Глава 1
Знакомство с разработкой full-stack

В этой главе:

- преимущества разработки full-stack;
- обзор компонентов стека MEAN;
- то, что делает стек MEAN столь интересным;
- предварительный обзор приложения, которое мы будем создавать на протяжении всей книги.

Если мы с вами похожи, то вам, наверное, не терпится скорее углубиться в какой-нибудь код и начать что-то создавать. Но сначала воспользуемся моментом, выясним, что же такое *разработка full-stack*, и взглянем на составные части стека, чтобы убедиться, что вы понимаете их все.

При упоминании разработки full-stack я подразумеваю разработку всех частей сайта или приложения. Full-stack начинается с БД и веб-сервера в прикладной части, центральной частью становится логика приложения и управления, а окончанием — пользовательский интерфейс в клиентской части.

Стек MEAN включает четыре основные технологии:

- **M**ongoDB — база данных;
- **E**xpress — веб-фреймворк;
- **A**ngularJS — фреймворк клиентской части;
- **N**ode.js — веб-сервер,

а также ряд вспомогательных.

MongoDB существует с 2007 года и активно поддерживается компанией MongoDB Inc., ранее известной как 10gen.

Express выпущен в 2009 году Т. Дж. Головайчуком и с тех пор стал самым популярным фреймворком для Node.js. Его исходный код открыт, он активно разрабатывается и поддерживается более чем 100 участниками проекта.

AngularJS относится к ПО с открытым исходным кодом и поддерживается корпорацией Google. Он существует с 2010 года и постоянно развивается и расширяется.

Node.js был создан в 2009 году, его разработка и поддержка спонсируются Joyent. Node.js использует в качестве ядра V8 — движок JavaScript с открытым исходным кодом компании Google.

1.1. Зачем изучать full-stack

Действительно, зачем изучать full-stack? Похоже, это огромный кусок работы! Ну да, это *действительно* огромный, но весьма полезный кусок работы. И не все в стеке MEAN так сложно, как вы можете подумать.

1.1.1. Краткая история веб-разработки

В давние времена, когда Интернет был совсем юным, люди не ждали многого от сайтов. Способ представления данных не был столь важен, гораздо больше всех заботило то, что происходило за кулисами. Обычно, если вы немного знали Perl и могли скомпоновать фрагмент HTML, то уже считались веб-разработчиком.

По мере распространения использования Интернета бизнес-компании начали сильнее интересоваться тем, как они в нем представлены. В сочетании с улучшенной поддержкой браузерами каскадных таблиц стилей (Cascading Style Sheets (CSS)) и JavaScript такой интерес привел к тому, что реализация клиентской части стала более сложной. Речь больше не шла о способности компоновать HTML — приходилось тратить время на CSS и JavaScript, убеждаясь в привлекательности внешнего вида сайтов и их правильном функционировании. И все это должно было работать в различных браузерах, значительно хуже совместимых, чем сейчас.

Именно тогда стали различать разработчиков клиентской и прикладной частей. Рост этого различия с течением времени демонстрирует рис. 1.1.

В то время как разработчики прикладной части концентрируют свое внимание на закулисной механике, разработчики клиентской части сосредоточены на обеспечении пользователю приятного опыта взаимодействия с сайтом. С течением времени требования к обоим лагерям выросли, что способствовало развитию этой тенденции. Разработчикам часто приходилось выбирать себе область компетенции и концентрироваться на ней.

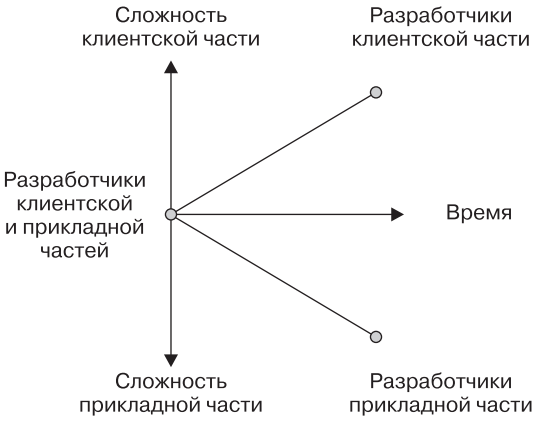

Рис. 1.1. Рост различий разработчиков клиентской и прикладной частей с течением времени

Как библиотеки и фреймворки помогают разработчикам

В 2000-е годы библиотеки и фреймворки стали популярными и широко применяются в большинстве распространенных языков программирования как на стороне клиента, так и на стороне сервера. Вспомните Dojo и jQuery для клиентского JavaScript, а также CodeIgniter для PHP и Ruby on Rails. Эти фреймворки были спроектированы для облегчения жизни разработчика, снижения входного порога. Хорошая библиотека или фреймворк уменьшают сложность разработки, ускоряя написание кода и снижая требования к наличию специальных знаний. Эта тенденция к упрощению привела к возрождению разработчиков full-stack, создававших как клиентскую часть, так и стоящую за ней логику приложения (рис. 1.2).

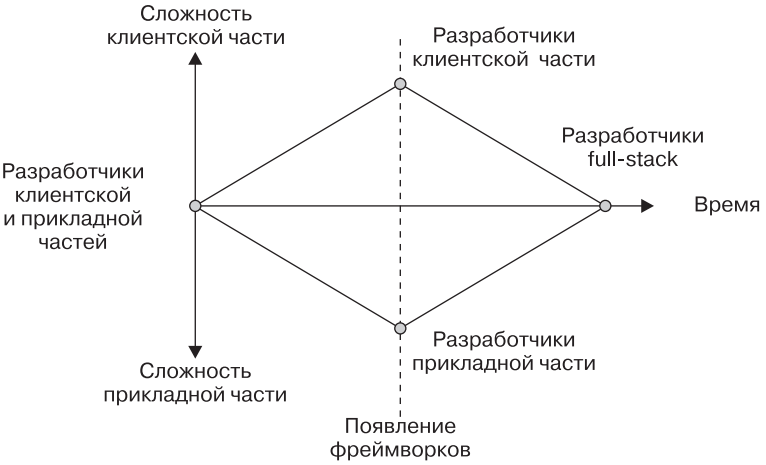

Рис. 1.2. Влияние фреймворков на отдельные блоки веб-разработки

Этот рисунок иллюстрирует скорее тенденцию, чем утверждение: «Все веб-разработчики должны быть разработчиками full-stack». Конечно, на протяжении всей истории Интернета существовали разработчики full-stack, а в перспективе, вероятнее всего, некоторые разработчики будут выбирать в качестве специализации клиентскую или серверную разработку. Я же стремлюсь показать, что благодаря использованию фреймворков и современных инструментов вам, чтобы быть хорошим веб-разработчиком, больше не надо выбирать одну из сторон.

Огромное преимущество использования подхода на основе фреймворков заключается в невероятной производительности в силу того, что можно рассмотреть приложение и его внутренние связи со всех сторон.

Продвижение кода приложения дальше по стеку

Продолжая тенденцию фреймворков, в последние несколько лет отмечаем развивающуюся тенденцию к вынесению логики приложения из серверной части в клиентскую. Можно рассматривать это как программирование прикладной части в клиентской. Наиболее популярные среди применяющих этот подход фреймворков JavaScript — AngularJS, Backbone и Ember.

Такое тесное сцепление кода приложения с клиентской частью на самом деле начинает размывать границу между теми, кто традиционно разрабатывает клиентскую и серверную части. Одна из причин популярности данного подхода — снижение благодаря этому нагрузки на сервер, а значит, снижение стоимости. Фактически при этом применяется привлечение необходимых приложению вычислительных мощностей за счет перенесения нагрузки на пользовательские браузеры.

Далее в книге я рассмотрю аргументы за и против этого подхода, включая то, когда уместно или неуместно применять одну из таких технологий.

1.1.2. Тенденция к разработке full-stack

Как уже говорилось, пути разработчиков клиентской и прикладной частей пересекаются и вполне можно быть профессионалом в обеих областях. Если вы фрилансер, консультант или трудитесь в небольшой команде, разносторонняя квалификация исключительно полезна, так как повышает вашу ценность для заказчиков. Умение разработать сайт или приложение от начала до конца позволяет вам полностью контролировать процесс и придает согласованность совместной работе различных частей, как если бы их не создавали по отдельности разные команды.

Если вы работаете в большой компании, то, вероятно, вам придется специализироваться (или по крайней мере сконцентрировать свои усилия) в одной области. Но, как правило, будет полезно знать, как созданные вами компоненты сочетаются с другими компонентами — это позволит лучше понимать требования и цели других команд и проекта в целом.

Наконец, строить приложения, основываясь на full stack, весьма полезно. В каждой его части есть свои задачи и проблемы, решение которых поддерживает интерес к разработке. Технологии и доступные на сегодняшний день инструменты усиливают эти ощущения и помогают создавать замечательные веб-приложения относительно легко и просто.

1.1.3. Преимущества разработки full-stack

Существует множество преимуществ изучения разработки full-stack. Для начинающих это, конечно, удовольствие от изучения нового и экспериментов с новыми технологиями. А также удовольствие от овладения чем-то совершенно особым и радостное возбуждение от возможности самостоятельно создать и запустить полномасштабное, ориентированное на работу с данными приложение.

Преимуществами работы в команде являются следующие:

- благодаря тому что вы понимаете, как устроены различные области и как они совместно функционируют, вы лучше видите более обширную картину;
- вы лучше понимаете, как действуют другие части команды и что им нужно для успешной работы;
- члены команды могут свободнее менять участок работы.

Дополнительные преимущества самостоятельной работы:

- вы можете самостоятельно создавать приложения от начала до конца, не завися от других людей;
- вы приобретаете больше навыков и имеете возможность предложить своим заказчикам большее число услуг.

В общем, можно многое сказать в пользу разработки full-stack. Большинство встречавшихся мне наиболее искусных разработчиков были разработчиками full-stack. Их всеобъемлющие знания и способность видеть более обширную картину давали им громадное преимущество перед другими.

1.1.4. Почему именно стек MEAN

Стек MEAN объединяет несколько лучших в своем классе современных веб-технологий в очень мощный и гибкий стек. Одна из лучших его черт: он не просто использует JavaScript в браузере — он использует JavaScript повсюду. С помощью стека MEAN вы можете программировать как клиентскую, так и серверную часть на одном и том же языке.

Основная технология, которая делает это возможным, — Node.js, внедряющая JavaScript в прикладную часть.

1.2. Знакомимся с Node.js: веб-сервер/веб-платформа

Node.js — это буква N в MEAN. То, что она стоит на последнем месте, отнюдь не значит, что она менее важна, чем прочие составляющие, — на самом деле это основа стека!

В двух словах: Node.js — программная платформа, позволяющая вам создать собственный веб-сервер и строить на нем веб-приложения. Node.js сам по себе не веб-сервер и не язык программирования. Он включает встроенную серверную библиотеку HTTP, следовательно, вам не нужно запускать отдельный веб-сервер, такой как Apache или Internet Information Services (IIS). В конечном счете вы получаете более полный контроль над работой своего веб-сервера за счет усложнения его установки и запуска, особенно в промышленной среде.

В случае PHP, например, вы можете легко найти виртуальный веб-хостинг, на котором запущен Apache, отправить туда файлы по FTP, и — вуаля! — ваш сайт работает. Это возможно, поскольку веб-хостинг заранее настроил Apache для вас и других пользователей. В случае Node.js все иначе, так как вы настраиваете сервер Node.js при создании приложения. Многие традиционные провайдеры веб-хостинга отстают в смысле поддержки Node.js, но несколько новых хостингов, работающих по принципу «платформа как услуга» (Platform as a Service (PaaS)), например Heroku, Nodejitsu и OpenShift, учитывают эту потребность. Подход к развертыванию промышленных сайтов на подобных PaaS-хостингах отличается от старой модели FTP, но если в нем разобраться, оказывается, что он довольно удобен. Мы будем развертывать сайт на Heroku в процессе продвижения по книге.

Альтернативный подход к хостингу приложения Node.js состоит в выполнении всей работы самостоятельно на выделенном сервере, на который можно установить все необходимое. Но описание администрирования промышленного сервера потребовало бы отдельной книги! Можно заменить любой из прочих компонентов альтернативной технологией, и это ни на что не повлияет, но отказ от Node.js приведет к изменению всего, что надстроено поверх него.

1.2.1. JavaScript: единый язык программирования для всего стека

Одна из главных причин популярности платформы Node.js такова: для нее программируют на знакомом большинству программистов языке — JavaScript. До сегодняшнего дня, если вы хотели быть разработчиком full-stack, вам нужно было в совершенстве знать по крайней мере два языка программирования: JavaScript для клиентской части и что-то еще, например PHP или Ruby, — для серверной.