Next.js 16 Migration Roadmap

Executive Summary

This document outlines a **complete and full migration** plan to move the QuoteVote frontend from **Vite + React 17** to **Next.js 16**. This is a **full framework replacement**, not a parallel setup. The old Vite-based application will be completely replaced with a Next.js 16 application.

Migration Scope

- **Framework**: Complete replacement of Vite → Next.js 16 (App Router)
- **UI Library**: Material-UI v4 → shadcn/ui
- **Language**: JavaScript (JSX) → TypeScript (TSX)
- **Routing**: React Router v5 → Next.js 16 App Router (file-based routing)
- **Build System**: Complete removal of Vite → Next.js 16 build system
- **React Version**: React 17 → React 19 (Next.js 16 requirement)
- **Architecture**: Client-side only → Next.js Server/Client Components architecture
- **Deployment**: SPA deployment → Next.js production deployment

Migration Philosophy

This is a **complete rewrite** within the existing codebase structure.

We will:
1. **Completely remove** Vite and all Vite-specific configurations
2. **Fully adopt** Next.js 16 App Router patterns
3. **Leverage** Server Components where possible for better performance
4. **Migrate** all client-side logic to Next.js Client Components
5. **Replace** all routing with Next.js file-based routing
6. **Remove** all React Router dependencies
7. **Eliminate** all Vite build artifacts and configurations

Testing Requirements (MANDATORY)

Testing is mandatory after each phase completion. Every phase includes:
- ✅ **Test Script Writing**: Write test scripts for all changes
- ✅ **Functionality Testing**: Verify all features work correctly
- ✅ **Integration Testing**: Test components work together
- ✅ **Regression Testing**: Ensure no existing functionality breaks
- ✅ **Test Suite Execution**: Run full test suite before proceeding

Testing Strategy:
- Write tests **immediately after** each migration step
- Test **before** moving to the next phase
- Maintain test coverage throughout migration
- Use automated testing where possible
- Manual testing for visual/UX validation

Current State Analysis

Technology Stack

Category	Current	Target
Framework	React 17.0.1	React 19 (via Next.js 16)
Build Tool	Vite 5.4.19	Next.js 16
UI Library	Material-UI v4 (@material-ui/core)	shadcn/ui
Language	JavaScript (JSX)	TypeScript (TSX)
Routing	React Router v5	Next.js 16 App Router
State Management	Redux Toolkit + Redux Persist	**Option A**: Redux Toolkit + Redux Persist (maintain) **Option B**: Zustand (recommended for Next.js)
GraphQL Client	Apollo Client v3	Apollo Client v3 (maintain)
Styling	SCSS + Material-UI styles	Tailwind CSS + CSS Modules
Testing	Vitest	Vitest (maintain)

Codebase Statistics

- **Total MUI Imports**: 984 matches across 247 files
- **Component Files**: ~200+ JSX components
- **Custom MUI Components**: `src/mui-pro/` directory (Material Dashboard Pro React template)
- **Routes**: 6 main routes defined in `routes.jsx`
- **Layouts**: 4 layouts (Auth, Admin, Scoreboard, TokenExpired)
- **Store Modules**: 4 Redux slices (user, ui, chat, filter)

Key Dependencies to Migrate

Material-UI Components (984 imports)

- `@material-ui/core` → shadcn/ui components
- `@material-ui/icons` → lucide-react or custom icons
- `@material-ui/lab` → shadcn/ui or custom implementations
- `makeStyles` → Tailwind CSS classes
- `ThemeProvider` → Tailwind CSS + CSS Variables

Routing

- `react-router-dom` v5 → Next.js App Router
- `history` package → Next.js navigation
- `useHistory` → `useRouter` from `next/navigation`
- `Route`, `Switch`, `Redirect` → File-based routing

Build & Dev Tools

- `vite.config.js` → `next.config.js`
- `index.html` → `app/layout.tsx` + `app/page.tsx`
- Vite plugins → Next.js plugins/configuration

Next.js 16 Key Features & Considerations

React 19 Compatibility

- **React 19 Requirements**: Next.js 16 uses React 19, which includes:
  - Automatic JSX Transform (no need for `import React`)
  - New `use()` hook for async operations
  - Enhanced Server Components
  - Improved form handling with `useFormStatus` and `useFormState`
  - Better hydration error messages
  - Support for `ref` as a prop (not just callback)

Next.js 16 Specific Features

- **Enhanced App Router**: Improved routing with better performance
- **Turbopack**: Faster builds (can be enabled with `turbo` flag)
- **Improved Caching**: Better default caching strategies
- **Partial Prerendering**: Automatic partial prerendering for better performance
- **Server Actions**: Enhanced server actions with better type safety
- **Metadata API**: Improved SEO and metadata handling
- **Image Optimization**: Better image optimization with `next/image`
- **Font Optimization**: Automatic font optimization

Breaking Changes from React 17 → React 19

- **Context API**: `useContext` behavior changes
- **Refs**: `ref` can now be a prop, not just a callback
- **Hooks**: Some hooks have updated behavior
- **Error Boundaries**: Improved error boundary handling
- **Suspense**: Enhanced Suspense support

Migration Considerations

- Update all components to use React 19 patterns
- Remove unnecessary `React` imports (automatic JSX transform)
- Update ref usage where applicable
- Test all hooks for compatibility
- Update error boundaries if needed

Migration Strategy

Full Framework Migration Approach

This migration is a **complete framework replacement**. The strategy involves:

1. **Complete Removal**: Remove all Vite, React Router, and old build system files
2. **Full Next.js Setup**: Establish Next.js 16 as the sole framework
3. **Server/Client Architecture**: Implement Next.js Server and Client Components pattern
4. **Complete Routing Migration**: Replace all React Router with Next.js App Router
5. **Full UI Migration**: Complete replacement of MUI with shadcn/ui
6. **Production Ready**: Deploy as a complete Next.js application

Phase-Based Execution

The migration will be executed in **6 phases** using a **fresh Next.js codebase approach**:

1. **Create Fresh Next.js Codebase** - Initialize new Next.js 16 project from scratch
2. **Complete Configuration Setup** - Configure TypeScript, Tailwind, Apollo, shadcn, and all necessary tools
3. **Migrate Components by Functionality** - Move components in groups based on functionality/similarity
4. **Migrate Features & Routes** - Move complete features and implement Next.js routing

5. **Integration & Testing** - Complete feature parity and comprehensive testing
6. **Production Deployment** - Full Next.js production setup and optimization

Migration Approach: Fresh Codebase Strategy

Why this approach?
- ✅ Clean Next.js project structure from the start
- ✅ No legacy code conflicts
- ✅ Easier to configure everything correctly
- ✅ Components migrated in logical groups
- ✅ Better organization and maintainability
- ✅ Easier to test incrementally

Phase 1: Create Fresh Next.js Codebase

1.1 Create New Next.js Project

Duration: 0.5 day

Tasks:

- [ ] **Create new Next.js 16 project** using: `npx create-next-app@latest . --typescript --tailwind --app --no-src-dir`

- [ ] **Verify project structure** (app/, public/, next.config.js, tailwind.config.ts, tsconfig.json, package.json)

- [ ] **Test basic Next.js setup**: Run `npm run dev`, `npm run build`, verify localhost:3000

Deliverables:
- ✅ Fresh Next.js 16 project created
- ✅ Basic project structure verified
- ✅ Dev server and build working

Phase 2: Complete Configuration Setup

2.1 TypeScript Configuration

Duration: 0.5 day

Tasks:

- [ ] **Update `tsconfig.json`** with path aliases and strict settings
- [ ] Verify TypeScript compilation: `npx tsc --noEmit`

2.2 Tailwind CSS Configuration

Duration: 0.5 day

Tasks:

- [ ] **Tailwind is already installed** (from create-next-app), verify configuration
- [ ] Update `app/globals.css`
- [ ] Add Tailwind directives (already present)
- [ ] Add CSS variables for theme colors
- [ ] Migrate critical SCSS variables to CSS variables
- [ ] Add global styles if needed

2.3 Install & Configure shadcn/ui

Duration: 0.5 day

Tasks:

- [ ] **Initialize shadcn/ui**
- [ ] **Configure `components.json`**
- [ ] **Install initial shadcn components**
- [ ] Verify components are in `components/ui/` directory

2.4 Apollo Client Configuration

Duration: 1 day

Tasks:

- [ ] Install Apollo Client dependencies
- [ ] Create `lib/apollo-client.ts` with Next.js 16 SSR compatibility
- [ ] Create `lib/apollo-provider.tsx` for Client Components

- [ ] Test Apollo Client initialization

2.5 State Management Configuration

Duration: 0.5-1 day

Tasks (Choose one approach):

- [ ] **Option A: Zustand (Recommended)**: Install Zustand, create store directory structure
- [ ] **Option B: Redux Toolkit**: Install Redux dependencies, create `lib/store.ts` with Redux store setup (Maintain)

2.6 Environment Variables & Path Configuration

Duration: 0.5 day

Tasks:

- [ ] Set up environment variables (`.env.local`, `.env.development`, `.env.production`)
- [ ] Map `REACT_APP_*` → `NEXT_PUBLIC_*`
- [ ] Create directory structure (components/, lib/, hooks/, store/, types/)
- [ ] Copy static assets from `src-old/public/` to `public/`

2.7 Root Layout Setup

Duration: 0.5 day

Tasks:

- [ ] Update `app/layout.tsx` with providers (Apollo, State Management)
- [ ] Test root layout renders correctly

2.8 Configuration Testing & Validation (MANDATORY)

Duration: 0.5-1 day

Tasks:

- [ ] **Write test scripts** for foundation setup
- [ ] **Functionality Testing**

- [ ] Verify Next.js dev server starts (`npm run dev`)
- [ ] Verify Next.js build succeeds (`npm run build`)
- [ ] Test root layout renders correctly
- [ ] Test basic routing works
- [ ] Verify shadcn/ui components can be imported
- [ ] Test Tailwind CSS classes apply correctly
- [ ] Verify Apollo Client initializes
- [ ] Test state management store initialization
- [ ] **Integration Testing**:
- [ ] Test all providers work together
- [ ] Verify no console errors on startup
- [ ] Test environment variables load correctly
- [ ] **Run test suite**: `npm test` or `npm run test:watch`

Deliverables:
- ✅ TypeScript configured with path aliases
- ✅ Tailwind CSS configured with theme
- ✅ shadcn/ui installed and configured
- ✅ Apollo Client configured for Next.js
- ✅ State management configured (Zustand or Redux)
- ✅ Environment variables set up
- ✅ Root layout with providers configured
- ✅ Directory structure created
- ✅ **Test scripts written and passing**
- ✅ **All configuration tests passing**

Phase 3: Migrate Components by Functionality

Migration Strategy: Component Groups

Components will be migrated in **logical groups** based on:
1. **Functionality** - Components that work together

2. **Similarity** - Components with similar patterns
3. **Dependencies** - Components with few dependencies first
4. **Reusability** - Shared/common components first

3.1 Group 1: Utility & Helper Components (No Dependencies)

Duration: 1-2 days

Components to migrate:

- [ ] `src-old/utils/*` → `lib/utils/*`
- [ ] Convert `.js` → `.ts`
- [ ] Add TypeScript types
- [ ] Test each utility function
- [ ] `src-old/hooks/*` → `hooks/*`
- [ ] Convert `.js` → `.ts`
- [ ] Add TypeScript types
- [ ] Test each hook
- [ ] Simple helper components (if any)

Tasks:

- [ ] Copy files from `src-old/` to new locations
- [ ] Convert to TypeScript
- [ ] Update imports
- [ ] Write tests for each utility/hook
- [ ] Verify functionality

Testing:

- [ ] Write test scripts for utilities
- [ ] Write test scripts for hooks
- [ ] Run functionality tests

3.2 Group 2: Common/Shared UI Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/Avatar.jsx` → `components/Avatar.tsx`
- [ ] `src-old/components/LoadingSpinner.jsx` → `components/LoadingSpinner.tsx`
- [ ] `src-old/components/Alert.jsx` → `components/Alert.tsx`
- [ ] `src-old/components/common/*` → `components/common/*`
- [ ] `src-old/components/Icons/*` → `components/Icons/*`

Tasks:

- [ ] Copy component files
- [ ] Convert MUI → shadcn/ui
- [ ] Convert JSX → TSX
- [ ] Update icons (MUI icons → lucide-react)
- [ ] Update styling (makeStyles → Tailwind)
- [ ] Add TypeScript types
- [ ] Test each component

Testing:

- [ ] Write test scripts for each component
- [ ] Visual regression testing
- [ ] Functionality testing

3.3 Group 3: Button & Action Components

Duration: 1-2 days

Components to migrate:

- [ ] `src-old/components/CustomButtons/*` → `components/CustomButtons/*`
- All button variants
- Icon buttons
- Action buttons

Tasks:

- [ ] Convert MUI Button → shadcn Button
- [ ] Update styling
- [ ] Update icons
- [ ] Convert to TypeScript
- [ ] Test all button variants

3.4 Group 4: Form Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/Login/*` → `components/Login/*`
- [ ] `src-old/components/SignupForm/*` → `components/SignupForm/*`
- [ ] `src-old/components/PasswordReset/*` → `components/PasswordReset/*`
- [ ] `src-old/components/ForgotPassword/*` → `components/ForgotPassword/*`

Tasks:

- [ ] Convert MUI form components → shadcn form components
- [ ] Update form validation
- [ ] Convert to TypeScript
- [ ] Test form submissions
- [ ] Test validation

3.5 Group 5: Layout Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/Navbars/*` → `components/Navbars/*`
- [ ] `src-old/mui-pro/Sidebar/*` → `components/Sidebar/*`
- [ ] `src-old/components/SubHeader.jsx` → `components/SubHeader.tsx`

Tasks:

- [ ] Convert MUI components → shadcn
- [ ] Update navigation logic (React Router → Next.js)
- [ ] Convert to TypeScript
- [ ] Test navigation

3.6 Group 6: Post & Content Components

Duration: 3-4 days

Components to migrate:

- [ ] `src-old/components/Post/*` → `components/Post/*`
- [ ] `src-old/components/Comment/*` → `components/Comment/*`
- [ ] `src-old/components/SubmitPost/*` → `components/SubmitPost/*`
- [ ] `src-old/components/Quotes/*` → `components/Quotes/*`

Tasks:

- [ ] Convert MUI → shadcn
- [ ] Update GraphQL integration
- [ ] Convert to TypeScript
- [ ] Test post creation, display, editing
- [ ] Test comment system

3.7 Group 7: Profile & User Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/Profile/*` → `components/Profile/*`
- [ ] `src-old/components/UserPosts/*` → `components/UserPosts/*`

Tasks:

- [ ] Convert MUI → shadcn

- [ ] Update user data fetching
- [ ] Convert to TypeScript
- [ ] Test profile features

3.8 Group 8: Chat & Messaging Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/Chat/*` → `components/Chat/*`
- [ ] `src-old/components/PostChat/*` → `components/PostChat/*`
- [ ] `src-old/components/BuddyList/*` → `components/BuddyList/*`

Tasks:

- [ ] Convert MUI → shadcn
- [ ] Update WebSocket subscriptions
- [ ] Convert to TypeScript
- [ ] Test real-time messaging

3.9 Group 9: Notification Components

Duration: 1-2 days

Components to migrate:

- [ ] `src-old/components/Notifications/*` → `components/Notifications/*`

Tasks:

- [ ] Convert MUI → shadcn
- [ ] Update notification logic
- [ ] Convert to TypeScript
- [ ] Test notifications

3.10 Group 10: Voting & Interaction Components

Duration: 2-3 days

Components to migrate:

- [ ] `src-old/components/VotingComponents/*` → `components/VotingComponents/*`
- [ ] `src-old/components/PostActions/*` → `components/PostActions/*`

Tasks:

- [ ] Convert MUI → shadcn
- [ ] Update voting logic
- [ ] Convert to TypeScript
- [ ] Test voting functionality

3.11 Group 11: Complex/Feature Components

Duration: 3-4 days

Components to migrate:

- [ ] `src-old/components/Carousel/*` → `components/Carousel/*`
- [ ] `src-old/components/RequestAccess/*` → `components/RequestAccess/*`
- [ ] `src-old/components/SearchContainer/*` → `components/SearchContainer/*`
- [ ] `src-old/components/Settings/*` → `components/Settings/*`

Tasks:

- [ ] Convert MUI → shadcn
- [ ] Handle complex interactions
- [ ] Convert to TypeScript
- [ ] Test all features

3.12 Testing After Each Group (MANDATORY)

After migrating each group:

- [ ] Write test scripts for the group
- [ ] Run functionality tests
- [ ] Visual regression testing

- [ ] Integration testing with previously migrated components
- [ ] Fix any issues before moving to next group

Phase 4: Migrate Features & Routes

4.1 Create Next.js Route Structure

Duration: 1-2 days

Tasks:

- [ ] **Create App Router structure** based on old routes:
- Root layout and page
- (auth)/ group: login, signup, forgot-password, password-reset
- (dashboard)/ group: search, post, notifications, profile, control-panel
- Special routes: unauth, error, logout

- [ ] **Migrate layout components**:
    - `src-old/layouts/Scoreboard.jsx` → `app/(dashboard)/layout.tsx`
    - `src-old/layouts/Auth.jsx` → `app/(auth)/layout.tsx`
    - `src-old/layouts/Admin.jsx` → `app/(dashboard)/control-panel/layout.tsx`
    - `src-old/layouts/TokenExpired.jsx` → `app/unauth/page.tsx`

4.2 Migrate View/Page Components

Duration: 2-3 days

Tasks:

- [ ] **Migrate page components** from `src-old/views/`:
- `src-old/views/SearchPage/*` → `app/(dashboard)/search/page.tsx`
- `src-old/views/PostsPage/*` → `app/(dashboard)/post/page.tsx`
- `src-old/views/Profile/*` → `app/(dashboard)/profile/page.tsx`
- `src-old/views/ControlPanel/*` → `app/(dashboard)/control-panel/page.tsx`
- `src-old/views/LoginPage/*` → `app/(auth)/login/page.tsx`
- `src-old/views/SignupPage/*` → `app/(auth)/signup/page.tsx`
- `src-old/views/ForgotPassword/*` → `app/(auth)/forgot-password/page.tsx`

- `src-old/views/PasswordResetPage/*` → `app/(auth)/password-reset/page.tsx`
- `src-old/views/LandingPage/*` → `app/page.tsx`

☐ **Convert each page**:
- Update imports (React Router → Next.js)
- Convert to Server/Client Components as needed
- Update data fetching for Next.js
- Convert to TypeScript
- Test each page

4.3 Update Navigation & Routing

Duration: 1-2 days

Tasks:

☐ **Update all navigation components**:
- Replace `react-router-dom` Link → `next/link`
- Replace `useHistory()` → `useRouter()` from `next/navigation`
- Replace `useLocation()` → `usePathname()` from `next/navigation`
- Replace `useParams()` → `params` prop in page components

☐ **Update route guards**:
- Convert `PrivateRoute` → Next.js middleware
- Implement authentication checks in layouts
- [ ] **Test all routes**:
- Verify all routes are accessible
- Test route guards
- Test navigation

4.4 Migrate GraphQL Queries & Mutations

Duration: 1-2 days

Tasks:

☐ **Copy GraphQL files**:
- `src-old/graphql/query.jsx` → `lib/graphql/queries.ts`
- `src-old/graphql/mutations.jsx` → `lib/graphql/mutations.ts`
- `src-old/graphql/subscription.jsx` → `lib/graphql/subscriptions.ts`

☐ **Convert to TypeScript**:
- Add proper types for queries
- Add proper types for mutations
- Add proper types for subscriptions

☐ **Update usage**:
- Update imports in components
- Ensure compatibility with Next.js App Router
- Test all GraphQL operations

4.5 Testing & Validation (MANDATORY)

Duration: 1 day

Tasks:

☐ **Write test scripts** for routing:
- Test all routes
- Test navigation
- Test route guards

☐ **Functionality Testing**:
- Test all pages render correctly
- Test navigation between pages
- Test route guards work
- Test GraphQL operations

☐ **Run test suite**

Deliverables:
- ✅ All routes migrated to Next.js App Router
- ✅ All pages migrated
- ✅ Navigation updated
- ✅ GraphQL operations working
- ✅ **Test scripts written and passing**
- ✅ **All functionality tests passing**

Phase 5: Integration & Testing

5.1 Complete Integration Testing

Duration: 2-3 days

Tasks:

- [ ] **Write comprehensive integration test scripts**
- [ ] **End-to-End Feature Testing**:
    - [ ] Test complete user authentication flow
    - [ ] Test post creation and display flow
    - [ ] Test comment system flow
    - [ ] Test voting system flow
    - [ ] Test search functionality flow
    - [ ] Test profile management flow
    - [ ] Test notification system flow
    - [ ] Test chat/messaging flow
- [ ] **Cross-Component Integration**:
    - [ ] Test components work together
    - [ ] Test state management across components
    - [ ] Test GraphQL operations across features
    - [ ] Test navigation between features
- [ ] **Performance Testing**:
    - [ ] Measure page load times
    - [ ] Test bundle sizes
    - [ ] Verify Core Web Vitals
    - [ ] Test image optimization
- [ ] **Run full test suite**: `npm test`

Deliverables:
- ✅ All features integrated and working
- ✅ End-to-end testing complete

- ✅ Cross-component integration verified
- ✅ Performance benchmarks met
- ✅ **Integration test scripts written and passing**
- ✅ **All functionality tests passing**

Phase 6: Production Deployment & Optimization

6.1 Next.js Production Optimization

Duration: 2-3 days

Tasks:

☐ **Implement Next.js optimizations**:
- Image optimization with `next/image`
- Font optimization
- Code splitting
- Dynamic imports for heavy components

☐ **Optimize bundle size**:
- Remove unused dependencies
- Tree-shake unused code
- Analyze bundle with `@next/bundle-analyzer`

☐ **Implement caching strategies**:
- API route caching
- Static page generation where appropriate
- ISR for dynamic content

6.2 SEO & Metadata (Next.js Native)

Duration: 1-2 days

Tasks:

☐ **Migrate `react-helmet-async` → Next.js Metadata API**:
- Remove `react-helmet-async` dependency
- Use Next.js `metadata` export in pages

☐ **Add metadata to all pages**:
- Title

- Description
- Open Graph tags
- Twitter Card tags

- [ ] **Implement structured data** (JSON-LD)

- [ ] **Test SEO** with tools

6.3 Complete Code Cleanup & Removal

Duration: 1-2 days

Tasks:

- [ ] **Completely remove old dependencies**:
- Uninstall all Material-UI packages
- Uninstall React Router packages
- Uninstall all Vite packages
- Remove all unused utilities and dependencies

- [ ] **Delete all old framework files**:
- Delete `src-old/` directory (backup of old code)
- Delete old SCSS files
- Delete old route files
- Remove all React Router route definitions

- [ ] **Update documentation**

- [ ] **Code review and refactoring**

6.4 Next.js Production Deployment

Duration: 1-2 days

Tasks:

- [ ] **Update build scripts** for Next.js production

- [ ] **Configure `next.config.js` for production** (Next.js 16)

- [ ] **Update CI/CD pipelines**

- [ ] **Create deployment documentation**

- [ ] **Set up monitoring and error tracking**

6.5 Final Testing & Validation (MANDATORY)

Duration: 1-2 days

Tasks:

- [ ] **Write production test scripts**
- [ ] **Functionality Testing**:
  - [ ] Test production build succeeds (`npm run build`)
  - [ ] Test production server starts (`npm run start`)
  - [ ] Verify all features work in production mode
  - [ ] Test error handling in production
  - [ ] Verify no console errors/warnings
- [ ] **Performance Testing**:
  - [ ] Measure Core Web Vitals (LCP, FID, CLS)
  - [ ] Test bundle size is acceptable
  - [ ] Verify image optimization works
  - [ ] Test caching strategies
- [ ] **SEO Testing**:
  - [ ] Verify metadata on all pages
  - [ ] Test Open Graph tags
  - [ ] Verify structured data
- [ ] **Security Testing**:
  - [ ] Verify environment variables are secure
  - [ ] Test authentication security
  - [ ] Verify no sensitive data exposed
- [ ] **Run full test suite**: Final validation before deployment

Deliverables:
- ✅ Optimized performance

- ✅ SEO implemented
- ✅ Codebase cleaned
- ✅ Ready for production
- ✅ **Production test scripts written and passing**
- ✅ **All functionality tests passing**
- ✅ **Performance benchmarks met**
- ✅ **Production build verified**

Component Migration Checklist (Reference)

MUI Component	shadcn/ui Component	Notes
`Button`	`Button`	Direct replacement
`Card`, `CardContent`, `CardHeader`	`Card`	Similar structure
`Dialog`	`Dialog`	Direct replacement
`TextField`	`Input`	May need `Label` component
`Select`	Custom with `Select`	Use Radix UI Select
`Switch`	`Switch`	Direct replacement
`Checkbox`	`Checkbox`	Direct replacement
`Avatar`	`Avatar`	Direct replacement
`Tooltip`	`Tooltip`	Direct replacement
`Menu`, `MenuItem`	`DropdownMenu`	Different API
`Tabs`, `Tab`	`Tabs`	Similar structure
`Snackbar`	`Toast`	Different API (use sonner)
`IconButton`	`Button` variant=`"ghost"`	Styling difference
`Typography`	Native HTML + Tailwind	Use semantic HTML
`Grid`, `GridItem`	Tailwind Grid	CSS Grid/Flexbox
`Paper`	`Card` or `div`	Styling with Tailwind
`AppBar`	Custom header	Build with shadcn components
`Drawer`	`Sheet`	Similar functionality
`Pagination`	Custom component	Build with shadcn Button
`CircularProgress`	`Skeleton` or custom	Loading states

MUI Component	shadcn/ui Component	Notes
`LinearProgress`	Custom component	Progress bars

Risk Assessment & Mitigation

High Risk Areas

1. **Complex MUI Components**
    - **Risk**: Custom MUI components in `mui-pro/` may not have direct shadcn equivalents
    - **Mitigation**: Build custom components using shadcn primitives.
2. **State Management with SSR**
    - **Risk**: Redux Persist may have issues with Next.js SSR
    - **Mitigation**: Use `next-redux-wrapper` or similar, test thoroughly
3. **GraphQL Subscriptions**
    - **Risk**: WebSocket connections may need adjustment for Next.js 16
    - **Mitigation**:
        - Test subscriptions early
        - Use Apollo Client's Next.js support
4. **Authentication Flow**
    - **Risk**: Token management and protected routes need careful handling
    - **Mitigation**: Implement middleware for route protection, test all auth flows
5. **Third-party Integrations**
    - **Risk**: Some libraries may not be Next.js compatible
    - **Mitigation**: Identify early, find alternatives or workarounds

Medium Risk Areas

1. **Performance**

- **Risk**: Bundle size may increase initially
- **Mitigation**: Monitor bundle size, use dynamic imports, optimize images

2. **Testing**
   - **Risk**: Tests may break during migration
   - **Mitigation**: Update tests incrementally, maintain test coverage

Timeline Estimate

Conservative Estimate

Phase	Duration	Dependencies
Phase 1: Create Fresh Next.js Codebase	0.5 day	None
Phase 2: Complete Configuration Setup	2-3 days	Phase 1
Phase 3: Migrate Components by Functionality	4-5 weeks	Phase 2
Phase 4: Migrate Features & Routes	1-2 weeks	Phase 3
Phase 5: Integration & Testing	1.5-2 weeks	Phase 4
Phase 6: Production Deployment	1-1.5 weeks	Phase 5
Total	**8-10 weeks**	

Aggressive Estimate (Parallel Work)

Phase	Duration	Dependencies
Phase 1: Create Fresh Next.js Codebase	0.5 day	None
Phase 2: Complete Configuration Setup	1-2 days	Phase 1
Phase 3: Migrate Components by Functionality	3-4 weeks	Phase 2
Phase 4: Migrate Features & Routes	1 week	Phase 3
Phase 5: Integration & Testing	1-1.5 weeks	Phase 4
Phase 6: Production Deployment	0.5-1 week	Phase 5
Total	**6-8 weeks**	

Note: Timeline assumes 1-2 developers working full-time.

Success Criteria

Technical Criteria

- [ ] All routes migrated and functional
- [ ] All components migrated to shadcn/ui
- [ ] TypeScript compilation with no errors
- [ ] All tests passing
- [ ] Bundle size within acceptable limits
- [ ] Performance metrics met (LCP, FID, CLS)
- [ ] SEO metadata implemented
- [ ] Cross-browser compatibility verified

Functional Criteria

- [ ] All features working as before
- [ ] Visual parity with current design
- [ ] Responsive design maintained
- [ ] Accessibility standards met
- [ ] Authentication flows working
- [ ] GraphQL operations functional

Quality Criteria

- [ ] Code review completed
- [ ] Documentation updated
- [ ] No critical bugs
- [ ] Performance improved or maintained
- [ ] Developer experience improved

Post-Migration Tasks

Immediate (Week 1)

- [ ] Monitor error rates
- [ ] Collect user feedback
- [ ] Fix critical bugs
- [ ] Performance monitoring

Short-term (Month 1)

- [ ] Optimize based on analytics
- [ ] Refine UI/UX based on feedback
- [ ] Add missing features
- [ ] Improve documentation

Long-term (Quarter 1)

- [ ] Plan next features
- [ ] Consider additional optimizations
- [ ] Evaluate new Next.js features
- [ ] Team training on new stack

Resources & References

Documentation

- Next.js 16 Documentation
- Next.js 16 Release Notes
- React 19 Documentation
- React 19 Upgrade Guide
- shadcn/ui Documentation
- Tailwind CSS Documentation
- Apollo Client Next.js Guide

- Redux Toolkit Documentation

Migration Guides

- React Router to Next.js
- Material-UI Migration Guide
- Vite to Next.js
- React 17 to React 19 Migration
- Next.js 15 to 16 Upgrade Guide

Tools

- TypeScript Playground
- Tailwind CSS Playground
- Bundle Analyzer

Notes

Complete Migration Approach

- **This is a FULL framework replacement**, not a parallel setup
- **Vite will be completely removed** - no dual build systems
- **React Router will be completely removed** - Next.js App Router is the only routing solution
- **Material-UI will be completely removed** - shadcn/ui is the only UI library
- **All old framework files will be deleted** after migration is complete
- The migration should be done incrementally to minimize risk
- Regular code reviews and testing are essential
- Communication with the team is crucial throughout the process
- **Plan for a complete cutover** - Next.js will be the only framework after migration

- Test thoroughly before removing old dependencies
- Ensure all features work in Next.js before deleting old code