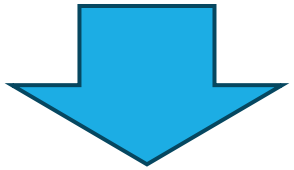
動機

ポケモンのタイプ相性は複雑（記憶頼りでは限界）
「相性」だけでなく「素早さ（先制できるか）」も重要な要素



膨大なデータをDBで管理し、瞬時に最適なポケモンを検索するシステムを構築する

ペルソナ

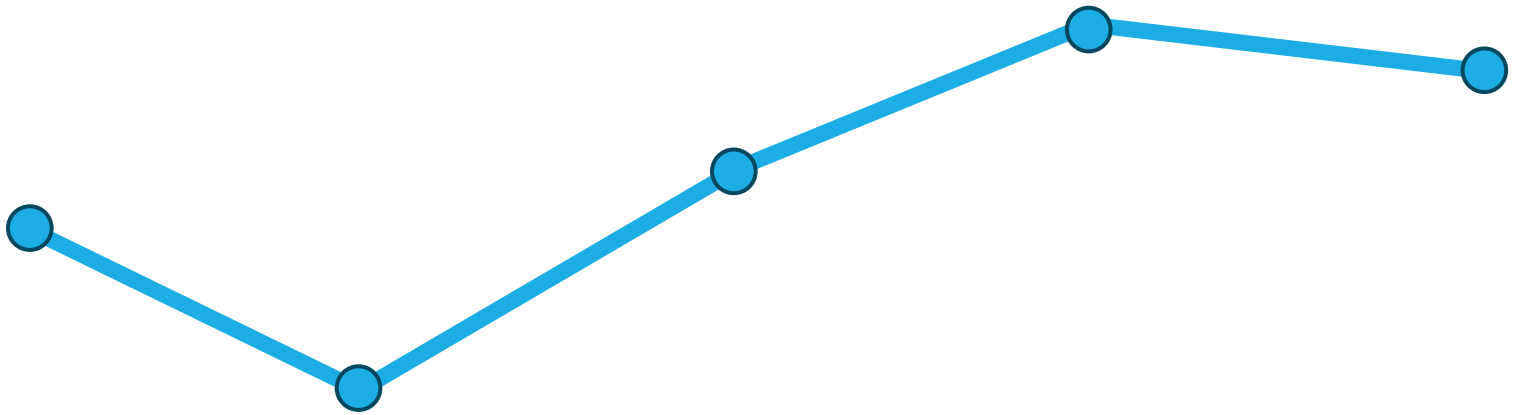
名前：都市大 サトシ（21歳）
属性：私立大学3年生 / ポケモン歴1年
性格：論理的に勝ちたいが、暗記が苦手
負けず嫌いで、ポケモンバトルで勝てるように努力は惜しまない

現状の課題 ：
「タイプ相性表」を見ながら戦うが、複合タイプ（例：ほのお/ひこう）の計算に時間がかかり、技選択時間が足りなくなる
「さっきの相手、スカーフ持ちだったっけ？」と記憶があいまいで、同じ戦法で負ける

ゴール：
相手のパーティを見て、3秒で「最適解」を選出したい

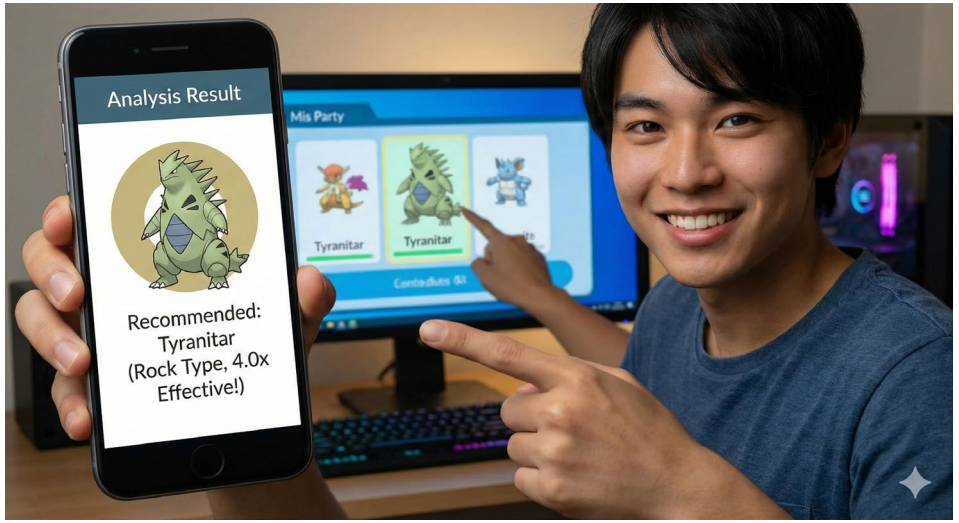
モチベショーングラフ

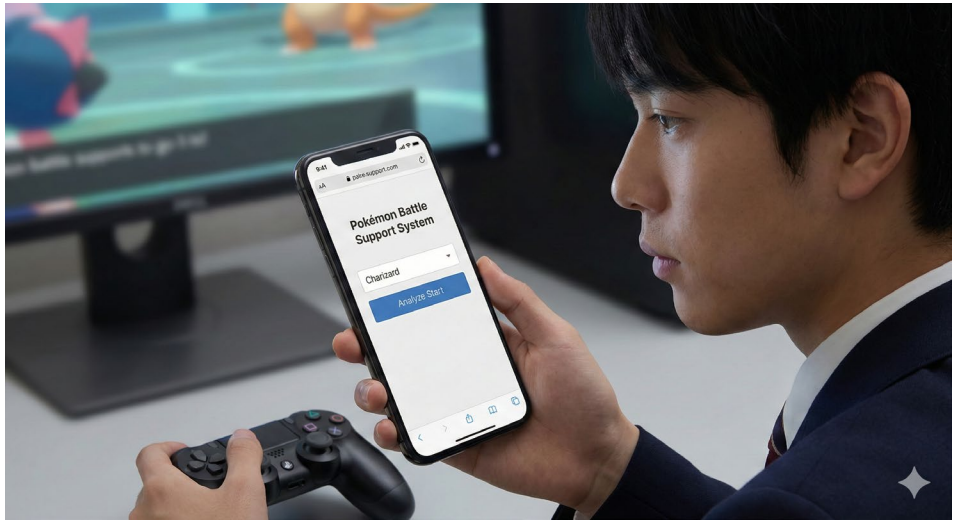
	マッチング	混乱	アプリ導入	勝利	記録・成長
モチベーション	0	−3	+3	+5	+4
状況	強敵と遭遇	弱点が思い出せず焦る	推奨ポケモン判明	有利に進め、勝利	メモで振り返り、勝利
心情	強そうだ、、、	弱点なんだっけ？	あった！これなら、、、	読み通り！	忘れないように、、、



ストーリーボード

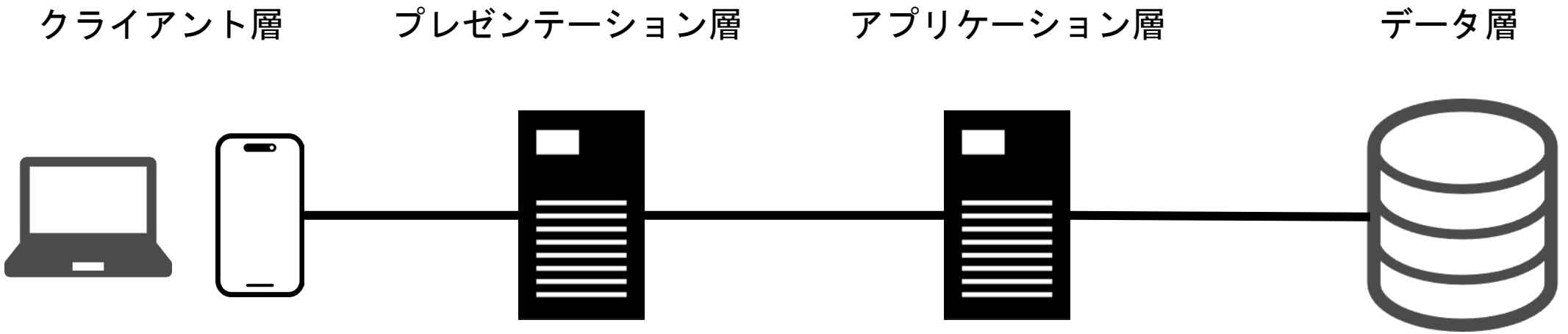

Charizard
Lv.20
126/140


Analysis Result
Recommended:
Tyranitar
(Rock Type, 4.0x
Effective!)
Mis Party
Tyranitar
Tyranitar


Pokémon Battle
Support System
Charizard
Analyze Start


You Win!
Memo
Pokémon Battle Support System
Tyranitar
Lv. 30
Description
Opponent Charizard had Solar Beam, need to be careful next time.

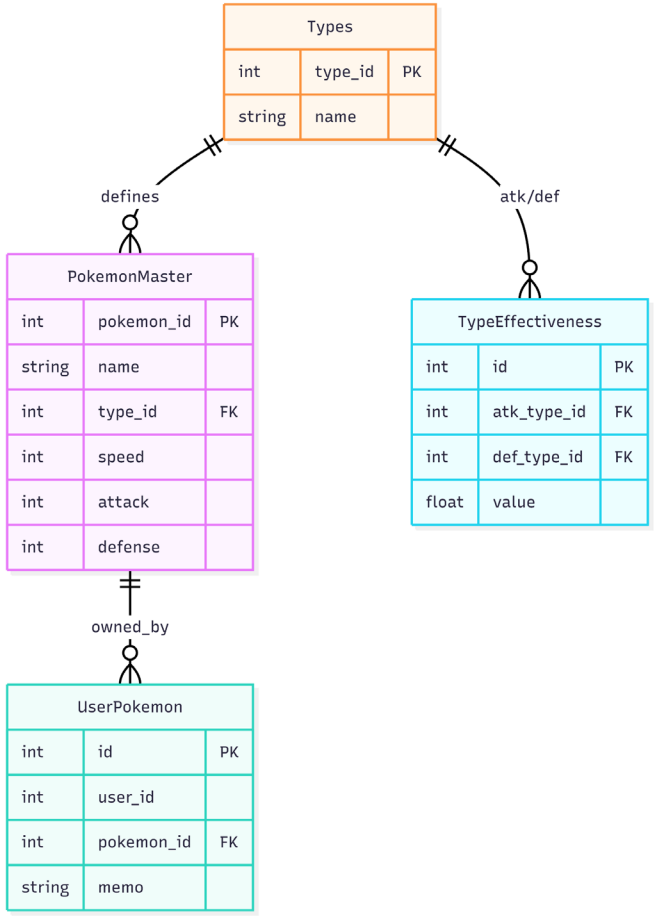
システム構成


クライアント層
プレゼンテーション層
アプリケーション層
データ層
Web Browser
Web Server (Render)
Gunicorn / Flask
App Server
Python (Flask)
ORM(SQLAlchemy)
Web Browser
PostgreSQL

非機能要件定義

項目	定義・目標値	実現方法
Availability (可用性)	99.9% (24H365D稼働)	Renderのクラウドインフラを利用し、常時稼働を実現
Performance (性能)	レスポンスタイム 200ms以内	インデックス最適化と軽量なクエリ設計（Outer Join活用）
RPO(目標復旧時点)	24時間	PostgreSQLの自動日次バックアップ機能を利用。データロストを最大1日分に抑える
RTO（目標復旧時間）	1時間以内	GitHub連携による自動デプロイ(CI/CD)。障害時はコード修正後、即座に再デプロイを行い復旧
DR (災害復旧)	リージョン障害対策	ソースコードはGitHub（別クラウド）で管理し、DBダンプがあれば別リージョンへ即時展開可能

データベース設計



Types：タイプ名が変わらないよう専用のテーブル

PokemonMater：ポケモンの情報
type_idが外部キー

UserPokemon：手持ちポケモン情報

TypeEffectiveness：相性計算用テーブル

感想・今後の展望

適切なDB設計（正規化）により、データの整合性と拡張性を確保できた

使用するサービスの選定に当初の予定よりも時間がかかった

タイプ相性・すばやさ以外の観点からも推奨ポケモンを解析する