```typescript
import { md5 } from '@env/crypto';
import { EventEmitter, Uri } from 'vscode';
import type { GravatarDefaultStyle } from './config';
import { Container } from './container';
import type { CommitAuthor } from './git/models/author';
import { getGitHubNoReplyAddressParts } from './git/remotes/github';
import { base64, equalsIgnoreCase } from './system/string';
import { configuration } from './system/vscode/configuration';
import { getContext } from './system/vscode/context';

const maxSmallIntegerV8 = 2 ** 30 - 1; // Max number that can be stored in V8's smis (small integers)

let avatarCache: Map<string, Avatar> | undefined;
const avatarQueue = new Map<string, Promise<Uri>>();

interface Avatar {
    uri?: Uri;
    fallback?: Uri;
    timestamp: number;
    retries: number;
}

const missingGravatarHash = '00000000000000000000000000000000';

const millisecondsPerMinute = 60 * 1000;
const millisecondsPerHour = 60 * 60 * 1000;
const millisecondsPerDay = 24 * 60 * 60 * 1000;

const retryDecay = [
    millisecondsPerDay * 7, // First item is cache expiration (since retries will be 0)
    millisecondsPerMinute,
    millisecondsPerMinute * 5,
    millisecondsPerMinute * 10,
    millisecondsPerHour,
    millisecondsPerDay,
    millisecondsPerDay * 7,
];

function getAvatarUriCore(
    email: string | undefined,
    repoPathOrCommit: string | { ref: string; repoPath: string } | undefined,
    options?: { cached?: boolean; defaultStyle?: GravatarDefaultStyle; size?: number },
): Uri | Promise<Uri> | undefined {
    ensureAvatarCache(avatarCache);

    // Double the size to avoid blurring on the retina screen
    const size = (options?.size ?? 16) * 2;

    if (!email) {
        const avatar = createOrUpdateAvatar(
            `${missingGravatarHash}:${size}`,
            undefined,
            size,
            missingGravatarHash,
            options?.defaultStyle,
        );
        return avatar.uri ?? avatar.fallback!;
    }

    const hash = md5(email.trim().toLowerCase());
    const key = `${hash}:${size}`;

    const avatar = createOrUpdateAvatar(key, email, size, hash, options?.defaultStyle);
    if (avatar.uri != null) return avatar.uri;

    if (
        !options?.cached &&
        repoPathOrCommit != null &&
        getContext('gitlens:repos:withHostingIntegrationsConnected')?.includes(
            typeof repoPathOrCommit === 'string' ? repoPathOrCommit : repoPathOrCommit.repoPath,
        )
    ) {
        let query = avatarQueue.get(key);
        if (query == null && hasAvatarExpired(avatar)) {
            query = getAvatarUriFromRemoteProvider(avatar, key, email, repoPathOrCommit, { size: size }).then(
                uri => uri ?? avatar.uri ?? avatar.fallback!,
            );
            avatarQueue.set(
                key,
                query.finally(() => avatarQueue.delete(key)),
            );
        }
        return query ?? avatar.fallback!;
    }
    return options?.cached ? avatar.uri : avatar.uri ?? avatar.fallback!;
}

function createOrUpdateAvatar(
    key: string,
    email: string | undefined,
    size: number,
    hash: string,
    defaultStyle?: GravatarDefaultStyle,
): Avatar {
    let avatar = avatarCache!.get(key);
    if (avatar == null) {
        avatar = {
            uri: email != null && email.length !== 0 ? getAvatarUriFromGitHubNoReplyAddress(email, size) : undefined,
            fallback: getAvatarUriFromGravatar(hash, size, defaultStyle),
            timestamp: 0,
            retries: 0,
        };
        avatarCache!.set(key, avatar);
    } else if (avatar.fallback == null) {
        avatar.fallback = getAvatarUriFromGravatar(hash, size, defaultStyle);
    }
    return avatar;
}

function ensureAvatarCache(cache: Map<string, Avatar> | undefined): asserts cache is Map<string, Avatar> {
    if (cache == null) {
        const avatars: [string, Avatar][] | undefined = Container.instance.storage
            .get('avatars')
            ?.map<[string, Avatar]>(([key, avatar]) => [
                key,
                {
                    uri: Uri.parse(avatar.uri),
                    timestamp: avatar.timestamp,
                    retries: 0,
                },
            ]);
        avatarCache = new Map<string, Avatar>(avatars);
    }
}

function hasAvatarExpired(avatar: Avatar) {
    return Date.now() >= avatar.timestamp + retryDecay[Math.min(avatar.retries, retryDecay.length - 1)];
}

function getAvatarUriFromGravatar(hash: string, size: number, defaultStyle?: GravatarDefaultStyle): Uri {
    return Uri.parse(
        `https://www.gravatar.com/avatar/${hash}?s=${size}&d=${defaultStyle ?? getDefaultGravatarStyle()}`,
    );
}

export function getAvatarUriFromGravatarEmail(email: string, size: number, defaultStyle?: GravatarDefaultStyle): Uri {
    return getAvatarUriFromGravatar(md5(email.trim().toLowerCase()), size, defaultStyle);
}

function getAvatarUriFromGitHubNoReplyAddress(email: string, size: number = 16): Uri | undefined {
    const parts = getGitHubNoReplyAddressParts(email);
    if (parts == null || !equalsIgnoreCase(parts.authority, 'github.com')) return undefined;
    return Uri.parse(
        `https://avatars.githubusercontent.com/${parts.userId ? `u/${parts.userId}` : parts.login}?size=${size}`,
    );
}

async function getAvatarUriFromRemoteProvider(
    avatar: Avatar,
    _key: string,
    email: string,
    repoPathOrCommit: string | { ref: string; repoPath: string },
    { size = 16 }: { size?: number } = {},
) {
    ensureAvatarCache(avatarCache);

    try {
        let account: CommitAuthor | undefined;
        if (typeof repoPathOrCommit !== 'string') {
            const remote = await Container.instance.git.getBestRemoteWithIntegration(repoPathOrCommit.repoPath);
            if (remote?.hasIntegration()) {
                account = await (
                    await remote.getIntegration()
                )?.getAccountForCommit(remote.provider.repoDesc, repoPathOrCommit.ref, {
                    avatarSize: size,
                });
            }
        }

        if (account?.avatarUrl == null) {
            // If we have no account assume that won't change (without a reset), so set the timestamp to "never expire"
            avatar.uri = undefined;
            avatar.timestamp = maxSmallIntegerV8;
            avatar.retries = 0;
            return undefined;
        }

        avatar.uri = Uri.parse(account.avatarUrl);
        avatar.timestamp = Date.now();
        avatar.retries = 0;

        if (account.email != null && equalsIgnoreCase(email, account.email)) {
            avatarCache.set(`${md5(account.email.trim().toLowerCase())}:${size}`, { ...avatar });
        }
        return avatar.uri;
    } catch {
        avatar.uri = undefined;
        avatar.timestamp = Date.now();
        avatar.retries++;
        return undefined;
    }
}

let defaultGravatarsStyle: GravatarDefaultStyle | undefined = undefined;
function getDefaultGravatarStyle() {
    if (defaultGravatarsStyle == null) {
        defaultGravatarsStyle = configuration.get('defaultGravatarsStyle', undefined, 'robohash');
    }
    return defaultGravatarsStyle;
}
```