Lucia

Tutorial: Google OAuth in SvelteKit

Before starting, make sure you've created the session and cookie API outlined in the Sessions page.

An example project based on this tutorial is also available. You can clone the example locally or open it in StackBlitz.

```
git clone git@github.com:lucia-auth/example-sveltekit-google-oauth.git
```

Create an OAuth App

Create an Google OAuth client on the Cloud Console. Set the redirect URI to `http://localhost:5173/login/google/callback`. Copy and paste the client ID and secret to your `.env` file.

```
# .env
GOOGLE_CLIENT_ID=""
GOOGLE_CLIENT_SECRET=""
```

Update database

Update your user model to include the user's Google ID and username.

```
interface User {
    id: number;
    googleId: string;
    name: string;
}
```

Setup Arctic

```
npm install arctic
```

Initialize the Google provider with the client ID, client secret, and redirect URI.

```
import { Google } from "arctic";
import { GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET } from "$env/static/private";

export const google = new Google(
    GOOGLE_CLIENT_ID,
    GOOGLE_CLIENT_SECRET,
    "http://localhost:5173/login/google/callback"
);
```

Sign in page

Create `routes/login/+page.svelte` and add a basic sign in button, which should be a link to `/login/google`.

```
<!-- routes/login/+page.svelte -->
<h1>Sign in</h1>
<a href="/login/google">Sign in with Google</a>
```

Create authorization URL

Create an API route in `routes/login/google/+server.ts`. Generate a new state and code verifier, and create a new authorization URL. Add the `openid` and `profile` scope to have access to the user's profile later on. Store the state and code verifier, and redirect the user to the authorization URL. The user will be redirected to Google's sign in page.

```
// routes/login/google/+server.ts
import { generateState, generateCodeVerifier } from "arctic";
import { google } from "$lib/server/oauth";

import type { RequestEvent } from "@sveltejs/kit";

export async function GET(event: RequestEvent): Promise<Response> {
    const state = generateState();
    const codeVerifier = generateCodeVerifier();
    const url = google.createAuthorizationURL(state, codeVerifier, ["openid", "prof

    event.cookies.set("google_oauth_state", state, {
        path: "/",
        httpOnly: true,
        maxAge: 60 * 10, // 10 minutes
        sameSite: "lax"
    });
    event.cookies.set("google_code_verifier", codeVerifier, {
```

```
		path: "/",
		httpOnly: true,
		maxAge: 60 * 10, // 10 minutes
		sameSite: "lax"
	});

	return new Response(null, {
		status: 302,
		headers: {
			Location: url.toString()
		}
	});
}
```

Validate callback

Create an API route in `routes/login/google/callback/+server.ts` to handle the callback. Check that the state in the URL matches the one that's stored. Then, validate the authorization code and stored code verifier. If you passed the `openid` and `profile` scope, Google will return a token ID with the user's profile. Check if the user is already registered; if not, create a new user. Finally, create a new session and set the session cookie to complete the authentication process.

```
// routes/login/google/callback/+server.ts
import { generateSessionToken, createSession, setSessionTokenCookie } from "$lib/se
import { google } from "$lib/server/oauth";
import { decodeIdToken } from "arctic";

import type { RequestEvent } from "@sveltejs/kit";
import type { OAuth2Tokens } from "arctic";

export async function GET(event: RequestEvent): Promise<Response> {
	const code = event.url.searchParams.get("code");
	const state = event.url.searchParams.get("state");
	const storedState = event.cookies.get("google_oauth_state") ?? null;
	const codeVerifier = event.cookies.get("google_code_verifier") ?? null;
	if (code === null || state === null || storedState === null || codeVerifier ===
		return new Response(null, {
			status: 400
		});
	}
	if (state !== storedState) {
		return new Response(null, {
			status: 400
		});
	}

	let tokens: OAuth2Tokens;
	try {
		tokens = await google.validateAuthorizationCode(code, codeVerifier);
	} catch (e) {
		// Invalid code or client credentials
		return new Response(null, {
			status: 400
		});
	}
	const claims = decodeIdToken(tokens.idToken());
	const googleUserId = claims.sub;
	const username = claims.name;

	// TODO: Replace this with your own DB query.
	const existingUser = await getUserFromGoogleId(googleUserId);

	if (existingUser !== null) {
		const sessionToken = generateSessionToken();
		const session = await createSession(sessionToken, existingUser.id);
		setSessionTokenCookie(event, sessionToken, session.expiresAt);
		return new Response(null, {
			status: 302,
			headers: {
				Location: "/"
			}
		});
	}

	// TODO: Replace this with your own DB query.
	const user = await createUser(googleUserId, username);

	const sessionToken = generateSessionToken();
	const session = await createSession(sessionToken, user.id);
	setSessionTokenCookie(event, sessionToken, session.expiresAt);
	return new Response(null, {
		status: 302,
		headers: {
			Location: "/"
		}
	});
}
```

Get the current user

If you implemented the middleware outlined in the Session cookies in SvelteKit page, you can get the current session and user from `Locals`.

```
// routes/+page.server.ts
import { redirect } from "@sveltejs/kit";

import type { PageServerLoad } from "./$types";
```

```
export const load: PageServerLoad = async (event) => {
    if (!event.locals.user) {
        return redirect(302, "/login");
    }

    return {
        user
    };
};
```

Sign out

Sign out users by invalidating their session. Make sure to remove the session cookie as well.

```
// routes/+page.server.ts
import { fail, redirect } from "@sveltejs/kit";
import { invalidateSession, deleteSessionTokenCookie } from "$lib/server/session";

import type { Actions, PageServerLoad } from "./$types";

export const load: PageServerLoad = async ({ locals }) => {
    // ...
};

export const actions: Actions = {
    default: async (event) => {
        if (event.locals.session === null) {
            return fail(401);
        }
        await invalidateSession(event.locals.session.id);
        deleteSessionTokenCookie(event);
        return redirect(302, "/login");
    }
};
```

```
<!-- routes/+page.svelte -->
<script lang="ts">
    import { enhance } from "$app/forms";
</script>

<form method="post" use:enhance>
```

```
    <button>Sign out</button>
</form>
```