Web プログラミング及び演習

情報科学部 情報科学科 コンピュータシステム専攻
K22120 牧野遥斗

2024 年 11 月 21 日

目次

1. 目標 2

2. 設計のコンセプト 2

2.1. ユーザー管理方法 2

2.2. イベントの追加 3

2.3. デザインについて 3

2.4. 月間カレンダー表示 3

3. データベースのテーブルとリレーション 4

3.1. データベース設計 4

3.2. マイグレーションファイル 4

3.3. モデルファイル 7

4. 画面の状態遷移 9

4.1. 画面構成 10

5. 達成度 11

5.1. 完成度の評価 11

6. 反省点 11

6.1. 反省点 11

6.2. 改善案 11

1. 目標

本プロジェクトの目標は、カレンダー管理アプリを作成して、PHP の基本的な使い方やデータベース操作のスキルを習得する。 具体的には以下の点を学習し、実践する。

1. PHP の基礎知識 PHP の構文やデータベースと連携する方法を学ぶ。 また、データを操作する際の効率的で安全な手法についても検討する。

2. データベース設計のスキル セキュリティを意識した設計を心がけ、データの整合性を保つための外部キー制約や正規化の手法を用いる。

3. ファイル構造と設計パターンの理解 コードの保守性と拡張性を高めるために、MVC（Model-View-Controller）設計パターンを意識してファイル構造を構築する。 これにより、開発プロセスの効率化とアプリのスケーラビリティ向上を目指す。

これらを通じて、PHP とデータベースの実践的なスキルを習得し、より高いレベルでのシステム開発に対応できる基礎を築く。

2. 設計のコンセプト

本アプリは、ユーザーが自身のイベントを効率的に管理できるよう、以下の機能を実装する。

- イベント管理機能 イベントの追加、編集、削除、および一覧表示
- カレンダー表示機能 月間および週間のカレンダー表示
- 認証機能 ユーザーのログイン・ログアウト、登録機能

これらの機能を実現するために、Laravel フレームワークを採用した。 Laravel を使用して、セキュアかつ効率的な開発が可能になる。

以下に、各機能の詳細設計や実装時に意識したポイントを説明する。

2.1. ユーザー管理方法

ユーザー情報を管理するために、以下を実施する。

- データベース構造の設計 新規に「ユーザー」テーブルを作成し、ユーザー名、パスワード、メールアドレスといった情報を保存する。

- セキュリティ対策 パスワードはハッシュ化（bcrypt などのアルゴリズムを利用）して保存する。 これにより、データベースが不正アクセスを受けても、パスワードが漏洩しにくい仕組みを構築する。

- Laravel の Auth 機能の活用 ユーザー認証には Laravel の既存 Auth 機能を利用して、効率的かつ安全なユーザー管理を実現する。 これにより、セッション管理や認証に関する複雑な実装を簡素化できる。

2.2. イベントの追加

イベント情報の保存先として、MySQL を使用します。以下の点を意識して開発を進める。

- データ整合性の確保 外部キー制約を用いて、データ間の一貫性を維持する。 たとえば、ユーザーが削除された場合、そのユーザーに紐づくイベントも自動で削除されるように設計する。
- 効率的なクエリ設計 イベントの検索や表示が高速に行えるよう、適切なインデックスの設計も検討する。

2.3. デザインについて

UI デザインには CSS フレームワークの Bootstrap を活用する。これにより、以下のメリットを得られる。

- 開発スピードの向上 CSS をゼロから作成する手間を省き、統一感のあるデザインを迅速に構築できる。
- レスポンシブ対応 Bootstrap のグリッドシステムを活用し、どのデバイスでも見やすいデザインを実現する。

加えて、ユーザー体験 (UX) を高めるため、シンプルで直感的な操作性を重視したレイアウトを採用する。

2.4. 月間カレンダー表示

月間カレンダーの実装には、FullCalendar という JavaScript ライブラリを使用する。 主な特徴と活用方法は以下の通りである。

- 機能の充実 月間ビューの表示やイベントのドラッグ＆ドロップによる操作が簡単に実現できる。
- 導入の容易さ CDN を用いて手軽に導入可能で、少ないコードで高機能なカレンダーを実装できる。

FullCalendar の利用により、カレンダー表示が直感的で視認性が高くなり、ユーザーの利便性を向上させる。 また、このライブラリをカスタマイズし、アプリのデザインや機能に適したカレンダー表示を目指す。

3. データベースのテーブルとリレーション

本プロジェクトでは、Laravel に搭載されている Eloquent ORM を使用してデータベースの操作を行う。Eloquent ORM を用いると、以下のような利点がある。

- コードの簡潔化 データベースのテーブルをオブジェクトとして扱えるため、SQL 文を直接記述する手間が省け、可読性の高いコードが書ける。
- リレーションの管理 テーブル間のリレーション（関連性）を簡単に設定し、複雑なデータ操作も直感的に行える。

3.1. データベース設計

本アプリでは以下の 2 つのテーブルを作成し、ユーザー情報とスケジュール情報を管理する。

- Users テーブル ユーザー情報（名前、メールアドレス、パスワードなど）を管理する。
- Schedules テーブル 各ユーザーに紐づくイベント情報（タイトル、日時、詳細など）を管理する。

これらのテーブルは、Laravel のマイグレーション機能を使用して定義・生成する。マイグレーションにより、データベースの構造を簡単に管理・変更できるため、開発中の仕様変更にも柔軟に対応可能となる。

3.2. マイグレーションファイル

Laravel では、マイグレーションファイルを用いてデータベースのテーブル構造をプログラムで定義する。以下は、Schedules テーブルと Users テーブルのマイグレーションファイルの一部を示す コード 1 コード 2。

マイグレーションでは、外部キー制約を設けると、データの整合性を保つ設計を行える。 例えば、Schedules テーブルでは user_id を Users テーブルの主キーと関連付けるような設計を行うと、ユーザーが削除された際にそのユーザーに紐づくスケジュールも自動で削除される。

コード 1: 2024_11_14_015911_create_schedules_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        if(!Schema::hasTable('schedules')) {
            Schema::create('schedules', function (Blueprint $table) {
                $table->id();
                $table->foreignId('user_id')->constrained()->cascadeOnDelete();
                $table->string('title');
                $table->text('description')->nullable();
                $table->dateTime('start');
                $table->dateTime('end');
                $table->timestamps();
            });
        }
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('schedules');
    }
};
```

コード 2: 0001_01_01_000000_create_users_table.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

3.3. モデルファイル

Eloquent ORM を用いるため、各テーブルに対応するモデルファイルを作成した。 これにより、複数のファイルや機能から簡単にデータベースの操作が行えるようになる。以下に、モデルファイルの一部を示す コード 3 コード 4。

コード 3: User.php

```php
<?php
namespace App\Models;
// use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array<int, string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
}

```

コード 4: Schedule.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class Schedule extends Model
{

    // Factory を使うための記述
    use HasFactory;

    //テーブル名
    protected $table = 'schedules';

    //可変項目
    protected $fillable =
    [
        'user_id',
        'title',
        'description',
        'start',
        'end',
    ];

    //日付の属性を変更
    protected $dates = ['start', 'end'];

    //リレーション
    public function user()
    {
        return $this->belongsTo('App\Models\User');
    }
}

```

4. 画面の状態遷移

以下の図形はスケジュール管理アプリの画面遷移図である 図 1 。 この図は、ユーザーがどのようにアプリ内を行き来し、スケジュールの登録や確認、編集を行えるかを示す。

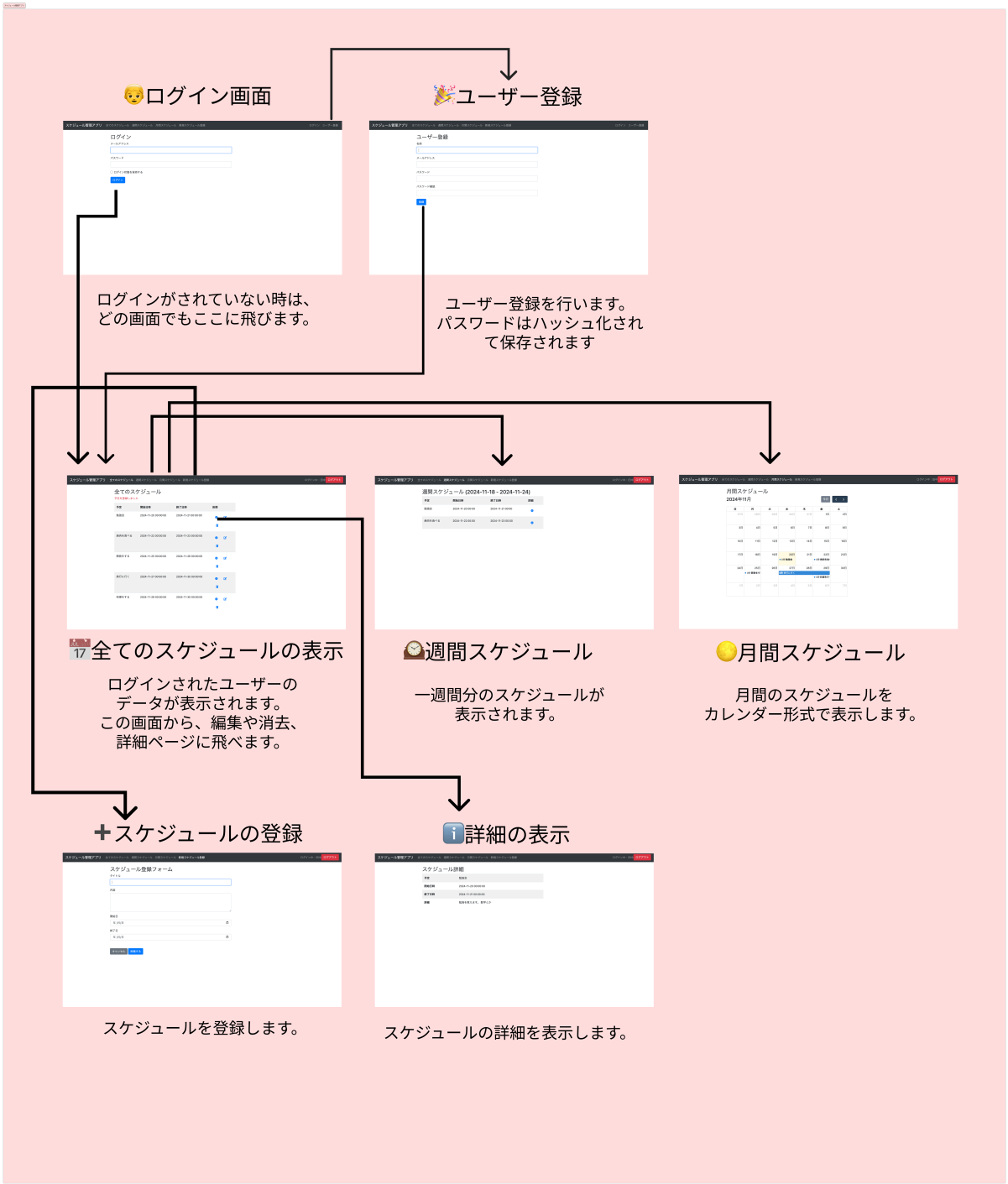

図 1: スケジュール管理アプリの画面遷移図

4.1. 画面構成

アプリ内には以下の種類の画面がある。それぞれの画面で特定の機能を担い、スケジュール管理を円滑に行えるよう設計を行なった。

1. ログイン画面 ユーザー認証を行う画面。登録済みのユーザーがメールアドレスとパスワードを入力してログインする。 Laravel の Auth 機能を活用し、セキュアな認証を実現した。

2. ユーザー登録画面 新規ユーザーを登録する画面。 ユーザー名、メールアドレス、パスワードを入力し、データベースに保存する。 入力されたパスワードはハッシュ化して保存し、セキュリティを考慮する。

3. 全てのスケジュールの表示画面 登録されている全スケジュールをリスト形式で表示する画面。

4. 週間スケジュール表示画面 特定の 1 週間におけるスケジュールをカレンダー形式で表示する画面。短期間の予定を把握しやすくするため、週間表示を採用した。

5. 月間スケジュール表示画面 1 か月単位のスケジュールを表示する画面。FullCalendar ライブラリを利用して月単位でのスケジュールを分かりやすく視覚化した。 予定の全体像を把握しやすいデザインを採用した。

6. スケジュール登録画面 新しいスケジュールを追加する画面。タイトル、日付、開始時刻、終了時刻、詳細などの項目を入力し、データベースに保存する。入力エラー時にはリアルタイムでフィードバックを行い、使いやすさを向上させた。エラーチェックを行い、データの整合性を保つ設計を行った。

7. スケジュール詳細表示画面 特定のスケジュールの詳細情報を確認できる画面。

本画面遷移図と設計に基づき、スケジュール管理アプリはユーザーが直感的に操作しやすく、視覚的な情報提示とスムーズなデータ操作を実現した。

5. 達成度

本プロジェクトでは、カレンダー管理アプリを完成させ、以下のような学びや成果を得られた

1. PHP の習得 PHP の基本構文やデータベースとの連携方法について理解を深められた。 特に、Laravel フレームワークを用いて、効率的かつセキュアなアプリケーション開発の基礎を学べた。

2. 主要機能の実装 ユーザー登録、スケジュールの登録、表示、編集、削除といった基本機能を実装した。 これにより、ユーザーがスケジュールを簡単に管理できるシステムを提供できた。

3. デザインの工夫 Bootstrap を活用し、使いやすさと統一感のあるデザインを実現した。 カレンダー表示には FullCalendar ライブラリを採用し、視覚的にわかりやすいインターフェースを構築した。

5.1. 完成度の評価

機能面とデザイン面のほとんどが計画通りに実装できたため、完成度は 95% 程度と考えた。一部改善点はあるものの、全体としてユーザーにとって使いやすいアプリケーションを作成できたと感じた。

6. 反省点

今回のプロジェクトではいくつかの課題が浮き彫りになった。次回以降の開発に活かすために、以下の反省点と改善案を挙げる。

6.1. 反省点

- モバイル端末への対応不足 現在のデザインは主に PC 向けに最適化されており、モバイル端末での利用時に使いにくい部分がある。具体的には、レスポンシブデザインを適切に適用できていないため、小さい画面では表示が崩れるケースが見られた。

- デバッグの不足 開発途中でのデバッグが不十分だったため、エラーが発生した際の対処ができていない部分がある。 一部のバグに気づかないまま進行してしまったが、後々修正の手間を増やす結果となった。

6.2. 改善案

- レスポンシブデザインの導入 次回は、Bootstrap のグリッドシステムや Flexbox を活用し、画面サイズに応じてレイアウトを調整できるレスポンシブデザインを実装する。また、モバイル端末専用の UI を意識し、ボタンの大きさや配置などにも配慮した設計を行って行きたい。

- バグを発生させないような実装 テスト駆動開発（TDD）やユニットテストを導入し、開発途中からバグを発生させないような設計を心がける。 また、エラーハンドリングを適切に行い、ユーザーにエラーが発生した際の適切なメッセージを表示するようにする。

これらの反省点と改善案を基に、次回のプロジェクトではより完成度の高いアプリケーションを目指して取り組みたい。また、モバイルユーザーにも配慮したデザインを導入し、幅広い利用者層に対応したサービスを提供したいと考える。