Authentication

Client-server authentication using JWT

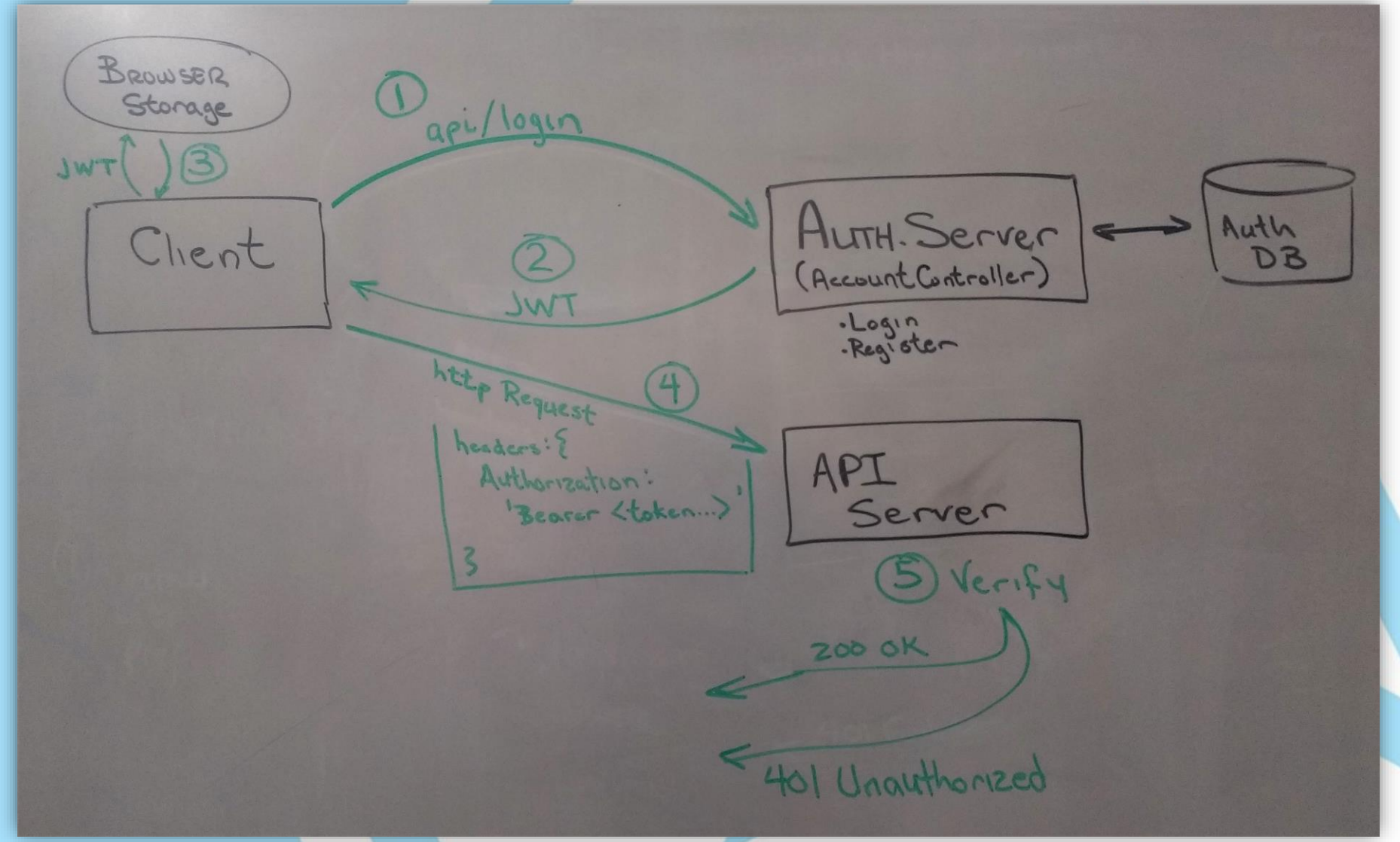
JWT - JSON Web Token

- Yes, JWT is an acronym within an acronym (better than JSONWT)
- Allows two parties to pass around information and guarantees that it has not been tampered with (digitally signed)
  - Not encrypted (although it could be)
- The information in the token is called Claims
  - A claim of who the user is
  - A claim of what roles the user has
  - Others if your application needs them
- Application uses Bearer Authentication
  - User who bears a valid token is authorized

The Process

1. The client requests authorization from an authorization server
   A. In our case, our API is also our authorization server
   B. Client passes credentials (encrypted, of course)
2. Server validates credentials; creates and returns a token (JWT)
   A. Token contains user name; may contain roles and more
3. Client stores the token locally (browser local storage)
4. With every API request, client passes token in the headers
   A. Authorization: Bearer <token>
5. With every API call, server verifies token and authorizes user
   A. Verify signature is valid (not tampered)
   B. Verify token has not expired

The Process (diagram)



Sample Project Setup

- Setup the DB
  - [optional] Modify Schema.sql to create a users table in your application database
  - In SSMS, run Schema.sql
  - [optional] Run data.sql to add a default user (uid: user, pwd: greatwall)
- Client (Vue project)
  - Open the file called ".env" at the root of the project
  - Modify the VUE_APP_REMOTE_API variable
- Server (.NET project)
  - Change the default route for AccountController to **[Route("api")]**

Protecting client routes

- If a route requires user to be logged in
  - Use **requiresAuth** on the route meta object
  - User will be re-directed to the login page if not logged in

```
19  const router = new Router({
20    mode: 'history',
21    base: process.env.BASE_URL,
22    routes: [
23      {
24        path: '/',
25        name: 'home',
26        component: Home,
27        meta: {
28          requiresAuth: true
29        }
30      },
31      {
32        path: "/login"
```

Protecting server methods

- If a method requires the user to be logged in
  - Use the [Authorize] attribute on the method
  - You can also use it on the class
  - Use Roles parameter for specific roles
  - [Authorize(Roles="Admin")]
  - Comma-separate for multiple roles

```
[HttpGet]
[Authorize]
public IActionResult Get()
{
    var result = $"Welcome back {User.Identity.Name}";
    return Ok(result);
}
/// <summary>
```