Лабораторная работа №1
«Сценарий транзакции»

Цель работы

Познакомиться с предметной областью, реализовать минимальный функционал без детального проектирования согласно паттерну «сценарий транзакции»

Задание для выполнения

Определить предметную область согласно варианту (или предложить свою). Реализовать несколько действий в рамках выбранной предметной области согласно паттерну «сценарий транзакции». Не требуется полноценное работоспособное приложение, достаточно пары функций.

Бэкенд реализовать на PHP7, фронтэнд – не играет роли, поэтому лучше всего простой html+js+css без фреймворков. Здесь и далее для нужд доставки запросов и ответов на бэкенде может использоваться любой PHP-фреймворк (рекомендуется Symfony), либо можно обойтись без фреймворков.

Варианты

Можно использовать следующие предметные области (или предложить свои) – по согласованию с преподавателем. Один человек – одна предметная область. Учтите, что в данной предметной области будут все последующие ваши лабораторные работы.

- доставка еды;
- продажа билетов на мероприятия;
- управление стадионом;
- учёт военнообязанных;
- управление парковкой;
- учёт абитуриентов вуза;
- разработка нефтегазовых месторождений;
- премия «Оскар»;
- космический туризм;
- публикация научных статей;
- регистратура поликлиники;
- старая добрая библиотека;
- управление автомойкой;
- учёт иностранных шпионов;
- диспетчерская городского транспорта;

- система трансферов федерации футбола;
- управление фитнес-центром;
- управление шиномонтажем;
- провайдер телекоммуникационных услуг;
- фермерское хозяйство;
- продажа игрушек;
- промышленная робототехника;
- продажа электрооборудования;
- детский сад;
- управление дорожным движением;
- арендное агенство;
- …

Теоретические сведения

Сценарий транзакции (transaction script) *– это способ организации бизнес-логики по процедурам, каждая из которых обслуживает один запрос, инициируемый слоем представления.*

Многие бизнес-приложения могут восприниматься как последовательности транзакций. Одна транзакция способна модифицировать данные, другая – воспринимать их в структурированном виде и т.д. Каждый акт взаимодействия клиента с сервером описывается определённым фрагментом логики. В одних случаях задача оказывается настолько же простой, как отображение части содержимого базы данных. В других могут предусматриваться многочисленные вычислительные и контрольные операции.

Сценарий транзакций организует логику вычислительного процесса преимущественно в виде единой процедуры, которая обращается к базе данных напрямую или при посредничестве кода тонкой оболочки. Каждой транзакции ставится в соответствие собственный **сценарий транзакций** (общие подзадачи могут быть вынесены в подчинённые процедуры).

Принцип действия

При использовании типового решения **сценарий транзакции** логика предметной области распределяется по транзакциям, выполняемым в системе. Если, например, пользователю необходимо заказать номер в гостинице, соответствующая процедура должна предусматривать действия по проверке наличия подходящего номера, вычислению суммы оплаты и фиксации заказа в базе данных.

Простые случаи не требуют особых объяснений. Разумеется, как и при написании иных программ, структурировать код по модулям следует осмысленно. Это не должно вызвать затруднений, если только транзакция не оказывается слишком сложной. Одно из основных преимуществ **сценария транзакций** заключается в том, что не приходится беспокоиться о наличии и

вариантах функционирования других параллельных транзакций. Задача – получить входную информацию, опросить базу данных, сделать выводы и сохранить результаты.

Где расположить **сценарий транзакций**, зависит от организации слоёв системы. Этим местом может быть страница сервера, сценарий CGI или объект распределённого сеанса. Предпочтительнее обособлять **сценарии транзакций** настолько строго, насколько это возможно. В самом крайнем случае можно размещать их в различных подпрограммах, а лучше – в классах, отличных от тех, которые относятся к слоям представления и источника данных. Помимо того, следует избегать вызовов, направленных из **сценариев транзакций** к коду логики представления; это облегчит тестирование **сценариев транзакций** и их возможную модификацию.

Существует два способа разнесения кода **сценариев транзакций** по классам. Наиболее общий, прямолинейный и удобный во многих ситуациях – использование одного класса для реализации нескольких **сценариев транзакций**. Второй, связан с разработкой собственного класса для каждого **сценария транзакций**: определяется тип, базовый по отношению ко всем командам, в котором предусматривается некий метод выполнения, удовлетворяющий логике **сценария транзакций**. Преимущества каждого подхода – возможность манипулировать экземплярами сценариев как объектами в период выполнения, хотя в системах, где бизнес-логика организована с помощью **сценариев транзакций**, подобная потребность возникает сравнительно редко. Разумеется, во многих языках модель классов можно полностью игнорировать, полагаясь, скажем, только на глобальные функции. Однако вполне очевидно, что аппарат создания объектов помогает преодолевать проблемы потоков вычислений и облегчает изоляцию данных.

Термин **сценарий транзакций** выбран потому, что в большинстве случаев приходится иметь дело с одним сценарием для каждой транзакции уровня системы базы данных. Пусть такой исход нельзя гарантировать на все сто процентов, но в первом приближении это верно.

Назначение

Главным достоинством типового решения **сценарий транзакции** является простота. Именно такой вид организации логики, эффективный с точки зрения восприятия и производительности, весьма характерен и естественен для небольших приложений.

По мере усложнения бизнес-логики становится все труднее содержать ее в хорошо структурированном виде. Одна из достойных внимания проблем связана с повторением фрагментов кода. Поскольку каждый **сценарий транзакции** призван обслуживать одну транзакцию, все общие порции кода неизбежно приходится воспроизводить вновь и вновь.

Проблема может быть частично решена за счет тщательного анализа кода, но наличие более сложной бизнес-логики требует применять *модель предметной области (Domain Model)*. Последняя предлагает гораздо больше

возможностей структурирования кода, повышения степени его удобочитаемости и уменьшения повторяемости.

Определить количественные критерии выбора конкретного типового решения довольно сложно, особенно если одни решения знакомы вам в большей степени, нежели другие [1].

Это решение предлагает использовать несколько скриптовых сценариев, обрабатывающих запросы пользователя к системе и возвращающих ему результат этой обработки. Важным условием здесь является то, что сценарии отвечают за весь производственный цикл ответа, а не делегируют обработку другим частям приложения.

Рассмотрим несколько примеров:

```
<?php
// Процедурный стиль
// controller/guestbook.php
...

switch($_REQUEST['action']){
  // Просмотр гостевой книги
  case 'index':
    $template = new Template('view/guestbook/index.php', [
      'posts'  =>  Db::getInstance()->select('SELECT  *  FROM  `guestbook` LIMIT 10 ORDER BY `added` DESC')
    ]);
    echo $template->render();
    break;

  // Добавление сообщения в гостевую книгу
  case 'add':
    $text = InputStringFilter($_POST['text']);
    Db::getInstance()->insert('guestbook', [
      'text' => $text,
      'added' => time(),
    ]);
    return header('Location: /guestbook.php?action=index');
    break;
```

```
<?php
// Объектный стиль
// controller/guestbook.php
class GuestbookController{
  // Просмотр гостевой книги
  public function indexAction(){
    $template = new Template('view/guestbook/index.php', [
      'posts'  =>  Db::getInstance()->select('SELECT  *  FROM  `guestbook` LIMIT 10 ORDER BY `added` DESC')
    ]);

    return $template->render();
  }

  // Добавление сообщения в гостевую книгу
  public function addAction(){
    $text = InputStringFilter($_POST['text']);
```

```
        Db::getInstance()->insert('guestbook', [
          'text' => $text,
          'added' => time(),
        ]);

        return header('Location: /guestbook.php?action=index');
      }
```

Подобный подход чаще всего используется в небольших проектах и отличается следующими возможностями:

– быстрая разработка, за счет исключения из архитектуры дополнительных компонентов, таких как модели, сервисы и т.д.;
– простота изучения (только до определенного момента), основанная на аккумуляции всей логики приложения в скрипты транзакций

К сожалению применение этой архитектуры в средних и крупных проектах влечет следующие проблемы:

– дублирование кода, вызванное отсутствием гибкости архитектуры и невозможностью частичного вынесения логики из сценариев для повторного использования;
– раздувание сценариев (как следствие предыдущего пункта) и усложнение процесса их изучения [2]

Список использованных источников

1. Фаулер, М. Шаблоны корпоративных приложений. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2016. – 544 с. : ил. – Парал. тит. англ.
2. Архитектура: Сценарий транзакции [Электронный ресурс]. – Режим доступа: https://bashka.github.io/posts/architecture-transaction/. – Дата доступа: 25.01.2020.