TABLE OF CONTENTS

· Raspberry-Pi:-Performance-tuning.md (Raspberry-Pi:-Performance-tuning.md)

· Release-0.10.3.md (Release-0.10.3.md)

· Release-0.11.md (Release-0.11.md)

· Release-0.12.md (Release-0.12.md)

· Release-0.13.md (Release-0.13.md)

· Report-Bugs-by-Creating-Issues.md

· Securesync:-How-Cross-Computer-Syncing-Works.md (Securesync:-How-Cross-Computer-Syncing-Works.md)

· Student-Data-Records.md

· Submitting-Pull-Requests.md

· Testing-new-code.md

· The-setup-codepath.md

· The-topic-tree.md

· Tips-and-tricks.md

Backbone-styles-and-tips.md

Principles

- Views encapsulate a portion of the DOM, and handle all user interactions with that part of the DOM. Be sure to scope any jQuery references to the current view (`this.$(".have-some-class")` or `this.$el.html("...")`) to avoid cross-view interference.
- Models serve as the canonical representation of state. Avoid storing app state in the DOM, in the view itself, or in global variables. State lives in the models, and views listen to the models to update the DOM when the state they care about changes. Similarly, when a user interaction needs to change state, it should do so by setting a value on a model. Other views can then respond to the state change as needed.
- Refresh as little of the DOM as possible when state changes, within pragmatic reason. No need to redraw a large complex template if adding/removing a class, or changing some text, will suffice.

Tips

- Use `listenTo` (http://backbonejs.org/#Events-listenTo) anytime you want to bind a view to an event. That way, when you later run `my_view.remove()` , the view is not only removed from the DOM, it also has its listeners unbound to avoid "phantom listeners" later.
- Some models encapsulate persistent state (e.g. a log record that is saved to the server), while others represent temporary state to do with the user's current interactions with the interface. It's best to keep this in separate models.
- Views can have subviews. For now, we generally just keep track of these subviews as an attribute on the parent view, or as a list of views as an attribute on the parent view (if there are many of the same type). This way, we can loop over them later, e.g. to "remove" them and thereby clear their event listeners.

Styles

- Name a View class ending with `View` (e.g. `ExercisePracticeView`). Name a Model ending with `Model` (e.g. `ExerciseDataModel`). Name a Collection ending with `Collection` (e.g. `AttemptLogCollection`).

- End a View's `render` method with a `return this`, so that the view can be rendered and inserted into the DOM in a single line.
- Attach View, Model, and Collection definitions directly to the window object, as it helps to explicitly show that they're in the global namespace (e.g. `window.CurrentUnitRowView = Backbone.View.extend({...`).

Branching-and-merging-strategy.md

Big questions:

- What do all the branches mean?
- What is our schedule and flow for merging branches?

Development happens on multiple branches in parallel. We have multiple types of branches:

- The `master` branch: Publicly released, production-ready code that will be installed by end users.
- "Release" branches: Branches that are being prepared for release, but not yet public.
- "Project" branches: Branches in development for particular pre-release projects, such as partner deployments.
- The `develop` branch: Very in-progress, potentially buggy code, to be released in a longer time window.

These branches are arranged in a roughly linear sequence, to simplify our merge/release strategy: develop, project, release, master. This means that when code is ready to be moved closer to production, we do a "forwards merge" along the direction of this chain. We also do "backwards merges" on a regular basis to bring changes (e.g. hotfixes) from the more public branches back into the development branches. Backwards merges can happen any time, but at least once a week (currently scheduled for Mondays).

Choosing where to branch off from and target a pull request depends on an assessment of three factors: how urgent the fix/feature is, how risky (in terms of introducing other bugs) the required changes are, and how long it will take to develop. A good heuristic is to see how urgent the code you're committing vs. how risky or potentially breaking it is:

```
  Master     Release                Projects     Develop
<-------------------------------------------------------->
  More urgent                                  More risky
```

For example, a critical hotfix that involves a small, low-risk code change should branch from and target back to the master branch, whereas a more involved, and simply "nice to have", feature should target the develop branch.

Here is a further breakdown of the type of fixes that will go to each branch, and the general flow of the code with the implemented merge strategy:

```
Master (hotfixes)
---->\
      \    Release branches
       \   (noncritical fixes,
        \   short-term public features)
         ->--------->\
                      \
                       \   Project branches
                        \  (project-specific features)
                         ->------>\
                                   \   Develop
                                    \  (long-term public features)
                                     ->------>
```

Code-Editor-Tips.md

Here are some tips on working with KA-Lite on your code editor of choice. If it's not listed here, you may ping using one of our Communication and Coordination links in case one of us uses your code editor.

Editing Tips for configuring Sublime Text 2 for KA Lite dev

The "static" folder directly under the "kalite" directory is where static files get assembled, leading to duplicate files, which can cause confusion about which file should be edited. To exclude that directory from file matches and searches, save your project (Project > Save Project) and then edit it (Project > Edit Project), adding the `folder_exclude_patterns` key seen below:

```
{
    "folders":
    [
        {
            "path": "ka-lite",
            "folder_exclude_patterns": ["kalite/static"]
        }
    ],
        "settings":
        {
                "tab_size": 4,
                "trim_trailing_white_space_on_save": true,
                "translate_tabs_to_spaces": true
        },
}
```

Editing tips in PyCharm 3.x

1. Set the `kalite/static` folder as `Excluded` on the `Preferences -> Project Structure` settings. This is related to the `The "static" folder` explanation above for Sublime Text.

Coding-guidelines-and-conventions.md

General Guidelines

Priority on efficiency

KA Lite is designed to function reasonably well on low power devices (such as a Raspberry Pi), meaning we want to avoid doing anything computationally intensive. Also, for the cross-device syncing operations, connection bandwidth and speed are often expensive and slow, so we should always try to minimize the amount of data needing to be transferred.

Avoiding non-python dependencies

Because we want an extremely low friction cross-platform installation process, we only want to depend on Python libraries that are pure Python (no compiled C modules, etc) and cross-platform (i.e., work on both Linux, OSX and Windows). This allows us to fully include any dependencies directly in the codebase, to simplify download and installation.

Soft dependencies on a package with binaries is fine; e.g., for efficiency reasons, the project takes advantage of `python-m2crypto` if available, but falls back to `python-rsa` (a pure Python implementation) otherwise.

Git commits

commit messages

Try to use present-tense, imperative-mood sentences for your commit messages, as this is what Git uses for its merge commits by default. For example: "**Create** a styling table for the frontpage" rather than "creates" or "created".

Python 2.6 restrictions

We have decided to drop support for Python 2.6. Developers can freely use Python 2.7 features, like `OrderedDict` and set literals.

i18n-everything

Internationalization is core to our project and is sprinkled throughout the code, in terms of features we've created and exposing strings to users.

In order to expose strings to users, please follow the following conventions:

- In `.py` files (python code),
    - Make sure to import `ugettext` (`from django.utils.translation import ugettext as _`)
    - Wrap any user-facing string with `_("String")` , with
- In `.html` files (Django template files)
    - Make sure to import `i18n` (`{% load i18n %}' on the second line of the template)
    - Wrap any user-facing string with `{% trans 'String' %}` or `{% blocktrans %}String{% endblocktrans %}`
- In `.js` files (JavaScript files)
    - Wrap any user-facing string with `gettext("String")`

Style guides

For code, we generally follow the PEP8 conventions (http://www.python.org/dev/peps/pep-0008/), with a few exceptions/clarifications:

- Limit line length to 119 characters (PEP8 limits lines to 79 characters, but this can lead to a lot of wrapping)
- We're somewhat flexible in where we put empty lines; the goal is to use empty lines as punctuation to separate semantic units.

For comments, we follow Google's Python Style Guide (http://google-styleguide.googlecode.com/svn/trunk/pyguide.html?showone=Comments#Comments), which contain docstring formatting instructions.

- Read more about expectations related to documentation and testing.

Naming conventions

Besides styles, variable names should be kept consistent.

Paths

- variables/functions for directory paths: `my_varname_dirpath`
- variables/functions for file paths: `my_varname_filepath`

File pointers:

- `with open('fil.txt') as fp:"`

Code Structure Guidelines

App-independence

App creation / editing

We aim to create apps with a very simple dependency structure. This allows for apps to represent features, and therefore to be turned on or off via the `INSTALLED_APPS` Django variable without causing unexpected failures.

In order to do this, we follow the following conventions:

- Encapsulation: each app should only use Django-defined globals and globally defined css rules. All other variables, resources, and styles should be defined within the app.
  - No new global Django `settings`; each app should define and use their own `settings` in a `settings.py` file.
  - All app templates should be defined at `{app}/templates/{app}/[template].html`
  - All static files (images, css, js) should be defined at `{app}/static/{app}/[static files]`
  - All data files should be defined at `{app}/data/{app}/[data files]`

Commits

In order to promote reduced inter-app dependencies, we suggest that **commits be separated by app**. In some cases, this is an absolute necessity (due to the use of `git subtree`); however, we use this as a project-wide convention.

Imports

Below are the general rules we follow. At the end, find an example that shows each.

We separate our imports into 3 sections:

1. Libraries that can be installed via `requirements.txt` (but are currently contained in `python-packages`)
2. Django imports
3. FLE apps (including KA Lite apps as well as apps in other FLE projects, such as `securesync` and `fle-utils`).

We have the following conventions:

- Imports within an app should use relative imports
- Imports across apps should use absolute import paths

We order our imports by:

- Putting all `import {module}` before `from {module} import {function}` lines.
- Alphabetizing within `import` and `from` sub-sections

Example:

```
import re  # First section is for external packages that could be installed via requirements.txt
import time
import unittest
from selenium import webdriver  # "from {module} import {function/class}" come after "import mod
from selenium.common.exceptions import NoSuchElementException
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support import expected_conditions, ui  # Note the careful alphabetizati
from selenium.webdriver.firefox.webdriver import WebDriver

from django.core.urlresolvers import reverse  # Django goes second.  No "import {module} stateme
from django.test import TestCase
from django.utils.translation import ugettext as _

from fle_utils.testing.browser import BrowserTestCase  # All FLE imports also appear here.
from kalite import i18n  # All KA Lite imports are prefaced with kalite.
from kalite.facility.models import Facility, FacilityGroup, FacilityUser
from kalite.main.models import ExerciseLog
from kalite.main.topic_tools import get_exercise_paths
```

Project-independent Libraries: Python-packages directory

This directory contains code from external projects, and as such should not be modified. Only code within the `ka-lite` directory should be modified. A few notable exceptions:

- `kalite/distributed/static/khan-exercises` - this generally should not be changed
- `python-packages/fle_utils` - this can be changed (they are our utilities), but it is best to change in the `fle-utils` repo directly (adding unit tests to test the functions), then to pull the changes into KA Lite via `git subtree pull`
- `python-packages/securesync` - same as `fle_utils` above.

CSS: overriding Khan Academy's styling.

The file `khan-site.css` is from khan-exercises, and we don't want to modify it, in case we want to update it from there in the future. Instead, most CSS styling goes in khan-lite.css, and will override any styles defined in `khan-site.css` .

For styles that will only ever be used on a single page, they can be defined in a `<style>` block inside `{% block headcss %}` . Avoid using inline (tag attribute) styles at all costs.

Django

MVT

Django uses model-view-template pattern, and naturally so do we. Some overall ideas for following MVT is:

- Fat models and skinny views. Centralize your logic in models, don't disperse it in views.

- Template logic is template logic! If you want to put a link, use the `{% url %}` template tage, don't resolve to solutions like putting the link in your context.
- Furthermore, template logic is also presentation logic. Whether you want 25 or 50 rows displayed on a page is not the view's responsibility but goes in the template. That way, local template overrides can control more, and UI designers can too (without bothering about Python code). If you find yourself putting presentation logic in your view, it's probably because you've forgotten about writing custom template tags :)

On templates

We have a custom `static` tag that automatically appends a hash to the static url. This makes sure that we reload the assets whenever we update the server through the update process. We strongly recommend that you enable this to make sure that your assets are reloaded on every software update. To do so, just do a `{% load kalite_staticfiles %}` instead of a `{% load staticfiles %}` in your templates.

Common-variables-and-values.md

`video_id` vs. `youtube_id`

Conceptually...

`video_id` is an abstract ID that corresponds to a video lesson; `youtube_id` corresponds to an actual video (from YouTube!). Usually `video_id` corresponds to many `youtube_ids` (one per language), and a `youtube_id` corresponds to a single `video_id`. However, this is not strictly so--one `youtube_id` can be used for multiple `video_ids` (if a single video encapsulates multiple video lessons), and a `video_id` could have multiple `youtube_ids` for a single language (if the first was not very good, and was re-recorded)

In our code....

- `video_id` should be used for all reporting and for completion.
- `Youtube_id` is used for metadata (exactly which video was watched that led to this `video_id` being completed?) and for streaming videos (including showing subtitles), downloading, and deleting videos.

Therefore, `youtube_id` is used mainly in the `i18n` and `updates` apps, while `video_id` is used largely in the `coachreports` and `caching` code, while both are used in `main`, `khanload`, and elsewhere.

Communication-and-Coordination.md

The dev team uses the following resources to stay in touch and coordinate:

- Our dev Google group (https://groups.google.com/a/learningequality.org/forum/#!forum/dev) is where we send notifications and updates. Sign up!
- Our dev calendar (https://www.google.com/calendar/embed?src=bGVhcm5pbmdlcXVhbGl0eS5vcmdfdGU2cmpkdnBkMXJhbjNzamNtMnQ3dTd0YTRAZ3JvdXAuY2FsZW5kYXIuZ29vZ2xl is used for scheduling dev meetings, due dates for deliverables, and milestones along the way.
- Our HipChat dev room (https://www.hipchat.com/gzQfGFgv1) is a live chatroom where you can stop by and ask questions or hang out with other developers.
- Our Waffle board (https://waffle.io/learningequality/ka-lite) which lists the status of all the issues.
- Soon to come: a dev roster, defining people, their areas of interest & expertise, and contact information.

Content-packs.md

Why content packs?

For 0.15 and below, we started packaging assessment exercise resources such as PNGs and images into their own zip file (called the assessment zip). These need to be installed before KA Lite is usable. We also had language packs -- files that are useful for different languages, such as subtitles, translation catalog files, and dubbed video mappings.

For 0.16, we decided to combine both language packs and the assessment zip into one pack, called the content pack. The reasons are:

- **performance**: previously, since translation and the topic tree bundling logic was separate, translation of the topic tree, as well as topic availability, has to be done on the fly. Now with the topic tree pre-translated, stored into database, and availability marked to an extent, the KA Lite server has to do less work, especially during startup.
- **non-english installer bundles**: with translation files bundled in with assessment resources, it will now be easier to create KA Lite installers with different languages bundled in.

Terminology

- `assessment resource zip` : the zip file used for bundling assessment resources. Only contains assessment resources, and `assessmentitems.json` , which contains individual questions for each exercise. This is the only zip file needed by KA Lite versions 0.15 and below to work.
- `language packs` : zip files containing language related resources. These include:
  - catalog files, for translating the interface and topic tree.
  - video subtitles
  - dubbed video mapping, to allow the admin to download videos dubbed in the language currently activated.
- `topic tree JSON files` : for 0.15 and below, the Khan Academy topic tree data has been stored in 3 different JSON files: `exercise.json` , `topics.json` , and `content.json` . These are read and translated during startup for all languages installed.
- `unpack_assessment_zip` : the management command used to extract the contents of the assessment resource zip, and place it in the right location to make it serveable by KA Lite.
- `languagepackdownload` : the management command used to extract the contents of a language pack, and place it in the right location. This makes them usable by Django and servable by KA Lite.
- `content pack` : A zip file that contains the data from the assessment resource zip, language packs, and the topic tree JSON files. The most important thing it contains is the **content db**, which is the topic tree in a SQLite database, translated and availability annotated. The content db also has the assessment item metadata, also translated. For non-english content, each video node has also been mapped to its dubbed counterpart.

A content pack may also have the following contents:

- catalog files, for on-the-fly interface translation
- video subtitles
- assessment resources

0.16

Lay of the land

As a transitionary release, 0.16 will have both `unpack_assessment_zip` and `retrievecontentpack`, to ensure that we have enough time for testing the installers. We have the following content packs available:

- `en.zip`: the full content pack. Includes the content db, and assessment resources. About 500 MB in size.
- `en-minimal.zip`: the minimal version of `en.zip`. Only includes the content db. This can be downloaded by adding the `--minimal` flag to `retrievecontentpack`, like so:

  ```
  bin/kalite manage retrievecontentpack download en --minimal
  ```

- `{fr,zh,pt-BR,de,etc.}.zip`: the various content packs, meant to replace 0.15's language packs. Includes the content db, interface catalog files, and subtitles.

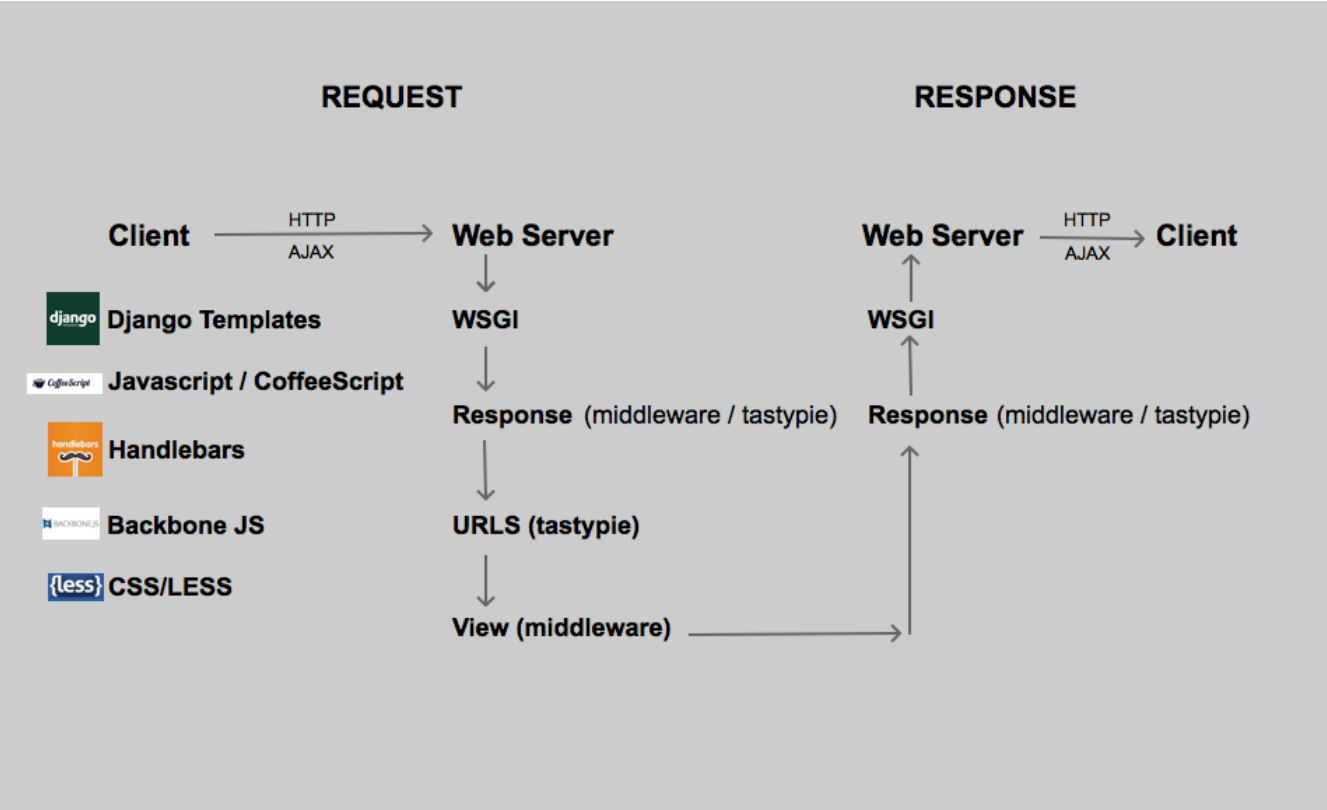
Migrating from `unpack_assessment_zip` to `retrievecontentpack`

As of this writing, the output of `make assets` is an sdist prebundled with the content db. To achieve that, it runs `retrievecontentpack download en --minimal`, fetching `en-minimal.zip` and then packaging the content db with the final sdist.

For installers, there are two options you can do, if you're using the sdist: continue using `unpack_assessment.zip` (link here (http://pantry.learningequality.org/downloads/ka-lite/0.16/content/khan_assessment.zip)), or bundle in `en.zip` (link here (http://pantry.learningequality.org/downloads/ka-lite/0.16/content/contentpacks/en.zip)) and unpack it using `kalite manage retrievecontentpack local en <en.zip path>`. Both have the same end result of including both content.db and the assessment resources. If you're not using the sdist, you can run just the `retrievecontentpack` step, or find a way to bundle both the content db, and then run use `unpack_assessment_zip` to unpack the assessment resource zips separately.

For the developer, it's recommended you use `bin/kalite manage retrievecontentpack download` moving forward.

Django-Architecture.md



References

- The Django Request-Response Cycle (http://irisbeta.com/article/245366784/the-django-request-response-cycle/)
- Django Request-Response Image (http://irisoccipital.s3.amazonaws.com/media/1048576/0174cf4a743911e4b422025f83075d2c.jpg)
- Request
    - Client
        - Django Templates
        - Javascript / CoffeeScript
        - Handlebars
        - Backbone JS
        - CSS/LESS
        - TODO(@AmodiaRichard): Bootstrap
    - Web Server
        - WSGI
        - Request (middleware / tastypie)
        - URLS (tastypie)
        - View (middleware)
- Response

- Web Server
  - WSGI
  - Response (middleware / tastypie)

Features-Matrix.md

This shows a list of features that already exist or doesn't exist, being developed, or to be deprecated per branch of the project repository.

Objective Prevent new/existing developers from asking which feature exists on which branch?

Legends

- Y or :+1: == exists
- - (dash) or :-1: == doesn't exist
- P or :clock1: == in progress
- D or :bomb: == to be deprecated
- ? or :question: == unknown
- I or :bulb: == idea to be implemented

Feature	master	develop	release-0.12.0	nalanda-rct3	wch	
sample feature	:+1:	:-1:	:question:	:bomb:	:clock1:	
khan-exercises	:+1:	:+1:	:+1:	:+1:	:+1:	:question:
Perseus-exercises	:question:	:question:	:question:	:question:	:question:	:question:
content log	:-1:	:+1:	:-1:	:-1:	:question:	
* content log - video	:-1:	:+1:	:-1:	:-1:	:question:	
* content log - audio	:-1:	:+1:	:-1:	:-1:	:question:	
* content log - pdf	:-1:	:+1:	:-1:	:-1:	:question:	
* content log - exercise	:-1:	:+1:	:-1:	:-1:	:question:	
coach reports	:question:	:question:	:question:	:question:	:question:	
scatter reports	:question:	:question:	:question:	:question:	:bulb:	

General-Coding-Resources.md

Getting Started

Codecademy (http://www.codecademy.com/) has courses to learn:

- HTML/CSS (http://www.codecademy.com/tracks/web) (*Front End Focus*)
- Javascript (http://www.codecademy.com/tracks/javascript) (*Front End Focus*)

- Python (http://www.codecademy.com/tracks/python) (*Back End Focus*)

from scratch. All the coding is done in the browser, through small, bite size chunks to get you independently coding.

Diving Deeper

Python

- Awesome Python Experience (http://awesomepython.com/)
- learnpython.org (http://www.learnpython.org/)
- Wikibook on Python Programming (http://en.wikibooks.org/wiki/Python_Programming)
- Learn Python the Hard Way (http://learnpythonthehardway.org/book/)

Django

The Django Tutorial (https://docs.djangoproject.com/en/1.7/intro/tutorial01/) is the best introduction to how Django works to create a web application. Completion of this tutorial will help you figure out your models from your views, and get you introduced the the Django templating language. (*Back End Focus*)

For a jumpstart on templating see this intro to Django templating (http://jeffcroft.com/blog/2006/feb/21/django-templates-an-introduction/). (*Front End Focus*) **(broken link)**

More templating tutorials (https://code.djangoproject.com/wiki/Tutorials#Templates).

jQuery

Codecademy's jQuery class (http://www.codecademy.com/tracks/jquery). (*Front End Focus*)

jQuery 101 (https://learn.jquery.com/javascript-101/) has a good overview of the basics of jQuery, the Javascript library that powers much of the front end code of KA Lite. (*Front End Focus*)

I can see my house from here!

Backbone

Backbone tutorials (http://backbonetutorials.com/) can help you get to grips with Backbone.js a library used in small but crucial parts of the KA Lite client side code. (*Front End Focus*)

d3

These tutorials (http://alignedleft.com/tutorials/d3/) will introduce you to data driven documents (d3) (http://d3js.org/), a Javascript library used to create dynamic data visualizations.

Getting-started.md

Firstly, thank you for your interest in contributing to KA Lite! The project was founded by volunteers dedicated to helping make educational materials more accessible to those in need, and every contribution makes a difference. The instructions below should get you up and running with the code in no time!

Setting up KA Lite for development

See: http://ka-lite.readthedocs.io/en/develop/developer_docs/environment.html (http://ka-lite.readthedocs.io/en/develop/developer_docs/environment.html)

Which branch should I work on?

The `develop` branch is reserved for active development. It's very unstable and may not be usable at any given point in time -- if you encounter an issue running it, please check with the development team to see if it's a known issue before reporting it.

When we get close to releasing a new stable version of KA Lite, we generally fork the develop branch into a new branch (like `0.16.x`). If you're working on an issue tagged for example with the `0.16.0` milestone, then you should target changes to the `0.16.x` branch. Changes to such branches will later be pulled into `develop` again. If you're not sure which branch to target, ask the dev team!

Putting some sample user data into KA Lite

To generate sample Learner and Facility data to help you test KA Lite features, run the `generaterealdata` management command from the command line:

```
./bin/kalite manage generaterealdata
```

Getting your changes back into KA Lite

See the guide for opening pull requests (https://github.com/learningequality/ka-lite/wiki/Submitting-Pull-Requests).

Next steps

- Once you've toyed around with things, check out our style & structure guide to understand more about the conventions we use.
- Now that you're up to speed on conventions, you're probably itching to make some contributions! Head over to the GitHub issues page (https://github.com/learningequality/ka-lite/issues) and take a look at the current project priorities. Try filtering by milestone. If you find a bug in your testing, please submit your own issue!
- Once you've identified an issue and you're ready to start hacking on a solution, read through our instructions for submitting pull requests so that you know you can cover all your bases when submitting your fix! You might wanna look at some common Git commands to get you up and running.

Helpful-Git-commands.md

You should clone from your forked repo (https://help.github.com/articles/fork-a-repo/) for this to work:

Set up your upstream remote

```
git remote add upstream git@github.com:learningequality/ka-lite.git
```

Update your repo with changes from upstream

BRANCH = branch you want to update

```
git checkout $BRANCH
git fetch upstream
git merge upstream/$BRANCH
```

Push your branch to your remote repo on GitHub

```
git push origin $BRANCH
```

Push your current branch to a branch that has a different name

```
git push origin $BRANCH:$DIFFERENT_BRANCH
```

Restore your working copy to the latest committed version on the current branch

```
git reset --hard HEAD
```

Find a string amongst all files tracked by Git

```
git grep $STRING
```

Home.md

Welcome to the KA Lite Wiki!

KA Lite is an open-source Python/Django project created and maintained by the nonprofit Learning Equality (http://learningequality.org/) that provides offline access to Khan Academy materials, for the 4.5 billion people around the world without Internet.

This wiki is the main source of documentation for ***developers*** contributing to the KA Lite project. It has been organized like an FAQ, with outbound links to resources beneath commonly asked questions.

How do I get started contributing?

There are two primary ways to contribute to KA Lite!

- Use KA Lite and give us feedback on our user documentation (http://ka-lite.readthedocs.org/en/latest/). If you'd like to change something in the documentation or seek clarification, you can email us directly, open an issue on github to discuss, or open a pull request to propose changes directly.
- If you're a developer, you can contribute code. See our Getting started page for instructions on setting up your dev environment and running the code.

What is KA Lite anyway?

- Check out the KA Lite website (https://learningequality.org/ka-lite/) to learn more about KA Lite!
- Read the Learning Equality homepage (https://learningequality.org/) page to understand why we started and what the ultimate vision is.

At a high-level, how does the app work?

- Check out the Project structure page for an overview of how the code works.

Where can I read more in-depth docs?

It's a work in progress, but we've created some specific docs on certain topics, and they're listed below:

- Inter-app dependencies
- Common variables and values
- Creating Issues
- Unit tests Part 1 (https://github.com/learningequality/ka-lite/wiki/Guide:-Writing-unit-tests---Part-1)
- Unit tests Part 2 (https://github.com/learningequality/ka-lite/wiki/Guide:-Writing-unit-tests-Part-2)
- Caching
- Benchmarking the Raspberry Pi
- Internationalization: Coding
- Securesync: How Cross Computer Syncing Works
- Student data records
- The setup codepath
- The topic tree

How do I contact you people for help and support?

- To get (and stay) in touch with the team, check out our page on communication & coordination.

Extra goodies

- Coding tutorial resources
- Tips and Tricks
- Code Editor Tips

Inter-app-dependencies.md

The relationship between apps should be explicit and intentional.

All apps within the `kalite` directory are dependent on `settings.py` (at least until #1644 is complete).

KA Lite app definitions

- `i18n` : knows about languages, language packs, and language mappings between `youtube_id` and `video_id`
- `facility` : authentication and user grouping system. Accesses `i18n` to allow users to set a default language limited to the languages currently available.
- `main` : associates `facility` users with khan academy (exercise & video logs) and login (userlog) data.

- `khanload` : allow importing of data from khan academy, either into the `main` data sources, or into the database (`main` models)
- `coachreports` : reports on top of `main` data
- `control_panel` : views on users, user groups, and hooks into their data.
- `updates` : allows updating of language packs (via `i18n`), videos (via `main`), and software.
- `distributed` : combines all apps together.

KA Lite inter-app dependencies

Project-independent (and so won't be included in the list of dependencies):

- `fle_utils` - no dependencies. internal utilities for FLE. pretty much everything depends on it :)
- `fle_utils/chronograph` - no dependencies.
- `fle_utils/config` - no dependencies. Similar to `settings.py` , but written into the database.
- `securesync` - depends on `config` (to store the crypto keys) and `fle_utils`

Downloaded, tweaked, but don't have intra-app dependencies (outside of `settings.py`):

- `django_cherrypy_wsgiserver` - the main server process for the distributed server

New modules for KA Lite

- `distributed` - Depends on all apps that have distributed server end-points (`coachreports` , `control_panel` , `facility` , `main` , etc.)
- `coachreports` - Depends on `main` (for models)
- `control_panel` - Depends on `main` (for models) and `coachreports` (to integrate them into the interface)
- `facility` - depends on `config` (for default facility) and `securesync`
- `i18n` - depends on `config` (for default language), `main` (for mappings of youtube_id to video_id)
- `kalite` - The project; depends on all apps :)
- `khanload` - Depends on `main` (so it can write into main's data structures; could be refactored)
- `main` - depends on `facility` (for facility user logins). SHOULD not have circular dependencies, but currently depends on `i18n` (to map youtube_ids to video_ids) and `khanload` (unnecessarily).
- `shared` - None / Not applicable
- `testing` - None / Not applicable
- `updates` - Depends on `main` ... but could be refactored to become an independent library

KA-Lite-Dev-Roster.md

Please add your name, interests/skills, and a link to your GitHub profile.

Jamie Alexandre (https://github.com/jamalex)

- Familiar with ka-lite, ka-lite-central, fle-home, and the RPi/Ubuntu-related repositories.
- Particular interest and experience, for ka-lite, in security, databases, syncing, and KA-related stuff.

Richard Tibbles (https://github.com/rtibbles)

- Familiar with ka-lite, ka-lite-central, and KA-API-Py.
- Particular interest and experience, for ka-lite, in front end, API endpoints, and munging KA topic trees.
- Coach reports, teacher tools, student reports, and enhancing pedagogical aspects of KA Lite.

Aron Asor (https://github.com/aronasorman)

- Familiar with ka-lite (i18n specially).
- Interests are building internal tools, improving dev processes.

Dylan Barth (https://github.com/dylanjbarth)

- Familiar with (in order of experience) fle-home (incl. map), ka-lite, ka-lite-central
- Particular interest and experience in UX, information architecture, and operational and learning processes (e.g. how the dev team works as a unit, how a developer can learn actively)

Cyril Pauya (https://github.com/cpauya)

- Familiar with ka-lite and ka-lite-central.
- As a newcomer, have touched unit-tests, middlewares, model migrations, api, front-end templates, and some deployment scripts on linux/osx.
- Interests for ka-lite in testing, bug-fixing, documentation, and deployment.

Major-differences-between-KA-Lite-and-Khan-Academy.md

Content

KA Lite contains a subset of content of Khan Academy. We currently do not contain the following content:

- SAT (200+ videos, 10+ exercises)
- Partner content (Stanford University)
- Partner content (Crash Course)
- Computer science ()

Functional

- No discussion area for students to interact on each video.
- Perseus exercise framework is not integrated limiting the number of Common Core Skills exercises available.
- Mastery model uses the older "streak"-based model, not the newer machine-learning model (http://david-hu.com/2011/11/02/how-khan-academy-is-using-machine-learning-to-assess-student-mastery.html) nor the latest spaced repetition model to achieve mastery.
- Common Core Skills and expanding library of Perseus created questions grouped under each skill.

Managing-the-Django-server.md

What are commands?

From the command prompt, in your ka-lite/kalite directory, if you run `python manage.py` , you will see a list of management commands that give you some control over your app.

How do I run them?

From the command prompt, in your ka-lite/kalite directory, run `python manage.py [command]` ,

What commands are available?

Here, we list out ALL of the available commands in three sets:

- KA Lite-related commands that we intend you to use
- KA Lite-related commands that we do not intend for you to use
- KA Lite-unrelated commands

KA Lite-related commands: *OK to use*

- [kalite] update - update your version of KA Lite. Must be online or provide a zip file. (v0.9.4+) zip_kalite - package your version of KA Lite into a zip file, to share with your friends! Includes local_settings.py, but no zone information nor data.
- [main]
    - apacheconfig - for configuring KA Lite to run under apache (by default, configured to run under a Python-based web server)
    - cache - manipulate the cache
    - subtitledownload - force downloading and installation of specified subtitles data (v0.9.4+)
    - videoscan - rescan the hard drive and database, to synchronize available video information.
- [securesync]
    - changelocalpassword - reset the password for a facility user (student, teacher account) e.g. changelocalpassword username
    - retrypurgatory - run only if you find errors in syncing
    - syncmodels - force models to synchronize immediately (online access required)

KA Lite-unrelated commands: *OK to use*

- [auth]
  - changepassword - reset the password for an admin account (not teacher, nor student)
  - createsuperuser - create a new admin account (not teacher, nor student)
- [django]
  - dumpdata - save your local data to a backup (JSON format)
  - loaddata - load your local data from a backup (JSON format)
  - validate - validate your basic server installation
- [south]
  - migrate - run in case your

KA Lite testing related commands: *DON'T use these!*

- [coachreports]
  - generatefakedata - generate exercise data for fake users
  - generaterealdata - generate exercise, video, and user login data for fake users
- [django_cherrypy_wsgiserver]
  - runcherrypyserver - runs the python-based web server (run via start.sh / start.bat instead)
- [main]
  - initdconfig -
  - khanload - download new topic data from Khan Academy. WARNING: may fail, and may destroy necessary KA Lite data!
  - videodownload - force downloading of videos selected from "update" UI
- [securesync]
  - generatekeys -
  - initdevice - run once, during installation.

Project-structure.md

KA Lite is a medium-size project, so keeping a well-defined structure is essential to making it understandable.

Below is an outline of the directory structure for the project, as well as how apps are currently structured.

Project Directories

The KA Lite project has the following subdirectories:

Code

- kalite (https://github.com/learningequality/ka-lite/tree/master/kalite) - Django apps we've created or downloaded and modified for the ka lite projects
- python-packages (https://github.com/learningequality/ka-lite/tree/master/python-packages) - Django apps and Python package dependencies for apps within `kalite`
- scripts (https://github.com/learningequality/ka-lite/tree/master/scripts) - OS-specific scripts for starting/stopping server (and other similar tasks)
- static-libraries (https://github.com/learningequality/ka-lite/tree/master/static-libraries) - Downloaded static libraries that are shared across all apps.

Resources

- content (https://github.com/learningequality/ka-lite/tree/master/content) - contains video files and video preview images
- docs (https://github.com/learningequality/ka-lite/tree/master/docs) - .md files for developers and KA Lite users
- locale (https://github.com/learningequality/ka-lite/tree/master/locale) - contains translations that are downloaded via language pack updates

Apps

KA Lite created / modified apps

Distributed server-specific - only used on the installable KA Lite server

- django_cherrypy_wsgiserver - wrapper around cherrypy for use in Django
- khanload - code and commands for downloading Khan Academy's topic tree and user data
- updates - sister app of chronograph; updates job status from back-end management commands to the front-end UI

Shared - Shared between both technologies

- coachreports - graphical displays of student progress
- control_panel - summaries of all data (usage and syncing)
- i18n - tools for implementing language packs, including interface translations, subtitles, and dubbed videos
- main - main website and student progress recording
- securesync - engine for syncing data, as well as defining users
- tests - framework for functional and performance testing
- utils - app-independent utilities (could be shared with the world!)

Library apps

These are located in the `python-packages` directory.

Shared FLE libraries

- fle_utils - includes:
  - chronograph

    - config
- playground
- securesync

True libraries - usually get via `sudo apt-get` , but we download and ship for offline completeness

- cherrypy
- collections
- django
- httplib2
- pytz
- requests
- rsa
- selenium
- south

External helpers - Collected from around the web, we use this code without modification. **NOTE** : many of these may belong to "true libraries" above.

- annoying
- dateutil
- debug_toolbar
- decorator
- django_extensions
- django_snippets
- git
- ifcfg
- iso8601
- khanacademy
- memory_profiler
- oauth
- pbkdf2
- polib
- postmark
- pyasn1

- werkzeug

App file structure

Apps are now self-contained entities with as few inter-app and global project dependencies as possible.

Files

- Each app contains relevant standard Django files (`forms.py` , `models.py` , `views.py` , `urls.py`)
- Some apps have both HTML views as well as API/JSON views. These are defined by `api_xxx.py` files, such as `api_views.py` , `api_urls.py` , etc.
- Any shared functions across the app/module with other apps should be defined within the `__init__.py` file
- App-specific **settings** should be put inside the app's `settings.py` file. This includes required middleware and context processors.
- App-related **template files** should be put inside the `{app}/templates/{app}/` folder (or any subfolders), and referenced as `{app}/template.html` from view functions.
- App-related **static files** should be put inside the `{app}/static/[css|images|js]/{app}/` folder, and referenced as `{% static 'static/[css|images|js]/{app}/file.ext' %}` from template files.

Report-Bugs-by-Creating-Issues.md

If you encounter a bug while testing or using KA Lite, we'd love it if you could report it to us! In GitHub, bugs can be reported through the "Issue" feature.

You can create a new issue by signing in (https://github.com/login) to GitHub, visiting our issues page (https://github.com/learningequality/ka-lite/issues) and clicking the green "New Issue" button. Below, we've explained how to create a really great bug report, which will help our team of open source developers get a fix in as quickly as possible!

Issue format:

Issue title: *1-sentence description of the issue or bug that you've found*

Issue Body:

- Branch: *which branch you saw the issue on (if you aren't sure what a branch is, you're probably on the 'master' branch*
- Expected Behavior: *1-2 sentence description of what you expected to happen*
- Current Behavior: *1-2 sentence description of what actually happened*
- Steps to reproduce: *steps a developer can take to reproduce the bug*
- Screenshot(s): *please include screenshots of the bug and/or error screens, if possible*

Example of a good issue

Issue title: Deleting a facility from the "overview" page shows a JSON page in the browser.

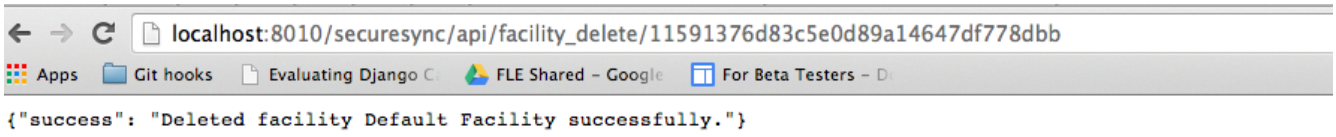
Issue Body:

- Branch: develop

- Expected Behavior: When clicking the "delete" button on the facility, the facility would be deleted and the "overview" page would be reloaded.
- Current Behavior: The facility is deleted, but the "overview" page no longer shows--a JSON blob is shown (see screen shot below).

Steps to reproduce:

1. Log in as the administrator.
2. Create a new facility
3. Click the "overview" navbar link.
4. Next to the facility, click on the "delete" icon.



Example of a bad issue

Issue title: Deleting a facility doesn't work.

Issue Body: I clicked the delete link but I just saw JSON.

Can't create a GitHub account

If you aren't able to create a GitHub account, please feel free to email our dev team a bug report. Please format your report as instructed above, and email the report to support [at] learningequality.org and we will create the issue for you as long as it's correctly formatted!

Student-Data-Records.md

Student Data Records

We keep logs of student progress in order to help teachers assess student progress and trouble. We limit these logs to a few pieces of basic information, in order to keep the app running with low required resources. We expose those data to teachers through our coach reports.

Here is documentation about the fields that we store when a student interacts with videos or exercises

To generate example student data, you can run the 'generaterealdata' management command by typing this in the command line: `bin/kalite manage generaterealdata`

Exercise Log

As students do exercises, they can check their answer and be right or wrong, or get hints.
If right: streak_progress and points continue incrementing. If hint or wrong: streak_progress and points get reset to 0.

- streak_progress: [# of consecutive correct answers (with no hints)] * 10% (10 consecutive answers required to get to 100%!)
- attempts: # of times the student tried answering (includes successful & failed answers)
- points: total points acquired on this exercise
- complete: If the student got streak_progress=100
- struggling: True IF attempts > 20 and not complete
- attempts_before_completion = models.IntegerField(blank=True, null=True)
- completion_timestamp = models.DateTimeField(blank=True, null=True)
- completion_counter = models.IntegerField(blank=True, null=True)

Video Logs:

- total_seconds_watched: integer specifying the total # of seconds a video was watched. If they replay the video, or leave the page and return, this number is added to (never reset)
- points: Points are received while watching a video, with a maximum number per video. If the video is rewatched, then the # of points is reset (not added to)
- complete = models.BooleanField(default=False)
- completion_timestamp = models.DateTimeField(blank=True, null=True)
- completion_counter = models.IntegerField(blank=True, null=True)

Submitting-Pull-Requests.md

Before submitting a Pull Request, please review each of the following items.

Getting started

First, you need to determine where to branch from before writing your code. This will depend on several things (whether it's a bug fix or a larger feature, where we are in the current release cycle, etc). See the strategy document for branching and merging (https://github.com/learningequality/ka-lite/wiki/Branching-and-merging-strategy) for guidance, but always ask if you're not sure.

Once you've decided what branch to work off of -- let's say it's `develop` -- do the following:

```
git checkout upstream/develop
git checkout -b name_of_your_feature
```

And after making and committing changes, push them to a branch on your fork using:

```
git push origin name_of_your_feature
```

Then, go to your fork and click the "New pull request" button. Someone will review your code and you can push new commits to your branch to have the updates added to the open PR (pull request). See GitHub's notes for how to create a Pull Request (https://help.github.com/articles/using-pull-requests).

Coding Guidelines and Conventions

We expect submitted code to follow our code conventions and styling (https://github.com/learningequality/ka-lite/wiki/Coding-guidelines-and-conventions). We also have careful guidelines about app structure and inter-app dependencies (https://github.com/learningequality/ka-lite/wiki/Coding-guidelines-and-conventions#code-structure-guidelines), including expectations about how commits are structured.

Documentation

For all new features and any major changes (including class or function additions) for bugfixes, we require documentation to be submitted along with code changes.

For comments, we follow Google's Python Style Guide (http://google-styleguide.googlecode.com/svn/trunk/pyguide.html?showone=Comments#Comments), which contain docstring formatting instructions.

We use docstrings & comments for each of the following

- New modules / apps: docstring in the module's `__init__.py`, explaining the high-level need, design, and dependencies.
- New files: docstring at the top of the file defining what lives inside, and any overall design.
- New functions/classes: docstring for each
- Inline: as needed

Here's an example of the standard docstring for a public function:

```
def public_fn_with_googley_docstring(name, state=None):
    """This function does something.

    Args:
       name (str):  The name to use.

    Kwargs:
       state (bool): Current state to be in.

    Returns:
       int.  The return code::

          0 -- Success!
          1 -- No good.
          2 -- Try again.

    Raises:
       AttributeError, KeyError

    A really great idea.  A way you might use me is

    >>> print public_fn_with_googley_docstring(name='foo', state=None)
    0

    BTW, this always returns 0.  **NEVER** use with :class:`MyPublicClass`.

    """
    return 0
```

Testing

Test Types

We have four types of tests in our repository.

- `Unit tests` - These tests are direct tests of function inputs and outputs. Unit tests are meant to test python functions that will be called by the Django framework. They're meant to do test most of the business logic inside our app.
- `API tests` - These tests use a programmable HTTP client to test API endpoints. In general, API tests have to make sure that they either

    1. have the correct input and output
    2. throw the right exceptions when given bad input
    3. call into the right function, which is sufficiently tested by unit tests.

    In general, both API and unit tests work in tandem. You don't want to duplicate testing across these two types of tests. For example, if you've tested a certain utility function, you just want to make sure that a view just calls that function with the right arguments. Re-testing the logic of the view function is just duplication of testing and should be avoided as much as possible.

- `Browser tests` - Also known as integration tests, wherein you test that the entire feature works, from frontend to backend. These tests use a browser to click links, submit forms, examine elements, and test end-user experience.
- `Ecosystem tests` - Another type of integration test, specifically for testing interactions between different server types. These tests must be run from our central server repository (https://github.com/fle-internal/ka-lite-central/wiki). These tests install multiple versions of KA Lite (including the central server) to test data sharing across installations.

Note that these 4 tests all work in tandem to fully test the code.

We use Django's extension of the `unittest` framework for `unit tests`, Django's `LiveServerTest` for any tests requiring KA Lite to be running (`API`, `Browser`, and `Ecosystem` tests), and `selenium` to launch and run `Browser` tests.

Test Requirements

All new features must have unit tests, and should use other types of tests as applicable (see below for examples). Regression tests (tests for bugs that have been found) can be in whatever test type is applicable as well.

Test Structure and Tools

Files should be contained within the app in a `tests` directory.

- If no such directory exists, create one! Copy an `__init__.py` from an existing test repository--it contains code needed for loading all tests into the `kalite.{app}.

Test Classes and Examples

`Unit Tests`

`API Tests`

`Browser Tests`

Code review guidelines

After submitting your pull request, someone will review your code and provide feedback. Here are the guidelines you and the code reviewers should follow: