Smart Bookmark App

A full-stack bookmark manager application built with Next.js and Supabase that allows users to save, manage, and sync their bookmarks across devices in real-time.

Live Demo: [Add your Vercel URL here]

Tech Stack

- **Frontend:** Next.js 14 (App Router), React, Tailwind CSS
- **Backend:** Supabase (PostgreSQL, Authentication, Realtime)
- **Authentication:** Google OAuth 2.0
- **Deployment:** Vercel
- **Database:** PostgreSQL with Row Level Security

Features

- ✅ **Google OAuth Authentication** - Secure sign-in without email/password
- ✅ **Add Bookmarks** - Save URLs with custom titles
- ✅ **Private & Secure** - Each user's bookmarks are completely private
- ✅ **Real-time Sync** - Changes appear instantly across all open tabs
- ✅ **Delete Bookmarks** - Remove bookmarks you no longer need
- ✅ **Responsive Design** - Works seamlessly on desktop and mobile
- ✅ **Dark Mode Support** - Automatic theme switching based on system preference

Architecture Overview

- **Next.js App Router:** Modern server and client component architecture
- **Supabase Auth:** Handles OAuth flow and session management

- **PostgreSQL with RLS:** Database-level security ensuring data privacy
- **Realtime Subscriptions:** WebSocket connections for live updates
- **Serverless Deployment:** Hosted on Vercel's edge network

Database Schema

Column	Type	Description
id	uuid	Primary key (auto-generated)
user_id	uuid	Foreign key to auth.users
title	text	Bookmark title
url	text	Bookmark URL
created_at	timestamp	Creation timestamp

Table 1: Bookmarks table structure

Row Level Security Policies:

1. **SELECT Policy:** Users can only view their own bookmarks
2. **INSERT Policy:** Users can only create bookmarks for themselves
3. **DELETE Policy:** Users can only delete their own bookmarks

Installation & Setup

Prerequisites

- Node.js 18+ and npm
- A Supabase account (free tier works)
- A Google Cloud Platform account for OAuth

Local Development

1. Clone the repository:

```
git clone https://github.com/yourusername/smart-bookmark-app.git
cd smart-bookmark-app
```

2. Install dependencies:

```
npm install
```

3. Configure environment variables:

Create a `.env.local` file in the root directory:

```
NEXT_PUBLIC_SUPABASE_URL=your_supabase_project_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
```

4. Set up Supabase:

- Create a new project at supabase.com
- Run the SQL schema (provided in the setup guide)
- Enable Google OAuth provider in Authentication settings
- Configure redirect URLs

5. Set up Google OAuth:

- Go to Google Cloud Console
- Create OAuth 2.0 credentials
- Add authorized redirect URI: `https://your-project-ref.supabase.co/auth/v1/callback`
- Add Client ID and Secret to Supabase

6. Run the development server:

```
npm run dev
```

Open http://localhost:3000 to see the app.

Deployment to Vercel

1. Push your code to GitHub
2. Import the repository in Vercel
3. Add environment variables (NEXT_PUBLIC_SUPABASE_URL, NEXT_PUBLIC_SUPABASE_ANON_KEY)
4. Deploy
5. Update Google OAuth redirect URIs with your Vercel domain
6. Update Supabase redirect URLs to include your production domain

Problems I Faced & How I Solved Them

1. Google OAuth Redirect Loop

Problem: After clicking "Sign in with Google," the app would redirect endlessly or show an "invalid redirect URL" error.

Root Cause: The OAuth redirect URI configured in Google Cloud Console did not match the actual callback URL that Supabase was using. Additionally, the production URL wasn't added to Supabase's allowed redirect URLs.

Solution:

- Added both development (http://localhost:3000/auth/callback) and production URLs to Google OAuth authorized redirect URIs
- Configured Supabase Authentication settings to include both local and production callback URLs
- Created a dedicated callback route handler at `app/auth/callback/route.js` using `createRouteHandlerClient`
- Ensured the `redirectTo` parameter in the OAuth call matched the configured callback URL exactly

Key Learning: OAuth requires exact URL matching - even trailing slashes matter. Always test with both localhost and production URLs.

2. Bookmarks Visible Across Different Users

Problem: When testing with multiple Google accounts, I could see bookmarks created by other users. This was a critical security flaw.

Root Cause: I had created the `bookmarks` table but forgot to enable Row Level Security (RLS) and create proper security policies. Without RLS, all authenticated users could access all rows.

Solution:

1. Enabled Row Level Security on the bookmarks table
2. Created three specific policies:
    - SELECT: `auth.uid() = user_id`
    - INSERT: `auth.uid() = user_id`
    - DELETE: `auth.uid() = user_id`

3. Tested with multiple Google accounts to verify isolation
4. Verified that users could only perform operations on their own bookmarks

Key Learning: RLS is essential for multi-tenant applications. Always enable RLS before deploying to production, and test with multiple user accounts.

3. Real-time Updates Not Working Across Tabs

Problem: When adding or deleting a bookmark in one browser tab, the changes didn't appear in other tabs until I manually refreshed the page.

Root Cause: Multiple issues compounded:

- Forgot to enable Realtime replication for the bookmarks table in Supabase
- Initial subscription didn't filter by `user_id`, causing it to receive events for all users (which RLS then blocked)
- Wasn't properly updating the local state when receiving INSERT and DELETE events

Solution:

1. Ran SQL command: `alter publication supabase_realtime add table bookmarks;`
2. Added user-specific filtering to the subscription:
3. Implemented proper event handlers:
    - INSERT events: Add new bookmark to state array
    - DELETE events: Remove bookmark from state array by id
4. Used `useEffect` with user dependency to re-establish subscription on auth changes
5. Properly cleaned up subscriptions using `removeChannel` in cleanup function

Key Learning: Supabase Realtime requires table-level enablement and proper cleanup. Always filter subscriptions by user to avoid unnecessary network traffic and potential security issues.

4. Authentication State Flickering

Problem: When refreshing the page, the UI briefly showed the "logged out" state before recognizing the user was authenticated, causing a jarring flash of the login screen.

Root Cause: The initial session check was asynchronous, and the component rendered the logged-out UI before the session was retrieved from storage.

Solution:

- Added a loading state to track initial auth check
- Used `supabase.auth.getSession()` on component mount to check for existing session
- Implemented `onAuthStateChange` listener for real-time auth updates
- Rendered a loading indicator during the initial auth check instead of the login screen
- Ensured the auth listener cleanup in `useEffect` return function

Key Learning: Always handle loading states for asynchronous authentication checks to improve user experience.

5. Next.js App Router Client/Server Component Confusion

Problem: Getting errors about using Supabase client in server components and vice versa. The documentation examples seemed contradictory.

Root Cause: Next.js App Router has different Supabase client creation methods for server components, client components, route handlers, and middleware.

Solution:

- Used `createClientComponentClient` in client components (page.js with 'use client')
- Used `createRouteHandlerClient` in API route handlers (auth callback)
- Kept all interactive UI logic in client components

- Used proper cookies import from next/headers for server-side clients

Key Learning: The Supabase Auth Helpers package provides specific client creators for each Next.js context. Using the wrong one causes hydration and authentication errors.

6. Environment Variables Not Loading in Production

Problem: After deploying to Vercel, the app showed "undefined" for Supabase URL and the login button did nothing.

Root Cause: Forgot to add environment variables in the Vercel dashboard. Local development worked because of `.env.local`, but Vercel doesn't have access to that file.

Solution:

1. Added all environment variables in Vercel project settings
2. Used the `NEXT_PUBLIC_` prefix for client-side accessible variables
3. Redeployed after adding the variables
4. Verified variables were properly injected by checking the Network tab

Key Learning: Environment variables must be configured separately for each deployment environment. Next.js only exposes variables with `NEXT_PUBLIC_` prefix to the browser.

7. URL Validation and Edge Cases

Problem: Users could submit invalid URLs (missing protocol, typos) which would break the link rendering and cause console errors.

Root Cause: No client-side validation beyond HTML5's basic `type="url"` validation, which isn't strict enough.

Solution:

- Added URL validation using the browser's URL constructor
- Automatically prepended "https://" if protocol was missing
- Provided user feedback for invalid URLs before submission

- Used try-catch blocks around URL parsing to handle edge cases gracefully

Key Learning: Never trust user input. Validate and sanitize URLs on both client and server side.

Project Structure

```
smart-bookmark-app/
├── app/
│ ├── layout.js # Root layout with metadata
│ ├── page.js # Main bookmark UI (client component)
│ ├── globals.css # Tailwind directives
│ └── auth/
│ └── callback/
│ └── route.js # OAuth callback handler
├── lib/
│ └── supabase.js # Supabase client configuration
├── .env.local # Environment variables (not committed)
├── .gitignore
├── package.json
├── next.config.js
├── tailwind.config.js
└── README.md
```

Key Features Implementation

Authentication Flow

1. User clicks "Sign in with Google"
2. Next.js redirects to Supabase Auth endpoint
3. Supabase redirects to Google OAuth consent screen
4. User authorizes the application
5. Google redirects back to Supabase with authorization code
6. Supabase exchanges code for session token
7. Callback route receives code and creates session
8. User is redirected to main application
9. Session persists across page refreshes via cookies

Real-time Synchronization

The app uses Supabase's Realtime feature with PostgreSQL's logical replication:

- Subscribes to INSERT and DELETE events on the bookmarks table
- Filters events by current user's ID to reduce network traffic
- Updates local React state immediately when events are received
- Maintains WebSocket connection for instant updates
- Automatically reconnects if connection is lost

Security Considerations

- **Row Level Security:** Database-enforced privacy at the data layer
- **OAuth 2.0:** No passwords stored, delegated authentication to Google
- **Server-side Session:** Auth tokens stored in HTTP-only cookies
- **CORS Protection:** Supabase restricts requests to authorized domains
- **Input Validation:** URL and title validation before database insertion

Future Enhancements

- Edit existing bookmarks (title and URL)
- Add categories/tags for organization
- Search and filter bookmarks
- Bookmark folders and nested organization
- Import/export bookmarks (JSON, HTML)
- Browser extension for one-click bookmarking
- Shared bookmark collections with other users
- Bookmark metadata (favicon, description, screenshot)

Performance Optimizations

- Used React's `useEffect` with proper dependencies to minimize re-renders
- Implemented optimistic UI updates for better perceived performance
- Utilized Vercel's Edge Network for fast global delivery
- Supabase connection pooling for efficient database queries
- Indexed `user_id` and `created_at` columns for fast queries

Testing

Manual testing performed:

1. Multi-user isolation (tested with 3 different Google accounts)
2. Real-time sync across 4 browser tabs simultaneously
3. Authentication flow with different browsers (Chrome, Firefox, Safari)
4. Mobile responsiveness on iOS and Android devices
5. Network resilience (tested with intermittent connectivity)
6. Edge cases (special characters in titles, very long URLs)

Lessons Learned

1. **Start with Security:** Enable RLS from day one, not as an afterthought
2. **Test with Real Users:** Multi-account testing revealed issues I wouldn't have found otherwise
3. **Read the Docs Carefully:** Next.js App Router requires different patterns than Pages Router
4. **Environment Configuration:** Keep a checklist of environment variables for each deployment environment
5. **Real-time is Complex:** WebSocket subscriptions need careful setup, filtering, and cleanup
6. **OAuth is Finicky:** Exact URL matching is crucial - save yourself hours of debugging

Contributing

This is a learning project, but suggestions and improvements are welcome! Feel free to:

- Open issues for bugs or feature requests
- Submit pull requests with improvements
- Share your own learning experiences

License

Acknowledgments

- Next.js team for the excellent App Router documentation
- Supabase for providing a fantastic backend-as-a-service platform
- Vercel for seamless deployment experience
- Google for OAuth 2.0 authentication infrastructure

Contact

Built by [Your Name] - GitHub Profile

Live Demo: [Add your deployed URL here]
GitHub Repository: [Add your repo URL here]

Built as a learning project to understand full-stack development with modern web technologies.