# Content Authoring Management and Publishing System—CAMPER

# The JUICE Ecosystem

JUICE is a Web Application that relies on a cloud-based content repository to serve content to its users. It is composed of a (1) Web Server running a MEAN stack (MongoDB database, Express Web application framework, AngularJS client-side JavaScript MVC framework, Node.js server-side JavaScript execution environment) and Nginx HTTP server, and (2) an Amazon S3-based content repository. The content repository contains three types of files—JSON files describing the organization of the site, JSON files describing the mini-lesson content proper, and media and text files supporting the content (e.g., images files).

JUICE Server

Content management  
(CAMPER)

JUICE User (Web Browser)

Content Repository (Amazon S3)

The three components mentioned above are the only ones required to run JUICE. Content description files can be created/modified by using specialized authoring screens, and the site structure can be customized by editing the corresponding JSON files.

However, manual editing is not optimal for creating and managing a large number of files. The JUICE team developed a specialized web application to support and streamline **c**ontent authoring, management, and publishing. This is called CAMPER.

CAMPER is built around a custom Quick Base (QB) application (<http://www.quickbase.com>). QB is an on-line database well suited to develop workflow management systems, and is scriptable in Javascript via a rich API.

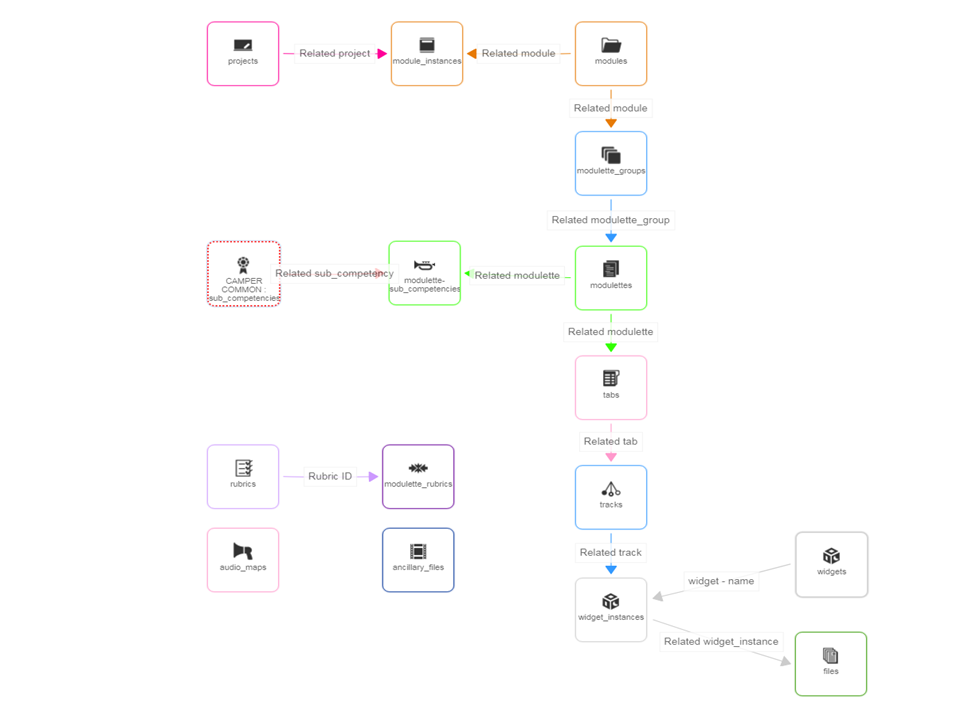
CAMPER consists of two Quick Base apps: CAMPER, and CAMPER COMMON. CAMPER holds the site structure and content information for the Core JUICE program. CAMPER COMMON holds code pages that are used by CAMPER, and also holds tags and competency information that might be used across multiple JUICE programs.

The remainder of this document provides an overview of CAMPER. Detailed documentation on how to use CAMPER to manage and publish content and create customized instances of JUICE is available in the **Working in CAMPER** documentation directory.

Note: As described later, CAMPER explicitly supports a development workflow relying on three JUICE server instances—a development (DEV) server, a quality insurance (QA) server, and a production (PROD) server. Only the DEV server instance is required to create and manage content.

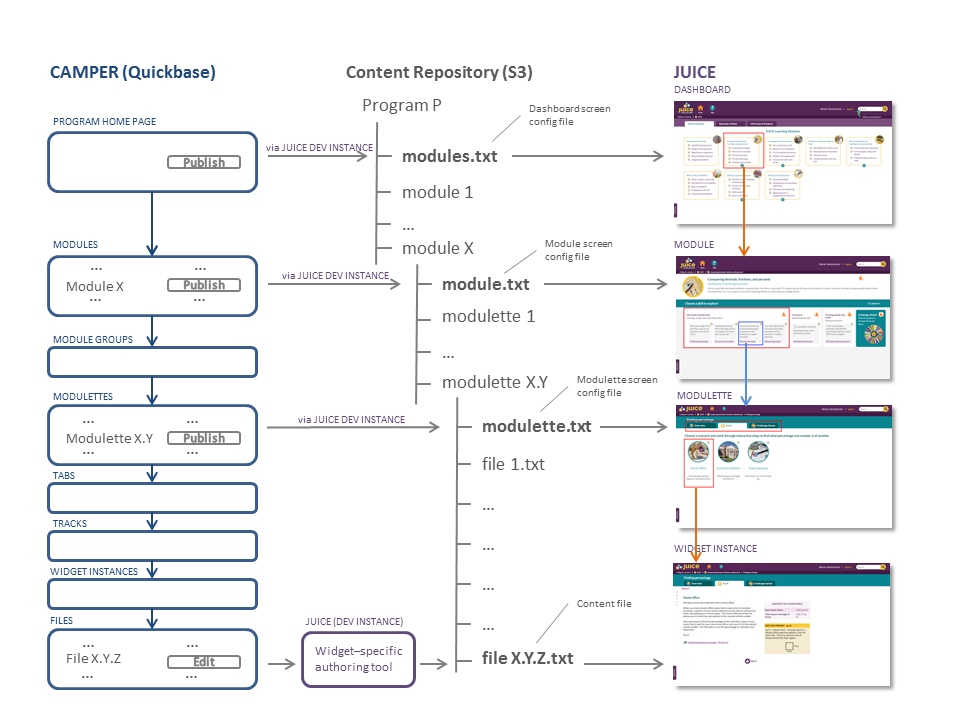
Information on how to re-create your own instance of CAMPER and CAMPER Common is available in the document **Re-creating CAMPER**.

# CAMPER and CAMPER COMMON Tables and Relationship Diagram



The tag fields in the modules, modulettes, and ancillary\_files tables pull values from CAMPER COMMON.

# Content Structure Authoring and Publishing



The diagram above illustrates the correspondence between the main QB pages, the directory structure and json files in S3, and the JUICE User Interface. The structure of a JUICE program is described entirely through the QB tables and relationships shown in the Tables and Relationship Diagram presented in section 2. Select QB pages display Publish buttons that, when activated, generate json files that encapsulate these content and relationships, and store them in a directory structure in S3. The JUICE application reads these files and presents the content to the users. A brief description of the essential features illustrated by the diagram follows.

CAMPER Home Page

The CAMPER Home Page displays a button to publish the program dashboard to S3. The program corresponding file, *modules.txt*, is located in the root of that program directory in the content repository in S3, and contains all the information required to build the programs Modules dashboard.

Modules Page

The Modules page displays the content of the *modules* table. Each record contains the description of a module, and a link to the modulette groups it contains (a modulette is the QB internal name for a mini-lesson). Each record displays a Publish button that generates a json files (*module.txt*) that contains all the information required to build the corresponding Module page in JUICE. This file is located in the root of the corresponding module sub-directory of the program directory in S3.

Modulette Groups Page

The Modulette Groups page displays the content of the *modulette\_groups* table. Each record contains the description of a modulette group, and a link to the modulettes it contains.

Modulettes page

The Modulettes page display the contents of the *modulettes* table. Each record contains the description of a modulette, and a link to the tabs it constains. Each record displays a Publish button that generates a json file (*modulette.txt*) that contains all the information required to build the corresponding JUICE mini-lesson page, including tabs and tracks information. This file is located in the root of the corresponding modulette sub-directory of the module directory in S3.

Tabs Page

The Tabs page displays the content of the *tabs* table. Each record contains the description of a tab in a modulette (a.k.a mini-lesson), and a link to the tracks it contains.

Tracks Page

The Tabs page displays the content of the *tracks* table. Each record contains the description of a track in a modulette, and a link to the modulette instances it contains. A modulette instance is the combination of a widget type (Overview, Try It, a specific Challenge Game type) and a file that describes the content to be “played” by that widget.

Widget Instances Page

The Widget Instances page displays the content of the *wiget\_instances* table. Each record contains a description of the instance, including widget type and a link to the content file to be played by the widget.

Files Page

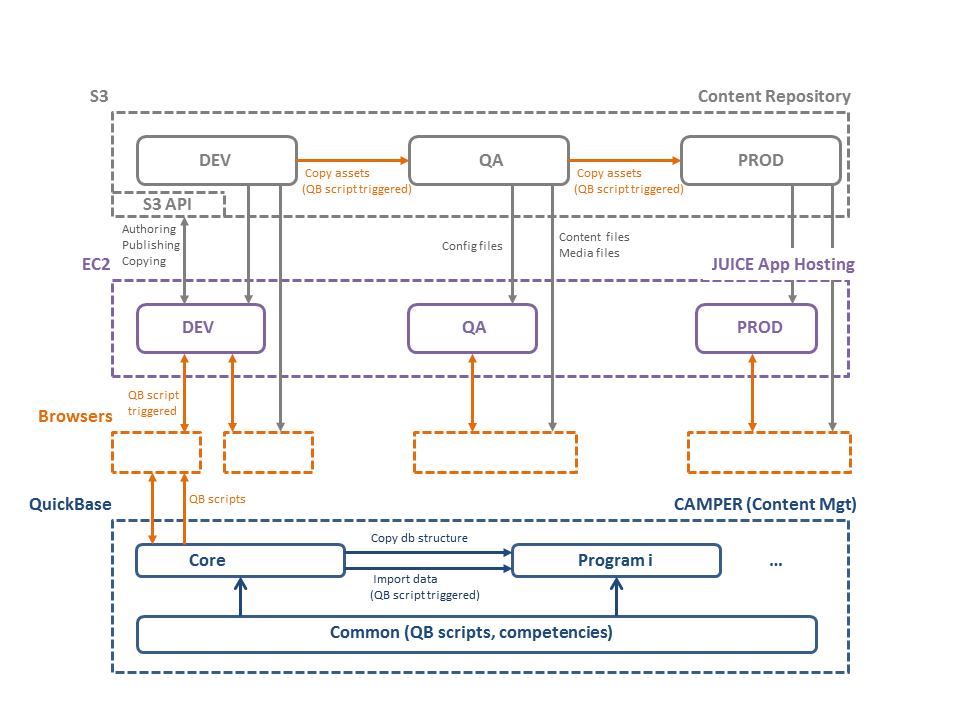
The Files page displays the content of the *files* table. Each record contains a description of the file, including related widget (i.e., content player) and a button allowing editing the content file to be played. Clicking the Edit button brings up a widget-specific authoring page (hosted on JUICE Dev server) that allows authoring of the content, as well as saving it to a file in S3. This file is located in the root of the corresponding modulette sub-directory of the module directory in S3.

Ancillary\_files Page

The Ancillary\_files page displays the content of the ancillary\_files table. Each record contains the link to a file, tags, title, and buttons that allow the file to be uploaded and copied to directories in S3. These files are used to provide additional learning resources and are usually linked to from within the main content files.

The rest of the CAMPER pages display supporting information and tools. See the *document Re-creating CAMPER* for more information about each table and what it does.

# JUICE Systems and Data Flow



As mentioned in the introduction the JUICE ecosystem, CAMPER has been designed to support a development workflow that encompasses three phases, each supported by its own JUICE server (an EC2 instance) and content repository (S3 bucket)—development (DEV), testing/quality assurance (QA) ,and production (PROD). This workflow is illustrated in the above diagram.

Authoring and publishing activities from CAMPER (discussed in the preceding section) interact with DEV server and bucket only. CAMPER provides additional scripts, activated by buttons embedded in QB pages, to copy files from DEV to QA buckets, and from QA to PROD when ready to be promoted.

You can create alternate JUICE programs with different content from the main Core JUICE program. Each alternate program is supported by an individual QB application created by copying the core JUICE database structure, and importing specific modules to the new program using custom scripts. See the document *Creating a new program for admins* for more information.

Core JUICE, as well as all derivative programs, rely on a number of common elements—code pages, and tables describing competencies, sub-competencies, and tags. These common elements are collected in a separate QB application, called CAMPER COMMON.