Complete Authentication System Files

Project Structure

```
your-project/
├── client/              # Frontend React app
│   ├── src/
│   │   ├── pages/
│   │   │   └── AuthPage.tsx
│   │   ├── hooks/
│   │   │   └── useAuth.tsx
│   │   └── lib/
│   │       └── auth-context.tsx
│   ├── vite.config.ts
│   └── package.json
├── server/              # Backend Express app
│   ├── db/
│   │   ├── index.ts
│   │   └── schema.ts
│   ├── routes/
│   │   └── auth.ts
│   ├── middleware/
│   │   └── auth.ts
│   ├── index.ts
│   ├── package.json
│   ├── tsconfig.json
│   └── drizzle.config.ts
├── .env.example
└── .gitignore
```

Backend Files

📁 **server/package.json**

```
json
```

```
{
  "name": "whats-the-move-backend",
  "version": "1.0.0",
  "description": "Backend for What's the Move campus event app",
  "main": "dist/index.js",
  "scripts": {
    "dev": "tsx watch index.ts",
    "build": "tsc",
    "start": "node dist/index.js",
    "db:generate": "drizzle-kit generate:pg",
    "db:push": "drizzle-kit push:pg",
    "db:migrate": "tsx db/migrate.ts",
    "db:studio": "drizzle-kit studio"
  },
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "cookie-parser": "^1.4.6",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "dotenv": "^16.3.1",
    "drizzle-orm": "^0.29.3",
    "postgres": "^3.4.3",
    "zod": "^3.22.4"
  },
  "devDependencies": {
    "@types/express": "^4.17.21",
    "@types/cors": "^2.8.17",
    "@types/cookie-parser": "^1.4.6",
    "@types/bcryptjs": "^2.4.6",
    "@types/jsonwebtoken": "^9.0.5",
    "@types/node": "^20.10.5",
    "typescript": "^5.3.3",
    "tsx": "^4.7.0",
    "drizzle-kit": "^0.20.9"
  }
}
```

📁 **server/index.ts**

typescript

```
import express from 'express';
import cors from 'cors';
import cookieParser from 'cookie-parser';
import dotenv from 'dotenv';
import authRoutes from './routes/auth';

// Load environment variables
dotenv.config({ path: '../.env' });

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:5173',
  credentials: true, // Important for cookies
}));
app.use(express.json());
app.use(cookieParser());

// Routes
app.use('/api', authRoutes);

// Health check
app.get('/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({ error: 'Route not found' });
});

// Error handling middleware
app.use((err: any, req: express.Request, res: express.Response, next: express.NextFunction) => {
  console.error('Error:', err);
  res.status(err.status || 500).json({
    error: err.message || 'Something went wrong!',
    ...(process.env.NODE_ENV === 'development' && { stack: err.stack })
  });
});

app.listen(PORT, () => {
  console.log(`✨ Server running on http://localhost:${PORT}`);
  console.log(`🔐 CORS enabled for ${process.env.CLIENT_URL || 'http://localhost:5173'}`);
});
```

📁 **server/db/index.ts**

typescript

```typescript
import { drizzle } from 'drizzle-orm/postgres-js';
import postgres from 'postgres';
import * as schema from './schema';

if (!process.env.DATABASE_URL) {
  throw new Error('DATABASE_URL is not set');
}

const queryClient = postgres(process.env.DATABASE_URL);
export const db = drizzle(queryClient, { schema });
```

📁 **server/db/schema.ts**

typescript

```typescript
import { pgTable, uuid, text, timestamp, integer, boolean } from 'drizzle-orm/pg-core';
import { createInsertSchema, createSelectSchema } from 'drizzle-zod';

export const users = pgTable('users', {
  id: uuid('id').defaultRandom().primaryKey(),
  email: text('email').notNull().unique(),
  password: text('password').notNull(),
  firstName: text('first_name').notNull(),
  lastName: text('last_name').notNull(),
  university: text('university'),
  graduationYear: integer('graduation_year'),
  emailVerified: boolean('email_verified').default(false),
  createdAt: timestamp('created_at').defaultNow().notNull(),
  updatedAt: timestamp('updated_at').defaultNow().notNull(),
});

// Type exports
export type User = typeof users.$inferSelect;
export type NewUser = typeof users.$inferInsert;

// Zod schemas for validation
export const insertUserSchema = createInsertSchema(users);
export const selectUserSchema = createSelectSchema(users);
```

📁 **server/routes/auth.ts**

typescript

```
import express from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { z } from 'zod';
import { db } from '../db';
import { users } from '../db/schema';
import { eq } from 'drizzle-orm';
import { authMiddleware } from '../middleware/auth';

const router = express.Router();

// Environment variables
const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key-change-in-production';
const JWT_EXPIRES_IN = '7d';
const NODE_ENV = process.env.NODE_ENV || 'development';

// Validation schemas
const loginSchema = z.object({
  email: z.string().email().toLowerCase(),
  password: z.string().min(1),
});

const registerSchema = z.object({
  email: z.string().email().toLowerCase(),
  password: z.string()
    .min(6, "Password must be at least 6 characters")
    .regex(/[A-Z]/, "Password must contain at least one uppercase letter")
    .regex(/[0-9]/, "Password must contain at least one number"),
  firstName: z.string().min(1).trim(),
  lastName: z.string().min(1).trim(),
  university: z.string().trim().optional(),
  graduationYear: z.number().min(2020).max(2030).optional(),
});

// Helper function to create JWT token
const createToken = (userId: string) => {
  return jwt.sign({ userId }, JWT_SECRET, { expiresIn: JWT_EXPIRES_IN });
};

// Helper function to set auth cookie
const setAuthCookie = (res: express.Response, token: string) => {
  res.cookie('auth-token', token, {
    httpOnly: true,
    secure: NODE_ENV === 'production',
    sameSite: 'lax',
    maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
    path: '/',
  });
};

// POST /api/register
router.post('/register', async (req, res) => {
  try {
    // Validate input
    const validatedData = registerSchema.parse(req.body);

    // Check if user already exists
    const existingUser = await db
      .select()
      .from(users)
      .where(eq(users.email, validatedData.email))
      .limit(1);

    if (existingUser.length > 0) {
      return res.status(409).json({
        error: 'An account with this email already exists'
      });
    }

    // Hash password
    const hashedPassword = await bcrypt.hash(validatedData.password, 12);

    // Create user
    const [newUser] = await db.insert(users).values({
      email: validatedData.email,
      password: hashedPassword,
      firstName: validatedData.firstName,
      lastName: validatedData.lastName,
      university: validatedData.university,
      graduationYear: validatedData.graduationYear,
    }).returning();

    // Create session token
    const token = createToken(newUser.id);

    // Set cookie
    setAuthCookie(res, token);
    // Return user (without password)
    const { password: _, ...userWithoutPassword } = newUser;

    res.status(201).json({
      message: 'Registration successful',
      user: userWithoutPassword,
    });

  } catch (error) {
    console.error('Registration error:', error);

    if (error instanceof z.ZodError) {
      return res.status(400).json({
        error: 'Invalid input data',
        details: error.errors
      });
    }

    res.status(500).json({
      error: 'Failed to create account. Please try again.'
    });
  }
});

// POST /api/login
router.post('/login', async (req, res) => {
  try {
    // Validate input
    const validatedData = loginSchema.parse(req.body);

    // Find user
    const [user] = await db
      .select()
      .from(users)
      .where(eq(users.email, validatedData.email))
      .limit(1);

    if (!user) {
      return res.status(401).json({
        error: 'Invalid email or password'
      });
    }
    // Check password
    const isValidPassword = await bcrypt.compare(
      validatedData.password,
      user.password
    );

    if (!isValidPassword) {
      return res.status(401).json({
        error: 'Invalid email or password'
      });
    }

    // Update last login
    await db
      .update(users)
      .set({ updatedAt: new Date() })
      .where(eq(users.id, user.id));

    // Create session token
    const token = createToken(user.id);

    // Set cookie
    setAuthCookie(res, token);

    // Return user (without password)
    const { password: _, ...userWithoutPassword } = user;

    res.json({
      message: 'Login successful',
      user: userWithoutPassword,
    });

  } catch (error) {
    console.error('Login error:', error);

    if (error instanceof z.ZodError) {
      return res.status(400).json({
        error: 'Invalid input data',
        details: error.errors
      });
    }

    res.status(500).json({
      error: 'Failed to login. Please try again.'
    });
  }
});

// POST /api/logout
router.post('/logout', (req, res) => {
  res.clearCookie('auth-token', { path: '/' });
  res.json({ message: 'Logout successful' });
});

// GET /api/user - Get current user
router.get('/user', authMiddleware, async (req, res) => {
  try {
    const userId = (req as any).userId;

    const [user] = await db
      .select()
      .from(users)
      .where(eq(users.id, userId))
      .limit(1);

    if (!user) {
      return res.status(404).json({ error: 'User not found' });
    }

    const { password: _, ...userWithoutPassword } = user;

    res.json({ user: userWithoutPassword });

  } catch (error) {
    console.error('Get user error:', error);
    res.status(500).json({ error: 'Failed to fetch user' });
  }
});

// GET /api/verify-auth - Check if user is authenticated
router.get('/verify-auth', authMiddleware, (req, res) => {
  res.json({ authenticated: true, userId: (req as any).userId });
});

export default router;
```

📁 **server/middleware/auth.ts**

typescript

```typescript
import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';

const JWT_SECRET = process.env.JWT_SECRET || 'your-secret-key-change-in-production';

export const authMiddleware = (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = req.cookies['auth-token'];

    if (!token) {
      return res.status(401).json({ error: 'Not authenticated' });
    }

    const decoded = jwt.verify(token, JWT_SECRET) as { userId: string };
    (req as any).userId = decoded.userId;

    next();
  } catch (error) {
    console.error('Auth middleware error:', error);
    res.status(401).json({ error: 'Invalid or expired token' });
  }
};
```

📁 **server/tsconfig.json**

json

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "commonjs",
    "lib": ["ES2022"],
    "outDir": "./dist",
    "rootDir": "./",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "resolveJsonModule": true,
    "moduleResolution": "node",
    "allowSyntheticDefaultImports": true,
    "types": ["node"]
  },
  "include": ["**/*.ts"],
  "exclude": ["node_modules", "dist"]
}
```

📁 **server/drizzle.config.ts**

typescript

```typescript
import type { Config } from 'drizzle-kit';
import dotenv from 'dotenv';

dotenv.config({ path: '../.env' });

export default {
  schema: './db/schema.ts',
  out: './drizzle',
  driver: 'pg',
  dbCredentials: {
    connectionString: process.env.DATABASE_URL!,
  },
  verbose: true,
  strict: true,
} satisfies Config;
```

Frontend Files

📁 **client/vite.config.ts**

typescript

```typescript
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
  server: {
    port: 5173,
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true,
        secure: false,
      },
    },
  },
});
```

📁 **client/src/lib/auth-context.tsx**

typescript

```
import React, { createContext, useContext, useEffect, useState } from 'react';

interface User {
  id: string;
  email: string;
  firstName: string;
  lastName: string;
  university?: string;
  graduationYear?: number;
}

interface AuthContextType {
  user: User | null;
  isLoading: boolean;
  error: string | null;
  login: (email: string, password: string) => Promise<void>;
  register: (data: any) => Promise<void>;
  logout: () => Promise<void>;
  refetchUser: () => Promise<void>;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const fetchUser = async () => {
    try {
      const response = await fetch('/api/user', {
        credentials: 'include',
      });

      if (response.ok) {
        const data = await response.json();
        setUser(data.user);
      } else {
        setUser(null);
      }
    } catch (err) {
      console.error('Failed to fetch user:', err);
      setUser(null);
    } finally {
      setIsLoading(false);
    }
  };

  useEffect(() => {
    fetchUser();
  }, []);

  const login = async (email: string, password: string) => {
    const response = await fetch('/api/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      credentials: 'include',
      body: JSON.stringify({ email, password }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Login failed');
    }

    setUser(data.user);
  };

  const register = async (registerData: any) => {
    const response = await fetch('/api/register', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      credentials: 'include',
      body: JSON.stringify(registerData),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.error || 'Registration failed');
    }

    setUser(data.user);
  };

  const logout = async () => {
    try {
      await fetch('/api/logout', {
        method: 'POST',
        credentials: 'include',
      });
      setUser(null);
    } catch (err) {
      console.error('Logout failed:', err);
    }
  };

  return (
    <AuthContext.Provider
      value={{
        user,
        isLoading,
        error,
        login,
        register,
        logout,
        refetchUser: fetchUser,
      }}
    >
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};
```

📁 **client/src/hooks/useAuth.tsx**

typescript

```
// This file re-exports from the auth context for consistency
export { useAuth } from '@/lib/auth-context';
```

📁 `client/src/main.tsx` **(Update your main app file)**

```typescript
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { AuthProvider } from '@/lib/auth-context';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <AuthProvider>
      <App />
    </AuthProvider>
  </React.StrictMode>
);
```

Environment & Config Files

📁 `.env.example`

```bash
# Server Configuration
NODE_ENV=development
PORT=5000

# Client URL (for CORS)
CLIENT_URL=http://localhost:5173

# Database (PostgreSQL)
DATABASE_URL=postgresql://username:password@localhost:5432/whats_the_move

# JWT Configuration
JWT_SECRET=generate-a-secure-random-string-here-at-least-32-chars

# Optional: Email Service (for future email verification)
# SMTP_HOST=smtp.gmail.com
# SMTP_PORT=587
# SMTP_USER=your-email@gmail.com
# SMTP_PASS=your-app-specific-password
```

📁 **.gitignore**

```
# Dependencies
node_modules/
.pnp
.pnp.js

# Production
dist/
build/

# Environment variables
.env
.env.local
.env.*.local

# Database
*.sqlite
*.sqlite3
*.db
drizzle/

# Logs
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

# IDE
.vscode/
.idea/
*.swp
*.swo
*~
.DS_Store

# Testing
coverage/
*.lcov
.nyc_output

# TypeScript
```

```
*.tsbuildinfo

# Package manager
package-lock.json
yarn.lock
pnpm-lock.yaml
```

Setup Instructions

1. Install Dependencies

```
bash

# Backend
cd server
npm install

# Frontend
cd ../client
npm install
```

2. Set Up PostgreSQL Database

```
bash

# Install PostgreSQL if not already installed
# Create a new database
psql -U postgres
CREATE DATABASE whats_the_move;
```

3. Configure Environment

```
bash

# Copy the example env file
cp .env.example .env

# Edit .env with your database credentials
# Generate a secure JWT secret:
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

4. Initialize Database

```bash
cd server
npm run db:push  # Creates tables from schema
```

5. Run Development Servers

```bash
# Terminal 1 - Backend
cd server
npm run dev

# Terminal 2 - Frontend
cd client
npm run dev
```

6. Test the Application

- Open http://localhost:5173
- Try registering a new account
- Login with your credentials
- Check that cookies are being set properly

Security Checklist

- ✅ Passwords hashed with bcrypt (12 rounds)
- ✅ JWT tokens in httpOnly cookies
- ✅ CORS configured properly
- ✅ Input validation with Zod
- ✅ SQL injection protection (Drizzle ORM)
- ✅ Environment variables for secrets
- ✅ Secure cookies in production
- ⬜ Rate limiting (add express-rate-limit)
- ⬜ Email verification (implement later)
- ⬜ Password reset functionality

- ⬜ Two-factor authentication

Common Issues & Solutions

Issue: Cannot connect to database

Solution: Check DATABASE_URL in .env and ensure PostgreSQL is running

Issue: CORS errors

Solution: Ensure CLIENT_URL in .env matches your frontend URL

Issue: Cookies not being set

Solution: Check that credentials: 'include' is in fetch requests and proxy is configured in vite.config.ts

Issue: JWT errors

Solution: Generate a new JWT_SECRET and restart the server

Production Deployment

For Backend (e.g., Railway, Render, Heroku):

1. Set NODE_ENV=production
2. Set all environment variables
3. Use SSL/TLS for database connection
4. Enable HTTPS

For Frontend (e.g., Vercel, Netlify):

1. Build with `npm run build`
2. Configure API endpoint to production URL
3. Set up proper redirects for SPA routing