Session cookies in SvelteKit

This page builds upon the API defined in the Basic session API page.

Cookies

CSRF protection is a must when using cookies. SvelteKit has basic CSRF protection using the `Origin` header is enabled by default.

Session cookies should have the following attributes:

- `HttpOnly`: Cookies are only accessible server-side
- `SameSite=Lax`: Use `Strict` for critical websites
- `Secure`: Cookies can only be sent over HTTPS (Should be omitted when testing on localhost)
- `Max-Age` or `Expires`: Must be defined to persist cookies
- `Path=/`: Cookies can be accessed from all routes

SvelteKit automatically sets the `Secure` flag when deployed to production.

> Lucia v3 used `auth_session` as the session cookie name.

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

// ...

export function setSessionTokenCookie(event: RequestEvent, token: string, expiresAt
    event.cookies.set("session", token, {
        httpOnly: true,
        sameSite: "lax",
        expires: expiresAt,
        path: "/"
    });
}

export function deleteSessionTokenCookie(event: RequestEvent): void {
    event.cookies.set("session", "", {
        httpOnly: true,
        sameSite: "lax",
        maxAge: 0,
        path: "/"
```

```
    });
}
```

Session validation

Session tokens can be validated using the `validateSessionToken()` function from the Basic session API page. If the session is invalid, delete the session cookie. Importantly, we recommend setting a new session cookie after validation to persist the cookie for an extended time.

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

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

export const load: PageServerLoad = async (event) => {
    const token = event.cookies.get("session") ?? null;
    if (token === null) {
        return new Response(null, {
            status: 401
        });
    }

    const { session, user } = await validateSessionToken(token);
    if (session === null) {
        deleteSessionTokenCookie(event);
        return new Response(null, {
            status: 401
        });
    }
    setSessionTokenCookie(event, token, session.expiresAt);

    // ...
};
```

We recommend handling session validation in the handle hook and passing the current auth context to each route.

```
// src/app.d.ts

import type { User } from "$lib/server/user";
```

```
import type { Session } from "$lib/server/session";

declare global {
    namespace App {
        interface Locals {
            user: User | null;
            session: Session | null;
        }
    }
}

export {};
```

```
// src/hooks.server.ts
import {
    validateSessionToken,
    setSessionTokenCookie,
    deleteSessionTokenCookie
} from "./lib/server/session";

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

export const handle: Handle = async ({ event, resolve }) => {
    const token = event.cookies.get("session") ?? null;
    if (token === null) {
        event.locals.user = null;
        event.locals.session = null;
        return resolve(event);
    }

    const { session, user } = await validateSessionToken(token);
    if (session !== null) {
        setSessionTokenCookie(event, token, session.expiresAt);
    } else {
        deleteSessionTokenCookie(event);
    }

    event.locals.session = session;
    event.locals.user = user;
    return resolve(event);
};
```

Both the current user and session will be available in loaders, actions, and endpoints.

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

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

export const load: PageServerLoad = async (event) => {
    if (event.locals.user === null) {
```

```
        return redirect("/login");
    }
    // ...
};

export const actions: Actions = {
    default: async (event) => {
        if (event.locals.user === null) {
            throw fail(401);
        }
        // ...
    }
};
```

```
// +server.ts
import { lucia } from "$lib/server/session";

export function GET(event: RequestEvent): Promise<Response> {
    if (event.locals.user === null) {
        return new Response(null, {
            status: 401
        });
    }
    // ...
}
```