KAPITOLA 6

Nákupní košík

Prvním krokem efektivního webového prodeje produktů je nákupní košík, který je přímo spojený s procesem dokončení objednávky. V této kapitole se dozvíme:

- Jak uspořádat a vytvořit nákupní košík
- Jak spravovat obsah nákupního košíku
- Jak se vypořádat s registrací návštěvníka a převedením jeho košíku na košík spojený s jeho uživatelským účtem

Nákupní košík by měl být relativně jednoduchým elementem. Jeho funkcí je uložit sadu produktů, které má zákazník v úmyslu zakoupit, a spojit je s tímto zákazníkem.

Nákupní košíky

Nákupní košíky představují velmi důležitý aspekt elektronických obchodů a na většině webů jsou prvním krokem v realizaci nákupu online. Existují však i mnohé další metody usnadňující elektronické obchodování, včetně následujících:

- **Platby jedním klepnutím** – příkladem tohoto typu plateb je například platební tlačítko služby PayPal nebo objednávání jedním klepnutím serveru Amazon. Platby jedním klepnutím, jaké poskytuje například uvedené platební tlačítko služby PayPal, přebírají veškerá data od zpracovatele platby a obecně se používají na webových stránkách

s malým výběrem produktů. Zákazník klepne na tlačítko na stránce produktu, jeho platba se zpracuje a zpracovatel platby předá administrátorovi informace o produktu, zákazníkovi, dodacích detailech a zaplacené částce – data o produktu a jeho ceně definuje platební tlačítko. Objednávky jedním klepnutím usnadňují zákazníkovi život a odstraňují nutnost projít celým procesem objednávky. Mají však také své nevýhody – zákazníci mohou snadno něco objednat omylem, musí si být předem jistí správností svých výchozích dodacích informací, není snadné přidávat kódy slevových kuponů bez dalších klepnutí a není možné brát do úvahy slevy za více objednávek nebo na hromadnou dodávku.

- **Platby za členství** – zpravidla se podobají předchozímu typu plateb – vyberete typ členství a poté zaplatíte. Většina předplacených služeb umožňuje snadno měnit možnosti odebírané služby a uživateli se tak účtuje částka odpovídající době, po kterou byl jeho účet nastavený na daný typ členství. Předplacené weby umožňují z obchodního úhlu pohledu přístup k produktům a službám po předplacenou dobu. To často znamená opakující se příjem a nižší poplatky za transakce. Například z webu poskytujícího ke stažení hudbu si uživatel může stáhnout 25 písniček měsíčně, kde každý z těchto nákupů je bez poplatků. V případě plateb za členství se transakční poplatky budou vztahovat pouze na méně frekventované, větší platby a celkově tak budou nižší.
- **Aukce** – aukce jsou založené na přihazování. To znamená, že se zákazník zaváže ke koupi produktu za určitou cenu, za předpokladu, že žádný jiný zákazník v době aukce nepřistoupí na vyšší cenu. Aukční weby jsou často automatizované, takže zákazník zadá maximální cenu a v průběhu aukce bude systém zvyšovat nabídku zákazníka s ohledem na jím stanovenou maximální hodnotu podle toho, jak budou přicházet ostatní nabídky.

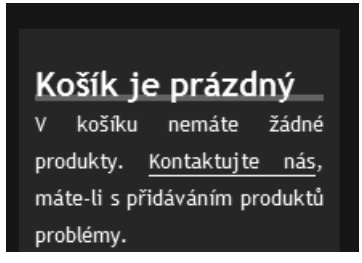
Pro náš framework bude tím nejvhodnějším řešením nákupní košík. Přece jenom se jedná o framework. Nic nám však nebrání v implementaci jiných metod realizace plateb.

Náš košík

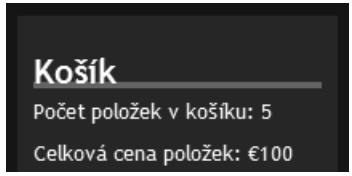
Na konci této kapitoly budeme mít stránku nákupního košíku a menší nákupní košík zobrazený na každé stránce.

Košík pro jednotlivé stránky

Pokud v košíku zákazníka nic není, zobrazí se mu na každé stránce textová hláška o prázdném košíku, jako je ta následující:



Pokud má zákazník ve svém košíku nějaké produkty, krátká zpráva shrnující obsah nákupního košíku zobrazovaná na každé stránce může vypadat takto:



A hlavní stránka košíku by měla vypadat takto:

Úvahy týkající se našeho nákupního košíku

V kapitole 4, *Varianty produktů a nahrávání uživatelských souborů na server*, jsme se zamysleli nad řadou věcí týkajících se nákupního košíku. Pojďme si je zrekapitulovat a podívat se na to, jak tyto myšlenky implementovat.

- **Skladové zásoby** – v některých případech potřebujeme určit, jestli je na skladě dostatečný počet kusů daného produktu, když ho uživatel přidá do svého nákupního košíku. Při přidávání produktu do nákupního košíku se můžeme jednoduše dotázat databáze produktů a ujistit se, že množství produktů skladem je větší, než kolik si zákazník přeje koupit, nebo není množství pro daný produkt relevantní (např. produkty digitálního charakteru, služby atd.).
- **Varianty produktů** – když zákazník přidá variantu produktu do svého nákupního košíku, musíme zaznamenat, o jakou variantu se jedná. Způsob, jakým jsou produkty uložené v košíku, musí rozlišovat produkty a varianty produktů, tak aby mohl uživatel zakoupit libovolný počet různých variant téhož produktu.

- **Úpravy produktů** – pokud zákazník upraví produkt, musíme zaznamenat veškerá jeho data, podobně jako v případě variant produktů.
- **Šablony** – potřebujeme několik různých šablon pro zobrazení košíku – prázdný košík, souhrn obsahu košíku zobrazovaný na každé stránce atd.
- **Mezisoučty** – potřebujeme vypočítat mezisoučty jednotlivých produktů v košíku.

Vytvoření košíku

Nákupní košík vytvoříme v několika krocích. Začneme u základní funkčnosti, kterou budeme při implementaci výše uvedených úvah postupně rozšiřovat.

Kdy vytvořit košík uživatele

Náš nákupní košík bude pravděpodobně zobrazený na většině stránek, a proto musíme zajistit sestavení obsahu nákupního košíku bez ohledu na to, kde přesně se uživatel v rámci webu, poháněného naším frameworkem, nachází. Přirozeně nemusíme tyto informace potřebovat pokaždé, ale ve většině případů tomu tak bude. Další úvahy směřují na adresu autentizace uživatelů – pokud je návštěvník přihlášený, bude se nákupní košík sestavovat jiným způsobem, a proto musíme zajistit, že se nákupní košík vytvoří až po provedení autentizace.

Databázová struktura

V kapitole 5, *Větší uživatelská přívětivost*, jsme se zabývali tvorbou seznamu přání. Ten však byl vhodný pouze pro standardní produkty, které nemohli zákazníci upravovat a které ani neměly žádné varianty. Nákupní košík by měl být v tomto ohledu dokonalejší. Je nezbytná jedna databázová tabulka pro svázání zákazníků a produktů. Vyjděme z naší tabulky seznamů přání a rozšiřme ji tak, aby byla vhodná pro nákupní košík.

Sloupec	Typ	Popis
`ID`	`Integer (Auto increment, Primary Key)`	
`Session_id`	`Varchar`	Pro vazbu na nepřihlášené zákazníky webu
`User_id`	`Integer`	Pro vazbu na přihlášené zákazníky webu
`Product_id`	`Integer`	Identifikátor produktu
`Quantity`	`Integer`	Množství, v jakém si přeje zákazník daný produkt zakoupit
`IPAddress`	`Varchar`	Tento sloupec je zapotřebí pro lepší vazbu produktů v košíku na nepřihlášené uživatele

Sloupec	Typ	Popis
`Timestamp`	`Timestamp`	Slouží pro rozlišení aktivního a už neplatného obsahu. Slouží tedy primárně pro nepřihlášené zákazníky a po uplynutí určitého času musíme začít předpokládat, že daný identifikátor relace a adresa IP už patří jinému zákazníkovi.

Tuto tabulku reprezentuje následující kód SQL:

```
CREATE TABLE `basket_contents` (
  `ID` INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `session_id` VARCHAR( 50 ) NOT NULL,
  `user_id` INT NOT NULL,
  `product_id` INT NOT NULL,
  `quantity` INT NOT NULL,
  `ip_address` VARCHAR( 50 ) NOT NULL,
  `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE = MYISAM;
```

Obsah košíku

Databázová struktura pro nákupní košík je připravená, nyní musíme zákazníkům umožnit s košíkem a jeho obsahem pracovat. To znamená především:

- Zobrazit obsah košíku
- Přidávat produkty do košíku
- Přidávat upravitelné produkty do košíku
- Přidávat varianty produktů do košíku
- Upravovat množství produktů v košíku

Pojďme tyto požadavky projít a implementovat je do našeho frameworku.

Zobrazení košíku

Může se zdát trochu zvláštní zabývat se zobrazením obsahu košíku, když naši zákazníci ještě ani nemají možnost přidávat produkty do svého košíku. Důvodem je, že pro přidání produktu do košíku potřebujeme metodu, která zkontroluje obsah košíku. Pokud zákazník přidá stejný produkt do svého nákupního košíku podruhé, musí se navýšit množství u již existující instance produktu v košíku, což znamená, že musíme být schopni rozhodnout, jestli daný produkt v košíku už je.

Kontrola obsahu nákupního košíku probíhá v následujících krocích:

1. Výběr relevantních dat z tabulky databáze pro nákupní košík.
2. Sestavení pole dat reprezentujících obsah košíku.

3. Nastavení několika proměnných, včetně celkové ceny obsahu košíku, příznaku prázdnosti košíku a příznaku provedené kontroly obsahu košíku.

Metoda checkBasket

Potřebujeme metodu, která zajistí kontrolu obsahu košíku, na jejímž základě budeme moci zákazníkovi zobrazit košík.

```
/**
 * Zkontroluje obsah nákupního košíku uživatele
 * @return void
 */
```

Prvním krokem je nastavení hodnot několika proměnných, které budou později zapotřebí. Evidujeme příznak signalizující provedenou kontrolu obsahu košíku, což zamezuje tomu, aby framework zbytečně volal tuto metodu znovu poté, co se už zavolala a data se zpracovala.

Potřebujeme také identifikátor relace uživatele a jeho adresu IP, především v případě nepřihlášených zákazníků. V případě přihlášeného uživatele potřebujeme znát jeho identifikátor.

```
public function checkBasket()
{
  // nastavení hodnoty proměnné signalizující, že kontrola
  // obsahu košíku už proběhla - zabraňuje zbytečnému opakovanému
  // volání metody
  // pokud tuto metodu zavoláme při načítání stránky, když
  // sestavujeme krátký souhrn obsahu košíku, nemusíme ji volat
  // znovu pro zobrazení hlavního košíku
  $this->basketChecked = true;

  // určení dat identifikujících uživatele
  $session_id = session_id();
  $ip_address = $_SERVER ['REMOTE_ADDR'];

  // pokud je zákazník přihlášený, bude se dotaz na databázi lišit
  if( $registry->getObject('authentication')->
    isLoggedIn() == true )
  {
    // zákazník je přihlášený, určíme jeho identifikátor
    $u = $this->registry->getObject('authentication')->getUserID();
```

Metoda `checkBasket` provede jednu ze dvou verzí dotazu pro vyhledávání produktů v košíku uživatele – jednu, pokud je zákazník přihlášený, která vyhledává produkty na základě identifikátoru zákazníka, a druhou, pokud je zákazník nepřihlášený, která používá adresu IP a identifikátor relace pro určení produktů náležících do nákupního košíku aktuálního zákazníka.

```
  $sql = "SELECT b.ID AS basket_id,
      b.quantity AS product_quantity,
```

```
      c.ID AS product_id,
      v.name AS product_name,
      p.stock AS product_stock,
      p.weight AS product_weight,
      p.price AS product_price,
      p.SKU AS product_sku
    FROM content_versions v, content c, content_types t,
         content_types_products p, basket_contents b
    WHERE c.active=1 AND c.secure=0 AND c.type=t.ID
      AND t.reference='product' AND p.content_version=v.ID
      AND v.ID=c.current_revision AND c.ID=b.product_id
      AND b.user_id={$u}";
}
else
{
  $sql = "SELECT b.ID AS basket_id,
      b.quantity AS product_quantity,
      c.ID AS
      product_id,
      v.name AS product_name,
      p.stock AS product_stock,
      p.weight AS product_weight,
      p.price AS product_price,
      p.SKU AS product_sku
  FROM content_versions v, content c, content_types t,
       content_types_products p, basket_contents b
  WHERE c.active=1 AND c.secure=0 AND c.type=t.ID
    AND t.reference='product' AND p.content_version=v.ID
    AND v.ID=c.current_revision AND c.ID=b.product_id
    AND b.user_id=0 AND b.session_id='{$session_id}'
    AND b.ip_address='{$ip_address}'";
}
// provedení dotazu
$this->registry->getObject('db')->executeQuery( $sql );
```

Pokud je množina výsledků dotazu neprázdná, není prázdný ani nákupní košík, a proto sestavíme jeho obsah průchodem přes jednotlivé výsledky. Toto chování budeme později muset, s ohledem na upravitelné produkty, pozměnit.

```
  if( $this->registry->getObject('db')->numRows() > 0 )
  {
    // v košíku jsou nějaké produkty,
    // nastaví se proměnná signalizující neprázdný košík
    $this->basketEmpty = false;
    while( $contents = $this->registry->getObject('db')->getRows() )
    {
```

```
      // každý z produktů přidáme do pole nákupního košíku
      $this->contents[ 'standard-' . $contents['product_id'] ] =
        array( 'unitcost' => $contents['product_price'],
          'subtotal' => ($contents['product_price']
                       * $contents['product_quantity']),
          'weight' => $contents['product_weight'],
          'quantity' => $contents['product_quantity'],
          'product' => $contents['product_id'],
          'basket' => $contents['basket_id'],
          'name' => $contents['product_name'] );
      $this->numProducts = $this->numItems + $contents['quantity'];
      $this->cost = $this->cost + ( $contents['price']
                                  * $contents['quantity'] );
    }
  }
}
```

Řadič

V současném stavu musí náš řadič být schopen:

- Detekovat, jestli se zákazník snaží zobrazit košík
- Získat z modelu obsah košíku
- Uložit obsah košíku do mezipaměti a spojit ho se značkami šablony

Výše uvedenou funkčnost zajišťuje následující metoda:

```
public function viewBasket()
{
  // sestavení pohledu s použitím patřičné šablony
  $this->registry->getObject('template')->
    buildFromTemplates('header.tpl.php', 'viewbasket.tpl.php',
      'footer.tpl.php');
```

Následně se z modelu košíku získá jeho obsah a sestaví se pole vhodné pro uložení do mezipaměti a zpracování v šablonovém systému.

```
  $contents = $this->basket->getContents();
  $products = array();
  foreach( $contents as $reference => $data )
  {
    $data['basket_id'] = $data['basket'];
    $data['basket'] = '';
    $products[] = $data;
  }
  // Předání obsahu košíku šablonovému systému a přiřazení
  // dalších údajů, jako je celková cena a počet produktů
```

```
  // do proměnných šablony
  $basketCache = $this->registry->getObject('db')->
    cacheData( $products );
  $this->registry->getObject('template')->getPage()->
    addTag( 'products', array( 'DATA', $basketCache ) );
  $this->registry->getObject('template')->getPage()->
    addTag( 'basket_total', $this->basket->getTotal() );
  $this->registry->getObject('template')->getPage()->
    addPPTag('shippingCost', $this->basket->getShippingCost());
}
```

Přidávání produktů

Přidání standardního produktu do košíku by mělo být jednoduchou záležitosti. Stačí pouze zaznamenat identifikátor produktu a informace o uživateli (jako je jeho identifikátor nebo, v případě, že není uživatel přihlášený, identifikátor jeho relace a adresa IP).

Vzhledem k tomu, že budeme pro práci s obsahem košíku používat pole, jehož klíče jsou identifikátory produktů, a další informace, jako je cena, název, množství a další, jsou jeho prvky, měli bychom tyto identifikátory opatřit prefixem, abychom se vyhnuli pozdějším problémům při přidávání variant produktů a upravitelných produktů. Pokud může být k produktu například připojený obrázek a do košíku přidáme dva tyto produkty, každý s jiným obrázkem, budeme chtít, aby je košík bral jako jeden produkt s množstvím dva a nikoli jako dva samostatné produkty. Naopak v případě produktu, který není možné upravovat, je právě tohle žádoucí chování.

Začneme s funkčností pro přidání standardního produktu do nákupního košíku.

Metoda addProduct

Když uživatel stiskne tlačítko **Přidat do košíku**, co je třeba udělat?

1. Ze všeho nejdřív musíme zkontrolovat, že je produkt platný a aktivní. Nemůžeme přece do nákupního košíku přidat produkt, který neexistuje.
2. Zadruhé, musíme zkontrolovat, jestli se jedná o upravitelný produkt, nebo o produkt s variantami (protože tato metoda bude sloužit pouze pro standardní produkty).
3. Na závěr ověříme, jestli už produkt v košíku je. Pokud ano, zvýšíme množství produktu. V opačném případě ho do košíku přidáme.

Následující kód reprezentuje odpovídající implementaci metody `addProduct`. Přestože je množství parametrem metody, ve většině případů se použije výchozí hodnota jedna, jako například když zákazník stiskne tlačítko **Přidat do košíku**, bude chtít do košíku přidat pouze jeden produkt.

```
/**
 * Přidá produkt do košíku
 * @param String productPath odkaz na produkt
 * @param int $quantity množství produktu
 * @return String zpráva pro řadič
```

```
 */
public function addProduct( $productPath, $quantity=1 )
{
  // už jsme volali metodu checkBasket?
  if( ! $this->basketChecked == true ) { $this->checkBasket(); }

  // kontrola, jestli produkt existuje
  $productQuery = "SELECT v.name AS product_name,
      c.ID AS product_id,
      p.allow_upload AS allow_upload,
      p.stock AS product_stock,
      p.weight AS product_weight,
      p.price AS product_price,
      p.SKU AS product_sku,
      p.featured AS product_featured,
      v.heading AS product_heading,
      v.content AS product_description,
      v.metakeywords AS metakeywords,
      v.metarobots AS metarobots,
      v.metadescription AS metadescription
    FROM content_versions v, content c,
         content_types t, content_types_products p
    WHERE c.active=1 AND c.secure=0 AND c.type=t.ID
      AND t.reference='product' AND p.content_version=v.ID
      AND v.ID=c.current_revision AND c.path='{$productPath}'";
  $this->registry->getObject('db')->executeQuery( $productQuery );
  if( $this->registry->getObject('db')->numRows() == 1 )
  {
    // určení identifikátoru atd.
    $data = $this->registry->getObject('db')->getRows();

    // kontrola, jestli už je v košíku
    if( array_key_exists( 'standard-' . $data['product_id'],
        $this->contents ) == true )
    {
      // kontrola zásob
      if( $data['product_stock'] == -1 ||
          $data['product_stock'] >= ( $this->contents['standard-' .
          $data['product_id']]['quantity'] + $quantity ) )
      {
        // zvýšení množství
        $this->contents['standard-'
          . $data['product_id']]['quantity'] =
         $this->contents['standard-'
          . $data['product_id']]['quantity'] + $quantity;
```

```
      // aktualizace databáze
      $this->registry->getObject('db')->
        updateRecords('basket_contents', array('quantity'=>
        $this->contents[ 'standard-' . $product]['quantity'] ),
        'ID = ' . $this->contents['standard-'.$product]['basket'] );
      return 'success';
    }
    else
    {
      // chybová zpráva
      return 'stock';
    }
  }
  else
  {
    if( $data['product_stock'] == -1 ||
      $data['product_stock'] >= $quantity )
    {
      // přidání produktu
      // vložení nového záznamu do košíku
      $s = session_id();
      $u = ( $this->registry->getObject('authentication')->
        isLoggedIn() ) ? $this->registry->
        getObject('authentication')->getUserID() : 0;
      $ip = $_SERVER['REMOTE_ADDR'];
      $item = array( 'session_id' => $s, 'user_id' => $u,
        'product_id' => $data['product_id'],
        'quantity' => $quantity, 'ip_address' => $ip );
      $this->registry->getObject('db')->
        insertRecords( 'basket_contents', $item );
      $bid = $this->registry->getObject('db')->lastInsertID();

      // přidání produktu do pole s obsahem košíku
      $this->contents['standard-' . $data['product_id'] ] =
        array( 'unitcost' => $data['product_price'],
          'subtotal' => ($data['product_price'] * $quantity ),
          'weight' => $data['product_weight'],
          'quantity' => $quantity,
          'product' => $data['product_id'],
          'basket' => $bid,
          'name' => $data['product_name'] );

      // vrátí informaci o úspěšném provedení operace
      return 'success';
```

```
      }
      else
      {
        // chybová zpráva
        return 'stock';
      }
    }
  }
  else
  {
    // produkt neexistuje, chybová zpráva
    return 'noproduct';
  }
}
```

Řadič

Kromě dříve uvedených funkcí musí být náš řadič v tomto stádiu schopen:

- Detekovat, jestli se zákazník snaží přidat produkt do košíku, tj. jestli stisknul tlačítko **Přidat do košíku**, a následně na to adekvátním způsobem reagovat. Bez toho by se produkty nikdy nedostaly do nákupního košíku uživatele.
- Předat data modelu, tak aby se produkt přidal do košíku. Musíme modelu košíku sdělit, který produkt si přeje zákazník do svého košíku přidat, aby tak věděl, se kterým produktem má asociovat aktuálního zákazníka.
- Zobrazit chybovou hlášku, pokud zadaný produkt neexistuje, tak aby zákazník věděl, že produkt se do jeho košíku nepřidal, a nebyl tudíž zmatený z jeho nepřítomnosti v košíku.
- Zobrazit chybovou hlášku, pokud je produkt vyprodaný, tak aby zákazník věděl, že produkt se do jeho košíku nepřidal, a nebyl tudíž zmatený z jeho nepřítomnosti v košíku.
- Pokud se jedná o platný produkt, který je skladem, musí se zobrazit zákazníkovi potvrzení o jeho přidání do košíku. Vhodné je také po úspěšném přidání produktu do košíku zákazníka přesměrovat na jinou stránku, jako je například stránka nákupního košíku nebo stránka produktů.

Přestože je toho zapotřebí udělat poměrně hodně, odpovídající kód je překvapivě jednoduchý. V první řadě musíme rozšířit **blok** `switch` **v konstruktoru řadiče tak, aby detekoval,** jestli zákazník přidává produkt do košíku.

```
case 'add-product':
  echo $this->addProduct( $urlBits[2], 1);
  break;
```

Metoda `addProduct` se tak zavolá, když se zákazník snaží přidat produkt do svého košíku. Tato metoda se pokusí přidat produkt do modelu košíku a v závislosti na výsledku zobrazí zákazníkovi patřičnou zprávu.

```
/**
 * Přidá produkt do košíku
 * @param String productPath odkaz na produkt
 * @param int $quantity množství produktu
 * @return String zpráva pro řadič
 */
```

Následně ověříme, jestli už proběhla kontrola košíku, aby se tak zabránilo jejímu případnému opakování. Obsah košíku musíme zkontrolovat proto, abychom určili, jestli zvyšujeme množství určitého produktu v košíku anebo přidáváme do košíku nový produkt.

```
public function addProduct( $productPath, $quantity=1 )
{
  // už jsme volali metodu checkBasket?
  if( ! $this->basket->isChecked == true )
    { $this->basket->checkBasket(); }

  // následně zavoláme metodu addProduct modelu košíku
  // a poznamenáme její návratovou hodnotu
  $response = $this->basket->addProduct( $productPath, $quantity );
```

V závislosti na návratové hodnotě se zákazníkovi zobrazí jedna z hlášek.

```
    if( $response == 'success' )
    {
      $this->registry->redirectUser('products/view/'
        . $productPath, 'Produkt přidán',
        'Produkt byl přidán do vašeho košíku', false );
    }
    elseif( $response == 'stock' )
    {
      $this->registry->getObject('template')->
        buildFromTemplates('header.tpl.php', 'message.tpl.php',
          'footer.tpl.php');
      $this->registry->getObject('template')->getPage()->
        addTag('header', 'Vyprodáno' );
      $this->registry->getObject('template')->getPage()->
        addTag('message', 'Je nám líto, ale tento produkt je
          vyprodaný a není možné ho přidat do vašeho košíku.' );
    }
    elseif( $response == 'noproduct' )
    {
      $this->registry->getObject('template')->
        buildFromTemplates('header.tpl.php', 'message.tpl.php',
        'footer.tpl.php');
      $this->registry->getObject('template')->getPage()->
        addTag('header', 'Produkt nenalezen' );
```

```
      $this->registry->getObject('template')->getPage()->
        addTag('message', 'Je nám líto, ale takový produkt neexistuje.' );
    }
  }
```

Etická poznámka

Je důležité, aby v nákupním košíku uživatele skončily pouze ty produkty, které sám do košíku přidal. Z vlastní zkušenosti znám několik webů, které se snaží automaticky označit další produkty, jež se následně přidají do košíku. Zákazník to může snadno přehlédnout a nechtěně si objednat produkty, které nechce. Výsledkem bude nespokojený zákazník, negativní ohlasy na webu a pošramocená pověst takovéhoto obchodu.

Přidávání upravitelných produktů

Dříve v knize jsme probírali upravitelné produkty dovolující zákazníkům připojovat k objednávce produktu nejenom soubory, ale také vlastní texty zadané do několika textových polí.

Abychom umožnili nákup takovýchto produktů, musíme provést několik věcí:

- Pozměnit strukturu tabulky databáze pro košík
- Změnit způsob, jakým zobrazujeme nákupní košík
- Změnit způsob, jakým přidáváme produkty do košíku
- Zpracovat informace o úpravě provedené zákazníkem, jednalo-li se o upravitelný produkt

Úprava databázové struktury košíku

V tabulce produktů máme dva zajímavé sloupce, `allow_upload` a `custom_text_inputs`. Do tabulky košíku bude zapotřebí přidat sloupec pro uložení informací o nahraném souboru a sloupec pro texty vložené uživatelem.

```
ALTER TABLE `basket_contents`
  ADD `uploaded_file` VARCHAR( 255 ) NOT NULL,
  ADD `custom_text_values` LONGTEXT NOT NULL
```

Zobrazení košíku

Při zobrazování košíku *může* být žádoucí zobrazit odlišné informace v závislosti na tom, jestli se jedná o upravitelný produkt. To znamená upravit metodu `checkBasket` modelu a také provést určité změny v řadiči.

Změny modelu mají za úkol detekovat, jestli se k záznamu v košíku váže soubor nebo nějaké uživatelské texty. Pokud ano, přidá se produkt do pole reprezentujícího košík nikoli s prefixem `standard`, nýbrž s prefixem `customized`. Smyslem jiného prefixu je zajistit, aby se po opětovném přidání téhož produktu do košíku nevytvořil v košíku duplikát tohoto produktu a místo toho se zohlednily úpravy, které zákazník provedl.

Úprava modelu

Model vyžaduje následující změny:

- Kontrola, jestli se jedná o upravitelný produkt. Pokud ano, musíme požadavek zpracovat odlišným způsobem.
- Kontrola, jestli byl připojen soubor. Jedná-li se o upravitelný produkt a zákazníci mohou k jeho objednávce připojit soubor (jako je obrázek nebo fotografie), musí model ověřit, jestli zákazník nahrál soubor, a případně ho zpracovat.
- Pokud se zákazník rozhodl nahrát na server soubor, musí se po nahrání přesunout do vhodného umístění, jako například do adresáře `basket_uploads`. Umístění tohoto souboru je nutné poznamenat v tabulce `basket_contents`.
- V závislosti na povaze obchodu a typu souborů, které dovolujeme zákazníkům nahrávat na server, může být vhodné evidovat, jestli se jedná o obrázek nebo ne. Jedná-li se o obrázek, můžeme zobrazit náhled tohoto obrázku ve stránce s výpisem obsahu košíku, tak aby zákazník viděl společně s produktem také obrázek, který nahrál. Dovoluje-li produkt, aby zákazník připojil k objednávce vlastní texty, musíme je zpracovat, serializovat a uložit je do databáze. Data se musí serializovat, aby se mohl do jediného sloupce tabulky uložit obsah libovolného počtu textových polí (ke každému produktu se může vázat i několik textových vstupů), včetně odpovídajících názvů těchto polí.
- Pokud se jedná o upravitelný produkt, měl by být v poli reprezentujícím košík uložený s prefixem `customizable-`. Díky tomu je možné standardní a upravitelné produkty rozlišit a zpracovat odlišným způsobem.
- Kromě prefixu a identifikátoru produktu musíme uložit také unikátní referenci signalizující, že lze produkt upravit výběrem některé z jeho variant. To nám v kombinaci s prefixem umožňuje uvést v košíku každou úpravu odděleně. Zákazník si konec konců může koupit tři trička, ale každé z nich v jiné velikosti nebo s jiným obrázkem. V takovém případě by se měla zobrazit v košíku jako tři samostatné produkty. Pokud by se jednalo o stejné verze triček, zobrazilo by se pouze jedno, v množství tři.

Řadič

Kromě výše uvedených funkcí musí být náš řadič v této fázi kapitoly schopen:

- Zobrazit společně s obsahem košíku zadané vlastní texty, aby mohl zákazník vidět, které produkty v jeho košíku byly upraveny a jakým způsobem. Zákazník může díky tomu upravit množství nebo zcela odstranit určité upravené verze z košíku, bude-li chtít.
- Zobrazit nahraný obrázek, ze stejného důvodu jako v předchozím případě. Kromě toho bude košík pro zákazníka atraktivnější.
- Zobrazit odkaz ke stažení souboru, který zákazník nahrál, tak aby mohl zákazník ověřit, že k objednávce připojil správný soubor ještě předtím, než bude pokračovat dál.

Přidávání variant produktů

V souvislosti s přidáváním produktů, které mohou mít varianty, do našeho nákupního košíku, je opět třeba provést řadu změn databáze a kódu.

Nová databázová tabulka

Tabulka, jako je ta, kterou jsme vytvořili v kapitole 4, *Varianty produktů a nahrávání uživatelských souborů na server*, spojující produkty s potencionálními variantami, je zapotřebí pro asociaci vybraných variant produktů s produkty v nákupním košíku zákazníka. Jediný rozdíl je v tom, že nepotřebujeme sloupec pro pořadí a rozdíl v ceně.

Sloupec	Typ	Popis
`Basket_id`	`Integer`	Identifikátor produktu v košíku zákazníka
`Attribute_id`	`Integer`	Identifikátor konkrétního atributu tohoto produktu signalizující, jak je daná instance produktu v košíku upravená nebo která varianta produktu to je

Tuto novou tabulku reprezentuje následující kód:

```
CREATE TABLE IF NOT EXISTS `basket_attribute_value_association` (
  `basket_id` int(11) NOT NULL,
  `attribute_id` int(11) NOT NULL,
  KEY `basket_id` (`basket_id`,`attribute_id`),
  KEY `attribute_id` (`attribute_id`)
) ENGINE=InnoDB
  DEFAULT CHARSET=utf8 COLLATE utf8_czech_ci
  COMMENT='Asociace záznamů košíku s hodnotami atributů';
```

Změny modelu

Model vyžaduje následující změny:

- Uložit produkty s variantami, přidané zákazníkem do košíku, do pole pod prefixem `variation-`. To umožňuje, aby se tyto produkty v košíku a v průběhu objednávky zobrazovaly odlišným způsobem.
- Kromě prefixu a identifikátoru produktu musíme uložit také unikátní referenci signalizující, že lze produkt upravit výběrem některé z jeho variant. To znamená, že pokud zákazník přidal do košíku dvě varianty téhož produktu, budou v něm uvedené jako dva samostatné produkty. Jejich identifikátory dovolují zákazníkovi upravit množství a také je z košíku odstranit. Pokud by tyto reference nebyly unikátní, odstranění jedné varianty produktu z košíku by odstranilo i ostatní, což neměl zákazník v úmyslu.
- Kontrola, jestli jsou s produkty v košíku spojené varianty, tak aby se detaily těchto variant zobrazily.

- Uložit reference na varianty v poli s obsahem košíku.
- Při přidávání produktu zkontrolovat, jestli zákazník nezvolil některou z variant produktu. Varianty produktu (velikost, barva atd.) se zobrazují ve formě nabídek v pohledu produktu. Když chce zákazník přidat některý z takovýchto produktů do košíku, učiní tak stisknutím tlačítka pro odeslání formuláře, který obsahuje také zvolené prvky nabídek. Tato data můžeme zkontrolovat a určit, které varianty a atributy je třeba uložit.

Řadič

Kromě funkcí diskutovaných dříve v této kapitole musí náš řadič při prohlížení košíku zobrazovat také atributy variant, aby si byl zákazník vědom toho, jaké přesně produkty (nebo jaké varianty produktů) se chystá koupit, a mohl změnit jejich množství a případně je z košíku odstranit.

Změna množství

Je žádoucí, aby mohl zákazník snadno změnit množství produktu poté, co ho přidá do košíku. Naší nadějí samozřejmě je, že zákazník navýší množství produktů a koupí jich více, stejně tak ale musí mít možnost množství snížit nebo produkty úplně odstranit, dokud nebude s obsahem košíku zcela spokojený a bude moci pokračovat v objednávce.

Při zobrazování košíku bude každý prvek pole s obsahem košíku představovat nový řádek tabulky v pohledu a tento řádek bude obsahovat umožňující změnu množství. Tento sloupec se bude odkazovat na klíč, který daný řádek má v poli s obsahem košíku. Tím je zajištěno, že se aktualizuje ten správný produkt. Konec konců, pokud máme dvě upravené instance produktu a chceme zakoupit dvě kopie jedné z nich, budeme chtít změnit množství tohoto záznamu v modelu a v databázi.

Následující kód v našem řadiči aktualizuje košík s použitím nových **množství** zadaných zákazníkem:

```
/**
 * Aktualizace obsahu nákupního košíku
 */
private function updateBasket()
{
  // Nejdřív se zaplní model košíku, pokud už se tak nestalo
  if( ! $this->basket->isChecked == true )
  { $this->basket->checkBasket(); }
```

Následně projdeme všechny záznamu v tabulce košíku (tj. každý unikátní produkt v košíku zákazníka) a u záznamů, kde zákazník změnil množství, toto množství aktualizujeme. Pokud zákazník odstraní údaj o množství produktu, odstraní se z košíku.

```
  foreach( $this->basket->getContents() as $pid => $data )
  {
    // určení identifikátoru produktu v košíku
    $bid = $data['basket'];
```

```
    if( intval( $_POST['qty_' . $bid ] ) == 0 )
    {
      $this->basket->removeProduct( $bid );
    }
    else
    {
      $this->basket->updateProductQuantity( $bid,
        intval( $_POST['qty_' . $bid] ) );
    }
  }
  // nastavením vlastnosti embedded na false si ušetříme další
  // zpracovávání
  $this->embedded = false;

  // uživatele následně přesměrujeme na stránku s obsahem
  // košíku, potvrzující provedení požadované změny

  $this->registry->redirectUser('basket', 'Košík aktualizován',
    'Váš nákupní košík byl aktualizován', false );
}
```

Je-li množství nulové, anebo pokud zákazník klepne na odkaz **odstranit produkt**, zavolá se metoda `removeProduct`, která zajistí odstranění produktu z košíku.

```
public function removeProduct( $bid )
{
  $this->basket->removeProduct( $bid );
  $this->registry->redirectUser('basket' , 'Produkt odstraněn',
    'Produkt byl odstraněn z košíku', false );
}
```

Blok `switch` konstruktoru musí také doznat určitých změn, aby zajistil volání patřičných metod v závislosti na stránce, kterou zákazník navštívil, ať už se jedná o stránku potvrzující aktualizaci košíku nebo o stránku potvrzující odstranění produktu z košíku.

```
switch( $urlBits[1] )
{
  case 'view':
    $this->viewBasket();
    break;
  case 'add-product':
    echo $this->addProduct( $urlBits[2], 1);
    break;
  case 'update':
    $this->updateBasket();
    break;
  case 'remove-product':
```

```
    $this->removeProduct( intval( $urlBits[2] ) );
    break;
  default:
    $this->viewBasket();
    break;
}
```

Obě metody produktu, pro **aktualizaci i odstranění**, spoléhají na metody modelu košíku. Tyto metody jsou velice jednoduché. Pro změnu množství produktu stačí jednoduše aktualizovat tabulku košíku novým množstvím.

```
public function updateProductQuantity( $basketItemId, $quantity )
{
  $s = session_id();
  $u = ( $this->registry->getObject('authenticate')->
    isLoggedIn() ) ? $this->registry->
    getObject('authenticate')->getUserID() : 0;
  $ip = $_SERVER['REMOTE_ADDR'];
  $changes = array( 'quantity' => $quantity );
  $this->registry->getObject('db')->
    updateRecords( 'basket_contents', $changes, " session_id='{$s}'
    AND user_id={$u} AND ID={$basketItemId} AND ip_address='{$ip}' ");
}
```

Pro odstranění produktu jednoduše odstraníme záznam z tabulky košíku, korespondující s danou instancí produktu v košíku zákazníka.

```
public function removeProduct( $basketItemId )
{
  $s = session_id();
  $u = ( $this->registry->getObject('authenticate')->isLoggedIn() )
    ? $this->registry->getObject('authenticate')->getUserID() : 0;
  $ip = $_SERVER['REMOTE_ADDR'];
  $this->registry->getObject('db')->
    deleteRecords( 'basket_contents', " session_id='{$s}'
    AND user_id={$u} AND ID={$basketItemId}
    AND ip_address='{$ip}' ", 1 );
}
```

Od návštěvníka k uživateli

Když na náš web zavítá návštěvník, přidá do svého košíku produkty a poté se zaregistruje, musíme tyto produkty převést do jeho uživatelského košíku. Ke košíku tak budou moci přistupovat také na jiných počítačích.

Metoda transferToUser

Tuto operaci usnadní metoda modelu košíku s názvem `transferToUser`, která vyhledá veškeré produkty v tabulce košíku spojené s relací uživatele a aktualizuje tyto záznamy tak, aby se vázaly k identifikátoru daného uživatele.

```
/**
 * Převede košík na zadaného uživatele
 * @param int user id
 * @return bool
 */
public function transferToUser( $user )
{
  $changes = array( 'user_id' => $user );
  $s = session_id();
  $ip = $_SERVER ['REMOTE_ADDR'];
  $this->registry->getObject('db')->
    updateRecords( 'basket_contents', $changes,
      " SESSION_ID='{$s}' AND ip='{$ip}' ");
  return true;
}
```

Možná jste si všimli, že se při vyhledávání produktů nezohledňuje stáří záznamů. Důvodem je, že by se zastaralé záznamy v tabulce košíku měly automaticky odstraňovat, což probereme v následující části kapitoly.

Provedení převodu

Metoda `transferToUser` by se měla volat v průběhu přihlašování uživatele. V řadiči budeme muset použít parametr `directCall`, abychom zajistili, že se řadič nebude pokoušet zobrazit stránku košíku, kterou se uživatele nesnaží zobrazit.

Úklid košíku

Nákupní košíky by se měly vyprazdňovat, zejména pak v těchto případech:

- Zákazník si to sám vyžádá
- Zákazník potvrdí objednávku
- Obsah košíku je zastaralý a neváže se k uživatelskému účtu

Zastaralé záznamy

Můžeme vytvořit metodu, která vyprázdní košík na základě požadavku od uživatele. Tu však nebude možné použít pro odstranění zastaralých záznamů, protože ty se nevážou ke konkrétní-

mu uživateli. Potřebujeme nějakým způsobem odstranit z tabulky košíku veškerá zastaralá data. Měli bychom tak učinit proto, že po nějaké době už zákazník stejně produkty ve svém košíku neuvidí z důvodu změny jeho adresy IP poskytovatelem připojení nebo zahájení nové relace s novým identifikátorem. V zásadě se jedná o osiřelé produkty košíku, které je třeba odstranit, aby byl obsah databáze stále aktuální a databáze neobsahovala nadbytečná data.

Zobrazení košíku na každé stránce

Většina elektronických obchodů zobrazuje na každé stránce malý nákupní košík, často v horní části stránky nebo po jejím boku, připomínající zákazníkovi, kolik produktů má ve svém košíku, jejich celkovou cenu a také sloužící jako odkaz umožňující zobrazení obsahu košíku a dokončení objednávky.

Abychom mohli na každé stránce zobrazit košík, potřebujeme:

- Šablonu prázdného košíku reflektující tuto skutečnost adekvátní zprávou pro případ, že zákazník nemá nic v košíku.
- Šablonu košíku zobrazující počet produktů v košíku, jejich celkovou cenu a odkaz umožňující zákazníkovi postoupit dál a dokončit objednávku.
- Značku šablony v našich hlavních šablonách na místě, kam se má vložit košík. Poté, co se ověří obsah košíku, vloží se na místo této značky patřičný košík nebo košík prázdný.
- Něco v našem frameworku, co by umožnilo propojení do košíku na každé stránce, ne pouze při přímém zavolání řadiče košíku. Jinak by zákazník mohl košík vidět pouze tehdy, pokud by otevřel stránku s obsahem košíku.

Funkčnost

Propojit košík s každou stránkou můžeme v rámci našeho frameworku voláním řadiče košíku ze souboru `index.php`, avšak bez použití parametru `directCall`. To by zabránilo frameworku ve zpracování adresy URL a spuštění předdefinovaných částí řadiče. Místo toho jednoduše zavoláme metodu `checkBasket`, importujeme relevantní šablonu a použijeme proměnné z modelu určující počet produktů a jejich celkovou cenu jako značky šablony. A malý košík je na stránce.

Následující kód patří do souboru `index.php`. Jednoduše vytvoří objekt `Basketcontroller` a následně zavolá metodu `smallBasket` tohoto objektu.

```
// košík
require_once('controllers/basket/controller.php');
$basket = new Basketcontroller( $registry, false );
$basket->smallBasket();
```

Do řadiče `Basketcontroller` musíme přidat metodu `smallBasket`, která zajistí přidání patřičných dat a šablon do pohledu.

```
/**
 * Malý košík - připravý malý vložený košík
 * @return void
 */
```

Tato metoda musí v prvé řadě zkontrolovat obsah košíku, aby získala počet produktů v něm a jejich celkovou cenu. Jak už jsme řešili dříve, poté, co obsah košíku prověříme, nastavíme proměnnou indikující tuto skutečnost, abychom zabránili zbytečnému opakování této operace. Nejdříve tedy zkontrolujeme, jestli už jsme obsah košíku neprověřili, a pokud ne, zavoláme metodu `checkBasket`, která připraví potřebné informace.

```
public function smallBasket()
{
  if( $this->basket->isChecked() == false )
  { $this->basket->checkBasket();  }

  // nastavení vlastnosti signalizující vložený košík
  $this->embedded = true;
```

Pokud košík není prázdný, vložíme do pohledu šablonu košíku a nastavíme patřičné hodnoty.

```
// kontrola, jestli je košík prázdný
if( $this->basket->isEmpty() == false )
{
  // košík není prázdný, použije se tedy šablona košíku
  // a nastaví se proměnné numBasketItems a basketCost šablony
  $this->registry->getObject('template')->
    addTemplateBit('basket', 'basket.tpl.php');
  $this->registry->getObject('template')->getPage()->
    addPPTag('numBasketItems', $this->basket->getNumProducts() );
  $this->registry->getObject('template')->getPage()->
    addPPTag('basketCost', $this->basket->getTotal());
  $this->registry->getObject('template')->getPage()->
    addPPTag('shippingCost', $this->basket->getShippingCost());
}
```

Pokud je košík prázdný, vložíme šablonu prázdného košíku.

```
  else
  {
    // košík je prázdný, použije se šablona prázdného košíku
    $this->registry->getObject('template')->
      addTemplateBit('basket', 'basket-empty.tpl.php');
  }
}
```

Shrnutí

V této kapitole jsme učinili první krok k tomu, aby naši zákazníci mohli provést objednávku: vytvořili jsme nákupní košík. Začali jsme s košíkem podporujícím standardní produkty a rozšířili ho o podporu produktů, které mohou zákazníci upravovat a které nabízejí různé varianty.

Nyní se můžeme přesunout k procesu objednávky, jejímž prvním krokem je právě tento nákupní košík, na kterém budeme stavět a umožníme tak zákazníkům učinit objednávku.