Table of Contents

REQUIREMENTS

1. DEPLOYABILITY

The issue-tracker must be easily deployable on any Unix like OS. Windows family based OS'es are explicitly out of the scope of the issue-tracker tool. Any issue-tracker instance should be configurable as easily as possible for the version it has.

1.1. A full system deployability in less than an hour

The full System should be ready for use in a "blank" OS host in less than an hour.

1.1.1. Deploy by simple unzip

The issue-tracker tool could be deployed by a simply unzip of the full package, which must have all of the documentation and scripts to provide assistance for the setup and the configuration of the tool.

1.1.2. Oneliner for prerequisite binaries check

All the binaries which are required for the running of the tool must be checked by a user-friendly binaries prerequisites check script

1.1.3. Oneliner for Perl modules check

All the required Perl modules, must be verifiable via a single runnable perl script.

1.1.4. Installation documentation

The installation of the required mysql and postgres db must be documented in the DevOps guide, which should have both markdown and pdf versions in the doc directory of the deployment package.

1.2. A full application clone should be ready for less than 5 minutes

A DevOps operator should be able to perform an application clone of the issue-tracker application in less than 5 minutes.

1.2.1. Shell script for postgres db creation

The creation of the postgres database should be doable via a single shell call.

1.3. A full system deployability in less than an hour

The full System should be ready for use in a "blank" OS host in less than an hour.

2. USER-FRIENDLINESS

The interaction with each endpoint and interface of an application instance should be as user-friendly as possble.
As abstract as it may sound the tool must be multi-dimensionally and vertically integrated regarding the questions what,how and why towards a new person interacting with the tool by the usage of code comments,links from the documentations and uuids to be used for simple grepping from the docs till the source code.

2.1. Oneliner shell calls

The interaction of the application on the shell should be designed and implemented so that most of the features and bigger entry points should be accessible via one-liners on the shell - for example the testers should be able to lunch all the unit-tests via a single one line call. The integration tests should be triggerable via single oneline call.

2.1.1. Database recreation and DDL scripts run one-liners

The developers should be able to create the database via a single oneline call

2.1.2. Table(s) load via aa single one-liner

The developers should be able to load a table to the database via a single oneline call

2.1.3. Testing one-liner call

The testers and the developers should be able to trigger all the unit or integration tests via a single one-line call.

2.1.4. Test messages user

Each test should obey the following convention:
- short message as descriptive within the context as possible - what is being tested
- a short technical example of the generated entry being tested (for example a dynamic url)

- a uuid to search for from the Feature document what exactly is being tested within the context of the features description.

3. RELIABILITY AND STABILITY

3.1. Zero tollerance towards crashing

Crashing in normally configured and operating environment must not be tollerated, as soon as any crash has occured a bug must be registered and the bug set with prio towards the features pipeline.

3.2. Logging

The application shoud support configuratble logging to STDOUT and STDERR for the following levels - debug,info,warn

3.2.1. run execution tracing

3.3. Daily backups

Any instance of the issue-tracker product should be configurable for daily database backups in less than 5 minutes via cron for ANY configured project

```
# add the following one-liner in the cron-scheduled script
# load the proj env vars
doParseCnfEnvVars <<path-to-proj-file>>
clear ; export tables=`curl -s -k http://$web_host:3000/$postgres_db_name/select-tables|jq -r '.| .dat| .[] | .table_name'|perl -ne
's/\s+/,/g;print'`;export do_truncate_tables=1 ; export rdbms_type=postgres ; export load_model=upsert ; perl
src/perl/issue_tracker/script/issue_tracker.pl --do json-to-db --tables $tables
```

4. SCALABILITY

4.1. Feature scalability

The addition of new features should be as scalable as possible.

4.2. Setup scalability

The creation of new instances of the application should be as easy as possible.

5. PERFORMANCE

5.1. Page load maximum time

Each page of the application should load for less than 0.3 seconds.

6. MULTI-INSTANCE OPERABILITY AND DEPLOYABILITY

6.1. Environment type self-awareness

Each deployed and running instance of the issue-tracker must "know" its own environment type - dev,tst, qas or prd to comply with the multi-instance architecture on a single host.

6.2. Cross running between instances of different types

The application layers should support as much as possible cross running between different application layer instances and database instances - for example a dev appalication layer should be able to fetch data from a prd database.

7. APPLICATION BACK-END REQUIREMENTS

7.1. List action

7.1.1. list-tables route

The application back-end should be capable of retrieving the list of tables of any database the issue-tracker application layer has access to.
http://192.168.56.120:3000/dev_issue_tracker/list-tables

7.1.2. list <<table>> route

The application back-end should be capable of retrieving the results set of the contents of a full table
http://192.168.56.120:3000/dev_issue_tracker/list/montly_issues

8. DATA REQUIREMENTS

8.1. CRUD

Any modelled item in the database must be capable for:
- create
- update
- delete
- search

9. UI REQUIREMENTS

9.1. Clarity on errors

The UI must present every error in a clear and consice way, so that the end-user would understand that an error has occured, however no msgs should be displayed when the data is saved properly.

9.2. CRUDS

The System must provide the needed UI interfaces to Create , Update , Delete and Search items in the system for the users having the priveledges for those actions

10. DOCUMENTATION

10.1. Documentation set

Each running instance MUST have the following documentation set :
- Features and Functionalities doc
- DevOps Guide
- Requirements
- SystemGuide

in at least the md and pdf file formats.

10.2. Always up-to-date

Each running instance MUST have its required documentation set up-to-date. No undocumented or hidden features are allowed.