Technical Implementation for Blurby/Blurb Coexistence

Overview

This document provides a deep technical analysis of the **Rails upgrade path from 2.3 to 7.0** for Blurby and its coexistence with Blurb 2.0, identifying potential risks, key implementation challenges, and mitigation strategies. The primary focus is ensuring smooth incremental upgrades while maintaining application stability, performance, and compatibility.

Introduction: Modernization and Coexistence Strategy

The modernization of **Blurby** is a crucial step toward improving maintainability, security, and development efficiency. However, as **Blurb 2.0 team have outlined**, completely replacing Blurby with Blurb 2.0 is not immediately feasible. Instead, a **coexistence strategy** has been proposed to allow both systems to function in parallel while minimizing the need for extensive changes to Blurby.

This coexistence strategy focuses on:

- **Enable introducing changes to Blurby**, as modifications are time-consuming and complex due to slow development cycles in the current version of the Blurby system.
- **Keeping bookstore purchases on Blurby** to ensure order integrity and accurate sales data.
- **Prioritizing user migration** through account linking rather than full data migration.
- **Using a split account and split project model**, ensuring that Blurb 2.0 and Blurby operate independently but can be linked when needed.
- **Ensuring that the modernization results in a stable, high-performing, extensible, and maintainable environment**, ready to support new features and facilitate rapid prototyping.

To align with these goals, the modernization process must not only upgrade Blurby's infrastructure but also introduce **new APIs and features** to enable compatibility between the two platforms.

Timeline Overview: Modernization and Coexistence Phases

The attached timeline provides a structured approach to executing the **Blurby modernization** while ensuring a smooth coexistence with Blurb 2.0. This initial approach responds to the first version of the proposed Blurb 2.0 / Blurby distribution roadmap; suggested people, roles, efforts, and priorities can be modified or redefined depending on the final decision. Thus, the major phases are as follows:

1. Environment and Infrastructure Preparation (Month 0 - Month 1)

- **Local environment setup and DB stabilization**: Ensure Blurby can run reliably in modernized local environments.
- **Pipeline stabilization**: handle CI/CD processes to support deployments.

2. Incremental Upgrade Process (Month 1 - Month 4)

- **New pipelines and IT setup**: Establish required infrastructure for modernized workflows.
- **Workflow mapping**: Describe business flows scoped on Blurby (distribution, distribution account, NAP Checkout and payment gateway, Legacy SKU, Admin tools), and how Blurby and Blurb 2.0 will interact.
- **QA/Test strategy upgrading and improving**: Enhance test coverage to support future Rails and Ruby versions.
- **Ruby Upgrades (2.7 → 3.1)**: Ensure compatibility with modern dependencies.
- **Rails Upgrades (3 → 7)**: Incrementally upgrade the application to modern Rails versions, addressing breaking changes and security improvements at each stage.
- **Background jobs migration**: Ensure asynchronous job processing aligns with the upgraded stack.

3. Coexistence Enablement and API Integration (Month 2 - Month 4)

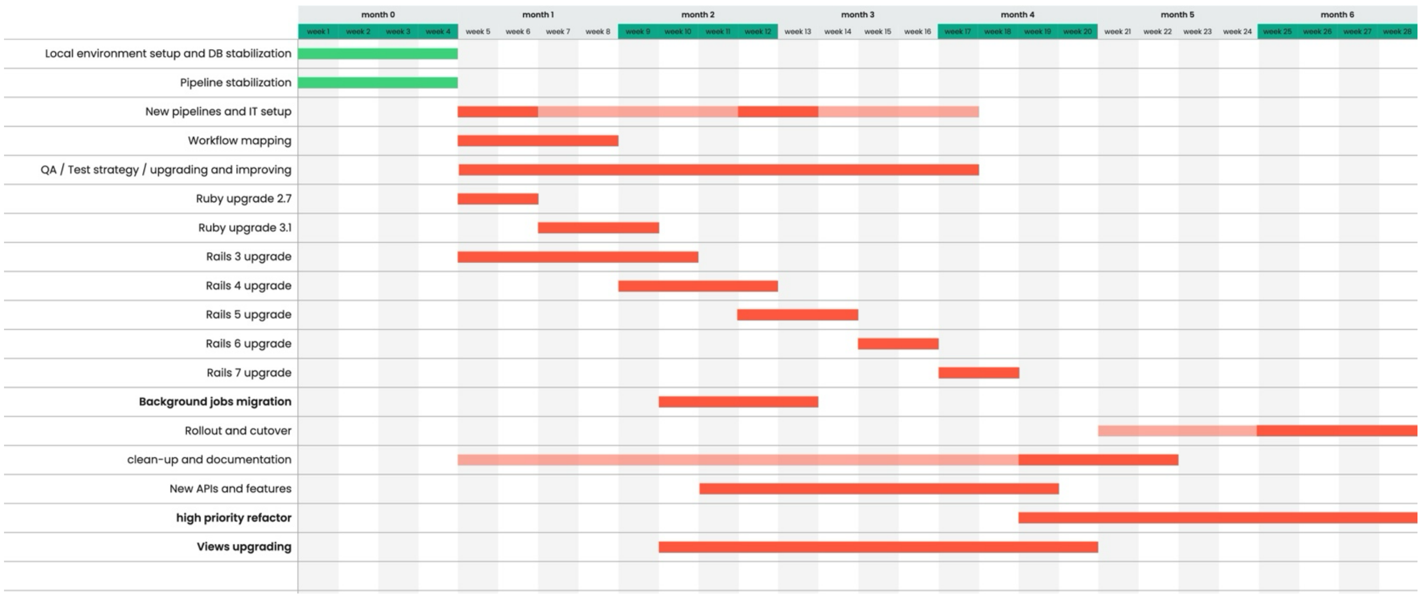
- **New APIs and features**: Develop APIs that allow Blurb 2.0 to integrate with Blurby without requiring a complete rewrite.
- **Views upgrading**: Enhance the front-end to accommodate the modernization efforts.

4. Handle tech debt and support cutover (Month 4 - Month 6)

- **High-priority refactors**: Modify key areas of Blurby to support coexistence.
- **Rollout and cutover**: Deploy changes gradually while maintaining system stability.
- **Clean-up and documentation**: Ensure all upgrades are well-documented and technical debt is addressed.

Key Considerations for Coexistence

- **Account linking**: Users will have separate accounts in Blurb 2.0 and Blurby but can link them when needed.
- **Projects linking**: Projects will exist separately in both systems but can be synced when required for distribution.
- **Orders linking**: Orders will exist on Blurby but needs to be linked.



Proposed Team Structure and Responsibilities

The modernization effort and coexistence strategy require a well-structured team to efficiently execute the roadmap. Below is the proposed team shape and their corresponding responsibilities:

Role	Responsibilities	Duration
Lead (SC - Johan Tique)	Oversees project execution, design and implement complex components or refactors, ensures technical alignment, manages risk mitigation, and handles communication between Blurby and Blurb 2.0 teams.	**6 months**
Automation Tester (SE/EN)	Helps doing the workflow mapping, Improves and maintains test coverage for the upgraded application for both Rails and Ruby versions, ensuring stable coexistence.	**6 months**
Developer Ruby (SE)	Focuses on Rails upgrades, database migrations, and API enhancements to support Blurb 2.0 integration.	**6 months**
Developer Ruby (EN)	Implements Rails upgrades, background jobs migration, performance optimization, and security upgrades.	**6 months**
Developer EmberJS + React (EN)	Upgrades Blurby's front-end components, aligning UI/UX with modern standards for improved user experience.	**3 months**

Team Alignment with the Timeline

- **First 3 months:** Core team focuses on **infrastructure preparation**, **Ruby/Rails upgrades**, and **pipeline stabilization**.
- **Months 2-4:** Developers shift towards **coexistence APIs**, **high-priority refactors**, and **background job migration**.
- **Months 3-6:** The front-end developer joins to complete **views upgrading**, while the team works on **final refinements, cutover, and documentation**.

Appendix

Upgrade Path: Challenges and Critical Breaking changes

1.1 Incremental Upgrade Path

Given the significant gap between **Rails 2.3 and Rails 7**, the upgrade must follow an incremental approach:

1. **Rails 2.3 → 3.2** *(Major breaking changes, fundamental rewrites required)*
2. **Rails 3.2 → 4.2** (Strong Parameters introduced, removal of protected attributes)
3. **Rails 4.2 → 5.2** (Zeitwerk, API mode introduced)
4. **Rails 5.2 → 6.1** (multiple DBs, parallel testing)
5. **Rails 6.1 → 7.0** (Strict autoloading, Hotwire, security enhancements)

1.2 Rails 2.3 to Rails 3.2

1. **Gemfile Structure & Bundler Introduction**
   - Rails 2 relied on `config/environment.rb` for dependency management.
   - Rails 3 introduced `Gemfile` and `Bundler` for managing dependencies.
2. **Routing DSL Changes**
   - `map.connect` in `routes.rb` replaced with `match`, `get`, `post` DSL.
3. **Removal of `vendor/plugins` & Migration to Gems**
   - Plugins in `vendor/plugins` must be converted into gems.
4. **ActiveRecord & Migration Changes**
   - `named_scope` replaced with `scope`.
5. **Controller & View Layer Updates**
   - `before_filter` renamed to `before_action`.
6. **Session Store & Cookie Security**
   - Rails 3 changed session handling, requiring explicit configuration.
7. **Mailer Changes & ActionMailer API Updates**
   - `deliver_*` methods replaced with `mail`.
8. **Testing & RSpec Upgrade**
   - RSpec 1.x is incompatible with Rails 3; requires migration to RSpec 2.x.

2. Rails 3.2 to Rails 4.2

1. **Strong Parameters Introduced**
   - Rails 4 deprecated `attr_accessible`, requiring Strong Parameters for mass assignment protection.
2. **Deprecation of ActiveRecord `find(:all)` and `find(:first)`**
   - Replaced with `User.all` and `User.first`.
3. **Changes to Callbacks in ActiveRecord**
   - Callbacks now run only if changes exist.
4. **Asset Pipeline Enhancements**
   - Improved asset compilation, deprecated older Sprockets versions.
5. **Removal of Rails Observers**
   - Now requires the `rails-observers` gem.
6. **Mailer Changes**
   - `default_url_options` now required.
7. **Testing & RSpec Upgrade**

- RSpec 2.x required, `should` syntax deprecated.

3. Rails 4.2 to Rails 5.2

1. **Zeitwerk Autoloader**
    - Classic autoloader deprecated; all applications must migrate to Zeitwerk.
    - Fix naming inconsistencies and ensure `autoload_paths` are correct.
2. **API Mode Introduced**
    - Rails 5 introduced `rails new --api` for API-only applications.
3. **ApplicationRecord Introduced**
    - All models now inherit from `ApplicationRecord` instead of `ActiveRecord::Base`.
4. **Controller Changes**
    - `before_filter` fully removed in favor of `before_action`.
5. **Test Framework Enhancements**
    - RSpec upgraded to 3.x, requiring syntax migration to `expect` style.
6. **Deprecation of Synchronous `deliver_now` in Mailers**
    - All mailers now default to async delivery unless explicitly set.
7. **Rails CLI & Binstubs**
    - `rails server` and `rails db:migrate` moved to `bin/` directory.
8. **ActiveJob Integration**
    - Standardized background job processing framework introduced.

4. Rails 5.2 to Rails 6.1

1. **Multiple Database Support**
    - Rails 6 introduced built-in support for multiple databases with `config/database.yml`.
2. **Parallel Testing Introduced**
    - RSpec and Minitest now support parallel execution.
3. **ActionMailbox Introduced**
    - Rich text content handling and inbound email processing added.
4. **Webpacker Becomes Default for JavaScript**
    - Sprockets is still supported but Webpacker is encouraged.
5. **Mail Delivery API Changes**
    - `deliver_now` and `deliver_later` improved to better handle failures.
6. **Default Host Configuration for Development**
    - New security constraints require setting `config.hosts` explicitly.
    - Add `config.hosts << "localhost"` in `config/environments/development.rb`.

5. Rails 6.1 to Rails 7.0

1. **Hotwire (Turbo & Stimulus) Replaces Rails UJS**
   - Rails 7 deprecates `rails-ujs` in favor of **Turbo & Stimulus**.
   - Migrate UJS event handlers to Turbo Streams and Turbo Frames.
2. **ActiveRecord Encryption Support**
   - ActiveRecord now natively supports encrypted attributes.
   - Example Migration:

```
class User < ApplicationRecord
  encrypts :email
end
```

   - Identify and encrypt sensitive fields where applicable.
3. **Asynchronous Query Loading**
   - ActiveRecord now supports async query loading using `.load_async`.
   - Optimize long-running queries by implementing `.load_async` where applicable.
4. **Mailer Configuration Changes**
   - `config.action_mailer.perform_deliveries` now defaults to `true` in production.
   - Ensure explicit production settings to avoid unintended email deliveries.
5. **Security Enhancements**
   - Content Security Policy (CSP) introduced to mitigate XSS attacks.
   - Define a strict CSP in `config/initializers/content_security_policy.rb`.
6. **Asset Pipeline Enhancements**
   - `config.assets.compile = true` is deprecated; assets must be precompiled.
   - Use CI/CD pipelines to handle asset compilation and serve via CDN.
7. **ActiveJob Defaults to Async Execution**
   - Background jobs now default to async execution instead of inline.
   - Configure a proper job queueing system (Sidekiq, DelayedJob, etc.).
8. **Removal of Deprecated APIs**
   - Methods like `update_attributes` and `reset_column_information` are removed.
   - Replace with `update` and `reset_column_information!` where necessary.

Analysis of Critical Upgradable Gems to Ruby 2.7/3.1

Gem	Responsibility	Risks in Upgrade
nokogiri (1.14.5)	XML/HTML parsing	Changes in API handling, potential XPath query changes
json (1.8.5)	JSON parsing and generation	Older syntax deprecated, potential encoding issues
yajl-ruby (~> 1.4.1)	JSON parsing with Yajl	Changes in strict mode handling, potential breakage in malformed JSON responses

Gem	Responsibility	Risks in Upgrade
ffi (1.13.1)	Foreign function interface (FFI) for calling C functions	API changes in handling pointers, potential OS-specific compatibility issues
pg (1.3.5)	PostgreSQL adapter for ActiveRecord	Changes in prepared statements, risk of connection pool management changes
rmagick (4.3.0)	Image manipulation library using ImageMagick	Memory leak concerns, deprecated method warnings
thin (1.8.2)	Lightweight web server for Rack applications	Compatibility with Rack versions, potential TLS/SSL changes
rexml (~> 3.1.9.1)	XML parsing in Ruby standard library	Strict XML parsing modes might break previously accepted malformed XML
aws-sdk	Amazon Web Services API client	Older versions incompatible with Ruby 2.7+, namespace restructuring required
braintree	Payment gateway integration	Older versions use deprecated Net::HTTP syntax, potential TLS/SSL compatibility issues
rack	Web server interface for Ruby applications	Breaking changes in middleware handling, deprecated API removals
rspec	Testing framework for Ruby	Requires migration from older `should` syntax to `expect`, possible spec failures due to API changes
Compass	Stylesheet authoring framework	Deprecated and replaced by `bourbon` or `autoprefixer`, requires migration
Sass	CSS preprocessor	Legacy versions deprecated in favor of `dart-sass`, syntax adjustments needed
bcrypt-ruby	Password hashing library	Replaced by `bcrypt`, migration needed to avoid dependency resolution issues