- Workflow: Controller (TrainController@search) + search-results.blade.php + Database
  - 1) User action and HTTP request
  - 2) Controller: TrainController@search
    - 2.1 Validate input
    - 2.2 Map station names → IDs
    - 2.3 Query routes and load trains
    - 2.4 Compute availability and pricing per route
    - 2.5 Return view with data
  - 3) View: resources/views/trains/search-results.blade.php
    - 3.1 Header with search summary
    - 3.2 Loop trains and render cards
  - 4) Database relations used in this flow
  - 5) Full sequence diagram (text)

Workflow: Controller (TrainController@search) + search-results.blade.php + Database

This document explains, end‑to‑end, how a search request flows from the browser to the controller, queries the database models, and renders the results view. It's written for students, with clear steps, diagrams-in-text, and code references from this project.

---

1) User action and HTTP request

- The user opens the Home page and submits the search form with:
  - **from_station** (string)
  - **to_station** (string)
  - **journey_date** (date)
  - **passengers** (1..4)
- The form posts to the named route `trains.search` which maps to `TrainController@search` (HTTP POST).

Route (routes/web.php):

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

---

2) Controller: TrainController@search

File: app/Http/Controllers/TrainController.php

2.1 Validate input

```
$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',
]);
```

- Protects against invalid data (e.g., same stations, missing date, passengers out of range).
- If validation fails, Laravel redirects back with errors.

2.2 Map station names → IDs

```
$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();
}
```

- We translate provided station names into `stations` table records (via `Station` model).

2.3 Query routes and load trains

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

- `TrainRoute` is `App\Models\Route` alias; it belongs to a `Train`.
- We fetch all active routes matching from→to.

2.4 Compute availability and pricing per route

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

$results = [];
foreach ($routes as $route) {
    $price = (float) $route->base_price;  // base fare from database

    $totalSeats = Coach::where('train_id', $route->train_id)->sum('total_seats');

    $bookedCount = BookingSeat::whereHas('booking', function ($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 = max(0, $totalSeats - $bookedCount);

    $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,
    ];
}
```

- We use 3 models together:
  - `Coach` to count total seats for a train
  - `BookingSeat` + `booking` relation to count already booked seats for that date/route
  - `Route` to get schedule/price
- This is pure read-only logic that composes DB data into a flat array for the view.

2.5 Return view with data

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

- Passes both `trains` (our computed list) and the original `searchParams` to the Blade.

---

3) View: resources/views/trains/search-results.blade.php

3.1 Header with search summary

```
<div class="card">
  <h2>Search Results</h2>
  <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>
</div>
```

- Confirms to the user what they searched for.

3.2 Loop trains and render cards

```
@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" style="margin-bottom:8px;">
      <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
```

- Each train card displays name, number, duration, price, and a button that carries forward:
  - journey_date
  - passengers
  - route_id
- This preserves the user's context into the next step.

---

4) Database relations used in this flow

- `Route` (App\Models\Route): belongs to a `Train`; has schedule fields, base_price, distance_km, duration_minutes, from_station_id, to_station_id, is_active.
- `Train` (App\Models\Train): has many `Route` and `Booking`.
- `Coach` (App\Models\Coach): belongs to Train; has total_seats.
- `BookingSeat` (App\Models\BookingSeat): belongs to Booking; used to count existing seats for the day.
- `Booking` (App\Models\Booking): has `booking_status` used to include pending/confirmed in the count.

The controller uses these to compute an up-to-date view of availability for the selected date.

---

5) Full sequence diagram (text)

Browser (POST /trains/search) → TrainController@search → Validate input → Map Station names to rows (stations table) → Query Route + Train → For each Route: sum Coach seats; count BookingSeat for date/route; compute availability and price → Return Blade with {trains, searchParams} → Blade loops and renders train cards + next-step link to seats

That’s the entire path from a user search to a rendered list of trains, grounded in the database.