Many to Many

To define a many-to-many relationship between the **DIARY_ENTRIES** and **EMOTIONS**, you need to set up a pivot table (*DIARY_ENTRY_EMOTIONS*).

Create the Models and Migrations

We need three tables: `diary_entries`, `emotions`, and a pivot table `diary_entry_emotions`.

`emotions`

1. Run the following command to create the model and migration for the emotions table

   > ./vendor/bin/sail artisan make:model Emotion -m

2. Open `Emotion.php` and define the properties

```
class Emotion extends Model
{
    use HasFactory;
    protected $table = 'emotions';
    protected $fillable = ['name', 'description'];
    protected $casts = [
        'created_at' => 'datetime',
        'updated_at' => 'datetime',
    ];
}
```

   Note: The protected $casts property on a model is used to specify how attributes should be cast to native types when you access them

3. Go to migration file for the `create_emotions_table.php` and define the schema

```
public function up(): void
{
    Schema::create('emotions', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->text('description')->nullable();
        $table->timestamps();
    });
}
```

4. Run a migration file:

```
./vendor/bin/sail artisan migrate
```

5. Seed the Emotions Data: create a seeder for the emotions table

   > ./vendor/bin/sail artisan make:seeder EmotionSeeder

   This will create a new seeder class called EmotionSeeder in the `database/seeders` directory.

6. Seed the Emotions Data: Define the Data in the Seeder by opening the newly created `EmotionSeeder.php` file located in the database/seeders directory and adding the following codes.

   > `use Illuminate\Support\Facades\DB;`

   > `use Carbon\Carbon;`

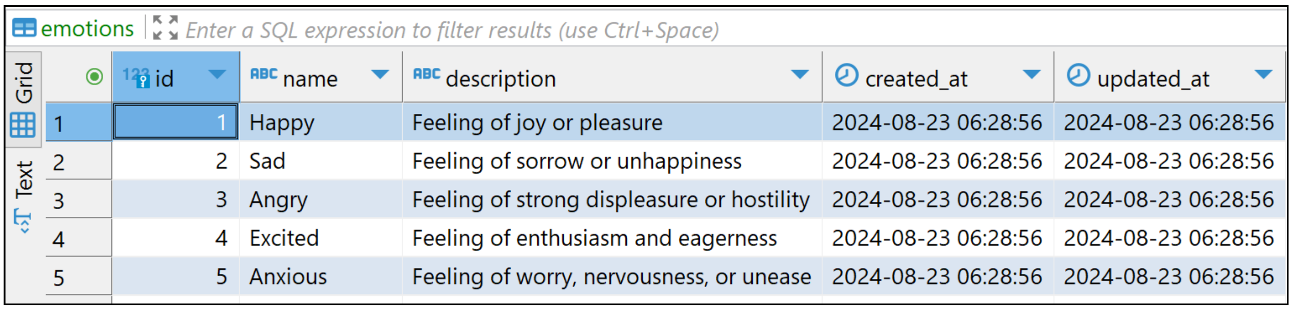
   ```
   class EmotionSeeder extends Seeder
   {
       public function run()
       {
           // Insert emotion data with timestamps
           DB::table('emotions')->insert([
               ['name' => 'Happy', 'description' => 'Feeling of joy or 
   pleasure', 'created_at' => Carbon::now(), 'updated_at' => Carbon::now()],
               ['name' => 'Sad', 'description' => 'Feeling of sorrow or 
   unhappiness', 'created_at' => Carbon::now(), 'updated_at' => Carbon::now()],
               ['name' => 'Angry', 'description' => 'Feeling of strong 
   displeasure or hostility', 'created_at' => Carbon::now(), 'updated_at' => 
   Carbon::now()],
               ['name' => 'Excited', 'description' => 'Feeling of enthusiasm 
   and eagerness', 'created_at' => Carbon::now(), 'updated_at' => 
   Carbon::now()],
               ['name' => 'Anxious', 'description' => 'Feeling of worry, 
   nervousness, or unease', 'created_at' => Carbon::now(), 'updated_at' => 
   Carbon::now()],
           ]);
       }
   }
   ```

   Note: Carbon is a date and time library included in Laravel that makes working with dates easier.

7. Run the Seeder

   > ./vendor/bin/sail artisan db:seed --class=EmotionSeeder

8. Check if the data is inserted



emotions | Enter a SQL expression to filter results (use Ctrl+Space)

	id	name	description	created_at	updated_at
1	1	Happy	Feeling of joy or pleasure	2024-08-23 06:28:56	2024-08-23 06:28:56
2	2	Sad	Feeling of sorrow or unhappiness	2024-08-23 06:28:56	2024-08-23 06:28:56
3	3	Angry	Feeling of strong displeasure or hostility	2024-08-23 06:28:56	2024-08-23 06:28:56
4	4	Excited	Feeling of enthusiasm and eagerness	2024-08-23 06:28:56	2024-08-23 06:28:56
5	5	Anxious	Feeling of worry, nervousness, or unease	2024-08-23 06:28:56	2024-08-23 06:28:56

`diary_entry_emotions`

1. Run the following command to create the migration for the 'diary_entry_emotions' table

> ./vendor/bin/sail artisan make:migration create_diary_entry_emotions_table --create=diary_entry_emotions

2. Go to migration file for the `create_diary_entry_emotions_table.php` and define the schema

```
public function up(): void
{
    Schema::create('diary_entry_emotions', function (Blueprint $table) {
        $table->id();
        $table->foreignId('diary_entry_id')->constrained('diary_entries')-
>onDelete('cascade');
        $table->foreignId('emotion_id')->constrained('emotions')-
>onDelete('cascade');
        $table->integer('intensity');  // Assuming intensity is a scale of
emotion, e.g. from 1 to 10.
        $table->timestamps();
    });
}
```

3. Run a migration file:

```
./vendor/bin/sail artisan migrate
```

Define Laravel Many to Many Relationship

In a many-to-many relationship, Laravel Eloquent automatically manages the pivot table (in this case, diary_entry_emotions) without the need for a specific model. You only define the relationship in the related models (DiaryEntry and Emotion), and Laravel will handle inserting and retrieving pivot data using the belongsToMany method.

Without Type Hinting

1. Define **belongsToMany** relationship: Go to `DiaryEntry` model

```
public function emotions()
{
    return $this->belongsToMany(Emotion::class, 'diary_entry_emotions',
'diary_entry_id', 'emotion_id')
                ->withPivot('intensity')
                ->withTimestamps();
}
```

Explaination

`withPivot('intensity')`

- Purpose: The withPivot() method is used to specify any additional columns in the pivot table that you want to access or manipulate when working with the relationship.
- Usage: By using withPivot('intensity'), you tell Laravel to retrieve the value of this intensity column whenever you access the `emotions()` relationship.
- Example: You can retrieve the intensity value associated with a particular emotion for a diary entry like this:

  ```
  $diaryEntry = DiaryEntry::find(1);

  foreach ($diaryEntry->emotions as $emotion) {
      echo $emotion->pivot->intensity; // Access intensity from pivot
  table
  }
  ```

`withTimestamps()`

- Purpose: The withTimestamps() method is used to automatically handle the created_at and updated_at columns in the pivot table. This is particularly useful when you want to track when a relationship was created or last updated.
- Usage: By using withTimestamps(), Laravel will automatically set the created_at and updated_at fields in the pivot table when adding or updating records in the relationship.

2. Define **belongsToMany** relationship: Go to `Emotion` model

```
public function diaryEntries()
{
    return $this->belongsToMany(DiaryEntry::class, 'diary_entry_emotions')
                ->withPivot('intensity')
                ->withTimestamps();
}
```

When defining a many-to-many relationship in Laravel, the foreign key names in the pivot table can be left blank because Laravel automatically assumes default naming conventions.

Update `DiaryEntryController` Controller

Update DiaryEntryController to Handle Emotions

1. Go to `DiaryEntryController` and modify the existing methods to handle emotions.:
   - Add this line: `use App\Models\Emotion;`
   - Index

     ```
     public function index()
     {
         $diaryEntries = Auth::user()->diaryEntries()->with('emotions')-
     >get();
         return view('diary.index', compact('diaryEntries'));
     }
     ```

     The with('emotions') method is used to eager-load the emotions related to each diary entry. This reduces the number of queries executed and improves performance.
   - Show the form for creating a new resource.

     ```
     public function create()
     {
         $emotions = Emotion::all(); // Fetch all emotions for selection
         return view('diary.create', compact('emotions')); // Pass emotions
     to the view
     }
     ```
   - Store a newly created resource in storage

     ```
     public function store(Request $request)
     {
         // Validate the request
         $validated = $request->validate([
             'date' => 'required|date',
             'content' => 'required|string',
             'emotions' => 'array', // Validate emotions as an array
             'intensity' => 'array', // Validate intensity as an array
         ]);

         // Create the diary entry
         $diaryEntry = Auth::user()->diaryEntries()->create([
             'date' => $validated['date'],
             'content' => $validated['content'],
         ]);

         // Handle emotions and intensities
     ```

```
    if (!empty($validated['emotions']) &&
!empty($validated['intensity'])) {
        foreach ($validated['emotions'] as $emotionId) {
            $intensity = $validated['intensity'][$emotionId] ?? null;

            // Attach emotions and intensities to the diary entry
            $diaryEntry->emotions()->attach($emotionId, ['intensity' =>
$intensity]);
        }
    }

    return redirect()->route('diary.index')->with('status', 'Diary
entry added successfully!');
}
```

Explaination:

1. This creates a new DiaryEntry record associated with the currently authenticated user. The `diaryEntries` method is an Eloquent relationship that retrieves the diary entries belonging to the user. The `create` method then inserts a new record into the diary_entries table with the validated date and content. (By calling `Auth::user()->diaryEntries()`, you're essentially telling Laravel that you want to create a new DiaryEntry for the user. Laravel will automatically set the foreign key (user_id) in the DiaryEntry to the ID of the currently authenticated user.)
2. Check if Emotions and Intensities are Provided: The method first ensures that both emotions and intensity arrays are not empty.
3. Then attach Emotions to Diary Entry: For each **emotionId** in the **emotions array**, it retrieves the corresponding intensity from the **intensity array**. The `attach` method is *used to add records to the pivot table* (diary_entry_emotions in this case), associating the emotionId with the newly created diaryEntry. The intensity value is also saved in the pivot table. Note: By default, attach inserts a new record into the pivot table. It does not remove or update existing records.

- Show the form for editing

```
public function edit(string $id)
{
    $diaryEntry = Auth::user()->diaryEntries()->with('emotions')-
>findOrFail($id);
    $emotions = Emotion::all(); // you must have a model called Emotion
to fetch all emotions
    return view('diary.edit', compact('diaryEntry', 'emotions'));
}
```

- Update the specified resource in storage

```php
public function update(Request $request, string $id)
{
    // Validate the request
    $validated = $request->validate([
        'date' => 'required|date',
        'content' => 'required|string',
        'emotions' => 'array', // Validate emotions as an array
        'intensity' => 'array', // Validate intensity as an array
    ]);

    // Find and update the diary entry
    $diaryEntry = Auth::user()->diaryEntries()->findOrFail($id);
    $diaryEntry->update([
        'date' => $validated['date'],
        'content' => $validated['content'],
    ]);

    // Sync emotions and intensities
    if (!empty($validated['emotions'])) {
        $emotions = [];
        foreach ($validated['emotions'] as $emotionId) {
            $intensity = $validated['intensity'][$emotionId] ?? null;
            $emotions[$emotionId] = ['intensity' => $intensity];
        }
        $diaryEntry->emotions()->sync($emotions);
    } else {
        // If no emotions are selected, clear all associated emotions
        $diaryEntry->emotions()->sync([]);
    }

    return redirect()->route('diary.index')->with('status', 'Diary
entry updated successfully!');
}
```

Explaination:

- `!empty($validated['emotions'])`: Checks if there are any selected emotions.
- `$emotions = [];`: Initializes an empty array to hold the emotions and their intensities.
- `foreach ($validated['emotions'] as $emotionId)`: Iterates over the selected emotions.
  - `$intensity = $validated['intensity'][$emotionId] ?? null;`: Retrieves the intensity for the current emotion. If no intensity is provided, it defaults to null.
  - `$emotions[$emotionId] = ['intensity' => $intensity];`: Sets the intensity for the emotion in the $emotions array.
- `$diaryEntry->emotions()->sync($emotions);`: Syncs the emotions with the diary entry. This updates the pivot table to reflect the changes:
  - Added Emotions: If new emotions are selected, they are added.
  - Removed Emotions: If some emotions are no longer selected, they are removed.
  - Updated Intensities: Existing emotions have their intensities updated.

- `$diaryEntry->emotions()->sync([]);`: If no emotions are selected, it clears all emotions associated with the diary entry.

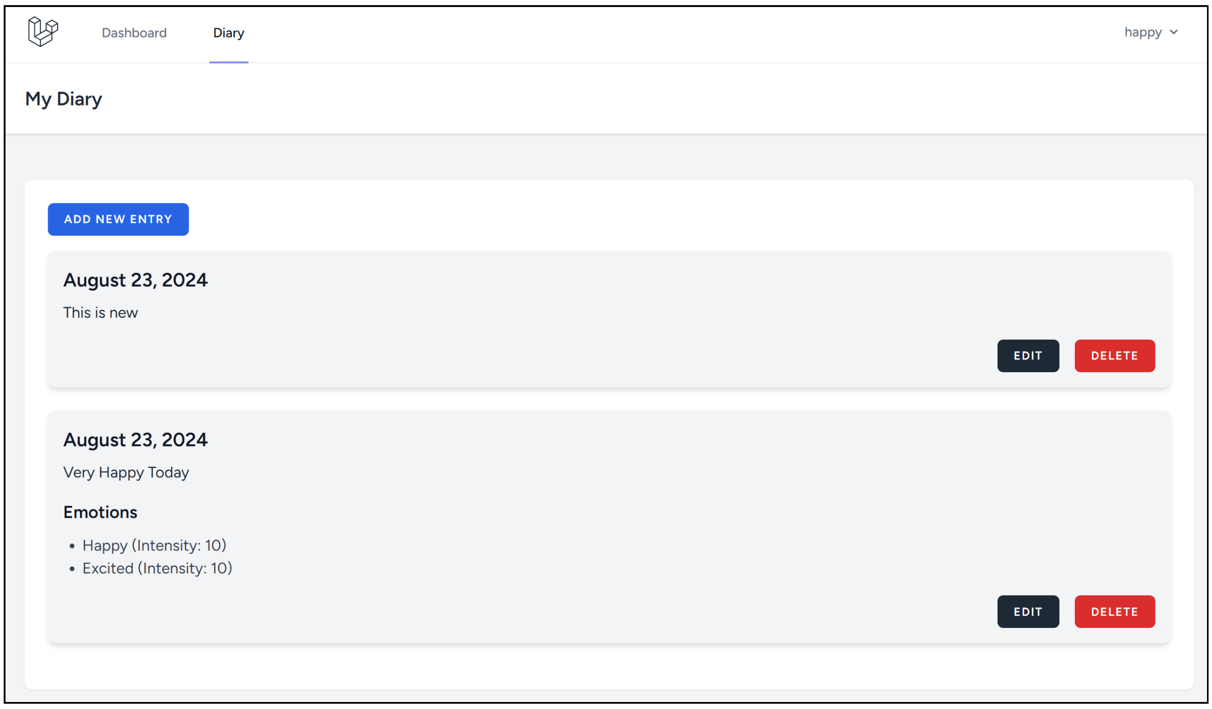
- Destroy (No Update)

  In Laravel, when you delete a record from a model that has a many-to-many relationship, you typically need to ensure that the related pivot table entries are also handled correctly. For the scenario where deleting a DiaryEntry should also delete its associated entries in the diary_entry_emotions pivot table, Laravel's default behavior handles this automatically if you have properly set up the database relationships with cascading delete.

Views

Modify the corresponding Blade views for each of the methods in the DiaryEntryController.

1. Modify a blade file `resources/views/diary/index.blade.php` by adding an emotion information section within the `@foreach($diaryEntries as $entry)` loop.



My Diary

August 23, 2024

This is new

August 23, 2024

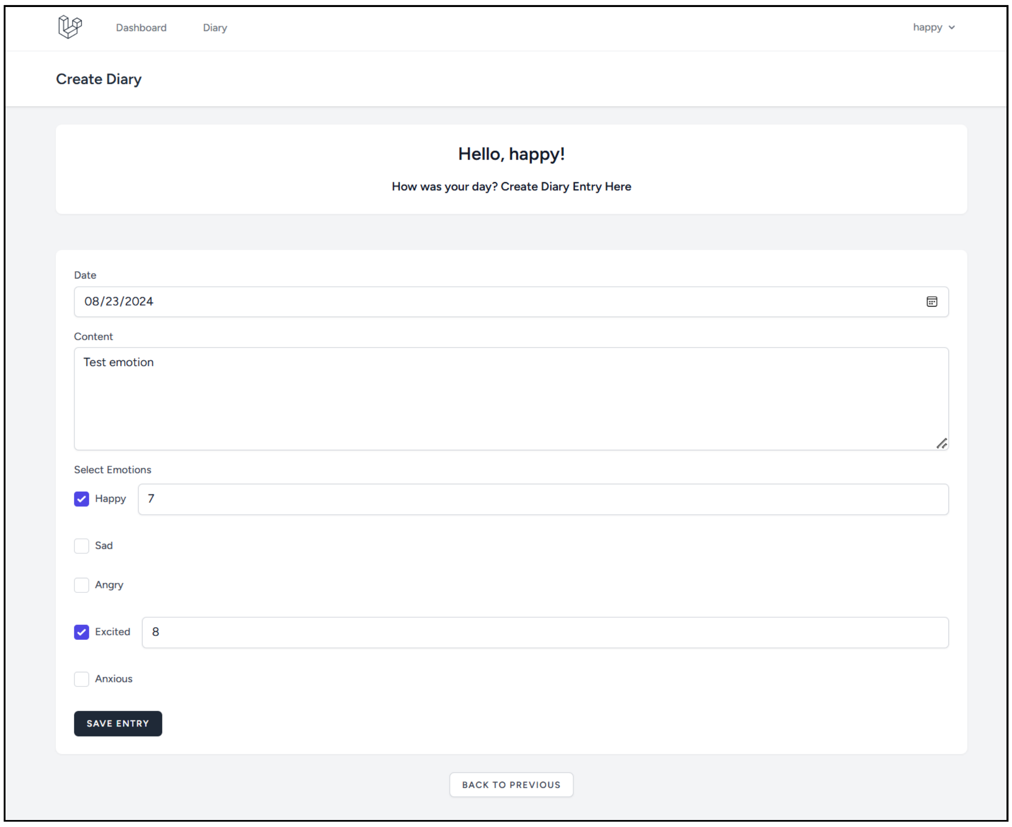
Very Happy Today

Emotions

- Happy (Intensity: 10)
- Excited (Intensity: 10)

```
<!-- Display emotions -->
@if($entry->emotions->isNotEmpty())
    <div class="mt-4">
        <h4 class="text-lg font-semibold mb-2">Emotions</h4>
        <ul class="list-disc pl-5">
            @foreach($entry->emotions as $emotion)
                <li class="text-gray-700 dark:text-gray-300">
                    {{ $emotion->name }} (Intensity: {{ $emotion->pivot-
>intensity }})
                </li>
            @endforeach
        </ul>
    </div>
@endif
```

2. Modify the blade file `resources/views/diary/create.blade.php` by adding a section within the existing form for selecting emotions. Place it directly under the `<form method="POST" action="{{ route('diary.store') }}">` tag.



```
{{-- Emotion --}}
<div class="mb-4">
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300
mb-2">Select Emotions</label>

    <!-- Grid layout for emotions -->
    <div class="grid grid-cols-1 gap-4">
        @foreach ($emotions as $emotion)
            <div class="flex items-center mb-4">
                <!-- Checkbox and label container -->
                <input type="checkbox" id="emotion_{{ $emotion->id }}"
name="emotions[]" value="{{ $emotion->id }}" class="h-5 w-5 text-indigo-600
border-gray-300 rounded dark:bg-gray-700 dark:border-gray-600
dark:focus:ring-indigo-600" onchange="toggleIntensityInput({{ $emotion->id
}})">
                <label for="emotion_{{ $emotion->id }}" class="ml-2 text-sm
font-medium text-gray-700 dark:text-gray-300">{{ $emotion->name }}</label>

                <!-- Intensity input container, initially hidden -->
                <div class="ml-4 hidden flex-1" id="intensity_container_{{
$emotion->id }}">
                    <input type="number" name="intensity[{{ $emotion->id
}}]" class="w-full border-gray-300 rounded-md shadow-sm dark:bg-gray-700
dark:text-gray-100 dark:border-gray-600 focus:ring-indigo-500 focus:border-
```

```
indigo-500" placeholder="Intensity" min="1" max="10">
                    </div>
            </div>
        @endforeach
    </div>

    @error('emotions')
        <div class="text-red-500 text-sm mt-2">{{ $message }}</div>
    @enderror
</div>

<script>
    // Function to toggle the visibility of the intensity input
    function toggleIntensityInput(emotionId) {
        var checkbox = document.getElementById('emotion_' + emotionId);
        var intensityContainer =
document.getElementById('intensity_container_' + emotionId);

        // Show intensity input if checkbox is checked
        if (checkbox.checked) {
            intensityContainer.classList.remove('hidden');
        } else {
            intensityContainer.classList.add('hidden');
        }
    }
</script>
```

3. Modify a blade file `resources/views/diary/edit.blade.php` by adding a section within the existing form for editing. Place it directly under the `<form method="POST" action="{{ route('diary.update', $diaryEntry) }}">` tag.

```
 {{-- Emotion --}}
 <div class="mb-4">
     <label class="block text-sm font-medium text-gray-700 dark:text-gray-
300 mb-2">Select Emotions</label>

     <!-- Grid layout for emotions -->
     <div class="grid grid-cols-1 gap-4">
         @foreach ($emotions as $emotion)
             <div class="flex items-center mb-4">
                 <!-- Checkbox and label container -->
                 <input type="checkbox" id="emotion_{{ $emotion->id }}"
name="emotions[]" value="{{ $emotion->id }}" class="h-5 w-5 text-indigo-600
border-gray-300 rounded dark:bg-gray-700 dark:border-gray-600
dark:focus:ring-indigo-600"
                     {{ in_array($emotion->id, old('emotions', $diaryEntry-
>emotions->pluck('id')->toArray())) ? 'checked' : '' }}
                     onchange="toggleIntensityInput({{ $emotion->id }})">
                 <label for="emotion_{{ $emotion->id }}" class="ml-2 text-sm
font-medium text-gray-700 dark:text-gray-300">{{ $emotion->name }}</label>
```

```
                <!-- Intensity input container, initially hidden -->
                <div class="ml-4 {{ in_array($emotion->id, old('emotions',
$diaryEntry->emotions->pluck('id')->toArray())) ? '' : 'hidden' }}"
id="intensity_container_{{ $emotion->id }}">
                    <input type="number" name="intensity[{{ $emotion->id
}}]" class="w-full border-gray-300 rounded-md shadow-sm dark:bg-gray-700
dark:text-gray-100 dark:border-gray-600 focus:ring-indigo-500 focus:border-
indigo-500" placeholder="Intensity" min="1" max="10"
                        value="{{ old('intensity.' . $emotion->id,
$diaryEntry->emotions->find($emotion->id)->pivot->intensity ?? '') }}">
                </div>
            </div>
        @endforeach
    </div>

    @error('emotions')
        <div class="text-red-500 text-sm mt-2">{{ $message }}</div>
    @enderror
</div>

<script>
    // Function to toggle the visibility of the intensity input
    function toggleIntensityInput(emotionId) {
        var checkbox = document.getElementById('emotion_' + emotionId);
        var intensityContainer =
document.getElementById('intensity_container_' + emotionId);

        // Show intensity input if checkbox is checked
        if (checkbox.checked) {
            intensityContainer.classList.remove('hidden');
        } else {
            intensityContainer.classList.add('hidden');
        }
    }

    // Initialize visibility based on existing emotions

document.querySelectorAll('input[type="checkbox"]').forEach(function(checkbo
x) {
        toggleIntensityInput(checkbox.value);
    });
</script>
```

Explaination:

1. When you specify $diaryEntry in the form action ({{ route('diary.update', $diaryEntry) }}), you are passing the specific diaryEntry instance (or its ID) to the named route. For example, it will generate `<form method="POST" action="http://localhost/diary/1">`