Created 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";

declare global {
  namespace Express {
    interface User extends SelectUser {}
  }
}

const scryptAsync = promisify(scrypt);

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}`;
}

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) {
```

```
const sessionSettings: session.SessionOptions = {
  secret: process.env.REPL_ID!,
  resave: false,
  saveUninitialized: false,
  store: storage.sessionStore,
};

if (app.get("env") === "production") {
  app.set("trust proxy", 1);
}

app.use(session(sessionSettings));
app.use(passport.initialize());
app.use(passport.session());

passport.use(
  new LocalStrategy(async (username, password, done) => {
    const user = await storage.getUserByUsername(username);
    if (!user || !(await comparePasswords(password, user.password))) {
      return done(null, false);
    } else {
      return done(null, user);
    }
  }),
);

passport.serializeUser((user, done) => done(null, user.id));
passport.deserializeUser(async (id: number, done) => {
  const user = await storage.getUser(id);
  done(null, user);
});
```

```
app.post("/api/register", async (req, res, next) => {
  const existingUser = await storage.getUserByUsername(req.body.username);
  if (existingUser) {
    return res.status(400).send("Username already exists");
  }

  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);
});
```

```
}
```

Created client/src/hooks/use-auth.tsx

```
import { createContext, ReactNode, useContext } from "react";
import {
  useQuery,
  useMutation,
  UseMutationResult,
} from "@tanstack/react-query";
import { insertUserSchema, User as SelectUser, InsertUser } 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 = Pick<InsertUser, "username" | "password">;

export const AuthContext = createContext<AuthContextType | null>(null);
export function AuthProvider({ children }: { children: ReactNode }) {
  const { toast } = useToast();
  const {
    data: user,
    error,
    isLoading,
  } = useQuery<SelectUser | undefined, Error>({
    queryKey: ["/api/user"],
```

```
  queryFn: getQueryFn({ on401: "returnNull" }),
});

const loginMutation = useMutation({
  mutationFn: async (credentials: LoginData) => {
    const res = await apiRequest("POST", "/api/login", credentials);
    return await res.json();
  },
  onSuccess: (user: SelectUser) => {
    queryClient.setQueryData(["/api/user"], user);
  },
  onError: (error: Error) => {
    toast({
      title: "Login failed",
      description: error.message,
      variant: "destructive",
    });
  },
});

const registerMutation = useMutation({
  mutationFn: async (credentials: InsertUser) => {
    const res = await apiRequest("POST", "/api/register", credentials);
    return await res.json();
  },
  onSuccess: (user: SelectUser) => {
    queryClient.setQueryData(["/api/user"], user);
  },
  onError: (error: Error) => {
    toast({
      title: "Registration failed",
      description: error.message,
```

```
      variant: "destructive",
    });
  },
});

const logoutMutation = useMutation({
  mutationFn: async () => {
    await apiRequest("POST", "/api/logout");
  },
  onSuccess: () => {
    queryClient.setQueryData(["/api/user"], null);
  },
  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;
}
```

Created 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 <Component />
}
```

Created shared/schema.ts

```
import { pgTable, text, serial, integer, boolean, timestamp } from "drizzle-orm/pg-core";
import { createInsertSchema } from "drizzle-zod";
import { z } from "zod";

export const users = pgTable("users", {
 id: serial("id").primaryKey(),
 username: text("username").notNull().unique(),
 password: text("password").notNull(),
});
export const events = pgTable("events", {
 id: serial("id").primaryKey(),
 title: text("title").notNull(),
 description: text("description").notNull(),
 location: text("location").notNull(),
 date: timestamp("date").notNull(),
 imageUrl: text("image_url").notNull(),
 category: text("category").notNull(),
```

```
  creatorId: integer("creator_id").notNull(),
});

export const attendees = pgTable("attendees", {
  id: serial("id").primaryKey(),
  userId: integer("user_id").notNull(),
  eventId: integer("event_id").notNull(),
  status: text("status").notNull(), // attending, maybe, declined
});

export const insertUserSchema = createInsertSchema(users).pick({
  username: true,
  password: true,
});

export const insertEventSchema = createInsertSchema(events).omit({
  id: true,
});

export const insertAttendeeSchema = createInsertSchema(attendees).omit({
  id: true,
});

export type InsertUser = z.infer<typeof insertUserSchema>;
export type User = typeof users.$inferSelect;
export type Event = typeof events.$inferSelect;
export type Attendee = typeof attendees.$inferSelect;

export const eventCategories = [
  "Conference",
  "Workshop",
  "Networking",
```

```
  "Social",
  "Tech Talk",
  "Other",
] as const;
```

Created server/storage.ts

```
import { users, events, attendees, type User, type InsertUser, type Event, type InsertEvent, type Attendee, type InsertAttendee } from "@shared/schema";
import { db } from "./db";
import { eq } from "drizzle-orm";
import session from "express-session";
import connectPg from "connect-pg-simple";
import { pool } from "./db";

const PostgresSessionStore = connectPg(session);

export interface IStorage {
  getUser(id: number): Promise<User | undefined>;
  getUserByUsername(username: string): Promise<User | undefined>;
  createUser(user: InsertUser): Promise<User>;

  // Event methods
  createEvent(event: InsertEvent): Promise<Event>;
  getEvent(id: number): Promise<Event | undefined>;
  getEvents(): Promise<Event[]>;
  getEventsByUser(userId: number): Promise<Event[]>;
  updateEvent(id: number, event: Partial<InsertEvent>): Promise<Event>;
  deleteEvent(id: number): Promise<void>;

  // Attendee methods
  addAttendee(attendee: InsertAttendee): Promise<Attendee>;
  getEventAttendees(eventId: number): Promise<Attendee[]>;
```

```
  updateAttendeeStatus(id: number, status: string): Promise<Attendee>;

  sessionStore: session.SessionStore;
}

export class DatabaseStorage implements IStorage {
  sessionStore: session.SessionStore;

  constructor() {
    this.sessionStore = new PostgresSessionStore({
      pool,
      createTableIfMissing: true,
    });
  }

  async getUser(id: number): Promise<User | undefined> {
    const [user] = await db.select().from(users).where(eq(users.id, id));
    return user;
  }

  async getUserByUsername(username: string): Promise<User | undefined> {
    const [user] = await db.select().from(users).where(eq(users.username, username));
    return user;
  }

  async createUser(insertUser: InsertUser): Promise<User> {
    const [user] = await db.insert(users).values(insertUser).returning();
    return user;
  }

  async createEvent(insertEvent: InsertEvent): Promise<Event> {
    const [event] = await db.insert(events).values(insertEvent).returning();
```

```
  return event;
}

async getEvent(id: number): Promise<Event | undefined> {
  const [event] = await db.select().from(events).where(eq(events.id, id));
  return event;
}

async getEvents(): Promise<Event[]> {
  return await db.select().from(events);
}

async getEventsByUser(userId: number): Promise<Event[]> {
  return await db.select().from(events).where(eq(events.creatorId, userId));
}

async updateEvent(id: number, event: Partial<InsertEvent>): Promise<Event> {
  const [updated] = await db
    .update(events)
    .set(event)
    .where(eq(events.id, id))
    .returning();
  return updated;
}

async deleteEvent(id: number): Promise<void> {
  await db.delete(events).where(eq(events.id, id));
}

async addAttendee(insertAttendee: InsertAttendee): Promise<Attendee> {
  const [attendee] = await db.insert(attendees).values(insertAttendee).returning();
  return attendee;
```

```
  }

  async getEventAttendees(eventId: number): Promise<Attendee[]> {
    return await db.select().from(attendees).where(eq(attendees.eventId, eventId));
  }

  async updateAttendeeStatus(id: number, status: string): Promise<Attendee> {
    const [updated] = await db
      .update(attendees)
      .set({ status })
      .where(eq(attendees.id, id))
      .returning();
    return updated;
  }
}

export const storage = new DatabaseStorage();

import type { Express } from "express";
import { createServer, type Server } from "http";
import { setupAuth } from "./auth";
import { storage } from "./storage";
import { insertEventSchema, insertAttendeeSchema } from "@shared/schema";

export function registerRoutes(app: Express): Server {
  setupAuth(app);

  // Event routes
  app.get("/api/events", async (req, res) => {
    const events = await storage.getEvents();
    res.json(events);
  });
```

```
app.get("/api/events/:id", async (req, res) => {
 const event = await storage.getEvent(parseInt(req.params.id));
 if (!event) return res.status(404).send("Event not found");
 res.json(event);
});

app.post("/api/events", async (req, res) => {
 if (!req.isAuthenticated()) return res.sendStatus(401);

 const parsed = insertEventSchema.safeParse(req.body);
 if (!parsed.success) {
  return res.status(400).json(parsed.error);
 }

 const event = await storage.createEvent({
  ...parsed.data,
  creatorId: req.user.id,
 });
 res.status(201).json(event);
});

app.put("/api/events/:id", async (req, res) => {
 if (!req.isAuthenticated()) return res.sendStatus(401);

 const event = await storage.getEvent(parseInt(req.params.id));
 if (!event) return res.status(404).send("Event not found");
 if (event.creatorId !== req.user.id) return res.sendStatus(403);

 const updated = await storage.updateEvent(event.id, req.body);
 res.json(updated);
});
```

```
app.delete("/api/events/:id", async (req, res) => {
  if (!req.isAuthenticated()) return res.sendStatus(401);

  const event = await storage.getEvent(parseInt(req.params.id));
  if (!event) return res.status(404).send("Event not found");
  if (event.creatorId !== req.user.id) return res.sendStatus(403);

  await storage.deleteEvent(event.id);
  res.sendStatus(204);
});

// Attendee routes
app.post("/api/events/:id/attend", async (req, res) => {
  if (!req.isAuthenticated()) return res.sendStatus(401);

  const parsed = insertAttendeeSchema.safeParse({
    userId: req.user.id,
    eventId: parseInt(req.params.id),
    status: req.body.status,
  });

  if (!parsed.success) {
    return res.status(400).json(parsed.error);
  }

  const attendee = await storage.addAttendee(parsed.data);
  res.status(201).json(attendee);
});

app.get("/api/events/:id/attendees", async (req, res) => {
  const attendees = await storage.getEventAttendees(parseInt(req.params.id));
```

```
    res.json(attendees);
  });

  const httpServer = createServer(app);
  return httpServer;
}


import { Switch, Route } from "wouter";
import { queryClient } from "./lib/queryClient";
import { QueryClientProvider } from "@tanstack/react-query";
import { Toaster } from "@/components/ui/toaster";
import { AuthProvider } from "./hooks/use-auth";
import { ProtectedRoute } from "./lib/protected-route";
import NotFound from "@/pages/not-found";
import AuthPage from "@/pages/auth-page";
import HomePage from "@/pages/home-page";
import EventPage from "@/pages/event-page";
import CreateEventPage from "@/pages/create-event-page";

function Router() {
  return (
    <Switch>
      <ProtectedRoute path="/" component={HomePage} />
      <ProtectedRoute path="/events/new" component={CreateEventPage} />
      <ProtectedRoute path="/events/:id" component={EventPage} />
      <Route path="/auth" component={AuthPage} />
      <Route component={NotFound} />
    </Switch>
  );
}
```

```
function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <AuthProvider>
        <Router />
        <Toaster />
      </AuthProvider>
    </QueryClientProvider>
  );
}

export default App;
```

```
import { useAuth } from "@/hooks/use-auth";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from
"@/components/ui/form";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { insertUserSchema } from "@shared/schema";
import { Redirect } from "wouter";
import { Loader2 } from "lucide-react";

export default function AuthPage() {
  const { user, loginMutation, registerMutation } = useAuth();

  if (user) {
    return <Redirect to="/" />;
```

```
}

return (
 <div className="min-h-screen grid md:grid-cols-2">
  <div className="flex items-center justify-center p-8">
   <Card className="w-full max-w-md">
    <CardHeader>
     <CardTitle>Welcome to EventHub</CardTitle>
    </CardHeader>
    <CardContent>
     <Tabs defaultValue="login">
      <TabsList className="grid w-full grid-cols-2">
       <TabsTrigger value="login">Login</TabsTrigger>
       <TabsTrigger value="register">Register</TabsTrigger>
      </TabsList>

      <TabsContent value="login">
       <AuthForm
        mode="login"
        onSubmit={(data) => loginMutation.mutate(data)}
        isPending={loginMutation.isPending}
       />
      </TabsContent>

      <TabsContent value="register">
       <AuthForm
        mode="register"
        onSubmit={(data) => registerMutation.mutate(data)}
        isPending={registerMutation.isPending}
       />
      </TabsContent>
     </Tabs>
```

```
        </CardContent>
       </Card>
      </div>

      <div className="hidden md:flex bg-gradient-to-br from-primary/90 to-primary items-center justify-center p-8">
       <div className="max-w-md text-center text-white">
        <h1 className="text-4xl font-bold mb-4">Create Amazing Events</h1>
        <p className="text-lg opacity-90">
         Join our platform to discover and create memorable events. Connect with
         like-minded people and make things happen.
        </p>
       </div>
      </div>
    </div>
  );
}

function AuthForm({ mode, onSubmit, isPending }: {
  mode: "login" | "register";
  onSubmit: (data: any) => void;
  isPending: boolean;
}) {
  const form = useForm({
    resolver: zodResolver(insertUserSchema),
    defaultValues: {
      username: "",
      password: "",
    },
  });

  return (
    <Form {...form}>
```

```
<form onSubmit={form.handleSubmit(onSubmit)} className="space-y-4 mt-4">
 <FormField
  control={form.control}
  name="username"
  render={({ field }) => (
   <FormItem>
    <FormLabel>Username</FormLabel>
    <FormControl>
     <Input {...field} />
    </FormControl>
    <FormMessage />
   </FormItem>
  )}
 />

 <FormField
  control={form.control}
  name="password"
  render={({ field }) => (
   <FormItem>
    <FormLabel>Password</FormLabel>
    <FormControl>
     <Input type="password" {...field} />
    </FormControl>
    <FormMessage />
   </FormItem>
  )}
 />

 <Button type="submit" className="w-full" disabled={isPending}>
  {isPending && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
  {mode === "login" ? "Login" : "Register"}
```

```
      </Button>
    </form>
  </Form>
 );
}
```

Created client/src/pages/home-page.tsx

```
import { Link } from "wouter";
import { useQuery } from "@tanstack/react-query";
import { Event } from "@shared/schema";
import { Button } from "@/components/ui/button";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { Calendar, MapPin } from "lucide-react";
import { format } from "date-fns";

const eventThumbnails = [
  "https://images.unsplash.com/photo-1513151233558-d860c5398176",
  "https://images.unsplash.com/photo-1519751138087-5bf79df62d5b",
  "https://images.unsplash.com/photo-1492684223066-81342ee5ff30",
  "https://images.unsplash.com/photo-1511795409834-ef04bbd61622",
  "https://images.unsplash.com/photo-1464047736614-af63643285bf",
  "https://images.unsplash.com/photo-1513623935135-c896b59073c1",
];

export default function HomePage() {
  const { data: events = [], isLoading } = useQuery<Event[]>({
    queryKey: ["/api/events"],
  });
```

```
return (
 <div className="min-h-screen bg-background">
  <header className="bg-gradient-to-r from-primary to-primary/90 text-white py-12">
   <div className="container mx-auto px-4">
    <div className="flex justify-between items-center">
     <div>
      <h1 className="text-4xl font-bold mb-2">Discover Events</h1>
      <p className="text-lg opacity-90">
       Find and join amazing events in your area
      </p>
     </div>
     <Link href="/events/new">
      <Button variant="secondary" size="lg">
       Create Event
      </Button>
     </Link>
    </div>
   </div>
  </header>

  <main className="container mx-auto px-4 py-8">
   {isLoading ? (
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
     {[1, 2, 3].map((i) => (
      <Card key={i} className="animate-pulse">
       <div className="h-48 bg-muted rounded-t-lg" />
       <CardContent className="p-4">
        <div className="h-6 bg-muted rounded mb-2" />
        <div className="h-4 bg-muted rounded w-2/3" />
       </CardContent>
      </Card>
     ))}
```

```
      </div>
    ) : (
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
       {events.map((event, index) => (
        <Link key={event.id} href={`/events/${event.id}`}>
         <Card className="cursor-pointer hover:shadow-lg transition-shadow">
          <img
           src={eventThumbnails[index % eventThumbnails.length]}
           alt={event.title}
           className="h-48 w-full object-cover rounded-t-lg"
          />
          <CardContent className="p-4">
           <CardTitle className="mb-2">{event.title}</CardTitle>
           <div className="space-y-2 text-muted-foreground">
            <div className="flex items-center gap-2">
             <Calendar className="h-4 w-4" />
             {format(new Date(event.date), "PPP")}
            </div>
            <div className="flex items-center gap-2">
             <MapPin className="h-4 w-4" />
             {event.location}
            </div>
           </div>
          </CardContent>
         </Card>
        </Link>
       ))}
      </div>
    )}
   </main>
  </div>
);
```

}

Created client/src/pages/create-event-page.tsx

```
import { useAuth } from "@/hooks/use-auth";
import { useMutation } from "@tanstack/react-query";
import { useLocation } from "wouter";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { insertEventSchema, eventCategories } from "@shared/schema";
import { Loader2 } from "lucide-react";
import { apiRequest, queryClient } from "@/lib/queryClient";
import { useToast } from "@/hooks/use-toast";

const venueImages = [
  "https://images.unsplash.com/photo-1720002337847-b9b6bfa12760",
  "https://images.unsplash.com/photo-1648862459464-f465d8a4a484",
  "https://images.unsplash.com/photo-1521727284875-14f6b020d1d6",
  "https://images.unsplash.com/photo-1672820415487-c534a8ee22ff",
];

export default function CreateEventPage() {
  const [, navigate] = useLocation();
  const { toast } = useToast();

  const mutation = useMutation({
```

```
  mutationFn: async (data: any) => {
    const res = await apiRequest("POST", "/api/events", data);
    return await res.json();
  },
  onSuccess: () => {
    queryClient.invalidateQueries({ queryKey: ["/api/events"] });
    toast({
      title: "Event created",
      description: "Your event has been created successfully",
    });
    navigate("/");
  },
});

const form = useForm({
  resolver: zodResolver(insertEventSchema),
  defaultValues: {
    title: "",
    description: "",
    location: "",
    date: "",
    category: "",
    imageUrl: venueImages[Math.floor(Math.random() * venueImages.length)],
  },
});

return (
  <div className="min-h-screen bg-background py-8">
    <div className="container mx-auto px-4">
      <Card className="mx-auto max-w-2xl">
        <CardHeader>
          <CardTitle>Create New Event</CardTitle>
```

```
        </CardHeader>
        <CardContent>
         <Form {...form}>
          <form onSubmit={form.handleSubmit((data) => mutation.mutate(data))}
className="space-y-6">
           <FormField
            control={form.control}
            name="title"
            render={({ field }) => (
             <FormItem>
              <FormLabel>Event Title</FormLabel>
              <FormControl>
               <Input {...field} />
              </FormControl>
              <FormMessage />
             </FormItem>
            )}
           />

           <FormField
            control={form.control}
            name="description"
            render={({ field }) => (
             <FormItem>
              <FormLabel>Description</FormLabel>
              <FormControl>
               <Textarea {...field} />
              </FormControl>
              <FormMessage />
             </FormItem>
            )}
           />
```

```
<div className="grid grid-cols-1 md:grid-cols-2 gap-4">
 <FormField
  control={form.control}
  name="date"
  render={({ field }) => (
   <FormItem>
    <FormLabel>Date</FormLabel>
    <FormControl>
     <Input type="datetime-local" {...field} />
    </FormControl>
    <FormMessage />
   </FormItem>
  )}
 />

 <FormField
  control={form.control}
  name="location"
  render={({ field }) => (
   <FormItem>
    <FormLabel>Location</FormLabel>
    <FormControl>
     <Input {...field} />
    </FormControl>
    <FormMessage />
   </FormItem>
  )}
 />
</div>

<FormField
 control={form.control}
```

```
          name="category"
          render={({ field }) => (
           <FormItem>
            <FormLabel>Category</FormLabel>
            <Select onValueChange={field.onChange} defaultValue={field.value}>
             <FormControl>
              <SelectTrigger>
               <SelectValue placeholder="Select a category" />
              </SelectTrigger>
             </FormControl>
             <SelectContent>
              {eventCategories.map((category) => (
               <SelectItem key={category} value={category}>
                {category}
               </SelectItem>
              ))}
             </SelectContent>
            </Select>
            <FormMessage />
           </FormItem>
          )}
         />

         <Button type="submit" className="w-full" disabled={mutation.isPending}>
          {mutation.isPending && <Loader2 className="mr-2 h-4 w-4 animate-spin" />}
          Create Event
         </Button>
        </form>
       </Form>
      </CardContent>
     </Card>
    </div>
```

```
    </div>
  );
}
```

Created client/src/pages/event-page.tsx

```
import { useQuery, useMutation } from "@tanstack/react-query";
import { useRoute } from "wouter";
import { Button } from "@/components/ui/button";
import { Card, CardContent } from "@/components/ui/card";
import { Badge } from "@/components/ui/badge";
import { Calendar, MapPin, Users, Tag } from "lucide-react";
import { format } from "date-fns";
import { Event, Attendee } from "@shared/schema";
import { apiRequest, queryClient } from "@/lib/queryClient";
import { useAuth } from "@/hooks/use-auth";
import { RadioGroup, RadioGroupItem } from "@/components/ui/radio-group";
import { Label } from "@/components/ui/label";
import { useToast } from "@/hooks/use-toast";

const eventThumbnails = [
  "https://images.unsplash.com/photo-1513151233558-d860c5398176",
  "https://images.unsplash.com/photo-1519751138087-5bf79df62d5b",
  "https://images.unsplash.com/photo-1492684223066-81342ee5ff30",
  "https://images.unsplash.com/photo-1511795409834-ef04bbd61622",
  "https://images.unsplash.com/photo-1464047736614-af63643285bf",
  "https://images.unsplash.com/photo-1513623935135-c896b59073c1",
];

export default function EventPage() {
  const [, params] = useRoute("/events/:id");
  const eventId = parseInt(params?.id || "0");
```

```
const { user } = useAuth();
const { toast } = useToast();

const { data: event, isLoading: isEventLoading } = useQuery<Event>({
 queryKey: [`/api/events/${eventId}`],
});

const { data: attendees = [], isLoading: isAttendeesLoading } = useQuery<Attendee[]>({
 queryKey: [`/api/events/${eventId}/attendees`],
});

const attendMutation = useMutation({
 mutationFn: async (status: string) => {
  const res = await apiRequest("POST", `/api/events/${eventId}/attend`, { status });
  return res.json();
 },
 onSuccess: () => {
  queryClient.invalidateQueries({ queryKey: [`/api/events/${eventId}/attendees`] });
  toast({
   title: "Status updated",
   description: "Your attendance status has been updated",
  });
 },
});

if (isEventLoading) {
 return (
  <div className="min-h-screen bg-background">
   <div className="container mx-auto px-4 py-8">
    <Card className="animate-pulse">
     <div className="h-64 bg-muted rounded-t-lg" />
     <CardContent className="p-6 space-y-4">
```

```
          <div className="h-8 bg-muted rounded w-2/3" />
          <div className="h-4 bg-muted rounded w-1/2" />
          <div className="h-20 bg-muted rounded" />
        </CardContent>
      </Card>
    </div>
  </div>
 );
}

if (!event) {
 return (
  <div className="min-h-screen bg-background flex items-center justify-center">
   <Card className="w-full max-w-md mx-4">
    <CardContent className="p-6">
     <h1 className="text-2xl font-bold text-center mb-2">Event Not Found</h1>
     <p className="text-center text-muted-foreground">
      The event you're looking for doesn't exist or has been removed.
     </p>
    </CardContent>
   </Card>
  </div>
 );
}

const userAttendee = attendees.find((a) => a.userId === user?.id);
const thumbnailIndex = event.id % eventThumbnails.length;

return (
 <div className="min-h-screen bg-background">
  <div className="container mx-auto px-4 py-8">
   <Card>
```

```
<img
 src={eventThumbnails[thumbnailIndex]}
 alt={event.title}
 className="w-full h-64 object-cover rounded-t-lg"
/>
<CardContent className="p-6 space-y-6">
 <div className="space-y-4">
  <div className="flex items-center justify-between">
   <h1 className="text-3xl font-bold">{event.title}</h1>
   <Badge variant="secondary">{event.category}</Badge>
  </div>

  <div className="flex flex-wrap gap-4 text-muted-foreground">
   <div className="flex items-center gap-2">
    <Calendar className="h-5 w-5" />
    {format(new Date(event.date), "PPP")}
   </div>
   <div className="flex items-center gap-2">
    <MapPin className="h-5 w-5" />
    {event.location}
   </div>
   <div className="flex items-center gap-2">
    <Users className="h-5 w-5" />
    {attendees.length} attendee{attendees.length !== 1 ? "s" : ""}
   </div>
   <div className="flex items-center gap-2">
    <Tag className="h-5 w-5" />
    {event.category}
   </div>
  </div>

  <p className="text-lg">{event.description}</p>
```

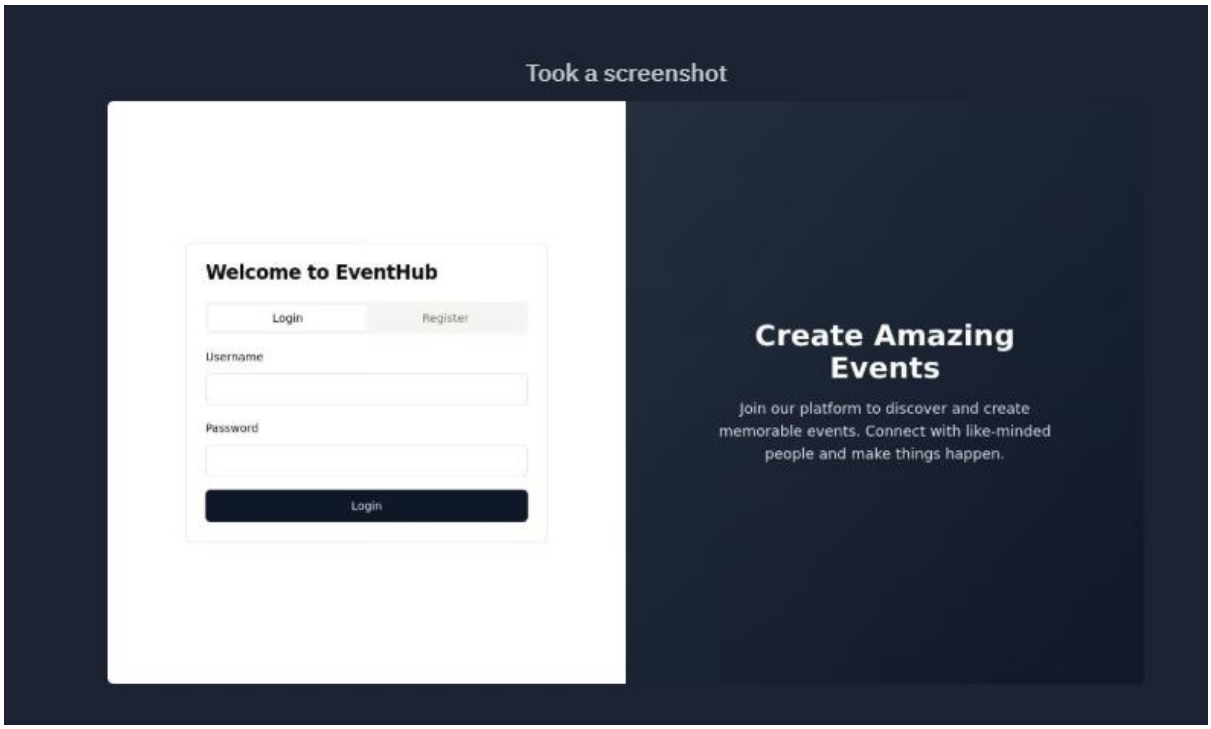
```
</div>

<div className="border-t pt-6">
 <h2 className="text-xl font-semibold mb-4">Your Attendance</h2>
 <RadioGroup
  defaultValue={userAttendee?.status || "attending"}
  onValueChange={(value) => attendMutation.mutate(value)}
  className="space-y-3"
 >
  <div className="flex items-center space-x-2">
   <RadioGroupItem value="attending" id="attending" />
   <Label htmlFor="attending">I'm attending</Label>
  </div>
  <div className="flex items-center space-x-2">
   <RadioGroupItem value="maybe" id="maybe" />
   <Label htmlFor="maybe">Maybe</Label>
  </div>
  <div className="flex items-center space-x-2">
   <RadioGroupItem value="declined" id="declined" />
   <Label htmlFor="declined">Can't make it</Label>
  </div>
 </RadioGroup>
</div>

<div className="border-t pt-6">
 <h2 className="text-xl font-semibold mb-4">Attendees</h2>
 {isAttendeesLoading ? (
  <div className="space-y-2">
   {[1, 2, 3].map((i) => (
    <div key={i} className="h-8 bg-muted rounded animate-pulse" />
   ))}
  </div>
```

```
          ) : (
            <div className="space-y-2">
             {attendees.map((attendee) => (
               <div
                 key={attendee.id}
                 className="flex items-center justify-between p-2 rounded hover:bg-muted"
               >
                 <span>User #{attendee.userId}</span>
                 <Badge
                  variant={
                    attendee.status === "attending"
                      ? "default"
                      : attendee.status === "maybe"
                      ? "secondary"
                      : "destructive"
                  }
                 >
                  {attendee.status}
                 </Badge>
               </div>
             ))}
            </div>
          )}
        </div>
      </CardContent>
    </Card>
  </div>
 </div>
 );
}
```

Configured Start application to run npm run dev


Took a screenshot
Welcome to EventHub
Login
Register
Username
Password
Login
Create Amazing Events
Join our platform to discover and create memorable events. Connect with like-minded people and make things happen.