Part 1:
Create and Register Controllers

Controllers are used to handle incoming requests and direct them to the appropriate responses in our project which is **Eventify**. Here, we have two main controllers: **HomeController** and **RoleController**.

- **HomeController**:
  - **Purpose**: Manages views for general pages, like the homepage and form.
  - **Methods**:
    - home(): Loads the homepage view. This view is the main page for users who are logged in or accessible to all visitors.
    - form(): Loads the form view, which is used for user input or registration.
  - **Code**:

HomeController.php

app > Http > Controllers > HomeController.php > ...

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

3 references | 0 implementations
class HomeController extends Controller
{
    1 reference | 0 overrides
    public function home(): Factory|View
    {
        return view(view: 'homepage');
    }

    1 reference | 0 overrides
    public function form(): Factory|View
    {
        return view(view: 'form');
    }
}
```

Note: When the route ***/home*** or ***/form*** is visited, HomeController loads the relevant view.

- **RoleController**:
  - **Purpose**: Manages the login process for different roles (Admin and Organizer). This controller validates user credentials and redirects users accordingly.
  - **Methods**:
    - adminLogin(Request $request): Handles admin login. It checks if the provided password is ‘admin’. If correct, it loads the admindashboard view, passing the username. If incorrect, it redirects back with an error.
    - organizerLogin(Request $request): Handles organizer login similarly but uses the password ‘organizer123’. If valid, it redirects to the homepage with the username; otherwise, it redirects back with an error.

- **Code:**

```php
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class RoleController extends Controller
{
    // Handle Admin Login
    public function adminLogin(Request $request): Factory|mixed|RedirectResponse|View
    {
        $username = $request->input(key: 'username');
        $password = $request->input(key: 'password');

        // Fixed Admin password check
        if ($password === 'admin') {
            return view(view: 'admindashboard', data: ['username' => $username]);  // Pass username to view on success
        } else {
            // Redirect to error page if credentials are incorrect
            return redirect()->route(route: 'error.page');
        }
    }

    // Handle Organizer Login
    public function organizerLogin(Request $request): Factory|mixed|RedirectResponse|View
    {
        $username = $request->input(key: 'username');
        $password = $request->input(key: 'password');

        // Fixed Organizer password check
        if ($password === 'organizer123') {
            return view(view: 'homepage', data: ['username' => $username]);  // Pass username to view on success
        } else {
            // Redirect to error page if credentials are incorrect
            return redirect()->route(route: 'error.page');
        }
    }
}
```

Note: Each method retrieves username and password from the request. It checks the password against a fixed value (acting as a simple authentication method). If successful, it redirects users to role-specific pages; if not, it routes to the error page.

- **Route Definitions: routes/web.php**
  - **Home Page Route**:
    php
    Code:

```php
Route::get(uri: '/home', action: [HomeController::class, 'home'] )->name(name: 'homepage');
```

Purpose: Maps the /home URL to the home method in HomeController.
Explanation: When a user visits /home, Laravel will call the home method in HomeController, which returns the homepage view. The route is named homepage for easy reference.

  - **User Form Route**:
    Code:

```php
Route::get(uri: '/user/form', action: [HomeController::class, 'form'] )->name(name: 'form');
```

Purpose: Maps the /user/form URL to the form method in HomeController.
Explanation: This route responds to a GET request at /user/form, triggering the form method to return the form view. The route is named form, making it easy to link to this route within the application.

  - **Admin Login Route**:
    Code:

```php
Route::post(uri: '/user/admin', action: [RoleController::class, 'adminLogin'])->name(name: 'login.admin');
Route::post(uri: '/user/organizer', action: [RoleController::class, 'organizerLogin'])->name(name: 'login.organizer');
```

Purpose: Maps the /user/admin URL to the **adminLogin** method in **RoleController.**
Explanation: This POST route is used when an admin submits a login form. The **adminLogin** method checks if the login credentials are correct. The route is named login.admin, allowing it to be referenced easily in forms and redirects.

Part 2:
Assign Controllers to Routes

- Use middleware (e.g., authentication) to protect specific routes and controllers.

```
Route::get(uri: '/some-route', action: [SomeController::class, 'someMethod'])->middleware(middleware: 'check.login.error');
```

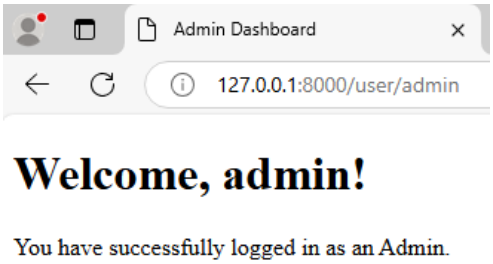
Explanation: Here, **check.login.error** middleware intercepts requests to */some-route* and checks for any session errors. If an error exists (a failed login attempt), it redirects the user to the error page. Otherwise, the request proceeds to someMethod in **SomeController**(default)**.**

```
// Error page route
Route::get(uri: '/error', action: function (): Factory|View {
    return view(view: 'error'); // Error view
})->name(name: 'error.page');
```

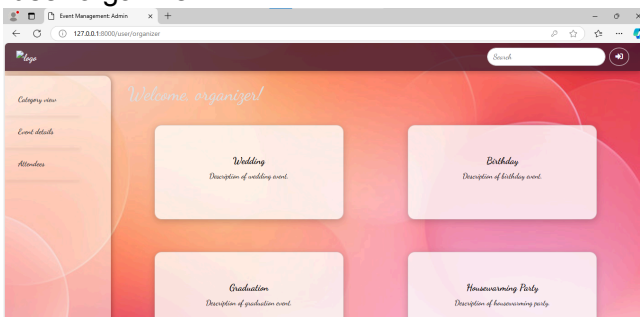
Explanation: This route provides a fallback view (error) when login errors are encountered, allowing middleware to redirect users here when necessary.

- Test routes (e.g. /, /dashboard) to ensure proper page loading.

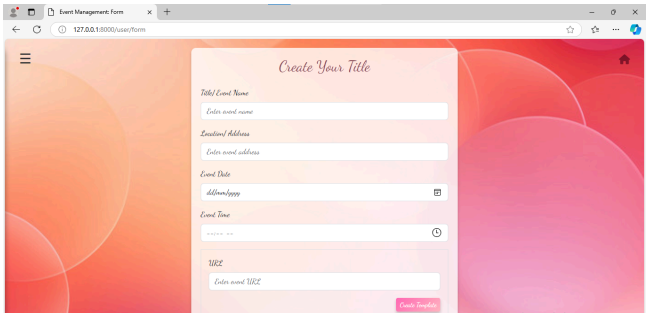
/admin/dashboard



/user/organizer



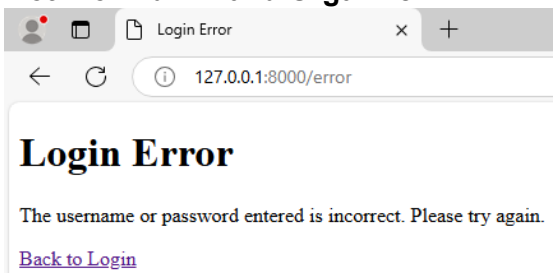
/user/form



- **Test middleware functionality**

- Attempt to access **/user/admin** and **/user/organizer** with an invalid session or login error. It redirects to the /error page due to the check.login.error middleware. This test ensures that protected routes behave as expected when accessed under different conditions.

Both for Admin and Organizer:



Part 3: Controllers with Parameters

- Modify DashboardController to accept dynamic parameters (e.g., user ID).
  - As the above mentioned, we used HomeController and RoleController.
  - We use characters/ strings as dynamic parameters in routes, like username and etc. We implement it in the RoleController.

1. In **RoleController**:
   a. Here, **{username}** is a string-based parameter, and it can accept anything that can be a valid part of a URL, such as admin or organizer.
   b. In RoleController, **$username** is passed as a string-based parameter, and we use it in our logic or pass it to the view.
2. **Use the String Parameter in Blade Views**

/views/admindashboard.php

```
web.php 1    HomeController.php    admindashboard.blade.php
resources > views > admindashboard.blade.php
1  <!-- resources/views/admin/dashboard.blade.php -->
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <title>Admin Dashboard</title>
6  </head>
7  <body>
8      <h1>Welcome, {{ $username }}!</h1>
9      <p>You have successfully logged in as an Admin.</p>
10 </body>
11 </html>
12
```

homepage.blade.php

```
homepage.blade.php
resources > views > homepage.blade.php
2   <html lang="en">
12  <body>
17  <div class="container-fluid">
18      <div class="row">
24          <div class="col-md-10 mt-4">
25              <div class="row g-4 justify-content-center px-4">
26              <h1>Welcome,  {{ $username }}!</h1>
```

Once passed the string parameter (username or any other string) to the view, it will be displayed or used within the view just like any other variable. This will display the string passed via the route (admin or organizer) in the views.

- **Test with String Parameters in Routes**
  - Testing of parameterized routes to load user-specific data is already shown in part 2