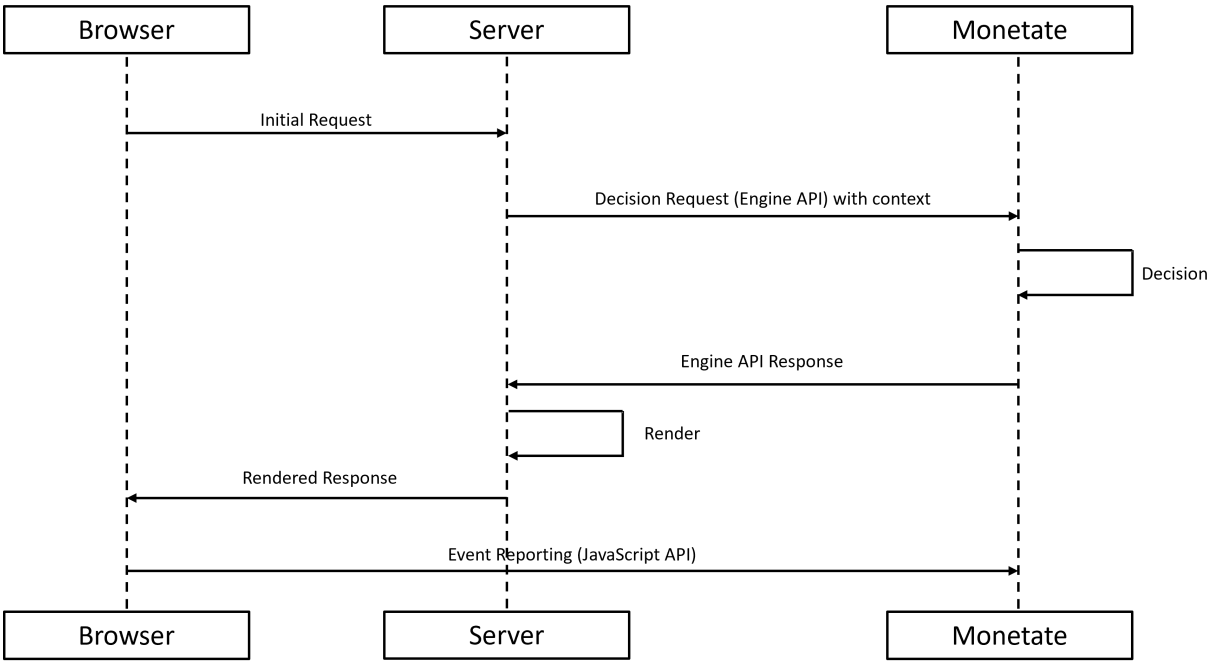
Types of Implementations

The Engine API was developed to be used across all channels, such as websites, mobile applications, client-telling apps, in-store displays, SMS services, or any other medium.

Hybrid Implementation

When used within a website, the Engine API should be implemented with the traditional Monetate tag-based approach. In this way the Monetate tag/JavaScript API can be relied on for all core reporting and analytics. Depending on your team's governance model, the Engine API can be used for some or all testing and personalization decisioning.

With hybrid implementations, the Engine API requests can be made entirely server-side, entirely within the browser, or some combination of the two. The exact implementation is dependent on your familiarity with your site architecture and how you wish to communicate with Monetate.



Regardless of whether the Engine API requests are made server-side or within the browser, the most important thing is to establish a governance model which controls when and where you do or do not use Monetate tag-based experiences (where actions run and can directly manipulate the DOM) or Omnichannel experiences (where the Engine API returns a decision and your site reacts to that decision to render the personalized content).

Considerations

- Monetate tag-based experiences allow the continued use of marketer-friendly tools (Action Builder, Target Builder, WYSIWYG editors, etc.) to directly manipulate the site with little to no knowledge of front-end Web development code and with little to no lead time.
- Omnichannel experiences require more development resources and time to ensure the correct context is

being passed and the correct logic is being applied to render the personalized content.
- Monetate tag-based experiences can sometimes experience flicker as the website and Monetate are "competing" to render the personalized content.
- Omnichannel experiences, when implemented properly, can completely eliminate the occurrence of flicker because of the ability to control what renders on the page and when.

FAQ

Q: How does Monetate ensure that sessions do not get double-counted when using a hybrid implementation (Engine API and Monetate tag/JavaScript API methods)?

A: So long as the `monetateID` (or `deviceId`) is persistent across both the Engine API and Monetate tag/JavaScript API implementations, Monetate reconciles that into a single session. Page view metrics are only recorded by the Monetate tag/JavaScript API, to prevent double counting.

Best Practices

- Implement JavaScript tag and API methods for reporting and analytics collection
- Implement Engine API requests when desired or necessary for decision requests and event recording (API events, managed impressions, recommendation item impressions, recommendation item clicks)
    - Depending on site architecture, Engine API requests may be made server-side, within the browser, or some combination of the two
- Establish an internal governance model for when or where to deploy experiences, such as in these examples:
    - Use Monetate tag solely for reporting and analytics and Engine API for all experiences
    - Use Engine API for high-impact, above-the-fold experiences and Monetate tag for lower impact
    - Define clear use cases for the agility of the Monetate tag versus the performance of Engine API
- If using a single-page application (SPA) framework, such as React or Vue, the same logic can be applied with the following additional considerations:
    - The JavaScript tag/API methods implementation requires retracks whenever the state of the page changes so that Monetate can re-evaluate for any tag-driven experiences and properly collect analytics
    - Engine API requests would be made from within the browser

Single-Page Application

Single-page applications, such as those built with React or Vue, can present unique challenges to how the traditional Monetate integration drives testing and personalization changes. While these challenges aren't insurmountable with just the tag-based approach, more clients pursue a hybrid implementation within their SPA.

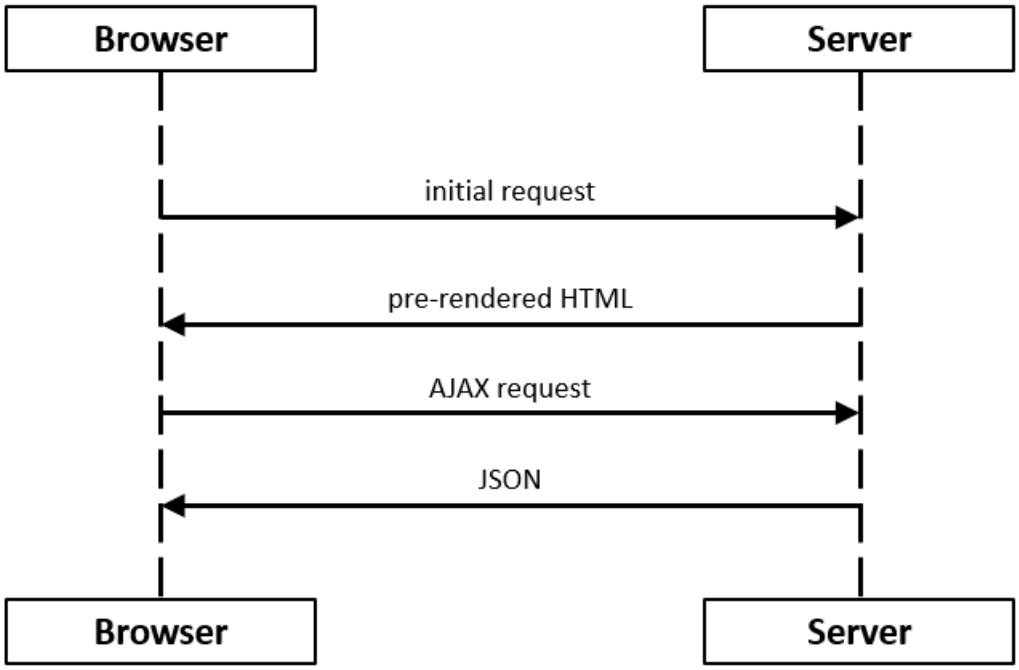
When using a hybrid implementation within an SPA, the Engine API may be called from the server on initial app load but from within the browser as the customer navigates the app or whenever the state changes.

Retracks on Engine API

- Every time something is sent to the Engine API, it's similar to a retrack. If a decision request is sent with a page view, it's similar to the track call (for example, selecting a color on a product details page).

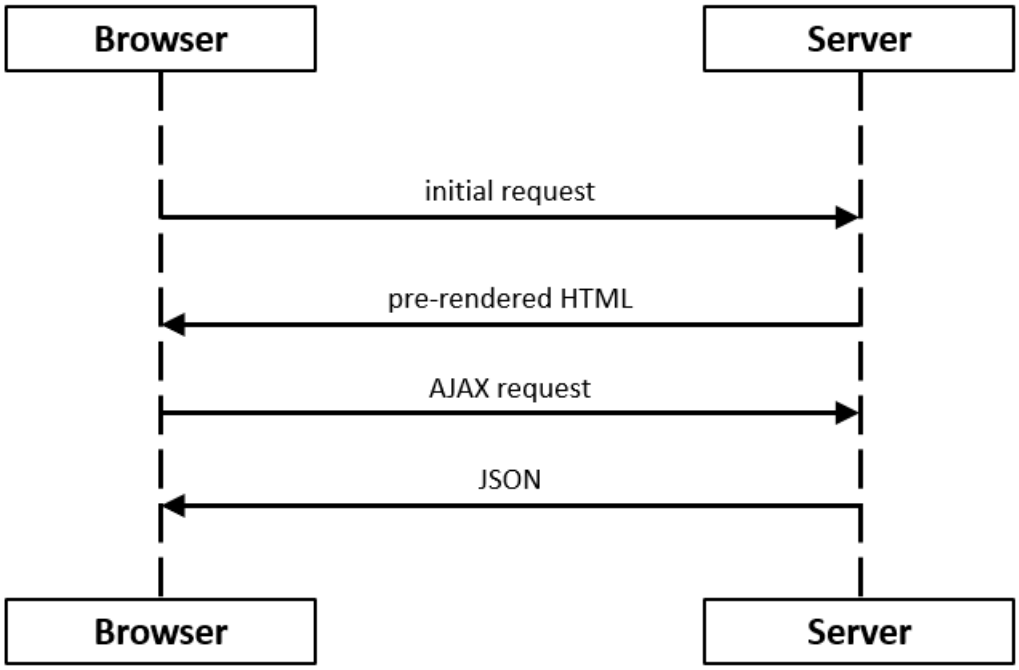
- A decision request can be sent without a page view (for example, just the color SKU), which only counts as single page view.

SPA Lifecycle



Pure Engine API

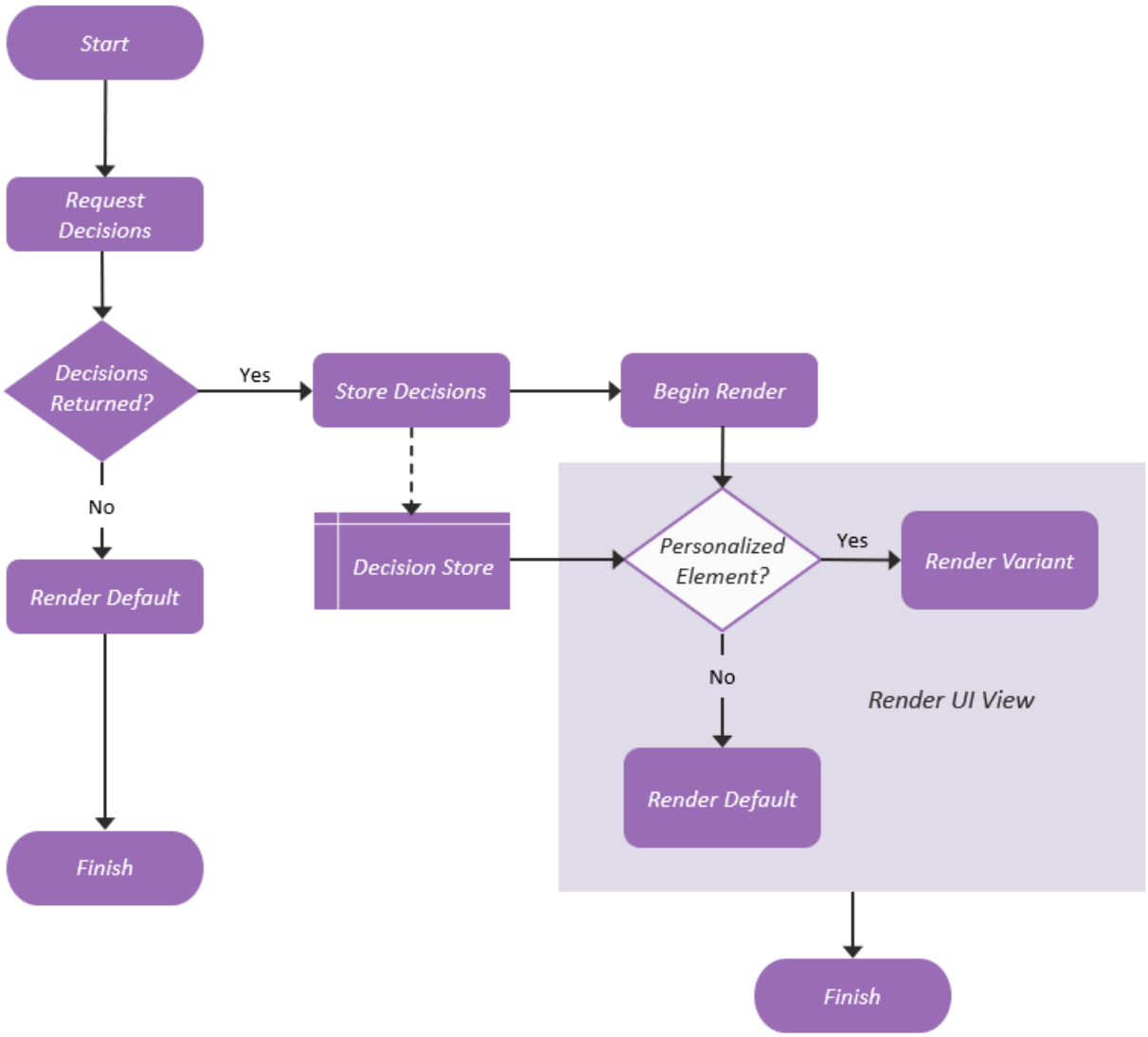
- Only the Engine API is implemented and is relied on for all decisioning and event reporting
- This method removes much of the "marketer friendliness" of the Monetate platform as the Builders are no longer available and no client-side actions can be created



Mobile Applications

It is recommend to take a "personalization-aware design" approach, where UI components check for personalization decisions prior to rendering, or render a loading indicator while awaiting the personalization decision. This personalization decision may involve replacing or otherwise overriding the components' default content, so it is advised to not display default content until the personalization decision has been received. If the application is structured in such a way that each UI component or structure can be checked for personalization overrides before rendering, code pushes simply to run experiences can be avoided.

- Identify default components and structures
- Identify which components and structures within the app can be personalized
- Create a store for personalization decisions that may override default components and structures
- Upon render, check the decision store for personalized components



- The importance of persisting a `monetateId` (or `deviceId`) remains
- Request must be made any time a user views a product (`monetate:context:ProductDetailView`), manages their cart contents (`monetate:context:Cart`) or makes a purchase (`monetate:context:Purchase`)—this enables Monetate to track this behavior for reporting/analytics as well as recommendation algorithms (as applicable)
- To test against specific device(s), include a testing custom variable by default

```
{
 "eventType": "monetate:context:CustomVariables",
 "customVariables": [
  {
   "variable": "qa_testing",
   "value": "true"
  }
 ]
}
```