Workflow: Search form → Controller (TrainController@search) → search-results.blade.php → Database

This document explains, step by step, how the search flow works from the Home page form submission to rendering the search results. It is written for students and includes code references and data flow.

---

1) User submits the search form (home.blade.php)

- File: `resources/views/home.blade.php`
- The form posts to `route('trains.search')`, i.e., `POST /trains/search`.
- Important inputs:
  - `from_station` (string)
  - `to_station` (string, must be different from from_station)
  - `journey_date` (date)
  - `passengers` (1..4)

Example HTML (simplified):

```
<form id="searchForm" action="{{ route('trains.search') }}" method="POST">
  @csrf
```

```
  <select name="from_station" required> ... </select>
  <select name="to_station" required> ... </select>
  <input type="date" name="journey_date" required min="{{ date('Y-m-d') }}">
  <select name="passengers" required> ... </select>
  <button type="submit">Search Trains</button>
</form>
```

When the user clicks Submit, the browser makes a POST request to `/trains/search` with those fields.

---

2) Route resolves to controller method

- File: `routes/web.php`
- Relevant line:

```
Route::post('/trains/search', [App\Http\Controllers\TrainController::class,
'search'])->name('trains.search');
```

This means Laravel calls `TrainController::search(Request $request)`.

---

3) Controller validation and station lookup

- File: `app/Http/Controllers/TrainController.php`
- Method: `public function search(Request $request)`

Key tasks performed:

1. Validate request data:

```
$data = $request->validate([
    'from_station' => 'required|string',
    'to_station' => 'required|string|different:from_station',
    'journey_date' => 'required|date',
    'passengers' => 'required|integer|min:1|max:4',
]);
```

2. Resolve station names to database IDs using the `stations` table/model:

```
$from = Station::where('name', $data['from_station'])->first();
$to = Station::where('name', $data['to_station'])->first();
if (!$from || !$to) {
    return back()->withErrors(['stations' => 'Please select valid stations.'])-
>withInput();
}
```

- Models used here: `App\Models\Station`.
- Why? The routes in DB reference stations by ID, not by name, so the controller converts names to IDs.

---

4) Query database for matching active routes

- After stations are resolved, we query the `routes` table (via `App\Models\Route` model aliased as `TrainRoute`) to find active routes from `from_station_id` to `to_station_id`:

```
$routes = TrainRoute::with('train')
    ->where('from_station_id', $from->id)
    ->where('to_station_id', $to->id)
    ->where('is_active', true)
    ->get();
```

- `with('train')` eager-loads the related `train` for each route so we can display train info without extra queries.

We also compute the effective `journeyDate` as a Y-m-d string:

```
$journeyDate = Carbon::parse($data['journey_date'])->toDateString();
```

---

5) Compute availability and prepare the view-model

For each matching route, the controller computes:

- Base price (`route.base_price`).
- Total seats available on the train across its coaches.
- Seats already booked for this train, route, and date.
- Remaining available seats.

Code excerpt:

```
$results = [];
foreach ($routes as $route) {
    $price = (float) $route->base_price;

    // Sum of all seats across coaches for this train
    $totalSeats = Coach::where('train_id', $route->train_id)->sum('total_seats');

    // Count booked seats by joining BookingSeat->booking with filters
    $bookedCount = BookingSeat::whereHas('booking', function ($q) use ($route,
$journeyDate) {
        $q->where('train_id', $route->train_id)
          ->where('route_id', $route->id)
          ->whereDate('journey_date', $journeyDate)
          ->whereIn('booking_status', ['confirmed', 'pending']);
    })->count();

    $available = $totalSeats - $bookedCount;
    if ($available < 0) { $available = 0; }

    $results[] = [
        'id' => $route->train->id,
        'name' => $route->train->name,
        'number' => $route->train->number,
        'from' => $from->name,
        'to' => $to->name,
        'departure' => $route->departure_time ? $route->departure_time-
>format('H:i') : '--:--',
        'arrival' => $route->arrival_time ? $route->arrival_time->format('H:i') :
'--:--',
        'duration' => sprintf('%dh %02dm', intdiv($route->duration_minutes, 60),
$route->duration_minutes % 60),
        'distance' => $route->distance_km . ' km',
        'price' => $price,
        'available_seats' => $available,
        'route_id' => $route->id,
```

```
    ];
}
```

- Models involved: `Coach` (to sum total seats), `BookingSeat` and its relationship to `Booking` (to count booked seats).
- Why use `whereHas('booking', ...)`? Because `booking_seats` records link to `bookings` which carry date/route/train filters.

Finally, return the view with the computed list and the original search params:

```
return view('trains.search-results', [
    'trains' => $results,
    'searchParams' => $data,
]);
```

---

6) Rendering results (search-results.blade.php)

- File: `resources/views/trains/search-results.blade.php`
- The Blade view receives:
  - `trains`: an array of associative arrays prepared by the controller.
  - `searchParams`: the exact form inputs, useful to display the route, date, and passenger count.

Important parts:

```
<p><strong>Route:</strong> {{ $searchParams['from_station'] }} → {{
$searchParams['to_station'] }}</p>
<p><strong>Date:</strong> {{ \Carbon\Carbon::parse($searchParams['journey_date'])-
>format('l, F j, Y') }}</p>
<p><strong>Passengers:</strong> {{ $searchParams['passengers'] }}</p>

@foreach($trains as $t)
  <div class="card">
    <div class="header">
      <div>
        <div class="name">{{ $t['name'] }}</div>
        <div class="muted">{{ $t['number'] }} • {{ $t['distance'] }}</div>
      </div>
      <div class="time">{{ $t['departure'] }} - {{ $t['arrival'] }}</div>
    </div>
```

```
    <div class="muted"><strong>Duration:</strong> {{ $t['duration'] }}</div>
    <div style="display:flex; justify-content:space-between; align-items:center;">
      <div class="price">৳{{ number_format($t['price']) }}</div>
      <a class="btn"
         href="{{ route('trains.seats', [
                'id' => $t['id'],
                'journey_date' => $searchParams['journey_date'],
                'passengers' => $searchParams['passengers'],
                'route_id' => $t['route_id'] ?? null
         ]) }}">
         Select Seats
      </a>
    </div>
  </div>
@endforeach
```

- Note the link to `trains.seats` carries forward `journey_date`, `passengers`, and `route_id`. This preserves context for the next page.

---

7) Database tables and relationships involved

- `stations` (via Station model): stores station names and IDs.
- `routes` (via Route model): stores active routes with from_station_id, to_station_id, times, base_price, etc.
- `trains` (via Train model): each route belongs to a train.
- `coaches` (via Coach model): used to sum total seats available per train.
- `booking_seats` and `bookings` (via BookingSeat and Booking models): used to count existing bookings for the given date/route/train.

Relationships used:

- Route belongsTo Train (`Route::train()`)
- BookingSeat belongsTo Booking (`BookingSeat::booking()`)
- Booking belongsTo Train/Route; Booking hasMany BookingSeat

---

8) End-to-end flow summary

1. User submits search form.

2. Route maps to `TrainController@search`.
3. Controller validates inputs → resolves station names to IDs.
4. Queries matching active routes, eager-loads train.
5. Computes price, total seats, booked seats, available seats.
6. Sends a clean view-model to `search-results.blade.php`.
7. Blade renders results and provides link to select seats, passing date/passenger count/route id.

This is a classic MVC flow: input → controller logic → DB queries via models → render Blade with data → user proceeds.