Events System

This directory contains the centralized event system for handling observer and adventurer actions in the application.

Events

ObserverEvent

- **Purpose**: Handles events related to observer characters
- **Broadcast**: Yes, broadcasts to private channel `observer.{observerId}`
- **Usage**:

```
ObserverEvent::dispatch($observerId, 'observation_started', $data);
```

AdventurerEvent

- **Purpose**: Handles events related to adventurer characters
- **Broadcast**: Yes, broadcasts to private channel `adventurer.{adventurerId}`
- **Usage**:

```
AdventurerEvent::dispatch($adventurerId, 'quest_completed', $data);
```

Listeners

ObserverEventListener

PROF

- **Purpose**: Processes ObserverEvent instances
- **Queue**: Yes, implements ShouldQueue for background processing
- **Actions Handled**:
  - `observation_started`
  - `observation_completed`
  - `data_collected`

AdventurerEventListener

- **Purpose**: Processes AdventurerEvent instances
- **Queue**: Yes, implements ShouldQueue for background processing
- **Actions Handled**:
  - `quest_started`
  - `quest_completed`
  - `level_up`
  - `item_found`

Usage Examples

Dispatching Events

```php
// For observer actions
use App\Events\ObserverEvent;

ObserverEvent::dispatch($userId, 'observation_started', [
    'target' => 'some_target',
    'location' => 'some_location'
]);

// For adventurer actions
use App\Events\AdventurerEvent;

AdventurerEvent::dispatch($userId, 'quest_completed', [
    'quest_id' => 123,
    'rewards' => ['gold' => 100, 'experience' => 50]
]);
```

Listening to Broadcasts

Events are broadcast to private channels, so you can listen to them in your frontend:

```javascript
// For observers
Echo.private('observer.' + observerId)
    .listen('.observer.action', (e) => {
        console.log('Observer action:', e.action, e.data);
    });

// For adventurers
Echo.private('adventurer.' + adventurerId)
    .listen('.adventurer.action', (e) => {
        console.log('Adventurer action:', e.action, e.data);
    });
```

PROF

Configuration

Events and listeners are automatically registered in `EventServiceProvider.php`. No additional configuration is required.

Best Practices

1. Always dispatch events asynchronously when possible
2. Include relevant data in the event payload
3. Use descriptive action names
4. Handle events in listeners with proper error handling

5. Log important events for debugging and auditing

Extending the System

To add new event types:

1. Create a new event class in this directory
2. Create a corresponding listener in the `Listeners` directory
3. Register the event-listener mapping in `EventServiceProvider.php`
4. Update this README with the new event details