# Personal Task Manager Documentation

## Table of Contents

1. [Project Overview](#project-overview)
2. [Technology Stack](#technology-stack)
3. [Project Structure](#project-structure)
4. [Database Schema](#database-schema)
5. [Authentication System](#authentication-system)
6. [Server-Side Implementation](#server-side-implementation)
7. [Client-Side Implementation](#client-side-implementation)
8. [UI Components](#ui-components)
9. [Task Management Features](#task-management-features)
10. [Styling and Theming](#styling-and-theming)
11. [Deployment Guidelines](#deployment-guidelines)

## Project Overview

The Personal Task Manager is a full-stack web application that enables users to manage their personal tasks efficiently. Users can sign up, log in, and access a private dashboard where they can create, edit, delete, and track the status of their tasks. The application provides a clean, responsive interface with modern UI components, and ensures data separation between different user accounts.

Key features include: - User authentication (register, login, logout) - Create, read, update, and delete (CRUD) operations for tasks - Task filtering (all, pending, completed) - Real-time status updates - Responsive design for multiple device types - Color-coded UI elements for better visual recognition

## Technology Stack

The application leverages the following technologies:

### Frontend

* **React**: JavaScript library for building user interfaces
* **TypeScript**: Superset of JavaScript with static typing
* **Vite**: Next-generation frontend tooling
* **Shadcn UI**: Component library built on Radix UI
* **Tailwind CSS**: Utility-first CSS framework
* **TanStack Query (React Query)**: Data fetching and state management
* **Wouter**: Small router for React applications
* **Zod**: TypeScript-first schema validation

### Backend

* **Node.js**: JavaScript runtime environment
* **Express**: Web application framework
* **Drizzle ORM**: TypeScript ORM for databases
* **Passport.js**: Authentication middleware for Node.js

### Database

* **In-memory storage** (development)
* **PostgreSQL** (production-ready configuration)

## Project Structure

The project follows a modern full-stack application structure, separating client and server code:

/  
├── client/ # Frontend React application  
│ ├── src/  
│ │ ├── components/ # Reusable UI components  
│ │ ├── hooks/ # Custom React hooks  
│ │ ├── lib/ # Utility functions  
│ │ ├── pages/ # Application pages  
│ │ ├── App.tsx # Main application component  
│ │ └── main.tsx # Entry point  
├── server/ # Backend Express application  
│ ├── auth.ts # Authentication logic  
│ ├── index.ts # Server entry point  
│ ├── routes.ts # API endpoints  
│ ├── storage.ts # Data storage interface  
│ └── vite.ts # Development server configuration  
├── shared/ # Shared code between client and server  
│ └── schema.ts # Database schema and types

## Database Schema

The database schema is defined in shared/schema.ts using Drizzle ORM with PostgreSQL adapter. This ensures type safety and consistency between the frontend and backend.

// shared/schema.ts  
import { pgTable, serial, text, timestamp, boolean } from 'drizzle-orm/pg-core';  
import { createInsertSchema } from 'drizzle-zod';  
import { z } from 'zod';  
  
// Users table  
export const users = pgTable("users", {  
 id: serial("id").primaryKey(),  
 name: text("name").notNull(),  
 email: text("email").notNull().unique(),  
 password: text("password").notNull(),  
});  
  
// Tasks table  
export const tasks = pgTable("tasks", {  
 id: serial("id").primaryKey(),  
 title: text("title").notNull(),  
 description: text("description"),  
 status: text("status").notNull().default("pending"),  
 userId: serial("user\_id").references(() => users.id).notNull(),  
 createdAt: timestamp("created\_at").defaultNow().notNull(),  
});  
  
// Schemas for input validation  
export const insertUserSchema = createInsertSchema(users).pick({  
 name: true,  
 email: true,  
 password: true,  
});  
  
export const insertTaskSchema = createInsertSchema(tasks).pick({  
 title: true,  
 description: true,  
 status: true,  
});  
  
export const updateTaskSchema = createInsertSchema(tasks).pick({  
 title: true,  
 description: true,  
 status: true,  
});  
  
// TypeScript types  
export type InsertUser = z.infer<typeof insertUserSchema>;  
export type User = typeof users.$inferSelect;  
export type InsertTask = z.infer<typeof insertTaskSchema>;  
export type Task = typeof tasks.$inferSelect;  
export type UpdateTask = z.infer<typeof updateTaskSchema>;

## Authentication System

The authentication system is implemented using Passport.js with a local strategy. User credentials are stored securely, with passwords hashed using scrypt.

### Server-Side Authentication

// server/auth.ts  
import passport from "passport";  
import { Strategy as LocalStrategy } from "passport-local";  
import { Express } from "express";  
import session from "express-session";  
import { scrypt, randomBytes, timingSafeEqual } from "crypto";  
import { promisify } from "util";  
import { storage } from "./storage";  
import { User as SelectUser } from "@shared/schema";  
  
// Password hashing  
async function hashPassword(password: string) {  
 const salt = randomBytes(16).toString("hex");  
 const buf = (await scryptAsync(password, salt, 64)) as Buffer;  
 return `${buf.toString("hex")}.${salt}`;  
}  
  
// Password verification  
async function comparePasswords(supplied: string, stored: string) {  
 const [hashed, salt] = stored.split(".");  
 const hashedBuf = Buffer.from(hashed, "hex");  
 const suppliedBuf = (await scryptAsync(supplied, salt, 64)) as Buffer;  
 return timingSafeEqual(hashedBuf, suppliedBuf);  
}  
  
export function setupAuth(app: Express) {  
 // Session setup  
 const sessionSettings: session.SessionOptions = {  
 secret: process.env.SESSION\_SECRET!,  
 resave: false,  
 saveUninitialized: false,  
 store: storage.sessionStore,  
 };  
  
 app.use(session(sessionSettings));  
 app.use(passport.initialize());  
 app.use(passport.session());  
  
 // Local strategy configuration  
 passport.use(  
 new LocalStrategy({  
 usernameField: 'email',  
 passwordField: 'password',  
 }, async (email, password, done) => {  
 const user = await storage.getUserByEmail(email);  
 if (!user || !(await comparePasswords(password, user.password))) {  
 return done(null, false, { message: "Invalid email or password" });  
 } else {  
 return done(null, user);  
 }  
 }),  
 );  
  
 // Serialization for session  
 passport.serializeUser((user, done) => done(null, user.id));  
 passport.deserializeUser(async (id: number, done) => {  
 const user = await storage.getUser(id);  
 done(null, user);  
 });  
  
 // Auth routes  
 app.post("/api/register", async (req, res, next) => {  
 const existingUser = await storage.getUserByEmail(req.body.email);  
 if (existingUser) {  
 return res.status(400).json({ message: "Email already in use" });  
 }  
  
 const user = await storage.createUser({  
 ...req.body,  
 password: await hashPassword(req.body.password),  
 });  
  
 req.login(user, (err) => {  
 if (err) return next(err);  
 res.status(201).json(user);  
 });  
 });  
  
 app.post("/api/login", passport.authenticate("local"), (req, res) => {  
 res.status(200).json(req.user);  
 });  
  
 app.post("/api/logout", (req, res, next) => {  
 req.logout((err) => {  
 if (err) return next(err);  
 res.sendStatus(200);  
 });  
 });  
  
 app.get("/api/user", (req, res) => {  
 if (!req.isAuthenticated()) return res.sendStatus(401);  
 res.json(req.user);  
 });  
}

### Client-Side Authentication

The client-side authentication uses a custom useAuth hook that manages authentication state and provides login, register, and logout functionality:

// client/src/hooks/use-auth.tsx  
import { createContext, ReactNode, useContext } from "react";  
import {  
 useQuery,  
 useMutation,  
 UseMutationResult,  
} from "@tanstack/react-query";  
import { InsertUser, User as SelectUser } from "@shared/schema";  
import { getQueryFn, apiRequest, queryClient } from "../lib/queryClient";  
import { useToast } from "@/hooks/use-toast";  
  
type AuthContextType = {  
 user: SelectUser | null;  
 isLoading: boolean;  
 error: Error | null;  
 loginMutation: UseMutationResult<SelectUser, Error, LoginData>;  
 logoutMutation: UseMutationResult<void, Error, void>;  
 registerMutation: UseMutationResult<SelectUser, Error, InsertUser>;  
};  
  
type LoginData = {  
 email: string;  
 password: string;  
};  
  
export const AuthContext = createContext<AuthContextType | null>(null);  
  
export function AuthProvider({ children }: { children: ReactNode }) {  
 const { toast } = useToast();  
   
 // Get current user  
 const {  
 data: user,  
 error,  
 isLoading,  
 } = useQuery<SelectUser | null, Error>({  
 queryKey: ["/api/user"],  
 queryFn: getQueryFn({ on401: "returnNull" }),  
 });  
  
 // Login mutation  
 const loginMutation = useMutation({  
 mutationFn: async (credentials: LoginData) => {  
 const res = await apiRequest("POST", "/api/login", credentials);  
 return await res.json();  
 },  
 onSuccess: (user: SelectUser) => {  
 queryClient.setQueryData(["/api/user"], user);  
 // Clear any existing tasks from cache on login  
 queryClient.removeQueries({ queryKey: ["/api/tasks"] });  
 toast({  
 title: "Login successful",  
 description: `Welcome back, ${user.name}!`,  
 variant: "success",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Login failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 // Register mutation  
 const registerMutation = useMutation({  
 mutationFn: async (userData: InsertUser) => {  
 const res = await apiRequest("POST", "/api/register", userData);  
 return await res.json();  
 },  
 onSuccess: (user: SelectUser) => {  
 queryClient.setQueryData(["/api/user"], user);  
 // Clear any existing tasks from cache on registration  
 queryClient.removeQueries({ queryKey: ["/api/tasks"] });  
 toast({  
 title: "Registration successful",  
 description: `Welcome, ${user.name}!`,  
 variant: "success",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Registration failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 // Logout mutation  
 const logoutMutation = useMutation({  
 mutationFn: async () => {  
 await apiRequest("POST", "/api/logout");  
 },  
 onSuccess: () => {  
 queryClient.setQueryData(["/api/user"], null);  
 // Clear any existing tasks from cache on logout  
 queryClient.removeQueries({ queryKey: ["/api/tasks"] });  
 toast({  
 title: "Logged out",  
 description: "You have been successfully logged out.",  
 variant: "info",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Logout failed",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 return (  
 <AuthContext.Provider  
 value={{  
 user: user ?? null,  
 isLoading,  
 error,  
 loginMutation,  
 logoutMutation,  
 registerMutation,  
 }}  
 >  
 {children}  
 </AuthContext.Provider>  
 );  
}  
  
export function useAuth() {  
 const context = useContext(AuthContext);  
 if (!context) {  
 throw new Error("useAuth must be used within an AuthProvider");  
 }  
 return context;  
}

## Server-Side Implementation

### API Routes

The server exposes several API endpoints for task management, protected by authentication:

// server/routes.ts  
import { Express, Request, Response } from "express";  
import { createServer, Server } from "http";  
import { setupAuth } from "./auth";  
import { storage } from "./storage";  
import { insertTaskSchema, updateTaskSchema } from "@shared/schema";  
  
export async function registerRoutes(app: Express): Promise<Server> {  
 // Set up authentication routes  
 setupAuth(app);  
  
 // Middleware to ensure user is authenticated  
 const isAuthenticated = (req: Request, res: Response, next: Function) => {  
 if (req.isAuthenticated()) {  
 return next();  
 }  
 res.sendStatus(401);  
 };  
  
 // Task routes  
 app.get("/api/tasks", isAuthenticated, async (req, res) => {  
 const tasks = await storage.getTasksByUserId(req.user!.id);  
 res.json(tasks);  
 });  
  
 app.post("/api/tasks", isAuthenticated, async (req, res) => {  
 const parseResult = insertTaskSchema.safeParse(req.body);  
 if (!parseResult.success) {  
 return res.status(400).json({ errors: parseResult.error.format() });  
 }  
  
 const task = await storage.createTask({  
 ...parseResult.data,  
 userId: req.user!.id,  
 });  
  
 res.status(201).json(task);  
 });  
  
 app.put("/api/tasks/:id", isAuthenticated, async (req, res) => {  
 const taskId = parseInt(req.params.id);  
 const task = await storage.getTaskById(taskId);  
  
 if (!task) {  
 return res.status(404).json({ message: "Task not found" });  
 }  
  
 if (task.userId !== req.user!.id) {  
 return res.status(403).json({ message: "Not authorized" });  
 }  
  
 const parseResult = updateTaskSchema.safeParse(req.body);  
 if (!parseResult.success) {  
 return res.status(400).json({ errors: parseResult.error.format() });  
 }  
  
 const updatedTask = await storage.updateTask(taskId, parseResult.data);  
 res.json(updatedTask);  
 });  
  
 app.delete("/api/tasks/:id", isAuthenticated, async (req, res) => {  
 const taskId = parseInt(req.params.id);  
 const task = await storage.getTaskById(taskId);  
  
 if (!task) {  
 return res.status(404).json({ message: "Task not found" });  
 }  
  
 if (task.userId !== req.user!.id) {  
 return res.status(403).json({ message: "Not authorized" });  
 }  
  
 await storage.deleteTask(taskId);  
 res.sendStatus(204);  
 });  
  
 const httpServer = createServer(app);  
 return httpServer;  
}

### Storage Interface

The application uses a memory-based storage system for development, with a clear interface that could be adapted for different database systems:

// server/storage.ts  
import { InsertUser, User, InsertTask, Task, UpdateTask } from "@shared/schema";  
import session from "express-session";  
import createMemoryStore from "memorystore";  
  
const MemoryStore = createMemoryStore(session);  
  
export interface IStorage {  
 getUser(id: number): Promise<User | undefined>;  
 getUserByEmail(email: string): Promise<User | undefined>;  
 createUser(user: InsertUser): Promise<User>;  
   
 getTasksByUserId(userId: number): Promise<Task[]>;  
 getTaskById(id: number): Promise<Task | undefined>;  
 createTask(task: InsertTask): Promise<Task>;  
 updateTask(id: number, taskData: UpdateTask): Promise<Task>;  
 deleteTask(id: number): Promise<void>;  
   
 sessionStore: session.SessionStore;  
}  
  
export class MemStorage implements IStorage {  
 private users: Map<number, User>;  
 private tasks: Map<number, Task>;  
 sessionStore: session.SessionStore;  
 currentUserId: number;  
 currentTaskId: number;  
  
 constructor() {  
 this.users = new Map();  
 this.tasks = new Map();  
 this.currentUserId = 1;  
 this.currentTaskId = 1;  
 this.sessionStore = new MemoryStore({  
 checkPeriod: 86400000, // Clear expired sessions every day  
 });  
 }  
  
 async getUser(id: number): Promise<User | undefined> {  
 return this.users.get(id);  
 }  
  
 async getUserByEmail(email: string): Promise<User | undefined> {  
 for (const user of this.users.values()) {  
 if (user.email === email) {  
 return user;  
 }  
 }  
 return undefined;  
 }  
  
 async createUser(insertUser: InsertUser): Promise<User> {  
 const id = this.currentUserId++;  
 const user: User = { ...insertUser, id };  
 this.users.set(id, user);  
 return user;  
 }  
  
 async getTasksByUserId(userId: number): Promise<Task[]> {  
 const userTasks: Task[] = [];  
 for (const task of this.tasks.values()) {  
 if (task.userId === userId) {  
 userTasks.push(task);  
 }  
 }  
 return userTasks;  
 }  
  
 async getTaskById(id: number): Promise<Task | undefined> {  
 return this.tasks.get(id);  
 }  
  
 async createTask(insertTask: InsertTask): Promise<Task> {  
 const id = this.currentTaskId++;  
 const createdAt = new Date();  
   
 const task: Task = {   
 ...insertTask,   
 id,  
 createdAt,  
 };  
   
 this.tasks.set(id, task);  
 return task;  
 }  
  
 async updateTask(id: number, taskData: UpdateTask): Promise<Task> {  
 const task = this.tasks.get(id);  
 if (!task) {  
 throw new Error("Task not found");  
 }  
  
 const updatedTask = { ...task, ...taskData };  
 this.tasks.set(id, updatedTask);  
 return updatedTask;  
 }  
  
 async deleteTask(id: number): Promise<void> {  
 this.tasks.delete(id);  
 }  
}  
  
export const storage = new MemStorage();

## Client-Side Implementation

### Routing

The application uses Wouter for routing, with a protected route component to handle authentication:

// client/src/App.tsx  
import { Switch, Route } from "wouter";  
import { QueryClientProvider } from "@tanstack/react-query";  
import { queryClient } from "./lib/queryClient";  
import { AuthProvider } from "./hooks/use-auth";  
import { Toaster } from "@/components/ui/toaster";  
import HomePage from "@/pages/home-page";  
import NotFound from "@/pages/not-found";  
import AuthPage from "@/pages/auth-page";  
import { ProtectedRoute } from "./lib/protected-route";  
  
function Router() {  
 return (  
 <Switch>  
 <ProtectedRoute path="/" component={HomePage} />  
 <Route path="/auth" component={AuthPage} />  
 <Route component={NotFound} />  
 </Switch>  
 );  
}  
  
export default function App() {  
 return (  
 <QueryClientProvider client={queryClient}>  
 <AuthProvider>  
 <Router />  
 <Toaster />  
 </AuthProvider>  
 </QueryClientProvider>  
 );  
}

### Protected Route Component

The ProtectedRoute component ensures that only authenticated users can access certain routes:

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

### Authentication Page

The authentication page includes both login and registration forms:

// client/src/pages/auth-page.tsx  
import { useAuth } from "@/hooks/use-auth";  
import { z } from "zod";  
import { useForm } from "react-hook-form";  
import { zodResolver } from "@hookform/resolvers/zod";  
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";  
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";  
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";  
import { Input } from "@/components/ui/input";  
import { Button } from "@/components/ui/button";  
import { Loader2 } from "lucide-react";  
import { useLocation } from "wouter";  
  
// Login form schema  
const loginSchema = z.object({  
 email: z.string().email({ message: "Please enter a valid email address" }),  
 password: z.string().min(6, { message: "Password must be at least 6 characters" })  
});  
  
// Registration form schema  
const registerSchema = z.object({  
 name: z.string().min(2, { message: "Name must be at least 2 characters" }),  
 email: z.string().email({ message: "Please enter a valid email address" }),  
 password: z.string().min(6, { message: "Password must be at least 6 characters" })  
});  
  
type LoginFormValues = z.infer<typeof loginSchema>;  
type RegisterFormValues = z.infer<typeof registerSchema>;  
  
export default function AuthPage() {  
 const { user, loginMutation, registerMutation } = useAuth();  
 const [location, navigate] = useLocation();  
   
 // Redirect if already logged in  
 if (user) {  
 navigate("/");  
 return null;  
 }  
   
 // Login form setup  
 const loginForm = useForm<LoginFormValues>({  
 resolver: zodResolver(loginSchema),  
 defaultValues: {  
 email: "",  
 password: "",  
 },  
 });  
   
 // Register form setup  
 const registerForm = useForm<RegisterFormValues>({  
 resolver: zodResolver(registerSchema),  
 defaultValues: {  
 name: "",  
 email: "",  
 password: "",  
 },  
 });  
   
 // Handle login submission  
 function onLoginSubmit(values: LoginFormValues) {  
 loginMutation.mutate(values);  
 }  
   
 // Handle registration submission  
 function onRegisterSubmit(values: RegisterFormValues) {  
 registerMutation.mutate(values);  
 }  
  
 return (  
 <div className="flex min-h-screen bg-slate-50">  
 <div className="flex flex-col justify-center flex-1 px-4 py-12 sm:px-6 lg:flex-none lg:px-20 xl:px-24">  
 <div className="w-full max-w-sm mx-auto lg:w-96">  
 <div className="mb-8">  
 <h1 className="text-3xl font-extrabold text-slate-900">TaskFlow</h1>  
 <p className="mt-2 text-sm text-slate-600">  
 Manage your tasks efficiently and stay organized  
 </p>  
 </div>  
   
 <Tabs defaultValue="login" className="w-full">  
 <TabsList className="grid w-full grid-cols-2 mb-4">  
 <TabsTrigger value="login">Login</TabsTrigger>  
 <TabsTrigger value="register">Register</TabsTrigger>  
 </TabsList>  
   
 <TabsContent value="login">  
 <Card>  
 <CardHeader>  
 <CardTitle>Login to your account</CardTitle>  
 <CardDescription>  
 Enter your credentials to access your tasks  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 <Form {...loginForm}>  
 <form onSubmit={loginForm.handleSubmit(onLoginSubmit)} className="space-y-4">  
 <FormField  
 control={loginForm.control}  
 name="email"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Email</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="yourname@example.com"   
 type="email"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={loginForm.control}  
 name="password"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Password</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="••••••••"   
 type="password"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <Button   
 type="submit"   
 className="w-full"   
 disabled={loginMutation.isPending}  
 >  
 {loginMutation.isPending ? (  
 <>  
 <Loader2 className="w-4 h-4 mr-2 animate-spin" />   
 Logging in...  
 </>  
 ) : (  
 "Login"  
 )}  
 </Button>  
 </form>  
 </Form>  
 </CardContent>  
 </Card>  
 </TabsContent>  
   
 <TabsContent value="register">  
 <Card>  
 <CardHeader>  
 <CardTitle>Create an account</CardTitle>  
 <CardDescription>  
 Sign up to start managing your tasks  
 </CardDescription>  
 </CardHeader>  
 <CardContent>  
 <Form {...registerForm}>  
 <form onSubmit={registerForm.handleSubmit(onRegisterSubmit)} className="space-y-4">  
 <FormField  
 control={registerForm.control}  
 name="name"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Full Name</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="John Doe"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={registerForm.control}  
 name="email"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Email</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="yourname@example.com"   
 type="email"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={registerForm.control}  
 name="password"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Password</FormLabel>  
 <FormControl>  
 <Input   
 placeholder="••••••••"   
 type="password"   
 {...field}   
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <Button   
 type="submit"   
 className="w-full"   
 disabled={registerMutation.isPending}  
 >  
 {registerMutation.isPending ? (  
 <>  
 <Loader2 className="w-4 h-4 mr-2 animate-spin" />   
 Creating account...  
 </>  
 ) : (  
 "Create Account"  
 )}  
 </Button>  
 </form>  
 </Form>  
 </CardContent>  
 </Card>  
 </TabsContent>  
 </Tabs>  
 </div>  
 </div>  
   
 <div className="relative flex-1 hidden w-0 lg:block">  
 <div className="absolute inset-0 flex flex-col justify-center p-16 bg-gradient-to-br from-primary/80 to-primary">  
 <div className="max-w-md p-8 mx-auto bg-white rounded-lg shadow-xl">  
 <h2 className="mb-4 text-2xl font-bold text-slate-900">Organize Your Life</h2>  
 <p className="mb-4 text-slate-600">  
 TaskFlow helps you manage your tasks, set priorities, and track your progress.   
 Stay organized, increase productivity, and never forget important deadlines.  
 </p>  
 <ul className="space-y-2 text-sm text-slate-600">  
 <li className="flex items-center">  
 <span className="inline-flex items-center justify-center w-5 h-5 mr-2 text-white bg-primary rounded-full">✓</span>  
 Create, edit, and organize tasks  
 </li>  
 <li className="flex items-center">  
 <span className="inline-flex items-center justify-center w-5 h-5 mr-2 text-white bg-primary rounded-full">✓</span>  
 Track task status and progress  
 </li>  
 <li className="flex items-center">  
 <span className="inline-flex items-center justify-center w-5 h-5 mr-2 text-white bg-primary rounded-full">✓</span>  
 Filter tasks by completion status  
 </li>  
 <li className="flex items-center">  
 <span className="inline-flex items-center justify-center w-5 h-5 mr-2 text-white bg-primary rounded-full">✓</span>  
 Secure and private task management  
 </li>  
 </ul>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
}

## UI Components

### Task Card Component

The TaskCard component displays individual tasks with options to edit, delete, and toggle status:

// client/src/components/task-card.tsx  
import { Card, CardContent } from "@/components/ui/card";  
import { Button } from "@/components/ui/button";  
import { Task } from "@shared/schema";  
import { Pencil, Trash, RefreshCw, CheckCircle } from "lucide-react";  
import { Badge } from "@/components/ui/badge";  
  
type TaskCardProps = {  
 task: Task;  
 onEdit: () => void;  
 onDelete: () => void;  
 onStatusToggle: () => void;  
};  
  
export function TaskCard({ task, onEdit, onDelete, onStatusToggle }: TaskCardProps) {  
 const isCompleted = task.status === "completed";  
   
 return (  
 <Card className="shadow-sm hover:shadow-md transition-all">  
 <CardContent className="p-5">  
 <div className="flex justify-between items-start mb-3">  
 <h3 className="font-medium text-slate-900 text-base">{task.title}</h3>  
 <div className="flex gap-1">  
 <Button   
 variant="ghost"   
 size="icon"   
 className="h-8 w-8 text-blue-500 hover:text-blue-700"   
 onClick={onEdit}  
 title="Edit task"  
 >  
 <Pencil className="h-4 w-4" />  
 </Button>  
 <Button   
 variant="ghost"   
 size="icon"   
 className="h-8 w-8 text-red-500 hover:text-red-700"   
 onClick={onDelete}  
 title="Delete task"  
 >  
 <Trash className="h-4 w-4" />  
 </Button>  
 </div>  
 </div>  
   
 <p className="text-sm text-slate-600 mb-4">  
 {task.description || "No description"}  
 </p>  
   
 <div className="flex items-center justify-between">  
 <Badge variant={isCompleted ? "success" : "pending"}>  
 {isCompleted ? "Completed" : "Pending"}  
 </Badge>  
 <Button   
 variant="ghost"   
 size="sm"   
 className={`text-xs flex items-center ${  
 isCompleted   
 ? "text-slate-600 hover:text-amber-600"   
 : "text-slate-600 hover:text-emerald-600"  
 }`}  
 onClick={onStatusToggle}  
 >  
 {isCompleted ? (  
 <>  
 <RefreshCw className="h-3 w-3 mr-1" /> Mark pending  
 </>  
 ) : (  
 <>  
 <CheckCircle className="h-3 w-3 mr-1" /> Mark completed  
 </>  
 )}  
 </Button>  
 </div>  
 </CardContent>  
 </Card>  
 );  
}

### Task Modals

Multiple dialog components for creating, editing and deleting tasks:

// client/src/components/task-modals.tsx  
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogDescription, DialogFooter } from "@/components/ui/dialog";  
import { AlertDialog, AlertDialogAction, AlertDialogCancel, AlertDialogContent, AlertDialogDescription, AlertDialogFooter, AlertDialogHeader, AlertDialogTitle } from "@/components/ui/alert-dialog";  
import { Button } from "@/components/ui/button";  
import { Loader2 } from "lucide-react";  
import { useMutation } from "@tanstack/react-query";  
import { useAuth } from "@/hooks/use-auth";  
import { apiRequest, queryClient } from "@/lib/queryClient";  
import { Task } from "@shared/schema";  
import { useToast } from "@/hooks/use-toast";  
import { TaskForm, TaskFormValues } from "./task-form";  
  
// New Task Dialog Component  
type NewTaskDialogProps = {  
 isOpen: boolean;  
 onOpenChange: (open: boolean) => void;  
};  
  
export function NewTaskDialog({ isOpen, onOpenChange }: NewTaskDialogProps) {  
 const { user } = useAuth();  
 const { toast } = useToast();  
   
 const createTaskMutation = useMutation({  
 mutationFn: async (taskData: TaskFormValues) => {  
 const res = await apiRequest("POST", "/api/tasks", {  
 ...taskData,  
 userId: user?.id,  
 });  
 return res.json();  
 },  
 onSuccess: () => {  
 queryClient.invalidateQueries({ queryKey: ["/api/tasks"] });  
 onOpenChange(false);  
 toast({  
 title: "Task created",  
 description: "Your new task has been created successfully.",  
 variant: "success",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Failed to create task",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 const handleSubmit = (data: TaskFormValues) => {  
 createTaskMutation.mutate(data);  
 };  
  
 return (  
 <Dialog open={isOpen} onOpenChange={onOpenChange}>  
 <DialogContent className="sm:max-w-[425px]">  
 <DialogHeader>  
 <DialogTitle>Create New Task</DialogTitle>  
 <DialogDescription>  
 Add a new task to your personal task list.  
 </DialogDescription>  
 </DialogHeader>  
   
 <div className="py-4">  
 <TaskForm   
 onSubmit={handleSubmit}  
 isPending={createTaskMutation.isPending}  
 />  
 </div>  
   
 <DialogFooter>  
 <Button  
 variant="outline"  
 onClick={() => onOpenChange(false)}  
 disabled={createTaskMutation.isPending}  
 >  
 Cancel  
 </Button>  
 <Button  
 type="submit"  
 onClick={() => {  
 document.querySelector('form')?.dispatchEvent(  
 new Event('submit', { cancelable: true, bubbles: true })  
 );  
 }}  
 disabled={createTaskMutation.isPending}  
 >  
 {createTaskMutation.isPending ? (  
 <>  
 <Loader2 className="mr-2 h-4 w-4 animate-spin" />  
 Creating...  
 </>  
 ) : (  
 "Create Task"  
 )}  
 </Button>  
 </DialogFooter>  
 </DialogContent>  
 </Dialog>  
 );  
}  
  
// Edit Task Dialog Component  
type EditTaskDialogProps = {  
 isOpen: boolean;  
 onOpenChange: (open: boolean) => void;  
 task: Task;  
 onUpdateTask: (taskData: TaskFormValues) => void;  
 isPending: boolean;  
};  
  
export function EditTaskDialog({   
 isOpen,   
 onOpenChange,   
 task,   
 onUpdateTask,  
 isPending  
}: EditTaskDialogProps) {  
 return (  
 <Dialog open={isOpen} onOpenChange={onOpenChange}>  
 <DialogContent className="sm:max-w-[425px]">  
 <DialogHeader>  
 <DialogTitle>Edit Task</DialogTitle>  
 <DialogDescription>  
 Make changes to your task here.  
 </DialogDescription>  
 </DialogHeader>  
   
 <div className="py-4">  
 <TaskForm   
 defaultValues={{  
 title: task.title,  
 description: task.description || "",  
 status: task.status as "pending" | "completed",  
 }}  
 onSubmit={onUpdateTask}  
 isPending={isPending}  
 />  
 </div>  
   
 <DialogFooter>  
 <Button  
 variant="outline"  
 onClick={() => onOpenChange(false)}  
 disabled={isPending}  
 >  
 Cancel  
 </Button>  
 <Button  
 type="submit"  
 onClick={() => {  
 document.querySelector('form')?.dispatchEvent(  
 new Event('submit', { cancelable: true, bubbles: true })  
 );  
 }}  
 disabled={isPending}  
 >  
 {isPending ? (  
 <>  
 <Loader2 className="mr-2 h-4 w-4 animate-spin" />  
 Updating...  
 </>  
 ) : (  
 "Update Task"  
 )}  
 </Button>  
 </DialogFooter>  
 </DialogContent>  
 </Dialog>  
 );  
}  
  
// Delete Task Dialog Component  
type DeleteTaskDialogProps = {  
 isOpen: boolean;  
 onOpenChange: (open: boolean) => void;  
 onConfirmDelete: () => void;  
 isPending: boolean;  
};  
  
export function DeleteTaskDialog({   
 isOpen,   
 onOpenChange,   
 onConfirmDelete,  
 isPending  
}: DeleteTaskDialogProps) {  
 return (  
 <AlertDialog open={isOpen} onOpenChange={onOpenChange}>  
 <AlertDialogContent>  
 <AlertDialogHeader>  
 <AlertDialogTitle>Confirm Deletion</AlertDialogTitle>  
 <AlertDialogDescription>  
 Are you sure you want to delete this task? This action cannot be undone.  
 </AlertDialogDescription>  
 </AlertDialogHeader>  
 <AlertDialogFooter>  
 <AlertDialogCancel disabled={isPending}>Cancel</AlertDialogCancel>  
 <AlertDialogAction  
 onClick={(e) => {  
 e.preventDefault();  
 onConfirmDelete();  
 }}  
 disabled={isPending}  
 className="bg-red-600 hover:bg-red-700 text-white"  
 >  
 {isPending ? (  
 <>  
 <Loader2 className="mr-2 h-4 w-4 animate-spin" />  
 Deleting...  
 </>  
 ) : (  
 "Delete"  
 )}  
 </AlertDialogAction>  
 </AlertDialogFooter>  
 </AlertDialogContent>  
 </AlertDialog>  
 );  
}

### Task Form Component

The task form component used for both creating and editing tasks:

// client/src/components/task-form.tsx  
import { z } from "zod";  
import { useForm } from "react-hook-form";  
import { zodResolver } from "@hookform/resolvers/zod";  
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";  
import { Input } from "@/components/ui/input";  
import { Textarea } from "@/components/ui/textarea";  
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";  
  
// Task form schema  
const taskFormSchema = z.object({  
 title: z.string().min(1, "Title is required").max(100, "Title is too long"),  
 description: z.string().optional(),  
 status: z.enum(["pending", "completed"]).default("pending"),  
});  
  
export type TaskFormValues = z.infer<typeof taskFormSchema>;  
  
type TaskFormProps = {  
 defaultValues?: Partial<TaskFormValues>;  
 onSubmit: (data: TaskFormValues) => void;  
 isPending?: boolean;  
};  
  
export function TaskForm({ defaultValues, onSubmit, isPending = false }: TaskFormProps) {  
 // Form with validation  
 const form = useForm<TaskFormValues>({  
 resolver: zodResolver(taskFormSchema),  
 defaultValues: {  
 title: "",  
 description: "",  
 status: "pending",  
 ...defaultValues,  
 },  
 });  
  
 return (  
 <Form {...form}>  
 <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4">  
 <FormField  
 control={form.control}  
 name="title"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Title</FormLabel>  
 <FormControl>  
 <Input placeholder="Task title" {...field} disabled={isPending} />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={form.control}  
 name="description"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Description</FormLabel>  
 <FormControl>  
 <Textarea   
 placeholder="Task details (optional)"   
 {...field}   
 disabled={isPending}  
 />  
 </FormControl>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
   
 <FormField  
 control={form.control}  
 name="status"  
 render={({ field }) => (  
 <FormItem>  
 <FormLabel>Status</FormLabel>  
 <Select   
 onValueChange={field.onChange}  
 defaultValue={field.value}  
 disabled={isPending}  
 >  
 <FormControl>  
 <SelectTrigger>  
 <SelectValue placeholder="Select status" />  
 </SelectTrigger>  
 </FormControl>  
 <SelectContent>  
 <SelectItem value="pending">Pending</SelectItem>  
 <SelectItem value="completed">Completed</SelectItem>  
 </SelectContent>  
 </Select>  
 <FormMessage />  
 </FormItem>  
 )}  
 />  
 </form>  
 </Form>  
 );  
}

### Navbar Component

The navigation bar component with logout functionality:

// client/src/components/navbar.tsx  
import { useAuth } from "@/hooks/use-auth";  
import { Button } from "@/components/ui/button";  
import { ListTodo } from "lucide-react";  
  
export function Navbar() {  
 const { user, logoutMutation } = useAuth();  
  
 const handleLogout = () => {  
 logoutMutation.mutate();  
 };  
  
 return (  
 <header className="bg-white shadow-sm border-b border-slate-200 sticky top-0 z-10">  
 <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-4">  
 <div className="flex justify-between items-center">  
 <div className="flex items-center">  
 <ListTodo className="text-primary h-6 w-6 mr-2" />  
 <h1 className="text-xl font-semibold text-slate-900">TaskFlow</h1>  
 </div>  
 <div className="flex items-center space-x-4">  
 <div className="flex items-center">  
 {user && (  
 <>  
 <span className="text-sm text-slate-700 mr-3">{user.name}</span>  
 <Button   
 variant="outline"   
 size="sm"  
 onClick={handleLogout}  
 disabled={logoutMutation.isPending}  
 className="bg-rose-50 text-rose-600 hover:bg-rose-100 border-rose-200 hover:text-rose-700"  
 >  
 {logoutMutation.isPending ? "Logging out..." : "Log out"}  
 </Button>  
 </>  
 )}  
 </div>  
 </div>  
 </div>  
 </div>  
 </header>  
 );  
}

### Filter Component

The filter component for filtering tasks by status:

// client/src/components/filters.tsx  
import { Button } from "@/components/ui/button";  
  
type FilterProps = {  
 currentFilter: "all" | "pending" | "completed";  
 onFilterChange: (filter: "all" | "pending" | "completed") => void;  
};  
  
export function Filters({ currentFilter, onFilterChange }: FilterProps) {  
 return (  
 <div className="flex space-x-2 items-center">  
 <span className="text-sm text-slate-500 mr-1">Filter:</span>  
 <div className="flex space-x-1 bg-slate-100 p-1 rounded-md">  
 <Button   
 variant={currentFilter === "all" ? "secondary" : "ghost"}   
 size="sm"   
 className="text-xs h-7"  
 onClick={() => onFilterChange("all")}  
 >  
 All  
 </Button>  
 <Button   
 variant={currentFilter === "pending" ? "secondary" : "ghost"}   
 size="sm"   
 className="text-xs h-7"  
 onClick={() => onFilterChange("pending")}  
 >  
 Pending  
 </Button>  
 <Button   
 variant={currentFilter === "completed" ? "secondary" : "ghost"}   
 size="sm"   
 className="text-xs h-7"  
 onClick={() => onFilterChange("completed")}  
 >  
 Completed  
 </Button>  
 </div>  
 </div>  
 );  
}

## Task Management Features

### Home Page with Task Management

The home page component includes functionality for displaying, filtering, creating, editing, and deleting tasks:

// client/src/pages/home-page.tsx  
import { useState } from "react";  
import { Navbar } from "@/components/navbar";  
import { Button } from "@/components/ui/button";  
import { Plus, Loader2 } from "lucide-react";  
import { useMutation, useQuery } from "@tanstack/react-query";  
import { Task } from "@shared/schema";  
import { apiRequest, queryClient } from "@/lib/queryClient";  
import { TaskCard } from "@/components/task-card";  
import { NewTaskDialog, EditTaskDialog, DeleteTaskDialog } from "@/components/task-modals";  
import { TaskFormValues } from "@/components/task-form";  
import { useToast } from "@/hooks/use-toast";  
import { Filters } from "@/components/filters";  
  
export default function HomePage() {  
 const { toast } = useToast();  
   
 // State for task modals  
 const [isNewTaskOpen, setIsNewTaskOpen] = useState(false);  
 const [isEditTaskOpen, setIsEditTaskOpen] = useState(false);  
 const [isDeleteTaskOpen, setIsDeleteTaskOpen] = useState(false);  
 const [editTask, setEditTask] = useState<Task | null>(null);  
 const [deleteTask, setDeleteTask] = useState<Task | null>(null);  
   
 // Filter state  
 const [filterStatus, setFilterStatus] = useState<"all" | "pending" | "completed">("all");  
  
 // Fetch tasks  
 const { data: tasks, isLoading, isError } = useQuery<Task[]>({  
 queryKey: ["/api/tasks"],  
 });  
  
 // Update task mutation  
 const updateTaskMutation = useMutation({  
 mutationFn: async ({ taskId, taskData }: { taskId: number; taskData: Partial<Task> }) => {  
 const res = await apiRequest("PUT", `/api/tasks/${taskId}`, taskData);  
 return res.json();  
 },  
 onSuccess: () => {  
 queryClient.invalidateQueries({ queryKey: ["/api/tasks"] });  
 setIsEditTaskOpen(false);  
 toast({  
 title: "Task updated",  
 description: "Your task has been updated successfully.",  
 variant: "success",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Failed to update task",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 // Delete task mutation  
 const deleteTaskMutation = useMutation({  
 mutationFn: async (taskId: number) => {  
 await apiRequest("DELETE", `/api/tasks/${taskId}`);  
 },  
 onSuccess: () => {  
 queryClient.invalidateQueries({ queryKey: ["/api/tasks"] });  
 setIsDeleteTaskOpen(false);  
 toast({  
 title: "Task deleted",  
 description: "Your task has been deleted successfully.",  
 variant: "info",  
 });  
 },  
 onError: (error: Error) => {  
 toast({  
 title: "Failed to delete task",  
 description: error.message,  
 variant: "destructive",  
 });  
 },  
 });  
  
 // Filter tasks based on status  
 const filteredTasks = tasks?.filter(task => {  
 if (filterStatus === "all") return true;  
 return task.status === filterStatus;  
 }) || [];  
  
 // Handle task edit  
 const handleEditTask = (task: Task) => {  
 setEditTask(task);  
 setIsEditTaskOpen(true);  
 };  
  
 // Handle task delete confirmation  
 const handleDeleteTask = (task: Task) => {  
 setDeleteTask(task);  
 setIsDeleteTaskOpen(true);  
 };  
  
 // Handle task status toggle  
 const handleToggleStatus = (task: Task) => {  
 const newStatus = task.status === "pending" ? "completed" : "pending";  
 updateTaskMutation.mutate({  
 taskId: task.id,  
 taskData: { ...task, status: newStatus },  
 });  
 };  
  
 return (  
 <div className="flex-1 flex flex-col min-h-screen">  
 <Navbar />  
   
 <main className="flex-1">  
 <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">  
 <div className="flex flex-col md:flex-row md:items-center md:justify-between mb-8 gap-4">  
 <div>  
 <h2 className="text-2xl font-bold text-slate-900">My Tasks</h2>  
 <p className="text-sm text-slate-500 mt-1">Manage all your tasks in one place</p>  
 </div>  
   
 <div className="flex flex-col sm:flex-row gap-3">  
 <Filters currentFilter={filterStatus} onFilterChange={setFilterStatus} />  
   
 <Button   
 onClick={() => setIsNewTaskOpen(true)}  
 className="flex items-center justify-center"  
 >  
 <Plus className="h-5 w-5 mr-1" /> New Task  
 </Button>  
 </div>  
 </div>  
  
 {isLoading ? (  
 <div className="flex justify-center items-center py-12">  
 <Loader2 className="h-8 w-8 animate-spin text-primary" />  
 </div>  
 ) : isError ? (  
 <div className="text-center py-12">  
 <p className="text-red-500">Failed to load tasks. Please try again.</p>  
 </div>  
 ) : filteredTasks.length > 0 ? (  
 <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">  
 {filteredTasks.map(task => (  
 <TaskCard   
 key={task.id}  
 task={task}  
 onEdit={() => handleEditTask(task)}  
 onDelete={() => handleDeleteTask(task)}  
 onStatusToggle={() => handleToggleStatus(task)}  
 />  
 ))}  
 </div>  
 ) : (  
 <div className="text-center py-12">  
 <div className="inline-flex items-center justify-center w-16 h-16 rounded-full bg-slate-100 mb-4">  
 <svg className="h-8 w-8 text-slate-400" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg">  
 <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M9 5H7a2 2 0 00-2 2v12a2 2 0 002 2h10a2 2 0 002-2V7a2 2 0 00-2-2h-2M9 5a2 2 0 002 2h2a2 2 0 002-2M9 5a2 2 0 012-2h2a2 2 0 012 2" />  
 </svg>  
 </div>  
 <h3 className="text-lg font-medium text-slate-900 mb-1">No tasks found</h3>  
 <p className="text-sm text-slate-500 mb-4">  
 {filterStatus !== "all"   
 ? `You don't have any ${filterStatus} tasks.`   
 : "You don't have any tasks yet. Create your first task to get started."}  
 </p>  
 <Button onClick={() => setIsNewTaskOpen(true)}>  
 <Plus className="h-5 w-5 mr-1" /> New Task  
 </Button>  
 </div>  
 )}  
 </div>  
 </main>  
  
 {/\* Task Modals \*/}  
 <NewTaskDialog   
 isOpen={isNewTaskOpen}   
 onOpenChange={setIsNewTaskOpen}   
 />  
   
 {editTask && (  
 <EditTaskDialog   
 isOpen={isEditTaskOpen}  
 onOpenChange={setIsEditTaskOpen}  
 task={editTask}  
 onUpdateTask={(taskData) => {  
 updateTaskMutation.mutate({  
 taskId: editTask.id,  
 taskData  
 });  
 }}  
 isPending={updateTaskMutation.isPending}  
 />  
 )}  
   
 {deleteTask && (  
 <DeleteTaskDialog   
 isOpen={isDeleteTaskOpen}  
 onOpenChange={setIsDeleteTaskOpen}  
 onConfirmDelete={() => {  
 deleteTaskMutation.mutate(deleteTask.id);  
 }}  
 isPending={deleteTaskMutation.isPending}  
 />  
 )}  
 </div>  
 );  
}

## Styling and Theming

### Custom Colors and Styling

The application uses custom colors defined in CSS variables for consistent styling:

/\* client/src/index.css \*/  
@tailwind base;  
@tailwind components;  
@tailwind utilities;  
  
:root {  
 --background: 0 0% 100%;  
 --foreground: 20 14.3% 4.1%;  
 --muted: 60 4.8% 95.9%;  
 --muted-foreground: 25 5.3% 44.7%;  
 --popover: 0 0% 100%;  
 --popover-foreground: 20 14.3% 4.1%;  
 --card: 0 0% 100%;  
 --card-foreground: 20 14.3% 4.1%;  
 --border: 20 5.9% 90%;  
 --input: 20 5.9% 90%;  
 --primary: 207 90% 54%;  
 --primary-foreground: 211 100% 99%;  
 --secondary: 60 4.8% 95.9%;  
 --secondary-foreground: 24 9.8% 10%;  
 --accent: 60 4.8% 95.9%;  
 --accent-foreground: 24 9.8% 10%;  
 --destructive: 0 84.2% 60.2%;  
 --destructive-foreground: 60 9.1% 97.8%;  
 --ring: 20 14.3% 4.1%;  
 /\* Custom colors for task statuses \*/  
 --success: 142 71% 45%;  
 --success-foreground: 0 0% 100%;  
 --pending: 222 89% 64%;  
 --pending-foreground: 0 0% 100%;  
 --radius: 0.5rem;  
}  
  
.dark {  
 --background: 240 10% 3.9%;  
 --foreground: 0 0% 98%;  
 /\* ... dark theme variables ... \*/  
}  
  
@layer base {  
 \* {  
 @apply border-border;  
 }  
  
 body {  
 @apply font-sans antialiased bg-background text-foreground;  
 }  
}

### Custom Badge Variants

Added custom badge variants for task status indicators:

// 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",  
 success:  
 "border-transparent bg-[hsl(var(--success))] text-[hsl(var(--success-foreground))] hover:bg-[hsl(var(--success))]/90",  
 pending:  
 "border-transparent bg-[hsl(var(--pending))] text-[hsl(var(--pending-foreground))] hover:bg-[hsl(var(--pending))]/90",  
 },  
 },  
 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 }

### Toast Notifications

The application uses a toast system with custom variants for different message types:

// client/src/components/ui/toast.tsx (variant section)  
const toastVariants = cva(  
 "group pointer-events-auto relative flex w-full items-center justify-between space-x-4 overflow-hidden rounded-md border p-6 pr-8 shadow-lg transition-all data-[swipe=cancel]:translate-x-0 data-[swipe=end]:translate-x-[var(--radix-toast-swipe-end-x)] data-[swipe=move]:translate-x-[var(--radix-toast-swipe-move-x)] data-[swipe=move]:transition-none data-[state=open]:animate-in data-[state=closed]:animate-out data-[swipe=end]:animate-out data-[state=closed]:fade-out-80 data-[state=closed]:slide-out-to-right-full data-[state=open]:slide-in-from-top-full data-[state=open]:sm:slide-in-from-bottom-full",  
 {  
 variants: {  
 variant: {  
 default: "border bg-background text-foreground",  
 destructive:  
 "destructive group border-destructive bg-destructive text-destructive-foreground",  
 success:   
 "border border-green-200 bg-green-50 text-green-800",  
 info:  
 "border border-blue-200 bg-blue-50 text-blue-800",  
 },  
 },  
 defaultVariants: {  
 variant: "default",  
 },  
 }  
)

## Deployment Guidelines

The application is designed to be easily deployed to any hosting environment that supports Node.js applications. Here are some key deployment considerations:

1. **Database Configuration**:
   * For production, replace the in-memory storage with a real PostgreSQL database connection.
   * Update the storage.ts file to use the PostgreSQL implementation instead of the MemStorage.
2. **Environment Variables**:
   * Set appropriate environment variables for the database connection.
   * Configure a proper session secret for production use.
   * Example environment variables include:
   * DATABASE\_URL=postgres://user:password@hostname:port/database  
     SESSION\_SECRET=your-secure-session-secret  
     NODE\_ENV=production
3. **Build Process**:
   * Run npm run build to create optimized production builds of both client and server.
   * The build process includes:
     + TypeScript compilation
     + Bundling of client-side assets
     + Minification and optimization
4. **Serving the Application**:
   * The server is designed to serve both the API and the static client files from the same origin.
   * In production, the Express server serves the pre-built client files from the dist directory.
5. **Security Considerations**:
   * Ensure the application is served over HTTPS.
   * Configure appropriate CORS settings if needed.
   * Implement rate limiting for authentication endpoints to prevent brute force attacks.

By following these guidelines, the Personal Task Manager application can be securely deployed to provide users with efficient task management capabilities.