インフラエンジニア

チームメンバー：渡邉諒（2442102）・田中誠真（2442054）

コンテナ型開発基盤の構築

概要:

アプリケーションの実行環境をイメージとしてコード化し、ホストOSに依存しないポータブルな開発環境を実現。

Dockerを用いたコンテナ型開発基盤の構築

インフラ構成のポイント:

OSの統一: 開発者全員が同じLinux環境で動作することを保証。

ライブラリ管理:

PythonやPostgreSQLのバージョンをコンテナ内で固定し、ライブラリの衝突（Dependency Hell）を防止。

メリット:

「自分の環境では動くが、他の環境では動かない」という問題を根本的に解決。

Docker Composeによるサービス連携の定義

役割:

docker-compose.yml を用いて、独立した2つのサービス（Web / DB）を定義し、一括で管理。

ネットワーク設計:

Isolated Network (隔離ネットワーク):

外部から直接DBコンテナへアクセスできないよう保護しつつ、Webコンテナからはdbというホスト名で通信可能な仮想ネットワークを構築。

環境変数の管理:

データベース名、ユーザー名、パスワード等の秘匿情報を環境変数として一元管理し、ソースコードからの分離を実現。

永続的データ管理 (Docker Volumes)

課題: コンテナ内のデータは、コンテナの破棄と共に消失してしまう。

解決策: Named Volumes (postgres_data) を実装。

詳細:ホスト側の領域とDBコンテナの /var/lib/postgresql/data をマウント。

結果:

開発中に docker-compose down を行っても、投稿した日報データや
ユーザーアカウントが維持される、信頼性の高いストレージ構成を確立。

正規化とリレーション設計

正規化への取り組み:

繰り返し項目（タグなど）や、重複する属性（カテゴリー名）を別テーブルに切り出し、第3正規形（3NF）まで正規化を実施。データの冗長性を排除。

多対多（Many-to-Many）の解決:

課題: 1つの日報に複数のタグがつき、1つのタグは複数の日報につく。

解決策: 中間テーブル Report_Tags を物理実装し、関連を管理。

これにより、タグごとの記事抽出や、記事ごとのタグ一覧取得を高速に行える設計とした。

データ整合性の担保 (Constraints)

実装した制約:

主キー制約 (Primary Key): 全テーブルにIDを設定し、行の一意性を保証。

外部キー制約 (Foreign Key):

Reports テーブルにおける user_id や category_id に設定。親データが存在しない子の登録を防ぎ、参照整合性を維持。

ユニーク制約 (Unique):

ユーザー名 (username) やタグ名 (name) の重複をデータベースレベルで禁止。

データ整合性の担保 (Constraints)

トランザクション:

記事投稿とタグ保存の処理において、原子性（Atomicity）を保証するためトランザクション制御を適用。