```
$subscriber->tags()->sync(
  $data->tags->toCollection()->pluck('id')
);
```

If you remember, the `$tags` property in the `SubscriberData` class is a `DataCollection`. This class comes from the laravel-data package, and it can be converted into a Laravel collection by calling the `toCollection` method. Since `tags` is a `belongsToMany` relationship in the `Subscriber` model, we can use the sync method:

- It will attach every tag from the given collection.
- And detach every other tag that the subscriber had earlier (in case of an update).

Updating A Subscriber

Now let's see how the `update` method looks in the `SubscriberController`:

```
public function update(
  SubscriberData $data,
  Request $request
): RedirectResponse {
  UpsertSubscriberAction::execute($data, $request->user());

  return Redirect::route('subscribers.index');
}
```

As you can see, it's the same as the `store`. That's because the action takes care of both actions. Why doesn't it have a `Subscriber $subscriber` argument from a route binding?

- The frontend will send the ID in the request.
- The `SubscriberData` loads this ID alongside the other attributes.
- The `updateOrCreate` in the action will run an `update` query.