

# PrairieLearn Okta Integration

Senior Design May 2024 - 33

Mitch Hudson  
Tyler Weberski  
Chris Costa  
Andrew Winters  
Carter Murawski  
Matt Graham

# Table of Contents

|   |          |
|---|----------|
| <b>Table of Contents</b>                  | <b>2</b> |
| <b>Introduction</b>                       | <b>3</b> |
| <b>GitHub Repository and Installation</b> | <b>3</b> |
| docker-compose-production.yml             | 3        |
| <b>Code Changes</b>                       | <b>4</b> |
| Dependencies                              | 4        |
| package.json                              | 5        |
| yarn.lock                                 | 5        |
| OIDC Authentication Flow                  | 6        |
| server.js                                 | 7        |
| authLoginOid.js                           | 8        |
| authCallbackOid.ts                        | 9        |
| OIDC Login Button                         | 9        |
| CSS Changes                               | 10       |
| HTML Changes                              | 11       |
| Enabling OIDC in PrairieLearn             | 12       |
| authLogin.html.ts                         | 13       |
| authLogin.ts                              | 14       |
| institution.ts                            | 14       |
| 20231114133024_oid_providers__add.sql     | 15       |
| Configuration                             | 16       |
| OIDC Server Configuration                 | 16       |
| User Info Configuration                   | 17       |
| Login Link Configuration                  | 17       |
| Logo Configuration                        | 17       |

## Introduction

This document outlines the steps taken to allow Okta SSO sign-ins for PrairieLearn. This works via a generic OpenID Connect (OIDC) authentication system that is set up for Okta by default. The system can be configured to allow any OIDC / OAuth2 authentication method. These configurations will be outlined in more detail in the [Configuration](#) section.

## GitHub Repository and Installation

All of the code changes are kept in the [GitHub repo](#). To use the code, you can follow the instructions on PrairieLearn's readthedocs page for installing natively and running with local code. In production, we had to frankenstein our docker compose file using the instructions found [here](#) and [here](#). In the end, our docker-compose-production.yml file looked like normal with an added volume for the PrairieLearn codebase going to /PrairieLearn instead of the config file. This caused some compilation issues, so we also had to follow the instructions [here](#) to compile the codebase locally before it would run. Finally, we followed the instructions [here](#) to get the autograders to work.

*NOTE: docker-compose-production.yml already has the autograder jobs environment set up, you just need to make the folder it links to and ensure it runs in the right user.*

### docker-compose-production.yml

```
version: '3.8'
services:
  pl:
    restart: always
    image: prairielearn/prairielearn:latest
    ports:
      - 3000:3000
    volumes:
      - postgres:/var/postgres
      - /var/run/docker.sock:/var/run/docker.sock
      - ${HOME}/pl_ag_jobs:/jobs
      - /home/pl/PL/PrairieLearn:/PrairieLearn
```

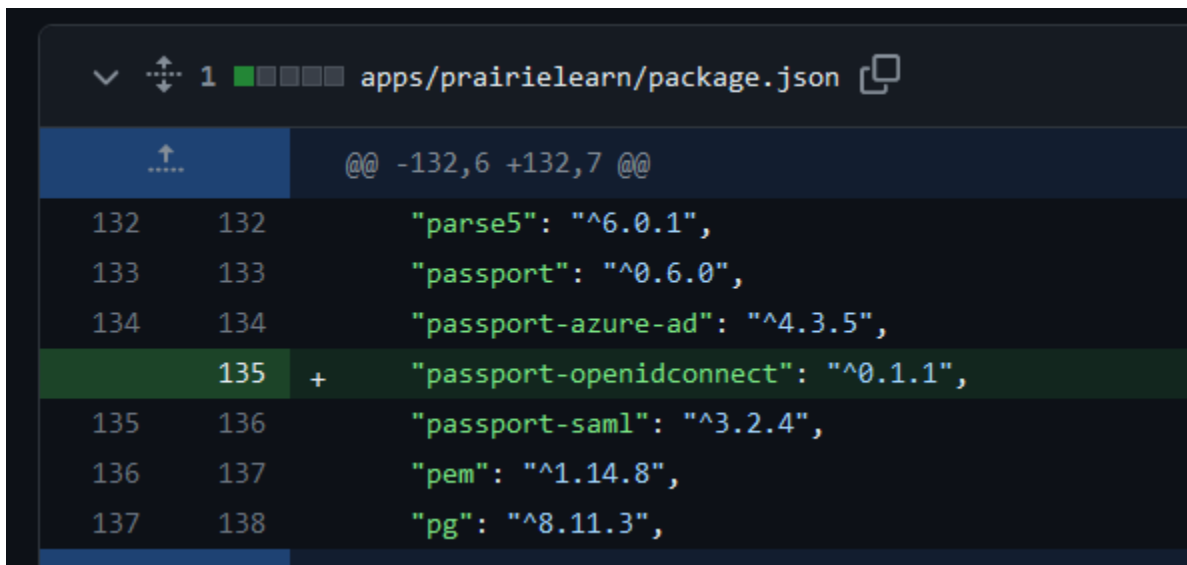
```
container_name: pl
environment:
  - HOST_JOBS_DIR=${HOME}/pl_ag_jobs
  - NODE_ENV=production

volumes:
  postgres:
```

## Code Changes

There are several key changes made to allow this authentication method to work. Mainly, we needed to add a new dependency for OIDC, update the login page to have buttons for OIDC logins, add OIDC as a trusted login method, and write code for handling the OIDC authentication flow. Finally, we added configuration variables so the authentication can be configured like any other authentication method, and added a migration to the SQL database to add OIDC as a login provider.

## Dependencies



```
apps/prairielearn/package.json
@@ -132,6 +132,7 @@
132 132   "parse5": "^6.0.1",
133 133   "passport": "^0.6.0",
134 134   "passport-azure-ad": "^4.3.5",
135 135 +  "passport-openidconnect": "^0.1.1",
135 136   "passport-saml": "^3.2.4",
136 137   "pem": "^1.14.8",
137 138   "pg": "^8.11.3",
```

*package.json changes*

## package.json

The file `apps/prairielearn/package.json` needs to be updated to include `passport-openidconnect`, with at least version 0.1.1. Passport is the library used by PrairieLearn to handle authentication, and it has a sub-library module for specifically OIDC.

```
13 yarn.lock
@@ -3286,6 +3286,7 @@ __metadata:
3286 3286   parse5: ^6.0.1
3287 3287   passport: ^0.6.0
3288 3288   passport-azure-ad: ^4.3.5
3289 + passport-openidconnect: ^0.1.1
3289 3290   passport-saml: ^3.2.4
3290 3291   pem: ^1.14.8
3291 3292   pg: ^8.11.3
@@ -12622,7 +12623,7 @@ __metadata:
12622 12623   languageName: node
12623 12624   linkType: hard
12624 12625
12625 - "oauth@npm:0.9.15":
12626 + "oauth@npm:0.9.15, oauth@npm:0.9.x":
12626 12627   version: 0.9.15
12627 12628   resolution: "oauth@npm:0.9.15"
12628 12629   checksum:
957c0d8d85300398dcb0e293953650c0fc3facc795bee8228238414f19f59cef5fd4ee8d17a972c142924c10c5f6ec50ef80f77f4a6cc6e3c98f9d22c027801c
@@ -13023,6 +13024,16 @@ __metadata:
13023 13024   languageName: node
13024 13025   linkType: hard
13025 13026
13027 + "passport-openidconnect@npm:^0.1.1":
13028 +   version: 0.1.1
13029 +   resolution: "passport-openidconnect@npm:0.1.1"
13030 +   dependencies:
13031 +     oauth: 0.9.x
13032 +     passport-strategy: 1.x.x
13033 +   checksum:
75fba69b7d5d36b0957cda611a456896986030328c47c04e9603ed26ee4d50f075a055f9357429fa31fc9c64693f1016b2f6e65153df729ac1d5708859c6d659
13034 +   languageName: node
13035 +   linkType: hard
13036 +
13026 13037 "passport-saml@npm:^3.2.4":
13027 13038   version: 3.2.4
13028 13039   resolution: "passport-saml@npm:3.2.4"
```

*yarn.lock changes*

## yarn.lock

Yarn.lock is a file that contains compilation and dependency information for yarn. All we added here was `passport-openidconnect` and `oauth`. The first section adds `passport-openidconnect` to the dependencies and the other part adds the definition for where the packages are found, as well as adds a new version to `oauth`.

## OIDC Authentication Flow

The OIDC authentication flow is handled through the use of Passport OpenID Connect. To set this up we tell passport to use a new strategy from passport-openidconnect in `apps/prairielearn/src/server.js`. Then, two files are added to hold the code for the endpoints needed by OIDC: `apps/prairielearn/src/pages/authLoginOid/authLoginOid.js` and `apps/prairielearn/src/pages/authCallbackOid/authCallbackOid.ts`.

```
apps/prairielearn/src/server.js
@@ -496,6 +496,11 @@ module.exports.initExpress = function () {
 496 496     app.use('/pl/auth/institution/:institution_id/saml', require('./ee/auth/saml/router').default);
 497 497     }
 498 498
 499 +   if (config.hasOid) {
 500 +     app.use('/pl/oidlogin', require('./pages/authLoginOid/authLoginOid'));
 501 +     app.use('/pl/oidcallback', require('./pages/authCallbackOid/authCallbackOid').default);
 502 +   }
 503 +
 499 504     app.use('/pl/lti', require('./pages/authCallbackLti/authCallbackLti'));
 500 505     app.use('/pl/login', require('./pages/authLogin/authLogin').default);
 501 506     if (config.devMode) {
@@ -2127,6 +2132,23 @@ if (require.main === module && config.startServer) {
 2127 2132     passport.use(strategy);
 2128 2133     }
 2129 2134     },
 2135 +   async () => {
 2136 +     if (config.hasOid) {
 2137 +       const { Strategy } = require('passport-openidconnect');
 2138 +       passport.use('oidconnect', new Strategy({
 2139 +         issuer: config.oidIssuer,
 2140 +         authorizationURL: config.oidAuthUrl,
 2141 +         tokenURL: config.oidTokenUrl,
 2142 +         userInfoURL: config.oidUserInfoUrl,
 2143 +         clientId: config.oidClientId,
 2144 +         clientSecret: config.oidClientSecret,
 2145 +         callbackURL: config.oidRedirectUrl,
 2146 +         scope: "openid profile"
 2147 +       }, (issuer, profile, cb) => {
 2148 +         cb(null, profile);
 2149 +       })))
 2150 +     }
 2151 +   },
 2130 2152     async function () {
 2131 2153       const pgConfig = {
 2132 2154         user: config.postgresqlUser,
```

### server.js changes

#### server.js

This file has two changes. First, we add the endpoints defined by authLoginOid.js and authCallbackOid.ts after checking that OIDC is enabled in the config. Second, in the stack of async () calls, we insert one that tells Passport to use a strategy defined by passport-openidconnect and the configuration variables defined in Configuration. This strategy is then labeled as "oidconnect" for later use.

```
apps/prairielearn/src/pages/authLoginOid/authLoginOid.js
...
1 + // @ts-check
2 + const express = require('express');
3 + const router = express.Router();
4 +
5 + const passport = require('passport');
6 +
7 + const { config } = require('../../lib/config');
8 +
9 + router.get('/',
10 +   function (req, res, next) {
11 +     if (!config.hasOid) {
12 +       return next(new Error(`OID login is not enabled`));
13 +     }
14 +
15 +     passport.authenticate(
16 +       'oidconnect',
17 +       { failureRedirect: '/pl', session: false }
18 +     )(req, res, next);
19 +   }, function (req, res) {
20 +     res.redirect('/pl');
21 +   }
22 + );
23 +
24 + module.exports = router;
```

*authLoginOid.js changes*

### authLoginOid.js

This file is used to define the endpoint using Passport. To do this, we use the NodeJS Express router made by PrairieLearn to add an endpoint. The endpoint is handled with a function that first checks if OIDC is enabled in the config, then uses passport to authenticate with the oidconnect strategy defined earlier, giving it failure and success redirects.



```

apps/prairielearn/src/pages/authCallbackOid/authCallbackOid.ts
... @@ -0,0 +1,51 @@
1 + // @ts-check
2 + import express = require('express');
3 + import asyncHandler = require('express-async-handler');
4 + import passport = require('passport');
5 + import { config } from '../../lib/config';
6 +
7 + import * as authnLib from '../../lib/authn';
8 +
9 + const router = express.Router();
10 +
11 + function authenticate(req, res): Promise<any> {
12 +   return new Promise((resolve, reject) => {
13 +     passport.authenticate(
14 +       'oidconnect',
15 +       {
16 +         failureRedirect: '/pl',
17 +         session: false
18 +       },
19 +       (err, user) => {
20 +         if (err) {
21 +           reject(err);
22 +         } else if (!user) {
23 +           reject(new Error("Login failed"));
24 +         } else {
25 +           resolve(user);
26 +         }
27 +       }
28 +     )(req, res);
29 +   });
30 + };
31 +
32 + router.get(
33 +   '/',
34 +   asyncHandler(async (req, res, _next) => {
35 +     const user = await authenticate(req, res);
36 +
37 +     const authnParams = {
38 +       uid: user[config.oidUidKey ?? ""],
39 +       name: user[config.oidNameKey ?? ""],
40 +       uin: user[config.oidUinKey ?? ""],
41 +       provider: 'OID',
42 +     };
43 +
44 +     await authnLib.loadUser(req, res, authnParams, {
45 +       redirect: true,
46 +       pl_authn_cookie: true,
47 +     });
48 +   }),
49 + );
50 +
51 + export default router;

```

### authCallbackOid.ts

This Typescript file handles the callback from the OIDC server, and logs the user into the user database. To do this, an authenticate function is created that uses Passport to continue the authentication flow, and either return a fail state or the user that logged in. This function is called in the router function, and the user is parsed for its authentication parameters, which are then loaded into the database.

## OIDC Login Button

The login button is configured using the configuration JSON file, and is handled in `apps/prairielearn/src/pages/authLogin/authLogin.html.ts`. The file is edited to show the button in the login page through HTML, CSS, and Typescript.

```
apps/prairielearn/src/pages/authLogin/authLogin.html.ts
@@ -92,6 +92,24 @@ function LoginPageContainer({
92 92 border-color: ${config.shibLinkColors.active.border};
93 93 color: ${config.shibLinkColors.active.text};
94 94 }
95 + .btn-oid {
96 + background-color: ${config.oidLinkColors.normal.background};
97 + border-color: ${config.oidLinkColors.normal.border};
98 + color: ${config.oidLinkColors.normal.text};
99 + }
100 + .btn-oid:hover {
101 + background-color: ${config.oidLinkColors.hover.background};
102 + border-color: ${config.oidLinkColors.hover.border};
103 + color: ${config.oidLinkColors.hover.text};
104 + }
105 + .btn-oid:focus {
106 + box-shadow: 0 0 0 0.2rem ${config.oidLinkColors.focus.shadow};
107 + }
108 + .btn-oid:active {
109 + background-color: ${config.oidLinkColors.active.background};
110 + border-color: ${config.oidLinkColors.active.border};
111 + color: ${config.oidLinkColors.active.text};
112 + }
95 113 .institution-header {
96 114 overflow: hidden;
97 115 text-align: center;
@@ -153,6 +171,17 @@ function GoogleLoginButton() {
```

*Adding CSS for the login button*

## CSS Changes

There is a large section of CSS code for the login page at the beginning of the file, and here we add some CSS that reads from the configuration JSON to get all of the color data.

```
.....  
↓  
↑  
.....  
@@ -153,6 +171,17 @@ function GoogleLoginButton() {  
153 171  `;  
154 172  }  
155 173  
174 + function OpenIDConnectLoginButton() {  
175 +   return html`  
176 +     <a class="btn btn-oid d-block position-relative" href="/pl/oidlogin" role="button">  
177 +       ${config.oidLinkLogo != null  
178 +         ? html`  
179 +         : html`<span class="social-icon"></span>`}  
180 +       <span class="font-weight-bold">${config.oidLinkText}</span>  
181 +     </a>  
182 +   `;  
183 + }  
184 +  
156 185  function MicrosoftLoginButton() {  
157 186  return html`  
158 187  <a class="btn btn-dark d-block position-relative" href="/pl/azure_login" role="button">  
.....  
↓  
↑  
.....  
@@ -195,6 +224,7 @@ export function AuthLogin({
```

*Adding HTML for the login button*

### HTML Changes

After the CSS changes, there is a section of functions that return the HTML for each of the login buttons. Here, we insert a function of our own that is modeled after the Shibboleth login button, with minor changes for the configuration variables.

## Enabling OIDC in PrairieLearn

PrairieLearn has several parts to the authentication system. Specifically, PrairieLearn uses a database to store active and supported authentication providers. To get PrairieLearn to recognize OIDC as a login method, we need to update the database and code that checks the database. To do this, we edit the `apps/prairielearn/src/pages/authLogin/authLogin.html.ts` file again, as well as `apps/prairielearn/src/ee/lib/institution.ts`, `apps/prairielearn/src/pages/authLogin/authLogin.ts`, and add a migration for the database at `apps/prairielearn/src/migrations/20231114133024_oid_providers__add.sql`.

```

@@ -195,6 +224,7 @@ export function AuthLogin({
195 224     ${config.hasShib && !config.hideShibLogin ? ShibLoginButton() : ''}
196 225     ${config.hasOauth ? GoogleLoginButton() : ''}
197 226     ${config.hasAzure && isEnterprise() ? MicrosoftLoginButton() : ''}
227 +     ${config.hasOid ? OpenIDConnectLoginButton(): ''}
198 228     </div>
199 229     ${institutionAuthnProviders?.length
200 230     ? html`
@@ -232,7 +262,8 @@ export function AuthLoginUnsupportedProvider({
232 262     const supportsShib = supportedProviders.some((p) => p.name === 'Shibboleth');
233 263     const supportsGoogle = supportedProviders.some((p) => p.name === 'Google');
234 264     const supportsAzure = supportedProviders.some((p) => p.name === 'Azure');
235 -
265 +     const supportsOid = supportedProviders.some((p) => p.name === 'OID');
266 +
236 267     const defaultProvider = supportedProviders.find((p) => p.is_default === true);
237 268     const hasNonDefaultProviders = supportedProviders.find(
238 269     (p) => p.name !== 'LTI' && p.is_default === false,
@@ -246,6 +277,7 @@ export function AuthLoginUnsupportedProvider({
246 277     defaultProvider?.name !== 'Shibboleth';
247 278     const showGoogle = config.hasOauth && supportsGoogle && defaultProvider?.name !== 'Google';
248 279     const showAzure = config.hasAzure && supportsAzure && defaultProvider?.name !== 'Azure';
280 +     const showOid = config.hasOid && supportsOid && defaultProvider?.name !== 'OID';
249 281
250 282     let defaultProviderButton: HtmlValue = null;
251 283     switch (defaultProvider?.name) {
@@ -260,6 +292,10 @@ export function AuthLoginUnsupportedProvider({
260 292         break;
261 293         case 'Azure':
262 294             defaultProviderButton = MicrosoftLoginButton();
295 +         break;
296 +         case 'OID':
297 +             defaultProviderButton = OpenIDConnectLoginButton();
298 +         break;
263 299     }
264 300
265 301     return LoginPageContainer({
@@ -291,6 +327,7 @@ export function AuthLoginUnsupportedProvider({
291 327     showShib ? ShibLoginButton() : '',
292 328     showGoogle ? GoogleLoginButton() : '',
293 329     showAzure ? MicrosoftLoginButton() : '',
330 +     showOid ? OpenIDConnectLoginButton() : '',
294 331     ]}
295 332     </div>
296 333     `

```

*authLogin.html.ts changes*

**authLogin.html.ts**

There are several places in this file where support for authentication methods are checked and used, and we need to update all of them to include our OIDC login method. These are fairly simple, since all login methods follow the same patterns.

```
apps/prairielearn/src/pages/authLogin/authLogin.ts
@@ -86,6 +86,9 @@ router.get(
86 86      case 'Shibboleth':
87 87          url = '/pl/shibcallback';
88 88          break;
89 +      case 'OID':
90 +          url = '/pl/oidlogin';
91 +          break;
89 92      default:
90 93          return null;
91 94      }
```

*authLogin.ts changes*

### authLogin.ts

Here we need to add the case for the OIDC login, and set the url to the /pl/oidlogin endpoint. This is what allows the OIDC login button to redirect to the login endpoint and start the OIDC authentication flow.

```
apps/prairielearn/src/ee/lib/institution.ts
@@ -48,6 +48,9 @@ export async function getSupportedAuth
48 48      if (row.name === 'Azure') {
49 49          return config.hasAzure;
50 50      }
51 +      if (row.name === 'OID') {
52 +          return config.hasOid;
53 +      }
51 54
52 55      // Default to true for all other providers.
53 56      return true;

```

*institution.ts changes*

### institution.ts

This is also a simple change that updates the getSupportedAuthenticationProviders function to check the OIDC config variable and properly report its status.

```
11 apps/prairielearn/src/migrations/20231114133024_oid_providers__add.sql
... @@ -0,0 +1,11 @@
1 + INSERT INTO
2 +   authn_providers (id, name)
3 + VALUES
4 +   (6, 'OID')
5 + ON CONFLICT DO NOTHING;
6 +
7 + INSERT INTO
8 +   institution_authn_providers (id, institution_id, authn_provider_id)
9 + VALUES
10 +   (6, 1, 6)
11 + ON CONFLICT DO NOTHING;
```

*Adding the SQL migration*

[20231114133024\\_oid\\_providers\\_\\_add.sql](#)

The final change made is adding an SQL migration that inserts OIDC as an authentication provider, as well as adding OIDC as an institution authentication provider. PrairieLearn's migration naming convention takes the current date in decreasing order of importance (yyyyMMddhhmmss) as the identifier, and adds a name with the function at the end.

The authn\_providers table is simply the id and name of the provider, in this case 6 is the last id (might change in the future) and the name is OID. The institution\_authn\_providers table links authentication providers to various institutions. In this case the id (6) is a new entry, the institution\_id (1) points to the default institution, and the authn\_provider\_id (6) points to our newly created provider.

## Configuration

To make the authentication system flexible, we added several configuration variables that set rules for the authentication. These options are set up in `apps/prairielearn/src/lib/config.ts` and link to variables that are set by the user in the `config.json` file passed into the PrairieLearn docker container. Here, there are several categories of variables used by the system. First, there are variables used to configure the OIDC server and enable the login method. Second, there is a set of variables that define the keys that the system looks for in the data returned by the OIDC server. Finally, there are variables that control the look of the login button.

### OIDC Server Configuration

| Variable                     | Description  |
|------------------------------|--|
| <code>hasOid</code>          | Boolean that controls whether or not to use OIDC as a login method. Should be true   |
| <code>oidIssuer</code>       | URL to the OIDC server. For Okta dev servers, it takes the form of <code>“https://&lt;devid&gt;.okta.com/oauth2/default”</code>  |
| <code>oidAuthUrl</code>      | URL to the OIDC authentication endpoint. This URL is used to start the authorization flow. For Okta dev servers, it takes the form of <code>“https://&lt;devid&gt;.okta.com/oauth2/default/v1/authorize”</code>                            |
| <code>oidTokenUrl</code>     | URL to the OIDC token endpoint. This URL is used to serve JWTs to the user after authentication. For Okta dev servers, it takes the form of <code>“https://&lt;devid&gt;.okta.com/oauth2/default/v1/token”</code>                          |
| <code>oidUserInfoUrl</code>  | URL to the OIDC user info endpoint. This URL returns the user information PrairieLearn needs to keep track of users. For Okta dev servers, it takes the form of <code>“https://&lt;devid&gt;.okta.com/oauth2/default/v1/userinfo”</code> . |
| <code>oidClientId</code>     | Client ID for server authentication. This links the PrairieLearn server to the correct OIDC group. Takes the form of a Base64 encoded string.  |
| <code>oidClientSecret</code> | Client secret for server authentication. This authenticates the PrairieLearn server with the OIDC server. Takes the form of a Base64 encoded string.   |
| <code>oidRedirectUrl</code>  | URL used by the OIDC server to redirect traffic back to the PrairieLearn server after authentication. Takes the form of <code>“https://&lt;domain&gt;/pl/oidcallback”</code> .   |



### User Info Configuration

| Variable   | Description  |
|------------|--|
| oidUidKey  | Key pointing to the UID of the logged in user in the JSON object returned by the user info endpoint. Default is “username” for Okta.     |
| oidNameKey | Key pointing to the name of the logged in user in the JSON object returned by the user info endpoint. Default is “displayName” for Okta. |
| oidUinKey  | Key pointing to the UIN of the logged in user in the JSON object returned by the user info endpoint. Default is “id” for Okta.           |

### Login Link Configuration

| Variable      | Description  |
|---------------|--|
| oidLinkText   | Text shown in the login button for OIDC. Default “Sign in with Okta”.  |
| oidLinkLogo   | Path to the SVG logo displayed alongside the login button. Default is “/images/okta_logo.svg”  |
| oidLinkColors | JSON object containing the various coloration rules for the login link. There are four sub-objects for this config: normal, hover, active, and focus. Normal, hover, and active all have three attributes: background, border, and text. Focus only has one attribute: shadow. Normal is used by default for the link, hover is used when the cursor is above the link, and active is used when the link is clicked. Focus is what shows the shadow when the link is in focus. The values for background, border, text, and shadow are just CSS color strings. Any CSS can be used here. |

### Logo Configuration

The logo used alongside the login link is stored in apps/prairielearn/public/images/ and should be an SVG type image. The default Okta logo is stored as okta\_logo.svg.

```
apps/prairielearn/src/lib/config.ts
@@ -258,6 +258,56 @@ const ConfigSchema = z.object({
 258 258     focus: { shadow: 'rgba(255, 83, 0, 0.35)' },
 259 259     },
 260 260     hasAzure: z.boolean().default(false),
 261 + /**
 262 +  * OID config variables are used for configuring OpenIDConnect servers.
 263 +  * Tested to work with Okta
 264 +  */
 265 +     hasOid: z.boolean().default(false),
 266 +     oidIssuer: z.string().nullable().default(null),
 267 +     oidAuthUrl: z.string().nullable().default(null),
 268 +     oidTokenUrl: z.string().nullable().default(null),
 269 +     oidUserInfoUrl: z.string().nullable().default(null),
 270 +     oidClientId: z.string().nullable().default(null),
 271 +     oidClientSecret: z.string().nullable().default(null),
 272 +     oidRedirectUrl: z.string().nullable().default(null),
 273 + /**
 274 +  * OID uid, name, and uin keys are defaults for Okta authentication
 275 +  */
 276 +     oidUidKey: z.string().default("username"),
 277 +     oidNameKey: z.string().default("displayName"),
 278 +     oidUinKey: z.string().default("id"),
 279 + /**
 280 +  * OpenIDConnect login link configuration
 281 +  */
 282 +     oidLinkText: z.string().default("Sign in with Okta"),
 283 +     oidLinkLogo: z.string().nullable().default("/images/okta_logo.svg"),
 284 +     oidLinkColors: z
 285 +     .object({
 286 +     normal: z.object({
 287 +     background: z.string(),
 288 +     border: z.string(),
 289 +     text: z.string(),
 290 +     }),
 291 +     hover: z.object({
 292 +     background: z.string(),
 293 +     border: z.string(),
 294 +     text: z.string(),
 295 +     }),
 296 +     active: z.object({
 297 +     background: z.string(),
 298 +     border: z.string(),
 299 +     text: z.string(),
 300 +     }),
 301 +     focus: z.object({
 302 +     shadow: z.string(),
 303 +     }),
 304 +     })
 305 +     .default({
 306 +     normal: { background: '#3F59E4', border: '#3F59E4', text: 'white' },
 307 +     hover: { background: '#1A31A9', border: '#1A31A9', text: 'white' },
 308 +     active: { background: '#13299C', border: '#13299C', text: 'white' },
 309 +     focus: { shadow: 'rgba(40, 64, 191, 0.35)' },
 310 +     }),
 261 311     hasOauth: z.boolean().default(false),
```

### *config.ts changes*