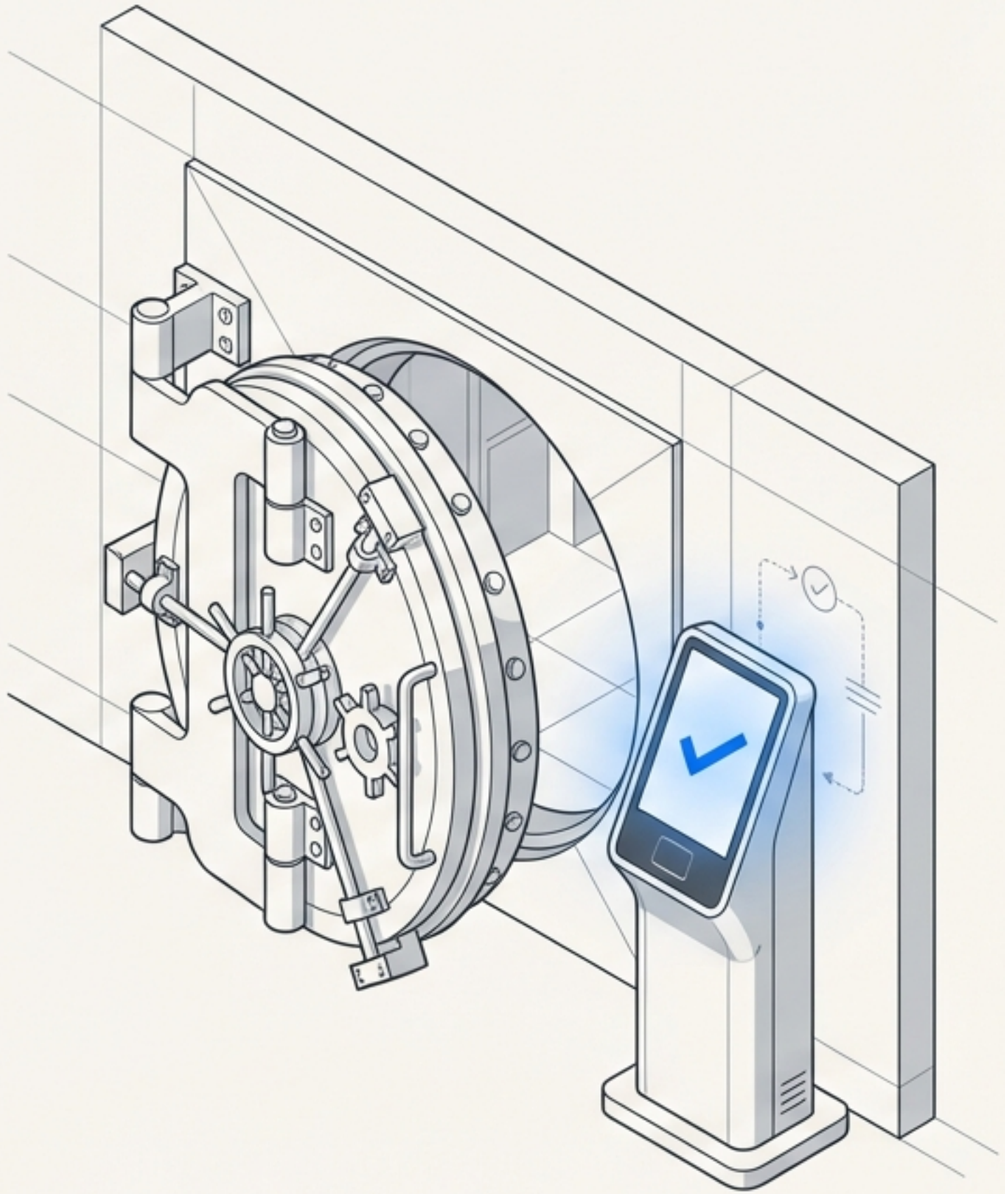
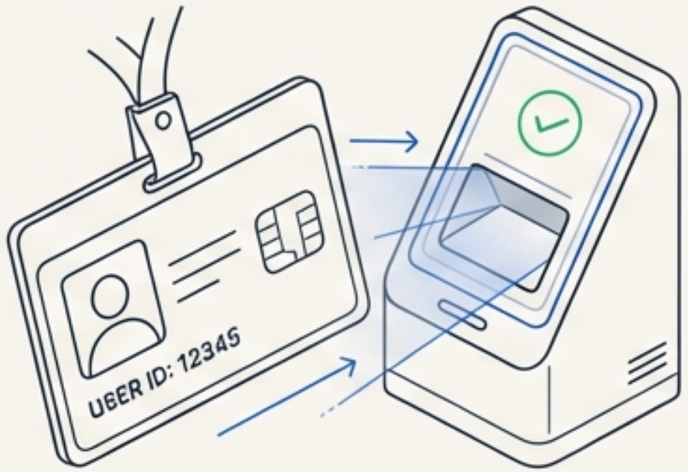
ID Will Be Checked at the Door

A definitive guide to Authentication and Passport.js in a Node/Express app.

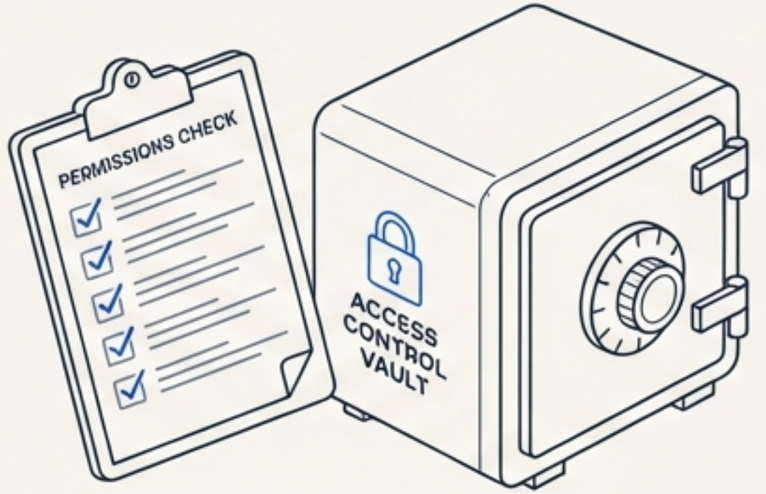




Authentication (AuthN)

Who are you?

- Validating identity (Login, Registration, Sessions).

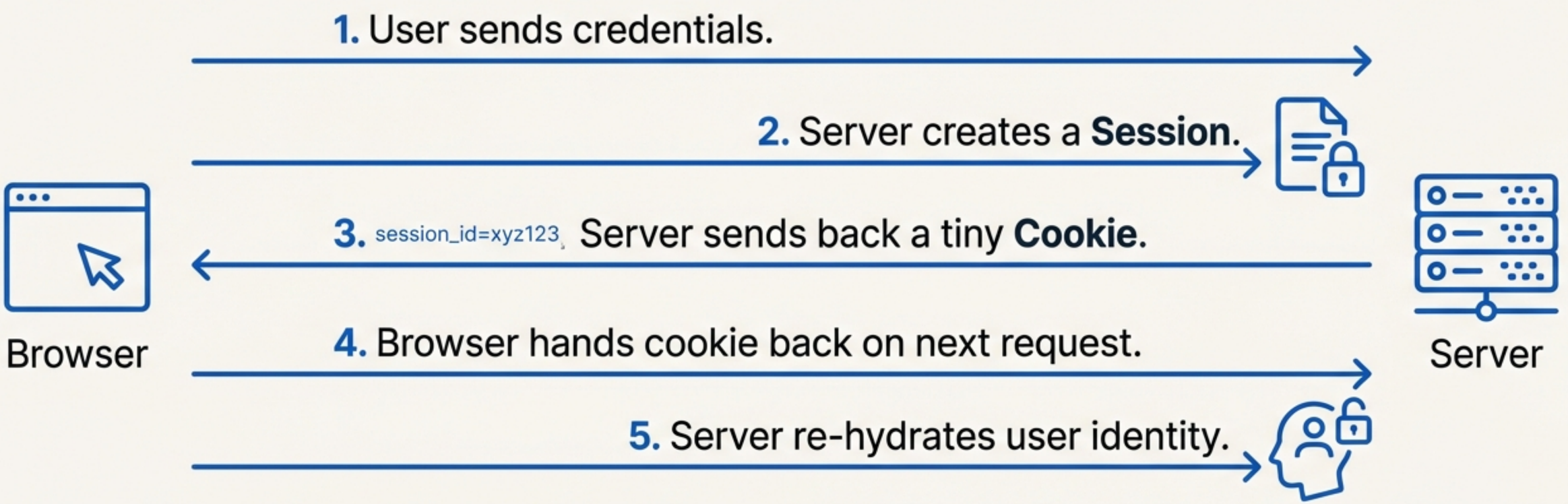


Authorization (AuthZ)

What can you do?

- Verifying permissions (Public vs. Admin routes).

Checking if a user exists is useless if you don't then verify they have the rights to drop a database table.

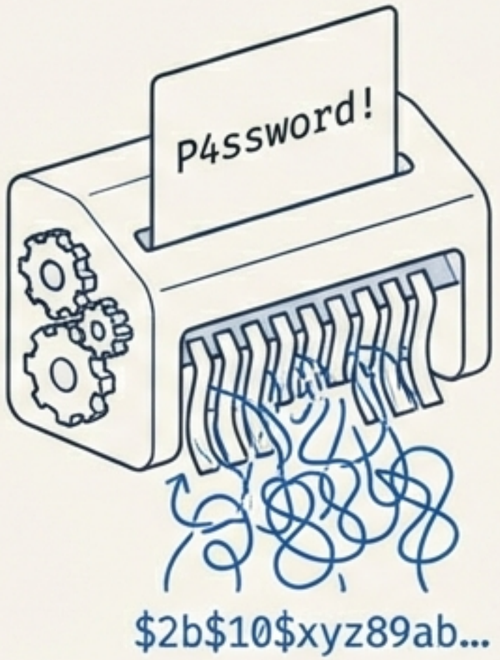


We use Session-based Auth because it perfectly suits tightly coupled, server-rendered Express apps.

Pro-Note: A cookie is just the slice of **data in the browser.** A **session** is the **actual data object** on the **server.**

Minimum Viable Security: Scramble the Keys

Storing plain text passwords is a catastrophic failure. We don’t encrypt (two-way street); we **hash** (one-way trip).




```
const hash = await
bcrypt.hash(password, 10);

const isMatch = await
bcrypt.compare(pwd, user.hash);
```

Hanging Promises! Forgetting **await** before `bcrypt.hash()` saves an unresolved promise to the database. That user is never logging in.

```
const session = require('express-session');

app.use(session({
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false
}));



}));
```

Secret: Cryptographically signs the cookie so users can't fake it. Must be in .env.

MemoryStore: The default storage. It is a development-only toy that leaks memory and destroys sessions on restart. Production requires connect-mongo.

The Passport.js Mental Model

The Bouncer (Passport) - Connects credential-checking to Express.

The Clipboard (Strategy) - The specific auth method (Local email/password).

The Wristband (Session) - The mechanism to serialize/deserialize the user.

What Passport DOES NOT do

It won't create your Mongoose User schema, it won't hash your passwords with bcrypt, and it won't decide which routes are admin-only. That is your job.

Data Prep & The Identity Transformer

Mongoose Schema

```
const UserSchema = new Schema({
  email: { type: String, unique: true },
  passwordHash: { type: String, required: true }
});
```

Passport Transform

```
passport.serializeUser((user, done) => {
  done(null, user._id.toString());
});

passport.deserializeUser(async (id, done) => {
  const user = await _userOps.getUserById(id);
  done(null, user); // Attaches to req.user
});
```

Serialization Mismatches! If you pack `user._id` into the cookie, you absolutely must query by `_id` when unpacking it.

The Strategy (Verify Callback)

```
passport.use('local', new LocalStrategy(
  { usernameField: 'email' },
  async (email, password, done) => {
    const user = await _userOps.getUserByEmail(email);
    if (!user) return done(null, false, { message: 'Invalid credentials.' });

    const isMatch = await bcrypt.compare(password, user.passwordHash);
    if (!isMatch) return done(null, false, { message: 'Invalid credentials.' });

    return done(null, user); // Success! Here is the verified user.
  }
));
```

Ambiguity is security. Never return “Email not found.” Always use “Invalid credentials” to prevent malicious enumeration of your user base.

Core Flow 1: Registration


```
try {
  await _userOps.createUser(email, password);
  res.redirect('/admin/login');
} catch (error) {
  if (error.message === 'Email_In_Use') {
    return res.status(409).render('admin/register', { error: 'Email already in use.' });
  }
}
```

Catch database unique constraint errors early to return polite feedback rather than crashing the app.

Core Flow 2: Login


```
router.post(
  '/login',
  passport.authenticate('local', {
    successRedirect: '/admin',
    failureRedirect: '/admin/login',
  })
);
```

Handing the Bouncer your ID. Once verified, Passport automatically injects the authenticated session and attaches req.user.

Core Flow 3: Logout

1. Invalidate Passport context
2. Destroy Session Memory
3. Clear Cookie & Redirect

```
adminRouter.get('/logout', (req, res, next) => {
  req.logout((err) => { // 1. Strip Passport state
    if (err) return next(err);
    req.session.destroy((err) => { // 2. Wipe server memory
      res.clearCookie('connect.sid'); // 3. Annihilate cookie
      res.redirect('/admin/login');
    });
  });
});
```

Authentication is temporary. Users must be able to securely revoke their sessions.

The Velvet Rope (Authorization Middleware)

```
function requireAuth(req, res, next) {
  if (req.isAuthenticated?.() &&
    req.user) return next();
  return res.redirect('/admin/login');
}
```

```
router.get('/login', renderLoginView); // Public Admin Route

<--- THE VELVET ROPE --->
router.use(requireAuth);

router.get('/', renderDashboardView); // Protected Route
```

The **Redirect Loop**! If you define your `/login` route below the `requireAuth` wall, the server will infinitely bounce the user.

Front-End UX (Views & State)

Navigation State

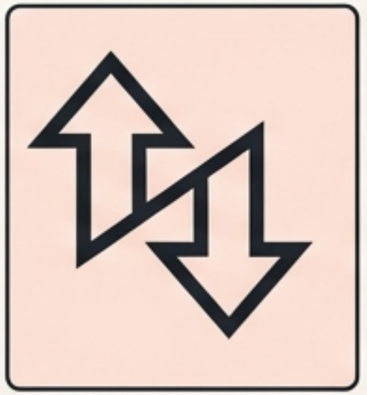
```
<% if (user) { %>
  <a href='/admin/logout'>Logout</a>
<% } else { %>
  <a href='/admin/login'>Login
<% } %>
```

Form Errors

```
<% if (error) { %>
  <div class='alert alert-danger'><%= error %></div>
<% } %>
```

`req.user` must be passed down to your views to conditionally render navigation items and private dashboard links.

The Bug Farm (Troubleshooting)



The Middleware Inversion

express-session MUST execute before passport.session() in server.js or the session has nowhere to live.

Forgetful Promises

Missing await on bcrypt.compare() universally accepts every invalid password.

Local Dev Failures

Cookies not sticking? Setting secure: true on cookies locally without an active SSL/TLS certificate will fail silently.

Why is req.user undefined?

Usually traces back to a serialization mismatch between _id and your database lookup.

The Integration Checklist

- [ ] Create Mongoose `User` schema (email, `passwordHash`).
- [ ] Configure `express-session` (with `.env` secrets).
- [ ] Initialize **passport** and `passport-local` LocalStrategy.
- [ ] Implement `serializeUser` and `deserializeUser`.
- [ ] Build `Register` & `Login` POST controllers with bcrypt.
- [ ] Mount the `requireAuth` Velvet Rope middleware.
- [ ] Construct the Logout controller (destroy session & cookie).

We aren't building a bank, but we aren't leaving the doors open either.
p.s., keep learning!