Board Spark AI - Architecture Documentation

Overview

Board Spark AI is a modern web application for managing board papers, compliance, and governance. Built with React 18, TypeScript, and Supabase, it follows a component-based architecture with emphasis on performance, security, and maintainability.

Technology Stack

Frontend

- **React 18.3.1** - UI library with concurrent features
- **TypeScript 5.8.3** - Type-safe JavaScript
- **Vite 5.4.19** - Fast build tool and dev server
- **TailwindCSS** - Utility-first CSS framework
- **shadcn/ui** - High-quality React components built on Radix UI

Backend & Infrastructure

- **Supabase** - Backend-as-a-Service
- PostgreSQL database with Row Level Security (RLS)
- Authentication with JWT tokens
- Real-time subscriptions
- File storage
- **Vercel** - Deployment platform with edge functions

State Management

- **TanStack Query (React Query)** - Server state management
- **React Context** - Auth state and global UI state
- **Local State** - Component-level state with useState/useReducer

Testing

- **Vitest** - Unit testing framework
- **Playwright** - End-to-end testing
- **React Testing Library** - Component testing

Project Structure

```
board-spark-ai/
├── src/
│   ├── components/          # React components
│   │   ├── ui/             # Reusable UI components (shadcn/ui)
│   │   ├── dashboard/      # Dashboard-specific components
│   │   ├── settings/       # Settings-specific components
│   │   ├── templates/      # Template management components
│   │   └── *.tsx           # Shared components
│   ├── pages/              # Route pages
│   ├── hooks/              # Custom React hooks
│   ├── lib/                # Utility functions and helpers
│   │   ├── utils.ts        # General utilities
│   │   ├── errorHandling.ts # Error handling utilities
│   │   ├── performance.ts  # Performance monitoring
│   │   ├── cache.ts        # Caching utilities
│   │   └── lazyLoad.tsx    # Lazy loading utilities
│   ├── types/              # TypeScript type definitions
│   ├── integrations/       # External service integrations
│   │   └── supabase/       # Supabase client and types
│   └── App.tsx             # Main application component
├── public/                 # Static assets
├── e2e/                    # End-to-end tests
├── supabase/             # Supabase migrations and config
└── docs/                 # Documentation
```

Core Architectural Patterns

1. Component Architecture

Component Hierarchy

```
App
├── AuthProvider (Context)
├── QueryClientProvider (React Query)
└── Router
    ├── Public Routes (Landing, Auth)
    └── Protected Routes
        ├── Dashboard
        ├── Board Papers
        ├── Compliance
        └── Settings
```

Component Types

Page Components (`src/pages/`)
- Top-level route components
- Handle data fetching and business logic
- Compose smaller components
- Example: `Dashboard.tsx` , `BoardPapers.tsx`

Feature Components (`src/components/`)
- Domain-specific components
- Encapsulate feature logic
- Example: `BoardPaperTemplateBuilder.tsx` , `COIManagement.tsx`

UI Components (`src/components/ui/`)
- Reusable, presentational components
- No business logic
- Highly composable
- Example: `Button` , `Card` , `Dialog`

Shared Components (`src/components/templates/`)
- Components shared across features
- Reduce code duplication
- Example: `SharedTemplateComponents.tsx`

2. State Management Strategy

Server State (React Query)

Used for all data from Supabase:

```
const { data, isLoading, error } = useQuery({
  queryKey: ['board-papers', orgId],
  queryFn: async () => {
    const { data, error } = await supabase
      .from('board_papers')
      .select('*')
      .eq('org_id', orgId);
    if (error) throw error;
    return data;
  },
  staleTime: 5 * 60 * 1000, // 5 minutes
});
```

Benefits:
- Automatic caching and invalidation
- Background refetching
- Optimistic updates
- Request deduplication

Client State (React Context)

Used for authentication and global UI state:

```
const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }) {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  // Auth logic...

  return (
    <AuthContext.Provider value={{ user, loading }}>
      {children}
    </AuthContext.Provider>
  );
}
```

Local State (useState/useReducer)

Used for component-specific UI state:
- Form inputs
- Modal visibility
- Accordion expansion
- Temporary UI state

3. Data Flow

```
User Action
    ↓
Component Event Handler
    ↓
React Query Mutation / API Call
    ↓
Supabase Client
    ↓
PostgreSQL + RLS
    ↓
Response
    ↓
React Query Cache Update
    ↓
Component Re-render
```

4. Performance Optimization

Code Splitting

```
// Lazy load heavy components
const Dashboard = lazyRoute(() => import('@/pages/Dashboard'));
const BoardPapers = lazyRoute(() => import('@/pages/BoardPapers'));
```

Memoization

```
// Memoize expensive computations
const metrics = useMemo(() => {
  return calculateDashboardMetrics(data);
}, [data]);

// Memoize callbacks
const handleSubmit = useCallback((values) => {
  submitForm(values);
}, []);

// Memoize components
const ExpensiveComponent = memo(({ data }) => {
  // Component logic
});
```

Caching Strategy

```
// Multi-level caching
// 1. React Query cache (5 minutes)
// 2. Browser cache (localStorage/sessionStorage)
// 3. Service Worker cache (offline support)

const data = await getCacheOrSet(
  'board-papers',
  async () => fetchBoardPapers(),
  5 * 60 * 1000 // 5 minutes
);
```

5. Security Architecture

Authentication Flow

```
1. User signs in → Supabase Auth
2. Receives JWT token
3. Token stored in httpOnly cookie
4. Token included in all API requests
5. Supabase validates token
6. RLS policies enforce access control
```

Row Level Security (RLS)

All database tables have RLS policies:

```
-- Example: Board members can only see their organization's data
CREATE POLICY "Users can view their org's board papers"
  ON board_papers FOR SELECT
  USING (
    org_id IN (
      SELECT org_id FROM profiles WHERE id = auth.uid()
    )
  );
```

Security Headers

Configured in `vercel.json`:
- Content Security Policy (CSP)
- X-Frame-Options: DENY
- X-Content-Type-Options: nosniff
- Referrer-Policy
- Permissions-Policy

6. Error Handling

Error Boundary

Catches React component errors:

```
<ErrorBoundary fallback={<ErrorFallback />}>
  <App />
</ErrorBoundary>
```

API Error Handling

Centralized error handling:

```
try {
  const { data, error } = await supabase.from('table').select();
  if (error) throw error;
  return data;
} catch (error) {
  const message = getUserFriendlyError(error);
  toast({ title: 'Error', description: message, variant: 'destructive' });
}
```

Database Schema

Core Tables

profiles
- User profile information
- Links to auth.users
- Organization membership

organizations
- Organization details
- Settings and preferences

boards
- Board definitions
- Linked to organizations

board_members
- Board membership
- Roles and permissions

board_papers
- Board paper documents
- Versioning and status

compliance_items
- Compliance tracking
- Due dates and status

document_snapshots
- Immutable document versions
- Checksums for integrity

Relationships

```
organizations
    ↓ (1:N)
boards
    ↓ (1:N)
board_papers
    ↓ (1:N)
document_snapshots

organizations
    ↓ (1:N)
profiles
    ↓ (N:M via board_memberships)
boards
```

API Integration

Supabase Client

```
import { createClient } from '@supabase/supabase-js';

export const supabase = createClient(
  import.meta.env.VITE_SUPABASE_URL,
  import.meta.env.VITE_SUPABASE_PUBLISHABLE_KEY
);
```

Query Patterns

Simple Query

```
const { data, error } = await supabase
  .from('board_papers')
  .select('*')
  .eq('org_id', orgId);
```

Join Query

```
const { data, error } = await supabase
  .from('board_papers')
  .select(`
    *,
    board:boards(*),
    created_by:profiles(*)
  `)
  .eq('org_id', orgId);
```

Real-time Subscription

```
const subscription = supabase
  .channel('board_papers')
  .on('postgres_changes', {
    event: '*',
    schema: 'public',
    table: 'board_papers'
  }, (payload) => {
    // Handle changes
  })
  .subscribe();
```

Build & Deployment

Development

```
npm run dev          # Start dev server
npm run test         # Run unit tests
npm run test:e2e     # Run E2E tests
npm run lint         # Lint code
```

Production Build

```
npm run build        # Build for production
npm run preview      # Preview production build
```

Deployment Pipeline

1. Push to GitHub
2. Vercel detects changes
3. Runs build process
4. Runs tests
5. Deploys to preview/production
6. Runs Lighthouse CI

Performance Targets

Lighthouse Scores

- Performance: 90+
- Accessibility: 90+
- Best Practices: 90+
- SEO: 90+

Core Web Vitals

- LCP (Largest Contentful Paint): < 2.5s
- FID (First Input Delay): < 100ms
- CLS (Cumulative Layout Shift): < 0.1

Bundle Size

- Initial JS: < 200KB gzipped

- Total JS: < 500KB gzipped

Testing Strategy

Unit Tests

- Utility functions
- Custom hooks
- Error handling
- Target: 70%+ coverage

Integration Tests

- Component interactions
- Form submissions
- API calls
- Target: Key user flows covered

E2E Tests

- Critical user journeys
- Authentication flows
- Board paper creation
- Compliance management

Future Improvements

Phase 3 (Current)

- Template refactoring
- Performance optimizations
- Caching strategies
- Documentation improvements

Phase 4 (Planned)

- Accessibility improvements (WCAG 2.1 AA)
- Mobile optimization
- Increased test coverage (80%+)
- PWA enhancements
- Internationalization (i18n)

Contributing

Code Style

- Use TypeScript for all new code
- Follow ESLint configuration
- Use Prettier for formatting
- Write JSDoc comments for public APIs

Git Workflow

1. Create feature branch from `main`
2. Make changes with descriptive commits
3. Run tests and linting
4. Create pull request
5. Code review
6. Merge to `main`

Commit Messages

Follow conventional commits:

```
feat: Add template management hook
fix: Resolve calendar date picker issue
docs: Update architecture documentation
perf: Optimize dashboard rendering
test: Add tests for phone validation
```

Resources

- React Documentation (https://react.dev)
- TypeScript Handbook (https://www.typescriptlang.org/docs/)
- Supabase Documentation (https://supabase.com/docs)
- TanStack Query (https://tanstack.com/query/latest)
- Tailwind CSS (https://tailwindcss.com/docs)
- shadcn/ui (https://ui.shadcn.com)

Last Updated: November 1, 2025
Version: 1.0.0