# 📚 Complete Code Documentation

## Directory: NewLinkInsightTracker-main (C:-main-main)

client/src/components/ui/slider.tsx

import \* as React from "react"  
import \* as SliderPrimitive from "@radix-ui/react-slider"  
  
import { cn } from "@/lib/utils"  
  
const Slider = React.forwardRef<  
 React.ElementRef<typeof SliderPrimitive.Root>,  
 React.ComponentPropsWithoutRef<typeof SliderPrimitive.Root>  
>(({ className, ...props }, ref) => (  
 <SliderPrimitive.Root  
 ref={ref}  
 className={cn(  
 "relative flex w-full touch-none select-none items-center",  
 className  
 )}  
 {...props}  
 >  
 <SliderPrimitive.Track className="relative h-2 w-full grow overflow-hidden rounded-full bg-secondary">  
 <SliderPrimitive.Range className="absolute h-full bg-primary" />  
 </SliderPrimitive.Track>  
 <SliderPrimitive.Thumb className="block h-5 w-5 rounded-full border-2 border-primary bg-background ring-offset-background transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50" />  
 </SliderPrimitive.Root>  
))  
Slider.displayName = SliderPrimitive.Root.displayName  
  
export { Slider }

client/src/components/ui/switch.tsx

import \* as React from "react"  
import \* as SwitchPrimitives from "@radix-ui/react-switch"  
  
import { cn } from "@/lib/utils"  
  
const Switch = React.forwardRef<  
 React.ElementRef<typeof SwitchPrimitives.Root>,  
 React.ComponentPropsWithoutRef<typeof SwitchPrimitives.Root>  
>(({ className, ...props }, ref) => (  
 <SwitchPrimitives.Root  
 className={cn(  
 "peer inline-flex h-6 w-11 shrink-0 cursor-pointer items-center rounded-full border-2 border-transparent transition-colors focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 focus-visible:ring-offset-background disabled:cursor-not-allowed disabled:opacity-50 data-[state=checked]:bg-primary data-[state=unchecked]:bg-input",  
 className  
 )}  
 {...props}  
 ref={ref}  
 >  
 <SwitchPrimitives.Thumb  
 className={cn(  
 "pointer-events-none block h-5 w-5 rounded-full bg-background shadow-lg ring-0 transition-transform data-[state=checked]:translate-x-5 data-[state=unchecked]:translate-x-0"  
 )}  
 />  
 </SwitchPrimitives.Root>  
))  
Switch.displayName = SwitchPrimitives.Root.displayName  
  
export { Switch }

client/src/components/ui/table.tsx

import \* as React from "react"  
  
import { cn } from "@/lib/utils"  
  
const Table = React.forwardRef<  
 HTMLTableElement,  
 React.HTMLAttributes<HTMLTableElement>  
>(({ className, ...props }, ref) => (  
 <div className="relative w-full overflow-auto">  
 <table  
 ref={ref}  
 className={cn("w-full caption-bottom text-sm", className)}  
 {...props}  
 />  
 </div>  
))  
Table.displayName = "Table"  
  
const TableHeader = React.forwardRef<  
 HTMLTableSectionElement,  
 React.HTMLAttributes<HTMLTableSectionElement>  
>(({ className, ...props }, ref) => (  
 <thead ref={ref} className={cn("[&\_tr]:border-b", className)} {...props} />  
))  
TableHeader.displayName = "TableHeader"  
  
const TableBody = React.forwardRef<  
 HTMLTableSectionElement,  
 React.HTMLAttributes<HTMLTableSectionElement>  
>(({ className, ...props }, ref) => (  
 <tbody  
 ref={ref}  
 className={cn("[&\_tr:last-child]:border-0", className)}  
 {...props}  
 />  
))  
TableBody.displayName = "TableBody"  
  
const TableFooter = React.forwardRef<  
 HTMLTableSectionElement,  
 React.HTMLAttributes<HTMLTableSectionElement>  
>(({ className, ...props }, ref) => (  
 <tfoot  
 ref={ref}  
 className={cn(  
 "border-t bg-muted/50 font-medium [&>tr]:last:border-b-0",  
 className  
 )}  
 {...props}  
 />  
))  
TableFooter.displayName = "TableFooter"  
  
const TableRow = React.forwardRef<  
 HTMLTableRowElement,  
 React.HTMLAttributes<HTMLTableRowElement>  
>(({ className, ...props }, ref) => (  
 <tr  
 ref={ref}  
 className={cn(  
 "border-b transition-colors hover:bg-muted/50 data-[state=selected]:bg-muted",  
 className  
 )}  
 {...props}  
 />  
))  
TableRow.displayName = "TableRow"  
  
const TableHead = React.forwardRef<  
 HTMLTableCellElement,  
 React.ThHTMLAttributes<HTMLTableCellElement>  
>(({ className, ...props }, ref) => (  
 <th  
 ref={ref}  
 className={cn(  
 "h-12 px-4 text-left align-middle font-medium text-muted-foreground [&:has([role=checkbox])]:pr-0",  
 className  
 )}  
 {...props}  
 />  
))  
TableHead.displayName = "TableHead"  
  
const TableCell = React.forwardRef<  
 HTMLTableCellElement,  
 React.TdHTMLAttributes<HTMLTableCellElement>  
>(({ className, ...props }, ref) => (  
 <td  
 ref={ref}  
 className={cn("p-4 align-middle [&:has([role=checkbox])]:pr-0", className)}  
 {...props}  
 />  
))  
TableCell.displayName = "TableCell"  
  
const TableCaption = React.forwardRef<  
 HTMLTableCaptionElement,  
 React.HTMLAttributes<HTMLTableCaptionElement>  
>(({ className, ...props }, ref) => (  
 <caption  
 ref={ref}  
 className={cn("mt-4 text-sm text-muted-foreground", className)}  
 {...props}  
 />  
))  
TableCaption.displayName = "TableCaption"  
  
export {  
 Table,  
 TableHeader,  
 TableBody,  
 TableFooter,  
 TableHead,  
 TableRow,  
 TableCell,  
 TableCaption,  
}

client/src/components/ui/tabs.tsx

import \* as React from "react"  
import \* as TabsPrimitive from "@radix-ui/react-tabs"  
  
import { cn } from "@/lib/utils"  
  
const Tabs = TabsPrimitive.Root  
  
const TabsList = React.forwardRef<  
 React.ElementRef<typeof TabsPrimitive.List>,  
 React.ComponentPropsWithoutRef<typeof TabsPrimitive.List>  
>(({ className, ...props }, ref) => (  
 <TabsPrimitive.List  
 ref={ref}  
 className={cn(  
 "inline-flex h-10 items-center justify-center rounded-md bg-muted p-1 text-muted-foreground",  
 className  
 )}  
 {...props}  
 />  
))  
TabsList.displayName = TabsPrimitive.List.displayName  
  
const TabsTrigger = React.forwardRef<  
 React.ElementRef<typeof TabsPrimitive.Trigger>,  
 React.ComponentPropsWithoutRef<typeof TabsPrimitive.Trigger>  
>(({ className, ...props }, ref) => (  
 <TabsPrimitive.Trigger  
 ref={ref}  
 className={cn(  
 "inline-flex items-center justify-center whitespace-nowrap rounded-sm px-3 py-1.5 text-sm font-medium ring-offset-background transition-all focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50 data-[state=active]:bg-background data-[state=active]:text-foreground data-[state=active]:shadow-sm",  
 className  
 )}  
 {...props}  
 />  
))  
TabsTrigger.displayName = TabsPrimitive.Trigger.displayName  
  
const TabsContent = React.forwardRef<  
 React.ElementRef<typeof TabsPrimitive.Content>,  
 React.ComponentPropsWithoutRef<typeof TabsPrimitive.Content>  
>(({ className, ...props }, ref) => (  
 <TabsPrimitive.Content  
 ref={ref}  
 className={cn(  
 "mt-2 ring-offset-background focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2",  
 className  
 )}  
 {...props}  
 />  
))  
TabsContent.displayName = TabsPrimitive.Content.displayName  
  
export { Tabs, TabsList, TabsTrigger, TabsContent }

client/src/components/ui/textarea.tsx

import \* as React from "react"  
  
import { cn } from "@/lib/utils"  
  
export interface TextareaProps  
 extends React.TextareaHTMLAttributes<HTMLTextAreaElement> {}  
  
const Textarea = React.forwardRef<HTMLTextAreaElement, TextareaProps>(  
 ({ className, ...props }, ref) => {  
 return (  
 <textarea  
 className={cn(  
 "flex min-h-[80px] w-full rounded-md border border-input bg-background px-3 py-2 text-sm ring-offset-background placeholder:text-muted-foreground focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:cursor-not-allowed disabled:opacity-50",  
 className  
 )}  
 ref={ref}  
 {...props}  
 />  
 )  
 }  
)  
Textarea.displayName = "Textarea"  
  
export { Textarea }

client/src/components/ui/toast.tsx

import \* as React from "react"  
import \* as ToastPrimitives from "@radix-ui/react-toast"  
import { cva, type VariantProps } from "class-variance-authority"  
import { X } from "lucide-react"  
  
import { cn } from "@/lib/utils"  
  
const ToastProvider = ToastPrimitives.Provider  
  
const ToastViewport = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Viewport>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Viewport>  
>(({ className, ...props }, ref) => (  
 <ToastPrimitives.Viewport  
 ref={ref}  
 className={cn(  
 "fixed top-0 z-[100] flex max-h-screen w-full flex-col-reverse p-4 sm:bottom-0 sm:right-0 sm:top-auto sm:flex-col md:max-w-[420px]",  
 className  
 )}  
 {...props}  
 />  
))  
ToastViewport.displayName = ToastPrimitives.Viewport.displayName  
  
const toastVariants = cva(  
 "group pointer-events-auto relative flex w-full items-center justify-between space-x-4 overflow-hidden rounded-md border p-6 pr-8 shadow-lg transition-all data-[swipe=cancel]:translate-x-0 data-[swipe=end]:translate-x-[var(--radix-toast-swipe-end-x)] data-[swipe=move]:translate-x-[var(--radix-toast-swipe-move-x)] data-[swipe=move]:transition-none data-[state=open]:animate-in data-[state=closed]:animate-out data-[swipe=end]:animate-out data-[state=closed]:fade-out-80 data-[state=closed]:slide-out-to-right-full data-[state=open]:slide-in-from-top-full data-[state=open]:sm:slide-in-from-bottom-full",  
 {  
 variants: {  
 variant: {  
 default: "border bg-background text-foreground",  
 destructive:  
 "destructive group border-destructive bg-destructive text-destructive-foreground",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 },  
 }  
)  
  
const Toast = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Root>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Root> &  
 VariantProps<typeof toastVariants>  
>(({ className, variant, ...props }, ref) => {  
 return (  
 <ToastPrimitives.Root  
 ref={ref}  
 className={cn(toastVariants({ variant }), className)}  
 {...props}  
 />  
 )  
})  
Toast.displayName = ToastPrimitives.Root.displayName  
  
const ToastAction = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Action>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Action>  
>(({ className, ...props }, ref) => (  
 <ToastPrimitives.Action  
 ref={ref}  
 className={cn(  
 "inline-flex h-8 shrink-0 items-center justify-center rounded-md border bg-transparent px-3 text-sm font-medium ring-offset-background transition-colors hover:bg-secondary focus:outline-none focus:ring-2 focus:ring-ring focus:ring-offset-2 disabled:pointer-events-none disabled:opacity-50 group-[.destructive]:border-muted/40 group-[.destructive]:hover:border-destructive/30 group-[.destructive]:hover:bg-destructive group-[.destructive]:hover:text-destructive-foreground group-[.destructive]:focus:ring-destructive",  
 className  
 )}  
 {...props}  
 />  
))  
ToastAction.displayName = ToastPrimitives.Action.displayName  
  
const ToastClose = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Close>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Close>  
>(({ className, ...props }, ref) => (  
 <ToastPrimitives.Close  
 ref={ref}  
 className={cn(  
 "absolute right-2 top-2 rounded-md p-1 text-foreground/50 opacity-0 transition-opacity hover:text-foreground focus:opacity-100 focus:outline-none focus:ring-2 group-hover:opacity-100 group-[.destructive]:text-red-300 group-[.destructive]:hover:text-red-50 group-[.destructive]:focus:ring-red-400 group-[.destructive]:focus:ring-offset-red-600",  
 className  
 )}  
 toast-close=""  
 {...props}  
 >  
 <X className="h-4 w-4" />  
 </ToastPrimitives.Close>  
))  
ToastClose.displayName = ToastPrimitives.Close.displayName  
  
const ToastTitle = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Title>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Title>  
>(({ className, ...props }, ref) => (  
 <ToastPrimitives.Title  
 ref={ref}  
 className={cn("text-sm font-semibold", className)}  
 {...props}  
 />  
))  
ToastTitle.displayName = ToastPrimitives.Title.displayName  
  
const ToastDescription = React.forwardRef<  
 React.ElementRef<typeof ToastPrimitives.Description>,  
 React.ComponentPropsWithoutRef<typeof ToastPrimitives.Description>  
>(({ className, ...props }, ref) => (  
 <ToastPrimitives.Description  
 ref={ref}  
 className={cn("text-sm opacity-90", className)}  
 {...props}  
 />  
))  
ToastDescription.displayName = ToastPrimitives.Description.displayName  
  
type ToastProps = React.ComponentPropsWithoutRef<typeof Toast>  
  
type ToastActionElement = React.ReactElement<typeof ToastAction>  
  
export {  
 type ToastProps,  
 type ToastActionElement,  
 ToastProvider,  
 ToastViewport,  
 Toast,  
 ToastTitle,  
 ToastDescription,  
 ToastClose,  
 ToastAction,  
}

client/src/components/ui/toaster.tsx

import { useToast } from "@/hooks/use-toast"  
import {  
 Toast,  
 ToastClose,  
 ToastDescription,  
 ToastProvider,  
 ToastTitle,  
 ToastViewport,  
} from "@/components/ui/toast"  
  
export function Toaster() {  
 const { toasts } = useToast()  
  
 return (  
 <ToastProvider>  
 {toasts.map(function ({ id, title, description, action, ...props }) {  
 return (  
 <Toast key={id} {...props}>  
 <div className="grid gap-1">  
 {title && <ToastTitle>{title}</ToastTitle>}  
 {description && (  
 <ToastDescription>{description}</ToastDescription>  
 )}  
 </div>  
 {action}  
 <ToastClose />  
 </Toast>  
 )  
 })}  
 <ToastViewport />  
 </ToastProvider>  
 )  
}

client/src/components/ui/toggle-group.tsx

import \* as React from "react"  
import \* as ToggleGroupPrimitive from "@radix-ui/react-toggle-group"  
import { type VariantProps } from "class-variance-authority"  
  
import { cn } from "@/lib/utils"  
import { toggleVariants } from "@/components/ui/toggle"  
  
const ToggleGroupContext = React.createContext<  
 VariantProps<typeof toggleVariants>  
>({  
 size: "default",  
 variant: "default",  
})  
  
const ToggleGroup = React.forwardRef<  
 React.ElementRef<typeof ToggleGroupPrimitive.Root>,  
 React.ComponentPropsWithoutRef<typeof ToggleGroupPrimitive.Root> &  
 VariantProps<typeof toggleVariants>  
>(({ className, variant, size, children, ...props }, ref) => (  
 <ToggleGroupPrimitive.Root  
 ref={ref}  
 className={cn("flex items-center justify-center gap-1", className)}  
 {...props}  
 >  
 <ToggleGroupContext.Provider value={{ variant, size }}>  
 {children}  
 </ToggleGroupContext.Provider>  
 </ToggleGroupPrimitive.Root>  
))  
  
ToggleGroup.displayName = ToggleGroupPrimitive.Root.displayName  
  
const ToggleGroupItem = React.forwardRef<  
 React.ElementRef<typeof ToggleGroupPrimitive.Item>,  
 React.ComponentPropsWithoutRef<typeof ToggleGroupPrimitive.Item> &  
 VariantProps<typeof toggleVariants>  
>(({ className, children, variant, size, ...props }, ref) => {  
 const context = React.useContext(ToggleGroupContext)  
  
 return (  
 <ToggleGroupPrimitive.Item  
 ref={ref}  
 className={cn(  
 toggleVariants({  
 variant: context.variant || variant,  
 size: context.size || size,  
 }),  
 className  
 )}  
 {...props}  
 >  
 {children}  
 </ToggleGroupPrimitive.Item>  
 )  
})  
  
ToggleGroupItem.displayName = ToggleGroupPrimitive.Item.displayName  
  
export { ToggleGroup, ToggleGroupItem }

client/src/components/ui/toggle.tsx

import \* as React from "react"  
import \* as TogglePrimitive from "@radix-ui/react-toggle"  
import { cva, type VariantProps } from "class-variance-authority"  
  
import { cn } from "@/lib/utils"  
  
const toggleVariants = cva(  
 "inline-flex items-center justify-center rounded-md text-sm font-medium ring-offset-background transition-colors hover:bg-muted hover:text-muted-foreground focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50 data-[state=on]:bg-accent data-[state=on]:text-accent-foreground",  
 {  
 variants: {  
 variant: {  
 default: "bg-transparent",  
 outline:  
 "border border-input bg-transparent hover:bg-accent hover:text-accent-foreground",  
 },  
 size: {  
 default: "h-10 px-3",  
 sm: "h-9 px-2.5",  
 lg: "h-11 px-5",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 size: "default",  
 },  
 }  
)  
  
const Toggle = React.forwardRef<  
 React.ElementRef<typeof TogglePrimitive.Root>,  
 React.ComponentPropsWithoutRef<typeof TogglePrimitive.Root> &  
 VariantProps<typeof toggleVariants>  
>(({ className, variant, size, ...props }, ref) => (  
 <TogglePrimitive.Root  
 ref={ref}  
 className={cn(toggleVariants({ variant, size, className }))}  
 {...props}  
 />  
))  
  
Toggle.displayName = TogglePrimitive.Root.displayName  
  
export { Toggle, toggleVariants }

client/src/components/ui/tooltip.tsx

import \* as React from "react"  
import \* as TooltipPrimitive from "@radix-ui/react-tooltip"  
  
import { cn } from "@/lib/utils"  
  
const TooltipProvider = TooltipPrimitive.Provider  
  
const Tooltip = TooltipPrimitive.Root  
  
const TooltipTrigger = TooltipPrimitive.Trigger  
  
const TooltipContent = React.forwardRef<  
 React.ElementRef<typeof TooltipPrimitive.Content>,  
 React.ComponentPropsWithoutRef<typeof TooltipPrimitive.Content>  
>(({ className, sideOffset = 4, ...props }, ref) => (  
 <TooltipPrimitive.Content  
 ref={ref}  
 sideOffset={sideOffset}  
 className={cn(  
 "z-50 overflow-hidden rounded-md border bg-popover px-3 py-1.5 text-sm text-popover-foreground shadow-md animate-in fade-in-0 zoom-in-95 data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=closed]:zoom-out-95 data-[side=bottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-left-2 data-[side=top]:slide-in-from-bottom-2",  
 className  
 )}  
 {...props}  
 />  
))  
TooltipContent.displayName = TooltipPrimitive.Content.displayName  
  
export { Tooltip, TooltipTrigger, TooltipContent, TooltipProvider }

client/src/hooks/use-auth.tsx

import { createContext, ReactNode, useContext } from "react";  
import {  
 useQuery,  
 useMutation,  
 UseMutationResult,  
} from "@tanstack/react-query";  
import { insertUserSchema, User } from "@shared/schema";  
import { getQueryFn, apiRequest, queryClient } from "../lib/queryClient";  
import { useToast } from "@/hooks/use-toast";  
import { z } from "zod";  
  
type AuthContextType = {  
 user: User | null;  
 isLoading: boolean;  
 error: Error | null;  
 loginMutation: UseMutationResult<User, Error, LoginData>;  
 logoutMutation: UseMutationResult<void, Error, void>;  
 registerMutation: UseMutationResult<User, Error, RegisterData>;  
};  
  
type LoginData = {  
 username: string;  
 password: string;  
};  
  
// Extended schema for registration with password confirmation  
const registerUserSchema = insertUserSchema.extend({  
 confirmPassword: z.string({ required\_error: "Please confirm your password" }),  
}).refine((data) => data.password === data.confirmPassword, {  
 message: "Passwords do not match",  
 path: ["confirmPassword"],  
});  
  
type RegisterData = z.infer<typeof registerUserSchema>;  
  
export const AuthContext = createContext<AuthContextType | null>(null);  
  
export function AuthProvider({ children }: { children: ReactNode }) {  
 const { toast } = useToast();  
 const {  
 data: user,  
 error,  
 isLoading,  
 } = useQuery<User | null, Error>({  
 queryKey: ["/api/user"],  
 queryFn: getQueryFn({ on401: "returnNull" }),  
 });  
  
 const loginMutation = useMutation({  
 mutationFn: async (credentials: LoginData) => {  
 const res = await apiRequest("POST", "/api/login", credentials);  
 return await res.json();  
 },  
 onSuccess: (user: User) => {  
 queryClient.setQueryData(["/api/user"], user);  
 toast({  
 title: "Welcome back!",  
 description: "You have successfully logged in.",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Login failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 const registerMutation = useMutation({  
 mutationFn: async (credentials: RegisterData) => {  
 // Remove confirmPassword before sending to API  
 const { confirmPassword, ...userData } = credentials;  
 const res = await apiRequest("POST", "/api/register", userData);  
 return await res.json();  
 },  
 onSuccess: (user: User) => {  
 queryClient.setQueryData(["/api/user"], user);  
 toast({  
 title: "Account created",  
 description: "You have successfully registered and logged in.",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Registration failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 const logoutMutation = useMutation({  
 mutationFn: async () => {  
 await apiRequest("POST", "/api/logout");  
 },  
 onSuccess: () => {  
 queryClient.setQueryData(["/api/user"], null);  
 queryClient.invalidateQueries();  
 toast({  
 title: "Logged out",  
 description: "You have been successfully logged out.",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Logout failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 return (  
 <AuthContext.Provider  
 value={{  
 user: user ?? null,  
 isLoading,  
 error,  
 loginMutation,  
 logoutMutation,  
 registerMutation,  
 }}  
 >  
 {children}  
 </AuthContext.Provider>  
 );  
}  
  
export function useAuth() {  
 const context = useContext(AuthContext);  
 if (!context) {  
 throw new Error("useAuth must be used within an AuthProvider");  
 }  
 return context;  
}

client/src/hooks/use-mobile.tsx

import \* as React from "react"  
  
const MOBILE\_BREAKPOINT = 768  
  
export function useIsMobile() {  
 const [isMobile, setIsMobile] = React.useState<boolean | undefined>(undefined)  
  
 React.useEffect(() => {  
 const mql = window.matchMedia(`(max-width: ${MOBILE\_BREAKPOINT - 1}px)`)  
 const onChange = () => {  
 setIsMobile(window.innerWidth < MOBILE\_BREAKPOINT)  
 }  
 mql.addEventListener("change", onChange)  
 setIsMobile(window.innerWidth < MOBILE\_BREAKPOINT)  
 return () => mql.removeEventListener("change", onChange)  
 }, [])  
  
 return !!isMobile  
}

client/src/hooks/use-toast.ts

import \* as React from "react"  
  
import type {  
 ToastActionElement,  
 ToastProps,  
} from "@/components/ui/toast"  
  
const TOAST\_LIMIT = 1  
const TOAST\_REMOVE\_DELAY = 1000000  
  
type ToasterToast = ToastProps & {  
 id: string  
 title?: React.ReactNode  
 description?: React.ReactNode  
 action?: ToastActionElement  
}  
  
const actionTypes = {  
 ADD\_TOAST: "ADD\_TOAST",  
 UPDATE\_TOAST: "UPDATE\_TOAST",  
 DISMISS\_TOAST: "DISMISS\_TOAST",  
 REMOVE\_TOAST: "REMOVE\_TOAST",  
} as const  
  
let count = 0  
  
function genId() {  
 count = (count + 1) % Number.MAX\_SAFE\_INTEGER  
 return count.toString()  
}  
  
type ActionType = typeof actionTypes  
  
type Action =  
 | {  
 type: ActionType["ADD\_TOAST"]  
 toast: ToasterToast  
 }  
 | {  
 type: ActionType["UPDATE\_TOAST"]  
 toast: Partial<ToasterToast>  
 }  
 | {  
 type: ActionType["DISMISS\_TOAST"]  
 toastId?: ToasterToast["id"]  
 }  
 | {  
 type: ActionType["REMOVE\_TOAST"]  
 toastId?: ToasterToast["id"]  
 }  
  
interface State {  
 toasts: ToasterToast[]  
}  
  
const toastTimeouts = new Map<string, ReturnType<typeof setTimeout>>()  
  
const addToRemoveQueue = (toastId: string) => {  
 if (toastTimeouts.has(toastId)) {  
 return  
 }  
  
 const timeout = setTimeout(() => {  
 toastTimeouts.delete(toastId)  
 dispatch({  
 type: "REMOVE\_TOAST",  
 toastId: toastId,  
 })  
 }, TOAST\_REMOVE\_DELAY)  
  
 toastTimeouts.set(toastId, timeout)  
}  
  
export const reducer = (state: State, action: Action): State => {  
 switch (action.type) {  
 case "ADD\_TOAST":  
 return {  
 ...state,  
 toasts: [action.toast, ...state.toasts].slice(0, TOAST\_LIMIT),  
 }  
  
 case "UPDATE\_TOAST":  
 return {  
 ...state,  
 toasts: state.toasts.map((t) =>  
 t.id === action.toast.id ? { ...t, ...action.toast } : t  
 ),  
 }  
  
 case "DISMISS\_TOAST": {  
 const { toastId } = action  
  
 // ! Side effects ! - This could be extracted into a dismissToast() action,  
 // but I'll keep it here for simplicity  
 if (toastId) {  
 addToRemoveQueue(toastId)  
 } else {  
 state.toasts.forEach((toast) => {  
 addToRemoveQueue(toast.id)  
 })  
 }  
  
 return {  
 ...state,  
 toasts: state.toasts.map((t) =>  
 t.id === toastId || toastId === undefined  
 ? {  
 ...t,  
 open: false,  
 }  
 : t  
 ),  
 }  
 }  
 case "REMOVE\_TOAST":  
 if (action.toastId === undefined) {  
 return {  
 ...state,  
 toasts: [],  
 }  
 }  
 return {  
 ...state,  
 toasts: state.toasts.filter((t) => t.id !== action.toastId),  
 }  
 }  
}  
  
const listeners: Array<(state: State) => void> = []  
  
let memoryState: State = { toasts: [] }  
  
function dispatch(action: Action) {  
 memoryState = reducer(memoryState, action)  
 listeners.forEach((listener) => {  
 listener(memoryState)  
 })  
}  
  
type Toast = Omit<ToasterToast, "id">  
  
function toast({ ...props }: Toast) {  
 const id = genId()  
  
 const update = (props: ToasterToast) =>  
 dispatch({  
 type: "UPDATE\_TOAST",  
 toast: { ...props, id },  
 })  
 const dismiss = () => dispatch({ type: "DISMISS\_TOAST", toastId: id })  
  
 dispatch({  
 type: "ADD\_TOAST",  
 toast: {  
 ...props,  
 id,  
 open: true,  
 onOpenChange: (open) => {  
 if (!open) dismiss()  
 },  
 },  
 })  
  
 return {  
 id: id,  
 dismiss,  
 update,  
 }  
}  
  
function useToast() {  
 const [state, setState] = React.useState<State>(memoryState)  
  
 React.useEffect(() => {  
 listeners.push(setState)  
 return () => {  
 const index = listeners.indexOf(setState)  
 if (index > -1) {  
 listeners.splice(index, 1)  
 }  
 }  
 }, [state])  
  
 return {  
 ...state,  
 toast,  
 dismiss: (toastId?: string) => dispatch({ type: "DISMISS\_TOAST", toastId }),  
 }  
}  
  
export { useToast, toast }

client/src/index.css

@tailwind base;  
@tailwind components;  
@tailwind utilities;  
  
@layer base {  
 \* {  
 @apply border-border;  
 }  
  
 body {  
 @apply font-sans antialiased bg-background text-foreground;  
 }  
}

client/src/lib/protected-route.tsx

import { useAuth } from "@/hooks/use-auth";  
import { Loader2 } from "lucide-react";  
import { Redirect, Route } from "wouter";  
  
export function ProtectedRoute({  
 path,  
 component: Component,  
}: {  
 path: string;  
 component: () => React.JSX.Element;  
}) {  
 const { user, isLoading } = useAuth();  
  
 if (isLoading) {  
 return (  
 <Route path={path}>  
 <div className="flex items-center justify-center min-h-screen">  
 <Loader2 className="h-8 w-8 animate-spin text-primary" />  
 </div>  
 </Route>  
 );  
 }  
  
 if (!user) {  
 return (  
 <Route path={path}>  
 <Redirect to="/auth" />  
 </Route>  
 );  
 }  
  
 return <Route path={path} component={Component} />;  
}

client/src/lib/queryClient.ts

import { QueryClient, QueryFunction } from "@tanstack/react-query";  
  
async function throwIfResNotOk(res: Response) {  
 if (!res.ok) {  
 const text = (await res.text()) || res.statusText;  
 throw new Error(`${res.status}: ${text}`);  
 }  
}  
  
export async function apiRequest(  
 method: string,  
 url: string,  
 data?: unknown | undefined,  
): Promise<Response> {  
 const res = await fetch(url, {  
 method,  
 headers: data ? { "Content-Type": "application/json" } : {},  
 body: data ? JSON.stringify(data) : undefined,  
 credentials: "include",  
 });  
  
 await throwIfResNotOk(res);  
 return res;  
}  
  
type UnauthorizedBehavior = "returnNull" | "throw";  
export const getQueryFn: <T>(options: {  
 on401: UnauthorizedBehavior;  
}) => QueryFunction<T> =  
 ({ on401: unauthorizedBehavior }) =>  
 async ({ queryKey }) => {  
 const res = await fetch(queryKey[0] as string, {  
 credentials: "include",  
 });  
  
 if (unauthorizedBehavior === "returnNull" && res.status === 401) {  
 return null;  
 }  
  
 await throwIfResNotOk(res);  
 return await res.json();  
 };  
  
export const queryClient = new QueryClient({  
 defaultOptions: {  
 queries: {  
 queryFn: getQueryFn({ on401: "throw" }),  
 refetchInterval: false,  
 refetchOnWindowFocus: false,  
 staleTime: Infinity,  
 retry: false,  
 },  
 mutations: {  
 retry: false,  
 },  
 },  
});

client/src/lib/utils.ts

import { clsx, type ClassValue } from "clsx"  
import { twMerge } from "tailwind-merge"  
  
export function cn(...inputs: ClassValue[]) {  
 return twMerge(clsx(inputs))  
}

client/src/main.tsx

import { createRoot } from "react-dom/client";  
import App from "./App";  
import "./index.css";  
  
createRoot(document.getElementById("root")!).render(<App />);

client/src/pages/auth-page.tsx

import { useState, useEffect } from "react";  
import { useLocation } from "wouter";  
import { useAuth } from "@/hooks/use-auth";  
import { zodResolver } from "@hookform/resolvers/zod";  
import { useForm } from "react-hook-form";  
import { z } from "zod";  
import { Button } from "@/components/ui/button";  
import { Card, CardContent } from "@/components/ui/card";  
import { Input } from "@/components/ui/input";  
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";  
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";  
import { Separator } from "@/components/ui/separator";  
  
import { insertUserSchema } from "@shared/schema";  
  
// Login schema  
const loginSchema = z.object({  
 username: z.string({ required\_error: "Username is required" }).min(3, "Username must be at least 3 characters"),  
 password: z.string({ required\_error: "Password is required" }).min(1, "Password is required"),  
});  
  
// Registration schema - use the shared schema to ensure consistency  
const registerSchema = insertUserSchema.extend({  
 confirmPassword: z.string({ required\_error: "Please confirm your password" }),  
}).refine((data) => data.password === data.confirmPassword, {  
 message: "Passwords do not match",  
 path: ["confirmPassword"],  
});  
  
export default function AuthPage() {  
 const [\_, setLocation] = useLocation();  
 const { user, loginMutation, registerMutation } = useAuth();  
 const [activeTab, setActiveTab] = useState<string>("login");  
 const [showPassword, setShowPassword] = useState(false);  
  
 const loginForm = useForm<z.infer<typeof loginSchema>>({  
 resolver: zodResolver(loginSchema),  
 defaultValues: {  
 username: "",  
 password: "",  
 },  
 });  
  
 const registerForm = useForm<z.infer<typeof registerSchema>>({  
 resolver: zodResolver(registerSchema),  
 defaultValues: {  
 username: "",  
 email: "",  
 password: "",  
 confirmPassword: "",  
 },  
 });  
  
 // If user is already logged in, redirect to dashboard  
 useEffect(() => {  
 if (user) {  
 setLocation("/");  
 }  
 }, [user, setLocation]);  
  
 const onLoginSubmit = async (data: z.infer<typeof loginSchema>) => {  
 await loginMutation.mutateAsync(data);  
 };  
  
 const onRegisterSubmit = async (data: z.infer<typeof registerSchema>) => {  
 await registerMutation.mutateAsync(data);  
 };  
  
 return (  
 <div className="min-h-screen w-full flex flex-col md:flex-row">  
 {/\* Left auth form section \*/}  
 <div className="w-full md:w-1/2 min-h-screen flex flex-col justify-center px-4 py-12 md:p-12 bg-background">  
 <div className="mb-8 flex justify-center md:justify-start">  
 <div className="flex items-center gap-2">  
 <svg className="w-10 h-10 text-primary" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">  
 <path d="M13.5 6L10 18.5M6.5 8.5L3 12L6.5 15.5M17.5 8.5L21 12L17.5 15.5" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round"/>  
 </svg>  
 <h1 className="text-xl font-bold">LinkInsight</h1>  
 </div>  
 </div>  
  
 <Card className="max-w-md mx-auto w-full">  
 <CardContent className="pt-6">  
 <Tabs defaultValue="login" value={activeTab} onValueChange={setActiveTab}>  
 <TabsList className="grid w-full grid-cols-2 mb-4">  
 <TabsTrigger value="login">Login</TabsTrigger>  
 <TabsTrigger value="register">Register</TabsTrigger>  
 </TabsList>  
  
 <TabsContent value="login">  
 <div className="space-y-4">  
 <div className="text-center mb-6">  
 <h2 className="text-2xl font-bold">Welcome back</h2>  
 <p className="text-muted-foreground">Sign in to access your dashboard</p>  
 </div>  
  
 <Form {...loginForm}>  
 <form onSubmit={loginForm.handleSubmit(onLoginSubmit)} className="space-y-4">  
 <FormField  
 control={loginForm.control}  
 name="username"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Username</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="username"   
 {...field}   
 autoComplete="username"  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <FormField  
 control={loginForm.control}  
 name="password"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Password</FormLabel>  
 <FormControl>  
 <div className="relative">  
 <Input   
 type={showPassword ? "text" : "password"}  
 placeholder="••••••••"   
 {...field}   
 autoComplete="current-password"  
 />  
 <button  
 type="button"  
 onClick={() => setShowPassword(!showPassword)}  
 className="absolute right-2 top-1/2 -translate-y-1/2 text-muted-foreground hover:text-foreground"  
 >  
 {showPassword ? (  
 <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round">  
 <path d="M2 12s3-7 10-7 10 7 10 7-3 7-10 7-10-7-10-7Z"/>  
 <circle cx="12" cy="12" r="3"/>  
 </svg>  
 ) : (  
 <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round">  
 <path d="M9.88 9.88a3 3 0 1 0 4.24 4.24"/>  
 <path d="M10.73 5.08A10.43 10.43 0 0 1 12 5c7 0 10 7 10 7a13.16 13.16 0 0 1-1.67 2.68"/>  
 <path d="M6.61 6.61A13.526 13.526 0 0 0 2 12s3 7 10 7a9.74 9.74 0 0 0 5.39-1.61"/>  
 <line x1="2" y1="2" x2="22" y2="22"/>  
 </svg>  
 )}  
 </button>  
 </div>  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <Button   
 type="submit"   
 className="w-full"   
 disabled={loginMutation.isPending}  
 >  
 {loginMutation.isPending ? (  
 <div className="flex items-center gap-2">  
 <span className="h-4 w-4 animate-spin rounded-full border-2 border-background border-t-transparent"></span>  
 Logging in...  
 </div>  
 ) : "Login"}  
 </Button>  
 </form>  
 </Form>  
  
 <div className="text-center">  
 <p className="text-sm text-muted-foreground">  
 Don't have an account?{" "}  
 <Button variant="link" className="p-0" onClick={() => setActiveTab("register")}>  
 Register  
 </Button>  
 </p>  
 </div>  
 </div>  
 </TabsContent>  
  
 <TabsContent value="register">  
 <div className="space-y-4">  
 <div className="text-center mb-6">  
 <h2 className="text-2xl font-bold">Create an account</h2>  
 <p className="text-muted-foreground">Sign up to get started</p>  
 </div>  
  
 <Form {...registerForm}>  
 <form onSubmit={registerForm.handleSubmit(onRegisterSubmit)} className="space-y-4">  
 <FormField  
 control={registerForm.control}  
 name="username"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Username</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="Choose a username"   
 {...field}   
 autoComplete="username"  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={registerForm.control}  
 name="email"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Email (optional)</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="your@email.com"   
 onChange={field.onChange}  
 onBlur={field.onBlur}  
 value={field.value ?? ""}  
 name={field.name}  
 ref={field.ref}  
 autoComplete="email"  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <FormField  
 control={registerForm.control}  
 name="password"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Password</FormLabel>  
 <FormControl>  
 <Input   
 type="password"   
 placeholder="••••••••"   
 {...field}   
 autoComplete="new-password"  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <FormField  
 control={registerForm.control}  
 name="confirmPassword"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Confirm Password</FormLabel>  
 <FormControl>  
 <Input   
 type="password"   
 placeholder="••••••••"   
 {...field}   
 autoComplete="new-password"  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <Button   
 type="submit"   
 className="w-full"   
 disabled={registerMutation.isPending}  
 >  
 {registerMutation.isPending ? (  
 <div className="flex items-center gap-2">  
 <span className="h-4 w-4 animate-spin rounded-full border-2 border-background border-t-transparent"></span>  
 Creating account...  
 </div>  
 ) : "Register"}  
 </Button>  
 </form>  
 </Form>  
  
 <div className="text-center">  
 <p className="text-sm text-muted-foreground">  
 Already have an account?{" "}  
 <Button variant="link" className="p-0" onClick={() => setActiveTab("login")}>  
 Login  
 </Button>  
 </p>  
 </div>  
 </div>  
 </TabsContent>  
 </Tabs>  
 </CardContent>  
 </Card>  
  
 <div className="mt-8 text-center">  
 <p className="text-sm text-muted-foreground">  
 Demo account: intern@dacoid.com / Test123  
 </p>  
 </div>  
 </div>  
  
 {/\* Right info section \*/}  
 <div className="hidden md:flex md:w-1/2 bg-primary/5 flex-col justify-center items-center p-12">  
 <div className="max-w-md">  
 <svg className="w-16 h-16 text-primary mb-6" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">  
 <path d="M13.5 6L10 18.5M6.5 8.5L3 12L6.5 15.5M17.5 8.5L21 12L17.5 15.5" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round"/>  
 </svg>  
   
 <h2 className="text-3xl font-bold mb-4">LinkInsight</h2>  
 <p className="text-xl text-muted-foreground mb-6">  
 Shorten, share, and track your links with powerful analytics.  
 </p>  
   
 <Separator className="my-8" />  
   
 <div className="space-y-6">  
 <div className="flex items-start gap-3">  
 <div className="mt-1 bg-primary/10 p-2 rounded-md">  
 <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" className="text-primary">  
 <path d="M9 6l6 6-6 6"/>  
 </svg>  
 </div>  
 <div>  
 <h3 className="font-medium">Create Short Links</h3>  
 <p className="text-muted-foreground">Transform long URLs into short, memorable links with optional custom aliases.</p>  
 </div>  
 </div>  
   
 <div className="flex items-start gap-3">  
 <div className="mt-1 bg-primary/10 p-2 rounded-md">  
 <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" className="text-primary">  
 <path d="M2 12h5"/>  
 <path d="M2 12a10 10 0 1 0 20 0 10 10 0 1 0-20 0Z"/>  
 <path d="M12 2a5 5 0 0 0-5 5v10"/>  
 </svg>  
 </div>  
 <div>  
 <h3 className="font-medium">Track Performance</h3>  
 <p className="text-muted-foreground">Monitor clicks, locations, and device usage with real-time analytics.</p>  
 </div>  
 </div>  
   
 <div className="flex items-start gap-3">  
 <div className="mt-1 bg-primary/10 p-2 rounded-md">  
 <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 24 24" fill="none" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round" className="text-primary">  
 <rect width="18" height="18" x="3" y="3" rx="2"/>  
 <path d="M3 9h18"/>  
 <path d="M3 15h18"/>  
 <path d="M9 3v18"/>  
 <path d="M15 3v18"/>  
 </svg>  
 </div>  
 <div>  
 <h3 className="font-medium">Generate QR Codes</h3>  
 <p className="text-muted-foreground">Instantly create and download QR codes for your shortened links.</p>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
}

client/src/pages/dashboard.tsx

import { useState } from "react";  
import { useAuth } from "@/hooks/use-auth";  
import { useLocation, Link } from "wouter";  
import Sidebar from "@/components/layout/sidebar";  
import Header from "@/components/layout/header";  
import StatsGrid from "@/components/stats-grid";  
import ClicksChart from "@/components/charts/clicks-chart";  
import DevicesChart from "@/components/charts/devices-chart";  
import CreateLinkForm from "@/components/create-link-form";  
import LinksTable from "@/components/links-table";  
import QrCodeModal from "@/components/qr-code-modal";  
import { useQuery } from "@tanstack/react-query";  
import { LinkWithAnalytics } from "@shared/schema";  
import { Button } from "@/components/ui/button";  
  
// Dashboard section component  
function DashboardSection() {  
 return (  
 <>  
 {/\* Stats Grid \*/}  
 <StatsGrid />  
  
 {/\* Charts \*/}  
 <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 mb-6">  
 <ClicksChart className="lg:col-span-2" />  
 <DevicesChart />  
 </div>  
 </>  
 );  
}  
  
// Links section component  
function LinksSection({   
 links = [] as LinkWithAnalytics[],   
 isLoading = false,   
 searchQuery = "",  
 onQrCodeClick = (link: LinkWithAnalytics) => {}  
}: {  
 links?: LinkWithAnalytics[];  
 isLoading?: boolean;  
 searchQuery?: string;  
 onQrCodeClick?: (link: LinkWithAnalytics) => void;  
}) {  
 return (  
 <>  
 <div className="mb-6">  
 <div className="flex items-center justify-between">  
 <div>  
 <h2 className="text-xl font-semibold">Your Links</h2>  
 <p className="text-muted-foreground">Manage and track all your shortened links</p>  
 </div>  
 <Link href="/create">  
 <Button>Create New Link</Button>  
 </Link>  
 </div>  
 </div>  
   
 <LinksTable   
 links={links}   
 isLoading={isLoading}  
 searchQuery={searchQuery}  
 onQrCodeClick={onQrCodeClick}  
 />  
 </>  
 );  
}  
  
// Create Link section component  
function CreateSection() {  
 return (  
 <>  
 <div className="mb-6">  
 <h2 className="text-xl font-semibold">Create New Link</h2>  
 <p className="text-muted-foreground">Create a new shortened link with optional settings</p>  
 </div>  
   
 <CreateLinkForm />  
 </>  
 );  
}  
  
// QR Codes section component  
function QrCodesSection({   
 links = [] as LinkWithAnalytics[],   
 isLoading = false,  
 onQrCodeClick = (link: LinkWithAnalytics) => {}  
}: {  
 links?: LinkWithAnalytics[];  
 isLoading?: boolean;  
 onQrCodeClick?: (link: LinkWithAnalytics) => void;  
}) {  
 return (  
 <>  
 <div className="mb-6">  
 <h2 className="text-xl font-semibold">QR Codes</h2>  
 <p className="text-muted-foreground">Generate and download QR codes for your links</p>  
 </div>  
   
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">  
 {isLoading ? (  
 <p>Loading QR codes...</p>  
 ) : links.length === 0 ? (  
 <p>No links found. Create some links to generate QR codes.</p>  
 ) : (  
 links.map(link => (  
 <div key={link.id} className="bg-card rounded-lg p-4 border">  
 <div className="mb-2 font-medium truncate">{link.originalUrl}</div>  
 <div className="text-sm text-muted-foreground mb-3">/{link.shortCode}</div>  
 <Button   
 variant="outline"   
 className="w-full"  
 onClick={() => onQrCodeClick(link)}  
 >  
 View QR Code  
 </Button>  
 </div>  
 ))  
 )}  
 </div>  
 </>  
 );  
}  
  
// Settings section component  
function SettingsSection() {  
 const { user } = useAuth();  
   
 return (  
 <>  
 <div className="mb-6">  
 <h2 className="text-xl font-semibold">Settings</h2>  
 <p className="text-muted-foreground">Manage your account settings</p>  
 </div>  
   
 <div className="bg-card rounded-lg p-6 border">  
 <h3 className="text-lg font-medium mb-4">Account Information</h3>  
 <div className="space-y-2">  
 <div>  
 <div className="text-sm text-muted-foreground">Username</div>  
 <div>{user?.username}</div>  
 </div>  
 {user?.email && (  
 <div>  
 <div className="text-sm text-muted-foreground">Email</div>  
 <div>{user.email}</div>  
 </div>  
 )}  
 <div>  
 <div className="text-sm text-muted-foreground">Member since</div>  
 <div>{user?.createdAt ? new Date(user.createdAt).toLocaleDateString() : "N/A"}</div>  
 </div>  
 </div>  
 </div>  
 </>  
 );  
}  
  
export default function Dashboard() {  
 const { user } = useAuth();  
 const [location] = useLocation();  
 const [sidebarOpen, setSidebarOpen] = useState(false);  
 const [qrModalOpen, setQrModalOpen] = useState(false);  
 const [selectedLink, setSelectedLink] = useState<LinkWithAnalytics | null>(null);  
 const [searchQuery, setSearchQuery] = useState("");  
  
 // Fetch links for the logged-in user  
 const { data: links, isLoading: isLoadingLinks } = useQuery<LinkWithAnalytics[]>({  
 queryKey: ["/api/links"],  
 });  
  
 // Handle QR code generation  
 const handleQrCodeClick = (link: LinkWithAnalytics) => {  
 setSelectedLink(link);  
 setQrModalOpen(true);  
 };  
   
 // Determine page title and content based on route  
 let pageTitle = "Dashboard";  
 let pageDescription = "Overview of your link analytics";  
   
 if (location === "/links") {  
 pageTitle = "Your Links";  
 pageDescription = "Manage and track all your shortened links";  
 } else if (location === "/create") {  
 pageTitle = "Create Link";  
 pageDescription = "Create a new shortened link";  
 } else if (location === "/qr-codes") {  
 pageTitle = "QR Codes";  
 pageDescription = "Generate and download QR codes";  
 } else if (location === "/settings") {  
 pageTitle = "Settings";  
 pageDescription = "Manage your account settings";  
 }  
  
 return (  
 <div className="min-h-screen">  
 {/\* Sidebar \*/}  
 <Sidebar   
 user={user}   
 isOpen={sidebarOpen}   
 onClose={() => setSidebarOpen(false)}   
 />  
  
 {/\* Main Content Area \*/}  
 <div className="lg:pl-64 flex flex-col min-h-screen">  
 {/\* Header \*/}  
 <Header   
 onMenuClick={() => setSidebarOpen(true)}   
 searchQuery={searchQuery}  
 onSearchChange={setSearchQuery}  
 />  
  
 {/\* Page Content \*/}  
 <main className="flex-1 p-4 md:p-6">  
 {/\* Page Header \*/}  
 <div className="mb-6">  
 <h1 className="text-2xl font-bold">{pageTitle}</h1>  
 <p className="text-muted-foreground">{pageDescription}</p>  
 </div>  
  
 {/\* Render different sections based on route \*/}  
 {location === "/" && (  
 <DashboardSection />  
 )}  
   
 {(location === "/" || location === "/links") && (  
 <LinksSection   
 links={links || []}  
 isLoading={isLoadingLinks}  
 searchQuery={searchQuery}  
 onQrCodeClick={handleQrCodeClick}  
 />  
 )}  
   
 {location === "/create" && (  
 <CreateSection />  
 )}  
   
 {location === "/qr-codes" && (  
 <QrCodesSection   
 links={links || []}  
 isLoading={isLoadingLinks}  
 onQrCodeClick={handleQrCodeClick}  
 />  
 )}  
   
 {location === "/settings" && (  
 <SettingsSection />  
 )}  
 </main>  
  
 {/\* Footer \*/}  
 <footer className="mt-auto px-4 py-4 text-sm text-muted-foreground text-center border-t border-gray-200 dark:border-gray-800">  
 &copy; {new Date().getFullYear()} LinkInsight. All rights reserved.  
 </footer>  
 </div>  
  
 {/\* QR Code Modal \*/}  
 <QrCodeModal   
 open={qrModalOpen}   
 onClose={() => setQrModalOpen(false)}   
 link={selectedLink}  
 />  
 </div>  
 );  
}

client/src/pages/not-found.tsx

import { Card, CardContent } from "@/components/ui/card";  
import { AlertCircle } from "lucide-react";  
  
export default function NotFound() {  
 return (  
 <div className="min-h-screen w-full flex items-center justify-center bg-gray-50">  
 <Card className="w-full max-w-md mx-4">  
 <CardContent className="pt-6">  
 <div className="flex mb-4 gap-2">  
 <AlertCircle className="h-8 w-8 text-red-500" />  
 <h1 className="text-2xl font-bold text-gray-900">404 Page Not Found</h1>  
 </div>  
  
 <p className="mt-4 text-sm text-gray-600">  
 Did you forget to add the page to the router?  
 </p>  
 </CardContent>  
 </Card>  
 </div>  
 );  
}

client/src/pages/redirect.tsx

import { useEffect, useState } from "react";  
import { useLocation } from "wouter";  
import { Card, CardContent } from "@/components/ui/card";  
import { Loader2 } from "lucide-react";  
import { Button } from "@/components/ui/button";  
  
export default function RedirectPage({ params }: { params: { shortCode: string } }) {  
 const [error, setError] = useState<string | null>(null);  
 const [\_, setLocation] = useLocation();  
  
 useEffect(() => {  
 const redirectUser = async () => {  
 try {  
 // Make request to redirect endpoint  
 const res = await fetch(`/r/${params.shortCode}`);  
   
 // If not a redirect response or error  
 if (!res.redirected && !res.ok) {  
 if (res.status === 404) {  
 setError("Link not found. This short URL may not exist.");  
 } else if (res.status === 410) {  
 setError("This link has expired and is no longer available.");  
 } else {  
 const data = await res.json();  
 setError(data.message || "An error occurred during redirection.");  
 }  
 }  
   
 // Otherwise redirect will happen automatically  
 } catch (err) {  
 setError("Something went wrong. Please try again later.");  
 console.error("Redirect error:", err);  
 }  
 };  
  
 redirectUser();  
 }, [params.shortCode]);  
  
 const goHome = () => {  
 setLocation("/dashboard");  
 };  
  
 return (  
 <div className="min-h-screen flex flex-col items-center justify-center bg-gray-50 dark:bg-gray-900 p-4">  
 <Card className="w-full max-w-md bg-white dark:bg-gray-800 shadow-lg">  
 <CardContent className="p-8 text-center">  
 <div className="mb-6">  
 <svg className="mx-auto w-16 h-16 text-primary" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">  
 <path d="M13.5 6L10 18.5M6.5 8.5L3 12L6.5 15.5M17.5 8.5L21 12L17.5 15.5" stroke="currentColor" strokeWidth="2" strokeLinecap="round" strokeLinejoin="round"/>  
 </svg>  
 </div>  
   
 {error ? (  
 <>  
 <h1 className="text-2xl font-bold mb-2">Redirect Failed</h1>  
 <p className="text-muted-foreground mb-6">{error}</p>  
 <Button onClick={goHome}>  
 Return to Dashboard  
 </Button>  
 </>  
 ) : (  
 <>  
 <h1 className="text-2xl font-bold mb-2">Redirecting you...</h1>  
 <p className="text-muted-foreground mb-6">You are being redirected to your destination. Please wait.</p>  
 <div className="flex justify-center mb-6">  
 <Loader2 className="w-12 h-12 text-primary animate-spin" />  
 </div>  
 <p className="text-sm text-muted-foreground">  
 If you are not redirected automatically, <Button variant="link" className="p-0" onClick={() => window.location.reload()}>click here</Button>.  
 </p>  
 </>  
 )}  
 </CardContent>  
 </Card>  
 </div>  
 );  
}

drizzle.config.ts

import { defineConfig } from "drizzle-kit";  
  
if (!process.env.DATABASE\_URL) {  
 throw new Error("DATABASE\_URL, ensure the database is provisioned");  
}  
  
export default defineConfig({  
 out: "./migrations",  
 schema: "./shared/schema.ts",  
 dialect: "postgresql",  
 dbCredentials: {  
 url: process.env.DATABASE\_URL,  
 },  
});

generate-docs.js

import fs from 'fs';  
import path from 'path';  
import { fileURLToPath } from 'url';  
  
  
const IGNORED\_FOLDERS = ['node\_modules'];  
const IGNORED\_FILES = [];  
  
function dumpCodeToDocs(dir, outputArray, processedFiles = new Set()) {  
 try {  
 const files = fs.readdirSync(dir);  
 console.log(`Found ${files.length} items in ${dir}`);  
   
 for (const file of files) {  
 const fullPath = path.join(dir, file).replace(/\\/g, '/');  
 const stat = fs.statSync(fullPath);  
  
 if (stat.isDirectory()) {  
 const shouldIgnore = IGNORED\_FOLDERS.some(ignoredFolder => {  
 const normalizedIgnoredFolder = `/${ignoredFolder.toLowerCase()}/`;  
 const normalizedPath = `/${fullPath.toLowerCase()}/`;  
 return normalizedPath.includes(normalizedIgnoredFolder);  
 });  
  
 if (shouldIgnore) {  
 console.log(`Skipping ignored directory: ${fullPath}`);  
 continue;  
 }  
  
 console.log(`Entering directory: ${fullPath}`);  
 dumpCodeToDocs(fullPath, outputArray, processedFiles);  
 } else if (  
 !IGNORED\_FILES.includes(file) &&  
 (file.endsWith('.js') ||  
 file.endsWith('.ts') ||  
 file.endsWith('.jsx') ||  
 file.endsWith('.tsx') ||  
 file.endsWith('.html') ||  
 file.endsWith('.css') ||  
 file.endsWith('.json') ||  
 file.endsWith('.env'))  
 ) {  
 const relativePath = path.relative(process.cwd(), fullPath).replace(/\\/g, '/');  
 if (processedFiles.has(fullPath)) {  
 console.warn(`Duplicate file found: ${relativePath}`);  
 continue;  
 }  
 processedFiles.add(fullPath);  
  
 console.log(`Processing file: ${relativePath}`);  
 outputArray.push(`\n\n---\n### 📄 ${relativePath}\n\n`);  
 const content = fs.readFileSync(fullPath, 'utf-8');  
 outputArray.push('```' + getLang(file) + '\n' + content + '\n```');  
 } else {  
 console.log(`Skipping file: ${fullPath} (does not match criteria)`);  
 }  
 }  
 } catch (err) {  
 console.error(`Error processing directory ${dir}:`, err);  
 }  
}  
  
function getLang(filename) {  
 if (filename.endsWith('.js')) return 'javascript';  
 if (filename.endsWith('.ts')) return 'typescript';  
 if (filename.endsWith('.json')) return 'json';  
 if (filename.endsWith('.css')) return 'css';  
 if (filename.endsWith('.html')) return 'html';  
 if (filename.endsWith('.env')) return 'bash';  
 return '';  
}  
  
function generateDocs(targetDir = '.', outputDir = process.cwd()) {  
 const outputArray = ['# 📚 Complete Code Documentation\n'];  
 const processedFiles = new Set();  
  
 const absolutePath = path.resolve(targetDir);  
 const dirName = path.basename(absolutePath);  
 outputArray.push(`## Directory: ${dirName} (${absolutePath})\n`);  
  
 console.log(`Starting to process directory: ${absolutePath}`);  
 dumpCodeToDocs(targetDir, outputArray, processedFiles);  
  
 if (outputArray.length <= 2) {  
 console.log('No files were found to process.');  
 return;  
 }  
  
 if (!fs.existsSync(outputDir)) {  
 fs.mkdirSync(outputDir, { recursive: true });  
 console.log(`Created output directory: ${outputDir}`);  
 }  
  
 const header = outputArray.slice(0, 2);  
 const contentChunks = outputArray.slice(2);  
  
 const total = contentChunks.length;  
 const chunkCount = 4; // ⬅️ Changed to 4 parts  
 const chunkSize = Math.ceil(total / chunkCount);  
  
 for (let i = 0; i < chunkCount; i++) {  
 const part = contentChunks.slice(i \* chunkSize, (i + 1) \* chunkSize);  
 const fileContent = [...header, ...part].join('\n');  
 const outputFile = path.join(outputDir, `code-documentation-${dirName}-part${i + 1}.md`);  
 fs.writeFileSync(outputFile, fileContent, 'utf-8');  
 const percent = ((part.length / total) \* 100).toFixed(1);  
 console.log(`✅ Generated ${outputFile} (${percent}%)`);  
 }  
}  
  
// Run with command line arguments  
const targetDir = process.argv[2] || '.';  
const outputDir = process.argv[3] || process.cwd();  
generateDocs(targetDir, outputDir);

index.js

// Simple Express server for Vercel deployment  
import express from 'express';  
import path from 'path';  
import dotenv from 'dotenv';  
import { fileURLToPath } from 'url';  
import mongoose from 'mongoose';  
import session from 'express-session';  
import MongoStore from 'connect-mongo';  
import passport from 'passport';  
import { Strategy as LocalStrategy } from 'passport-local';  
import { UserModel } from './server/db/models/user.model.js';  
  
// Load environment variables  
dotenv.config();  
  
const \_\_dirname = path.dirname(fileURLToPath(import.meta.url));  
const app = express();  
  
// Middleware  
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
  
// Session configuration  
const sessionConfig = {  
 secret: process.env.SESSION\_SECRET || 'default\_secret\_for\_development',  
 resave: false,  
 saveUninitialized: false,  
 cookie: {  
 maxAge: 1000 \* 60 \* 60 \* 24, // 1 day  
 httpOnly: true  
 }  
};  
  
// Use MongoDB for sessions if available  
if (process.env.MONGODB\_URI) {  
 try {  
 sessionConfig.store = MongoStore.create({  
 mongoUrl: process.env.MONGODB\_URI,  
 collectionName: 'sessions'  
 });  
 console.log('Using MongoDB for session storage');  
 } catch (err) {  
 console.error('Failed to create MongoDB session store:', err);  
 }  
}  
  
app.use(session(sessionConfig));  
  
// Setup Passport  
app.use(passport.initialize());  
app.use(passport.session());  
  
// Connect to MongoDB  
async function connectDB() {  
 if (process.env.MONGODB\_URI) {  
 try {  
 await mongoose.connect(process.env.MONGODB\_URI);  
 console.log('Connected to MongoDB');  
 } catch (err) {  
 console.error('MongoDB connection error:', err);  
 }  
 } else {  
 console.log('No MongoDB URI provided, running with in-memory storage');  
 }  
}  
  
connectDB();  
  
// Passport configuration  
passport.use(new LocalStrategy(async (username, password, done) => {  
 try {  
 const user = await UserModel.findOne({   
 $or: [{ username }, { email: username }]   
 });  
   
 if (!user) {  
 return done(null, false, { message: 'Invalid username or password' });  
 }  
   
 const isValid = await user.comparePassword(password);  
 if (!isValid) {  
 return done(null, false, { message: 'Invalid username or password' });  
 }  
   
 return done(null, user);  
 } catch (err) {  
 return done(err);  
 }  
}));  
  
passport.serializeUser((user, done) => {  
 done(null, user.id);  
});  
  
passport.deserializeUser(async (id, done) => {  
 try {  
 const user = await UserModel.findById(id);  
 done(null, user);  
 } catch (err) {  
 done(err);  
 }  
});  
  
// API routes  
app.get('/api/user', (req, res) => {  
 if (req.isAuthenticated()) {  
 const { \_id, username, email } = req.user;  
 return res.json({  
 id: \_id.toString(),  
 username,  
 email  
 });  
 }  
 return res.status(401).json({ message: 'Not authenticated' });  
});  
  
// Serve static files  
app.use(express.static(path.join(\_\_dirname, 'dist', 'public')));  
  
// Handle all other routes - send to index.html  
app.get('\*', (req, res) => {  
 res.sendFile(path.join(\_\_dirname, 'dist', 'public', 'index.html'));  
});  
  
// Start server  
const PORT = process.env.PORT || 5000;  
app.listen(PORT, () => {  
 console.log(`Server running on port ${PORT}`);  
});  
  
export default app;