Fullstack Development

Authentication / Authorization

Part 3: Persisting auth's state

Part 3: Social signing up/in

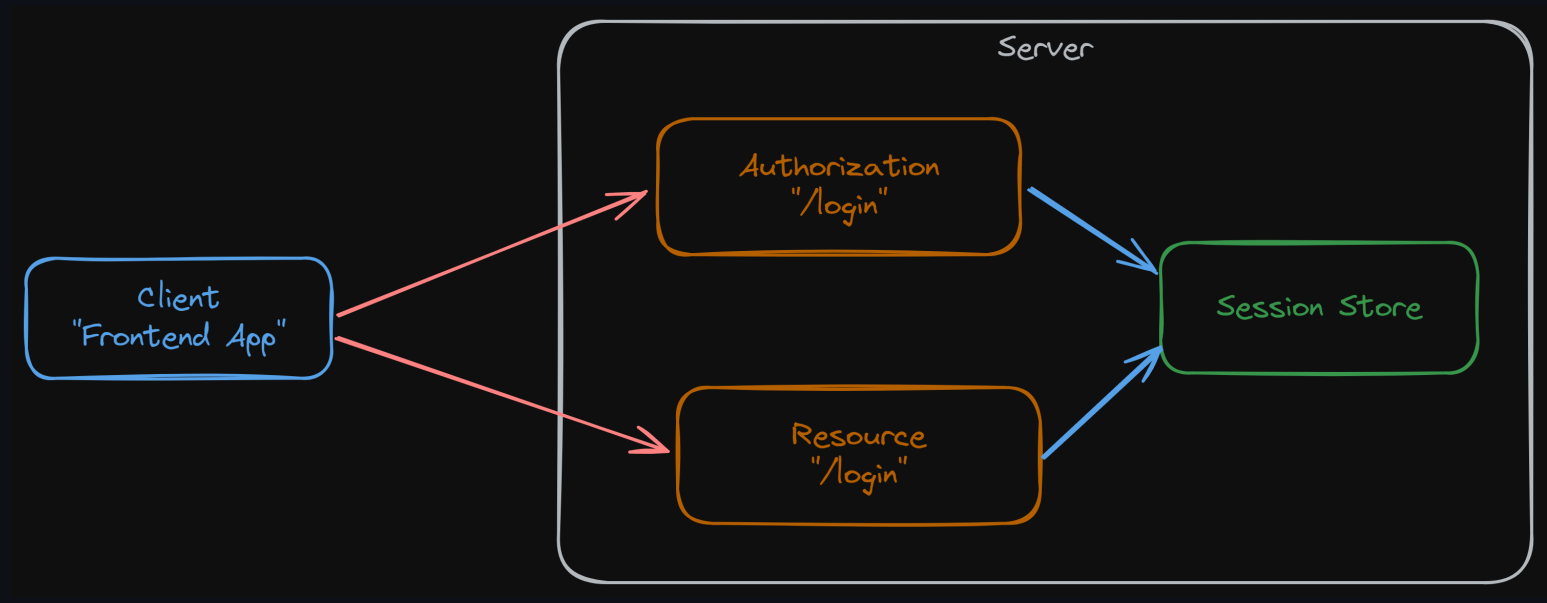
Section 3A: Session-based vs token-based

Session based

- Server is responsible for creating and maintaining the user's authentication state (i.e. in a database).
- After user sign-in, the server sets a cookie that contains the session ID and sends it to the browser.
  - The browser will include it in all further requests.
  - The server will use the cookie to identify the current user session from the database.

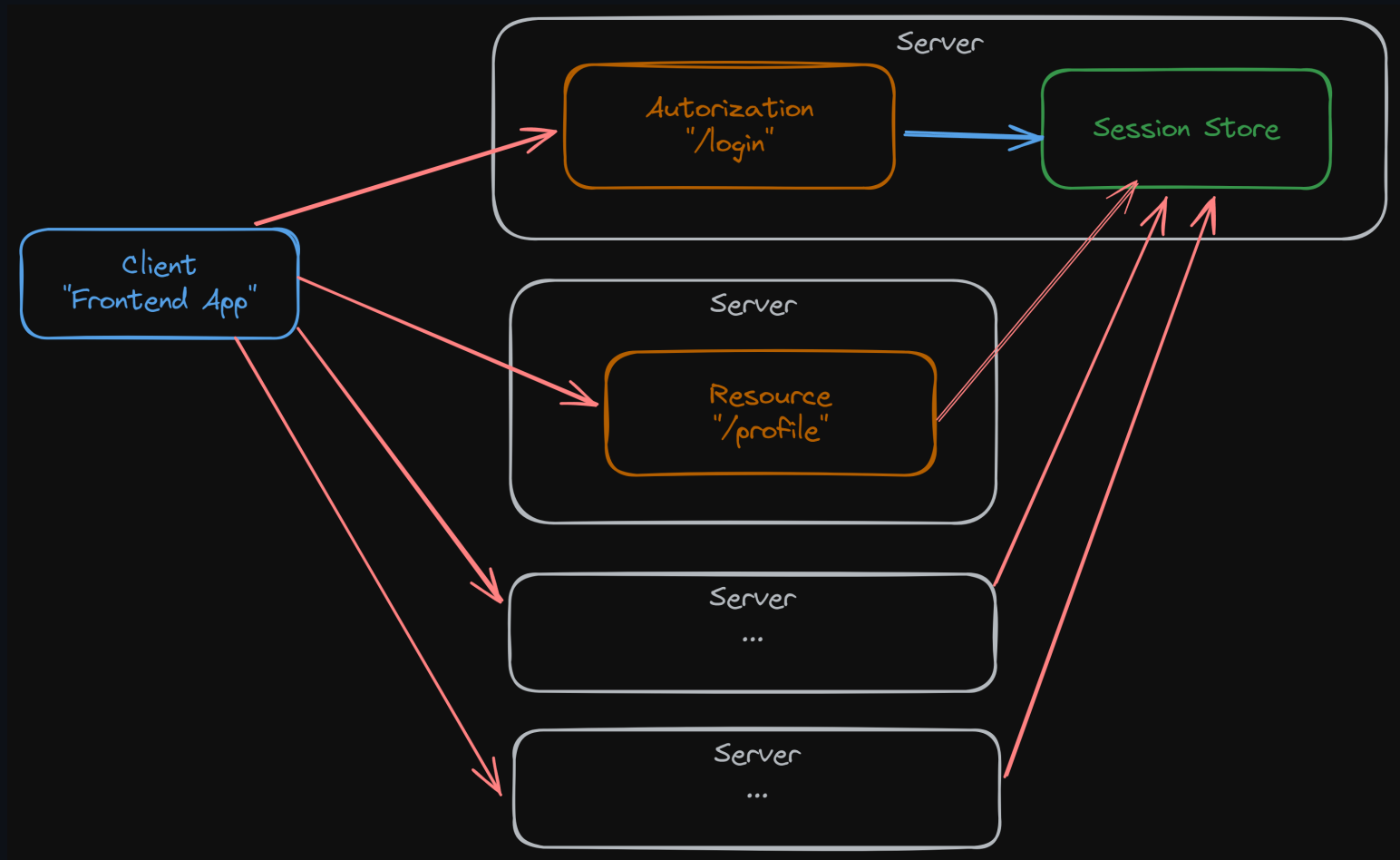
Session based

- Users' auth states are in DB.
- Need to query DB at every request.

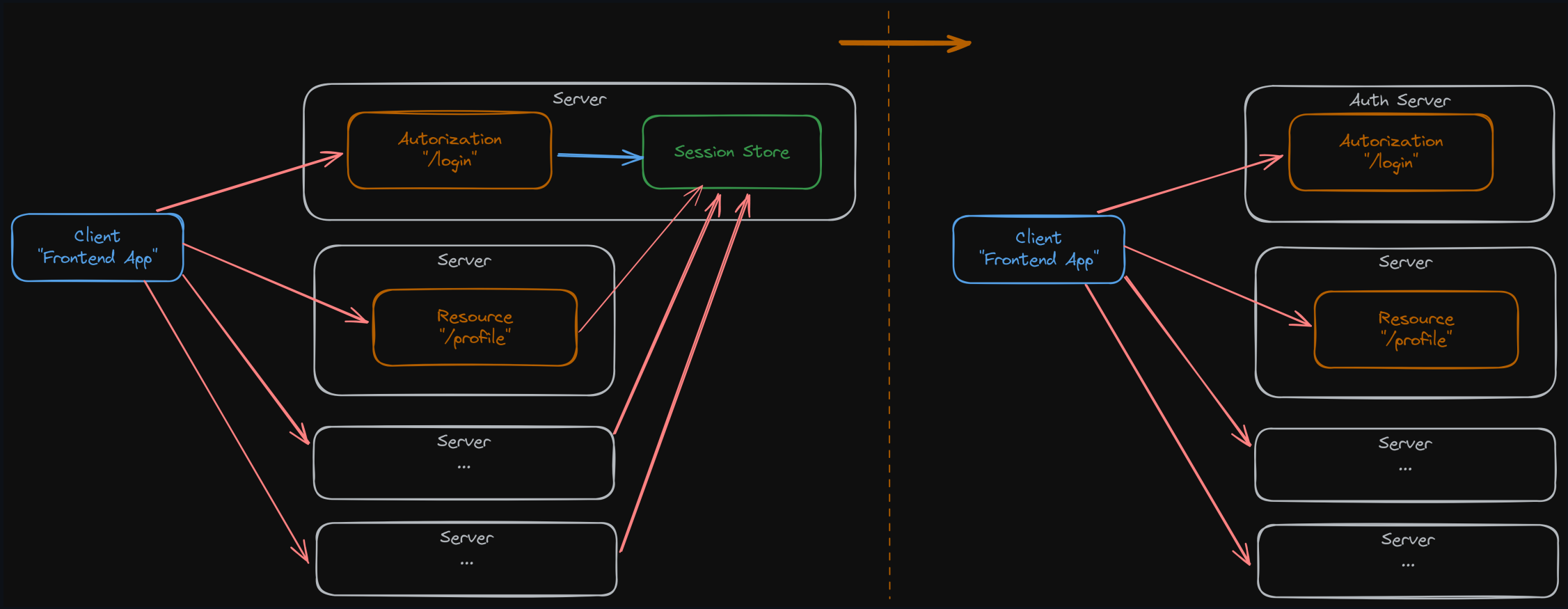


Session based

- This could be a problem in distributed system with centralized `auth` server.
- Session store could be overloaded.



Can do something like this?



Note that the right system is not exactly what you want to do.

Token-based

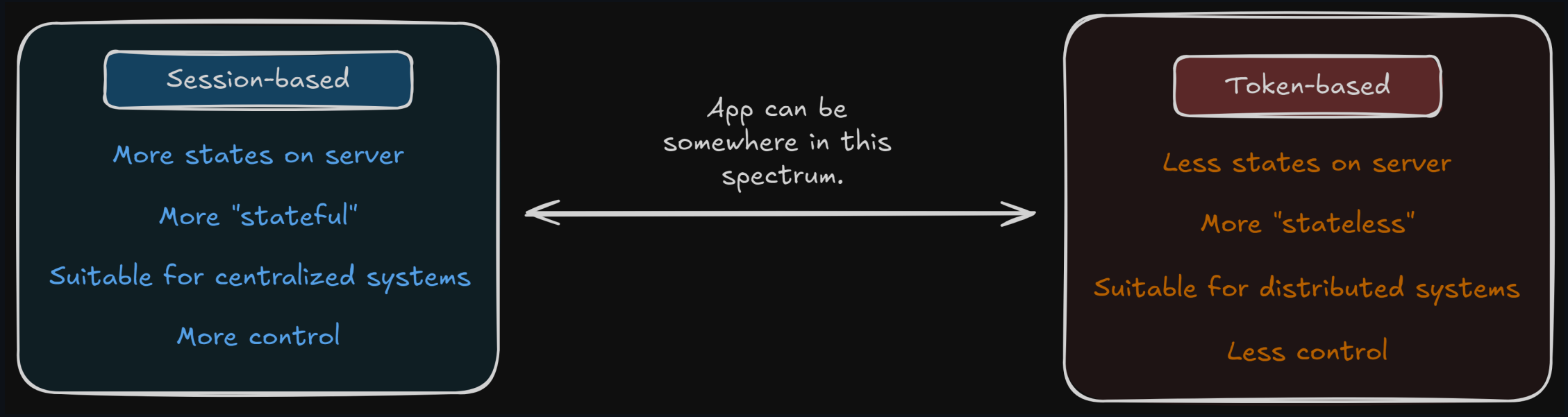
- `token` is a cryptographically signed piece of data that contains information about the authenticated user and their access permissions.
- The server will only have to verify the validity of the token rather than having it stored in a database.
  - Reduces the amount of state that needs to be stored on the server.
- While other token formats exist, JSON Web Tokens (JWTs) have become the prevailing standard for token-based approach.

JWT Test

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

Clarification

- It is better to think about where you put users' `auth` state.
  - `Session-based`: more states in server (*"stateful"*)
  - `Token-based`: more states in client (*stateless*)
- Using JWTs does not automatically means you are using token-based approach.
  - You can put JWTs in session cookie.
- The system can contain both approaches.



- When going token-based approach, you are losing **control** over user's state and you are making your system **less secured**.

Please do not do this.

- It is tempting to go **100% stateless** using token-based approach (JWT) to avoid dealing to storing information on server.
  - **You don't know who is using your system!**
- Also, be aware of these concerns (Ref1, Ref2).
  - Cannot really log out users.
  - Cannot really block users.
  - Stale data
  - Limited storage
  - JWT could be decrypted at some point.

Considering token-based approch?

- Do you have distributed system with centralized auth server?
  - If no, go session-based.
- You are concerned about overloading your database.
  - Have you considered `redis`?

Considering token-based approach?

- Have you consider the fact that modern token secuity is quite complex (*and will require database anyway*)?
  - Refresh tokens (revokable)
  - Allowed/Revoked lists
  - Token rotation
  - Token behavior detection

Bottom line

> If you don't know who are using your apps, please do something about it.

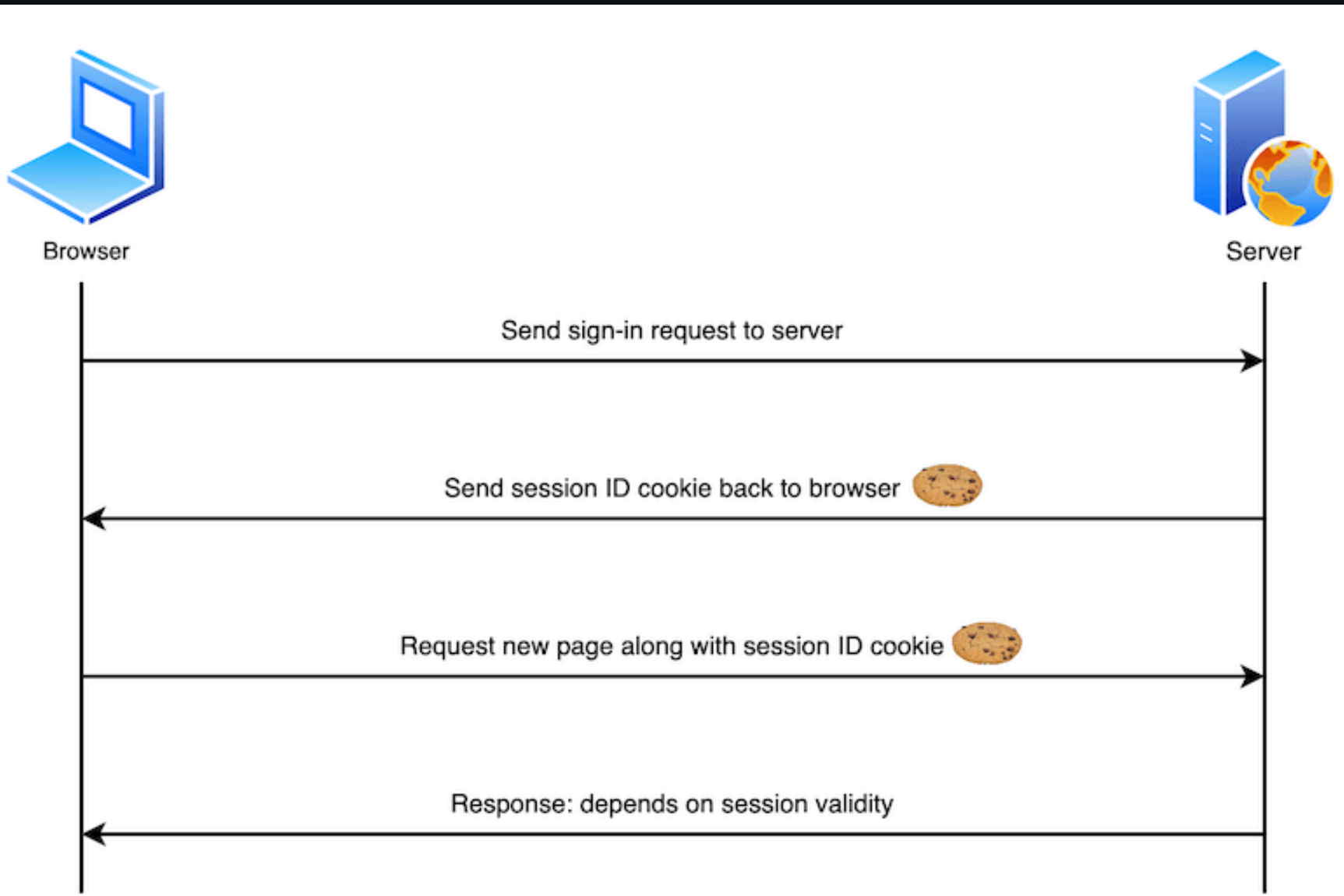
> Part 3: Persisting auth's state

Section 3B: Session management with `express-session`

Cookie

- A small piece of data a server sends to a user's web browser.
- The browser may:
  - Store cookies
  - Create new cookies
  - Modify existing ones
  - Send it back to the server with later requests.
- Cookies enable web applications to store limited amounts of data and remember state information
  - By default the HTTP protocol is `stateless`.

Cookie


Browser
Server
Send sign-in request to server
Send session ID cookie back to browser
Request new page along with session ID cookie
Response: depends on session validity

Cookie mechanism

- Server `response` header

```
HTTP/1.1 200 OK
Set-Cookie: connect.sid=s%3AUDOk...; Path=/; Expires=Fri, 30 Aug 2024 02:57:01
GMT; HttpOnly; SameSite=Lax
```

- Subsequent browser `request` header

```
GET / HTTP/1.1
Cookie: connect.sid=s%3AUDOk
```

Cookie attributes

- `Path=<path-value>`
  - Path that must exist in the requested URL for the browser to send the Cookie header
- `Expires=<date>`
  - Maximum lifetime
- `Max-Age=<number>`
  - The number of seconds until the cookie expires.
- `Secure`
  - Sent cookie to the server only with `https` schema.

Cookie attributes

- `HttpOnly`
  - Forbids JavaScript from accessing the cookie (`Document.cookie`).
  - Prevent against cross-site scripting (XSS).
- `SameSite`
  - Controls whether or not a cookie is sent with cross-site requests
  - `Strict` / `Lax` / `None`
  - Will come back to this later.

Setup

- `git clone -b session https://github.com/fullstack-67/auth-mpa-v2.git auth-session`
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

Highlighted package

`package.json`

```
{
  "express-session": "^1.18.0"
}
```

Usage

session.ts

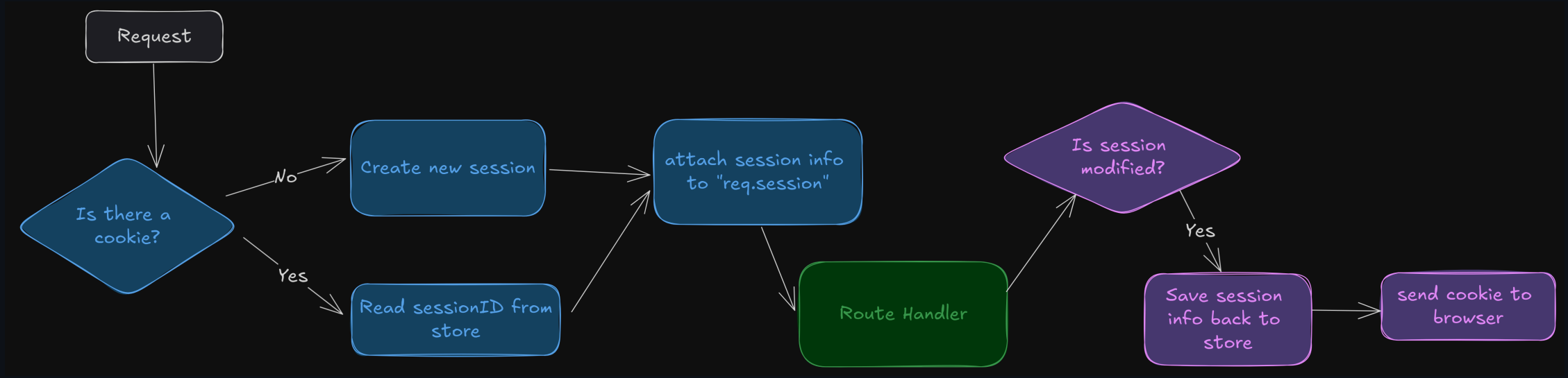
```ts
import session from "express-session";
// ...
const sessionIns = session({
  // Options
});
```

index.ts

```ts
app.use(sessionIns);
```

How does `express-session` work?

How does `express-session` work?



Session store

- Storage mechanism for sessions.
- If you don't supply anything, it just uses a `memory` store.
  - Not persisted across server restarts
- Other choices

Experiments

- Clear all cookies in browser and visit the `url`.
  - No cookie sent from server.
- Set `count` in `req.session`
  - Cookie saved in store.
  - Cookie sent from server.
- Open new tab/window.
  - Cookie are sent with client requests.
- Open Edge.
  - New sessionse are created.
- Set `useragent`.

Session options

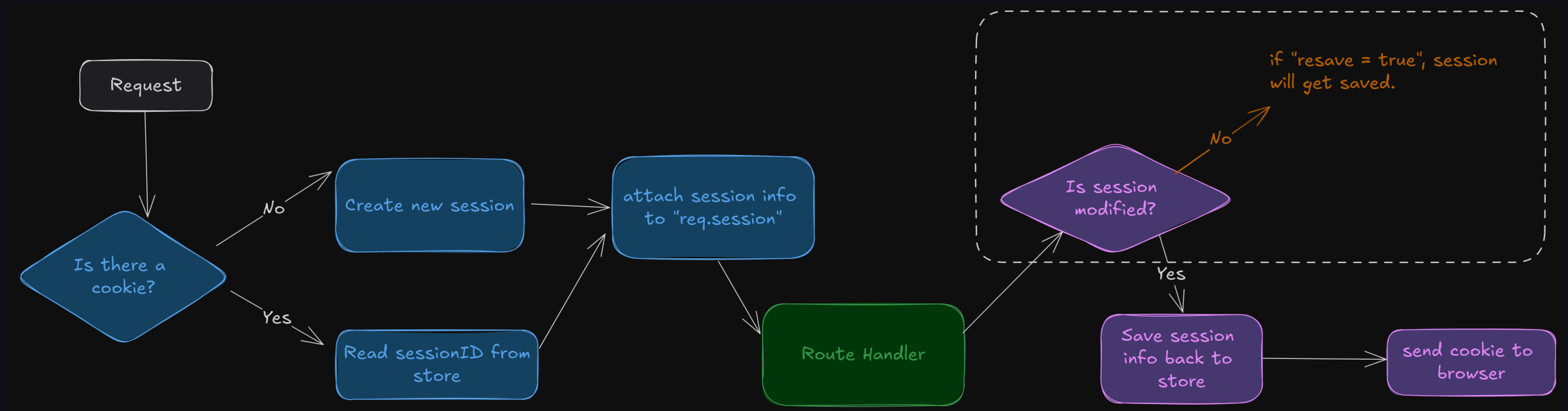
./src/auth/session.ts

```
const sessionIns = session({
  secret: "My Super Secret",
  cookie: {
    path: "/",
    httpOnly: true,
    secure: NODE_ENV === "production" ? true : false,
    maxAge: 60 * 60 * 1000,
    sameSite: "lax",
  },
  saveUninitialized: false,
  resave: false,
  store: SQLiteStoreInstance as session.Store,
});
```

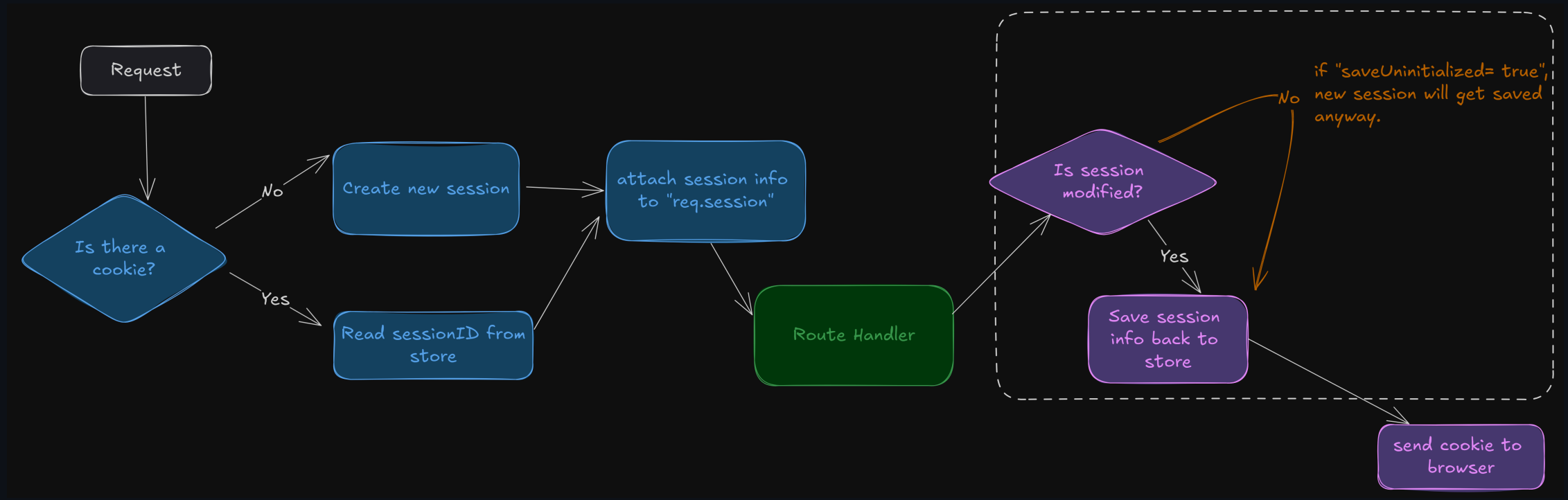
Session options

- `saveUninitialized`
  - Forces a session that is "uninitialized" to be saved to the store.
  - A session is uninitialized when it is new but not modified.
- `resave`
  - Forces the session to be saved back to the session store, even if the session was never modified during the request.

saveUninitialized



resave


Request
Is there a cookie?
No
Create new session
Yes
Read sessionID from store
attach session info to "req.session"
Route Handler
Is session modified?
No
if "saveUninitialized= true", new session will get saved anyway.
Yes
Save session info back to store
send cookie to browser

Remaining task

- We need a way to link authentication state to session.

Part 3: Persisting auth's state

Section 3C: Session + authentication

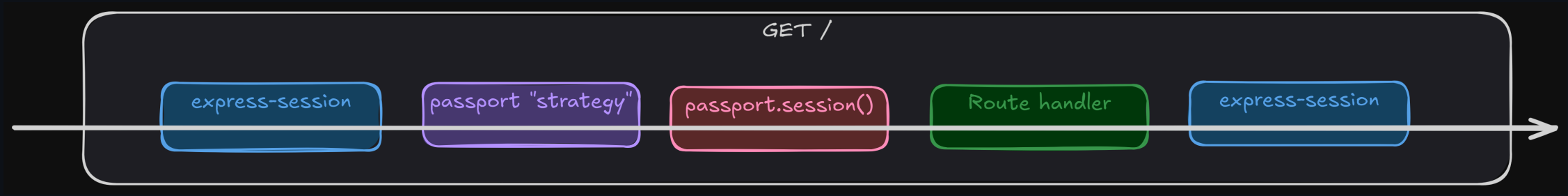
We need to

- Store user information in a session store when user sign in.
- Retrive user information for route handlers for subsequent requests.
- Destroy sessions when users log out.

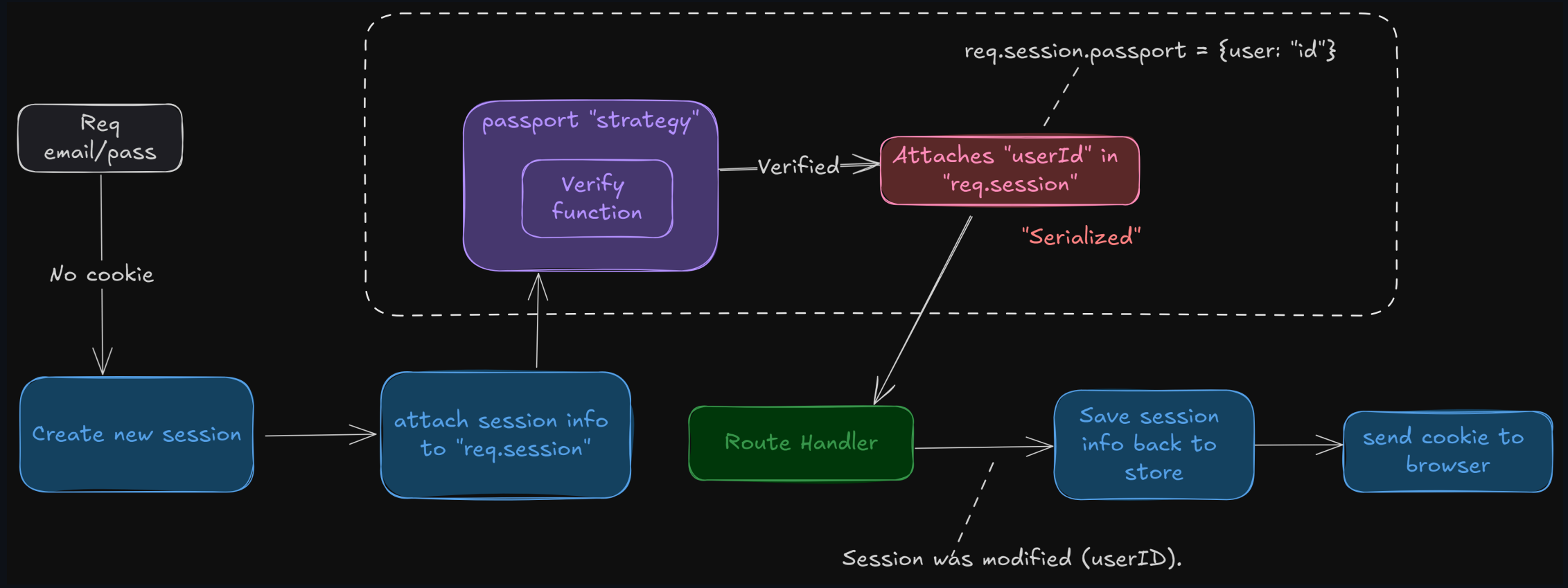
Two session middlewares

- `express-session`
  - Middleware to save/retrieve session from a session store.
- `passport.session()`
  - Middleware to append/retrive user information from session information.
- `REF1`, `REF2`

Structure

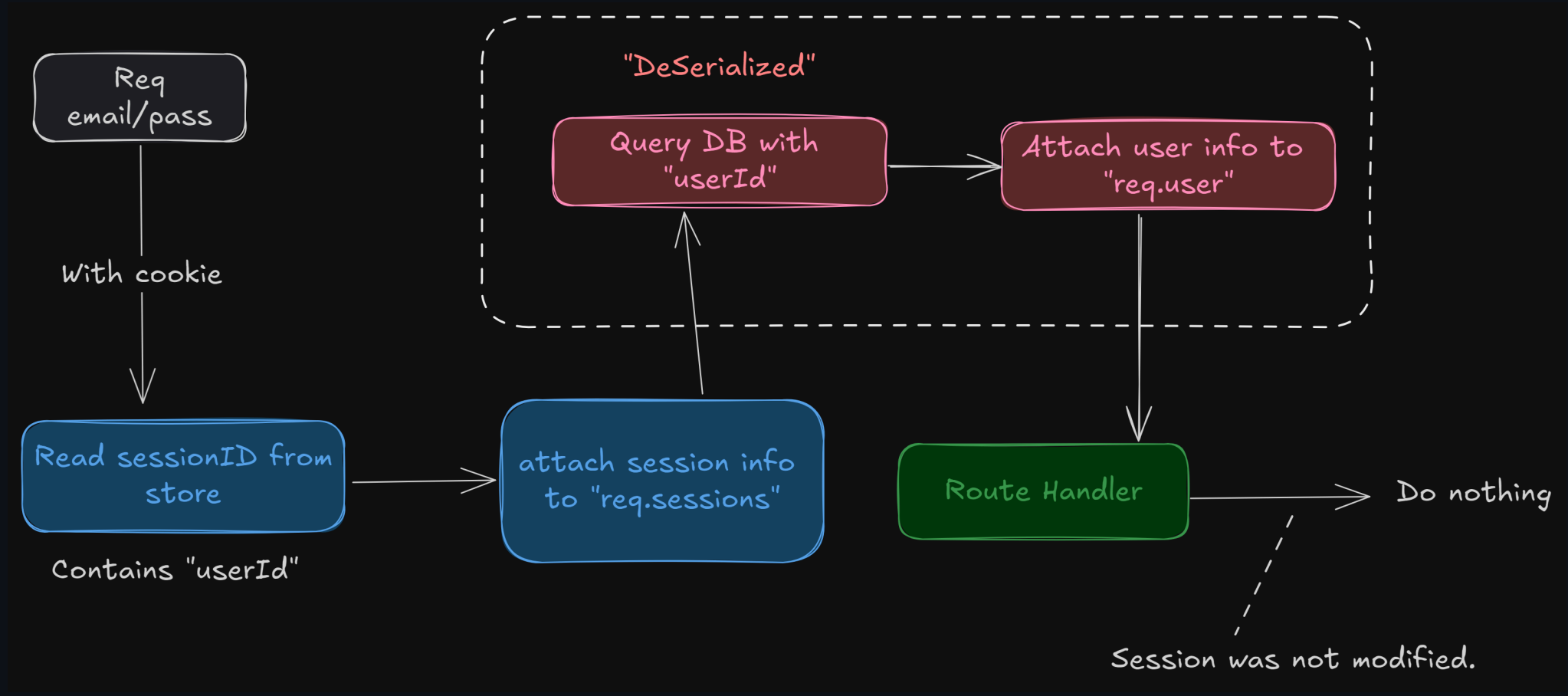

GET /
express-session
passport "strategy"
passport.session()
Route handler
express-session

User login



Note: Serialization

Subsequent requests



Note: Deserialization

Setup

- `git clone -b signin-credential-session https://github.com/fullstack-67/auth-mpa-v2.git auth-signin-credential-session`
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

Registering middlewares

./src/index.ts

```
// * express-session
app.use(sessionIns);

// * Passport
app.use(passportIns.initialize());
app.use(passportIns.session()); // 👉👉👉 passport.session
```

Serialization

./src/auth/passport.ts

```
passportIns.serializeUser(function (user, done) {
  // Sending user.id to session.
  done(null, user.id);
  // You can put all user info in the session
  // done(null, user);
});
```

Note: Putting all user info in session is not recommended.

Deserialization

./src/auth/passport.ts

```
passportIns.deserializeUser<string>(async function (id, done) {
  const query = await dbClient.query.usersTable.findFirst({
    where: eq(usersTable.id, id),
  });
  if (!query) {
    done(null, false);
  } else {
    done(null, query);
  }
});
```

Experiments

- Sign in
  - Note the user `id` in `req.session.passport`
- Sign out
- Sign in from two browsers.
  - Try removing other sessions. *(Cool!)*

`SameSite` cookie revisited

- `Strict`
  - Sent on first-party request only.
- `Lax`
  - Sent on third-party requests from top-level navigation and `GET` requests.
- `None`
  - No restriction.

Experiments

- With `strict` cookie (`./src/auth/session.ts`)
  - Redirect through `<a>` tag result in "guest" view.
    - Subsequent navigation will yield "user" view.
  - If navigate by changing `url` in the address bar results in correct "user" view.
- With `lax` cookie
  - Redirect through `<a>` tag will yield correct "user" view.

Typescript tips

- Extending `req.user` types with `drizzle` orm.

`./src/types/express.d.ts`

```
type usersTableType = typeof usersTable.$inferSelect;
declare global {
  namespace Express {
    interface User extends usersTableType {} 👉👉👉
  }
}
```

Typescript tips

- Extending `req.session` types

`./src/types/session.d.ts`

```
declare module "express-session" {
  interface SessionData {
    useragent?: Details;
    createdAt?: number;
    loginType: LoginType;
    passport?: { user: string };
  }
}
```

Tips

- I use `sessionID` that contains `userID` so that I can easily query all sessions that belongs to the same users.

`./src/auth/session.ts`

```
const generateSessionKey = (req: Request) => {
  const userId = req.user?.id ?? nanoid();
  const randomId = nanoid();
  return `sid:${userId}:${randomId}`;
};
```

- Query all user sessions

./db/repositories.ts

```
export async function getAllUserSessions(userId: string) {
  // ...
  const results = await dbClient
    .select()
    .from(sessionsTable)
    .where(like(sessionsTable.sid, likeString));
  // ...
}
```

Part 3: Persisting auth's state

Section 34: Putting everything together

Moving parts

- `Auth` Framework
  - `passport`
- `Auth` strategies
  - `passport-local`
  - `passport-oauth2`
- Session management
  - `express-session`
  - `passport.session()`

Setup

- `git clone -b main https://github.com/fullstack-67/auth-mpa-v2.git auth-main`
- `pnpm i`
- Fill in `.env`
- `npm run db:reset`
- `npm run dev`

Features

- Signup by credential.
- Login by credentials
- Login by social providers
- Session show all sessions same user.
- Delete another session.