

```

1 import { md5 } from '@env/crypto';
2 import { EventEmitter, Uri } from 'vscode';
3 import type { GravatarDefaultStyle } from './config';
4 import type { StoredAvatar } from './constants.storage';
5 import { Container } from './container';
6 import type { CommitAuthor } from './git/models/author';
7 import { getGitHubNoReplyAddressParts } from './git/remotes/github';
8 import { debounce } from './system/function';
9 import { filterMap } from './system/iterable';
10 import { base64, equalsIgnoreCase } from './system/string';
11 import { configuration } from './system/vscode/configuration';
12 import { getContext } from './system/vscode/context';
13 import type { ContactPresenceStatus } from './vsls/vsls';
14
15 const maxSmallIntegerV8 = 2 ** 30 - 1; // Max number that can be stored in V8's smis (small int
16
17 let avatarCache: Map<string, Avatar> | undefined;
18 const avatarQueue = new Map<string, Promise<Uri>>();
19
20 interface Avatar {
21   uri?: Uri;
22   fallback?: Uri;
23   timestamp: number;
24   retries: number;
25 }
26
27 const missingGravatarHash = '00000000000000000000000000000000';
28
29 const millisecondsPerMinute = 60 * 1000;
30 const millisecondsPerHour = 60 * 60 * 1000;
31 const millisecondsPerDay = 24 * 60 * 60 * 1000;
32
33 const retryDecay = [
34   millisecondsPerDay * 7, // First item is cache expiration (since retries will be 0)
35   millisecondsPerMinute,
36   millisecondsPerMinute * 5,
37   millisecondsPerMinute * 10,
38   millisecondsPerHour,
39   millisecondsPerDay,
40   millisecondsPerDay * 7,
41 ];
42
43 function getAvatarUriCore(
44   email: string | undefined,
45   repoPathOrCommit: string | { ref: string; repoPath: string } | undefined,
46   options?: { cached?: boolean; defaultStyle?: GravatarDefaultStyle; size?: number },
47 ): Uri | Promise<Uri> | undefined {
48   ensureAvatarCache(avatarCache);
49
50   // Double the size to avoid blurring on the retina screen
51   const size = (options?.size ?? 16) * 2;
52
53   if (!email) {
54     const avatar = createOrUpdateAvatar(
55       `${missingGravatarHash}:${size}`,
56       undefined,
57       size,
58       missingGravatarHash,
59       options?.defaultStyle,
60     );
61     return avatar.uri ?? avatar.fallback!;
62   }
63
64   const hash = md5(email.trim().toLowerCase());
65   const key = `${hash}:${size}`;
66
67   const avatar = createOrUpdateAvatar(key, email, size, hash, options?.defaultStyle);

```

```

68     if (avatar.uri != null) return avatar.uri;
69
70     if (
71         !options?.cached &&
72         repoPathOrCommit != null &&
73         getContext('gitlens:repos:withHostingIntegrationsConnected')?.includes(
74             typeof repoPathOrCommit === 'string' ? repoPathOrCommit : repoPathOrCommit.repoPath,
75         )
76     ) {
77         let query = avatarQueue.get(key);
78         if (query == null && hasAvatarExpired(avatar)) {
79             query = getAvatarUriFromRemoteProvider(avatar, key, email, repoPathOrCommit, { size: size });
80             uri => uri ?? avatar.uri ?? avatar.fallback!,
81         };
82         avatarQueue.set(
83             key,
84             query.finally(() => avatarQueue.delete(key)),
85         );
86     }
87
88     return query ?? avatar.fallback!;
89 }
90
91 return options?.cached ? avatar.uri : avatar.uri ?? avatar.fallback!;
92 }
93
94 function createOrUpdateAvatar(
95     key: string,
96     email: string | undefined,
97     size: number,
98     hash: string,
99     defaultStyle?: GravatarDefaultStyle,
100 ): Avatar {
101     let avatar = avatarCache!.get(key);
102     if (avatar == null) {
103         avatar = {
104             uri: email != null && email.length != 0 ? getAvatarUriFromGitHubNoReplyAddress(email, size) :
105             fallback: getAvatarUriFromGravatar(hash, size, defaultStyle),
106             timestamp: 0,
107             retries: 0,
108         };
109         avatarCache!.set(key, avatar);
110     } else if (avatar.fallback == null) {
111         avatar.fallback = getAvatarUriFromGravatar(hash, size, defaultStyle);
112     }
113     return avatar;
114 }
115
116 function ensureAvatarCache(cache: Map<string, Avatar> | undefined): asserts cache is Map<string, Avatar> {
117     if (cache == null) {
118         const avatars: [string, Avatar][] | undefined = Container.instance.storage
119             .get('avatars')
120             ?.map<[string, Avatar]>(([key, avatar]) => [
121                 key,
122                 {
123                     uri: Uri.parse(avatar.uri),
124                     timestamp: avatar.timestamp,
125                     retries: 0,
126                 },
127             ]);
128         avatarCache = new Map<string, Avatar>(avatars);
129     }
130 }
131
132 function hasAvatarExpired(avatar: Avatar) {
133     return Date.now() >= avatar.timestamp + retryDecay[Math.min(avatar.retries, retryDecay.length)];
134 }

```

```

135
136 function getAvatarUriFromGravatar(hash: string, size: number, defaultStyle?: GravatarDefaultStyle) {
137     return Uri.parse(
138         `https://www.gravatar.com/avatar/${hash}?s=${size}&d=${defaultStyle ?? getDefaultGravatarStyle}`
139     );
140 }
141
142 export function getAvatarUriFromGravatarEmail(email: string, size: number, defaultStyle?: GravatarDefaultStyle) {
143     return getAvatarUriFromGravatar(md5(email.trim().toLowerCase()), size, defaultStyle);
144 }
145
146 function getAvatarUriFromGitHubNoReplyAddress(email: string, size: number = 16): Uri | undefined {
147     const parts = getGitHubNoReplyAddressParts(email);
148     if (parts == null || !equalsIgnoreCase(parts.authority, 'github.com')) return undefined;
149
150     return Uri.parse(
151         `https://avatars.githubusercontent.com/${parts.userId ? `u/${parts.userId}` : parts.login}`
152     );
153 }
154
155 async function getAvatarUriFromRemoteProvider(
156     avatar: Avatar,
157     _key: string,
158     email: string,
159     repoPathOrCommit: string | { ref: string; repoPath: string },
160     { size = 16 }: { size?: number } = {},
161 ) {
162     ensureAvatarCache(avatarCache);
163
164     try {
165         let account: CommitAuthor | undefined;
166         if (typeof repoPathOrCommit !== 'string') {
167             const remote = await Container.instance.git.getBestRemoteWithIntegration(repoPathOrCommit);
168             if (remote?.hasIntegration()) {
169                 account = await (
170                     await remote.getIntegration()
171                 ).getAccountForCommit(remote.provider.repoDesc, repoPathOrCommit.ref, {
172                     avatarSize: size,
173                 });
174             }
175         }
176
177         if (account?.avatarUrl == null) {
178             // If we have no account assume that won't change (without a reset), so set the timestamp
179             avatar.uri = undefined;
180             avatar.timestamp = maxSmallIntegerV8;
181             avatar.retries = 0;
182
183             return undefined;
184         }
185
186         avatar.uri = Uri.parse(account.avatarUrl);
187         avatar.timestamp = Date.now();
188         avatar.retries = 0;
189
190         if (account.email != null && equalsIgnoreCase(email, account.email)) {
191             avatarCache.set(`${md5(account.email.trim().toLowerCase())}:${size}`, { ...avatar });
192         }
193
194         return avatar.uri;
195     } catch {
196         avatar.uri = undefined;
197         avatar.timestamp = Date.now();
198         avatar.retries++;
199
200         return undefined;
201     }

```

```
202 }
203
204 let defaultGravatarsStyle: GravatarDefaultStyle | undefined = undefined;
205 function getDefaultGravatarStyle() {
206     if (defaultGravatarsStyle == null) {
207         defaultGravatarsStyle = configuration.get('defaultGravatarsStyle', undefined, 'robohash')
208     }
209     return defaultGravatarsStyle;
210 }
```