Blinkist - Coding Challenge

Implementing A/B Testing in a Next.js Application

Christofer Nguyen

March 6, 2024

1 Introduction

This document outlines the development approach for integrating A/B testing into a blog platform, with the aim of determining the most effective content formats to inspire readers towards becoming users. The chosen tech stack for this project is Next.js, utilizing a hybrid rendering approach that combines Server-Side Rendering (SSR) and Client-Side Rendering (CSR) to serve dynamic content efficiently while maintaining high performance and scalability.

2 Choosing Next.js

Next.js was selected for this project due to its flexibility, performance, and developer experience. Key factors influencing this decision include:

- **Hybrid Rendering:** Next.js supports both SSR and CSR, allowing us to optimize our application's performance and user experience by choosing the most appropriate rendering method for each page or component.
- **SEO and Performance:** SSR in Next.js improves SEO by serving fully rendered pages to the browser, making content immediately crawlable by search engines. It also enhances the initial load time, which is critical for retaining users.
- **Developer Experience:** Features like file-based routing, automatic code splitting, and a wide range of plugins and integrations make development in Next.js fast and efficient.

3 Technical Approach

3.1 Hybrid Rendering

Our application leverages Next.js's hybrid rendering capabilities to serve the initial request using SSR for fast page loads and SEO benefits. Subsequent navigation and interactions utilize CSR, allowing users to interact with dynamic content without reloading the page. This approach ensures a seamless user experience while maintaining high performance.

3.2 A/B Testing Implementation

The A/B testing logic is implemented as follows:

1. **Variation Serving:** Upon the initial page load the client sends attaches to the cookie a user id. The server then responds by randomly selecting a variation of the content to serve to the user. This selection is stored on redis with the user id as key and the variation as value. This way, the user sees the same version even if the page is reloaded. While maintaining the server as the source of truth.

    - On the server check if there is a variation stored in redis by the unique user id. If not, randomly assign one (e.g., control or test) and save it to redis.

2. **Analytics Tracking:** Both page views and interactions with the buttons and links are tracked using the 'analytics-api.ts'. In this implementation version, external API calls will not be made. Instead, outputs will be logged directly to the browser's console. This data is crucial for determining the effectiveness of each content variation.

    - Integrate the provided analytics-api.ts for tracking. Ensure calls to the analytics API for both pageviews and buttons/links.
    - Attach an event handler to buttons and links to track clicks.

3. **Tracking Unique Interactions:** In order to make sure that the CTR are calculated correctly given what is known.

    - For page views: Each time a page loads, send a pageview event with the user id for the visitor.
    - For "Sign up" clicks: Track the event with the same user id and ensure that it's only counted once per user by checking against previously stored interactions.
        - The first click on the "Sign up" button by a user is tracked as an event. Store a flag on the cookie upon the first click to prevent counting additional clicks from the same user.
    - For other interactable clicks: Each time a click event occurs, send a event name with the user id for the visitor.
        - Assign a unique identifier (e.g., UUID) to each visitor and store this in the cookies. This ID is used to track the user's actions uniquely across their visit on the same device.

3.3 Assumptions

- **Technical Assumptions:**
  - The use of `cookies` for persisting the assigned variation and other user-specific data assumes clients have this capability enabled.
  - Without a user authentication system, it's assumed that the same user accessing the site from different devices will be treated as separate users.
  - The presence of an actual backend service capable of receiving, processing, and storing analytics events is assumed for the "imaginary API" but outside of the scope of this project.
  - Non-technical content editors can edit simple HTML and JSON files, given clear documentation or guidance.
- **Behavioral Assumptions:**
  - Visitors have given their consent for tracking, in compliance with GDPR and other privacy regulations.
  - Tracking clicks on the "Sign up" button is assumed to be the primary action of interest for determining the success of content variations.
  - Manual content edits by content editors are accurate and do not introduce errors affecting site display or functionality.

4 Guidance for Content Editors

Content editors can initiate A/B tests by editing simple JSON configurations, specifying the variations for a given article. Content editors can manipulate the content of the pages by editing the JSON files which exists on the server. By selecting which variation to edit (control or test), they can choose which variable to change.

```
{
  "control": {
    "title": "5 Books That Will Change Your Perspective",
    "signUpText": "Inspired to read more?
  },
  "test": {
    "title": "Learn fast and learn smart!",
    "signUpText": "Sign up today and explore a world of knowledge."
  }
}
```

5 Conclusion

The integration of A/B testing into the blog platform using Next.js represents a step towards creating a data-driven content strategy. By leveraging the hybrid rendering capabilities of Next.js, we have developed a solution that balances performance with dynamic content delivery, ultimately aiming to enhance user engagement and conversion rates. This approach follows the idea of A/B testing, where the goal is to compare the performance of different variations of content to see which one achieves better engagement.

6 Future Improvements

While the current implementation provides a basic foundation for A/B testing within a Next.js application, there are several areas where the solution can be further enhanced. Future improvements could include:

- **Enhanced Analytics:** Upgrade the analytics tracking to include more detailed metrics, such as time spent on page and scroll depth, to gain deeper insights into user engagement with content variations.
- **Cross-Device User Tracking:** Implement a more sophisticated method for tracking users across different devices to maintain a consistent experience and improve the accuracy of conversion tracking.
- **Content Editor Tools:** Enhance tools and interfaces for content editors, making it simpler to create and manage content variations without technical assistance.

7 GitHub Repository

The complete source code and additional documentation are available on GitHub at: `https://github.com/cncoding101/blinkist-coding-challenge`