Введение

В курсе “Веб-разработчик. Core 2.0.” мы рассмотрим все технологии, которые необходимы для создания современного сайта.

Сразу приготовьтесь, что это довольно долгий путь. Чтобы стать мастером, хорошим специалистом - минимум нужно 2 года. Но это прежде всего зависит от интенсивности обучения. Рекомендуем делать все указанные упражнения. Также будет плюсом, если Вы параллельно с прохождением курса будете создавать небольшой проект для своих нужд. Например, ведение дневника.

По большому счету, данный курс сэкономит Вам массу времени, т.к. основной упор будет направлен не на изучение всех спецификаций по языкам программирования, а на практическое использование технологий - где брать нужную информацию, как решать задачу. Это не заменяет чтение классических книг по программированию, а скорее дает правильное направление в плане вашего развития как программиста.

Самый важный момент, который Вы должны запомнить и о чем не пишут в книгах по программированию - необходимо ориентироваться не на функции и технологии, а на потребности конечного потребителя, клиента. Выбирайте решения, которые максимально быстро и недорого решают задачу клиента, а не те решения, в которых можно задействовать крутые технологии (которые обычно не вполне освоены Вами и Вы смутно представляете их преимущества, например, Angular JS). Вы должны четко понимать, зачем это все делаете, какой экономический смысл Вашей деятельности, **какую пользу** Вы приносите клиенту. В этом курсе мы будем использовать только те технологии, которые необходимы для создания таких приложений.

Пару слов хотелось бы сказать о карьерной лестнице программиста. Тут есть несколько вариантов:

1. Путь специалиста. Вы проходите последовательно ступени стажер, junior, middle, senior, lead developer. Определите (хотя бы субъективно) на каком уровне Вы находитесь. И работайте над теми навыками, которые позволят Вам перейти на ступень выше.
2. Путь предпринимателя-специалиста. То есть Вы работаете на себя. Вероятнее всего Вы начнете как одиночка-фрилансер. В этом случае, на мой взгляд, необходимо очень хорошо знать один полный профиль разработки. То есть Вы владеете одним стеком разработки. В нашем случае это ASP.NET, SQL Server, jQuery, Bootstrap. Поэтому не тратьте силы на изучение нескольких параллельных языков (ASP.NET и PHP), считая,

что так Вы больше найдете клиентов. У меня всегда вызывают подозрения “многостаночники” - на поверку оказывается, что они знают всего понемногу, поверхностно.

Основные принципы разработки

Очень коротко пройдемся по принципам разработки. В процессе разработки Вы сами сможете их дополнить своими собственными принципами. Вот они:

1. *Боритесь со сложностью в проекте, стремитесь к простоте.* Если можно сделать проще, то почему бы так и не делать? Если решение сложное, то его очень сложно удержать в голове - необходимо разбить на небольшие подзадачи. С простотой граничит понятие "скорость". Простота и скорость - это образующие принципы и на их основе строится все остальное. Чем сложнее решение, тем сложнее будет разобраться другому программисту, необходимо писать больше комментариев, и больше будет потенциальных ошибок.. Это тоже прямая зависимость от простоты - чем проще, тем сложнее допустить ошибку.

2. *Откажитесь от магии*, от перекладывания ответственности на систему - глюки, сбои, проблемы хостера и т.д. Вы должны четко понимать, что сбои в работе программы просто так не возникают, скорее это ошибка кода (с вероятностью 90%), а не ошибка системы. Это очень важно, всегда исправляйте ошибки. Например, если в коде появилась какая-то непонятная область, то никогда не пропускайте ее - разберитесь с тем, как она работает. Копайте до конца. Если не понимаете, как что-то работает в системе - спросите у коллег, не пускайте дело на самотек. Это избавит от проблем и ошибок в работе программы.

Также изучите все нюансы технологии. Чем больше Вы их будете знать, тем быстрее будете находить те места, в которых может возникнуть ошибка при той или иной ситуации.

3. *Не повторяйтесь.*. Не пишите один и тот же код несколько раз. По мере того как Вы растете как программист, растет и ваша кодовая база. Появляются свои библиотеки кода, которые можно использовать в разных проектах. Когда делаете какую-то функциональность (отправка СМС, выход с сайта и т.д.) - сразу подумайте, будете ли Вы ее использовать в будущем в других проектах. Если ответ - да, то оформляйте функциональность в виде законченного блока (класса), который можно использовать, просто подключив его к проекту.

Задание 1. Зайдите на любой крупный сайт и составьте список повторяющейся функциональности. Должно быть не менее 20 элементов.

Например:

1) Отправка СМС

2) Регистрация и так далее

4. *Мыслите от клиента.* Вы должны четко понимать, зачем клиенту необходим этот продукт. Чем лучше Вы понимаете предметную область, бизнес-логику, смысловое значение продукта - тем более лучшие решения Вы будете предлагать клиенту для решения его потребности. Не ленитесь изучать специфику бизнеса клиента. Задавайте вопросы по его тематике. Также необходимо учитывать, что клиент может и не иметь технического образования, поэтому необходимо его понимать при ведении диалога. То есть клиенту могут быть незнакомы такие понятия как: база данных, первичный ключ, таблица N и т.д. По этой причине придерживайтесь того языка, который понятен клиенту. Соответственно, свои требования и предложения озвучивайте в терминах понятных клиенту. В этом плане полезно использовать анкетирование. При работе с клиентом все вопросы и ответы оформляются в виде анкеты. Это выглядит проще и понятней. Также можно дополнительно создавать прототипы (схематичные макеты). То есть при диалоге можно быстро составить прототип и после его положительного утверждения уже дальше реализовывать полноценный проект.

Проводя диалог с клиентом, Вы должны четко понимать и разбираться в той предметной области, для которой производится разработка программы. Данное взаимодействие с клиентом в дальнейшем может помочь Вам в развитии профессиональных навыков, знаний в той или иной сфере. А это уже открывает путь к профессии бизнес-аналитика.

Разрабатывая программу для клиента, Вы должны понимать, что оказываете услугу, а поэтому должны оказывать ее качественно: это не только написание кода, но и общение с клиентом, представление отчетов по работе. Все эти моменты повышают Вашу ценность как специалиста.

Задание 2. Напишите, что, на ваш взгляд, больше всего ценит клиент в сотрудничестве с командой разработки.

5. *Не нужно быть оптимистом при оценке проекта*. Когда оцениваете задачу, во-первых, не торопитесь дать необдуманную оценку. Разбейте задачу на составляющие. Подумайте о возможных узких местах. Только после этого давайте оценку задачи.

Есть такой прием, как фокус-фактур оценки. Т.е. берется отношение оценочного времени на время затраченное на выполнение проекта, и затем полученный коэффициент умножается на оценочной время для другой задачи.

Никогда не нужно идти у поводу у клиента в плане оценки, т.к. он мыслит другими категориями. Например, клиент предлагает сделать невозможный функционал или пытается уменьшить срок на разработку, но это чревато приведению к ухудшению кода, к срыву задачи, или даже всего проекта. Всегда считайте оценку проекта, отталкиваясь от его сложности, а не пожеланий клиента.

6. *Сервис, а не просто программирование*. Мы этого уже несколько раз коснулись, но не лишним будет это подчеркнуть - мы работаем для клиентов, а не просто пишем код для каталога интернет-магазина! Что такое сервис? Сервис - это закрытие проблемы клиента быстро, качественно и без особых трудностей.

Задание 3. Напишите, что конкретно Вы можете делать для повышения сервиса (не менее 10 пунктов).

Задание 4 (для продвинутых). Напишите особенности сервиса в кафе. В чем он выражается?

Cheat-коды, или Что поможет упростить разработку

1. Необязательно знать какой-то синтаксис, код какой-то задачи и т.д. Для действий, с которыми незнакомы, можно использовать следующие ресурсы, программы, технологии:

- **Stack Overflow** - база, содержащая огромное количество проблем и решений для разработчиков с описанием программных интерфейсов, сервисов и платформ; не думайте, что решаете какую-то уникальную проблему. 99% вероятность того, что ее давно решили и решение есть на Stack Overflow.
- **Google** - просто вбейте в поиск свою проблему. Никогда не используйте поиск по Рунету, ищете на англоязычных сайтах. Главное при поиске - это правильно написать запрос. То есть ищем только в Google и только на английском языке.

Только помните момент - основную базу знаний по технологиям вы должны знать хорошо. Т.е. не думайте, что можно ничего не знать и все искать в Google при необходимости. Так вы очень медленно будете продвигаться. В Google ищем только нетипичные решения, которые встречаются относительно редко. Основные знания у вас должны быть автоматическими. Именно по этому мы назвали курс Core - это ядро ваших знаний, которые должны стать для вас родными.

2. Стремитесь к JavaScript разработке. Так как она не будет зависеть от платформы и позволяет делать сложные страницы. Хорошие JS программисты реально всем нужны. Сейчас очень много бекендщиков, сам по себе бекендщик не так ценен (если вы только не работаете узко по специальности). Еще момент - бекенд для большинства сайтов относительно типовой. Если бизнес-логика в приложении не очень сложная, то все сильно упрощается на стороне backend. Frontend - всегда на виду у клиента. Хорошая работа frontend гораздо лучше видна, чем гениальная работа бекендщика.

3. Используйте различные инструменты, облегчающие процесс разработки, например, DropBox (передача файлов), Clip2Net (снятие скриншотов и видео с экрана), skype mp3 recorder (запись общения с клиентом/менеджером через Skype), TeamViewer (удаленное подключение к клиенту), Axure RP (для прототипирования и создания макетов веб-сайтов и веб-приложений), Focus Booster (для концентрации на задаче), Toggl (для учета своего времени) Google Disk (облачное хранилище данных, совместная работа с документами), Punto Switcher (для быстрой работы с кодом и просто текстом). Инструментов очень много, постоянно пополняйте свою коллекцию таких инструментов.

Задание 5. Предложите не менее 10 полезных инструментов для разработчиков и полезные сервисы для работы в интернете.

Основные качества хорошего разработчика

- Количество ошибок и доработок. Отслеживайте эту метрику. Сколько ошибок после Вас находит клиент или тестер? Постоянно работайте над уменьшением этого показателя. Используйте различные техники, позволяющие уменьшить количество доработок за Вами. Важное пожелание: тестируйте за собой код. Проверяйте его на сервере. Не полагайтесь на свой талант и уверенность, что все работает - просто проверьте. Это крайне важная привычка. В нашей команде мы внедрили такое правило: сделал задачу - запиши видеоскрин с проверкой работы системы. Тем самым вы неявно можете протестировать свой код + дать другим членам команды уверенность, что ваша задача была как минимум вами протестирована.
- Применение типовых решений. 99% кода - это уже решенные когда-то задачи. Ваша задача - изучить эти решения и правильно их применять. Обычно большего не требуется. Типовое решение позволяет уменьшить количество ошибок и затраты на продукт. Изучайте типовые решения и пытайтесь понять принципы их работы. Т.е. не применяйте слепо эти решения, а применяйте принципы, заложенные в них. Если вы используйте какую-либо систему - в первую очередь изучите по максимуму ее возможности, документацию по ней. Это значительно упростит вам работу с системой в дальнейшим. Старайтесь в этом плане поменьше быть первооткрывателем. Еще раз - большинство ваших действий уже кто-то делал до вас, и ваша задача - просто найти это решение и правильно его внедрить в свой код.
- Создание повторяющегося кода. Если Вы постоянно переписываете один и тот же код - это плохо. В идеале код надо писать 1 раз и затем просто его использовать с разными параметрами. Наращивайте свою кодовую базу (свои типовые решения).
- Скорость реализации. Учитесь работать быстро. Скорость - это критически важное свойство хорошего разработчика. Одну и ту же задачу можно решить быстро, а можно очень-очень долго тянуть. Сделайте скорость своим приоритетом, засекайте время реализации разных компонентов. Например, мой личный рекорд по таблицам CRUD - 7 простых таблиц (фильтр, редактирование, сортировка, добавление) за 1 час 20 минут.
- Умение писать отчеты и работать прозрачно. Заказчик и менеджер должны четко понимать, в какой стадии находится Ваша работа. Давайте им обратную связь. Научитесь лаконично и прозрачно писать отчеты. Это не значит, что надо замарать кучу бумаги - отчет должен быть не более листа. В нем не должно быть ничего лишнего. Пишите в формате: что сделал, что буду делать, проблемы. Сначала потренируйтесь для себя - и оцените сами насколько у Вас получается.
- Культура кода. Очень редко программист пишет код в одиночку. Рано или поздно другие люди будут поддерживать Ваш код. Тут и выясняется, насколько хорошо Вы пишете. Хороший код написан, как увлекательная книга. Здесь практически не нужны комментарии-костыли. Каждый блок находится на своем месте. Все переменные понятны по своим названиям. Отступы соблюдены. Функции правильно выделены. Прочитайте книгу С. Макконелла “Совершенный

код”. При этом важно, чтобы Вы писали именно в соответствии с нотацией проекта, а не общепризнанными правилами. Т.е. не нужно вводить свои правила в существующий проект. Если внешние ключи называют categoryID, то не надо писать по другому (category_id, category и др). Если весь проект написан в единой нотации, то очень повышается скорость работы на проекте. Вам не надо проверять как называется то или иное поле или таблица - вы просто заранее знаете как оно должно называться.

- Писать с учетом производительности. Очень часто начинающие разработчики и разработчики средней руки пишут код, который работает. Клиент принимает работу. Но через некоторое время, когда данных становится больше - все начинает жутко тормозить. Причина - потому что об этом не подумали, когда делали систему. Мы тоже через это проходили и до сих пор проходим. Изучите тему оптимизации кода: LINQ, SQL, веб запросы, кеширование и т.д. В нашем более углубленном курсе мы посвятим время на проработку этого вопроса. Но все-таки скажу про пару полезных моментов относительно производительности:
    1. Изучите очень хорошо трассировку в ASP.NET и средства отладки.
    2. Всегда используйте в проекте средство MiniProfiler - оно сразу покажется ваши проблемные места.
    3. Очень осторожно используйте LINQ и Отложенную загрузку (LazyLoadiang) - это приводит к дикому количеству дублирующих запросов SQL.
- Обещания и репутация. Репутация программиста складывается из того, какие задачи он решает и как он это делает. Если Вы что-то обещаете - то делайте это вовремя. Репутация нарабатывается годами, а потерять ее можно за неделю. Работайте на свою репутацию осознанно. Через некоторое время сама репутация будет двигать Вас вперед. Самый важный фактор, влияющий на вашу репутацию - это как Вы реагируете на проблемы. Самое плохое, что Вы можете сделать - это просто пропасть и не отвечать. С таким человеком никто не захочет работать. При удаленном способе работы вдвойне важно быть всегда на связи.
- Первоклассный разработчик думает об удобстве пользования для конечного клиента. Какие-то моменты могут быть не прописаны в задании, но Вы знаете как можно сделать более удобно - сделайте это сразу. Дайте клиенту Wow эффект. Это очень ценный момент для любого клиента. Изучайте смежные дисциплины (именно на это направлен наш курс **Extend,** где рассматривают основы юзабилити, дизайна, менеджмента проектов, производительности и т.д.)

Пару слов о фрилансе

Понятно, что в начале у Вас будут 2 проблемы. Как сделать, чтобы это работало и Где искать клиентов. В этом курсе мы будем рассматривать только первый вопрос (т.е. Вашу техническую подготовку). Второй вопрос требует отдельной проработки. Со своей стороны могу сказать, что если

Вы действительно хорошо закроете первый вопрос и будете работать по всем тем принципам, которые указаны здесь - у Вас в любом случае будет шанс работать с нами.

И второй момент - это готовьтесь к тому, что нужно будет знать полный стек веб-технологий. Есть куча backend средних программистов, которые никому не нужны, т.к. не знают frontend разработку. Конечно, у Вас должна быть какая-то сильная сторона (например, работа со сложными запросами SQL), но в идеале Вы должны уметь работать по всему стеку технологий, чтобы в итоге Вы могли в одиночку сделать небольшой портал. В курсе мы и будем рассматривать один полный стек веб-разработки. Возможно чуть позже мы издадим новый курс касательно того, как начать свою карьеру на фрилансе (как оформить свой профиль, свою резюме, где и как искать клиентов, как обрабатывать заказы, юридические и финансовые моменты).

Как повысить свою производительность

Для программиста - это критически важный вопрос. Разница между 2 программистами может быть просто огромной. Есть программисты, которые делают одну CRUD таблицу за 2 часа (это был один из наших “новичков”, который декларировал нам опыт в 4 года!), а есть те, которые делают 7 таких таблиц за 80 минут (это случай из практики я писал об этом выше).

Применяйте правильные принципы при организации работы.

- Полная концентрация. Уберите все лишнее, что может Вас отвлекать. Телефон, skype, социальные сети, закройте все вкладки браузера. Заприте дверь на ключ. Задраить люк. В общем минимум 3 часа Вас вообще ничто не должно отвлекать.
- Ставьте конкретные цели. Заранее определите, что по итогам дня Вы должны сделать. Чем конкретнее, тем лучше. Также определяйте свой фокус на недели и на месяцы.
- Считайте свое время. Записывайте все, что Вы делаете по работе и анализируйте насколько быстро Вам это удается сделать. Сравнивайте себя с самим собой 3 месяца назад. Лучше ли Вы стали делать задачи?
- Делайте отчеты по своему прогрессу: что изучили, что разработали, что нового внедрили, какие новые компоненты общего пользования сделали. Планируйте свое обучение. Не нужно сразу кидаться на новую интересную технологию - запланируйте ее изучение в новом месяце. Концентрируйтесь на текущих целях.
- Изучайте свои слабые и сильные стороны. Работайте над ними целенаправленно. Берите проекты, где их можно проработать по максимуму. Описывайте все это в своем резюме и продумайте, что еще можно улучшить в вашем резюме. Изучите резюме других людей.
- Создайте идеальные условия труда. Самый лучший монитор (или 2 монитора), эргономичная клавиатура (если необходимо), хорошие микрофон и наушники, удобное кресло и т.д. Сделайте так, чтобы работать было максимально удобно.

- Изучайте способы быстрой работы: горячие клавиши, печатать на клавиатуре вслепую, сниппеты и т.д. Заметьте какие типовые операции Вы больше всего делайте за день и сознательно применяйте для них горячие клавиши (создать скрин, запустить проект, остановить проект, вызов скайпа, закрытие/открытие вкладки браузера и т.д.)

Попробуйте неделю жить по таким правилам и оцените эффект на своем опыте.

Задание 6. Напишите самые полезные горячие клавиши в Gmail, Skype.

Структура курса

Каждая глава книги - это отдельная технология. Мы не претендуем на полноту материала - здесь даются только базовые навыки и направления для развития. Каждая технология будет содержать следующие подразделы:

- Описание: общие сведения о технологи и зачем она нужна.
- Примеры: типовые примеры и задачи. Показаны несколько распространенных примеров по использованию этой технологии.
- Советы: советы и подсказки по использованию.
- Ссылки: что нужно изучить и попробовать, чтобы узнать технологию на более-менее приличном уровне.
- Задачи: задания по рассматриваемой технологии, которые позволят Вам более основательно ее проработать.

При выполнении заданий Вам необходимо создать папку на Google Docs и собирать в ней все результаты заданий.

Вернемся к технологиям. Все технологии условно разбиты на 3 уровня:

1. Уровень доступа к данным (DAL)
   1.1. SQL Server
   1.2. SQL
   1.3. LINQ
   1.4. ADO.NET
   1.5. Entity Framework
   1.6. Кеширование
2. Уровень обработки вызовов с клиентской части (браузера)
   2.1. ASP.NET
   2.2. C#
   2.3. Web Methods
   2.4. Сериализация
3. Уровень пользовательского интерфейса (UI).

3.1. HTML
3.2. CSS и верстка
3.3. JS
3.4. jQuery
3.5. Ajax
3.6. Паттерны JS
3.7. Формирование разметки в ASP.NET

В некоторой степени они перемешиваются, но тем не менее так Вам проще будет понять назначение технологий, которые мы будем рассматривать. В целом это похоже на MVC подход, но я намеренно не загружаю Вас лишними на данный момент терминами.

Задание 7. Найдите в Интернете что такое паттерн MVC:)