Лабораторна робота №4

**Тема:** Магазин на Yii2

**Мета:** Створити власний магазин з допомогою Yii2

Хід роботи

До створення магазину на Yii2 я підійшов по принципу ‘database first’, тобто перш за все сформував базу даних з допомогою міграцій.

Таблиці:

user

**public function** safeUp()  
{  
 $this->createTable('user', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'email' => $this->string(64),  
 'username' => $this->string(32),  
 'hash' => $this->string(60),  
 'salt' => $this->string(32),  
 'auth\_key' => $this->string(32),  
 'access\_token' => $this->string(32),  
 'verification\_token' => $this->string(32),  
 'role' => $this->tinyInteger(1)->unsigned()->notNull()->defaultValue(0),  
 'status' => $this->tinyInteger(1)->unsigned()->notNull()->defaultValue(0),  
 ]);  
}

product, brand, category, features

**public function** safeUp()  
{  
 $this->createTable('product', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'title' => $this->string(32),  
 'description' => $this->text(),  
 'price' => $this->float(2)->unsigned(),  
 'amount' => $this->integer(11)->unsigned(),  
 'category\_id' => $this->integer(11)->unsigned(),  
 'brand\_id' => $this->integer(11)->unsigned(),  
 'created\_at' => $this->string(16),  
 'updated\_at' => $this->string(16),  
 'image' => $this->string(64),  
 ]);  
  
 $this->createTable('brand', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'name' => $this->string(32),  
 'contact' => $this->string(64)  
 ]);

$this->createTable('category', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'name' => $this->string(32),  
 ]);  
  
 $this->createTable('feature', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'name' => $this->string(32),  
 ]);  
  
 $this->createTable('category\_feature', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'category\_id' => $this->integer(11)->unsigned(),  
 'feature\_id' => $this->integer(11)->unsigned()  
 ]);  
  
 $this->createTable('product\_feature', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'product\_id' => $this->integer(11)->unsigned(),  
 'feature\_id' => $this->integer(11)->unsigned(),  
 'value' => $this->string(16)->unsigned(),  
 ]);  
}

order

**public function** safeUp()  
{  
 $this->createTable('order', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'description' => $this->string(255),  
 'status' => $this->tinyInteger(1)->unsigned()->notNull()->defaultValue(0),  
 'delivery\_type' => $this->tinyInteger(1)->unsigned()->notNull()->defaultValue(0),  
 'destination' => $this->string(64),  
 'user\_id' => $this->integer(11)->unsigned(),  
 'created\_at' => $this->string(16)  
 ]);  
  
 $this->createTable('order\_item', [  
 'id' => $this->primaryKey(11)->unsigned(),  
 'order\_id' => $this->integer(11)->unsigned(),  
 'product\_id' => $this->integer(11)->unsigned(),  
 'amount' => $this->integer(11)->unsigned()  
 ]);  
}

relations

**public function** safeUp()  
{  
 // creates index for column `order\_id` in table `order\_item`  
 $this->createIndex(  
 'idx-order\_item-order\_id',  
 'order\_item',  
 'order\_id'  
 );  
  
 //add foreign key for table `order`  
 $this->addForeignKey(  
 'fk-order\_item-order\_id',  
 'order\_item',  
 'order\_id',  
 'order',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `product\_it` in table `order\_item`  
 $this->createIndex(  
 'idx-order\_item-product\_id',  
 'order\_item',  
 'product\_id'  
 );  
  
 //add foreign key for table `product`  
 $this->addForeignKey(  
 'fk-order\_item-product\_id',  
 'order\_item',  
 'product\_id',  
 'product',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `user\_id` in table `order`  
 $this->createIndex(  
 'idx-order-user\_id',  
 'order',  
 'user\_id'  
 );  
  
 //add foreign key for table `user`  
 $this->addForeignKey(  
 'fk-order-user\_id',  
 'order',  
 'user\_id',  
 'user',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `category\_id` in table `product`  
 $this->createIndex(  
 'idx-product-category\_id',  
 'product',  
 'category\_id'  
 );  
  
 //add foreign key for table `category`  
 $this->addForeignKey(  
 'fk-product-category\_id',  
 'product',  
 'category\_id',  
 'category',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `category\_id` in table `category\_feature`  
 $this->createIndex(  
 'idx-category\_feature-category\_id',  
 'category\_feature',  
 'category\_id'  
 );  
  
 //add foreign key for table `category`  
 $this->addForeignKey(  
 'fk-category\_feature-category\_id',  
 'category\_feature',  
 'category\_id',  
 'category',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `feature\_id` in table `category\_feature`  
 $this->createIndex(  
 'idx-category\_feature-feature\_id',  
 'category\_feature',  
 'feature\_id'  
 );  
  
 //add foreign key for table `feature`  
 $this->addForeignKey(  
 'fk-category\_feature-feature\_id',  
 'category\_feature',  
 'feature\_id',  
 'feature',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `brand\_id` in table `product`  
 $this->createIndex(  
 'idx-product-brand\_id',  
 'product',  
 'brand\_id'  
 );  
  
 //add foreign key for table `brand`  
 $this->addForeignKey(  
 'fk-product-brand\_id',  
 'product',  
 'brand\_id',  
 'brand',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
 // creates index for column `product\_id` in table `product\_feature`  
 $this->createIndex(  
 'idx-product\_feature-product\_id',  
 'product\_feature',  
 'product\_id'  
 );  
  
 //add foreign key for table `product\_feature`  
 $this->addForeignKey(  
 'fk-product\_feature-product\_id',  
 'product\_feature',  
 'product\_id',  
 'product',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
  
 // creates index for column `feature\_id` in table `product\_feature`  
 $this->createIndex(  
 'idx-product\_feature-feature\_id',  
 'product\_feature',  
 'feature\_id'  
 );  
  
 //add foreign key for table `feature`  
 $this->addForeignKey(  
 'fk-product\_feature-feature\_id',  
 'product\_feature',  
 'feature\_id',  
 'feature',  
 'id',  
 'CASCADE',  
 'CASCADE'  
 );  
}

rating

**public function** safeUp()  
{  
 $this->createTable('rating', [  
 'id' => $this->primaryKey()->unsigned(),  
 'product\_id' => $this->integer(11)->unsigned(),  
 'user\_id' => $this->integer(11)->unsigned(),  
 'value' => $this->integer(11)->unsigned(),  
 ]);  
  
 // creates index for column `product\_id`  
 $this->createIndex(  
 'idx-rating-product\_id',  
 'rating',  
 'product\_id'  
 );  
  
 // add foreign key for table `product`  
 $this->addForeignKey(  
 'fk-rating-product\_id',  
 'rating',  
 'product\_id',  
 'product',  
 'id',  
 'CASCADE'  
 );  
  
 // creates index for column `user\_id`  
 $this->createIndex(  
 'idx-rating-user\_id',  
 'rating',  
 'user\_id'  
 );  
  
 // add foreign key for table `user`  
 $this->addForeignKey(  
 'fk-rating-user\_id',  
 'rating',  
 'user\_id',  
 'user',  
 'id',  
 'CASCADE'  
 );  
}

comment

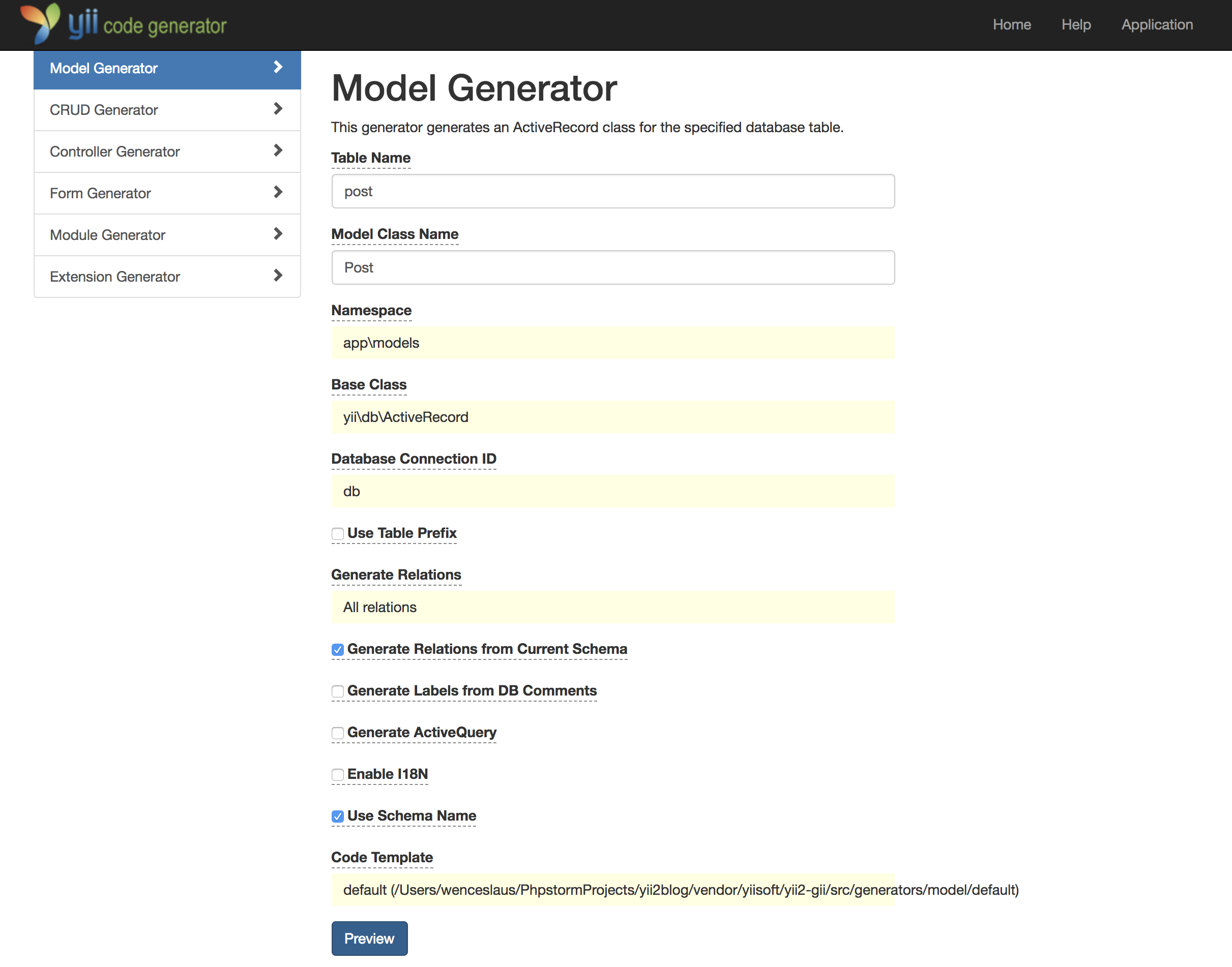
**public function** safeUp()  
{  
 $this->createTable('comment', [  
 'id' => $this->primaryKey(),  
 'product\_id' => $this->integer(11)->unsigned(),  
 'user\_id' => $this->integer(11)->unsigned(),  
 'content' => $this->text(),  
 'created\_at' => $this->string(16),  
 ]);  
  
 // creates index for column `product\_id`  
 $this->createIndex(  
 'idx-comment-product\_id',  
 'comment',  
 'product\_id'  
 );  
  
 // add foreign key for table `product`  
 $this->addForeignKey(  
 'fk-comment-product\_id',  
 'comment',  
 'product\_id',  
 'product',  
 'id',  
 'CASCADE'  
 );  
  
 // creates index for column `user\_id`  
 $this->createIndex(  
 'idx-comment-user\_id',  
 'comment',  
 'user\_id'  
 );  
  
 // add foreign key for table `user`  
 $this->addForeignKey(  
 'fk-comment-user\_id',  
 'comment',  
 'user\_id',  
 'user',  
 'id',  
 'CASCADE'  
 );  
}

discount

**public function** safeUp()  
{  
 $this->createTable('discount', [  
 'id' => $this->primaryKey()->unsigned(),  
 'category\_id' => $this->integer(11)->unsigned(),  
 'user\_id' => $this->integer(11)->unsigned(),  
 'percent' => $this->integer(11)->unsigned(),  
 'created\_at' => $this->string(16),  
 ]);  
  
 // creates index for column `category\_id`  
 $this->createIndex(  
 'idx-discount-category\_id',  
 'discount',  
 'category\_id'  
 );  
  
 // add foreign key for table `category`  
 $this->addForeignKey(  
 'fk-discount-category\_id',  
 'discount',  
 'category\_id',  
 'category',  
 'id',  
 'CASCADE'  
 );  
  
 // creates index for column `user\_id`  
 $this->createIndex(  
 'idx-discount-user\_id',  
 'discount',  
 'user\_id'  
 );  
  
 // add foreign key for table `user`  
 $this->addForeignKey(  
 'fk-discount-user\_id',  
 'discount',  
 'user\_id',  
 'user',  
 'id',  
 'CASCADE'  
 );  
}

Після створення таблиць з допомогою команди php yii migrate/fresh можна згенерувати моделі в Gii Code Generator:

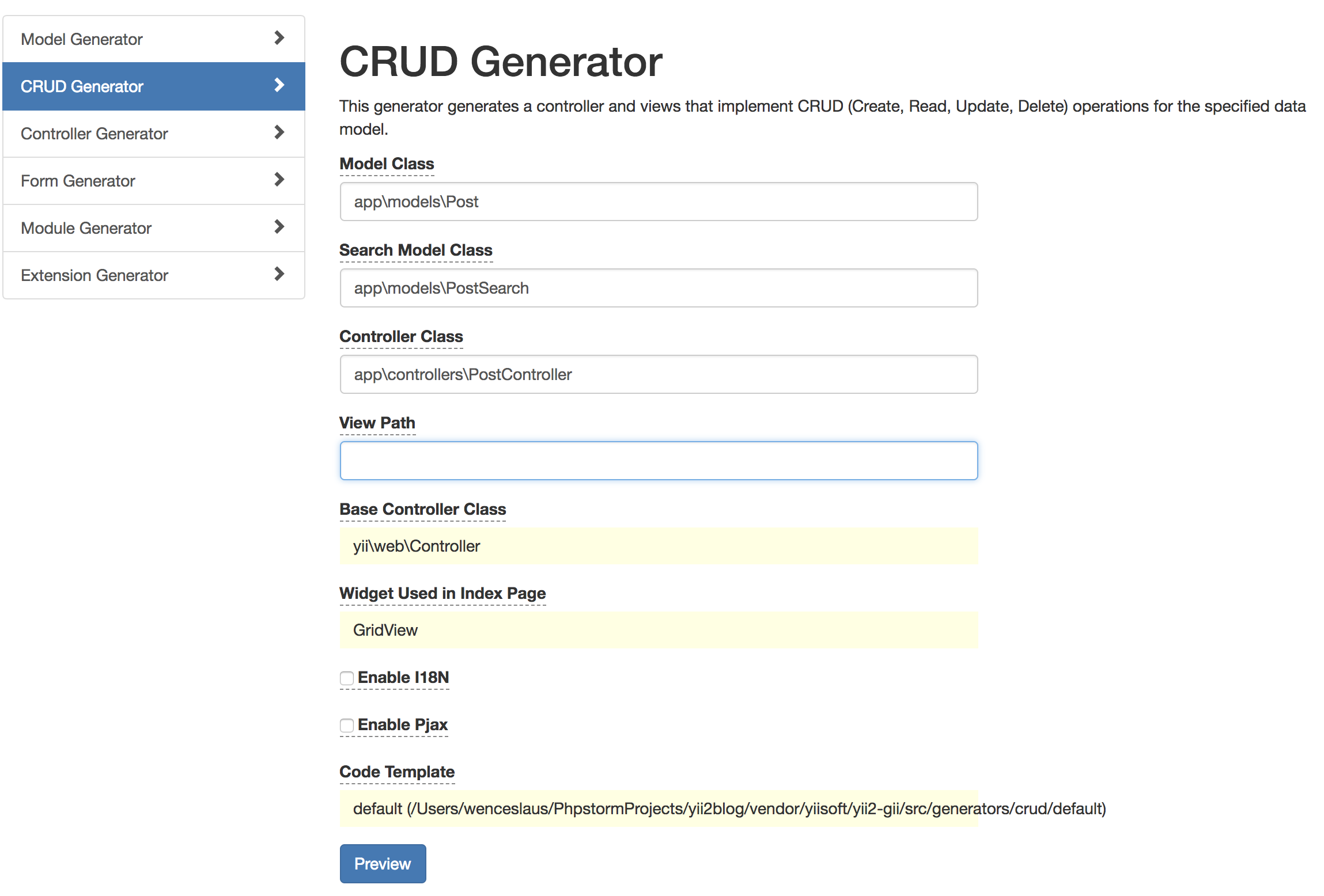
- в адресній строці після index.php дописуємо рендер ‘ ?r=gii ’;



На заголовки типу Table Name, Model Class Name і т.д. можна навестись для отримання інструкцій і слідуючи їм згенеруємо моделі для всіх таблиць.

Далі ми генеруємо CRUD (абр. Create Read Update Delete), CRUD це інтерфейс для зручного керування даними. Разом з ним згенеруються і контролери для моделей, отже окремо їх генерувати не потрібно.

Як правило CRUD будуть користуватись редактори чи адміністратори.



Тепер можна рефакторити згенеровані моделі та CRUD. Розгляд кожного елементу магазину є нераціональним і дуже об’ємним тому я поясню лише основний функціонал.

Загальний опис

На фреймворку я реалізував цілком робочий прототип потенційного магазину. В його функціонал входить базовий набір функцій магазина: керування продуктами, категоріями, характеристиками, користувачами та замовленнями.

Динамічні категорії і характеристики

Особливістю створення категорій і характеристик є те що я реалізував їх динамічно. Тобто можна створити будь-яку категорію і призначити цій категорії будь-які характеристики, отже сайт можна адаптувати під будь-який тип магазину/товарів.

З допомогою додатку для Yii2 я реалізовую зв’язок багато-до-багатьох, його звичайно ще можна реалізувати через link(), але так це займає менше місця в коді.

**В поведінках прописуємо додаток.**

**public function** behaviors()  
{  
 **return** [  
 [  
 'class' => ManyToManyBehavior::*className*(),  
 'relations' => [  
 'feature\_ids' => 'features',  
 ],  
 ],  
 ];  
}

**Використовуємо зв’язок категорії з характеристиками.**

**public function** getFeatures()  
{  
 **return** $this->hasMany(Feature::*className*(), ['id' => 'feature\_id'])->via('categoryFeatures');  
}

Процес створення характеристик і категорій в загальному стандартний.

Створення продукту

Головною складністю проекту є створення продукту, складність саме в підвантаженні характеристик відповдіно до категорій і реалізації створення і оновлення записів.

Потрібно прописати зв’язки моделі продукту що зв’яжуть її з характеристиками, при цьому потрібно враховувати що характеристики(саме назви) зберігаються в окремій таблиці, і мають проміжну таблицю яка вказує яка характеристика описує який продукт і її значення.

**Характеристики:**

**public function** getCategoryFeatures()  
{  
 **return** $this->hasMany(Feature::*className*(), ['id' => 'feature\_id'])->via('features');  
}

**Проміжна таблиця з значеннями**

**public function** getFeatures()  
{  
 **return** $this->hasMany(ProductFeature::*className*(), ['product\_id' => 'id']);  
}

При цьому в характеристик є свої зв’язки в моделі і їх я використовую щоб зручно добиратись до назви характеристики і її значення для відповідного продукту.

Найскладнішим є процес create/update. Перш за все я створив 2 форми(моделі) для прийняття даних продукту і прийняття даних характеристик.

Форма для продукту є стандартною моделлю що пропускає через себе дані і потім передає в саме модель продукту для збереження.

Форма для характеристик – динамічна модель. Вона генерується в контролері в залежності від обраної користувачем категорії.

Слід зауважити що використовувати форми-моделі слід тоді коли ми будемо отримувати від користувача дані які не будуть на пряму зберігатись у базу даних.

В моєму випадку причиною є також специфіка процесу create/update, оскільки в формі-моделі реалізовано методи які повертають певні сигнали про зміну категорію і потребу в підвантаженні нових input-ів для відповідних характеристик.

В загальному реалізовані такі методи які описувати в моделі продукту було б зовсім нелогічно з точки зору семантики фреймворку.

ProductForm.php

*/\*\*  
 \* Model ProductForm - form behind the Product Create/Update action.  
 \** ***@package*** *app\models  
 \*  
 \** ***@property*** *Category $category  
 \*/***class** ProductForm **extends** Model  
{  
// General  
 **public** $imageFile, $title, $description, $price, $amount, $category\_id, $brand\_id;  
// Only on update  
 **public** $source, $id, $imagePreview;  
// Utility  
 **public** $loadCategory;  
  
 */\*\*  
 \* ProductForm constructor.  
 \** ***@param*** *Product|null $product  
 \*/* **public function** \_\_construct($product = **null**)  
 {  
 **parent**::*\_\_construct*();  
 **if** ($product) {  
 $this->source = $product;  
 $this->id = $product->id;  
 $this->imagePreview = $product->image;  
 $this->setAttributes($product->attributes, **false**);  
 }  
 }  
  
 */\*\*  
 \** ***@inheritdoc*** *\*/* **public function** rules()  
 {  
 **return** [  
 [['id'], 'integer'],  
 [['price'], 'number'],  
 [['amount', 'category\_id', 'brand\_id'], 'integer'],  
 [['title'], 'string', 'max' => 32],  
// Length restriction may be redundant as descriptions is of text type.  
 [['description'], 'string', 'max' => 255],  
 [['brand\_id'], 'exist', 'skipOnError' => **true**, 'targetClass' => Brand::*className*(), 'targetAttribute' => ['brand\_id' => 'id']],  
 [['category\_id'], 'exist', 'skipOnError' => **true**, 'targetClass' => Category::*className*(), 'targetAttribute' => ['category\_id' => 'id']],  
 [['title', 'description', 'price', 'amount'], 'required'],  
 [['imageFile'], 'image'],  
 [['imagePreview'], 'safe'],  
 [['loadCategory'], 'boolean'],  
 ];  
 }

**Тут реалізовані методи які повертають сигнали про потребу в завантаженні нової категорії до представлення форми loadsCategory() і loadsSourceCategory()**

*/\*\*  
 \* Perform a check if this form loads category features after user selected a category.  
 \** ***@return*** *bool  
 \*/* **public function** loadsCategory()  
 {  
 **if** ($this->loadCategory) {  
 $this->loadCategory = **false**;  
 **return true**;  
 }  
 **return false**;  
 }  
  
 **public function** loadsSourceCategory()  
 {  
 **if** ($this->category\_id == $this->source->category\_id)  
 **return true**;  
 **return false**;  
 }  
  
 */\*\*  
 \** ***@param*** *ProductFeatureForm $featureForm  
 \** ***@param*** *bool $runValidation  
 \** ***@return*** *int|bool if saved successfully return saved product id, else return false  
 \*/* **public function** save($featureForm, $runValidation = **true**)  
 {  
 $product = **new** Product();  
 $product->setAttributes($this->attributes);  
  
 **if** ($imageFile = UploadedFile::*getInstance*($this, 'imageFile')) {  
 $product->image = 'uploads/' . $imageFile->baseName . '.' . $imageFile->extension;  
 }  
  
 **if** ($product->save($runValidation) && $featureForm->save($product)) {  
 **if** ($imageFile) $imageFile->saveAs($product->image);  
 **return** $product->id;  
 }  
 **return false**;  
 }  
  
 */\*\*  
 \** ***@param*** *ProductFeatureForm $featureForm  
 \** ***@param*** *bool $runValidation  
 \** ***@return*** *bool|int  
 \*/* **public function** update($featureForm, $runValidation = **true**)  
 {  
 $product = $this->source;  
 $product->setAttributes($this->attributes);  
  
 **if** ($imageFile = UploadedFile::*getInstance*($this, 'imageFile')) {  
 $product->image = 'uploads/' . $imageFile->baseName . '.' . $imageFile->extension;  
 }  
  
 **if** ($product->save($runValidation) && $featureForm->update($product)) {  
 **if** ($imageFile) $imageFile->saveAs($product->image);  
 **return** $product->id;  
 }  
 \Yii::*$app*->session->setFlash('danger', 'ProductForm Update failed.');  
 **return false**;  
 }  
  
 */\*\*  
 \** ***@return*** *null|Category  
 \*/* **public function** getCategory()  
 {  
 **return** Category::*findOne*($this->category\_id);  
 }  
  
}

ProductFeatureForm.php

**class** ProductFeatureForm **extends** DynamicModel  
{  
 */\*\*  
 \* ProductFeatureForm constructor.  
 \** ***@param*** *Category|null $category  
 \** ***@param*** *Product|null $product  
 \** ***@param*** *array $config  
 \** ***@throws*** *\Exception  
 \*/*

**Слід звернути увагу на конструктор, він динамічно створює формі-моделі атрибути в залежності від характеристик що належать якійсь категорії.**

**public function** \_\_construct($category = **null**, $product = **null**, $config = [])  
 {  
 **if** ($category) {  
 **parent**::*\_\_construct*($category->getFeatureNames(), $config);  
 **if** ($product) {  
 **if** ($product->category\_id == $category->id) {  
 $this->setAttributes($product->getFeaturesAsAttributes(), **false**);  
 } **else** {  
 **throw new** \Exception('Given category is not the same as given product\'s category!');  
 }  
 }  
 } **else** {  
 **if** ($category = Category::*find*()->one()) {  
 **parent**::*\_\_construct*($category->getFeatureNames(), $config);  
 } **else** {  
 **parent**::*\_\_construct*();  
 }  
 }  
 $this->generateRules();  
 }  
  
 **public function** generateRules()  
 {  
 **foreach** ($this->attributes **as** $name => $value) {  
 $this->addRule($name, 'string', ['max' => 32]);  
 $this->addRule($name, 'required');  
 }  
 }  
  
 */\*\*  
 \** ***@param*** *Product $product  
 \** ***@return*** *bool succession of procedure  
 \*/* **public function** save($product)  
 {  
 **try** {  
 **foreach** ($this->attributes **as** $name => $value) {  
 $product->link('categoryFeatures', Feature::*find*()->where(['name' => $name])->one(), ['value' => $value]);  
// *TODO: Remove debug code.*// echo $name . ' ' . $value . '<br>';  
 }  
// die;  
 **return true**;  
 } **catch** (\Exception $e) {  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \** ***@param*** *Product $product  
 \** ***@return*** *bool succession of procedure  
 \*/* **public  
 function** update($product)  
 {  
 **try** {  
 **foreach** ($product->getFeaturesAsAttributes() **as** $name => $value) {  
 $product->unlink('categoryFeatures', Feature::*find*()->where(['name' => $name])->one(), **true**);  
 **echo** $name . ' ' . $value . '<br>';  
 }  
// *TODO: Remove debug code. 2*// foreach ($this->attributes as $name => $value) {  
// $product->link('categoryFeatures', Feature::find()->where(['name' => $name])->one(), ['value' => $value]);  
// echo $name . ' ' . $value . '<br>';  
// }  
// die;  
 $this->save($product);  
 **return true**;  
 } **catch** (\Exception $e) {  
 **return false**;  
 }  
 }  
}

В загальному в двох формах-моделях реалізовано створення/оновлення продукту з його характеристиками.

В контролері все пов’язується і реалізовується підвантаження характеристик та передача моделей до представлення форми.

ProductController.php

*/\*\*  
 \* Creates a create Product model.  
 \* If creation is successful, the browser will be redirected to the 'view' page.  
 \** ***@return*** *mixed  
 \** ***@throws*** *\Exception  
 \*/***public function** actionCreate()  
{  
 $productForm = **new** ProductForm();  
 $featureForm = **new** ProductFeatureForm();  
  
 **if** ($productForm->load(Yii::*$app*->request->post())) {  
 //Check if the submit was partial, only to load a different category.  
 **if** ($productForm->loadsCategory()) {  
 $featureForm = **new** ProductFeatureForm($productForm->category);  
 **return** $this->render('create', [  
 'product' => $productForm,  
 'features' => $featureForm,  
 ]);  
 }  
  
 **if** ($featureForm->load(Yii::*$app*->request->post()) && $product\_id = $productForm->save($featureForm)) {  
 **return** $this->redirect(['view', 'id' => $product\_id]);  
 }  
  
 }  
  
 **return** $this->render('create', [  
 'product' => $productForm,  
 'features' => $featureForm,  
 ]);  
}  
  
*/\*\*  
 \* Updates an existing Product model.  
 \* If update is successful, the browser will be redirected to the 'view' page.  
 \** ***@param*** *integer $id  
 \** ***@return*** *mixed  
 \** ***@throws*** *NotFoundHttpException if the model cannot be found  
 \** ***@throws*** *\Exception  
 \*/***public function** actionUpdate($id)  
{  
 $productForm = **new** ProductForm($this->findModel($id));  
 $featureForm = **new** ProductFeatureForm($productForm->category, $productForm->source);  
  
 **if** ($productForm->load(Yii::*$app*->request->post())) {  
 //Check if the submit was partial, only to load a different category.  
 **if** ($productForm->loadsCategory()) {  
 $featureForm = $productForm->loadsSourceCategory()  
 ? **new** ProductFeatureForm($productForm->category, $productForm->source)  
 : **new** ProductFeatureForm($productForm->category);  
 **return** $this->render('update', [  
 'product' => $productForm,  
 'features' => $featureForm,  
 ]);  
 }  
  
 $featureForm = **new** ProductFeatureForm($productForm->category);  
 **if** ($featureForm->load(Yii::*$app*->request->post()) && $product\_id = $productForm->update($featureForm)) {  
 **return** $this->redirect(['view', 'id' => $product\_id]);  
 }  
  
 }  
  
 **return** $this->render('update', [  
 'product' => $productForm,  
 'features' => $featureForm,  
 ]);  
}

Представлення форми я адаптував до всього процесу.

\_form.php

*/\** ***@var*** *$this yii\web\View \*/  
/\** ***@var*** *$product ProductForm \*/  
/\** ***@var*** *$features ProductFeatureForm \*/  
/\** ***@var*** *$form ActiveForm \*/***?>**<div class="product-form">  
  
 **<?php** $form = ActiveForm::*begin*(['options' => ['enctype' => 'multipart/form-data', 'id' => 'product-form']]); **?>** <img class="center-block col-lg-6" id="preview"  
 src="**<?=** (**isset**($product->source->image)) ? $product->source->image : '' **?>**">  
  
 **<?=** $form->field($product, 'imageFile')->fileInput(['onchange' => 'loadFile(event)']) **?>  
  
 <?=** $form->field($product, 'title')->textInput(['maxlength' => **true**]) **?>  
  
 <?=** $form->field($product, 'brand\_id')->dropDownList(ArrayHelper::*map*(Brand::*find*()->all(), 'id', 'name')) **?>  
  
 <?=** $form->field($product, 'description')->textarea() **?>  
  
 <?=** $form->field($product, 'price')->textInput() **?>  
  
 <?=** $form->field($product, 'amount')->textInput() **?>  
  
 <?=** $form->field($product, 'category\_id')->dropDownList(ArrayHelper::*map*(Category::*find*()->all(), 'id', 'name'), ['onchange' => 'loadFeatures(event)']) **?>** <!-- Hidden checkbox used with javaScript -->  
 <div class="hidden">  
 **<?=** $form->field($product, 'loadCategory')->checkbox(['id' => 'loadCategory-checkbox', 'class' => 'hidden']) **?>** </div>  
  
 **<?php** //Dynamically render features by chosen category.  
 **foreach** ($features->attributes **as** $name => $value) {  
 **echo** $form->field($features, $name)->textInput();  
 } **?>** <div class="form-group">  
 **<?=** Html::*submitButton*('Save', ['class' => 'btn btn-success']) **?>** </div>  
  
 **<?php** ActiveForm::*end*(); **?>**</div>

**Для того щоб реалізувати динамічне підвантаження input-ів для моделей характеристик продукту я дописав JavaScript в кінці представлення форми і створив декілька прихованих CheckBox input-ів, щоб передавати сигнали про зміну категорії в форму-модель ProductForm.php.**

<script>  
 **var** loadFile = **function** (event) {  
 **var** output = document.getElementById('preview');  
 output.src = URL.createObjectURL(event.target.files[0]);  
 };  
 **var** loadFeatures = **function** (event) {  
 **var** form = document.getElementById('product-form');  
 **var** requestCategory = document.getElementById('loadCategory-checkbox');  
 requestCategory.checked = **true**;  
 form.submit();  
 }  
</script>

Особливості замовлення та реєстрації користувача

В якості функціоналу я реалізував відправлення пошти на пошку користувача у випадку завершення або вікладення замовлення та для підтвердження реєстрації користувача.

Order.php

**public function** sendEmailNotification()  
{  
 **if** ($this->isPending()) {  
 **return** Yii::*$app*->mailer  
 ->compose(['html' => 'order-notify-pending-html'], ['user' => $this->user, 'order' => $this])  
 ->setTo($this->getUserEmail())  
 ->setFrom(Yii::*$app*->params['adminEmail'])  
 ->setSubject('Your order is pending!')  
 ->send();  
 }  
 **if** ($this->isCompleted()) {  
 **return** Yii::*$app*->mailer  
 ->compose(['html' => 'order-notify-completed-html'], ['user' => $this->user, 'order' => $this])  
 ->setTo($this->getUserEmail())  
 ->setFrom(Yii::*$app*->params['adminEmail'])  
 ->setSubject('Your order is completed!')  
 ->send();  
 }  
 **return false**;  
}

User.php

**public function** validateVerificationToken($token)  
{  
 **return** $this->verification\_token == $token;  
}  
  
*/\*\*  
 \* Sends mail with verification message  
 \** ***@return*** *bool if mail was sent successfully  
 \*/***public function** sendEmailVerification()  
{  
 **return** Yii::*$app*->mailer  
 ->compose(['html' => 'user-verify-html'], ['user' => $this])  
 ->setTo($this->email)  
 ->setFrom(Yii::*$app*->params['adminEmail'])  
 ->setSubject('Account verification.')  
 ->send();  
}

Вище показано методи для відправлення пошти та верифікації юзера. В обох їх контролерах звичайно є дії(action) для здійснення цих операцій, але вони по суті прості і лише викликають ці методи та дають повідомлення про успішну відправку чи невдачу.

Для верифікації в контролері сайту є дія верифікації яка приймає токен.

SiteController.php

**public function** actionVerifyUser($token)  
{  
 */\*\** ***@var*** *User $user \*/* **if** ($user = User::*findIdentityByVerificationToken*($token))  
 **if** ($user->verify()) {  
 Yii::*$app*->session->setFlash('success', 'Account verified!');  
 Yii::*$app*->user->login($user);  
 **return** $this->render('about');  
 }  
 Yii::*$app*->session->setFlash('danger', 'Invalid token.');  
 **return** $this->render('about');  
}

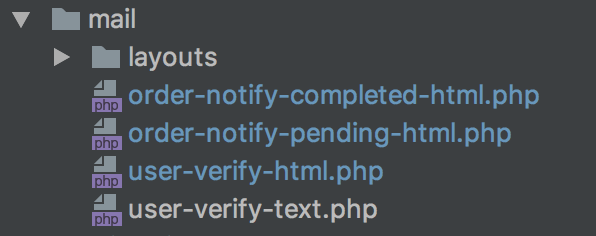
Посилання на цю дію згенеровано і відіслано через шаблон пошти: 

Рисунок 1. Папка з поштовими шаблонами

user-verify-html.php

**<?php  
use** yii\helpers\Html;  
*/\*\** ***@var*** *$user \app\models\User \*/*$link = Yii::*$app*->urlManager->createAbsoluteUrl(['site/verify-user', 'token' => $user->verification\_token]);  
**?>**<div class="password-reset" style="font-size: 12px; font-family: Arial, Helvetica, sans-serif;">  
 <h3 style="font-size: 18px">Hello **<?=** Html::*encode*($user->username) **?>**!</h3>  
  
 <p>Please, follow the link below to verify your email:</p>  
  
 <p>**<?=** Html::*a*($link, $link) **?>**</p>  
</div>

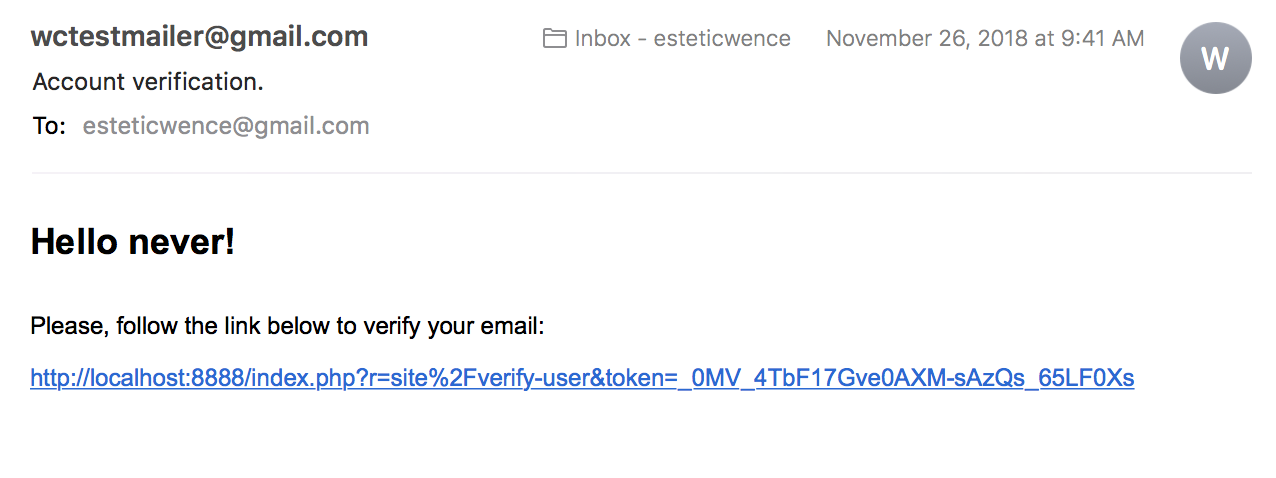


Рисунок 2. Пошта про підтвердження користувача

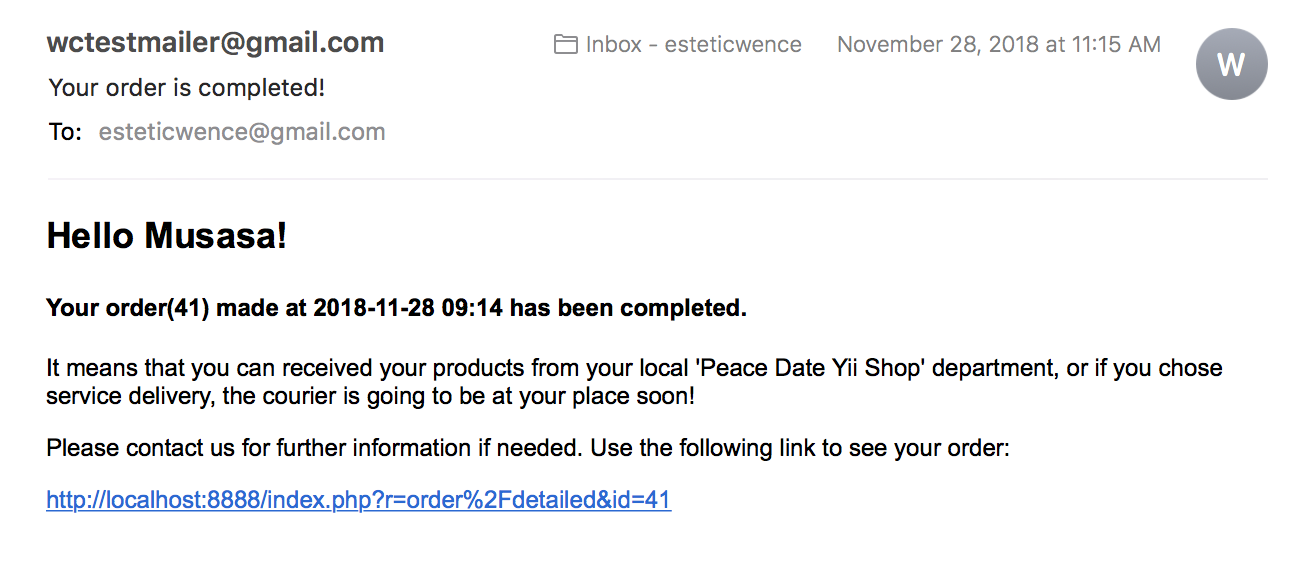


Рисунок 3. Пошта про виконання замовлення

Зручність CRUD

Для зручності CRUD-ів різних об’єктів я розширив GridView і добавив фільтри і сортування по атрибутам із зв’язаних таблиць та розрахунковим атрибутам.

Для прикладу візьмемо середовище продукту, воно вийшло найбільш об’ємним.

Перш за все потрібно в модели прописати всі зв’язки і get методи для створення атрибутів що беруться з зв’язаних таблиць або розраховуються.

Product.php

**public function** getBrandName()  
{  
 **return** $this->brand->name;  
}  
  
**public function** getCategoryName()  
{  
 **return** $this->category->name;  
}

В дійсності атрибутів більше але як приклад підійдуть і ці.

У представленні індекста CRUD-а дописуємо в GridView наші атрибути.

index.php

*/\** ***@var*** *$this yii\web\View \*/  
/\** ***@var*** *$searchModel ProductSearch \*/  
/\** ***@var*** *$dataProvider ActiveDataProvider \*/*$this->title = 'Products';  
$this->params['breadcrumbs'][] = $this->title;  
**?>**<div class="product-index">  
  
 <h1>**<?=** Html::*encode*($this->title) **?>**</h1>  
 **<?php** // echo $this->render('\_search', ['model' => $searchModel]); **?>** <p>  
 **<?=** Html::*a*('Create Product', ['create'], ['class' => 'btn btn-success']) **?>** </p>  
  
 **<?=** GridView::*widget*([  
 'dataProvider' => $dataProvider,  
 'filterModel' => $searchModel,  
 'columns' => [  
 ['class' => 'yii\grid\SerialColumn'],  
  
 'title',  
 'categoryName',  
 'brandName',  
 'price',  
 'amount',  
 'totalRating',  
  
 ['class' => 'yii\grid\ActionColumn'],  
 ],  
 ]); **?>**</div>

По суті Yii2 бере всі методи get/set з моделей і перетворює їх в атрибути, їх можна викликати через метод а можна через власне атрибут, в останньому випадку потрібно не забути прописати @property в документації до моделі

Product.php

*/\*\*  
 \* This is the model class for table "product".  
 \*  
 \** ***@property*** *int $id  
 \** ***@property*** *string $title  
 \** ***@property*** *string $description  
 \** ***@property*** *double $price  
 \** ***@property*** *int $amount  
 \** ***@property*** *int $category\_id  
 \** ***@property*** *int $brand\_id  
 \** ***@property*** *string $created\_at  
 \** ***@property*** *string $updated\_at  
 \*  
 \** ***@property*** *OrderItem[] $orderItems  
 \** ***@property*** *Brand $brand  
 \** ***@property*** *string $brandName  
 \** ***@property*** *Category $category  
 \** ***@property*** *string $categoryName  
 \** ***@property*** *Feature[] $categoryFeatures  
 \** ***@property*** *ProductFeature[] $features  
 \** ***@property*** *Comment[] $comments  
 \** ***@property*** *Rating[] $ratings  
 \*  
 \** ***@property*** *array $feature  
 \*  
 \** ***@property*** *string $image  
 \*/***class** Product **extends** ActiveRecord  
{

Далі потрібно просто розширити модель пошуку яка використовується в GridView у налаштуванні ‘filterModel’.

ProductSearch.php

*/\*\*  
 \* ProductSearch represents the model behind the search form of `app\models\Product`.  
 \** ***@property*** *string $brandName  
 \** ***@property*** *string $categoryName  
 \** ***@property*** *float $fromPrice  
 \** ***@property*** *float $toPrice  
 \*/***class** ProductSearch **extends** Product  
{  
 */\*\*  
 \* {****@inheritdoc****}  
 \*/* **public** $brandName;  
 **public** $categoryName;  
 **public** $fromPrice, $toPrice;  
 **public** $fromRating, $toRating;  
  
 //used on index view  
 **public** $sortBrand, $sortTitle, $sortPrice, $sortRating;  
  
 **const** *SORT\_NONE* = 0,  
 *SORT\_ASC* = 1,  
 *SORT\_DESC* = 2;  
  
 **public function** rules()  
 {  
 **return** [  
 [['id', 'amount', 'category\_id', 'brand\_id'], 'integer'],  
 [['title', 'description', 'created\_at', 'updated\_at'], 'safe'],  
 [['price'], 'number'],  
 [['brandName', 'categoryName'], 'safe'],  
 [['fromPrice', 'toPrice', 'fromRating', 'toRating'], 'number'],  
 [['sortBrand', 'sortTitle', 'sortPrice', 'sortRating'], 'integer'],  
 ];  
 }  
  
 */\*\*  
 \* {****@inheritdoc****}  
 \*/* **public function** scenarios()  
 {  
 // bypass scenarios() implementation in the parent class  
 **return** Model::*scenarios*();  
 }  
  
 */\*\*  
 \** ***@return*** *$this|array  
 \*/  
  
 /\*\*  
 \* Creates data provider instance with search query applied  
 \*  
 \** ***@param*** *array $params  
 \*  
 \** ***@return*** *ActiveDataProvider  
 \*/* **public function** search($params)  
 {  
 $query = Product::*find*();  
  
 // add conditions that should always apply here  
  
 $dataProvider = **new** ActiveDataProvider([  
 'query' => $query,  
 ]);  
  
 $dataProvider->setSort([  
 'attributes' => [  
 'id',  
 'title',  
 'brandName' => [  
 'asc' => ['brand.name' => *SORT\_ASC*],  
 'desc' => ['brand.name' => *SORT\_DESC*],  
 'label' => 'Brand',  
 ],  
 'categoryName' => [  
 'asc' => ['category.name' => *SORT\_ASC*],  
 'desc' => ['category.name' => *SORT\_DESC*],  
 'label' => 'Category',  
 ],  
 'description',  
 'price',  
 'amount',  
 'totalRating' => [  
 'asc' => ['AVG(rating.value)' => *SORT\_ASC*],  
 'desc' => ['AVG(rating.value)' => *SORT\_DESC*],  
 'label' => 'Rating',  
 ],  
 ],  
 ]);  
  
 $this->load($params);  
  
 // explicit sorting for index listView of products  
 **if** ($this->sortBrand != **self**::*SORT\_NONE*)  
 **if** ($this->sortBrand == **self**::*SORT\_ASC*) {  
 $dataProvider->getSort()->setAttributeOrders(['brandName' => *SORT\_ASC*]);  
 } **elseif** ($this->sortBrand == **self**::*SORT\_DESC*) {  
 $dataProvider->getSort()->setAttributeOrders(['brandName' => *SORT\_DESC*]);  
 }  
  
 **if** ($this->sortTitle != **self**::*SORT\_NONE*)  
 **if** ($this->sortTitle == **self**::*SORT\_ASC*) {  
 $dataProvider->getSort()->setAttributeOrders(['title' => *SORT\_ASC*]);  
 } **elseif** ($this->sortTitle == **self**::*SORT\_DESC*) {  
 $dataProvider->getSort()->setAttributeOrders(['title' => *SORT\_DESC*]);  
 }  
  
 **if** ($this->sortPrice != **self**::*SORT\_NONE*)  
 **if** ($this->sortPrice == **self**::*SORT\_ASC*) {  
 $dataProvider->getSort()->setAttributeOrders(['price' => *SORT\_ASC*]);  
 } **elseif** ($this->sortPrice == **self**::*SORT\_DESC*) {  
 $dataProvider->getSort()->setAttributeOrders(['price' => *SORT\_DESC*]);  
 }  
  
 **if** ($this->sortRating != **self**::*SORT\_NONE*)  
 **if** ($this->sortRating == **self**::*SORT\_ASC*) {  
 $dataProvider->getSort()->setAttributeOrders(['totalRating' => *SORT\_ASC*]);  
 } **elseif** ($this->sortRating == **self**::*SORT\_DESC*) {  
 $dataProvider->getSort()->setAttributeOrders(['totalRating' => *SORT\_DESC*]);  
 }  
  
 **if** (!$this->validate()) {  
 // uncomment the following line if you do not want to return any records when validation fails  
 // $query->where('0=1');  
 **return** $dataProvider;  
 }  
  
 // grid filtering conditions  
 $query->andFilterWhere([  
 'id' => $this->id,  
 'price' => $this->price,  
 'amount' => $this->amount,  
 'category\_id' => $this->category\_id,  
 'brand\_id' => $this->brand\_id,  
 ]);  
  
 $query->andFilterWhere(['>=', 'price', $this->fromPrice])  
 ->andFilterWhere(['<=', 'price', $this->toPrice]);  
  
 $query->andFilterWhere(['like', 'title', $this->title])  
 ->andFilterWhere(['like', 'description', $this->description])  
 ->andFilterWhere(['like', 'created\_at', $this->created\_at])  
 ->andFilterWhere(['like', 'updated\_at', $this->updated\_at]);  
  
 //related field for brandName property  
 $query->joinWith('brand');  
 $query->andFilterWhere(['like', 'brand.name', $this->brandName]);  
  
  
 //related field for categoryName property  
 $query->joinWith('category');  
 $query->andFilterWhere(['like', 'category.name', $this->categoryName]);  
  
 //related field for rating property  
 $query->joinWith('ratings');  
 $query->andFilterHaving(['>=', 'AVG(rating.value)', $this->fromRating])  
 ->andFilterHaving(['<=', 'AVG(rating.value)', $this->toRating]);  
  
 $query->groupBy('product.id');  
  
 **return** $dataProvider;  
 }  
  
}

Вище ми просто розписали сортування по атрибутам які нам потрібно і розширили запит фільтрації для додаткових атрибутів.

Аналогічним образом і здійснено функціонал в інших моделях, наприклад в категорії таким образом вираховується кількість унікальних продуктів і сумарна кількість, та здійснюється пошук і сортування за ними.

Знижки

Реалізацію знижок можна пояснити на словах, якщо користувач здійснює певну кількість покупок в певній категорії – йому начисляється знижка на продукти в цій категорії. Для вірності відображення цін у замовлень порівнюються атрибути ‘created\_at’ (дати створення) знижок та замовлення, щоб зрозуміти чи діяла знижка на певне замовлення. Знижки можна адмініструвати через CRUD.

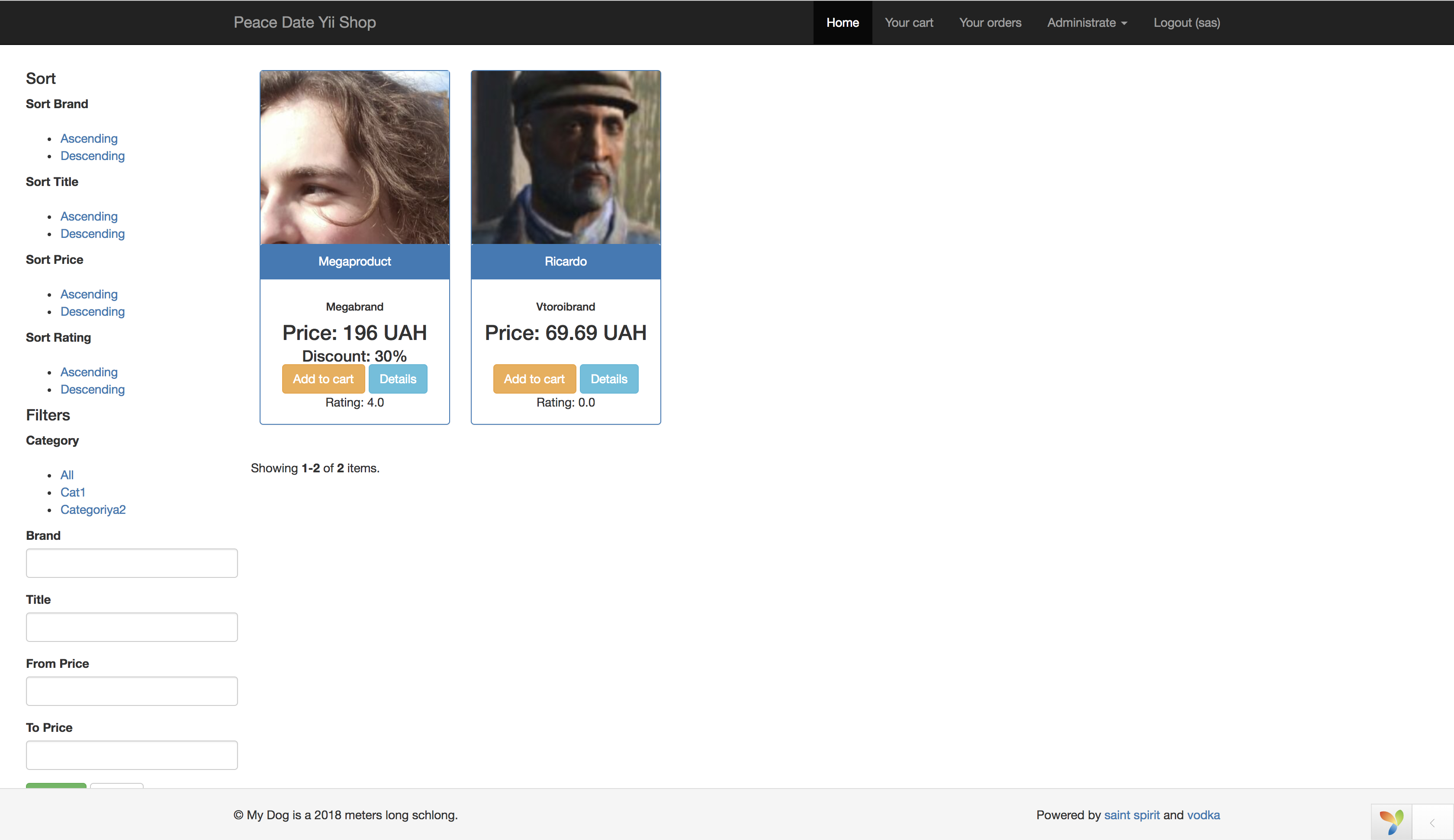
Результат

Рисунок 4. Головна сторінка з продуктами

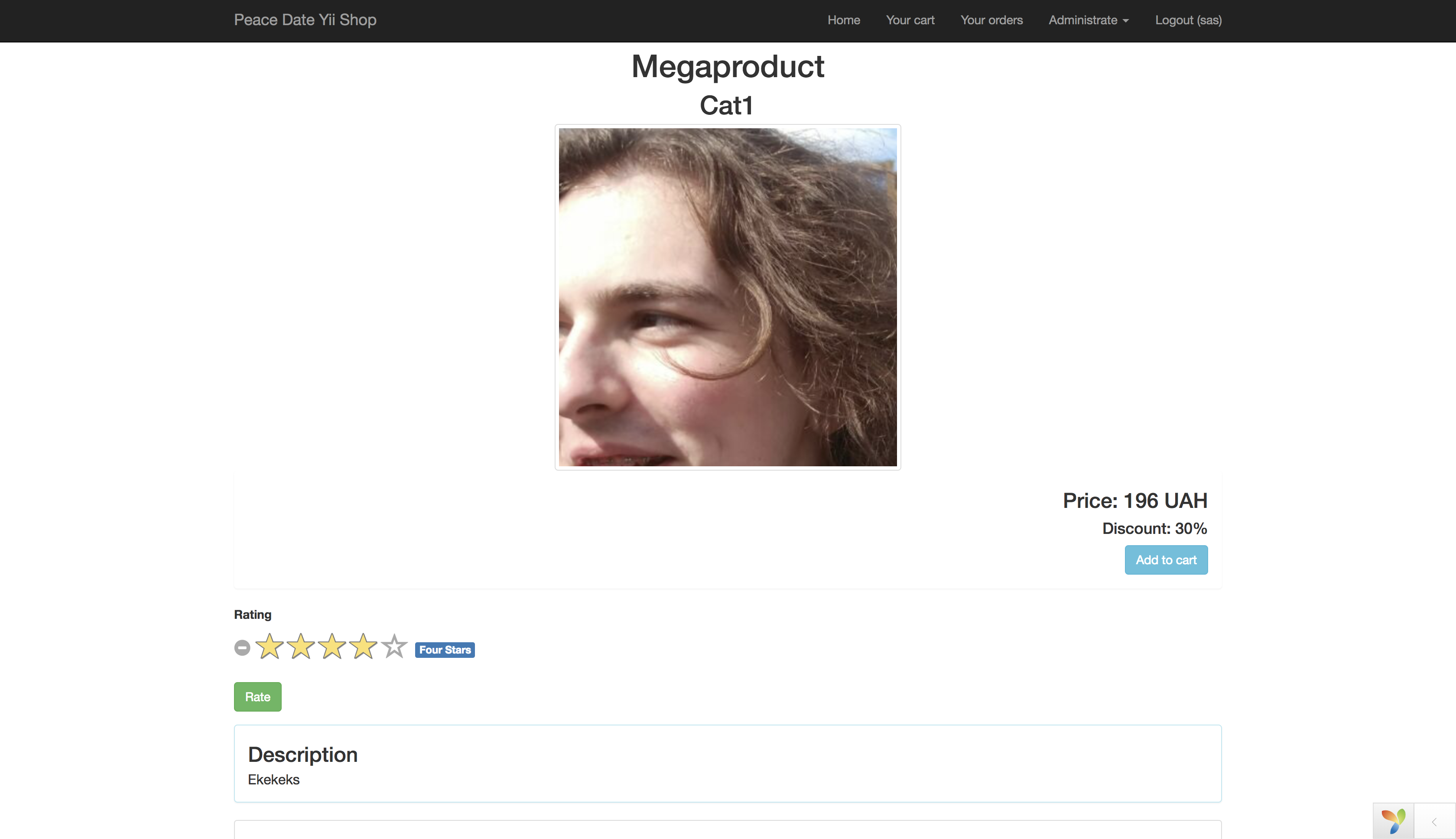


Рисунок 5. Перегляд продукту

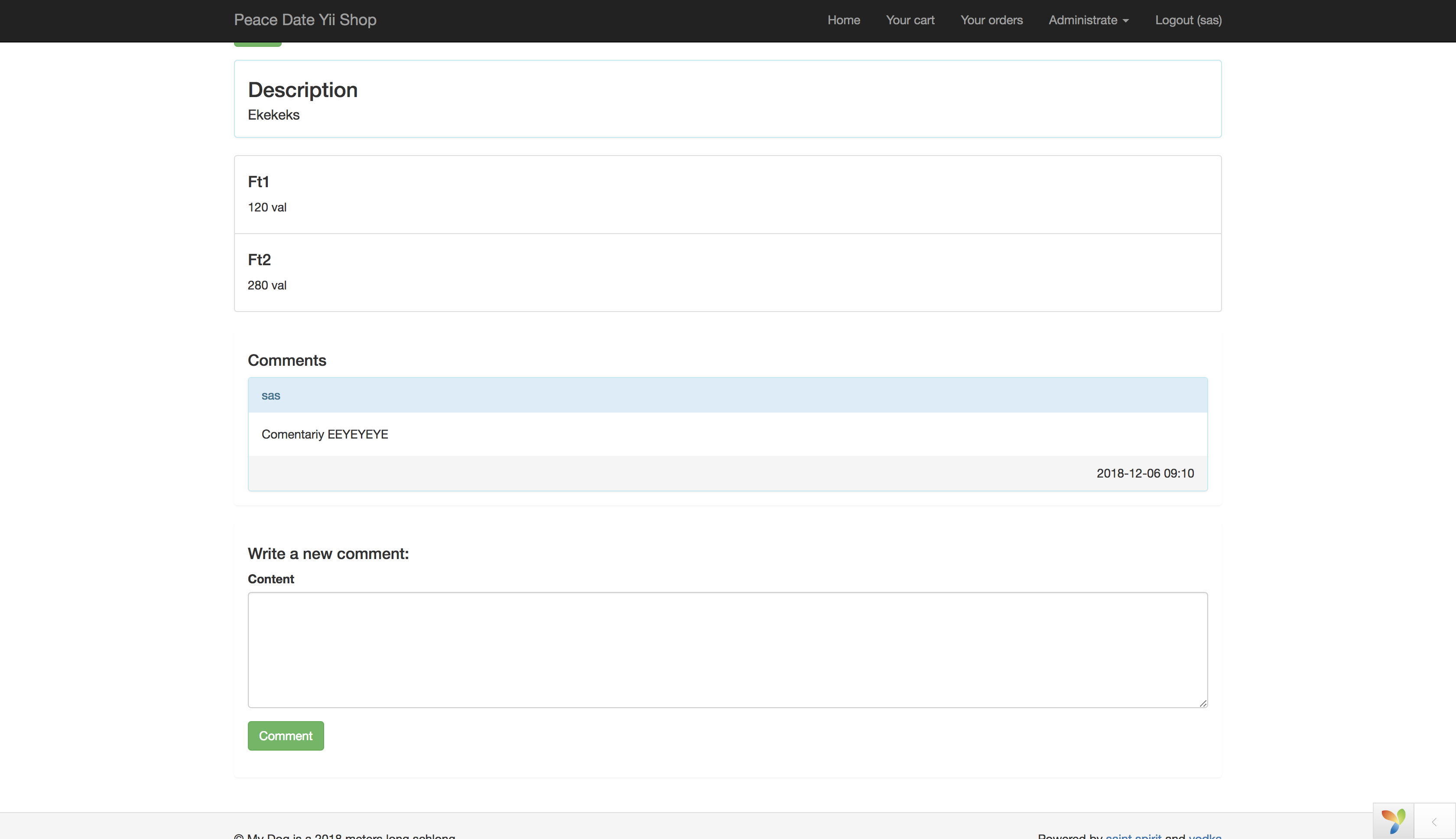


Рисунок 6. Коментарі до продукту

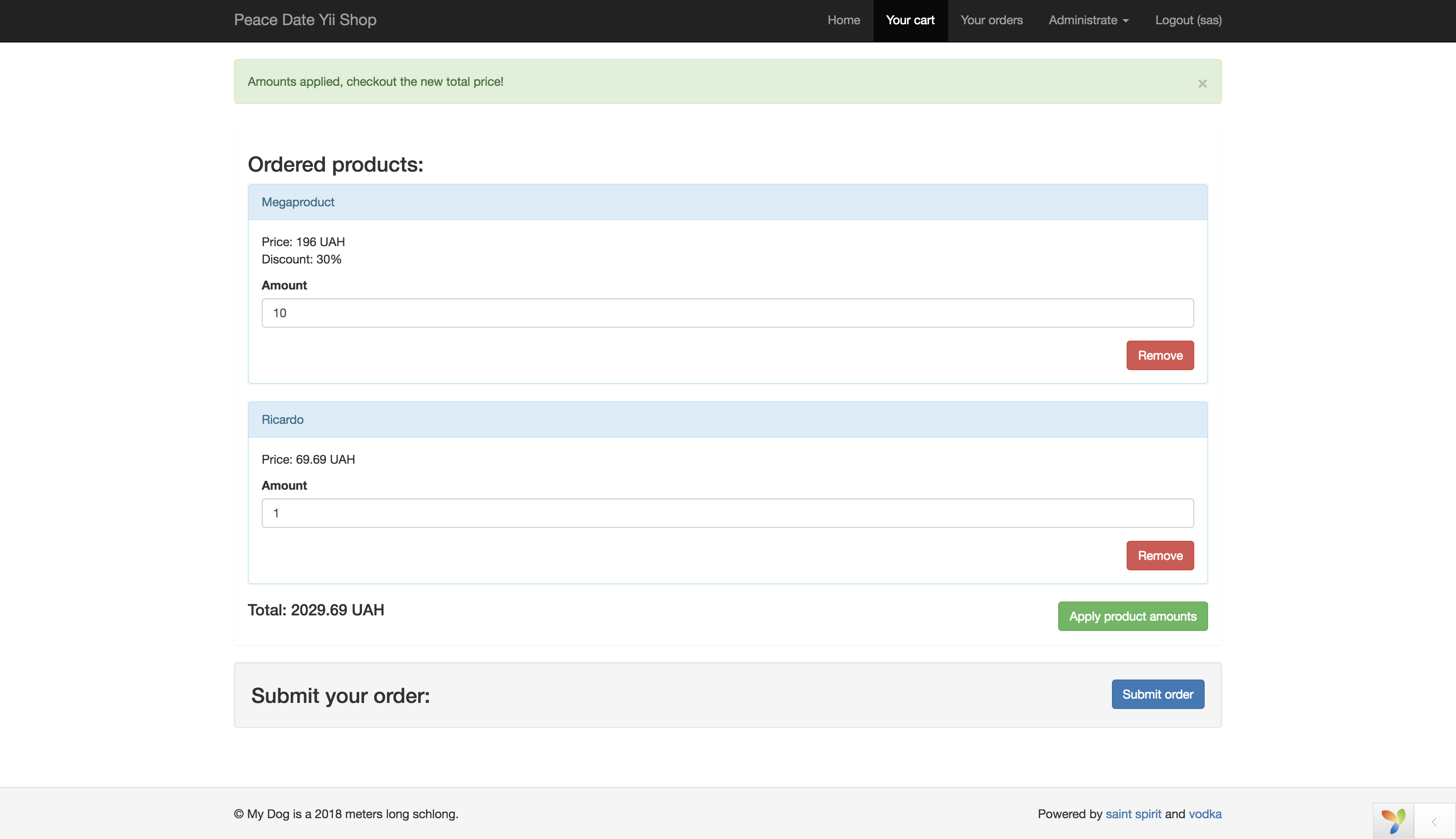


Рисунок 7. Кошик користувача

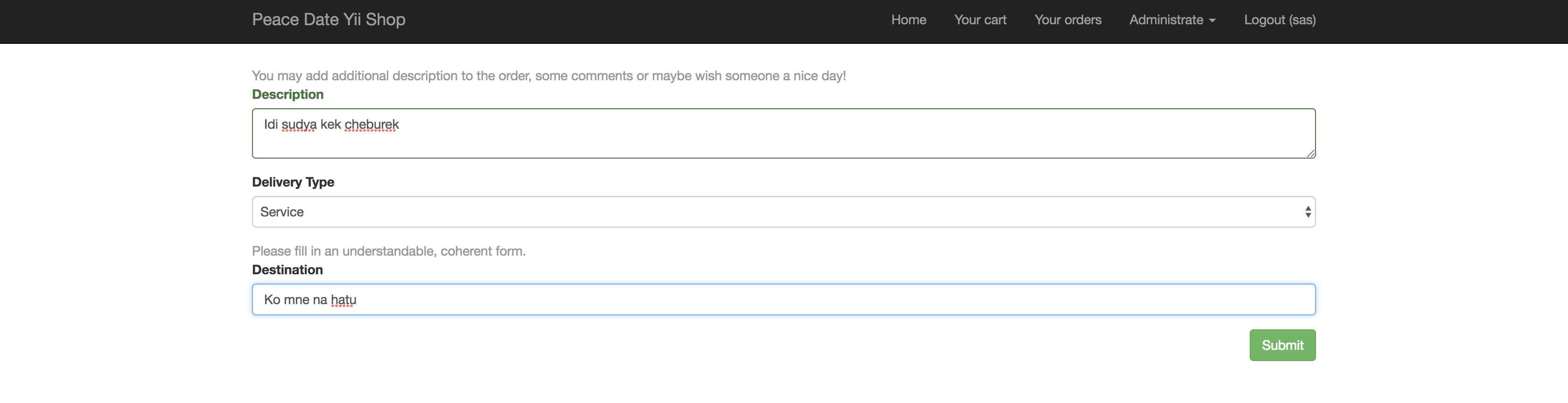


Рисунок 8. Підтвердження замовлення

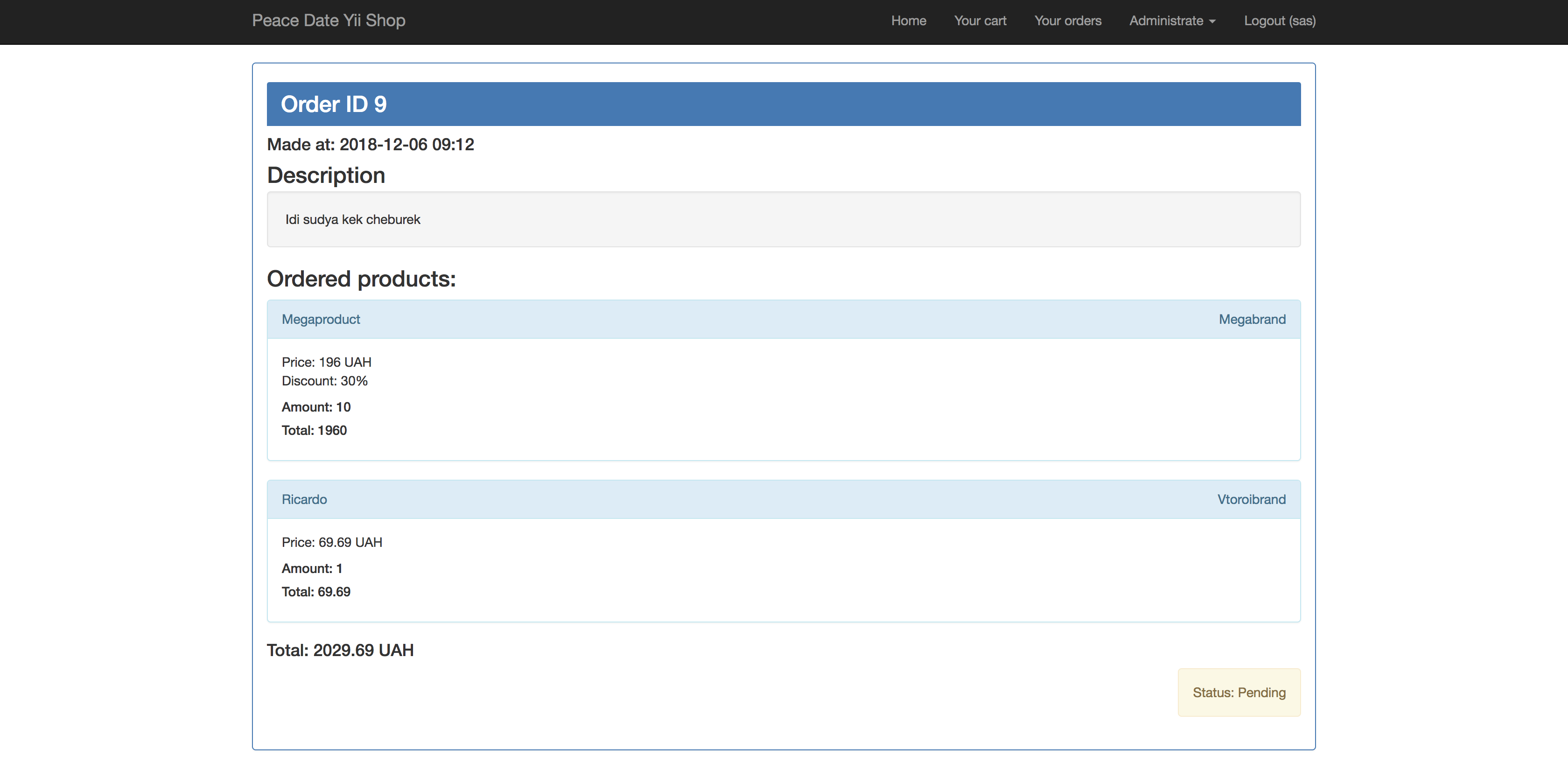


Рисунок 9. Перегляд замовлення



Рисунок 10. CRUD Замовлень

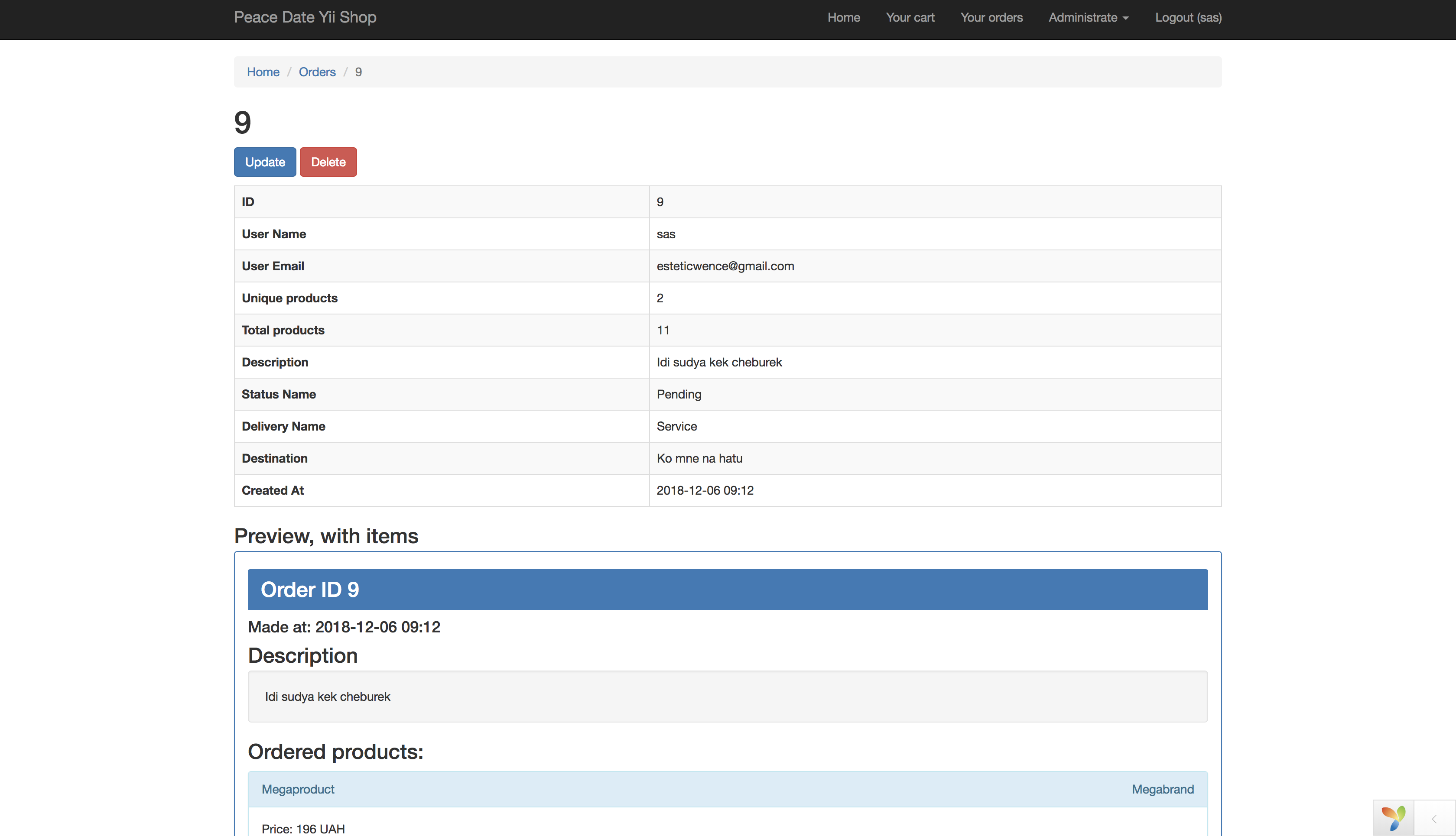


Рисунок 11. Детальний перегляд замовлення(1)

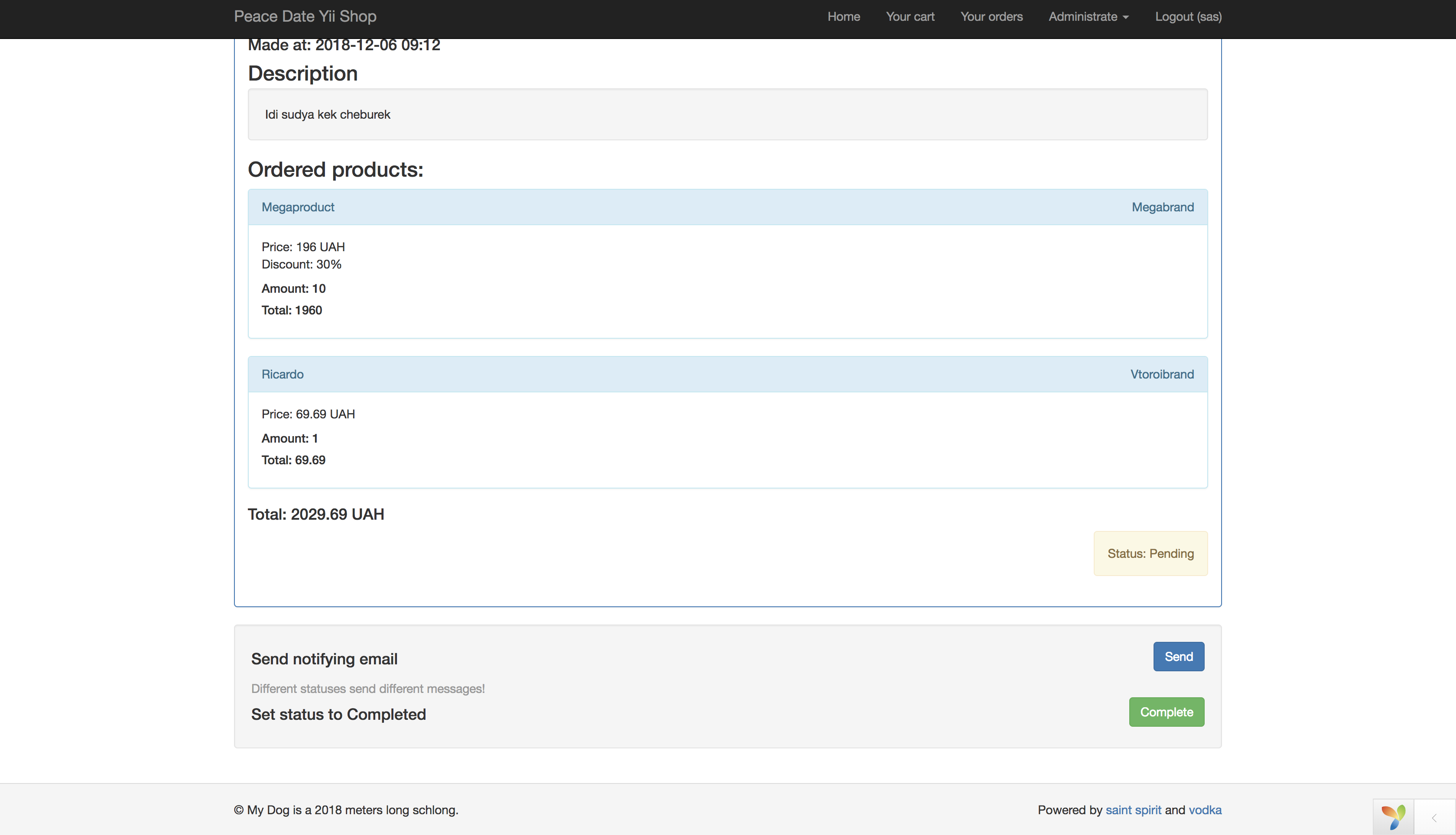


Рисунок 12. Детальний перегляд замовлення (2)

Використані джерела:

<https://github.com/voskobovich/yii2-many-to-many-behavior> | Many-To-Many Behaviour

<https://github.com/kartik-v/yii2-widget-rating> | Rating Widget

Висновок: на даній лабораторній роботі я спробував створити власний веб-магазин на основі Yii2.