

Loan Management System

Project Implementation Report

Date: 21.01.2026

Table of Contents

1	Executive Summary	3
2	Task Breakdown & Status	3
2.1	Task 1: Loan List Component (Completed)	3
2.2	Task 2: Loan Application Form (Completed)	5
2.3	Task 3: Loan Details & Performance (Completed)	7
2.4	Bonus Task 1: Error Handling (Completed)	7
2.5	Bonus Task 2: Unit Tests (Completed)	8
2.6	Bonus Task 3: Loading Interceptor (Completed)	8
3	Additional Features & Improvements	8
3.1	Infrastructure & Standards	8
3.2	Accessibility (WCAG 2.1 AA)	8
4	Tools & Technologies Used	9
5	Testing Methodology	9
5.1	How to Run Automated Tests	9
5.2	Troubleshooting	9

Table of Figures

Figure 2.1: Loans List UI	4
Figure 2.2: Loans List Filtering	4
Figure 2.3: Loan Application For Creating a Loan	5
Figure 2.4: Loan Application Validations	6
Figure 2.5: Update Loan Form	6
Figure 2.6: Loan Details Page with Monthly Payment Formula	7
Figure 5.1: Automated Test Run on Karma	10

1 EXECUTIVE SUMMARY

This document details the implementation of the Loan Management Application. The project successfully migrated a basic setup into a production-ready, scalable Angular application. Key achievements include full CRUD functionality for Loans, a robust modular architecture, global error handling, and strict adherence to WCAG 2.1 Level AA Accessibility standards.

2 TASK BREAKDOWN & STATUS

2.1 Task 1: Loan List Component (Completed)

Objective: Create a responsive loan list with filtering and sorting.

Implementation:

- Created `LoanListComponent` at `/loans`.
- Displayed all required fields (ID, Name, Amount, Rate, Status, Date).
- Implemented Client-side Filtering:
 - Status: Dropdown for Active/Pending/Closed.
 - Borrower Name: Search input.
- Implemented Client-side Sorting:
 - Sortable columns for Amount and Date with visual indicators (↑/↓).
- Enhancement: Added ARIA labels for accessible sorting and filtering.

Key Files

- `src/app/modules/loans/components/loan-list/loan-list.component.ts`
(Logic)
- `src/app/modules/loans/components/loan-list/loan-list.component.html`
(Template)
- `src/app/modules/loans/components/loan-list/loan-list.component.css`
(Styles)

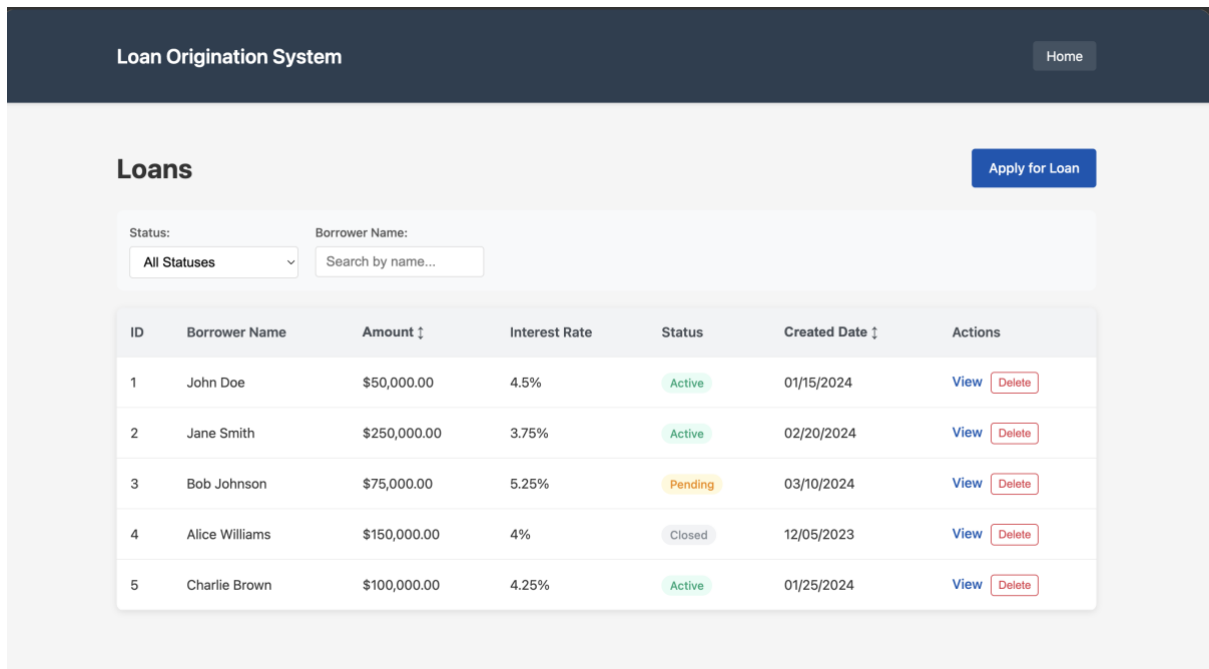


Figure 2.1: Loans List UI

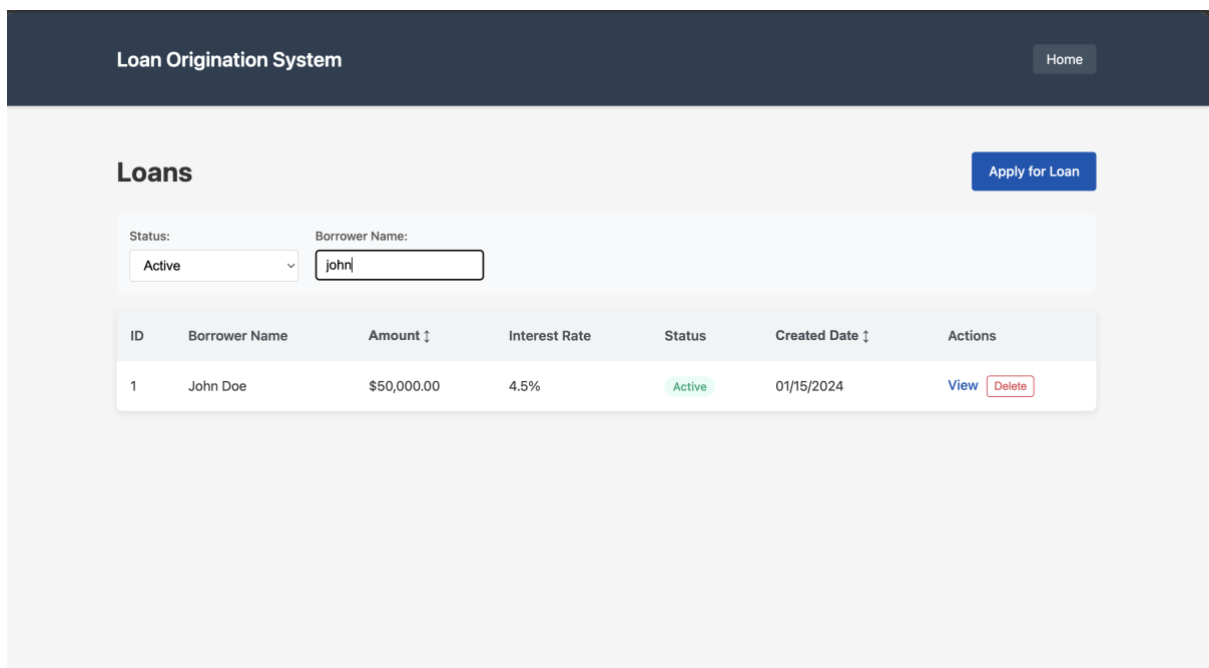


Figure 2.2: Loans List Filtering

2.2 Task 2: Loan Application Form (Completed)

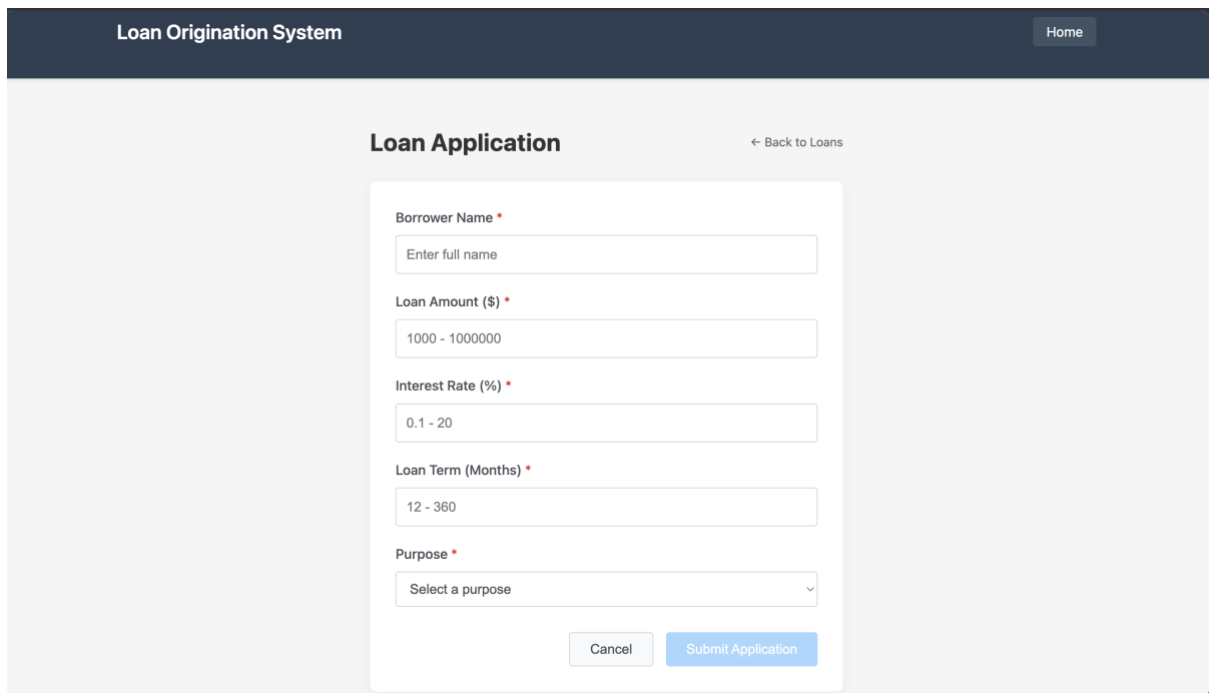
Objective: Create a loan application form with validation.

Key Files:

- `src/app/modules/loans/components/loan-application/loan-application.component.ts` (Logic)
- `src/app/modules/loans/components/loan-application/loan-application.component.html` (Template)
- `src/app/modules/loans/components/loan-application/loan-application.component.css` (Styles)
- `src/app/modules/loans/components/loan-application/loan-application.component.spec.ts` (Unit Tests)

Implementation:

- Utilized Reactive Forms for robust state management.
- Implemented real-time validation for all fields.
- Refactored to support "Edit Mode" for existing loans.



The screenshot displays the 'Loan Origination System' interface. At the top, a dark blue header contains the text 'Loan Origination System' on the left and a 'Home' button on the right. Below the header, the main content area has a light gray background. Centered in this area is a white form titled 'Loan Application' with a '← Back to Loans' link to its right. The form contains five input fields, each with a red asterisk indicating a required field: 'Borrower Name' (placeholder: 'Enter full name'), 'Loan Amount (\$)' (placeholder: '1000 - 1000000'), 'Interest Rate (%)' (placeholder: '0.1 - 20'), 'Loan Term (Months)' (placeholder: '12 - 360'), and 'Purpose' (a dropdown menu with 'Select a purpose' and a downward arrow). At the bottom of the form are two buttons: a gray 'Cancel' button and a blue 'Submit Application' button.

Figure 2.3: Loan Application For Creating a Loan

Loan Application

← Back to Loans

Borrower Name *

Kyler Ren

Loan Amount (\$) *

|1000 - 1000000

Amount is required

Interest Rate (%) *

0.1

Loan Term (Months) *

12

Purpose *

Select a purpose

Purpose is required

Cancel

Submit Application

Figure 2.4: Loan Application Validations

Loan Origination System

Home

Update Loan

← Back to Loans

Borrower Name *

Kyler Ren

Loan Amount (\$) *

2009

Interest Rate (%) *

0.1

Loan Term (Months) *

12

Purpose *

Refinance

Cancel

Update Loan

Figure 2.5: Update Loan Form

2.3 Task 3: Loan Details & Performance (Completed)

Objective: Detailed view with optimizations and lazy loading.

Key Files:

- `src/app/modules/loans/components/loan-detail/loan-detail.component.ts`
- `src/app/modules/loans/components/loan-detail/loan-detail.component.html`
- `src/app/modules/loans/components/loan-detail/loan-detail.component.css`

Implementation:

- **Lazy Loading:** Configured in `src/app/modules/loans/loans-routing.module.ts`
- **Performance:** Used `onPush` change detection strategy.
- **Calculations:** Implemented monthly payment formula.

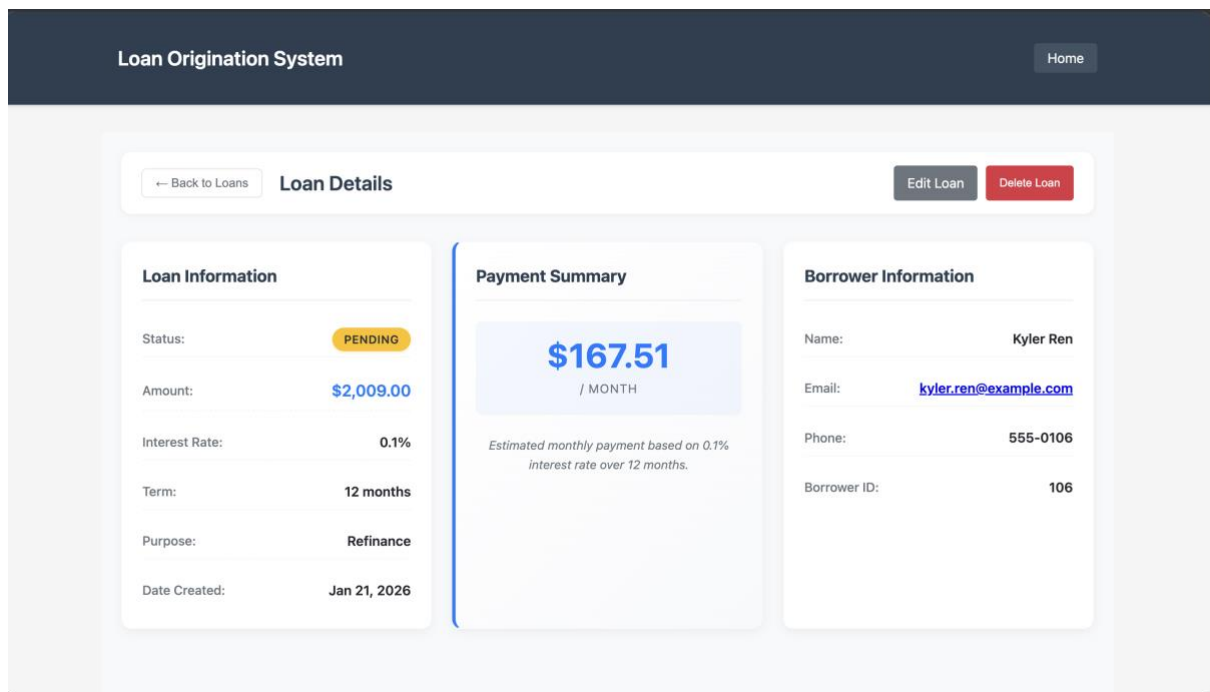


Figure 2.6: Loan Details Page with Monthly Payment Formula

2.4 Bonus Task 1: Error Handling (Completed)

Key Files:

- `src/app/core/handlers/global-error-handler.ts`
- `src/app/core/services/notification.service.ts`
- `src/app/shared/components/toast/toast.component.ts`

Implementation:

- Global error handler catches exceptions and displays user-friendly Toasts.

2.5 Bonus Task 2: Unit Tests (Completed)

Key Files:

- `src/app/core/services/loan.service.spec.ts`
- `src/app/modules/loans/components/loan-application/loan-application.component.spec.ts`

2.6 Bonus Task 3: Loading Interceptor (Completed)

Implementation:

- Tests verify API logic and Form validation constraints.
- Bonus Task 3: Loading Interceptor (Completed)

Key Files:

- `src/app/core/interceptors/loading.interceptor.ts`
- `src/app/core/services/loading.service.ts`
- `src/app/shared/components/spinner/spinner.component.ts`

Implementation:

- Interceptor tracks active requests and toggles the global spinner.

3 ADDITIONAL FEATURES & IMPROVEMENTS

3.1 Infrastructure & Standards

Core Services: `src/app/core/services/loan.service.ts`

Environment Config: `src/environments/environment.ts` , `set-env.js`

Architecture: Refactored into Core (Singletons), Shared (UI), and Feature modules.

Full CRUD Operations

- Delete Feature:
 - `src/app/shared/components/confirmation-dialog/confirmation-dialog.component.ts`
 - `src/app/core/services/dialog.service.ts`
- **Implementation:** Implemented safe deletion with a reusable Confirmation Dialog.

3.2 Accessibility (WCAG 2.1 AA)

- **Audit Scope:** All Component HTML files (`*.component.html`) and TypeScript logic for focus management.
- **Implementation:**

- **Focus Trapping:** `ConfirmationDialogComponent`.
- **Semantic Landmarks:** `AppComponent` (`src/app/app.component.html`).
- **ARIA Attributes:** Applied to tables, forms, and buttons across all modules.

4 TOOLS & TECHNOLOGIES USED

- Framework: Angular 15+
- Language: TypeScript
- State Management: RxJS (Observables, Subjects)
- Testing: Jasmine & Karma
- Build: Angular CLI

5 TESTING METHODOLOGY

- **Automated:** Unit tests run via `ng test` to verify logic in isolation.
- **Manual:** Verified all user flows (Create -> List -> Detail -> Edit -> Delete) in Chrome. Verified accessibility behavior using keyboard-only navigation.

5.1 How to Run Automated Tests

The application uses Karma as the test runner and Jasmine as the assertion framework. We have configured support for both Chrome and Microsoft Edge.

Command

Run the following command in the frontend directory:

```
npm run test or ng test
```

Note: This will attempt to launch both Chrome and Edge. You can modify `karma.conf.js` to select only one.

5.2 Troubleshooting

- Chrome: If you encounter Cannot start Chrome, set or check the `CHROME_BIN` environment variable.
- Edge: If you encounter Cannot start Edge, ensure Microsoft Edge is installed and `EDGE_BIN` is set if needed.
- For CI/CD environments, you may need to configure Karma to use `ChromeHeadless` or `EdgeHeadless`.

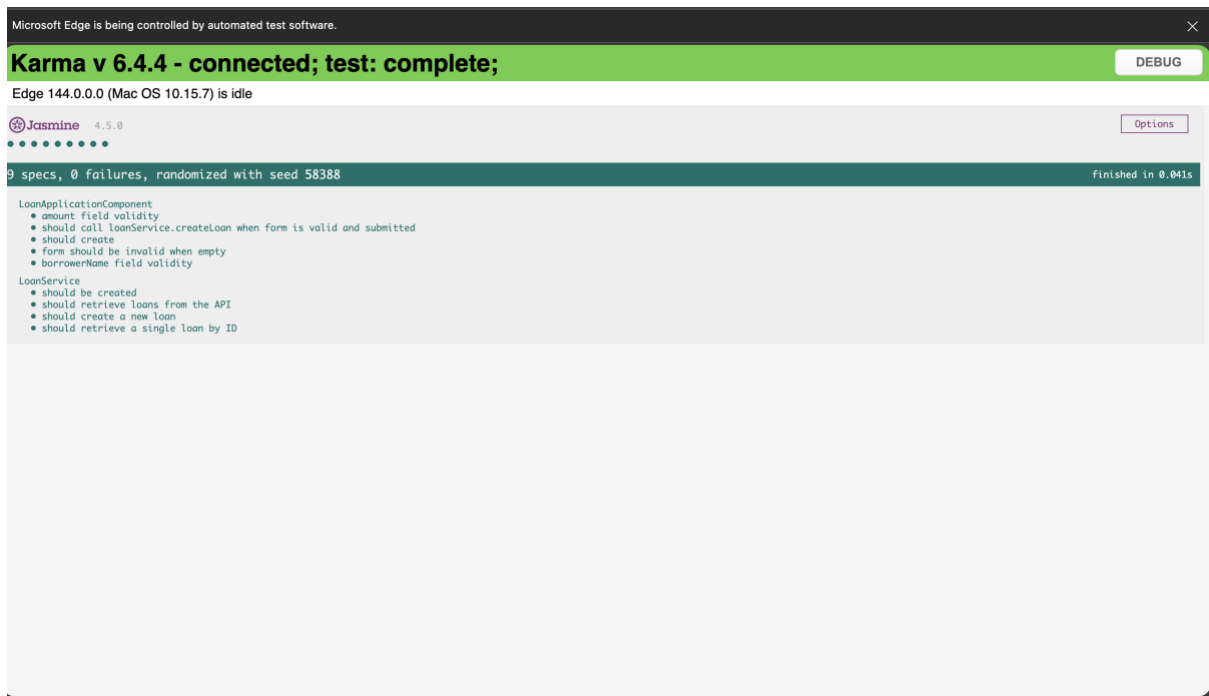


Figure 5.1: Automated Test Run on Karma