{

$this->\_items = $this->storage->load();

}

private function saveItems()

{

$this->storage->save($this->\_items);

}

}

2. It will work only with own items. Instead of built-in storing items to session it will delegate this responsibility to any external storage class, which will implement the Storageinterface interface.

3. The cart class just gets the storage object in its own constructor, saves it instance into private $storage field and calls its load() and save() methods.

4. Define a common cart storage interface with the required methods:

<?php

namespace app\cart\storage;

interface StorageInterface {

/\*\*

\* @return array of cart items \*/

public function load();

/\*\*

\* @param array $items from cart \*/

public function save(array $items);

}

5. Create a simple storage implementation. It will store selected items in a server session:

<?php

namespace app\cart\storage; use yii\web\Session;

class SessionStorage implements StorageInterface {

private $session; private $key;

public function construct(Session $session, $key)

{

$this->key = $key;

$this->session = $session;

}

public function load()

{

return $this->session->get($this->key, []);

}

public function save(array $items)

{

$this->session->set($this->key, $items);

}

}

6. The storage gets any framework session instance in the constructor and uses it later for retrieving