Pine Framework

Pine framework has a big mission, and it is, write business once independent of tools, framework, database, application server,... then use the implemented business into different architecture or structure.

The framework comprised three main part as follows:

- Core component
- Core implementation component (like JEE, Spring, ...)
- Product component

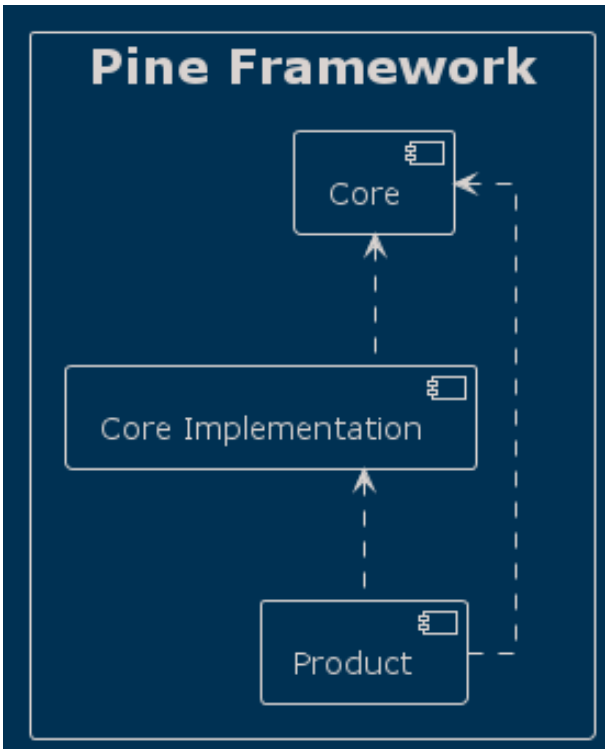

Figure 1. Pine Framework components

1. Core Component

Core component is a basic component so that other productions and the specific implementations must be implemented that, therefore they depend on the Core component indeed **Core component is a main component**.

It included a few components and all of them are written in pure Java language. It is an abstract component, means, it has provided facilities, abstractions, utilities, ... in order to develop productions as much as soon.

Core components are as follows:

- document

- i18n
- helper
- model
- contract
- business
- test

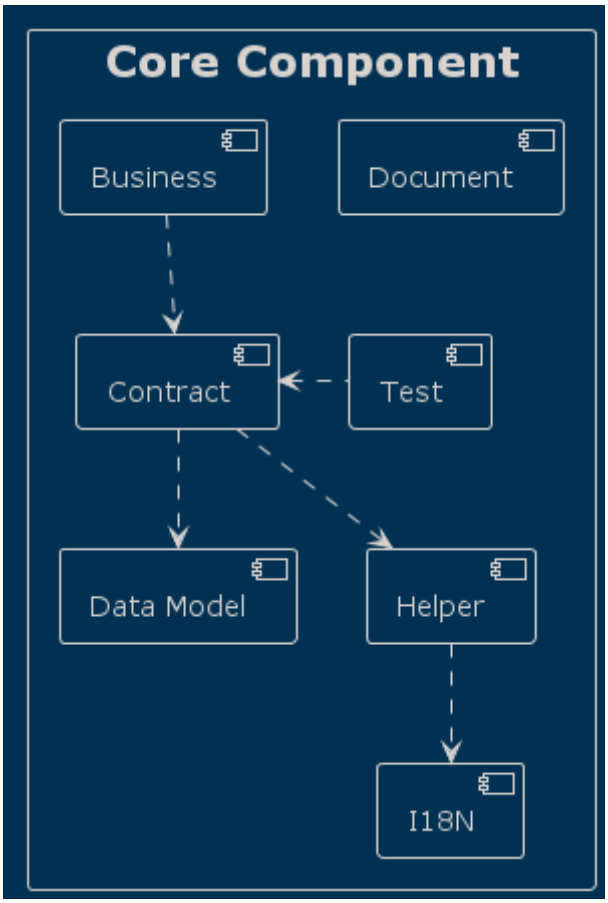

Figure 2. Core components

The Core component can support four environments, and they are as follows:

- Development
- Test
- UAT
- Production

1.1 I18n

This component has designed for adding internationalization ability to Pine framework. The component is included name of fields, error messages, info messages, etc. It currently supports English and Persian languages as a built-in languages, therefore in order to add the other language you have to add a properties file to resources folder.

If you want to set default message file then you have to create a properties file with **i18n_lan.properties** format. It means the name must be started with *i18n* plus underscore sign (_) *and two alphabet language code. Then, to introduce file you have to define an environment variable named _I18N_LANG* with two alphabet language code as a value.

Linux/Unix

```
export I18N_LANG=en
echo "export I18N_LANG=en" >> ${HOME}/.bashrc
```

Windows

```
set I18N_LANG=en
setx /M I18N_LANG "en"
```

Also, you can add more files with different start part and same language that you defined via *I18N_LANG*. To get this, you must add files to resources folder then define an environment variable named *I18N_FILES* with name of files as a value. The value supports comma separate.

Linux/Unix

```
export I18N_FILES="test_i18n1,test_i18n2,..."
echo "export I18N_FILES=test_i18n1,test_i18n2,..." >> ${HOME}/.bashrc
```

Windows

```
set I18N_FILES="test_i18n1,test_i18n2,..."
setx /M I18N_FILES "test_i18n1,test_i18n2,..."
```