Example Implementation

This guide explains how to integrate an external user system with Replyke. For demonstration purposes, we'll use Clerk as the authentication provider. However, the logic can be adapted for other systems, with slight variations based on their specific requirements.

Client-Side Integration

Overview

In this example, we check for a logged-in user from the external authentication system (Clerk). If a user is logged in, a request is made to the backend to retrieve a signed token, which is then consumed by the Replyke `AuthProvider`.

Implementation

Here's the React client-side implementation:

```
import { useEffect, useState } from "react";
import { AuthProvider, ReplykeProvider } from "@replyke/react-js";
import axios from "axios";
import { useUser as useUserClerk, useAuth as useAuthClerk } from "@clerk/clerk-react";

function ContextProvider({ children }: { children: React.ReactNode }) {
  const { user: userClerk } = useUserClerk();
  const { getToken } = useAuthClerk();

  const [signedToken, setSignedToken] = useState<string | null>(null);

  useEffect(() => {
    const generateJwt = async () => {
      try {
        const token = await getToken(); // Get the Clerk session token

        const path = "<YOUR_SERVER_ENDPOINT>"; // Replace with your server's endpoint
        const response = await axios.get(path, {
          headers: {
            Authorization: `Bearer ${token}`, // Attach token to Authorization header
          },
        });
```

```
        if (response) setSignedToken(response.data);
      } catch (error) {
        console.error("Error fetching data:", error);
        throw error; // Re-throw the error for further handling
      }
    };

    if (userClerk) generateJwt();
  }, [userClerk, getToken]);

  return (
    <ReplykeProvider projectId="<YOUR_PROJECT_ID>" signedToken={signedToken}>
      {children}
    </ReplykeProvider>
  );
}

export default ContextProvider;
```

Key Points:

1. **Token Management**: The Clerk session token is retrieved using `getToken`.
2. **API Call**: A `GET` request is sent to the server with the token included in the `Authorization` header. For other systems, ensure their tokens are passed as required.
3. **State Management**: The signed token received from the server is stored in the component state and consumed by the Replyke `AuthProvider`.

Server-Side Integration

On the server, we:

1. Use Clerk middleware to handle authentication.
2. Implement an endpoint to issue a signed token for Replyke.

Middleware Setup

To enable Clerk's middleware:

```
// Apply Clerk middleware
app.use(clerkMiddleware());
```

This ensures requests are authenticated before reaching protected routes.

Endpoint Definition

We protect the route using Clerk's `requireAuth` middleware and delegate the logic to a controller:

```
import { Router } from "express";
import { requireAuth } from "@clerk/express";
import signUserJwt from "../controllers/auth/signUserJwt";

const router = Router();

router.get("/sign-token", requireAuth(), signUserJwt);

export default router;
```

Controller Logic

The controller generates a signed JWT token:

```
import { clerkClient } from "@clerk/express";
import { Request as ExReq, Response as ExRes } from "express";
import jwt from "jsonwebtoken";

export default async (req: ExReq, res: ExRes) => {
  try {
    const { userId } = req.auth;

    const user = await clerkClient.users.getUser(userId);
    const { emailAddresses } = user;

    const email = emailAddresses.find(
      (email) => email.id === user.primaryEmailAddressId
    )?.emailAddress;

    const projectId = process.env.REPLYKE_PROJECT_ID;
    const privateKeyPem = Buffer.from(
      process.env.REPLYKE_PROJECT_PRIVATE_KEY!,
      "base64"
    ).toString("utf-8");

    const payload = {
      sub: userId, // External user's ID
      iss: projectId, // Project ID
      aud: "replyke.com", // Audience
      userData: {
```

```
        email,
      },
    };

    const token = jwt.sign(payload, privateKeyPem, {
      algorithm: "RS256",
      expiresIn: "5m",
    });

    return res.status(200).send(token);
  } catch (err: any) {
    console.error("Error generating token: ", err);
    return res.status(500).send({ error: "Server error" });
  }
};
```

Key Points:

1. **Middleware**: Clerk's middleware attaches the user's ID to the request object.
2. **Private Key**: The private key is converted from a base64 string to a PEM format.
3. **JWT Payload**:
   - `sub`: User ID.
   - `iss`: Project ID.
   - `aud`: Always set to `replyke.com`.
   - `userData`: Optional user details. In this example we pass the email only, but we can pass all properties that are avaialable on the User object (e.g. name, username, location, metadata etc.)
4. **Token Signing**: The JWT is signed using `RS256` with a short expiration time (5 minutes).

Customizing for Other Systems

To adapt this integration for other authentication systems:

1. Replace Clerk-specific logic (e.g., `getToken`, `requireAuth`, and user details extraction) with equivalent methods from your chosen system.
2. Ensure your authentication middleware populates the request with user data.
3. Follow Replyke's JWT payload structure for compatibility.

Conclusion

This example demonstrates a secure and modular approach to integrating an external user system with Replyke. While we used Clerk for demonstration, the same principles apply to other systems with minor adjustments. This flexibility ensures seamless integration with your existing authentication setup.

Last updated on May 6, 2025