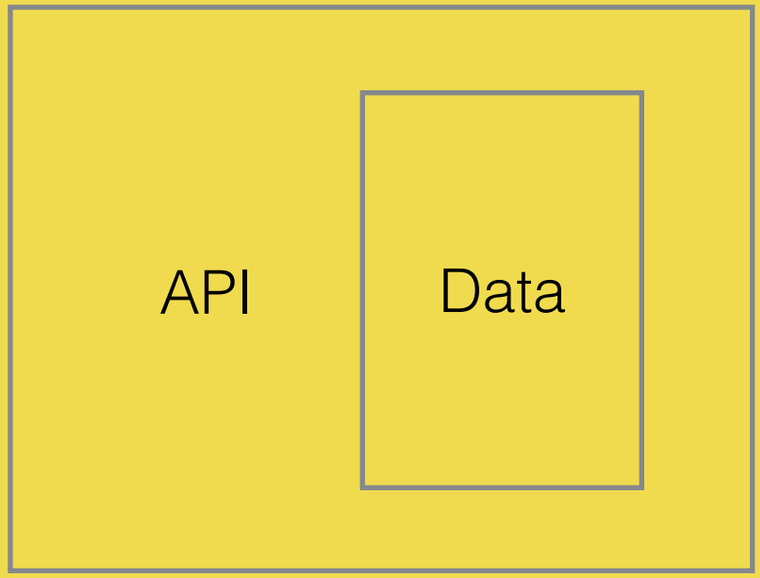
Authentication

With JSON Web Tokens

Mats Julian Olsen, @mewwts
23.05.2015

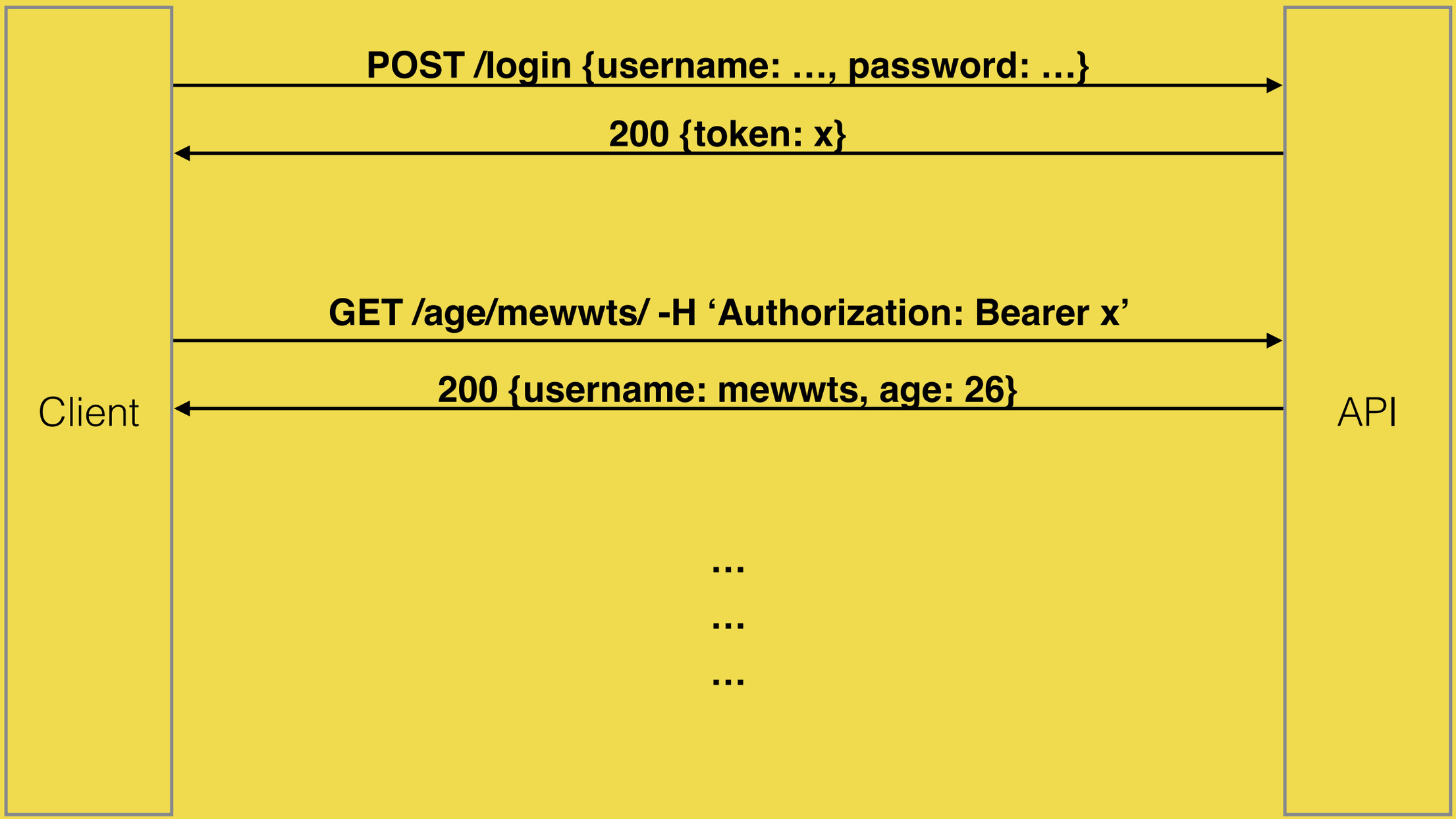
Authentication

- REST APIs exist to encapsulate data.
- How to make sure that only authorized users get the data?



Token Based Authentication

- Users log in with e.g. username and password.
- Client gets a string in return (a token).
- Client puts the token in the “Authorization” header in all subsequent requests.


POST /login {username: …, password: …}
200 {token: x}
GET /age/mewwts/ -H ‘Authorization: Bearer x’
200 {username: mewwts, age: 26}
Client
API
…
…
…

Tokens

- **Before**:
  - Randomly generated string used as key in a key-value store.
  - Upon request one would look up in a DB the properties for that token.
- **Now**:
  - JSON Web Tokens (JWTs) encode information such as username, scope and expiry time *in* the token.
  - Upon request the token is verified, and no DB interaction is necessary.

JSON Web Tokens

Header

Claims

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3M
iOiJtYXRzIiwiYWRtaW4iOnRydWUsImlhdCI6MTQz
MjE5Mzg1MywiZXhwIjoxNDMyMTkzOTEzfQ.q8slu
QbFJ0Z9dZ-MAGdbTUdAUjFkW6P6rmjGI0wOy6w

Signature (JWS)

JSON Web Tokens

```
var header = {
  “typ”: “JWT”,
  “alg”: "HS256"
}
```

```
var claims = {
 “iss”: "mats",
 “admin”: true,
 “iat”: 1432193853,
 “exp”: 1432193913
}
```

```
var signature = HMACSHA256(
 base64UrlEncode(header) + ‘.’ +
 base64UrlEncode(claims),
 “mySecret”
)
```

JSON Web Tokens

- Usually include **expiration time** and user information.
- JWTs are **signed** to ensure that they aren't tampered with.
- If no one knows the secret, you can trust the information in the token.
- No need to lookup user information in DB.
- Can encrypt the token if you want to store sensitive information.

How to implement authentication in your API

With a database and Passport.

Code is available at github.com/mewwts/auth-examples

What you need

A. Function that retrieves a user object from a database of your choice.

B. Passport “local strategy” to validate login attempts.

C. Function that can create a token.

D. Passport “bearer strategy” to validate tokens.

Passport

- Uses strategies to authenticate requests.
- We must give the strategy a function it will use to verify the requests.
- You could probably define your own middleware that does this

A. Get user from DB

```
function find(db, username, callback) {
  return db.get(
        'SELECT * FROM users WHERE username=?',
        username,
        callback
    );
}
```

B. Local Strategy

```
function createLocalStrategy(db) {
  return new LocalStrategy(function (username, password, done) {
    db.find(username, function (err, dbUser) {
      if (err) { return done(err); }
      if (!dbUser) { return done(null, false); }
      bcrypt.hash(password, dbUser.salt, null, function (err, res) {
        if (err) { return done(err); }
        if (res !== dbUser.password) { return done(null, false); }
        return done(null, {username: dbUser.username});
      });
    });
  });
}
```

C. Create a JWT

```
function issue(jwt_secret, jwt_expiry) {
  var claims = {iss: req.user.username, admin: true};
  return jwt.sign(
    obj,
    jwt_secret,
    {expiresInMinutes: jwt_expiry}
  );
}
```

D. Bearer Strategy

```
function createBearerStrategy(jwt_secret) {
  return new BearerStrategy(function (token, done) {
    jwt.verify(token, jwt_secret, function (err, decoded) {
      if (err) { return done(err); }
      return done(null, {username: decoded.iss});
    });
  });
}
```

Demo Outline

1. Client make a request to /login with a body containing the properties username and password.

2. Get a token back

3. Client makes subsequent requests with the token set in the ‘Authorization’ header.

4. Access a restricted endpoint

Thanks.

github.com/mewwts{/auth-examples}
@mewwts