

# BellsNotice - Comprehensive Project Documentation

## Table of Contents

1. Project Overview
  2. Technical Architecture
  3. Database Schema
  4. Application Features
  5. User Roles & Permissions
  6. API Integration
  7. Deployment & Configuration
  8. Development Workflow
- 

## Project Overview

### Purpose

BellsNotice is a modern, web-based notice board application designed for educational institutions, specifically Bells University of Technology. The platform facilitates seamless communication between students, faculty representatives (reps), and administrators by providing a centralized system for creating, sharing, and managing notices.

### Core Objectives

- Provide a user-friendly interface for viewing and searching notices
- Enable authorized representatives to publish notices efficiently
- Implement a request-based system for non-representative users to suggest notices
- Support media-rich content including images, videos, and documents
- Maintain organized categorization through tags and filtering
- Ensure secure authentication and role-based access control

### Target Audience

- **Regular Users:** Students who can view, search, save, and comment on notices
  - **Representatives (Reps):** Authorized users who can create, edit, and manage notices, plus handle notice requests
  - **Administrators:** System administrators with full control over all notices and requests
-

## Technical Architecture

### Technology Stack

#### Frontend Framework

- **Next.js 16.0.10**: React-based full-stack framework with Server Components
- **React 19.2.0**: Latest React for building interactive user interfaces
- **TypeScript 5**: Type-safe JavaScript for better code quality and maintainability

#### UI Component Library

- **Radix UI**: Headless UI components for accessible design
- **Tailwind CSS 4.1.9**: Utility-first CSS framework for rapid styling
- **Lucide React**: Comprehensive icon library
- **shadcn/ui**: Pre-built component library built on Radix UI and Tailwind

#### Backend & Database

- **Supabase**: Backend-as-a-Service providing:
  - PostgreSQL database
  - Authentication system
  - File storage (Supabase Storage)
  - Real-time subscriptions
- **@supabase/ssr**: Server-side rendering support for Supabase

#### Additional Libraries

- **React Hook Form 7.60.0**: Form management with validation
- **Zod 3.25.76**: Schema validation
- **date-fns 4.1.0**: Date manipulation and formatting
- **recharts 2.15.4**: Data visualization for analytics
- **sonner 1.7.4**: Toast notifications
- **next-pwa 5.6.0**: Progressive Web App capabilities

#### Project Structure

```
bellsnote/
├── app/                # Next.js App Router pages
│   ├── auth/          # Authentication pages
│   ├── dashboard/     # User dashboard
│   ├── filter/        # Notice filtering
│   ├── notice/        # Notice creation and viewing
│   ├── notices/       # All notices listing
│   ├── profile/       # User profiles
│   └── request-notice/ # Notice request submission
```

saved/	# Saved notices
search/	# Search functionality
admin/	# Admin panel
globals.css	# Global styles
layout.tsx	# Root layout
components/	# Reusable components
ui/	# shadcn/ui components
pages/	# Page-specific components
navigation/	# Navigation components
lib/	# Utility functions
supabase/	# Supabase client configuration
utils.ts	# Helper utilities
hooks/	# Custom React hooks
public/	# Static assets
styles/	# Additional styles

## Key Design Patterns

**1. Server-Side Rendering (SSR)** Next.js App Router uses Server Components by default, improving performance and SEO. Client components are marked with “use client” directive when interactivity is needed.

## 2. Supabase Client Pattern

```
// Browser client for client-side operations
createClient() from @/lib/supabase/client

// Server client for server-side operations
createClient() from @/lib/supabase/server
```

**3. Authentication State Management** Authentication state is managed through: - Supabase Auth for user sessions - Session storage for admin authentication - Context and hooks for state distribution

**4. Role-Based Access Control** User roles are stored in the `profiles` table and checked before protected operations: - `user_type: "rep"` - Can create and manage notices - `user_type: "user"` - Read-only access, can request notices - Admin access through separate authentication flow

---

## Database Schema

### Core Tables

`profiles`

- id: UUID (primary key, references auth.users)
- email: text
- display\_name: text
- user\_type: text ('user' | 'rep')
- level: text ('100' | '200' | '300' | '400' | '500')
- program: text ('undergraduate' | 'postgraduate')
- college: text (COLNAS, COLMANS, COLFAST, COLENG, COLENVS)
- department: text
- matric\_number: text
- profile\_image\_url: text
- read\_receipt\_visibility: boolean
- created\_at: timestamp

#### notices

- id: UUID (primary key)
- title: text (required)
- description: text (required)
- author\_id: UUID (references profiles)
- view\_count: integer (default: 0)
- is\_important: boolean (default: false)
- is\_featured: boolean (default: false)
- expires\_at: timestamp (nullable)
- created\_at: timestamp

#### notice\_media

- id: UUID (primary key)
- notice\_id: UUID (references notices)
- media\_type: text ('image' | 'video' | 'file')
- media\_url: text
- is\_link: boolean
- created\_at: timestamp

#### notice\_requests

- id: UUID (primary key)
- title: text
- description: text
- requester\_id: UUID (references profiles)
- rep\_id: UUID (references profiles)
- status: text ('pending' | 'approved' | 'rejected')
- response\_message: text (nullable)
- responded\_at: timestamp (nullable)
- notice\_id: UUID (references notices, nullable)
- created\_at: timestamp

#### notice\_request\_media

- id: UUID (primary key)
- request\_id: UUID (references notice\_requests)
- media\_type: text
- media\_url: text
- created\_at: timestamp

#### tags

- id: UUID (primary key)
- name: text (unique)
- created\_at: timestamp

#### notice\_tags

- notice\_id: UUID (references notices)
- tag\_id: UUID (references tags)
- created\_at: timestamp

#### comments

- id: UUID (primary key)
- content: text
- user\_id: UUID (references profiles)
- notice\_id: UUID (references notices)
- created\_at: timestamp

#### reactions

- id: UUID (primary key)
- user\_id: UUID (references profiles)
- notice\_id: UUID (references notices)
- created\_at: timestamp

#### saved\_notices

- user\_id: UUID (references profiles)
- notice\_id: UUID (references notices)
- created\_at: timestamp

#### Database Relationships

- **One-to-Many:** Profile → Notices, Profile → Comments
- **Many-to-Many:** Notices ↔ Tags (through notice\_tags)
- **Many-to-Many:** Users ↔ Saved Notices (through saved\_notices)
- **One-to-One:** NoticeRequest → Notice (when approved)

## Application Features

### 1. Authentication System

#### User Login

- Supports login via email or matric number
- Matric number lookup in profiles table to retrieve associated email
- Secure password authentication through Supabase Auth
- Session persistence with automatic token refresh

#### Admin Login

- Separate authentication flow using admin password
- Session-based authentication stored in sessionStorage
- Redirects to admin panel upon successful login

#### Registration

- Multi-step registration form
- Fields: Email, Display Name, Account Type, Level, Program, College, Department, Matric Number
- Dynamic department selection based on college choice
- Email-based account creation

### 2. Notice Management

#### Viewing Notices

- **Home Page:** Displays important, featured, and latest notices
- **Important Notices:** Highlighted in red with carousel navigation
- **Featured Notices:** Highlighted in blue with carousel navigation
- **Latest Notices:** Random selection of 12 notices
- **Individual Notice View:** Full details with media, comments, and reactions

#### Creating Notices (Reps & Admins)

- Rich content creation with title and description
- Media upload support:
  - Image uploads (multiple files)
  - Video uploads (multiple files)
  - Document uploads (multiple files)
  - URL support for all media types
- Tag-based categorization
- Mark as important (admin only)
- Expiry date setting
- Automatic view counter

### **Editing Notices**

- Full edit capabilities for notice authors
- Update title, description, tags, and media
- Modify important/featured status (admin only)

### **Deleting Notices**

- Cascade deletion of associated:
  - Media files
  - Comments
  - Tags
  - Reactions

## **3. Notice Request System**

### **Submitting Requests**

- Regular users can request notices through representatives
- Include title, description, and media
- Select target representative
- Track request status

### **Managing Requests (Reps)**

- View pending requests with timestamps
- Accept requests to publish as notices
- Decline with optional feedback message
- Media automatically transfers to published notice
- Request history tracking

## **4. Search & Filtering**

### **Search Functionality**

- Full-text search across notice titles and descriptions
- Real-time search results
- Highlighted matching text

### **Advanced Filtering**

- Filter by tags
- Filter by date range
- Filter by department
- Filter by author
- Combined filter application

## **5. User Interactions**

### **Comments**

- Add comments to notices
- View comment threads
- Delete own comments
- Display commenter profile information

### **Reactions**

- Like notices with heart reactions
- View reaction count
- Toggle reactions on/off

### **Saved Notices**

- Save notices for later viewing
- Access saved notices from dashboard
- Remove from saved list

## **6. Profile Management**

### **Profile Viewing**

- Public profile pages
- Display user information:
  - Display name and profile picture
  - Academic information (level, program, college, department)
  - Published notices count
  - Comments count

### **Profile Editing**

- Update display name
- Upload profile picture
- Modify academic information
- Change matric number
- Toggle read receipt visibility

### **Account Deletion**

- Permanent account deletion
- Cascade deletion of all user data
- Confirmation dialog for safety

## 7. Dashboard

### User Dashboard

- Profile management
- Comment history
- Saved notices
- Read receipt settings

### Rep Dashboard

- All user features plus:
- My Notices management
- Notice request handling (pending/processed)
- Analytics and statistics
- View count tracking

## 8. Admin Panel

### Notice Management

- Create, edit, delete all notices
- Mark notices as important/featured
- View all notices with full control

### Request Oversight

- View all notice requests
- Override rep decisions
- Direct notice publishing

## 9. Analytics

### Notice Statistics

- View count per notice
- Creation date tracking
- Status indicators (important/featured)
- Performance metrics dashboard

---

## User Roles & Permissions

### Regular User (user\_type: “user”)

**Can:** - View all notices - Search and filter notices - Comment on notices - React to notices - Save notices - Request notices through reps - View other user profiles - Edit own profile - Delete own comments

**Cannot:** - Create notices directly - Edit or delete notices - Manage notice requests - Mark notices as important/featured - Access admin panel

#### **Representative (Rep) (user\_type: “rep”)**

**Can (all user permissions plus):** - Create notices - Edit own notices - Delete own notices - View and manage notice requests - Accept/decline notice requests - View analytics for own notices - Upload media to notices - Add tags to notices

**Cannot:** - Mark notices as important (admin only) - Mark notices as featured (admin only) - Edit other users’ notices - Access admin panel

#### **Administrator**

**Can (all rep permissions plus):** - Create, edit, delete all notices - Mark any notice as important - Mark any notice as featured - Access admin panel - Override rep decisions on requests - View all notice requests - Full system oversight

---

## **API Integration**

### **Supabase Client Configuration**

#### **Browser Client (Client Components)**

```
// @/lib/supabase/client.ts
import { createBrowserClient } from "@supabase/ssr"

export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  )
}
```

#### **Server Client (Server Components)**

```
// @/lib/supabase/server.ts
import { createServerClient } from "@supabase/ssr"
import { cookies } from "next/headers"

export function createClient() {
  const cookieStore = cookies()
  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {

```

```

        cookies: {
          get(name: string) {
            return cookieStore.get(name)?.value
          },
        },
      },
    )
  }
}

```

## Key Database Operations

### Authentication

```

// Sign In
const { error } = await supabase.auth.signInWithPassword({
  email,
  password
})

```

```

// Sign Out
await supabase.auth.signOut()

```

```

// Get User
const { data: { user } } = await supabase.auth.getUser()

```

### Notice Operations

```

// Create Notice
const { data: newNotice } = await supabase
  .from("notices")
  .insert({
    title,
    description,
    author_id: user.id,
    is_important: false,
    view_count: 0
  })
  .select()
  .single()

// Fetch Notices
const { data: notices } = await supabase
  .from("notices")
  .select("*")
  .eq("is_important", true)
  .order("created_at", { ascending: false })
  .limit(20)

```

```

// Update Notice View Count
await supabase
  .from("notices")
  .update({ view_count: currentView + 1 })
  .eq("id", noticeId)

```

## Media Uploads

```

// Upload to Supabase Storage
const { data, error } = await supabase.storage
  .from("bellsnotice")
  .upload(`images/${fileName}`, file)

// Get Public URL
const { data: { publicUrl } } = supabase.storage
  .from("bellsnotice")
  .getPublicUrl(`images/${fileName}`)

```

## Comments & Reactions

```

// Add Comment
await supabase
  .from("comments")
  .insert({
    content,
    user_id: user.id,
    notice_id: noticeId
  })

// Add Reaction
await supabase
  .from("reactions")
  .insert({
    user_id: user.id,
    notice_id: noticeId
  })

// Check if user reacted
const { data } = await supabase
  .from("reactions")
  .select("*")
  .eq("user_id", user.id)
  .eq("notice_id", noticeId)
  .single()

```

## Deployment & Configuration

### Environment Variables

Required environment variables:

```
NEXT_PUBLIC_SUPABASE_URL=your-supabase-project-url  
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-supabase-anon-key
```

### Deployment Platforms

#### Vercel (Recommended)

1. Connect GitHub repository to Vercel
2. Configure environment variables in Vercel dashboard
3. Deploy with automatic CI/CD
4. Custom domain configuration available

### Manual Deployment

```
# Install dependencies
```

```
npm install
```

```
# Build for production
```

```
npm run build
```

```
# Start production server
```

```
npm start
```

### Storage Configuration

#### Supabase Storage Buckets

- **bellsnote:** Main storage bucket for all media
  - **profiles/:** User profile images
  - **images/:** Notice images
  - **videos/:** Notice videos
  - **files/:** Notice documents

**Public Access Rules** Ensure storage policies allow: - Public read access for all files - Authenticated write access for users - Admin write access for all operations

### PWA Configuration

The application is configured as a Progressive Web App: - Service Worker registration for offline support - Web App Manifest for installability - Custom app icons - Theme color configuration

## Development Workflow

### Getting Started

#### Prerequisites

- Node.js 18+ installed
- Supabase account and project
- Git for version control

#### Installation

```
# Clone the repository
git clone https://github.com/Greatness0123/bellsnotice.git

# Navigate to project directory
cd bellsnotice

# Install dependencies
npm install

# Set up environment variables
cp .env.example .env
# Edit .env with your Supabase credentials

# Run development server
npm run dev
```

#### Development Commands

```
npm run dev      # Start development server
npm run build    # Build for production
npm start        # Start production server
npm run lint     # Run ESLint
```

### Code Style & Best Practices

#### TypeScript

- Use TypeScript for all new files
- Define interfaces for data structures
- Avoid **any** type when possible
- Use proper typing for Supabase queries

#### Component Organization

- Keep components focused and single-purpose
- Extract reusable logic into custom hooks
- Use Server Components by default

- Mark components with “use client” only when necessary

### **State Management**

- Use React hooks (useState, useEffect) for local state
- Use Supabase real-time subscriptions for live updates
- Minimize prop drilling with context where appropriate

### **Error Handling**

- Implement try-catch blocks for async operations
- Display user-friendly error messages
- Log errors for debugging
- Handle edge cases gracefully

### **Testing Considerations**

#### **Manual Testing Checklist**

- ☐ User registration and login
- ☐ Notice creation, editing, deletion
- ☐ Media upload functionality
- ☐ Search and filter operations
- ☐ Comment and reaction features
- ☐ Notice request workflow
- ☐ Admin panel functionality
- ☐ Mobile responsiveness
- ☐ Dark mode compatibility
- ☐ PWA installation

### **Performance Optimization**

- Image optimization with Next.js Image component
- Lazy loading for large content lists
- Code splitting with dynamic imports
- Caching strategies for frequently accessed data
- Database query optimization

### **Security Best Practices**

#### **Authentication**

- Never expose service role keys on client-side
- Use Supabase Row Level Security (RLS) policies
- Implement proper session management
- Secure admin authentication flow

## Data Validation

- Validate all user inputs on both client and server
- Use Zod schemas for form validation
- Sanitize user-generated content
- Implement rate limiting for API calls

## Storage Security

- Implement Supabase storage policies
  - Validate file types and sizes
  - Use virus scanning for uploads
  - Set appropriate file permissions
- 

## Future Enhancements

### Planned Features

1. **Email Notifications:** Notify users of new notices and responses
2. **Push Notifications:** Real-time alerts for important notices
3. **Advanced Analytics:** Enhanced analytics with charts and graphs
4. **Notice Templates:** Pre-defined templates for common notice types
5. **Batch Operations:** Bulk actions for notice management
6. **Export Functionality:** Export notices to PDF/Word
7. **Integration:** Integration with university systems
8. **Mobile App:** Native mobile application

### Performance Improvements

1. **Caching Layer:** Implement Redis caching for frequently accessed data
2. **CDN Integration:** Content delivery network for media files
3. **Database Optimization:** Index optimization and query tuning
4. **Lazy Loading:** Improved lazy loading strategies
5. **Bundle Size Reduction:** Code splitting and tree shaking

### User Experience Enhancements

1. **Onboarding Flow:** Guided tour for new users
  2. **Accessibility Improvements:** Enhanced WCAG compliance
  3. **Dark Mode Toggle:** Improved dark mode implementation
  4. **Customizable Themes:** User-selectable color schemes
  5. **Keyboard Shortcuts:** Power user keyboard navigation
  6. **Offline Mode:** Enhanced offline capabilities
-

## Support & Contributing

### Getting Help

- Review this documentation thoroughly
- Check Supabase documentation for backend issues
- Consult Next.js documentation for framework questions
- Review code comments for implementation details

### Contributing Guidelines

1. Fork the repository
2. Create a feature branch
3. Make your changes with proper documentation
4. Test thoroughly before submitting
5. Submit a pull request with clear description

### Issue Reporting

When reporting issues, include: - Clear description of the problem - Steps to reproduce - Expected vs actual behavior - Screenshots if applicable - Browser and environment details

---

### License

This project is developed for Bells University of Technology. All rights reserved.

---

*Last Updated: January 2025 Version: 0.1.0 Author: Greatness*