Урок 5

Взаимодействие компонентов страницы, практика.

Изучение правил и практик составления системной документации. Реализация механизмов взаимодействия элементов страницы.

© geekbrains.ru

На предыдущих уроках мы уже овладели объектной моделью AJAX и мощным инструментом регулярных выражений, но пока все эти технологии мы рассматривали по отдельности. Теперь стоит собрать всё воедино и применить полученные знания на практике.
Для начала мы создадим один из ключевых модулей для любого интернет-магазина – корзину покупателя.

Соглашения и документация

В работе над сайтом очень часто участвует далеко не один человек. Как правило, есть один или несколько разработчиков на BackEnd, и такая же картина на FrontEnd. Мы уже увидели, что эти два мира умеют общаться посредством определенных протоколов, например, JSON.
Допустим, какой-то функционал уже реализован, но у FrontEnd программиста появляется потребность в получении нового параметра в JSON-ответе. Он идёт к BackEnd программисту, договаривается с ним, свойство реализуется. При продолжении разработки метод дорабатывают, поле меняется или вовсе уходит, а функционал на релизе разваливается.
Чтобы избежать подобной ошибки, при работе с AJAX (да и не только) используется особый тип документации – спецификация. Она описывает все виды общения FrontEnd и BackEnd следующим образом:

Название обмена	Добавление товара в корзину
URL	/basket/add/
Тип запроса	POST, asynchronous

Передаваемые данные	{
	user_id : 1,
	product_id : 2,
	price: 123,
	quantity : 1
	}

Ожидаемый ответ	{
	result: 1
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_code : 100,
	error_message : “Невозможно добавить товар в корзину”
	}

Коды ошибок	100 – ошибка SQL

Подобная таблица решает целый комплекс возможных проблем.
Во-первых, программисты становятся более независимы друг от друга. FrontEnd разработчик может спокойно разрабатывать свой код, не обращая внимания на то, готова ли принимающая часть на BackEnd. Ведь всегда можно подменить ответ заглушкой с нужной структурой, а структура уже зафиксирована.
Во-вторых, реализуется возможность автоматического тестирования, ведь протокол не меняется, а значит его можно запрограммировать.
В-третьих, другие разработчики, которые никогда не работали с данным функционалом ранее, спокойно могут прочесть документацию и понять суть того, с чем им предстоит работать.
Именно поэтому обширная и подробная спецификация является залогом надёжного ПО и крепкого сна программиста.
Такую же спецификацию мы заведём и для нашего модуля корзины, чтобы знать, к чему мы идём в итоге.

Спецификация модуля корзины

Логично, что сама по себе сущность корзины обладает тремя методами:

1. Получить текущую корзину.

2. Положить товар в корзину.
3. Удалить товар из корзины.

Их мы и опишем в нашей спецификации.

Название обмена	**Добавление товара в корзину**
URL	/basket/get/ /* но пока заменим заглушкой */
Тип запроса	POST, asynchronous

Передаваемые данные	{
	"id_user" : 123,
	}

Ожидаемый ответ	{
	result: 1,
	basket : [
	{
	id_product : 123,
	price : 100
	}
	],
	amount: 100
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Название обмена	Добавление товара в корзину
URL	/basket/add/ /* но пока заменим заглушкой */
Тип запроса	POST, asynchronous

Передаваемые данные	{
	"id_product" : 123,
	"quantity" : 1
	}

Ожидаемый ответ	{
	result: 1,
	full_price : 123
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Название обмена	Удаление товара из корзины
URL	/basket/add/ /* но пока заменим заглушкой */
Тип запроса	POST, asynchronous

Передаваемые данные	{
	id_product : 123,
	full_price : 321
	}

Ожидаемый ответ	{
	result: 1
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “ Сообщение об ошибке ”
	}

На данном этапе мы опустим создание кодов ошибок, т.к. проект наш не настолько богат кодом. Более того, нарастить коды достаточно легко при наличии спецификации и возвращаемого ответа об ошибке.
Сама по себе корзина традиционно представляет собой блок HTML-кода в шапке страницы, который отображает количество товара в корзине и его общую стоимость. Поскольку работаем мы через AJAX, нам надо будет предусмотреть легкий доступ к необходимым элементам страницы для их изменения по факту получения ответа от сервера.

Подготовка

На стороне HTML корзина будет представлена простым блоком:

```
<div id="basket"></div>
```

Её мы намеренно оставляем пустой, т.к. корзина у нас будет заполняться серверными скриптами. При загрузке основного контента страницы мы будем сразу же посылать запрос на получение корзины, чтобы разгрузить обмен данными и отдать страницу максимально быстро. Если вы уверены, что получение корзины не займёт много серверного времени, то смело можете добавлять получение корзины в функционал загрузки страницы.
Также нам потребуются три заглушки с данными:

- исходная корзина;
- ответ на добавление товара;
- ответ на удаление товара.

Разработка

На занятии по AJAX мы формировали элемент, содержащий вёрстку прямо в JS-коде. С точки зрения читаемости и стандартизации кода – это очень плохой подход. Мы ведь помним, что залог спокойствия в разделении логики на слои. Но к счастью, JavaScript предоставляет нам отличные инструмент работы с DOM-моделью. А учитывая то, что мы уже знакомы с jQuery, то можно применять весь комплекс доступного функционала сразу же.
Создание нового элемента в jQuery делается через встроенный метод, который замкнут сам на себя. Т.е. мы можем создать div следующим образом:

```
var my_div = $('<div/>', {
    id: 'foo',
    href: 'http://google.com',
    title: 'Become a Googler',
    rel: 'external',
    text: 'Go to Google!'
});
```

После этого нам достаточно просто поместить полученный элемент в DOM-модель при помощи метода appendTo():

```
my_div.appendTo('#my_selector');
```

Так что мы сможем данным образом создавать нужные нам элементы в модуле корзины сразу же по получению ответа от сервера.
Теперь остаётся переписать обращение по AJAX к серверу и обработку ответа.

Как только мы получили содержимое корзины, нужно научиться этим содержимым управлять. Для этого в объекте корзины мы создадим параметр, который будет содержать в себе текущие товары.
Здесь будет иметь место один очень важный нюанс.
Когда мы получаем данные о корзине внутри AJAX-запроса, передать их в прототип сразу же не получиться. Ведь ссылка this будет указывать не на текущий объект корзины, а на объект AJAX. Здесь нам поможет параметр context.

```
$.get({
        url: 'basket.content.json',
        context: this,
    ...
        dataType: "json"
    });
```

Такая запись говорит о том, что this будет обращаться к внешней области видимости. Таким образом, ответ сможет общаться с прототипом напрямую.
Научим наше приложение добавлять товар в корзину. Эта процедура будет состоять из следующих шагов:

1. Отловить событие “добавление товара”.
2. Собрать данные о товаре.
3. Добавить товар в составляющие элементы корзины.
4. Обновить общую стоимость корзины.

Домашнее задание

1. Написать метод удаления товара из корзины.
2. Создать модуль сбора отзывов:
    a. модуль может выводить отзывы (пока из json-заглушки);
    b. модуль может добавлять отзывы;
    c. модуль может одобрять отзывы;
    d. модуль может удалять отзывы;
    e. модуль подчиняется следующим соглашениям.

Название обмена	Добавить отзыв
URL	review.add.json
Тип запроса	POST, asynchronous

Передаваемые данные	{
	"id_user" : 123,
	“text” : “”
	}

Ожидаемый ответ	{
	result: 1,
	userMessage: "Ваш отзыв был передан на модерацию"
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Название обмена	Одобрить отзыв
URL	review.submit.json
Тип запроса	POST, asynchronous

Передаваемые данные	{
	" id_comment" : 123, // ID отзыва, который одобряется
	}

Ожидаемый ответ	{
	result: 1
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Название обмена	Удалить отзыв
URL	review.delete.json
Тип запроса	POST, asynchronous

Передаваемые данные	{
	" id_comment" : 123, // ID отзыва, который удаляется
	}

Ожидаемый ответ	{
	result: 1
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Название обмена	Показать все отзывы
URL	review.list.json
Тип запроса	POST, asynchronous
Передаваемые данные	{ }

Ожидаемый ответ	{
	comments: [
	{
	id_comment: 123,
	text: ‘Текст отзыва’
	}
	]
	}

Ответ в случае системной ошибки	{
	result : 0,
	error_message : “Сообщение об ошибке”
	}

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. “jQuery. Подробное руководство по продвинутому JavaScript” - Беэр Бибо, Иегуда Кац