Social Authentication Integration (Google, GitHub, Facebook, LinkedIn, Twitter, Apple)

This document contains complete, ready-to-add code snippets tailored to your project structure (Node.js + Express + TypeScript + PostgreSQL using SQL queries). Place each snippet into the indicated file path in your repository.

---

0) Required packages

Run:

```
npm install passport passport-google-oauth20 passport-github2 passport-facebook
passport-linkedin-oauth2 passport-twitter passport-apple express-session
# if you use typescript dev packages
npm install -D @types/express-session @types/passport @types/passport-twitter
```

---

1) `.env.example`

```
# server
PORT=5000
SERVER_URL=http://localhost:5000
FRONTEND_URL=http://localhost:5173
DASHBOARD_URL=http://localhost:3000

# JWT / Cookie
JWT_SECRET_KEY=your_jwt_secret
JWT_EXPIRES_IN=1d
COOKIE_EXPIRES_IN=7

# Postgres
DB_USER=postgres
DB_HOST=localhost
DB_NAME=your_db
DB_PASSWORD=your_password
DB_PORT=5432

# Google
GOOGLE_CLIENT_ID=your_google_client_id
GOOGLE_CLIENT_SECRET=your_google_client_secret
```

```
# GitHub
GITHUB_CLIENT_ID=your_github_client_id
GITHUB_CLIENT_SECRET=your_github_client_secret

# Facebook
FACEBOOK_CLIENT_ID=your_facebook_app_id
FACEBOOK_CLIENT_SECRET=your_facebook_app_secret

# LinkedIn
LINKEDIN_CLIENT_ID=your_linkedin_client_id
LINKEDIN_CLIENT_SECRET=your_linkedin_client_secret

# Twitter (X)
TWITTER_CONSUMER_KEY=your_twitter_api_key
TWITTER_CONSUMER_SECRET=your_twitter_api_secret

# Apple
APPLE_CLIENT_ID=your_apple_service_id
APPLE_TEAM_ID=YOUR_TEAM_ID
APPLE_KEY_ID=YOUR_KEY_ID
# paste the .p8 content with \n escaped or load from file
APPLE_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----"
```

---

2) `src/config/passport.ts`

```
import passport from "passport";
import { Strategy as GoogleStrategy } from "passport-google-oauth20";
import { Strategy as GitHubStrategy } from "passport-github2";
import { Strategy as FacebookStrategy } from "passport-facebook";
import { Strategy as LinkedInStrategy } from "passport-linkedin-oauth2";
import { Strategy as TwitterStrategy } from "passport-twitter";
// passport-apple types may not ship with TS types; import as any
import AppleStrategy from "passport-apple" as any;
import database from "./db";

// Helper: create or find user by email
async function findOrCreateUserByEmail(profile: any, providerName: string) {
  const email = profile.emails?.[0]?.value ?? profile.email;
  const name = profile.displayName ?? `${profile.name?.givenName ?? ""} $
{profile.name?.familyName ?? ""}`.trim();

  if (!email) {
    // Providers like Twitter might not return email by default; caller should
```

```
handle this case.
    throw new Error("Email not returned from provider");
  }

  // find user
  const result = await database.query(`SELECT * FROM users WHERE email = $1`,
[email]);
  if (result.rows.length > 0) return result.rows[0];

  // not found -> create
  const insert = await database.query(
    `INSERT INTO users (name, email, password, avatar) VALUES ($1, $2, $3, $4)
RETURNING *`,
    [name || email, email, 'SOCIAL_LOGIN', JSON.stringify({ provider:
providerName, photo: profile.photos?.[0]?.value ?? null })]
  );
  return insert.rows[0];
}

// GOOGLE
passport.use(
  new GoogleStrategy(
    {
      clientID: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/google/callback`,
    },
    async (accessToken, refreshToken, profile, done) => {
      try {
        const user = await findOrCreateUserByEmail(profile, 'google');
        done(null, user);
      } catch (err) {
        done(err as Error, null);
      }
    }
  )
);

// GITHUB
passport.use(
  new GitHubStrategy(
    {
      clientID: process.env.GITHUB_CLIENT_ID!,
      clientSecret: process.env.GITHUB_CLIENT_SECRET!,
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/github/callback`,
    },
    async (accessToken, refreshToken, profile, done) => {
      try {
```

```
        const user = await findOrCreateUserByEmail(profile, 'github');
        done(null, user);
      } catch (err) {
        done(err as Error, null);
      }
    }
  )
);

// FACEBOOK
passport.use(
  new FacebookStrategy(
    {
      clientID: process.env.FACEBOOK_CLIENT_ID!,
      clientSecret: process.env.FACEBOOK_CLIENT_SECRET!,
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/facebook/callback`,
      profileFields: ['id', 'displayName', 'email', 'photos'],
    },
    async (accessToken, refreshToken, profile, done) => {
      try {
        const user = await findOrCreateUserByEmail(profile, 'facebook');
        done(null, user);
      } catch (err) {
        done(err as Error, null);
      }
    }
  )
);

// LINKEDIN
passport.use(
  new LinkedInStrategy(
    {
      clientID: process.env.LINKEDIN_CLIENT_ID!,
      clientSecret: process.env.LINKEDIN_CLIENT_SECRET!,
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/linkedin/callback`,
      scope: ['r_liteprofile', 'r_emailaddress'],
    },
    async (accessToken, refreshToken, profile, done) => {
      try {
        const user = await findOrCreateUserByEmail(profile, 'linkedin');
        done(null, user);
      } catch (err) {
        done(err as Error, null);
      }
    }
  )
);
```

```
// TWITTER (OAuth 1.0a)
passport.use(
  new TwitterStrategy(
    {
      consumerKey: process.env.TWITTER_CONSUMER_KEY!,
      consumerSecret: process.env.TWITTER_CONSUMER_SECRET!,
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/twitter/callback`,
      includeEmail: true,
    },
    async (token, tokenSecret, profile, done) => {
      try {
        const user = await findOrCreateUserByEmail(profile, 'twitter');
        done(null, user);
      } catch (err) {
        done(err as Error, null);
      }
    }
  )
);

// APPLE
// passport-apple expects a different verify signature; provider may not send
email on subsequent logins
try {
  // eslint-disable-next-line @typescript-eslint/no-var-requires
  const Apple = require('passport-apple');
  passport.use(
    new Apple({
      clientID: process.env.APPLE_CLIENT_ID,
      teamID: process.env.APPLE_TEAM_ID,
      keyID: process.env.APPLE_KEY_ID,
      privateKey: process.env.APPLE_PRIVATE_KEY?.replace(/\\n/g, '\n'),
      callbackURL: `${process.env.SERVER_URL}/api/v1/social/apple/callback`,
      scope: ['name', 'email'],
    },
    async (accessToken: any, refreshToken: any, idToken: any, profile: any,
done: any) => {
      try {
        // Apple returns emails only on first sign-in. idToken may contain
email.
        const userProfile = profile || (idToken && idToken.payload) || {};
        const user = await findOrCreateUserByEmail(userProfile, 'apple');
        done(null, user);
      } catch (err) {
        done(err, null);
      }
    }))
```

```
} catch (e) {
  // if package not installed or not available, log but allow server to run
  console.warn('passport-apple not configured or installed', e);
}

// session serialization — we store user id
passport.serializeUser((user: any, done) => {
  done(null, user.id);
});

passport.deserializeUser(async (id: string, done) => {
  try {
    const result = await database.query(`SELECT * FROM users WHERE id = $1`,
[id]);
    done(null, result.rows[0] || null);
  } catch (err) {
    done(err as Error, null);
  }
});

export default passport;
```

---

3) `src/routes/socialAuthRouter.ts`

```
import express from 'express';
import passport from 'passport';
import { socialLoginSuccess } from '../controllers/socialController';

const router = express.Router();

// Google
router.get('/google', passport.authenticate('google', { scope: ['profile',
'email'] }));
router.get('/google/callback', passport.authenticate('google', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

// GitHub
router.get('/github', passport.authenticate('github', { scope:
['user:email'] }));
router.get('/github/callback', passport.authenticate('github', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

// Facebook
router.get('/facebook', passport.authenticate('facebook', { scope:
```

```
['email'] }));
router.get('/facebook/callback', passport.authenticate('facebook', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

// LinkedIn
router.get('/linkedin', passport.authenticate('linkedin'));
router.get('/linkedin/callback', passport.authenticate('linkedin', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

// Twitter
router.get('/twitter', passport.authenticate('twitter'));
router.get('/twitter/callback', passport.authenticate('twitter', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

// Apple
router.get('/apple', passport.authenticate('apple'));
router.post('/apple/callback', passport.authenticate('apple', {
failureRedirect: `${process.env.FRONTEND_URL}/login` }), socialLoginSuccess);

export default router;
```

---

4) `src/controllers/socialController.ts`

```
import { Request, Response } from 'express';
import { generateToken } from '../utils/jwtToken';

export const socialLoginSuccess = (req: Request, res: Response) => {
  // req.user is set by passport
  const user = req.user as any;
  if (!user) return res.redirect(`${process.env.FRONTEND_URL}/login`);

  // Reuse existing generateToken to set cookie and return JSON
  // Option 1: redirect to frontend with token query param (if you prefer)
  // const token = create token manually and redirect to FRONTEND_URL?token=...

  // We will call generateToken so cookie is set and JSON is returned.
  // If you prefer redirect, change generateToken to return token string and
redirect accordingly.
  return generateToken(user, 200, 'Logged in via social provider', res);
};
```

---

5) Update `src/app.ts` to initialize passport and session

Find your current app.ts and add/update these lines (near top after dotenv and before routers):

```
import session from 'express-session';
import passport from './config/passport';

// ... existing code

app.use(
  session({
    secret: process.env.SESSION_SECRET ?? 'session_secret_change_me',
    resave: false,
    saveUninitialized: false,
    cookie: { secure: false },
  })
);

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

// mount social router
import socialAuthRouter from '../src/routes/socialAuthRouter';
app.use('/api/v1/social', socialAuthRouter);
```

Note: In production set `cookie.secure = true` and use HTTPS.

---

6) Minor adjustments: `src/config/db.ts` (already exists)

No change needed — passport uses your existing `database` client.

---

7) User table considerations

Your `users` table currently has `password TEXT NOT NULL`. For social users you may want to allow null or a sentinel value. Run one migration or query manually:

```
ALTER TABLE users ALTER COLUMN password DROP NOT NULL;
UPDATE users SET password = 'LOCAL' WHERE password IS NULL;
```

Or keep `SOCIAL_LOGIN` as inserted above.

---

8) Front-end quick usage (React)

On your login page, open social endpoint in same window to let passport set cookies:

```
// Sign in button
const handleGoogle = () => { window.location.href = `$
{process.env.REACT_APP_API_BASE}/api/v1/social/google`; }
```

After provider redirect -> backend will set cookie and respond JSON. If you prefer redirect to frontend with token query param, modify `socialController.socialLoginSuccess` to `const token = jwt.sign(...); res.redirect(` ${process.env.FRONTEND_URL}/dashboard?token=${token} `)`

---

9) Notes & troubleshooting

- Ensure redirect URIs configured at provider dashboards match the callback URLs defined (exact match). Example: `http://localhost:5000/api/v1/social/google/callback`.
- Twitter may not return email by default; you must request email access in Twitter App settings and include `includeEmail: true` in the strategy options (done above).
- Apple requires a paid Apple developer account and correct private key formatting. The `.p8` key must be embedded or read from a file and `\\n` sequences replaced with real newlines when loading.
- If a provider does not return email, you can show an intermediate page asking user to provide email and then link account.

---

10) Next step (optional)

If you want, I can produce the exact TypeScript files ready to paste (with imports/exports matching your project's tsconfig paths), or I can create an updated `authController` that merges social login with existing local login flow (for example: link existing accounts by email). Tell me which you prefer and I will generate the files accordingly.

End of document.