|画像アップロード| B["APIサーバー"] B -->|ジョブ送信| C["RabbitMQ"] C -->|ジョブ配信| D["Worker"] D -->|画像処理・アップロード| E["MinIO"]

処理の流れ

---

2. クラス仕様図（Mermaid 記法）

```
classDiagram
    class APIServer {
      +uploadHandler(w, r)
      +main()
      -amqpURL : string
      -queueName : string
    }
    class Worker {
      +main()
      +connectRabbitMQ()
      +processImage(msg)
      +uploadToMinIO(imageData)
      -amqpURL : string
      -queueName : string
      -minioEndpoint : string
      -minioAccess : string
      -minioSecret : string
      -minioBucket : string
    }
```

APIサーバーと Worker の主なメソッドとフィールド

---

3. 処理フロー図（Mermaid 記法）

```
sequenceDiagram
    participant U as ユーザー
    participant A as APIサーバー
    participant R as RabbitMQ
    participant W as Worker
    participant M as MinIO

    U->>A: 画像アップロード
    A->>U: 「画像受け付け」メッセージ
    A->>R: 画像データを送信
    R->>W: ジョブ配信
    W->>W: 画像処理（リサイズ・サムネイル生成）
    W->>M: 処理済み画像をアップロード
```

一連の流れ(シーケンス図)

5. 使用言語・フレームワーク等

- **プログラミング言語:**
  - Go (Golang)
- **フレームワーク／ライブラリ:**
  - APIサーバー: `net/http`（標準ライブラリ）
  - RabbitMQ クライアント: `github.com/streadway/amqp`
  - 画像処理: `github.com/disintegration/imaging`
  - MinIO クライアント: `github.com/minio/minio-go/v7`
- **インフラ／運用:**
  - コンテナ化: Docker
  - ローカル実行: Docker Compose
  - 本番環境: Kubernetes を利用したオーケストレーション（自動スケーリング、自己修復、ローリングアップデート）
- **その他ツール:**
  - MinIO: オブジェクトストレージサービス（Webコンソールで管理・確認可能）
  - RabbitMQ: メッセージブローカー（ジョブキュー管理用）

---

6. 補足

- **エラーハンドリングとリトライ:**
  - Worker コンポーネントでは、RabbitMQ 接続に失敗した場合のリトライロジックを実装することで、起動タイミングのズレによる接続拒否を回避。
- **将来的な拡張:**
  - APIサーバーに処理済み画像の一覧表示やギャラリー機能を実装し、ユーザーが結果を確認できるようにする。
  - 署名付き URL の生成により、セキュアにオブジェクトストレージ内の画像へアクセス可能とする。

-->

仕様

---

1. システム概要

本システムは、ユーザーがブラウザから画像をアップロードすると、バックグラウンドで画像のリサイズ、サムネイル生成などの処理を実施し、結果画像をオブジェクトストレージ（MinIO）に保存するサービスです。
各コンポーネントはマイクロサービスとして分離され、RabbitMQ を用いて非同期に連携することで、Kubernetes の自動スケーリングや自己修復機能を活かした運用が可能です。

1. アーキテクチャ図（Mermaid 記法）

```
graph LR
    A["ブラウザ （ユーザーUI)"] -->|画像アップロード| B["APIサーバー"]
    B -->|ジョブ送信| C["RabbitMQ"]
    C -->|ジョブ配信| D["Worker"]
    D -->|画像処理・アップロード| E["MinIO"]
```

処理の流れ

2. クラス仕様図（Mermaid 記法）

```
classDiagram
    class APIServer {
      +uploadHandler(w, r)
      +main()
      -amqpURL : string
      -queueName : string
    }
    class Worker {
      +main()
      +connectRabbitMQ()
      +processImage(msg)
      +uploadToMinIO(imageData)
      -amqpURL : string
      -queueName : string
      -minioEndpoint : string
      -minioAccess : string
      -minioSecret : string
      -minioBucket : string
    }
```

APIサーバーと Worker の主なメソッドとフィールド

3. 処理フロー図（Mermaid 記法）

```
sequenceDiagram
    participant U as ユーザー
    participant A as APIサーバー
    participant R as RabbitMQ
    participant W as Worker
    participant M as MinIO

    U->>A: 画像アップロード
```

```
    A->>U: 「画像受け付け」メッセージ
    A->>R: 画像データを送信
    R->>W: ジョブ配信
    W->>W: 画像処理（リサイズ・サムネイル生成）
    W->>M: 処理済み画像をアップロード
```

一連の流れ(シーケンス図)

---

5. 使用言語・フレームワーク等

- **プログラミング言語:**
  - Go (Golang)
- **フレームワーク／ライブラリ:**
  - APIサーバー: `net/http`（標準ライブラリ）
  - RabbitMQ クライアント: `github.com/streadway/amqp`
  - 画像処理: `github.com/disintegration/imaging`
  - MinIO クライアント: `github.com/minio/minio-go/v7`
- **インフラ／運用:**
  - コンテナ化: Docker
  - ローカル実行: Docker Compose
  - 本番環境: Kubernetes を利用したオーケストレーション（自動スケーリング、自己修復、ローリングアップデート）
- **その他ツール:**
  - MinIO: オブジェクトストレージサービス（Webコンソールで管理・確認可能）
  - RabbitMQ: メッセージブローカー（ジョブキュー管理用）

---

6. 補足

- **エラーハンドリングとリトライ:**
  - Worker コンポーネントでは、RabbitMQ 接続に失敗した場合のリトライロジックを実装することで、起動タイミングのズレによる接続拒否を回避。
- **将来的な拡張:**
  - APIサーバーに処理済み画像の一覧表示やギャラリー機能を実装し、ユーザーが結果を確認できるようにする。
  - 署名付き URL の生成により、セキュアにオブジェクトストレージ内の画像へアクセス可能とする。
