Episode 2 - Creating Post, Category, Tag, PostTag and RBAC tables

Hello,
In the previous episode, we have successfully installed and set up our Yii application. At the end we also run our first migration to test database connection and create `User` table. In this episode, we will learn more about migrations, create our own migrations and run RBAC migrations. Also, we will prepare all other tables which we will need to create our Yii blog application.

Database migrations

During the course of developing and maintaining a database-driven application, the structure of the database being used evolves just like the source code does. For example, during the development of an application, a new table may be found necessary; after the application is deployed to production, it may be discovered that an index should be created to improve the query performance; and so on. Because a database structure change often requires some source code changes, Yii supports the so-called database migration feature that allows you to keep track of database changes in terms of database migrations which are version-controlled together with the source code. [1]

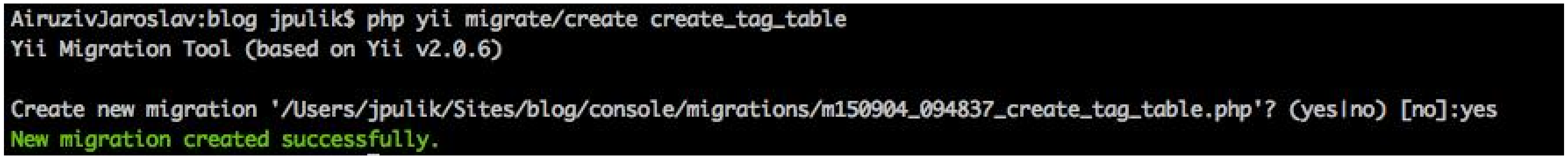
Creating migrations

Tag

First, we will create new migration for `Tag` table. To generate new migration we have to run this command:

```
php yii migrate/create create_tag_table
```

You should see:



```
AiruzivJaroslav:blog jpulik$ php yii migrate/create create_tag_table
Yii Migration Tool (based on Yii v2.0.6)

Create new migration '/Users/jpulik/Sites/blog/console/migrations/m150904_094837_create_tag_table.php'? (yes|no) [no]:yes
New migration created successfully.
```

This will generate the new create_tag_table migration file in `console/migrations/` directory with this name:

`m<YYMMDD_HHMMSS>_create_tag_table` [2]

```
<?php
```

```
use yii\db\Migration;



class m160525_190407_create_tag_table extends Migration
{

    public function up()
    {
        $this->createTable('tag_table', [
            'id' => $this->primaryKey(),
        ]);
    }



    public function down()
    {
        $this->dropTable('tag_table');
    }
}
```

There are also `safeUp()` and `safeDown()` methods, but we won't use them because we don't actually need a transaction migration.

As the official documentation describes: [3]

In the migration class, you are expected to write code in the up() method that makes changes to the database structure. You may also want to write code in the down() method to revert the changes made by up(). The up() method is invoked when you upgrade the database with this migration, while the down() method is invoked when you downgrade the database.

We will use the new migration syntax introduced in Yii 2.0.6. For our tag table we need only few columns like id, name, created_at and updated_at. Our migration file will look like:

```
<?php
```

```php
use yii\db\Migration;

class m150904_094837_create_tag_table extends Migration
{
    public function up()
    {
        $this->createTable('tag', [
            'id' => $this->primaryKey(),
            'name' => $this->string(64)->notNull()->unique(),
            'created_at' => $this->datetime()->notNull(),
            'updated_at' => $this->datetime(),
        ]);
    }

    public function down()
    {
        $this->dropTable('tag');
    }
}
```

Category

Run

```
php yii migrate/create create_category_table
```

to generate new migration for Category table.

Our `Category` migration will have this content:

```php
<?php

use yii\db\Migration;
```

```
class m150904_102410_create_category_table extends Migration
{
    public function up()
    {
        $this->createTable('category', [
            'id' => $this->primaryKey(),
            'name' => $this->string(64)->notNull()->unique(),
            'slug' => $this->string(64)->notNull()->unique(),
            'meta_description' => $this->string(160),
            'created_at' => $this->datetime()->notNull(),
            'updated_at' => $this->datetime(),
        ]);
    }

    public function down()
    {
        $this->dropTable('category');
    }
}
```

Post

Run

```
php yii migrate/create create_post_table
```

to generate new migration for Post table.

Our `Post` migration will have this content:

```
<?php

use yii\db\Migration;
```

```
class m150904_102648_create_post_table extends Migration
{
    public function up()
    {
        $this->createTable('post', [
            'id' => $this->primaryKey(),
            'title' => $this->string(128)->notNull()->unique(),
            'slug' => $this->string(128)->notNull()->unique(),
            'lead_photo' => $this->string(128),
            'lead_text' => $this->text(),
            'content' => $this->text()->notNull(),
            'meta_description' => $this->string(160),
            'created_at' => $this->datetime()->notNull(),
            'updated_at' => $this->datetime(),
            'created_by' => $this->integer()->notNull(),
            'updated_by' => $this->integer(),
            'category_id' => $this->integer()->notNull()
        ]);

        $this->createIndex('post_index', 'post', ['created_by', 'updated_by']);
        $this->addForeignKey('fk_post_category', 'post', 'category_id', 'category', 'id', 'CASCADE', 'CASCADE');
        $this->addForeignKey('fk_post_user_created_by', 'post', 'created_by', 'user', 'id', 'CASCADE', 'CASCADE');
        $this->addForeignKey('fk_post_user_updated_by', 'post', 'updated_by', 'user', 'id', 'CASCADE', 'CASCADE');
    }

    public function down()
    {
        $this->dropForeignKey('fk_post_category', 'post');
        $this->dropForeignKey('fk_post_user_created_by', 'post');
        $this->dropForeignKey('fk_post_user_updated_by', 'post');
        $this->dropTable('post');
    }
}
```

PostTag

Run

```
php yii migrate/create create_post_tag_table
```

to generate new migration for PostTag table.

Our `PostTag` migration will have this content:

```
<?php

use yii\db\Migration;

class m150906_141330_create_post_tag_table extends Migration
{
    public function up()
    {
        $this->createTable('post_tag', [
            'id' => $this->primaryKey(),
            'post_id' => $this->integer()->notNull(),
            'tag_id' => $this->integer()->notNull()
        ]);

        $this->createIndex('post_tag_index', 'post_tag', ['post_id', 'tag_id']);
        $this->addForeignKey('fk_post_tag_post', 'post_tag', 'post_id', 'post', 'id', 'CASCADE', 'CASCADE');
        $this->addForeignKey('fk_post_tag_tag', 'post_tag', 'tag_id', 'tag', 'id', 'CASCADE', 'CASCADE');
    }

    public function down()
    {
        $this->dropForeignKey('fk_post_tag_post', 'post_tag');
        $this->dropForeignKey('fk_post_tag_tag', 'post_tag');
        $this->dropTable('post_tag');
    }
}
```

Applying migrations

Run command

```
php yii migrate/up
```

in your console to apply newly created migrations.

This command will list all migrations that have not been applied so far. If you confirm that you want to apply these migrations, it will run the up() or safeUp() method in every new migration class, one after another, in the order of their timestamp values. If any of the migrations fails, the command will quit without applying the rest of the migrations.

RBAC (Role-Based Access Control) tables migration [4]

Before executing RBAC tables migration we must setup Yii `authManager` component. Yii provides two types of authorization managers: `yii\rbac\PhpManager` and `yii\rbac\DbManager`. The former uses a PHP script file to store authorization data, while the latter stores authorization data in a database. In our blog application we will store our RBAC data in database - so we will use DbManager. To configure authManager to work with DbManager we must add this code to `common/config/main.php`:

```
'components' => [


    'authManager' => [
        'class' => 'yii\rbac\DbManager',
    ],


],
```

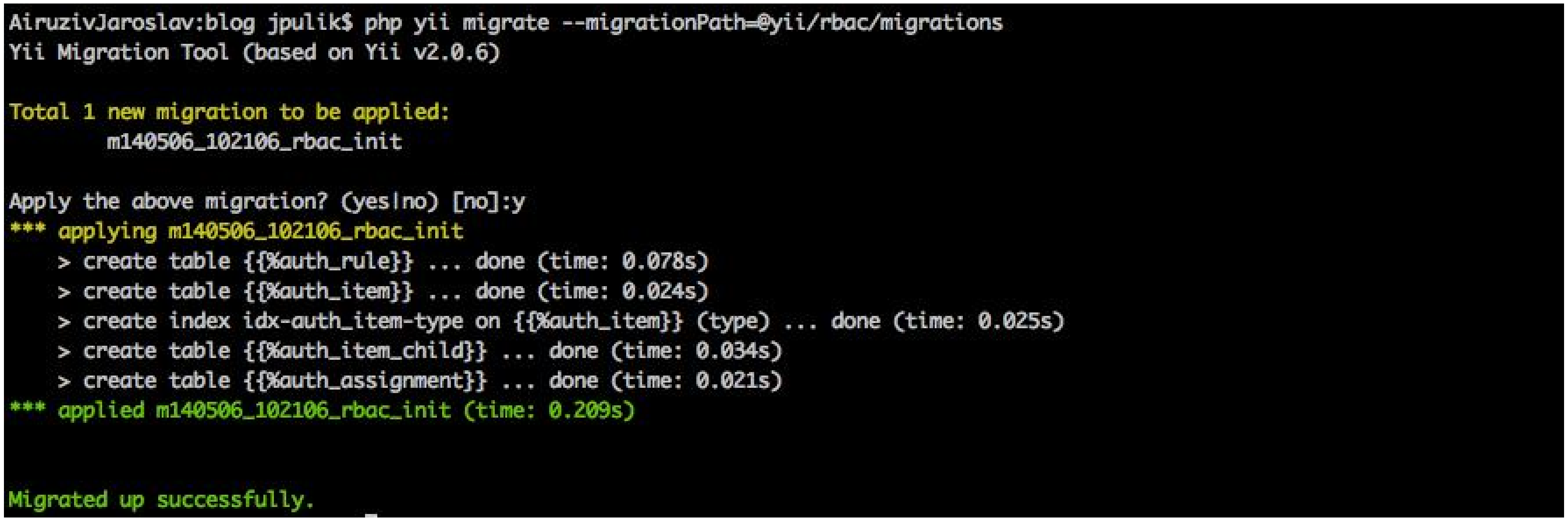
After setting up the `authManager` we can now run RBAC migration:

```
php yii migrate --migrationPath=@yii/rbac/migrations
```

This migration will create following tables:

1. **itemTable**: the table for storing authorization items. Defaults to `auth_item`.
2. **itemChildTable**: the table for storing authorization item hierarchy. Defaults to `auth_item_child`.
3. **assignmentTable**: the table for storing authorization item assignments. Defaults to `auth_assignment`.
4. **ruleTable**: the table for storing rules. Defaults to `auth_rule`.

If your migration was successful you should see:



```
AiruzivJaroslav:blog jpulik$ php yii migrate --migrationPath=@yii/rbac/migrations
Yii Migration Tool (based on Yii v2.0.6)

Total 1 new migration to be applied:
        m140506_102106_rbac_init

Apply the above migration? (yes|no) [no]:y
*** applying m140506_102106_rbac_init
    > create table {{%auth_rule}} ... done (time: 0.078s)
    > create table {{%auth_item}} ... done (time: 0.024s)
    > create index idx-auth_item-type on {{%auth_item}} (type) ... done (time: 0.025s)
    > create table {{%auth_item_child}} ... done (time: 0.034s)
    > create table {{%auth_assignment}} ... done (time: 0.021s)
*** applied m140506_102106_rbac_init (time: 0.209s)


Migrated up successfully.
```

The `authManager` can now be accessed via `Yii::$app->authManager`. We will set up our roles, permissions and rules in the next episode focused specially to RBAC. We have only created necessary RBAC tables with built in migrations for now.

Now check your database, you should have these 10 tables: `auth_assignment`, `auth_item`, `auth_item_child`, `auth_rule`, `category`, `migration`, `post`, `post_tag`, `tag` and `user`.

Perfect, we are done for this episode! We have successfully learned how to create an apply database migrations. We have also run migration for Yii `authManager` component. We will use `authManager` to provide Role-Baces Access Control in the next episode.

We will continue building our blog in next episode. If do you have any questions regarding to this episode, please write them below to the comments section.

Download files from this episode: episode_02.zip.