# 📚 Complete Code Documentation

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

.env

# MongoDB connection URL  
# Replace with your actual MongoDB Atlas connection string  
MONGODB\_URI=mongodb+srv://vinayst05:rsI3L0SfYfzZCobg@linkshortener.cqcswc4.mongodb.net/?retryWrites=true&w=majority&appName=linkshortener  
  
# Session secret for authentication  
SESSION\_SECRET=8e7f2c6f3bfa4d25b1a6e3e2c9b7d8a1  
  
# Node environment (development, production, test)  
NODE\_ENV=development

api/index.js

// Vercel serverless function entry point  
import express from 'express';  
import path from 'path';  
import { fileURLToPath } from 'url';  
import dotenv from 'dotenv';  
import mongoose from 'mongoose';  
import session from 'express-session';  
import MongoStore from 'connect-mongo';  
import memorystore from 'memorystore';  
import passport from 'passport';  
import { Strategy as LocalStrategy } from 'passport-local';  
import bcrypt from 'bcrypt';  
  
// Load environment variables  
dotenv.config();  
  
const \_\_filename = fileURLToPath(import.meta.url);  
const \_\_dirname = path.dirname(\_\_filename);  
  
// Initialize Express  
const app = express();  
  
// Middleware  
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
  
// Configure session  
const sessionConfig = {  
 secret: process.env.SESSION\_SECRET || 'development\_secret',  
 resave: false,  
 saveUninitialized: false,  
 cookie: {  
 maxAge: 24 \* 60 \* 60 \* 1000, // 1 day  
 secure: process.env.NODE\_ENV === 'production'  
 }  
};  
  
// Handle session store based on environment  
if (process.env.MONGODB\_URI) {  
 try {  
 // Use MongoDB session store with optimized settings for serverless  
 sessionConfig.store = MongoStore.create({  
 mongoUrl: process.env.MONGODB\_URI,  
 collectionName: 'sessions',  
 touchAfter: 24 \* 3600, // Only update session every 24 hours unless data changes  
 ttl: 14 \* 24 \* 60 \* 60, // 14 days  
 autoRemove: 'native',  
 stringify: false, // Don't stringify session  
 crypto: {  
 secret: process.env.SESSION\_SECRET || 'default-encryption-key'  
 },  
 connectionOptions: {  
 useNewUrlParser: true,  
 useUnifiedTopology: true,  
 serverSelectionTimeoutMS: 10000,  
 socketTimeoutMS: 20000,  
 family: 4,  
 maxPoolSize: 1 // Important for serverless  
 }  
 });  
 console.log('Using MongoDB session store with serverless optimizations');  
 } catch (err) {  
 // Fallback to memory store  
 console.error('Failed to create MongoDB session store:', err);  
 console.error(`Error details: ${err.message}`);  
   
 const MemoryStore = memorystore(session);  
 sessionConfig.store = new MemoryStore({  
 checkPeriod: 86400000 // 24 hours  
 });  
 console.log('Falling back to memory session store');  
 }  
} else {  
 // Use memory store  
 const MemoryStore = memorystore(session);  
 sessionConfig.store = new MemoryStore({  
 checkPeriod: 86400000 // 24 hours  
 });  
 console.log('Using memory session store (MongoDB URI not provided)');  
}  
  
// Set up session middleware  
app.use(session(sessionConfig));  
  
// Initialize Passport  
app.use(passport.initialize());  
app.use(passport.session());  
  
// Connect to MongoDB - Optimized for Vercel serverless  
let db = null;  
let isConnecting = false;  
let connectionPromise = null;  
  
// Function to parse MongoDB connection string to log sanitized version  
function getSanitizedMongoURI(uri) {  
 if (!uri) return "undefined";  
 try {  
 const parsed = new URL(uri);  
 // Only show host and database, hide credentials  
 return `mongodb://${parsed.host}${parsed.pathname}`;  
 } catch (e) {  
 return "invalid-uri-format";  
 }  
}  
  
async function connectToMongoDB() {  
 if (!process.env.MONGODB\_URI) {  
 console.log('No MongoDB URI provided - using in-memory storage');  
 return null;  
 }  
  
 // Provide helpful diagnostics about the MongoDB URI  
 const sanitizedURI = getSanitizedMongoURI(process.env.MONGODB\_URI);  
 console.log(`Attempting connection to: ${sanitizedURI}`);  
   
 // If we already have a connection, use it  
 if (mongoose.connection.readyState === 1) {  
 console.log('Using existing MongoDB connection');  
 return mongoose.connection;  
 }  
   
 // If we're already connecting, wait for that connection  
 if (isConnecting && connectionPromise) {  
 console.log('Connection to MongoDB in progress, waiting...');  
 try {  
 return await connectionPromise;  
 } catch (err) {  
 console.log('Existing connection promise failed, retrying...');  
 // Reset connection state and continue to new connection attempt  
 isConnecting = false;  
 connectionPromise = null;  
 }  
 }  
   
 console.log('Initiating new MongoDB connection...');  
   
 // Disconnect any existing connection first  
 if (mongoose.connection.readyState !== 0) {  
 console.log('Cleaning up existing connections first...');  
 await mongoose.disconnect();  
 }  
   
 try {  
 isConnecting = true;  
   
 // IMPORTANT: For Vercel serverless, we need to avoid connection pooling  
 // More info: https://vercel.com/guides/using-databases-with-vercel#mongodb  
 connectionPromise = mongoose.connect(process.env.MONGODB\_URI, {  
 serverSelectionTimeoutMS: 15000, // Reduced timeout for faster failure  
 socketTimeoutMS: 30000,  
 connectTimeoutMS: 15000,  
 family: 4, // Force IPv4 (can help with some networks)  
 maxPoolSize: 1, // Critical for serverless - keep minimal connections  
 minPoolSize: 0, // Don't keep connections open  
 maxIdleTimeMS: 10000, // Close idle connections quickly  
 serverApi: {  
 version: '1', // Use latest stable API version  
 strict: true,  
 deprecationErrors: true,  
 },  
 // These options are deprecated but included for backward compatibility  
 useNewUrlParser: true,  
 useUnifiedTopology: true,  
 });  
   
 // Wait for connection  
 await connectionPromise;  
 console.log('✅ Connected to MongoDB successfully');  
   
 // Setup error handling  
 mongoose.connection.on('error', (err) => {  
 console.error('MongoDB connection error:', err);  
 isConnecting = false;  
 connectionPromise = null;  
 });  
   
 mongoose.connection.on('disconnected', () => {  
 console.log('MongoDB disconnected');  
 isConnecting = false;  
 connectionPromise = null;  
 });  
   
 return mongoose.connection;  
 } catch (error) {  
 isConnecting = false;  
 connectionPromise = null;  
   
 // Detailed error logging for diagnostics  
 console.error('❌ MongoDB connection failed:');  
 console.error(`Error type: ${error.name}`);  
 console.error(`Message: ${error.message}`);  
   
 if (error.name === 'MongoServerSelectionError') {  
 console.error('This is likely due to IP whitelisting or network issues.');  
 console.error('Make sure you\'ve added 0.0.0.0/0 to your MongoDB Atlas IP whitelist.');  
 }  
   
 if (process.env.NODE\_ENV !== 'production') {  
 console.error('Stack trace:', error.stack);  
 }  
   
 // In production, we'll continue with in-memory fallback  
 console.log('Continuing with in-memory storage fallback');  
 return null;  
 }  
}  
  
// Simple user model for Vercel deployment  
const userSchema = new mongoose.Schema({  
 username: { type: String, required: true, unique: true },  
 email: { type: String, unique: true, sparse: true },  
 password: { type: String, required: true },  
 createdAt: { type: Date, default: Date.now }  
});  
  
userSchema.methods.comparePassword = async function(candidatePassword) {  
 return bcrypt.compare(candidatePassword, this.password);  
};  
  
const User = mongoose.models.User || mongoose.model('User', userSchema);  
  
// Simple link model for URL shortening  
const linkSchema = new mongoose.Schema({  
 shortCode: { type: String, required: true, unique: true },  
 originalUrl: { type: String, required: true },  
 userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },  
 createdAt: { type: Date, default: Date.now },  
 expiresAt: { type: Date, default: null },  
 clickCount: { type: Number, default: 0 },  
 customAlias: { type: String, default: null },  
 isActive: { type: Boolean, default: true }  
});  
  
const Link = mongoose.models.Link || mongoose.model('Link', linkSchema);  
  
// Simple click model for analytics  
const clickSchema = new mongoose.Schema({  
 linkId: { type: mongoose.Schema.Types.ObjectId, ref: 'Link', required: true },  
 timestamp: { type: Date, default: Date.now },  
 ipAddress: { type: String, default: null },  
 userAgent: { type: String, default: null },  
 device: { type: String, default: null },  
 browser: { type: String, default: null },  
 os: { type: String, default: null },  
 referrer: { type: String, default: null },  
 country: { type: String, default: null }  
});  
  
const Click = mongoose.models.Click || mongoose.model('Click', clickSchema);  
  
// Passport config  
passport.serializeUser((user, done) => {  
 done(null, user.\_id);  
});  
  
passport.deserializeUser(async (id, done) => {  
 try {  
 const user = await User.findById(id);  
 done(null, user);  
 } catch (err) {  
 done(err, null);  
 }  
});  
  
passport.use(new LocalStrategy(  
 async (username, password, done) => {  
 try {  
 const user = await User.findOne({  
 $or: [{ username }, { email: username }]  
 });  
   
 if (!user) {  
 return done(null, false, { message: 'Invalid credentials' });  
 }  
   
 const isMatch = await user.comparePassword(password);  
 if (!isMatch) {  
 return done(null, false, { message: 'Invalid credentials' });  
 }  
   
 return done(null, user);  
 } catch (err) {  
 return done(err);  
 }  
 }  
));  
  
// Connect to MongoDB when the app starts  
connectToMongoDB();  
  
// 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' });  
});  
  
// Standard auth endpoints (from client)  
app.post('/api/login', (req, res, next) => {  
 console.log('Login attempt received at /api/login');  
 passport.authenticate('local', (err, user, info) => {  
 if (err) return next(err);  
 if (!user) return res.status(401).json({ message: info?.message || 'Authentication failed' });  
   
 req.login(user, (err) => {  
 if (err) return next(err);  
 return res.json({  
 id: user.\_id.toString(),  
 username: user.username,  
 email: user.email  
 });  
 });  
 })(req, res, next);  
});  
  
// Vercel-specific auth endpoints  
app.post('/api/auth/login', (req, res, next) => {  
 console.log('Login attempt received at /api/auth/login');  
 passport.authenticate('local', (err, user, info) => {  
 if (err) return next(err);  
 if (!user) return res.status(401).json({ message: info?.message || 'Authentication failed' });  
   
 req.login(user, (err) => {  
 if (err) return next(err);  
 return res.json({  
 id: user.\_id.toString(),  
 username: user.username,  
 email: user.email  
 });  
 });  
 })(req, res, next);  
});  
  
app.post('/api/register', async (req, res, next) => {  
 console.log('Registration attempt received at /api/register');  
 try {  
 const { username, password, email } = req.body;  
  
 if (!username || !password) {  
 return res.status(400).json({ message: "Username and password are required" });  
 }  
  
 // Check if user already exists  
 const existingUser = await User.findOne({   
 $or: [{ username }, { email }]   
 });  
   
 if (existingUser) {  
 return res.status(400).json({ message: "Username or email already exists" });  
 }  
  
 // Hash password  
 const salt = await bcrypt.genSalt(10);  
 const hashedPassword = await bcrypt.hash(password, salt);  
  
 // Create user  
 const newUser = await User.create({  
 username,  
 password: hashedPassword,  
 email,  
 createdAt: new Date()  
 });  
  
 // Remove password from response  
 const userResponse = {  
 id: newUser.\_id.toString(),  
 username: newUser.username,  
 email: newUser.email,  
 createdAt: newUser.createdAt  
 };  
  
 req.login(newUser, (err) => {  
 if (err) return next(err);  
 res.status(201).json(userResponse);  
 });  
 } catch (error) {  
 console.error('Registration error:', error);  
 next(error);  
 }  
});  
  
app.post('/api/logout', (req, res) => {  
 req.logout((err) => {  
 if (err) return res.status(500).json({ message: 'Error logging out' });  
 return res.json({ message: 'Logged out successfully' });  
 });  
});  
  
app.post('/api/auth/logout', (req, res) => {  
 req.logout((err) => {  
 if (err) return res.status(500).json({ message: 'Error logging out' });  
 return res.json({ message: 'Logged out successfully' });  
 });  
});  
  
// Link management endpoints  
app.post('/api/links', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const { originalUrl, customAlias, expiresAt } = req.body;  
   
 if (!originalUrl) {  
 return res.status(400).json({ message: 'Original URL is required' });  
 }  
   
 // Validate URL  
 try {  
 new URL(originalUrl);  
 } catch (err) {  
 return res.status(400).json({ message: 'Invalid URL format' });  
 }  
   
 const userId = req.user.\_id;  
   
 // Generate short code or use custom alias  
 let shortCode;  
 if (customAlias) {  
 // Check if custom alias is already taken  
 const existingLink = await Link.findOne({ shortCode: customAlias });  
 if (existingLink) {  
 return res.status(400).json({ message: 'Custom alias already in use' });  
 }  
 shortCode = customAlias;  
 } else {  
 // Generate random short code  
 const generateCode = () => {  
 const chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';  
 let result = '';  
 for (let i = 0; i < 7; i++) {  
 result += chars.charAt(Math.floor(Math.random() \* chars.length));  
 }  
 return result;  
 };  
   
 // Ensure unique short code  
 let isUnique = false;  
 while (!isUnique) {  
 shortCode = generateCode();  
 const existing = await Link.findOne({ shortCode });  
 isUnique = !existing;  
 }  
 }  
   
 // Create new link  
 const link = new Link({  
 shortCode,  
 originalUrl,  
 userId,  
 expiresAt: expiresAt ? new Date(expiresAt) : null,  
 customAlias: customAlias || null  
 });  
   
 await link.save();  
   
 res.status(201).json({  
 id: link.\_id.toString(),  
 shortCode: link.shortCode,  
 originalUrl: link.originalUrl,  
 userId: link.userId.toString(),  
 createdAt: link.createdAt,  
 expiresAt: link.expiresAt,  
 clickCount: link.clickCount,  
 customAlias: link.customAlias  
 });  
 } catch (error) {  
 console.error('Error creating link:', error);  
 res.status(500).json({ message: 'Failed to create shortened URL' });  
 }  
});  
  
app.get('/api/links', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const userId = req.user.\_id;  
 const links = await Link.find({ userId });  
   
 res.json(links.map(link => ({  
 id: link.\_id.toString(),  
 shortCode: link.shortCode,  
 originalUrl: link.originalUrl,  
 userId: link.userId.toString(),  
 createdAt: link.createdAt,  
 expiresAt: link.expiresAt,  
 clickCount: link.clickCount,  
 customAlias: link.customAlias  
 })));  
 } catch (error) {  
 console.error('Error fetching links:', error);  
 res.status(500).json({ message: 'Failed to fetch links' });  
 }  
});  
  
app.delete('/api/links/:id', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const linkId = req.params.id;  
 const userId = req.user.\_id;  
   
 const link = await Link.findOne({ \_id: linkId, userId });  
 if (!link) {  
 return res.status(404).json({ message: 'Link not found or not owned by user' });  
 }  
   
 await Link.deleteOne({ \_id: linkId });  
 await Click.deleteMany({ linkId });  
   
 res.status(200).json({ message: 'Link deleted successfully' });  
 } catch (error) {  
 console.error('Error deleting link:', error);  
 res.status(500).json({ message: 'Failed to delete link' });  
 }  
});  
  
// URL redirection endpoint  
app.get('/r/:shortCode', async (req, res) => {  
 try {  
 const { shortCode } = req.params;  
 const link = await Link.findOne({ shortCode });  
   
 if (!link) {  
 return res.status(404).json({ message: 'Link not found' });  
 }  
   
 // Check if link has expired  
 if (link.expiresAt && new Date() > link.expiresAt) {  
 return res.status(410).json({ message: 'Link has expired' });  
 }  
   
 // Increment click count  
 link.clickCount += 1;  
 await link.save();  
   
 // Log click  
 const userAgent = req.headers['user-agent'];  
 const device = userAgent?.match(/Mobile|Android|iPhone|iPad|iPod/i)  
 ? userAgent.match(/iPad/i) ? 'Tablet' : 'Mobile'  
 : 'Desktop';  
   
 const browser = userAgent?.match(/Chrome/i)  
 ? 'Chrome'  
 : userAgent?.match(/Firefox/i)  
 ? 'Firefox'  
 : userAgent?.match(/Safari/i)  
 ? 'Safari'  
 : userAgent?.match(/Edge/i)  
 ? 'Edge'  
 : userAgent?.match(/MSIE|Trident/i)  
 ? 'Internet Explorer'  
 : 'Unknown';  
   
 const os = userAgent?.match(/Windows/i)  
 ? 'Windows'  
 : userAgent?.match(/Mac/i)  
 ? 'Mac'  
 : userAgent?.match(/Linux/i)  
 ? 'Linux'  
 : userAgent?.match(/Android/i)  
 ? 'Android'  
 : userAgent?.match(/iOS/i)  
 ? 'iOS'  
 : 'Unknown';  
   
 // Get client IP address  
 const ip = req.headers['x-forwarded-for'] || req.socket.remoteAddress;  
 const ipAddress = typeof ip === 'string' ? ip.split(',')[0].trim() : undefined;  
   
 // Create click record  
 const click = new Click({  
 linkId: link.\_id,  
 ipAddress: ipAddress || null,  
 userAgent: userAgent || null,  
 device,  
 browser,  
 os,  
 referrer: req.headers.referer || null  
 });  
   
 // Save click asynchronously (don't wait for it)  
 click.save().catch(err => console.error('Error saving click:', err));  
   
 // Redirect to original URL  
 res.redirect(link.originalUrl);  
 } catch (error) {  
 console.error('Error processing redirect:', error);  
 res.status(500).json({ message: 'Failed to process redirect' });  
 }  
});  
  
// Analytics endpoints  
app.get('/api/links/:id/clicks', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const linkId = req.params.id;  
 const userId = req.user.\_id;  
   
 // Verify the link belongs to the user  
 const link = await Link.findOne({ \_id: linkId, userId });  
 if (!link) {  
 return res.status(404).json({ message: 'Link not found or not owned by user' });  
 }  
   
 // Get all clicks for the link  
 const clicks = await Click.find({ linkId });  
   
 res.json(clicks.map(click => ({  
 id: click.\_id.toString(),  
 linkId: click.linkId.toString(),  
 timestamp: click.timestamp,  
 ipAddress: click.ipAddress,  
 userAgent: click.userAgent,  
 device: click.device,  
 browser: click.browser,  
 os: click.os,  
 referrer: click.referrer,  
 country: click.country  
 })));  
 } catch (error) {  
 console.error('Error fetching clicks:', error);  
 res.status(500).json({ message: 'Failed to fetch click data' });  
 }  
});  
  
app.get('/api/links/:id/stats/clicks', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const linkId = req.params.id;  
 const userId = req.user.\_id;  
   
 // Verify the link belongs to the user  
 const link = await Link.findOne({ \_id: linkId, userId });  
 if (!link) {  
 return res.status(404).json({ message: 'Link not found or not owned by user' });  
 }  
   
 // Aggregate clicks by date  
 const clickStats = await Click.aggregate([  
 { $match: { linkId: link.\_id } },  
 {  
 $group: {  
 \_id: {  
 $dateToString: { format: "%Y-%m-%d", date: "$timestamp" }  
 },  
 count: { $sum: 1 }  
 }  
 },  
 { $sort: { "\_id": 1 } }  
 ]);  
   
 // Format the results  
 const stats = clickStats.map(item => ({  
 date: item.\_id,  
 count: item.count  
 }));  
   
 res.json(stats);  
 } catch (error) {  
 console.error('Error fetching click stats:', error);  
 res.status(500).json({ message: 'Failed to fetch click statistics' });  
 }  
});  
  
app.get('/api/links/:id/stats/devices', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const linkId = req.params.id;  
 const userId = req.user.\_id;  
   
 // Verify the link belongs to the user  
 const link = await Link.findOne({ \_id: linkId, userId });  
 if (!link) {  
 return res.status(404).json({ message: 'Link not found or not owned by user' });  
 }  
   
 // Aggregate clicks by device  
 const deviceStats = await Click.aggregate([  
 { $match: { linkId: link.\_id } },  
 {  
 $group: {  
 \_id: "$device",  
 count: { $sum: 1 }  
 }  
 }  
 ]);  
   
 // Calculate total for percentage  
 const total = deviceStats.reduce((sum, item) => sum + item.count, 0);  
   
 // Format the results  
 const stats = deviceStats.map(item => ({  
 device: item.\_id || 'Unknown',  
 count: item.count,  
 percentage: total > 0 ? Math.round((item.count / total) \* 100) : 0  
 }));  
   
 res.json(stats);  
 } catch (error) {  
 console.error('Error fetching device stats:', error);  
 res.status(500).json({ message: 'Failed to fetch device statistics' });  
 }  
});  
  
app.get('/api/dashboard/stats', async (req, res) => {  
 try {  
 if (!req.isAuthenticated()) {  
 return res.status(401).json({ message: 'Authentication required' });  
 }  
   
 const userId = req.user.\_id;  
   
 // Find all links by user  
 const links = await Link.find({ userId });  
   
 // Calculate statistics  
 const now = new Date();  
 const expirationThreshold = new Date();  
 expirationThreshold.setDate(now.getDate() + 7); // Links expiring in 7 days  
   
 const totalLinks = links.length;  
 const totalClicks = links.reduce((sum, link) => sum + (link.clickCount || 0), 0);  
 const activeLinks = links.filter(link =>   
 (!link.expiresAt || link.expiresAt > now)  
 ).length;  
 const expiringLinks = links.filter(link =>   
 link.expiresAt && link.expiresAt > now && link.expiresAt < expirationThreshold  
 ).length;  
   
 res.json({  
 totalLinks,  
 totalClicks,  
 activeLinks,  
 expiringLinks  
 });  
 } catch (error) {  
 console.error('Error fetching dashboard stats:', error);  
 res.status(500).json({ message: 'Failed to fetch dashboard statistics' });  
 }  
});  
  
// Health check endpoint  
app.get('/api/health', (req, res) => {  
 res.status(200).json({  
 status: 'ok',  
 timestamp: new Date().toISOString(),  
 environment: process.env.NODE\_ENV || 'development',  
 mongodb: !!mongoose.connection.readyState  
 });  
});  
  
// Error handling middleware  
app.use((err, req, res, next) => {  
 console.error('Unhandled error:', err);  
   
 // Detailed error logging for diagnostics  
 if (err) {  
 console.error(`Error type: ${err.name || 'Unknown'}`);  
 console.error(`Message: ${err.message || 'No message'}`);  
   
 if (err.code) {  
 console.error(`Error code: ${err.code}`);  
 }  
   
 if (process.env.NODE\_ENV !== 'production') {  
 console.error('Stack trace:', err.stack);  
 }  
 }  
   
 // MongoDB connection errors  
 if (err.name === 'MongoServerSelectionError' ||   
 err.name === 'MongoNetworkError' ||  
 err.name === 'MongooseServerSelectionError') {  
 return res.status(503).json({  
 status: 'error',  
 message: 'Database connectivity issue. Please try again later.',  
 error: process.env.NODE\_ENV === 'production' ? 'Database error' : err.message  
 });  
 }  
   
 // Validation errors  
 if (err.name === 'ValidationError') {  
 return res.status(400).json({  
 status: 'error',  
 message: 'Invalid data provided',  
 error: process.env.NODE\_ENV === 'production' ? 'Validation failed' : err.message  
 });  
 }  
   
 // Default error response  
 res.status(err.status || 500).json({  
 status: 'error',  
 message: process.env.NODE\_ENV === 'production'   
 ? 'An unexpected error occurred'   
 : err.message || 'Unknown error',  
 });  
});  
  
// Vercel needs to handle both API routes and static files  
if (process.env.NODE\_ENV === 'production') {  
 // Serve static files - check both possible static file locations  
 const primaryDistPath = path.join(\_\_dirname, '..', 'dist');  
 const fallbackDistPath = path.join(\_\_dirname, '..', 'dist', 'public');  
   
 // First check if primary dist path exists  
 try {  
 if (require('fs').existsSync(path.join(primaryDistPath, 'index.html'))) {  
 console.log('Using primary dist path:', primaryDistPath);  
 app.use(express.static(primaryDistPath));  
 } else if (require('fs').existsSync(path.join(fallbackDistPath, 'index.html'))) {  
 console.log('Using fallback dist path:', fallbackDistPath);  
 app.use(express.static(fallbackDistPath));  
 } else {  
 console.log('No static files found in either location');  
 }  
 } catch (err) {  
 console.error('Error checking static file paths:', err);  
 }  
   
 // Handle all other routes by serving the index.html  
 app.get('\*', (req, res) => {  
 if (req.path.startsWith('/api/')) {  
 return res.status(404).json({ message: 'API endpoint not found' });  
 }  
   
 // Try to send the index.html from the appropriate path  
 try {  
 if (require('fs').existsSync(path.join(primaryDistPath, 'index.html'))) {  
 return res.sendFile(path.join(primaryDistPath, 'index.html'));  
 } else if (require('fs').existsSync(path.join(fallbackDistPath, 'index.html'))) {  
 return res.sendFile(path.join(fallbackDistPath, 'index.html'));  
 } else {  
 return res.status(404).send('Frontend build not found');  
 }  
 } catch (err) {  
 console.error('Error serving index.html:', err);  
 return res.status(500).send('Error serving application');  
 }  
 });  
}  
  
// Handle server startup when running directly (not as a Vercel function)  
if (process.env.START\_SERVER === 'true') {  
 const PORT = process.env.PORT || 5000;  
 app.listen(PORT, '0.0.0.0', () => {  
 console.log(`Server running on port ${PORT}`);  
 });  
}  
  
export default app;

build.js

import { build } from 'vite'  
import { fileURLToPath } from 'url'  
import { dirname, resolve } from 'path'  
  
const \_\_filename = fileURLToPath(import.meta.url)  
const \_\_dirname = dirname(\_\_filename)  
  
async function buildApp() {  
 // Build the client  
 await build({  
 root: resolve(\_\_dirname, 'client'),  
 build: {  
 outDir: resolve(\_\_dirname, 'dist/public'),  
 emptyOutDir: true  
 }  
 })  
}  
  
buildApp()

client/index.html

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1" />  
 </head>  
 <body>  
 <div id="root"></div>  
 <script type="module" src="/src/main.tsx"></script>  
 <!-- This script injects a replit badge into the page, please feel free to remove this line -->  
 <script type="text/javascript" src="https://replit.com/public/js/replit-badge-v3.js"></script>  
 </body>  
</html>

client/src/App.tsx

import { Switch, Route } from "wouter";  
import { queryClient } from "./lib/queryClient";  
import { QueryClientProvider } from "@tanstack/react-query";  
import { Toaster } from "@/components/ui/toaster";  
import NotFound from "@/pages/not-found";  
import { AuthProvider } from "@/hooks/use-auth";  
import { ProtectedRoute } from "./lib/protected-route";  
import Dashboard from "@/pages/dashboard";  
import AuthPage from "@/pages/auth-page";  
import RedirectPage from "@/pages/redirect";  
  
function Router() {  
 return (  
 <Switch>  
 <Route path="/auth" component={AuthPage} />  
 <Route path="/r/:shortCode" component={RedirectPage} />  
 <ProtectedRoute path="/" component={Dashboard} />  
 <ProtectedRoute path="/links" component={Dashboard} />  
 <ProtectedRoute path="/create" component={Dashboard} />  
 <ProtectedRoute path="/qr-codes" component={Dashboard} />  
 <ProtectedRoute path="/settings" component={Dashboard} />  
 <Route component={NotFound} />  
 </Switch>  
 );  
}  
  
function App() {  
 return (  
 <QueryClientProvider client={queryClient}>  
 <AuthProvider>  
 <Router />  
 <Toaster />  
 </AuthProvider>  
 </QueryClientProvider>  
 );  
}  
  
export default App;

client/src/components/charts/clicks-chart.tsx

import { useState } from "react";  
import { useQuery } from "@tanstack/react-query";  
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";  
import { Skeleton } from "@/components/ui/skeleton";  
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";  
import { LinkWithAnalytics, ClickStat } from "@shared/schema";  
import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from "recharts";  
  
interface ClicksChartProps {  
 className?: string;  
}  
  
export default function ClicksChart({ className }: ClicksChartProps) {  
 const [timeRange, setTimeRange] = useState("7");  
   
 // Fetch links with click stats  
 const { data: links, isLoading } = useQuery<LinkWithAnalytics[]>({  
 queryKey: ["/api/links"],  
 });  
  
 // Prepare data for chart  
 const prepareChartData = () => {  
 if (!links || links.length === 0) return [];  
   
 // Combine all link click stats  
 const allClicksData: ClickStat[] = [];  
   
 links.forEach(link => {  
 // Skip links without click stats  
 if (!link.clickStats) return;  
   
 link.clickStats.forEach(stat => {  
 const existingIndex = allClicksData.findIndex(s => s.date === stat.date);  
 if (existingIndex >= 0) {  
 // Add to existing date  
 allClicksData[existingIndex].count += stat.count;  
 } else {  
 // Add new date entry  
 allClicksData.push({ ...stat });  
 }  
 });  
 });  
   
 // Sort by date  
 return allClicksData.sort((a, b) => new Date(a.date).getTime() - new Date(b.date).getTime());  
 };  
   
 const chartData = prepareChartData();  
   
 // Format date for x-axis  
 const formatXAxis = (dateStr: string) => {  
 const date = new Date(dateStr);  
 return date.toLocaleDateString('en-US', { weekday: 'short' });  
 };  
   
 // Custom tooltip for chart  
 const CustomTooltip = ({ active, payload, label }: any) => {  
 if (active && payload && payload.length) {  
 return (  
 <div className="bg-background border border-border p-2 rounded shadow-sm">  
 <p className="font-medium">{new Date(label).toLocaleDateString()}</p>  
 <p className="text-primary">Clicks: {payload[0].value}</p>  
 </div>  
 );  
 }  
 return null;  
 };  
  
 return (  
 <Card className={className}>  
 <CardHeader className="flex flex-row items-center justify-between">  
 <CardTitle>Clicks Over Time</CardTitle>  
 <Select  
 value={timeRange}  
 onValueChange={setTimeRange}  
 >  
 <SelectTrigger className="w-[150px]">  
 <SelectValue placeholder="Select range" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="7">Last 7 days</SelectItem>  
 <SelectItem value="30">Last 30 days</SelectItem>  
 <SelectItem value="90">Last 90 days</SelectItem>  
 </SelectContent>  
 </Select>  
 </CardHeader>  
 <CardContent>  
 {isLoading ? (  
 <div className="h-[300px] flex items-center justify-center">  
 <Skeleton className="h-[250px] w-full" />  
 </div>  
 ) : chartData.length === 0 ? (  
 <div className="h-[300px] flex items-center justify-center">  
 <p className="text-muted-foreground">No click data available yet</p>  
 </div>  
 ) : (  
 <div className="h-[300px]">  
 <ResponsiveContainer width="100%" height="100%">  
 <BarChart  
 data={chartData}  
 margin={{ top: 10, right: 10, left: 0, bottom: 30 }}  
 >  
 <CartesianGrid strokeDasharray="3 3" vertical={false} stroke="var(--border)" />  
 <XAxis   
 dataKey="date"   
 tick={{ fill: 'var(--muted-foreground)' }}   
 tickFormatter={formatXAxis}  
 axisLine={{ stroke: 'var(--border)' }}  
 />  
 <YAxis   
 tick={{ fill: 'var(--muted-foreground)' }}   
 axisLine={{ stroke: 'var(--border)' }}  
 />  
 <Tooltip content={<CustomTooltip />} />  
 <Bar  
 dataKey="count"  
 name="Clicks"  
 fill="hsl(var(--primary))"  
 radius={[4, 4, 0, 0]}  
 />  
 </BarChart>  
 </ResponsiveContainer>  
 </div>  
 )}  
 </CardContent>  
 </Card>  
 );  
}

client/src/components/charts/devices-chart.tsx

import { useQuery } from "@tanstack/react-query";  
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";  
import { Skeleton } from "@/components/ui/skeleton";  
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";  
import { LinkWithAnalytics, DeviceStat } from "@shared/schema";  
import { PieChart, Pie, Cell, ResponsiveContainer, Legend, Tooltip } from "recharts";  
import { useState } from "react";  
  
export default function DevicesChart() {  
 const [selectedView, setSelectedView] = useState("all");  
   
 // Fetch links with device stats  
 const { data: links, isLoading } = useQuery<LinkWithAnalytics[]>({  
 queryKey: ["/api/links"],  
 });  
   
 // Prepare data for chart  
 const prepareChartData = () => {  
 if (!links || links.length === 0) return [];  
   
 // Combine all device stats  
 const deviceCounts: Record<string, number> = {};  
   
 links.forEach(link => {  
 // Skip links without device stats  
 if (!link.deviceStats) return;  
   
 link.deviceStats.forEach(stat => {  
 deviceCounts[stat.device] = (deviceCounts[stat.device] || 0) + stat.count;  
 });  
 });  
   
 // Convert to array for pie chart  
 const chartData = Object.entries(deviceCounts).map(([device, count]) => ({  
 device,  
 count  
 }));  
   
 // Calculate percentages  
 const total = chartData.reduce((sum, item) => sum + item.count, 0);  
   
 return chartData.map(item => ({  
 ...item,  
 percentage: total > 0 ? Math.round((item.count / total) \* 100) : 0  
 }));  
 };  
   
 const deviceData = prepareChartData();  
   
 // Colors for the pie chart  
 const COLORS = [  
 'hsl(var(--primary))',  
 'hsl(var(--secondary))',  
 'hsl(var(--accent))',  
 'hsl(var(--chart-1))',  
 'hsl(var(--chart-2))'  
 ];  
   
 // Custom tooltip for chart  
 const CustomTooltip = ({ active, payload }: any) => {  
 if (active && payload && payload.length) {  
 return (  
 <div className="bg-background border border-border p-2 rounded shadow-sm">  
 <p className="font-medium">{payload[0].name}</p>  
 <p className="text-primary">{payload[0].value} clicks ({payload[0].payload.percentage}%)</p>  
 </div>  
 );  
 }  
 return null;  
 };  
  
 return (  
 <Card>  
 <CardHeader className="flex flex-row items-center justify-between">  
 <CardTitle>Devices</CardTitle>  
 <Select  
 value={selectedView}  
 onValueChange={setSelectedView}  
 >  
 <SelectTrigger className="w-[150px]">  
 <SelectValue placeholder="All links" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="all">All links</SelectItem>  
 <SelectItem value="popular">Popular links</SelectItem>  
 </SelectContent>  
 </Select>  
 </CardHeader>  
 <CardContent>  
 {isLoading ? (  
 <div className="h-[260px] flex items-center justify-center">  
 <Skeleton className="h-[200px] w-[200px] rounded-full" />  
 </div>  
 ) : deviceData.length === 0 ? (  
 <div className="h-[260px] flex items-center justify-center">  
 <p className="text-muted-foreground">No device data available yet</p>  
 </div>  
 ) : (  
 <div className="h-[260px]">  
 <ResponsiveContainer width="100%" height="100%">  
 <PieChart>  
 <Pie  
 data={deviceData}  
 cx="50%"  
 cy="50%"  
 innerRadius={60}  
 outerRadius={80}  
 fill="#8884d8"  
 paddingAngle={5}  
 dataKey="count"  
 nameKey="device"  
 label={({ device, percentage }) => `${percentage}%`}  
 >  
 {deviceData.map((entry, index) => (  
 <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />  
 ))}  
 </Pie>  
 <Tooltip content={<CustomTooltip />} />  
 </PieChart>  
 </ResponsiveContainer>  
 </div>  
 )}  
   
 <div className="grid grid-cols-3 gap-2 mt-4">  
 {deviceData.map((item, index) => (  
 <div key={index} className="flex items-center space-x-2">  
 <span   
 className="w-3 h-3 rounded-full"   
 style={{ backgroundColor: COLORS[index % COLORS.length] }}  
 ></span>  
 <span className="text-sm">{item.device}</span>  
 </div>  
 ))}  
 </div>  
 </CardContent>  
 </Card>  
 );  
}

client/src/components/create-link-form.tsx

import { useState } from "react";  
import { useForm } from "react-hook-form";  
import { zodResolver } from "@hookform/resolvers/zod";  
import { z } from "zod";  
import { useMutation } from "@tanstack/react-query";  
import { apiRequest, queryClient } from "@/lib/queryClient";  
import { useToast } from "@/hooks/use-toast";  
  
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";  
import { Form, FormControl, FormDescription, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";  
import { Input } from "@/components/ui/input";  
import { Button } from "@/components/ui/button";  
import { Link2Icon } from "lucide-react";  
import { insertLinkSchema } from "@shared/schema";  
  
// Form schema for create link form  
const createLinkSchema = z.object({  
 originalUrl: z.string()  
 .url("Please enter a valid URL including http:// or https://")  
 .min(1, "URL is required"),  
 customAlias: z.string().optional(),  
 expiresAt: z.string().optional(),  
});  
  
type CreateLinkFormValues = z.infer<typeof createLinkSchema>;  
  
export default function CreateLinkForm() {  
 const { toast } = useToast();  
 const [customAliasAvailable, setCustomAliasAvailable] = useState<boolean | null>(null);  
  
 const form = useForm<CreateLinkFormValues>({  
 resolver: zodResolver(createLinkSchema),  
 defaultValues: {  
 originalUrl: "",  
 customAlias: "",  
 expiresAt: "",  
 },  
 });  
  
 const createLinkMutation = useMutation({  
 mutationFn: async (values: CreateLinkFormValues) => {  
 const res = await apiRequest("POST", "/api/links", values);  
 return res.json();  
 },  
 onSuccess: () => {  
 toast({  
 title: "Link created successfully",  
 description: "Your shortened link is now ready to use.",  
 });  
 form.reset();  
 setCustomAliasAvailable(null);  
   
 // Invalidate queries to refresh the links list and stats  
 queryClient.invalidateQueries({ queryKey: ["/api/links"] });  
 queryClient.invalidateQueries({ queryKey: ["/api/dashboard/stats"] });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Failed to create link",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 const onSubmit = (values: CreateLinkFormValues) => {  
 createLinkMutation.mutate(values);  
 };  
  
 // Check if custom alias contains spaces or special chars  
 const validateCustomAlias = (value: string) => {  
 if (!value) {  
 setCustomAliasAvailable(null);  
 return true;  
 }  
   
 // Check for spaces and special characters except dash and underscore  
 const isValid = /^[a-zA-Z0-9\_-]+$/.test(value);  
 setCustomAliasAvailable(isValid);  
 return isValid || "Custom alias can only contain letters, numbers, dashes and underscores";  
 };  
  
 return (  
 <Card className="mb-6">  
 <CardHeader>  
 <CardTitle>Create New Short Link</CardTitle>  
 </CardHeader>  
 <CardContent>  
 <Form {...form}>  
 <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">  
 <div className="grid grid-cols-1 md:grid-cols-4 gap-4">  
 <FormField  
 control={form.control}  
 name="originalUrl"  
 render={({ field }) => (  
 <FormItem className="md:col-span-2">  
 <FormLabel>Long URL</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="https://example.com/your-long-url-goes-here"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <FormField  
 control={form.control}  
 name="customAlias"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Custom Alias (Optional)</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="my-custom-name"   
 {...field}   
 onChange={(e) => {  
 field.onChange(e);  
 validateCustomAlias(e.target.value);  
 }}  
 />  
 </FormControl>  
 {customAliasAvailable === false && (  
 <FormDescription className="text-destructive">  
 Only letters, numbers, dashes and underscores allowed  
 </FormDescription>  
 )}  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
  
 <FormField  
 control={form.control}  
 name="expiresAt"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Expiration (Optional)</FormLabel>  
 <FormControl>  
 <Input   
 type="date"   
 {...field}   
 min={new Date().toISOString().split('T')[0]}  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
 </div>  
  
 <div className="flex justify-end">  
 <Button   
 type="submit"  
 disabled={createLinkMutation.isPending || customAliasAvailable === false}  
 className="flex items-center gap-2"  
 >  
 {createLinkMutation.isPending ? (  
 <>  
 <span className="h-4 w-4 animate-spin rounded-full border-2 border-background border-t-transparent"></span>  
 Creating...  
 </>  
 ) : (  
 <>  
 <Link2Icon className="h-4 w-4" />  
 Create Short Link  
 </>  
 )}  
 </Button>  
 </div>  
 </form>  
 </Form>  
 </CardContent>  
 </Card>  
 );  
}

client/src/components/layout/header.tsx

import { useState, useEffect } from "react";  
import { Button } from "@/components/ui/button";  
import { Input } from "@/components/ui/input";  
import {   
 Menu,   
 Search,   
 BellIcon,   
 SunIcon,   
 MoonIcon   
} from "lucide-react";  
  
interface HeaderProps {  
 onMenuClick: () => void;  
 searchQuery: string;  
 onSearchChange: (query: string) => void;  
}  
  
export default function Header({ onMenuClick, searchQuery, onSearchChange }: HeaderProps) {  
 const [theme, setTheme] = useState<"light" | "dark">("light");  
   
 // Initialize theme from localStorage or system preference  
 useEffect(() => {  
 const storedTheme = localStorage.getItem("theme");  
 if (storedTheme === "dark" ||   
 (!storedTheme && window.matchMedia("(prefers-color-scheme: dark)").matches)) {  
 setTheme("dark");  
 document.documentElement.classList.add("dark");  
 } else {  
 setTheme("light");  
 document.documentElement.classList.remove("dark");  
 }  
 }, []);  
   
 // Toggle theme  
 const toggleTheme = () => {  
 const newTheme = theme === "light" ? "dark" : "light";  
 setTheme(newTheme);  
   
 if (newTheme === "dark") {  
 document.documentElement.classList.add("dark");  
 localStorage.setItem("theme", "dark");  
 } else {  
 document.documentElement.classList.remove("dark");  
 localStorage.setItem("theme", "light");  
 }  
 };  
  
 return (  
 <header className="bg-background border-b border-border sticky top-0 z-10">  
 <div className="flex items-center justify-between px-4 py-3">  
 {/\* Mobile menu button \*/}  
 <div className="flex items-center lg:hidden">  
 <Button variant="ghost" size="icon" onClick={onMenuClick}>  
 <Menu className="h-5 w-5" />  
 <span className="sr-only">Open menu</span>  
 </Button>  
 </div>  
  
 {/\* Search \*/}  
 <div className="flex items-center space-x-3 ml-auto mr-auto lg:ml-0">  
 <div className="relative">  
 <Search className="absolute left-3 top-1/2 -translate-y-1/2 h-4 w-4 text-muted-foreground" />  
 <Input  
 type="text"  
 placeholder="Search links..."  
 className="pl-9 pr-4 py-2 w-full lg:w-64"  
 value={searchQuery}  
 onChange={(e) => onSearchChange(e.target.value)}  
 />  
 </div>  
 </div>  
  
 {/\* Right side actions \*/}  
 <div className="flex items-center space-x-3">  
 <Button   
 variant="ghost"   
 size="icon"   
 onClick={toggleTheme}   
 aria-label="Toggle dark mode"  
 >  
 {theme === 'dark' ? (  
 <SunIcon className="h-5 w-5" />  
 ) : (  
 <MoonIcon className="h-5 w-5" />  
 )}  
 </Button>  
   
 <Button   
 variant="ghost"   
 size="icon"   
 aria-label="Notifications"  
 >  
 <BellIcon className="h-5 w-5" />  
 </Button>  
 </div>  
 </div>  
 </header>  
 );  
}

client/src/components/layout/sidebar.tsx

import { useLocation } from "wouter";  
import { User } from "@shared/schema";  
import { useAuth } from "@/hooks/use-auth";  
import { cn } from "@/lib/utils";  
import { Button } from "@/components/ui/button";  
import { ScrollArea } from "@/components/ui/scroll-area";  
import { Avatar, AvatarFallback } from "@/components/ui/avatar";  
import {   
 LayoutDashboard,   
 Link as LinkIcon,   
 PlusCircle,   
 QrCode,   
 Settings,   
 LogOut,  
 X  
} from "lucide-react";  
  
interface SidebarProps {  
 user: User | null;  
 isOpen: boolean;  
 onClose: () => void;  
}  
  
export default function Sidebar({ user, isOpen, onClose }: SidebarProps) {  
 const [location, setLocation] = useLocation();  
 const { logoutMutation } = useAuth();  
   
 const getInitials = (name: string) => {  
 return name.split('@')[0].substring(0, 2).toUpperCase();  
 };  
   
 const handleLogout = async () => {  
 await logoutMutation.mutateAsync();  
 setLocation("/auth");  
 };  
   
 const navigationItems = [  
 {  
 name: "Dashboard",  
 href: "/",  
 icon: <LayoutDashboard className="h-5 w-5" />,  
 },  
 {  
 name: "My Links",  
 href: "/links",  
 icon: <LinkIcon className="h-5 w-5" />,  
 },  
 {  
 name: "Create Link",  
 href: "/create",  
 icon: <PlusCircle className="h-5 w-5" />,  
 },  
 {  
 name: "QR Codes",  
 href: "/qr-codes",  
 icon: <QrCode className="h-5 w-5" />,  
 },  
 {  
 name: "Settings",  
 href: "/settings",  
 icon: <Settings className="h-5 w-5" />,  
 },  
 ];  
  
 return (  
 <>  
 {/\* Mobile overlay \*/}  
 {isOpen && (  
 <div   
 className="fixed inset-0 bg-black/50 backdrop-blur-sm z-40 lg:hidden"  
 onClick={onClose}  
 />  
 )}  
   
 {/\* Sidebar \*/}  
 <aside   
 className={cn(  
 "fixed inset-y-0 left-0 z-50 w-64 bg-background border-r border-border transform transition-transform duration-300 lg:translate-x-0",  
 isOpen ? "translate-x-0" : "-translate-x-full"  
 )}  
 >  
 <div className="flex flex-col h-full">  
 {/\* Logo and mobile close button \*/}  
 <div className="p-4 border-b border-border flex items-center justify-between">  
 <div className="flex items-center space-x-3">  
 <svg className="w-8 h-8 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>  
   
 <button   
 className="lg:hidden rounded-sm opacity-70 hover:opacity-100 focus:outline-none"   
 onClick={onClose}  
 >  
 <X className="h-5 w-5" />  
 <span className="sr-only">Close sidebar</span>  
 </button>  
 </div>  
  
 {/\* Navigation \*/}  
 <ScrollArea className="flex-1 py-2">  
 <nav className="px-2">  
 <ul className="space-y-1">  
 {navigationItems.map((item) => (  
 <li key={item.href}>  
 <Button  
 variant={item.href === location ? "secondary" : "ghost"}  
 className={cn(  
 "w-full justify-start text-base",  
 item.href === location   
 ? "bg-primary/10 text-primary hover:bg-primary/20"  
 : "hover:bg-gray-100 dark:hover:bg-gray-800"  
 )}  
 onClick={() => setLocation(item.href)}  
 >  
 {item.icon}  
 <span className="ml-3">{item.name}</span>  
 </Button>  
 </li>  
 ))}  
 </ul>  
 </nav>  
 </ScrollArea>  
  
 {/\* User profile \*/}  
 <div className="p-4 border-t border-border">  
 <div className="flex items-center space-x-3">  
 <Avatar>  
 <AvatarFallback className="bg-primary/10 text-primary">  
 {user ? getInitials(user.username) : "UN"}  
 </AvatarFallback>  
 </Avatar>  
 <div className="flex-1 min-w-0">  
 <p className="text-sm font-medium truncate">  
 {user?.username || "User"}  
 </p>  
 <p className="text-xs text-muted-foreground truncate">  
 {user?.email || user?.username || "Unknown"}  
 </p>  
 </div>  
 <Button  
 variant="ghost"  
 size="icon"  
 onClick={handleLogout}  
 disabled={logoutMutation.isPending}  
 title="Logout"  
 >  
 <LogOut className="h-5 w-5 text-muted-foreground" />  
 <span className="sr-only">Logout</span>  
 </Button>  
 </div>  
 </div>  
 </div>  
 </aside>  
 </>  
 );  
}

client/src/components/links-table.tsx

import { useState } from "react";  
import { useMutation } from "@tanstack/react-query";  
import { Link, LinkWithAnalytics } from "@shared/schema";  
import { apiRequest, queryClient } from "@/lib/queryClient";  
import { useToast } from "@/hooks/use-toast";  
  
import {  
 Card,  
 CardContent,  
 CardHeader,  
 CardTitle,  
 CardFooter,  
} from "@/components/ui/card";  
import {  
 Table,  
 TableBody,  
 TableCell,  
 TableHead,  
 TableHeader,  
 TableRow,  
} from "@/components/ui/table";  
import {  
 Select,  
 SelectContent,  
 SelectItem,  
 SelectTrigger,  
 SelectValue,  
} from "@/components/ui/select";  
import { Button } from "@/components/ui/button";  
import { Badge } from "@/components/ui/badge";  
import { Skeleton } from "@/components/ui/skeleton";  
import {  
 Copy,  
 BarChart2,  
 QrCode,  
 Trash2,  
 ChevronRight,  
 ChevronLeft,  
 Download,  
} from "lucide-react";  
import {  
 Pagination,  
 PaginationContent,  
 PaginationEllipsis,  
 PaginationItem,  
 PaginationLink,  
 PaginationNext,  
 PaginationPrevious,  
} from "@/components/ui/pagination";  
import {  
 Tooltip,  
 TooltipContent,  
 TooltipProvider,  
 TooltipTrigger,  
} from "@/components/ui/tooltip";  
import {  
 AlertDialog,  
 AlertDialogAction,  
 AlertDialogCancel,  
 AlertDialogContent,  
 AlertDialogDescription,  
 AlertDialogFooter,  
 AlertDialogHeader,  
 AlertDialogTitle,  
 AlertDialogTrigger,  
} from "@/components/ui/alert-dialog";  
  
interface LinksTableProps {  
 links: LinkWithAnalytics[];  
 isLoading: boolean;  
 searchQuery: string;  
 onQrCodeClick: (link: LinkWithAnalytics) => void;  
}  
  
export default function LinksTable({  
 links,  
 isLoading,  
 searchQuery,  
 onQrCodeClick,  
}: LinksTableProps) {  
 const { toast } = useToast();  
 const [filter, setFilter] = useState<string>("all");  
 const [page, setPage] = useState(1);  
 const [linkToDelete, setLinkToDelete] = useState<LinkWithAnalytics | null>(null);  
 const itemsPerPage = 5;  
  
 // Delete link mutation  
 const deleteLinkMutation = useMutation({  
 mutationFn: async (linkId: number) => {  
 await apiRequest("DELETE", `/api/links/${linkId}`);  
 },  
 onSuccess: () => {  
 toast({  
 title: "Link deleted",  
 description: "The link has been successfully deleted.",  
 });  
   
 // Invalidate queries to refresh the links list and stats  
 queryClient.invalidateQueries({ queryKey: ["/api/links"] });  
 queryClient.invalidateQueries({ queryKey: ["/api/dashboard/stats"] });  
   
 // Close dialog  
 setLinkToDelete(null);  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Failed to delete link",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 // Filter links based on filter and search query  
 const filteredLinks = links  
 .filter((link) => {  
 if (filter === "active") {  
 return !link.isExpired;  
 } else if (filter === "expired") {  
 return link.isExpired;  
 } else if (filter === "expiring") {  
 return link.isExpiringSoon;  
 }  
 return true;  
 })  
 .filter((link) => {  
 if (!searchQuery) return true;  
 const query = searchQuery.toLowerCase();  
 return (  
 link.originalUrl.toLowerCase().includes(query) ||  
 link.shortCode.toLowerCase().includes(query) ||  
 (link.customAlias && link.customAlias.toLowerCase().includes(query))  
 );  
 });  
  
 // Calculate pagination  
 const totalPages = Math.ceil(filteredLinks.length / itemsPerPage);  
 const paginatedLinks = filteredLinks.slice(  
 (page - 1) \* itemsPerPage,  
 page \* itemsPerPage  
 );  
  
 // Copy link to clipboard  
 const copyToClipboard = (shortCode: string) => {  
 const fullUrl = `${window.location.origin}/r/${shortCode}`;  
 navigator.clipboard.writeText(fullUrl).then(  
 () => {  
 toast({  
 title: "Copied to clipboard",  
 description: "Link has been copied to your clipboard.",  
 });  
 },  
 (err) => {  
 console.error("Could not copy text: ", err);  
 toast({  
 title: "Failed to copy",  
 description: "Could not copy link to clipboard.",  
 variant: "destructive",  
 });  
 }  
 );  
 };  
  
 // Format date  
 const formatDate = (dateString: Date | null | undefined) => {  
 if (!dateString) return "N/A";  
 return new Date(dateString).toLocaleDateString();  
 };  
  
 // Get status badge for link  
 const getStatusBadge = (link: LinkWithAnalytics) => {  
 if (link.isExpired) {  
 return (  
 <Badge variant="destructive">Expired</Badge>  
 );  
 } else if (link.isExpiringSoon) {  
 return (  
 <Badge variant="outline" className="bg-amber-100 text-amber-800 dark:bg-amber-900/30 dark:text-amber-400 border-amber-400">  
 Expires Soon  
 </Badge>  
 );  
 } else {  
 return (  
 <Badge variant="outline" className="bg-green-100 text-green-800 dark:bg-green-900/30 dark:text-green-400 border-green-400">  
 Active  
 </Badge>  
 );  
 }  
 };  
  
 return (  
 <Card>  
 <CardHeader className="flex flex-col md:flex-row md:items-center md:justify-between gap-4">  
 <CardTitle>Recent Links</CardTitle>  
 <div className="flex items-center space-x-2">  
 <Select value={filter} onValueChange={setFilter}>  
 <SelectTrigger className="w-[180px]">  
 <SelectValue placeholder="Filter links" />  
 </SelectTrigger>  
 <SelectContent>  
 <SelectItem value="all">All links</SelectItem>  
 <SelectItem value="active">Active links</SelectItem>  
 <SelectItem value="expired">Expired links</SelectItem>  
 <SelectItem value="expiring">Expiring soon</SelectItem>  
 </SelectContent>  
 </Select>  
 <Button variant="outline" size="sm" className="flex items-center gap-2">  
 <Download className="h-4 w-4" />  
 <span>Export</span>  
 </Button>  
 </div>  
 </CardHeader>  
 <CardContent className="p-0">  
 <div className="overflow-x-auto">  
 <Table>  
 <TableHeader>  
 <TableRow>  
 <TableHead>Link details</TableHead>  
 <TableHead>Created</TableHead>  
 <TableHead>Clicks</TableHead>  
 <TableHead>Status</TableHead>  
 <TableHead>Actions</TableHead>  
 </TableRow>  
 </TableHeader>  
 <TableBody>  
 {isLoading ? (  
 Array(5)  
 .fill(0)  
 .map((\_, index) => (  
 <TableRow key={index}>  
 <TableCell>  
 <div className="flex flex-col gap-1">  
 <Skeleton className="h-5 w-36" />  
 <Skeleton className="h-4 w-48" />  
 </div>  
 </TableCell>  
 <TableCell>  
 <Skeleton className="h-4 w-24" />  
 </TableCell>  
 <TableCell>  
 <Skeleton className="h-4 w-12" />  
 </TableCell>  
 <TableCell>  
 <Skeleton className="h-6 w-20" />  
 </TableCell>  
 <TableCell>  
 <div className="flex items-center space-x-2">  
 <Skeleton className="h-8 w-8 rounded-full" />  
 <Skeleton className="h-8 w-8 rounded-full" />  
 <Skeleton className="h-8 w-8 rounded-full" />  
 <Skeleton className="h-8 w-8 rounded-full" />  
 </div>  
 </TableCell>  
 </TableRow>  
 ))  
 ) : paginatedLinks.length === 0 ? (  
 <TableRow>  
 <TableCell colSpan={5} className="text-center py-8 text-muted-foreground">  
 {searchQuery  
 ? "No links found matching your search query"  
 : "No links created yet. Create your first short link above!"}  
 </TableCell>  
 </TableRow>  
 ) : (  
 paginatedLinks.map((link) => (  
 <TableRow key={link.id}>  
 <TableCell>  
 <div className="flex flex-col">  
 <a  
 href={`/r/${link.shortCode}`}  
 target="\_blank"  
 rel="noopener noreferrer"  
 className="text-primary font-medium text-sm hover:underline truncate max-w-xs"  
 title={`${window.location.origin}/r/${link.shortCode}`}  
 >  
 {window.location.host}/r/{link.shortCode}  
 </a>  
 <span  
 className="text-muted-foreground text-xs truncate max-w-xs"  
 title={link.originalUrl}  
 >  
 {link.originalUrl.length > 50  
 ? `${link.originalUrl.substring(0, 50)}...`  
 : link.originalUrl}  
 </span>  
 </div>  
 </TableCell>  
 <TableCell className="text-sm text-muted-foreground">  
 {formatDate(link.createdAt)}  
 </TableCell>  
 <TableCell>  
 <div className="text-sm font-medium">{link.clickCount}</div>  
 </TableCell>  
 <TableCell>{getStatusBadge(link)}</TableCell>  
 <TableCell>  
 <div className="flex items-center space-x-2">  
 <TooltipProvider>  
 <Tooltip>  
 <TooltipTrigger asChild>  
 <Button  
 variant="ghost"  
 size="icon"  
 onClick={() => copyToClipboard(link.shortCode)}  
 >  
 <Copy className="h-4 w-4" />  
 <span className="sr-only">Copy link</span>  
 </Button>  
 </TooltipTrigger>  
 <TooltipContent>  
 <p>Copy link</p>  
 </TooltipContent>  
 </Tooltip>  
 </TooltipProvider>  
  
 <TooltipProvider>  
 <Tooltip>  
 <TooltipTrigger asChild>  
 <Button variant="ghost" size="icon">  
 <BarChart2 className="h-4 w-4" />  
 <span className="sr-only">View analytics</span>  
 </Button>  
 </TooltipTrigger>  
 <TooltipContent>  
 <p>View analytics</p>  
 </TooltipContent>  
 </Tooltip>  
 </TooltipProvider>  
  
 <TooltipProvider>  
 <Tooltip>  
 <TooltipTrigger asChild>  
 <Button  
 variant="ghost"  
 size="icon"  
 onClick={() => onQrCodeClick(link)}  
 >  
 <QrCode className="h-4 w-4" />  
 <span className="sr-only">Generate QR code</span>  
 </Button>  
 </TooltipTrigger>  
 <TooltipContent>  
 <p>Generate QR code</p>  
 </TooltipContent>  
 </Tooltip>  
 </TooltipProvider>  
  
 <AlertDialog  
 open={linkToDelete?.id === link.id}  
 onOpenChange={(open) => {  
 if (!open) setLinkToDelete(null);  
 }}  
 >  
 <AlertDialogTrigger asChild>  
 <Button  
 variant="ghost"  
 size="icon"  
 className="text-muted-foreground hover:text-destructive"  
 onClick={() => setLinkToDelete(link)}  
 >  
 <Trash2 className="h-4 w-4" />  
 <span className="sr-only">Delete</span>  
 </Button>  
 </AlertDialogTrigger>  
 <AlertDialogContent>  
 <AlertDialogHeader>  
 <AlertDialogTitle>Delete link</AlertDialogTitle>  
 <AlertDialogDescription>  
 Are you sure you want to delete this link? This cannot be undone.  
 </AlertDialogDescription>  
 </AlertDialogHeader>  
 <AlertDialogFooter>  
 <AlertDialogCancel>Cancel</AlertDialogCancel>  
 <AlertDialogAction  
 onClick={() => {  
 if (linkToDelete) {  
 deleteLinkMutation.mutate(linkToDelete.id);  
 }  
 }}  
 className="bg-destructive text-destructive-foreground hover:bg-destructive/90"  
 >  
 {deleteLinkMutation.isPending ? "Deleting..." : "Delete"}  
 </AlertDialogAction>  
 </AlertDialogFooter>  
 </AlertDialogContent>  
 </AlertDialog>  
 </div>  
 </TableCell>  
 </TableRow>  
 ))  
 )}  
 </TableBody>  
 </Table>  
 </div>  
 </CardContent>  
 {totalPages > 1 && (  
 <CardFooter className="flex items-center justify-between px-6 py-4">  
 <div className="text-sm text-muted-foreground">  
 Showing {Math.min((page - 1) \* itemsPerPage + 1, filteredLinks.length)} to{" "}  
 {Math.min(page \* itemsPerPage, filteredLinks.length)} of {filteredLinks.length} results  
 </div>  
 <Pagination>  
 <PaginationContent>  
 <PaginationItem>  
 <PaginationPrevious  
 onClick={() => setPage((p) => Math.max(1, p - 1))}  
 aria-disabled={page === 1 ? "true" : "false"}  
 className={page === 1 ? "pointer-events-none opacity-50" : ""}  
 />  
 </PaginationItem>  
   
 {Array.from({ length: Math.min(5, totalPages) }, (\_, i) => {  
 // Calculate page numbers to show based on current page  
 let pageNum;  
 if (totalPages <= 5) {  
 pageNum = i + 1;  
 } else if (page <= 3) {  
 pageNum = i + 1;  
 } else if (page >= totalPages - 2) {  
 pageNum = totalPages - 4 + i;  
 } else {  
 pageNum = page - 2 + i;  
 }  
   
 return (  
 <PaginationItem key={i}>  
 <PaginationLink  
 onClick={() => setPage(pageNum)}  
 isActive={page === pageNum}  
 >  
 {pageNum}  
 </PaginationLink>  
 </PaginationItem>  
 );  
 })}  
   
 {totalPages > 5 && page < totalPages - 2 && (  
 <PaginationItem>  
 <PaginationEllipsis />  
 </PaginationItem>  
 )}  
   
 <PaginationItem>  
 <PaginationNext  
 onClick={() => setPage((p) => Math.min(totalPages, p + 1))}  
 aria-disabled={page === totalPages ? "true" : "false"}  
 className={page === totalPages ? "pointer-events-none opacity-50" : ""}  
 />  
 </PaginationItem>  
 </PaginationContent>  
 </Pagination>  
 </CardFooter>  
 )}  
 </Card>  
 );  
}

client/src/components/qr-code-modal.tsx

import { useState } from "react";  
import { LinkWithAnalytics } from "@shared/schema";  
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogDescription } from "@/components/ui/dialog";  
import { Button } from "@/components/ui/button";  
import { useToast } from "@/hooks/use-toast";  
import { Download, Share2 } from "lucide-react";  
import { QRCodeCanvas } from "qrcode.react";  
  
interface QrCodeModalProps {  
 open: boolean;  
 onClose: () => void;  
 link: LinkWithAnalytics | null;  
}  
  
export default function QrCodeModal({ open, onClose, link }: QrCodeModalProps) {  
 const { toast } = useToast();  
 const [isDownloading, setIsDownloading] = useState(false);  
 const [isSharing, setIsSharing] = useState(false);  
   
 if (!link) return null;  
   
 const shortUrl = `${window.location.origin}/r/${link.shortCode}`;  
   
 const handleDownload = () => {  
 setIsDownloading(true);  
   
 try {  
 const canvas = document.querySelector('#qr-code-canvas canvas');  
 if (!canvas) {  
 throw new Error("QR code canvas not found");  
 }  
   
 const dataUrl = (canvas as HTMLCanvasElement).toDataURL("image/png");  
 const downloadLink = document.createElement("a");  
   
 downloadLink.href = dataUrl;  
 downloadLink.download = `qrcode-${link.shortCode}.png`;  
 document.body.appendChild(downloadLink);  
 downloadLink.click();  
 document.body.removeChild(downloadLink);  
   
 toast({  
 title: "QR Code downloaded",  
 description: "Your QR code has been saved successfully.",  
 });  
 } catch (error) {  
 toast({  
 title: "Download failed",  
 description: "There was an error downloading the QR code.",  
 variant: "destructive",  
 });  
 console.error("QR code download error:", error);  
 } finally {  
 setIsDownloading(false);  
 }  
 };  
   
 const handleShare = async () => {  
 setIsSharing(true);  
   
 try {  
 if (navigator.share) {  
 await navigator.share({  
 title: "Shared QR Code",  
 text: `Check out this link: ${shortUrl}`,  
 url: shortUrl,  
 });  
   
 toast({  
 title: "Link shared",  
 description: "Your link has been shared successfully.",  
 });  
 } else {  
 // Fallback for browsers that don't support navigator.share  
 navigator.clipboard.writeText(shortUrl);  
   
 toast({  
 title: "Link copied",  
 description: "Link copied to clipboard for sharing.",  
 });  
 }  
 } catch (error) {  
 if ((error as Error).name !== 'AbortError') {  
 toast({  
 title: "Share failed",  
 description: "There was an error sharing the QR code.",  
 variant: "destructive",  
 });  
 console.error("QR code share error:", error);  
 }  
 } finally {  
 setIsSharing(false);  
 }  
 };  
  
 return (  
 <Dialog open={open} onOpenChange={onClose}>  
 <DialogContent className="sm:max-w-md">  
 <DialogHeader>  
 <DialogTitle>QR Code</DialogTitle>  
 <DialogDescription>  
 Scan this QR code to access your shortened URL  
 </DialogDescription>  
 </DialogHeader>  
   
 <div className="flex flex-col items-center p-4">  
 <div className="bg-white p-4 rounded-lg mb-4" id="qr-code-canvas">  
 <QRCodeCanvas  
 value={shortUrl}  
 size={200}  
 level="H"  
 includeMargin={true}  
 imageSettings={{  
 src: "https://raw.githubusercontent.com/Achraf-haddar/bitly-clone/main/client/src/assets/logo.png",  
 height: 24,  
 width: 24,  
 excavate: true,  
 }}  
 />  
 </div>  
   
 <p className="text-sm text-muted-foreground mb-4 text-center break-all">  
 <span>Scan to access:</span>{" "}  
 <strong className="text-primary font-medium">  
 {shortUrl}  
 </strong>  
 </p>  
   
 <div className="flex items-center space-x-2">  
 <Button  
 className="flex items-center gap-2"  
 onClick={handleDownload}  
 disabled={isDownloading}  
 >  
 {isDownloading ? (  
 <span className="h-4 w-4 animate-spin rounded-full border-2 border-background border-t-transparent"></span>  
 ) : (  
 <Download className="h-4 w-4" />  
 )}  
 <span>Download</span>  
 </Button>  
   
 <Button  
 variant="outline"  
 className="flex items-center gap-2"  
 onClick={handleShare}  
 disabled={isSharing}  
 >  
 {isSharing ? (  
 <span className="h-4 w-4 animate-spin rounded-full border-2 border-primary border-t-transparent"></span>  
 ) : (  
 <Share2 className="h-4 w-4" />  
 )}  
 <span>Share</span>  
 </Button>  
 </div>  
 </div>  
 </DialogContent>  
 </Dialog>  
 );  
}

client/src/components/stats-grid.tsx

import { useQuery } from "@tanstack/react-query";  
import { Card, CardContent } from "@/components/ui/card";  
import { Skeleton } from "@/components/ui/skeleton";  
import { ArrowUpIcon, LinkIcon, MousePointerClick, CheckIcon, Clock } from "lucide-react";  
  
type DashboardStats = {  
 totalLinks: number;  
 totalClicks: number;  
 activeLinks: number;  
 expiringLinks: number;  
};  
  
export default function StatsGrid() {  
 const { data: stats, isLoading } = useQuery<DashboardStats>({  
 queryKey: ["/api/dashboard/stats"],  
 });  
  
 return (  
 <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4 mb-6">  
 {/\* Total Links \*/}  
 <Card>  
 <CardContent className="pt-6 px-5">  
 <div className="flex items-center justify-between">  
 <p className="text-muted-foreground text-sm">Total Links</p>  
 <div className="w-10 h-10 rounded-lg bg-primary/10 flex items-center justify-center">  
 <LinkIcon className="h-5 w-5 text-primary" />  
 </div>  
 </div>  
   
 {isLoading ? (  
 <Skeleton className="h-8 w-16 mt-2" />  
 ) : (  
 <h3 className="text-2xl font-bold mt-2">{stats?.totalLinks || 0}</h3>  
 )}  
   
 <p className="text-sm text-green-600 dark:text-green-400 flex items-center mt-1">  
 <ArrowUpIcon className="h-4 w-4 mr-1" />  
 <span>New opportunities</span>  
 </p>  
 </CardContent>  
 </Card>  
  
 {/\* Total Clicks \*/}  
 <Card>  
 <CardContent className="pt-6 px-5">  
 <div className="flex items-center justify-between">  
 <p className="text-muted-foreground text-sm">Total Clicks</p>  
 <div className="w-10 h-10 rounded-lg bg-indigo-100 dark:bg-indigo-900/30 flex items-center justify-center">  
 <MousePointerClick className="h-5 w-5 text-indigo-600 dark:text-indigo-400" />  
 </div>  
 </div>  
   
 {isLoading ? (  
 <Skeleton className="h-8 w-16 mt-2" />  
 ) : (  
 <h3 className="text-2xl font-bold mt-2">{stats?.totalClicks || 0}</h3>  
 )}  
   
 <p className="text-sm text-green-600 dark:text-green-400 flex items-center mt-1">  
 <ArrowUpIcon className="h-4 w-4 mr-1" />  
 <span>Expanding reach</span>  
 </p>  
 </CardContent>  
 </Card>  
  
 {/\* Active Links \*/}  
 <Card>  
 <CardContent className="pt-6 px-5">  
 <div className="flex items-center justify-between">  
 <p className="text-muted-foreground text-sm">Active Links</p>  
 <div className="w-10 h-10 rounded-lg bg-green-100 dark:bg-green-900/30 flex items-center justify-center">  
 <CheckIcon className="h-5 w-5 text-green-600 dark:text-green-400" />  
 </div>  
 </div>  
   
 {isLoading ? (  
 <Skeleton className="h-8 w-16 mt-2" />  
 ) : (  
 <h3 className="text-2xl font-bold mt-2">{stats?.activeLinks || 0}</h3>  
 )}  
   
 <p className="text-sm text-green-600 dark:text-green-400 flex items-center mt-1">  
 <ArrowUpIcon className="h-4 w-4 mr-1" />  
 <span>Working links</span>  
 </p>  
 </CardContent>  
 </Card>  
  
 {/\* Expiring Soon \*/}  
 <Card>  
 <CardContent className="pt-6 px-5">  
 <div className="flex items-center justify-between">  
 <p className="text-muted-foreground text-sm">Expiring Soon</p>  
 <div className="w-10 h-10 rounded-lg bg-amber-100 dark:bg-amber-900/30 flex items-center justify-center">  
 <Clock className="h-5 w-5 text-amber-600 dark:text-amber-400" />  
 </div>  
 </div>  
   
 {isLoading ? (  
 <Skeleton className="h-8 w-16 mt-2" />  
 ) : (  
 <h3 className="text-2xl font-bold mt-2">{stats?.expiringLinks || 0}</h3>  
 )}  
   
 <p className="text-sm text-amber-600 dark:text-amber-400 flex items-center mt-1">  
 <Clock className="h-4 w-4 mr-1" />  
 <span>Within 7 days</span>  
 </p>  
 </CardContent>  
 </Card>  
 </div>  
 );  
}

client/src/components/ui/accordion.tsx

import \* as React from "react"  
import \* as AccordionPrimitive from "@radix-ui/react-accordion"  
import { ChevronDown } from "lucide-react"  
  
import { cn } from "@/lib/utils"  
  
const Accordion = AccordionPrimitive.Root  
  
const AccordionItem = React.forwardRef<  
 React.ElementRef<typeof AccordionPrimitive.Item>,  
 React.ComponentPropsWithoutRef<typeof AccordionPrimitive.Item>  
>(({ className, ...props }, ref) => (  
 <AccordionPrimitive.Item  
 ref={ref}  
 className={cn("border-b", className)}  
 {...props}  
 />  
))  
AccordionItem.displayName = "AccordionItem"  
  
const AccordionTrigger = React.forwardRef<  
 React.ElementRef<typeof AccordionPrimitive.Trigger>,  
 React.ComponentPropsWithoutRef<typeof AccordionPrimitive.Trigger>  
>(({ className, children, ...props }, ref) => (  
 <AccordionPrimitive.Header className="flex">  
 <AccordionPrimitive.Trigger  
 ref={ref}  
 className={cn(  
 "flex flex-1 items-center justify-between py-4 font-medium transition-all hover:underline [&[data-state=open]>svg]:rotate-180",  
 className  
 )}  
 {...props}  
 >  
 {children}  
 <ChevronDown className="h-4 w-4 shrink-0 transition-transform duration-200" />  
 </AccordionPrimitive.Trigger>  
 </AccordionPrimitive.Header>  
))  
AccordionTrigger.displayName = AccordionPrimitive.Trigger.displayName  
  
const AccordionContent = React.forwardRef<  
 React.ElementRef<typeof AccordionPrimitive.Content>,  
 React.ComponentPropsWithoutRef<typeof AccordionPrimitive.Content>  
>(({ className, children, ...props }, ref) => (  
 <AccordionPrimitive.Content  
 ref={ref}  
 className="overflow-hidden text-sm transition-all data-[state=closed]:animate-accordion-up data-[state=open]:animate-accordion-down"  
 {...props}  
 >  
 <div className={cn("pb-4 pt-0", className)}>{children}</div>  
 </AccordionPrimitive.Content>  
))  
  
AccordionContent.displayName = AccordionPrimitive.Content.displayName  
  
export { Accordion, AccordionItem, AccordionTrigger, AccordionContent }

client/src/components/ui/alert-dialog.tsx

import \* as React from "react"  
import \* as AlertDialogPrimitive from "@radix-ui/react-alert-dialog"  
  
import { cn } from "@/lib/utils"  
import { buttonVariants } from "@/components/ui/button"  
  
const AlertDialog = AlertDialogPrimitive.Root  
  
const AlertDialogTrigger = AlertDialogPrimitive.Trigger  
  
const AlertDialogPortal = AlertDialogPrimitive.Portal  
  
const AlertDialogOverlay = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Overlay>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Overlay>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPrimitive.Overlay  
 className={cn(  
 "fixed inset-0 z-50 bg-black/80 data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0",  
 className  
 )}  
 {...props}  
 ref={ref}  
 />  
))  
AlertDialogOverlay.displayName = AlertDialogPrimitive.Overlay.displayName  
  
const AlertDialogContent = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Content>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Content>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPortal>  
 <AlertDialogOverlay />  
 <AlertDialogPrimitive.Content  
 ref={ref}  
 className={cn(  
 "fixed left-[50%] top-[50%] z-50 grid w-full max-w-lg translate-x-[-50%] translate-y-[-50%] gap-4 border bg-background p-6 shadow-lg duration-200 data-[state=open]:animate-in data-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95 data-[state=closed]:slide-out-to-left-1/2 data-[state=closed]:slide-out-to-top-[48%] data-[state=open]:slide-in-from-left-1/2 data-[state=open]:slide-in-from-top-[48%] sm:rounded-lg",  
 className  
 )}  
 {...props}  
 />  
 </AlertDialogPortal>  
))  
AlertDialogContent.displayName = AlertDialogPrimitive.Content.displayName  
  
const AlertDialogHeader = ({  
 className,  
 ...props  
}: React.HTMLAttributes<HTMLDivElement>) => (  
 <div  
 className={cn(  
 "flex flex-col space-y-2 text-center sm:text-left",  
 className  
 )}  
 {...props}  
 />  
)  
AlertDialogHeader.displayName = "AlertDialogHeader"  
  
const AlertDialogFooter = ({  
 className,  
 ...props  
}: React.HTMLAttributes<HTMLDivElement>) => (  
 <div  
 className={cn(  
 "flex flex-col-reverse sm:flex-row sm:justify-end sm:space-x-2",  
 className  
 )}  
 {...props}  
 />  
)  
AlertDialogFooter.displayName = "AlertDialogFooter"  
  
const AlertDialogTitle = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Title>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Title>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPrimitive.Title  
 ref={ref}  
 className={cn("text-lg font-semibold", className)}  
 {...props}  
 />  
))  
AlertDialogTitle.displayName = AlertDialogPrimitive.Title.displayName  
  
const AlertDialogDescription = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Description>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Description>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPrimitive.Description  
 ref={ref}  
 className={cn("text-sm text-muted-foreground", className)}  
 {...props}  
 />  
))  
AlertDialogDescription.displayName =  
 AlertDialogPrimitive.Description.displayName  
  
const AlertDialogAction = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Action>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Action>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPrimitive.Action  
 ref={ref}  
 className={cn(buttonVariants(), className)}  
 {...props}  
 />  
))  
AlertDialogAction.displayName = AlertDialogPrimitive.Action.displayName  
  
const AlertDialogCancel = React.forwardRef<  
 React.ElementRef<typeof AlertDialogPrimitive.Cancel>,  
 React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Cancel>  
>(({ className, ...props }, ref) => (  
 <AlertDialogPrimitive.Cancel  
 ref={ref}  
 className={cn(  
 buttonVariants({ variant: "outline" }),  
 "mt-2 sm:mt-0",  
 className  
 )}  
 {...props}  
 />  
))  
AlertDialogCancel.displayName = AlertDialogPrimitive.Cancel.displayName  
  
export {  
 AlertDialog,  
 AlertDialogPortal,  
 AlertDialogOverlay,  
 AlertDialogTrigger,  
 AlertDialogContent,  
 AlertDialogHeader,  
 AlertDialogFooter,  
 AlertDialogTitle,  
 AlertDialogDescription,  
 AlertDialogAction,  
 AlertDialogCancel,  
}

client/src/components/ui/alert.tsx

import \* as React from "react"  
import { cva, type VariantProps } from "class-variance-authority"  
  
import { cn } from "@/lib/utils"  
  
const alertVariants = cva(  
 "relative w-full rounded-lg border p-4 [&>svg~\*]:pl-7 [&>svg+div]:translate-y-[-3px] [&>svg]:absolute [&>svg]:left-4 [&>svg]:top-4 [&>svg]:text-foreground",  
 {  
 variants: {  
 variant: {  
 default: "bg-background text-foreground",  
 destructive:  
 "border-destructive/50 text-destructive dark:border-destructive [&>svg]:text-destructive",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 },  
 }  
)  
  
const Alert = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement> & VariantProps<typeof alertVariants>  
>(({ className, variant, ...props }, ref) => (  
 <div  
 ref={ref}  
 role="alert"  
 className={cn(alertVariants({ variant }), className)}  
 {...props}  
 />  
))  
Alert.displayName = "Alert"  
  
const AlertTitle = React.forwardRef<  
 HTMLParagraphElement,  
 React.HTMLAttributes<HTMLHeadingElement>  
>(({ className, ...props }, ref) => (  
 <h5  
 ref={ref}  
 className={cn("mb-1 font-medium leading-none tracking-tight", className)}  
 {...props}  
 />  
))  
AlertTitle.displayName = "AlertTitle"  
  
const AlertDescription = React.forwardRef<  
 HTMLParagraphElement,  
 React.HTMLAttributes<HTMLParagraphElement>  
>(({ className, ...props }, ref) => (  
 <div  
 ref={ref}  
 className={cn("text-sm [&\_p]:leading-relaxed", className)}  
 {...props}  
 />  
))  
AlertDescription.displayName = "AlertDescription"  
  
export { Alert, AlertTitle, AlertDescription }

client/src/components/ui/aspect-ratio.tsx

import \* as AspectRatioPrimitive from "@radix-ui/react-aspect-ratio"  
  
const AspectRatio = AspectRatioPrimitive.Root  
  
export { AspectRatio }

client/src/components/ui/avatar.tsx

import \* as React from "react"  
import \* as AvatarPrimitive from "@radix-ui/react-avatar"  
  
import { cn } from "@/lib/utils"  
  
const Avatar = React.forwardRef<  
 React.ElementRef<typeof AvatarPrimitive.Root>,  
 React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Root>  
>(({ className, ...props }, ref) => (  
 <AvatarPrimitive.Root  
 ref={ref}  
 className={cn(  
 "relative flex h-10 w-10 shrink-0 overflow-hidden rounded-full",  
 className  
 )}  
 {...props}  
 />  
))  
Avatar.displayName = AvatarPrimitive.Root.displayName  
  
const AvatarImage = React.forwardRef<  
 React.ElementRef<typeof AvatarPrimitive.Image>,  
 React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Image>  
>(({ className, ...props }, ref) => (  
 <AvatarPrimitive.Image  
 ref={ref}  
 className={cn("aspect-square h-full w-full", className)}  
 {...props}  
 />  
))  
AvatarImage.displayName = AvatarPrimitive.Image.displayName  
  
const AvatarFallback = React.forwardRef<  
 React.ElementRef<typeof AvatarPrimitive.Fallback>,  
 React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Fallback>  
>(({ className, ...props }, ref) => (  
 <AvatarPrimitive.Fallback  
 ref={ref}  
 className={cn(  
 "flex h-full w-full items-center justify-center rounded-full bg-muted",  
 className  
 )}  
 {...props}  
 />  
))  
AvatarFallback.displayName = AvatarPrimitive.Fallback.displayName  
  
export { Avatar, AvatarImage, AvatarFallback }

client/src/components/ui/badge.tsx

import \* as React from "react"  
import { cva, type VariantProps } from "class-variance-authority"  
  
import { cn } from "@/lib/utils"  
  
const badgeVariants = cva(  
 "inline-flex items-center rounded-full border px-2.5 py-0.5 text-xs font-semibold transition-colors focus:outline-none focus:ring-2 focus:ring-ring focus:ring-offset-2",  
 {  
 variants: {  
 variant: {  
 default:  
 "border-transparent bg-primary text-primary-foreground hover:bg-primary/80",  
 secondary:  
 "border-transparent bg-secondary text-secondary-foreground hover:bg-secondary/80",  
 destructive:  
 "border-transparent bg-destructive text-destructive-foreground hover:bg-destructive/80",  
 outline: "text-foreground",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 },  
 }  
)  
  
export interface BadgeProps  
 extends React.HTMLAttributes<HTMLDivElement>,  
 VariantProps<typeof badgeVariants> {}  
  
function Badge({ className, variant, ...props }: BadgeProps) {  
 return (  
 <div className={cn(badgeVariants({ variant }), className)} {...props} />  
 )  
}  
  
export { Badge, badgeVariants }

client/src/components/ui/breadcrumb.tsx

import \* as React from "react"  
import { Slot } from "@radix-ui/react-slot"  
import { ChevronRight, MoreHorizontal } from "lucide-react"  
  
import { cn } from "@/lib/utils"  
  
const Breadcrumb = React.forwardRef<  
 HTMLElement,  
 React.ComponentPropsWithoutRef<"nav"> & {  
 separator?: React.ReactNode  
 }  
>(({ ...props }, ref) => <nav ref={ref} aria-label="breadcrumb" {...props} />)  
Breadcrumb.displayName = "Breadcrumb"  
  
const BreadcrumbList = React.forwardRef<  
 HTMLOListElement,  
 React.ComponentPropsWithoutRef<"ol">  
>(({ className, ...props }, ref) => (  
 <ol  
 ref={ref}  
 className={cn(  
 "flex flex-wrap items-center gap-1.5 break-words text-sm text-muted-foreground sm:gap-2.5",  
 className  
 )}  
 {...props}  
 />  
))  
BreadcrumbList.displayName = "BreadcrumbList"  
  
const BreadcrumbItem = React.forwardRef<  
 HTMLLIElement,  
 React.ComponentPropsWithoutRef<"li">  
>(({ className, ...props }, ref) => (  
 <li  
 ref={ref}  
 className={cn("inline-flex items-center gap-1.5", className)}  
 {...props}  
 />  
))  
BreadcrumbItem.displayName = "BreadcrumbItem"  
  
const BreadcrumbLink = React.forwardRef<  
 HTMLAnchorElement,  
 React.ComponentPropsWithoutRef<"a"> & {  
 asChild?: boolean  
 }  
>(({ asChild, className, ...props }, ref) => {  
 const Comp = asChild ? Slot : "a"  
  
 return (  
 <Comp  
 ref={ref}  
 className={cn("transition-colors hover:text-foreground", className)}  
 {...props}  
 />  
 )  
})  
BreadcrumbLink.displayName = "BreadcrumbLink"  
  
const BreadcrumbPage = React.forwardRef<  
 HTMLSpanElement,  
 React.ComponentPropsWithoutRef<"span">  
>(({ className, ...props }, ref) => (  
 <span  
 ref={ref}  
 role="link"  
 aria-disabled="true"  
 aria-current="page"  
 className={cn("font-normal text-foreground", className)}  
 {...props}  
 />  
))  
BreadcrumbPage.displayName = "BreadcrumbPage"  
  
const BreadcrumbSeparator = ({  
 children,  
 className,  
 ...props  
}: React.ComponentProps<"li">) => (  
 <li  
 role="presentation"  
 aria-hidden="true"  
 className={cn("[&>svg]:w-3.5 [&>svg]:h-3.5", className)}  
 {...props}  
 >  
 {children ?? <ChevronRight />}  
 </li>  
)  
BreadcrumbSeparator.displayName = "BreadcrumbSeparator"  
  
const BreadcrumbEllipsis = ({  
 className,  
 ...props  
}: React.ComponentProps<"span">) => (  
 <span  
 role="presentation"  
 aria-hidden="true"  
 className={cn("flex h-9 w-9 items-center justify-center", className)}  
 {...props}  
 >  
 <MoreHorizontal className="h-4 w-4" />  
 <span className="sr-only">More</span>  
 </span>  
)  
BreadcrumbEllipsis.displayName = "BreadcrumbElipssis"  
  
export {  
 Breadcrumb,  
 BreadcrumbList,  
 BreadcrumbItem,  
 BreadcrumbLink,  
 BreadcrumbPage,  
 BreadcrumbSeparator,  
 BreadcrumbEllipsis,  
}

client/src/components/ui/button.tsx

import \* as React from "react"  
import { Slot } from "@radix-ui/react-slot"  
import { cva, type VariantProps } from "class-variance-authority"  
  
import { cn } from "@/lib/utils"  
  
const buttonVariants = cva(  
 "inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-md text-sm font-medium 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 [&\_svg]:pointer-events-none [&\_svg]:size-4 [&\_svg]:shrink-0",  
 {  
 variants: {  
 variant: {  
 default: "bg-primary text-primary-foreground hover:bg-primary/90",  
 destructive:  
 "bg-destructive text-destructive-foreground hover:bg-destructive/90",  
 outline:  
 "border border-input bg-background hover:bg-accent hover:text-accent-foreground",  
 secondary:  
 "bg-secondary text-secondary-foreground hover:bg-secondary/80",  
 ghost: "hover:bg-accent hover:text-accent-foreground",  
 link: "text-primary underline-offset-4 hover:underline",  
 },  
 size: {  
 default: "h-10 px-4 py-2",  
 sm: "h-9 rounded-md px-3",  
 lg: "h-11 rounded-md px-8",  
 icon: "h-10 w-10",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 size: "default",  
 },  
 }  
)  
  
export interface ButtonProps  
 extends React.ButtonHTMLAttributes<HTMLButtonElement>,  
 VariantProps<typeof buttonVariants> {  
 asChild?: boolean  
}  
  
const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(  
 ({ className, variant, size, asChild = false, ...props }, ref) => {  
 const Comp = asChild ? Slot : "button"  
 return (  
 <Comp  
 className={cn(buttonVariants({ variant, size, className }))}  
 ref={ref}  
 {...props}  
 />  
 )  
 }  
)  
Button.displayName = "Button"  
  
export { Button, buttonVariants }

client/src/components/ui/calendar.tsx

import \* as React from "react"  
import { ChevronLeft, ChevronRight } from "lucide-react"  
import { DayPicker } from "react-day-picker"  
  
import { cn } from "@/lib/utils"  
import { buttonVariants } from "@/components/ui/button"  
  
export type CalendarProps = React.ComponentProps<typeof DayPicker>  
  
function Calendar({  
 className,  
 classNames,  
 showOutsideDays = true,  
 ...props  
}: CalendarProps) {  
 return (  
 <DayPicker  
 showOutsideDays={showOutsideDays}  
 className={cn("p-3", className)}  
 classNames={{  
 months: "flex flex-col sm:flex-row space-y-4 sm:space-x-4 sm:space-y-0",  
 month: "space-y-4",  
 caption: "flex justify-center pt-1 relative items-center",  
 caption\_label: "text-sm font-medium",  
 nav: "space-x-1 flex items-center",  
 nav\_button: cn(  
 buttonVariants({ variant: "outline" }),  
 "h-7 w-7 bg-transparent p-0 opacity-50 hover:opacity-100"  
 ),  
 nav\_button\_previous: "absolute left-1",  
 nav\_button\_next: "absolute right-1",  
 table: "w-full border-collapse space-y-1",  
 head\_row: "flex",  
 head\_cell:  
 "text-muted-foreground rounded-md w-9 font-normal text-[0.8rem]",  
 row: "flex w-full mt-2",  
 cell: "h-9 w-9 text-center text-sm p-0 relative [&:has([aria-selected].day-range-end)]:rounded-r-md [&:has([aria-selected].day-outside)]:bg-accent/50 [&:has([aria-selected])]:bg-accent first:[&:has([aria-selected])]:rounded-l-md last:[&:has([aria-selected])]:rounded-r-md focus-within:relative focus-within:z-20",  
 day: cn(  
 buttonVariants({ variant: "ghost" }),  
 "h-9 w-9 p-0 font-normal aria-selected:opacity-100"  
 ),  
 day\_range\_end: "day-range-end",  
 day\_selected:  
 "bg-primary text-primary-foreground hover:bg-primary hover:text-primary-foreground focus:bg-primary focus:text-primary-foreground",  
 day\_today: "bg-accent text-accent-foreground",  
 day\_outside:  
 "day-outside text-muted-foreground opacity-50 aria-selected:bg-accent/50 aria-selected:text-muted-foreground aria-selected:opacity-30",  
 day\_disabled: "text-muted-foreground opacity-50",  
 day\_range\_middle:  
 "aria-selected:bg-accent aria-selected:text-accent-foreground",  
 day\_hidden: "invisible",  
 ...classNames,  
 }}  
 components={{  
 IconLeft: ({ ...props }) => <ChevronLeft className="h-4 w-4" />,  
 IconRight: ({ ...props }) => <ChevronRight className="h-4 w-4" />,  
 }}  
 {...props}  
 />  
 )  
}  
Calendar.displayName = "Calendar"  
  
export { Calendar }

client/src/components/ui/card.tsx

import \* as React from "react"  
  
import { cn } from "@/lib/utils"  
  
const Card = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => (  
 <div  
 ref={ref}  
 className={cn(  
 "rounded-lg border bg-card text-card-foreground shadow-sm",  
 className  
 )}  
 {...props}  
 />  
))  
Card.displayName = "Card"  
  
const CardHeader = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => (  
 <div  
 ref={ref}  
 className={cn("flex flex-col space-y-1.5 p-6", className)}  
 {...props}  
 />  
))  
CardHeader.displayName = "CardHeader"  
  
const CardTitle = React.forwardRef<  
 HTMLParagraphElement,  
 React.HTMLAttributes<HTMLHeadingElement>  
>(({ className, ...props }, ref) => (  
 <h3  
 ref={ref}  
 className={cn(  
 "text-2xl font-semibold leading-none tracking-tight",  
 className  
 )}  
 {...props}  
 />  
))  
CardTitle.displayName = "CardTitle"  
  
const CardDescription = React.forwardRef<  
 HTMLParagraphElement,  
 React.HTMLAttributes<HTMLParagraphElement>  
>(({ className, ...props }, ref) => (  
 <p  
 ref={ref}  
 className={cn("text-sm text-muted-foreground", className)}  
 {...props}  
 />  
))  
CardDescription.displayName = "CardDescription"  
  
const CardContent = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => (  
 <div ref={ref} className={cn("p-6 pt-0", className)} {...props} />  
))  
CardContent.displayName = "CardContent"  
  
const CardFooter = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => (  
 <div  
 ref={ref}  
 className={cn("flex items-center p-6 pt-0", className)}  
 {...props}  
 />  
))  
CardFooter.displayName = "CardFooter"  
  
export { Card, CardHeader, CardFooter, CardTitle, CardDescription, CardContent }

client/src/components/ui/carousel.tsx

import \* as React from "react"  
import useEmblaCarousel, {  
 type UseEmblaCarouselType,  
} from "embla-carousel-react"  
import { ArrowLeft, ArrowRight } from "lucide-react"  
  
import { cn } from "@/lib/utils"  
import { Button } from "@/components/ui/button"  
  
type CarouselApi = UseEmblaCarouselType[1]  
type UseCarouselParameters = Parameters<typeof useEmblaCarousel>  
type CarouselOptions = UseCarouselParameters[0]  
type CarouselPlugin = UseCarouselParameters[1]  
  
type CarouselProps = {  
 opts?: CarouselOptions  
 plugins?: CarouselPlugin  
 orientation?: "horizontal" | "vertical"  
 setApi?: (api: CarouselApi) => void  
}  
  
type CarouselContextProps = {  
 carouselRef: ReturnType<typeof useEmblaCarousel>[0]  
 api: ReturnType<typeof useEmblaCarousel>[1]  
 scrollPrev: () => void  
 scrollNext: () => void  
 canScrollPrev: boolean  
 canScrollNext: boolean  
} & CarouselProps  
  
const CarouselContext = React.createContext<CarouselContextProps | null>(null)  
  
function useCarousel() {  
 const context = React.useContext(CarouselContext)  
  
 if (!context) {  
 throw new Error("useCarousel must be used within a <Carousel />")  
 }  
  
 return context  
}  
  
const Carousel = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement> & CarouselProps  
>(  
 (  
 {  
 orientation = "horizontal",  
 opts,  
 setApi,  
 plugins,  
 className,  
 children,  
 ...props  
 },  
 ref  
 ) => {  
 const [carouselRef, api] = useEmblaCarousel(  
 {  
 ...opts,  
 axis: orientation === "horizontal" ? "x" : "y",  
 },  
 plugins  
 )  
 const [canScrollPrev, setCanScrollPrev] = React.useState(false)  
 const [canScrollNext, setCanScrollNext] = React.useState(false)  
  
 const onSelect = React.useCallback((api: CarouselApi) => {  
 if (!api) {  
 return  
 }  
  
 setCanScrollPrev(api.canScrollPrev())  
 setCanScrollNext(api.canScrollNext())  
 }, [])  
  
 const scrollPrev = React.useCallback(() => {  
 api?.scrollPrev()  
 }, [api])  
  
 const scrollNext = React.useCallback(() => {  
 api?.scrollNext()  
 }, [api])  
  
 const handleKeyDown = React.useCallback(  
 (event: React.KeyboardEvent<HTMLDivElement>) => {  
 if (event.key === "ArrowLeft") {  
 event.preventDefault()  
 scrollPrev()  
 } else if (event.key === "ArrowRight") {  
 event.preventDefault()  
 scrollNext()  
 }  
 },  
 [scrollPrev, scrollNext]  
 )  
  
 React.useEffect(() => {  
 if (!api || !setApi) {  
 return  
 }  
  
 setApi(api)  
 }, [api, setApi])  
  
 React.useEffect(() => {  
 if (!api) {  
 return  
 }  
  
 onSelect(api)  
 api.on("reInit", onSelect)  
 api.on("select", onSelect)  
  
 return () => {  
 api?.off("select", onSelect)  
 }  
 }, [api, onSelect])  
  
 return (  
 <CarouselContext.Provider  
 value={{  
 carouselRef,  
 api: api,  
 opts,  
 orientation:  
 orientation || (opts?.axis === "y" ? "vertical" : "horizontal"),  
 scrollPrev,  
 scrollNext,  
 canScrollPrev,  
 canScrollNext,  
 }}  
 >  
 <div  
 ref={ref}  
 onKeyDownCapture={handleKeyDown}  
 className={cn("relative", className)}  
 role="region"  
 aria-roledescription="carousel"  
 {...props}  
 >  
 {children}  
 </div>  
 </CarouselContext.Provider>  
 )  
 }  
)  
Carousel.displayName = "Carousel"  
  
const CarouselContent = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => {  
 const { carouselRef, orientation } = useCarousel()  
  
 return (  
 <div ref={carouselRef} className="overflow-hidden">  
 <div  
 ref={ref}  
 className={cn(  
 "flex",  
 orientation === "horizontal" ? "-ml-4" : "-mt-4 flex-col",  
 className  
 )}  
 {...props}  
 />  
 </div>  
 )  
})  
CarouselContent.displayName = "CarouselContent"  
  
const CarouselItem = React.forwardRef<  
 HTMLDivElement,  
 React.HTMLAttributes<HTMLDivElement>  
>(({ className, ...props }, ref) => {  
 const { orientation } = useCarousel()  
  
 return (  
 <div  
 ref={ref}  
 role="group"  
 aria-roledescription="slide"  
 className={cn(  
 "min-w-0 shrink-0 grow-0 basis-full",  
 orientation === "horizontal" ? "pl-4" : "pt-4",  
 className  
 )}  
 {...props}  
 />  
 )  
})  
CarouselItem.displayName = "CarouselItem"  
  
const CarouselPrevious = React.forwardRef<  
 HTMLButtonElement,  
 React.ComponentProps<typeof Button>  
>(({ className, variant = "outline", size = "icon", ...props }, ref) => {  
 const { orientation, scrollPrev, canScrollPrev } = useCarousel()  
  
 return (  
 <Button  
 ref={ref}  
 variant={variant}  
 size={size}  
 className={cn(  
 "absolute h-8 w-8 rounded-full",  
 orientation === "horizontal"  
 ? "-left-12 top-1/2 -translate-y-1/2"  
 : "-top-12 left-1/2 -translate-x-1/2 rotate-90",  
 className  
 )}  
 disabled={!canScrollPrev}  
 onClick={scrollPrev}  
 {...props}  
 >  
 <ArrowLeft className="h-4 w-4" />  
 <span className="sr-only">Previous slide</span>  
 </Button>  
 )  
})  
CarouselPrevious.displayName = "CarouselPrevious"  
  
const CarouselNext = React.forwardRef<  
 HTMLButtonElement,  
 React.ComponentProps<typeof Button>  
>(({ className, variant = "outline", size = "icon", ...props }, ref) => {  
 const { orientation, scrollNext, canScrollNext } = useCarousel()  
  
 return (  
 <Button  
 ref={ref}  
 variant={variant}  
 size={size}  
 className={cn(  
 "absolute h-8 w-8 rounded-full",  
 orientation === "horizontal"  
 ? "-right-12 top-1/2 -translate-y-1/2"  
 : "-bottom-12 left-1/2 -translate-x-1/2 rotate-90",  
 className  
 )}  
 disabled={!canScrollNext}  
 onClick={scrollNext}  
 {...props}  
 >  
 <ArrowRight className="h-4 w-4" />  
 <span className="sr-only">Next slide</span>  
 </Button>  
 )  
})  
CarouselNext.displayName = "CarouselNext"  
  
export {  
 type CarouselApi,  
 Carousel,  
 CarouselContent,  
 CarouselItem,  
 CarouselPrevious,  
 CarouselNext,  
}

client/src/components/ui/chart.tsx

import \* as React from "react"  
import \* as RechartsPrimitive from "recharts"  
  
import { cn } from "@/lib/utils"  
  
// Format: { THEME\_NAME: CSS\_SELECTOR }  
const THEMES = { light: "", dark: ".dark" } as const  
  
export type ChartConfig = {  
 [k in string]: {  
 label?: React.ReactNode  
 icon?: React.ComponentType  
 } & (  
 | { color?: string; theme?: never }  
 | { color?: never; theme: Record<keyof typeof THEMES, string> }  
 )  
}  
  
type ChartContextProps = {  
 config: ChartConfig  
}  
  
const ChartContext = React.createContext<ChartContextProps | null>(null)  
  
function useChart() {  
 const context = React.useContext(ChartContext)  
  
 if (!context) {  
 throw new Error("useChart must be used within a <ChartContainer />")  
 }  
  
 return context  
}  
  
const ChartContainer = React.forwardRef<  
 HTMLDivElement,  
 React.ComponentProps<"div"> & {  
 config: ChartConfig  
 children: React.ComponentProps<  
 typeof RechartsPrimitive.ResponsiveContainer  
 >["children"]  
 }  
>(({ id, className, children, config, ...props }, ref) => {  
 const uniqueId = React.useId()  
 const chartId = `chart-${id || uniqueId.replace(/:/g, "")}`  
  
 return (  
 <ChartContext.Provider value={{ config }}>  
 <div  
 data-chart={chartId}  
 ref={ref}  
 className={cn(  
 "flex aspect-video justify-center text-xs [&\_.recharts-cartesian-axis-tick\_text]:fill-muted-foreground [&\_.recharts-cartesian-grid\_line[stroke='#ccc']]:stroke-border/50 [&\_.recharts-curve.recharts-tooltip-cursor]:stroke-border [&\_.recharts-dot[stroke='#fff']]:stroke-transparent [&\_.recharts-layer]:outline-none [&\_.recharts-polar-grid\_[stroke='#ccc']]:stroke-border [&\_.recharts-radial-bar-background-sector]:fill-muted [&\_.recharts-rectangle.recharts-tooltip-cursor]:fill-muted [&\_.recharts-reference-line\_[stroke='#ccc']]:stroke-border [&\_.recharts-sector[stroke='#fff']]:stroke-transparent [&\_.recharts-sector]:outline-none [&\_.recharts-surface]:outline-none",  
 className  
 )}  
 {...props}  
 >  
 <ChartStyle id={chartId} config={config} />  
 <RechartsPrimitive.ResponsiveContainer>  
 {children}  
 </RechartsPrimitive.ResponsiveContainer>  
 </div>  
 </ChartContext.Provider>  
 )  
})  
ChartContainer.displayName = "Chart"  
  
const ChartStyle = ({ id, config }: { id: string; config: ChartConfig }) => {  
 const colorConfig = Object.entries(config).filter(  
 ([\_, config]) => config.theme || config.color  
 )  
  
 if (!colorConfig.length) {  
 return null  
 }  
  
 return (  
 <style  
 dangerouslySetInnerHTML={{  
 \_\_html: Object.entries(THEMES)  
 .map(  
 ([theme, prefix]) => `  
${prefix} [data-chart=${id}] {  
${colorConfig  
 .map(([key, itemConfig]) => {  
 const color =  
 itemConfig.theme?.[theme as keyof typeof itemConfig.theme] ||  
 itemConfig.color  
 return color ? ` --color-${key}: ${color};` : null  
 })  
 .join("\n")}  
}  
`  
 )  
 .join("\n"),  
 }}  
 />  
 )  
}  
  
const ChartTooltip = RechartsPrimitive.Tooltip  
  
const ChartTooltipContent = React.forwardRef<  
 HTMLDivElement,  
 React.ComponentProps<typeof RechartsPrimitive.Tooltip> &  
 React.ComponentProps<"div"> & {  
 hideLabel?: boolean  
 hideIndicator?: boolean  
 indicator?: "line" | "dot" | "dashed"  
 nameKey?: string  
 labelKey?: string  
 }  
>(  
 (  
 {  
 active,  
 payload,  
 className,  
 indicator = "dot",  
 hideLabel = false,  
 hideIndicator = false,  
 label,  
 labelFormatter,  
 labelClassName,  
 formatter,  
 color,  
 nameKey,  
 labelKey,  
 },  
 ref  
 ) => {  
 const { config } = useChart()  
  
 const tooltipLabel = React.useMemo(() => {  
 if (hideLabel || !payload?.length) {  
 return null  
 }  
  
 const [item] = payload  
 const key = `${labelKey || item.dataKey || item.name || "value"}`  
 const itemConfig = getPayloadConfigFromPayload(config, item, key)  
 const value =  
 !labelKey && typeof label === "string"  
 ? config[label as keyof typeof config]?.label || label  
 : itemConfig?.label  
  
 if (labelFormatter) {  
 return (  
 <div className={cn("font-medium", labelClassName)}>  
 {labelFormatter(value, payload)}  
 </div>  
 )  
 }  
  
 if (!value) {  
 return null  
 }  
  
 return <div className={cn("font-medium", labelClassName)}>{value}</div>  
 }, [  
 label,  
 labelFormatter,  
 payload,  
 hideLabel,  
 labelClassName,  
 config,  
 labelKey,  
 ])  
  
 if (!active || !payload?.length) {  
 return null  
 }  
  
 const nestLabel = payload.length === 1 && indicator !== "dot"  
  
 return (  
 <div  
 ref={ref}  
 className={cn(  
 "grid min-w-[8rem] items-start gap-1.5 rounded-lg border border-border/50 bg-background px-2.5 py-1.5 text-xs shadow-xl",  
 className  
 )}  
 >  
 {!nestLabel ? tooltipLabel : null}  
 <div className="grid gap-1.5">  
 {payload.map((item, index) => {  
 const key = `${nameKey || item.name || item.dataKey || "value"}`  
 const itemConfig = getPayloadConfigFromPayload(config, item, key)  
 const indicatorColor = color || item.payload.fill || item.color  
  
 return (  
 <div  
 key={item.dataKey}  
 className={cn(  
 "flex w-full flex-wrap items-stretch gap-2 [&>svg]:h-2.5 [&>svg]:w-2.5 [&>svg]:text-muted-foreground",  
 indicator === "dot" && "items-center"  
 )}  
 >  
 {formatter && item?.value !== undefined && item.name ? (  
 formatter(item.value, item.name, item, index, item.payload)  
 ) : (  
 <>  
 {itemConfig?.icon ? (  
 <itemConfig.icon />  
 ) : (  
 !hideIndicator && (  
 <div  
 className={cn(  
 "shrink-0 rounded-[2px] border-[--color-border] bg-[--color-bg]",  
 {  
 "h-2.5 w-2.5": indicator === "dot",  
 "w-1": indicator === "line",  
 "w-0 border-[1.5px] border-dashed bg-transparent":  
 indicator === "dashed",  
 "my-0.5": nestLabel && indicator === "dashed",  
 }  
 )}  
 style={  
 {  
 "--color-bg": indicatorColor,  
 "--color-border": indicatorColor,  
 } as React.CSSProperties  
 }  
 />  
 )  
 )}  
 <div  
 className={cn(  
 "flex flex-1 justify-between leading-none",  
 nestLabel ? "items-end" : "items-center"  
 )}  
 >  
 <div className="grid gap-1.5">  
 {nestLabel ? tooltipLabel : null}  
 <span className="text-muted-foreground">  
 {itemConfig?.label || item.name}  
 </span>  
 </div>  
 {item.value && (  
 <span className="font-mono font-medium tabular-nums text-foreground">  
 {item.value.toLocaleString()}  
 </span>  
 )}  
 </div>  
 </>  
 )}  
 </div>  
 )  
 })}  
 </div>  
 </div>  
 )  
 }  
)  
ChartTooltipContent.displayName = "ChartTooltip"  
  
const ChartLegend = RechartsPrimitive.Legend  
  
const ChartLegendContent = React.forwardRef<  
 HTMLDivElement,  
 React.ComponentProps<"div"> &  
 Pick<RechartsPrimitive.LegendProps, "payload" | "verticalAlign"> & {  
 hideIcon?: boolean  
 nameKey?: string  
 }  
>(  
 (  
 { className, hideIcon = false, payload, verticalAlign = "bottom", nameKey },  
 ref  
 ) => {  
 const { config } = useChart()  
  
 if (!payload?.length) {  
 return null  
 }  
  
 return (  
 <div  
 ref={ref}  
 className={cn(  
 "flex items-center justify-center gap-4",  
 verticalAlign === "top" ? "pb-3" : "pt-3",  
 className  
 )}  
 >  
 {payload.map((item) => {  
 const key = `${nameKey || item.dataKey || "value"}`  
 const itemConfig = getPayloadConfigFromPayload(config, item, key)  
  
 return (  
 <div  
 key={item.value}  
 className={cn(  
 "flex items-center gap-1.5 [&>svg]:h-3 [&>svg]:w-3 [&>svg]:text-muted-foreground"  
 )}  
 >  
 {itemConfig?.icon && !hideIcon ? (  
 <itemConfig.icon />  
 ) : (  
 <div  
 className="h-2 w-2 shrink-0 rounded-[2px]"  
 style={{  
 backgroundColor: item.color,  
 }}  
 />  
 )}  
 {itemConfig?.label}  
 </div>  
 )  
 })}  
 </div>  
 )  
 }  
)  
ChartLegendContent.displayName = "ChartLegend"  
  
// Helper to extract item config from a payload.  
function getPayloadConfigFromPayload(  
 config: ChartConfig,  
 payload: unknown,  
 key: string  
) {  
 if (typeof payload !== "object" || payload === null) {  
 return undefined  
 }  
  
 const payloadPayload =  
 "payload" in payload &&  
 typeof payload.payload === "object" &&  
 payload.payload !== null  
 ? payload.payload  
 : undefined  
  
 let configLabelKey: string = key  
  
 if (  
 key in payload &&  
 typeof payload[key as keyof typeof payload] === "string"  
 ) {  
 configLabelKey = payload[key as keyof typeof payload] as string  
 } else if (  
 payloadPayload &&  
 key in payloadPayload &&  
 typeof payloadPayload[key as keyof typeof payloadPayload] === "string"  
 ) {  
 configLabelKey = payloadPayload[  
 key as keyof typeof payloadPayload  
 ] as string  
 }  
  
 return configLabelKey in config  
 ? config[configLabelKey]  
 : config[key as keyof typeof config]  
}  
  
export {  
 ChartContainer,  
 ChartTooltip,  
 ChartTooltipContent,  
 ChartLegend,  
 ChartLegendContent,  
 ChartStyle,  
}