状態の表現方法
- v0 Drawでの画面遷移図作成（主要3画面にフォーカス）

・演習2（30分）：

- 主要3画面の画面遷移図の作成（15分）
    - ステップ1（5分）：主要3画面間の遷移フローを整理

        - ステップ2（10分）：v0 Drawで画面遷移図を生成・調整（プロンプト例を参考）
    - ワイヤーフレームと画面遷移図の完成（15分）
        - ステップ1（5分）：全体の整合性を確認（ワイヤーフレームと遷移図の対応）
        - ステップ2（10分）：必要な調整とチーム内共有

・完成イメージ：

- ワイヤーフレーム（v0 Draw）
    - 主要3画面（トップ、一覧、詳細）
- 画面遷移図（v0 Draw）
    - 主要3画面間の遷移関係
    - 主要な状態遷移（簡易的）

コマ4：開発環境構築

・目標：

- プロジェクトの基本構造を理解する
- 開発環境のセットアップを行う
- Render.comでのデプロイを意識した、public（フロントエンド）とapi（バックエンド）のシンプルなディレクトリ構造を理解し、作成できる
- v0 First Draft でのUI生成を見越した、非常にシンプルなHTML雛形（ヘッダー、メイン、フッター程度）を作成し、表示確認できる

・説明パート1（10-15分）：

- プロジェクトの基本構造
    - publicとapiディレクトリ構成の意図と役割
    - ファイル命名規則
    - バージョン管理の基本
- 開発環境のセットアップ
    - VS Codeでのディレクトリ作成、基本ファイルの作成（public/index.html, public/css/style.css, public/js/app.js, api/index.php）
    - live-serverの使い方
    - Gitの基本操作

・演習1（30分）：

- プロジェクトフォルダの作成と基本ファイルの配置（15分）

        - ステップ1（5分）：ルートディレクトリ（anime-connect）および public, api ディレクトリを作成
        - ステップ2（10分）：public内にシンプルな index.html（ヘッダー、メイン、フッター構成), 空の css/style.css, js/app.js を作成。api内に動作確認用の index.php を作成。
    - Gitリポジトリの初期化（15分）
        - ステップ1（5分）：git init を実行
        - ステップ2（10分）：作成した基本ファイルをステージングし、最初のコミット（Initial commit）を行う

・説明パート2（10-15分）：

    - HTMLの基本構造
        - セマンティックHTML
        - メタデータの設定
        - v0 First Draftを見越したシンプルな雛形としてのスタイリングとスクリプトの基本
    - live-serverの詳細な使い方（publicディレクトリを指定して起動など）

・演習2（30分）：

    - live-serverでの動作確認（15分）
        - ステップ1（5分）：live-server を public ディレクトリで起動
        - ステップ2（10分）：ブラウザで index.html が表示され、CSS（簡単な背景色など）とJS（コンソールログなど）が適用されていることを確認
    - 動作確認と調整（15分）
        - ステップ1（5分）：live-serverでの確認
        - ステップ2（10分）：必要な調整（HTML雛形の微調整など）

・完成イメージ：

    - プロジェクトの基本構造
        - public/api ディレクトリ構成
        - 基本ファイル（index.html, style.css, app.js, index.php）
    - 動作するHTMLページ
        - 基本的なレイアウト（シンプルな雛形、v0 First Draft適用前）
        - スタイリング（一部適用）
        - JavaScript実行確認（コンソールログ）
    - Gitリポジトリ
        - 初期コミット
        - 基本的な設定

Day 2: UI実装・DOM操作基礎

コマ1：v0 First DraftによるUI生成とDOM操作基礎

・目標：

- v0 First Draft を使用して基本的なHTML/CSS構造を生成できる
- 生成されたHTML/CSSを理解し、微調整ができる
- JavaScriptの基本的なDOM操作（要素の取得と内容の書き換え）を理解できる
- セマンティックHTMLの基本とアクセシビリティ（alt属性など）の重要性を理解する
- シンプルなCSS変数の使い方を理解する

・説明パート1（10-15分）：

- v0 First Draft の概要と使い方
  - プロンプト作成のポイント（テーマ、必須要素、デザインテイストの指示）
  - HTML/CSS生成とコード確認
- 生成されたコードの確認ポイント
  - HTML構造の把握
  - CSSの基本的な当たり方
- セマンティックHTMLとアクセシビリティ
  - header, main, footer, nav タグの適切な使用
  - img タグの alt属性の重要性

・演習1（30分）：v0 First DraftによるUI生成とHTML/CSSの微調整

- ステップ1（10分）：v0 First Draftのプロンプトを作成し、「シンプルな作品紹介ページ」のHTML/CSSを生成（ヘッダー、作品カード複数、フッターを含む）
- ステップ2（10分）：生成された index.html を開き、セマンティックな構造に修正（適切なタグ使用、alt属性追加など）。画像パスを修正し、public/images 等にサンプル画像を配置。
- ステップ3（10分）：生成された style.css を開き、主要な色やフォントサイズをCSS変数として定義し、既存スタイルに適用。

・説明パート2（10-15分）：

- JavaScriptによるDOM操作の基本
  - HTML要素の取得（document.getElementById()）
  - 要素内容の書き換え（innerHTML）
- テンプレートリテラルの活用方法
- JSON形式のデータ構造（JavaScriptの配列とオブジェクト）の基本

・演習2（30分）：JavaScriptによる基本的なDOM操作

- ステップ1（10分）：public/script.js を作成し、作品情報（タイトル、説明、画像URL等）の配列オブジェクトをハードコーディングで準備。
- ステップ2（10分）：index.html のメインコンテンツエリアに、JavaScriptで作品情報を表示するためのコンテナ要素（例： ）を追加し、script.js を読み込む設定を行う。
- ステップ3（10分）：script.js に、準備した作品データをループ処理し、テンプレートリテラルと innerHTML を使って #works-container に各作品情報をHTMLとして描画するコードを記述。live-server で表示確認。

・完成イメージ：

- v0 First Draftで生成・調整されたHTML/CSS（public/index.html, public/css/style.css）
  - セマンティックなHTML構造（適切なタグ使用、alt属性設定）
  - CSS変数を利用したスタイリング
- JavaScriptによるDOM操作でデータが表示されたページ（public/script.js の実行結果を確認）
  - ハードコードされたJSONデータがHTML上に表示される
  - 画像とテキスト情報が正しく紐づいている

コマ2：セマンティックHTMLと基本的なアクセシビリティ

・目標：

- セマンティックHTML（<header>, <main>, <footer>, <nav>, <h1>〜<h6>, <article>など）の重要性を理解し、適切に使用できる
- 画像に対する alt属性の適切な設定方法を理解し、実装できる
- Webアクセシビリティの基本的な考え方を理解する

・説明パート1（10-15分）：

- セマンティックHTMLの重要性
  - マシンリーダブル（検索エンジン、支援技術による解釈の容易性）
  - SEOへの貢献
  - アクセシビリティ向上
- 主要なセマンティックタグの役割と使い方
  - 構造を示すタグ：<header>, <main>, <footer>, <nav>, <aside>
  - コンテンツを区切るタグ：<article>, <section>
  - 見出しタグ：<h1>〜<h6> の階層構造

- と の適切な使い分け、セマンティックタグへの置き換え指針

・演習1（30分）：セマンティックHTMLの適用

- ステップ1（10分）：Day2-1で作成した public/index.html を開き、ページ全体の主要な構造部分（ヘッダー、メインコンテンツ、フッター、ナビゲーションエリア）を、それぞれ <header>, <main>, <footer>, <nav> タグで適切にマークアップする。
- ステップ2（10分）：メインコンテンツ内の各セクションや独立したコンテンツブロック（例：作品カード）を、<section> や <article> タグで囲み、タイトルを見出しタグ（<h1>～<h6>）で階層的に正しく設定する。
- ステップ3（10分）：必要に応じて、リスト構造を持つ部分を <ul>, <ol>, <li> タグでマークアップし直し、全体の構造をセマンティックに改善する。

・説明パート2（10-15分）：

- Webアクセシビリティの基本概念
  - なぜアクセシビリティが重要か（多様なユーザーへの配慮）
  - WCAGなどのガイドライン概要（簡単に触れる程度）
- <img> タグの alt属性の適切な記述方法
  - 画像の内容を的確に伝えるテキスト（良い例、悪い例）
  - 装飾目的の画像の場合の対応（alt=""）
- 簡単なアクセシビリティチェック方法
  - キーボード操作（Tabキーでのフォーカス移動、フォーカスインジケータの確認）
  - CSSを無効にした状態でのコンテンツの可読性確認

・演習2（30分）：画像へのalt属性の追加とアクセシビリティチェック

- ステップ1（10分）：public/index.html 内の静的に配置された全ての <img> タグを確認し、その画像の内容を的確に説明する alt属性を追加する。
- ステップ2（10分）：Day2-1で作成した public/script.js を修正し、JavaScriptで動的に生成している <img> タグにも、各作品の内容に応じた適切な alt属性が設定されるようにする。
- ステップ3（10分）：ブラウザで index.html を表示し、キーボードの Tab キーのみで主要なインタラクティブ要素（リンク、ボタンなど）にアクセスできるか、またフォーカスが視覚的に示されるかを確認。CSSを一時的に無効化し、HTMLの構造だけで情報が伝わるか確認する。

・完成イメージ：

- セマンティックHTMLで構造化された public/index.html

    - 主要構造が <header>, <main>, <footer>, <nav> で正しくマークアップされている
    - コンテンツブロックが <article>, <section> で区切られ、見出し（<h1>-<h6>）が適切に設定されている
- 全ての画像（静的および動的に生成されるもの）に、内容を的確に伝える alt属性が設定されている
- キーボード操作での基本的なアクセシビリティが確保されている

コマ3：CSS変数の活用とデザイントークン基礎

・目標：

- CSS変数の基本的な定義方法（:root）と使用方法（var()関数）を習得する
- 主要な色やフォントサイズ、スペーシングなどをCSS変数で管理するメリットを理解する
- デザイントークンの基本的な考え方と、CSS変数による実現方法を理解する
- CSSの保守性と再利用性を高める意識を持つ

・説明パート1（10-15分）：

- CSS変数の概要とメリット
    - 一元管理による保守性の向上
    - 再利用性によるDRY（Don't Repeat Yourself）の実現
    - 可読性の向上（変数名による意味の明確化）
    - テーマ変更の容易さ
- CSS変数の定義方法
    - :root セレクタ内でのグローバルな定義（--variable-name: value;）
- CSS変数の使用方法
    - var() 関数の使い方（property: var(--variable-name);）
- デザイントークンの概念紹介
    - UIの一貫性を保つための設計値（色、フォント、スペース、ボーダーなど）
    - CSS変数を使ったデザイントークンの具体的な実現例

・演習1（30分）：主要なスタイル値のCSS変数化

- ステップ1（10分）：Day2-2で作成・修正した public/style.css を開き、CSSファイルの先頭の :root セレクタ内に、サイトの主要な色（プライマリーカラー、セカンダリーカラー、テキスト基本色、背景色、カード背景色など）をCSS変数として定義する。
- ステップ2（10分）：同様に :root 内に、主要なフォントサイズ（本文基本サイズ、h1, h2, カードタイトル等の見出しサイズ）や、共通的に使用するスペーシン

グ値（例：--space-small: 8px, --space-medium: 16px）をCSS変数として定義する。
- ステップ3（10分）：style.css 内の既存のスタイルルールで、直接記述されていた色、フォントサイズ、margin/paddingなどの値を、ステップ1・2で定義した適切なCSS変数（var()）を使って置き換える。

・説明パート2（10-15分）：

- CSS変数のカスケーディングとスコープ（ローカル変数の定義も可能であることを簡単に紹介）
- CSS変数を使ったテーマ変更のデモンストレーション
  - :root で定義したテーマカラー変数の値を変更するだけで、サイト全体の基調色が一括で変わる様子を実演
- CSS変数のデバッグ方法
  - ブラウザの開発者ツール（インスペクタ）で、適用されているCSS変数とその実際の値を確認する方法

・演習2（30分）：CSS変数の管理と応用

- ステップ1（10分）：:root で定義したCSS変数のうち、いくつかの色やフォントサイズの値を変更してみて、ブラウザ上でサイト全体のデザインが一括で変更されることを確認し、変数管理の利便性を体験する。
- ステップ2（10分）：（オプション）CSS変数を利用して、特定のUIコンポーネント（例：ボタン、アラートメッセージなど）の簡単なバリエーションを作成してみる。例えば、プライマリーボタンとセカンダリーボタンで背景色を変数で切り替えるなど。
- ステップ3（10分）：（チーム討議・任意）今後プロジェクトでCSS変数を運用していく上で、デザイントークンとしてどのような値を管理すべきか、また変数名の命名規則（例：--color-primary, --font-size-lgなど）やコメントによる分類など、簡単な管理ルールについてチームで話し合ってみる。

・完成イメージ：

- CSS変数が効果的に使用された public/style.css
  - :root に主要な色、フォントサイズ、スペーシング等のデザイントークンがCSS変数として定義されている
  - 各スタイルルール内で var() 関数により変数が広範囲に使用されている
- CSS変数の値を変更することで、サイトの見た目が容易に変更できることを確認済み
- （任意）チームで定めたCSS変数の簡単な命名規則や管理方法に関するメモやドキュメント

コマ4：JavaScriptによるJSONデータの表示とDOM操作応用

・目標：

- JavaScriptを使用してHTML要素を取得し、内容を書き換える基本的なDOM操作を確実に習得する。
- JavaScript内に定義されたシンプルなJSONデータ（配列オブジェクト）をループ処理し、各要素をHTMLに描画できる。
- テンプレートリテラルを使って効率的かつ可読性の高いHTML文字列を生成できる。
- public フォルダに配置されたHTML、CSS、JavaScriptファイルが連携して動作する仕組みを理解する。

・説明パート1（10-15分）：

- これまでの総括とファイル連携の確認：
  - Day2で作成してきた public/index.html（セマンティック構造），public/css/style.css（CSS変数使用），public/script.js の役割分担の再確認。
  - HTMLからCSSとJSファイルを正しく読み込む記述方法のレビュー。
- JavaScriptでのデータ構造：
  - 配列とオブジェクトを組み合わせたJSONライクなデータ（作品リストなど）の定義方法。

・演習1（30分）：作品データの準備と表示コンテナの確認

- ステップ1（5分）：Day2-1で作成し、index.html から読み込んでいる public/script.js ファイルを開く。
- ステップ2（15分）：script.js の先頭に、複数の作品情報（例：id, title, imageUrl, description をキーに持つオブジェクト）を要素とする配列（例：const worksData = [...]）をハードコーディングで定義する（2〜3件程度）。画像URLはプレースホルダーでも可。
- ステップ3（10分）：public/index.html を確認し、メインコンテンツエリア（例：<main> タグ内）に、JavaScriptから作品データを挿入するためのコンテナ要素（例： ）が配置され、一意の id 属性が付与されていることを確認する（なければ追加）。

・説明パート2（10-15分）：

- DOM操作によるデータ表示処理のステップ解説：
  1. データ（配列オブジェクト）を準備する。
  2. HTML内の表示対象コンテナ要素を document.getElementById() で取得する。
  3. HTML文字列を組み立てるための変数を初期化する。

4. データの配列をループ処理（例：forEach メソッド）する。
5. ループ内で、各データ要素を元にテンプレートリテラルを使ってHTML部品（例：作品カード）の文字列を生成する。
6. 生成したHTML部品の文字列を、組み立て用変数に追記していく。
7. ループ終了後、組み立てたHTML文字列全体を、取得したコンテナ要素の innerHTML プロパティに代入する。

- window.onload イベントハンドラの役割：
  - HTML文書全体の読み込みと解析が完了してからJavaScriptの処理を実行させるための基本的な方法。

・演習2（30分）：JavaScriptによる作品データの動的表示

- ステップ1（15分）：script.js 内に、演習1で定義した worksData 配列を引数に取り、説明パート2で解説されたステップに従って、各作品情報を <article class="card">...</article> のようなHTML文字列として生成し、それらを連結して #works-container の innerHTML に設定する関数（例：function displayWorks(data)）を作成する。各作品の画像 <img> タグには、作品タイトルなどを含む適切な alt属性をテンプレートリテラル内で設定する。
- ステップ2（10分）：ページの読み込みが完了したタイミングで（例：window.onload = function() { ... } 内で）、作成した displayWorks 関数に worksData を渡して呼び出すコードを記述する。
- ステップ3（5分）：live-server で public/index.html をブラウザで表示し、worksData で定義した全作品がカード形式で正しく表示されること、各画像の alt属性が設定されていること、ブラウザの開発者コンソールにエラーが出ていないことを確認する。

・完成イメージ：

- public/index.html と public/style.css （Day2-3までの状態をベースとして大きな変更はなし）
- public/script.js に、JavaScriptの配列オブジェクトとしてハードコーディングされた複数件の作品データが定義されている。
- ブラウザで index.html を表示すると、script.js 内の作品データが、JavaScriptのDOM操作によって動的にHTML上に描画されている状態。
  - 各作品情報（画像、タイトル、説明文など）が作品カードとして表示される。
  - 動的に生成された画像にも、適切な alt属性が設定されている。
  - ブラウザの開発者コンソールにエラーが表示されていない。

Day 3: API実装・フロント連携

コマ1：PHP基本復習とAPI準備（ルーティング概念）

・目標：

- PHPの基本的な文法（変数、配列、連想配列、foreach）を再確認する。
- JSON形式のデータをPHPで扱う方法（json_encode, json_decode）を理解する。
- HTTPリクエスト（特に $_SERVER['REQUEST_URI']）の基本を理解し、簡単なルーティングの概念を把握する。
- api フォルダ内でのPHPファイル作成と基本的な動作確認ができる。

・説明パート1（10-15分）：

- PHPの役割と基本：
  - サーバーサイド言語としてのPHPの概要と、Webアプリケーションにおける位置づけ（HTMLを動的に生成、データベース連携など）。
  - PHPスクリプトの基本的な書き方（<?php ... ?>）と実行方法（Webサーバー経由またはCLI）。
- PHPの基本構文（復習）：
  - 変数宣言（$variableName）とデータ型（文字列、数値、ブーリアン、配列、連想配列）。
  - 文字列の連結（.）と変数展開（"Hello, $name!"）。
  - echo または print での出力。
- 配列と連想配列の操作：
  - インデックス配列と連想配列の作成方法。
  - 要素へのアクセス方法（インデックス指定、キー指定）。
  - foreach ループを使った配列・連想配列の反復処理。

・演習1（30分）：PHP基本構文の演習

- ステップ1（10分）：anime-connect/api/ フォルダ内に practice.php というファイルを作成する。PHPの変数を使って自己紹介文（名前、年齢、好きなアニメなど）を作成し、echo で表示するコードを記述する。Webブラウザまたはコマンドライン（php api/practice.php）で実行し、表示を確認する。
- ステップ2（10分）：practice.php に追記。複数のアニメタイトルを格納したインデックス配列を作成し、foreach ループを使って各タイトルをリスト形式（例：- アニメタイトル）で表示する。
- ステップ3（10分）：practice.php に追記。キャラクターの情報（例：名前、登場作品、声優）を格納した連想配列を作成し、各情報（キーと値）を取り出して整形して表示する。

・説明パート2（10-15分）：

- PHPでのJSONデータの取り扱い：
  - `json_encode()` 関数：PHPの配列や連想配列をJSON形式の文字列に変換する方法。日本語文字化け対策のための `JSON_UNESCAPED_UNICODE` オプションの説明。
  - `json_decode()` 関数：JSON形式の文字列をPHPの連想配列またはオブジェクトに変換する方法。第二引数に `true` を指定すると連想配列になることの説明。
- ルーティングの基本概念と簡易実装：
  - ルーティングとは何か（URLのパスに基づいて、実行する処理を振り分ける仕組み）。なぜAPI開発で重要か。
  - `$_SERVER['REQUEST_URI']` 変数を使った、現在のリクエストURI（パス部分）の取得方法。セキュリティ上の注意点にも軽く触れる（直接利用の危険性）。
  - `if/elseif/else` 文と文字列比較（`===`）を使った、URIパスに応じた処理の切り替え（簡易ルーティング）のデモンストレーション。

・演習2（30分）：JSON操作と簡易ルーティング体験

- ステップ1（10分）：`practice.php` に追記。演習1で作成したキャラクター情報の連想配列を `json_encode()` でJSON文字列に変換し表示。次に、与えられたサンプルJSON文字列（例：`'{"id":1,"item_name":"フィギュア","price":5000}'`）を `json_decode()` でPHPの連想配列に変換し、各要素の値を取り出して表示する。
- ステップ2（15分）：Day1-4で作成した `anime-connect/api/index.php` の雛形を修正する。`$_SERVER['REQUEST_URI']` の値を取得し、例えば `/api/index.php/user` にアクセスされた場合はユーザー情報（名前、IDなどを含むダミーの連想配列をJSONエンコードしたもの）を、`/api/index.php/item` にアクセスされた場合は商品情報（商品名、価格などを含むダミーの連想配列をJSONエンコードしたもの）を返すように `if/elseif` 文で処理を分岐させる。デフォルトのパス（上記以外）では、既存の「API is working!」メッセージを返すようにする。PHPの `header('Content-Type: application/json');` が設定されていることを確認。
- ステップ3（5分）：Webブラウザのアドレスバーに `http://localhost/anime-connect/api/index.php/user` や `http://localhost/anime-connect/api/index.php/item`（ポート番号は環境による）のように入力し、アクセスするパスによってブラウザに表示されるJSONの内容が切り替わることを確認する。

・完成イメージ：

- PHPの基本構文（変数、配列、連想配列、`foreach` ループ）の動作を記述・確認した `api/practice.php` ファイル。

- PHPでJSONデータのエンコード・デコード処理を記述・確認した api/practice.php ファイル（またはその一部）。
- api/index.php に、リクエストされたURLのパス（例：/user, /item）に応じて、異なる構造のダミーJSONデータをレスポンスとして返す、ごく簡易的なルーティング処理が実装されている。
- ブラウザから異なるURLパスで api/index.php にアクセスすることにより、返却されるJSONデータの内容が実際に変化することを確認済み。

コマ2：PHPでのGETリクエスト処理API実装とDB連携基礎

・目標：

- PHPでGETリクエストを処理する基本的なAPIエンドポイントを実装できる。
- PDO（PHP Data Objects）を使ったデータベース（MySQL）への接続と、簡単なSELECT文によるデータ取得方法を理解する。
- 取得したデータをJSON形式でレスポンスとして返す方法を習得する。
- APIの基本的な構造（ルーター、コントローラー、モデル）の役割分担を理解する（今回は簡易的に実装）。
- CORS（Cross-Origin Resource Sharing）の基本的な概念と、PHPでのヘッダー設定方法を理解する。

・説明パート1（10-15分）：

- APIとは何か（再確認）：クライアント（例：ブラウザのJavaScript）とサーバー（PHP）がデータを交換するための明確な窓口（インターフェース）。
- RESTful APIの基本原則（HTTPメソッドとリソース）：
  - GETメソッド：リソース（データ）の取得を要求する。
  - リソース指向のURL設計例： /mangas（マンガ一覧), /mangas/1（IDが1のマンガ)。
- PHPにおけるAPIの基本的な構成要素（簡易的なMVC風）：
  - index.php（フロントコントローラー/ルーター）：全てのリクエストを受け付け、適切な処理に振り分ける。
  - Controllerクラス（例: MangaController.php）：リクエストの種類に応じてモデルを呼び出し、ビジネスロジックを実行し、最終的なレスポンス（JSONなど）を生成する。
  - Modelクラス（例: MangaModel.php）：データベースとのやり取り（データの取得、登録、更新、削除）を専門に担当する。
- PDO（PHP Data Objects）の概要：
  - 様々な種類のデータベース（MySQL, PostgreSQLなど）に、統一された方法でアクセスするためのPHP拡張機能。
  - MySQLへの接続文字列、ユーザー名、パスワードを使った接続方法。

    - SQLインジェクション対策としてのプリペアドステートメントの重要性（簡単に触れる）。

・演習1（30分）：データベース準備とPDO接続・データ取得モデル作成

- ステップ1（10分）：Docker環境等でMySQLデータベースサーバーを起動する。指定されたSQL文（`CREATE TABLE mangas ...` およびサンプルデータの `INSERT` 文）を実行し、アプリケーションで使用する `mangas` テーブルを作成し、初期データを投入する。（SQLは別途提供想定）
- ステップ2（10分）：`anime-connect/api/` フォルダ内に `Database.php` を作成する。このファイルに、PDOを使ってMySQLデータベースに接続するための情報（ホスト名、データベース名、ユーザー名、パスワードは各自の環境に合わせて設定）を保持し、`getConnection()` メソッドでPDO接続オブジェクトを返す `Database` クラスを実装する。接続エラー時の例外処理も記述する。
- ステップ3（10分）：`anime-connect/api/` フォルダ内に `MangaModel.php` を作成する。コンストラクタで `Database` クラスをインスタンス化してデータベース接続を取得する。`getAllMangas()` メソッド（`mangas` テーブルの全件をSELECTして返す）と `getMangaById($id)` メソッド（指定されたIDのマンガ情報をSELECTして返す）を実装する。各メソッドはプリペアドステートメントを使用し、結果を連想配列の配列または単一の連想配列として返す。

・説明パート2（10-15分）：

- APIエンドポイントの設計例：
    - `/api/mangas`（GET）：全てのマンガ情報を取得。
    - `/api/mangas/{id}`（GET）：指定したIDのマンガ情報を取得。
- PHPでのGETリクエスト処理フロー：
    1. ルーター（`index.php`）がリクエストURIとHTTPメソッドを解析。
    2. 適切なコントローラーのメソッドを呼び出す。
    3. コントローラーが、必要に応じてリクエストパラメータ（例：URL内のID）を取得。
    4. コントローラーがモデルのメソッドを呼び出してデータ取得を依頼。
    5. モデルがデータベースからデータを取得してコントローラーに返す。
    6. コントローラーが取得したデータを `json_encode()` でJSON文字列に変換。
    7. コントローラーがHTTPヘッダー（`Content-Type: application/json`, ステータスコード）を設定し、JSONレスポンスをクライアントに送信。
- HTTPステータスコードの使い分け：200 (OK), 404 (Not Found), 500 (Internal Server Error)など。
- CORS (Cross-Origin Resource Sharing) の解説：
    - なぜCORSが必要か（異なるオリジンからのリクエスト制限と、それを安全に許可する仕組み）。

- PHPの header() 関数を使った基本的なCORS許可設定（Access-Control-Allow-Origin: *, Access-Control-Allow-Methods, Access-Control-Allow-Headers）。
- プリフライトリクエスト（OPTIONS メソッド）への対応の必要性。

・演習2（30分）：GETリクエスト処理APIコントローラーとルーティング実装

- ステップ1（15分）：anime-connect/api/ フォルダ内に MangaController.php を作成する。コンストラクタで MangaModel をインスタンス化する。リクエストURIとHTTPメソッドに応じて、getAllMangas() または getMangaById() を呼び出し、その結果（またはエラー情報）をJSON形式で出力するメソッド（例：listAll() と getOneById($id)）を実装する。これらのメソッド内では、適切なHTTPステータスコードを設定し、CORS関連のヘッダーと Content-Type: application/json ヘッダーを送信する。
- ステップ2（10分）：anime-connect/api/index.php （Day3-1で簡易ルーティングを試したファイル）を本格的に修正。リクエストURI（$_SERVER['REQUEST_URI']）とリクエストメソッド（$_SERVER['REQUEST_METHOD']）を解析する。/api/mangas というパスでGETリクエストが来た場合は MangaController の listAll() メソッドを呼び出す。/api/mangas/{id} というパス（{id}は数値）でGETリクエストが来た場合は、IDを抽出し MangaController の getOneById($id) メソッドを呼び出すように、ルーティング処理を実装する。該当しないパスの場合は404エラーを返す。
- ステップ3（5分）：WebブラウザまたはPostmanのようなAPIテストツールを使用し、実装したエンドポイント（例：/api/mangas, /api/mangas/1（存在するID), /api/mangas/999（存在しないID））にGETリクエストを送信し、期待されるJSONレスポンスとHTTPステータスコードが返ってくることを確認する。

・完成イメージ：

- 指定された構造の mangas テーブルがMySQLデータベースに作成され、サンプルデータが挿入されている状態。
- api/Database.php: PDOを使用してMySQLデータベースへの接続を管理するクラス。
- api/MangaModel.php: mangas テーブルから全件データおよびID指定で個別データをSELECT文で取得するメソッド（プリペアドステートメント使用）を持つクラス。
- api/MangaController.php: MangaModel を利用して作品データを取得し、結果をJSON形式でHTTPレスポンスとして返すメソッド（HTTPヘッダー、ステータスコード、CORS設定を含む）を持つクラス。
- api/index.php: /api/mangas および /api/mangas/{id} というURLパスへのGETリクエストを適切に解析し、MangaController の対応するメソッドを呼び出すルーティング機能を持つフロントコントローラ。
- 各APIエンドポイント（一覧取得、ID指定取得、存在しないIDでの取得）にリクエストを送信し、それぞれのケースで期待通りのJSONデータまたはエラーメッセージ、そして適切なHTTPステータスコードが返却されることを確認済み。

コマ3：Postmanを使ったAPIテスト（GETリクエスト）

・目標：

- Postmanの基本的な使い方（リクエスト作成、送信、レスポンス確認）を習得する。
- Day3-2で作成したPHP APIのGETエンドポイント（一覧取得、ID指定取得）を手動でテストできる。
- APIテストにおける正常系（期待通りの動作）と異常系（エラーケース）の基本的な考え方を理解する。
- レスポンスのステータスコード（200, 404等）とJSONボディの内容をPostmanで確認・検証する方法を理解する。
- （オプション）PostmanのTestsタブで簡単な自動検証スクリプトを記述できる。

・説明パート1（10-15分）：

- APIテストの重要性と目的：
  - APIが仕様通りに正しく動作することを確認する（品質保証）。
  - バグや意図しない挙動を早期に発見する。
  - APIのドキュメントとしての役割も果たす（リクエスト例、レスポンス例）。
- Postmanの概要と基本インターフェース：
  - API開発・テストのための主要なGUIツール。
  - ワークスペース、コレクション、リクエストの概念。
  - リクエスト作成画面の主要要素：HTTPメソッド選択、リクエストURL入力欄、Paramsタブ、Authorizationタブ、Headersタブ、Bodyタブ。
  - レスポンス表示エリアの主要要素：Bodyビュー（Pretty, Raw, Preview）、Cookiesタブ, Headersタブ, Test Resultsタブ、ステータスコード表示。
- Postmanでのリクエスト送信とレスポンス確認の基本フロー：
  - 新規リクエストを作成し、メソッドとURLを指定。
  - 必要に応じてヘッダーやボディを設定（GETでは主にURLとヘッダー）。
  - 「Send」ボタンでリクエストを送信。
  - 返ってきたレスポンスのステータスコード、ボディ（JSONなど）、ヘッダーを確認する。

・演習1（30分）：Postmanの準備と作品一覧取得API（/api/mangas）のテスト

- ステップ1（10分）：Postmanを起動し、新しいコレクション（例：Manga API DEV Tests）を作成する。Day3-2でローカル環境に構築したPHP APIのベースURL（例：http://localhost/anime-connect/api や http://localhost:8000/api 等、各自の環境に合わせる）を確認する。
- ステップ2（10分）：コレクション内に、作品一覧を取得する /mangas エンドポイントへのGETリクエストを作成する（例：GET {{baseURL}}/mangas）。リクエストを

送信し、返ってきたレスポンスのステータスコード、JSONボディの内容（空の配列またはデータ配列）、Content-Typeヘッダーなどを確認する。

- ステップ3（10分）：（オプション）Postmanの「Tests」タブに、レスポンスのステータスコードが 200 であること、レスポンスのContent-Typeヘッダーが application/json であることなどを検証する簡単なJavaScriptのテストスクリプトを記述し、「Send」後にTest Resultsタブで結果を確認する。

・説明パート2（10-15分）：

- APIテストケース設計の基本：
  - 正常系テスト：APIが期待通りに機能する場合のテスト（例：正しいパラメータでデータを取得できる）。
  - 異常系テスト：APIが予期しない状況や不正な入力に対して、適切にエラーハンドリングできるかのテスト（例：存在しないリソースへのアクセス、不正な形式のID指定）。
- Postmanでのパスパラメータとクエリパラメータの扱い：
  - パスパラメータ（例：/mangas/:id）の指定方法（URLに直接記述）。
  - クエリパラメータ（例：/mangas?sort=title）の指定方法（Paramsタブまたはurlに直接記述）。
- Postmanテストスクリプト入門：
  - pm.test("テスト名", function(){...}); の基本構造。
  - レスポンスオブジェクト（pm.response）の主要プロパティ（pm.response.code, pm.response.json()など）。
  - アサーションライブラリChaiJSの基本的な使い方（pm.expect().to.have.status(), pm.expect().to.be.an('array') など）。

・演習2（30分）：作品詳細取得API（/api/mangas/:id）のテスト（正常系・異常系）

- ステップ1（10分）：正常系テストとして、データベースに存在するマンガのID（例：1）をパスパラメータに指定したGETリクエスト（例：GET {{baseURL}}/mangas/1）をコレクション内に新規作成し送信する。ステータスコードが200 OKであり、レスポンスボディに該当IDのマンガ情報（JSONオブジェクト）が正しく返されることを確認する。（オプション：Testsタブで、返却されたJSONオブジェクトが特定のキー id, title を持つことなどを検証する。）
- ステップ2（10分）：異常系テストとして、データベースに存在しないマンガのID（例：99999）をパスパラメータに指定したGETリクエスト（例：GET {{baseURL}}/mangas/99999）を新規作成し送信する。APIの仕様通りにステータスコードが404 Not Foundであり、レスポンスボディにエラーメッセージ（例：{"message":"Manga not found."}）を含むJSONが返されることを確認する。（オプション：Testsタブでステータスコードとエラーメッセージの内容を検証する。）
- ステップ3（10分）：異常系テストとして、IDとして数値ではない不正な形式の文字列（例：abc）をパスパラメータに指定したGETリクエスト（例：GET {{baseURL}}/m

angas/abc）を新規作成し送信する。Day3-2のAPI実装がどのように応答するか（例：404エラー、または他のエラーステータス）を確認し、その応答がAPIの仕様として適切かどうかを考察・議論する。

・完成イメージ：

- Postman上に Manga API DEV Tests （または類似の名前の）コレクションが作成され、その中に複数のリクエストが保存されている。
- /api/mangas エンドポイントへのGETリクエストが作成され、正常なレスポンス（ステータスコード200、JSON配列、適切なヘッダー）が得られることを手動またはテストスクリプトで確認済み。
- /api/mangas/:id エンドポイントへのGETリクエストが、以下の3つのケースについて作成・テストされている：
    1. 存在するIDを指定：正常なレスポンス（ステータスコード200、該当IDのJSONオブジェクト）の確認済み。
    2. 存在しないIDを指定：エラーレスポンス（ステータスコード404、エラーメッセージJSON）の確認済み。
    3. 不正な形式のIDを指定：APIの応答（ステータスコード、エラーメッセージ等）を確認し、その挙動を把握済み。
- （オプション）いくつかのリクエストには、PostmanのTestsタブに基本的なアサーション（ステータスコードチェック、JSON形式チェック、キー存在チェックなど）を行うテストスクリプトが記述されている。

コマ4：Fetch APIによるフロントエンド連携（GET）

・目標：

- JavaScriptのFetch API（.then().catch()構文）を使用して、Day3-2で作成したPHP API（/api/mangas）からデータを非同期に取得できる。
- 非同期処理の基本的な概念（Promiseベースの処理フロー）を理解する。
- 取得したJSONデータを解析し、JavaScriptを使ってHTMLのDOM要素（例：リスト）に動的に描画できる。
- APIリクエスト中のローディング表示、および通信エラーやAPIエラー発生時の簡単なエラーハンドリングを実装できる。
- publicフォルダ内のHTML/JSと、apiフォルダ内のPHPが連携して動作する一連の流れを体験する。

・説明パート1（10-15分）：

- フロントエンドとバックエンド(API)連携の概要：
    - なぜ連携が必要か：静的なHTMLではなく、動的に変化するデータを表示するWebアプリケーションを実現するため。

    - HTML/CSS/JavaScript（フロントエンド）と PHP/データベース（バックエンドAPI）の役割分担の再確認。
- JavaScriptにおける非同期処理の重要性：
    - 同期処理の問題点：時間のかかる処理（例：APIリクエスト）がUIの描画やユーザー操作をブロックしてしまう。
    - 非同期処理の利点：時間のかかる処理をバックグラウンドで行い、その間もUIは応答性を保つ。
- Fetch APIの基本的な使い方（Promiseベース）：
    - `fetch(apiEndpointUrl)`：APIにリクエストを送信。Promiseオブジェクトを返す。
    - `.then(response => { ... })`（1つ目のthen）：レスポンスオブジェクトを受け取る。`response.ok` でHTTPステータス成功確認。`response.json()` でレスポンスボディをJSONとしてパース（これもPromiseを返す）。
    - `.then(data => { ... })`（2つ目のthen）：パースされた実際のデータ（JavaScriptオブジェクト/配列）を受け取り、処理する。
    - `.catch(error => { ... })`：リクエスト送信時、レスポンス処理時、JSONパース時などに発生したエラーを一括で捕捉する。
- `DOMContentLoaded` イベントリスナーの役割：HTMLドキュメント全体の読み込みと解析が完了してからJavaScriptの処理を実行させるための標準的な方法。

・演習1（30分）：HTML準備とFetch APIでのデータ取得・コンソール表示

- ステップ1（10分）：Day2で作成した `public/index.html` をベースに（または新規に）、マンガ作品一覧を表示するための空の `<ul>` 要素（例: `<ul id="manga-list"></ul>`）をメインコンテンツエリアに配置する。また、ローディングメッセージを表示するための `<div>` 要素（例: `<div id="loading-message" style="display:none;">読み込み中...</div>`）と、エラーメッセージを表示するための `<div>` 要素（例: `<div id="error-message" style="display:none;"></div>`）も近くに配置する。`public/js/app.js`（Day2で作成・編集したファイル）を `index.html` から `defer` 付きで読み込んでいることを確認。
- ステップ2（15分）：`public/js/app.js` の既存の処理を一旦コメントアウトするか、末尾に追記する形で、`DOMContentLoaded` イベントリスナー内に新しい処理を記述する。Day3-2で作成したPHP APIの作品一覧取得エンドポイントのURL（例：`const apiUrl = '/api/mangas';` または `http://localhost:xxxx/api/mangas`）を定義する。`fetch()` を使用してこのURLからデータを非同期に取得し、レスポンスをJSONに変換後、取得したデータ全体をブラウザの開発者コンソールに `console.log()` で出力する。Fetchの `.catch()` を使用して、通信エラーやJSON変換エラーが発生した場合はエラーオブジェクトを `console.error()` で出力する。
- ステップ3（5分）：ローカルサーバー（live-serverやPHPのビルトインサーバー等）で `public/index.html` をブラウザで表示する。開発者コンソールを開き、PHP APIから取得したマンガデータの配列（またはエラーメッセージ）が正しく表示されることを確認する。

・説明パート2（10-15分）：

- JavaScriptによるDOM操作を用いたデータ表示：
  - `document.getElementById()` や `document.querySelector()` で、HTML内の特定の要素（例：`<ul>`）を取得する方法。
  - `document.createElement('li')` で新しいHTML要素（例：リストアイテム）をメモリ上に作成する方法。
  - `element.textContent = "表示内容";` や `element.innerHTML = "<li>表示内容</li>";` で要素のテキスト内容やHTML内容を設定する方法。
  - `parentElement.appendChild(childElement)` で、作成した要素をDOMツリー内の指定した親要素の子として追加し、画面に表示する方法。
  - 取得したデータの配列を `forEach` ループで反復処理し、各データアイテムに対して上記のDOM操作を行い、複数のリストアイテムを動的に生成・表示する一般的な流れ。
- 簡単なUX（ユーザーエクスペリエンス）向上のためのTips：
  - ローディング表示：APIリクエスト開始時に「読み込み中」メッセージを表示し、データ取得完了（成功または失敗）時に非表示にすることで、ユーザーに処理中であることを伝える。
  - エラー表示：API通信失敗時やエラーレスポンス受信時に、画面上に適切なエラーメッセージを表示することで、ユーザーに状況を伝える。
- （補足）XSS（クロスサイトスクリプティング）対策としてのHTMLエスケープの重要性：ユーザー入力や外部APIから取得したデータを `innerHTML` で表示する際には注意が必要であること（今回は詳細には踏み込まず、`textContent` の利用を推奨する程度に留める）。

・演習2（30分）：取得データのHTML描画とローディング/エラー表示実装

- ステップ1（15分）：`public/js/app.js` の `fetch().then(data => { ... })` ブロック（データ取得成功時の処理）内を修正。まず、`id="manga-list"` の `<ul>` 要素を取得する。取得したマンガデータの配列（`data`）を `forEach` ループで処理し、各マンガオブジェクトについて、そのタイトル（と可能であれば簡単な説明や画像URLの一部）を含む `<li>` 要素を動的に生成する。生成した各 `<li>` 要素を `<ul>` 要素に `appendChild` して、ブラウザ上にマンガの一覧を表示する。データが空配列だった場合のフォールバック表示（例：「表示するマンガがありません。」）も考慮する。
- ステップ2（10分）：ローディング表示機能を実装する。APIリクエストを開始する直前（`fetch()` の呼び出し前）に、`id="loading-message"` の `<div>` 要素を表示状態（`style.display = 'block';`）にする。そして、データ取得が完了した後（2つ目の `.then()` の最初または `.catch()` の最初）に、このローディングメッセージを非表示状態（`style.display = 'none';`）にする処理を追加する。

- ステップ3（5分）：エラー表示機能を実装する。fetch().catch(error => { ... }) ブロック内で、id="error-message" の 要素を取得し、その textContent にエラーメッセージ（例：error.message や固定のエラー文）を設定し、表示状態にする。データ取得成功時には、このエラーメッセージがクリアされるか非表示になっていることを確認する。ブラウザで public/index.html をリロードし、ローディング表示、マンガ一覧表示、エラー発生時の表示（例：APIサーバーを一時停止してみるなど）が期待通りに動作することを確認する。

・完成イメージ：

- public/index.html ファイルには、マンガ一覧を表示するための <ul> 要素、ローディングメッセージ用の 要素、エラーメッセージ用の 要素が適切に配置されており、public/js/app.js が読み込まれている。
- public/js/app.js ファイルには以下のJavaScript処理が記述されている：
  - DOMContentLoaded イベント発生後、Fetch APIを使用して /api/mangas エンドポイントから非同期にマンガデータを取得する。
  - データ取得中は、「読み込み中...」のようなローディングメッセージが画面に表示され、取得完了後に非表示になる。
  - 取得に成功したマンガデータ（各マンガのタイトル等）が、JavaScriptによって動的にHTMLの リスト形式で index.html 上に描画される。
  - APIからのデータ取得に失敗した場合（ネットワークエラー、APIサーバーエラー等）、画面上に適切なエラーメッセージが表示される。
- ブラウザで public/index.html を表示すると、まずローディングメッセージが表示され、その後、PHP APIから取得されたマンガの一覧が（データがあれば）画面にリスト表示される。API接続に失敗した場合はエラーメッセージが表示される。

Day 4: 復習課題（新規テーマでの企画・設計・API準備）

コマ1：新テーマ「ゲーム攻略情報共有サイト」要件定義とUI設計（簡易版）

・目標：

- 新しいテーマ「ゲーム攻略情報共有サイト」の基本的な要件定義（目的、利用者、主要2機能、情報項目）をv0 Docs等を使って記述できる。
- v0 Draw等を利用して、主要2機能に対応する2画面（一覧表示、情報登録フォーム）のシンプルなワイヤーフレームを作成できる。
- 作成したワイヤーフレームのUI/UXについて、基本的な観点（情報の整理、操作の直感性、分かりやすさ）からセルフレビューできる。

- Day1で学んだ要件定義・UI設計のプロセスを、新しいテーマで再現・実践する。

・説明パート1（10-15分）：

- 復習課題の全体概要：
  - Day1〜Day3で習得した知識とスキル（要件定義、UI設計、HTML/CSS/JSによるフロントエンド実装、PHPによるAPI実装、DB連携、Fetch APIによる非同期通信）を総合的に活用する。
  - 新しいテーマ「ゲーム攻略情報共有サイト」を題材に、小規模なWebアプリケーションの企画・設計から実装までの一連の流れを体験する（Day4では主に企画・設計と実装準備、Day5でデプロイを想定）。
  - 今回は機能を大幅に絞り込んだ「簡易版」として取り組み、基本的な要素の実装に集中する。
- Day4の進め方とコマ1のゴール：
  - Day4では、新テーマの企画・設計（コマ1）、フロントエンドの静的実装（コマ2）、バックエンドAPIの準備（コマ3）、そしてフロントエンドとAPIの連携準備（コマ4）を段階的に行う。
  - コマ1の具体的なゴールは、新テーマ「ゲーム攻略情報共有サイト」の超簡易的な要件定義（3W+What）と、主要2画面（ゲーム一覧表示画面、ゲーム攻略情報登録フォーム画面）のワイヤーフレームを作成すること。

・演習1（30分）：v0 Docs（またはマークダウン）での簡易要件定義

- ステップ1（15分）：新しいテーマ「ゲーム攻略情報共有サイト」について、v0 Docsを利用するか、シンプルなマークダウンファイル（例：game-requirements.md）を作成し、以下の項目についてチームで話し合いながら簡潔に記述する。
  - Who（主な利用者）：例：自分自身、特定のゲーム仲間、少人数のコミュニティなど。
  - Why（サイトの目的・解決したい課題）：例：個人がプレイしたゲームの攻略メモや感想を記録・整理したい。特定のゲームの攻略情報を仲間内で共有したい。
  - What（主要機能）：今回は機能を絞り込み、「登録されたゲーム攻略情報の一覧表示機能」と「新しいゲーム攻略情報を登録する機能」の主要2機能に限定する。
  - What（ゲーム攻略情報として扱う主なデータ項目）：例：ゲームタイトル（テキスト・必須）、対応プラットフォーム（テキスト・任意）、攻略メモ（長文テキスト・任意）、クリア状況（選択式：未クリア/プレイ中/クリア済）、個人的評価（5段階評価など・任意）。
- ステップ2（15分）：作成した簡易要件定義の内容（記述した項目、機能、データ項目）をチーム内で共有し、曖昧な点がないか、主要2機能を実現するためにデータ項目に不足がないかなどを相互に確認し、必要であれば追記・修正する。

・説明パート2（10-15分）：

- ワイヤーフレーム作成のポイントの再確認：
    - 要件定義で定めた主要機能とデータ項目を基に、ユーザーがスムーズに操作できる画面の流れと情報配置を考える。
    - 今回は主要2画面（一覧表示、登録フォーム）に集中。ヘッダーやフッター、ナビゲーションなどの共通パーツは極力シンプルにするか、省略しても良い。
    - v0 DrawなどのAIツールを活用する場合は、どのようなプロンプトで指示すれば意図したワイヤーフレームが効率的に生成できるかを考える。
- 作成したワイヤーフレームのUI/UXセルフチェック観点（再確認）：
    - 情報の整理：必要な情報が過不足なく、分かりやすく整理されて配置されているか。
    - 操作の直感性：ユーザーが次に何をすべきか迷わずに操作を進められるか。
    - 見た目の分かりやすさ：要素のグルーピングや優先順位が視覚的に伝わるか。

・演習2（30分）：v0 Draw（または手描き/作図ツール）でのワイヤーフレーム作成とレビュー

- ステップ1（20分）：演習1で定義した「ゲーム攻略情報一覧表示画面」と「ゲーム攻略情報登録フォーム画面」の2つのワイヤーフレームを、v0 Draw（推奨）、手描き、または他の使い慣れた作図ツール（Figma, draw.ioなど）で作成する。ワイヤーフレームには、各画面で必要となる主要な情報要素（例：ゲームタイトル表示エリア、攻略メモ表示エリアなど）や操作要素（例：新規登録ボタン、登録実行ボタン、一覧へ戻るリンクなど）を具体的に配置する。
- ステップ2（10分）：作成した2画面のワイヤーフレームをチーム内で共有し、説明パート2で触れたUI/UXの観点（情報の整理、操作の直感性、分かりやすさ）に基づいて相互にレビューを行う。改善点や気づいた点があれば、お互いにフィードバックし、可能であればワイヤーフレームに反映する。

・完成イメージ：

- マークダウン形式（またはv0 Docsの出力結果）で記述された「ゲーム攻略情報共有サイト」に関する簡易要件定義書。
    - 「Who（主な利用者）」「Why（サイトの目的）」「What（主要2機能）」「What（ゲーム攻略情報の主なデータ項目）」がそれぞれ簡潔に明記されている。
- 「ゲーム攻略情報一覧表示画面」のワイヤーフレーム（画像ファイル、v0 Drawの共有URL、または手描きをスキャンしたものなど）。
    - 登録されているゲームのタイトル（やその他主要情報の一部）がリスト形式で表示されるエリアがある。

    - 新しいゲーム攻略情報を登録するための画面へ遷移するボタンまたはリンクが配置されている。
- 「ゲーム攻略情報登録フォーム画面」のワイヤーフレーム（画像ファイル、v0 Drawの共有URL、または手描きをスキャンしたものなど）。
    - ゲームタイトル、対応プラットフォーム、攻略メモ、クリア状況、個人的評価などの情報を入力するためのフォーム部品（テキスト入力欄、選択肢など）が配置されている。
    - 入力した情報を登録するための実行ボタン、および一覧表示画面へ戻るためのボタンまたはリンクが配置されている。
- 作成したワイヤーフレームについて、チーム内でUI/UXの観点からレビューを行い、その結果（口頭での合意、または簡単なメモ書き程度）が得られている。

コマ2：新テーマUI実装（v0 First Draft活用とCSS調整）

・目標：

- v0 First Draft を使用して、Day4-1で作成したワイヤーフレーム（一覧画面、登録フォーム画面）から基本的なHTML/CSSの骨子を生成できる。
- 生成されたHTML/CSSの構造を理解し、意図に合わせてクラス名の追加や軽微なHTML構造の修正ができる。
- サイトの基本的なデザイントークン（主要カラー、基本フォントサイズ、基本余白など）をCSSカスタムプロパティとして定義し、生成されたCSSに適用できる。
- 作成した2画面間で基本的なページ遷移（リンク設定）ができる。
- Day2で学んだv0 First Draftの活用方法とCSSの基礎知識を、新しいテーマで実践する。

・説明パート1（10-15分）：

- v0 First DraftによるUI生成プロセスの再確認：
    - Day4-1で作成したワイヤーフレームや簡易要件定義をインプットとして、v0 First Draftにどのような指示（プロンプト）を与えれば、目的のHTML/CSSの骨子を効率的に生成できるか。
    - 生成したい要素（ヘッダー、メインコンテンツエリア、リスト項目、フォーム部品、ボタンなど）、おおまかなレイアウト構造、希望するデザインの雰囲気（シンプル、モダンなど）をプロンプトに含めることの重要性。
- デザイントークンとCSSカスタムプロパティ（CSS変数）の戦略的な活用（復習）：
    - なぜデザイントークンを定義するのか：サイト全体で色、フォント、スペーシングなどのスタイル値に一貫性を持たせ、保守性・再利用性を高めるため。
    - CSSカスタムプロパティの定義方法（`:root` セレクタ内でのグローバルな定義：`--main-color: #RRGGBB;`）と、`var()` 関数を使った利用方法（例：`background-color: var(--main-color);`）の再確認。

    - 今回の簡易サイトで定義すべき基本的なデザイントークンの種類（プライマリーカラー、アクセントカラー、基本テキスト色、基本背景色、基本フォントファミリー、基本フォントサイズ、主要な余白単位など）。

・演習1（30分）：v0 First DraftによるUI骨子生成とデザイントークン定義

- ステップ1（15分）：Day4-1で作成した「ゲーム攻略情報一覧表示画面」と「ゲーム攻略情報登録フォーム画面」のワイヤーフレームを基に、v0 First Draftにそれぞれの画面に対応するHTMLとCSSの生成を指示する。プロンプトには、ワイヤーフレームで定義した要素（例：一覧表示のためのリスト構造、フォームのためのラベルと入力フィールド、ナビゲーション用のボタンやリンクなど）と、それらの大まかな配置関係を記述する。生成されたHTMLコードを、それぞれ `public/game_list.html` および `public/game_form.html` というファイル名で保存する。生成されたCSSコードは、一旦 `public/css/game_style_raw.css` のような一時ファイルに保存するか、または直接 `public/css/game_style.css` にコピー＆ペーストする（後で調整するため）。
- ステップ2（15分）：`public/css/game_style.css` ファイルを作成または開き、ファイルの先頭部分に、今回の「ゲーム攻略情報共有サイト」で使用するシンプルなデザイントークンをCSSカスタムプロパティとして定義する。少なくとも以下の項目について、具体的な値を設定する。
    - `--primary-text-color`（例: `#212529`）
    - `--secondary-text-color`（例: `#6c757d`）
    - `--primary-bg-color`（例: `#ffffff`）
    - `--secondary-bg-color`（例: `#e9ecef`）
    - `--accent-color1`（例: `teal`）
    - `--accent-color2`（例: `coral`）
    - `--font-family-sans-serif`（例: `"Helvetica Neue", Arial, sans-serif`）
    - `--base-font-size`（例: `1rem`）
    - `--spacing-xs`（例: `0.25rem`）
    - `--spacing-sm`（例: `0.5rem`）
    - `--spacing-md`（例: `1rem`）
    - `--spacing-lg`（例: `1.5rem`）
    - `--border-radius`（例: `0.25rem`）

・説明パート2（10-15分）：

- v0 First Draftが生成したHTML/CSSの調整戦略：
    - HTML構造の検証：生成されたHTMLがセマンティックであるか、不要なネストやタグがないかを確認。必要に応じてクラス名を追加したり、IDを付与したりして、CSSでのスタイリングやJavaScriptでの操作がしやすいように調整する。

    - CSSの検証とリファクタリング：生成されたCSSセレクタが適切か、冗長なスタイル指定がないかを確認。汎用的なクラスセレクタへの置き換えや、スタイルルールのグルーピングなどを検討する。
    - デザイントークン（CSS変数）の具体的な適用箇所：サイト全体のフォント設定、背景色、主要な要素（ヘッダー、ボタン、カードなど）の色や余白、境界線などに、定義したCSS変数を一貫して適用していく。
- 簡単なページ間ナビゲーションの実装：HTMLの `<a>` タグを使用し、`href` 属性に遷移先のHTMLファイルパスを指定することで、基本的な画面遷移を実現する。

・演習2（30分）：HTML/CSSの調整、デザイントークン適用、画面間リンク設定

- ステップ1（15分）：v0 First Draftで生成・保存した `public/game_list.html` と `public/game_form.html` の内容を確認し、HTML構造を必要に応じて微調整する（例：意味のあるクラス名の付与、不要なdivの削除、リスト構造の確認など）。次に `public/css/game_style.css`（または `game_style_raw.css` からコピーした内容）を確認し、v0が生成したスタイルルールをベースに、演習1で定義したデザイントークン（CSSカスタムプロパティ）を適用するように書き換える。例えば、`body` のフォントファミリーや基本文字色、コンテナ要素の最大幅や中央寄せ、ボタンの基本スタイルなどにデザイントークンを使用する。
- ステップ2（10分）：「ゲーム一覧画面」（`game_list.html`）の主要な要素（例：各ゲーム情報のカード、新規登録ボタン）や、「ゲーム登録フォーム画面」（`game_form.html`）の主要な要素（例：フォームの各入力欄、登録ボタン、一覧へ戻るボタン）に対して、デザイントークンを用いた具体的なスタイリング（背景色、文字色、パディング、マージン、ボーダーなど）を `public/css/game_style.css` に追加・修正していく。
- ステップ3（5分）：`public/game_list.html` 内に、「新しいゲーム情報を登録する」ためのリンクまたはボタンを設置し、その `href` 属性を `game_form.html` に設定する。同様に、`public/game_form.html` 内に、「ゲーム一覧に戻る」ためのリンクまたはボタンを設置し、その `href` 属性を `game_list.html` に設定する。ローカルサーバーでHTMLファイルを開き、ブラウザで表示を確認。基本的なスタイリングが適用されていること、および画面間のリンクで正しく遷移できることを確認する。

・完成イメージ：

- `public/game_list.html`: v0 First Draftによって生成され、HTML構造が基本的なレベルで調整された「ゲーム攻略情報一覧表示」画面のHTMLファイル。適切なクラス名が付与されている。
- `public/game_form.html`: v0 First Draftによって生成され、HTML構造が基本的なレベルで調整された「ゲーム攻略情報登録フォーム」画面のHTMLファイル。適切なクラス名が付与されている。
- `public/css/game_style.css`:

    - ファイルの先頭部分に、サイトの主要なデザイントークン（色、フォント、余白、ボーダーラディウスなど）が複数のCSSカスタムプロパティとして定義されている。
    - 上記で定義したカスタムプロパティを広範囲に使用して、game_list.html および game_form.html の基本的な見た目（全体のレイアウト、文字の装飾、ボタンやフォーム部品の基本的なスタイル、カード型要素のスタイルなど）が、統一感を持って整えられている。
- ブラウザで public/game_list.html および public/game_form.html を表示した際に、定義したデザイントークンに基づいた基本的なスタイリングが適用されており、かつ、両画面間で設定したリンク（例：「新規登録へ」「一覧へ戻る」）によって相互にページ遷移ができる状態。

Day 5: デプロイ・発表

コマ1：重要概念復習とデプロイ準備

目標

- 3層構造Webアプリケーションの各層の役割を再確認する
- API、JSON、Fetch通信の基本概念を復習し、理解を深める
- Render.comへのデプロイに向けた準備と心構えを整える
- ローカル環境とデプロイ環境の違い（特にAPI URL、DB接続情報）を理解する

説明パート（Q&A形式でのディスカッションを想定）

- **3層構造（Three-Tier Architecture）の復習**
    - 各層（プレゼンテーション層、アプリケーション層、データ層）の名称と主な役割
    - 本講座で作成したアプリにおける各層の具体的な対応技術・ファイル群
    - publicフォルダとapiフォルダの分離と3層構造の関連性
- **API（Application Programming Interface）の復習**
    - APIの定義とWebアプリケーション開発における重要性
    - フロントエンドとバックエンドのAPI連携の仕組み（リクエストとレスポンス）
- **JSON（JavaScript Object Notation）の復習**
    - JSONの定義と特徴（軽量性、可読性、JavaScriptとの親和性、多言語サポート）
    - APIでのデータ交換におけるJSONの利用理由
- **Fetch APIと非同期処理の復習**
    - Fetch APIの役割と目的（HTTPリクエスト送信・レスポンス受信）
    - 非同期処理の概念と必要性（ユーザー体験の向上）

    - `.then()`, `.catch()`, `async/await` と非同期処理の関係
- **デプロイの基本**
    - デプロイの定義と目的
    - Render.comの概要とPaaSとしての役割
    - 静的サイト（Static Site）とWebサービス（Web Service）の違いと、本講座アプリでの使い分け
    - ローカル環境とデプロイ環境の主な違い（API URL、DB接続情報、環境変数など）

演習内容（ディスカッションと確認）

1. **重要概念のQ&Aセッション（30-40分）**
    - 上記の「説明パート」に挙げた各項目について、講師からの質問に答えたり、チーム内で相互に説明し合ったりする形で理解度を確認する。
    - 特に、自身が担当した箇所や理解が曖昧な部分について積極的に質問し、疑問点を解消する。
2. **デプロイ準備のためのチェックリスト確認（15-20分）**
    - プロジェクトのファイル構成（`public/api`分離）が正しいか確認。
    - フロントエンドのJSコード内のAPI呼び出しURLが、デプロイ後の変更可能性を考慮した記述になっているか（または変更箇所を特定できているか）確認。
    - DB接続情報（Render.comのDBサービス利用時）のローカルとデプロイ環境での分離方法（環境変数の利用など）について理解を深める。
    - コード内に不要なファイルや機密情報（パスワード直書きなど）が含まれていないか確認。
    - Gitリポジトリの状態（コミット漏れがないか、ブランチ戦略など）を確認。

完成イメージ

- 3層構造、API、JSON、Fetch API、デプロイといった重要概念について、自身の言葉で説明できるレベルで理解が深まっている。
- Render.comへのデプロイ作業をスムーズに進めるための知識が整理され、具体的な準備項目が明確になっている。
- ローカル環境とデプロイ環境の違いを認識し、対応が必要な箇所（APIの向き先、DB設定等）を把握できている。
- チーム内でデプロイに関する疑問点や懸念事項が共有され、解消に向けた見通しが立っている。

コマ2：Render.comへのデプロイ実践（バックエンドAPIとDB）

目標

- Render.comでアカウントを作成し、基本的なダッシュボード操作に慣れる
- Render.com上でPostgreSQLデータベースを準備し、接続情報を取得できる
- バックエンドAPI（apiフォルダ）をRender.comのWeb Serviceとしてデプロイできる
- デプロイ時に環境変数を設定し、PHPコードからDB接続情報を読み込めるようにする
- デプロイしたAPIのエンドポイントにPostman等でアクセスし、動作確認ができる

説明パート

- **Render.comの概要とアカウント作成**
  - Render.comのサービス紹介（PaaS）、無料プランの範囲
  - アカウント作成手順（GitHubアカウント連携推奨）
  - ダッシュボードの主要機能（サービス作成、ログ確認、設定変更など）
- **Render.comでのPostgreSQLデータベース作成**
  - 「New +」->「PostgreSQL」からのデータベース作成フロー
  - インスタンス名、DB名、ユーザー名、リージョン設定のポイント
  - 無料プランの選択と制限事項
  - 作成後に表示される接続情報（Internal/External Connection String、ホスト、ポート、ユーザー、パスワード等）の確認と保管方法
- **バックエンドAPI（PHP）のWeb Serviceとしてのデプロイ準備**
  - PHPコードの修正（Database.php等）：DB接続情報をハードコーディングから環境変数経由で取得するように変更（getenv()）。
  - PDOのDSNをMySQL用からPostgreSQL用に修正（pgsql:host=...）
  - api/index.php（エントリポイント）の確認：CORS設定、エラーハンドリング、リクエストルーティング処理。
  - apiフォルダをGitリポジトリのルートまたはサブディレクトリとしてプッシュする準備。
- **Render.comでのWeb Service作成とデプロイ**
  - 「New +」->「Web Service」からのサービス作成フロー
  - GitHubリポジトリとの連携
  - 設定項目：サービス名、リージョン、ブランチ、ルートディレクトリ（api等）、ランタイム（PHP）、ビルドコマンド（composer install等、必要に応じて）、スタートコマンド（PHPサーバー起動コマンド、Render.comがindex.phpを自動認識する場合も）
  - 無料プランの選択
- **環境変数の設定**
  - Web Serviceの「Environment」セクションでの環境変数設定方法
  - 設定する環境変数：DB_HOST, DB_NAME, DB_USER, DB_PASSWORD, DB_PORT, FRONTEND_URL（後で設定）
- **デプロイプロセスの確認とログ**
  - 「Create Web Service」後のデプロイプロセスとログの確認ポイント

    - デプロイ成功後に提供されるAPIのURL（例: https://your-api.onrender.com）
- **データベースへのテーブル作成**
    - psqlコマンドラインツールやpgAdmin等のGUIツールを使ったRender.com上のPostgreSQLへの接続方法
    - 既存のテーブル作成SQL（MySQL用）をPostgreSQL用に修正する際の注意点（AUTO_INCREMENT -> SERIAL, データ型、CURRENT_TIMESTAMPの扱いなど）
    - 修正したSQLを実行してテーブルを作成

演習内容

1. **Render.comアカウント作成とDB準備（20分）**
    - Render.comにサインアップ（GitHub連携推奨）
    - ダッシュボードから新しいPostgreSQLデータベースを作成（Freeプラン）。リージョンはTokyoを選択。
    - 作成されたDBの接続情報（Internal Connection Stringなど）を控える。
2. **バックエンドAPIのコード修正とデプロイ準備（30分）**
    - ローカルのapiフォルダ内のPHPコードを修正：
        - Database.php（または同等のDB接続クラス）で、DB接続情報を環境変数から取得するように変更。
        - PDOのDSNをPostgreSQL用に修正。
        - （必要であれば）テーブル作成SQL文をPostgreSQL用に修正し、別途保存しておく。
    - 修正したapiフォルダ（全体、またはapiディレクトリのみ）をGitHubリポジトリにプッシュ。
3. **Render.comへのWeb ServiceとしてのAPIデプロイ（30分）**
    - Render.comで新しいWeb Serviceを作成。
    - GitHubリポジトリと連携し、apiコードが含まれるブランチ、ルートディレクトリを指定。
    - ランタイムにPHPを選択。ビルドコマンド、スタートコマンドを適切に設定（またはRenderのデフォルトに期待）。
    - 環境変数に、手順1で控えたDB接続情報（DB_HOST等）を設定。
    - Web Serviceを作成し、デプロイを開始。デプロイログでエラーがないか確認。
    - デプロイ成功後、提供されたAPIのURLを控える。
4. **データベースマイグレーション（テーブル作成）（20分）**
    - PostgreSQLクライアントツール（psql, pgAdmin, DBeaver等）を使用して、Render.com上のPostgreSQLデータベースに接続。
    - 準備しておいたPostgreSQL用のテーブル作成SQLを実行し、必要なテーブル（例: mangas, games）を作成。
5. **API動作確認（10分）**

- Postmanやブラウザから、デプロイされたAPIのURL（例: https://your-api.onrender.com/mangas）にアクセスし、正常にレスポンスが返ってくるか（空の配列でもOK）、またはエラーメッセージが適切かを確認。

完成イメージ

- Render.com上にPostgreSQLデータベースが作成され、接続情報が取得できている。
- バックエンドAPI（PHP）がRender.comのWeb Serviceとしてデプロイされ、公開URLが発行されている。
- Web Serviceの環境変数にデータベース接続情報が正しく設定されている。
- デプロイされたPostgreSQLデータベース上に、アプリケーションに必要なテーブル構造が作成されている。
- デプロイされたAPIのエンドポイントにアクセスすると、期待される初期応答（データが空でもJSON形式の応答や、適切なエラーメッセージ）が返ってくる。

コマ3：フロントエンドのデプロイと最終連携確認

目標

- フロントエンド（publicフォルダ）をRender.comのStatic Siteとしてデプロイできる
- デプロイしたフロントエンドのJavaScriptコード内で、API呼び出し先のURLをデプロイ済みのバックエンドAPIのURLに正しく修正できる
- バックエンドAPIの環境変数 FRONTEND_URL に、デプロイしたフロントエンドのURLを設定し、CORS設定を適切に行える
- デプロイされたWebアプリケーション（フロントエンド＋バックエンドAPI＋DB）全体の動作確認を行い、ローカル同様に機能することを検証する
- 「作ったものがWebで公開され、誰でもアクセスできる状態になる」という一連の体験を完了する

説明パート

- **フロントエンド（Static Site）のデプロイ準備**
  - publicフォルダ内のJavaScriptファイル（例: app.js, list.js, form.js）で、APIを呼び出している箇所のURLを確認。
  - ローカル開発用のAPI URL（例: http://localhost/api/...）から、Day5コマ2でデプロイしたバックエンドAPIの公開URL（例: https://your-api.onrender.com/...）へ変更する必要性を理解する。
- **Render.comでのStatic Site作成とデプロイ**
  - 「New +」->「Static Site」からのサービス作成フロー。
  - GitHubリポジトリとの連携。

    - 設定項目：サービス名、ブランチ、ルートディレクトリ（public等）、ビルドコマンド（通常は空欄または静的サイトジェネレータのコマンド）、パブリッシュディレクトリ（public等）。
    - 無料プランの選択。
- **API呼び出しURLの修正と反映**
    - ローカルのJavaScriptファイル内のAPIベースURLを、デプロイしたバックエンドAPIのURLに書き換える。
    - 変更をコミットし、GitHubにプッシュすることで、Render.comのStatic Siteに自動的に再デプロイがかかることを確認（または手動でトリガー）。
- **CORS設定の最終確認と調整**
    - バックエンドAPI（Web Service）の環境変数 `FRONTEND_URL` に、デプロイした静的サイトのURL（例: `https://your-frontend.onrender.com`）を正確に設定する。
    - PHP側の `Access-Control-Allow-Origin` ヘッダーが、この `FRONTEND_URL` を正しく参照しているか確認。
- **デプロイ後の全体動作確認ポイント**
    - フロントエンドURLにアクセスし、画面が表示されるか。
    - APIからのデータ取得・表示（一覧表示など）が正常に行われるか。
    - フォームからのデータ登録（POSTリクエスト）が正常に行われ、データがDBに保存されるか。
    - ブラウザの開発者ツール（ネットワークタブ、コンソール）でエラーが出ていないか。

演習内容

1. **フロントエンドのAPI呼び出しURL修正（15分）**
    - ローカルの `public` フォルダ内のJavaScriptファイル（APIを呼び出しているファイル全て）を開く。
    - APIのベースURLを指定している箇所を、Day5コマ2で控えた「デプロイ済みバックエンドAPIのURL」に書き換える。
    - 変更内容をGitにコミットし、GitHubリポジトリにプッシュする。
2. **Render.comへのStatic Siteとしてのフロントエンドデプロイ（25分）**
    - Render.comで新しいStatic Siteを作成。
    - GitHubリポジトリと連携し、publicコードが含まれるブランチ、ルートディレクトリ（例: `public`）を指定。
    - ビルドコマンドやパブリッシュディレクトリを適切に設定（シンプルなHTML/CSS/JSの場合はデフォルトまたはpublicで可）。
    - Static Siteを作成し、デプロイを開始。デプロイログでエラーがないか確認。
    - デプロイ成功後、提供されたフロントエンドの公開URLを控える。
3. **バックエンドAPIのCORS設定更新（10分）**

- Render.comのダッシュボードで、Day5コマ2で作成したバックエンドAPI（Web Service）の設定を開く。
- 環境変数 FRONTEND_URL の値を、手順2で控えた「デプロイ済みフロントエンドのURL」に設定（または更新）する。
- 設定変更後、Web Serviceが再起動または設定を再読み込みするのを待つ。

4. **デプロイ済みアプリケーションの全体動作確認（30分）**
    - デプロイされたフロントエンドのURLにブラウザでアクセス。
    - ゲーム一覧表示機能が動作し、APIから取得したデータ（初期データまたは登録済みデータ）が正しく表示されるか確認。
    - ゲーム登録フォームから新しいデータを入力・送信し、データが正常に登録されるか（エラーが出ないか、一覧に反映されるか）確認。
    - ブラウザの開発者ツールを開き、コンソールにエラーが出ていないか、ネットワークタブでAPIリクエスト・レスポンスが期待通りかを確認。
    - （任意）Postman等で、デプロイ済みAPIに対して直接リクエストを送信し、DBのデータを確認。
5. **トラブルシューティング（必要に応じて）**
    - API連携がうまくいかない場合（CORSエラー、404エラー、500エラーなど）、それぞれのログ（ブラウザコンソール、APIのランタイムログ、DBログ）を確認し、原因を特定して修正する。
        - CORSエラー：FRONTEND_URL の設定ミス、API側のヘッダー設定漏れなど。
        - API URL間違い：フロントエンドのJS内のURL設定ミス。
        - DB接続エラー：API側の環境変数設定ミス、DB側の準備不足など。

完成イメージ

- フロントエンドアプリケーション（HTML/CSS/JS）がRender.comのStatic Siteとしてデプロイされ、公開URLが発行されている。
- フロントエンドのJavaScriptコード内のAPI呼び出し先が、デプロイ済みのバックエンドAPIの正しいURLに設定されている。
- バックエンドAPIのCORS設定が、デプロイ済みのフロントエンドURLを許可するように正しく設定されている。
- Webブラウザからデプロイ済みフロントエンドURLにアクセスすると、バックエンドAPIおよびデータベースと連携したWebアプリケーションが完全に動作する状態になっている（データの表示、登録など）。
- ローカル開発環境とほぼ同様のユーザー体験が、公開されたWeb環境で実現できている。

コマ4：成果発表と講座全体の振り返り

目標

- 5日間の演習で制作・デプロイしたWebアプリケーションの概要、工夫した点、学んだことをまとめた発表資料を作成できる。
- ライトニングトーク形式（3〜5分程度）で、デプロイした自身のアプリケーションをデモンストレーションしながら成果を発表できる。
- KPT法（Keep, Problem, Try）を用いて、個人の学習成果と課題を客観的に整理する。
- 今後の継続的な学習やスキルアップに向けた具体的なアクションプラン（Try）を立てる。

説明パート

- **成果発表の目的と進め方**
    - 5日間の学びと成果を共有し、達成感を味わう。
    - ライトニングトーク（LT）形式での発表（3〜5分/人）。
    - 発表資料（スライド3〜5枚程度）の準備について。
    - デモンストレーションのポイント（主要機能、こだわった点）。
- **発表資料の構成案（例）**
    1. タイトルページ（アプリ名、発表者名、スクリーンショット等）
    2. 作ったもの紹介（コンセプト、主な機能、こだわった点/頑張った点、デモURL）
    3. 開発を通して感じたこと・学んだこと（面白かったこと、難しかったこと、今後やってみたいこと）
    4. （任意）謝辞など
- **ライトニングトークのポイント**
    - 時間配分（自己紹介とアプリ紹介、デモンストレーション、感想とまとめ）。
    - スムーズなデモのための事前準備。
    - 「何を作り、どこを頑張り、何を感じたか」を自分の言葉で伝える。
- **KPT法による振り返り**
    - KPT法（Keep, Problem, Try）のフレームワーク説明。
        - Keep: 良かったこと・続けたいこと。
        - Problem: 課題・改善点。
        - Try: 次に挑戦すること・試したいこと。
    - 振り返りの観点（技術的学び、開発プロセス、ツール活用、AI協調、自己学習、Web公開経験、LT経験など）。
- **今後の学習リソース紹介（情報共有）**
    - 公式ドキュメント（MDN, PHP.net, Render.com等）。
    - オンライン学習プラットフォーム（Progate, Udemy, freeCodeCamp等）。
    - 技術ブログ、コミュニティ、書籍など。

演習内容

1. **発表資料作成（30-40分）**
    - 各自、DAY5コマ2・3でデプロイした自身のWebアプリケーションについて、説明パートの構成案を参考に発表資料（スライド3〜5枚程度）を作成する。
    - スクリーンショットやデモURL（QRコードも可）を盛り込む。
    - 特に「こだわった点/頑張った点」や「感じたこと/学んだこと」を自分の言葉で記述する。
2. **ライトニングトーク準備・練習（20-30分）**
    - 作成した資料をもとに、3〜5分のライトニングトークの構成を考え、話す内容をまとめる。
    - 実際に声に出して練習し、時間内に収まるように調整する。
    - デモンストレーションの手順を確認し、スムーズに行えるように練習する。
3. **成果発表会（ライトニングトーク）（60分〜 ※人数による）**
    - 各自、準備した資料とデプロイ済みアプリケーションを使ってライトニングトークを行う。
    - 発表後、質疑応答やフィードバックの時間を設ける（任意）。
4. **KPT法による個人振り返り（15分）**
    - 5日間の講座全体（企画〜デプロイ、発表まで）を振り返り、Keep, Problem, Tryをそれぞれ3つ以上書き出す。
    - 具体的なエピソードや感情も交えながら記述する。
5. **KPT共有とディスカッション（20分）**
    - グループ内で各自のKPTを共有し、共感点や新たな気づきを話し合う。
    - 特に「Try」について、具体的なアクションや参考情報などを相互にアドバイスし合う。
6. **クロージング・今後のアナウンス（10分）**
    - 講師からの総括、質疑応答。
    - 今後の学習に役立つリソースの共有（再掲）。

完成イメージ

- 各自が作成したWebアプリケーションの発表資料（スライド3〜5枚程度）。
- デプロイ済みのアプリケーションを用いたライトニングトークの実施経験。
- KPT法に基づいて整理された個人の学習記録（Keep, Problem, Tryが具体的に記述されている）。
- 今後の学習やスキルアップに向けた具体的な目標（Try）が設定されている。
- 講座全体の達成感と、次のステップへのモチベーションが得られている。

SupabaseとRender.comについて

当初案では、Supabaseで公開という想定だったが、render.com とか glitchのほうがいいか？

デプロイ先の選定についてですね。Render.comやGlitchへの変更をご検討されているとのこと、それぞれの特徴を踏まえて学生の「驚き体験」と講座の目的に最適なものを考えてみましょう。

各プラットフォームの特徴と考察

- Supabase:
    - **メリット:** PostgreSQLデータベース、認証、ストレージなどが統合されており、本格的なバックエンド機能を比較的簡単に構築・利用できる。BaaS (Backend as a Service) としての側面が強い。
    - **デメリット:** 今回の講座の主眼は「素のPHP APIと静的フロントエンドの分離」であり、Supabaseの高機能なバックエンド機能は必ずしも必要ないかもしれません。データベースもDockerでMySQLを簡易的に利用する案や、最悪JSONファイルで代用するレベルも視野に入れているため、Supabaseの学習コストやセットアップが学生にとって少し重荷になる可能性があります。
    - **講座コンセプトとの整合性:** 「PHP+SQL」という学生の既存スキルセットを活かす点ではMySQLの方が親和性が高いかもしれません。SupabaseのDB機能を使う場合、PostgreSQLの癖やSupabase固有のAPI操作などを覚える必要が出てきます。
- Render.com:
    - メリット:
        - **静的サイトとWebサービス (PHP APIなど) のデプロイが容易:** GitHubリポジトリと連携させるだけで、ビルドからデプロイまで自動化できます。
        - **無料枠が充実:** 小規模なアプリケーションであれば、無料枠で十分に運用可能です。
        - **PHPのサポート:** PHPランタイムをサポートしているため、作成したPHP APIをそのままデプロイしやすいです。
        - **データベースサービスも提供:** 必要であればPostgreSQLなどのデータベースもRender上で簡単に作成・連携できます（ただし、これも学習コストは考慮）。
        - **分かりやすいUI/UX:** 学生にとっても比較的直感的に操作しやすいインターフェースです。
    - デメリット:

        - 無料枠の場合、スリープ状態になることがある（長時間アクセスがない場合）。ただし、学生が短期間でアクセスして体験する分には大きな問題にならないでしょう。
        - ビルドやデプロイに多少時間がかかる場合がある。
    - **講座コンセプトとの整合性:** 静的フロントエンドとPHP APIを別々のサービスとしてデプロイし、それらを連携させるという3層構造の体験には非常に適しています。「プログラムをコントロールできる」という感覚も得やすいでしょう。
- Glitch:
    - メリット:
        - **非常に手軽で迅速なデプロイ:** ブラウザ上でコードを編集し、リアルタイムで変更が反映されます。Remix機能で簡単にテンプレートからプロジェクトを開始できます。
        - **Node.jsベースの環境が主だが、PHPも工夫次第で動作可能:** ただし、PHPをネイティブサポートしているわけではないため、セットアップに一手間かかる可能性があります（例: `index.js` からPHPを呼び出すなど）。
        - **コラボレーション機能:** 複数人での同時編集が可能です（今回の講座では必須ではないかもしれません）。
        - **「動いた！」という即時性:** コードを少し変えるだけで、すぐに結果が見えるため、初学者には達成感を得やすいです。
    - デメリット:
        - **PHPのネイティブサポートではない:** PHP APIをデプロイするには工夫が必要で、講座の趣旨である「素のPHP API」をそのまま活かしにくい可能性があります。
        - **永続的なストレージやデータベースには不向き:** あくまで手軽なプロトタイピングや小規模アプリ向けです。
        - **無料プランの制限:** プロジェクトがスリープしたり、リソース制限があったりします。
    - **講座コンセプトとの整合性:** 「手軽さ」「即時性」という点では優れていますが、PHP APIのデプロイという点では、Render.comに比べて手間がかかる可能性があります。静的フロントエンドのホスティングには非常に適しています。

提案

学生のスキルレベル（偏差値40程度、英語は苦手、Webアプリ開発歴2ヶ月）と講座の目的（3層構造の理解、プログラムのコントロール感）を考慮すると、以下の選択肢が考えられます。

1. **Render.com を最有力候補とする**
    - **理由:**
        - 静的サイト（HTML/CSS/JS）とPHP APIを別々のサービスとして、比較的容易にデプロイできるため、3層構造の分離を明確に体験できます。
        - GitHub連携による自動デプロイは、現代的な開発フローの初歩を体験する良い機会になります。
        - 無料枠で 충분히利用可能です。
        - 学生が将来的に自分のポートフォリオサイトや小規模なWebアプリを公開する際にも役立つ実践的なスキルが身につきます。
    - **懸念点と対策:**
        - デプロイ設定（ビルドコマンドや起動コマンドなど）で少しつまづく可能性があるため、詳細な手順書や教員によるサポートが必要です。
        - データベースをどうするかですが、Render.com上で無料のPostgreSQLインスタンスを作成してPHPから接続する、あるいは前回提案したように、講座内ではJSONファイルで代用し、DB連携は「今後の発展」として示す形でも良いでしょう。MySQLに慣れている学生には、Render.comのPostgreSQLは少し戸惑うかもしれませんが、基本的なSQL操作はほぼ同じです。
2. **Glitch を静的フロントエンドの公開に限定して利用し、PHP APIはローカル環境（またはPHP内蔵サーバーを外部公開するngrokのようなツール）で動かす**
    - **理由:**
        - Glitchは静的サイトの公開が非常に手軽で、「作ったものがすぐに見える」という驚き体験を提供しやすいです。
        - PHP APIのデプロイの複雑さを回避できます。
    - **懸念点と対策:**
        - APIがローカルにある場合、他の学生がアクセスできないため、「公開」の範囲が限定的になります。ngrokのようなツールを使えば一時的に外部公開できますが、設定や安定性に課題が残る場合があります。

        - 3層構造の「完全な分離と連携」の体験としては少し弱くなります。

3. Supabase を使う場合の再検討

    - もしSupabaseを利用するのであれば、その機能をフルに活用するのではなく、主に**PostgreSQLデータベースの提供元**および**静的ファイルのホスティング先**として限定的に利用し、PHP APIのデプロイという点では、Render.comなどにデプロイする、あるいはSupabase Functions (Edge Functions) を使ってJavaScript/TypeScriptでAPIを書く（これはPHPから逸脱しますが）という選択肢も考えられます。
    - ただ、講座のコンセプトである「素のPHP API」を活かすには、やはり、Render.comのようなPHPを直接デプロイできる環境の方がシンプルかもしれません。

結論としてのおすすめ

現時点では、**Render.com** が最もバランスが良く、講座の目的と学生の体験価値を高める上で有力な選択肢だと考えます。

- 静的フロントエンドとPHP APIをそれぞれデプロイすることで、レイヤー分離を明確に意識させることができます。
- GitHub経由でのデプロイは、バージョン管理と連携した開発フローの初歩を体験できます。
- 無料枠で十分対応可能です。

ただし、どのプラットフォームを選択するにしても、学生が迷わないように**詳細な手順書（スクリーンショット付き推奨）**を作成し、TAによる手厚いサポート体制を整えることが成功の鍵となります。

デプロイ作業自体はDAY5の演習時間内（例えば1コマ分）で行い、残りの時間で相互レビューや発表、振り返りを行うのが良いでしょう。

最終的な判断は、講座の具体的な時間配分やTAのリソース、そして何よりも「学生に何を持ち帰ってほしいか」という最も重要なポイントに照らし合わせてご決定いただければと思います。