



Reddit Clone - Technical Documentation

A Full-Stack Reddit Clone Built with Next.js 16 & Supabase



[GitHub Repository](#)



[Live Demo](#)



Table of Contents

- [Project Overview](#)
- [Technology Stack](#)
- [Application Architecture](#)
- [Authentication System](#)
- [Database Integration](#)
- [Core Features & Implementation](#)
- [File Structure](#)
- [Important Code Sections](#)

Project Overview

This is a feature-rich Reddit clone application built using modern web technologies. It implements core Reddit functionalities including user authentication, post creation, commenting, image uploads, and real-time search capabilities.

Project Type: Full-Stack Web Application

Framework: Next.js 16 with App Router

Backend: Supabase (PostgreSQL + Authentication + Storage)

Deployment: Vercel

▶ Key Features

- **User Authentication:** Sign up, login, and logout with Supabase Auth
- **Post Management:** Create, edit, and delete posts with optional images
- **Comment System:** Add and delete comments on posts
- **Image Uploads:** Upload images to Supabase Storage
- **Search Functionality:** Real-time search for posts
- **Route Protection:** Middleware-based authentication
- **Responsive Design:** Tailwind CSS v4 for styling



Technology Stack

Next.js 16

React framework with App Router, Server Components, and Server Actions

React 19

Latest React with concurrent features

TypeScript

Type-safe development

Supabase

PostgreSQL database, authentication, and storage

Tailwind CSS v4

Utility-first CSS framework

React Hook Form

Form state management

Zod

Schema validation

TanStack Query

Data fetching and caching

Sonner

Toast notifications

► Package.json Dependencies

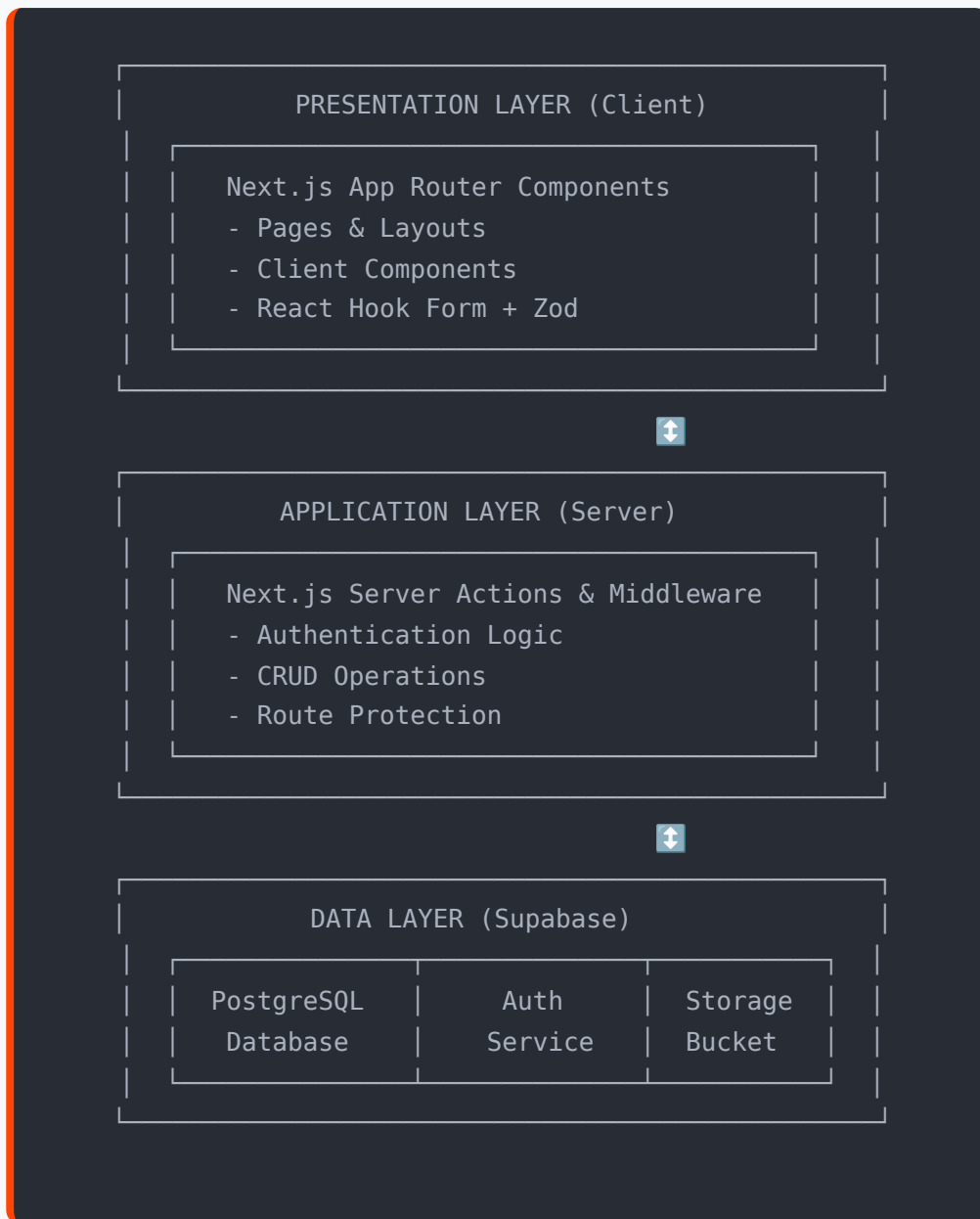
```
{  
  "dependencies": {  
    "@hookform/resolvers": "^5.2.2",
```

```
"@supabase/ssr": "^0.7.0",
"@supabase/supabase-js": "^2.57.4",
"@tanstack/react-query": "^5.89.0",
"lucide-react": "^0.544.0",
"next": "^16.0.10",
"react": "^19.2.3",
"react-dom": "^19.2.3",
"react-hook-form": "^7.63.0",
"sonner": "^2.0.7",
"uuid": "^13.0.0",
"zod": "^4.1.11"
}
}
```



Application Architecture

► Three-Tier Architecture



▶ Data Flow

1. **User Action:** User interacts with UI (form submission, button click)
2. **Client Component:** React Hook Form captures and validates data
3. **Server Action:** Next.js Server Action receives data
4. **Validation:** Zod schema validates input
5. **Authentication:** Supabase verifies user session
6. **Database Operation:** CRUD operation executed on Supabase
7. **Revalidation:** Next.js revalidates affected routes
8. **UI Update:** Client receives updated data and re-renders



Authentication System

► Overview

The application uses Supabase Authentication with custom middleware for route protection. It implements both server-side and client-side authentication flows.

► 1. Middleware for Route Protection

src/middleware.ts

```
import { createServerClient } from "@supabase/ssr"
import { NextRequest, NextResponse } from "next/server"

export const middleware = async (request: NextRequest) => {
  let supabaseResponse = NextResponse.next({request})

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_PUBLISHABLE_KEY!,
    {
      cookies: {
        getAll(){
          return request.cookies.getAll()
        },
        setAll(cookiesToSet){
          cookiesToSet.forEach(({name,value})=>
            request.cookies.set(name,value))
          cookiesToSet.forEach(({name,value,options})=>
            supabaseResponse.cookies.set(name,value,options))
        }
      }
    }
  )
}
```

```

    )

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

    const protectedRoutes = [/^\/create\/?$/]

    if(!user && protectedRoutes.some(route =>
      route.test(request.nextUrl.pathname))) {
      const newUrl = request.nextUrl.clone()
      newUrl.pathname = "/auth/login"
      return NextResponse.redirect(newUrl)
    }

    return supabaseResponse
  }
}

```

How Middleware Works

- Runs on every request before page rendering
- Creates Supabase client with cookie handling
- Checks if user is authenticated
- Protects routes like `/create`
- Redirects unauthenticated users to login

► 2. Supabase Client Configuration

Server Client

`utils/supabase/server-client.ts`

```

import { createServerClient } from "@supabase/ssr";
import { cookies } from "next/headers";

```

```

export const createClient = async () => {
  const cookieStore = await cookies();

  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_PUBLISHABLE_KEY!,
    {
      cookies: {
        getAll() {
          return cookieStore.getAll()
        },
        setAll(cookiesToSet) {
          try {
            cookiesToSet.forEach(({name,value,options}) => {
              cookieStore.set(name,value,options)
            })
          } catch { }
        }
      }
    }
  )
}

```

Browser Client

[utils/supabase/browserclient.ts](#)

```

import { createBrowserClient } from "@supabase/ssr";

export const createClient = () => {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_PUBLISHABLE_KEY!
  )
}

```

► 3. Authentication Actions

Sign Up Action

src/actions/sign-up.ts

```
'use server'

export const SignUp = async (userdata: z.infer<typeof signUpSchema>) => {
  const supabase = await createClient()
  const {data:{user},error} = await supabase.auth.signUp({
    email: userdata.email,
    password: userdata.password,
  })

  if(user && user.email){
    // Create user profile in database
    await supabase.from('users')
      .insert([
        {
          id: user.id,
          email: user.email,
          username: userdata.username
        }
      ])
  }

  if(error) throw error
  redirect("/")
}
```

Log In Action

src/actions/log-in.ts

```
'use server'

export const LogIn = async(userData: z.infer<typeof logInSchema>) => {
  const parsedData = logInSchema.parse(userData)
  const supabase = await createClient()
```

```
const {data:{user},error} = await supabase.auth
    .signInWithPassword(userData)

if(error) throw error
redirect("/")
}
```

► 4. Validation Schemas

src/actions/schema.ts

```
import z from "zod";

export const logInSchema = z.object({
  email: z.email(),
  password: z.string()
    .min(6, "Your password must be a minimum of 6 characters")
})

export const signUpSchema = z.object({
  email: z.email(),
  username: z.string()
    .regex(/^[a-zA-Z]+$/, "Username should contain only letters")
  password: z.string()
    .min(6, "Password must be minimum of 6 characters")
})
```



Database Integration

► Database Schema

The application uses three main tables in Supabase:

Table	Purpose	Key Fields
users	User profiles	id (UUID), email, username
posts	Post content	id, title, content, slug, user_id, image, created_at
comments	Post comments	id, comment_section, commentor_name, user_id, slug

► Database Queries

`utils/supabase/queries.ts`

Get Home Posts

```
export const getHomePosts = async (supabase: ReturnType<typ
  return await supabase.from('posts')
    .select('id,title,slug,users("username"),image')
    .order('created_at',{ascending:false})
}
```

Get Single Post with Comments

```
export const getSinglePost = async (slug: string) => {
  const supabase = createClient()
  return await supabase.from('posts')
    .select(`
      id,title,content,slug,user_id,
      users("username"),
      image,
      comments("comment_section","commentor_name","sl
    `)
    .eq('slug',slug)
    .single()
}
```

Search Posts

```
export const getSearchPost = async (searchTerm: string) =>
  const supabase = createClient()
  return await supabase.from('posts')
    .select('title,slug')
    .ilike('title',`${searchTerm}%`)
}
```

💡 Key Features:

- Type-safe queries with TypeScript
- Automatic joins with foreign key relationships
- Real-time search with `ilike`
- Sorted by creation date

✨ Core Features & Implementation

► 1. Post Management

Create Post Action

src/actions/create-post.ts

```
'use server'

export const CreatePost = async (userdata: z.infer<typeof p
  const parsedData = postSchema.parse(userdata)
  const slug = slugify(parsedData.title)

  // Handle image upload
  const imageFile = userdata.image?.get("image")
  if(!(imageFile instanceof File) && imageFile !== null)
    throw new Error("malformed image File")
  }

  const publicImageUrl = imageFile instanceof File
    ? await uploadImage(imageFile)
    : null

  // Verify authentication
  const supabase = await createClient()
  const {data:{user}} = await supabase.auth.getUser();
  if(!user) throw new Error("Not Authorized")

  // Insert post
  await supabase.from('posts')
    .insert([{
      user_id: userId,
      slug: slug,
```

```

        ...parsedData,
        image: publicImageUrl
      })
      .throwOnError()

    revalidatePath("/")
    redirect(`/${slug}`)
  }

```

Slug Generation

utils/slugify.ts

```

export const slugify = (text: string) => {
  return text
    .toLowerCase()
    .trim()
    .replace(/[^\w\s-]/g, '')
    .replace(/[\s_]/g, '-')
    .replace(/-+/g, '-')
}

```



Post Creation Flow

1. User submits form with title, content, and optional image
2. Data validated with Zod schema
3. Title converted to URL-friendly slug
4. Image uploaded to Supabase Storage (if provided)
5. Post inserted into database
6. Homepage revalidated
7. User redirected to new post

► 2. Image Upload System

utils/supabase/upload-image.ts

```
import { v4 as uuid } from 'uuid';

export const uploadImage = async (image: File) => {
  const supabase = await createClient();

  // Generate unique filename
  const imageName: string[] = image.name.split('.')
  const path: string = `${imageName[0]}-${uuid()}.${imageName[1]}`

  // Upload to Supabase Storage
  const {data,error} = await supabase.storage
    .from('images')
    .upload(path, image)

  if(error) throw error

  // Get public URL
  const {data:{publicUrl}} = await supabase.storage
    .from('images')
    .getPublicUrl(data.path);

  return publicUrl
}
```

► 3. Comment System

src/actions/create-comment.ts

```
'use server'

export const CreateComment = async ({
```

```

        comment_section,
        slug,
        id
    }: {
        comment_section: string,
        id: number,
        slug: string
    }) => {
        try {
            const parsedData = commentSchema.parse({ comment_section: comment_section, id: id, slug: slug })
            const supabase = await createClient()
            const { data: { user } } = await supabase.auth.getUser()

            if (!user) throw new Error("Not Authorized")

            // Fetch user profile
            const { data: profile } = await supabase
                .from("users")
                .select("username,id")
                .eq("id", user.id)
                .single()

            // Insert comment
            await supabase
                .from("comments")
                .insert([
                    {
                        slug,
                        comment_section: parsedData.comment_section,
                        commentor_name: profile?.username || "Anonymous",
                        user_id: profile?.id || "Anonymous"
                    }
                ])
                .throwOnError()

            revalidatePath(`/${slug}`)
        } catch (err) {
            console.error("ERROR IN CreateComment:", err)
            throw err
        }
    }
}

```

▶ 4. Search Functionality

Real-time search implemented with TanStack Query for efficient data fetching and caching.

```
src/components/Search/index.tsx
```

- Debounced search input
- Case-insensitive partial matching
- Instant results dropdown
- Navigation to post on selection

File Structure

```
Reddit_Clone_Supabase/
|
├─ src/
|   │
|   └─ actions/                                # Server Actions
|       │
|       ├── create-comment.ts
|       ├── create-post.ts
|       ├── delete-comments.ts
|       ├── delete-post.ts
|       ├── edit-posts.ts
|       ├── log-in.ts
|       ├── log-out.ts
|       ├── sign-up.ts
|       └─ schema.ts
|
|   └─ app/                                    # Next.js App Router
|       │
|       ├── (main)/
|       │   │
|       │   ├── [slug]/
|       │   │   │
|       │   │   ├── edit/
|       │   │   │   │
|       │   │   │   ├── EditForm/
|       │   │   │   └─ page.tsx
|       │   │   ├── DeleteButton/
|       │   │   ├── EditButton/
|       │   │   └─ page.tsx
|       │   ├── create/
|       │   │   └─ page.tsx
|       │   ├── layout.tsx
|       │   └─ page.tsx
|       │
|       └─ auth/
|           │
|           ├── login/
|           │   │
|           │   ├── LoginForm/
|           │   └─ page.tsx
|           └─ signup/
```

```

├── |   |   |   |   └─ SignupForm/
├── |   |   |   |   └─ page.tsx
├── |   |   |   └─ layout.tsx
├── |   |   |
├── |   |   └─ layout.tsx
├── |   |   └─ globals.css
├── |   |
├── |   └─ components/                # React Components
├── |       |   └─ AccountLinks/
├── |       |   └─ Header/
├── |       |   └─ LogOut/
├── |       |   └─ Logo/
├── |       |   └─ Search/
├── |       |   └─ comments/
├── |       |       |   └─ CommentSection/
├── |       |       |   └─ index.tsx
├── |       |       └─ Home/
├── |   └─ middleware.ts              # Route Protection
├── |
├── └─ utils/
├──     |   └─ supabase/
├──     |       |   └─ browserclient.ts
├──     |       |   └─ server-client.ts
├──     |       |   └─ queries.ts
├──     |       |   └─ upload-image.ts
├──     |       |   └─ datatypes.types.ts
├──     |       └─ slugify.ts
├── |
├── └─ providers/
├──     |   └─ query-client-provider.tsx
├── |
├── └─ public/                        # Static Assets
├── └─ package.json
├── └─ tailwind.config.ts
├── └─ tsconfig.json
├── └─ next.config.ts

```

Important Code Sections

► 1. App Layout with Query Provider

src/app/layout.tsx

```
import { QueryClientProvider } from "../../providers/query-  
import { Toaster } from "sonner";  
  
export default function RootLayout({  
  children,  
}: Readonly<{ children: React.ReactNode }>) {  
  return (  
    <html lang="en">  
      <body className="p-4 pt-6 antialiased font-sans">  
        <Toaster richColors/>  
        <QueryClientProvider>  
          {children}  
        </QueryClientProvider>  
      </body>  
    </html>  
  );  
}
```

► 2. Home Page Server Component

src/app/(main)/page.tsx

```
import { getHomePosts } from "../../utils/supabase/query-  
  
export const revalidate = 60; // Revalidate every 60 seconds
```

```

export default async function Home() {
  const supabase = await createClient();
  const {data, error} = await getHomePosts(supabase)

  return (
    <div>
      {data && <HomePosts posts={data} />}
    </div>
  );
}

```

► 3. Single Post Page with Comments

src/app/(main)/[slug]/page.tsx

```

const singlePost = async ({params}:{params:{slug:string}})
  const {slug} = await params
  const {data, error} = await getSinglePost(slug)
  const supabase = await createClient();
  const {data:{user}} = await supabase.auth.getUser();

  const isAuthor = user?.id === data?.user_id

  return (
    <div className="container-narrow">
      {/* Post Header */}
      <div className="card">
        <h1>{data.title}</h1>
        <p>{data.users?.username}</p>
      </div>

      {/* Post Image */}
      {data?.image && <img src={data.image} alt={data
        title}>}

      {/* Post Content */}
      <div className="card">{data.content}</div>
    </div>
  );
}

```

```

        {/* Author Actions */}
        {isAuthor && (
            <>
                <EditButton slug={slug}/>
                <DeleteButton postId={data.id}/>
            </>
        )}

        {/* Comments */}
        {user && <CommentSection slug={slug} id={data.id}
        {data.comments} && (
            <Comments
                comments={data.comments}
                currentUserId={user?.id}
                postAuthorId={data.user_id}
                slug={slug}
            />
        )}
    </div>
)
}

```

► 4. Header Component

src/components/Header/index.tsx

```

const Header = () => {
    return(
        <header className="header-nav">
            <div className="container-main py-4">
                <div className="flex items-center justify-between">
                    <Logo />
                    <div className="flex-1 max-w-2xl">
                        <SearchInput />
                    </div>
                    <AccountLinks />
                </div>
            </div>
        </header>
    )
}

```

```
        </div>
      </header>
    )
  }
```

► 5. Validation Schemas

src/actions/schema.ts

```
export const postSchema = z.object({
  title: z.string().min(3, "Title must have 3 characters"),
  content: z.string().optional(),
  image: z.instanceof(FormData).optional(),
})

export const commentSchema = z.object({
  comment_section: z.string().min(1, "Comment cannot be empty"),
})
```



Deployment & Environment

► Environment Variables Required

```
NEXT_PUBLIC_SUPABASE_URL=your_supabase_project_url  
NEXT_PUBLIC_SUPABASE_PUBLISHABLE_KEY=your_supabase_anon_key
```

► Build Configuration

package.json

```
{  
  "scripts": {  
    "dev": "next dev --turbopack",  
    "build": "next build --turbopack",  
    "start": "next start"  
  }  
}
```

Deployment Platform: Vercel

Live URL: <https://puzzle-game-vibe-coding.vercel.app/login>

Build Tool: Turbopack (Next.js 16 default)

► Key Deployment Features

- **Automatic Deployments:** Push to main triggers deployment
- **Edge Runtime:** Middleware runs on edge for low latency
- **ISR:** Pages revalidate every 60 seconds

- **Environment Variables:** Configured in Vercel dashboard



Security Considerations

► Implemented Security Measures

1. Authentication & Authorization

- JWT-based authentication via Supabase
- Middleware-level route protection
- Server-side user verification for all mutations
- Author-only edit/delete permissions

2. Data Validation

- Zod schemas for all user inputs
- Type-safe database queries with TypeScript
- File type validation for image uploads
- SQL injection prevention via Supabase client

3. Secure File Uploads

- UUID-based unique filenames
- Stored in Supabase Storage bucket
- File type checking before upload

- Public URL generation after successful upload

▶ **Best Practices Followed**

- Environment variables for sensitive data
- Server-side rendering for sensitive operations
- HTTPS enforced on Vercel
- Row-level security policies in Supabase
- Error handling without exposing stack traces



Performance Optimizations

▶ Next.js 16 Features Utilized

- **Turbopack:** Faster builds and hot module replacement
- **React Server Components:** Reduced client-side JavaScript
- **Server Actions:** Direct server mutations without API routes
- **Incremental Static Regeneration:** 60-second revalidation
- **Automatic Image Optimization:** Next.js Image component

▶ Database Optimizations

- Efficient queries with selective field selection
- Database indexing on frequently queried fields (slug, created_at)
- Single query for posts with user data (joins)
- TanStack Query for client-side caching

▶ Frontend Optimizations

- Tailwind CSS for minimal CSS bundle size
- Code splitting via Next.js App Router
- Debounced search input
- Toast notifications via lightweight Sonner library

Key Takeaways

► Architecture Highlights

Modern Stack

Using latest versions of Next.js 16, React 19, and Tailwind CSS v4

Type Safety

Full TypeScript coverage with Zod validation







Scalable Backend

Supabase provides production-ready PostgreSQL, auth, and storage

DX Optimized

Turbopack for fast builds, Server Actions for simple mutations

► Code Quality Features

-  Consistent error handling with try-catch
-  Proper TypeScript types throughout
-  Reusable server actions
-  Clean separation of concerns (actions, queries, components)
-  Path revalidation after mutations
-  Form validation with React Hook Form + Zod

► Production Readiness

 Ready for Production

- Deployed on Vercel with HTTPS
- Environment variables configured
- Database hosted on Supabase cloud
- Authentication fully implemented
- Error boundaries and loading states
- Mobile-responsive design



Additional Resources

► Official Documentation

- Next.js 16: <https://nextjs.org/docs>
- Supabase: <https://supabase.com/docs>
- Tailwind CSS v4: <https://tailwindcss.com/docs>
- React Hook Form: <https://react-hook-form.com>
- Zod: <https://zod.dev>

► Project Links

- GitHub Repository:
https://github.com/VaishaliGovindaraj/Reddit_Clone_Supabase
- Live Demo: <https://puzzle-game-vibe-coding.vercel.app/login>



Document Generated: December 2025



Project Owner: VaishaliGovindaraj



Version: 0.1.0

End of Documentation

Reddit Clone - Technical Documentation

Generated with ❤️ for developers