Supply Chain Club - Tech Stack Design Plan

Phase 1: Frontend-First Approach (Current)

Core Frontend Stack

- **React 18**+ with **TypeScript** - Modern, type-safe development
- **Vite** - Fast build tool and development server
- **Tailwind CSS** - Utility-first styling for rapid UI development
- **React Router** - Client-side routing for multi-page functionality

State Management & Data Fetching

- **Zustand** or **Redux Toolkit** - Lightweight state management
- **TanStack Query (React Query)** - Server state management and caching
- **Axios** - HTTP client for API calls (future-ready)

UI Components & Styling

- **Headless UI** or **Radix UI** - Accessible, unstyled components
- **Framer Motion** - Smooth animations and transitions
- **React Hook Form** - Efficient form handling with validation
- **Zod** - Runtime type validation for forms and API responses

Development Tools

- **ESLint** + **Prettier** - Code linting and formatting
- **Husky** + **lint-staged** - Pre-commit hooks for code quality
- **Storybook** (optional) - Component development and documentation

Deployment

- **Vercel** or **Netlify** - Easy frontend deployment with CI/CD
- **GitHub Actions** - Automated testing and deployment pipeline

Phase 2: Full-Stack Evolution (Future)

Backend Stack

- **Node.js** with **Express.js** - RESTful API server

- **TypeScript** - Consistent typing across frontend and backend
- **MongoDB** with **Mongoose** - Document database with ODM
- **JWT** - Authentication and authorization
- **bcryptjs** - Password hashing
- **Helmet** - Security middleware
- **CORS** - Cross-origin resource sharing configuration

Database & Storage

- **MongoDB Atlas** - Cloud-hosted MongoDB
- **Cloudinary** or **AWS S3** - Image and file storage
- **Redis** (optional) - Caching and session storage

API Development

- **Express Validator** - Request validation middleware
- **Morgan** - HTTP request logging
- **Compression** - Response compression
- **Rate Limiting** - API protection

Phase 3: Mobile App Expansion

Mobile Development

- **React Native** with **TypeScript** - Cross-platform mobile development
- **Expo** - Managed React Native workflow for easier development
- **React Navigation** - Navigation library for mobile screens

Shared Components Strategy

```
src/
├── components/
│   ├── shared/          # Components used by both web and
mobile
│   ├── web/             # Web-specific components
│   └── mobile/          # Mobile-specific components
├── hooks/               # Shared custom hooks
├── utils/               # Shared utility functions
├── types/               # Shared TypeScript types
└── services/            # API calls and business logic
```

Code Sharing Approach

- **Monorepo with Nx** or **Lerna** - Manage web and mobile apps together

- **Shared packages** for business logic, types, and utilities
- **Platform-specific UI** components while sharing core logic

Recommended Project Structure

Initial React App Structure

```
supply-chain-club/
├── public/
├── src/
│   ├── components/
│   │   ├── ui/           # Reusable UI components
│   │   ├── layout/       # Layout components
│   │   └── features/     # Feature-specific components
│   ├── pages/            # Page components
│   ├── hooks/            # Custom React hooks
│   ├── services/         # API services
│   ├── types/            # TypeScript type definitions
│   ├── utils/            # Helper functions
│   ├── store/            # State management
│   └── styles/           # Global styles
├── tests/
├── docs/
└── deployment/
```

Development Workflow Recommendations

Version Control & Collaboration

- **Git Flow** or **GitHub Flow** for branching strategy
- **Conventional Commits** for clear commit messaging
- **Pull Request** templates for code review process

Testing Strategy

- **Vitest** - Fast unit testing framework
- **React Testing Library** - Component testing
- **Cypress** or **Playwright** - End-to-end testing (future)

Performance & SEO Considerations

- **React.lazy()** - Code splitting for better performance
- **React Helmet** - Meta tags and SEO optimization

- **Web Vitals** monitoring - Performance tracking

Migration Path Strategy

Phase 1 → Phase 2 Transition

1. Set up Express.js API alongside existing React app
2. Gradually move hardcoded data to API endpoints
3. Implement authentication and user management
4. Add database integration for dynamic content

Phase 2 → Phase 3 Transition

1. Extract shared business logic into separate packages
2. Set up React Native project with Expo
3. Create mobile-specific UI components
4. Implement shared navigation patterns

Cost Considerations

Free Tier Options

- **Vercel/Netlify** - Frontend hosting
- **MongoDB Atlas** - 512MB free cluster
- **GitHub** - Code repository and Actions (2000 minutes/month)
- **Cloudinary** - 25GB storage and 25GB bandwidth

Paid Upgrades (Future)

- **MongoDB Atlas** - $9+/month for production clusters
- **Vercel Pro** - $20+/month for advanced features
- **Apple Developer** - $99/year for iOS app distribution
- **Google Play** - $25 one-time registration fee

Success Metrics & Milestones

Phase 1 Goals

- [ ] Responsive design across devices
- [ ] Fast loading times (<3 seconds)
- [ ] Accessible UI components
- [ ] Mobile-friendly navigation

Phase 2 Goals

- [ ] User authentication system
- [ ] Dynamic content management
- [ ] API documentation
- [ ] Database backup strategy

Phase 3 Goals

- [ ] Cross-platform mobile app
- [ ] Push notifications
- [ ] Offline functionality
- [ ] App store deployment

Alternative Considerations

If React Native Proves Challenging

- **Progressive Web App (PWA)** - Web app that feels native on mobile
- **Capacitor** - Wrap your React app as a native mobile app
- **Flutter** - If you want to explore non-React mobile development

Backend Alternatives to MERN

- **Next.js Full-Stack** - API routes with React for simpler architecture
- **Supabase** - Backend-as-a-Service with PostgreSQL
- **Firebase** - Google's BaaS platform with real-time features

This tech stack design provides a clear evolution path from a simple React website to a full-featured web and mobile platform, while maintaining consistency in your technology choices and maximizing code reuse opportunities.