ГИПЕРМЕДИА: ПОВТОРНОЕ ЗНАКОМСТВО

В наши дни технология гипермедиа стала универсальной и почти так же распространена, как электричество.

Миллиарды людей ежедневно используют системы на основе гипермедиа, в основном на уровне взаимодействия с языком гипертекстовой разметки *HTML* (Hypertext Markup Language), передаваемом по протоколу *HTTP* (Hypertext Transfer Protocol); для этого используется веб-браузер, подключенный к Сети.

Такие системы используются для получения новостей, общения с друзьями, покупок в интернете, игр, отправки электронной почты и т. д. Разнообразие и количество онлайн-сервисов, предоставляемых через гипермедиа, потрясает воображение.

Тем не менее, несмотря на такую распространенность, тема гипермедиа сама по себе остается на удивление малоизученной; в основном ею занимаются специалисты. Да, можно без труда найти множество пособий о том, как писать HTML, создавать ссылки и формы и т. д. Однако HTML очень редко рассматривается как *технология гипермедиа*, а в более широком смысле и как комплекс систем гипермедиа.

Ситуация заметно отличается от той, что была на заре веб-разработки, когда такие концепции, как *REST* (Representational State Transfer) или *HATEOAS* (Hypermedia As The Engine of Application State, «гипермедиа как ядро состояния приложения»), часто обсуждались, уточнялись и вызывали споры между веб-разработчиками.

Как ни печально, сегодня многие ругают самый популярный язык гипермедиа — HTML: это громоздкий, устаревший язык разметки, который приходится применять в пользовательских интерфейсах веб-приложений, уже почти полностью создаваемых на JavaScript. Так получилось, что HTML присутствует в браузере и мы вынуждены с ним работать.

Такой подход удручает, и мы надеемся убедить вас в том, что гипермедиа — не просто устаревшая технология, которую приходится принимать и использовать.

Мы хотим показать, что гипермедиа — невероятно революционный, простой и *гибкий* способ построения надежных приложений: *гипермедиа-управляемых приложений*.

Хочется верить, что к концу этой книги вы, как и мы, почувствуете, что гипермедиа может стать хорошим выбором архитектуры приложения, которое вы планируете разрабатывать. Вариант гипермедиа-управляемого приложения на основе *системы гипермедиа* (такой, как веб) вполне жизнеспособен; более того, нередко это отличное решение для *современного* веб-приложения.

(И как будет показано в части, посвященной Hyperview, не только для веб-приложения.)

Что такое гипермедиа?

> Гипертекст — новая форма письма, выводимая на экран компьютера, которая ветвится или выполняет действие по запросу читателя. Гипертекст является нелинейной формой письма; он приносит практическую пользу только при выводе на экран.
>
> *Тед Нельсон,* https://archive.org/details/SelectedPapers1977/page/n7/mode/2up

Начнем с самого начала: что такое гипермедиа?

Гипермедиа — информационная среда (например, текст) с возможностью *нелинейного перехода* из одной своей точки в другую по встроенным гиперссылкам. Префикс «гипер-» происходит от греческого префикса «ὑπερ-», означающего «за пределами» или «сверх-». Он указывает, что гипермедиа выходит за рамки обычных, пассивно потребляемых информационных сред, таких как журналы или газеты.

Гиперссылки — классический пример того, что называется *элементом управления гипермедиа* (hypermedia control).

> *Элемент управления гипермедиа* — это элемент гипермедиа, описывающий некоторое взаимодействие (или управляющий этим взаимодействием), часто с удаленным сервером. Информация об этом взаимодействии напрямую и полностью кодируется внутри самого элемента.

Элементы управления — то, что отличает гипермедиа от других видов информационных средств.

Возможно, вам более знаком термин *«гипертекст»*; приведенное выше определение взято со страницы в «Википедии», посвященной гипертексту. Гипертекст составляет подкатегорию гипермедиа, и бóльшая часть этой книги будет по-

священа тому, как строить современные приложения на основе гипертекста, например HTML (Hypertext Markup Language) или HXML (гипертекста, используемого мобильной гипермедиа-системой Hyperview).

Гипертекст, как HTML, работает в сочетании с другими технологиями, необходимыми для функционирования всей системы гипермедиа: сетевыми протоколами (такими, как HTTP), другими типами информационных сред (например, графикой или видео), серверами гипермедиа (то есть серверами, предоставляющими API гипермедиа), полнофункциональными клиентами гипермедиа (например, веб-браузерами) и т. д.

По этой причине для описания базовой архитектуры приложений с применением гипертекста мы предпочитаем более широкий термин *системы гипермедиа*, чтобы подчеркнуть роль системной архитектуры в конкретной используемой разновидности гипермедиа.

Именно *архитектура* системы гипермедиа в целом не пользуется должным вниманием и игнорируется многими современными веб-разработчиками.

Краткая история гипермедиа

Как возникла идея гипермедиа?

Хотя у современной концепции гипертекста и более общей концепции гипермедиа было много предшественников, отправной точкой для формирования современных представлений о гипермедиа многие считают статью Ванневара Буша (Vannevar Bush) «As We May Think», вышедшую в 1945 году в журнале «Atlantic».

В этой статье Буш описал устройство, которое он назвал Memex. Это устройство, использовавшее сложную механическую систему катушек и микропленок в сочетании с системой кодирования, давало бы возможность пользователю переходить между взаимосвязанными кадрами контента. Устройство Memex так никогда и не было создано, но его идея послужила вдохновением для последующей проработки концепции гипермедиа.

Термины «гипертекст» и «гипермедиа» были введены в 1963 году Тедом Нельсоном (Ted Nelson), который работал над системой редактирования гипертекста в Университете Брауна, а позднее создал *FRESS* (File Retrieval and Editing System) — невероятно передовую для своего времени систему гипермедиа. (Не исключено, что это была первая цифровая система с возможностью «отмены» (undo).)

Пока Нельсон работал над своими идеями, Дуглас Энгельбарт (Douglas Engelbart) трудился в Стэнфордском исследовательском институте, пытаясь воплотить

в реальность устройство Memex Ванневара Буша. В 1968 году Энгельбарт провел «демонстрацию века» в Сан-Франциско, штат Калифорния.

Энгельбарт продемонстрировал ряд невероятных технологических достижений:

- дистанционное редактирование текста совместно с коллегами в Менло-Парке;
- видео- и аудиочат;
- интегрированную систему окон с возможностью изменения размеров окон и других параметров;
- гипертекст, в котором по щелчку на подчеркнутом фрагменте происходит переход к новому контенту.

Несмотря на бурные овации потрясенной после выступления аудитории, прошли десятилетия, прежде чем продемонстрированные Энгельбартом технологии получили массовое распространение.

Современная реализация

В 1990 году Тим Бернерс-Ли (Tim Berners-Lee), работавший в CERN, опубликовал первый веб-сайт. Он работал над идеей гипертекста около 10 лет. Наконец, в отчаянии от того факта, что ученым так трудно делиться результатами своих исследований, он нашел подходящий момент и организационную поддержку для создания Всемирной паутины:

> “Создание Всемирной паутины в действительности было актом отчаяния, потому что во время моей работы в CERN ситуация была очень сложной. Многие технологии, задействованные во Всемирной паутине (гипертекст, интернет, многошрифтовые текстовые объекты), уже были спроектированы. Мне оставалось лишь объединить их. Это было обобщением, переходом на более высокий уровень абстракции, попыткой рассматривать все системы документации как части более крупной воображаемой системы документации.
>
> *Тим Бернерс-Ли*
> https://britishheritage.org/tim-berners-lee-the-world-wide-web

К 1994 году его творение стало развиваться настолько быстро, что Бернерс-Ли основал W3C — группу компаний и исследователей, работавших над улучшением Паутины. Все стандарты, созданные W3C, не требовали лицензионных отчислений. Любой желающий мог адаптировать и реализовать их, что привело к укреплению открытой, ориентированной на сотрудничество природы Всемирной паутины.

В 2000 году Рой Филдинг, тогда работавший в Калифорнийском университете в Ирвайне, опубликовал во Всемирной паутине свою судьбоносную диссертацию «Architectural Styles and the Design of Network-based Software Architectures» («Архитектурные стили и дизайн программных архитектур сетевой среды»). Филдинг работал над HTTP-сервером с открытым исходным кодом Apache и в своей работе описывал то, что считал *новой особой сетевой архитектурой*, появившейся в ранней версии Паутины.

Филдинг работал над первыми спецификациями HTTP. В своей статье для определения сетевой модели гипермедиа он использовал термин REST (REpresentational State Transfer).

Работа Филдинга стала опорой для первых веб-разработчиков, получивших в свое распоряжение язык для обсуждения новой технической среды, в которой они строили приложения.

Мы подробно обсудим ключевые идеи Филдинга в главе 2, а заодно попытаемся навести порядок с определениями REST, HATEOAS и гипермедиа.

Самый успешный гипертекст: HTML

> “ В начале была гиперссылка, и гиперссылка была с веб-средой, и гиперссылка была веб-средой. И это было хорошо.
>
> *Rescuing REST From the API Winter*
> https://intercoolerjs.org/2016/01/18/rescuing-rest.html

HTML, над которым работали Бернерс-Ли, Филдинг и многие другие, создавался вокруг гипермедиа. Изначально разметка HTML представляла собой гипермедиа-текст, предназначенный только для чтения и используемый для публикации академических документов. Эти документы связывались при помощи якорных тегов, которые создавали между ними *гиперссылки*. При помощи этих ссылок пользователи могли быстро перемещаться от одного документа к другому.

В спецификации HTML 2.0 появилось понятие тега `form`, присоединившегося к якорному тегу (то есть гиперссылке) как второй элемент управления. Введение тега `form` открыло возможность построения *приложений* в веб-среде, так как оно предоставляло механизм *обновления* ресурсов, а не только их чтения.

В этот момент Всемирная паутина из интересной документоориентированной системы превратилась в перспективную *архитектуру приложений*.

В наши дни HTML является самой широко используемой технологией гипермедиа. Естественно, предполагается, что наш читатель достаточно хорошо знаком с ней. Вам не нужно быть экспертом в области HTML (или CSS), чтобы понять приведенный в книге код, но чем лучше вы разбираетесь в основных тегах и концепциях HTML, тем больше пользы принесет вам книга.

Сущность HTML как гипермедиа

Рассмотрим два определяющих гипермедиа-элемента (*элемента управления*) HTML — якорный тег и тег формы — более подробно.

Якорные теги

Якорные теги встречаются настолько часто, что кажутся набившими оскомину. И все же стоит рассмотреть механику работы гиперссылок, чтобы настроиться на более глубокое понимание гипермедиа.

Возьмем простой якорный тег, встроенный в документ HTML.

Листинг 1. Простая гиперссылка

```
<a href="https://hypermedia.systems/">
  Системы гипермедиа
 </a>
```

Якорный тег состоит из собственно тега `<a></a>`, а также атрибутов и контента внутри тега. Особый интерес представляет атрибут `href`, который задает *гипертекстовую ссылку* на другой документ или фрагмент документа. Именно этот атрибут превращает якорный тег в элемент гипермедиа.

Типичный браузер будет интерпретировать этот якорный тег следующим образом:

- Вывести текст «Системы гипермедиа» в такой форме, чтобы пользователь видел, что на нем можно щелкнуть.
- Когда пользователь щелкает на этот текст, выдать запрос `HTTP GET` к URL https://hypermedia.systems/.
- Принять HTML-контент из тела ответа HTTP на этот запрос и заменить весь экран в браузере новым документом, с обновлением адресной строки новым URL-адресом.

Якорные ссылки обеспечивают основной механизм перемещения в современной веб-среде — переход по ссылке от документа к документу или от ресурса к ресурсу.

Схематично взаимодействие пользователя с якорным тегом/гиперссылкой выглядит так:

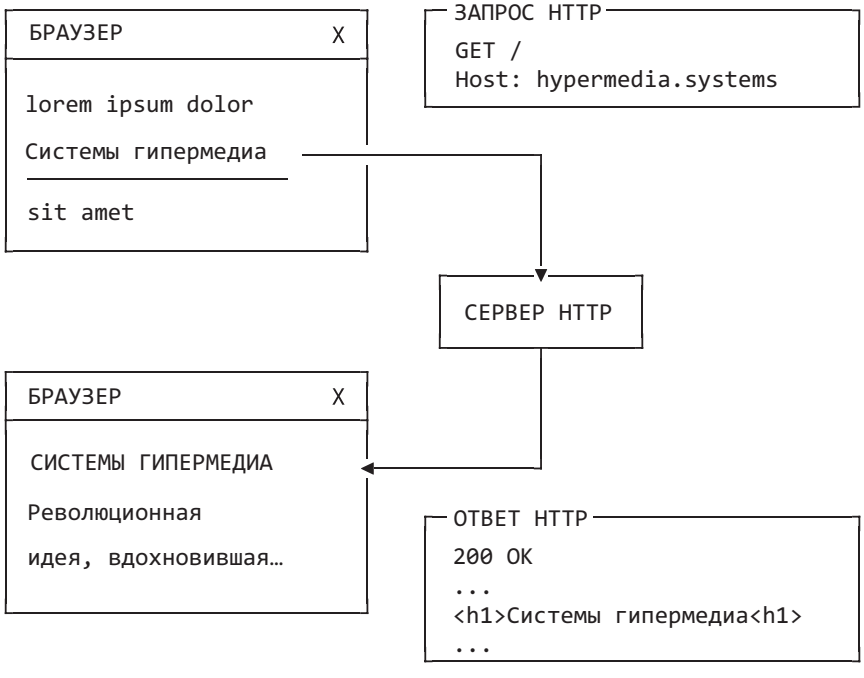

Рис. 1. Запрос HTTP GET в действии

При щелчке на ссылке браузер (или, как иногда говорят, *клиент гипермедиа*) инициирует `GET`-запрос HTTP к URL-адресу, закодированному в атрибуте `href` ссылки.

Обратите внимание, что запрос HTTP включает дополнительные данные (*метаданные*). Они описывают, что именно браузеру нужно получить от сервера, в форме заголовков. Эти заголовки и HTTP более подробно рассматриваются в главе 2.

Затем с *сервера гипермедиа* поступает *гипермедийный ответ* на запрос — разметка HTML — для новой страницы. На первый взгляд этот момент может показаться незначащим и очевидным, но это исключительно важный признак *системы гипермедиа*, действительно отвечающей условиям REST: клиент и сервер должны взаимодействовать через гипермедиа!

Теги форм

Якорные теги обеспечивают *навигацию* между документами или ресурсами, но не позволяют обновлять эти ресурсы. За эту функциональность отвечает тег `form`.

Простой пример формы HTML:

Листинг 2. Простая форма

```
<form action="/signup" method="post">
 <input type="text" name="email" placeholder="Укажите адрес электронной почты,
                          чтобы зарегистрироваться..."/>
 <button>Регистрация</button>
</form>
```

Как и якорный тег, тег формы состоит из собственно тега `<form></form>`, а также атрибутов и контента внутри тега. Обратите внимание: у тега формы нет атрибута `href`, но есть атрибут `action`, указывающий, куда следует направить запрос HTTP. Кроме того, он содержит атрибут `method`, который точно указывает, какой «метод» HTTP должен использоваться. В приведенном примере форма дает команду браузеру выдать запрос `POST`.

В отличие от якорных тегов, контент и теги *внутри* формы могут влиять на гипермедиа-взаимодействие между формой и сервером. *Значения* тегов `input` и других тегов (например, `select`) при отправке данных формы будут включены в запрос HTTP как параметры URL в случае `GET` и как часть тела запроса в случае `POST`. В отличие от якорного тега, это позволяет форме включать в запрос произвольный объем информации, полученной от пользователя.

Типичный браузер будет интерпретировать этот тег формы и его содержимое примерно следующим образом:

- Вывести текстовое поле ввода и кнопку «Регистрация» для пользователя.
- Когда пользователь отправит данные формы, щелкнув на кнопке «Регистрация» или нажав клавишу Enter, пока элемент `input` обладает фокусом, направить запрос HTTP `POST` по пути `/signup` «текущего» сервера.
- Принять HTML-контент из тела ответа HTTP и заменить весь экран в браузере новым документом, с обновлением адресной строки новым URL-адресом.

Этот механизм позволяет пользователю выдавать запросы на *обновление состояния* ресурсов на сервере. Заметим, что, несмотря на новый тип запроса, весь обмен данными между клиентом и сервером все еще происходит исключительно через *гипермедиа*.

Именно тег формы делает возможным существование гипермедиа-управляемых приложений.

Вероятно, опытный веб-разработчик заметит, что мы опустили некоторые подробности и трудные случаи. Например, ответ на отправку формы часто *перенаправляет* клиента на другой URL-адрес.

Это правда, и мы еще займемся техническими подробностями форм в следующих главах, а пока и этого простого примера будет достаточно для демонстрации базового механизма обновления состояния системы исключительно через гипермедиа.

Диаграмма взаимодействия выглядит так:

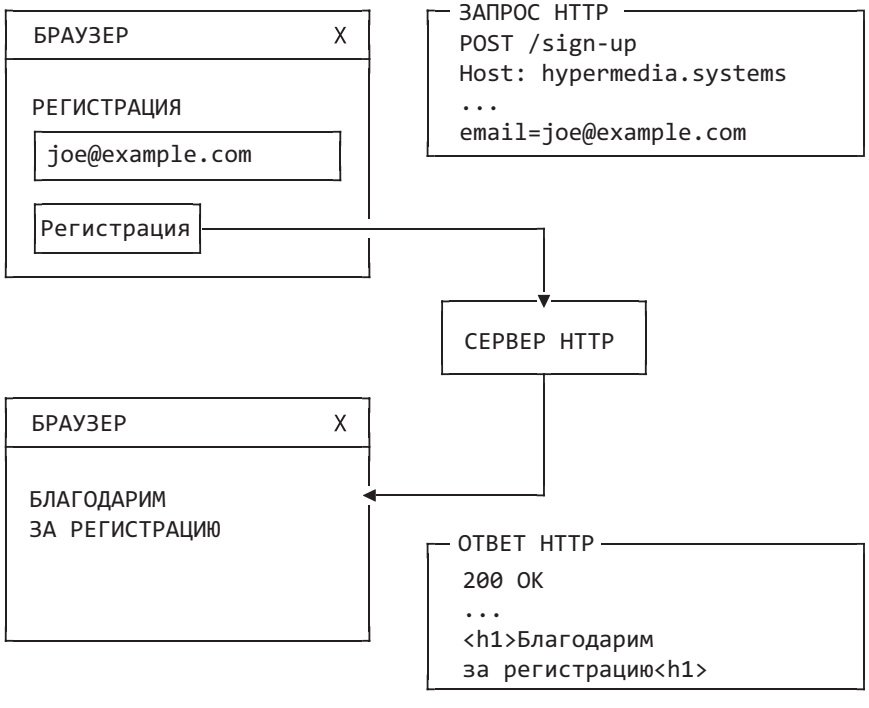

Рис. 2. Запрос HTTP POST в действии

Приложения Web 1.0

Как человеку, интересующемуся веб-разработкой, вам, вероятно, уже знакомы приведенные выше диаграммы и рассуждения. Возможно, кому-то этот материал даже покажется скучным. Но сделайте шаг назад и обратите внимание на тот факт, что этими двумя элементами гипермедиа — якорями и формами — ограничиваются *все* собственные возможности базового HTML для взаимодействия пользователя с сервером.

Всего два тега!

Тем не менее они обеспечили Всемирной паутине на заре ее развития рост с экспоненциальной скоростью и возможность предоставлять невероятно огромный объем динамической онлайн-функциональности миллиардам людей.

Это убедительное доказательство мощи гипермедиа. Даже сегодня, когда в веб-разработке все чаще на ведущие позиции выходят большие JavaScript-ориентированные интерфейсные фреймворки, многие предпочитают использовать для достижения своих целей простой базовый HTML, и нередко результат их полностью устраивает.

Эти два тега наделяют HTML колоссальной выразительностью.

Что не относится к гипермедиа?

Итак, ссылки и формы — два основных механизма гипермедиа, предназначенные для взаимодействия с сервером и доступные в HTML.

Теперь рассмотрим другой подход: взаимодействие с сервером посредством выдачи запроса HTTP через JavaScript. Для этого мы будем использовать `fetch()`[1] — популярный API для выдачи запросов «асинхронного JavaScript и XML», или запросов *AJAX* (*Asynchronous JavaScript and XML*), доступный во всех современных браузерах.

Листинг 3. JavaScript

```
<button onclick="fetch('/api/v1/contacts/1') ❶
        .then(response => response.json()) ❷
        .then(data => updateUI(data))">  ❸
  Fetch Contact
</button>
```

❶ Выдача запроса.

❷ Преобразование ответа в объект JavaScript.

❸ Вызов функции `updateUI()` с объектом.

Кнопка имеет атрибут `onclick`, в котором хранится код JavaScript, выполняемый при щелчке по ней.

JavaScript выдает запрос AJAX HTTP `GET` к пути `/api/v1/contacts/1` с использованием `fetch()`. Запрос AJAX похож на «обычный» запрос HTTP, но он выдается браузером скрыто. Пользователь не видит индикатор запроса от браузера, как в случае с обычными ссылками и формами. Кроме того, в отличие от запросов, выдаваемых этими элементами управления, за обработку ответа от сервера отвечает код JavaScript.

Хотя в акроним AJAX входит XML, в наши дни ответ HTTP на такой запрос почти наверняка будет закодирован в формате JSON (JavaScript Object Notation), а не в XML.

[1] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Ответ HTTP на этот запрос может выглядеть примерно так:

Листинг 4. JSON

```
{ ❶
 "id": 42, ❷
 "email" : "json-example@example.org" ❸
}
```

❶ Начало объекта JSON.
❷ Свойство — в данном случае с именем id и значением 42.
❸ Другое свойство — адрес электронной почты контакта с этим id.

Приведенный выше код JavaScript преобразует текст JSON, полученный от сервера, в объект JavaScript; для этого текст передается в вызове метода `json()`. Далее новый объект JavaScript передается методу `updateUI()`.

Метод `updateUI()` отвечает за обновление пользовательского интерфейса на основании данных, закодированных в объекте JavaScript, — возможно, с выводом данных контакта в разметке HTML, генерируемой клиентским шаблоном в приложении JavaScript.

Подробности того, что делает функция `updateUI()`, нам неважны.

А что важно и что является *критической* особенностью взаимодействий с сервером на основе JSON — то, что в них не используются гипермедиа. JSON API не возвращает гипермедийный ответ. В нем нет гиперссылок или других элементов гипермедиа. Здесь JSON API скорее является *API данных*.

Так как ответ закодирован в формате JSON, а *не* в формате гипермедиа, методу JavaScript `updateUI()` нужна информация, как преобразовать эти контактные данные в HTML.

В частности, коду `updateUI()` необходимы сведения о *внутренней структуре* и смысле данных. Он должен знать:

- точную структуру и имена полей в объекте данных JSON;
- какими отношениями они связаны;
- как обновить локальные данные, которым соответствуют эти новые данные;
- как отобразить эти данные в браузере;
- какие дополнительные действия / конечные точки API могут вызываться с этими данными.

Вкратце, логика `updateUI()` должна обладать полной информацией о конечной точке API, имеющей путь `/api/v1/contact/1`, причем эта информация должна предоставляться сторонним источником не внутри самого ответа. В результате

между кодом `updateUI()` и API возникают особые отношения, называемые *сильной связанностью* (tight coupling): если формат ответа JSON изменится, то код `updateUI()` почти наверняка тоже придется изменять.

Одностраничные приложения

Хотя приведенный фрагмент кода JavaScript выглядит очень скромно, это органическое начало целой большой концепции создания веб-приложений. С него начинаются *одностраничные приложения* (Single Page Application, SPA). Веб-приложение уже не осуществляет навигацию *между* страницами с использованием гипермедийных элементов управления, как это было со ссылками и формами.

Вместо этого приложение обменивается *обычными данными* с сервером, а затем обновляет контент *в пределах* одной страницы.

Когда такая стратегия или архитектура принимается для всего приложения, все фактически происходит на одной странице и приложение становится «одностраничным».

Архитектура одностраничных приложений стала чрезвычайно популярна и в последнее десятилетие доминирует при построении веб-приложений. Об этом говорят высокий уровень внимания отрасли к этой архитектуре и частота ее обсуждений.

В наши дни в подавляющем большинстве одностраничных приложений для управления пользовательским интерфейсом применяются намного более сложные фреймворки, чем показано в этом простом примере. Такие популярные библиотеки, как React, Angular, Vue.js и т. д., стали распространенным — и можно сказать, стандартным — способом построения веб-приложений.

С такими сложными фреймворками разработчики обычно применяют более тщательно проработанную модель на стороне клиента, а именно объекты JavaScript, хранимые локально в памяти браузера и представляющие «модель» или «предметную область» приложения. Эти объекты JavaScript обновляются кодом JavaScript, а фреймворк «реагирует» на эти изменения обновлением пользовательского интерфейса.

Когда пользовательский интерфейс (UI) обновляется пользователем, эти изменения также распространяются в объекты модели, устанавливая «двусторонний» механизм связывания: модель может обновить пользовательский интерфейс, а пользовательский интерфейс может обновить модель.

Это намного более сложный подход к веб-клиенту, чем гипермедиа. Обычно он почти полностью упраздняет нижележащую инфраструктуру гипермедиа, доступную в браузере.