

Headless Architecture

Brown-Bag Session

Presented by:

K.C.Ashish Kumar



ashishkumarkc.com

Agenda

- **What is “Headless” architecture**
- **Why “Headless” architecture**
- **What can you do with Headless**
- **Headless Technology Stack**
- **Traditional rendering vs Headless rendering**
- **Page rendering approach - ATG/Endeca**
- **Security with Headless (RESTful Services)**
- **SEO with Headless**
 - HTML Snapshot Approach
 - Isomorphic Rendering
- **Benefits of going Headless**
 - Performance Metrics
 - Delivery Approach



Headless Architecture



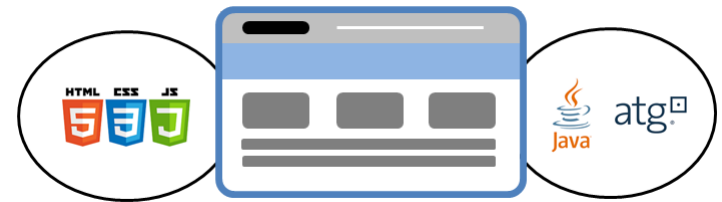
What is “Headless” architecture

The concept of Headless website is hardly new, and is more broadly known as running a Decoupled / API First Architecture.

The notion of a “headless” website refers to a situation where:

- There is a traditional database-driven backend used to maintain the content for the site
- The content for the site is accessible via web-service API, usually in a RESTful manner and in a mashup-friendly format such as JSON.
- The end-user experience is delivered by a JavaScript application rendering the output of this API into HTML, frequently making use of a modern application framework like Backbone, Ember, Angular, or Knockout.

In practice, this adds an additional layer of abstraction between end-user and website, extending an existing trend. The separation of content and display on back-end started over 10 years ago; the headless web continues this development.



Why “Headless” architecture

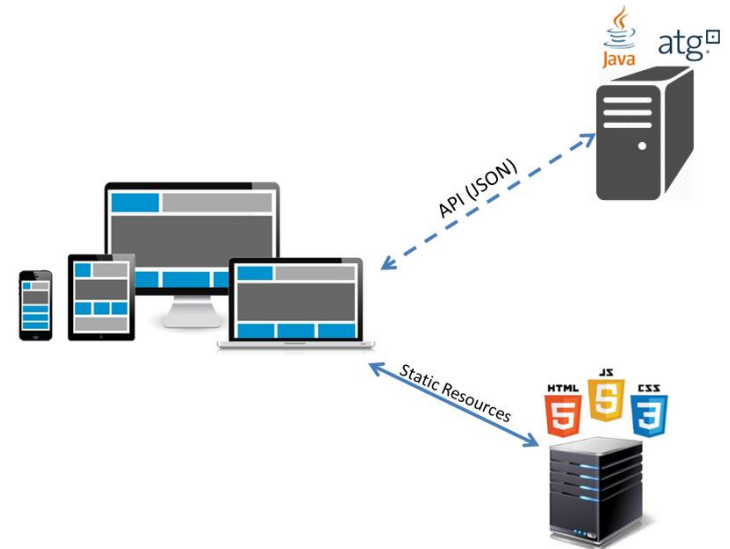
The Problem:

Traditional application design was device centric i.e. for each device channel separate set of UI (JSP) implementations were required with similar set of business rules. To launch a new channel it required more efforts to prepare UI and integrate with backend business logic.

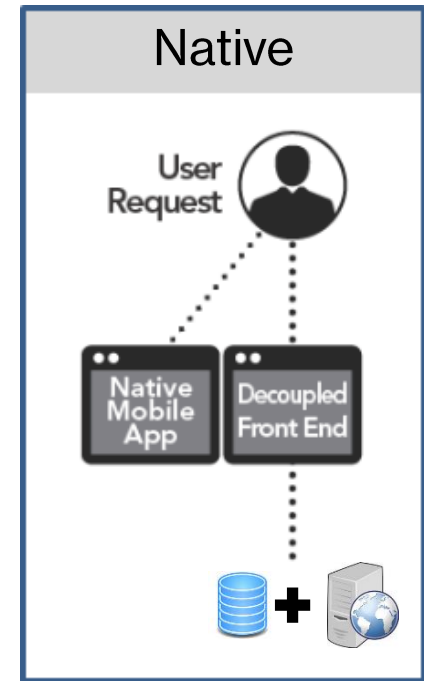
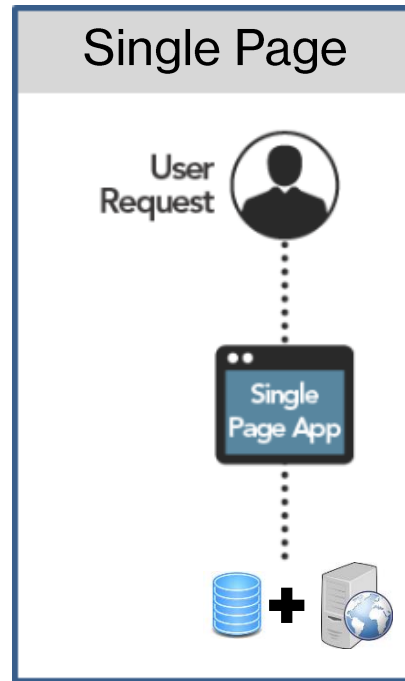
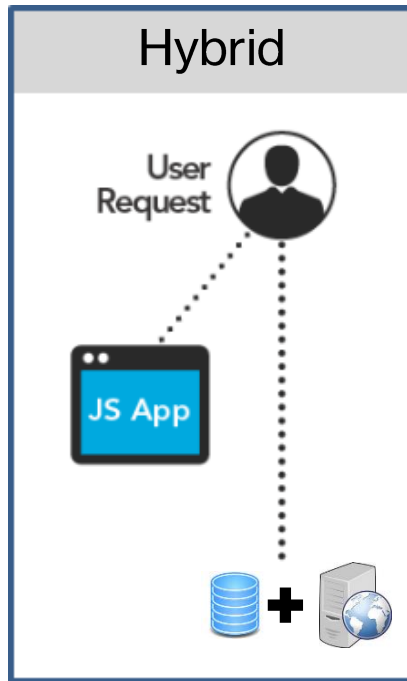
The solution:

Separating the UI and backend business logic with “separation of concern”.

- Build APIs for all the business functionalities and build the UI for the user experience for each channel using the API’s responses.



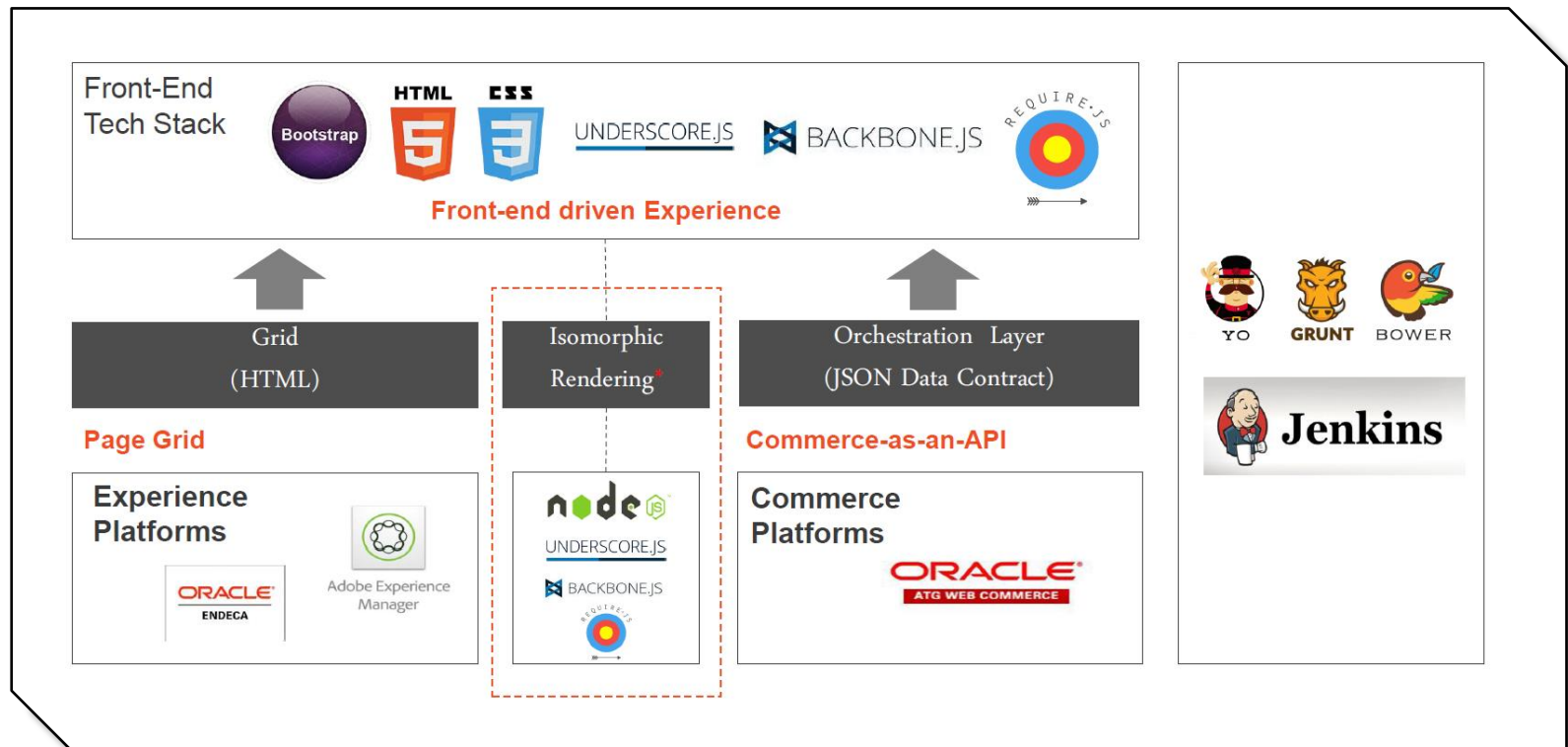
What can you do with Headless



- Hybrid: a portion of the website is highly interactive and communicates with the backend via API.
- Single page app: the end-user experience is a complete application-in-browser, usually leveraging one of the newer JS frameworks such as Angular, Ember, or Backbone.
- Native app: the frontend is a native mobile app. Users may or may not also interact directly with the website.



Headless Technology Stack

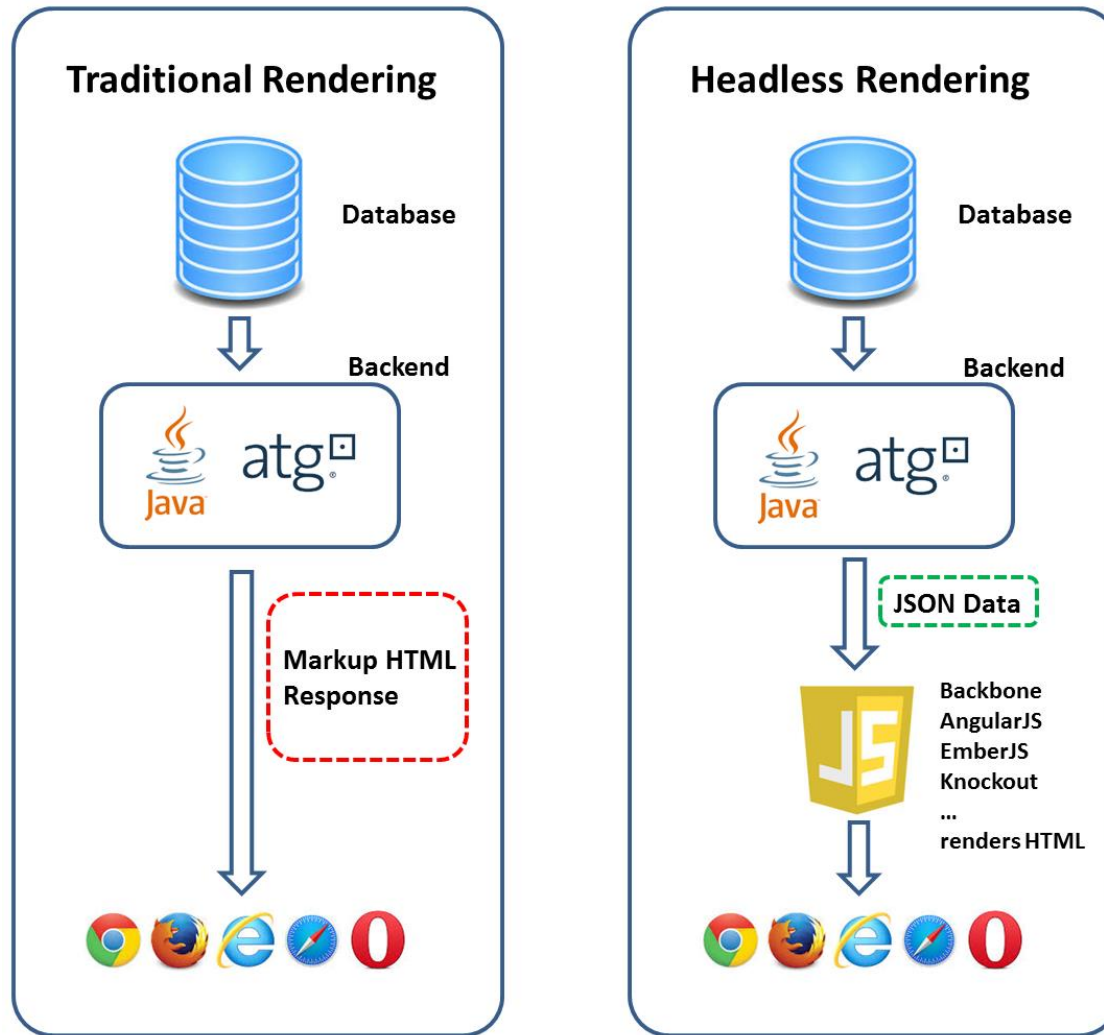


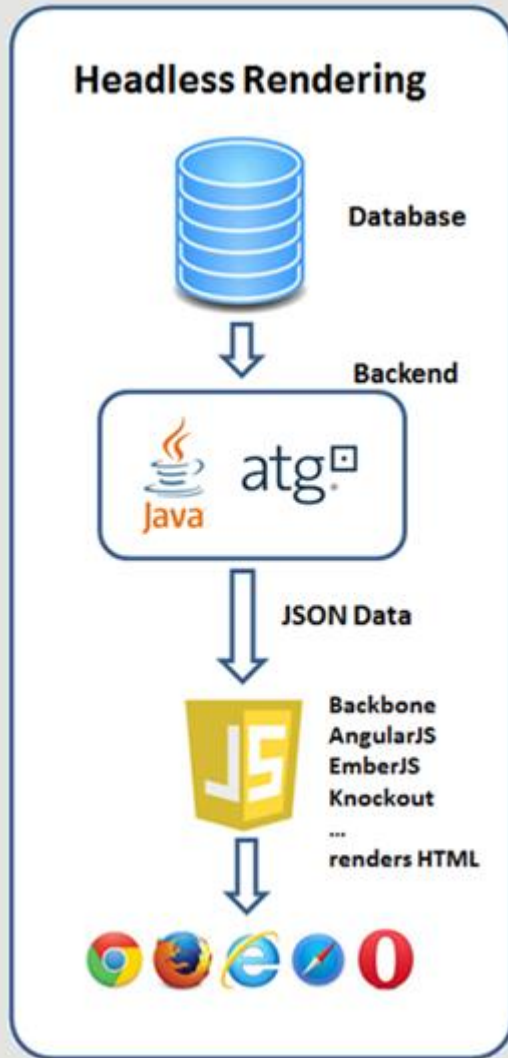
Headless Technology Stack ... contd

- Bootstrap – Bootstrap is the HTML, CSS, and JS framework for developing responsive, mobile first projects on the web
- UnderscoreJS – is JS Library that provides several functions that support function binding, javascript templating, creating quick indexes, deep equality testing as well as helpers such as map, filter, invoke.
- BackboneJS – is a JS Library based on MVP (Model View Presenter) design paradigm that gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.
- RequireJS – is a JS file and module loader. Helps improve the speed and quality of your code by providing ability to have discrete JS files/modules and still able to optimize code in just one or a few HTTP calls.
- NodeJS – is a platform built on Chrome's JS runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.



Traditional rendering vs Headless rendering





Backend Deals with:

- Content Modeling
- Storage
- Backup
- Authorization
- ...

Frontend deals with:

- HTML/CSS/JS
- Typography
- Layouts
- Sliders
- ...



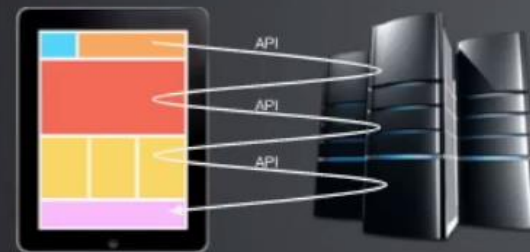
Traditional rendering vs Headless rendering . . . contd

Traditional Rendering



In the traditional rendering, complete page response (*HTML*) is fetched from backend servers

Headless Rendering



In Headless rendering, the response data (*JSON*) related to each individual component of the screen is fetched separately in the form of asynchronous requests



Traditional rendering vs Headless rendering . . . contd

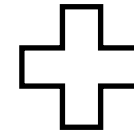
Traditional Rendering

```
<div id="container">
  <ul class="deviceDetails">
    <li class="item">
      <span>Manufacturer:</span>
      <span>Apple</span>
    </li>
    <li class="item">
      <span>Model:</span>
      <span>iPhone 6S</span>
    </li>
    <li class="item">
      <span>Color:</span>
      <span>Silver</span>
    </li>
    <li class="item">
      <span>Retail Price:</span>
      <span>$649.99</span>
    </li>
    <li class="item">
      <span>Capacity:</span>
      <span>16GB</span>
    </li>
  </ul>
</div>
```

Bytes sent from Server: **631**

Headless Rendering

```
var deviceDetail = {
  'Manufacturer': 'Apple',
  'Model': 'iPhone 6S',
  'Color': 'Silver',
  'Retail Price': '$649.99',
  'Capacity': '16GB'
}
```



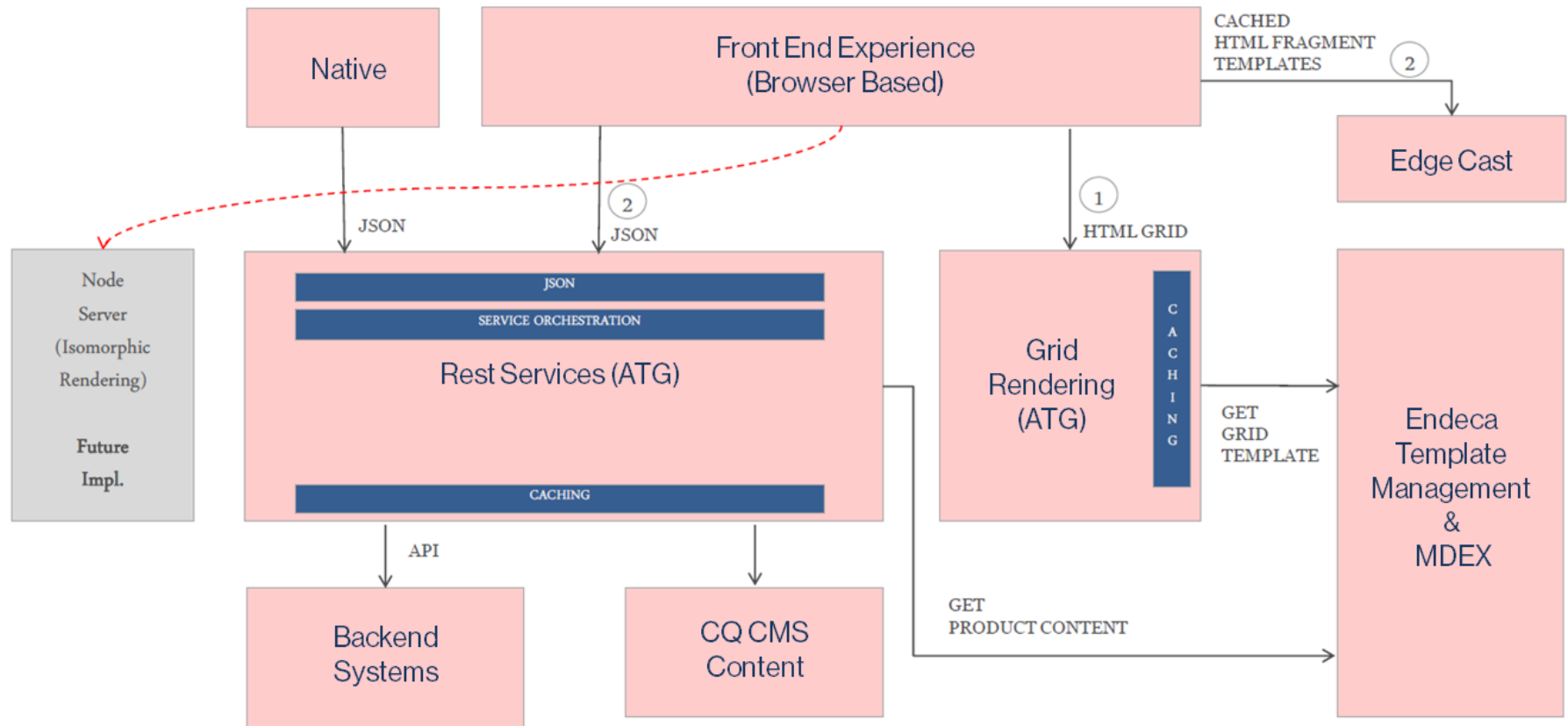
Client Side Templating:

**UnderscoreJS,
Moustache,
Handlebars, ...**

Bytes sent from Server: **152**



Page Rendering Approach – ATG/Endeca



Security with Headless (RESTful Services)

- ✓ **OOTB RESTSecurityManager using ACL (logged in or logged out users)**
We can use OOTB the REST Security Managers and ACL. This would control what services needs to exposed for logged in users vs anonymous users. Like AAL & EUP services can be controlled using this.
- ✓ **HTTPS vs HTTP services**
We can define what services needs to be secure vs what may not be. Like catalog look up can be on http while checkout related would be on HTTPS. We would maintain a list of HTTP and HTTPS services in a component and if a user tries to access HTTPS services on HTTP, it won't be served.
- ✓ **Not exposing the context /rest/**
We can write a wrapper module which will be invoked from the client. This module will only interact with the REST module. The idea is that the services cannot be invoked outside the firewall.
- ✓ **Token embedded in the header**
Each service call from the client can embed a token in a header that would be matched in the pipeline. If not matched access would be restricted.
- ✓ **Not exposing repository Services**
We should not expose any repository as a Service to avoid fetching data to the Nth level. Like printing the entire catalog.



SEO with Headless

In the past few years, AJAX has become extremely popular with web developers looking to create more dynamic and responsive experiences by exchanging small amounts of data with the web server. Often AJAX is used as a technique for creating interactive **Headless** web applications.

Exchanging data asynchronously requires the execution of scripts when particular page events occur.

The Problem: Search engines face challenges when it comes to interpreting or executing this kind of code. Content contained within an AJAX-enabled page was often not accessible nor indexed.

Solution:

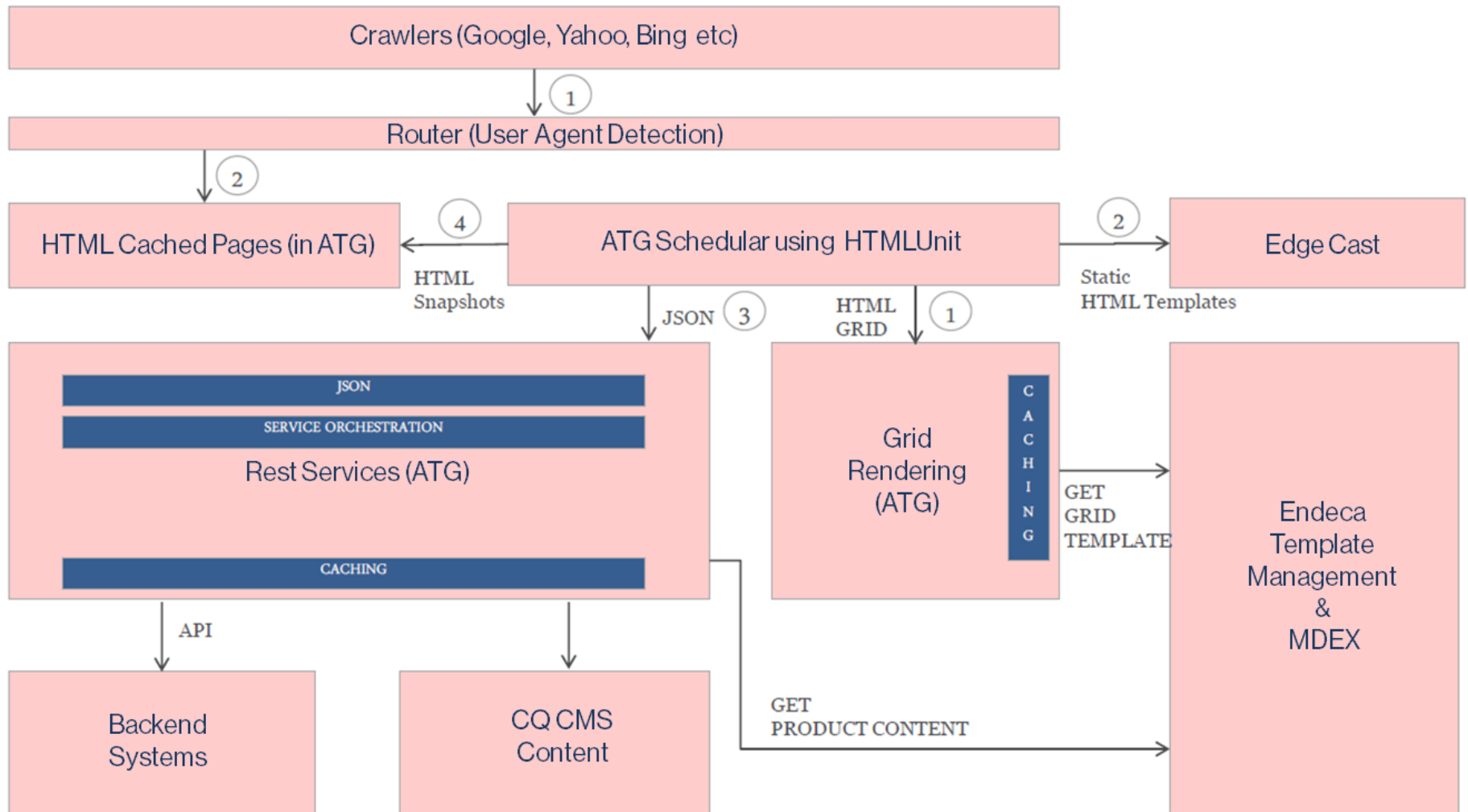
- **HTML Snapshot**

HTML Snapshot is the technique of rendering the precompiled HTML page for a URL when the request is detected from a Search Bot like GoogleBot, . . .

- **Isomorphic Rendering**



HTML Snapshot Approach



Isomorphic Rendering

Javascript was traditionally the language of the web browser, performing computations directly on a user's machine. This is referred to as "client-side" processing.

With the advent of Node.js, JavaScript has become a compelling "**server-side**" language as well, which was traditionally the domain of languages like Java, Python and PHP.

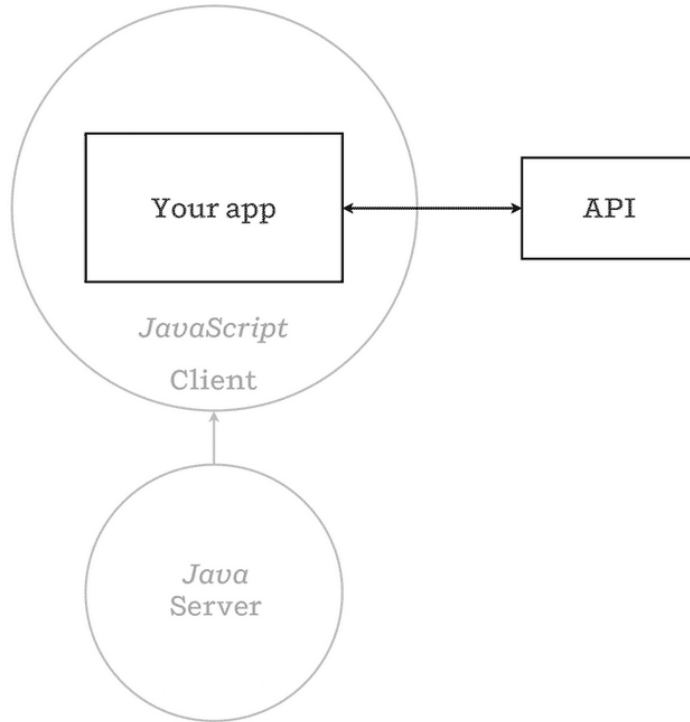
In web development, an isomorphic application is one whose code (in this case, JavaScript) can run both in the server and the client.

In an isomorphic application, the first request made by the web browser is processed by the server while subsequent requests are processed by the client

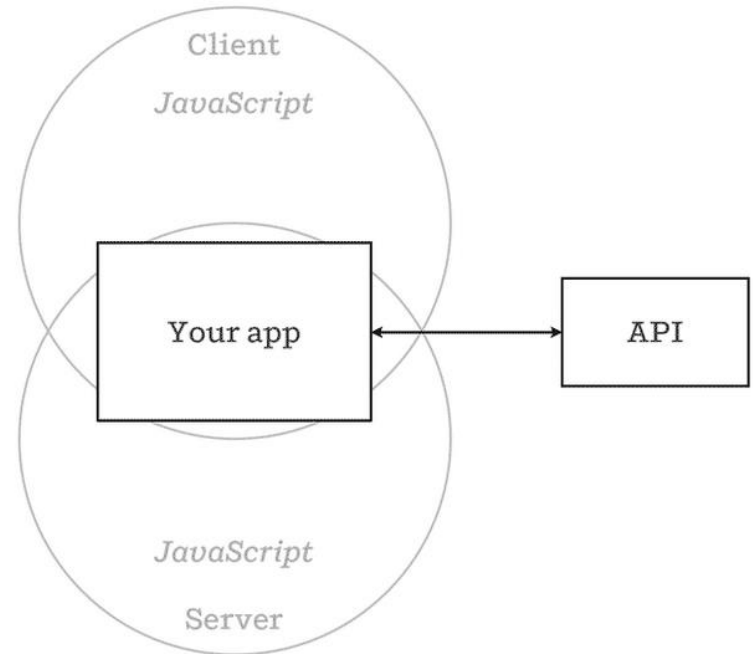


Isomorphic Rendering ... contd

Non-Isomorphic (Normal Approach)



Isomorphic Approach



Benefits of going Isomorphic:

- The first page request is fast and subsequent ones are even faster; as opposed to common SPAs, where the first request is used to load the application and then a round trip is made to fetch what was requested.
- SEO of a Headless Application is possible as JS is also executed at server-side and thereby content rendered from server can be directly indexed by Search-Bots.
- There is less code, as it is shared by both the client and the server



Benefits of going Headless

By shifting responsibility for the user experience completely into the browser, the headless model provides a number of benefits:

- **Value:** future-proof your website implementation, lets you design the site without re-implementing the backend.
- **Clear separation:** Sets frontend developers free from the conventions and structures of the backend. Headless development not only eliminates “div-itis”, it gives frontend specialists full control over the user experience using their native tools.
- **Speeds up the site:** by shifting display logic to the client-side and streamlining the backend. An application focussed on delivering content can be much more responsive than one that assembles completely formatted responses based on complex rules.
- **Faster, Cheaper, Better:** Simplify the process to build and operate great digital experiences by “separation of concern”.

Headless website development has the potential to unleash the creative power of frontend developers to deliver richer, faster, and more responsive user experiences.



Benefits of going Headless ... contd



Easy to Embed

Since the data is in a mashup friendly format like JSON, it is very easy to embed in existing application



Code Maintenance easy

With Headless there is separation of the logic and the presentation layer which makes maintenance of code easy



Works with all frameworks

Whether it comes to backend or frontend, Headless works with all frameworks. For the backend all it needs is a RESTful API and for the front end it needs JS templating mechanism



Performant

The main advantage of going Headless is performance. Since with Headless, the backend sends only the required data and not the markup, there is considerable difference in bytes transmitted over network



Secure

The RESTful API provides various security options and configurations to secure the responses

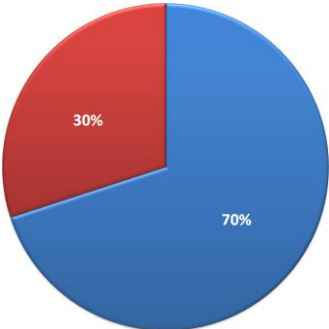
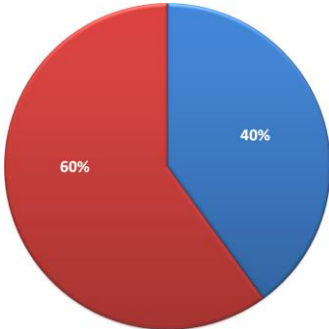
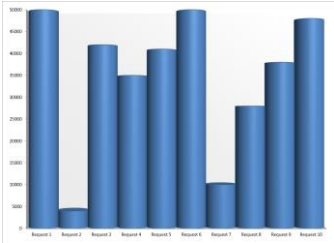
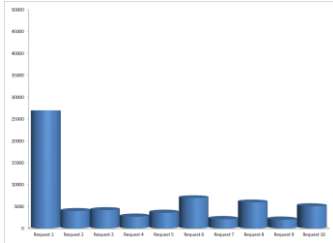


Customizable

Separation of concerns is one of the important aspects of going Headless. With a MVP architecture, you can customize your application as per your needs in very less time



Performance Metrics

Metric	Traditional Approach	Headless Approach	Legend
Code Execution :			<div><div></div> Server-Side</div> <div><div></div> Client-Side</div>
Bytes transferred per request :			<div><div></div> Request 1</div> <div>...</div> <div><div></div> Request n</div>
Rendering Speeds :	<div>Size: 1,506.47 KB Time: 10.22 seconds</div>	<div>Size: 203.92 KB Time: 7.15 seconds</div>	



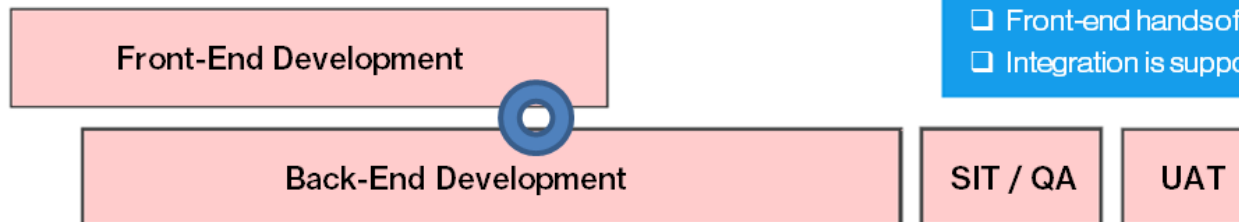
Delivery Approach (Traditional vs Headless)

Traditional Approach



- ☐ Front-end code is required to be broken down and converted into JSP
- ☐ This results in project churn between front-end and back-end teams determining browser specific and experience related bugs

Headless Approach



- ☐ Teams are able to work independently
- ☐ Front-end handoff complete browser tested code
- ☐ Integration is supported by JSON data contract



Questions ?



Thank you.

