escola superior
de tecnologia e gestão
instituto politécnico
de leiria

Laravel

Authentication and Authorization

Marco Monteiro

Contributors

- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

Summary

1. Authentication
2. Authorization

1 – AUTHENTICATION

Laravel Authentication

- Laravel authentication core is made of "**guards**" and "**providers**"
  - **Guards** – define how users are authenticated for each request. Examples: web session guard (default), token, …
  - **Providers** - define how users are retrieved from your persistent storage. Examples: Eloquent (default), active directory (LDAP server), …
- Authentication configuration is located at `config/auth.php`.
- Laravel includes **starter-kits** that generates the resources (controllers, views, etc.) required for the authentication layer (login, register, recover password, etc).
  - Laravel **Breeze** – simple and minimal. Uses Tailwind CSS
  - Laravel Jetstream – more complex and more functionalities
  - Laravel Fortify – headless (no UI) backend

Laravel Breeze – How to use

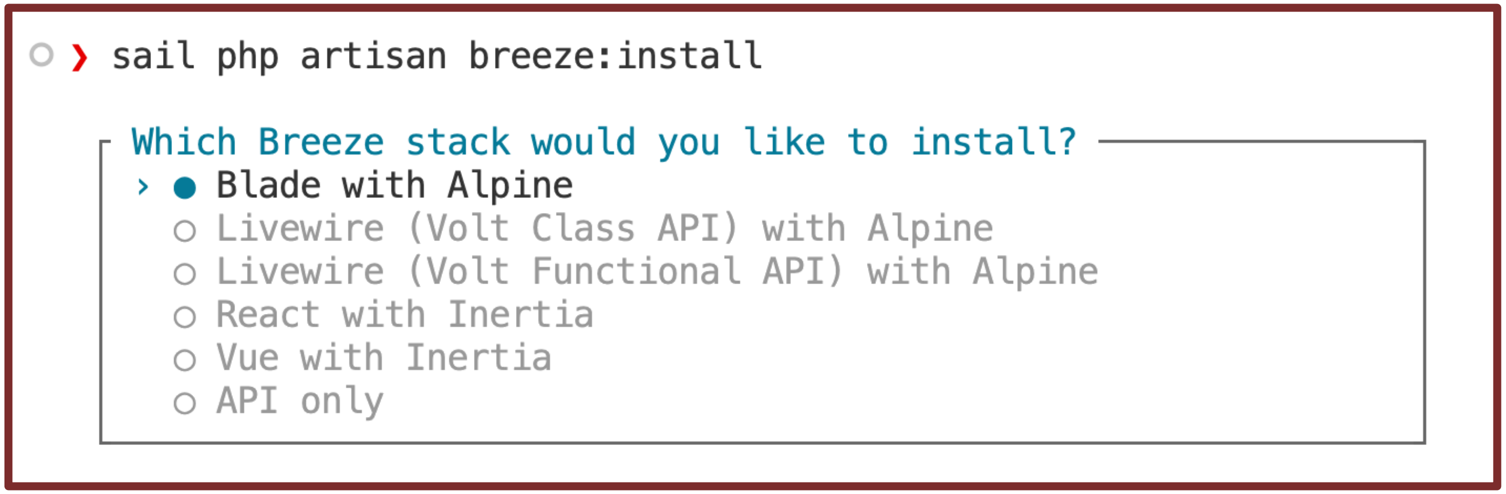
1 - Install Laravel/breeze package:

```
composer require laravel/breeze --dev
```

2 - Generate authentication scaffold

```
php artisan breeze:install
```

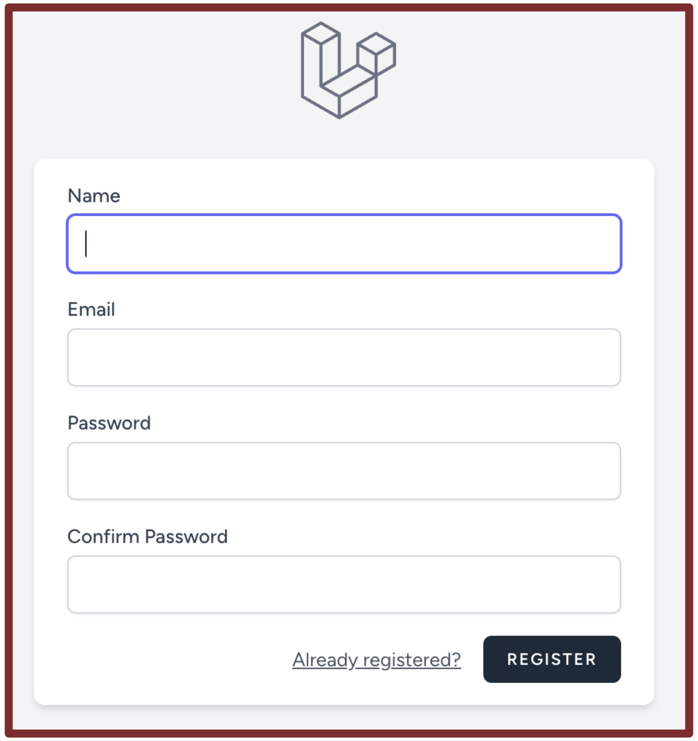
- Several frontend stacks are available - for current course, choose **Blade**

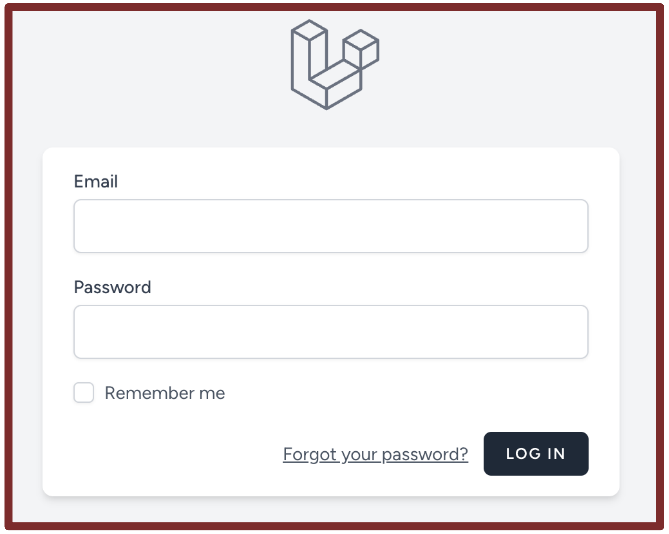


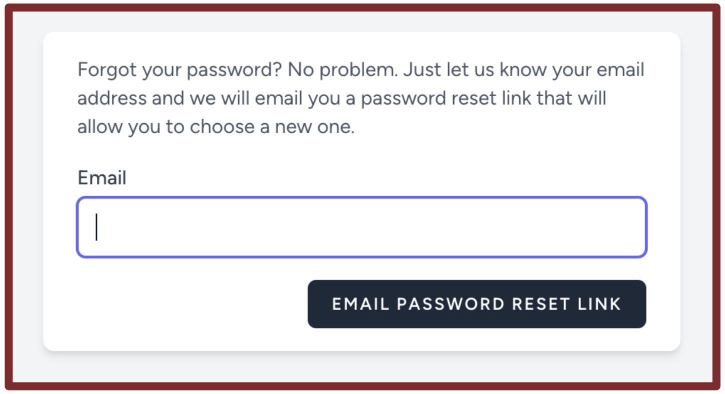
```
○ ❯ sail php artisan breeze:install

  ┌ Which Breeze stack would you like to install? ─┐
  │ › ● Blade with Alpine                          │
  │   ○ Livewire (Volt Class API) with Alpine      │
  │   ○ Livewire (Volt Functional API) with Alpine │
  │   ○ React with Inertia                         │
  │   ○ Vue with Inertia                           │
  │   ○ API only                                   │
  └────────────────────────────────────────────────┘
```

Laravel Breeze

- Laravel Breeze generates a **login**, **registration**, **email verification**, **password confirmation**, **reset password**, **profile**, **change password** and a **logout**.





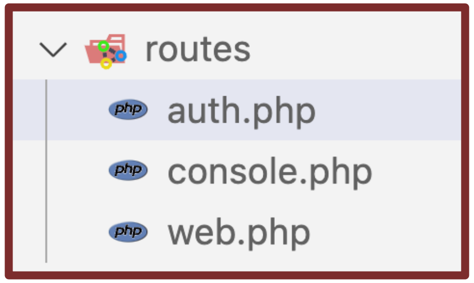

Forgot your password? No problem. Just let us know your email address and we will email you a password reset link that will allow you to choose a new one.

Laravel Breeze - Routes

- Laravel Breeze adds the following code to `routes/web.php`:

```php
require __DIR__.'/auth.php';
```

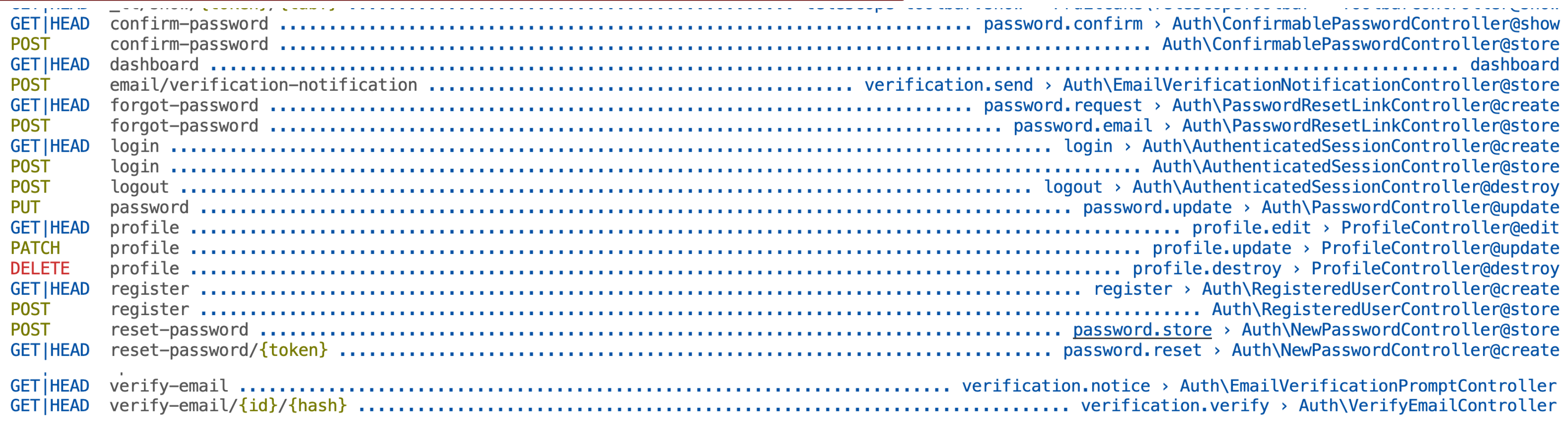
which imports the routes defined on the `routes/auth.php` file:



```
routes
    auth.php
    console.php
    web.php
```

- Routes created by Laravel Breeze:

```
php artisan route:list
```

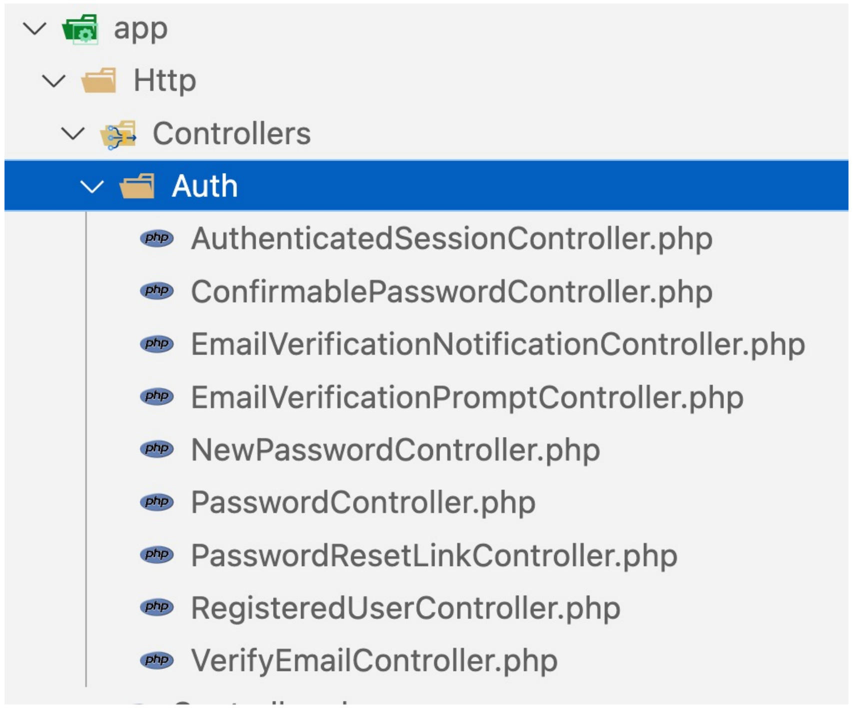


```
GET|HEAD   confirm-password ................................................ password.confirm › Auth\ConfirmablePasswordController@show
POST       confirm-password ................................................................. Auth\ConfirmablePasswordController@store
GET|HEAD   dashboard ........................................................................................................ dashboard
POST       email/verification-notification ..................................... verification.send › Auth\EmailVerificationNotificationController@store
GET|HEAD   forgot-password ................................................ password.request › Auth\PasswordResetLinkController@create
POST       forgot-password ................................................... password.email › Auth\PasswordResetLinkController@store
GET|HEAD   login ................................................................ login › Auth\AuthenticatedSessionController@create
POST       login ..................................................................... Auth\AuthenticatedSessionController@store
POST       logout ............................................................. logout › Auth\AuthenticatedSessionController@destroy
PUT        password ............................................................. password.update › Auth\PasswordController@update
GET|HEAD   profile ................................................................... profile.edit › ProfileController@edit
PATCH      profile ................................................................. profile.update › ProfileController@update
DELETE     profile ............................................................... profile.destroy › ProfileController@destroy
GET|HEAD   register ..................................................... register › Auth\RegisteredUserController@create
POST       register ................................................................. Auth\RegisteredUserController@store
POST       reset-password .................................................. password.store › Auth\NewPasswordController@store
GET|HEAD   reset-password/{token} ......................................... password.reset › Auth\NewPasswordController@create

GET|HEAD   verify-email ............................................... verification.notice › Auth\EmailVerificationPromptController
GET|HEAD   verify-email/{id}/{hash} .......................................... verification.verify › Auth\VerifyEmailController
```

Laravel Breeze – Models & Controllers

- Uses the **model** `app/Models/`**`User`**`.php` without any modification

- Generates the **controllers** on folder `app/Http/Controllers/Auth`



```
app
  Http
    Controllers
      Auth
        AuthenticatedSessionController.php
        ConfirmablePasswordController.php
        EmailVerificationNotificationController.php
        EmailVerificationPromptController.php
        NewPasswordController.php
        PasswordController.php
        PasswordResetLinkController.php
        RegisteredUserController.php
        VerifyEmailController.php
```

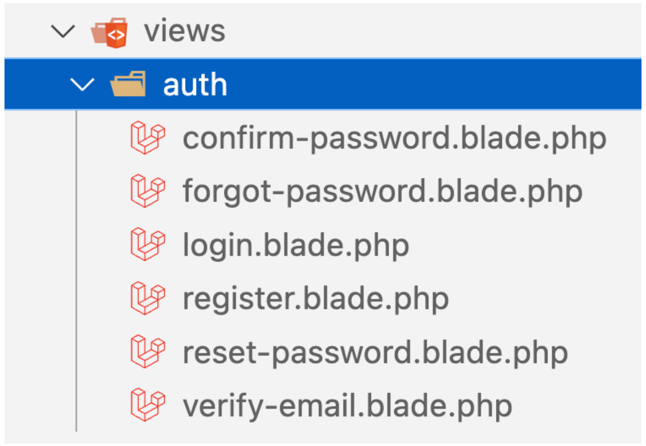
AuthenticatedSessionController
handles the login and logout

. . .

RegisteredUserController
handles the registration of a new user

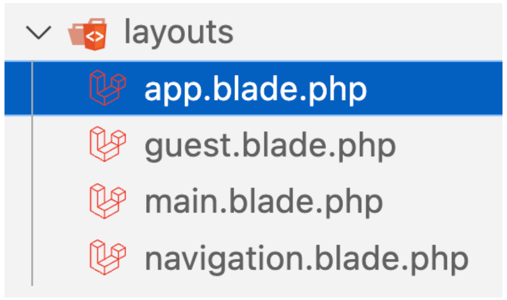
Laravel Breeze – Views

- Generates **views** on folder `resources/views/auth`:



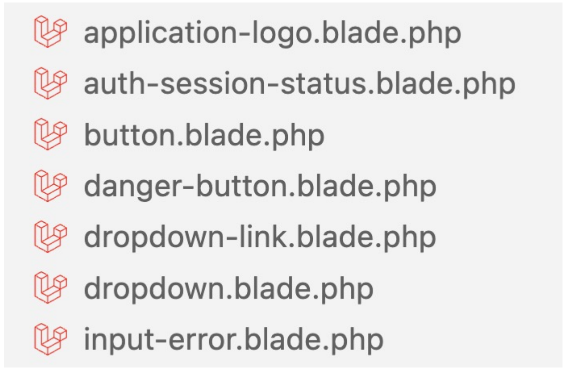
```
views
  auth
    confirm-password.blade.php
    forgot-password.blade.php
    login.blade.php
    register.blade.php
    reset-password.blade.php
    verify-email.blade.php
```

... the layouts "app", "guest" and "navigation":



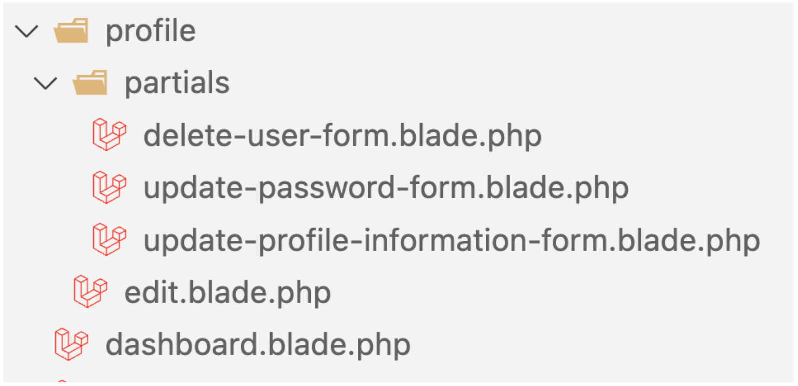
```
layouts
  app.blade.php
  guest.blade.php
  main.blade.php
  navigation.blade.php
```

... several view components:



```
application-logo.blade.php
auth-session-status.blade.php
button.blade.php
danger-button.blade.php
dropdown-link.blade.php
dropdown.blade.php
input-error.blade.php
```

... profile related views and the dashboard view:



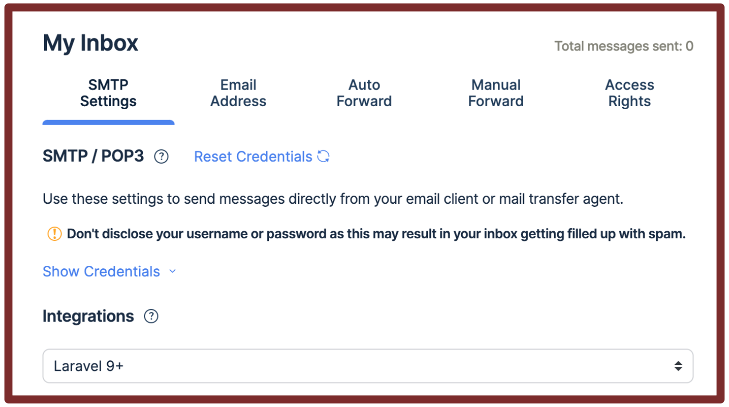
```
profile
  partials
    delete-user-form.blade.php
    update-password-form.blade.php
    update-profile-information-form.blade.php
  edit.blade.php
dashboard.blade.php
```

Email

- Some authentication features (e.g. email verification, password confirmation, reset password) require sending emails.
- Laravel breeze generated code will send, automatically, email messages for the user.
- We have to configure an email account, that Laravel will use to send the messages
- When developing, we can use a service like **Mailtrap** (https://mailtrap.io/) to send email messages to a "dummy" mailbox

Mailtrap.io

- Register on the Mailtrap.io service and get the SMTP settings for Laravel



My Inbox

Total messages sent: 0

SMTP / POP3 Reset Credentials

Use these settings to send messages directly from your email client or mail transfer agent.

Don't disclose your username or password as this may result in your inbox getting filled up with spam.

Show Credentials

Integrations

Laravel 9+

- Copy these setting to the **.env** file. Example:

```
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=XXXXXXXX
MAIL_PASSWORD=YYYYYYYY
MAIL_FROM_ADDRESS="noreppply@mycompany.com"
MAIL_FROM_NAME="${APP_NAME}"
```

- That's it. Laravel is prepared to send email.
  Laravel Breeze will send email automatically.
  For instance, try the feature "Forgot your password"

Authenticated User

- To access the model of the **authenticated user** (User model):
- With the **Auth** facade class:

```
use Illuminate\Support\Facades\Auth;

// Get the currently authenticated user...
$user = Auth::user();
$name = $user?->name;
// Get the currently authenticated user's ID...
$id   = Auth::id();
```

- Through a **Request** instance:

```
public function update(Request $request) {
    // Get the currently authenticated user...
    $user = $request->user();
    $name = $user?->name;
}
```

Auth facade

- Some methods of the **Auth** facade class:

`Auth::user()`	Current user Model
`Auth::id()`	Current user ID
`Auth::check()`	Returns true if current user is authenticated
`Auth::attempt()`	Tries to manually authenticate a user (passing an array with the credentials). Returns true if authentication was successful.
`Auth::logout()`	Log users out. Clear authentication information in user's session
`Auth::login($user)`	Login an existing user instance
`Auth::loginUsingId(1)`	Login a user just by passing its ID

Authentication & Blade

- Blade include 2 directives **@auth** and **@guest** to determine of current user is authenticated or a guest

```
@guest
   <a href="{{ route('login') }}">Login</a>
   <a href="{{ route('register') }}">Register</a>
@else
   . . . Content when user has authenticated
   {{ Auth::user()->name }}
   . . .
@endguest
```

2 – AUTHORIZATION

Authorization with Middleware, Gates and Policies

Authorization

- Authentication
  - Verifies the identity of a user or service
- Authorization
  - Process by which an entity such as a user or a service gets permission to perform a restricted operation

Authorization - middleware

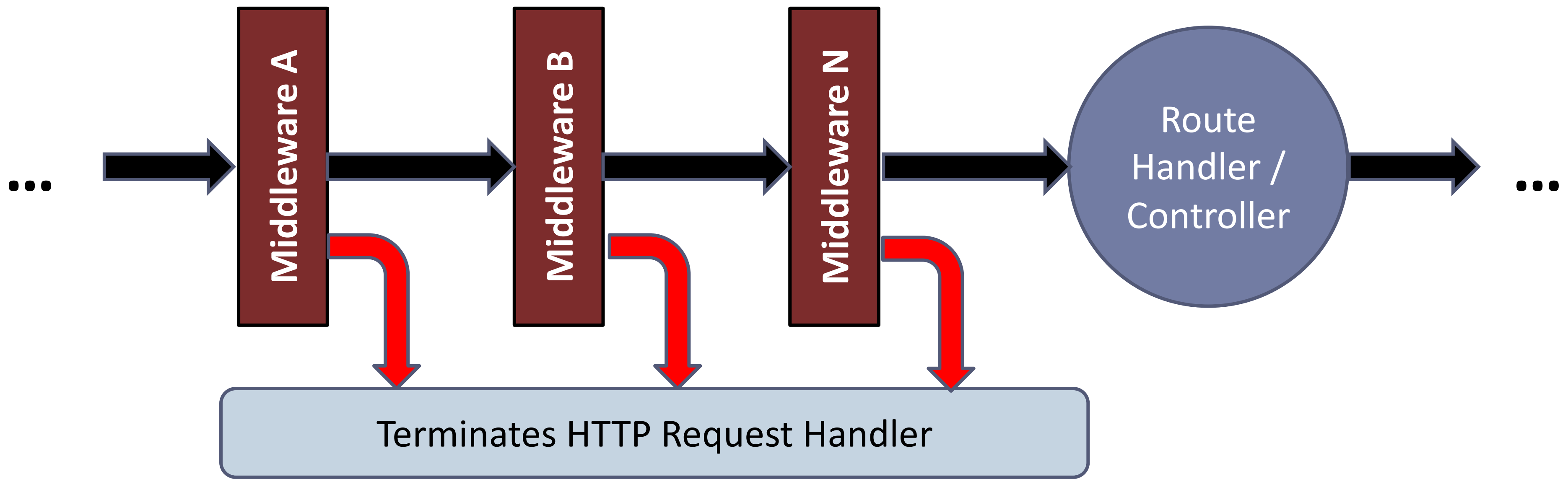
- Routes can be protected by attaching middleware
- Laravel includes (already registered) the **auth** and **verified** middleware
  - **auth** – route only accessible to authenticated user
  - **verified** – route only accessible if user has verified the e-mail (verification feature must be configured - https://laravel.com/docs/verification)
- Examples:

```
Route::get('profile', …)->middleware('auth');
```

```
// Apply auth and verified middleware to several routes
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('profile', …);
    …
});
```

Middleware

- Middleware filters HTTP requests
  - They are not used exclusively for authentication – e.g. EncryptCookies; ConvertEmptyStringsToNull; TrimStrings; ValidateCsrfToken; etc.
  - Each HTTP Request handled by Laravel, will execute a set of middlewares.
  - Each middleware executes specific code, and then, either continues the execution flow to the next middleware or terminates the request



Custom middleware

- We can create our own middleware.
- Example: middleware **admin** - route only accessible to administrators

```
php artisan make:middleware Admin
```

```
use Illuminate\Auth\Access\AuthorizationException;
. . .
class Admin
{
    . . .
    public function handle($request, Closure $next)
    {
        if ($request->user() &&  $request->user()->admin) {
            return $next($request);
        }
        throw new AuthorizationException();
    }
}
```

- More details on https://laravel.com/docs/middleware

Authorization

- Laravel provides specialized services to implement authorization: **gates** and **policies**
  - **Gates** provide a simple, closure (*inline function*) based approach (like route closures) to authorization
    - For global authorization actions
  - **Policies** classes (like controller classes) that organize authorization logic around a particular model or resource.
    - Authorization actions associated to eloquent models
    - Example: authorization to update a customer model
  - Applications can mix gates and policies

Gates

- **Gates** - determine if a specific user is authorized to perform a given action.
- Example 1: create a gate with the name: "**admin**"
  - Gates are defined in the method `boot()` of the class `App\Providers\AppServiceProvider`
  - Gates **always receive a user instance** as their **1st argument**

```
public function boot(): void
{
  Gate::define('admin', function (User $user) {
    // Only "administrator" users can "admin"
    return $user->admin;
  });
}
```

If function returns **true** then user is **authorized**.
If function returns **false** then user is **not authorized**.

Gates

- Example 2: create a gate with the name: "**view-account**"
  - Gates always receive a user instance as their 1st argument
  - Gates can receive (optionally) **additional arguments**, such as a relevant model instance

```
public function boot(): void
{
  Gate::define('view-account',
    function (User $user, Account $account) {
        // Only account owner can view the account
        return $user->id == $account->owner_id;
    });
}
```

If user is not authenticated, the gate will fail, and user is not authorized (it fails before entering the gate code)

Authorization on routes – "can"

- To protect a route with a gate, we apply the **middleware** "**can**".
- Examples:

*Gate "**admin**"*

```
Route::get('/users', …)
    ->middleware('can:admin');
```

*Gate "**view-account**"*

```
Route::get('/account/{account}', …)
    ->middleware('can:view-account,account');
```

Route parameter "account" is passed on to the gate as the second argument of the gate

Alternative syntax

```
Route::get('/account/{account}', …)
    ->can('view-account', 'account');
```

Policies

- Policies are like gates, but are defined as classes with a set of authorization actions around a model or resource

- To **create** a Policy:

```
php artisan make:policy AccountPolicy
```

  - This creates the file: `app/Policies/AccountPolicy.php`

  - To create a policy with basic CRUD policy methods:

```
artisan make:policy AccountPolicy --model=Account
```

Policies

- Methods of the Policy class refer to actions that need to be protected.

  Example: Can the user **view** the account?

```
class AccountPolicy   {   . . .
  public function view(User $user, Account $account)
  {
    return $user->id == $account->owner_id;
  } . . .
```

- 1st argument is **always** the **authenticated user** model instance
- 2nd argument (**optional**) is a **model instance** that represents the model to be protected.

If method returns **true** then user is **authorized**.
If method returns **false** then user is **not authorized**.

Policies

- Example of policy class with several methods
  - Can the user **view** the account *(only the account owner)*
  - Can the user **update** the account *(only the account owner, if age > 18)*
  - Can the user **create** an account. *(only admin users)*

```
class AccountPolicy   {   . . .
  public function view(User $user, Account $account) {
    return $user->id == $account->owner_id;
  }
  public function update(User $user, Account $account) {
    return ($user->id == $account->owner_id) &&
           ($user->age >= 18);
  }
  public function create(User $user) {
    return $user->isAdmin;
  }. . .
```

- Note that "**create**" policy method **does not have the 2nd argument**. This means that it does not depend on an instance of the model.

Policies

- Typical policy methods / typical controller methods
  - We can add policy methods as required

Policy Method	Controller Method
viewAny	index
view	show
create	create
create	store
update	edit
update	update
delete	destroy
restore	
forceDelete	

Authorization on routes – "can"

- To protect a route with a policy, we apply the middleware "**can**". *(similar to gates)*

```
Route::get('/accounts/{account}', …)
   ->middleware('can:view,account');
Route::put('/accounts/{account}', …)
   ->middleware('can:update,account');
Route::post('/accounts', …)
   ->middleware('can:create,App\Models\Account');
```

- Alternative syntax

```
Route::get('/accounts/{account}', …)
   ->can('view', 'account');
Route::put('/accounts/{account}', …)
   ->can('update', 'account');
Route::post('/accounts', …)
   ->can('create', Account::class);
```

Guest/anonymous users

- By default, all gates and policies automatically return false for anonymous users (the incoming HTTP request was not initiated by an authenticated user)
- Gates and policies can support anonymous users by declaring an "optional" type-hint for the user argument

```
class PostPolicy
{
  public function update(?User $user, Post $post)
  {
    return optional($user)->id === $post->user_id;
  }
}
```

- **$user** is null for guest/anonymous users

Authorization with User model

- Method **can** of the **User** model – returns true/false
  - Both $user and Auth::user() have an instance of User model

```
if (Auth::user()->can('view', $account)) {
   // restricted access feature
}
```

Policy AccountPolicy method = view

```
if ($user->can('view-account', $account)) {
   // restricted access feature
}
```

Gate "view-account"

```
if (Auth::user()->can('admin')) {
   // restricted access feature
}
```

Gate "admin"

```
if ($user->can('create', Account::class)) {
   // restricted access feature
}
```

Policy AccountPolicy method = create

Authorization with Gate authorize

- Method **authorize** of the Gate class

```
use Illuminate\Support\Facades\Gate;

class CourseController extends Controller {

public function show(Request $request, Account $account)
{
    Gate::authorize('view', $account);
    // Only executes this, if authorized to
}
```

  - Gate authorize method uses **gates** or **policies** to check for authorization. If user is not authorized, code is **terminated** with status code 403 (throws an AuthorizationException)

Authorize resource controllers

- To authorize all actions of a resource controller

```
use App\Models\Post;
use Illuminate\Foundation\Auth\Access\AuthorizesRequests;

class PostController extends \Illuminate\Routing\Controller
{
    use AuthorizesRequests;

    public function __construct()
    {
        $this->authorizeResource(Post::class, 'post');
    }
}
```

  - Authorizes all 7 "typical" routes of a resource controller, as long as the policy class also handles all "typical" policies
  -

Authorization on Blade templates

- To show/hide sections of the page based on gates or policies, Blade adds two directives:
  - **@can** – section visible when user is authorized
  - **@cannot** – section to hide when user is not authorized

- Examples using the Gate "view-account":

```
@can('view-account', $account)
    <!-- The Current User Can View the Account Information -->
@else
    <!-- The Current User Can Create New account -->
@endcan
```

```
@cannot('view-account', $account)
    <!-- The Current User Cannot View the Account Information -->
@endcannot
```

Authorization on Blade templates

- Mores examples (using gates and policies)

```
@can('create', App\Models\Account::class)
  <!-- The Current User Can Create New account -->
@endcan
```

```
@can('view', $account)
    <!-- The Current User Can View the Account Information -->
@elsecan('create', App\Models\Account::class)
    <!-- The Current User Can Create New account -->
@endcan
```

```
@can('admin')
    <!-- The Current User Can "admin" -->
@endcan
```

```
@cannot('view', $account)
    <!-- The Current User Cannot View the Account Information -->
@endcannot
```