Need-To-Know for

pal development

Stacey Beard

Git Version Control

Branching

- Opal development is organized in the following development stages:

Personal branch	Team branch	Staging	PreProd	Prod
• Used by one person for development.	• **Optional** • If working in a team, can be used to merge everyone's personal branches to test them together.	• Once a project is done and ready to be included in Opal, it's merged into staging. • Staging is tested by running the code in a browser.	• Once staging has been tested, it's merged into PreProd. • PreProd is built into the Opal-dev app, then tested on mobile devices. • Only developers have access to this app.	• Once PreProd has been tested, it's merged into Prod. • Prod is built into the Opal app, then tested on mobile devices. • After testing, a new version of Opal is released to patients.

Branching

- We use the following branch names for each development stage:

Personal branch	Team branch	Staging	PreProd	Prod
Your project name, followed by an underscore, followed by your first name, all in lowercase e.g.: **questionnaires_anna**	The project name, in lowercase e.g.: **questionnaires**	Branch name: **staging** (qplus and opal-listener)	Branch names: **opal_pre_prod** (qplus) **PreProd** (opal-listener)	Branch name: **master** (qplus and opal-listener)

Your Personal Branch

- During your opal project, you will only be working in the **Personal Branch** and **Team Branch** stages (don't touch the opal_pre_prod or master branches). You won't need a team branch if you're the only one working on your project.
- Create a new personal branch:
  - Branch off of opal_student (to start a new project) or an existing project branch (to continue an existing project)
  - Branch name: your project name, followed by an underscore, followed by your first name, all in lowercase
  - e.g.: questionnaires_anna

Commits

- Make commits to your branch after each logical change.
- To keep your code safe, push your branch to the origin (using the same branch name) regularly.

Commit style:

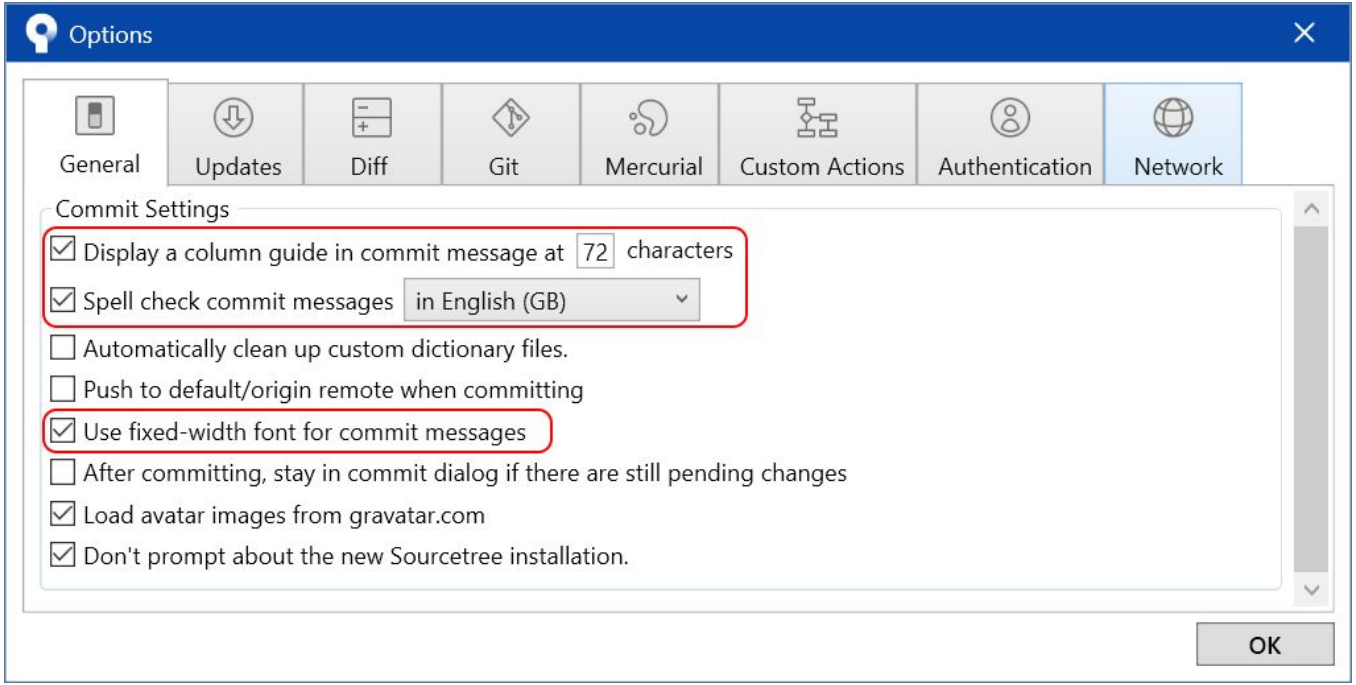
Make informative commits, using good commit style. A new student or developer who continues your work later should be able to understand what you've done.
Follow this commit guide: *How to Write a Git Commit Message*

Sourcetree Configurations

- Sourcetree can be configured to help you write commit messages

Go to **Tools** > **Options** and scroll down

Select these options



Committing

- **Always double-check your code changes before commiting.**
- A GUI like SourceTree can help you do this.

Pending files, sorted by file status

Search

Staged files | Unstage All | Unstage Selected

www/js/controllers/auth/logOutController.js

www/js/controllers/educational/educationalMateria...

Unstaged files | Stage All | Stage Selected

www/js/controllers/home/homeController.js

www/js/controllers/loading/loadingController.js

www/js/controllers/tabs/generalTabController.js

www/js/controllers/tabs/personalTabController.js

www/js/services/educationalMaterialService.js

www/js/services/hospitalsService.js

www/js/services/labResultsService.js

www/js/controllers/tabs/generalTabController.js

Hunk 1 : Lines 12-29 | Stage hunk | Discard hunk

```
12  12          .module('MUHCApp')
13  13          .controller('GeneralTabController', GeneralTabController);
14  14
15      -       GeneralTabController.$inject = ['$scope', 'Announcements', 'Upd
    15  +       GeneralTabController.$inject = ['$scope', 'Announcements', 'Upd
    16  +           'MetaData', 'UserPreferences', 'Hospitals'];
16  17
17      -       function GeneralTabController($scope, Announcements, UpdateUI, 
    18  +       function GeneralTabController($scope, Announcements, UpdateUI, 
    19  +                                     MetaData, UserPreferences, Hospit
18  20              var vm = this;
19  21
20  22              vm.goToPatientCharter = goToPatientCharter;
21  23              vm.goToParking = goToParking;
22  24              vm.generalDeviceBackButton = generalDeviceBackButton;
23  25              vm.goToUrl = goToUrl;
    26  +           vm.enableModule = enableModule;
24  27
25  28              activate();
26  29
```

Hunk 2 : Lines 129-137 | Stage hunk | Discard hunk

```
126 129
127 130         }
128 131
```

Stacey Beard <staceylynn800@hotmail.com>

Commit options...

Push changes immediately to origin/all-in-one-stacey

Commit

Committing

- Don't commit:
  - Personal configurations (except config.js)
  - Shortcuts (like hardcoded email and password)
  - Console.logs used for debugging

Don't commit this! →

www/js/controllers/loading/loadingController.js

Hunk 1 : Lines 27-33 | Stage hunk | Discard hunk

```
27  27                  UpdateUI.init()
28  28                      .then(function() {
29  29
    30  +                   console.log('STATE GO HOME');
30  31                      $state.go('Home');
31  32                      RequestToServer.sendTargetedRequestWithResponse('Sh
32  33
```

Hunk 2 : Lines 41-49 | Stage hunk | Discard hunk

```
40  41                      PlanningSteps.initializePlanningSequence();
41  42
42  43
    44  +                   console.log('HIDE LOADING MODAL');
43  45                      loadingmodal.hide();
    46  +                   console.log('CLEAR TIMEOUT');
44  47                      clearTimeout(timeOut);
45  48                  });
46  49              }
```

Stashing

- Stashing can be a great way to save code that you want to put aside for later without having to commit it.
- Read more here: Git Stash
- Sourcetree provides a nice interface for managing your stashes (they're all listed in the left menu panel).
- Always stash leftover changes before switching branches to make sure you don't lose them.

WebStorm (or equivalent IDE)

Useful features

- Webstorm offers many useful features that can make development easier. Similar IDEs may have similar features, using different key combinations.

Features

- Global search in a project. You can specify a custom scope to exclude files in the 'lib' folder.
- CTRL+Click (⌘+Click) on a function call to open the function in the file where it's defined.
- CTRL+Shift+N (Shift+⌘+O) to search for a file by name.
- And many more.

Terminal

- When running your local copy of the app, you can open your two projects (qplus and opal-listener) in two separate windows. Each will have a terminal where you can run the code.

qplus

File Edit View Navigate Code Refactor Run Tools VCS Window Help

opal-listener > listener > server.js

Project

- opal-listener D:\Opal\Student
  - listener
    - api
      - request
      - response
      - apiHospitalUpdate.js
      - apiPatientUpdate.js
      - main.js
      - processApiRequest.js
      - sqlInterface.js
    - cron
    - logs

server.js × | config.json × | logger.js ×

```
1   /*  Filename     :   server.js...
19
20  const mainRequestApi     = require(
21  const processApi         = require(
22  const admin              = require(
23  const utility            = require(
24  const q                  = require(
25  const config             = require(
26  const logger             = require(
27  const cp                 = require(
28
29  const FIREBASE_DEBUG = !!process.
30
31  /*****************************
```

processRequest()

Terminal

```
D:\Opal\Student Install\opal-listener>cd listener

D:\Opal\Student Install\opal-listener\listener>node server.js
```

6: TODO | Terminal | 9: Version Control | Event Log

opal-listener

File Edit View Navigate Code Refactor Run Tools VCS Window Help

qplus > www > index.html

Project

- lib
- test
- views
  - education
  - general
  - home
  - init
  - login
  - navigators
  - personal
  - settings
  - tabs

index.html ×

```
1   <!DOCTYPE html>
2   <html lang="en" >
3   <head>
4       <title>OPAL</title>
5
6       <meta charset="utf-8">
7       <meta http-equiv="X-UA-Compatible"
8
9       <!--
10      Caching Meta
11      ==========================
12      <meta http-equiv="cache-control"
13      <meta http-equiv="cache-control"
14      <meta http-equiv="expires" content
```

Terminal

```
D:\Opal\Student Install\qplus>gulp serve
```

TERMINALS

6: TODO | Terminal | 9: Version Control | Event Log

Merge tool

- WebStorm has a tool to help you fix conflicts when merging branches.
- Read this to learn how to use it: [WebStorm - Resolving Conflicts]

Merge Revisions for C:\IdeaProjects\help-sources\intellij-platform\topics\Install_and_set_up__product_.xml

Highlight words

3 changes. 1 conflict

Local Changes (Read-only) | LF | Result | CRLF | Changes from Server (revision cf342407e563428a6a781d1abbbe3f7a5... LF

Accept Left Side
Accept Right Side
Resolve using Left
Resolve using Right
Resolve Automatically
Ignore
Compare with Clipboard
Annotate

Ignore
Ctrl+click to resolve conflict

Accept
Ctrl+click to resolve conflict

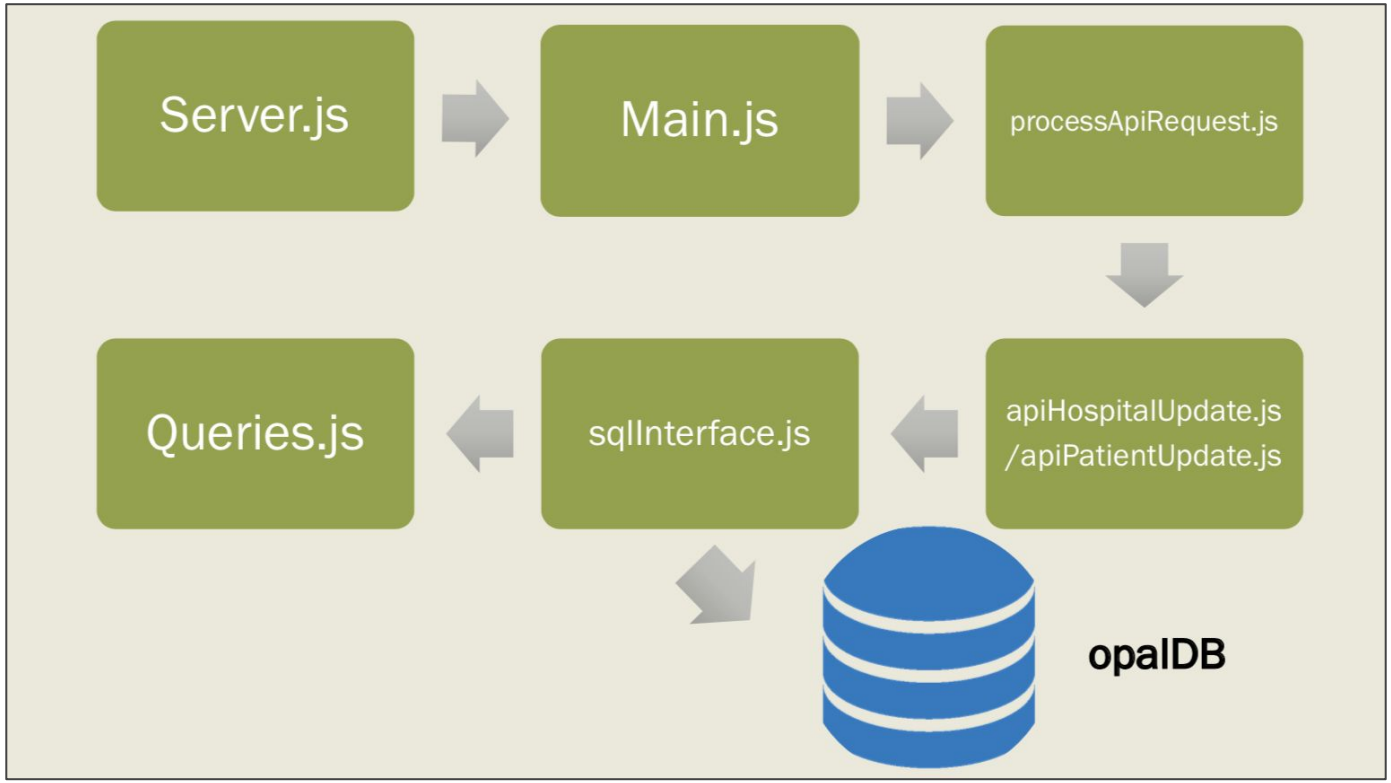
Accept Left | Accept Right | Apply | Abort

Opal Listener

Information Flow

- To understand how the listener works, start at `server.js` and start reading through the code.
- `server.js` is the file that runs when starting the listener with the command `node server.js`

Information Flow



API

- All requests are listed in `processApiRequest.js`
  - `API` contains all authenticated requests (made after the patient has logged in)
  - `securityAPI` contains all unauthenticated requests (made when the patient is not logged in)

Authenticated

```
const API = {
    'DeviceIdentifier': apiHospitalUpdate.updateDeviceIdentifier,
    'Log': apiPatientUpdate.logActivity,
    'LogPatientAction': apiPatientUpdate.logPatientAction,
    'Login': apiPatientUpdate.login,
    'Logout': apiHospitalUpdate.logout,
    'Resume': apiPatientUpdate.resume,
    'Refresh': apiPatientUpdate.refresh,
    'AccountChange': apiHospitalUpdate.accountChange,
    'CheckCheckin': apiPatientUpdate.checkCheckin,
    'Checkin': apiHospitalUpdate.checkIn,
    'CheckinUpdate': apiPatientUpdate.checkinUpdate,
    'DocumentContent': apiPatientUpdate.getDocumentsContent,
    'Feedback': apiHospitalUpdate.inputFeedback,
    'LabResults': apiPatientUpdate.getLabResults,
    'MapLocation': apiPatientUpdate.getMapLocation,
    'Message': apiHospitalUpdate.sendMessage,
    'NotificationsAll': apiHospitalUpdate.getAllNotifications,
    'NotificationsNew': apiHospitalUpdate.getNewNotifications,
    'EducationalPackageContents': apiPatientUpdate.getPackageContents,
    'Questionnaires': apiPatientUpdate.getQuestionnaires,
    'QuestionnaireRating': apiHospitalUpdate.inputEducationalMaterialRating,
    'QuestionnaireAnswers': apiHospitalUpdate.inputQuestionnaireAnswers,
    'Read': apiHospitalUpdate.updateReadStatus,
    'PFPMembers': apiPatientUpdate.getPatientsForPatientsMembers,
};
```

Unauthenticated

```
exports.securityAPI = {
    'PasswordReset': security.resetPasswordRequest,
    'SecurityQuestion': security.securityQuestion,
    'SetNewPassword': security.resetPasswordRequest,
    'VerifyAnswer': security.resetPasswordRequest
};
```

Adding a New Request

- When working on an Opal project, you will likely need to add new requests to the listener.
- To add a new request:
  - Give it an appropriate name, and add it to the API in `processApiRequest.js` (most of the time, you won't need to use the security API).
  - Create a function in `apiHospitalUpdate.js` or `apiPatientUpdate.js` for your request.
  - Create one or many functions in `sqlInterface.js` for your request. This is where the bulk of the request work gets done.
  - Add any DB queries to `queries.js`.

Adding a New Request

- After these steps, link all the functions together by using the same data flow as the existing Opal requests
  - `processApiRequest.js` calls `apiHospitalUpdate.js` or `apiPatientUpdate.js`, which call `sqlInterface.js`, which uses the function `runSqlQuery` to run queries in `queries.js`
- Your best bet is to imitate what is being done in the other Opal requests.

Do's and Don'ts of New Requests

Do

- Add new requests when needed
- Edit new requests that were created as part of the project you're working on (by previous contributors)
- Use parameters to make the most of a single request
- Split the requests you want to create into logical, single-purpose requests

Don't

- Edit existing requests (talk to an Opal team member first if you think it's necessary)
- Delete existing requests
- Make many separate requests that do similar things (use parameters instead)
- Make one large request that does many different things

Debugging in the Listener

- Debugging in the listener can be challenging.
  - The console outputs hundreds and hundreds of long lines.
  - Sometimes, what you want to look at gets cut off the top of the terminal because the output is too long.
- Solution: use the listener logs to send all output to a file.

Logger.js file →

logger.js

```
17  const levels = {
18      error: 0,
19      warn: 1,
20      info: 2,
21      verbose: 3,
22      debug: 4,
23  };
24
25  winston.configure({
26      transports: [
27          // Output all debug events and higher to console
28          new (winston.transports.Console)({
29              level: 'debug',
30              json: false,
31              timestamp: true
32          }),
33          // Store all ---debug--- events and higher to the opal.log file
34          new (winston.transports.File)({
35              filename: './logs/opal-info.log',
36              level: 'debug',
37              json: true,
38              timestamp: true
39          })
40      ]
41  });
```

Each logger message is similar to a console output, with the addition of a level that specifies its level of importance (error is the most important; debug is the least)

This section sends logs to an output file. Change the level to debug to send all logs (of level debug and higher, which is all of them) to a file

Important: don't commit this change!

Debugging in the Listener

- After saving and re-starting the listener, all logger messages will get sent to the file opal-info.log, in the listener's logs folder.
- CAUTION: This file will become huge very fast (hundreds of megabytes; tens of thousands of lines).
  - Built-in text editors and WebStorm can't handle big text files with long lines. I recommend using Notepad++ or a similar advanced text editor to read the log instead.
  - It's safe to delete the file whenever it gets too big (this also makes it easier to find what you're looking for); the file will simply get re-created automatically.

Debugging in the Listener

- To debug the listener:
  - Stop the listener.
  - Make the change to the logger.js file.
  - Save your changes.
  - Delete the existing opal-info.log file.
  - Restart the listener.
  - Use Opal frontend to repeat the steps that created the issue.
  - Stop the listener.
  - Open opal-info.log using an advanced text editor like Notepad++. Search through the log for errors. You may want to toggle between word-wrap and no word-wrap.

Debugging in the Listener

- If you can't find the problem, you can add logger outputs of your own to print out variables. Make sure to use JSON.stringify(obj) when printing objects.

Example of logging (first argument is the logging level, second argument is the contents to log):

```
logger.log('debug', 'results: ' + JSON.stringify(results));
```

It's good to add logger output to your code, just make sure to pick an appropriate logging level.

Opal Frontend

Requests

- Requests are sent to the app via the service "RequestToServer".
- To send a request, use the RequestToServer API:

Examples:

```
RequestToServer.sendRequest('Read',{
    'Id':serNum,
    'Field':'Announcements'
});

RequestToServer.sendRequestWithResponse('LabResults')
    .then(function (response) {
```

Gulp

- In qplus, gulp is used to automate or bundle together certain tasks.
- You can change these task by editing `gulpfile.js`.

Examples:

If the app is refreshing endlessly on a loop, set livereload to false:

```
gulp.task('connect', function() { … livereload …
```

If you want to use a different browser than chrome, edit the "open" task:

```
gulp.task('open', function() { …
```

Bower Components

- Bower components are used to provide access to third-party packages.
- Do not update or install bower components without speaking to an Opal team member first. Making changes to bower components can cause your Opal installation to stop working.
- Note: if for any reason you need to make changes to the bower components, back them up first. It's much easier to restore bower components from a zip file than it is to re-install them from scratch.

Running Two or More Instances of qplus at Once

- It's sometimes useful for development to have two instances of the app running at the same time.
- For example, you can be logged in as a patient in one instance and as a caregiver in the other, to instantly see the results of sharing data between accounts without having to log out of one and into the other.
- You can run two or more instances of qplus by using different ports:

Use gulp serve (port 9000 by default) and http-server (with port 9001)
or
Use http-server (with port 9000) and http-server (with port 9001)
(You can run as many instances of http-server on different ports as you need)

JavaScript

Undefined Attributes

- Always check that attributes you're referencing aren't undefined.

Example:

```
if (user.firstName === "Jane") // do something
```

This will crash if the user has no firstName attribute

Undefined Attributes

- There are several ways of checking if attributes are undefined

Check for falsiness:

```
if (!user.firstName) // no first name
```

Caution! This will work for strings or objects, but the number 0 will evaluate to false, which may not be the behavior you want.

Undefined Attributes

- There are several ways of checking if attributes are undefined

Check typeof:

```
if (typeof user.firstName === 'undefined') …
```

Other methods: Checking for undefined

Documentation

Opal Documentation

- Document everything you're doing during your project in a Google document. This can include:
  - Description or introduction to your project.
  - Choices that you made while designing how your project will work, and their justifications.
  - Evaluations of pros and cons.
  - GUI mockups of views.
  - Implementation details (what you did, and how).
  - List of known bugs / things that are incomplete in your project.
  - Need-to-Know for future development
  - Suggestions of future directions

Opal Documentation

Detailed documentation is the best way to guarantee that your project will live on after you.

"What do I include?" It's simple: include everything that you would want to know if you were a new person picking up your project where you left off.

End of Need-To-Know for Opal Development