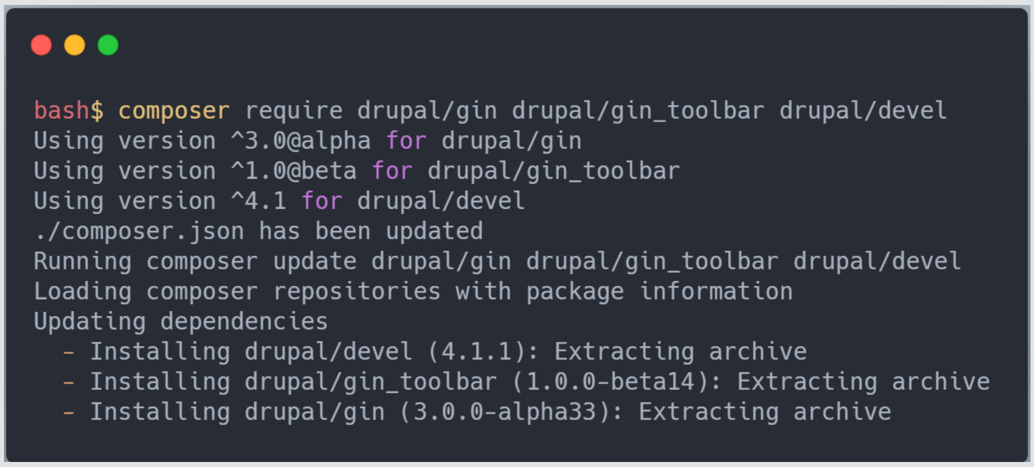
Building our first content entity

👉 This section teaches you how to define a custom entity and create it in the database. At the end you will be able to create your own custom entity with custom tailored **base fields** and revisions support.

💻 If you follow this course **by doing**:
After you installed the *recommended* drupal 9 and set-up drupal/gin
And the following module:

- drupal/devel



```
bash$ composer require drupal/gin drupal/gin_toolbar drupal/devel
Using version ^3.0@alpha for drupal/gin
Using version ^1.0@beta for drupal/gin_toolbar
Using version ^4.1 for drupal/devel
./composer.json has been updated
Running composer update drupal/gin drupal/gin_toolbar drupal/devel
Loading composer repositories with package information
Updating dependencies
  - Installing drupal/devel (4.1.1): Extracting archive
  - Installing drupal/gin_toolbar (1.0.0-beta14): Extracting archive
  - Installing drupal/gin (3.0.0-alpha33): Extracting archive
```

And enable them (devel via drush, to enable gin go to the backend). Make gin the default theme (also the administration theme) via admin/settings/appearance**.** Also in the theme settings, set the toolbar as “horizontal, modern toolbar” and disable the “Users can override Gin settings”.

The first question that gets raised is why would we use custom content entities. Isn’t the core node entity with it’s subtypes (bundles) enough?

If we’d have a simple website with just some blog posts and a portfolio, I’d always recommend to use the core Node content entity. It is the de facto out-of-the-box solution for this.

But our platform aims to have full control over all pages that create, edit and delete content, as well as the overviews. Custom entities give us more power to define our own access functions.

We build a platform that allows users to create offers as well as to make a bid on offers. It would not make sense to use Nodes with bundles like this:

Entity Node
    Bundle Offer
    Bundle Bid

I'd have to add numerous access checks to make sure users only get access to their own Offer entities and only their own Bids because they are from the same Entity. Drupal's node behaviour wasn't really meant to separate access between these kind of node types as well. No, instead we do:

Entity Offer
Entity Bid
...

Proper modelling of our data is crucial. The **Entity API** provides us all the tools for doing this.

We start with creating a content entity 'Offer'.

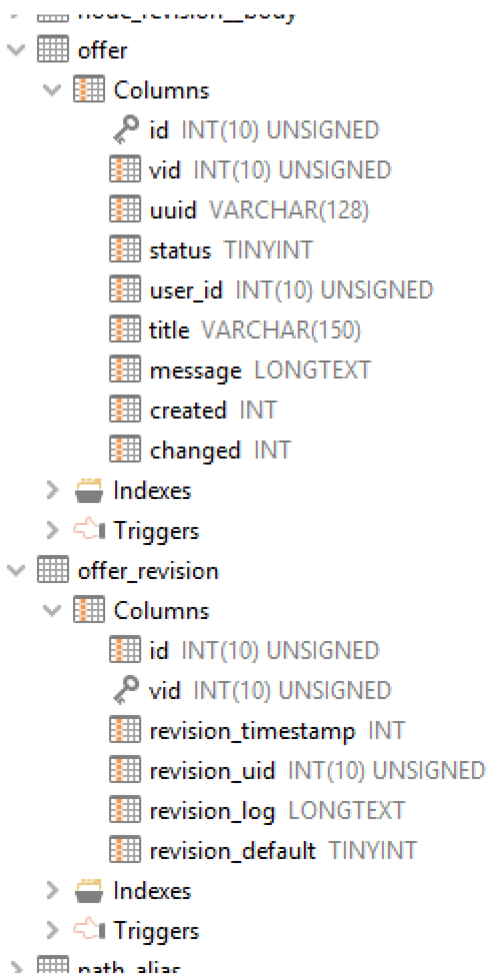
Disable the offer module if it is already enabled from a previous chapter. Picking up new entities works best when enabling the module. In theory this should also work with just clearing caches, but I experienced that sometimes this just doesn't work.

A file named **Offer.php** file inside **modules/custom/offer/src/Entity** will define our entity. Copy this code to define the entity:

```php
<?php
/**
* @file
* Contains \Drupal\offer\Entity\Offer.
*/

namespace Drupal\offer\Entity;

use Drupal\Core\Entity\EditorialContentEntityBase;
use Drupal\Core\Field\BaseFieldDefinition;
use Drupal\Core\Entity\EntityTypeInterface;
use Drupal\Core\Entity\ContentEntityInterface;
use Drupal\Core\Entity\EntityStorageInterface;

/**
* Defines the offer entity.
*
* @ingroup offer
*
* @ContentEntityType(
*   id = "offer",
*   label = @Translation("Offer"),
*   base_table = "offer",
*   data_table = "offer_field_data",
*   revision_table = "offer_revision",
*   revision_data_table = "offer_field_revision",
*   entity_keys = {
*     "id" = "id",
*     "uuid" = "uuid",
*     "label" = "title",
*     "revision" = "vid",
*     "status" = "status",
*     "published" = "status",
*     "uid" = "uid",
*     "owner" = "uid",
*   },
*   revision_metadata_keys = {
*     "revision_user" = "revision_uid",
*     "revision_created" = "revision_timestamp",
*     "revision_log_message" = "revision_log"
*   },
* )
*/

class Offer extends EditorialContentEntityBase {

 public static function baseFieldDefinitions(EntityTypeInterface
$entity_type) {
   $fields = parent::baseFieldDefinitions($entity_type); // provides id
and uuid fields

   $fields['user_id'] = BaseFieldDefinition::create('entity_reference')
     ->setLabel(t('User'))
     ->setDescription(t('The user that created the offer.'))
     ->setSetting('target_type', 'user')
     ->setSetting('handler', 'default')
     ->setDisplayOptions('view', [
       'label' => 'hidden',
       'type' => 'author',
       'weight' => 0,
     ])
     ->setDisplayOptions('form', [
       'type' => 'entity_reference_autocomplete',
       'weight' => 5,
       'settings' => [
         'match_operator' => 'CONTAINS',
         'size' => '60',
         'autocomplete_type' => 'tags',
         'placeholder' => '',
       ],
     ])
     ->setDisplayConfigurable('form', TRUE)
     ->setDisplayConfigurable('view', TRUE);

   $fields['title'] = BaseFieldDefinition::create('string')
     ->setLabel(t('Title'))
     ->setDescription(t('The title of the offer'))
     ->setSettings([
       'max_length' => 150,
       'text_processing' => 0,
     ])
     ->setDefaultValue('')
     ->setDisplayOptions('view', [
       'label' => 'above',
       'type' => 'string',
       'weight' => -4,
     ])
     ->setDisplayOptions('form', [
       'type' => 'string_textfield',
       'weight' => -4,
     ])
     ->setDisplayConfigurable('form', TRUE)
     ->setDisplayConfigurable('view', TRUE);

   $fields['message'] = BaseFieldDefinition::create('string_long')
     ->setLabel(t('Message'))
     ->setRequired(TRUE)
     ->setDisplayOptions('form', [
       'type' => 'string_textarea',
       'weight' => 4,
       'settings' => [
         'rows' => 12,
       ],
     ])
```

```
      ->setDisplayConfigurable('form', TRUE)
      ->setDisplayOptions('view', [
        'type' => 'string',
        'weight' => 0,
        'label' => 'above',
      ])
      ->setDisplayConfigurable('view', TRUE);

    $fields['status'] = BaseFieldDefinition::create('boolean')
      ->setLabel(t('Publishing status'))
      ->setDescription(t('A boolean indicating whether the Offer entity
is published.'))
      ->setDefaultValue(TRUE);

    $fields['created'] = BaseFieldDefinition::create('created')
      ->setLabel(t('Created'))
      ->setDescription(t('The time that the entity was created.'));

    $fields['changed'] = BaseFieldDefinition::create('changed')
      ->setLabel(t('Changed'))
      ->setDescription(t('The time that the entity was last edited.'));

    return $fields;
  }
}
```

Enable the module. Now our offer table is created and two extra database tables were added:


```
offer
  Columns
    id INT(10) UNSIGNED
    vid INT(10) UNSIGNED
    uuid VARCHAR(128)
    status TINYINT
    user_id INT(10) UNSIGNED
    title VARCHAR(150)
    message LONGTEXT
    created INT
    changed INT
  Indexes
  Triggers
offer_revision
  Columns
    id INT(10) UNSIGNED
    vid INT(10) UNSIGNED
    revision_timestamp INT
    revision_uid INT(10) UNSIGNED
    revision_log LONGTEXT
    revision_default TINYINT
  Indexes
  Triggers
```

Now let's proceed with the CRUD operations. For every offer, we'd like to have an add, edit and delete form. But first, we secure the access.