QTO SYSTEM GUIDE

1. INTRO

1.1. Purpose

The purpose of this guide is to provide description of the qto system and application architecture.

1.2. Audience

Target audience of this document is comprised of the architects and System designers of a potential or current system, comprised on deployed and operating qto instances. Developers and devops operators.

THE INFORMATION SYSTEM AS CYBERNETIC ORGANISM

2. ABSTRACTION

This section is essential for you to understand the very basic motives and principles behind the design of the qto applications and systems. Elon Musk stated once in an interview, that any organisation might be viewed as a cybernetic organism, ran to a certain degree by an Artificial Intelligence Instance...
Qto has nothing to do with the modern AI definition, yet it's design is inspired by this cybernetic organism or biome metaphor.

2.1. The qto Information System - definition

There are multiple definitions for "Information System" - most of those apply to the qto information system, yet the following definition provides additional content, which might not necessary be applicable to any other non-qto IS.
The qto information system is the interactive combination of the people , processes & policies, hardware & networks , source code, binaries, configuration and data managed by any running instance of the qto application.

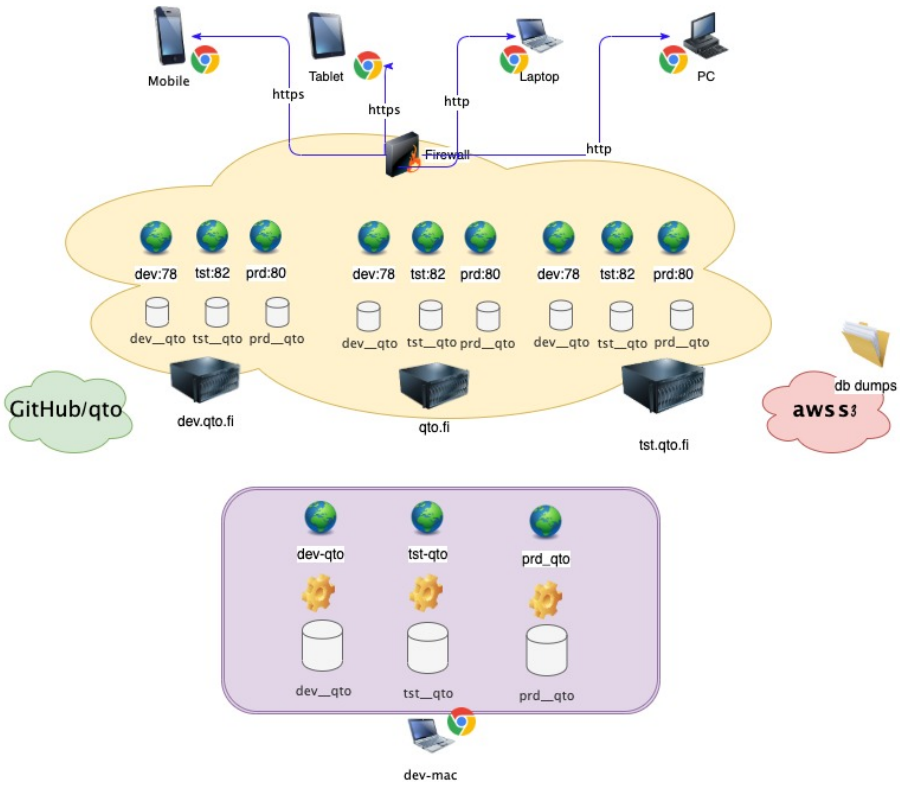
3. THE QTO SYSTEM INFRASTRUCTURE

This section describes the current system infrastructure.

3.1. System architectural overview by SIWA dia

The following diagram implements the Simplest Possible Way of describing Architecture principle - it's sole purpose is to quickly provide an overview of the existing infrastructure built with the help of the qto application as well as provide visual tool for communication related to the application.

Figure 1: The qto insfrastructure



3.2. Infrastructural components

3.2.1. End-user clients

The end-users can access any qto application via their browsers. All of the functionalities are available in mobile browsers (smart-phones or tablets not older than 2 years)

3.2.2. Application Layer and databases hosted in AWS

Both the application layer and the database(s) are hosted on the same amazon ec2 host (this will change in the future, as the architecture supports databases hosting in RDS or in separate hosts with TCP data channel)

3.2.3. Local Development and testing

The development and testing are done in an ubuntu 18.04 vm running on top of mac - the binary configuration for the vm in described in the bootstrap script.
You could also develop and test in other Unix-like OS('s) - GentOs, MacOS etc, as long as you could figure out how-to install and provision postgres , the required OS binaries and the Perl modules for the application layer, as the provided deployment script has been aimed and tested only for the latest Ubuntu LTS.

3.2.4. Source code in GitHub

The source code for the qto project is hosted in GitHub with the most open licensing possible:
https://github.com/YordanGeorgiev/qto

4. ARCHITECTURE

4.1. ICOCM architecture definition

The Input-Output Control Model architecture is and application architecture providing the highest possible abstraction for almost any software artefact, by dividing its main executing components based on their abstract responsibilities - Input, Conversion, Output , Control and Model.

4.1.1. The Control components

The Control components control the control flow in the application. The instantiate the Model and pass it to the Readers , Converters and Writers for output.

4.1.2. The Model components

The model components model the global DATA of the application - that is no configuration, nor control flow nor anything else should be contained within the model.
Should you encounter data, which is not modelled yet , you should expand the Model and NOT provide different data storage and passing techniques elsewhere in the code base ...

4.1.3. The Input Components

The Input Components are generally named as "Readers". Their responsibility is to read the application data into Model(s). The Input components must be as generic as possible.

4.1.4. The Output Components

The Output Components are generally named as "Writers" Their responsibility is to write the already processed data from the Models into the output media. The output components must be as generic as possible.

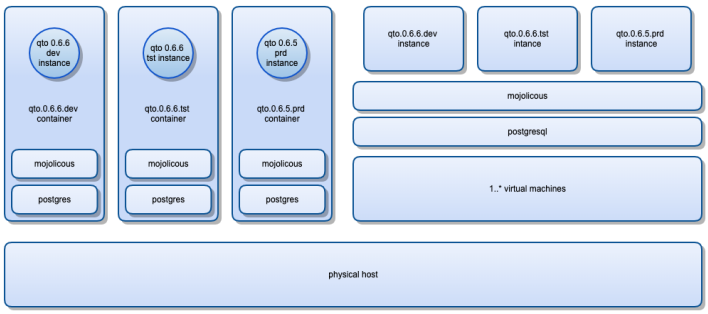
4.1.5. The Conversion Components

The Conversion components are generally called "converters". Their responsibility is to convert from one run-time data structure to another. The conversion might also contain certain business logic.

4.2. Multi-instance setup

The multi-instance setup refers to the capability of any installed and setup instance of the qto application to "know" its version , environment type - development , testing and production) and owner.

Figure: 2 qto multi-instance setup



4.2.1. Multi-environment naming convention

Each database used by the qto application has an <<environment abbreviation>> suffix referring to its environment type. Running application layers against different db versions should be supported as much as possible.

5. FULL STACK DESCRIPTION

5.1. OS stack

The Qto Application supports ONLY Ubuntu 18.04 as of v0.7.9. - that is all the deployment automations have been tested ONLY against this OS. Any middle level full stack developer should be able to deploy and operate the Qto Application on most of the reason Linux distributions and even probably BSD, yet this will require substantial rewrite of the existing deployment scripts as the aim is to have single code base for those scripts and configurability if multiple OS's will be supported.

5.2. The Perl Stack

Qto runs preferably on the newest possible Perl version on Ubuntu, yet "use 5.12.0;" is placed at the entry point of the shell. script. The deployment script installs the latest perl binary for Ubuntu as non-root. Bothe the back-end of the web application and the shell scripts use the same Perl code base.

5.2.1. The perl binary

"use 5.12.0;" is placed at the entry point of the shell. script. The deployment script installs the latest perl binary for Ubuntu as non-root.

5.2.2. The perl modules

All the perl modules installations have been included in the deployment script and over automated installations have been performed without errors. The list of perl modules used by the deployer script to install is stored in the following "list" file:
src/bash/deployer/qto/cnf/bin/perl-modules.lst

5.3. The Mojolicious web framework

Qto uses Mojolicious as it's main engine - the reason is simply the fact that Mojolicious IS the best web framework out there. For those willing to argue this fact, simply extend this statement with the "for the needs of the Qto Application" phrase.

5.4. The browser run-time

Qto strives to support all non-older then 2 years browsers. The majority of the testing is however conducted with Chromium based mobile and desktop browsers.

5.4.1. HTML 5

5.4.2. JavaScript

Simple dynamic tricks have been implemented with vanilla JavaScript to avoid complexity of the Vue applications, by however providing the needed client side reactivity and functionality desired by modern look and feel.

5.4.2.1. Vue

The client side reactivity of the pages is implemented with Vue. As soon as you understand that the vue code "lives" with the Mojolicious templates of the back-end (which as a techniques is not uncommon, that is the back-end templates to be "build" first on the backend, but then to "inject" client-side code) the whole magic of "double-dynamism" turns out to be a systematic approach to reduce complexity by the divide and conquer principle.

5.4.2.2. VueX

The view doc page is implemented with VueX.

5.4.3. name...

5.5. NodeJS

Client side ui automation is implemented with a small npm project using couple of npm modules (mocha and puppeteer)

_

6. APPLICATION CONTROL FLOW

This section provides a generic control flow description for the shell based and ui based control flows.

6.1. Shell control flow

The shell control flow is based on the control model input output architecture. The usual pattern is to call a single <<doSomeAction>> shell function, which is stored in a some-action.func.sh file and loaded dynamically based on this naming convention. Maximun of 2 level nesting is used in the functional calls, that is once an shell action is evoked it could call only 1 funtion, which might or might be not a shell action function , BUT not more, to address unneeded complexity written in bash.

6.1.1. Front-End

The Mojolicious Web Framework runs on top of a perl instance, which serves the back-end requests and passes back and forth json, as well as the ui Mojo templates dynamically, which combined with the vue template create the generic ui loaded in modern html 5, web sockets and ajax capable browsers.
That said the html,js , vue code is mixed with some perl Mojolicious template code several times per file unit and the logic of building the file units into html pages in defined by the Mojolicious templating system, but once one could grasp the concept the developing of the front-end is more or less html,javascript , vue centric.

6.1.2. Back-End

The id's of the tables which ARE VISIBLE to the end users ui are big integers, which are formed by the concatenation of the year, month, day, hour, minutes and second in which the row in the table is created.
The primary keys are however GUID's to provide the underlying expandability for cross-domain, cross-db data transfers.

7. SECURITY

This section provides an overview of the security of a system operating the qto application.

7.1. Non-security mode

In the non-security mode the application does NOT authenticate any one. Both run over https and http. Of course if you use http all the traffic will be in plain text ...

7.2. Simple native security mode

In this mode the authentication is performed against the project defined in the login page (where project is actually the database to which the application layer can connect to). This means that one user can have access to multiple project databases and be authenticated agains some of them thus being able to navigate with the same browser from project to project. In this mode the user credentials - email and password are stored in the users table with a blowfish encryption.
Sessions are used for storing the state of the authentication, which is handled by the Mojolicious web framework - all data gets serialised to json and stored Base64 encoded on the client-side, but is protected from unwanted changes with a HMAC-SHA1 signature.