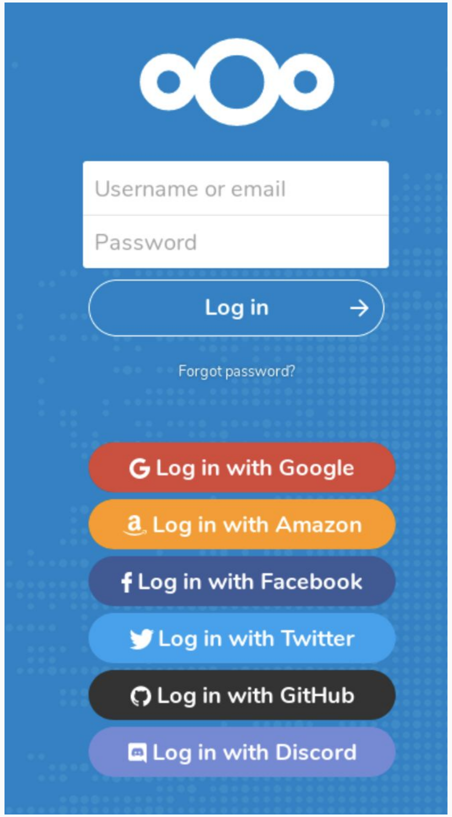
CSC302 - OAuth

Jonathan Libby, Nancy Zhao, David Chen

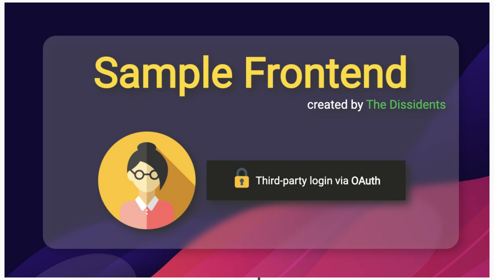
General Product Concept

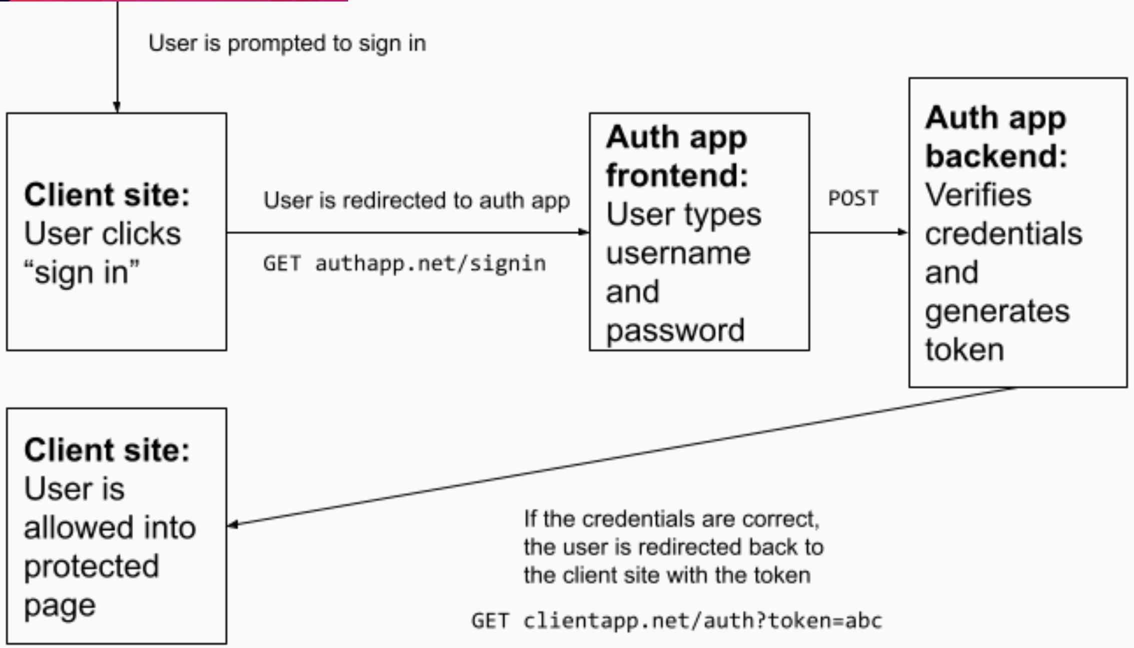
Simple implementation of **OAuth:** a website that other websites can use to authenticate users



From nextcloud.com

What and Why?

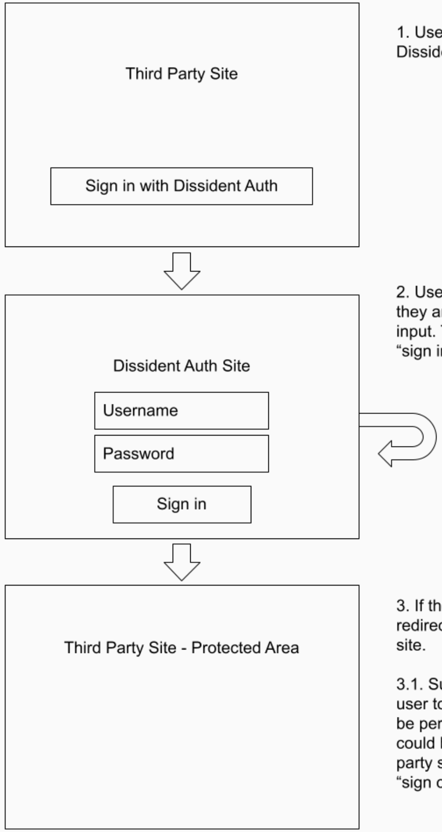



User is prompted to sign in
Client site: User clicks “sign in”
User is redirected to auth app
GET authapp.net/signin
Auth app frontend: User types username and password
POST
Auth app backend: Verifies credentials and generates token
If the credentials are correct, the user is redirected back to the client site with the token
GET clientapp.net/auth?token=abc
Client site: User is allowed into protected page

Functionality

Demo

CSC302 The Dissidents - OAuth Service Acceptance Criteria Storyboard


Third Party Site
Sign in with Dissident Auth
Dissident Auth Site
Username
Password
Sign in
Third Party Site - Protected Area

1. User visits third party site and clicks “sign in with Dissident Auth” button

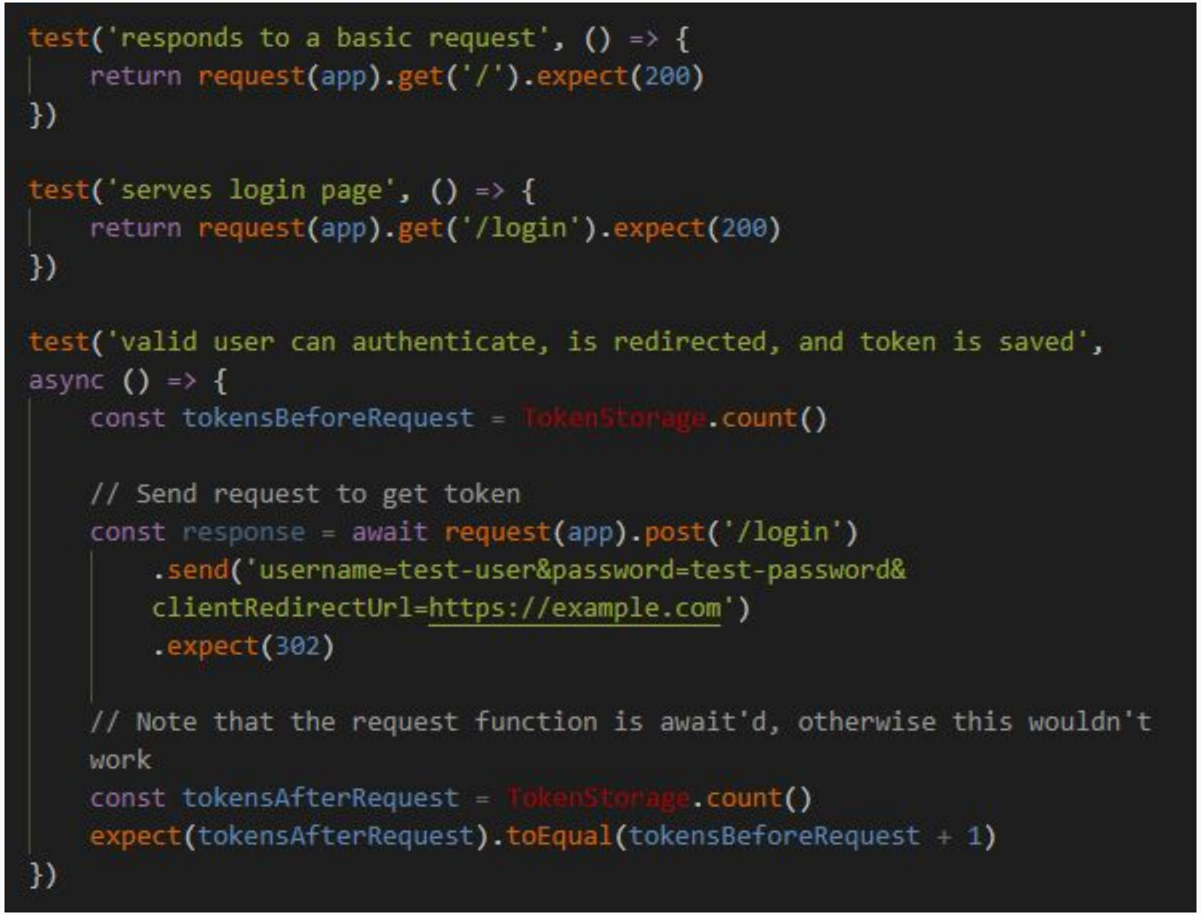
2. User is redirected to Dissident Auth site, where they are presented with a username and password input. The user enters their credentials and clicks “sign in”.

2.1. If the user’s credentials are incorrect, a nonspecific error message is presented on the page and the inputs are cleared (e.g. “Username **or** password incorrect”). The user can continue to attempt to authenticate until successful. A stretch goal could be rate limiting these auth requests, potentially with exponential backoff.

3. If the user’s credentials are correct, they are redirected to the protected area of the third party site.

3.1. Subsequent page refreshes will redirect the user to the signin page. Authentication state will not be persisted on the third party site. A stretch goal could be storing the authentication state in the third party site session to persist it until the user clicks “sign out”.

Acceptance Criteria

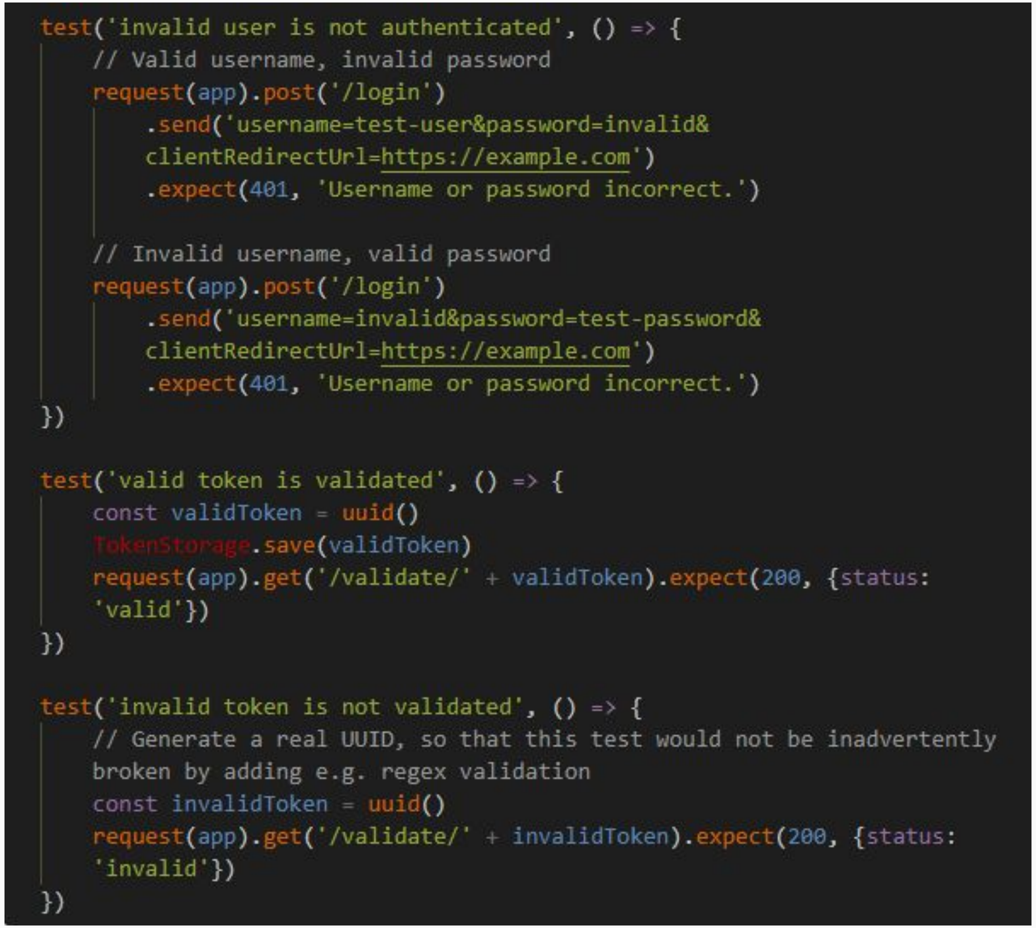

```
test('responds to a basic request', () => {
    return request(app).get('/').expect(200)
})

test('serves login page', () => {
    return request(app).get('/login').expect(200)
})

test('valid user can authenticate, is redirected, and token is saved',
async () => {
    const tokensBeforeRequest = TokenStorage.count()

    // Send request to get token
    const response = await request(app).post('/login')
        .send('username=test-user&password=test-password&
        clientRedirectUrl=https://example.com')
        .expect(302)

    // Note that the request function is await'd, otherwise this wouldn't
    work
    const tokensAfterRequest = TokenStorage.count()
    expect(tokensAfterRequest).toEqual(tokensBeforeRequest + 1)
})
```



```
test('invalid user is not authenticated', () => {
    // Valid username, invalid password
    request(app).post('/login')
        .send('username=test-user&password=invalid&
        clientRedirectUrl=https://example.com')
        .expect(401, 'Username or password incorrect.')

    // Invalid username, valid password
    request(app).post('/login')
        .send('username=invalid&password=test-password&
        clientRedirectUrl=https://example.com')
        .expect(401, 'Username or password incorrect.')
})

test('valid token is validated', () => {
    const validToken = uuid()
    TokenStorage.save(validToken)
    request(app).get('/validate/' + validToken).expect(200, {status:
    'valid'})
})

test('invalid token is not validated', () => {
    // Generate a real UUID, so that this test would not be inadvertently
    broken by adding e.g. regex validation
    const invalidToken = uuid()
    request(app).get('/validate/' + invalidToken).expect(200, {status:
    'invalid'})
})
```

Automated Tests

Team Concept

- What was our philosophy around teamwork and the division of tasks?

Milestones and Task Creation

Team Milestones

1. Set up environment, install instructions
2. Make and grant auth requests
3. Verify auth, generate and send token
4. Validate token, redirect to sample page

How did we determine which functionality to include in each milestone?

- Iterating on client and server, and documentation

How did we determine who to assign tasks to?

- Subtasks by client, server, deployment, storage
- Team effort on documenting progress

Technical Decision Making

- How did we decide which technologies to use for each feature?
  - Mostly familiarity, for speed and ease of development

- **Client:** Simple HTML/CSS/JS frontend
- **Server:** Express, Node.JS

Adjustments Made Along the Way

- Readability is crucial
- Detailed meeting notes saves lives
- Not everything can be meticulously planned
- Sometimes tasks will be delayed
    - It is important to learn how to prioritize
    - The difference between deadlines and guidelines

Conclusion

Thank you!