Fullstack Development

Authentication / Authorization

Authentication - `authen`

- A process of verifying user identity.
- Who is the user?
- Is the user really who he/she represents himself to be?

Authorization - `author`

- A process of verifying a user's access level.
- Is user `X` authorized to access resource `R`?
- Is user `X` authorized to perform operation `P`?

Note

`authen` and `author` do not exist separately.

- Users try to access protected APIs:
  - Applications might need to allow user based on role (`author`) but also need to know user identities (`authen`).
- Social login (i.e. Google):
  - Users verify themselves to Google (`authen`) but authorize applications (`author`) to access their resources.

Approach

Rather than talking about `authen` vs `author`, let's focus on requirements:

- How do users sign up/in with credentials?
- How do users sign up/in with social accounts?
- How do we persist users' auth states?
  - So that users don't need to sign in at every request.

Part 1: Signing up/in with credential

Situation

- User fill in username and password.
- Your app creates user entry in database.
- *How do you store password?*
    - (and also compare it?)

Part 1: Signing up/in with credential

Section 1A: How to store password

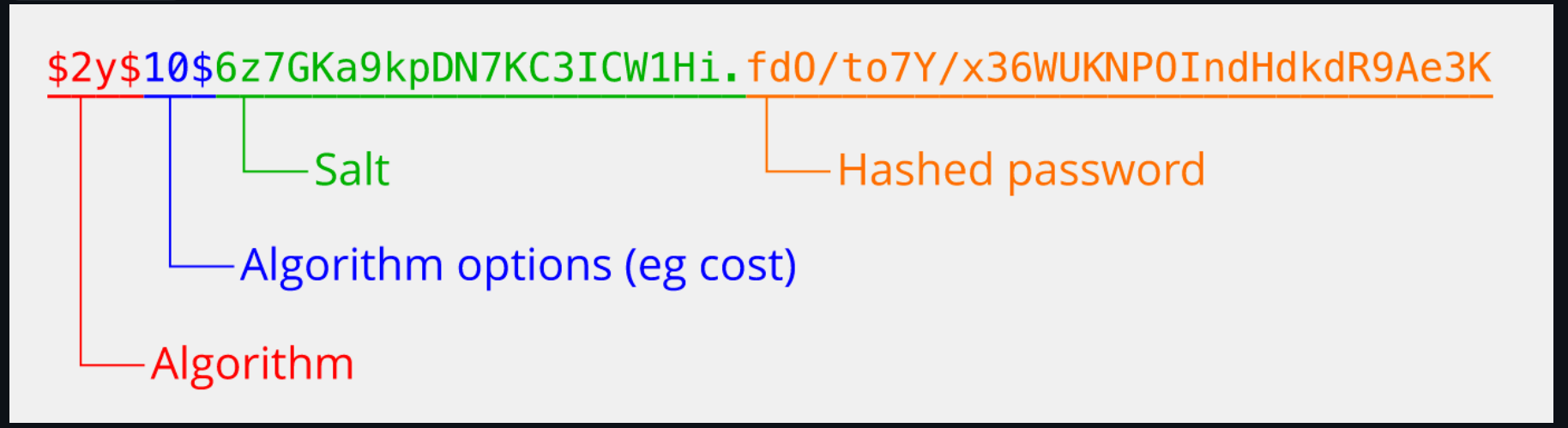
6 levels of safety

Technique	Ranking	Vunerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor (`bcrypt`)	B+	*Infinity stone* 🤣
?	A	

Adapted from source

Note (1)

- SHA256
- Rawinbow table attack
- `bcrypt` hash



Note (2)

- It should be noted that the resulting "hash" contain `salt`.
  - The inclusion of `salt` is so that we do not need to keep track of it.
- But this also leave room for hacker to use it to regenerate rainbow table on the fly.
  - This is why we need cost factor.

Note (3)

- In `bcrypt`, "salt rounds" (also known as the "cost factor" or "work factor") refer to the number of iterations or rounds the hashing algorithm performs when generating a password hash.

`bcrypt` example

- `git clone https://github.com/fullstack-68/auth-bcrypt.git`
- `pnpm i`
- `npx run hash`
- `npx compare`

Note on the code

- Increasing time to generate (and compare) hash with incrasing `saltRounds`.
- The use of `bcrypt.compare`

Part 1: Signing up/in with credential

Section 1B: Implementation with `passport`

`passport`

- Most popular authentication middleware for `express`.
- Minimal and modular
- 500+ strategies (click at button)
- Confusing and poorly documented 🤣
  - Hidden manual

Let's see it

- `git clone https://github.com/fullstack-68/auth-signin-credential.git`
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

Side note about the project

- MPA - HTMX
- Use `SQLite` + `drizzle`.
  - Checkout the schema.
- Try debugging in VSCode.
  - See `launch.json`.

Highlighed packages

`package.json`

```
{
  "passport": "^0.7.0",
  "passport-local": "^1.0.0"
}
```

(Not chaning from last year)

Middleware

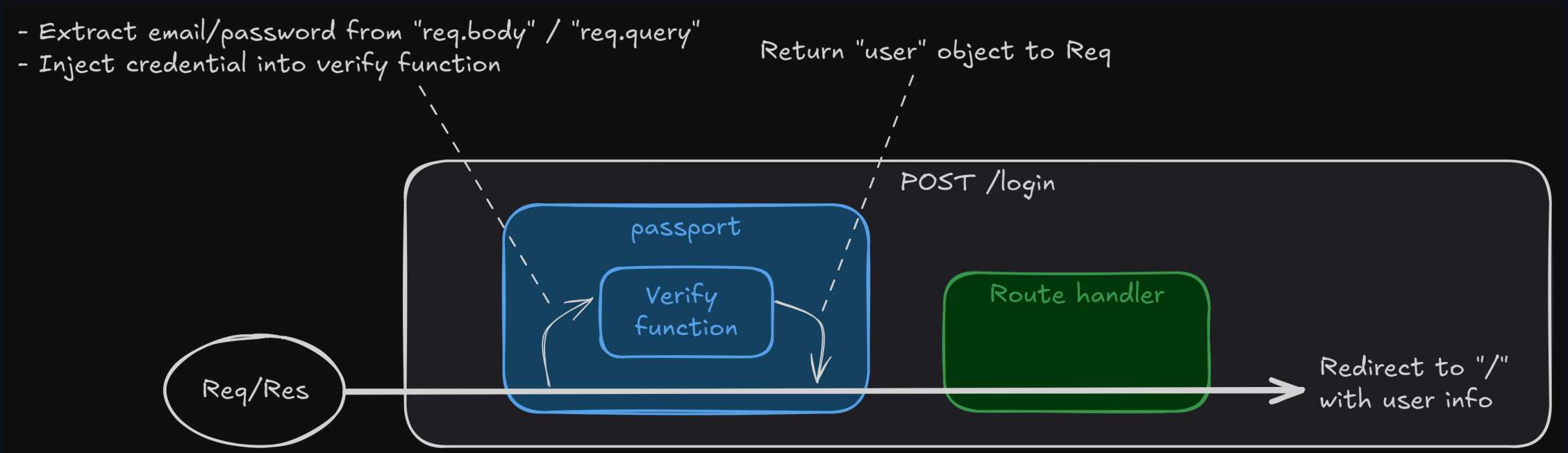
src/index.ts

```
passport.use(
  new LocalStrategy(
    {
      // Options
    },
    async function (email, password, done) {
      // Verify email / password
    }
  )
);
//
app.use(passport.initialize());
```

Available options

Route

```
app.post(
  "/login",
  passport.authenticate("local", { session: false }),
  function (req, res) {
    // * Passport will attach user object in the request
  }
);
```


- Extract email/password from "req.body" / "req.query"
- Inject credential into verify function
Return "user" object to Req
POST /login
passport
Verify function
Route handler
Req/Res
Redirect to "/" with user info

Can we do better?

Technique	Ranking	Vunerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor (`bcrypt`)	B+	*Infinity stone*
Not storing password	A	👈👈👈

Next: Part 2