

詳細仕様書

1. システム概要

1.1. このシステムの目的

このシステムは、ユーザーが自宅でフィットネスを行うのを助けるデスクトップアプリケーションです。ユーザーは、様々なエクササイズガイドを受けながら、リアルタイムでポーズの正確性をチェックし、その運動履歴を記録・管理することができます。健康的なライフスタイルをサポートし、自宅での運動習慣を促進することを目的としています。

1.2. 主な機能の説明

このアプリケーションは、以下の主要な機能を提供します。

- **ユーザー認証・管理:** ユーザーは安全にログイン、新規アカウント登録、パスワード変更、および個人プロフィール情報の編集が可能です。
- **エクササイズ情報提供:** 豊富なエクササイズリストから、各エクササイズの詳細な説明、画像、動画URL、効果、注意点などを参照できます。
- **リアルタイムポーズ検出・フィードバック:** カメラ映像を通じてユーザーのポーズをリアルタイムで検出し、正しいポーズとの比較に基づいて視覚的および音声によるフィードバックを提供します。
- **運動履歴記録・表示:** 実施したエクササイズの期間、消費カロリー、達成状況などを自動的に記録し、ユーザーはいつでも過去の運動履歴を確認できます。
- **参照ポーズ記録:** 新しいエクササイズやカスタムエクササイズのために、ユーザー自身のポーズを「正しいポーズ」としてシステムに登録する機能です。

2. 画面仕様

2.1. 画面一覧

このシステムは、以下の9つの主要な画面で構成されています。

- ログイン画面 (`LoginWindow`)
- 新規登録画面 (`RegisterWindow`)
- メイン画面 (エクササイズ一覧) (`VirtualFitnessCoachWindow`)
- エクササイズ詳細画面 (`ExerciseDetailWindow`)
- ポーズ検出画面 (`PoseDetectionWindow`)
- エクササイズ履歴画面 (`ExerciseHistoryWindow`)
- マイページ画面 (`MyPageWindow`)
- プロファイル編集画面 (`ProfileEditWindow`)
- パスワード変更画面 (`PasswordChangeWindow`)

2.2. 各画面の説明

2.2.1. ログイン画面 (`LoginWindow`)

- **目的:** ユーザーがシステムにアクセスするための認証を行います。

- **説明:** ユーザー名とパスワードの入力フィールド、ログインボタン、新規登録画面への遷移ボタン、ゲストログインボタンが配置されています。
- **UI要素:**
 - `Gtk::Entry` (ユーザー名入力)
 - `Gtk::Entry` (パスワード入力, `set_visibility(false)` で非表示)
 - `Gtk::Button` (ログイン)
 - `Gtk::Button` (新規登録)
 - `Gtk::Button` (ゲストログイン)
 - `Gtk::Label` (エラーメッセージ表示用)
- **操作フロー:**
 1. ユーザーがユーザー名とパスワードを`Gtk::Entry`に入力します。
 2. 「ログイン」ボタンをクリック (`signal_clicked`) します。
 3. システムは入力値を`AuthManager::authenticate_user`メソッドに渡し、認証を試みます。
 4. 認証が成功した場合、`VirtualFitnessCoachWindow` (メイン画面) へ遷移します。
 5. 認証が失敗した場合、`Gtk::Label`にエラーメッセージ (例: 「ユーザー名またはパスワードが違います」) を表示します。
 6. 「新規登録」ボタンをクリックすると、`RegisterWindow`へ遷移します。
 7. 「ゲストログイン」ボタンをクリックすると、認証なしで`VirtualFitnessCoachWindow`へ遷移します (機能未実装)。

- スクリーンショット挿入場所:



バーチャルエクササイズコーチ...

ログイン

ユーザー名:

パスワード:

☒ IDとパスワードを記憶する

ログイン

新規登録

ゲストとして続行

2.2.2. 新規登録画面 (**RegisterWindow**)

- **目的:** 新しいユーザーアカウントを作成します。
- **説明:** ユーザー名、メールアドレス、パスワード、パスワード確認の入力フィールド、登録ボタン、戻るボタンが配置されています。
- **UI要素:**
 - **Gtk::Entry** (ユーザー名入力)
 - **Gtk::Entry** (メールアドレス入力)
 - **Gtk::Entry** (パスワード入力, **set_visibility(false)**)
 - **Gtk::Entry** (パスワード確認入力, **set_visibility(false)**)
 - **Gtk::Button** (登録)
 - **Gtk::Button** (戻る)
 - **Gtk::Label** (エラーメッセージ表示用)
- **操作フロー:**
 1. ユーザーが各**Gtk::Entry**に情報を入力します。
 2. 「登録」ボタンをクリック (**signal_clicked**) します。
 3. システムは入力値のバリデーション（例: パスワードの一致、メールアドレス形式）を行います。
 4. **AuthManager::register_user**メソッドを呼び出し、ユーザー登録を試みます。
 5. 登録が成功した場合、**LoginWindow**へ戻ります。
 6. 登録が失敗した場合、**Gtk::Label**にエラーメッセージ（例: 「ユーザー名またはメールアドレスはすでに使われています」）を表示します。
 7. 「戻る」ボタンをクリックすると、**LoginWindow**へ戻ります。

- スクリーンショット挿入場所:

2.2.3. メイン画面（バーチャルフィットネスコーチ画面）（`VirtualFitnessCoachWindow`）

- **目的:** エクササイズの一覧を表示し、ユーザーがエクササイズを選択できるようにします。
- **説明:** エクササイズがリスト形式で表示され、各エクササイズには名前と画像が含まれます。上部にはマイページ、履歴、ログアウトへのナビゲーションボタンがあります。
- **UI要素:**
 - `Gtk::ListBox` (エクササイズリスト表示)
 - `Gtk::Button` (マイページ)
 - `Gtk::Button` (履歴)
 - `Gtk::Button` (ログアウト)
 - `Gtk::Image` (エクササイズ画像)
 - `Gtk::Label` (エクササイズ名)
- **操作フロー:**
 1. 画面表示時 (`on_show`) に、`DatabaseManager::fetch_exercises`メソッドを呼び出し、データベースからエクササイズデータを取得します。

2. 取得したエクササイズデータは`Gtk::ListBox`に表示されます。各行は`Exercise`オブジェクトに対応します。
3. ユーザーがリスト内のエクササイズを選択し、クリック (`signal_row_activated`) すると、選択されたエクササイズのIDを`ExerciseDetailWindow`に渡し、詳細画面へ遷移します。
4. 「マイページ」ボタンをクリックすると、`MyPageWindow`へ遷移します。
5. 「履歴」ボタンをクリックすると、`ExerciseHistoryWindow`へ遷移します。
6. 「ログアウト」ボタンをクリックすると、`LoginWindow`へ戻ります。

- スクリーンショット挿入場所:



2.2.4. エクササイズ詳細画面 (`ExerciseDetailWindow`)

- **目的:** 選択されたエクササイズの詳細情報を提供し、ポーズ検出を開始する入り口となります。
- **説明:** エクササイズ名、説明、指示、動画URL、画像、消費カロリー目安などが表示されます。「ポーズ検出開始」ボタンと「戻る」ボタンがあります。
- **UI要素:**

- `Gtk::Label` (エクササイズ名、説明、指示、ヒント、効果、間違い、バリエーション、消費カロリー目安など)
- `Gtk::Image` (エクササイズ画像)
- `Gtk::LinkButton` (動画URL)
- `Gtk::Button` (ポーズ検出開始)
- `Gtk::Button` (戻る)
- 操作フロー:
 1. メイン画面から渡されたエクササイズIDに基づいて、`DatabaseManager::fetch_exercises`を呼び出し、詳細なエクササイズ情報を取得します。
 2. 取得した情報を各`Gtk::Label`や`Gtk::Image`に表示します。
 3. 「ポーズ検出開始」ボタンをクリック (`signal_clicked`) します。
 4. システムは`DatabaseManager::insert_exercise_session_start`を呼び出し、新しい運動セッションをデータベースに記録し、`exercise_history_id`を取得します。
 5. 取得した`exercise_history_id`と`Exercise`オブジェクトを`PoseDetectionWindow`に渡し、ポーズ検出画面へ遷移します。
 6. 「戻る」ボタンをクリックすると、`VirtualFitnessCoachWindow`へ戻ります。

- スクリーンショット挿入場所:



2.2.5. ポーズ検出画面 (PoseDetectionWindow)

- **目的:** ユーザーのポーズをリアルタイムで検出し、フィードバックを提供し、参照ポーズを記録します。
- **説明:** カメラからのライブ映像が`Gtk::DrawingArea`に表示され、その上に検出された関節の骨格とフィードバックメッセージがオーバーレイされます。カウントダウン表示、記録ボタン、終了ボタンがあります。
- **UI要素:**
 - `Gtk::Label` (タイトル、ステータスメッセージ)
 - `Gtk::DrawingArea` (カメラ映像とポーズ検出結果の描画)
 - `Gtk::Button` (参照ポーズを記録)
 - `Gtk::Button` (閉じる/終了)
- **操作フロー:**
 1. 画面表示時 (`on_realize`) に、`start_camera`メソッドが呼ばれ、カメラ (`cv::VideoCapture`) が起動します。

2. `Glib::signal_timeout().connect`で約30FPS (33ms) ごとに`update_frame`メソッドが呼び出されます。
3. `update_frame`内で、カメラからフレームを取得し、TensorFlow Liteモデル (`tf_lite_interpreter`) を使ってポーズ検出を行います。
4. 検出されたキーポイント (`last_detected_keypoints`) は、データベースから取得した参照ポーズ (`DatabaseManager::fetch_reference_pose`) と比較されます。
5. 比較結果に基づき、`error_message_queue`にエラーメッセージが追加され、`status_label`に表示されます。音声フィードバック (`system("say -v Kyoko ...")`) も提供されます。
6. ポーズが正しい状態が続くと、`m_correct_pose_duration_ms`が加算されます。
7. 「参照ポーズを記録」ボタンをクリックすると、5秒のカウントダウン (`on_countdown_timer`) 後に3秒間のポーズ記録 (`on_recording_timer`) が行われます。記録されたポーズの平均値が `DatabaseManager::insert_reference_pose`を使ってデータベースに保存されます。
8. 「閉じる」ボタンをクリックすると、カメラが停止 (`stop_camera`) し、画面が閉じます。この際、`m_correct_pose_duration_ms`に基づいて最終的な消費カロリーが計算され、`DatabaseManager::update_exercise_session_end`で運動履歴が更新されます。

- スクリーンショット挿入場所:  ポーズ検出画面

2.2.6. エクササイズ履歴画面 (`ExerciseHistoryWindow`)

- 目的: ユーザーの過去のエクササイズ記録を一覧で表示します。
- 説明: 実施日時、エクササイズ名、期間、消費カロリー、ステータス (例: 実行、試行) などが `Gtk::ListBox`に表示されます。
- UI要素:
 - `Gtk::ListBox` (エクササイズ履歴リスト)
 - `Gtk::Label` (各履歴項目の詳細)
 - `Gtk::Button` (戻る)
- 操作フロー:
 1. 画面表示時 (`on_show`) に、`DatabaseManager::fetch_exercise_history`メソッドを呼び出し、現在のユーザーの運動履歴を取得します。
 2. 取得した履歴データは`Gtk::ListBox`に表示されます。
 3. 「戻る」ボタンをクリックすると、`MyPageWindow`または`VirtualFitnessCoachWindow`へ戻ります。

- スクリーンショット挿入場所:



2.2.7. マイページ画面 (MyPageWindow)

- **目的:** ユーザーの個人情報と運動統計を表示し、関連する設定画面へナビゲートします。
- **説明:** ユーザー名、メールアドレス、総運動時間、総消費カロリーなどの情報が表示されます。プロフィール編集、パスワード変更、履歴表示、ログアウトへのボタンがあります。
- **UI要素:**
 - `Gtk::Label` (ユーザー名、メールアドレス、統計情報など)
 - `Gtk::Button` (プロフィール編集)
 - `Gtk::Button` (パスワード変更)
 - `Gtk::Button` (履歴を見る)
 - `Gtk::Button` (ログアウト)
 - `Gtk::Button` (戻る)
- **操作フロー:**
 1. 画面表示時 (`on_show`) に、`DatabaseManager` からユーザー情報と運動統計 (例: `fetch_statistics`) を取得し、`Gtk::Label` に表示します。
 2. 「プロフィール編集」 ボタンをクリックすると、`ProfileEditWindow` へ遷移します。
 3. 「パスワード変更」 ボタンをクリックすると、`PasswordChangeWindow` へ遷移します。
 4. 「履歴を見る」 ボタンをクリックすると、`ExerciseHistoryWindow` へ遷移します。
 5. 「ログアウト」 ボタンをクリックすると、`LoginWindow` へ戻ります。
 6. 「戻る」 ボタンをクリックすると、`VirtualFitnessCoachWindow` へ戻ります。

- スクリーンショット挿入場所:



2.2.8. プロファイル編集画面 (ProfileEditWindow)

- **目的:** ユーザーのプロファイル情報（身長、体重、目標など）を更新します。
- **説明:** 身長、体重、フィットネスレベル、目標などの入力フィールド、保存ボタン、キャンセルボタンが配置されています。
- **UI要素:**
 - `Gtk::Entry` (身長、体重などの入力)
 - `Gtk::ComboBoxText` (フィットネスレベル選択)
 - `Gtk::TextView` (目標入力)
 - `Gtk::Button` (保存)
 - `Gtk::Button` (キャンセル)
 - `Gtk::Label` (エラーメッセージ表示用)
- **操作フロー:**
 1. 画面表示時 (`on_show`) に、現在のユーザープロファイル情報 (`DatabaseManager::fetch_user_profile`) を取得し、各UI要素に表示します。
 2. ユーザーが情報を編集し、「保存」ボタンをクリック (`signal_clicked`) します。
 3. システムは入力値のバリデーションを行い、`DatabaseManager::update_user_profile`メソッドを呼び出してデータベースを更新します。
 4. 更新が成功した場合、`MyPageWindow`へ戻ります。
 5. 更新が失敗した場合、`Gtk::Label`にエラーメッセージを表示します。
 6. 「キャンセル」ボタンをクリックすると、変更を破棄して`MyPageWindow`へ戻ります。



- **スクリーンショット挿入場所:**

2.2.9. パスワード変更画面 (`PasswordChangeWindow`)

- **目的:** ユーザーのパスワードを安全に変更します。
- **説明:** 現在のパスワード、新しいパスワード、新しいパスワード確認の入力フィールド、変更ボタン、キャンセルボタンが配置されています。
- **UI要素:**
 - `Gtk::Entry` (現在のパスワード入力, `set_visibility(false)`)
 - `Gtk::Entry` (新しいパスワード入力, `set_visibility(false)`)
 - `Gtk::Entry` (新しいパスワード確認入力, `set_visibility(false)`)
 - `Gtk::Button` (変更)

- `Gtk::Button` (キャンセル)
- `Gtk::Label` (エラーメッセージ表示用)
- 操作フロー:
 1. ユーザーが各`Gtk::Entry`にパスワードを入力します。
 2. 「変更」ボタンをクリック (`signal_clicked`) します。
 3. システムは入力値のバリデーション（例: 新しいパスワードの一致）を行い、`AuthManager::change_password`メソッドを呼び出してパスワード変更を試みます。
 4. 変更が成功した場合、`MyPageWindow`へ戻ります。
 5. 変更が失敗した場合、`Gtk::Label`にエラーメッセージを表示します。
 6. 「キャンセル」ボタンをクリックすると、変更を破棄して`MyPageWindow`へ戻ります。



- スクリーンショット挿入場所:

3. 機能仕様

3.1. 機能一覧

- ユーザー認証機能 (`AuthManager`)
- ユーザープロフィール管理機能 (`ProfileEditWindow`, `DatabaseManager`)
- エクササイズ情報表示機能 (`VirtualFitnessCoachWindow`, `ExerciseDetailWindow`, `DatabaseManager`)
- ポーズ検出・フィードバック機能 (`PoseDetectionWindow`, TensorFlow Lite)
- エクササイズ履歴記録・表示機能 (`ExerciseHistoryWindow`, `DatabaseManager`)
- 参照ポーズ記録機能 (`PoseDetectionWindow`, `DatabaseManager`)

3.2. 各機能の詳細処理

3.2.1. ユーザー認証機能 (`AuthManager`)

- ログイン:
 - 入力: `username` (`Glib::ustring`), `password` (`Glib::ustring`)
 - 処理:
 1. `DatabaseManager::fetch_users`を呼び出し、入力された`username`に一致するユーザーをデータベースから取得します。
 2. ユーザーが見つかった場合、`AuthManager::verify_password`メソッドを使って、入力された`password`とデータベースに保存されている`password_hash`を比較します。
 3. パスワードが一致すれば、ユーザーIDを返します。

4. 一致しない場合やユーザーが見つからない場合は、認証失敗として処理します。

- 出力: ユーザーID (int) または認証失敗を示す値

- 新規登録:

- 入力: `username` (Glib::ustring), `email` (Glib::ustring), `password` (Glib::ustring)
- 処理:
 1. 入力された`username`と`email`がデータベースにすでに存在しないか、`DatabaseManager::fetch_users`で確認します。
 2. `AuthManager::hash_password`メソッドを使って、入力された`password`をハッシュ化します。
 3. `DatabaseManager::execute_query`を使って、新しいユーザー情報を`users`テーブルに挿入します。
- 出力: 登録成功/失敗 (bool)

- パスワード変更:

- 入力: `user_id` (int), `old_password` (Glib::ustring), `new_password` (Glib::ustring)
- 処理:
 1. `DatabaseManager::fetch_users`で`user_id`のユーザー情報を取得し、`AuthManager::verify_password`で`old_password`が正しいか確認します。
 2. `AuthManager::hash_password`で`new_password`をハッシュ化します。
 3. `DatabaseManager::execute_query`を使って、`users`テーブルの`password_hash`を更新します。
- 出力: 変更成功/失敗 (bool)

3.2.2. ユーザープロフィール管理機能 (`ProfileEditWindow`, `DatabaseManager`)

- プロファイル編集:

- 入力: `User`オブジェクト (更新されたプロフィール情報を含む)
- 処理:
 1. `ProfileEditWindow`でユーザーが入力した身長、体重、フィットネスレベル、目標などの情報を`User`オブジェクトに設定します。
 2. `DatabaseManager::update_user_profile`メソッドを呼び出し、`user_profiles`テーブルの対応するレコードを更新します。
- 出力: 更新成功/失敗 (bool)

3.2.3. エクササイズ情報表示機能 (`VirtualFitnessCoachWindow`, `ExerciseDetailWindow`, `DatabaseManager`)

- エクササイズ一覧表示:

- 入力: なし
- 処理:
 1. `VirtualFitnessCoachWindow`の初期化時または表示時に、`DatabaseManager::fetch_exercises`メソッドを呼び出し、`exercises`テーブルからすべてのエクササイズ情報を`std::vector<Exercise>`として取得します。
 2. 取得した`Exercise`オブジェクトのリストを`Gtk::ListBox`にバインドし、各エクササイズの`name`と`image_path`を表示します。
- 出力: `std::vector<Exercise>` (エクササイズオブジェクトのリスト)

- エクササイズ詳細表示:

- 入力: `exercise_id` (int)

- **処理:**

1. `ExerciseDetailWindow`は、渡された`exercise_id`を使って
`DatabaseManager::fetch_exercises`を呼び出し、特定のエクササイズ情報を取得します。
2. 取得した`Exercise`オブジェクトの各プロパティ (`name`, `instructions`, `video_url`, `calories_burned_estimate`など) を画面上の対応する`Gtk::Label`や`Gtk::Image`に表示します。

- **出力:** `Exercise`オブジェクトの詳細情報

3.2.4. ポーズ検出・フィードバック機能 (`PoseDetectionWindow`, `TensorFlow Lite`)

- **入力:** カメラからのリアルタイム映像 (`cv::Mat`フレーム)

- **処理:**

1. `PoseDetectionWindow::update_frame`メソッドが約33msごとに実行されます。
2. `cv::VideoCapture`から現在のフレームを取得し、`TensorFlow Lite`モデル (`movenet_singlepose_lightning.tflite`) の入力形式に合わせて前処理 (リサイズ、正規化) を行います。
3. `tflite_interpreter->Invoke()`を呼び出し、ポーズ検出モデルを実行します。
4. モデルの出力 (17個のキーポイントの座標と信頼度) を取得し、`last_detected_keypoints`に保存します。
5. `DatabaseManager::fetch_reference_pose`を呼び出し、現在エクササイズ中の参照ポーズデータを取得します。
6. `last_detected_keypoints`と参照ポーズの各関節の距離を計算し、`JOINT_ERROR_THRESHOLD`と比較します。
7. 閾値を超えた関節がある場合、`error_message_queue`にエラーメッセージ (例: 「左肩を調整してください」) を追加し、`m_is_pose_correct`を`false`に設定します。
8. `error_message_queue`が空の場合 (すべての関節が正しい場合)、`m_is_pose_correct`を`true`に設定し、`m_correct_pose_duration_ms`を33ms加算します。
9. `Gtk::DrawingArea`にカメラ映像、検出された骨格、およびフィードバックメッセージを描画します。
10. `m_is_pose_correct`が`REQUIRED_CORRECT_FRAMES`以上連続した場合、`DatabaseManager::update_exercise_session_end`を呼び出し、`performed_seconds` (`m_correct_pose_duration_ms`を秒に変換したもの) と計算された`calories_burned`で運動履歴を更新します。

- **出力:** `cv::Mat` (骨格描画済みフレーム), `Gtk::Label` (フィードバックメッセージ), 音声フィードバック

3.2.5. エクササイズ履歴記録・表示機能 (`ExerciseHistoryWindow`, `DatabaseManager`)

- **履歴記録:**

- **入力:** `user_id` (int), `exercise_id` (int), `correct_performed_seconds` (int), `calories_burned` (int), `duration_minutes` (int), `status` (std::string)

- **処理:**

1. エクササイズ開始時、`ExerciseDetailWindow`から
`DatabaseManager::insert_exercise_session_start`を呼び出し、`exercise_history`テーブルに初期レコードを挿入し、`m_exercise_history_id`を取得します。

2. エクササイズ終了時 (`PoseDetectionWindow::on_hide`) または正しいポーズが一定時間維持された場合、`DatabaseManager::update_exercise_session_end`を呼び出し、`m_exercise_history_id`を使って`exercise_history`テーブルのレコードを更新します。
 - 出力: なし
- 履歴表示:
 - 入力: `user_id` (int)
 - 処理:
 1. `ExerciseHistoryWindow`の表示時 (`on_show`) に、`DatabaseManager::fetch_exercise_history`メソッドを呼び出し、`user_id`に紐づくすべての運動履歴を`std::vector<std::map<std::string, std::string>>`として取得します。
 2. 取得した履歴データを`Gtk::ListBox`に表示します。
 - 出力: `std::vector<std::map<std::string, std::string>>` (運動履歴のリスト)

3.2.6. 参照ポーズ記録機能 (`PoseDetectionWindow`, `DatabaseManager`)

- 入力: カメラからのリアルタイム映像 (ユーザーのポーズ)
- 処理:
 1. `PoseDetectionWindow`で「参照ポーズを記録」ボタンがクリックされると、`record_reference_pose`メソッドが呼び出されます。
 2. 既存の参照ポーズがあるか`DatabaseManager::fetch_statistics`で確認し、ユーザーに上書きの確認ダイアログ (`Gtk::MessageDialog`) を表示します。
 3. ユーザーが記録を続行すると、5秒のカウントダウン (`on_countdown_timer`) が開始されます。
 4. カウントダウン後、`on_recording_timer`が3秒間、約100msごとに実行され、`last_detected_keypoints`を`recorded_poses`に複数回保存します。
 5. 記録期間終了後、`recorded_poses`の中央のポーズを代表として選択します。
 6. `DatabaseManager::execute_query`で既存の参照ポーズを削除し、`DatabaseManager::insert_reference_pose`を使って新しい参照ポーズの各関節データ (`x`, `y`, `confidence`) を`reference_poses`テーブルに保存します。
- 出力: 記録成功/失敗メッセージ (`Gtk::Label`)

4. データ仕様

4.1. 使用するデータベース

- PostgreSQL (バージョン14を想定)

4.2. テーブル仕様

4.2.1. `users` テーブル

- 説明: システムに登録されたユーザーの基本情報と認証情報を管理します。
- データ項目:
 - `id`: SERIAL (主キー) - ユーザーを一意に識別する自動採番の整数ID。
 - `username`: VARCHAR(50) (ユニーク, 必須) - ユーザーがログインに使用する名前。50文字まで。
 - `email`: VARCHAR(100) (ユニーク, 必須) - ユーザーのメールアドレス。100文字まで。

- **password_hash**: VARCHAR(255) (必須) - パスワードのハッシュ値。セキュリティのため、元のパスワードは保存しません。
- **first_name**: VARCHAR(50) - ユーザーの名。50文字まで。
- **last_name**: VARCHAR(50) - ユーザーの姓。50文字まで。
- **created_at**: TIMESTAMP (デフォルト: 現在時刻) - アカウントが作成された日時。
- **updated_at**: TIMESTAMP (デフォルト: 現在時刻) - ユーザー情報が最後に更新された日時。
- **last_login**: TIMESTAMP - ユーザーが最後にログインした日時。
- **is_active**: BOOLEAN (デフォルト: TRUE) - アカウントが現在有効であることを示すフラグ。

4.2.2. **user_sessions** テーブル

- **説明**: ユーザーのログインセッション情報を管理し、ログイン状態の維持とセキュリティを確保します。
- **データ項目**:
 - **id**: SERIAL (主キー) - セッションを一意に識別する自動採番の整数ID。
 - **user_id**: INTEGER (外部キー: **users.id**, 必須) - このセッションが属するユーザーのID。 **users** テーブルの **id** を参照します。
 - **token_hash**: VARCHAR(255) (必須) - 認証トークンのハッシュ値。
 - **expires_at**: TIMESTAMP (必須) - セッションの有効期限。この日時を過ぎるとセッションは無効になります。
 - **created_at**: TIMESTAMP (デフォルト: 現在時刻) - セッションが作成された日時。
 - **is_valid**: BOOLEAN (デフォルト: TRUE) - セッションが現在有効であることを示すフラグ。

4.2.3. **user_profiles** テーブル

- **説明**: ユーザーの身体情報やフィットネスに関する詳細なプロフィール情報を管理します。
- **データ項目**:
 - **id**: SERIAL (主キー) - プロファイルを一意に識別する自動採番の整数ID。
 - **user_id**: INTEGER (外部キー: **users.id**, ユニーク, 必須) - このプロフィールが属するユーザーのID。 **users** テーブルの **id** を参照します。各ユーザーは1つのプロフィールのみ持ちます。
 - **age**: INTEGER - ユーザーの年齢。
 - **height_cm**: INTEGER - ユーザーの身長（センチメートル）。
 - **weight_kg**: DECIMAL(5,2) - ユーザーの体重（キログラム）。小数点以下2桁まで。
 - **fitness_level**: VARCHAR(20) (デフォルト: 'beginner') - ユーザーのフィットネスレベル（例: 'beginner', 'intermediate', 'advanced'）。
 - **goals**: TEXT - ユーザーの運動目標（自由記述）。
 - **preferences**: TEXT - ユーザーの運動に関する好み（自由記述）。
 - **created_at**: TIMESTAMP (デフォルト: 現在時刻) - プロファイルが作成された日時。
 - **updated_at**: TIMESTAMP (デフォルト: 現在時刻) - プロファイル情報が最後に更新された日時。

4.2.4. **exercises** テーブル

- **説明**: アプリケーションで提供される各エクササイズの詳細情報を管理します。
- **データ項目**:
 - **id**: SERIAL (主キー) - エクササイズを一意に識別する自動採番の整数ID。
 - **name**: VARCHAR(255) (必須) - エクササイズの名前。255文字まで。
 - **image_path**: VARCHAR(255) - エクササイズに関連する画像ファイルのパス。255文字まで。

- **category**: VARCHAR(100) - エクササイズのカテゴリ（例: 'Yoga', 'Strength Training'）。100文字まで。
- **primary_muscle**: VARCHAR(100) - 主に鍛えられる筋肉。100文字まで。
- **secondary_muscles**: TEXT - 補助的に鍛えられる筋肉（複数可）。
- **equipment**: TEXT - エクササイズに必要な器具。
- **difficulty_level**: VARCHAR(50) - エクササイズの難易度（例: 'Easy', 'Medium', 'Hard'）。50文字まで。
- **instructions**: TEXT - エクササイズの詳細な手順。
- **tips**: TEXT - エクササイズを行う上でのヒントやコツ。
- **video_url**: VARCHAR(255) - エクササイズの説明動画へのURL。255文字まで。
- **reps_sets_suggestion**: VARCHAR(255) - 推奨される回数やセット数。255文字まで。
- **benefits**: TEXT - エクササイズから得られる効果。
- **common_mistakes**: TEXT - エクササイズ中によくある間違い。
- **variations**: TEXT - エクササイズのバリエーション。
- **calories_burned_estimate**: INTEGER - 1分間あたりに消費される推定カロリー（整数値）。
- **created_at**: TIMESTAMP (デフォルト: 現在時刻) - エクササイズ情報が作成された日時。
- **updated_at**: TIMESTAMP (デフォルト: 現在時刻) - エクササイズ情報が最後に更新された日時。

4.2.5. exercise_history テーブル

- **説明**: 各ユーザーが行ったエクササイズの履歴を記録します。
- **データ項目**:
 - **id**: SERIAL (主キー) - 運動履歴を一意に識別する自動採番の整数ID。
 - **user_id**: INTEGER (外部キー: **users.id**, 必須) - 運動を行ったユーザーのID。
 - **exercise_id**: INTEGER (外部キー: **exercises.id**, 必須) - 行われたエクササイズのID。
 - **session_date**: TIMESTAMP (デフォルト: 現在時刻) - 運動セッションが開始された日時。
 - **duration_minutes**: INTEGER - 運動セッションの合計期間（分）。
 - **calories_burned**: INTEGER - このセッションで消費された推定カロリー（整数値）。
 - **notes**: TEXT - 運動に関するユーザーのメモ。
 - **status**: VARCHAR(50) (デフォルト: '確認') - 運動セッションの達成状況（例: '確認', '試行', '実行'）。
 - **performed_seconds**: INTEGER - 正しいポーズを維持した合計時間（秒）。この値に基づいてカロリーが計算されます。


4.2.6. reference_poses テーブル

- **説明**: 各エクササイズにおける「正しいポーズ」の基準となる関節の座標と信頼度を保存します。ポーズ検出の比較対象として使用されます。
- **データ項目**:
 - **id**: SERIAL (主キー) - 参照ポーズデータを一意に識別する自動採番の整数ID。
 - **exercise_id**: INTEGER (外部キー: **exercises.id**, 必須) - この参照ポーズが属するエクササイズのID。
 - **keypoint_index**: INTEGER (必須) - 検出された関節のインデックス（例: 0:鼻, 1:左目, ..., 16:右足首）。
 - **x**: REAL (必須) - 関節のX座標。カメラ映像の幅に対する正規化された値（0.0～1.0）。
 - **y**: REAL (必須) - 関節のY座標。カメラ映像の高さに対する正規化された値（0.0～1.0）。
 - **confidence**: REAL (必須) - 検出された関節の信頼度（0.0～1.0）。

- **frame_number**: INTEGER (デフォルト: 0) - 参照ポーズが記録されたフレーム番号。静止ポーズの場合は通常0。

5. 処理フロー


5.1. ログイン処理フロー

- **説明**: ユーザーがアプリケーションにログインする際の処理の流れを示します。
- **フローチャート挿入場所**:  ログインフロー

```
sequenceDiagram
    actor User
    participant LoginWindow
    participant AuthManager
    participant DatabaseManager
    participant VirtualFitnessCoachWindow

    User->>LoginWindow: ユーザー名とパスワードを入力し、ログインボタンをクリック
    LoginWindow->>AuthManager: authenticate_user(username, password)を呼び出す
    AuthManager->>DatabaseManager: fetch_users(username)でユーザー情報を取得
    DatabaseManager-->>AuthManager: ユーザー情報 (password_hashなど)
    AuthManager->>AuthManager: 入力パスワードとpassword_hashを比較 (verify_password)
    alt 認証成功
        AuthManager-->>LoginWindow: ユーザーIDを返す
        LoginWindow->>VirtualFitnessCoachWindow: メイン画面を表示
    else 認証失敗
        AuthManager-->>LoginWindow: 認証失敗を通知
        LoginWindow->>LoginWindow: エラーメッセージを表示
    end
end
```

5.2. ポーズ検出・記録処理フロー

- **説明**: ユーザーがエクササイズを開始し、ポーズ検出と運動履歴の記録が行われる際の処理の流れを示します。
- **シーケンス図挿入場所**:  ポーズ検出シーケンス

```
sequenceDiagram
    actor User
    participant ExerciseDetailWindow
    participant PoseDetectionWindow
    participant Camera
    participant TensorFlowLiteModel
    participant DatabaseManager

    User->>ExerciseDetailWindow: エクササイズを選択し、「ポーズ検出開始」をク
```

```

リック
    ExerciseDetailWindow->>DatabaseManager:
insert_exercise_session_start(user_id, exercise_id)
    DatabaseManager-->>ExerciseDetailWindow: exercise_history_idを返す
    ExerciseDetailWindow->>PoseDetectionWindow:
PoseDetectionWindow(exercise, exercise_history_id)を生成・表示

    loop 約33msごと (update_frame)
        PoseDetectionWindow->>Camera: フレームを取得 (cap >> frame)
        Camera-->>PoseDetectionWindow: cv::Matフレーム
        PoseDetectionWindow->>TensorFlowLiteModel: フレームを渡し、ポーズ
を検出 (tflite_interpreter->Invoke())
        TensorFlowLiteModel-->>PoseDetectionWindow: キーポイントデータ
(last_detected_keypoints)
        PoseDetectionWindow->>DatabaseManager:
fetch_reference_pose(exercise_id)で参照ポーズを取得
        DatabaseManager-->>PoseDetectionWindow: 参照ポーズデータ
        PoseDetectionWindow->>PoseDetectionWindow: 検出ポーズと参照ポーズ
を比較し、フィードバックを生成
        alt ポーズが正しい
            PoseDetectionWindow->>PoseDetectionWindow:
m_correct_pose_duration_msを更新
        else ポーズが正しくない
            PoseDetectionWindow->>PoseDetectionWindow: エラーメッセージを
表示・音声再生
        end
        PoseDetectionWindow->>PoseDetectionWindow: 画面に映像とフィードバ
ックを描画
    end

    opt 参照ポーズ記録
        User->>PoseDetectionWindow: 「参照ポーズを記録」ボタンをクリック
        PoseDetectionWindow->>PoseDetectionWindow: カウントダウン表示
        loop 3秒間 (on_recording_timer)
            PoseDetectionWindow->>PoseDetectionWindow: 検出ポーズを
recorded_posesに保存
        end
        PoseDetectionWindow->>DatabaseManager: 既存の参照ポーズを削除
(DELETE FROM reference_poses)
        PoseDetectionWindow->>DatabaseManager: 新しい参照ポーズを挿入
(insert_reference_pose)
        DatabaseManager-->>PoseDetectionWindow: 記録結果
        PoseDetectionWindow->>PoseDetectionWindow: 記録成功/失敗メッセー
ジを表示
    end

    User->>PoseDetectionWindow: 「閉じる」ボタンをクリック (またはウィンドウ
を閉じる)
    PoseDetectionWindow->>PoseDetectionWindow: カメラを停止
(stop_camera)
    PoseDetectionWindow->>DatabaseManager:
update_exercise_session_end(history_id, status, correct_seconds,
calories_burned, duration_minutes)

```

DatabaseManager-->>PoseDetectionWindow: 更新結果
PoseDetectionWindow-->>ExerciseDetailWindow: 画面を閉じる

6. エラー・例外処理

6.1. 予想されるエラー

- **データベース接続エラー:**
 - 発生箇所: `DatabaseManager`のコンストラクタ、または各データベース操作メソッド。
 - 原因: データベースサーバーが起動していない、接続情報が間違っている、ネットワークの問題など。
- **認証失敗:**
 - 発生箇所: `AuthManager::authenticate_user`。
 - 原因: ユーザー名またはパスワードがデータベースと一致しない。
- **ユーザー登録失敗:**
 - 発生箇所: `AuthManager::register_user`。
 - 原因: ユーザー名やメールアドレスがすでに登録されている、入力値のバリデーションエラー。
- **ポーズ検出モデルロード失敗:**
 - 発生箇所: `PoseDetectionWindow`のコンストラクタ。
 - 原因: `movenet_singlepose_lightning.tflite`ファイルが見つからない、またはファイルが破損している。
- **カメラ起動失敗:**
 - 発生箇所: `PoseDetectionWindow::start_camera`。
 - 原因: カメラデバイスが見つからない、他のアプリケーションがカメラを使用中、カメラドライバの問題。
- **参照ポーズ未登録:**
 - 発生箇所: `PoseDetectionWindow::update_frame`でのポーズ比較時。
 - 原因: 特定のエクササイズに対して参照ポーズがデータベースに保存されていない。
- **データ入力エラー:**
 - 発生箇所: ユーザー入力がある各画面（例: `RegisterWindow`, `ProfileEditWindow`, `PasswordChangeWindow`）。
 - 原因: ユーザーが必須項目を空にした、数値フィールドに文字を入力した、メールアドレスの形式が不正など。

6.2. 対応方法

- **エラーメッセージ表示:**
 - ユーザーインターフェース上で、エラーの内容を具体的に示すメッセージ（例: 「ユーザー名またはパスワードが間違っています。」）を`Gtk::Label`に表示します。
 - `PoseDetectionWindow`では、`update_status_label`メソッドを使用して、画面下部に赤字でエラーメッセージを表示します。
- **ログ出力:**
 - すべてのエラーは`std::cerr`に出力され、アプリケーションのログファイル（もし設定されていれば）にも記録されます。これにより、開発者は問題の原因を詳細に追跡できます。
 - 例: `std::cerr << "Error: Could not open camera" << std::endl;`
- **リトライ:**
 - データベース接続エラーなど、一時的な問題の可能性がある場合は、ユーザーに再試行を促すダイアログを表示します。

- **デフォルト値:**
 - プロファイル編集などで入力値が不正な場合、可能な範囲で以前の値に戻すか、適切なデフォルト値を適用します。
- **アプリケーションの終了:**
 - ポーズ検出モデルのロード失敗やデータベースへの接続不可など、アプリケーションの継続が困難な致命的なエラーが発生した場合は、ユーザーに通知した上で安全にアプリケーションを終了します。

7. セキュリティ

7.1. 認証・認可の方法

- **パスワードのハッシュ化:**
 - ユーザーが登録またはパスワードを変更する際、入力されたパスワードは直接データベースに保存されません。代わりに、`AuthManager::hash_password`メソッド内で強力なハッシュアルゴリズム（例: PBKDF2, bcryptなど）を使用してハッシュ化されます。
 - ログイン時には、入力されたパスワードも同様にハッシュ化され、データベースに保存されているハッシュ値と`AuthManager::verify_password`メソッドで比較されます。これにより、パスワードの漏洩リスクを低減します。
- **セッション管理:**
 - ログインが成功すると、サーバーサイドでセッションIDが生成され、クライアントにはJWT（JSON Web Token）のような認証トークンが発行されます。このトークンは`user_sessions`テーブルにハッシュ化されて保存され、有効期限が設定されます。
 - 以降のAPIリクエストにはこのトークンが含まれ、サーバーはトークンの有効性を検証することで、ユーザーのログイン状態と認可を管理します。

7.2. データ保護の仕組み

- **データベースのアクセス制御:**
 - アプリケーションは、`DatabaseManager`クラスを通じてのみPostgreSQLデータベースにアクセスします。直接的なデータベースアクセスは制限され、アプリケーションが使用するデータベースユーザーには必要最小限の権限のみが付与されます。
- **入力値の検証:**
 - ユーザーからの入力データは、データベースに保存される前に、各画面のUIロジックおよび`AuthManager`や`DatabaseManager`のメソッド内で厳格なバリデーション（検証）が行われます。これにより、SQLインジェクションや不正なデータ挿入などのセキュリティリスクを軽減します。
- **機密データの暗号化:**
 - パスワード以外の機密データ（例: 将来的に追加される可能性のある個人を特定できる情報）については、データベース保存時や通信時に暗号化を検討します。

8. 今後の課題

- **ポーズ検出の精度向上と多様なエクササイズへの対応:**
 - 現在のMoveNetモデルの精度をさらに高めるためのチューニングや、より複雑なエクササイズに対応できる新しいAIモデル（例: MediaPipe Pose, OpenPoseなど）の導入を検討します。これにより、ユーザーへのフィードバックの質を向上させます。
- **エクササイズの種類とコンテンツの拡充:**
 - ユーザーのモチベーション維持のため、ヨガ、筋力トレーニング、有酸素運動など、様々なカテゴリーの新しいエクササイズを定期的に追加します。また、エクササイズ動画の質向上や、多言語対応も視野に

入れます。

- **ユーザーインターフェース/ユーザーエクスペリエンス (UI/UX) の改善:**

- GTKMMのテーマやスタイルをカスタマイズし、よりモダンで直感的なデザインに刷新します。アニメーションの追加や、より分かりやすいナビゲーション構造の導入により、ユーザーエクスペリエンスを向上させます。

- **運動計画と進捗管理機能の強化:**

- ユーザーの年齢、体重、フィットネスレベル、目標に基づいて、パーソナライズされた運動計画を自動生成する機能を追加します。週次/月次のレポート機能や、目標達成度を視覚的に表示するダッシュボード機能も検討します。

- **マルチユーザー対応とプロフィール切り替え:**

- 一つのアプリケーションインスタンスで複数のユーザーが簡単にログイン・ログアウトし、それぞれのプロフィールと履歴にアクセスできる機能を実装します。

- **クラウド連携とデータ同期:**

- ユーザーの運動履歴やプロフィールデータをクラウドサービス（例: AWS, Google Cloud）と連携し、データのバックアップ、異なるデバイス間でのデータ同期、およびWebベースのダッシュボード提供を可能にします。

- **ゲーミフィケーション要素の導入:**

- バッジ、レベルアップ、チャレンジ、フレンドとのランキングなど、ゲーミフィケーション要素を導入することで、ユーザーの継続的な利用を促進し、運動をより楽しくします。