IT academy **IABS**

DAY 19

WEB PROGRAMMER

15-55 years

Содержание

День 19.

День 19.

Дипломный проект

По окончанию курса вы должны выбрать и реализовать дипломный проект. Защита дипломного проекта будет проходить публично. Будут приглашены студенты других групп и возможные работодатели.

Для выполнения проектов вы можете объединяться в группы или реализовывать проект самостоятельно.

Дипломные проекты могут как только frontend и backend, так и full stack. Если вы выбираете узкую специализацию дипломного проекта - на вас накладываются дополнительные требования. К примеру, выбрав только frontend вы должны выбрать сложный макет, использовать анимации, адаптивность и оживить страницы интересными написанными самостоятельно скриптами.

Варианты дипломных проектов:

1. Сделать сайт на фреймворке yii2
- багтрекер
- сайт онлайн бронирования столиков в кафе
- сайт преподавателя для контроля домашних заданий
2. Сверстать макет и натянуть на вордпресс
3. Сверстать макет использовать javascript

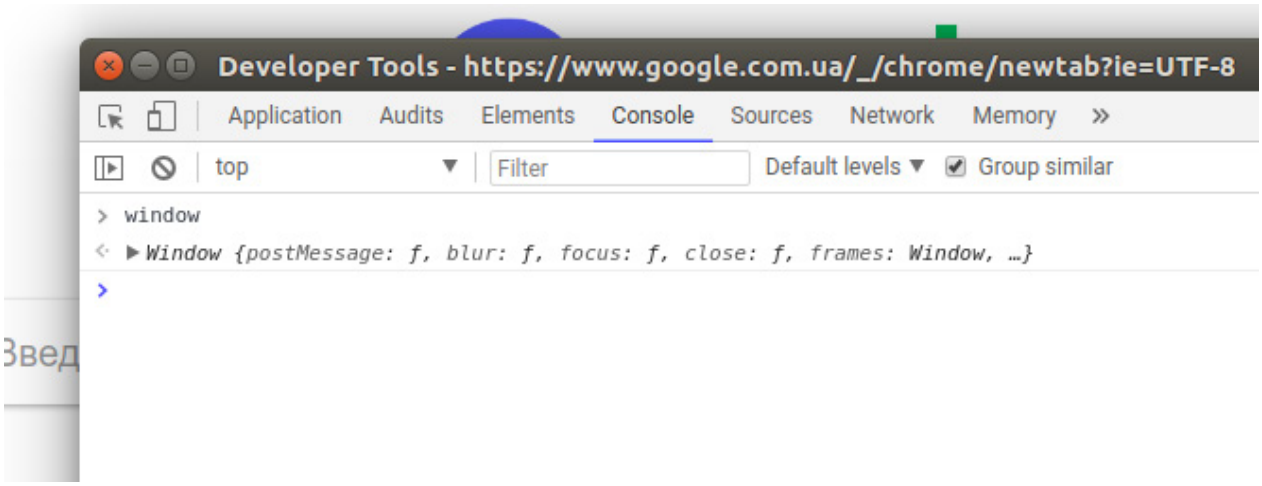
Javascript

Объекты javascript в браузере

Javascript или ECMAScript(так называется стандарт на основании которого реализован язык Javascript) - это язык, который не обязательно должен работать в браузере. Есть варианты языка которые работают на сервере или в микроконтроллерах. Но мы сейчас изучаем работу с языком в браузере.

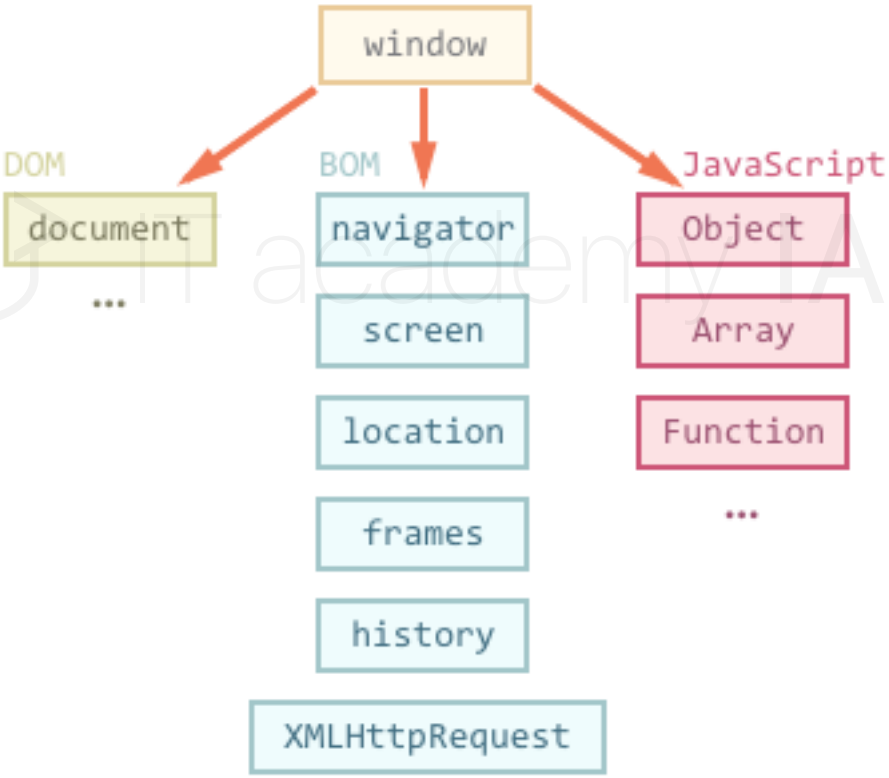
JS - это ООП язык, его ООП отличается от ООП языка php. Но есть и классы и объекты и методы. Если в для вызова метода вы писали
$obj->method();
То в JS это будет obj.method();, как видите разница небольшая.
Когда js работает в браузере у него есть ряд уже инициализированных объектов.

Главный объект станицы - это window, он включает в себя все остальные. Поэтому его мы можем упускать при обращении.
Для изучения объектов нам понадобиться консоль разработчика. Её можно открыть в бразуере Chrome клавишей F12.



Есть несколько вкладок, но сейчас нас интересует вкладка Console. Переключитесь на нее, здесь вы можете писать код JS и он сразу же будет выполняться.

Напишите window. Посмотрите что в нем содержиться.



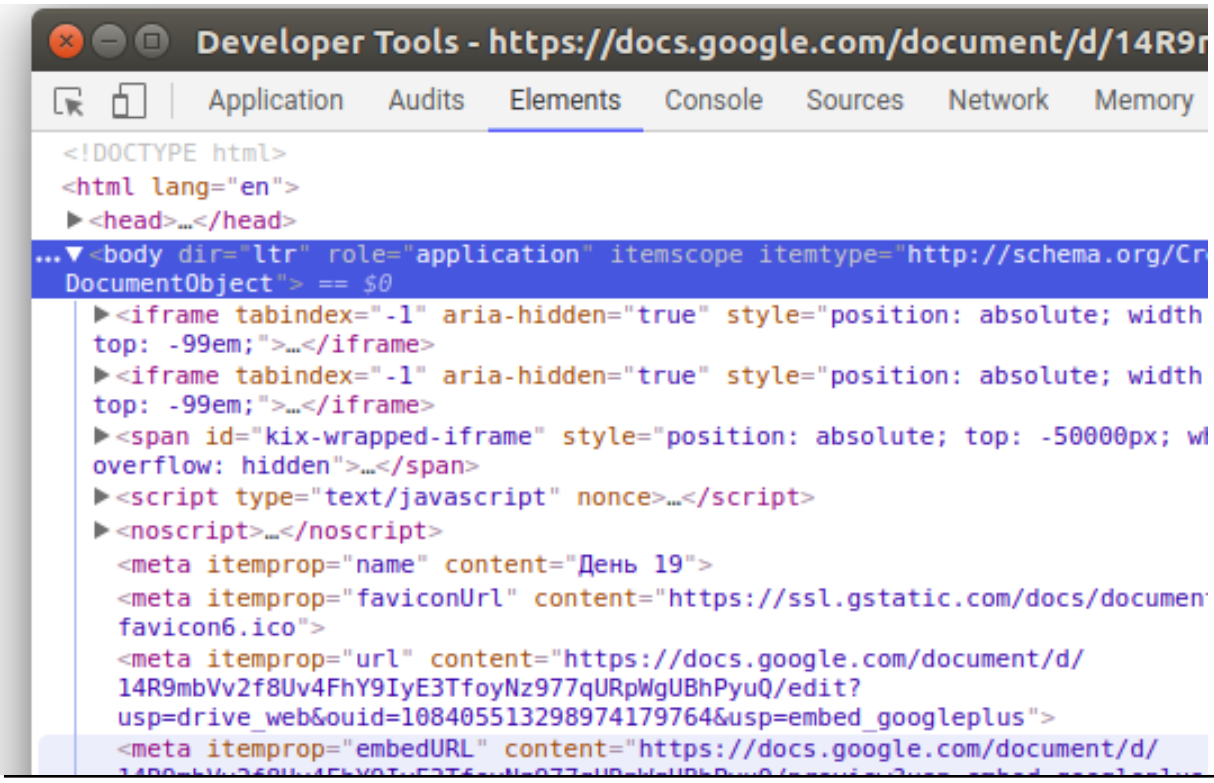
Эти объекты делят на три группы - DOM(Document Object Model), BOM (Browser Object Model) и объекты языка Javascript.
Поскольку все объекты принадлежат window, браузерная версия позволяет нам это не записывать.
Напишите в коснсле window.document и нажмите Enter и просто document. Как видите разницы между объектами нет.
BOM объекты нужны для работы с браузером. К примеру посмотрите на объекты navigator или history.

Посмотреть методы объектов тоже можно в консоле используя свойство proto

```
▼History {length: 2, scrollRestoration: "auto", state: null}
    length: 2
    scrollRestoration: "auto"
    state: null
  ▼__proto__: History
    ▶back: ƒ back()
    ▶forward: ƒ forward()
    ▶go: ƒ go()
      length: (...)
    ▶pushState: ƒ pushState()
    ▶replaceState: ƒ replaceState()
      scrollRestoration: (...)
      state: (...)
    ▶constructor: ƒ History()
      Symbol(Symbol.toStringTag): "History"
    ▶get length: ƒ ()
    ▶get scrollRestoration: ƒ ()
    ▶set scrollRestoration: ƒ ()
    ▶get state: ƒ ()
    ▶__proto__: Object
```

Объект history управляет историей браузера, откройте хабр и погуляйте по его страницам, а теперь выполните в консоли history.back(); как видите произошло тоже самое что и при клинике на кнопку назад.

Обратите внимание, что Chrome Dev Tools содержит и другие вкладки.
На вкладке Elements отображается HTML дерево документа.



Во вкладке Sources находятся загруженные странице ресурсы. А во вкладке Network запросы, которая страница отправляет на сервер. Мы еще поговорим о том, что при помощи javascript можно сделать так, чтобы страница отправляла на сервер запросы при этом не перезагружалась.

DOM дерево

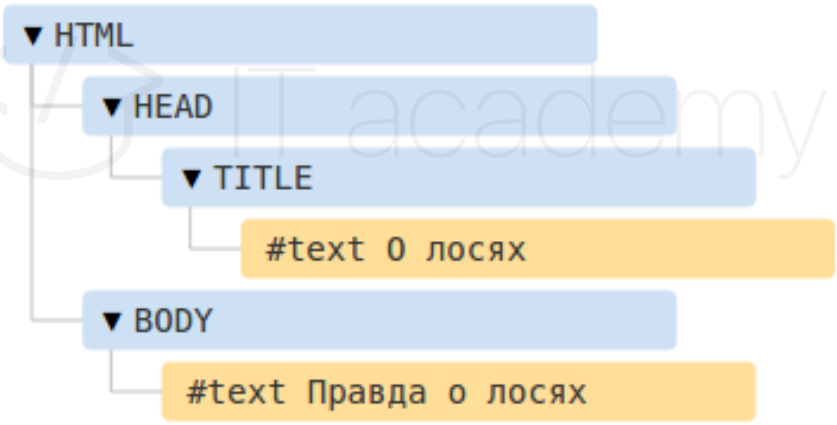
DOM - объектная модель документа представляет нам HTML дерево в виде объектов.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

Пример DOM

Построим, для начала, дерево DOM для следующего документа.
<!DOCTYPE HTML>
<html><head><title>О лосях</title></head><body>Правда о лосях</body></html>



DOM нужен для того, чтобы манипулировать страницей – читать информацию из HTML, создавать и изменять элементы.

Узел HTML можно получить как document.documentElement, а BODY – как document.body.

Получив узел, мы можем что-то сделать с ним.

Например, можно поменять цвет BODY и вернуть обратно:

```
document.body.style.backgroundColor = 'red';
alert( 'Поменяли цвет BODY' );

document.body.style.backgroundColor = '';
alert( 'Сбросили цвет BODY' );
```

DOM предоставляет возможность делать со страницей всё, что угодно.

Позже мы более подробно рассмотрим различные свойства и методы DOM-узлов.

Практическая работа

Откройте https://codepen.io/vromnichev24/pen/gxVMEE

Поиск по id
id1.style.backgroundColor = 'green';
window['id2'].style.backgroundColor = 'green';
document.getElementById('id3').style.backgroundColor = 'green';

Поиск по тегу
document.getElementsByTagName('input')[0].value = 5;
// или так
var form = document.getElementById('idform');
form.getElementsByTagName('input')[2].value = 6;

Поиск по CSS селектору querySelectorAll, querySelector
document.querySelector("input:last-child").value = 5;

Есть варианты искать по getElementsByClassName, поиск по атрибуту name getElementsByName

Дочерние элементы можно найти в свойстве children элемента. К примеру
var el1 = document.getElementById('idform').children[1];

Yii

Создание миграций

Сначала мы создадим новую миграцию для Tag таблицы. Чтобы создать новую миграцию, мы должны запустить эту команду:

php yii migrate/create create_tag_table

```
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.
```

Это создаст новый файл миграции create_tag_table в console/migrations/каталоге с таким именем: m<YYMMDD_HHMMSS>_create_tag_table

```
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');
  }
}
```

Как описано в официальной документации:

http://www.yiiframework.com/doc-2.0/guide-db-migrations.html#creating-migrations

В классе миграции вы должны написать код в методе up (), который вносит изменения в структуру базы данных. Вы также можете написать код в методе down (), чтобы вернуть изменения, сделанные функцией up (). Метод up () вызывается при обновлении базы данных с помощью этой миграции, тогда как метод down () вызывается при понижении базы данных.

Мы будем использовать новый синтаксис миграции, введенный в Yii 2.0.6. Для нашей таблицы тегов нам нужно всего несколько столбцов, таких как id, name, created_at и updated_at. Наш файл миграции будет выглядеть так:


```
<?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');
  }
}
```

Создаем таблицу для категорий

php yii migrate/create create_category_table

Код миграции:

```
<?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');
    }
}
```


php yii migrate/create create_post_table
Код миграции:

```
<?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', 'CAS-
CADE', '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');
    }
}
```

php yii migrate/create create_post_tag_table
Код миграции:

```
<?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', 'CAS-
CADE');
         $this->addForeignKey('fk_post_tag_tag', 'post_tag', 'tag_id', 'tag', 'id', 'CASCADE', 'CAS-
CADE');
    }

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

Выполним все миграции
php yii migrate/up

Миграция таблиц RBAC (на основе ролей)

Перед выполнением миграции таблиц RBAC необходимо настроить authManager компонент Yii . Yii предоставляет два типа менеджеров авторизации: yii\rbac\PhpManager и yii\rbac\DbManager. Первый использует файл сценария PHP для хранения данных авторизации, в то время как последний хранит данные авторизации в базе данных. В нашем блоге мы будем хранить наши данные RBAC в базе данных, поэтому мы будем использовать DbManager. Чтобы настроить authManager для работы с DbManager, мы должны добавить этот код в common/config/main.php:

```
'components' => [
   // ... other components
   'authManager' => [
      'class' => 'yii\rbac\DbManager',
   ],
   // ... other components
],
```

После настройки authManager мы можем запустить миграцию RBAC:
php yii migrate --migrationPath=@yii/rbac/migrations

Эта миграция будет создавать следующие таблицы:

1. itemTable : таблица для хранения элементов авторизации. По умолчанию auth_item.
2. itemChildTable : таблица для хранения иерархии элементов авторизации. По умолчанию auth_item_child.

```
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.
```

Теперь authManager можно получить доступ через Yii::$app->authManager. Мы создадим наши роли, разрешения и правила в следующем эпизоде, специально посвященном

Настройка прав доступа

Мы уже создали наш компонент authManager в предыдущем эпизоде. Теперь мы можем начать создавать роли, разрешения и их взаимные соединения.

Мы хотим иметь 3 роли: пользователь, автор и администратор. Роль пользователь - это роль по умолчанию после регистрации. «Автор» - привилегированная роль, которая может добавлять новые сообщения или редактировать собственные сообщения. И «admin» - это роль, которая может делать все, что «автор» может, но также может обновлять ВСЕ записи.

Для создания ролей, разрешений и их взаимных связей нам нужно создать новую миграцию с таким методом up. Давайте сначала разберем что тут происходит:

```
$auth = Yii::$app->authManager;

    /**
     * Permissions
     */

    // create and add "createPost" permission
    $createPost = $auth->createPermission('createPost');
    $createPost->description = 'User can create a post';
    $auth->add($createPost);

    // create and add "updatePost" permission
    $updatePost = $auth->createPermission('updatePost');
    $updatePost->description = 'User can update post';
    $auth->add($updatePost);

    /**
     * Roles
     */

    // create and add "user" role
    $user = $auth->createRole('user');
    $auth->add($user);

    // create and add "author" role
    $author = $auth->createRole('author');
    $auth->add($author);

    // create and add "admin" role
    $admin = $auth->createRole('admin');
    $auth->add($admin);

    /**
     * Mutual connections
     */

    // "author" can create new Post
    $auth->addChild($author, $createPost);
```

```
// “admin” can do everything what “author” can
    $auth->addChild($admin, $author);
    // ... and ...
    // “admin” can update ALL Posts
    $auth->addChild($admin, $updatePost);
```

Если вы хотите получить дополнительную информацию о конфигурации RBAC, я рекомендую вам ознакомиться с официальным руководством
http://www.yiiframework.com/doc-2.0/guide-security-authorization.html#configuring-rbac

Эта миграция создаст таблицы auth_itemи auth_item_child и заполнит их правилами.

Кроме того, мы должны автоматически назначать роль пользователя каждому новому Пользователю, зарегистрированному в нашем блоге. Для этого нам нужно обновить frontend\models\SignupForm действие s signup(). Нам просто нужно добавить 3 новые строки:

```
$auth = \Yii::$app->authManager;
$userRole = $auth->getRole('user');
$auth->assign($userRole, $user->getId());
```

Теперь весь метод signup () должен выглядеть так:

```
public function signup()
{
    if (!$this->validate()) {
        return null;
    }

    $user = new User();
    $user->username = $this->username;
    $user->email = $this->email;
    $user->setPassword($this->password);
    $user->generateAuthKey();
    $user->status = User::STATUS_ACTIVE;

    if ($user->save()) {
        // the following three lines were added:
        $auth = \Yii::$app->authManager;
        $userRole = $auth->getRole('user');
        $auth->assign($userRole, $user->getId());

        return $user;
    }

    return null;
}
```

Правила

Правила добавляют дополнительное ограничение для ролей и разрешений. Правило - это класс, наследующийся от yii\rbac\Rule. Он должен реализовать этот execute() вметод. В иерархии, которую мы создали ранее, автор не может редактировать свой собственный пост. Давайте исправим это. Сначала нам нужно правило, чтобы убедиться,

что пользователь является автором сообщения. Для этого нам нужно создать console/rbac/AuthorRule.php файл. Кроме того, вам нужно будет создать папку rbac в каталоге консоли. AuthorRule.php должен содержать:

```
<?php

namespace console\rbac;

use yii\rbac\Rule;

/**
 * Checks if authorID matches user passed via params
 */
class AuthorRule extends Rule
{
    public $name = 'isAuthor';

    /**
     * @param string|integer $user the user ID.
     * @param Item $item the role or permission that this rule is associated with
     * @param array $params parameters passed to ManagerInterface::checkAccess().
     * @return boolean a value indicating whether the rule permits the role or permission it is associated with.
     */
    public function execute($user, $item, $params)
    {
        return isset($params['model']) ? $params['model']->createdBy->id == $user : false;
    }
}
```

В приведенном выше правиле проверяется, создается ли сообщение пользователем $user. Мы создадим новое разрешение updateOwnPost и связам с ним новое правило. К нашей миграции мы должны дописать вот такой код:

```
$auth = Yii::$app->authManager;

    // add the rule
    $rule = new \console\rbac\AuthorRule();
    $auth->add($rule);

    // add the "updateOwnPost" permission and associate the rule with it.
    $updateOwnPost = $auth->createPermission('updateOwnPost');
    $updateOwnPost->description = 'Update own post';
    $updateOwnPost->ruleName = $rule->name;
    $auth->add($updateOwnPost);

    // get the "updatePost" permission
    $updatePost = $auth->getPermission('updatePost');

// get the "author" role
    $author = $auth->getRole('author');
```

```
// "updateOwnPost" will be used from "updatePost"
  $auth->addChild($updateOwnPost, $updatePost);

  // allow "author" to update their own posts
  $auth->addChild($author, $updateOwnPost);
```

Создайте миграцию ./yii migrate/create init_rbac

Она должна иметь вид:

```
<?php

use yii\db\Migration;


/**
 * Class m180313_200245_init_rbac
 */
class m180313_200245_init_rbac extends Migration
{
  /**
   * {@inheritdoc}
   */
  public function safeUp()
  {
    $auth = Yii::$app->authManager;

    /**
     * Права
     */

// Права на создание поста
    $createPost = $auth->createPermission('createPost');
    $createPost->description = 'User can create a post';
    $auth->add($createPost);

    // Права на редактирование поста
    $updatePost = $auth->createPermission('updatePost');
    $updatePost->description = 'User can update post';
    $auth->add($updatePost);
/**
     * Roles
     */

    // Создание роли Пользователь
    $user = $auth->createRole('user');
    $auth->add($user);

// Создание роли Автор
    $author = $auth->createRole('author');
    $auth->add($author);
```

```
// Создание роли Админ
        $admin = $auth->createRole('admin');
        $auth->add($admin);

        /**
         * Связываем права и роли
         */

        // Автор может создавать посты
        $auth->addChild($author, $createPost);

        // Админ делает тоже что и автор
        $auth->addChild($admin, $author);
        // ... and ...
        // Админ может редактировать все посты
        $auth->addChild($admin, $updatePost);
        //Создали наше правило
                $rule = new \console\rbac\AuthorRule();
                $auth->add($rule);

                // Привязали правило к праву
                $updateOwnPost = $auth->createPermission('updateOwnPost');
                $updateOwnPost->description = 'Update own post';
                $updateOwnPost->ruleName = $rule->name;
                $auth->add($updateOwnPost);

                // Получили право на редактирование поста
                $updatePost = $auth->getPermission('updatePost');

                // Получили роль автора
                $author = $auth->getRole('author');

                // Связали права редактирования поста с правилом
                $auth->addChild($updateOwnPost, $updatePost);

                // Связали роль автора с правилом
                $auth->addChild($author, $updateOwnPost);

    }

    /**
     * {@inheritdoc}
     */
    public function safeDown()
    {
        $auth = Yii::$app->authManager;
        $auth->removeAll();
    }

}
```

Запустите миграции ./yii migrate

Сейчас самое подходящее время для регистрации в вашем блоге. После регистрации вы должны автоматически войти в систему, и вы должны назначить роль пользователя в auth_assignment таблице. Вручную измените свою роль на «admin». Вы можете сделать это, изменив значение «user» на «admin» в item_name столбце.

Чтобы проверить, может ли пользователь создать новое сообщение:

```
if (\Yii::$app->user->can('createPost')) {
    // create post
}
```

Чтобы проверить, может ли пользователь обновлять сообщение, нам нужно передать дополнительный параметр:

```
if (\Yii::$app->user->can('updatePost', ['model' => $post])) {
    // update post
}
```

Мы продолжим строить наш блог на следующем уроке.

Домашнее задание

1. Задачник День 19