

УДК 004.7

DOI: 10.37005/2071-9612-2020-4-25-36

ПЕРСПЕКТИВЫ РАЗВИТИЯ CMS

Гутовский Дмитрий Игоревич¹, Добрынин Владимир Николаевич²

¹Аспирант;

Государственный университет «Дубна»;

141980, Московская обл., г. Дубна, ул. Университетская, д. 19;

e-mail: mfhnb@mail.ru.

²Профессор;

Государственный университет «Дубна»;

141980, Московская обл., г. Дубна, ул. Университетская, д. 19;

e-mail: arbatsolo@yandex.ru.

В данной статье описаны перспективы развития CMS и дано краткое описание этапов становления Интернет до появления CMS. Приведён возможный вариант развития CMS по MODx-сценарию.

Ключевые слова: CMS, MODx, WEB, HTML, CSS, PHP, управление содержимым, вёрстка, Интернет, безопасность, WEB 1.0, WEB 2.0.

Для цитирования:

Гутовский, Д. И. Перспективы развития CMS / Д. И. Гутовский, В. Н. Добрынин // Системный анализ в науке и образовании: сетевое научное издание. – 2020. – № 4. – С. 25–36. – URL : <http://sanse.ru/download/412>. – DOI : 10.37005/2071-9612-2020-4-25-36.

THE PROSPECTS OF A CMS DEVELOPMENT

Gutovskiy Dmitriy¹, Dobrinin Vladimir²

¹Graduate student;

Dubna State University;

141980, Dubna, Moscow reg., Universitetskaya str., 19;

e-mail: mfhnb@mail.ru.

²Professor;

Dubna State University;

141980, Dubna, Moscow reg., Universitetskaya str., 19;

e-mail: arbatsolo@yandex.ru.

This article contains the description of the prospects of a CMS development and the short description of the stages of the Internet-formation before an appearance of the CMS. And the assumed way of a CMS development along the MODx-scenario is described in this article.

Keywords: CMS, MODx, WEB, HTML, CSS, PHP, content management, typesetting, the Internet, security, WEB 1.0, WEB 2.0.

For citation:

Gutovskiy D.I., Dobrinin V.N. The prospects of a CMS development. System Analysis in Science and Education, 2020;(4):25–36(In Russ). Available from: <http://sanse.ru/download/412>. DOI: 10.37005/2071-9612-2020-4-25-36.

Введение

CMS (Content Management System – система управления содержимым) – это программный комплекс, позволяющий управлять содержимым сайта [1]. Как-правило, *CMS* содержат ещё и компоненты, предназначенные для управления пользовательскими правами и настройками (однако, это зависит от сложности системы) [5]. Изначально, подобных систем не было. На заре появления *WEB*-технологий (начиная с 1992 года, когда язык *HTML (Hyper-Text Markup Language)* вышел в свет), никто не думал о создании сложных систем, работающих в Интернет-пространстве [5]. Интернет был весьма небыстр и дорог, а сайты служили лишь набором страниц, с включённым в них гипертекстом (текстом, со ссылками). Первые *CMS* появились с наступлением эпохи *WEB 2.0* (в конце 90-х годов 20-ого столетия) [5]. Однако, для лучшего понимания развития *WEB*, сначала необходимо чётко понять, почему Интернет развивается столь быстрыми темпами, а актуальность *WEB*-приложений растёт с каждым днём [15 – 17].

WEB-программирование является одной из важнейших и наиболее перспективных разновидностей программирования [5]. Это связано с тем, что архитектура *WEB*-сайтов и приложений имеет кроссплатформенную структуру для пользователя [5]. Пользователю не нужно задумываться о том, какое оборудование у него (компьютеры различных типов, смартфоны, планшеты и т.д.). Он не обязан использовать конкретные операционные системы (ОС). Такой подход отличается от концепции стандартных приложений, которые приходится портировать на множество различных платформ. Всё, что требуется от пользователя, это иметь стандартный *WEB*-браузер (актуальной версии) [5, 13]. Такая кроссплатформенность стала возможной, так как в 90-е годы Интернет приобрёл современные очертания, благодаря появлению языка гипертекстовой разметки (*HTML*). Позднее, к нему присоединился язык задания стилей (*CSS*), а потом, стандартизировался и язык программирования, работающий на стороне браузера [5, 13]. Изначально, таких языков было много, но начиная с конца 1-ого десятилетия 21-ого века, язык программирования *JavaScript* стал основным для *WEB*-страниц, постепенно вытесняя *Adobe Flash*, *Java-applet* и другие языки и технологии [10]. Несмотря на то, что *WEB*-страницы не стали идеально кросс-браузерными (даже к 2020-ому году), индивидуальных тонкостей вёрстки под разные браузеры осталось очень мало [5]. Сейчас (в 2020-ом), браузеры различаются эффективностью работы своих ядер (движков), которые и интерпретируют связку из *HTML + CSS + JavaScript* (далее – клиентскую часть сайта). Сами же стандарты клиентской части сайта, предназначенные для любого стандартного браузера, установились чётко. Это позволило избавиться от множества проблем несовместимости [5].

Конечно, клиентская часть *WEB* не является образцом вычислительной эффективности, и это является платой за кроссплатформенность. Поэтому, всё прикладное программное обеспечение (ПО), постепенно переходит к двум тенденциям. В первом случае, пользователю не требуется высокой вычислительной эффективности. Таких приложений большинство. Они нужны в большей части сфер человеческой деятельности и призваны решать различные повседневные задачи. К таким приложениям можно отнести различные офисные средства, а также клиентские части коммуникационного ПО, такого как социальные сети, программы для просмотра электронной почты и т.д. Несмотря на то, что для работы, например, той же электронной почты, требуются большие вычислительные мощности, (это нужно, например, для шифрования передаваемых данных), все эти тяжёлые вычисления производятся на серверах, а клиент получает только результат работы серверной части. Как и в случае с *WEB*, архитектура коммуникационного ПО является клиент-серверной, и она, в большинстве случаев, использует общепринятую модель Интернета [7].

С развитием *WEB*-технологий, отпадает необходимость писать различные версии клиентской части для коммуникационных программ, и они переходят на *WEB*-архитектуру. Классические приложения требуются в том случае, если от оборудования необходимо «выжать максимум». При таком подходе, программы оптимизируются под конкретные ОС и типы оборудования. Но эти программы, чаще всего, работают на серверах, а их интерфейс, опять же, реализован в *WEB*-архитектуре. Если речь идёт о приложениях, предназначенных для расчётов, но работающих на локальном рабочем месте, например, различные *CAD*-программы, то там используется классическая архитектура. Это происходит до тех пор, пока это приложение не перейдёт от локальной версии, к серверной, а на сторону клиента не начнут выдавать *WEB*-интерфейс.

Из вышесказанного можно сделать вывод, что *WEB*-архитектура охватывает, всё большую часть ПО, при том, что важность оно не подлежит сомнению. Трудно было бы представить современный мир без Интернета, в любой сфере человечества. Однако, кроссплатформенной является только клиентская часть, чего нельзя сказать о серверной. Далее, будут рассматриваться проблемы, которые возникают на стороне сервера (и возможные пути их решения).

Историческое развитие *WEB*-архитектуры не имело множество разветвлений (вплоть до 2005 года). Оно шло по естественному пути усложнения серверной и клиентской части. Как понятно из вышесказанного, серверная часть *WEB* появилось как закономерный ответ на быстро растущую сложность задач, возлагаемых на Интернет-технологии. В то же время, сразу было понятно, что аналогичная полная стандартизация (как в клиентской части) не будет возможна с серверной. Однако, с появлением *CMS MODx* (в 2005 году), появился альтернативный взгляд на *CMS* и логику построения серверной части *WEB*, в целом. Но в *MODx*-подходе есть один недостаток, не позволяющий ему завоевать наибольшую популярность среди *WEB*-разработчиков. Естественно, что потребность в стандартизации компонентов серверной части растёт, ввиду увеличения сложности *WEB*-приложений. Но нет точного и однозначного понимания того, как стандартизировать какие компоненты. Решением данной проблемы, как раз, может стать подход, используемый в *MODx*. Однако, обо всём по порядку [14, 16].

Концепция возникновения Интернет

Предпосылки к возникновению Интернет и особенности массового спроса на сетевые коммуникации

После окончания Второй Мировой Войны, практически сразу началась гонка вооружений. Это привело к разработке принципиально новых видов оружия и потребовало значительных вычислительных мощностей для его расчёта. Помимо вычислительной мощности, требовалось и объединение умственных ресурсов различных групп учёных и научно-исследовательских институтов. Плюс к этому, противоборствующие силы должны были быть готовы к неожиданному поражению, с выводом из строя центра управления. Таким образом, возникла необходимость, не только в объединении большого количества информационно-вычислительных комплексов, но и в децентрализации их расположения. Это привело к появлению сети *Arpanet*, ставшей прародителем Интернета. Однако, с развитием различных отраслей науки, сеть начала становиться более глобальной, выходя за рамки конкретных НИИ и стран. При этом, единого стандарта, протоколов передачи данных и т.д., всё ещё не было. Стало понятным, что дальнейшее развитие всемирной паутины неизбежно. Требовалось создание стандарта, который бы позволил использовать её в разном программном и аппаратном окружении. Началом Интернета, можно условно считать 1992 год, когда официально вышел 1-ый вариант языка *HTML* (языка гипертекстовой разметки). Это справедливо, если говорить не о развитии сети (на низком уровне), а именно о развитии *WEB* (как токового, в современном понимании). Хотя, первый *HTML*-сайт появился 6 августа 1991 года. С выходом языка *HTML*, Интернет стал по-настоящему единым (хотя, он был и до этого).

Таким образом, после появления единого Интернета, информация стала гораздо более доступной. Это связано с тем, что не было специфических требований к программной и аппаратной части. Плюс к этому, была решена задача децентрализованной работы, так как сервера были в разных географических местах. Задача объединения информационных и умственных ресурсов, соответственно, тоже решалась лёгким доступом к глобальным источникам информации. Добыча оной, ранее, отнимала большое количество времени.

Несмотря на явные позитивные проявления Интернета, с его развитием связан и ряд определённых социальных проблем. Во-первых, часто бывает так, что общество неверно трактует предназначение глобальной сети. Нужно понимать, что Интернет – всего лишь технология, позволяющая получить лёгкий доступ к различным источникам информации. Однако, никто не может гарантировать ценность и правильность этой информации. Эта проблема не нова, и она затрагивает любые способы передачи информации, начиная от самых древних. Но с увеличением количества доступной информации, увеличивается и количество дезинформации. Плюс к этому, часто может возникать ложное понятие о том, что увеличение скорости получения информации, увеличит и скорость её освоения. То есть, может возникнуть впечатление, что стало легко дойти до уровня специалиста в той или иной

области, просто найдя информацию в Интернете. Однако, этот миф разрушительен. Часто бытующее мнение, о том, что сегодня главное – не много знать, а уметь быстро взять те или иные сведения, ухудшает качество специалистов. Так как, даже при условии заучивания наизусть, информация не станет знанием, до тех пор, пока не будет понятен способ её применения.

Вообще, на начальных этапах развития Интернет, возникала проблема с его недостаточным ресурсом. Работники различных индустрий быстро почувствовали удобство *WEB* и стали пытаться возложить на него те задачи, к которым он ещё не был готов (например, идеи видеоконференций с отличным звуком и видео, для чёткой детализации, начали появляться ещё в конце 90-х годов, когда скорость сети была совершенно недостаточной для этого).

Интернет и Всемирная паутина периода 1991 – 1997

В начале 90-х годов, когда Интернет стал приобретать привычный (в современном понимании) вид, *WEB* – сайты были статическими [5]. У них не было серверной части, соответственно и не было *CMS*. Несмотря на то, что физическое расположение этих сайтов было на удалённых серверах, они имели только клиентскую часть, которая передавалась на компьютеры конечных пользователей без изменений. Сайт, который очень часто представлял из себя лишь набор страниц гипертекста (текста со ссылками), был «намертво» свёрстан. Он не имел средств для управления содержимым (без участия верстальщика) [5]. Для изменения информации на подобном сайте, приходилось изменять сам код страницы (её вёрстку). Это было неудобно, да и не позволяло сделать большинство приложений, удовлетворяющих современным требованиям. Например, нельзя было сделать клиентскую часть соц. сети или другой коммуникационной системы, используя статические страницы. Ведь в этом случае, необходимо, чтобы содержимым мог управлять любой пользователь. Типичный пользователь не станет верстать, а захочет писать на простом, понятном ему языке. При этом, *WEB*-страницы должны будут генерироваться динамически, исходя из содержимого и индивидуальных настроек системы для пользователя [4, 6, 7].

Постепенно (к концу 90-х), *WEB*-сайты начали обзаводиться серверной частью. Конечной задачей серверной части (и по сей день), является генерация клиентской части [5]. Практически любой современный *WEB*-сайт имеет серверную часть, но с точки зрения конечного пользователя, а точнее *WEB*-браузера, работающего на его компьютере, сайт остался набором *HTML*-страниц, приходящих с сервера (хоть и в более усложнённом виде) [5]. В общем случае, браузеру нет никакой разницы, написаны ли страницы вручную полностью, или сгенерированы серверной частью *WEB*-приложения. Браузер, всё равно, получит связку из *HTML + CSS + JavaScript*, не имея никакого понятия о серверной части. Клиентская часть, никак не будет зависеть от средств её получения (языки клиентской части от этого не меняются) [2]. Это позволило сохранить кроссплатформенность *WEB*, добавив возможность управления и существенно расширив спектр задач, возлагаемых на *WEB*-архитектуру и Интернет, в целом. Далее, рассмотрим подробнее историю возникновения клиентской части сайтов.

Сам по себе, язык *HTML* произошёл от *SGML (Standard Generalized Markup Language* – стандартный обобщённый язык разметки). *SGML* является наследником, разработанного в 1969 году в *IBM* языка *GML (Generalized Markup Language)*. Первая, официально вышедшая версия *HTML*, это – *HTML 2.0*. Этот язык был разработан Консорциумом Всемирной Паутины (*W3C*). Этот орган, и по сей день, является законодателем спецификации *HTML* [4, 5].

Как уже было сказано выше, клиентская часть сайта состоит из 3-х компонент, одной из которых является *CSS*. *CSS* является языком для задания стилей *WEB*-страниц [13]. Его первая версия была официально опубликована 17 декабря 1996 года. Несмотря на это, язык *CSS*, ещё долгое время (до середины нулевых годов) поддерживался не полностью в некоторых браузерах (особенно, в *Microsoft Internet Explorer*, до появления версии 7) [18]. Ранее, когда *CSS* не было, или он не поддерживался повсеместно, для достижения различных стиливых эффектов, прибегали ко всяческим *HTML*-ухищрениям. В *HTML*, даже существовали специальные тэги и атрибуты, позволяющие задавать стиливые правила, например, выравнивать элементы на странице, задавать фон и т.д. [5, 13]. Однако, такие стиливые *HTML*-конструкции были разрешены до версии *HTML 4*. Несмотря на то, что в 4-ой версии *HTML* [5, 13], ещё не было запрещено использование стиливых конструкций, этот подход являлся нежелательным. Он был оставлен в рабочем виде только для того, чтобы сайты постепенно перестраивались на новый стандарт, без резкого обрушения инфраструктуры [5, 13]. В *HTML5*, который

является актуальным на момент 2020 года, стилевые *HTML*-конструкции однозначно являются запрещёнными, а все стилевые эффекты берёт на себя *CSS* [13]. Однако, даже в эру *HTML5* и *CSS3*, всё ещё встречаются сайты и системы управления, работающие с запрещённым стилевым подходом [13].

К сожалению, использование стилевых конструкций *HTML*, встречается не только среди начинающих *WEB*-разработчиков, но и на сайтах крупных компаний. Это несмотря на то, что такой подход не соответствует спецификации современного *HTML*. Хотя, с момента запрета стилевых конструкций *HTML* прошло более 10 лет, *WEB*-инфраструктура чрезвычайно велика, и не все приложения успели адаптироваться к новым реалиям [5]. Плюс к этому, устаревший стиливой стандарт несёт целый ряд проблем. Одной из основных, является нарушение семантических правил *HTML* [5]. Ранее, во времена зарождения *WEB*, о семантике *HTML*-тэгов особо не задумывались. Это связано с тем, что не было такого большого количества сайтов, поисковых систем, специализированных программ для *WEB*. Дифференциация компонентов сайта, также была менее выраженной, по причине большей примитивности тогдашних систем [5]. Сейчас, одним из основных критериев «хорошей» вёрстки является её правильная семантика. Недостаточно того, чтобы она имела корректный внешний вид и адаптировалась под различные штатные ситуации. Требование к соблюдению семантики вызвано тем, что помимо данных, отображаемых пользователю, тэги *HTML*-документа содержат множество мета-данных, несущих скрытую, но не менее важную информацию [13]. Примером такой семантической нагрузки тэгов может служить обычный документ в известной программе *Microsoft Word*. Если, например, заголовки не сделать с верной опцией, а просто выделить текст, допустим, жирным начертанием, то *Word* не поймёт, что это начало главы и не внесёт этот «ложный заголовок» в авто-оглавление. Аналогичная ситуация и с *WEB*-страницами. Скрытые мета-данные используются, начиная от поисковых систем, заканчивая специализированными браузерами для слабовидящих и т.д. Одним из ярчайших примеров того, как пытались реализовать стили, без нормальной поддержки *CSS*, в конце 90-х годов (во времена *HTML 3*), это – табличная вёрстка [5]. Сейчас, она запрещена в явном виде, но использование её свойств, при помощи *CSS* – это правильный подход, если того требует ситуация, и это удобнее в конкретном дизайне.

Программный компонент клиентской части сайта, тоже не всегда был стандартизован и представлен языком *JavaScript* [4, 5, 13]. Например, в конце 90-х (во времена актуальности *Internet Explorer 5*), были весьма популярными такие технологии и языки как *Visual-Basic*, *Java-applet* и *Adobe Flash*. Хотя, *Adobe Flash* оставался популярным, вплоть до появления *HTML5*. В 2020 году, *Adobe Flash*, практически потерял свою актуальность из-за множества недостатков данной технологии, особенно с точки зрения *SEO* и безопасности, ну а *Visual-Basic* и *Java-applet*, давно перестали использовать в клиентской части сайта (ещё в конце нулевых) [4, 5, 13]. В эру *HTML5* и *CSS3*, *JavaScript* является безальтернативным языком программирования для клиентской части *WEB*-сайтов. Помимо этого, вся клиентская часть, достаточно строго стандартизована, и её кросс-браузерность, близка к абсолютной. Хотя, «разногласия» между разными браузерами, всё ещё встречаются, как и необходимость нарушения *HTML*-семантики, в исключительных случаях, даже при наличии *CSS3* [4, 5, 13].

Но даже в конце 90-х, *WEB*-архитектура была весьма перспективным и кроссплатформенным направлением. Уже тогда было понятно, что сайт может быть, нечто более сложной системой, чем простейший набор статических страниц. И уже в 1998 году, многие крупные *WEB*-сайты имели не только клиентскую, но и серверную часть. Человечество, постепенно, приближалось к эпохе *WEB 2.0* [4, 5].

Эпоха *WEB 2.0*

Технологические достижения в области массовой доступности к информационному пространству на основе Интернет и инструментов управления контентом (CMS)

В современном обществе, в условиях молниеносного развития Интернета, *WEB*-приложения становились всё сложнее. Это привело человечество к эпохе, когда управлять сайтом, в общем случае, может любой, без специальной подготовки. Эта эпоха провозглашена, как эпоха «*WEB 2.0*» [5].

В эпоху *WEB 2.0* (в которой мы находимся и в 2020 году), подавляющее большинство *WEB*-сайтов имеют функционал, который позволяет обыкновенным пользователям всячески управлять его содержимым. При этом, не прибегая к специфическим приёмам, специальным программам и не имея квалифицированной подготовки [5]. Системы, позволяющие пользователям управлять содержимым сайтов, были названы *CMS* (*Content Management System* – система управления содержимым). Первые *CMS*, появившиеся в конце 90-х годов, были узкоспециализированными. Например, появившаяся в 1999 году *CMS WordPress*, была предназначена только для построения блогов, хотя позже, стала развиваться и как система для создания форумов [1, 2, 12, 15].

Клиентская часть *WEB*-приложений, со времён эпохи *WEB 1.0*, продолжает развиваться, и никаких резких изменений в ней не происходит [5; 13]. Концептуально, *HTML* не меняется с самого своего начала, то есть, с 1992 года. *CSS*, тоже развивается постепенно, по возможности, не заменяя старые технологии, а добавляя новые. При этом, изменения языка программирования для работы в браузерах, не сильно затронули сайты. Это связано с тем, что во времена, когда этих языков было много, плагинов, работающих на стороне клиента, было мало, и они успешно перешли на *JS*. Благодаря постепенному развитию клиентской части, которая изначально появилась как стандарт для *WEB*, сайты сохраняют кроссплатформенность, и концепции стандартов не меняются [5].

Но, как уже было сказано выше, эпоха *WEB 2.0* подразумевает возможность управления содержимым. Для этого нужна серверная часть, которая будет динамически генерировать *WEB*-страницы, в зависимости от требований пользователя и функционала *WEB*-приложения. Страницы больше не передаются «как-есть», и задача серверной части (на основе какой технологии она бы не была сделана), это – генерация страниц, представляющих из себя клиентскую часть. Серверная часть не имеет стандарта, она может быть написана с применением множества различных технологий и языков программирования. Конечно, так как клиентская часть стандартна, то в итоге, браузеру всё равно, как была получена страница. Работа сайта на стороне клиента, в общем случае, не зависит от того, какие серверные приложения лежат в основе конкретного сайта [5].

Любой *WEB*-сайт, вне зависимости от его направленности и технологии серверной части, имеет *CMS*, если есть хоть какая-то возможность управлять содержимым, не переделывая вёрстку напрямую [5]. *CMS* могут быть написаны под конкретный проект, если он весьма крупный (как Яндекс, Google и им подобные), или имеет узко-специфический функционал. Однако, большинство средне-статистических проектов (сайтов различных (даже крупных) компаний, сайтов гос. учреждений и т.д.) имеют в своей основе какую-то готовую *CMS*. Системы подобного рода разрабатываются в различных странах мира, в том числе и в России, например, *Bitrix* (от компании «1С»), *HostCMS* (от компании «Хостмэйк»), *NetCAT* (от компании «НетКэт») и т.д. Существует ошибочное мнение, что если сайт основан не на общеизвестной *CMS* (в простонародье «движке»), то он не имеет *CMS*. Однако, это не так – по определению [1, 5]. Например, известная российская соц. сеть ВКонтакте имеет *CMS*, которая была написана специально для неё, на языке *PHP* [1, 2, 4]. Как уже было сказано ранее, без *CMS* существуют только статические сайты, которых практически нет в эпоху *WEB 2.0* (если только это не чистая визитка). Конечно, даже в 2020-ом году, остались статические *WEB*-сайты, но это – скорее исключение, чем правило.

В данную эпоху, автоматически решилось (по большей части) проблема копирования сайта. Так как сайт более не является набором статических страниц, а браузер читает только клиентскую часть, то нельзя полностью скопировать *WEB*-сайт, не получив доступ к серверу. То, что доступно клиенту, как и в эпоху *WEB 1.0*, это – только сами страницы. Но в отличие от статического сайта, в котором вся вёрстка была «на ладони», страницы современного сайта могут быть сгенерированы под огромное множество различных ситуаций. Даже если, гипотетически, пройти по всем *WEB*-страницам, доступным в конкретный момент времени, для определённой группы пользователей (которые будут пытаться скопировать клиентскую часть), не факт, что все опции включены, и вся возможная вёрстка конкретного сайта передастся на сторону клиента. С точки зрения теории, механизм копирования клиентской части *WEB*-сайта не изменился, со времён появления *HTML*. Однако, нарастающая сложность *WEB*-сайтов, в том числе, приводит к вариативности и усложнению клиентской части. А это, может делать копирование оной, просто нецелесообразным [5]. Плюс к этому, современные *CMS*, часто поддерживают не только средства, связанные напрямую с управлением сайтом, но и множество приёмов «запутывания» конечного кода [14]. Хотя, сжать исходный код перед выкладыванием на сервер, можно было и раньше, но сам код был легче, не было столько функций и различных метаданных, используемых для работы этих функций. Соответственно, и копирование было легче. Конечно,

но, переход к эпохе *WEB 2.0*, не имеет прямого влияния на копирование клиентской части сайта, но резкое снижение целесообразности такого акта, произошло вместе с появлением серверной части сайта. Сайты стали настолько сложны, что легче создать верстку самостоятельно, взяв дизайн искомого сайта за основу [5, 10, 13].

Потребности развития *CMS* в условиях становления информационного общества

CMS, как, впрочем, и все остальные компоненты серверной части *WEB*, непрерывно и стремительно развиваются. Однако, именно развитие систем управления имеет наиболее выраженное влияние на *WEB*-индустрию, в целом. Во-первых, именно с ними контактирует пользователь *WEB*-приложения, а значит, возникает проблема в необходимости привыкания к новым реалиям, у тех людей, для которых это не является профессией. Во-вторых, *CMS* очень много (гораздо больше, чем языков для их написания), и одной из глобальных задач в области развития *CMS*, является их классификация и унификация [16].

С одной стороны, многообразие серверных технологий даёт возможность для выбора более оптимальных из них, и как следствие – шанс более эффективной разработки приложений подобного рода. Однако, это очень часто может послужить снижением эффективности, так как та или иная технология может быть выбрана не по причине оптимальности для конкретного класса задач, а по причине известности для конкретной группы разработчиков. Языков программирования, используемых для написания серверной части *WEB*, не так уж и много (их количество измеряется десятками). Плюс к этому, у них много схожего (имеются в виду те языки, которые более-менее актуальны и популярны). Их можно, относительно хорошо выучить, сделав упор на самый часто-применяемый из них. Но на основе этих языков, создано множество программ и технологий, которых уже тысячи, и выучивание всего этого многообразия, представляется гораздо менее реалистичным. Эта проблема очень сильно касается, именно, *CMS* [11].

CMS могут иметь абсолютно различную архитектуру, как с точки зрения своего ядра, так и с точки зрения *API* (*Application Programming Interface* – интерфейса прикладного программирования). Разнообразие *CMS*, с точки зрения архитектуры их ядер и языков программирования, на которых они написаны, повышает эффективность разработки сайтов. Так как увеличивается разнообразие алгоритмов работы с различными типами данных, что позволяет выбрать те, которые наиболее эффективны из них. В свою очередь, разнообразие *API*, напротив, может резко понизить эффективность разработки модулей для разных *CMS*, и как следствие, для основанных на них сайтов [3].

Понятно, что изучить все *CMS* невозможно (их тысячи), но и отказываться от существующих систем (или от разработки новых) – не самое лучшее решение. У каждой *CMS* есть свои преимущества и недостатки... Основная проблема разнообразия *CMS* заключается в том, что на их *API* нет абсолютно никакого стандарта [1; 2; 3; 6]. Разработчик, например, для *CMS WordPress* не сможет создать шаблон для *Joomla* (и наоборот). При этом, если не брать в расчёт знания клиентской части (которыми должны обладать *WEB*-разработчики), и базовые навыки какого-то серверного языка (в случае с *WordPress* и *Joomla* – это *PHP*), то эти группы разработчиков не имеют ничего общего. При этом, ранга значимости между ними нет, так как нельзя однозначно сказать, какая из этих *CMS* будет востребована больше, в очередном проекте.

API большинства стандартных *CMS* имеют целый ряд проблем, сильно снижающих эффективность разработки *WEB*-сайтов и приложений на их основе. Среди этих проблем, можно явно выделить следующие:

Сущности, которыми апеллируют стандартные *CMS*, являются сложносоставными [1, 2, 6], следовательно, они часто могут преумножаться без явной необходимости. Например, есть некоторая сущность, под названием «объявление», а есть «статья». И та, и другая, состоят из примитивных элементов, таких как название, описание и т.д., а вообще, всё это – текст. Но в стандартном подходе, эти 2 сущности будут чётко разделены, и за работу с ними будут отвечать разные функции *API* в конкретной *CMS*. Это сильно усложнит ядро самой системы и коды модулей, разрабатываемых под неё;

Слабая преемственность между версиями одной *CMS* – тоже частая проблема [1, 2, 6]. Из-за сложного *API* с избыточным количеством функций, часто приходится модернизировать это *API*, удаляя и добавляя новые сущности, функции для работы с ними, или просто изменяя часть *API* для работы с конкретным набором функционала. Соответственно, это мешает обновлять *CMS*, так как многие модули, написанные под старые версии, не будут корректно (или вообще не будут) работать в новой версии *CMS*;

Проблемы с безопасностью [6] у многих *CMS* возникают из-за вышеупомянутых проблем с обновлениями (есть известные прорехи какой-нибудь *CMS*, а обновить на исправленную версию не получается, так как большая часть сайта может перестать работать). Однако, есть и тот факт, что из-за неповоротливого *API*, во многих *CMS* нельзя штатно зашифровать их характерные черты, например, пути расположения модулей или специфические названия классов, формируемых редакторами содержимого и т.д. Всё это, приводит к рассекречиванию модели (а иногда и версии *CMS*). Что, в свою очередь, даёт возможность потенциальному злоумышленнику узнать сведения о том, какая серверная часть у конкретного *WEB*-приложения. А это увеличивает шанс поразить сайт, используя известные уязвимости для *CMS*, на которой он базируется [6].

Усложнённая разработка модулей под стандартные *CMS* связана с большим количеством функций в *API*. Даже если речь не идёт об изучении множества систем, а требуется разработка под конкретную, стандартный подход, используемый в подавляющем большинстве современных *CMS* (на момент 2020 года), сильно усложняет способы разработки, как самих модулей, так и ядра *CMS* [8, 9].

Конечно, вышеприведённый список не объясняет всех проблем. Да и у тех, что есть в списке, нет подробного объяснения. Однако, это и не сравнительный анализ конкретных групп *CMS*, а описание недостатков стандартного подхода, чтобы далее показать попытки, предпринимаемые человечеством для решения этих проблем.

Одной из наиболее ярких и популярных систем, где все эти проблемы решаются достаточно легко, является *MODx* [14]. *API* данной *CMS* имеет кардинально другой подход, но систем с аналогичным устройством, гораздо меньше, чем стандартных. Этот факт связан с невозможностью автоматической установки шаблонов на подобные системы. Подробнее об этом, будет пояснено в следующем разделе.

Все *CMS*, вне зависимости от их направленности и языка программирования, используемого для их разработки, могут быть разделены на 2 группы. Одна относится к *MODx*-концепции, а другая – к не *MODx*-концепции. Естественно, что такое разделение является условным, и черты обеих концепций могут, в той или иной степени, присутствовать в различных *CMS*. Однако, «перевес» в ту или иную сторону, всё равно виден.

Большинство современных *CMS* не имеют абсолютно чёткой направленности. Они могут совмещать в себе несколько «родственных» структур сайта, например, блоги и форумы. Однако, их сложносоставной *API* не даёт далеко отойти от той структуры, для которой изначально была предназначена данная система. Конечно, делать Интернет-магазин на *CMS*, предназначенной для создания соц. сети – не самое рациональное решение. Но несмотря на это, у любых *CMS* могут быть специфические особенности, позволяющие наиболее эффективно работать с конкретными типами данных. В добавок ко всему, написанные под конкретную систему модули, могут сильно облегчить разработку потенциального проекта на чужеродной *CMS*. Однако, при сложности *API* у большинства систем управления, создание сущностей, не заложенных в систему изначально, может быть настолько затратным, что сведёт на нет все преимущества от конкретных модулей ядра или специфических дополнений [1, 2, 3].

Границы между *CMS* разной направленности, так и не стёрты в стандартном подходе, однако, эти проблемы были успешно решены в *CMS MODx*, и с того момента, впервые появилась по-настоящему универсальная система. Однако, такие универсальные системы, так или иначе являются аналогами *MODx*, повторяя её логику. Проблемы плохой универсальности у стандартных *CMS*, не решены до сих пор (2020 год) [1, 3, 7, 8]. Можно было бы отказаться от традиционного подхода построения *API* у *CMS*, полностью перейдя на *MODx*-концепцию, но всё упирается в 1 явный недостаток, подробнее о котором далее.

Методология и технологии *MODx* – ответ на потребности эффективного управления контентом массовым потребителем

В 2005 году, появилась *CMS*, которая имела кардинально иной подход к логике работы серверной части *WEB*. Эта *CMS* называется *MODx*. *MODx* начала создаваться разработчиками Рэймондом Ирвингом (англ. *Raymond Irving*) и Райаном Трашом (англ. *Ryan Thrash*), которые начали работу над этим проектом, ещё в 2004 году. Однако, изначально, *MODx* вышла как надстройка над *CMS Etomite*, но через год отделилась и стала отдельной самостоятельной системой. *Etomite*, в отличии от *MODx* была закрытой системой, и разработчики *Etomite* не одобряли начинаний Рэймонда и Райна [14, 16].

Сначала, проект *MODx CMS* имел одну ветку, которая называлась *MODx Evolution* [14]. Но через 5 лет (в 2010 году), появляется *MODx Revolution*, которая была полностью переписана, с точки зрения кода, но унаследовала концепцию. *MODx Revolution* является преемницей *Evolution*, с точки зрения логики *API* и концепции разработки модулей под неё, однако, с точки зрения ядра – это совершенно другая система [14]. Несмотря на это, большинство модулей, написанных для *Evolution*, легко были перенесены на *Revolution*. Это связано с тем, что *API MODx*-подобных систем позволяют легко разрабатывать модули под них и переносить, даже на другие системы с аналогичной концепцией (подробнее об этом, будет пояснено далее). В 2020 году, по-прежнему, сохранены и поддерживаются обе ветки проекта *MODx*, однако, наиболее популярной является *MODx Revolution* [14, 16]. Далее, не будет приводиться конкретики разделения на эти ветви, так как то, что написано в данной работе, справедливо для обеих веток *MODx*, да и в большинстве случаев, для других *CMS* подобного рода.

MODx относится не просто к *CMS*, она является ещё и *CMF* (*Content Management Framework* – система, позволяющая создавать собственные *CMS*). *MODx* является универсальной системой управления, для создания *WEB*-приложений любой сложности и направленности. Основным принципом *MODx*-концепции является принцип атомарности. Сущности, с которыми работают в рамках *MODx API*, являются базовыми, а не составными, как заведено в стандартных *CMS*. Из базовых сущностей, таких как текст, изображение, число и т.д., можно собрать любую составную сущность, такую как «товар», «урок», «отзыв» и т.д. Естественно, что в *MODx API* нет специфического набора функций, отвечающих за каждую сложносоставную сущность, но есть простой и понятный интерфейс, позволяющий быстро создать набор сущностей базовых типов и из этих примитивов собрать любую комбинацию [14, 16].

Всё базируется на простейших объектах, одним из которых является «переменная шаблона» [14; 16]. Как и в стандартных языках программирования, переменная шаблона в *MODx* имеет название и тип. Значение этой переменной подставляется при интерпретации шаблона, во время генерации *WEB*-страниц, на те места, где она вызывается. Разработчику не требуется изучать множество документации со сложным набором индивидуальных функций под каждый тип содержимого. Ему требуется только выучить элементарный синтаксис по работе с переменными шаблона, и некоторыми другими объектами, присущими *MODx*-подобным системам [14, 16]. Появившаяся в 2005 году, *MODx* задала кардинально другой подход к созданию *WEB*-приложений, в котором нет избыточного кодирования, в котором не нужно запоминать *API* каждой конкретной *CMS*, и где можно легко портировать, практически любые шаблоны и модули с одной *CMS* на другую, если она соответствует *MODx*-концепции. Вообще, в *MODx*-концепции можно выделить 9 основных принципов:

1. Атомарность сущностей вывода.
2. Атомарность сущностей политики.
3. Адаптивность.
4. Преемственность.
5. Независимость функционала от интерфейса.
6. Независимость от адресов файловой системы сайта.
7. Функциональная расширяемость.
8. Равносильность (панели управления сайтом и «пользовательских» страниц).
9. Декомпозиция контекстов.

Несмотря на 9 принципов, главными являются 1-ые 2, так как при их соблюдении, остальные 7 вытекают автоматически. Их можно соблюсти, довольно легко, а значит, если первые 2 принципа соблюдены, то *CMS*, скорее всего, будет относиться к *MODx*-подобным.

Проблемы развития и пути решения *MODx*

Из-за невозможности автоматической установки шаблона, *MODx*-концепция – не самая популярная. Но, благодаря приведённым выше принципам, разработка под такие системы существенно легче, чем под стандартные [14]. Можно провести аналогию с графическим растровым редактором *Paint*. Представьте, что инструменты для рисования (кисти, распылители, карандаш, линии и т.д.), будут привязаны к цветам. Допустим, у нас будет синий карандаш и красная кисть, таким образом, нарисовать произвольную картину будет существенно сложнее. Аналогичная ситуация, в общем случае, складывается в стандартных *CMS*. Основные недостатки не *MODx*-концепции возникают из-за смешения логики приложения на разных уровнях и сильной связности компонентов. *MODx*-подобные системы лишены всех этих недостатков. Единственный недостаток *MODx*-концепции можно описать так, что если приложение создаётся не «под ключ», а делаются «кубики конструктора», из которых будет собираться сайт, человеком, не имеющем отношения к *WEB*-разработке, то *MODx* – не лучший выбор [14].

Одной из основных подзадач, в развитии *MODx*-концепции, является создание инструмента, позволяющего выполнять автоматическую установку шаблонов на *MODx* (и ей подобные *CMS*). В этом случае, *MODx*-концепция имеет все шансы стать единственной в данном классе приложений, а иная не будет иметь никакого смысла. Ведь во всём остальном, стандартные *CMS* уступают в плане эффективности разработки модулей под них. Естественно, что в данном случае речь идёт именно об *API CMS*, но не об их ядрах. Особенности ядра *CMS* индивидуальны у каждой системы, что даёт ей эффективность в определённом классе задач, но если все *CMS* будут иметь *MODx*-подобный *API*, то это никак не скажется на эффективности их ядер, а разработка приложений на их основе, облегчится в разы.

Речь, ни в коем случае, не идёт об ограничении разнообразия *CMS*, упразднении различных языков программирования серверной части *WEB*, или о строгой стандартизации оной. Задача заключается лишь в стандартизации *API CMS*, так как для серверной части, полная стандартизация невозможна. Для клиентской части, полную стандартизацию удалось реализовать, так как в большинстве случаев, при работе приложений подобного рода, на стороне клиента не происходит тяжёлых вычислений, требующих тонкой оптимизации. На стороне сервера же, происходят основные ресурсоёмкие действия, которым нужна высокая вычислительная эффективность, поэтому, стандартизация языков программирования и архитектур для всех ядер *CMS* – не лучший выбор, а стандартизация *API* не даст снижений производительности и окажет лишь положительное влияние на всю *WEB*-индустрию.

Заключение

Подводя итог, можно сказать, что закономерным путём развития *CMS* станет стандартизация их *API*, путём создания нового стандартного языка для *API* всех *CMS*. Для будущего стандартного языка *API*, логично выбрать подход, используемый в *MODx*. Благодаря целому ряду преимуществ, связанных с атомарностью сущностей, работающих в рамках подобных систем, этот подход – наиболее подходящий для данной задачи. Однако, помимо создания и описания самого языка, следует ряд задач, подлежащих обязательному выполнению (хотя бы частично). Во-первых, требуется создать приложение, лишаящее *MODx* её единственного недостатка (отсутствия автоматической установки шаблонов). Во-вторых, требуется создать систему трансформации шаблонов, которая бы переводила их из старого специфического *API* конкретной *CMS*, в новый стандартный. Безусловно, что из-за большого разнообразия различных не *MODx*-подобных *CMS* с их специфическими *API*, реализовать такую трансформационную систему будет невозможно в рамках одной небольшой группы разработчиков, работающих с конкретной *CMS*. Поэтому, предполагается модульная архитектура, и система будет постепенно расширяться, пополняя свой банк *API*, до тех пор, пока все *CMS* не перейдут на новый стандарт. Однако, расширение системы будет происходить, уже силами разработчиков, связанных с конкретными *CMS*. Без создания подобной системы трансформации, даже в случае успеш-

ной реализации основной задачи стандартизации, будет гораздо меньше шансов на скорый выход нового стандарта в свет. Ведь в этом случае, не будет возможности постепенного перехода на него. Несмотря на сложность поставленной задачи, в случае успешной реализации, рано или поздно, переход на новый стандарт произойдёт, так как это принесёт целый ряд преимуществ:

CMS будет выбираться из условий эффективности для конкретного класса задач, а не по принципу известности для определённой группы разработчиков;

Можно будет без труда сменить *CMS*, перенеся весь функционал, настройки и пользовательские данные, если это необходимо;

Можно будет объединять результаты наработок из разных проектов и разработчиков модулей для разных систем (даже при условии отличия языков программирования, на которых написаны эти *CMS*);

Снижения разнообразия и эффективности серверных решений различных архитектур не произойдёт, так как стандартизируется только *API CMS*;

Конечно, это – далеко не все преимущества, которые будут в случае появления нового стандарта, а актуальность данной проблемы вытекает из актуальности *WEB* в целом, которая, на момент написания данной статьи, говорит сама за себя.

Список литературы

1. Левин, А. Создаем сайт быстро и качественно / А. Левин. – Питер, 2011. – 240 с.
2. Горнаков, С. Г. Осваиваем популярные системы управления сайтом / С. Г. Горнаков. – ДМК-Пресс, 2009. – 333 с.
3. Горнаков, С. Г. Создание сайтов на основе системы UMI CMS / С. Г. Горнаков. – Эксмо, 2009. – 240 с.
4. Дари, К. AJAX и PHP. Разработка динамических веб-приложений / Кристиан Дари, Богдан Бринзаре. – Символ-Плюс, 2009. – 336 с.
5. Дронов, В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов / В. А. Дронов. – СПб. : БХВ-Петербург, 2011. – 416 с.
6. Рамел, Д. Joomla! для профессионалов / Дэн Рамел. – ИД «Вильямс», 2014. – 441 с.
7. Черепанова, И. uCoz. Создание сайтов / И. Черепанова. – 3-е изд. – Эксмо, 2011. – 544 с.
8. Колисниченко, Д. Выбираем лучший бесплатный движок для сайта. CMS Joomla! и Drupal / Денис Колисниченко. – М. : БХВ-Петербург. – 2010. – 288 с.
9. Декстер, М. Joomla!: программирование / Марк Декстер, Луис Лэндри. – ИД «Вильямс», 2013. – 592 с.
10. Мейер, Э. CSS. Каскадные таблицы стилей. Карманный справочник / Э. Мейер. – 4-е изд. – ИД «Вильямс», 2016. – 288 с.
11. Ромашов, В. CMS Drupal. Система управления содержимым сайта / Виктор Ромашов. – М. : Питер, 2016. – 533 с.
12. Уильямс, Б. WordPress для профессионалов. Разработка и дизайн сайтов / Б. Уильямс, Д. Дэмстра, Х. Стэрн – Питер, 2014. – 461 с.
13. Эндрю, Р. CSS: 100 и 1 совет / Рейчел Эндрю. – 3-е изд. СПб: Символ-Плюс, 2010. – 336 с.
14. Шпак, Ю. А. Web-разработка средствами MODx / Ю. А. Шпак. – МК-Пресс, Корона-Век, 2012. – 400 с.
15. WordPress.org : [Официальный сайт CMS WordPress для разработчиков]. – URL : <https://codex.wordpress.org/> (дата обращения: 21.12.2016).
16. MODX.RU : [Официальный русский сайт CMS MODx]. – URL : <https://modx.ru/> (дата обращения: 17.01.2017).

17. Wix.com : [Официальный русский сайт CMS WiX] / Wix.com, Inc. – URL : <http://ru.wix.com/> (дата обращения: 10.03.2017).
18. Microsoft : [Официальный сайт компании Microsoft]. – URL : <https://microsoft.com/> (дата обращения: 21.05.2019).