Table of Contents

Introduction and Background 3
- Current Application State 3
- Migration Drivers 3
- Project Objectives 4

Current Architecture and Codebase Analysis 4
- Architecture Overview 4
- Dependency Analysis 4
- Codebase Health and Technical Debt 5

Migration Strategy and Approach 5
- Incremental Migration Methodology 5
- Risk Mitigation 6
- Tools and Frameworks 6
- Tentative Timeline 7

Compatibility and Dependency Management 7
- Ember.js and Addon Versions 7
- Third-Party Dependencies 7
- Backend and Service Integrations 8

Risk Assessment and Mitigation 8
- Potential Risks 8
- Mitigation Strategies 8
- Impact and Likelihood 9
- Contingency Plans 9

Testing and Quality Assurance Plan 10
- Testing Frameworks and Tools 10
- Testing Types 10
- Quality Measurement and Tracking 10

Implementation Roadmap and Timeline 11
- Phase 1: Assessment (2 weeks) 11
- Phase 2: Upgrade Core (4 weeks) 11
- Phase 3: Feature Migration (8 weeks) 11
- Phase 4: Testing & Optimization (4 weeks) 11

Stakeholder Communication and Training 12
- Communication Plan 12

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

Training Program ---- 13
Post-Migration Support and Maintenance ---- **13**
Issue Tracking and Resolution ---- 13
Maintenance Schedule ---- 13
Performance and Stability Monitoring ---- 14
Conclusion and Next Steps ---- **14**
Key Takeaways ---- 14
Immediate Actions ---- 14
Measuring Success ---- 14

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

Introduction and Background

This document outlines a proposal from Docupal Demo, LLC for migrating your existing Ember.js application to a more modern and supported version. Our goal is to provide a clear path for upgrading your application, ensuring its long-term maintainability, performance, and access to the latest features.

Current Application State

Your application is currently running on Ember.js version 2.18, utilizing the Ember CLI build tool. While this version has served its purpose, it is now outdated and no longer receives long-term support (LTS) from the Ember.js core team.

Migration Drivers

Several key factors necessitate this migration:

- **LTS Requirement:** Maintaining an application on an unsupported Ember.js version poses security risks and limits access to critical bug fixes and security patches. Upgrading to a supported LTS version (4.x) ensures ongoing stability and security.
- **Performance Improvements:** Newer Ember.js versions include significant performance enhancements, leading to a faster and more responsive user experience.
- **Modern Features:** Upgrading unlocks access to modern Ember.js features and best practices, allowing for more efficient development and improved code maintainability.
- **Reduced Technical Debt:** By migrating, we can address accumulated technical debt and improve the overall architecture of the application, making it easier to maintain and extend in the future.

Project Objectives

The primary objectives of this migration project are:

- **Upgrade to Ember.js 4.x:** The immediate goal is to successfully migrate the application to a supported Ember.js 4.x version.

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

- **Improve Application Performance:** We aim to optimize the application's performance, resulting in faster load times and a smoother user experience.
- **Reduce Technical Debt:** Refactoring and modernizing the codebase will reduce technical debt, making the application more maintainable and scalable.

Current Architecture and Codebase Analysis

Our analysis of the current application provides a clear picture of its architecture, dependencies, and overall code health. This understanding is crucial for planning an effective migration strategy.

Architecture Overview

The application primarily follows the Model-View-Controller (MVC) architectural pattern. We also see the adoption of component-based architecture in certain areas. This hybrid approach suggests a gradual shift towards componentization, which is a positive trend for maintainability. However, inconsistencies in applying these patterns could lead to increased complexity.

Dependency Analysis

The application relies on Ember.js version 2.18. This version is significantly outdated, missing many performance improvements and features available in newer Ember versions. We also noted the use of Ember Data for data management and jQuery for DOM manipulation. A detailed list of all dependencies, including community addons and their respective versions, is available in the package.json file. Reviewing these dependencies is important to identify potential compatibility issues during the migration.

Codebase Health and Technical Debt

Our assessment revealed several pain points and areas of technical debt.

- **Outdated Dependencies**: Using Ember.js 2.18 means the application is missing out on modern features and performance enhancements. It also increases the risk of security vulnerabilities.

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

- **Performance Bottlenecks**: Certain complex components exhibit performance issues. These bottlenecks likely stem from inefficient rendering or data handling.
- **Lack of Automated Testing**: The existing test coverage is insufficient. This makes it difficult to ensure the application's stability during and after the migration.

The chart visualizes the distribution of technical debt across key areas. Addressing these issues will be a key focus of the migration process.

Migration Strategy and Approach

We propose an incremental migration strategy for DocuPal Demo, LLC's Ember.js application. This approach minimizes disruption and risk. It allows for a gradual transition to a modern Ember.js version. We will focus on upgrading the application route by route. This ensures that existing functionality remains stable while new features are developed on the updated framework.

Incremental Migration Methodology

Our incremental migration will follow these steps:

1. **Assessment and Planning:** A thorough review of the current application. Identification of dependencies, deprecated features, and potential migration challenges. We will create a detailed migration plan. This plan will outline the order of route upgrades and resource allocation.
2. **Ember CLI Update:** Utilize Ember CLI to update the application's core dependencies. This includes Ember.js, Ember Data, and other essential packages. We will address any compatibility issues that arise during the update process.
3. **Route-by-Route Upgrades:** Each route will be upgraded independently. This involves updating templates, components, and controllers to align with the latest Ember.js conventions. Feature flags will be used to toggle between the old and new versions of each route. This allows for controlled testing and rollback if necessary.
4. **Component Modernization:** Refactor existing components to improve performance and maintainability. We will leverage modern Ember.js features such as tracked properties and native classes.

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

5. **Testing and Quality Assurance:** Rigorous testing will be conducted after each route upgrade. This includes unit tests, integration tests, and end-to-end tests. We will use automated testing tools to ensure code quality and prevent regressions.
6. **Deployment and Monitoring:** Upgraded routes will be deployed to a staging environment for user acceptance testing. Once approved, the changes will be rolled out to production. We will closely monitor the application's performance and stability after each deployment.
7. **Rollback Plan:** We will ensure that we have a rollback plan for each route upgrade in case of any issues. This will allow us to quickly revert to the previous version if necessary.

Risk Mitigation

We will minimize risks and downtime through:

- **Feature Flags:** To control the release of new features and allow for easy rollback.
- **Comprehensive Testing:** To identify and fix issues before they impact users.
- **Rollback Plans:** To quickly revert to the previous version if necessary.

Tools and Frameworks

We will leverage the following tools and frameworks to support the migration:

- **Ember CLI:** For managing the application's build process and dependencies.
- **Ember Observer:** For assessing the compatibility of Ember.js add-ons.
- **ember-cli-update:** For automating the update process.
- **VS Code with Ember Language Server:** For code editing and debugging.

Tentative Timeline

The following table provides a high-level timeline for the migration project:

Phase	Duration	Start Date	End Date
Assessment and Planning	2 weeks	2025-08-26	2025-09-09
Ember CLI Update	1 week	2025-09-10	2025-09-17
Route-by-Route Upgrades	12 weeks	2025-09-18	2025-12-10

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

Phase	Duration	Start Date	End Date
Component Modernization	8 weeks	2025-12-11	2026-02-05
Testing and Quality Assurance	4 weeks	2026-02-06	2026-03-06
Deployment and Monitoring	Ongoing	2026-03-07	Ongoing

Compatibility and Dependency Management

The migration to Ember.js 4.12 (LTS) requires careful consideration of compatibility and dependency management. Our approach prioritizes a stable and functional application throughout the upgrade process.

Ember.js and Addon Versions

We will target Ember.js version 4.12, an LTS (Long Term Support) release, ensuring stability and continued support. A key aspect involves updating Ember Data and other essential addons to versions compatible with Ember.js 4.12. This includes a thorough review of each addon's compatibility matrix and upgrade guides. We will document the specific versions of each addon that are compatible and will be used in the migrated application.

Third-Party Dependencies

Each third-party dependency will undergo a detailed evaluation. We will assess compatibility with Ember.js 4.12. Where possible, dependencies will be updated to compatible versions. If a dependency is incompatible and no update is available, we will explore suitable replacements that offer similar functionality. This ensures minimal disruption and continued feature parity.

Backend and Service Integrations

Potential integration challenges with backend APIs and other services are anticipated. The primary concern is API version compatibility. To mitigate this, we will implement thorough integration testing throughout the migration. This testing will focus on ensuring that the updated Ember.js application interacts correctly with

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

all backend services. Any necessary adjustments to API calls or data structures will be identified and addressed promptly. We will document all integration points and testing results.

Risk Assessment and Mitigation

Migrating to a newer Ember.js version carries inherent risks. We have identified potential issues that could impact the project timeline and the quality of the final product. We will actively monitor and control these risks throughout the migration process.

Potential Risks

The primary risks associated with this Ember.js migration are:

- **Dependency Conflicts:** Updating Ember.js often requires updating its dependencies. These updates can introduce conflicts between different packages, leading to application errors.
- **Unexpected Breakages:** Even with thorough planning, changes in Ember.js or its dependencies can cause unexpected breakages in existing application functionality.
- **Performance Regressions:** New Ember.js versions might introduce changes that negatively impact the application's performance. This could result in slower loading times or reduced responsiveness.

Mitigation Strategies

To minimize the impact of these risks, we will implement the following strategies:

- **Continuous Integration:** We will use a continuous integration (CI) system to automatically build and test the application after each code change. This allows us to quickly identify and address dependency conflicts and unexpected breakages.
- **Monitoring Tools:** We will use monitoring tools to track the application's performance before and after the migration. This will help us identify and resolve any performance regressions.
- **Regular Code Reviews:** We will conduct regular code reviews to ensure that all code changes are thoroughly tested and adhere to best practices. This will help prevent the introduction of new bugs and improve the overall quality of the

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

codebase.

- **Detailed Migration Plan:** The migration will be broken down into smaller, manageable steps. Each step will be carefully planned and executed to minimize the risk of introducing errors.
- **Comprehensive Testing:** We will perform extensive testing throughout the migration process. This will include unit tests, integration tests, and end-to-end tests to ensure that all application functionality is working as expected.

Impact and Likelihood

The following chart visualizes the potential risks based on their impact and likelihood:

Contingency Plans

Despite our best efforts, some risks may still materialize. In such cases, we have developed contingency plans to minimize their impact:

- **Rollback Plan:** If a major issue is discovered after a migration step, we will have a rollback plan in place to quickly revert to the previous version of the application.
- **Escalation Process:** If a risk cannot be resolved by the development team, it will be escalated to senior management for further assistance.

By proactively identifying and mitigating these risks, we aim to ensure a smooth and successful Ember.js migration.

Testing and Quality Assurance Plan

We will implement a comprehensive testing and quality assurance plan. This plan will ensure the stability and reliability of the application during and after the Ember.js migration. Our strategy includes multiple layers of testing, focusing on identifying and resolving issues early in the development lifecycle.

Testing Frameworks and Tools

We will use established testing frameworks to maintain code quality. Ember QUnit will be our primary unit testing framework. We will also use Ember ⊠⊠⊠⊠⊠⊠⊠ for integration and end-to-end testing. These tools offer robust features for writing and

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

running tests, ensuring comprehensive test coverage.

Testing Types

Our testing strategy will cover three key areas:

- **Unit Testing:** We will test individual components and functions in isolation. This will verify that each part of the application works correctly.
- **Integration Testing:** We will test the interactions between different parts of the application. This will ensure that components work together as expected.
- **End-to-End (E2E) Testing:** We will simulate real user scenarios to test the application's overall functionality. This will validate the entire user experience from start to finish.

Quality Measurement and Tracking

We will use several metrics to measure and track the quality of the migration:

- **Code Coverage:** We will monitor the percentage of code covered by tests. This will help us identify areas that require more testing.
- **Performance Metrics:** We will track key performance indicators, such as page load times and response times. This will ensure that the application performs efficiently.
- **User Feedback:** We will collect feedback from users to identify any issues or areas for improvement. This will help us to fine-tune the application and improve user satisfaction. We will implement mechanisms for collecting and analyzing user feedback throughout the migration process.

By combining these testing approaches and quality assurance measures, we aim to deliver a stable and reliable Ember.js application.

Implementation Roadmap and Timeline

Our Ember.js migration will proceed in four distinct phases. Each phase has defined objectives, deliverables, and resource allocations. A dedicated team consisting of four developers, a QA engineer, and a project manager will handle the migration.

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

Phase 1: Assessment (2 weeks)

The initial phase involves a thorough assessment of the current Ember.js application. We will analyze the codebase, dependencies, and existing infrastructure. This assessment will identify potential risks and challenges associated with the migration. The key deliverable is a detailed migration plan.

Phase 2: Upgrade Core (4 weeks)

This phase focuses on upgrading the core Ember.js framework and its core dependencies. We will address any breaking changes and ensure the application's basic functionality remains intact. This phase is critical and must be completed before feature migrations can begin.

Phase 3: Feature Migration (8 weeks)

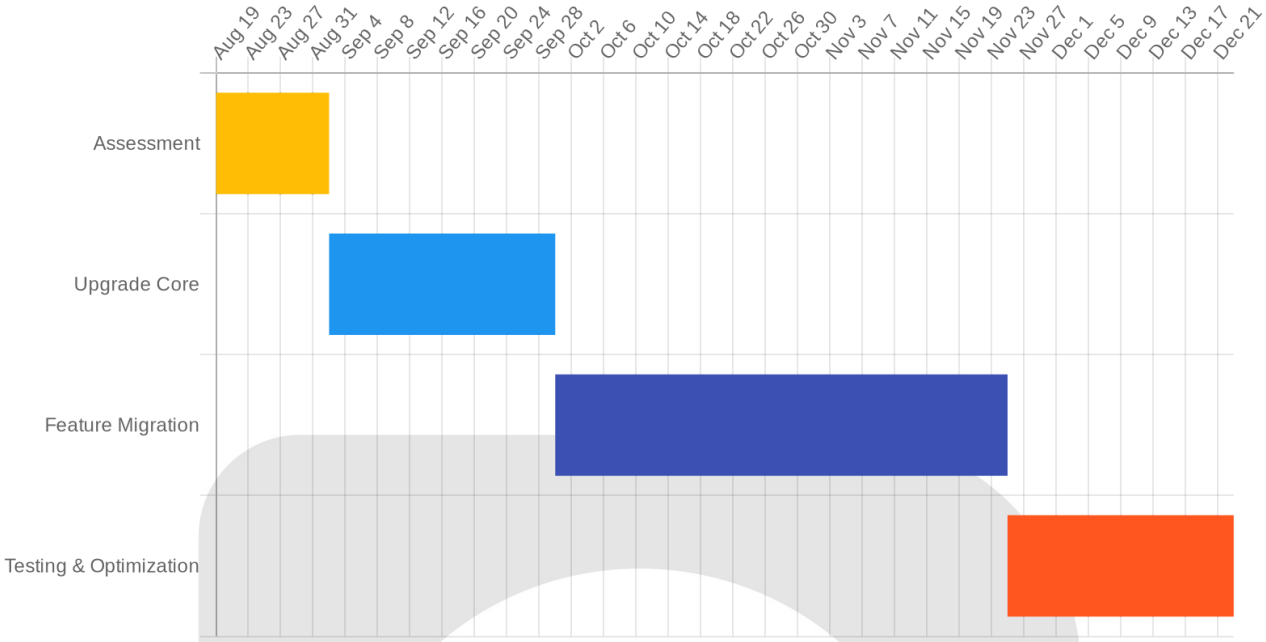
During this phase, we will migrate individual features and components to the new Ember.js version. We will prioritize features based on their impact and complexity. Thorough testing will be conducted after each feature migration to ensure functionality and stability.

Phase 4: Testing & Optimization (4 weeks)

The final phase involves comprehensive testing of the entire application. This includes unit, integration, and user acceptance testing. We will also optimize the application for performance and stability. The deliverable is a fully migrated and optimized Ember.js application.

The following is a grant chart visualizing the timeline for the migration:

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



Stakeholder Communication and Training

Effective communication and comprehensive training are crucial for a successful Ember.js migration. Our plan focuses on keeping all stakeholders informed and equipping our team with the skills needed for the updated framework.

Communication Plan

We will use multiple channels to keep everyone updated. Expect daily stand-ups for the development team. Weekly progress reports will summarize accomplishments, challenges, and upcoming tasks. A dedicated Slack channel will facilitate quick questions and answers. Bi-weekly stakeholder meetings will provide a platform for demonstrations, discussions, and feedback. Key stakeholders include the CTO, development team lead, product owner, and select end-users. This ensures all perspectives are considered throughout the migration.

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

Training Program

To ensure a smooth transition, we will provide targeted training. The development team will receive training on Ember.js 4.x features. This will cover new APIs, best practices, and tooling. Modern JavaScript concepts will also be covered. This will include ES6+ features and related libraries. We will emphasize testing best practices to maintain application quality. Training materials will include documentation, tutorials, and hands-on exercises. Training will be tailored to different skill levels. This will ensure everyone can contribute effectively to the migration effort. The training will address common migration challenges. It will also cover strategies for troubleshooting and debugging.

Post-Migration Support and Maintenance

Following the Ember.js migration, Docupal Demo, LLC will provide comprehensive support and maintenance to ensure the application's ongoing stability and performance. Our approach focuses on proactive monitoring, timely issue resolution, and continuous improvement.

Issue Tracking and Resolution

We will use a centralized issue tracker, Jira, to manage and resolve any post-migration issues. A dedicated support team will be responsible for monitoring the issue tracker, prioritizing issues, and coordinating with the development team for resolution. Our team will categorize issues based on severity and impact, ensuring critical issues are addressed promptly.

Maintenance Schedule

Our maintenance plan includes weekly maintenance releases to address bug fixes and minor enhancements. We will also schedule quarterly major updates to incorporate new features, performance improvements, and Ember.js version upgrades. Security patches will be applied as needed to address any vulnerabilities.

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

Performance and Stability Monitoring

To maintain optimal application performance and stability, we will use New Relic and the Ember Performance Monitoring addon. These tools will provide real-time insights into application performance, allowing us to identify and resolve bottlenecks proactively. Regular load testing will be conducted to ensure the application can handle expected traffic volumes. We will establish performance baselines and track key metrics to identify any performance regressions.

Conclusion and Next Steps

This migration is essential. It addresses maintainability, boosts performance, and strengthens security. Stakeholder backing is crucial for success.

Key Takeaways

We've detailed a clear path. This path leads to a modern, efficient Ember.js application. The migration enhances user experience and reduces technical debt.

Immediate Actions

Upon approval, we need to act swiftly. First, we will secure the allocated budget. Next, we will assign the necessary resources. Then, we will communicate the detailed migration plan to all stakeholders. This ensures everyone is informed and aligned.

Measuring Success

Post-migration, success will be measured objectively. We'll track performance improvements using defined metrics. A decrease in bug reports will indicate enhanced stability. Increased developer satisfaction, measured via survey, will show improved development workflows. These factors will confirm a successful migration.

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