Sessions with Drizzle ORM

Users will use a session token linked to a session instead of the ID directly. The session ID will be the SHA-256 hash of the token. SHA-256 is a one-way hash function. This ensures that even if the database contents were leaked, the attacker won't be able to retrieve valid tokens.

Declare your schema

Create a session table with a field for an ID, user ID, and expiration.

MySQL

```
import mysql from "mysql2/promise";
import { mysqlTable, int, varchar, datetime } from "drizzle-orm/mysql-core";
import { drizzle } from "drizzle-orm/mysql2";

import type { InferSelectModel } from "drizzle-orm";

const connection = await mysql.createConnection();
const db = drizzle(connection);

export const userTable = mysqlTable("user", {
    id: int("id").primaryKey().autoincrement()
});

export const sessionTable = mysqlTable("session", {
    id: varchar("id", {
        length: 255
    }).primaryKey(),
    userId: int("user_id")
        .notNull()
        .references(() => userTable.id),
    expiresAt: datetime("expires_at").notNull()
});

export type User = InferSelectModel<typeof userTable>;
export type Session = InferSelectModel<typeof sessionTable>;
```

PostgreSQL

```
import pg from "pg";
import { pgTable, serial, text, integer, timestamp } from "drizzle-orm/pg-core";
import { drizzle } from "drizzle-orm/node-postgres";

import type { InferSelectModel } from "drizzle-orm";

const pool = new pg.Pool();
const db = drizzle(pool);

export const userTable = pgTable("user", {
    id: serial("id").primaryKey()
});

export const sessionTable = pgTable("session", {
    id: text("id").primaryKey(),
    userId: integer("user_id")
        .notNull()
        .references(() => userTable.id),
    expiresAt: timestamp("expires_at", {
        withTimezone: true,
        mode: "date"
    }).notNull()
});

export type User = InferSelectModel<typeof userTable>;
export type Session = InferSelectModel<typeof sessionTable>;
```

SQLite

```
import sqlite from "better-sqlite3";
import { sqliteTable, integer, text } from "drizzle-orm/sqlite-core";
import { drizzle } from "drizzle-orm/better-sqlite3";

import type { InferSelectModel } from "drizzle-orm";

const sqliteDB = sqlite(":memory:");
const db = drizzle(sqliteDB);

export const userTable = sqliteTable("user", {
    id: integer("id").primaryKey()
});

export const sessionTable = sqliteTable("session", {
    id: text("id").primaryKey(),
    userId: integer("user_id")
        .notNull()
        .references(() => userTable.id),
    expiresAt: integer("expires_at", {
        mode: "timestamp"
    }).notNull()
});
```

```
export type User = InferSelectModel<typeof userTable>;
export type Session = InferSelectModel<typeof sessionTable>;
```

Install dependencies

This page uses Oslo for various operations to support a wide range of runtimes. Oslo packages are fully-typed, lightweight, and has minimal dependencies. These packages are optional and can be replaced by runtime built-ins.

```
npm i @oslojs/encoding @oslojs/crypto
```

Create your API

Here's what our API will look like. What each method does should be pretty self explanatory.

```
import type { User, Session } from "./db.js";

export function generateSessionToken(): string {
    // TODO
}

export async function createSession(token: string, userId: number): Promise<Session
    // TODO
}

export async function validateSessionToken(token: string): Promise<SessionValidatio
    // TODO
}

export async function invalidateSession(sessionId: string): Promise<void> {
    // TODO
}

export async function invalidateAllSessions(userId: number): Promise<void> {
    // TODO
}

export type SessionValidationResult =
    | { session: Session; user: User }
    | { session: null; user: null };
```

The session token should be a random string. We recommend generating at least 20 random bytes from a secure source (**DO NOT USE** `Math.random()`) and encoding it with base32. You can use any encoding schemes, but base32 is case insensitive unlike base64 and only uses alphanumeric letters while being more compact than hex encoding.

The example uses the Web Crypto API for generating random bytes, which is available in most modern runtimes. If your runtime doesn't support it, similar runtime-specific alternatives are available. Do not use user-land RNGs.

- `crypto.randomBytes()` for older versions of Node.js.
- `expo-random` for Expo.
- `react-native-get-random-bytes` for React Native.

```
import { encodeBase32LowerCaseNoPadding } from "@oslojs/encoding";

// ...

export function generateSessionToken(): string {
    const bytes = new Uint8Array(20);
    crypto.getRandomValues(bytes);
    const token = encodeBase32LowerCaseNoPadding(bytes);
    return token;
}
```

> You can use UUID v4 here but the RFC does not mandate that IDs are generated using a secure random source. Do not use libraries that are not clear on the source they use. Do not use other UUID versions as they do not offer the same entropy size as v4. Consider using `Crypto.randomUUID()`.

The session ID will be SHA-256 hash of the token. We'll set the expiration to 30 days.

```
import { db, userTable, sessionTable } from "./db.js";
import { eq } from "drizzle-orm";
import { encodeBase32LowerCaseNoPadding, encodeHexLowerCase } from "@oslojs/encodin
import { sha256 } from "@oslojs/crypto/sha2";

// ...

export async function createSession(token: string, userId: number): Promise<Session
    const sessionId = encodeHexLowerCase(sha256(new TextEncoder().encode(token)));
    const session: Session = {
        id: sessionId,
        userId,
        expiresAt: new Date(Date.now() + 1000 * 60 * 60 * 24 * 30)
    };
    await db.insert(sessionTable).values(session);
```

```
    return session;
}
```

Sessions are validated in 2 steps:

1. Does the session exist in your database?
2. Is it still within expiration?

We'll also extend the session expiration when it's close to expiration. This ensures active sessions are persisted, while inactive ones will eventually expire. We'll handle this by checking if there's less than 15 days (half of the 30 day expiration) before expiration.

For convenience, we'll return both the session and user object tied to the session ID.

```
import { db, userTable, sessionTable } from "./db.js";
import { eq } from "drizzle-orm";
import { encodeBase32LowerCaseNoPadding, encodeHexLowerCase } from "@oslojs/encodin
import { sha256 } from "@oslojs/crypto/sha2";

// ...

export async function validateSessionToken(token: string): Promise<SessionValidatio
    const sessionId = encodeHexLowerCase(sha256(new TextEncoder().encode(token)));
    const result = await db
        .select({ user: userTable, session: sessionTable })
        .from(sessionTable)
        .innerJoin(userTable, eq(sessionTable.userId, userTable.id))
        .where(eq(sessionTable.id, sessionId));
    if (result.length < 1) {
        return { session: null, user: null };
    }
    const { user, session } = result[0];
    if (Date.now() >= session.expiresAt.getTime()) {
        await db.delete(sessionTable).where(eq(sessionTable.id, session.id));
        return { session: null, user: null };
    }
    if (Date.now() >= session.expiresAt.getTime() - 1000 * 60 * 60 * 24 * 15) {
        session.expiresAt = new Date(Date.now() + 1000 * 60 * 60 * 24 * 30);
        await db
            .update(sessionTable)
            .set({
                expiresAt: session.expiresAt
            })
            .where(eq(sessionTable.id, session.id));
    }
    return { session, user };
}
```

Finally, invalidate sessions by simply deleting it from the database.

```
import { db, userTable, sessionTable } from "./db.js";
import { eq } from "drizzle-orm";

// ...

export async function invalidateSession(sessionId: string): Promise<void> {
    await db.delete(sessionTable).where(eq(sessionTable.id, sessionId));
}

export async function invalidateAllSessions(userId: number): Promise<void> {
    await db.delete(sessionTable).where(eq(sessionTable.userId, userId));
}
```

Here's the full code:

```
import { db, userTable, sessionTable } from "./db.js";
import { eq } from "drizzle-orm";
import { encodeBase32LowerCaseNoPadding, encodeHexLowerCase } from "@oslojs/encodin
import { sha256 } from "@oslojs/crypto/sha2";

import type { User, Session } from "./db.js";

export function generateSessionToken(): string {
    const bytes = new Uint8Array(20);
    crypto.getRandomValues(bytes);
    const token = encodeBase32LowerCaseNoPadding(bytes);
    return token;
}

export async function createSession(token: string, userId: number): Promise<Session
    const sessionId = encodeHexLowerCase(sha256(new TextEncoder().encode(token)));
    const session: Session = {
        id: sessionId,
        userId,
        expiresAt: new Date(Date.now() + 1000 * 60 * 60 * 24 * 30)
    };
    await db.insert(sessionTable).values(session);
    return session;
}

export async function validateSessionToken(token: string): Promise<SessionValidatio
    const sessionId = encodeHexLowerCase(sha256(new TextEncoder().encode(token)));
    const result = await db
        .select({ user: userTable, session: sessionTable })
        .from(sessionTable)
        .innerJoin(userTable, eq(sessionTable.userId, userTable.id))
        .where(eq(sessionTable.id, sessionId));
    if (result.length < 1) {
        return { session: null, user: null };
```

```
    }
    const { user, session } = result[0];
    if (Date.now() >= session.expiresAt.getTime()) {
        await db.delete(sessionTable).where(eq(sessionTable.id, session.id));
        return { session: null, user: null };
    }
    if (Date.now() >= session.expiresAt.getTime() - 1000 * 60 * 60 * 24 * 15) {
        session.expiresAt = new Date(Date.now() + 1000 * 60 * 60 * 24 * 30);
        await db
            .update(sessionTable)
            .set({
                expiresAt: session.expiresAt
            })
            .where(eq(sessionTable.id, session.id));
    }
    return { session, user };
}

export async function invalidateSession(sessionId: string): Promise<void> {
    await db.delete(sessionTable).where(eq(sessionTable.id, sessionId));
}

export async function invalidateAllSessions(userId: number): Promise<void> {
    await db.delete(sessionTable).where(eq(sessionTable.userId, userId));
}

export type SessionValidationResult =
    | { session: Session; user: User }
    | { session: null; user: null };
```

Using your API

When a user signs in, generate a session token with `generateSessionToken()` and create a session linked to it with `createSession()`. The token is provided to the user client.

```
import { generateSessionToken, createSession } from "./session.js";

const token = generateSessionToken();
const session = createSession(token, userId);
setSessionTokenCookie(token);
```

Validate a user-provided token with `validateSessionToken()`.

```
import { validateSessionToken } from "./session.js";

const token = cookies.get("session");
if (token !== null) {
```

```
    const { session, user } = validateSessionToken(token);
}
```

To learn how to store the token on the client, see the Session cookies page.