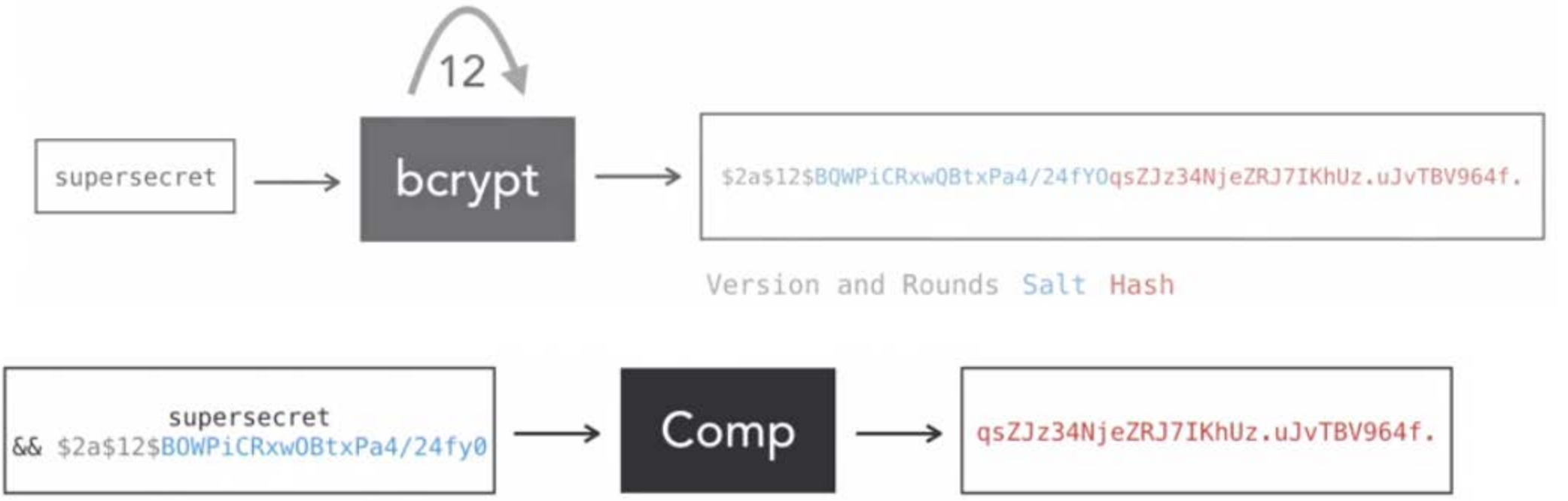
Popular Express Middleware

- **body-parser:** parse HTTP request body
- **compression:** compress HTTP responses
- **cookie-parser:** parse cookie header and populate req.cookies
- **multer:** handle multipart form data
- **serve-favicon:** serve a favicon
- **session:** establish server-based sessions (development only)
- **helmet:** helps secure your apps by setting various HTTP headers
- **passport:** authentication using "strategies" such as OAuth, OpenID, and many others

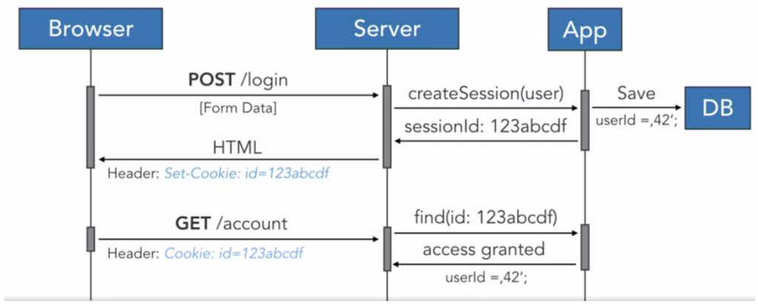
Using bcrypt to Hash and Compare Passwords

https://auth0.com/blog/hashing-in-action-understanding-bcrypt



Understanding Cookies and Sessions

Authentication Flow



Properties of Cookies

Set-Cookie:

- id=123abcdf;
- Expires=Wed, 14 Nov 2018 00:30:00 GMT;
- Secure;
- HttpOnly;

Reading Cookies in Express

```
const cookieParser = require(,cookie-parser‘);
app.use(cookieParser()); // gives us req.cookies
```

Handling Sessions in Express

```
const session = require(,express-session’);

const MongoStore = require('connect-mongo')(session);
app.use(session({
  secret: 'foo',
  store: new MongoStore(options)
}));
```

Introduction to Passport

Conceptual Overview

Using Passport in Express

- **passport.initialize();**
  Returns a middleware function that uses the request (req) object to store passport internal data in it.
- **passport.session();**
  Looks for a previously serialized user in the current session and uses a provided deserialization function to provide the user in *req.user* to all following middlewares and routes.

Authentication vs. Authorization

Authentication

- *Who* is this user?

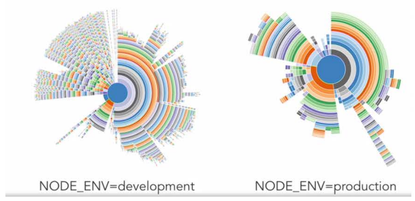
Authorization

- *What* is the user allowed to do?

Protecting Routes

Tuning Express Performance

Set NODE_ENV to production



NODE_ENV=development NODE_ENV=production

The drastic effects of omitting NODE_ENV in your Express.js applications

Daniel Khan — Digital experience
July 22, 2015

Most developers learn best by examples, which naturally tend to simplify matters and omit things that aren't essential for understanding. This means that the "Hello World" example, when used as starting point for an application, may be not suitable for production scenarios at all.

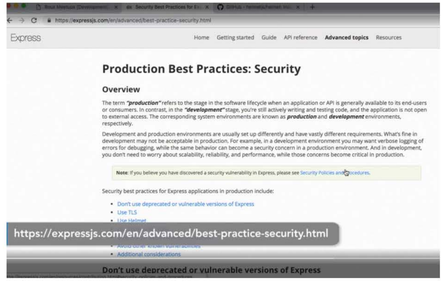
I started using Node.js like that and I have to confess that it took me almost two years to quantify the huge performance impact of omitting a single environment variable. In fact it was

https://lnkd.in/gxumB9A

Compress Server Responses

```
// npm install –save compression
const compression = require('compression');
const express = require('express');
const app = express();
app.use(compression());
```

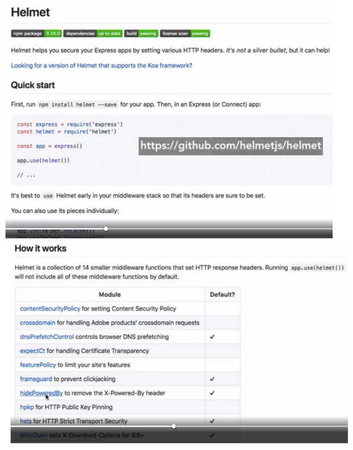
Server → gzip/deflate → Browser

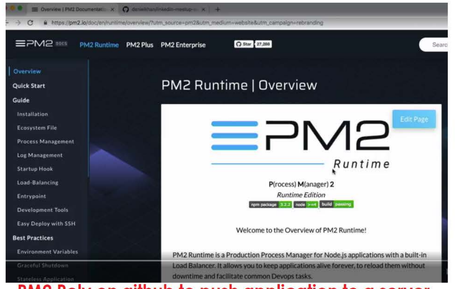



```
car-booth-20-mac:meetup danielkhan$ npm audit

=== npm audit security report ===

found 0 vulnerabilities
```

Further Recommendations

- Cache request results
- Don't use synchronous functions
- Don't use console.log() because it's synchronous
- Handle errors and exceptions properly
- Use a cluster





PM2 Rely on github to push application to a server

```
],
deploy: {
  production: {
    user: 'nodejs',
    host: 'meetup.demo.khan.io',
    ref: 'origin/master',
    repo: 'https://github.com/danielkhan/linkedin-meetup-sample',
    path: '/home/nodejs/deploy',
    'post-deploy': 'cp ../.env ./ && npm install && pm2 startOrRestart ecosystem.con
  },
},
```

runninng in http

Best practice for Prod: run node application behing web proxy such as NGINX - better to manage SSL and TLS

Prod Web Server: NGINX

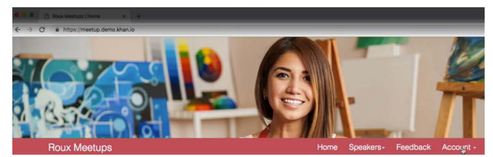
Provide free ssl self-signed certificate to server

Proxy configuration on, nginx

```
if (app.get('env') === 'production') {
  app.set('trust proxy', 'loopback');
  app.use(session({
    secret: 'another very secret 12345',
    name: 'sessionId',
    proxy: true,
    cookie: { secure: true },
    resave: true,
    saveUninitialized: false,
    store: new MongoStore({ mongooseConnection: mongoose.connection }),
  }));
} else {
```

```
car-booth-20-mac:meetup danielkhan$ pm2 update
car-booth-20-mac:meetup danielkhan$ pm2 deploy production
```

running in https behing proxy



Setting Up a Deployment Environment

1. Linux host (example: an EC2 instance with Ubuntu)
2. Install Node.js and PM2 (npm install -g pm2).
3. /home/nodejs/deploy owned by a user “nodejs”
4. Add local SSH public key to authorized_keys on server.
5. Push your project to a Git repository (example: GitHub).
6. Copy .env to the deployment directory.
7. Whitelist the server IP on MongoDB Atlas.