Securing access of our entities

> 👉 In this important section you will be taught how **entity access** works. At the end of the section you will be able to create custom permissions for users and translate it towards your entity. In the use case of this project it means that authors can only see/edit/delete their own created entities.

There is something worth noticing about our entities. While the author of the entity will be the owner (thanks to the *PreCreate()* function in our Offer entity) he has no exclusive access towards viewing, or even editing and deleting the entity.

While drupal will typically provide separate “view”, “create”, “edit” and “delete” options we will (for now) make this 1 single permission: ***administer own offers***.

But we did not specify which access this means towards the entity itself. Let us make sure that everyone with this access can create offers and more importantly that they can only edit and delete their own offers and not those of others.

First, add a file **modules/custom/offer/offer.permissions.yml** with the following:

```
administer own offers:
 title: 'Create/edit/delete own offers'
```

Second, add the following methods to the **custom/offer/src/Offer** class at the bottom, these are two methods that are used quite a lot. The first one is to make sure the user id gets stored as the author of the entity, The other ones are typical methods to quickly get info about the author of an entity:

```
/**
 * {@inheritdoc}
 *
 * Makes the current user the owner of the entity
 */
public static function preCreate(EntityStorageInterface
$storage_controller, array &$values) {
  parent::preCreate($storage_controller, $values);
  $values += array(
    'user_id' => \Drupal::currentUser()->id(),
  );
}
```

```
/**
* {@inheritdoc}
*/
public function getOwner() {
 return $this->get('user_id')->entity;
}

/**
* {@inheritdoc}
*/
public function getOwnerId() {
 return $this->get('user_id')->target_id;
}
```

In the end we want full CRUD access for our authenticated users. When accessing an entity in drupal, there are 4 operations that can be requested:

- view
- update
- edit
- delete

Add the following to the annotations of your entity inside the **modules/custom/offer/src/Entity/Offer.php** file:

```
*   handlers = {
*     "access" = "Drupal\offer\OfferAccessControlHandler",
*   }
```

This file will handle access towards our entity. Add a file called **OfferAccessControlHandler.php** inside **modules/custom/offer/src**:

```
<?php

namespace Drupal\offer;

use Drupal\Core\Access\AccessResult;
use Drupal\Core\Entity\EntityAccessControlHandler;
use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Session\AccountInterface;

/**
```

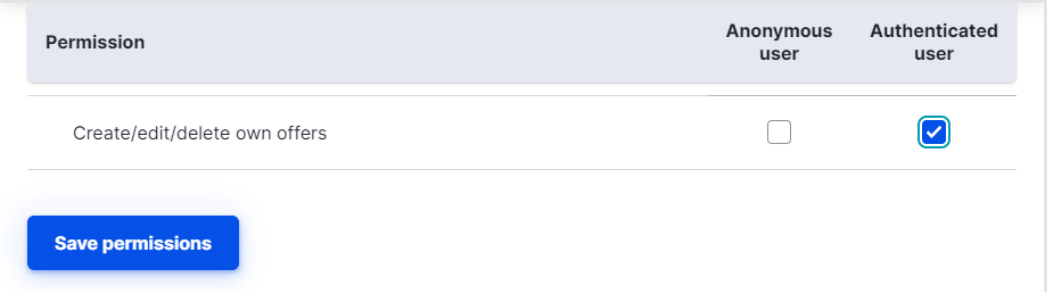
```
* Access controller for the offer entity. Controls create/edit/delete
access for entity and fields.
*
* @see \Drupal\offer\Entity\Offer.
*/
class OfferAccessControlHandler extends EntityAccessControlHandler {

 /**
  * {@inheritdoc}
  *
  * Link the activities to the permissions. checkAccess is called with
the
  * $operation as defined in the routing.yml file.
  */
 protected function checkAccess(EntityInterface $entity, $operation,
AccountInterface $account) {

   $access = AccessResult::forbidden();

   switch ($operation) {
     case 'view':
       if ($account->hasPermission('administer own offers')) {
         $access = AccessResult::allowedIf($account->id() ==
$entity->getOwnerId())->cachePerUser()->addCacheableDependency($entity);
       }
       break;
     case 'update': // Shows the edit buttons in operations
       if ($account->hasPermission('administer own offers')) {
         $access = AccessResult::allowedIf($account->id() ==
$entity->getOwnerId())->cachePerUser()->addCacheableDependency($entity);
       }
       break;
     case 'edit': // Lets me in on the edit-page of the entity
       if ($account->hasPermission('administer own offers')) {
         $access = AccessResult::allowedIf($account->id() ==
$entity->getOwnerId())->cachePerUser()->addCacheableDependency($entity);
       }
       break;
     case 'delete': // Shows the delete buttons + access to delete this
entity
       if ($account->hasPermission('administer own offers')) {
         $access = AccessResult::allowedIf($account->id() ==
$entity->getOwnerId())->cachePerUser()->addCacheableDependency($entity);
       }
       break;
   }
```

```
    return $access;
  }

  /**
   * {@inheritdoc}
   *
   * Separate from the checkAccess because the entity does not yet exist,
it
   * will be created during the 'add' process.
   */
  protected function checkCreateAccess(AccountInterface $account, array
$context, $entity_bundle = NULL) {
    return AccessResult::allowedIfHasPermission($account, 'administer own
offers');
  }

}

?>
```

This access controller gives us a variety of power towards our entity. We now have full control in code on who can access different modes of our entity.

If you take a closer look, it is here that we integrate our permission (administer own offers) with our view/edit/update/delete access. As an extra we add a check to make sure there is only access to own entities.

💻 Add the newly created “administer own offers” permission to all authenticated users via admin/people/permissions.



In a later stage of the software, we can create different user roles for which entire access to the CRUD section can be granted with one click.

With our entity access completely nailed, we are about to use these access checks in our routing and crud forms. Let’s move on to the next chapter!