Web Application Security

Web Security Aspects

- **Authentication** (**Identity verification**):
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he/she claims to be
- **Authorization** (**Controlling access**):
  - Determine if the user should be granted access to a particular resource/functionality.
- **Confidentiality**:
  - Encrypt sensitive data (e.g., credit card details) to prevent unauthorized access in transit or in storage
- **Data Integrality**:
  - Sign sensitive data (e.g., authentication token) to prevent the content from being tampered (e.g., changed in transit)

Token based Authentication & Authorization

JWT

Token based Authentication & Authorization

- After a successful authentication a **JSON Web Token (JWT)** is issued by the server and communicated to the client
- JWT token is a signed json object that contains:
  - Claims (i.e., information about *issuer* and the *user*)
  - Signature (encrypted hash for tamper proof & authenticity)
  - An expiration time
- Client must send JWT in an **HTTP authorization header** with subsequent Web API requests
- Web API (i.e., a resource) **validates** the received token and makes authorization decisions (typically based on the user's **role**)

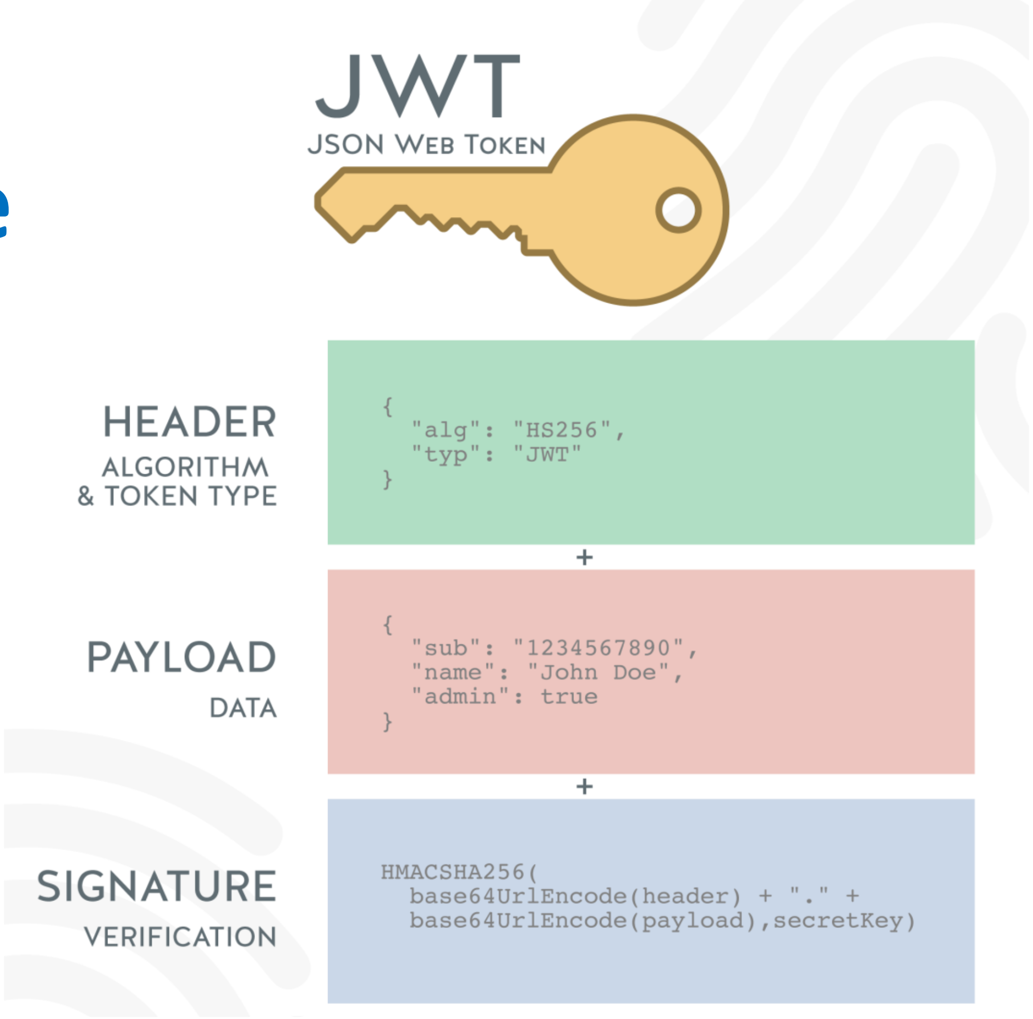
Web pages Session Management using JWT

- Implements stateless sessions by:
  - Creating utility functions (`createSession`, `verifySession`, `deleteSession`) to manage sessions
  - Using the jose library to create and verify signed/encrypted JWTs containing user object and expiration
  - Storing the JWT in a secure, HttpOnly cookie using `cookies()` from `next/headers`
  - Redirecting users after login/signup or when access is denied using `redirect()` from `next/navigation`

Advantages of Token based Security

- A primary reason for using token-based authentication is that it is **stateless** and **scalable** authentication mechanism
  - It is suitable for SPA, Web APIs, Web pages and mobile apps
  - The token is stored on the client-side
  - The claims in a JWT are encoded as a **JSON** object that contains information that is useful for making authorization decisions
  - JWT is a simple and widely useful security token format with libraries available in most programming languages
- Can be used for **Single Sign-On:**
  - Sharing the JWT between different applications

JWT Structure



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

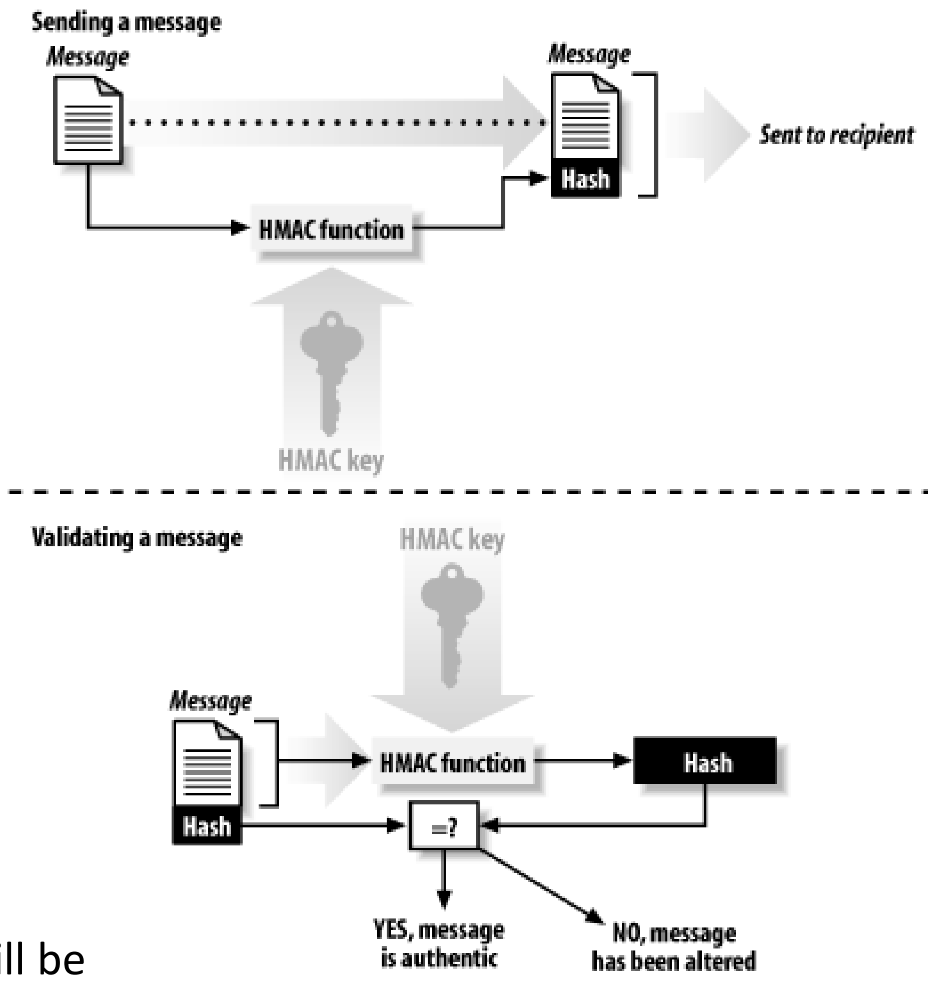
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

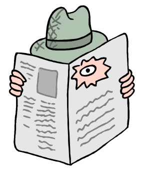
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),secretKey)
```

$$\underbrace{\texttt{eyJhbGciOiJub25lIn0}}_{\textbf{Header}}\texttt{.}\underbrace{\texttt{eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD}}_{\textbf{Payload}}\texttt{.}\underbrace{\texttt{4MTkzODAsDQogImh0dHA6Ly9leGFt}}_{\textbf{Signature}}$$

Hash-based Message Authentication Code (HMAC)

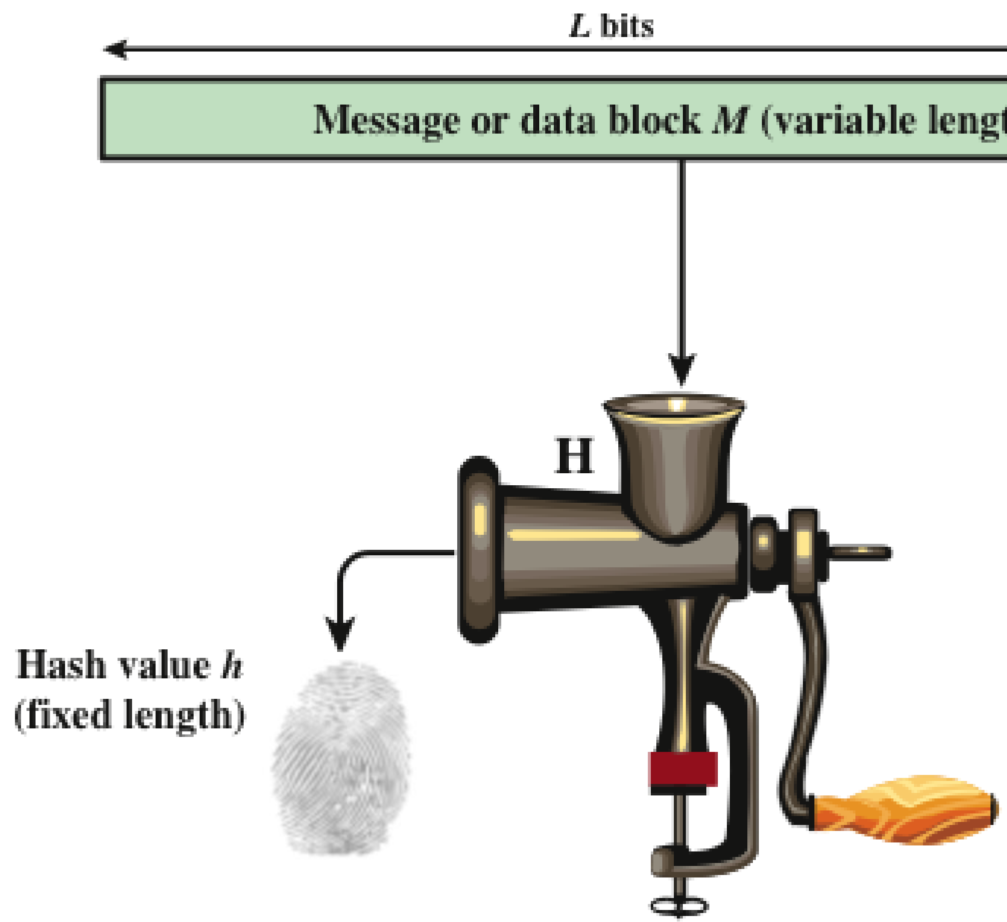
- **HMAC-SHA256** is often used for **signing JWT** to ensure its integrity
- HMAC-SHA256 is a *cryptographic hash function* that uses SHA256 hashing and **a *secret key*** *to generate a MAC (i.e., JWT signature)*
- The MAC is appended to the message sent
- MAC provides **message integrity**: Any manipulations of the message during transit will be detected by the receiver





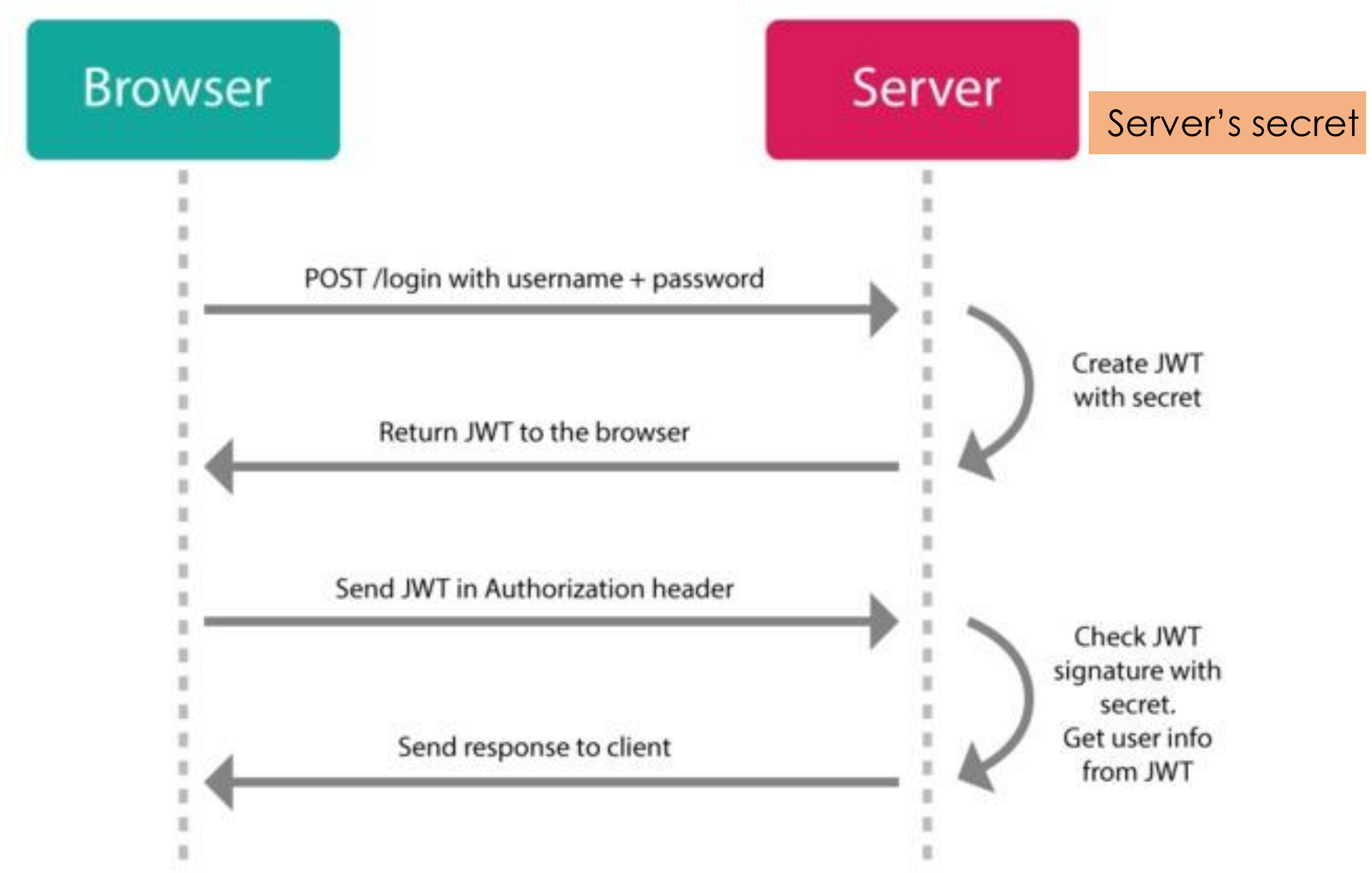
An attacker who alters the message will be **unable** to alter the associated MAC value without knowledge of the secret key

Hashing



Hash functions are used to compute a digest of a message. It takes a variable size input, produce fixed size pseudorandom output

JWT for Web API

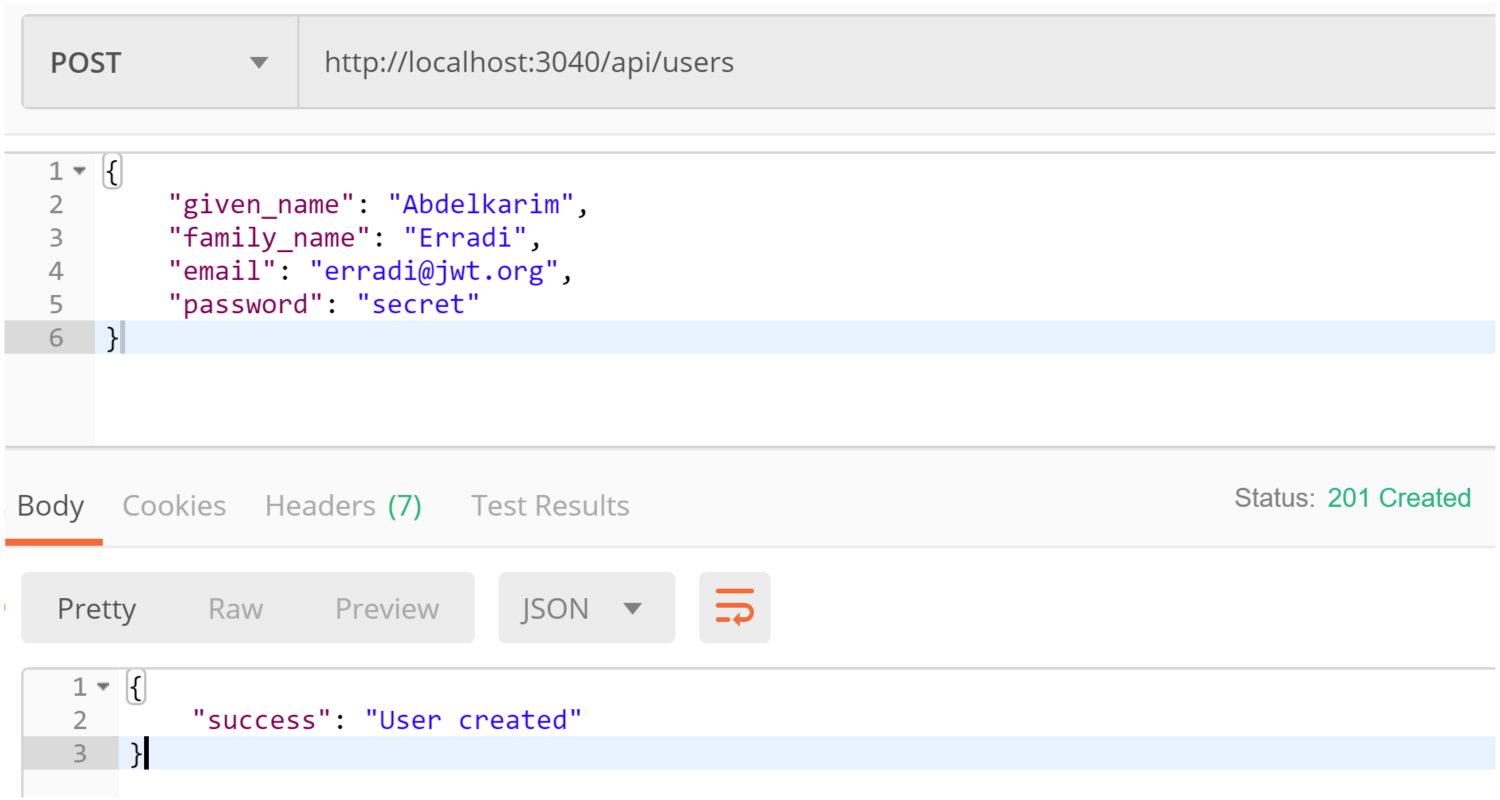

Browser
Server
Server's secret
POST /login with username + password
Create JWT with secret
Return JWT to the browser
Send JWT in Authorization header
Check JWT signature with secret. Get user info from JWT
Send response to client

- Every request to a Web API must include a **JWT** in the Authorization header
- Web API checks that the JWT token is valid
- Web API uses info in the token (e.g., **role**) to make authorization decisions

Sign-Up Example

- Sign up @ http://localhost:3040/api/users

Try it with Postman



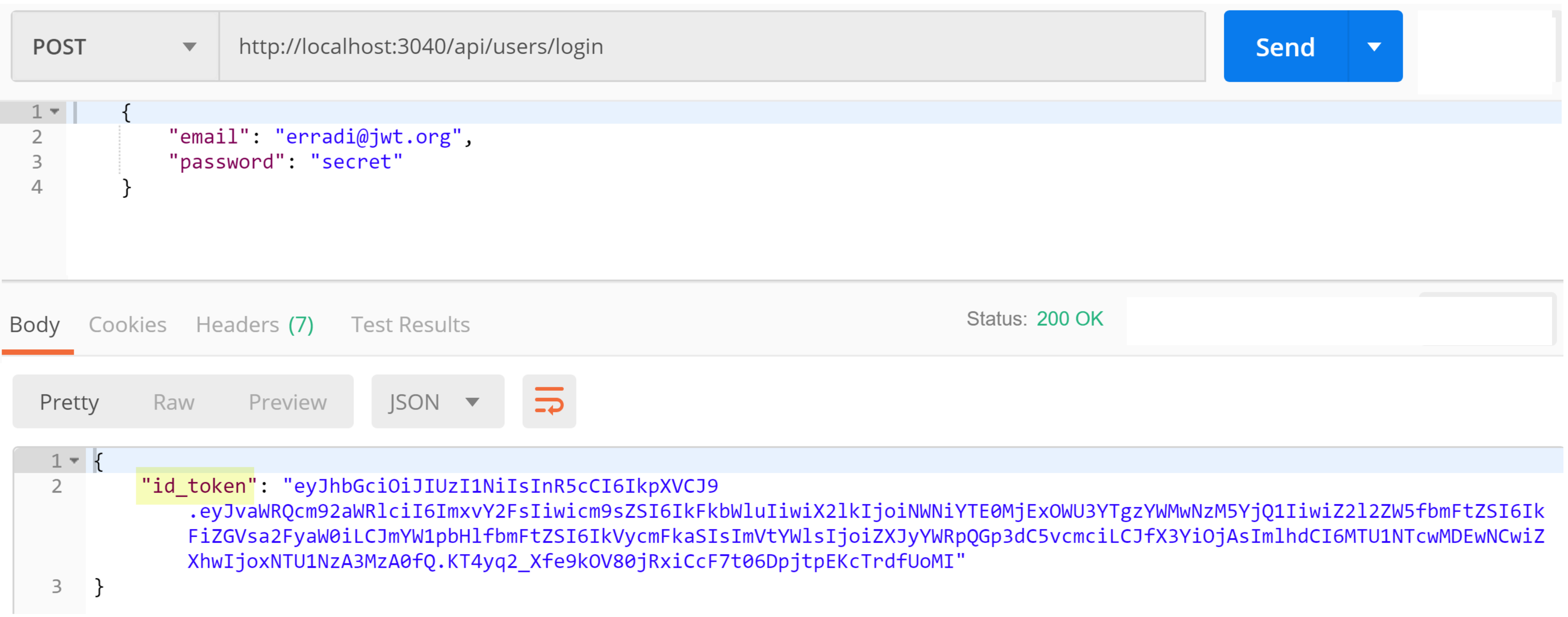
http://localhost:3040/api/users

```
{
    "given_name": "Abdelkarim",
    "family_name": "Erradi",
    "email": "erradi@jwt.org",
    "password": "secret"
}
```

```
{
    "success": "User created"
}
```

Successful Login to get JWT

- Sign in @ http://localhost:3040/api/users/login



http://localhost:3040/api/users/login

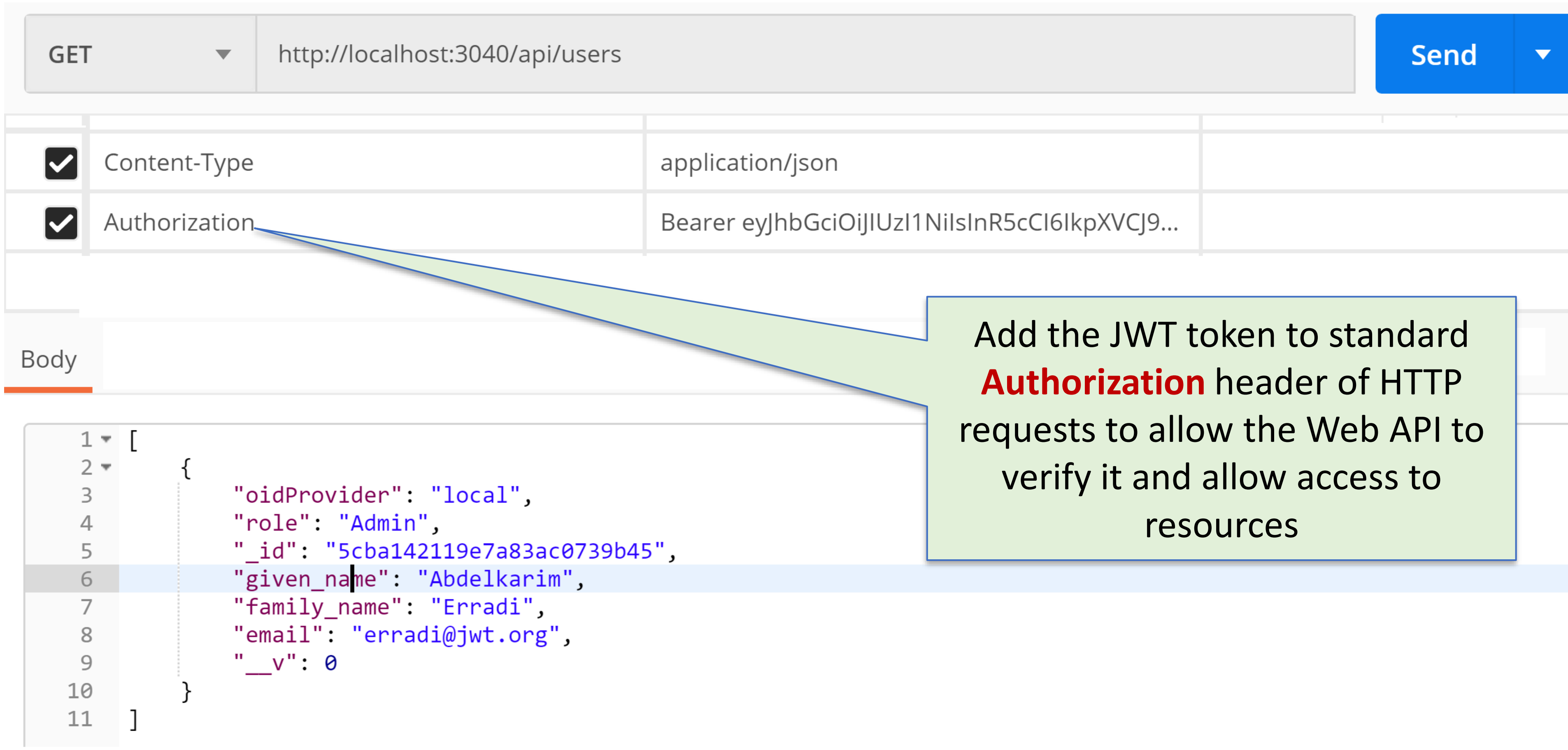
```
{
    "email": "erradi@jwt.org",
    "password": "secret"
}
```

Status: 200 OK

```
{
    "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJvaWRQcm92aWRlciI6ImxvY2FsIiwicm9sZSI6IkFkbWluIiwiX2lkIjoiNWNiYTE0MjExOWU3YTgzYWMwNzM5YjQ1IiwiZ2l2ZW5fbmFtZSI6Ik
        FiZGVsa2FyaW0iLCJmYW1pbHlfbmFtZSI6IkVycmFkaSIsImVtYWlsIjoiZXJyYWRpQGp3dC5vcmciLCJfX3YiOjAsImlhdCI6MTU1NTcwMDEwNCwiZ
        XhwIjoxNTU1NzA3MzA0fQ.KT4yq2_Xfe9kOV80jRxiCcF7t06DpjtpEKcTrdfUoMI"
}
```

Use JWT to Access Protected Resource

- Get users http://localhost:3040/api/users



```
http://localhost:3040/api/users
```

✓	Content-Type	application/json
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

```
[
    {
        "oidProvider": "local",
        "role": "Admin",
        "_id": "5cba142119e7a83ac0739b45",
        "given_name": "Abdelkarim",
        "family_name": "Erradi",
        "email": "erradi@jwt.org",
        "__v": 0
    }
]
```

Add the JWT token to standard **Authorization** header of HTTP requests to allow the Web API to verify it and allow access to resources

Storing JWT in Browser Local Storage

Local Storage allows storing a set of name value pairs directly accessible with **client-side** JavaScript

- **Store**

```
localStorage.id_token = "eyJhbnR5cCI…."
```

- **Retrieve**

```
Console.log(localStorage.id_token)
```

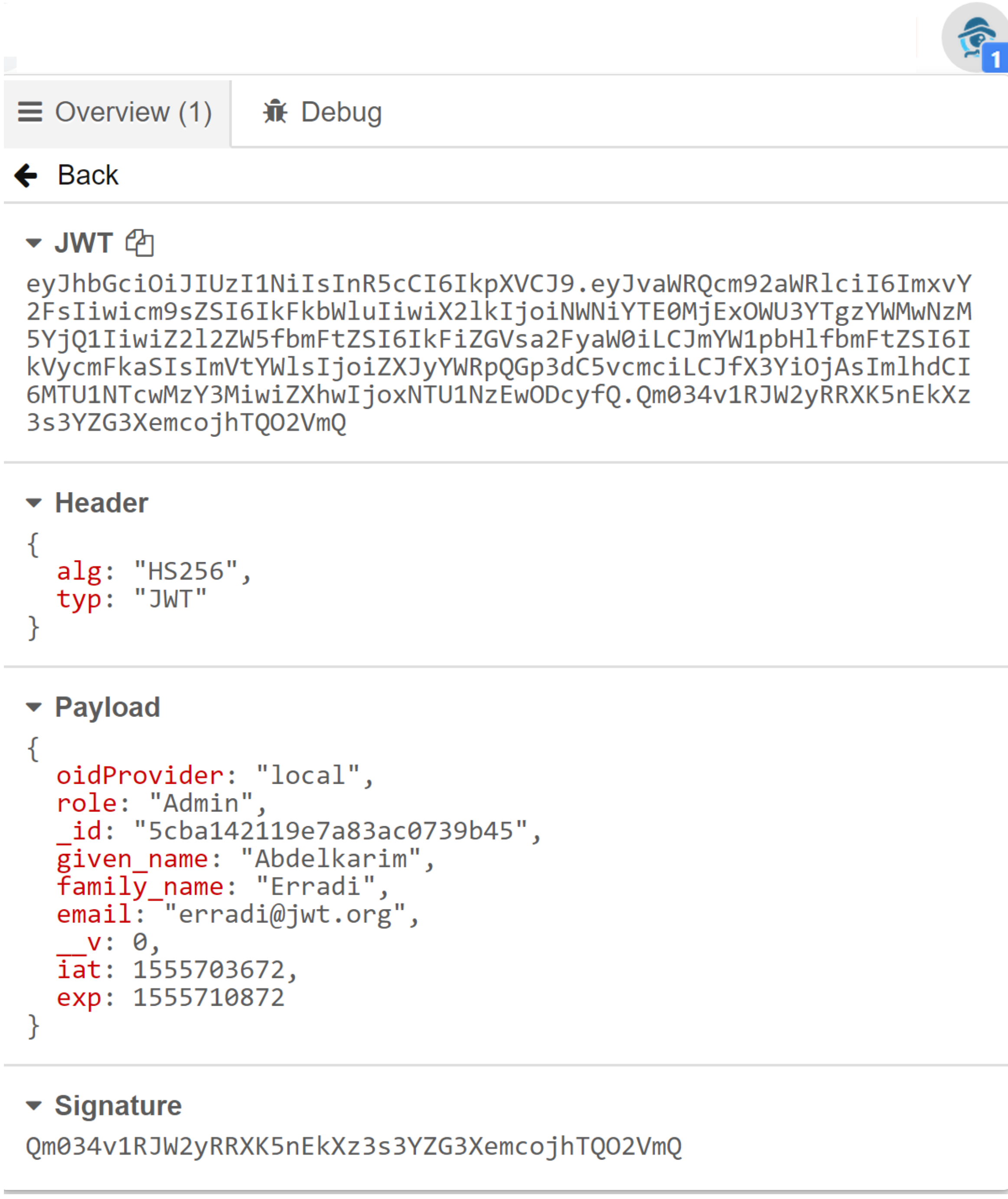
- **Remove**

```
delete localStorage.id_token
```

- **Remove all saved data**

```
localStorage.clear();
```

JWT Inspector is a chrome extension that lets you **decode** and **inspect** JWT in requests, and local storage



JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY
2FsIiwicm9sZSI6IkFkbWluIiwiX2lkIjoiNWNiYTE0MjExOWU3YTgzYWMwNzM
5YjQ1IiwiZ2l2ZW5fbmFtZSI6IkFiZGVsa2FyaW0iLCJmYW1pbHlfbmFtZSI6I
kVycmFkaSIsImVtYWlsIjoiZXJyYWRpQGp3dC5vcmciLCJfX3YiOjAsImlhdCI
6MTU1NTcwMzY3MiwiZXhwIjoxNTU1NzEwODcyfQ.Qm034v1RJW2yRRXK5nEkXz
3s3YZG3XemcojhTQO2VmQ
```

Header

```
{
  alg: "HS256",
  typ: "JWT"
}
```

Payload

```
{
  oidProvider: "local",
  role: "Admin",
  _id: "5cba142119e7a83ac0739b45",
  given_name: "Abdelkarim",
  family_name: "Erradi",
  email: "erradi@jwt.org",
  __v: 0,
  iat: 1555703672,
  exp: 1555710872
}
```

Signature

```
Qm034v1RJW2yRRXK5nEkXz3s3YZG3XemcojhTQO2VmQ
```

401 vs. 403

- ***401 Unauthorized***

- Should be returned in case of failed authentication

- ***403 Forbidden***

- Should be returned in case of failed authorization

- The user is authenticated but not authorized to perform the requested operation on the given resource

Middleware.js to Check Authentication

- Use middleware.js to check if the user is **authenticated** and **authorized** before handling their request

```
const protectedRoutes = ["/", "/assessments", "/comments", "/workload-
report"];

export function middleware(req) {
  const token = req.cookies.get("auth_token")?.value;
  const path = req.nextUrl.pathname;

  console.log("Middleware - Request Path:", path);
  // Check if the current path is a protected route
  const isProtectedRoute = protectedRoutes.some(
    (route) => path === route || (route !== "/" && path.startsWith(route))
  );
  // Redirect to login if accessing a protected route without a token
  if (isProtectedRoute && !token) {
    return NextResponse.redirect(new URL("/login", req.url));
  }
  // Continue if authenticated or accessing public route
  return NextResponse.next();
}
```

Resources

- Good resource to learn about JWT

https://jwt.io/

- JWT Handbook

https://auth0.com/resources/ebooks

- **Auth.js** to authenticate using GitHub, Google, …etc.

- Top 10 Web Application Security Risks

https://owasp.org/www-project-top-ten/