Web Application Security

Outline

1. Token based Token based Authentication & Authorization (JWT)
2. Authorization for Node.js App

Web Security Aspects

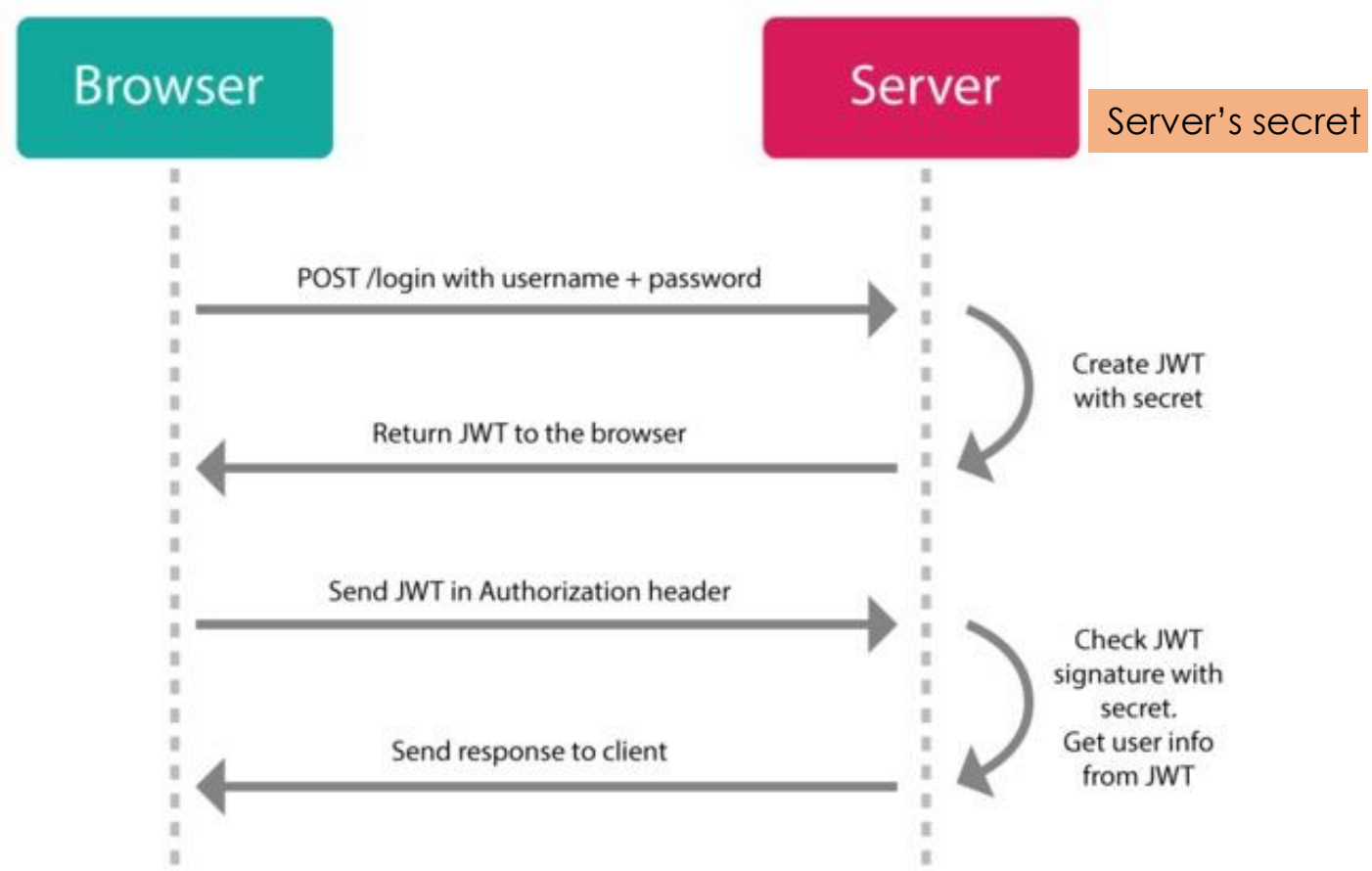
- **Authentication (Identity verification)**:
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he/she claims to be
- **Authorization**:
  - Determine if the user should be granted access to a particular resource/functionality.
- **Confidentiality**:
  - Encrypt sensitive data to prevent unauthorized access in transit or in storage

Token based Authentication & Authorization

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**)

JSON Web Token (JWT)



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

JWT Structure

HEADER
ALGORITHM & TOKEN TYPE

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

+

PAYLOAD
DATA

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

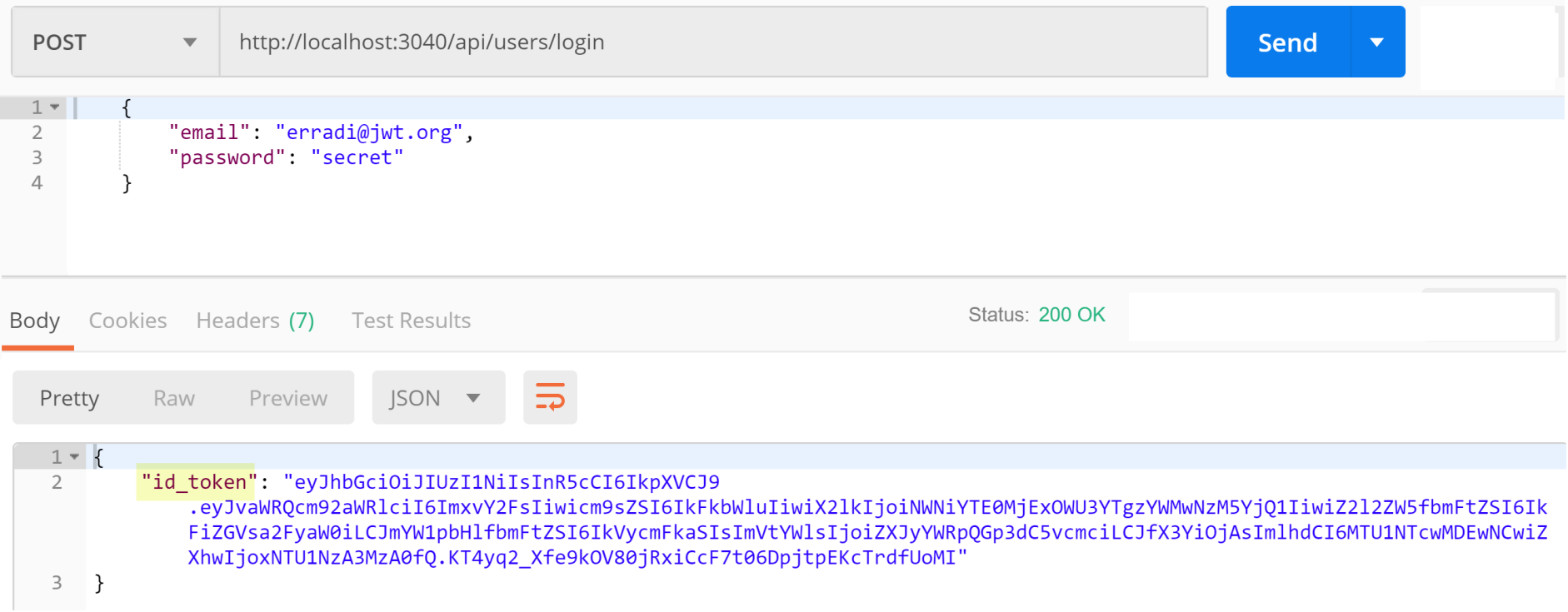
+

SIGNATURE
VERIFICATION

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

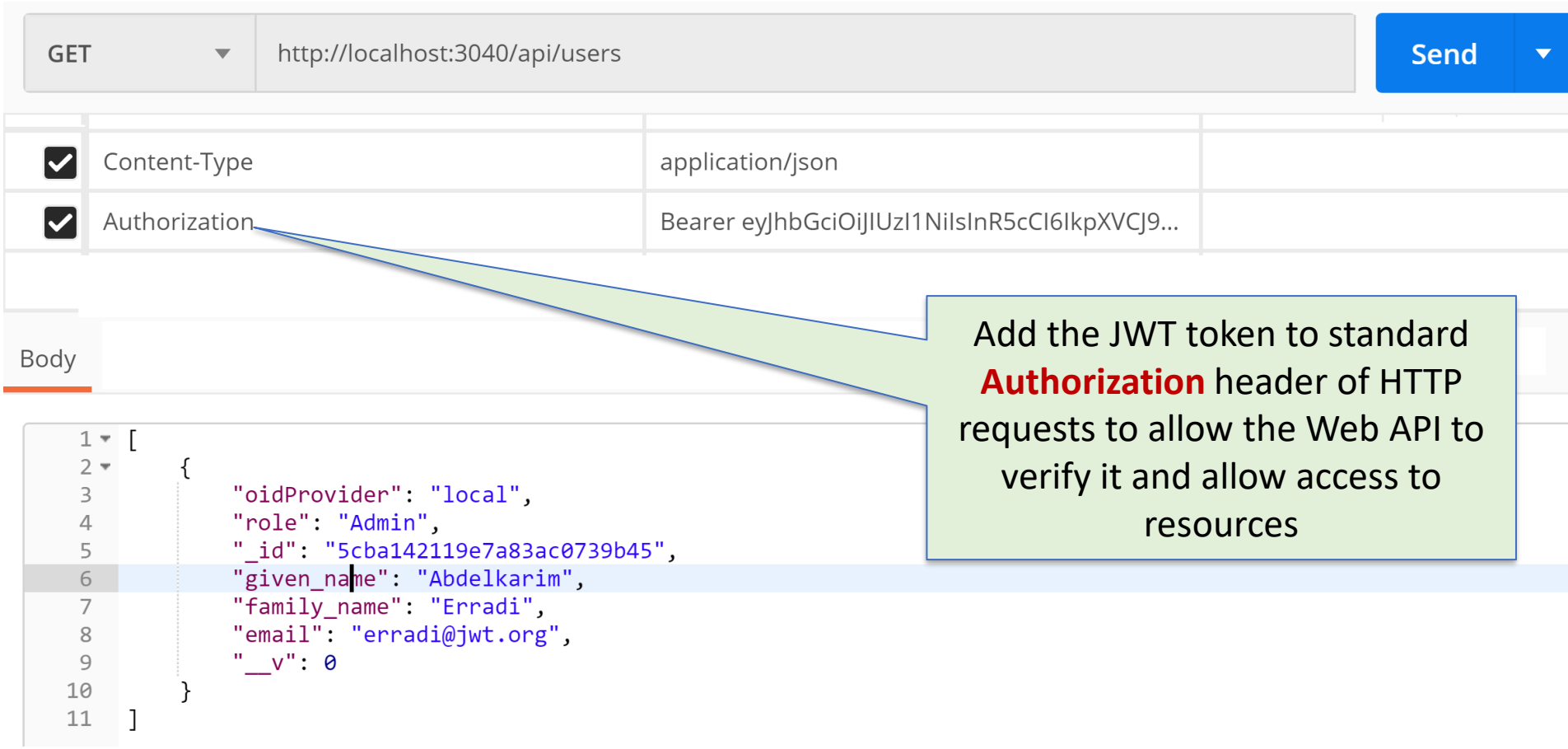
Successful Login to get JWT

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



Use JWT to Access Protected Resource

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



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();
```

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

Node.js Middleware to Check Authorization

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

```
isAuthenticated(req, res, next) {
    let id_token = req.headers.authorization;
    console.log("received id_token: ", id_token);
    if (!id_token) {
        res.status(401).json({error: "Unauthorized. Missing JWT Token"});
        return;
    }
    try {
            id_token = id_token.split(" ")[1];
            //Decode and verify jwt token using the secret key
            const decodedToken = jwt.verify(id_token, keys.jwt.secret);
            //Assign the decoded token to the request to make the user details
            //available to the request handler
            req.user = decodedToken;
            console.log("decodedToken: ", decodedToken);
            next();
    } catch (e) {
        res.status(403).json({error: "Forbidden. Invalid JWT Token"});
    }
}
```

```
router.get('/users', isAuthenticated, async function (req, res) {
    if (req.user.role == 'Admin') {
        const users = await userRepository.getUsers();
        res.json(users);
    } else {
        res.status(403).json({ error: "Access denied" });
    }
});
```

Resources

- JWT Handbook

https://auth0.com/resources/ebooks/jwt-handbook

- Good resource to learn about JWT

https://jwt.io/