

# 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

`authn` 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 (`authn`).
- Social login (i.e. Google):
  - Users verify themselves to Google (`authn`) but authorize applications (`author`) to access their resources.

# Approach

Rather than talking about `authn` 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	Vulnerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor ( <code>bcrypt</code> )	B+	<i>Infinity stone</i> 🚀
?	A	

Adapted from [source](#)

# Note (1)

- SHA256
- Rainbow table attack
- `bcrypt` hash

The diagram shows a bcrypt hash string: `$2y$10$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K`. The string is color-coded to show its components: red for the algorithm (`$2y`), blue for algorithm options (`$10`), green for the salt (`$6z7GKa9kpDN7KC3ICW1Hi`), and orange for the hashed password (`.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K`). Lines connect these colored segments to their respective labels: Algorithm (red), Algorithm options (eg cost) (blue), Salt (green), and Hashed password (orange).

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f d0/to7Y/x36WUKNP0IndHdkdR9Ae3K

Algorithm

Algorithm options (eg cost)

Salt

Hashed password

## 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.

## bcrypt example

- `git clone -b bcrypt https://github.com/fullstack-67/auth-mpa-v2.git auth-bcrypt`
- `pnpm i`
- `npx tsx ./src/hash.ts`
- `npx tsx ./src/compare.ts`

## Note on the code

- Promisify the callback style.
- Increasing time to generate (and compare) hash with increasing `saltRounds`.
- The use of `bcrypt.compare`
- Use of `debug` package.

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 poor documented 🤔
  - Hidden manual



# Let's see it

- `git clone -b signin-credential https://github.com/fullstack-67/auth-mpa-v2.git auth-signin-credential`
- `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`.

# Highlighted packages

package.json

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

# Middleware

src/index.ts

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

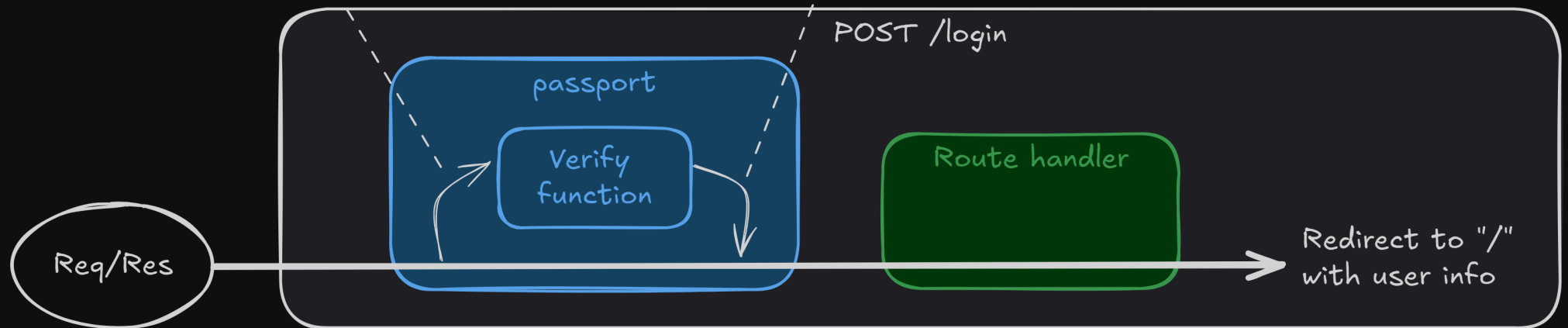
## Available options

# 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



**Can we do better?**

Technique	Ranking	Vulnerability
Plain text	F	All
Encryption	D	Stolen key
Hashing	C	Rainbow table attack
Salting	B	Fast computer
Salting + Cost Factor ( <code>bcrypt</code> )	B+	<i>Infinity stone</i>
<b>Not storing password</b>	A	👉👉👉



**Next: Part 2**