SkinX Blog - Enhanced Version

A modern blog platform built with Next.js and Express.js, featuring a clean design and robust functionality.

Features

- User authentication (login/register)
- Create, read, update, delete blog posts
- Advanced post filtering and search
- Tag-based organization with popular tags
- Responsive, modern UI with transitions
- Toast notifications for feedback
- Comprehensive error handling
- Smart pagination
- Client-side state management

Tech Stack

Backend

- Node.js with Express.js
- TypeScript
- Prisma ORM with PostgreSQL
- JWT authentication
- Zod validation
- bcrypt for password hashing

Frontend

- Next.js 15 with React 19
- TypeScript
- Tailwind CSS for styling
- Context API for state management
- Toast notifications
- Responsive design system
- Client-side search and filtering

Getting Started

Prerequisites

- Node.js (v18 or higher)
- Docker and Docker Compose (for PostgreSQL)

Database Setup with Docker

1. Navigate to backend directory:

```
cd backend
```

2. Start the PostgreSQL database:

```
docker compose up -d
```

This will start a PostgreSQL container with the following configuration:

- Database name: skinxdb
- Username: skinx
- Password: skinxpass
- Port: 5432 (accessible at localhost:5432)

The database data is persisted using a Docker volume named `pgdata`.

Backend Setup

1. Navigate to backend directory:

```
cd backend
```

2. Install dependencies:

```
npm install
```

3. Create `.env` file:

```
DATABASE_URL="postgresql://skinx:skinxpass@localhost:5432/skinxdb"
JWT_SECRET="your-super-secret-jwt-key-at-least-32-characters-long"
PORT=4000
```

4. Run database migrations:

```
npm run prisma:migrate
```

5. Generate Prisma client:

```
npm run prisma:gen
```

6. Seed the database with sample data:

```
npm run seed
```

7. Start the server:

```
npm run dev
```

Frontend Setup

1. Navigate to frontend directory:

```
cd frontend
```

2. Install dependencies:

```
npm install
```

3. Create `.env.local` file:

```
NEXT_PUBLIC_API_BASE=http://localhost:4000
```

4. Start the development server:

```
npm run dev
```

Project Structure

Backend (`/backend`)

```
src/
├── middleware/
│   ├── errorHandler.ts      # Simple error handling
│   ├── requireAuth.ts       # JWT authentication
│   └── validation.ts        # Request validation
├── routes/
│   ├── auth.ts             # Authentication routes
│   └── posts.ts            # Blog post routes
├── validation/
```

```
│   └── schemas.ts          # Zod validation schemas
└── index.ts                # Main server file
```

Frontend (`/frontend`)

```
src/
├── app/
│   ├── login/              # Enhanced login page
│   ├── posts/
│   │   ├── page.tsx        # Posts listing with filters
│   │   └── [id]/           # Individual post pages
│   ├── layout.tsx          # Root layout with providers
│   └── page.tsx            # Home page
├── components/
│   ├── Navigation.tsx      # Navigation with auth state
│   ├── LoadingSpinner.tsx # Loading indicators
│   ├── Toast.tsx          # Toast notifications
│   └── ClientLayout.tsx    # Client-side layout wrapper
├── contexts/
│   ├── AuthContext.tsx     # Authentication context
│   └── ToastContext.tsx    # Toast notifications context
├── lib/
│   └── api.ts              # Enhanced API utilities
└── utils/
    ├── auth.ts             # Auth helpers
    └── error.ts            # Error handling utilities
```

Available Scripts

Backend

- `npm run dev` - Start development server with nodemon
- `npm run build` - Build TypeScript to JavaScript
- `npm run start` - Start production server
- `npm run prisma:gen` - Generate Prisma client
- `npm run prisma:migrate` - Run database migrations
- `npm run seed` - Seed database with sample data

Frontend

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

API Endpoints

Authentication

- `POST /auth/login` - User login
- `POST /auth/register` - User registration
- `GET /auth/me` - Get current user info

Posts

- `GET /posts` - Get all posts with filtering
  - Query params: q (search), tag, page, pageSize
- `GET /posts/:id` - Get single post

Features in Detail

Post Management

- Create, edit, and delete posts
- Rich text content support
- Tag-based organization
- Post drafts and publishing
- Automatic excerpt generation

Search and Filtering

- Full-text search across posts
- Tag-based filtering
- Popular tags showcase
- Smart pagination
- URL-synchronized filters

User Experience

- Toast notifications for actions
- Loading states and spinners
- Smooth transitions
- Responsive design
- Error feedback
- Client-side navigation

Authentication

- Secure login and registration
- JWT-based authentication
- Protected routes
- Persistent sessions
- User context

License

MIT License