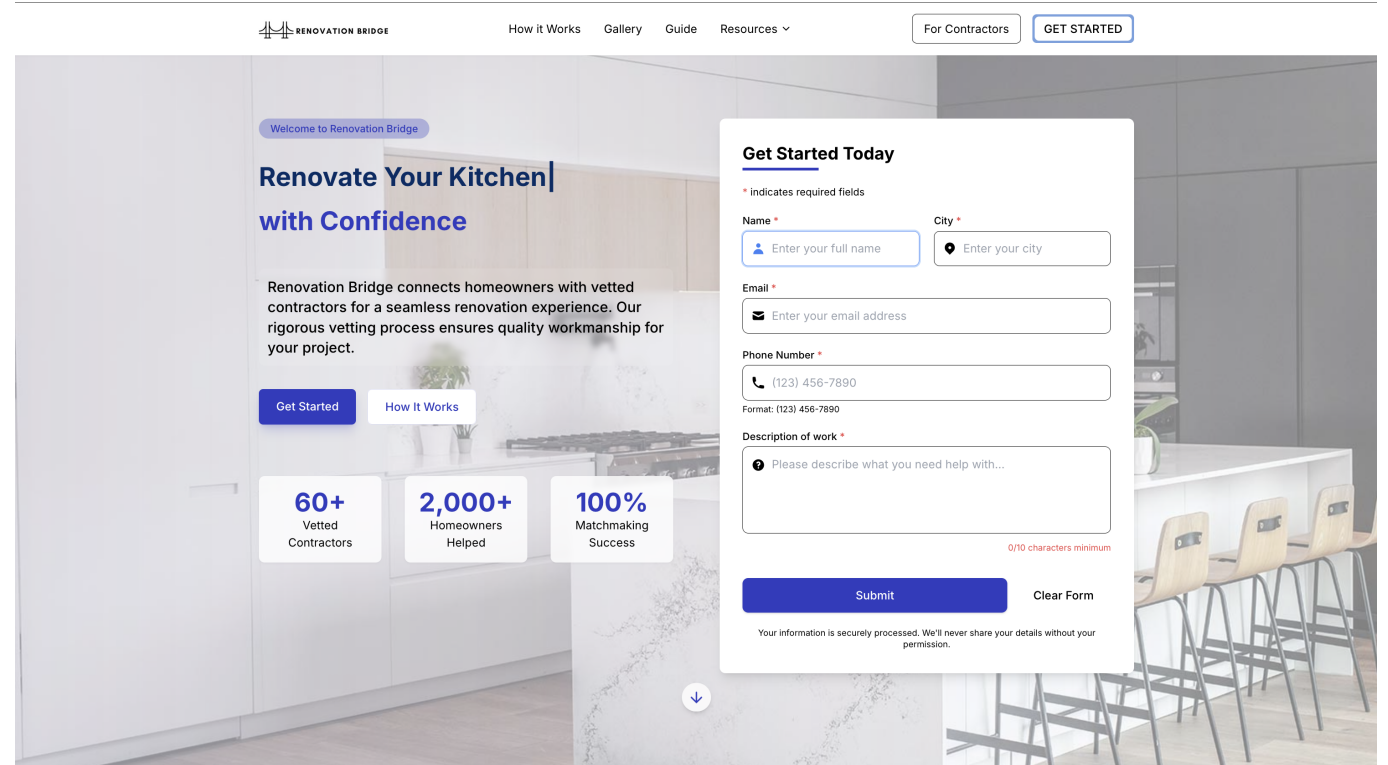
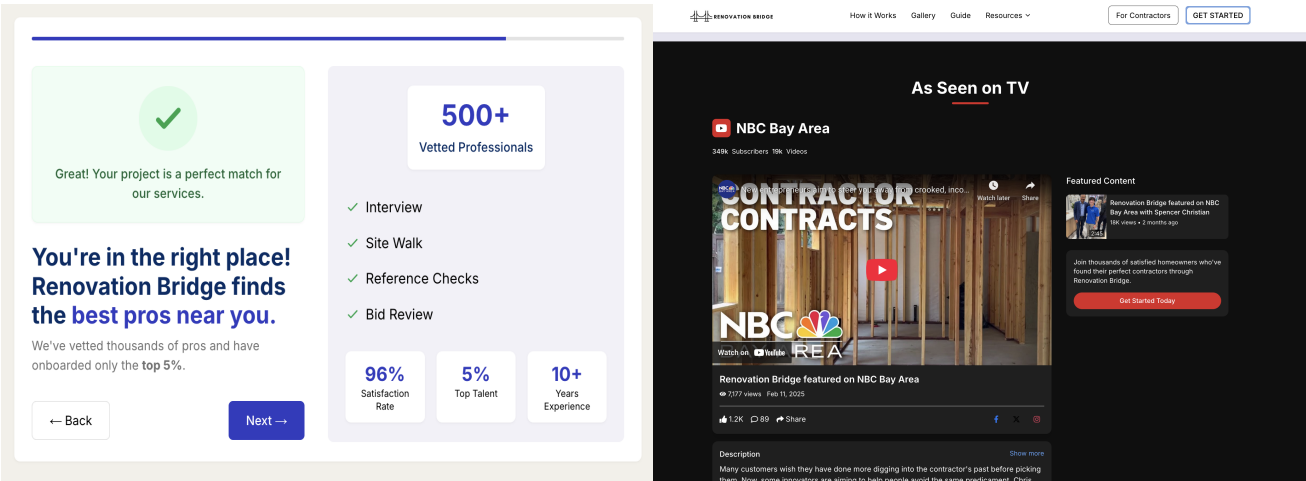


Renovation Bridge

Connecting homeowners with quality contractors for seamless renovation experiences



Modern, responsive homepage designed to convert visitors into leads



Project showcase and contractor management interface

Design System

Our carefully crafted color palette ensures a professional and trustworthy appearance:

- **Primary Blue #313bc0** - Trust and professionalism
- **Secondary Dark Blue #002c66** - Depth and stability
- **Gray #717171** - Balance and readability
- **Black #000000** - Contrast and emphasis

- **Cream** #f2f0e9 - Warmth and approachability
- **Lavender** #e5e4f0 - Softness and sophistication

Tech Stack

- **Framework:** Next.js 14.2+
- **UI Library:** React 18
- **Styling:** Tailwind CSS 3.4+
- **Animations:** Framer Motion 12.5+
- **Forms:** React Hook Form 7.50+
- **Icons:** React Icons 5.0+
- **Image Processing:** Sharp 0.33+
- **SEO:** next-sitemap, JSON-LD structured data
- **HTML Parsing:** jsdom
- **Testing:** Jest, React Testing Library, Playwright for E2E
- **Deployment:** Vercel
- **GraphQL Client:** GraphQL Request

Project Structure

```
renovationbridge/
├── public/                # Static assets
│   ├── robots.txt        # Search engine crawling rules
│   └── sitemap.xml        # Auto-generated sitemap
├── src/
│   ├── app/              # Next.js App Router pages
│   │   ├── api/          # API routes including revalidation
│   │   ├── blog/         # Blog posts and articles
│   │   ├── contractors/  # Contractor-specific pages
│   │   └── resources/    # Resource and guide pages
│   ├── components/       # Reusable UI components
│   │   ├── forms/        # Form components
│   │   ├── layout/       # Layout components
│   │   ├── sections/     # Page sections
│   │   ├── seo/          # SEO components (JsonLd, PageSeo)
│   │   └── ui/           # UI elements
│   ├── lib/              # Core functionality
│   │   ├── db/           # Database connections
│   │   ├── models/       # Data models
│   │   ├── structured-data.ts # JSON-LD schema generators
│   │   ├── seo-utils.ts  # SEO utility functions
│   │   └── utils/        # Utility functions
│   ├── styles/           # Global styles
│   └── utils/            # Helper functions
├── scripts/              # Deployment and utility scripts
├── secrets/              # Environment secrets
├── .github/              # GitHub Actions workflows
├── next-sitemap.config.js # Sitemap configuration
└── config files          # Various configuration files
```

Getting Started

1. Clone the repository

```
git clone https://github.com/yourusername/renovationbridge.git
cd renovationbridge
```

2. Install dependencies

```
npm install
```

3. Configure environment variables

```
cp .env.example .env.local
# Edit .env.local with your configuration settings
```

4. Start development server

```
npm run dev
# or
./start.sh
```

5. Open <http://localhost:3000>

Local Testing

Renovation Bridge includes a comprehensive suite of testing tools to ensure code quality and functionality.

Using the Testing Script

A convenient testing script is provided to run various testing operations:

```
# Make the script executable (first time only)
chmod +x scripts/testing.sh

# Run the testing script
./scripts/testing.sh
```

The script provides the following options:

1. Start development server
2. Run linting

3. Run unit tests
4. Run end-to-end tests
5. Run all tests

Manual Testing Commands

You can also run these commands directly:

```
# Lint the codebase
npm run lint

# Run unit tests
npm test

# Run unit tests with watch mode
npm test -- --watch

# Run end-to-end tests
npm run test:e2e

# Check TypeScript types
npm run type-check
```

Testing Strategies

- **Unit Tests:** Located in `__tests__` directories alongside the components they test
- **Integration Tests:** Found in the `tests/integration` directory
- **End-to-End Tests:** Uses Cypress located in the `cypress` directory

Mock Data for Testing

For testing with mock data:

```
# Start the development server with mock data
npm run dev:mock
```

This uses mock API responses from the `mocks` directory, allowing you to test the application without connecting to actual APIs.

🌟 Key Features

- **Smart Lead Generation** - Optimized contact forms and CTAs
- **Contractor Management** - Dedicated dashboard for contractors
- **Project Showcase** - Beautiful gallery of completed renovations including:
 - Interactive image galleries with lightbox functionality
 - Detailed project information and specifications
 - Organized by project type (kitchen, bathroom, full home)

- Responsive design for optimal viewing on all devices
- **Resource Library** - Valuable guides and articles
- **Mobile-First Design** - Perfect experience on all devices
- **Advanced SEO** - Complete with:
 - Automatic sitemap generation
 - Robots.txt configuration
 - JSON-LD structured data (Organization, LocalBusiness, Services)
 - Comprehensive OpenGraph and Twitter card metadata
 - Canonical URLs
 - Dynamic meta descriptions
- **Incremental Static Regeneration** - Fast page loads with dynamic content

Deployment

Vercel Deployment

Renovation Bridge is configured for seamless deployment on Vercel's platform.

1. Prerequisites

- A [Vercel account](#)
- Vercel CLI (optional, for command-line deployment)

```
npm install -g vercel
```

2. Connect Repository to Vercel

- Go to [Vercel Dashboard](#)
- Click "Add New" > "Project"
- Select your GitHub repository
- Configure the project:
 - Framework Preset: Next.js
 - Root Directory: ./
 - Build Command: `npm run build`
 - Output Directory: .next

3. Set Environment Variables

- In Vercel dashboard, navigate to your project
- Go to Settings > Environment Variables
- Add all required environment variables from `.env.production.local`
- Key variables to include:

```
MONGODB_URI
JWT_SECRET
REVALIDATION_SECRET
```

```
API_URL
API_TOKEN
```

4. Deploy Using the Script

```
# Run the deployment script
./scripts/test-deploy.sh
# Select option 3 for production deployment
```

5. Manual Deployment Using Vercel CLI

```
# Login to Vercel
vercel login

# Deploy to development environment
vercel

# Deploy to production
vercel --prod
```

6. Automatic Deployments

- Every push to the **main** branch will trigger an automatic deployment
- Preview deployments are created for pull requests
- Configure branch deployments in the Vercel dashboard under Settings > Git

7. Revalidation API

- Content revalidation is configured at **/api/revalidate**
- Use this endpoint with the appropriate secret token to refresh content
- Example:

```
curl -X POST https://your-domain.vercel.app/api/revalidate \
-H "Authorization: Bearer YOUR_REVALIDATION_SECRET" \
-H "Content-Type: application/json" \
-d '{"contentType":"post","slug":"example-post"}
```

8. Monitoring

- Monitor your deployment through the Vercel dashboard
- View logs, analytics, and performance metrics
- Set up status alerts for your production deployment

Recent Updates

Vercel SpeedInsights Integration (Added)

Added Vercel SpeedInsights to monitor and improve web performance:

- **What Changed:** Integrated the `@vercel/speed-insights` package with a custom DSN for self-hosting
- **Files Modified:**
 - `src/app/layout.tsx` - Added the SpeedInsights component with custom DSN
- **Implementation Details:**
 - Self-hosted with custom DSN: `ZNga99anB7eSaJjchi9phAZv6n7`
 - Automatically collects Web Vitals metrics
 - Provides insights on real-user performance
 - Zero impact on page loading performance
 - Privacy-focused analytics with no cookie requirements

Benefits: This integration enables real-time monitoring of site performance metrics, helping identify and address performance bottlenecks to improve user experience.

Case-Sensitivity Fix for Gallery URLs (June 2023)

Fixed an issue where gallery project pages were generating 404 errors when accessed with different letter casing in URLs:

- **What Changed:** Updated the URL slug matching logic to be case-insensitive, normalizing all gallery URLs
- **Files Modified:**
 - `src/app/gallery/[slug]/page.tsx` - Improved slug matching with case normalization
 - `src/features/gallery/services/galleryImageService.ts` - Enhanced project name lookup to handle case variations

Usage Example: The following URLs now all correctly display the same project page:

- `/gallery/alamo`
- `/gallery/Alamo`
- `/gallery/ALAMO`

This ensures consistent user experience regardless of how the URL is typed or linked.



SEO Implementation

The project includes comprehensive SEO features:

1. Sitemap Generation

- Automatic sitemap.xml generation using next-sitemap
- Custom priority configurations for different page types
- Custom change frequency settings

2. Robots.txt

- Properly configured robots.txt file in public directory

- Controls search engine crawling behavior

3. Structured Data

- JSON-LD implementation for rich search results
- Schemas for Organization, LocalBusiness, Service, FAQ, and BreadcrumbList
- Reusable structured data components

4. Meta Tags

- Enhanced metadata in layout.tsx
- OpenGraph and Twitter card meta tags
- Canonical URL implementation
- Dynamic meta descriptions

Project Gallery

The project showcase gallery features:

- Grid-based showcase of completed projects
- Filtering by project category
- Detailed project pages with image galleries
- Responsive design optimized for all devices
- Lightbox image gallery with navigation
- Project metadata including:
 - Services provided
 - Project duration
 - Completion date
 - Category/type of project
- Automatic image gallery generation based on available images
- Image paths follow a consistent pattern:
 - `public/images/gallery/Project-[ProjectName]/[projectname]-[n].jpg`

Feature Module Organization

Gallery code is now organized into a feature module structure:

```
src/features/gallery/
├── components/      # UI components specific to gallery
│   ├── ImageGallery.tsx      # Reusable gallery grid with lightbox
│   └── ProjectGalleryTemplate.tsx # Project detail layout
├── hooks/           # Custom hooks for gallery functionality
│   ├── useGalleryProjects.ts # Hook for project listing/filtering
│   └── useProjectDetails.ts  # Hook for getting project details
├── services/        # Data services for gallery
│   ├── galleryDataService.ts # Project data and metadata
│   └── galleryImageService.ts # Image path handling
├── types/           # Type definitions
│   └── index.ts      # Shared gallery types
└── index.ts         # Barrel file exporting public API
```


This modular approach provides:

- Clear separation of concerns
- Well-defined interfaces between components
- Improved maintainability
- Easier testing
- Reusable components and hooks

To use gallery components and services, import from the feature module:

```
import {  
  ImageGallery,  
  ProjectGalleryTemplate,  
  useGalleryProjects,  
  useProjectDetails  
} from '@features/gallery';
```

Built by [Ethan Bleier](#)

Testing

The project uses a comprehensive testing strategy:

1. Unit and Component Testing

- Jest for test runner and assertions
- React Testing Library for component testing
- Run tests with:

```
# Run all tests  
npm test  
  
# Watch mode for development  
npm run test:watch  
  
# Generate test coverage report  
npm run test:coverage
```

2. End-to-End Testing

- Playwright for browser-based end-to-end testing
- Tests run against multiple browsers (Chrome, Firefox, Safari)
- Mobile device simulation for responsive testing
- Enhanced timeout configuration for reliable mobile testing
- Run E2E tests with:

```
# Run all E2E tests
npm run test:e2e

# Run tests in a specific browser
npx playwright test --project=chromium

# Run only mobile tests
npx playwright test --project="Mobile Chrome" --project="Mobile Safari"

# Run tests in UI mode for debugging
npx playwright test --ui
```

3. Test Structure

- Unit/component tests in `src/__tests__`
- E2E tests in `e2e/`
- Component tests paired with their respective components
- Integration tests for key user flows

Deployment Notes

GitHub Actions Build Issues

If you encounter errors like `Field 'browser' doesn't contain a valid alias configuration` or `src/index.json doesn't exist` during GitHub Actions builds, this is likely due to webpack resolution issues. We've added specific webpack configuration in `next.config.js` to address this:

```
webpack: (config, { isServer }) => {
  // Ensures webpack resolves modules correctly
  config.resolve.modules = ['node_modules', '.']

  // Explicitly tell webpack how to resolve
  config.resolve.extensions = ['.js', '.jsx', '.ts', '.tsx']

  return config
},
```

This configuration helps webpack properly resolve module paths in the GitHub Actions environment.