# Table of Contents





+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

# Introduction

This document outlines a proposal from Docupal Demo, LLC to optimize the performance of Acme, Inc's Ember.js application. Our primary goal is to significantly improve application speed, reduce load times, and create a better user experience.

## The Need for Ember.js Optimization

Ember.js performance is critical for ACME-1. A fast and responsive application is essential for maintaining user engagement. It also supports complex user interactions and helps ACME-1 stay competitive in the market.

## Addressing Current Challenges

Currently, ACME-1 faces challenges with slow initial load times. Users are also experiencing sluggishness during UI interactions. Inefficient data handling further contributes to these performance issues. This proposal directly addresses these pain points through targeted optimization strategies. Our approach will focus on streamlining code, optimizing data flow, and leveraging Ember.js best practices to create a faster, more efficient application for ACME-1.

# Current Performance Assessment

ACME-1's Ember.js application performance has been thoroughly assessed. We used Ember Inspector, Chrome DevTools, and custom performance timers. These tools provided detailed insights into the application's behavior. Our assessment focused on identifying key bottlenecks and comparing performance against industry standards.

## Key Performance Indicators

Our analysis revealed several areas needing improvement. Excessive template re-renders significantly impact UI responsiveness. Inefficient data fetching slows down data loading. Large asset sizes increase initial load times.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

## Performance Benchmarks

| Metric | Current Value | Industry Standard |
|---|---|---|
| Initial Load Time | 4.5 seconds | 2.5 seconds |
| UI Responsiveness | 60 ms | 16 ms |
| CPU Usage (Idle) | 15% | 5% |

These figures highlight the gap between the current state and optimal performance.

## Detailed Findings

### Template Re-renders

The application experiences a high number of unnecessary template re-renders. This leads to increased CPU usage and reduced UI responsiveness. Optimizing component rendering and data binding strategies are crucial.

### Data Fetching

Current data fetching methods are not efficient. Multiple requests for small amounts of data cause delays. Implementing more efficient data fetching strategies, such as batching or GraphQL, can improve performance.

### Asset Sizes

Large asset sizes contribute to slow initial load times. Unoptimized images and excessive JavaScript code are the main culprits. Compressing assets and implementing code splitting can reduce asset sizes.

### Comparison to Industry Standards

The application lags behind industry standards in initial load time and UI responsiveness. This impacts user experience and can lead to user frustration. Addressing the identified bottlenecks is essential to meet industry benchmarks.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

# Rendering Optimization Strategies

Efficient rendering is crucial for ACME-1 to deliver a smooth and responsive user experience. Several strategies can be employed to optimize Ember.js rendering without compromising functionality.

## Code Splitting and Route Optimization

We propose splitting the application code into smaller, manageable chunks. This approach reduces the initial load time, as the browser only downloads the code necessary for the current route. Route optimization ensures that transitions between different sections of the application are seamless and quick, enhancing perceived performance. Lazy loading assets like images and components that are not immediately visible will further improve initial load times.

## Minimizing Template Re-renders

Unnecessary template re-renders can significantly impact performance. We recommend implementing the following best practices:

- **Immutable Data:** Utilize immutable data structures to ensure that changes trigger re-renders only when necessary.
- **Trackable Properties:** Employ trackable properties to precisely control when components update, avoiding unnecessary re-renders.
- **Avoid Unnecessary Observers:** Carefully evaluate the need for observers, as they can trigger frequent updates. Consider alternative approaches like computed properties where appropriate.

## Ember Run-Loop Management

The Ember run-loop manages the execution of tasks within the application. Inefficient management of the run-loop can lead to delays and performance bottlenecks. To optimize the run-loop, we suggest the following:

- **Debouncing:** Debounce actions that trigger frequent updates, such as user input, to avoid excessive rendering.
- **Throttling:** Throttling limits the rate at which a function is executed, preventing performance issues caused by rapid updates.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

- **Run-Loop Awareness:** Understand how Ember's run-loop works to schedule tasks efficiently, and avoid long-running operations within the run-loop.

# Data Management and Ember Data Optimization

Data management is crucial for ACME-1's Ember.js application performance. We will focus on efficient data fetching, caching strategies, and Ember Data optimization.

## Efficient Data Fetching

We recommend using JSON:API with included resources to reduce network requests. GraphQL is another strong option, allowing clients to request only the necessary data. This avoids over-fetching and improves load times.

This chart shows the improvement in data load times using JSON:API and GraphQL compared to the current implementation. The numbers represent milliseconds.

## Caching Strategies

Implementing robust caching mechanisms will significantly improve responsiveness. We will leverage in-memory caching for frequently accessed data. This reduces the need to fetch data from the server repeatedly. Ember Data's built-in caching can be configured to suit ACME-1's specific needs. We will explore additional caching layers like local storage or service worker caching for further optimization.

## Ember Data Optimization

Optimizing Ember Data configurations is key to minimizing overhead. We will fine-tune adapter and serializer settings to reduce the amount of data transferred. Efficient relationship management is also essential. We will ensure that relationships are correctly defined and loaded only when needed. This prevents unnecessary data fetching and improves application performance. Furthermore,



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

payload size optimization can be achieved through techniques such as field selection and data compression. We will conduct a thorough analysis of ACME-1's data models to identify areas for improvement.

# Build and Deployment Optimization

Effective build and deployment strategies are vital for optimizing Ember.js application performance. We will focus on key areas to improve speed and efficiency for ACME-1.

## Build Optimization

We will enhance your application's build process. Minification will reduce the size of your JavaScript and CSS files. Tree shaking will eliminate unused code, leading to smaller bundles. Utilizing production builds ensures that development-specific code is excluded, further reducing the application's footprint. These steps combined result in faster load times and improved overall performance.

## Asset Management

Optimizing how assets are handled is critical. Image optimization techniques, such as compression and appropriate file formats, will reduce image sizes without sacrificing quality. We will implement Content Delivery Networks (CDNs) to distribute assets globally, reducing latency for users regardless of their location. Furthermore, we will leverage browser caching to store assets locally, minimizing the need to re-download them on subsequent visits.

## FastBoot for Server-Side Rendering

FastBoot will be integrated to improve server-side rendering. This offers several benefits, including improved Search Engine Optimization (SEO) as search engines can crawl rendered content more easily. Users will experience a faster initial render, leading to a better perceived performance. First-time visitors will benefit from a fully rendered page served directly from the server, improving their initial experience and reducing the time to interactive.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

### Further Build Process Improvements

To further optimize the build process, we will explore advanced techniques. Code splitting will divide the application into smaller chunks, allowing browsers to download only the necessary code for each route or feature. This reduces the initial load time and improves responsiveness. By implementing these strategies, ACME-1 can achieve faster startup times and a more efficient application.

# Asynchronous and Lazy Loading Enhancements

Asynchronous and lazy loading techniques can significantly improve ACME-1's Ember.js application performance. We propose strategies to reduce initial load times and enhance responsiveness by loading resources only when they are needed.

### Asynchronous Rendering

Ember.js provides several APIs to facilitate asynchronous rendering. We recommend leveraging RSVP.hash to manage multiple promises within routes, ensuring data dependencies are resolved before rendering components. Additionally, Ember.run.later can defer non-critical UI updates, preventing blocking of the main thread. The {{await}} helper offers a clean and declarative way to handle asynchronous values directly in templates. Implementing these techniques will prevent long loading times.

### Route-Based Lazy Loading

We will implement route-based lazy loading to load modules and components only when a specific route is accessed. This approach minimizes the initial payload size. The Ember Router will be configured to load route definitions and associated assets on demand.

### Deferred Asset Loading

Non-critical assets, such as images and fonts, will be loaded using lazy loading techniques. This involves loading these assets only when they are about to become visible in the viewport. We will use libraries like ember-lazy-load or native browser



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

APIs like IntersectionObserver to efficiently manage deferred asset loading.

### Addressing Potential Pitfalls

Implementing deferred loading requires careful consideration to avoid common pitfalls. We will prioritize loading critical resources to ensure a usable initial experience. User experience will be closely monitored to prevent delays. Robust error handling will be implemented to gracefully manage loading failures.

# Benchmarking and Metrics for Continuous Improvement

To ensure sustained performance gains, ACME-1 requires ongoing monitoring and regular benchmarking. This section outlines the key performance indicators (KPIs) and methods Docupal Demo, LLC will use to track improvements and identify areas for further optimization.

### Key Performance Indicators (KPIs)

Docupal Demo, LLC will monitor the following KPIs:

- **Page Load Time:** The time it takes for a page to fully load.
- **Time to First Interaction:** The time it takes for a user to be able to interact with the page.
- **Frames Per Second (FPS):** A measure of the smoothness of animations and transitions.

### Monitoring and Benchmarking

Docupal Demo, LLC will benchmark performance regularly, such as monthly or after major feature releases. This allows for tracking the impact of changes and identifying regressions. Performance data will be visualized using area charts to easily identify trends and anomalies over time.

### Tools for Automated Monitoring

Docupal Demo, LLC will leverage tools to automate performance monitoring and reporting. These tools include:



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

- **New Relic:** A comprehensive monitoring platform for web applications.
- **Datadog:** A monitoring and analytics platform for cloud-scale applications.
- **Custom Monitoring Scripts:** Tailored scripts to collect specific performance data relevant to ACME-1's Ember.js application.

These tools will provide real-time insights into application performance, enabling proactive identification and resolution of issues.

# Conclusion and Recommendations

Our analysis identified key areas for performance improvement within ACME-1's Ember.js application. We propose a focused optimization strategy to address these areas.

## Prioritized Actions

We recommend prioritizing the following actions:

- **Optimize data fetching:** Improve the efficiency of data retrieval processes.
- **Reduce template re-renders:** Minimize unnecessary re-rendering of templates.
- **Implement lazy loading:** Load resources only when needed to reduce initial load time.

## Tracking and Reporting

To ensure transparency and track progress, we will use project management tools. Regular performance reports will provide detailed insights into the impact of the optimizations. This will include metrics such as page load times and rendering performance.

## Long-Term Maintenance

Sustained performance requires ongoing effort. We suggest the following for long-term maintenance:

- **Continuous monitoring:** Proactively identify and address potential performance bottlenecks.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

- **Regular code reviews:** Ensure code quality and adherence to performance best practices.
- **Stay updated:** Keep up-to-date with the latest Ember.js updates and best practices.

# Appendix and References

## Supporting Resources

This proposal is supported by several key resources within the Ember.js ecosystem and broader web development community. These resources provide detailed information and best practices that underpin the optimization strategies outlined.

- Ember.js Official Documentation: https://emberjs.com/api/
- Ember.js Guides: https://guides.emberjs.com/
- Ember Observer: https://emberobserver.com/

## Code Samples and Configurations

Examples of optimized Ember.js components, efficient data fetching techniques, and recommended build configurations are available upon request. These code snippets will illustrate the practical implementation of the proposed performance enhancements.

## External Tools and Libraries

The following external tools and libraries are recommended to support the optimization efforts:

- Lodash: A JavaScript utility library delivering modularity, performance & extras.
- Moment.js: A JavaScript date library for parsing, validating, manipulating, and formatting dates.
- Image Optimization Tools: Tools for compressing and optimizing images to reduce file sizes and improve loading times.



+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country

## Additional Information

Supplementary information, including references to Ember.js documentation, profiling tools, and relevant articles on web performance optimization, can be provided as needed. Profiling tools like Ember Inspector and browser developer tools will be used to measure the impact of optimizations.





+123 456 7890
+123 456 7890
info@website.com
websitename.com
P.O. Box 283 Demo
Frederick, Country