ESS QualityNet Workspace Developers

HCMS - QNP Technical Guide

HCMS - QNP Technical Guide.docx

QNP Workspace Developer Onboarding

QNP Workspace Developer Onboarding.docx

QNP Workspace - PRS 2.0 API Documentation

QNP Workspace - PRS 2.0 API Documentation.docx

How does SAFe work?

Source: SAFe Explained in Five Minutes (video)

- Iterations (i.e. sprints) are two weeks long.
  - Agile teams start each iteration by planning what they can deliver in an iteration planning event.
  - The product owner represents the customer for the team and gives guidance for what needs to be built.
  - Daily sync up meetings are used to report on progress.
  - At the end of each iteration, teams demo what they have completed in an Iteration Review.
  - Before the next iteration, the teams meets to discuss what to improve for the next iteration in an Iteration Retrospective.
  - The Scrum Master is the coach of the team.
- An Agile Release Train (ART) is a long-term, dedicated cross-functional team that works toward a singular goal. In other words, it includes all the teams involved in a project. The train is made up of multiple agile teams (not just software developers). They work on a fixed schedule and share the vision, product backlog, and roadmap defined by SAFe.
- A Program Increment (PI) is typically 5 iterations long, but our PIs are quarterly (i.e. 3 months).
  - The ART comes together at the beginning of a PI to plan the work for the next PI.
  - At the end of each PI, the teams come together for a retrospective and discuss how to improve.
- The ART comes together at the end of each iteration for a System Demo.
- Innovation & Planning (I&P): These are 2 weeks long and are used for feature enhancements or improvements. We don't necessarily work on delivering anything during I&P.

Use A Password Manager (It's Free!)

With all of the passwords that we have to use and the strict security requirements we have to follow, it is a good idea to use a password manager. There are plenty of good, free password managers that you can use just for work. Bitwarden is a good option, but there are others as well.

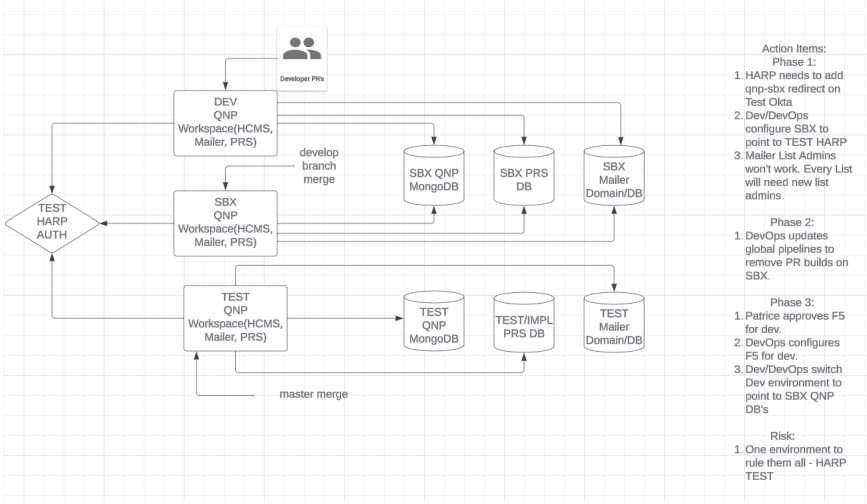
Install Zscaler

Zscaler Installer - User Instructions

Environments & URLs

Overview

Updated 31 Jan 2024



Due to the issues and inconsistencies we are facing on HARP SBX, we have decided to move away from using HARP SBX completely. So the login pages for all of our lower environments (DEV, SBX, TEST) will redirect to the HARP TEST login page.

Which branches does my code get merged into and which environments is my code available in?

- Developer PRs will remain in their own branch (i.e. they will not be merged into another branch) and the code will be available to view/test in the DEV environment.
- After a PR has been approved and you click the "Merge pull request" button in GitHub, your branch will be merged into the `develop` branch and your code will be available to view/test in the SBX environment.
- Code that is merged into `master` is available to view/test in the TEST environment.

Which database URLs does my code point to?

For code that is in the...

- DEV environment, the database URLs point to the SBX databases.
- SBX environment, the database URLs point to the SBX databases.
- TEST environment, the database URLs point to the TEST databases.

Refer to the diagram above.

DEV

- Public URL: https://dev-web-public-qnp.ess.hcqis.org
- Admin Portal: https://dev-web-public-qnp.ess.hcqis.org/admin (redirects to HARP Test Login Page @ https://test.harp.cms.gov/login/login?ADO=QNP2)
- QNP Gateway: https://dev-hcms-apigateway.ess.hcqis.org/gateway/ (Valid Token)
- PRS Gateway: https://dev-prs-apigateway.ess.hcqis.org/gateway/ (Valid Token)
- QNP ALB: https://app-dev-qnp.ess.hcqis.org (expired token)
- PRS ALB: https://app-dev-prs.ess.hcqis.org (expired token)

Notes (Dev & QA Team)

- Since the admin portal redirects to the HARP TEST login page, you need to login with your TEST environment credentials.
  - The first time you login to the DEV environment, your screen might loop between the DEV Admin Portal and the HARP TEST login page a bunch of times. If so, then follow the steps under the Troubleshooting section (below) to fix it.
- To access both public/admin portal(s) - YOU HAVE TO BE LOGGED INTO ZSCALER (since this is not a cms.gov URL)
- QNP Database is pointing to TEST MongoDB.
- QNP Mailer is pointing to TEST Mailman (same domain for dev mailing lists - @test-mailer.qualitynet.org).
- PRS is pointing to IMPL Database (same as PRS Test).
- Since QNP Dev is pointing to TEST Database - CRUD operations and Publish operations on QNP Dev will reflect on QNP TEST as well at the data layer.
- All developer PR's & develop branch merges will be deployed on dev & sbx (sbx is still down).

Dev Team Only

- Everyone will need to create TEST HARP account(s) (if you don't have one already) and request same roles as SBX for local development).
- For local UI development you will still access the Admin UI by going to http://localhost:4200/admin and logging in with your TEST environment credentials.

- The UI for local development will now redirect from TEST HARP login via `ADO=QNP2local` (refer to the `ess-hcms-public-ui/src/configs/config.json` file)

Troubleshooting

The first time you navigate to https://dev-web-public-qnp.ess.hcqis.org/admin, you will either see a browser warning for the URL or an empty screen or your screen might loop between the DEV Admin Portal and the HARP TEST login page a bunch of times. It expects to see .cms.gov URL but dev has a .ess.hcqis.org url. Do the following to fix this issue:

1. Open a new tab in your browser and copy and paste the following URL into the new tab: https://dev-hcms-apigateway.ess.hcqis.org/gateway/userProfile
2. Click on Advanced and hit proceed to *ess.hcqis.org site.
3. Go back to your other tab where you are trying to login and you should see everything working now.
4. You can close the new tab that you opened.

These defects have been logged to track the issues with this new DEV environment

- https://qnetjira.cms.gov/browse/QNWRSPC-4404 - QNP/AHCAH/Alzh
- https://qnetjira.cms.gov/browse/ESSPRS-1451 - PRS/Mailer - This has a few known issues logged which is being worked on.

How to do development work on localhost using SBX resources

If you are working on `localhost` in the DEV environment and you are unable to access the resources that you need (e.g. unable to login to QNP Admin, you get error messages stating that the page is unable to load data or resources) even after following the troubleshooting steps to login, then some resources might be down in the DEV environment. By default your `localhost` environment will point to DEV resources. So as a workaround, you can point your local UI development environment (Public UI, PRS UI, Mailer UI, CRS UI) to SBX resources, like this:

1. In your UI repo, replace the configs in your `node_modules/@qnp/qnp-common/configs/config.json` file with the configs that are shown after these steps.
    a. NOTE: If the following configs do not work, then ping the Pajama Ninjas channel to see if anyone has updated configs that will work.
2. Delete the `.angular` folder from your local UI directory.
3. Run `ng serve` (or `npm start`).
4. Now when you try to login at `localhost:4200/admin` it should point to the SBX login and callback to your `localhost` environment.

```
// Replace node_modules/@qnp/qnp-common/configs/config.json
configs with these:

{
  "production": false,
  "envName": "local",
  "apiGateway": "https://sbx-hcms-apigateway.cms.gov/gateway",
  "prsApiGateway": "https://sbx-prs-apigateway.cms.gov/gateway",
  "publicGateway": "https://sbx-qualitynet.cms.gov/publicgateway
",
  "contentTypesApi": "contentTypes",
  "programsApi": "programs",
  "metadataApi": "metadata",
  "contentApi": "contentManager",
  "contentSubmissionApi": "contentSubmission",
  "clinicians": "/ess-crs/clinicians",
  "publicContentApi": "/public",
  "userProfileEndPoint": "/userProfile",
  "publishApi": "/publish",
  "mailmanAdminApi": "/mailman/admin",
  "listsApi": "/lists",
  "bulkOpsApi": "/bulk-ops",
  "publishPageEndpoint": "/publishPage",
  "previewPageEndpoint": "/previewPage",
  "uiLogout": "https://test-qualitynet.cms.gov",
  "minutesBeforeLogout": 2,
  "issuer": "https://cms-test.okta.com/oauth2
/aus1mjgkubZcmA2d4297",
  "clientId": "0oa1mjexjwOeeLVXg297",
  "redirectUrl": "http://localhost:4200/callback/",
  "oktaBaseUrl": "https://cms-test.okta.com/",
  "OKTA_FEATURES_REMEMBER_ME": false,
  "OKTA_TERMS_CONDITIONS_ENDPOINT": "terms/",
  "OKTA_REDIRECT_ENDPOINT": "redirect/",
  "OKTA_REDIRECT_PARAM_NAME": "appPageName",
  "OKTA_TERMS_CONDITIONS_CHECKBOX_CONTENT": "I agree to the",
  "OKTA_TERMS_CONDITIONS_LINK_CONTENT": "Terms and Conditions",
  "IS_OKTA_HELP_CONTENT_AVAILABLE": false,
  "OKTA_HELP_ENDPOINT": "help",
  "OKTA_HELP_CONTENT": "HCMS Help",
  "HARP_BASE_URL": "https://test.harp.qualitynet.org",
  "HARP_APPLICATION_NAME": "HARP",
  "HARP_SIGNUP_APP_NAME": "HARP Registration",
  "HARP_RECOVERY_APP_NAME": "HARP Account Recovery",
  "HARP_SIGNUP_ENDPOINT": "register/profile-info",
  "HARP_RECOVERY_ENDPOINT": "login/account-recovery",
  "HARP_SIGNUP_HEADER_CONTENT": "Don't have an account?",
  "HARP_SIGNUP_LINK_CONTENT": "Sign Up",
  "HARP_RECOVERY_CONTENT": "Having trouble logging in?",
  "reCaptchaSiteKey": "6LfoYnEUAAAAAPTg1H5_G_VWhUCctmB-xxaDRmxz",
  "gaEnv": "dev",
  "HARP_LOGIN_REDIRECT": "https://test.harp.cms.gov/login/login?
ADO=QNP2local",
  "HARP_LOGOUT_REDIRECT": "https://test.harp.cms.gov/login/ado-
logout?ADO=QNP2local"
}
```

GitHub

How to login to GitHub

Follow these steps to access GitHub repos:

1. Got to https://idm.cms.gov/ and login with your HARP credentials (Ventera email address and HARP prod password).
2. Once you are logged in you will be taken to the “CMS.gov | IDM” dashboard where you will see a card with the GitHub logo and the title “CCSQ GitHub Enterprise Managed User”.
3. Click on that card to access our GitHub repos.
4. Scroll down until you find the repos for “ccsq-ess” and click on it.
5. From here you can search for the specific repo that you need.

NOTE: You might also be able to go to this link and sign in directly, but this needs to be verified to see if it works: GitHub | Sign in with HARP credentials | Sign in to Ctrs for Medicare-Medicaid Svcs, QNET EMU

How to setup your computer to clone Git repos

Create your personal access token for GitHub

To understand how to clone a Git repo using a personal access token (see the instructions above), read this short article: How To: Clone GIT Repo Using Personal Access Token.

This article from GitHub is a good resource for creating a personal access token: Authorizing a personal access token for use with SAML single sign-on.

GitHub has moved away from authenticating via username and password. Use the below steps to create your personal token for accessing all API's.

- Login to GitHub CCSQ-ESS via HARP authentication.
- Click on the user profile icon (top right).
- Navigate into "Settings".
- Click on "Developer Settings" on the bottom of the left navigation.
- Click on the "Personal access tokens"-> "Tokens (classic)" link on the left navigation.
- Click on Generate new token (classic) and provide a name for it.
- Add any permissions as needed.
- Set the expiration to "No expiration."
- Click on "Generate token" at the bottom of the screen.
- Note: Make sure to save this token some place as this cannot be retrieved once you move away from this screen.
- Once you completed the above steps, open a console/terminal and run the following command to store the credentials on your local credential manager
    - `git config --global credential.https://github.com.username` *`your-github-username`*
    - provide and token and you should be all set.

Cache your GitHub credentials in Git

Instead of configuring your GitHub username globally and then providing your access token each time you need to work with Git (see the instructions above), you should cache your GitHub credentials using GitHub CLI. Follow these instructions: Caching your GitHub credentials in Git. *(NOTE: If you are using WSL2, then you can follow the Linux instructions and run all the following commands inside your Ubuntu shell in Windows Terminal.)*

After you install GitHub CLI, and run `gh auth login`, you should see the following prompts in your terminal. Use the same responses that are listed here:

```
? What account do you want to log into? GitHub.com
? You're already logged into github.com. Do you want to re-authenticate? Yes
? What is your preferred protocol for Git operations? HTTPS
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'workflow'.
? Paste your authentication token: ************************************
- gh config set -h github.com git_protocol https
 Configured git protocol
 Logged in as <your-github-username>
```

At this point, when you try to clone a repo you will get this error:

```
Cloning into 'ess-hcms-public-ui'...
remote: The `ccsq-ess' organization has enabled or enforced SAML SSO. To access
remote: this repository, visit https://github.com/enterprises/cms-qnet-emu/sso?
authorization_request=BCIAXBBGBGXBC4ED73MA2S3FAI2TLA5PN5ZGOYLONF5GC5DJN5XF62LEZYCZCBEKVVRXEZLEMVXHI2LBNRPWSZGOJ26Q
T2NPMNZGKZDFNZ2GSYLML52HS4DFVNHWC5LUNBAWGY3FONZQ
remote: and try your request again.
fatal: unable to access 'https://github.com/ccsq-ess/ess-hcms-public-ui.git/': The requested URL returned error:
403
```

Copy the URL shown after the message "To access this repository, visit" and paste it into a browser. You will be redirected to GitHub where you will be required to authenticate. Once you are authenticated into GitHub, go back to your terminal and try cloning the repo again. This time it should work.

Clone a repo with your cached GitHub credentials

Now you can clone GitHub repos like you normally do. You should clone the repos using the HTTPS option. For example,

```
git clone https://github.com/example-user/example-repo.git
```

How to get access to AWS Console

You will need access to an AWS account for various reasons (e.g. to get the MongoDB URL, username, and password from AWS Parameter Store). To get access to an AWS account, send an email to Antonio (Tony) Franco and Praveen Kuninti and CC Sharath Kamarapu. Ask for access to the AWS account/role that corresponds to the project you are working on and the environment you need access to:

- **QNP SBX/DEV:** Account: `adele-ado1-qnp-sbx`, Role: `ADFS-ADELE-ADO1-ADOADMIN-QNP-SBX`
- **QNP TEST:** Account: `adele-ado1-qnp-test`, Role: `ADFS-ADELE-ADO1-ADOADMIN-QNP-TEST`

- **QNP IMPL:** Account: `adele-ado1-qnp-impl`, Role: `ADFS-ADELE-ADO1-ADOADMIN-QNP-IMPL`
- **QNP PROD:** Account: `adele-ado1-qnp-prod`, Role: `ADFS-ADELE-ADO1-READONLY-QNP-PROD`
- **PRS SBX/DEV/TEST/IMP:** Account: `cms-hids-adele-ado1`, Role: `ADFS-ADELE-ADO1-ADOADMIN`
- **PRS PROD:** Account: `hids-adele-ado1-prod`, Role: `ADFS-ADELE-ADO1-ADOADMIN-PROD`

NOTE: SBX and DEV are on the same AWS accounts, so you don't need to request additional access for the DEV environment. The SBX credentials will work for DEV.

Where to find your LDAP ID (or HCQIS ID)

The security team will probably ask you for your LDAP ID as part of your AWS Console access request. The security team should have provided you with an LDAP ID as part of your on-boarding, but if not then this is how you can get your LDAP ID:

1. Login to https://idm.cms.gov/ with your HARP PROD credentials.
2. Once you are logged in, click on your name in the top, right corner and select “Settings”.
3. Your LDAP ID is the ID number labeled HCQIS ID.

What you need to login to AWS Console

In order to login to AWS Console you will need these 3 things:

1. Your Active Directory ID
2. Your Active Directory password
3. A VIP Access Security Code from the VIP Access app

Active Directory (AD) Enrollment & VIP Access

NOTE: If you get stuck at any point in the following process or if this process has changed, then you can reach out to someone from the security team and they should be able to help you out.

AD Enrollment

Once your AWS Console access request has been processed by the security team and you have been granted access to AWS Console you should receive two emails from Perry Brookins:

1. One of the emails will have instructions to enroll in AD, which is how you get your AWS Console login credentials.
2. The other email will have some information that you need in order to complete the AD enrollment.
3. You should also receive instructions for setting up your VIP Access app (a 2-factor authentication app) and your VIP Access credential. Those instructions might be in a separate email, but if you do not receive those instructions, then ask Perry Brookins for those instructions.

NOTE: When you get to the part in the AD enrollment process where you need to enter your HCQIS Account, you will be able to find your HCQIS account ID in the email that has your AD information. It might be labeled “Active Directory ID” and it is in the format `abc1234` (i.e. 3 numbers followed by 4 letters). Take that number and add `@qnet.qualnet.org` to the end to make the HCQIS Account (e.g. `abc1234@qnet.qualnet.org`). That is what needs to be entered into that “HCQIS Account” field.

Setup VIP Access App and Credential

After completing the Active Directory enrollment process you will have to set up the VIP Access app and credential, which is how you get the security codes for logging into AWS Console. If you did not receive those instructions from the security team, then you can follow these instructions:

1. Download the Symantec VIP Access app onto your Ventera laptop: https://vip.symantec.com/
2. Open the VIP Access app. You will see a Credential ID and a Security Code inside the app.
3. Go to https://vipssp.qualnet.org/vipssp/ and login with the Active Directory credentials that you created. For the `Username` field you should enter only your Active Directory ID (i.e. just the `abc1234` part).
4. Once you are logged in, click on “REGISTER” underneath the “VIP Credential” heading.
5. On the next screen you will be able to create a VIP Credential on your laptop.
    a. Give your new credential a name (e.g. VIP Access - Laptop).
    b. Enter the Credential ID and Security Code from your VIP Access app into the corresponding fields in the credential registration form.
    c. Click “Submit”.

You can also install the VIP Access app on your phone and create another credential using the phone app. This can be helpful in case your laptop is not available for some reason. If needed, a member of the security team can reset these for you.

You can also refer to these instructions: AWS Console login instructions

Login to AWS Console

Now you can login to AWS Console at https://sts.qualnet.org/adfs/ls/IdpInitiatedSignOn.aspx.

- Under “Sign in to one of the following sites” select “HCQIS Amazon Web Services”.
- Use the following credentials to sign in
    - **Username:** Your HCQIS Account (e.g. abc1234@qnet.qualnet.org).
    - **Password:** The password that you set during Active Directory enrollment.

- When prompted, enter the security code from your VIP Access app.

How to access AWS resources locally (i.e. how to set AWS credentials locally)

After sending a request through Postman or the UI, if you get an error that includes the following message, then you need to set AWS credentials on your computer:

```
CredentialsError: Missing credentials in config, if using AWS_CONFIG_FILE, set AWS_SDK_LOAD_CONFIG=1
```

This is how to set AWS credentials on your computer:

WSL2 Prerequisites

If you are using WSL2, then you need to install a few things first:

1. Log out of Zscaler.
2. Install `pip`:
    a. Run `sudo apt update`
    b. Then run `sudo apt install python3-pip`. This will install Python pip along with some other missing packages. (Source: https://askubuntu.com/a/1427334)
3. Install the `krb5` package: `sudo apt-get install libkrb5-dev -y` (Source: https://stackoverflow.com/a/74856221/23067953)
4. Install `pipx` for your operating system: https://github.com/pypa/pipx
    a. You can get your Ubuntu version with this command: `lsb_release -a`
5. Install `aws-adfs` following the `pipx` instructions: https://github.com/venth/aws-adfs

Setting up AWS credentials

1. Make sure you have access to AWS Console. You can get access from the security team. See the section How to get access to AWS Console.
2. Install the AWS CLI for your operating system. If you get errors and are unable to install AWS CLI, then you might need to log out of Zscaler or you can set Zscaler certs.
    a. AWS CLI installation command on WSL2 (Ubuntu): `sudo snap install aws-cli --classic`
3. Make sure you are logged into Zscaler.
4. Install `aws-adfs` from here https://github.com/venth/aws-adfs
5. Once `aws-adfs` is installed, run the following command in your terminal to create (or update) the credentials and tokens in your local `.aws/credentials` file: `aws-adfs login --adfs-host sts.qualnet.org --no-sspi --region us-east-1 --s3-signature-version s3v4`
    a. NOTE: If you get an `ExpiredToken` error when you send a request to an API, then you might need to update your credentials with the command above and continue with the rest of the steps below to get the API to pick up your updated credentials/tokens.
6. You should see a long message in your terminal with this at the end: `ERROR: Cannot extract saml assertion from request's response. Re-authentication needed`?
7. That will be followed by separate prompts to enter your AWS `Username` (`abc1234@qnet.qualnet.org`), `Password` and a Symantec VIP Access Code. Enter that information and press Enter.
    a. If you have access to multiple environments, then pick the environment you need access to. The list shows numbers associated with each env role. To see all buckets, run `aws s3 ls`
    b. If you do NOT have access to multiple environments, then you won't be prompted to select an environment.
8. NOTE: This is where your `.aws` folder will be located:
    a. Windows: `C:\Users\<USERNAME>\.aws`
    b. Mac & Linux (WSL2): `~/.aws`
9. Restart the API(s) you are working on (the credentials will be picked up at start up) and try sending your request again (through Postman or the UI).
10. You should have access now and the request should work.

Local development setup

Install Node.js

If you want to use Volta as your Node.js manager, then you will either need to be logged out of Zscaler or have the proper certificates installed on your computer before you can install Volta and the version of Node.js that you need.

VSCode

You can keep code formatting consistent and avoid formatting diffs on PRs by install some extensions and configuring them.

Install extensions

Enable code formatting and linting locally by installing these 3 extensions:

- EditorConfig for VS Code
- ESLint
- Prettier – Code Formatter

Enable format on save

Enable “Format On Save" to automatically format code:

1. Navigate to "File" > "Preferences" > "Settings".
2. On the "Settings" page either navigate to "Text Editor" > "Formatting" or search for "editor.formatOnSave".
3. Check the box next to "Format On Save".

Change default formatter

If formatting isn’t working, you may need to change VSCode’s default formatter:

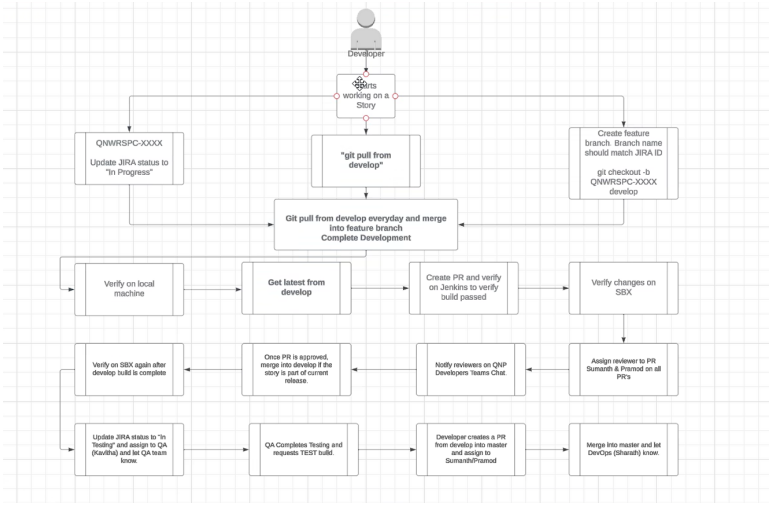
1. Press `Ctrl` + `Shift` + `P`
2. Search for "Format Document With..." and select that option.
3. Select "Configure Default Formatter...".
4. Select "Prettier - Code formatter".

Test your default formatter

You can test that your default formatter is set correctly:

1. Open a file that you want to test your formatting on.
2. Open the command pallete: `Ctrl + Shift + P`
3. Search for and select "Format Document".

Development Workflow



TODOS

- Ajay asked about the number of commits in our PRs. Pramod talked about having a commit at the beginning and at the end of a story and then stashing the other commits in between. Pramod also talked about other commits that are added in response to changes that need to be made during code reviews. Ask for clarification about how we should handle the number of commits in our PRs.
- I need to find out how and when to merge directly into the develop branch. Ajay mentioned something about force pushing, which can trigger a build in the develop branch. Is that correct?

Step 1: Complete preliminary requirements

Update the story status

Each story has a status dropdown box that starts out as “To Do”. That should be changed to “In Progress” when you are ready to start working on the story.

Complete the Impact Analysis

Each story also has a section toward the top labeled “Impact Analysis”. This needs to be completed when you are ready to start working on the story. By that time you should have enough information to complete the Impact Analysis.

NOTES:

- The Impact Analysis questions typically only apply to the API stories.
- This question in the Impact Analysis: “Is this new code, service , API (Y/N)?” should probably say something like “Is this a new microservice (Y /N)?” The answer to this question will usually be “N”.
- For UI stories, the answer to each of the Impact Analysis questions is “N”. When answering this portion for a UI story: “Security Impact Summary (Provide a summary of the impact caused by the story/enabler/bug)” you can answer with something like this: “This is a UI story that only affects the frontend code.”

Once you have answered the Impact Analysis questions, make sure to also set the “SIA Status” of the story to “Pending Security Review”. Click “Edit” toward the top of the story and scroll down until you find the “SIA Status” field.

Evaluate UI stories for possible UX design needs

Some UI stories might need some UX design work to be done before you can complete them. Try to determine as early as possible whether or not a UI story will need UX work done before you start it. Contact a member from the UX team who is assigned to your project and let them know about the UX design needs for your story.

Here is a good habit to get into: At the beginning of a new iteration, look at the stories in the next iteration and try to determine which UI stories will need UX work. Let a UX team member know as early as possible about the UX needs for your story. It might even be a good idea to collaborate with a UX team member and ask them if they think the UI story will need some UX design work.

If you don't realize that a UI story needs UX design work until after you have started working on the story, then that is okay too. Just do your best.

Step 2: Update your local integration branch

IMPORTANT: If your story involves multiple repos, then make sure to follow these steps for each repo that is part of your story.

NOTE: If you need to merge the latest code from `develop` into your feature branch that already exists, then follow the instructions under Step 5. When you have completed those instructions, then continue by going to Step 4.

1. Start out each day by pulling the latest code from the integration branch (`develop`). If you are creating a new feature branch, then you can branch off of the updated `develop` branch.
2. Check which branch you are on (and make sure that any old branches have been deleted): `git branch`
3. Switch to the `develop` branch, if you are not already there: `git switch develop`
4. Get the latest commits from the remote repo using one of the following options:
    a. Get updates from only the remote `develop` branch: `git fetch origin develop`
    b. Get updates from all branches and remove (i.e. prune) any branches that have been deleted: `git fetch -p`
5. Merge your remote tracking branch (`origin/develop`) into your local `develop` branch: `git merge origin/develop`
    a. **NOTE:** If there is new code to merge, then this should result in a fast-forward merge. However, if there are merge conflicts in `develop`, then there is no sense in resolving them. So, instead of dealing with merge conflicts, follow these instructions to overwrite your local `develop` branch with the remote `develop` files: How do I force "git pull" to overwrite local files?

Step 3: Create your feature branch

1. Create your feature branch, which should be based on your integration branch, and switch to it: `git switch -c <feature-branch>`
    a. The name of your feature branch should be the Jira story number (e.g. `QNWRSPC-4293`).
    b. You can also prefix your branch name with your first and last name initials followed by an underscore and then the story number (e.g. `fl_QNWRSPC-4293`).
2. Install (or update) dependencies in your feature branch with `npm install` (API code) or `npm install --force` (UI code). If you pull new updates from `develop` and do not update your dependencies, then your code won’t have the latest updates applied, which could result in errors.
3. Run your code with `npm run dev` (API code) or `npm start` (UI code).

Troubleshooting Build or Compile Errors

When you run `npm run dev` or `npm start` if you get build or compile errors, then try the following:

1. Delete the following folders/files:
    a. `.angular` folder (UI repos only)
    b. `node_modules` folder
    c. `package-lock.json` file
2. Clear the npm cache: `npm cache clean --force`
3. Install dependencies:
    a. `npm install --force` (UI repos)
    b. `npm install` (API repos)

Step 4: Work on your feature branch

NOTE: You need to be logged into Zscaler in order to access things inside the VPN (e.g. Git repos, database).

Default acceptance criteria for each story

Each story has some default acceptance criteria that needs to be implemented before the story can be considered complete, even if these acceptance criteria are not explicitly specified in the story:

1. Unit tests. See the API Testing or UI Testing sections below for details.
2. API documentation with Swagger (for API stories only). See the Swagger API Documentation section below for details.

Include the JIRA story number in your commit messages

When running `git commit` remember to include the JIRA story number in your commit message. This will help track commits in JIRA (your commits will appear in the JIRA story) and will help with any audits in the future. For example:

```
git commit -m "add ids to paragraphs #QNWRSPC-2312"
```

Push code to GitHub early and often

Run `git status` to see which files need to be added, committed, and pushed to your remote branch. All work completed should be checked in and pushed to GitHub by the end of the day, if possible.

You probably won't need to add, commit, or push any config files (e.g. `package.json`, `package-lock.json`, `config.js`), so you can just ignore those. If those config files are updated by a tech lead, then when you run `git fetch` and merge any new updates into your branch, those config files will get updated in your branch too.

Working on API Stories

See the Working on API Stories section below for details.

Working on UI Stories

See the Working on UI Stories section below for details.

Error: remote: Write access to repository not granted

When you push your code up to the remote repo, if you get an error like this:

```
Error:
remote: Write access to repository not granted.
fatal: unable to access 'https://github.com/ccsq-ess/name-of-some-repo.git/': The requested URL returned error:
403
```

…then that means that you do not have access to push code to that repo. You will need to ping Sharath Kamarapu (from the DevOps team) and let him know which repos you need write access to so you can push your code up to them.

Step 5: Prepare your feature branch to merge back into the develop branch

IMPORTANT: Make sure to do this for each of the GitHub repos that belong to the story.

Prepare your feature branch to merge back into the `develop` branch by merging the latest commits in the `develop` branch into your feature branch.

1. Switch to the `develop` branch: `git switch develop`
2. Get the latest commits from the remote repo using one of the following options:

    a. Get updates from only the remote `develop` branch: `git fetch origin develop`
    b. Get updates from all branches and remove (i.e. prune) any branches that have been deleted: `git fetch -p`
3. Merge `origin/develop` into your local develop branch: `git merge origin/develop`
4. Switch back to your feature branch: `git switch <feature-branch>`
5. Then merge `develop` into your feature branch: `git merge develop`
6. Be prepared to fix merge conflicts, if there are any. See the section How to handle merge conflicts for more information.
7. Update dependencies in your feature branch with `npm install` (API code) or `npm install --force` (UI code) otherwise your code won't have the latest updates applied, which could result in errors.
8. Once the merge conflicts have been resolved and your dependencies have been updated, run your code with `npm run dev` (API code) or `npm start` (UI code).
    a. NOTE: When you run `npm run dev` or `npm start` if you get build (or compile) errors, then follow these steps: Troubleshooting Build or Compile Errors
9. Perform final testing (manual and automated) to ensure that everything still works after merging your code.
10. Check the status of your local branch with `git status` and follow Git's hints to add, commit, and/or push your code as necessary.
    a. You only need to include `package.json` files if a package has been added or if a package version has been updated. Package updates will usually be handled by team leads, but there might be situations where you need to update packages (e.g. when working on Common UI changes).
    b. Do not add, commit, or push `package-lock.json` files since new `package-lock.json` files will be generated in the Jenkins build pipeline.

Step 6: Create a pull request

IMPORTANT: Make sure to do this for each of the GitHub repos that belong to the story.

After your code is working locally you need to create a PR and make sure that it passes all the build and deployment stages in Jenkins.

1. Create a PR for your branch in GitHub. From the repo's main page click the "Pull requests" tab >> click "New pull request".
2. On the "Compare changes" page, make sure that the "base" branch is `develop` and the "compare" branch is your feature branch.
3. Click "Create pull request".
4. Enter your story information in the PR's "Title" field, which you can get from the Jira story:
    a. Format: `<Story ID>: <Story Title>`
    b. Example: `QNWRSPC-4236: API - Search by Content Type & Name`
5. You do not need to add a description.
6. Do not assign your PR to anyone yet.
7. Click the "Create pull request" button.
8. When the PR is created, you will see that it runs through some checks. It looks like there might be some errors initially, but just wait a few moments for some of those things to clear.
9. Attach a screenshot of the unit test coverage report to your PR.

How to remove files from a pull request

If you go to the "Files changed" tab in your PR and find that you accidentally checked some files into a PR that you did not want to include (e.g. `package-lock.json`), then you can follow these steps to remove those files from your PR:

Switch to the branch from which you created the pull request:

```
git checkout pull-request-branch
```

Overwrite the modified file(s) with the unmodified version from the `develop` branch:

```
git checkout origin/develop -- package-lock.json path/to/other/file.js
```

Commit and push it to the remote:

```
git commit -m "Removed modified files from pull request"
git push origin pull-request-branch
```

***Source:** Remove a modified file from pull request*

Remember that once you push your changes your PR will be updated and the Jenkins build pipeline will automatically re-run. This time the files that you overwrote in Git will be reverted back to their unmodified versions and they will be removed from your PR.

Step 7: Verify that the build & deploy pipeline completes successfully

In order to see the Jenkins build stages…

1. In your PR you should see a line that says, "Unit Tests and Sonar Quality Analysis" with a link labeled "Details" next to it.
    a. If you don't see a "Unit Tests and Sonar Quality Analysis" line, then refresh your page or find the line where it says, "All checks have passed" and click "Show all checks" to the right of it. Now you should see the "Details" link.
2. Click on the "Details" link to the right of "Unit Tests and Sonar Quality Analysis".

3. Clicking the “Details” link will take you to the Jenkins server (CloudBees). You might have to login (with your email address and HARP PROD environment password) to see the Jenkins build and deploy pipeline. If you are already logged in through idm.cms.gov (see the section How to login to GitHub), then you shouldn’t have to log into Jenkins again.
4. Once you are logged in, click on the name of the repo in the breadcrumbs (e.g. ess-hcms-public-ui, ess-hcms-data-api) to see the build and deploy pipeline.
5. The build stages will show which stages are passing or failing due to errors.
    a. If your code builds without errors, then you will see a green checkmark with a circle around it next to the page heading. Your code will also be deployed into DEV.
    b. If there are errors, then you will see a red X with a circle around it and a “Failed” box somewhere in the build pipeline.

NOTES:

- Once a PR has been created for your branch, then anytime you push code to that remote branch it will automatically start a new Jenkins build process. You can go through this process (edit code, push to remote branch, start build process) as many times as you need.
- If something goes wrong with the Jenkins build pipeline, then you can fix any errors in your code that are preventing it from building and push to your remote branch again.
- The last two stages in the Jenkins build pipelines are under the control of the DevOps team. If there are any failures in those stages, then contact someone from the DevOps team (e.g. Sharath Kamarapu) and let them know about the failure.

Step 8: Manually verify your changes in DEV

After the build and deployment have completed successfully, then you need to manually verify that everything is working in DEV.

API Changes

For API changes, you need to manually verify your changes by sending requests through Postman. In Postman set the “Environment” option (the dropdown box in the top, right corner) to the corresponding option:

These **Environments** (host URLs) point to the internal ALB URL and can be used with expired tokens:

QNP ALB (Amazon Load Balancer)

- DEV Data Cluster: https://data-dev-qnp.ess.hcqis.org
- DEV App Cluster: https://app-dev-qnp.ess.hcqis.org
- SBX Data Cluster: https://data-sbx-qnp.ess.hcqis.org
- SBX App Cluster: https://app-sbx-qnp.ess.hcqis.org

PRS ALB

- DEV Data Cluster: https://data-dev-prs.ess.hcqis.org
- DEV App Cluster: https://app-dev-prs.ess.hcqis.org

The Amazon Load Balancer (ALB) URLs are not publicly accessible and cannot be hit directly by the UI. They are only enabled on SBX for developers and testers to access.

UI Changes

For UI changes, you need to manually verify your changes by doing this:

If the code for this story is located in the Admin Portal, then login to the Admin Portal first:

1. For QNP, go to the QNP Admin site in DEV (https://dev-web-public-qnp.ess.hcqis.org/admin). You can refer to the Environments & URLs section for the different URLs.
2. Login with your HARP TEST credentials (because the login page in DEV redirects to the HARP Test Login Page at https://test.harp.cms.gov/login/login?ADO=QNP2).

Whether the code for this story is located in the Admin Portal or the public side, navigate to the location where you made your changes and test that everything is working as it should.

TODO: Find out what these URLs are used for:

These **Environments** (host URLs) point to the Gateway URL and have to be used with valid and un-expired tokens:

1. Gateway - Admin: https://sbx-hcms-apigateway.cms.gov/gateway
2. Gateway - Public: https://sbx-qualitynet.cms.gov/publicgateway

The Gateway URLs are what the UI calls to and they are accessible from the public Internet. So calls to those URLs require an active token.

Step 9: Assign pull request & perform code review

IMPORTANT: Make sure to do this for each of the GitHub repos that belong to the story.

After your manual verifications are complete, then go back to your pull request in GitHub and tag a reviewer for your pull request.

For API stories, tag Pramod Pradhan as the reviewer. For UI stories, tag Sumanth Venkata as the reviewer. Then ping the reviewer to let them know that your PR is ready for review. For example, you could write something like this:

I have tagged you as the reviewer for these 3 PRs. Will you please review these when you have a moment? Thank you!

- *https://github.com/ccsq-ess/ess-hcms-data-api/pull/123*
- *https://github.com/ccsq-ess/ess-hcms-content-manager-api/pull/456*
- *https://github.com/ccsq-ess/ess-hcms-public-api/pull/789*

Create a new comment in the Jira story giving a progress update. You might want to say something like, “I have created a PR for this story and have tagged <Reviewer’s Name> as the reviewer.” It would also be a good idea to include the PR URLs in your comment. The PR URLs can help you remember which repos need to be included in a master merge.

The reviewer will leave comments in the PR when necessary. You can make any necessary edits and push your code up to the remote branch, which will automatically add the new commits to your PR. Then when you are ready for the next review, ping your reviewer.

Step 10: Verify that your PR should be included in the next release

Once the reviewer approves your PR, you will get an email that says, “@reviewer_username approved this pull request.” The “Merge pull request” button in your PR will be enabled, which will allow you to merge your feature branch into the `develop` branch, but **DO NOT** click that button yet.

We will have a code freeze every other Thursday (by COB) for the upcoming release. That means that no new code merges should occur after the code freeze (because a `master` merge will be performed after the code freeze). Before you merge any code into the `develop` branch, you need to confirm with your Scrum Master and/or the QA team if your code will be going into the next release. You can ping Kavitha with something like this:

Hi Kavitha. I have a story that is ready for testing, but I want to find out if it should be included in the next release. This is the story: https://qnetjira.cms.gov/browse/QNWRSPC-4287

Kavitha will let you know if your PR should be included in the next release or not.

If your PR is NOT supposed to be included in the next release…

- Then leave your code in the feature branch without merging it into `develop` (i.e. **DO NOT** click the "Merge pull request" button in your PR).
- Leave a comment in the Jira story explaining the situation. Something like this will work: "This story is ready for testing, but it won't be included in the next release. Kavitha will create a QA story to test this story for release <X.X.X>." (Insert the corresponding release number in place of the <X.X.X>.)
- If Kavitha has already created a QA story that links back to your story, then...
    - You should see a QA story under the "Issue Links" section on your Jira story page. You can click the story link and read the description to confirm that the QA story is the one that is supposed to verify your PR.
    - You can mark your story as "Done".
- If Kavitha has **NOT** already created a QA story that links back to your story, then she will usually let you know what she would like for you to do. For example, she might tell you that she will create a QA story and that you can mark your story as "Done". Depending on what Kavitha tells you to do, you will usually take one of two actions:
    - Ping Kavitha back and let her know that you will mark your story as "Done" after she has created her QA story (which you will see under the "Issue Links" section in your story). Then you can make a comment in your story that provides an update; something like this: "I will mark this story as "Done" after Kavitha has created her QA story."
    - Or assign your story to Kavitha and ping the story number to her saying something like this: "I have assigned <story link> to you. You can create a QA story to test this story when you are ready." Kavitha will create a QA story that links back to your original story and she will mark the original story as "Done".
- After the QA team has tested and verified your code for the given release, then Kavitha will ping you and let you know when your PR can be merged into the `develop` branch.
    - NOTE: If necessary, the DevOps team can handle your PR merge.

This process will avoid cherry picking and reverting code that should not be included in a particular release.

If your PR is supposed to be included in the next release…

1. Click the “Merge pull request” button. (Use the default button option to “Create a merge commit”.)
2. Click the “Confirm merge” button.
3. After you click the “Confirm merge” button, you should see a new message in its place that says, “Pull request successfully merged and closed” with a button to the right of it labeled “Delete branch”. Click that button to delete your old feature branch.
    a. NOTE: There is no sense in keeping a messy repo with a bunch of old branches. Your commit history will be included in your integration branch, so you should delete any old feature branches that are no longer being used.
4. Go to https://qnetjenkins.cms.gov/ess-master/job/hcms/job/develop/ and find the `develop` build pipeline for your repo.
5. After the `develop` build is complete, check your code in DEV again to verify that everything is still working and that nothing in DEV has broken.

NOTE: *We talked about using this process after we had both DEV and SBX environments available.* ***TODO: Ask Pramod for clarification about this process to see if it has been recorded correctly.*** *We are already using most of these steps, but we need clarification about the* `release` *branch and whether we are using that or not. If we are using a* `release` *branch, then we also need clarification about how to use that in this development flow when we merge code.*

1. *PRs from feature branches will go into the DEV environment. Code reviews will take place in the DEV environment.*
2. *We will have a* `release` *branch that will go into both the DEV and SBX environments.*
3. *PRs that have been reviewed but* ***NOT*** *approved for the next release need to stay in their own feature branch, so do not click the "Merge pull request" button.*
4. *PRs that have been reviewed and approved for the next release need to be merged into the* `release` *branch.*
    a. *Before you merge any code into the* `release` *branch, you need to confirm with your Scrum Master and the QA team if your code will be going into the next release. If not, it's better to leave your code in the feature branch until it is ready to be included in the next release at which point your code can be merged into the* `release` *branch. This will avoid cherry picking and reverting code.*
5. *The* `release` *branch will then be merged into the* `develop` *branch and the* `develop` *branch will be merged into* `master` *when the release is ready.*

Step 11: QA Testing

After your code has been merged into the `develop` branch, it will be deployed into the DEV environment.

NOTE: For API stories, the Swagger docs have to be approved and merged into DEV before the QA team can test any APIs. So as long as the Swagger docs are ready, then the QA team can take it from here.

Now do the following:

1. In the Jira story:
    a. Assign the story to Kavitha Kommineri (she is on the QA team).
    b. Change the status from "In Progress" to "Testing".
    c. Put a note in the "Comments" section that says something like this: "The PR has been approved and I have assigned this story to Kavitha for testing."
2. In Teams, ping Kavitha and let her know that your story has been approved and merged into DEV and that it is ready to be tested.

Working on API Stories

Things To Know

/config/config.js

Each API repo has a `/config/config.js` file where all of the environment variables are loaded. These env vars include parameters like the MongoDB URL, username, and password that you can get from the AWS Parameter Store. The values for each of the env vars will be different for each environment that the code is deployed to (e.g. SBX, TEST, IMPL, PROD). The env vars will get loaded when the containers startup.

NEW_RELIC_APP_NAME. Not starting!

When you run the APIs locally you will see this error in the terminal. New Relic is something used by the DevOps team. This error won't impact your local development and you can ignore it.

Connect to MongoDB from a data cluster API & Run your API code

You will need to connect to MongoDB through your API code. Also, in order to inspect the MongoDB database, you will need to connect to MongoDB through a MongoDB GUI client.

Store the environment variables in a .env file

In order to complete the following steps, you will need access to AWS Console. See the section How to get access to AWS Console for details.

NEW INSTRUCTIONS (the old instruction are below, if you need them)

All environments are now migrated to MongoDB Atlas.

Environment Variables in .env file:

Look at the params that are in the `/src/resources/ssm.params.js` file and include those in your `.env` file. For example:

```
# For app cluster APIs, these environment variables are typically used:

wso2ei_url=
services_alb_url=
```

```
# For data cluster APIs, these environment variables are typically used (and there might be some others that
you need to include):

# After a few releases Atlas will be the default database and we will get rid of this flag
ess_qnp_mongo_atlas_feature_flag=Active

# SBX
mongodb_url=
mongodb_username=
mongodb_password=

# TEST
mongodb_url=
mongodb_username=
mongodb_password=

publish_bucket=
preview_bucket=
access_bucket=
```

```
# If the following params are present in the /src/resources/ssm.params.js file, then you will need to include
these too:

okta_hcms_redirect_url=
okta_hcms_url=
okta_hcms_issuer_id=
okta_hcms_client_id=
okta_hcms_client_secret=
```

1. Create a `.env` file in your repo. This `.env` file will contain hardcoded values for your local development environment. Copy and paste the env vars above into your `.env` file.
2. For each of those env vars, you need to get the values from AWS.
3. After you have been granted access to AWS, go to https://sts.qualnet.org/adfs/ls/IdpInitiatedSignOn.aspx and login with your AWS Console credentials.
4. Each environment variable in your `.env` file has a corresponding variable in AWS Parameter Store.
5. Copy an environment variable name from your `.env` file (e.g. `wso2ei_url`) and go back to AWS Console.
6. In AWS Console, search for “parameter store”.
7. In the Parameter Store, paste the variable name that you copied (e.g. `wso2ei_url`) in the search bar underneath the “My parameters” heading and press Enter.
    a. NOTE: Make sure that the region in AWS Console is set to **US East (N. Virginia) us-east-1** otherwise you might not see the "My parameters" heading or the search bar.
8. Click on the search result for `/dev/qnp/wso2ei_url`.
9. Copy the variable value (which is under the "Value" heading).
10. In your text editor, inside your `.env` file (the one you created above), paste that value next to the corresponding env variable.
11. Do the same with the other env variables in your `.env` file.

OLD INSTRUCTIONS:

Environment Variables in .env file:

```
MONGODB_URL=
MONGODB_USERNAME=
MONGODB_PASSWORD=
PUBLISH_S3_BUCKET_NAME=
ACCESS_S3_BUCKET_NAME=
PREVIEW_S3_BUCKET_NAME=
```

1. Create a `.env` file in the data cluster repo. This `.env` file will contain hardcoded values for your local development environment. Copy and paste the env vars above into your `.env` file.
2. For each of those env vars, you need to get the values from AWS.
3. After you have been granted access to AWS, go to https://sts.qualnet.org/adfs/ls/IdpInitiatedSignOn.aspx and login with your AWS Console credentials.

4. In your text editor, open the `/templates/config.j2` file in the data cluster repo. This file contains env vars along with references to their values. The value references with the following format contain the variable names that are stored in AWS Parameter Store: `'{{mongodb_url}}'`
5. Copy the AWS variable name from your `/templates/config.j2` file (e.g. `mongodb_url`) and go back to AWS Console.
6. In AWS Console, search for “parameter store”.
7. In the Parameter Store, paste the variable name that you copied (e.g. `mongodb_url`) in the search bar underneath the “My parameters” heading and press Enter.
8. Click on the search result for `/dev/qnp/mongodb_url`.
9. Copy the variable value (which is under the "Value" heading).
10. In your text editor, inside your `.env` file (the one you created above), paste that value next to the corresponding env variable.
11. Do the same with the other env variables in your `.env` file.

Whitelist IP addresses for MongoDB Atlas

You will need to get two IP addresses whitelisted:

1. Your Public IP: Open a browser window and navigate to https://www.whatismyip.com/. You should see "My ISP: <Your ISP>".
2. Your Zscaler IP: Log in to Zscaler, open an incognito browser window and navigating to https://www.whatismyip.com/. You should see "My ISP: Zscaler Inc."

Send those two IP addresses to Sharath and ask him to whitelist your IPs for MongoDB Atlas in both the SBX and TEST environments.

NOTE: We have not had any luck getting Studio 3T to work with Atlas, but you can use Compass or any other MongoDB client that is supported by MongoDB Atlas.

How to connect to MongoDB & run your API code

The data cluster repos (e.g. the Data API repo) will connect to the MongoDB instance and then app cluster APIs (i.e. “passthrough” APIs) will make API calls to the data cluster repos.

For example, if you are working on an app cluster API, then you would start the data cluster API first (`npm run dev`), then you would start the app cluster API (`npm run dev`). However, there are four problems with this (with their solutions below each problem):

Problem #1: Missing environment variables

Problem #1:

The `/src/resources/ssm.params.js` file in the data cluster repo has a bunch of env variables that are expecting values to be passed into the environment.

Solution #1:

Create a `.env` file in the data cluster repo that has hardcoded values for your local development environment. You already did this above.

Problem #2: Conflicting port numbers

Problem #2:

Both of those repos will run on the same port locally (see the port variable in the `/bin/www` file in each repo).

Solution #2:

Change the port of the data cluster repo (on line 24 of the `/bin/www.js` file) from `3000` to `3001`.

Problem #3: What URL should the `wso2ei_url` env variable point to?

Problem #3:

Your `.env` file in the app cluster API repo (e.g. Content Manager API repo) has this env variable: `wso2ei_url=https://data-dev-qnp.ess.hcqis.org`

The `https://data-dev-qnp.ess.hcqis.org` URL points to the data cluster URL in the DEV environment, which is not part of your local development environment.

So what URL should you use in order to point to the data cluster API for local development? That depends.

Solution #3:

1. If you are only working on an endpoint for an app cluster API (i.e. you are not changing anything in a data cluster API), then you can leave the `wso2ei_url` environment variable as is (e.g. `wso2ei_url=https://data-dev-qnp.ess.hcqis.org`) and any requests that are sent to the app cluster API will be forwarded to the data cluster API URL in the DEV environment.
2. If you are working on an endpoint for an app cluster API along with an endpoint in the data cluster API, then you will need to comment out the `wso2ei_url=https://data-dev-qnp.ess.hcqis.org` environment variable and add this environment variable below it (which will point to your local data cluster API): `wso2ei_url=http://localhost:3001`

Problem #4: Missing SSL certificates

Problem #4:

When you run an app cluster API (e.g. Content Manager API) you will get a really long error that has this message toward the top (which indicates missing SSL certificates): `Error: Cannot find module '../../certs/QNET-Issuer-CA-v1.pem'`

Solution #4:

The app cluster APIs talk to the data cluster APIs via HTTPS, which requires SSL certificates. In your cloned app cluster API repo, create a folder called `certs` inside the project root directory and add the following two SSL certificates (which you can get from Pramod):

1. QNET-ROOT-CA-v1.pem
2. QNET-Issuer-CA-v1.pem

TODO: Find out where to get the SSL certificates.

How to create a connection to MongoDB in your Studio 3T Free GUI client

In order to test CRUD operations against the MongoDB database, you will need to connect to a MongoDB GUI. Studio 3T Free is a GUI that many people on our team use, but you can use something else if you prefer.

NOTE: When we move to MongoDB Atlas, then the process to create a connection might be different from what is explain below—it might use an automatic configuration.

1. Open Studio 3T Free.
2. Click “Connect”. This will open a new window titled “Connection Manager”.
3. Click “New Connection”
4. “Manually configure my connection settings” >> “Next”
5. **Connection name**: “QNP – DEV”
6. Under the **Server** tab. The connection configurations will depend on the `mongodb_url` value in your `.env` file.
    a. If your `mongodb_url` value is a single URL:
        i. **Connection Type**: Standalone
        ii. **Server**: The `mongodb_url` value from your `.env` file.
        iii. **Port**: The port that is specified in your `mongodb_url` value. If no port is specified, then try 27017 (the default MongoDB port) or 27031 (the port number that is often used on our team).
    b. If your `mongodb_url` value is made up of multiple URLs (e.g. 3 URLs) separated by commas:
        i. **Connection Type**: Replica Set
        ii. **Members**: To the right of the "Members" textbox, click the "Add" button to open an "Add New Member" modal. You will see a "Server" field and a "Port" field. Now if you look in your data cluster API’s `.env` file for `mongodb_url` you will see that the value of that variable is 3 URLs separated by commas. Paste the first URL in the "Server" field and the port number in the port field. Do the same thing for the other two URLs. You should see three members in the "Members" textbox.
7. Under the **Authentication** tab:
    a. **Authentication Mode**: Basic (SCRAM-SHA-256)
    b. **User name**: Paste the value for the `mongodb_username` environment variable from your data cluster API's `.env` file.
    c. **Password**: Paste the value for the `mongodb_password` environment variable from your data cluster API's `.env` file.
    d. **Authentication DB**: admin
8. Click “Save”.
9. **IMPORTANT:** Make sure that you are logged into Zscaler otherwise you won’t be able to connect to the database.
10. In the “Connection Manager” window, click “Connect”.
11. Close the “Welcome” or “Quickstart” window to reveal the “Open connections” side panel.
12. The “Open connections" panel is where you will see your databases and the collections inside those databases. Look for the “hcms” database and expand it.
13. From there you can inspect any of the collections you need to work with.
    a. Double-click on the name of a collection to open a Mongo Shell window where you can perform queries against the database.
14. You can disconnect from the database by right clicking the database in the “Open connections” panel >> “Disconnect”.

Studio 3T Tutorials

Use Postman for API development & testing

We develop API stories first and then any corresponding UI stories after. So after you have created your API endpoints, you can test those endpoints with Postman. Make sure that your API is running locally and then you can run your Postman requests against your API.

Pramod has a Postman Collection that he can share with you. Each URL in that collection is prefixed with `{{host}}`. You can change that `host` variable by clicking on the **Environments** dropdown box in the top, right corner of Postman and selecting the environment that you want to run the API in. (The default value of that dropdown box is “No Environment”.) When you change the environment, Postman will prefix the APIs with the corresponding host (e.g. http://locahost:3000, http://locahost:3001, https://app-sbx-qnp.ess.hcqis.org).

How to create new Postman environments

You can create new environments (which will replace the `host` variable) by clicking the “Environments” tab in the left column.

1. Replace “New Environment” (above the “Filter variables” search box) with the name of the environment that you are creating. This name will appear in the **Environments** dropdown box in the top, right corner.
2. **Variable**: Set this to `host`.
3. **Type**: Leave this at `default`.
4. **Initial value**: Set this to the URL host that you want for this environment.
5. **Current value**: This will be set to the Initial value by default when you click “Save”.
6. Click “Save”.

What is the difference between the Initial value and the Current value in the Environments settings above?

When you enter the Initial value and press Save, then Postman will copy that Initial value over to the Current value. You can change the Current value any time and Postman will save a history of variable values.

How to use auth tokens with Postman

The data cluster APIs do NOT need to use an auth token because authentication is not verified in the data cluster. However, authentication is verified in the app cluster, so you will need to use an auth token with any endpoints that run in the app cluster.

To get an auth token and set it in Postman:

1. Log into the HCMS. For example, if you are working in the DEV environment, then go to https://dev-web-public-qnp.ess.hcqis.org/admin/hcms/content-dashboard.
    a. Once you are logged in, open the "Network" tab and refresh the page to load the network requests.
    b. Under the Network tab, look under the "Name" column and find the “userProfile” request. Click on "userProfile".
    c. A panel will open to the right. Click the “Headers” tab.
    d. Expand the “Request Headers” section.
    e. Find the “Authorization” header and copy everything after “Bearer”.
2. Back in Postman…
    a. Click the “Environment” tab in the left tab pane.
    b. Click “Globals” and add a new variable:
        i. **Variable**: Give the variable a name (e.g. `token_sa` or `token_system_admin`).
        ii. **Type**: Leave this at `default`.
        iii. **Initial value**: Paste the auth token in this field.
        iv. **Current value**: This will be set to the Initial value by default when you click “Save”.
        v. Click “Save”.
    c. When you select an endpoint in Postman that needs to use the auth token, you need to set that token.
        i. Select the endpoint.
        ii. Click the “Authorization” tab (under the URL for the endpoint).
            1. **Type**: Select “Bearer Token”.
            2. **Token**: Set this to `{{your_token_name}}`.
            3. Click “Save”.

How to set the "authorization_code" in the body of a Postman request

This applies to the `{{host}}/userProfile/getRefreshToken` endpoint.

To get an `authorization_code` and set it in Postman:

1. Login to any environment (e.g. the DEV environment: https://dev-web-public-qnp.ess.hcqis.org/admin).
2. After you verify that you are trying to login through your MFA device, click the "Stop loading this page" button in the browser when you see "/callback/" in the URL (e.g. https://dev-web-public-qnp.ess.hcqis.org**/callback/**#id_token=...).
3. Copy the URL and paste it into a text editor. You will see a long URL that has some query params.
4. Search for "code" and copy the string after **`code=`** and before **`&state=`**

5. Copy and paste that string into Postman. It will look something like this:

```
{
    "authorization_code": "NOCcuYvW6jy2SqP6ok4uB52LhtMiQzt-IzsjnbeYZO4"
}
```

NOTES: You can only use the `authorization_code` once and then it will expire. So...

- If you don't stop the page from loading when you see "/callback/" in the URL, then you will need to log out and try again.
- If you need to send another Postman request that includes the `authorization_code`, then you will need to log out and go through this process again from the beginning.

If you are testing the `{{host}}/userProfile/getRefreshToken?devLocal=true` endpoint in Postman (notice the `?devLocal` query param), then you can also follow these steps to get the `authorization_code`:

1. Open the `ess-hcms-public-ui` repo and go to the `/src/app/callback/callback.component.ts` file.
2. In that file, look for this line: **`const refreshPromise = this.userService.getRefreshToken(code);`**
3. Insert **`debugger`** above that line.
4. Run the Public UI locally and open DevTools.
5. Login at http://localhost:4200/admin. DevTools should open the "Sources" tab automatically and stop where you placed **`debugger`**.
6. In the right panel you should see a variable named "code" underneath one of the "Block" sections.
7. Copy that value for "code" and paste it into Postman.

How to set the "refresh_token" in the body of a Postman request

This applies to the `{{host}}/userProfile/renewToken` endpoint.

To get a `refresh_token` and set it in Postman:

1. Follow the instructions to set the `authorization_code` (above).
2. The response from the `{{host}}/userProfile/getRefreshToken` endpoint will be a JSON object that contains a "refresh_token" property.
3. You will need to use the value from that "refresh_token" property in your Postman request.

Postman Tutorials

- Postman Overview

API Data Validation

The `ajv` library is used for data validation with the API endpoints.

API Testing

You can run the test script for your app cluster API with `npm run test`. There are a lot of tests in the `/src/tests/` folder and you might not be able to see the results of your tests. However, you can run only the tests in your suite (or individual tests) with this command:

```
npm run test -- -g 'name of suite or test'
```

If you need to debug any `nock` mocks, you can set the `DEBUG` environment variable to `nock.*` as described here https://github.com/nock/nock#debugging. This will log each step of the matching process.

Run this in your terminal first: `export DEBUG=nock.*`

Then you can run `npm run test` or `npm run test -- -g 'name of suite or test'` to see the detailed matching output.

You can use test-driven development (TDD), if you are comfortable with that, but TDD is not required.

For API stories, you only need to create unit tests. You do not need to create integration tests or end-to-end tests.

API testing tutorials & resources

1. Search for something like “mocha and chai test driven development tutorial”.
2. Mocking External HTTP Requests in Node Tests with Nock
3. Test a Node RESTful API with Mocha and Chai (including Chai HTTP). This lists many assertions that you can use with the Chai HTTP library.
4. API mock testing with Nock
    a. Also look under the heading “Clearing mocks and blocks” to understand how to clear mocks after each test.

Libraries used for API unit testing

- Mocha: Testing framework.
- Chai: Assertion library.
- Nock: Mocking library.
- Istanbul (with nyc CLI): We use Istanbul and nyc for test coverage reports. Istanbul is a JavaScript code coverage tool that works by analyzing your codebase and letting you know where unit tests are needed in your app. nyc is Istanbul’s command line interface.

API Unit Test Requirements

1. Unit tests need to have 80% code coverage.
2. Unit tests only need to be written for app cluster APIs (i.e. the “passthrough” endpoints). We still need to figure out how to write tests that actually work for the data cluster APIs.
    a. Unit tests do not work with the data cluster APIs right now, so you don’t need to write unit tests for the data cluster APIs right now. Pramod understands this situation, so if you have questions you can ask him about this.
3. Make sure that the story points for your story include an estimate of time/effort for unit testing as well.

API unit testing notes

- Unit tests, including mocks, are located inside the `/src/tests/` folder.
- Use the necessary libraries (detailed above) for your unit tests.
- Use the red, green, refactor cycle when developing your tests.
- Tests should provide specific inputs (e.g. payload, route parameters, user inputs) and assert on the expected outputs.
- Remember the Setup, Execute, Verify, Teardown phases for each test. Or Arrange, Act, Assert.
    - When necessary, use `before*` and `after*` hooks for setup and teardown.
- In order to create unique test suite names, you might want to use this naming convention:
    - `'HTTP_Method /route/as/defined/in/the/routes.js/file'`
    - For example: `'POST /content/searchContentObjectsForPage'`
- The app cluster APIs check for an `Authorization` header (the data cluster APIs do not). So when you run tests for an app cluster API, you will need to set the `Authorization` header and a `Bearer` token. That token has to include your system admin role. If you look at the other tests in the file, they should have a `Bearer` token that you can use in your test scripts.
- Since app cluster APIs will be the only APIs that have unit tests, you need to mock the data cluster APIs that will be called by the app cluster APIs. See the example below.
    - How do I use mocks in unit tests and still have confidence that the code/system is being tested accurately? Unit tests are intended to only test small parts of an app. Mocks are the mechanisms that are used to isolate your unit tests to only a small part of an app. So, although unit tests can give you some level of confidence, they are not intended to give you the same level of confidence as integration tests or end-to-end tests.

Example of data cluster API mocking:

```
// Define the expected response body.
const responseBody = [
  { "contentId": "6ah805690795ae00iefj", "contentName": "Content Name 1" },
  { "contentId": "3b294bfc1543e8003d29", "contentName": "Content Name 2" },
  { "contentId": "4k9376b61543e8002k09", "contentName": "Content Name 3" },
];

// Define the data cluster URL that should be mocked when it is called during the test.
nock(contentSubmissionUrl)
  .post('/content/searchContentObjectsForPage/')
  .reply(200, responseBody);
```

API Test Coverage Reports

1. When you run `npm run test`, an `lcov` report will be created that shows the number of lines of code that are covered by tests.
2. You can view the test coverage report by opening the `/coverage/icov-report/index.html` file in VSCode…
    a. Right-click on the `index.html` file and select “Reveal in File Explorer”.
    b. Double-click the `index.html` file inside File Explorer to open it in a browser.
    c. NOTE: If you are using WSL2, then “Reveal in File Explorer” will not work. You can do this instead:
        i. Create a bookmark in your browser with this URL: `file://wsl.localhost/Ubuntu`
        ii. Right-click the `index.html` file and select “Copy Path”.
        iii. Open a browser tab, click your bookmark that you just created, remove the trailing slash (because the browser will insert a trailing slash), and paste the path that you copied.
        iv. Press Enter and you should see the `lcov` report.
    d. There is an “All files” heading at the top of the page. The test coverage percentage is the first percentage underneath that heading.

Swagger API Documentation

The Swagger API is in the Public API repo in GitHub.

1. Create a new branch that is named after your story. (See the Step 2: Update your local integration branch and Step 3: Create your feature branch.)
2. Change the port that the Swagger docs will run on in the `/bin/www` file. Since the app cluster API is probably running on port `3000` and the data cluster API is probably running on port `3001`, you could probably use port `3002`.
3. Run the Public API repo locally: `npm run dev`
4. To see the Swagger UI locally, open a browser window and go to `http://localhost:<PORT>/public/api-docs/qnp/` (e.g. `http://localhost:3002/public/api-docs/qnp/`).
5. In the Swagger UI, find the section where your API belongs (e.g. QualityNet Portal (QNP) – Admin).
6. Then open the `/swagger/qnp/swagger.json` file and do a search for the section title (e.g. "QualityNet Portal (QNP) – Admin"). You should see a field like this: `"name": "QualityNet Portal (QNP) - Admin"`, but you need the next field, which looks like this: `"tags": ["QualityNet Portal (QNP) - Admin"]`. That field is associated with the endpoints that have been documented already.
7. Find an endpoint that is similar to the one you need to document. Copy and paste that endpoint where you want to place your new endpoint documentation.
8. Edit your endpoint documentation until it is finished.
9. Test your documented endpoint in the Swagger UI. Swagger allows you to test your endpoint directly in the Swagger UI to make sure that your documentation is accurate.
    a. In the `/swagger/qnp/swagger.json` file, find the `"servers"` array.
    b. Copy the first object in that array and paste it at the beginning of the array.
    c. Change the `url` property of your pasted object to the same URL where your app cluster API is running (e.g. `http://localhost:3000`). You can change the `description` property to something that makes sense (e.g. Local Development Server), but that is not required.
    d. Make sure that your app cluster API is running (`npm run dev`). If you are developing a data cluster API along with your app cluster API, then make sure that your data cluster API is also running.
    e. In the Swagger UI, find the "Servers" select box at the top of the page and select the `localhost` server that you just configured.
    f. Click the "Authorize" button to the right of that select box.
    g. In the popup window that appears, paste a Bearer token inside the "Value" input field. (You can use the Bearer token that you use in Postman.) Click "Authorize" then "Close".
        i. NOTE: You can change the token by clicking the "Logout" button in the popup window and then following the same steps to enter a token.
    h. Your API endpoint in the Swagger UI has a button labeled "Try it out". Click that button, then scroll down a little and click the blue "Execute" button.
    i. If everything was documented correctly, then you should see the same response that you see in Postman.

Working on UI Stories

Things To Know

UI error handling notes

- Errors are handled inside the component class methods. Each method should use an error block to handle the error and provide a toast message for user feedback. The `ngx-toastr` package is used to display toast messages. You can search the `ess-hcms-public-ui` repo for "ngx-toastr" to see examples of toast messages.
- Note that rxjs has deprecated the old `subscribe()` method arguments and we are now encouraged to use the new format where you pass an object with `next`, `error`, and/or `complete` keys to the `subscribe()` method. See the last example on this page for details: https://rxjs.dev/deprecations/subscribe-arguments.
- If a service returns an error, then the error should be returned to the component's method that called the service. The error block will then handle the error that is returned by the service.
- We do NOT use a remote logging service, so there is no need to log UI errors.

Angular Tutorials

- Tour of Heroes application and tutorial
- Angular for React Developers: Differences and Component API comparison

How to run Angular for local development

There are 2 repos that you will need to use:

1. The Public UI repo (which is the main app): https://github.com/ccsq-ess/ess-hcms-public-ui
2. The Common UI repo (which is the UI component library): https://github.com/ccsq-ess/ess-qnp-common-ui

Step 1: Setup your local environment

Make sure to have Node.js installed in your local environment. Angular requires Node.js version 12.x or later.

Install Angular CLI globally with `npm install -g @angular/cli`

Step 2: Setup the Common UI repo (optional)

NOTE: This step is only necessary if you are doing work in the Common UI repo. Otherwise, you can skip this step.

Clone the Common UI repo

If you have not already cloned the Common UI repo to your local computer, then do that now.

Update your cloned Common UI repo

If you have already cloned the Common UI repo before and you are now working on a new story, then you need to make sure that your local integration branch is up-to-date. Follow the instructions in the section Update your local integration branch.

Run the Common UI library

1. In a terminal window, navigate to the Common UI's root directory.
2. Run `npm run watch` to start the local development environment. This command will keep watching for any changes in the source files and rebuild the library.

Step 3: Setup the Public UI repo

Clone the Public UI repo

If you have not already cloned the Public UI repo to your local computer, then do that now.

Update your cloned Public UI repo

If you have already cloned the Public UI repo to your local computer and you are now working on a new story, then you need to make sure that your local integration branch is up-to-date. Follow the instructions in the section Update your local integration branch.

Configure your local Public UI repo to use your local Common UI library (optional)

NOTE: This step is only necessary if you are doing work in the Common UI repo. Otherwise, you can skip this step.

In the Public UI repo…

1. Open the `package.json` file and find the line with `"@qnp/qnp-common"`.
2. Change its value from `"@qnp/qnp-common": "0.0.xx"` to `"@qnp/qnp-common": "file:../ess-qnp-common-ui/dist/qnp-common"` . This will point to the local build directory of your Common UI library and will create a symlink to your local copy of the Common UI library when you run `npm start`.
    a. NOTE: Make sure that the path that points to the `ess-qnp-common-ui` library (e.g. `"file:../ess-qnp-common-ui/dist/qnp-common"`) is correct and relative to the repo location on your local computer.
3. Now that your app is using your local copy of the `"@qnp/qnp-common"` library, any changes you make to the Common UI library will be reflected in your local Public UI app.

Update dependencies

1. In a terminal window, navigate to the root directory of your Public UI repo.
2. Run `npm install` to install the required dependencies from the `package.json` file.

NOTES:

- If you recently ran `npm install` ***before*** you updated the value of `"@qnp/qnp-common"` to point to `"file:../ess-qnp-common-ui/dist/qnp-common"`, then you will have to run `npm install` again in your Public UI so its dependencies will include your local copy of Common UI.
- You might need to run `npm install --force` instead of `npm install` to get it to work.

Step 4: Run the Public UI app

Run the Public UI app locally using `npm start`. The app will be available at http://localhost:4200 by default. However, you might need to make some additional changes if you are working on a UI story that is paired with an API story:

Run code for both a UI story and an API story

If you are working on a UI story that is paired with an API story, then requests should be sent to your local API code. This is how you configure that:

In the `*.service.ts` files you will see two URL variables:

1. One that points to the API Gateway (`this.config.apiGateway`)
2. One that points to your local API code (http://localhost:3000).

Make sure to comment out the API Gateway variable and uncomment the `localhost` variable so your requests will go to your local API code.

Once that is done, then you need to run your API code locally first, then run your UI code locally.

Run code for a UI story only

If you are working on a UI story only (i.e. there is no API story that is paired with the UI story), then requests should be sent to the API Gateway URL.

You don't have to make any other changes in order for your UI code to work. If you get errors with any requests (e.g. data does not load), then make sure that the API Gateway variable is being used in your `*.service.ts` files (see the explanation in the section above).

Step 5: Test a local login

Go to http://localhost:4200/admin and try logging in using your TEST credentials to make sure everything is working as expected.

UI Testing

You can use test-driven development (TDD), if you are comfortable with that, but TDD is not required.

Testing Tutorials

1. Angular Testing (with Karma & Jasmine)
    - HTTP client - Test requests
2. Karma JS Testing: What, Why, and How to Get Going Right Now
3. Test the Component logic (and Services) using SpyOn
4. Angular Unit Testing : Jasmine & Karma. This tutorial shows how to use Karma and Jasmine in the browser.
5. Jasmine Cheatsheet

Libraries used for UI unit testing

- Karma: Test runner.
- Jasmine: Testing framework.

UI Unit Test Requirements

1. Unit tests need to have 80% code coverage.
2. Make sure that the story points for your story include an estimate of time/effort for unit testing as well.

UI unit testing notes

- For UI stories, you only need to create unit tests. You do not need to create integration tests or end-to-end tests. That means that all function calls to the backend and their responses need to be mocked or spied.
- It looks like you only need to create tests for the component class. (See Component class testing.) In other words, you would test that data in the DOM is being changed accurately or that queries to the backend are returning the correct type of data.
- Do not spend very much time, if any, on Component DOM testing. You might get stuck for a long time trying to figure out how to do things like simulating user interactions with the screen.
    - You probably don't need to test things like a modal showing or hiding in the DOM because that could get into DOM testing, which gets pretty hairy, and it's probably not too important to test.
    - However, if you do need to test if a modal is showing or hiding in the DOM, for example, then you would test something like this: an `openModal()` function is called and that the `showModal` variable is set to `true`.
    - You would NOT try to do a DOM query for the button that opens the modal and then check if an element inside the modal is populated with actual data from the database (which gets into integration or end-to-end testing).
- Use the red, green, refactor cycle when developing your tests.
- Tests should provide specific inputs and assert the expected outputs.
- Remember the Setup, Execute, Verify, Teardown phases for each test. Or Arrange, Act, Assert.
    - When necessary, use `before*` and `after*` hooks for setup and teardown.
- In order to create unique test suite names, you might want to use this naming convention: `'#nameOfFunctionUnderTest() should…'`
- When writing tests for services, keep your tests simple. There doesn't appear to be very much documentation on the different types of assertions that you can use with the `subscribe()` method and the `HttpTestingController` testing package (i.e. `httpMock`). So you should probably just use the same assertions that the other tests in the `spec.ts` files use.
- You can run a single test by prefixing a `describe` or `it` block with an `f`, which stands for focus. So a `describe()` block would be `fdescribe()` and an `it()` block would be `fit()`. This will come in handy when you want to…
    - See the output for only your test.
    - Debug your tests. This will allow you to find the output of your `console.log()` statements without sifting through the output of all the other tests in the file.
- **IMPORTANT:** Remember to remove the `f` in front of any `describe()` or `it()` blocks before you submit your code for review.

How to run UI tests

1. Open a terminal window.
2. Navigate to the root directory of your cloned `ess-hcms-public-ui` repo.

3. Run `npm run test:dev`
4. After a few moments a Chrome window will open with your Karma test results.

How to run UI tests with WSL2

If you are using WSL2, then follow these steps to get Angular tests to work:

Installation

1. Start here: Setting Up WSL for Angular
    a. You can also reference this article: Using Graphical User Interfaces like Cypress' in WSL2
2. Follow this post to install Chrome in WSL2: Use Google Chrome in Ubuntu on Windows Subsystem Linux
3. When installing VcXsrv, you can also reference this tutorial: How to use GUI apps in WSL2 (forwarding X server).
4. Follow these instructions to set Zscaler certs. If you have not set the Zscaler certs, then you won't be able to run UI tests while Zscaler is running or if the Angular app is running.

Run Tests

1. Open XLaunch (this is the name that is given to VcXsrv after it is installed) and click Next, Next, Next, Finish to use the default launch settings (refer to the first tutorial above for details).
    a. NOTE: After you run XLaunch the first time, it does not appear that you need to run it again before you run `google-chrome` in your terminal (step 5 below). But it does not hurt anything to run XLaunch again.
2. ***IMPORTANT NOTE:*** *If you are not able to get the following terminal commands to work with the Linux Bash terminal in VSCode, then you might need to use the Linux Bash terminal inside Windows Terminal.*
3. Once VcXsrv is running, open your Linux command line and run Chrome with this command: `google-chrome`. You should see a Chrome browser open up that is running in WSL2 but displaying inside your Windows operating system. This version will look a little different and the Chrome icon in the Taskbar will have a little Linux icon on top of it.
4. Open another Linux terminal window, navigate to the root directory of your cloned `ess-hcms-public-ui` repo, and run `npm run test:dev`. After a few moments another Chrome window will open with your Karma test results and you should see the test results in the terminal.
    a. NOTES:
        i. If the tests are building and/or loading slowly, then it might be helpful to stop any code repos that are running in other terminals.
        ii. The tests appear to refresh periodically, which also reloads the browser. So be aware of that.
5. In the browser window with the Karma test results, search for your test and click on it to see the results for only that test. The terminal window where you ran `npm run test:dev` will also display errors that occur in your test scripts.
6. How to close the tests and the browser:
    a. You can close the tests by typing `Ctrl+C` in your terminal where you ran `npm run test:dev`.
    b. You can close the WSL2 Chrome browser by typing `Ctrl+C` in your terminal where you ran `goolge-chrome`.

Code Coverage Reports

1. When you run the unit tests (with `npm run test:dev`) a `coverage` folder will be created at the root of the Angular project. If `npm run test:dev` did not create a `coverage` folder, then try running `ng test --watch=false --code-coverage`
2. In VS Code, navigate to the `coverage` folder, right-click on the `index.html` file and select "Reveal in File Explorer".
3. In the File Explorer window that appears, double-click on the `index.html` file to open it in a browser where you will see the code coverage report.
4. In the code coverage report under the "File" column, click on the file that corresponds to the unit tests for your story to see the report for only your unit test(s).
5. Take a screenshot and include it in your PR.

Making changes to the Common UI library and creating PRs

The Common UI library is shared among the other UI repos (Public UI, PRS UI, Mailer UI, CRS UI). If you make edits to Common UI, then there are a few things that you have to do before creating a PR.

1. Follow the steps under How to run Angular for local development to setup Common UI for local development.
2. After you make your changes to Common UI, you need to run the unit tests in each of the UI repos to make sure everything still works. NOTE: At the time of this writing tests are not setup in Common UI. So if you run unit tests in Common UI and you get strange behavior (e.g. your terminal displays a screen of scrambled lines instead of output for unit tests), then you do not need to run any Common UI unit tests.
    a. You need to have a local clone of each UI repo.
    b. For each UI repo, open the `package.json` file and find the line with `"@qnp/qnp-common"`.
    c. Change its value from `"@qnp/qnp-common": "0.0.xxx"` to `"@qnp/qnp-common": "file:../ess-qnp-common-ui/dist/qnp-common"`. This will point to the local build directory of your Common UI library and will create a symlink to your local copy of the Common UI library.
        - NOTE: Make sure that the path that points to the `ess-qnp-common-ui` library (e.g. `"file:../ess-qnp-common-ui/dist/qnp-common"`) is correct and relative to the repo location on your local computer.
    d. In Common UI, you need to run the build command (`npm run build`) to build the latest code into the `dist/qnp-common` folder. However, that is probably not necessary since you already did the same thing essentially when you ran the build/watch command (`npm run watch`) while you were making your edits.
    e. In the UI repo, if you have not done so already, run `npm install --force` to install the dependencies in the UI repo with the `"@qnp/qnp-common"` package pointing to your local copy (e.g. `"file:../ess-qnp-common-ui/dist/qnp-common"`).
    f. Once the UI repo is pointing to your local copy of the `"@qnp/qnp-common"` library, you can run `npm run test:dev` in the UI repo and any unit tests that use the Common UI library should reference your local Common UI changes.
3. In the Common UI library repo:
    a. You need to bump the version up by 1 in both the `package.json` and `projects/qnp-common/package.json` files.

    b. Create a PR for your Common UI changes.
        i. NOTE: You should not include the `package-lock.json` file in your PR. You just need to update the version number in the pa ckage.json and `projects/qnp-common/package.json` files and make sure to include those files in your PR.
    c. After your Common UI PR has been approved, you should merge it. This will merge your Common UI PR into `develop`.
4. Now you need to deploy the new version of Common UI.
    a. Go to https://qnetjenkins.cms.gov/ess-master/job/common/job/ess-qnp-common-ui/ and press the "Build Now" button in the left column.
    b. You should see the build process start. If it passes, then continue to the next step. If it fails, then fix the errors and try again.
5. After the deployment is complete, then you need to determine if the Common UI version should be updated in all the UI repos or only the UI repo (s) that were involved in your story. If you need help figuring this out, then you can ask in the Pajama Ninjas channel in Microsoft Teams.

    a. NOTE: Recent NGINX updates have automated the process for updating the Jenkinsfile Program Increment (PI) version number, so we no longer need to manually update the PI number after the deployment is complete. However, this is the process that we used to follow, which is kept here in case it is ever needed: After the deployment is complete you will be sent a Jenkinsfile Program Increment (PI) version number. Update the PI version number in each UI repo. Refer to this PR for an example: https://github.com/ccsq-ess/ess-hcms-public-ui/pull/952/files.
6. For each UI repo that needs to reference the new Common UI version:

    a. Prepare your feature branch to merge back into the develop branch. NOTE: Before you update the dependencies, make sure that the `@q np/qnp-common` version number is updated in the `package.json` file so it matches the new version that is now deployed.
    b. Create a pull request. Make sure to include the `package.json` file in your PR, but not the `package-lock.json` file. A new `package-lock.json` file will be generated during the Jenkins build pipeline.
    c. After creating your pull request, follow the remaining steps until your story is complete.

Git Branch Update Methods: Merge & Rebase

Merge Approach

1. **Fetch Remote Changes:** `git fetch origin develop`
    - Fetches updates from `develop` branch.
2. **Merge into Local Branch:** `git merge FETCH_HEAD`
    - Merges the fetched changes into your branch.
3. **Push to Remote:** `git push`
    - Pushes the merged changes to remote.

Rebase Approach

1. **Fetch Remote Changes:** `git fetch origin develop`
    - Fetches updates from `develop` branch.
2. **Rebase Local Branch:** From your feature branch, run `git rebase origin/develop`
    - This will reapply your feature branch changes on top of the remote `develop` branch.
3. **Force Push to Remote:** `git push --force-with-lease`
    - Safely force-pushes the rebased branch.

How to handle merge conflicts

Git provides hints to resolve merge conflicts

If you run into code conflicts, then Git will provide you with some hints in the terminal as to what you should do after you have resolved all the conflicts. For example:

```
Resolve all conflicts manually, mark them as resolved with "git add/rm <conflicted_files>"
```

You can also run `git status` at any time to find out what you still need to do.

Tips for handling merge conflicts in VS Code

The EXTREMELY helpful guide to merge conflicts

How to handle merge conflicts with lock files

If there are merge conflicts with a lock file, then that can get really complicated to resolve. It might be best to accept all the code in the develop branch's lock file and then after the merge has been completed you can delete the lock file and regenerate a new one (with `npm install`) that takes into account all of the packages that are being used after the merge. But you should ask your team lead if they have a better approach to handling merge conflicts in lock files.

What does HEAD represent in a merge conflict?

If you have a merge conflict, VS Code will look like this:

```
<<<<<<< HEAD (Current Change)

I love branches

=======

Branches rock!!!

>>>>>>> develop (Incoming Change)
```

The changes under `HEAD` represent the changes in the branch that you are currently switched into. For example, if you are switched into the `feature-a` branch and you run `git merge develop`, then the changes under `HEAD` represent the changes in the `feature-a` branch.

In VS Code, should you select "Accept Current Change" or "Accept Incoming Change"?

That depends on whether you are performing a merge or a rebase.

In the case of a merge where you are switched inside your feature branch and you are merging `develop` into your feature branch, "Current Changes" are changes from your feature branch and "Incoming Changes" are changes from the `develop` branch.

See this StackOverflow post: GitHub: Difference between Accept current changes and Incoming changes

---

Accessibility

Familiarize yourself with Accessibility Guidelines

- W3C - Introduction to Web Accessibility
- Section508.gov - What is Section 508?
- Section 508 Standards
- WCAG Guidelines

Understanding Assistive Technologies

Learn about different assistive technologies that people use to access the web: Introduction to Web Accessibility

Accessibility Tools and Plugins

- WAVE - A suite of evaluation tools that help authors make their web content more accessible to individuals with disabilities.
- axe Accessibility Checker - A browser extension for automated accessibility testing.
- JAWS Screen Reader. Ask Steve Haering (from the security team) for access to the shared team license.
    - Download JAWS
    - JAWS Training
        - Getting Started With JAWS
        - Surf's up! Surfing the Internet with JAWS! - Web Pages
        - Surf's up! Surfing the Internet with JAWS! - Videos
    - JAWS Keyboard Commands Quick Reference Guide

Accessibility with Angular

These resources can help you understand how to make Angular apps accessible.

- Angular, Accessibility, and You
- Building Accessible Single Page Apps

How to perform a master merge

This process is basically the same as any other PR that gets merged into the `develop` branch. For more details you can refer to the instructions about PRs under the Development Workflow.

1. Once we are ready for a release, Kavitha will ping us (either in Slack or Teams) and request a master merge for the upcoming release.
2. One of the developers needs to create a PR from the `develop` branch into the `master` branch. This should be done for each repo that will be included in the release.
    a. On the “Compare changes” page, make sure that the “base” branch is `master` and the “compare” branch is `develop`.
    b. The PR’s “Title” field can be "master merge X.X.X release" (replace the X.X.X with the release number).
3. After the PR is created, we typically ping the other developers (in the Pajama Ninjas channel) who have also contributed code to the repo for the upcoming release and ask them to verify if their stories have been included in the master merge PR. But this is not required.
4. Check for conflicts in the PR. If there are conflicts, then those need to be resolved before the PR can be merged.
5. The PR needs to be approved by at least one other developer before it can be merged. If you created the PR, then you will need to ask another developer to approve it. If someone else created the PR, then you might be asked to approve it (and maybe even merge it).
6. Once the PR has been approved and merged into `master`, then:
    a. Respond to Kavitha's ping to let her know which PRs have been merged.
    b. The master build pipeline does not automatically build, so you need to let Sharath Kamarapu know that the repos that are part of the master merge are ready for test deployment. For example, you can post something like this in the Pajama Ninjas channel: Sharath Kama rapu, Public-UI, PRS-UI and PRS-API are ready for test deployment.

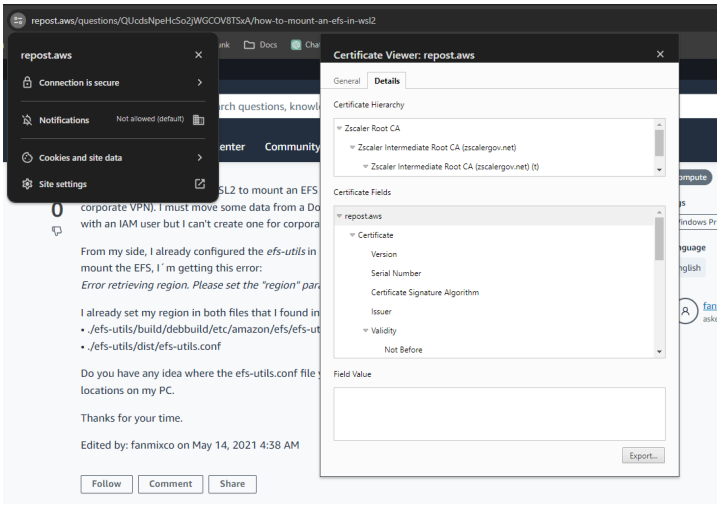
NOTES:

- Anyone can perform a master merge, but typically one of the developers who has worked on a story that will be included in the release should create the PR for their repos into `master`. For example, if I am one of the developers (or the only developer) who has contributed code to the Public UI repo for the upcoming release, then I can/should handle the master merge for the Public UI repo.

WSL2 Tips

Set Zscaler certificates (prevent the need to log out of Zscaler when installing software or running apps)

1. Get the following cert files: **Zscaler Root CA.crt** and **Zscaler Intermediate Root CA (zscalergov.net).crt**
    a. While connected to Zscaler, open this site in Chrome https://repost.aws/questions/QUcdsNpeHcSo2jWGCOV8TSxA/how-to-mount-an-efs-in-wsl2 (see the image below). (You might be able to open any site in Chrome, but I haven't been able to find the certs in the few sites that I tried.)
    b. Click on the "View site information" icon at the beginning of the URL bar.
    c. In the popup that appears click "Connection is secure".
    d. Click "Certificate is valid".
    e. Click the "Details" tab.
    f. Inside the "Certificate Hierarchy" box, select "Zscaler Root CA" and click the "Export..." button.
    g. Inside the "Certificate Hierarchy" box, select "Zscaler Intermediate Root CA (zscalergov.net)" and click the "Export..." button.
    h. Those certs should now be in your Windows "Downloads" folder.
2. Move those cert files to your `/usr/local/share/ca-certificates` folder in WSL2. This should be done from within WSL2 otherwise you will run into permissions errors.
    a. Open a WSL2 terminal. (You can use Windows Terminal for this.)
    b. Navigate to your `ca-certificates` folder: `cd /usr/local/share/ca-certificates/`
    c. Move the **Zscaler Root CA.crt** cert (note the period at the end of this command): `sudo mv /mnt/c/Users/<your_user_profile>/Downloads/Zscaler\ Root\ CA.crt .`
    d. Move the **Zscaler Intermediate Root CA (zscalergov.net).crt** cert (note the period at the end of this command): `sudo mv /mnt/c/Users/<your_user_profile>/Downloads/Zscaler\ Intermediate\ Root\ CA\ \(zscalergov.net\).crt .`
3. Your Zscaler certs should be set now!



S3 Error: The difference between the request time and the current time is too large

If you get this error when trying to access S3, then you can follow the steps from this StackOverflow answer, which are specific to Ubuntu (in WSL2): https://stackoverflow.com/a/66259987/23067953

How to take time off (PTO)

1. Create a new meeting invite in Outlook.
    a. In the left column of Outlook, click on the “Calendar” icon.
    b. Under the “Home” tab, click “New Meeting”.
    c. In the “New Meeting” window that pops up:
        i. Set “Show as” to "Free".
        ii. Update "Title" as “Your Name – OOO” (e.g. John Smith – OOO).
        iii. On the “Required” line, add the new Team Calendar email address: ESS_Team_Calendar@ventera.com
        iv. On the “Optional” line, add any other contacts that should be notified.
        v. Set the dates and times that you will be out of the office.
        vi. On the “Location” line, you can remove any locations that are listed, including any Microsoft Teams Meeting links.
2. Send the invite.

The robotic process automation (RPA) built for the QuickSight integration will run and identify new meeting invites that have been sent to the team calendar. The RPA will accept the meeting invite and send you a confirmation email and will also update QuickSight to reflect your out of office time.

NOTE: You no longer need to complete the CSV file. You can now add the team calendar to your OOO meeting invite in Outlook.

Logging PTO and Submitting Timesheets in Advance

You can log PTO in advance in Unanet. If you are going to be out on PTO when your timesheet is due, then you can also submit your timesheet before you leave.

We have also been encouraged to download and utilize the Unanet app to help ensure that our PTO gets logged and our timesheets are submitted on-time.

Unanet GovCon Mobile App

Google Play: https://play.google.com/store/apps/details?id=com.unanet.unanet&hl=en_US&pli=1

App Store: https://apps.apple.com/us/app/unanet-govcon/id1533322144

FAQ

Q: If my OOO spans over two weeks should I send multiple invites, one for each week?

A: No, you do not need to send multiple invites. For example, if I am out Thu-Wed of the next week, then one invite showing my start time as Thu and end time as Wed is ok. It's also fine if your OOO spans multiple months. Just create one invite for the dates you are out, and you're all set.

Q: What if I am out for a half day?

A: You can still send the invite to the new calendar. QuickSight will show that you are out for the entire day as it does with the CSV file.

Q: How do I change or cancel my OOO?

A: As you normally would, make the updates you need to and be sure to send the updated event. If cancelling, select the send cancellation option. The updates you make will be processed and reflected in QuickSight.

Q: What are the meeting acceptance notifications?

A: When you add or update an event, you will eventually receive a notification that the meeting was accepted by the team calendar. This is just an indicator that your meeting was processed and added to QuickSight.

Q: Am I able to view this calendar?

A: Yes, you may add the calendar and view upcoming OOO events if you would like. The calendar can be added in your Outlook client from the Address Book (search for "ESS Team Calendar").