Authenticating and managing users

Dancer2::Plugin::Auth::Extensible

Andy Beverley andy@andybev.com

Why use Auth::Extensible?

- Minimal code for initial implementation
- Several providers available
- Can use existing DBIC schema
- Many user management features
- Sorry, only for Dancer(2)

Features - what did I need?

- Encrypted passwords
- Roles
- Time of last login
- Ability to update user details
- Ability to create user and send welcome email
- Password resets
- Password expiry

Basic example

config.yml

```
Plugins:
  Auth::Extensible:
    disable_roles: 1
    realms:
      config:
        provider: Config
        users:
          - user: andy
            pass: secret # Can be encrypted
```

Application

```
use Dancer2::Plugin::Auth::Extensible;
get '/restricted' => require_login sub {
    return "You're in";
};
```

Basic example continued

- Routes automatically added:
  - /login
  - /logout
- Automatic redirect to built-in login
- Default pages can be changed

Roles

```
get '/beer' => require_role BeerDrinker => sub {
    ...
};


get '/drink' => require_any_role
    [qw(BeerDrinker VodaDrinker)] => sub
{
    ...
};
```

DBIC example

- Full features only possible with DBIC
- Users table and resultsource required
- Username and password columns

Config.yml

```
Plugins:
  Auth::Extensible:
    disable_roles: 1
    realms:
      dbic:
        provider: DBIC
        # optionally specify table/column names
```

Password resets

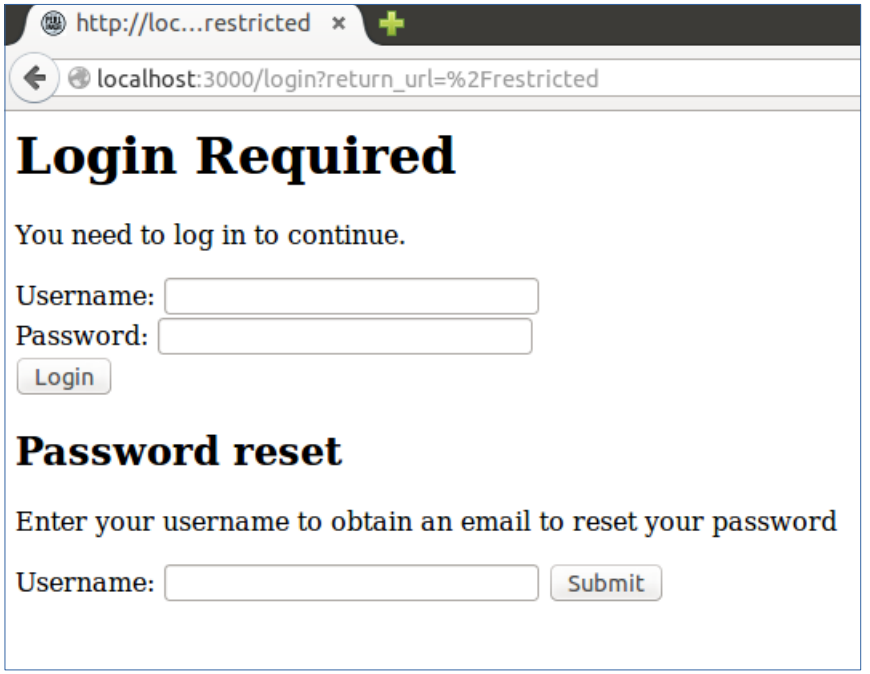
- Requires DBIC provider

Config.yml

```
Plugins:
  Auth::Extensible:
    disable_roles: 1
    reset_password_handler: 1
    mailer:
      module: Mail::Message
      options:
        via: sendmail
    mail_from: nobody@example.com
    realms:
      dbic:
        provider: DBIC
```

Password resets continued

- Functions without any code, just config parameters
- Can specify functions to create email content or complete emails



Password functions

- Check current user's password:

```
user_password password => 'mysecret'
```

- Check a password:

```
user_password username => 'jsmith',
    password => 'bigsecret'
```

- Change a password:

```
user_password username => 'jbloggs',
    password => 'old', new_password => 'secret'
```

Create users

```
# Simple user
create_user
   username => "jsmith",
   realm    => "dbic",
   surname  => "Smith";


# With welcome email (requires pw reset configured)
create_user
   username      => "jsmith",
   email         => "john@you.com",
   email_welcome => 1;
```

Password expiry

config.yml

```
users_pwchanged_column: pw_changed
password_expiry_days: 60
```

Application

```
hook before => sub {
    if (logged_in_user_password_expired)
    {
        redirect '/password_update'
            unless request->uri eq '/password_update';
    }
}
```

Time of last successful login

config.yml

```
record_lastlogin: 1
users_lastlogin_column: lastlogin
```

Application

```
my $datetime = logged_in_user_lastlogin();
```

Summary

- Lots of functionality with minimal configuration and code
- Can be customised as your application grows

Questions?