Lucia

Throttling

After each failed attempt, the user has to wait longer before their next attempt.

Memory storage

`timeoutSeconds` holds the number of seconds to lock out the user for.

```
export class Throttler<_Key> {
    public timeoutSeconds: number[];

    private storage = new Map<_Key, ThrottlingCounter>();

    constructor(timeoutSeconds: number[]) {
        this.timeoutSeconds = timeoutSeconds;
    }

    public consume(key: _Key): boolean {
        let counter = this.storage.get(key) ?? null;
        const now = Date.now();
        if (counter === null) {
            counter = {
                index: 0,
                updatedAt: now
            };
            this.storage.set(key, counter);
            return true;
        }
        const allowed = now - counter.updatedAt >= this.timeoutSeconds[counter.inde
        if (!allowed) {
            return false;
        }
        counter.updatedAt = now;
        counter.index = Math.min(counter.index + 1, this.timeoutSeconds.length - 1)
        this.storage.set(key, counter);
        return true;
    }

    public reset(key: _Key): void {
        this.storage.delete(key);
    }
}

interface ThrottlingCounter {
    index: number;
    updatedAt: number;
}
```

Here, on each failed sign in attempt, the lockout time gets extended with a max of 5 minutes.

```
const throttler = new Throttler<number>([1, 2, 4, 8, 16, 30, 60, 180, 300]);

if (!throttler.consume(userId)) {
    throw new Error("Too many requests");
}
const validPassword = verifyPassword(password);
if (!validPassword) {
    throw new Error("Invalid password");
}
throttler.reset(user.id);
```

Redis

We'll use Lua scripts to ensure queries are atomic. `timeoutSeconds` holds the number of seconds to lock out the user for.

```
-- Returns 1 if allowed, 0 if not
local key                     = KEYS[1]
local now                     = tonumber(ARGV[1])

local timeoutSeconds = {1, 2, 4, 8, 16, 30, 60, 180, 300}

local fields = redis.call("HGETALL", key)
if #fields == 0 then
    redis.call("HSET", key, "index", 1, "updated_at", now)
    return {1}
end
local index = 0
local updatedAt = 0
for i = 1, #fields, 2 do
    if fields[i] == "index" then
        index = tonumber(fields[i+1])
    elseif fields[i] == "updated_at" then
        updatedAt = tonumber(fields[i+1])
    end
end
local allowed = now - updatedAt >= timeoutSeconds[index]
if not allowed then
    return {0}
end
index = math.min(index + 1, #timeoutSeconds)
```

```
redis.call("HSET", key, "index", index, "updated_at", now)
return {1}
```

Load the script and retrieve the script hash.

```
const SCRIPT_SHA = await client.scriptLoad(script);
```

Reference the script with the hash.

```
export class Throttler {
    private storageKey: string;

    constructor(storageKey: string) {
        this.storageKey = storageKey;
    }

    public async consume(key: string): Promise<boolean> {
        const result = await client.EVALSHA(SCRIPT_SHA, {
            keys: [`${this.storageKey}:${key}`],
            arguments: [Math.floor(Date.now() / 1000).toString()]
        });
        return Boolean(result[0]);
    }

    public async reset(key: string): Promise<void> {
        await client.DEL(key);
    }
}
```

Here, on each failed sign in attempt, the lockout time gets extended.

```
const throttler = new Throttler<number>("login_throttler");

if (!throttler.consume(userId)) {
    throw new Error("Too many requests");
}
const validPassword = verifyPassword(password);
if (!validPassword) {
    throw new Error("Invalid password");
}
throttler.reset(user.id);
```