ECS 162
Web Programming

5/24

Last time

- What does it mean to be logged in?
  - Browser has a session cookie which she attaches to every HTTP request
  - Server has a record of the session, with a timeout
  - Server pipeline has stages for recognizing still-valid session cookies and connecting each session with its user, and the user’s data
  - Later server pipeline stages might use user data (eg. to get right cards out of database), and/or require a valid session and user to work at all.

Login process

- Today - How do we start up a session?
- Steps:

1. Authenticate – user gives password to Google
2. Server gets user profile information from Google
3. New user? If so, enter in user database
4. Set up Server session info

Authentication

- Two options: local or third party.
- Third party (eg. login with Google, login with Facebook, login with github…) is usually better for simple apps like ours.
- Why?

Authentication

- Two options: local or third party.
- Third party (eg. login with Google, login with Facebook, login with github…) is usually better for simple apps like ours.
- Why?

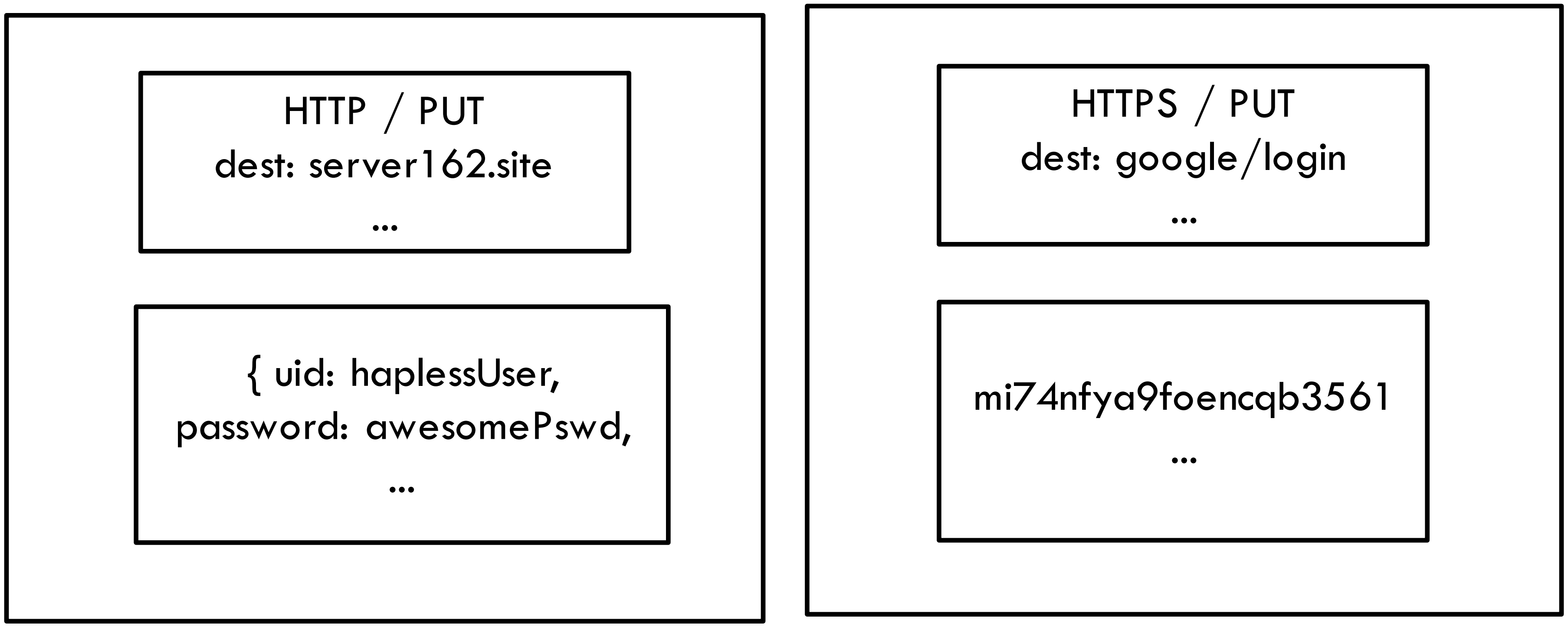
1. We don't want to be responsible for passwords or other information that might get compromised.

- Oauth2 is an open standard for authorization through third parties. Controversial, not particularly secure, what everybody uses….

No passwords in HTTP requests!

- Recall that an HTTP request is just a text file sent over the internet.
- Anyone snooping anywhere it passes through can read it.
- So it would be a disaster for the user to put his password (even a password for Lango) into an HTTP request.
- Instead, we need HTTPS (secure HTTP). In an HTTPS request/response, the body of the request and the body of the response are encrypted.

HTTP vs HTTPS



Never do this!

Password is publicly visible.

Perfectly fine.

Password in encrypted in body.

HTTPS

- So, another reasons we don’t do local login:

2. Our site only supports HTTP, so we can’t do local logins. The user could never send us a password.

- Instead, they will send their password directly to Google, in an HTTPS message.
- We redirect user to Google, they login, Google redirects them back to us.

Permission for Lango from Google

- We need to register Lango for login with Google.
- They already know about Lango since it uses the Translation API.
- We can add login service to that existing project.
- If we wanted user data they consider private, they would want to know more about the app, but since we will ask for minimal profile data, this is instantaneous (if complicated...directions on Assn 6 page).

Login Data object

```
const googleLoginData = {
    clientID: ’[server’s id key]',
    clientSecret: ’[another secret key]',
    callbackURL: '/auth/redirect'
};
```

- Lango is the client.
- The two keys are the credentials we got from Google, identifying our server.
- The callback URL is where we want Google to return to on the server after successful login.

Let passport store login data

```
passport.use(
    new GoogleStrategy(googleLoginData,
                                        gotProfile) );
```

- On start up, we store this data with Passport
- gotProfile is a callback function we'll need later in the process; might as well store it now.

Beginning of login process

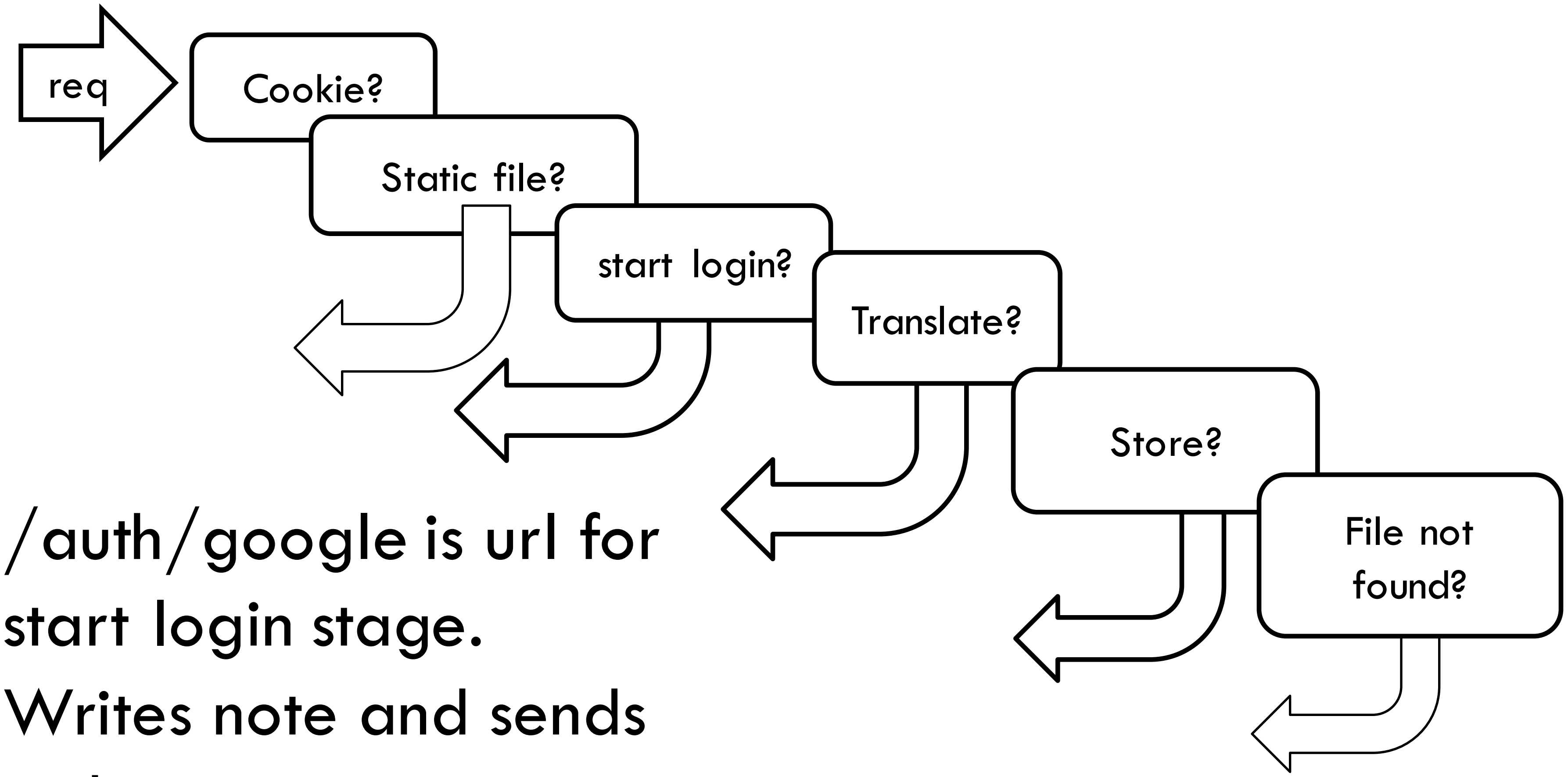
```
app.get('/auth/google',
  passport.authenticate('google',{ scope: ['profile'] }) );
```

- Pipeline stage that responds to URL:

http://server162.site:[port]/auth/google

- Kicks off login process

Stage to pipeline to start login



/auth/google is url for start login stage. Writes note and sends redirect response to browser.

Beginning of login process

```
app.get('/auth/google',
  passport.authenticate('google',{ scope: ['profile'] }) );
```

- passport.authenticate prepares a URL for Browser to send to Google, and sends it in a redirect response.

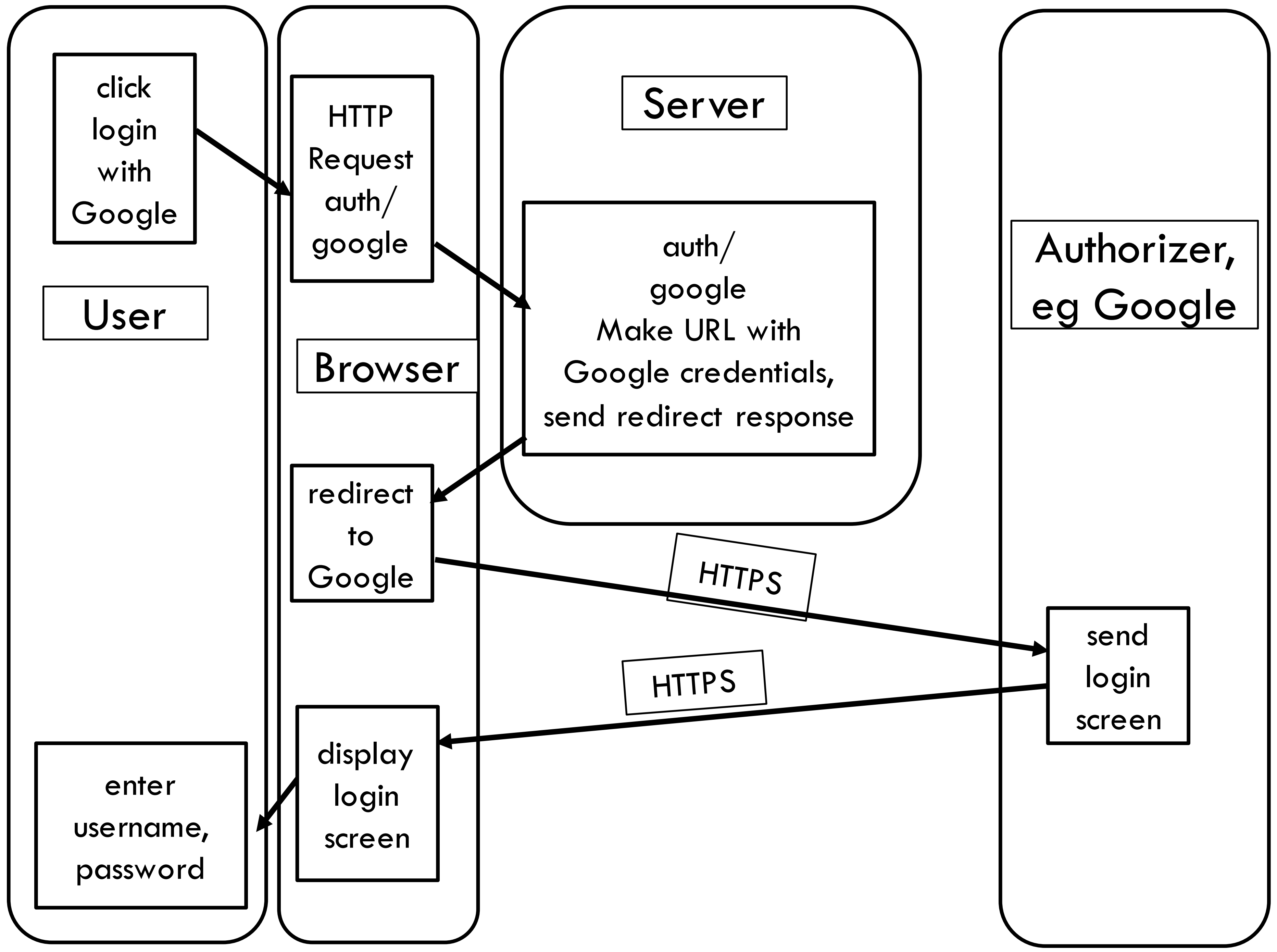
“note” for Google

- URL is like a note that Lango gives to the Browser, to show to Google:

“This user wants to log in to Lango with Google. We’re OK with that. We’d like to get their profile data. If they succeed in logging in, please send them back to us, at server162.site:[port]/auth/redirect”

The actual HTTP response

Return code: 302
Redirect address:
https://accounts.google.com/o/oauth2/v2/auth?response_type=code&redirect_uri=http://server162.site:30057/auth/redirect&scope=profile&client_id=[actual cliend id]


click login with Google
User
HTTP Request auth/ google
Browser
redirect to Google
display login screen
enter username, password
Server
auth/ google Make URL with Google credentials, send redirect response
HTTPS
HTTPS
Authorizer, eg Google
send login screen

Why all this redirect business?

- Why doesn’t Lango contact Google directly and tell them that the user wants to log in?

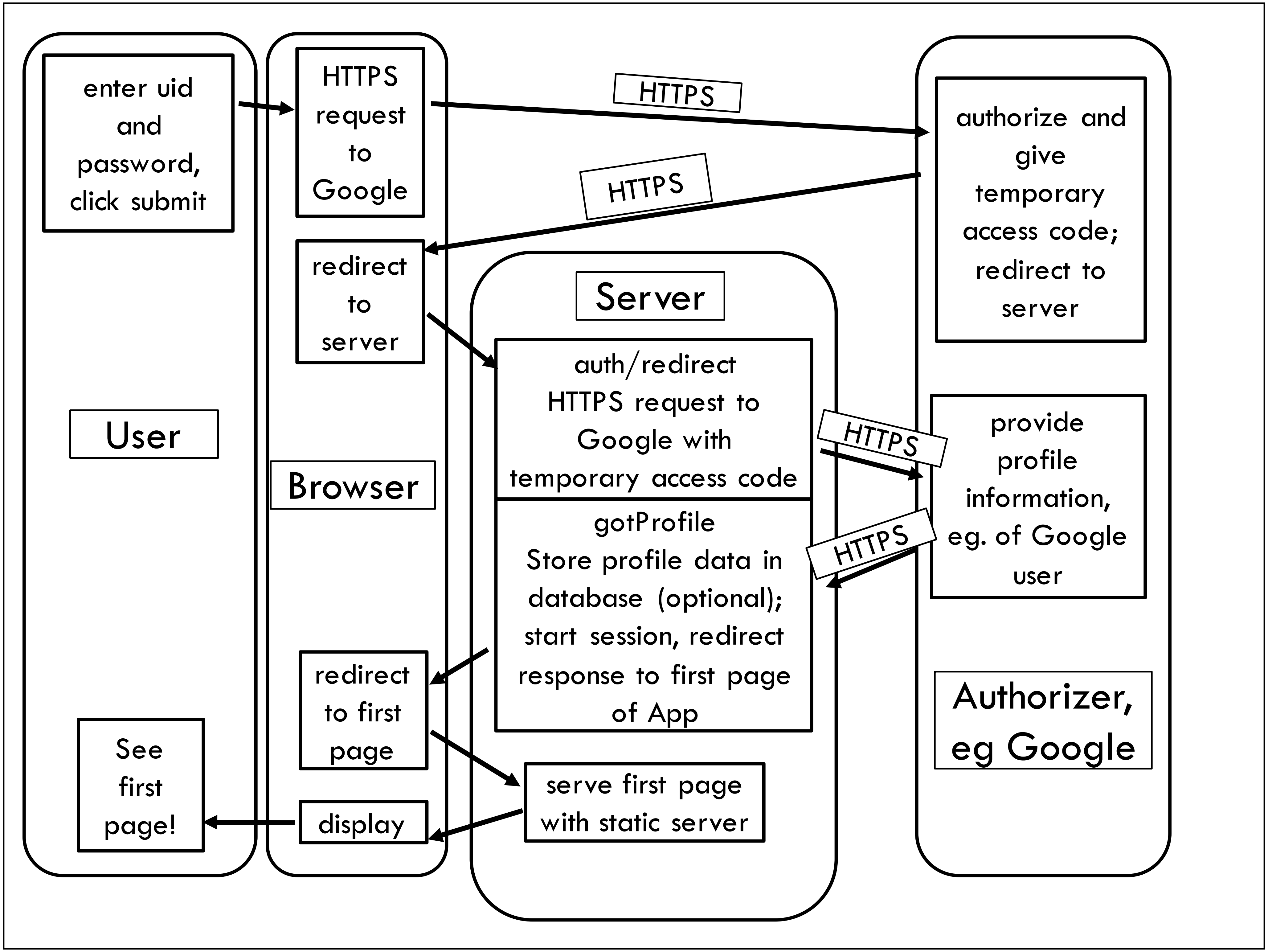
Why all this redirect business?

- Why doesn't Lango contact Google directly and tell them that the user wants to log in?

Google can't contact the Browser. Google is the server in the login transaction, and has to be contacted by the browser.

Instead, we stay in the simple request-response model.

  - *The Browser sent an HTTP request to the Server. The Server responds to that request (with a redirect).*
  - *The Browser sends an HTTPS request to Google, which it responds to in a perfectly normal way.*


enter uid and password, click submit
HTTPS request to Google
HTTPS
authorize and give temporary access code; redirect to server
HTTPS
redirect to server
Server
auth/redirect HTTPS request to Google with temporary access code
HTTPS
provide profile information, eg. of Google user
HTTPS
gotProfile Store profile data in database (optional); start session, redirect response to first page of App
User
Browser
redirect to first page
serve first page with static server
See first page!
display
Authorizer, eg Google

HTTP redirect to auth/redirect

Return code: 302

Redirect address:

http://server162.site:30057/auth/redirect?code=[temporary_profile_access_code&scope=profile+https://www.googleapis.com/auth/userinfo.profile

- It's redirecting to Server, but telling it how to request profile information from Googe for user.

Pipeline stage for auth/redirect

```
app.get('/auth/redirect’,
        passport.authenticate('google'),
        function (req, res) {
            console.log('Logged in and using cookies!')
            res.redirect('/user/hello.html');
        });
```

- Has two handler functions, done one after the other.
- res.redirect in the second one sends off the response object, to the app page user should see after login.

Action behind the scenes...

passport.authenticate('google')

- This first handler for auth/redirect sends off an HTTPS request to Google to get the user’s profile data, using the temporary code that Google sent us.
- The callback for this request is the gotProfile callback we defined way back when.
- After this we can do database and session set up!

What is profile information?

```
{ id: '106646915971583712375',
  displayName: 'Nina Amenta',
  name: { familyName: 'Amenta',
             givenName: 'Nina' },
  photos: [ { value:
'https://lh6.googleusercontent.com/-
Jk9n2DtloaA/AAAAAAAAAAI/AAAAAAAAA0g/81n7
GPscyYk/photo.jpg' } ] }
```

- What’s that picture?

Photo

How do we feel about this?

- It was pretty easy for us to get this info.
- Do we think of all this as public?

Boilerplate

- Most of this code is boilerplate, that is, stuff we copy, paste and modify.
- Database and session set-up has calls functions that let us customize
  - What do we put in our database for each user,
  - What goes into the req.user field for late pipeline stages to use.
- More on that next time.