System design

We have a local service, installed on the OS, which listens to wss requests from any Code Awareness clients. These clients in turn can take the form on plugins for text editors, add-ons for Microsoft Office products, and more.

 The local service also connects to the Code Awareness API via wss when active, and via https when idle. The local service becomes idle after 10 minutes of inactivity (no clients requests for changes or updates).

An example, when the client is a VSCode extension:

VSCode <- [ WSS ] -> CΩ Local <- [ WSS ] -> CΩ API

VSCode -> [ HTTP ] -> CΩ API

## Authorization:

* the client sends a user/password/origin via wss to the local service
* the local service authorizes the origin with the Code Awareness server and retains the tokens
* SECURITY CHECK: maybe the local service then generates a new series of tokens associated with the origin, and sends them back to the client.
* SECURITY CHECK: after a client authorizes an origin, all other clients can use that authorization, so the user doesn’t have to login to CodeAwareness in both VSCode and vim, for example.

Types of authorization:

* SSO (github, gitlab, etc)
* Swarm auth

 Repo diffs:

* OPTION 1. the local service monitors the repo folder
* OPTION 2. the client sends a wss signal when the user saves a file, deletes a file, or creates a new file / folder.
* when the user opens a file, the client sends that file path to the local service;
* the local service will identify an existing git repo: if the repo has already been authorized, it won’t do anything; otherwise it will request client to authenticate.
* once authenticated, the local service keeps a record of the working repo in an indexDB or encrypted local storage.
* once an origin is authenticated, the local service will send a commit log containing the most recent SHA values (~100 SHA values) to the API.
* after the commit log, the LS will also send the diffs against the common SHA to the API, as a zip file; these diffs include untracked files, in order to expand awareness on new files and folders being created by one peer.
* TODO: local service should monitor or be made aware of changing of branch, committing, rebasing, etc. so that it sends a new commit log to the API with that <origin, branch>
* the user may open files from different origins; the client will always send the file path to the local service, when the text editor has switched to a different file.
* the local service will in turn send the peers and diffs for that file back to the client (peers = list of people who have changed that file since common SHA), (diffs = array of line numbers containing changes)

## PPT like files (direct shares):

As originator:

* the client authenticates the file path
* the client requests a direct share, with access groups
* the local service uploads the file to the API and initiates the group share
* the API sends back a link for each group
* the local service stores these links with the file path

As collaborator:

* the user receives a link via email, or QR code
* the client sends the link to the local service
* local service will download the file from the API and send the download path to the client
* the client opens the file
* TODO: if the user then saves this file somewhere else (different path), the local service should store that new path paired with the original link (origin). At the moment the PPT API does not allow us to intercept a save operation.

## Peer diffs:

* the client sends request for diffs of a single peer
* local service runs a custom git diff command and sends the result to the client
* the client can also send a request for the peer file, in which case the local service applies git patch on top of the common SHA, saves that file in a temp folder, and sends the path back to the client
* the client may request to merge a chunk or an entire file, or replace local version with peer’s file

## Branch diffs:

* the client sends request for diffs against a local branch
* local service performs and sends diffs
* alternatively the client can request the file as it exists in the target branch

## VSCode specific:

* add custom SCM, to track the aggregate of all changed files and folder for the peers involved

## Code Awareness API:

* the system records changes made by a user, to extract essential metrics and use them in AI pairings and recommendation system

## OPTIMIZATIONS:

* in order to reduce the noise, the user may choose to ignore changes made by a list of peers, or list of branches
* similarly, the user may choose to ignore changes at peers that are not in active branch at that peer (a single peer may have worked on several branches).

## FUTURE:

* comments are stored on Code Awareness, rather than inside the code files themselves.
  + issue: code comments make the code prettier, and easier to find the blocks of code you’re looking for
  + benefits: a user may comment not just on a line of code, but also on a file, folder, module, concept, etc, and link these comments together via #tags
  + thumbs up / down to proposed code chunks allow people to give feedback to pieces of growing codebase