SocialLink Documentation

# Overview

SocialLink is a platform that allows users to create a single link that directs to a personalized landing page containing links to their various social media profiles, websites, or other online content. This is especially useful on platforms like Instagram or Twitter, where you can only have one bio link.

# Table of Contents

1. Features  
2. Architecture  
3. User Flow  
4. API Endpoints  
5. Database Schema  
6. Frontend Structure  
7. Backend Structure  
8. Security Considerations  
9. Deployment Strategy  
10. Scalability and Performance

# Features

## User Authentication and Profile Management

Sign up, login, logout, and password reset.  
Profile creation and management (name, bio, profile picture).

## Link Management

Add, edit, and delete links.  
Select from a list of popular social media platforms and websites.  
Customizable titles, URLs, and descriptions.  
Reorder links through a drag-and-drop interface (optional).

## Profile Sharing

Unique URL for each user (e.g., sociallink.com/username).  
Public profiles viewable by anyone with the URL.  
Generate a short link for easy sharing.

## Customizable Themes (Optional)

Users can choose from various themes or customize the appearance of their profile page.

# Architecture

## High-Level Architecture

Frontend: React.js  
State Management: Context API or Redux  
UI Framework: Material-UI or Bootstrap  
Backend: Node.js with Express.js  
Database: MongoDB  
Authentication: JWT (JSON Web Tokens)

# User Flow

## 1. Sign Up/Log In

User visits the website and signs up or logs in.  
Upon successful authentication, the user is redirected to their dashboard.

## 2. Dashboard (Profile Management)

User can edit their profile (name, bio, profile picture).  
Profile information is displayed on the public profile page.

## 3. Link Management

User clicks on an "Add Link" button to create a new link.  
User selects from a list of predefined social media platforms and websites, or adds a custom URL.  
User inputs the title and URL (and optional description) for the link.  
User can edit or delete existing links.  
User can reorder links through a drag-and-drop interface (optional).

## 4. Profile Sharing

User shares their unique URL (e.g., sociallink.com/username).  
Visitors can view the user’s landing page with all their links.

# API Endpoints

## User Authentication

POST /api/users/register - Register a new user  
POST /api/users/login - Authenticate a user and return a token  
POST /api/users/reset-password - Send password reset email  
POST /api/users/reset-password/:token - Reset password

## Profile Management

GET /api/users/profile - Get logged-in user’s profile  
PUT /api/users/profile - Update logged-in user’s profile

## Link Management

GET /api/links - Get all links for the logged-in user  
POST /api/links - Add a new link  
PUT /api/links/:id - Update a link  
DELETE /api/links/:id - Delete a link

## Profile Sharing

GET /api/profile/:username - Get public profile by username

# Database Schema

## User Schema

Fields:  
Name: String  
Email: String (unique)  
Password: String (hashed)  
Profile Picture: String (URL)  
Bio: String  
CreatedAt: Date  
UpdatedAt: Date

## Link Schema

Fields:  
UserId: ObjectId (reference to User)  
Title: String  
URL: String  
Platform: String (optional - predefined platforms like Facebook, Twitter, etc.)  
Description: String  
CreatedAt: Date  
UpdatedAt: Date

# Frontend Structure

## Components

Auth Components: Login, Register, Reset Password  
Profile Components: EditProfile, ViewProfile  
Link Components: AddLink, EditLink, LinkList, ReorderLinks (optional)  
Layout Components: Header, Footer  
Pages: Home, NotFound

## State Management

Use Context API or Redux for managing authentication state and user data

# Backend Structure

## API Endpoints

User Authentication: Register, Login, Logout, Password Reset  
Profile Management: Get Profile, Update Profile  
Link Management: Add Link, Get Links, Update Link, Delete Link  
Profile Sharing: Get Public Profile by Username

## Middleware

Authentication middleware to protect routes and ensure only logged-in users can access certain endpoints

# Security Considerations

1. Authentication:  
Use JWT tokens for user authentication and authorization  
Store tokens securely in HTTP-only cookies or local storage

2. Authorization:  
Protect API endpoints to ensure only authenticated users can access/modify their data

3. Data Validation:  
Validate user input on both frontend and backend to prevent malicious data entry

4. Rate Limiting:  
Implement rate limiting to prevent abuse of API endpoints

# Deployment Strategy

1. Frontend:  
Deploy on a static site hosting service (e.g., Netlify, Vercel)

2. Backend:  
Deploy on a cloud platform (e.g., Heroku, AWS, DigitalOcean)  
Use a managed database service like MongoDB Atlas

3. Continuous Integration/Continuous Deployment (CI/CD):  
Set up CI/CD pipelines to automate testing and deployment

# Scalability and Performance

1. Scalability:  
Use load balancers and horizontal scaling for the backend  
Optimize database queries and use indexing

2. Performance:  
Implement caching strategies (e.g., Redis) for frequently accessed data  
Use a content delivery network (CDN) for serving static assets