# Sessions

# Session management

- Client sends multiple requests to server
- Save some “state” information
  - logged in
  - choice of background colour
  - …
- Server customizes responses based on client session information

Storage:

- Client-side session: completely stored in cookie
- Server-side session: stored on server, looked up from cookie

# Cookies

- Set by server with Set-Cookie header
- Must be returned by client with each request
- Can be used to store information:
  - theme, background colour, font size: simple no security issues
  - user permissions, username: can also be set in cookie
    - must not be possible to alter!

# Example: Flask

```
from flask import session

# Set the secret key to some random bytes. Keep this really secret!
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

@app.route('/')
def index():
    if 'username' in session:
        return f'Logged in as {session["username"]}'
    return 'You are not logged in'
```

# Example: Flask

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        session['username'] = request.form['username']
        return redirect(url_for('index'))
    return '''
        <form method="post">
            <p><input type=text name=username>
            <p><input type=submit value=Login>
        </form>
    '''

@app.route('/logout')
def logout():
    # remove the username from the session if it's there
    session.pop('username', None)
    return redirect(url_for('index'))
```

# Security issues

- Can user modify Cookie?
  - Can set any username
- If someone else gets Cookie, can they log in as user?
  - Timeout
  - Source IP
- Cross-site requests
  - Attacker can create page to automatically submit request to another site
  - If user is logged in on other site when they visit attack page, will automatically invoke action
  - Verify on server that request came from legitimate start point

# Server-side information

- Maintain client information at server
- Cookie only provides minimal lookup information
- Not easy to alter
- Requires persistent storage at server
- Multiple backends possible
  - File storage
  - Database
  - Redis, other caching key-value stores

# Enforce authentication

- Some parts of site must be ***protected***
- How?
  - Enforce existence of specific token for access to those views
- Views:
  - determined by controller
- Protect access to controller!
  - Flask controller - Python function
  - Protect function - add wrapper around it to check auth status
    - Decorator!

# Example - flask_login

```
from flask_login import login_required, current_user
...
@main.route('/profile')
@login_required
def profile():
    return render_template('profile.html', name=current_user.name)
```

# Example - flask_login

```
from flask_login import login_user, logout_user, login_required
...
@auth.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('main.index'))
```

# Transmitted data security

- Assume connection can be “tapped”
- Attacker should not be able to read data
- HTTP GET URLs not good:
  - logged on firewalls, proxies etc
- HTTP POST, Cookies etc:
  - if wire can be made safe, then good enough

How to make the wire safe?