卒業論文

旅程の共有と準備を支援する旅行アプリの開発

関西学院大学工学部

情報工学課程 西谷研究室

37022463 山本果音

2026 年 3 月

概 要

旅行アプリ「旅しお」では旅行計画を簡単に立てることが可能であるが, 予定作成時の情報整理や位置情報の提示が不十分であり, 細かい旅程を決めるまで一貫して利用するのには向いていない. 本研究では, 旅行前の計画段階において, ユーザーがスムーズに情報を整理・共有できることを目的とし, デジタル版しおりの開発を行った. 具体的な特徴として, アカウント機能や共有機能, 旅行名や説明文をキーとした検索機能, 旅行日付に基づく時系列整理, 目的地の地図表示機能, 移動時間表示機能, 訪問順序入れ替え機能, チェックリスト機能を実装し, 新たな予定を立てやすくした. 結果としてユーザーは旅行計画を効率的に立てることができ, 一貫した利用が可能になった.

目 次

図 目 次

第1章　序論

修学旅行などのイベントでは, 参加者に対して事前にしおりが配布されることが多い. しおりは旅程の把握や所持品の確認, 集合時間の共有などを目的とした重要な情報媒体であり, 旅行の準備や当日の行動において欠かせない役割を果たしている. 近年では, スマートフォンの普及により, 紙媒体に代わってWebサービスを活用した旅行データの閲覧・共有のニーズが高まっており, 「旅しお」[1] のようなWebアプリも登場している. しかし, これらのサービスにはいくつかの課題が存在する.

1つ目は, 旅行名や説明文に対するキーワード検索機能が存在しないため, ユーザーはスクロール操作によって目的のしおりを探す必要がある点である.

2つ目は, 地図表示機能がないため, 目的地の位置関係や全体像を視覚的に理解することが難しく, 利用者は外部ツールや手作業で補完しなければならない.

3つ目は, 移動時間の自動算出機能が存在しないため, 各目的地間の所要時間をユーザー自身が都度手動で調べる必要がある点である. これにより, 旅行全体のスケジュールを立てる際に手間がかかり, 計画の精度や効率性が低下する可能性がある.

これらの課題は, 旅行計画の際にユーザーにとって大きな障壁となっており, データの整理や視覚的な理解の低下が考えられる. そこで本研究では, 視覚的な理解と効率的に旅行計画を行える旅行支援アプリの開発を目指す.

第2章　手法

2.1　開発手法

本研究では, 限られた開発期間内で効率的に成果を上げるために, 機能ごとに優先順位を設定し, それぞれについて計画・設計・実装・テストの工程を小刻みに繰り返すアジャイル手法を採用した. 各段階で得られた知見や, 研究室内のメンバーからのフィードバックをもとに, 仕様や UI の見直しを行いながら, 柔軟に改善を重ねた. これにより, 利用者の視点を反映した機能の追加や調整が可能となり, アプリケーションの完成度と実用性を段階的に高めることができた.

2.1.1　Django

開発環境として, Python で実装されたサーバーサイド Web アプリケーションフレームワークである Django を選定した [2]. Django を選定した理由は以下の 3 つである.

1 つ目は, ユーザー認証や管理画面, データベース操作などの機能が初期状態で組み込まれており, 追加設定を行わずとも迅速に開発を開始できる点である. これにより, 開発初期の環境構築にかかる時間を大幅に削減することが可能となった.

2 つ目は, データベース操作をすべて Python で記述できるため, 学習コストが低い点である. Django では, モデルの定義からマイグレーションの実行までを一貫して Python で記述できるため, SQL の詳細な知識がなくても複雑なデータ構造の設計や変更が容易に行える.

3 つ目は, セキュリティ対策がフレームワークに標準で備わっている点である. Django はクロスサイトスクリプティング (XSS) や SQL インジェクションなどの脆弱性に対して, 初期状態で対策が施されている. XSS 攻撃は, 攻撃者が作成したクライアント用のスクリプトを別のユーザーのブラウザ上で実行させる攻撃であり, CSRF 攻撃は悪意ある利用者

がユーザー自身の意図や操作を介さずに, そのユーザーの認証情報を用いたリクエストを強制的に送信させる攻撃である [3]. これらの対策が Django に標準搭載されていることで, 開発者が個別にセキュリティ機能を実装する必要がなく, 安全性の高い Web アプリケーションを効率的に構築することができた.

Django は MTV（Model–Template–View）アーキテクチャに基づいており, それぞれ「アプリ内で扱うデータの構造を定義し, データベースとの連携を担うモデル層（Model)」「ユーザーの操作に応じて処理を行い, 適切なデータを返すビュー層（View)」「HTML で記述された画面を表示するテンプレート層（Template)」の 3 つの層に分かれている. たとえば, 旅行作成機能では, テンプレートで旅行タイトル・説明・日付などのテキストフィールドなど表示し, ユーザーが入力したデータをビューが受け取って処理し, モデルとして整形してデータベースに保存する. 保存されたデータは, 旅行一覧や詳細ページで再びビューによってデータベースから取得され, テンプレートに渡されて HTML 内でデータが表示される. この一連の流れにより, 旅行データの作成 (Create)・取得 (Read)・更新 (Update)・削除 (Delete) といった基本的な CRUD 操作が実現されている. このように Django の MTV アーキテクチャは, 機能ごとの責任範囲を明確に分離することで, コードの可読性や再利用性を高めている. 図 2.1 は Django の MTV 構成 [4] を参考にした.

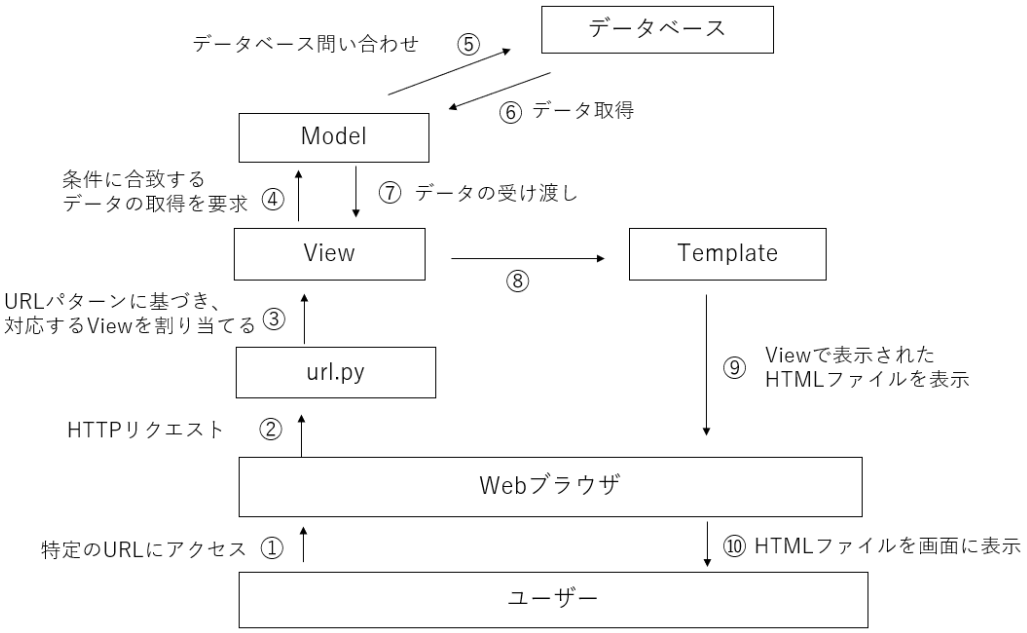

図 2.1: Django の MTV 構成に基づくリクエスト処理の流れ.

また, 実装には HTML, JavaScript, CSS を使用し, UI ライブラリとしては Bootstrap を

採用した. 開発にあたってはCopilot[5]を参照しながら, 実装を進めた.

第3章　結果と考察

3.1　アプリ機能

今回, 開発したアプリでは大きく分けて7つの機能を実装した. それぞれの機能は図3.1の通りである. それぞれの機能について次節以降で記述する.

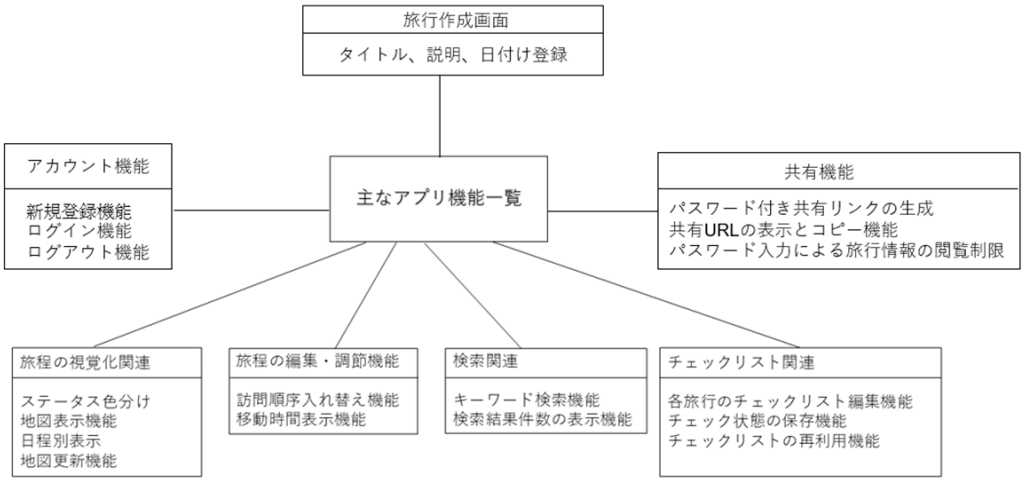

図 3.1: アプリ機能一覧.

3.1.1　アカウント機能

本アプリケーションでは, ユーザー認証機能としてログイン機能と新規登録機能を実装した.

図3.2に示したとおり, ログイン画面では登録済みのユーザーがユーザー名とパスワードを入力することで, 個別の旅行プランやチェックリストにアクセスできるようになっている.

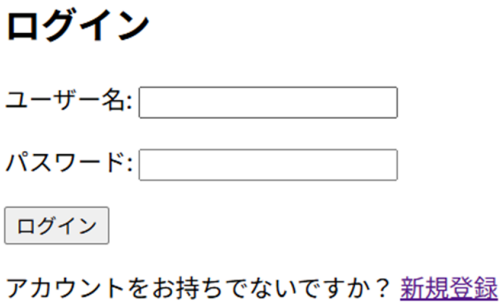

図 3.2: ログイン画面.

一方, まだ登録を行っていないユーザーに対しては, 図 3.3 のように新規登録画面を用意しており, ユーザー名・パスワード・確認用パスワードの 3 項目を入力することでアカウントを作成できる仕組みとした. パスワードの確認入力を設けることで, 入力ミスによるログイン失敗を防ぎ, ユーザー体験の向上を図っている.

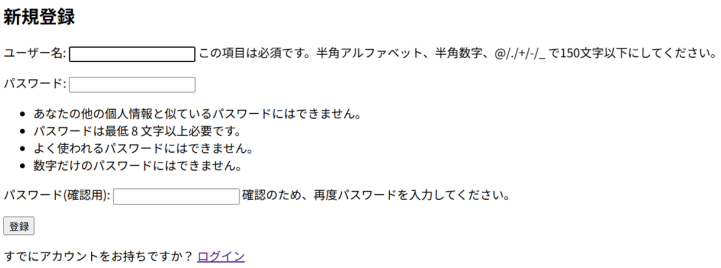

図 3.3: 新規登録画面.

図 3.4 に示したとおりログアウト機能も実装しており, ユーザーがログアウト操作を行うと, 確認ダイアログが表示されるようになっている. このダイアログでは「ログアウトしますか?」というメッセージとともに, OK またはキャンセルの選択肢が提示される. 専用のログアウトページは設けていないが, ユーザーの意図しないログアウトを防ぐための確認ステップを導入することで, 操作ミスの抑制とユーザー体験の向上を図っている.

図 3.4: ログアウト画面.

3.1.2 旅行作成画面

図 3.5 は旅行作成画面である. この画面では, タイトル, メモ, 旅行期間が入力できる. 開始日と終了日から旅行の期間を計算し, のちに説明する旅行詳細画面で日程ごとの計画を表示できるようにしている.

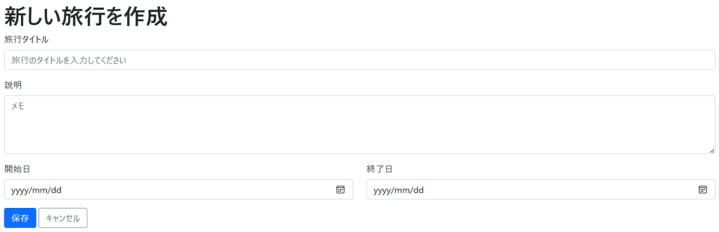

図 3.5: 旅行作成画面.

3.1.3 旅行一覧画面

まず, 図 3.6 に示したとおり登録された旅行期間に基づいて, 各旅行を「予定」「旅行中」「終了」「未定」の 4 つのステータスに分類し, 色分けによって一覧画面上に表示する機能を実装した. これにより, ユーザーは「どこに行ったか」「これからどこに行くか」といった旅行の進行状況を一目で把握できるようになった. 具体的には,「旅行中」は緑,「予定」

は青,「終了」はグレー,「未定」は黄色となっている．さらに, 色だけでなくアイコンを加えることでより視覚的にわかりやすい工夫を行った.

図 3.6: 色とアイコンで視覚的に整理された旅行一覧画面.

旅行一覧画面に多くの旅行が表示されている場合, 目的の旅行を探し出すのに時間がかかることがある．そこで, 図 3.7 のように旅行名や説明欄に含まれる単語をもとに部分一致によるキーワード検索を行い, 該当するデータを絞り込めるようにした．いずれの検索においても, 大文字・小文字を区別しない部分一致検索を実現しているため, ユーザーは入力時に文字の大文字・小文字を意識する必要がない．そのため, 検索漏れを防ぐことができ, よりスムーズに目的のデータへアクセスできるようになった．また, 検索結果の件数を取得し,「〇件」という形式で表示することで, 検索結果の量を一目で把握できるようにした.

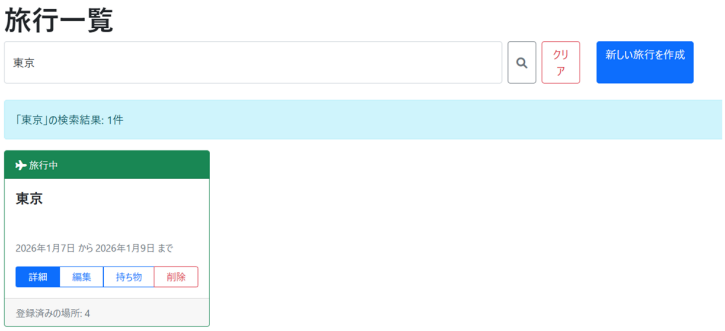

図 3.7: 検索結果画面.

3.1.4 旅行詳細画面

旅行作成時に入力された期間により, 旅行の日数分だけタブを動的に生成する機能を実装した. また, 旅行に含まれる観光場所を「全体」「1 日目」「2 日目」など日程ごとに切り替えて表示し, ユーザーはタブを操作することで旅行全体の流れを把握できるようにした. 各地点には名称や住所, 地点間の所要時間が表示されている. それぞれの日程に追加された地点は右側の地図上に表示され, 訪問場所と訪問順序が視覚化されるように実装した. また, ユーザーは各日程の訪問場所の追加や削除ができ, それに応じて地点間の所要時間や地図上の表示も更新される. これにより, 異なる日程の操作を誤って行ってしまうことや, 非現実的なスケジュールや移動の偏りにも気づきやすくなる.

図 3.8 のように地図上に地点を表示するためには, まず Google Maps JavaScript API[6] を読み込む必要がある. 本システムでは,

```
<script src="https://maps.googleapis.com/maps/api/js?
     key={{ google_maps_api_key }}&callback=initMap"
     async defer></script>
```

のように API を読み込んでいる.

key パラメータに Google Maps API の利用に必要な API キーを指定しており, Google 側での認証に使用される. また, callback パラメータには,API の読み込み完了後に自動的に実行される初期化関数 initMap を指定している. これにより, 地図の表示処理を API の読み込み完了後に確実に実行することが可能となる.

Google Maps API が読み込まれると, コールバック関数として指定された initMap() 関数が自動的に実行される.

initMap() では以下の処理が行われている.

```
function initMap() {
    // 日本の中心あたりを初期表示
    var map = new google.maps.Map(document.getElementById('map'), {
 zoom: 5,
 center: {lat: 36.2048, lng: 138.2529}
```

```
    });
}
```

この関数では, google.maps.Map コンストラクタを用いて地図オブジェクトを生成している. 初期表示位置として日本の中央付近を設定し, ズームレベルは 5（日本全土が見渡せる広域表示）としている.

図 3.8: 地図表示.

図 3.9 のように, タブの切り替えと地図表示を連動させることで, ユーザーはその日の観光地だけを地図上で確認できる. これにより, 日程ごとの旅程を迷わず管理できる.

これらの機能により, ユーザーは旅行の計画段階において, 旅程を効率的に把握・調整できるようになり, 直感的な旅行プランニングが可能となった.

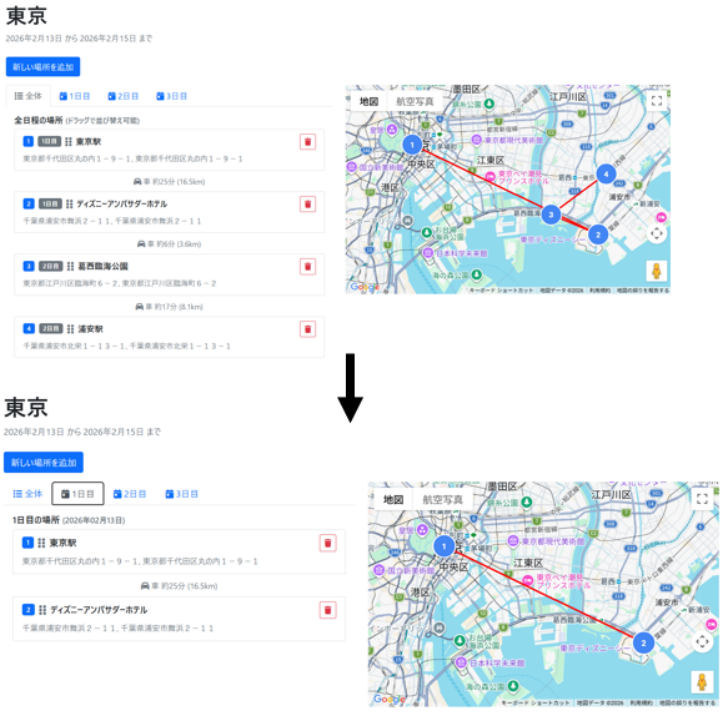

図 3.9: 全体から 1 日目のタブを押したときの変化.

3.1.5 旅行詳細画面 - 編集機能

旅行の計画を柔軟に調整できるよう, 旅程編集に関する 2 つの機能がある.

まず 1 つ目は, 観光場所の訪問順序をドラッグすることで自由に入れ替える機能である. これにより, ユーザーは移動効率や興味関心に応じて旅程を柔軟に再構成できるようになる. 例えばメモ帳などで訪問順序を管理すると, 訪問順序を変更するたびに項目を削除し, 都度追加し直す必要があるが, それが無くなることで操作性が向上し旅程編集の負担が大幅に軽減される.

この機能の実装には, SortableJS というドラッグ&ドロップ機能を提供する JavaScript ライブラリを用いている. SortableJS は, 要素の並べ替えを直感的に行える軽量なライブラリであり, CDN(Content Delivery Network) と呼ばれる仕組みを通じて読み込まれている. CDN とは, インターネット上に分散配置されたサーバー群から, ライブラリなどのファイルを高速かつ安定して配信する仕組みである. よって, 開発者はライブラリを自分でダウンロードして設置することなく, 外部の URL を指定するだけで簡単に利用できる.

ドラッグ開始時の処理は以下である.

```
onStart: function(evt) {
        console.log('Drag started');
        isDragging = true;

        // 所要時間を非表示にする
        const activeTabPane = document.querySelector('.tab-pane.active');
        const durationElements = activeTabPane ?
        activeTabPane.querySelectorAll('.travel-duration')
        : [];
        durationElements.forEach(element => {
     element.style.display = 'none';
        });
    }
```

ドラッグ終了時には,以下の処理を実行する.

```
onEnd: function(evt) {
  console.log('Drag ended, oldIndex:', evt.oldIndex, 'newIndex:', evt.newIndex);
  isDragging = false;

  // 所要時間を再表示
  const activeTabPane = document.querySelector('.tab-pane.active');
  const durationElements = activeTabPane ?
  activeTabPane.querySelectorAll('.travel-duration')
  : [];
  durationElements.forEach(element => {
      element.style.display = 'block';
  });

  updateOrderNumbers();
```

```
  if (evt.oldIndex !== evt.newIndex) {
      updatePlaceOrder();
  } else {
      // ドラッグしたが順序が変わらなかった場合も所要時間を再計算
      updateTravelDurations();
  }
     }
```

順序番号の更新処理は以下である.

この関数は, 現在表示されているタブ内のすべての場所カードに対して, 配列のインデックスに 1 を加えた値を順序番号として設定する.

サーバーへの順序保存処理は以下のようになっている.

```
function updatePlaceOrder() {
    if (isUpdating) {
 console.log('Update already in progress, skipping...');
 return;
    }

    isUpdating = true;
    console.log('Starting order update...');

    const sortableContainers = document.querySelectorAll('[id^="sortable-places"]');
    sortableContainers.forEach(container => {
 if (container && container.sortable) {
     container.sortable.option('disabled', true);
 }
    });

    const activeTabPane = document.querySelector('.tab-pane.active');
    const items = activeTabPane ? activeTabPane.querySelectorAll('.sortable-item') :
```

```
    console.log('Found items for reordering:', items.length);

    const placeOrders = [];

    items.forEach((item, index) => {
 const placeId = parseInt(item.dataset.placeId);
 const day = parseInt(item.dataset.day);
 const order = index + 1;
 console.log(`Item ${index}: placeId=${placeId}, day=${day}, newOrder=${order}`);

 placeOrders.push({
     place_id: placeId,
     day: day,
     order: order
 });
    });
}
```

２つ目は, 図 3.10 に示したとおり, 各観光地間の移動時間を自動で算出・表示する機能である. 算出されるのは車での移動を想定した所要時間であり, ユーザーは移動にかかる時間を考慮しながら, 現実的かつ無理のないスケジュールを立てることが可能である.

これらは Mapbox の Directions API[7] を活用して開発を行っている. 以下にその処理の流れを示す.

ユーザーが現在選択しているタブに表示されている観光地リストから, 各地点の座標（経度・緯度）を配列として取得する. この配列の先頭を出発地, 末尾を目的地, それ以外を経由地として分割し, Directions API に渡す形式に整形する.

travelDuration(type, start, waypoints, dest) は,Mapbox Directions API を使ってルートデータを取得する関数である.

●type：移動手段（例：driving, walking, cycling）

●start：出発地の座標（{ lon, lat }）

●waypoints：経由地の配列

●dest：目的地の座標

経由地は任意の個数を指定できるため, 配列として管理している. 指定形式に変換した座標データと移動手段をもとに, 以下の形式の URL に対してリクエストを送信する.

`https://api.mapbox.com/directions/v5/\{profile\}/\{coordinates}`

に対してリクエストを送信することで, 複数地点間の移動時間・距離を一度に取得できる.

・profile: 移動手段を指定するパラメータであり, driving（車）, walking（徒歩）, cycling（自転車）の 3 種類が用意されている. 本アプリでは driving を使用している.

・coordinates: 出発地・経由地・目的地の座標を「経度, 緯度」形式でセミコロン（;）区切りで列挙する.

この形式により, 複数地点間のルートを 1 回のリクエストでまとめて取得することが可能となる.

API からのレスポンスには routes 配列が含まれており, 最適なルートが routes[0] に格納されている. この中の legs 配列には, 各区間（出発地→経由地→目的地）ごとの詳細データが含まれており, それぞれの leg には以下のデータが含まれる.

・duration: 所要時間（秒単位）

・distance: 移動距離（メートル単位）

取得した各区間のデータをもとに, 所要時間は分単位に, 距離はキロメートル単位（小数点 1 桁）に変換し, 「車 約 24 分（16.5km）」のような形式で, 観光地カードの間に動的に挿入して表示している. 移動時間が可視化されたことで, 訪問地の組み合わせや順序の検討がより直感的かつ効率的に行えるようになった.

これらの機能は, 旅行の編集する上で不可欠な要素であり, ユーザーが旅程全体の流れを把握しながら, 日程ごとの詳細な計画を直感的に編集できる.

図 3.10: 車による移動時間の自動算出結果を表示した画面.

3.1.6 旅行詳細画面 - 地点登録機能

地点登録機能では, Mapbox の Search Box API[8] を活用している.

施設名の入力画面は図 3.11 である. これにより, ユーザーは住所だけでなく, 観光地名やカテゴリなどを入力することで, 目的の場所を効率的に検索できる.

一度は完全無料の OpenStreetMap の Nominatim を利用して実装を行ったが, 目的の観光地が検索にヒットしにくいという課題があった. そこで Mapbox の Search Box API へ切り替えたことにより, 検索精度が向上しより多くの地名を正確に取得できるようになった. この改善はユーザー体験の向上につながると考える.

精度面では Google Maps API の方が OpenStreetMap や Mapbox には勝っている一方, Google Maps API はコスト的なハードルがある. しかし, Mapbox の Search Box API は月 10 万回まで無料で利用できるためコスト面でも導入しやすいという利点がある.

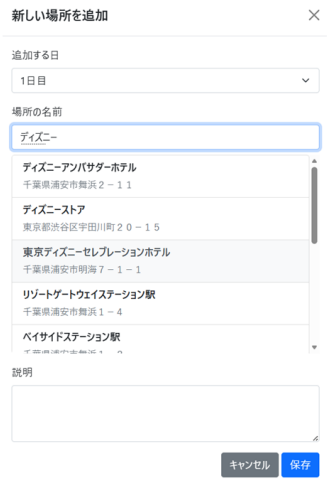

図 3.11: 施設名の入力画面.

3.1.7　チェックリスト機能

旅行準備を効率的に進められるよう, チェックリストに関する複数の機能を実装した.

まず, 図 3.12 のように旅行ごとに標準のチェックリスト設定を適用できる機能を実装した. これにより, ユーザーは毎回ゼロからリストを作成する必要がなく, あらかじめ用意された基本項目をもとに効率よく準備を開始できるようになった. さらに, 基本項目をテンプレート化することで, 今回だけ書き忘れたという漏れを防ぐ効果も期待できる.

図 3.12: チェックリスト設定画面.

また本アプリでは, ユーザーごとに持ち物チェックリストを個別に管理できるようにしている. そのため, 新規ユーザーが作成された際に, あらかじめ用意された共通のチェック

リスト項目を自動的に複製し, ユーザー専用のデータとして登録する仕組みを実装した. この処理は Django のシグナル機能 [9] を用いて実現しており, 以下のような関数を定義している. 本実装では, シグナル機能を利用することで, ユーザーが新しく登録された際に, 初期チェックリストをコピーする処理を自動で実行できるようになった.

```
@receiver(post_save, sender=User)
def create_user_checklist(sender, instance, created, **kwargs):
```

この関数は, User モデルのインスタンスが新規作成されたタイミング (post_save シグナル) で呼び出され, 以下の処理を実行する.

1.user=None かつ is_system_default=True の条件に一致する, システム共通のチェックリスト項目を取得する.

```
if created:
    # システムデフォルト（user=None）のアイテムを取得
    system_items = UserChecklistItem.objects.filter(
 user=None,
 is_system_default=True
    )
```

2. 各項目を, 作成されたユーザー用に複製し, user=instance として新たなインスタンスを生成する.

```
user_items = []


    for item in system_items:
 user_items.append(
     UserChecklistItem(
  user=instance,
  category=item.category,
  name=item.name,
  order=item.order,
```

```
  is_system_default=False  # ユーザー用なので False
    )
)
```

3.bulk_create() を用いて, 複数のチェックリスト項目を一括でデータベースに登録する. これにより, データベースへのアクセス回数を抑え, 高速かつ効率的に登録処理を行うことができる.

```
if user_items:
   UserChecklistItem.objects.bulk_create(user_items)
```

このようにすることで, ユーザーはアカウント作成直後から, 基本的な持ち物が登録されたチェックリストを利用できるようになり, 利便性が向上する. また, チェックリストはユーザーごとに独立して管理されるため, 各自が項目の追加・削除・並び替えを自由に行える.

また, 各旅行ごとに持ち物や準備項目を自由に編集できるチェックリスト編集機能を実装した. これにより, ユーザーは旅行の目的や季節, 同行者に応じて, 図 3.13 のように必要な項目を柔軟に追加することも可能となった. 状況に応じたカスタマイズが行えることで, より実際のニーズに即した準備ができるようになり, 忘れ物や準備漏れの防止にもつながる.

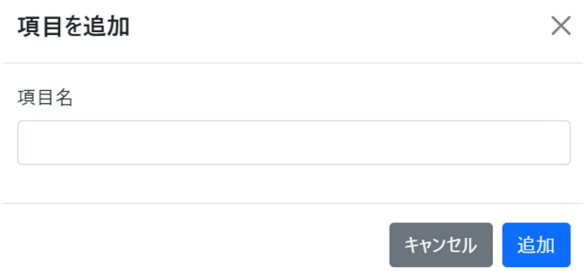

図 3.13: チェックリスト追加画面.

さらに, 図 3.14 に示したとおり, 各項目のチェック状況を保持する機能を実装し, ユーザーが確認済みの持ち物を記録できるようにした. これにより, 準備の進捗状況を可視化

しながら, 持ち物の確認状況を一目で把握できるようになり, 確認漏れや二重確認の負担を軽減することが可能となった. チェックの状態は旅行ごとに保存されるため, 複数の旅行を並行して準備する場合でも, 各旅程の進捗を個別に管理できる.

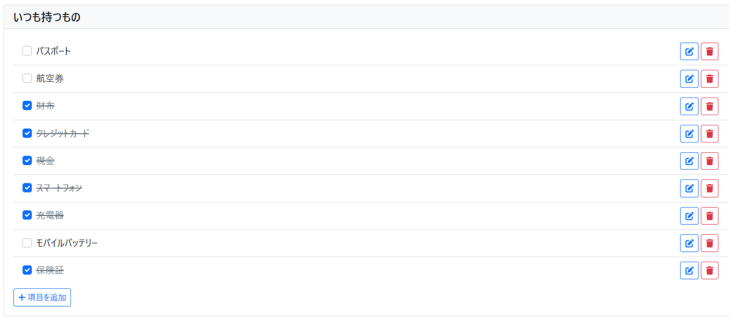

図 3.14: チェックリストの管理画面.

これらの機能は, 旅行前の準備を支援する上で重要な役割を果たしており, ユーザーが安心して出発できるよう, 実用性と柔軟性を兼ね備えたチェックリスト管理環境を提供している.

3.1.8 共有機能

本アプリでは, 図 3.15 に示したとおりユーザーが作成した旅行計画を他者と共有するためのリンク生成機能を実装した. この機能により, ユーザーはパスワード付きの専用 URL を発行し, 家族や友人と旅行データを安全かつ手軽に共有できる. 共有リンクの作成時には, ユーザーがあらかじめ用意された選択肢から有効期間を選択できるように設計しており, 設定された期限を過ぎたリンクに対しては, システムが自動的に無効と判断し, アクセスを制限する仕組みを導入している.

図 3.16 に示したとおり, ユーザーがパスワードを入力して共有リンクを作成すると, 専用 URL が画面上に表示される. また, ワンクリックで URL をコピーできるボタンを設置しており, 手動入力によるミスを防ぎつつ, スムーズな共有を実現している. コピー完了後には, コピーしましたと表示されるため, 操作結果が視覚的に分かりやすく, ユーザーの安心感にもつながっている.

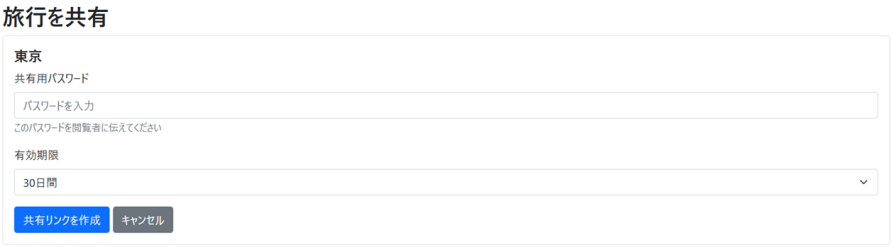

図 3.15: リンク作成画面.

図 3.16: 共有 URL 表示画面.

図 3.17 のように, このリンクを受け取った相手は, 共有側が作成したパスワードを入力することで旅行計画の詳細を閲覧できる. ただし, パスワードが正確に入力されない場合は画面にメッセージが表示されて共有された旅行画面をみることができない.

図 3.17: パスワード入力画面.

観光場所の追加や削除などによりデータを編集した場合でも, リンクが有効期限内であれば常に最新の旅行データが反映されるようになっている. これにより, 共有先のユーザーも常に最新の旅程を確認でき, 変更内容の伝達漏れを防ぐことができる.

また, データを更新するたびに新たなリンクを作成する必要がないため, 複数のリンクが発生して混乱することもなく, スムーズな共有が可能となっている.

共有された側のユーザーは, 図 3.18 に示したとおり旅行データを閲覧することはできるが, 内容の編集はできないように制限している. これにより, データの改ざんや誤操作を防ぎつつ, 安全かつ一方向的な共有を実現している.

図 3.18: 共有表示画面.

3.1.9 UI 設計とスタイルの工夫

本システムでは, Bootstrap 5.1.3[10] を採用し, ユーザーインターフェースの視覚的理解を促進した.

観光地データなどの表示に Bootstrap のカードコンポーネントを活用し, データの視認性と整理性を高めている. 例えば, 旅行一覧や旅行詳細の訪問場所をカード形式で表示することで, 各データのまとまりを視覚的に明確化した. これにより, ユーザーは一目でデータの区切りを理解できる.

さらに Font Awesome[11] を併用し, ボタンにアイコンを付与することで, 言語に依存しない直感的な操作を可能にした.

加えて, 旅行詳細画面の並び替え機能ではユーザー体験を向上させるため, 図 3.19 のような CSS アニメーションを実装した.

以下は実装内容である.

```
.sortable-ghost {
    opacity: 0.4;
}

.sortable-drag {
    transform: rotate(5deg);
}
```

```
.drag-handle:hover {
    color: #007bff !important;
}
```

1 .sortable-ghost クラスは, SortableJS がドラッグ中に元の位置に自動的に付与するクラスである. 本スタイルでは opacity: 0.4 を指定することで, 元の位置に残る要素を半透明にし, ユーザーに「この位置にあった要素が移動中である」ことを視覚的に示している. これにより, 並べ替え操作中の空間的な把握が容易となる.

2 .sortable-drag クラスは, 現在ドラッグされている要素に対して自動的に付与される. 本スタイルでは transform: rotate(5deg) を指定し, 要素を時計回りに 5 度回転させている. これにより, ドラッグ中の要素が他の静的な要素と視覚的に区別され, ユーザーの操作対象が明確になる.

3 .drag-handle クラスは, ユーザーが要素をドラッグする際のつかみ部分として HTML 上で指定されている. 本スタイルでは, マウスがその要素の上にあるときに青色に変化するように設定しており, 視覚的に操作可能な領域であることを示している. !important を付与することで, 他のスタイル指定よりも優先的に適用されるようにしている.

図 3.19: UI 設計とスタイルの工夫.

3.2　ユーザー評価

ユーザーからのフィードバックでは, 旅行の持ち物やスケジュール, 行き先といった情報に関して, 再利用のしやすさや視認性の高さが評価される傾向が見られた. たとえば, 毎回同じ持ち物をチェックリストに書くのが手間であるため, 定番の持ち物を簡単に確認できることが便利であるという声があり, 持ち物情報のテンプレート化や保存機能のニーズが示された. また, 地図が表示されて, 日程ごとにスケジュールを個別に確認できる点が評価されており, 地図とスケジュールの連携表示は視覚的な理解を促進する機能として有効であると考えられる. これは, 地図情報とスケジュールが連動して表示されることで, 訪問先の位置関係や移動の流れを直感的に把握しやすくなることを意味している. 特に, 複数の目的地を巡る旅行では, 地図上での視覚的な確認が移動ルートの最適化や時間配分の検討に役立つと考えられる.

第4章　まとめ

本研究では，python で書かれたサーバー側 Web アプリケーションフレームワークである Django を開発環境として，視覚的な理解と効率的に旅行計画を行える旅行支援アプリを開発した.

開発手法としては, 機能ごとに優先順位を設定し, 重要度の高いものから順に実装を進めることで, ユーザーに対して価値を早期に提供することを重視した. このアプローチにより, 開発初期から研究室内での試用とフィードバックが可能となり, 得られた意見を反映しながら, より実用性と完成度の高いアプリへと発展させることができた.

機能面では地図の表示や色分け機能によって旅行全体の構成を視覚的に把握しやすくし，キーワード検索機能や地名検索機能により，目的地の絞り込みやデータの整理を効率的に行えるようにした. また, Django の認証機能を活用してログイン機能を実装し, ユーザーごとのデータ管理やアクセス制御を可能にした. さらに，チェックリストや日程ごとのタブ表示，移動時間の自動算出といった機能を組み合わせることで，旅行前の準備から当日の行動管理，旅行後の振り返りまでを一貫してサポートできる設計とした.

結果として，旅行計画を可視化し，ユーザーが直感的に全体像を把握できるアプリケーションを開発することができた.

謝辞

本研究を進めるにあたり, 御指導いただきました西谷滋人教授には心から感謝いたします. また，実際にアプリを使用していただき，貴重なご意見や改善のヒントをくださった西谷研究室の皆さまにも，深く感謝いたします．皆さまからの率直なフィードバックが, 本研究の質を高める大きな支えとなりました．お礼申し上げます.

参考文献

[1] 旅しお - https://tabisio.com/ (accessd on 18 Feb 2025).

[2] Django - https://www.djangoproject.com/ (accessd on 18 Feb 2025).

[3] Security in Django - https://docs.djangoproject.com/ja/6.0/topics/security/ (accessd on 18 Feb 2025).

[4] MTV を理解しよう- https://djangobrothers.com/tutorials/memo_app/mtv/ (accessd on 18 Feb 2025).

[5] Copilot - https://copilot.microsoft.com/ (accessd on 18 Feb 2025).

[6] Google Maps API - https://developers.google.com/maps/documentation/javascript/overview?hl=ja (accessd on 18 Feb 2025).

[7] Directions API - https://docs.mapbox.com/api/navigation/directions/ (accessd on 18 Feb 2025).

[8] Search Box API - https://docs.mapbox.com/api/search/search-box/ (accessd on 18 Feb 2025).

[9] Signals - https://docs.djangoproject.com/ja/6.0/topics/signals/ (accessd on 18 Feb 2025).

[10] Bootstrap - https://getbootstrap.jp/ (accessd on 18 Feb 2025).

[11] Font Awesome - https://fontawesome.com/ (accessd on 18 Feb 2025).