УДК 004.7

ОПРЕДЕЛЕНИЕ ОСНОВНЫХ КОНЦЕПЦИЙ CMS

Гутовский Дмитрий Игоревич[1], Добрынин Владимир Николаевич[2]

[1]Аспирант;
ГБОУ ВО МО «Университет «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: mfhnb@mail.ru.

[2]Профессор;
ГБОУ ВО МО «Университет «Дубна»,
Институт системного анализа и управления;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: arbatsolo@yandex.ru.

Статья посвящена выделению основных концепций систем управления содержимым сайта (CMS). Описываются сходства и различия логических принципов работы CMS, выявляются их классификации, подбираются признаки, которые позволят произвести оптимальный выбор CMS, для WEB-сайтов любой направленности.

Ключевые слова: CMS, сайт, WEB, HTML, CSS, PHP, управление содержимым, вёрстка, Интернет.

HIGHLIGHTING OF CMS CONCEPTIONS

Gutovskiy Dmitriy[1], Dobrinin Vladimir[2]

[1]Graduate student;
Dubna State University,
Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: mfhnb@mail.ru.

[2]Professor;
Dubna State University,
Institute of the system analysis and management;
141980, Dubna, Moscow reg., Universitetskaya str., 19;
e-mail: arbatsolo@yandex.ru.

The abstract is containing the highlight of main content system management (CMS) conceptions. Wrote about similarities and differences of the logical principles of CMS working, highlighting of these classifications, selecting signs for optimal select the CMS for anything sites.

Keywords: CMS, site, WEB, HTML, CSS, PHP, content management, typesetting, the Internet.

Введение

Практически любой современный сайт имеет функции, которые позволяют менять содержимое на нём, не прибегая к написанию кода. На начальных этапах развития *WEB*, сайты были статическими. Статический сайт представлял из себя полностью свёрстанные страницы, которые располагались на сервере и передавались клиенту (браузеру компьютера) без изменений. Понятно, что такой вариант не подходит в подавляющем большинстве случаев. Представим типичный современный сайт, например, социальную сеть или Интернет-магазин. В случае со статическим сайтом, например, Интернет-магазина, каждую новую страничку с товаром приходилось бы верстать заново (подключая некоторые заранее сделанные компоненты). Но, если в Интернет-магазин, гипотетически можно нанять соответствующий штат верстальщиков, то представить себе статическую социальную сеть, невозможно в принципе. Статический сайт, теоретически, может быть визиткой, которая свёрстана

1 раз и не меняется, но даже в самых простых сайтах-визитках, иногда приходится изменять содержимое страницы, и если владелец сайта не является верстальщиком, то ему бы приходилось каждый раз прибегать к помощи сторонних лиц. Из вышесказанного становится понятно, что эпоха статических сайтов давно прошла и на смену ей пришли динамические сайты. В случае с динамическим сайтом, на сервере не находятся готовые страницы (они могут быть, но только для некоторых случаев, например, для страницы с ошибкой 404). Большинство страниц сайта генерируется динамическим образом, при помощи серверного приложения. Однако, для браузера, который не имеет понятия о серверной части сайта, нет никакой разницы, как была получена страница, что было написано вручную, а что сгенерировано сервером, так как результат – всё равно является *WEB*-страницей, состоящей из связки *HTML+CSS+JS* (в современном случае). Таким образом, появляется понятие «шаблона». Шаблон представляет из себя каркас страницы, в котором заранее свёрстаны элементы, не подлежащие изменению, а вместо динамически-выводимой информации, прописаны специальные конструкции, по которым серверное приложение вставляет те части содержимого, которые необходимо вывести в определённом месте страницы.

Например, в социальной сети есть правила вывода списка друзей, которые состоят из набора необходимой для этого разметки (*HTML*), правил внешнего оформления (*CSS*) и могут содержать элементы программной части, выполняемые браузером (*JS*). Эти правила вывода и являются частью шаблона, для пользовательских страниц социальной сети. Конкретные друзья каждого пользователя вставляются динамически. Эти данные, как-правило, достаются из базы данных (БД). В результате серверное приложение оставляет без изменений каркас шаблона, а на место конструкций для вывода динамического содержимого, вставляет данные из БД, по тем правилам, которые написаны в конкретных частях шаблона, для этого приложения. У пользователя сайта, который имеет право изменять контент, есть панель управления, где он вставляет нужное содержимое, а оно выводится потом на конечной странице, по правилам шаблона. Это серверное приложение, позволяющее управлять содержимым сайта, носит название *CMS* (*Content Management System*). *CMS* может быть написана с нуля, под конкретный проект, но чаще всего это не так, так как не все проекты настолько сложные и дорогостоящие, чтобы полное написание *CMS* было оправдано. В большинстве случаев используется готовая *CMS* (движок) сайта. Однако, это не означает, что сайт полностью собирается из готовых решений. Если сайт, более-менее профессиональный и индивидуальный, то часть его серверной компоненты пишется самостоятельно, с учётом правил *CMS*. Как-правило, самостоятельно пишется именно шаблон для *CMS*. Вся проблема в том, что сайты бывают различной направленности, например, блог, форум, Интернет-магазин и т. д. Помимо конечных страниц, которые видят все (даже неавторизированные пользователи), нужно ещё правильно предусмотреть функционал управления содержимым, для сайта конкретной направленности. Понятно, что *CMS* можно классифицировать по множеству различных критериев, но есть концептуальные особенности, по которым можно раскласифицировать все *CMS*, вне зависимости от того, индивидуальны они или общеизвестны. В данной статье выведены и описаны 2 основных концепции, которые не следует привязывать к конкретным *CMS*, название этих систем управления, приведено для примера, по причине их популярности. В данной статье не рассматриваются подробные приёмы изготовления шаблонов (посадки вёрстки) на *CMS*. Описываются, именно общие, концептуальные различия.

MODx-концепция

Под *MODx*-концепцией понимается способ интерпретации шаблонов, который используется в *CMS MODx*.

Все *CMS*, независимо от их направленности, можно разделить на 2 группы. Одну группу можно отнести к *CMS* с *MODx*-концепцией, другую – с не *MODx*-концепцией. *CMS* первой группы гораздо меньше, чем второй, а *MODx* взят для примера, как бесплатная и наиболее популярная *CMS* подобного вида.

При переводе *HTML*-вёрстки в шаблон, для какой-либо *CMS*, возникают две основных составляющих. Первая передаётся клиенту без изменений, вторая генерируется динамически. Представим весь шаблон в виде секций (в абстрактном смысле, а не в тэгах *<section></section>*). Определим некоторые секции заранее, например, заголовок статьи. Допустим, что данный заголовок всегда выводится одинаково (в пределах действующего шаблона), пусть это будет тэг *<h1>* с классом *page-*

title. В этом случае, тэг заголовка и его класс передаются без изменений, а информация, находящаяся внутри заголовка, генерируется динамически.

Так как клиентская часть сайта, представляющая из себя связку *HTML+CSS+JS*, не зависит от способа её получения, то с точки зрения браузера нет разницы в двух вышеупомянутых составляющих шаблона. В итоге, та часть, что передаётся без изменений, не зависит от того, какая *CMS* используется, и есть ли она вообще. Динамическая же часть встраивается в статический каркас принципиально по-разному. В случае с не *MODx*-концепцией, для вывода определённой информации, нужно вставить в соответствующие места шаблона, заранее зарезервированные конструкции, отвечающие за определённый пласт содержимого, например, вывод пользовательского контента или виджетов боковой панели. Даже если создаётся новый информационный раздел, которого нет в используемой *CMS*, то для его создания и последующего вывода, применяются специфические, для данной *CMS*, наборы функций и параметров. В *CMS* с не *MODx*-концепцией нет штатных средств, позволяющих создать новые информационные сущности, поэтому требуется знать *API* (*Application Programming Interface*), и разрабатывать шаблон по строго-определённым правилам. Естественно, что невозможно знать всех функций, в любых версиях *CMS*, и такой подход сильно усложняет разработку шаблонов и модулей, а также увеличивает риск несовместимости с разными версиями *CMS* (как прямой, так и обратной).

Второй недостаток не *MODx*-концепции, вытекает из первого и заключается в строгой иерархии файлов в шаблонах и плагинах. Например, в *WordPress* темы должны находиться в папке */wp-content/themes*, файлы в этих темах тоже должны быть с определёнными названиями. Становится сложнее замаскировать *CMS*, что ведёт к снижению безопасности.

Третий недостаток заключается в избыточной дифференциации содержимого. Та часть шаблона, которая отвечает за динамически генерируемое содержимое, в свою очередь, подразделяется на множество разделов, имеющих разные правила поведения на сайте. Например, в *WordPress*, информация из виджетов или опций темы, выводится с одинаковым содержимым, на всех страницах, что их поддерживают. Записи различных типов, уже выводятся по другим правилам. Соответственно, функции для создания и вывода таких объектов, также различны. Всё это не только усложняет код, но и увеличивает нагрузку на сервер. Плюс к этому, дифференциации под любую ситуацию предусмотреть невозможно, что ведёт к неиспользованию некоторых разделов (например, виджетов), но созданию своих.

Следующий недостаток, вытекающий из предыдущего, заключается в том, что модераторам становится сложнее использовать сайт с не *MODx*-концепцией, так как создание своих сущностей, в таких *CMS* достаточно проблематично, а имеющиеся могут использоваться не по назначению. Допустим, при наличии боковой колонки, модератор идёт в соответствующий раздел, пытаясь найти там необходимые функции. Но, может быть так, что боковая колонка создаваемой темы не имеет привычных виджетов, что должны быть одинаковыми на разных страницах. Допустим, что виджеты разные на страницах. Тогда, в терминологии *WordPress*, это уже не виджеты.

Ещё один недостаток заключается в неполном переопределении модулей, и наиболее заметен на *CMS Joomla*. В данной *CMS*, которая также относится к не *MODx*-концепции, имеется разделение шаблона на визуальные позиции, что призвано сделать редактирование более удобным, позволяя разнести модули по нужным позициям, а не привязывать определённые группы содержимого к конкретным участкам. Например, есть модуль поиска по сайту, куда его вставить, а точнее на какую позицию, например, в «шапку» или «подвал», решает разработчик, хотя можно создать и фиксированные модули в шаблоне. Это ближе к *MODx*-концепции, но основная проблема остаётся. Сами модули, а точнее функции вывода определённой информации в них, не являются произвольными. Даже если создаётся произвольный модуль, то он, либо содержит группу имеющихся модулей, либо подчиняется специфическому *API*, для создания новых сущностей. У каждого модуля в *Joomla* имеется стандартный вид (представление) вывода. Для того, чтобы разработчики могли изменить дизайн конкретного модуля под оформление определённой темы, в *Joomla* имеется функция переопределения. Дело в том, что набор встроенных модулей в разных версиях данной *CMS*, не как не стандартизирован. Это может привести к тому, что в новой версии, какой-либо модуль может быть упразднён или изменён, что приведёт к разрушению переопределений, а также зависящих от него, других модулей. Это, в свою очередь, приведёт к некорректной работе шаблона (или её отсутствию). Из этого же следует недостаток для всех *CMS* с не *MODx*-концепцией, заключающийся в том, что *API* разных *CMS* могут меняться от версии к версии, функции могут изменять и отменять.

Также, при создании шаблона для *CMS* с не *MODx*-концепцией, необходимо учитывать возможные конфликты с различными плагинами. Дело в том, что из-за жёсткой дифференциации содержимого, плагины выполняют не только генерацию определённой информации и функционала, но несут избыточный код, который не касается их напрямую. Например, в плагине фотогалереи может быть встроенная библиотека, дающая изображениям различные стилевые эффекты. Никто не может гарантировать, что стилевые файлы, подключаемые подобными плагинами, не создадут конфликтов с основным стилевым оформлением темы.

То же самое касается и *JS*-библиотек. Плюс к этому, некоторые плагины в *CMS* с не *MODx*-концепцией, зачастую завязываются на определённый участок страницы. Дело в том, что предполагается использовать все типичные сущности *CMS*, в создаваемой теме. То есть, если изготовляемая тема, например, для *WordPress*, не предполагает областей вывода меню, а оно реализовано путём вставки в виджеты, то некоторые плагины для расширения функционала меню могут отказаться работать. В худшем случае сайт может утратить работоспособность, и для её возврата, предстоит деактивировать плагин, путём прямого запроса к БД или захода на сервер по какому-либо файловому протоколу, с последующим удалением файлов плагина. Однако, если речь идёт о *CMS*, типа *Joomla*, то там, в отличии от того же *WordPress*, присутствует ещё и обязательная регистрация некоторых типов пакетов, с занесением сведений в соответствующие таблицы БД. Поэтому, удаление файлов плагина, в этом случае, может полностью не снять проблему.

При замене тем, для вышеупомянутых *CMS*, тоже могут возникать проблемы. Дело в том, что в некоторых *CMS*, в частности в *WordPress*, могут иметься дополнительные ключи для некоторых сущностей, в частности – виджетов. Допустим, что заполнялся сайт на *WordPress*, а после замены темы на другую, все виджеты пропали. Допустим, что разделов для виджетов (чаще – боковых колонок), в обеих темах равное количество, но их содержимое не перераспределится, так как идентификаторы данных разделов не обязаны совпадать (идентификаторы, в рамках *CMS*, не путать с понятием *ID* в *HTML*).

Ещё, при использовании *CMS*, не соответствующих *MODx*-концепции, помимо функционального переопределения, может возникать проблема с переопределением стилей некоторых разделов, что может привести к необходимости изменять статическую часть вёрстки, например, изменить классы для пунктов меню. Всё это может приводить к нарушению единого стиля вёрстки, например, при использовании БЭМ.

Из предыдущего пункта вытекает ещё одна проблема, а именно – возможные нарушения семантики в конечной вёрстке. Переопределение может не быть, так как, в некоторых случаях, функция генерации определённого контента может не возвращать результат, а сразу выводить его на сторону клиента. В этом случае приходится писать плагин для фильтрации выводной информации и замены её «на лету». Если проблему с несоответствием классов, пусть даже в нарушение единой стилистики вёрстки, можно решить модернизацией *CSS*, то с семантикой придётся только фильтровать и заменять. Это сильно понизит производительность сайта, так как не всякие секции подлежат кэшированию на сервере.

Несмотря на все вышеуказанные недостатки, *CMS*, которые не придерживаются *MODx*-концепции, имеют некоторые преимущества. Наиболее важным из них, является возможность автоматической установки шаблонов и различных плагинов. То есть, написанный шаблон, может быть установлен на сайт простым пользователем. Если сайт делается не «под ключ», а изготавливается модуль, на котором будут строить сайты люди, не имеющие отношение к *WEB*-разработке, то *MODx* и ей подобные *CMS* – не самый лучший выбор.

Ещё одна ситуация, в пользу не *MODx*-подобных *CMS*, если сайт имеет чёткую стандартную структуру и функциональность, например, блог, форум или Интернет-магазин. Поскольку в специализированных *CMS* имеется набор предопределённого функционала. Однако, многие специализированные *CMS*, типа *WordPress*, давно перестали быть таковыми, и теоретически, могут стать основой сайтов любой сложности и направленности. Однако, на практике, всё гораздо сложнее. Если функционал, например, блога отличается от стандартного не сильно, например, добавляется новый раздел с фотогалереей, которого нет в конкретной *CMS*, то разработка ещё не теряет целесообразность. Однако, при большом количестве произвольных полей ввода данных, при создании множества несвойственных, для выбранной *CMS*, разделов, разработка сильно усложняется и снижается итоговая производительность сайта.

В *MODx*-концепции, динамически генерируемая часть – гораздо более гибко управляемая, нежели в противоположной концепции. Дело в том, что в *MODx* и её аналогах, имеются штатные средства, для создания, ни только контента, но и функционала сайта. Первым, и наиболее ярким примером, является создание переменных шаблона. Разработчику не нужно писать множество кода, для создания своего дополнительного поля. Требуется, лишь выбрать соответствующую опцию, средствами графического интерфейса. При создании переменной шаблона указывается произвольное название, описание и подсказка для пользователя. Настраиваются параметры ввода и вывода, то есть уже имеются необходимые типы данных, которые могут быть введены пользователем, нужно просто выбрать требуемый. Если пользователь, например, в каком-то поле должен выбрать логотип, то разработчиком просто указывается тип ввода «изображение», после чего, пользователь получает нужное диалоговое окно. Дифференциации разделов, на различные типы записей, разделы для виджетов, опции темы и т. д. – нет. Разработчик сам определяет поведение, название и другие параметры всех разделов.

Также, нет зарезервированной иерархии для шаблонов и модулей. В *MODx*-концепции размещать файлы разрабатываемых компонентов можно где угодно, в пределах пространства файлов сайта. Также, есть выбор, позволяющий разместить динамически генерируемую часть, не в виде файлов, а в соответствующих таблицах БД сайта. Обычно, так и делают, при создании сайта на *MODx*-подобных *CMS*. Данный подход позволяет не выдавать стандартную файловую структуру на сайте, а также не передаёт иерархию дополнительных плагинов и компонентов, что положительно сказывается на безопасности. Если в случае с большинством *CMS*, используются функции, выводящие разную информацию, то в *MODx*-подобных *CMS*, имеется анализатор, который разбирает нужный компонент, например, шаблон, подставляя на места вызова динамических компонентов, результаты их работы. Это аналогично высокоуровневым языкам программирования, использующим императивный подход. Все переменные шаблона абсолютно равнозначны в возможных настройках работы с ними. Вне зависимости от того, является ли переменная зарезервированной или созданной, её вызов происходит одинаково и может быть произведён в произвольном месте вёрстки. Зарезервированные переменные шаблона, можно и не выводить, хотя они базовые и используются на сайте любой направленности. Трудно представить себе страницу сайта, не имеющую названия или содержимого.

Также, в *CMS* с *MODx*-концепцией используются общепринятые сущности, такие как чанки и сниппеты. Для их вызова, также не требуется знать сложного синтаксиса специфических функций. Чанки и сниппеты могут быть произвольно названы, и вызваны – где угодно, в приделах шаблона (или его чанка). Чанк – фрагмент клиентской части, как правило – фрагмент шаблона. Сниппет – фрагмент серверной части, отвечающий за отдельный функционал. В случае с *MODx*, серверным языком является *PHP*. Можно написать любой функционал, на чистом *PHP*, вызвать его в любом месте шаблона, а в качестве входных параметров, будут выступать названия переменных в сниппете. Такой подход даёт возможность использования вызовов сниппетов, как шорт-кодов.

Также, в *MODx* нет стандартного специфического функционала, например, вывода меню. Все нужные функции, характерные для разных типов сайтов, добавляются на сайт, средствами установки соответствующих дополнений, что ещё больше облегчает разработку. В *MODx*-концепции нет перегрузки специфическими сущностями, поэтому *CMS,* основанные на них, подходят для любых типов сайтов.

Так как в *MODx*-концепции нет чёткой дифференциации разделов, то можно удовлетворить любые требования заказчиков. Абсолютно не важно, какое содержимое выводится. Всё оно содержится в переменных шаблона. В каком участке вёрстки вывести содержимое, и как оно будет отображаться (одинаково для всех страниц или по-разному), решает разработчик, в соответствии с требованиями заказчика.

Также, в *MODx*-концепции очень гибкая настройка прав. Дело в том, что в отличии от специализированных CMS, нет конкретного разделения на предопределённые группы пользователей. Разработчик получает набор базовых (атомарных) правил, позволяющих отдельно настроить права для разных групп, вплоть до возможности видеть (или не видеть) определённые страницы и элементы на них.

Так как базовый функционал *MODx*-подобных *CMS* не несёт в себе избыточных возможностей, то нет потребности частого изменения *MODx API*. Функционал таких систем не нацелен на вывод

пользовательских данных, а напоминает своеобразный язык программирования, в котором происходит интерпретация шаблонов и модулей, функционал которых произволен и разработан под конкретный сайт. Благодаря этому, практически никогда не возникает проблемы несовместимости с более новыми версиями. Например, в *CMS Joomla* иногда приходится долгое время не производить обновлений, рискуя безопасностью сайта и конфиденциальных данных, во избежание того, чтобы не перестал работать тот или иной модуль (например, шаблон). В *MODx* же можно безбоязненно обновляться, весь специфический функционал, созданный разработчиком конкретного сайта, будет работать без изменений. Также, в экстренных случаях, гораздо легче решается возможность «отката» на более старую версию *CMS*. Хотя, в любом случае, лучше делать резервную копию всего сайта, перед подобными операциями, вне зависимости от платформы.

Применение *MODx* и аналогичных *CMS* оправдано в том случае, если делается конкретный сайт. Если фрагменты этого сайта (например, шаблон) не предполагается использовать на сторонних сайтах обычными пользователями. Универсальность *MODx*-подобных *CMS* очень сильно облегчает разработку, но накладывает отпечаток в виде отсутствия стандартов на весь функционал, кроме базового. Конечно, правила вызова переменных шаблона, или характерные таблицы в БД будут, но из-за отсутствия стандартных разделов, файловой структуры шаблонов и плагинов, теряется возможность автоматической установки шаблонов. Однако, плагины и компоненты, не отвечающие за вывод информации напрямую, устанавливаются автоматически. В *MODx*-концепции, обычно используются только такие плагины, а прямой вывод результатов, не оформляют в виде плагинов (хотя, это не запрещено). Аналогично тому, как используют функции в стандартных языках. Таким образом, разработчик делает определённый функционал, который оптимизирован под конкретный сайт.

Также, применение *CMS* с *MODx*-концепцией позволяет осуществлять изменение функционала, с сохранением содержимого, гораздо быстрее, чем *CMS*, типа *WordPress, Joomla* и т. д. Допустим, что при изготовлении темы для WordPress задумывалась секция виджетов, например, в боковой колонке. Классическое поведение данного раздела устраивало заказчика. По прошествии некоторого времени, было выдвинуто требование, чтобы виджеты боковой колонки были разные на определённых страницах. Из-за чёткой дифференциации разделов в *CMS*, с не *MODx*-концепцией, такая передела будет проблематичной. Дело в том, что когда появилось требование разграничить информацию в боковой панели, то содержимое этой панели, в терминах *WordPress*, перестало быть виджетами, приобретя свойства отдельного типа записей. За вывод другого класса информации, в *CMS* с не *MODx*-концепцией, отвечают другие функции, в отличии от *MODx*, в котором и то, и другое, выводилось бы в переменных шаблона. После переделки виджетов в записи, в теме для *WordPress*, контент пришлось бы переносить вручную, что может стать серьёзным препятствием, если проект давно запущен.

Задача переноса сайта на другой хостинг, или домен, также облегчается при использовании *CMS*, следующих *MODx*-концепции. Дело в том, что большинство индивидуального функционала (части шаблонов, плагины вывода конкретного контента и т. д.), в *MODx*-подобных *CMS*, обычно хранятся не в виде файлов, а в виде записей в БД сайта. Хотя, файловое хранение тоже возможно. В качестве примера можно рассмотреть прайс-лист. Допустим, что для его реализации не применялось дополнительных файлов клиентской части (*CSS, JS*), весь функционал этого раздела был реализован на стороне сервера. Вывод производился с использованием имеющихся в шаблоне стилей. В этом случае, для *CMS*, типа *WordPress*, пришлось бы написать плагин, сохранив его в файл *PHP* (один из вариантов), или добавить этот функционал в тему, опять же, изменив её файлы. В случае с *MODx*, всю реализацию можно было бы произвести с использованием динамически генерируемых полей. Даже если плагин для реализации таких полей будет в виде файла, то сама реализация прайс-листа будет в виде записи полей, в соответствующей таблице БД. Благодаря отсутствию чёткой дифференциации разделов, при помощи одного и того же плагина могут быть реализованы множество функций сразу (прайс-лист, слайдер, меню, боковая колонка виджетов, список новостей и т. д.). При добавлении всех этих возможностей файловая структура сайта не обязана меняться, если только не изменять статическую клиентскую часть (например, *CSS*-файлы). Дело в том, что независимо от концепции, текстовый контент сайта, как правило, будет храниться в БД. Поэтому, её в любом случае придётся переносить. Но, при отсутствии плагинов в отдельных файлах, чаще придётся менять только БД, что облегчит перенос и резервное копирование всего сайта. Также, такой подход позволяет более тонко балансировать нагрузку между *WEB*-сервером и СУБД.

Централизованное управление группой сайтов или глобальных подразделов, также решается с использованием *MODx*-подобных *CMS*. Дело в том, что в таких системах присутствует понятие

контекста. Контекст может быть полностью (или частично) изолирован от другого. При помощи таких настроек можно реализовать многоязычность (пример частичной изоляции) или отдельные сайты (пример полной изоляции), но управление такими сайтами можно будет осуществлять в пространстве одной *CMS*, при этом создав группу «глобальных модераторов», которые смогут управлять несколькими сайтами (разделами) сразу.

Ещё одним преимуществом, также связанным с тонкой настройкой прав доступа, в *MODx*-подобных *CMS*, является полноценное разделение файловой структуры. Как уже было сказано выше, нет предопределённой дифференциации, не только для разделов сайта, но и для политик безопасности и ролей пользователей. Можно создать группу пользователей, отвечающих строго за определённый участок в файловой структуре сайта, причём этот участок может быть размещён даже на другом сервере. Правда, это может усложнить работу, при ни совсем правильной настройке прав, приводя к избыточному копированию файлов. Допустим, что есть несколько групп пользователей, каждые из которых отвечают за свои разделы сайта (предположим, что разделы – отдельные страницы), и у каждой группы имеется доступ к отдельному источнику файлов. Предположим, что на нескольких страницах имеется одинаковое изображение. В этом случае, если не предусмотреть «проброс» значений конкретных полей на разных страницах, возникает необходимость повторно загружать это изображение в каждой, из вышеупомянутых групп пользователей.

Более гибкая система кэширования, позволяет ещё больше снизить нагрузку на сервер, и как следствие – повысить скорость работы сайта на *MODx*-подобных системах. Опять же благодаря тому, что разная информация выводится одними и теми же средствами, то под каждый сниппет, плагин и т. д. в зависимости от конкретных параметров, можно настроить индивидуальное кэширование данных. Один и тот же сниппет, выводящий, например, меню сайта и список новостей, в первом случае может кэшироваться, а во втором – нет. Ведь для вывода и того и другого, не нужно использовать разные функции, и их базовые параметры не будут различны.

Более гибкая система управления настройками, в *CMS* с *MODx*-концепцией может быть реализована, за счёт того, что как было сказано выше, нет дифференциации полей (переменных шаблона). Они равнозначны, поэтому могут в любой момент быть включены разработчиком в любой из шаблонов. То же самое касается чанков и сниппетов. Например, имеется общая, для страниц определённого типа, группа виджетов. Нужно встроить её же, но в другой шаблон, с теми же значениями (содержимым), но выводящихся по-другому. При том, правила изменения этих значений, также разные для различных шаблонов. Например, в одном случае, можно изменять индивидуально, а во втором – на всех страницах сразу. Как бы не разнились правила и последовательность вывода определённых групп содержимого, их не обязательно создавать заново, как, например, типы записей в *WordPress*. Можно вызывать одни и те же поля, просто делать это в разных чанках, с различными модификаторами, или обращаться к полям через дополнительные плагины, в конкретном чанке.

Несмотря на отсутствие автоматической установки шаблонов, на *MODx*-подобных системах можно создавать удобные сайты-конструкторы. Понятно, что не сделать разделов и полей на все случаи жизни, но можно воспользоваться подходом, аналогичным тому, что применяется в тренажёрах по вёрстке. Сверстав типовые элементы, аналогичные сетке, используемой при изготовлении макетов сайтов, не давая привязку к ним, для конкретных типов и категорий, получится сайт, в котором модератор произвольно настраивает каждую секцию. В качестве опорных функций можно создать базовые, типа задания фона, ширины, и расположения конкретной секции. Понятно, что с точки зрения вёрстки и оптимизации функционала, такой сайт будет неповоротливым, его будет неудобно заполнять, но в данной ситуации, речь не идёт о реальном сайте. Такой демонстрационный проект можно использовать для уточнения требований заказчика. Создание такого «конструктора прототипов», также будет усложнено на не *MODx*-подобных *CMS*, так как заранее не известен уровень вложенности для ячеек и строк.

Приложения и примеры

В данном разделе будет описан схематичный сравнительный пример простейшего случая адаптации вёрстки к *CMS MODx* и к абстрактной стандартной *CMS*. Пример на *MODx Revolution,* хотя в редакции *Evolution* принцип аналогичен, а отличия в синтаксисе минимальны. В качестве альтерна-

тивной концепции взята абстрактная *CMS*, так как подобных *CMS* очень много, функции различны, а основные черты концепции схожи. Допустим есть простейший шаблон страницы:

```
<!DOCTYPE html>
<html lang="ru-RU">
<head>
<meta charset="utf-8" />
<title>Example</title>
</head>
<body>
<div class="content">
<h1 class="page-title">Название страницы</h1>
Текст с содержимым:
<img class="page-thumbnail" src="image.bmp" alt="text" />
</div>
</body>
</html>
```

Рассмотрим пример адаптации к *MODx Revolution*:

```
<!DOCTYPE html>
<html lang="ru-RU">
<head>
<meta charset="utf-8" />
<title>[[*pagetitle]]</title>
</head>
<body>
<div class="content">
<h1 class="page-title">[[*longtitle]]</h1>
[[*content]]
<img class="page-thumbnail" src="[[*page-thumbnail]]" alt="[[*page-thumbnail-alt-text]]" />
</div>
</body>
</html>
```

В данном случае поля *pagetitle* и *longtitle* уже имеются в *CMS*. Поля *page-thumbnail* и *page-thumbnail-alt-text*, созданы разработчиком. Как видно, их вызов происходит одинаково. В случае с не *MODx*-концепцией, за вывод этих значений отвечали бы разные функции, или определённая функция с разными входными параметрами. Названия функций и параметров были бы чётко зарезервированы, но даже в случае создания своего (произвольного) поля, например, для миниатюры страницы, нужно было бы использовать набор зарезервированных функций, для включения этих полей в панель управления сайтом, и для вывода их значений на странице. Естественно, что для разных типов содержимого, такие функции были бы различны, да и способы работы с ними пришлось бы искать в соответствующей документации, так как в не *MODx*-подобных *CMS*, не предполагается среда разработки, позволяющая управлять функционалом *CMS* через графический интерфейс.

Шаблон, адаптированный к работе с *MODx,* не содержит функций, которые выводят информацию сразу, на сторону клиента, он содержит переменные, а *MODx,* являющийся интерпретатором, вставляет на места переменных те значения, которые выбрал конечный пользователь сайта, находясь в панели управления. Саму панель управления, при этом, можно настроить под любые требования заказчика, как угодно расположив и разбив по категориям поля для ввода информации. Фрагменты шаблона (чанки), тоже могут быть выделены, как угодно, в зависимости от условий удобства при разработке. Понятное дело, что можно написать серверный код в виде сниппета, который будет выводить информацию на экран напрямую, а не возвращать результат для дальнейшей обработки, а потом, вызвать этот сниппет в определённом месте шаблона, однако, это делается не часто, и только в конкретных ситуациях. Распространённые плагины для *MODx,* например, плагин для организации поиска по сайту *SimpleSearch*, не выводит информацию на экран, как и большинство других плагинов, он отвечает только за информацию, а всё «представление» пишет разработчик, в соответствующих чанках. В какой-то степени, *MODx*-концепция напоминает подход *MVC* (*Model View Controller*).

Естественно, что в любой *CMS* можно встретить черты из обеих концепций. Но, как-правило, явно доминирует одна из них. Например, в качестве крайней степени не *MODx*-концепции, можно рассматривать конструкторы сайтов, типа *WiX,* а *MODx* – как противоположность таким решением.

Дополнительную информацию о *MODx*-концепции, можно найти в статье «Общие рекомендации для проектирования и реализации *WEB*-сайтов», в разделе «Преимущества и недостатки *MODx*».

Заключение

В итоге, можно обобщённо сказать, что *MODx*-концепция выгодна в следующих случаях:

- Разработка сайта ведётся «под ключ» и его модули не предполагается выкладывать для общего пользования.
- Структура сайта не может быть однозначно отнесена к стандартным (блог, форум, и т. д.), или там много элементов, свойства которых выбиваются из предполагаемой структуры.
- Сайт содержит множество шаблонов с дизайнами, под разные типы страниц.
- Масштабный и сложный проект, предполагающий множество разделений прав и ролей пользователей и повышенные требования к системе безопасности.
- Чётких требований к сайту нет, они постоянно меняются, разработка ведётся в режиме «экстремального программирования».
- Шаблоны для разных типов страниц сильно отличаются друг от друга.
- Дизайн сайта подходит под типичную структуру, но является слишком сложным и предполагает обширный функционал.
- Требования к интерфейсу панели управления не определены.
- Создание демонстрационной версии для построения прототипа проекта.

CMS, не работающих на *MODx*-концепции, целесообразно использовать в следующих ситуациях:

- Разработка конечного сайта не предполагается, требуется создать набор модулей, из которых будет собран готовый сайт, без участия специалистов.
- Структуру разрабатываемого сайта можно чётко отнести к определённому типу, функционал которой, полностью (или большей частью) реализован в выбранной *CMS.*
- Уровень пользования будущих модераторов сильно ниже среднего, при условии привыкания к панели управления конкретной *CMS* (сейчас, практически не актуально, так как на уровне пользователя (соц. сети, форумы и т. д.) осваивает большинство людей).
- Учебные демонстрационные цели, не связанные с программированием, а поясняющие – что такое *CMS.*

Список литературы

1. Ромашов В. CMS Drupal / Система управления содержимым сайта. — М.: Питер, 2016. — С. 533.
2. Колисниченко Д. Выбираем лучший бесплатный движок для сайта / CMS Joomla! и Drupal. — М.: БХВ-Петербург, 2012. — С. 288.
3. Официальный сайт CMS WordPress для разработчиков. — [Электронный ресурс]. URL: https://codex.wordpress.org/ (дата обращения: 21.06.2019).
4. Официальный русский сайт CMS MODx. — [Электронный ресурс]. URL: https://modx.ru/ (дата обращения: 17.04.2019).
5. Официальный русский сайт CMS WiX. — [Электронный ресурс]. URL: http://ru.wix.com/ (дата обращения: 15.03.2019).