Backstage notes:

Frontend make request to proxy which is a plugin in backend.

- these request are made with proxyheader: which is localhost:7007/api/proxy (prometheys does it like this)
- Request is made from frontend are just pages. like localhost:3000/grafana/whatever. is going to grafana page, similar to how localhost:3000/catalog goes to catalog page which is a frontend plugin. These pages are defined in App.tsc in packages/app

Backend plugin:

- they can make request to ourside.
- formate: locahost:7007/api/<plugin id>/<router paths defined in that plugin>
- ex: proxy is a backend plugin used by frontend elements to make requests outside.

iframe: can't make requests to backend. It directly embeds whatever you place.

- if you place: url = /grafana/<whatever> if will show localhost:3000/grafana/<whatever> so if that page is not defined in App.tsx you see backstags 404
- if you pleac: url = http://localhost:7007/api/proxy/<a defined end point in app-confg>/request.
  - here backend will take that request. where proxy will get it cause of plugin id and since proxy plugin had routes defined for different path added in app-config. this request is made outside.
- There is no way, in proxy plugin to change response recived. so if response blocks the request you see nothing in the iframe

Middleware:

- extending the proxy plugin with a route made for iframe. which takes tokens from app-config and update headers in response. then iframe will show the contents.

So, Plan, Pick the proxy plugin and extend the page with middleware.

these settings are required for iframe to stop putting frontend domain infront of urls.

```
  csp:
    connect-src: ["'self'", 'http:', 'https:']
    frame-src: ["'self'", 'http:', 'https:']    # allows iframes to load content fro
m these sources
    frame-ancestors: ["'self'", 'http://localhost:3000', 'http://localhost:7007']
# allows your frontend to embed backend pages
    # Content-Security-Policy directives follow the Helmet format: https://hel
metjs.github.io/#reference
    # Default Helmet Content-Security-Policy values can be removed by setti
ng the key to false
```

After this: if iframe has `http://localhost:7007/api/....` it will start like that. Before it use to become like this: `http://localhost:3000/so` `me defaultpage/http://localhost:7007/api/...`

Issue:
proxy plugin already creates routes for any endpoint we add in app-config proxy.endpoints. right. Facts. now our exertion will add a router for a particular endpoint. If this endpoint is also mentioned in app-config then proxy plugin will create 2 routers for same endpoint

Thought:

> either we prevent people to make that endpoint. >> now we dont have source for token for this request so we will have to rely on cookiee and manually adding credentials.

Import note on dangenoulsy allowed auth module: defaut has been changed.

https://backstage.io/docs/next/releases/v1.28.0/#breaking-proxy-backend-plugin-protected-by-default:~:text=The new default is require%2C replacing the old dangerously-allow-unauthenticated. This means some previously

permitted requests may now result in 401 Unauthorized responses. This does not apply if backend.auth.dangerouslyDisableDefaultAuthPolicy is set to true

Router for module that server to iframe a hardcoded response properly.

```
import { PluginEnvironment } from '../types';
import { createProxyMiddleware } from 'http-proxy-middleware';
import { InputError } from '@backstage/errors';
import { z } from 'zod';
import express from 'express';
import Router from 'express-promise-router';
import Request from 'express-promise-router';
import Response from 'express-promise-router';
import NextFunction from 'express-promise-router';
import { HttpAuthService, HttpRouterService, LoggerService  } from '@bac
kstage/backend-plugin-api';

// res.setHeader('X-Frame-Options', 'ALLOWALL');
// res.setHeader('Content-Security-Policy', "frame-ancestors *");

export async function createRouter({
  logger,
  httpAuth,
  httpRouter
  // todoListService,
}: {
  logger: LoggerService;
  httpAuth: HttpAuthService;
  httpRouter: HttpRouterService;
  // todoListService: TodoListService;
}): Promise<express.Router> {
  const router = Router();
  router.use(express.json());

  httpRouter.addAuthPolicy({
    path: '/myiframe',
    allow: 'unauthenticated',
  });
```

```
router.get('/myiframe', async (req, res) => {
  let user = 'Anonymous';
  try {
    const creds = await httpAuth.credentials(req, { allow: ['user'] });
    user = creds.principal.userEntityRef;
  } catch {
    // no creds available, keep "Anonymous"
  }

  logger.info(`Serving /myiframe for user: ${user}`);

  res.send(`
    <!DOCTYPE html>
    <html>
      <head>
        <title>Hello World</title>
      </head>
      <body>
        <h1>Hello World!</h1>
        <p>This page is served from the backend router.</p>
        <p>User: ${user}</p>
      </body>
    </html>
  `);
});

// Proxy endpoint with dynamic target
// router.use(
//   '/myiframe/proxy',
//   createProxyMiddleware({
//     logProvider: () => logger,
//     logLevel: 'debug',
//     changeOrigin: true,
//     target: 'http://localhost', // dummy, replaced dynamically
//     router: (req: Request) => {
//       const urlParam = req.query.url as string;
//       if (!urlParam) {
```

```
    //       throw new Error('Missing "url" query parameter');
    //     }
    //     return urlParam.startsWith('http') ? urlParam : `http://${urlParam}`;
    //   },
    //   pathRewrite: (_path, req) => req.url || '/',
    // }),
    // );

  return router;
}
```

module code:

```
const iframeHandlerModule = createBackendModule({
  pluginId: 'proxy',
  moduleId: 'iframe-handler',
  register: reg => {
    reg.registerInit({
      deps: {
        logger: coreServices.logger,
        httpRouter: coreServices.httpRouter,
        httpAuth: coreServices.httpAuth,
      },
      async init({ logger, httpRouter, httpAuth }) {
        httpRouter.use(await createRouter({ logger , httpAuth, httpRouter }));
        httpRouter.addAuthPolicy({ path: '/api/proxy/myiframe', allow: 'unauth
enticated', });
      },
    });
  }
});
```

This router serve the iframe with the embedded url value, i.e., it shows the Oauth app of github that I have here: https://sample-webpage-2.onrender.com/. Redirects do not work though.

```
export async function createRouter({
  logger,
  httpAuth,
  httpRouter
  // todoListService,
}: {
  logger: LoggerService;
  httpAuth: HttpAuthService;
  httpRouter: HttpRouterService;
  // todoListService: TodoListService;
}): Promise<express.Router> {
  const router = Router();
  router.use(express.json());

  httpRouter.addAuthPolicy({
    path: '/myiframe',
    allow: 'unauthenticated',
  });

  router.get('/myiframe', async (req, res) => {
      let user = 'Anonymous';
      try {
        const creds = await httpAuth.credentials(req, { allow: ['user'] });
        user = creds.principal.userEntityRef;
      } catch {
        // no creds available, keep "Anonymous"
      }
      const { url } = req.query;
      if (!url || typeof url !== 'string') {
        res.status(400).send('Missing ?url param');
        return;
      }

      logger.info(`Serving /myiframe for user: ${user}`);
      // res.setHeader('X-Frame-Options', 'ALLOWALL');
      // res.setHeader('Content-Security-Policy', "frame-ancestors *");
```

```
    try {
      const upstreamRes = await fetch(url);
      const body = await upstreamRes.text();

      // Override headers to allow embedding
      res.setHeader('X-Frame-Options', 'ALLOWALL');
      res.setHeader('Content-Security-Policy', "frame-ancestors *");
      res.setHeader('Content-Type', upstreamRes.headers.get('content-typ
e') || 'text/html');

      logger.info(`Serving /myiframe for user: ${user}, url: ${url}`);
      res.send(body);
    } catch (e: any) {
      logger.error(`Failed to fetch ${url}: ${e.message}`);
      res.status(500).send(`Failed to fetch ${url}`);
    }


    // res.send(`
    //   <!DOCTYPE html>
    //   <html>
    //     <head>
    //       <title>Hello World</title>
    //     </head>
    //     <body>
    //       <h1>Hello World!</h1>
    //       <p>This page is served from the backend router.</p>
    //       <p>User: ${user}</p>
    //     </body>
    //   </html>
    // `);
  });
  return router;
}
```

```
router.use(
  '/myiframe',
  createProxyMiddleware({
    target: 'https://sample-webpage-2.onrender.com', // or dynamic target
    changeOrigin: true,
    selfHandleResponse: false, // let proxy handle the response fully
    onProxyReq(proxyReq, req, res) {
      // optionally add headers or auth tokens
      if (req.headers.cookie) {
        proxyReq.setHeader('cookie', req.headers.cookie);
      }
    },
    onProxyRes(proxyRes, req, res) {
      const cookies = proxyRes.headers['set-cookie'];
      if (cookies) {
        res.setHeader('Set-Cookie', cookies);
      }
      // optionally modify headers, e.g., allow embedding
      proxyRes.headers['X-Frame-Options'] = 'ALLOWALL';
      proxyRes.headers['Content-Security-Policy'] = "frame-ancestors *";
    },
    cookieDomainRewrite: '', // ensures cookies are set for the browser do
main
    followRedirects: true,
  })
);
```

```
router.use(
  '/myiframe', (req, res, next) => {
    const url = req.query.url;
    if (!url || typeof url !== 'string') {
      return res.status(400).send('Missing ?url param');
    }

    createProxyMiddleware({
      target: url, // or dynamic target
      changeOrigin: true,
```

```
      selfHandleResponse: false, // let proxy handle the response fully
      onProxyReq(proxyReq) {
        // optionally add headers or auth tokens
        if (req.headers.cookie) {
          proxyReq.setHeader('cookie', req.headers.cookie);
        }
      },
      onProxyRes(proxyRes, req, res) {
        const cookies = proxyRes.headers['set-cookie'];
        if (cookies) {
          res.setHeader('Set-Cookie', cookies);
        }
        // optionally modify headers, e.g., allow embedding
        res.setHeader('X-Frame-Options', 'ALLOWALL');
        res.setHeader('Content-Security-Policy', 'frame-ancestors *');
      },
      cookieDomainRewrite: '', // ensures cookies are set for the browser do
main
      followRedirects: true,
    })(req, res, next);
  }
 );
```

```
router.use('/myiframe', (req, res, next) => {
  logger.info("this is cookiee info", req.cookies);
  const url = req.query.url;
  if (!url || typeof url !== 'string') return res.status(400).send('Missing ?url
param');

  createProxyMiddleware({
    target: url,
    changeOrigin: true,
    selfHandleResponse: true, // ← take control of the response
    onProxyRes: async (proxyRes, req, res) => {
      let body = '';
      proxyRes.on('data', chunk => (body += chunk));
      proxyRes.on('end', () => {
        // set headers BEFORE sending body
```

```
        res.setHeader('X-Frame-Options', 'ALLOWALL');
        res.setHeader('Content-Security-Policy', 'frame-ancestors *');
        const cookies = proxyRes.headers['set-cookie'];
        if (cookies) res.setHeader('Set-Cookie', cookies);

        res.status(proxyRes.statusCode || 200).send(body);
      });
    },
  })(req, res, next);
```

Rediection will happen.
cookie can’t be found if cookie has setting of that order.

1. **Backstage already has authentication**

- Backstage typically uses OAuth (GitHub, Google, etc.) to log in users.
- When a user logs in, Backstage issues a **session cookie** (e.g., `backstage-session`) or JWT to track that user.

---

2. **Your iframe proxy / external site**

- The external site (e.g., `https://sample-webpage-2.onrender.com`) likely also expects **its own authentication cookie**.
- This cookie is independent of Backstage’s session.

---

3. **Problem**

- If the upstream site uses GitHub OAuth and redirects for login:
  - Your proxy will get a **302 redirect** to GitHub.
  - Browser tries to follow the redirect, but the cookie that the upstream expects **is not present**.
  - Also, your proxy may fail if it tries to rewrite headers while the redirect is happening — which causes errors like `ERR_HTTP_HEADERS_SENT`.

---

4. **Can Backstage authentication help?**

- Only if the external site accepts **Backstage’s session** as a valid login.

- In most cases, the external site doesn't know Backstage's session, so it will still redirect to GitHub.
- Therefore, your proxy **cannot bypass the upstream authentication** just because the user is logged in Backstage.

---

5. **Workarounds**

1. **Use a separate service account** to fetch the upstream content and serve it via proxy:
   - Your proxy logs in (server-side) to the upstream once and reuses a session cookie.
   - Users don't have to authenticate directly to the upstream.
2. **Pass user credentials to upstream** (only if upstream supports OAuth tokens):
   - Backstage stores the user's GitHub token.
   - Proxy attaches it to requests as a header or cookie.
3. **CORS / Same-origin issues**:
   - Browser cannot send upstream cookies automatically if they're cross-origin.
   - Server-side proxy is usually required.

✅ Key Takeaway

Backstage authentication **does not automatically give access to the external site**.

If the upstream site requires OAuth login, your proxy either:

- Needs to handle the login **server-side**, or
- Redirect the user **through the OAuth flow**, which will trigger redirects you must handle.

What about token. It will bypass all the requirments of cookiee. There is no need to save cookiee or send cookies. Just need to update the header to enable embedding.

Problem: redirect (303) >> /auth/github (404).
or cookie related issue, 503, 502 etc.

First of, we are abandoning all our work on: iframe to show page from frontend, frontend page to make request to backend then backend uses our iframe-handler backend plugin to make request to outside.

Instead, Now
We are going to provide backend url in iframe.
so it localhost:7007/api/proxy/myiframe?url=whatever in iframe instead of localhost:3000/iframe-handler?url=whatever.

so as per the new url, you can see, proxy plugin will be used and router for myiframe path will be required.

So now we are craete a module for proxy plugin which will add a router to rotuers of proxy plugin.