How to implement the authentication

Please find below the steps to understand the authentication process :

User entity

The interface's users are represented by a User entity. It implements the UserInterface & PasswordAuthenticatedUserInterface which provides methods used by the firewall and authentication system for credentials checks.

```
/**
 * @ORM\Entity(repositoryClass=UserRepository::class)
 * @ORM\Table(name="`user`")
 * @UniqueEntity("username")
 *
 * @method string getUserIdentifier()
 */
class User implements PasswordAuthenticatedUserInterface, UserInterface
```

Setup security.yml

Provider

```
security:
    # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
    providers:
        database:
            entity:
                class: 'App\Entity\User'
                property: 'username'
```

Indicates to Symfony where can be found the user, in User entity and the defines which attribute is used for authentication.

Password encryption

```
security:
    password_hashers:
        App\Entity\User:
            algorithm: 'bcrypt'
```

Bcrypt encoder is used to encrypt the passwords before recording in Database

Firewall

```
security:
    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: database
            custom_authenticator: App\Security\UserAuthenticator
            logout:
                path: app_logout
```

The firewall defines the authentication's process. The credentials are entered in the form_login with the route app_login : login.

Roles hierarchy

```
security:
    role_hierarchy:
        ROLE_ADMIN: ROLE_USER
```

indicates that the ROLE_ADMIN has the same rights than the ROLE_USER and above.

Access control

```
security:
    access_control:
        - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/users, roles: ROLE_ADMIN}
        - { path: ^/, roles: ROLE_USER }
```

The access control is set here : * the path /login is accessible to anyone * the path /users only to logged user with the ROLE_ADMIN * the path / to all logged users

Security Controller

The SecurityController : namespace Appdefines how works the authentication process and the error message sent to the View.

Login Form

The form UserType display the fields to be completed to create a new User :

```
     public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('username', TextType::class, ['label' => "Nom d'utilisateur"])
            ->add('password', RepeatedType::class, [
                'type' => PasswordType::class,
                'invalid_message' => 'Les deux mots de passe doivent correspondre.',
                'required' => $options['passwordRequired'],
                'first_options' => ['label' => 'Mot de passe'],
                'second_options' => ['label' => 'Tapez le mot de passe à nouveau'],
                'mapped'=>false,
            ])
            ->add('email', EmailType::class, ['label' => 'Adresse email'])
        ;

        if ($options['withRoleChoice']) {
            $builder->add('role', ChoiceType::class, [
                'choices' => [
                    'Utilisateur simple' => 'ROLE_USER',
                    'Administrateur' => 'ROLE_ADMIN',
                ],
            ]);
        }
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => User::class,
        ]);
        $resolver->setDefault('withRoleChoice', false)->setAllowedTypes('withRoleChoice',
['boolean']);
```

```
        $resolver->setDefault('passwordRequired', true)-
>setAllowedTypes('passwordRequired', ['boolean']);
    }
```

Voters

Voters are configured in App. Rights are configured here

Configure support

To enable support of a voter, simply add right name in returned array

```
  public const TASKS_LIST = 'TASKS_LIST';
  public const TASK_EDIT = 'TASK_EDIT';
  public const TASK_TOGGLE = 'TASK_TOGGLE';
  public const TASK_DELETE = 'TASK_DELETE';
  public const TASK_CREATE = 'TASK_CREATE';
---
  protected function supports(string $attribute, $subject): bool
  {
      // replace with your own logic
      // https://symfony.com/doc/current/security/voters.html
      return in_array(
          $attribute,
          [
              self::TASKS_LIST,
              self::TASK_EDIT,
              self::TASK_TOGGLE,
              self::TASK_DELETE,
              self::TASK_CREATE,
          ]
      ) && ($subject instanceof \App\Entity\Task || null == $subject);
  }
```

Configure verification

change below code with your logic

```
protected function voteOnAttribute(string $attribute, $subject, TokenInterface $token)
{
    $user = $token->getUser();
    // if the user is anonymous, do not grant access
    if (!$user instanceof UserInterface) {
        return false;
    }

    // ... (check conditions and return true to grant permission) ...
    switch ($attribute) {
        case self::TASKS_LIST:
        case self::TASK_CREATE:
            return true;
        case self::TASK_EDIT:
        case self::TASK_TOGGLE:
        case self::TASK_DELETE:
            if (!$subject instanceof Task) {
                return false;
            }

           return ($this->security->isGranted('ROLE_ADMIN') && null == $subject-
>getAuthor()) || $subject->getAuthor() === $user;
    }

    return false;
}
```

Use voter

Once voter is developed, to use it in your code, you need to use :

- @IsGranted("ROLE_NAME") from Sensio) in your controller method declaration
- $security->isGranted("ROLE_NAME) from Symfony\Component\Security\Core directly in your code

```
/**
  * @Route("/tasks/create", name="task_create", methods={"GET","POST"})
  * @IsGranted("ROLE_USER")
  */
public function create()
```

OR

```
/**
  * @Route("/tasks/create", name="task_create", methods={"GET","POST"})
  * @IsGranted("ROLE_USER")
  */
public function create(Security $security){
    if (!$security->isGranted("ROLE_USER"){
        // Do something
    }
}
```

In the first case, you will get 401 or 403 before your code execution. Useful to reduce access to whole functionality. In the second case, your code will be executed, and you can choose what to do if user's rights aren't enough.