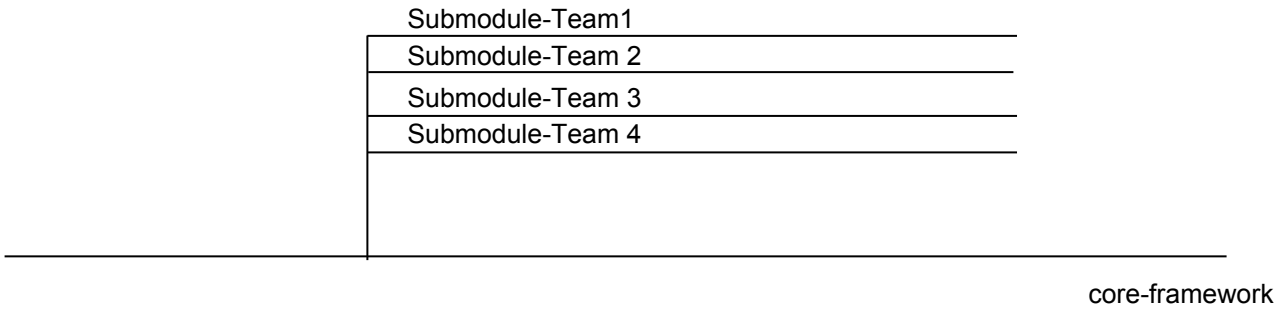
Application

Governance - by John marquez

Index

Index

App Structure. NPM modules: Submodules



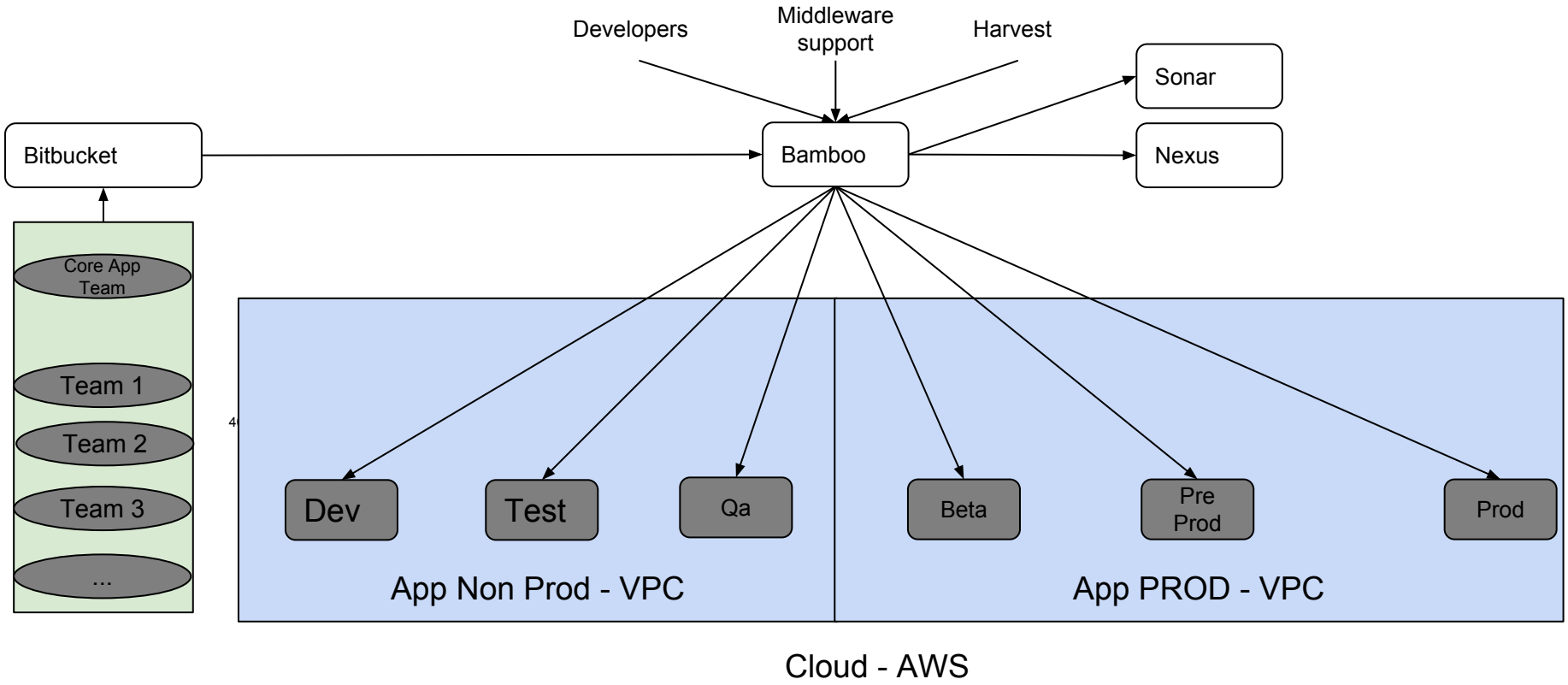
- One repository per project
- One shared Dev/IT/QA
- Small deployments
- Each project manage its own deployments
- Each team manage its own integrations

Each project will publish to nexus and maintain its own versions in the package.json file that exists in each repository.

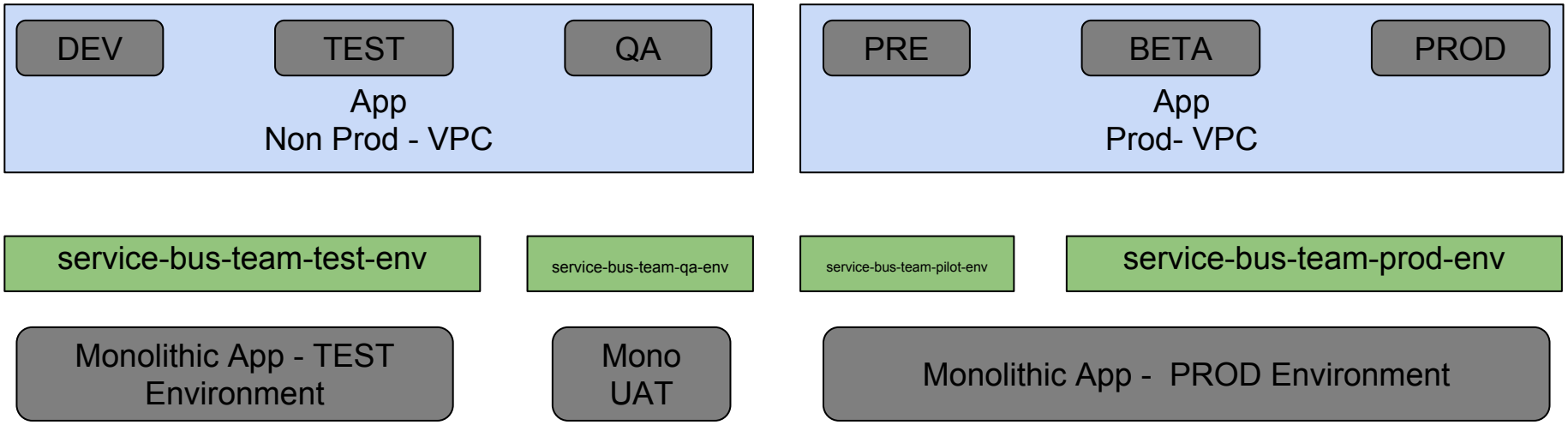
Framework will remove the links to each subrepository, deployments will be handled using the module version published in nexus.

Index

Continuous integration with BitBucket & Bamboo

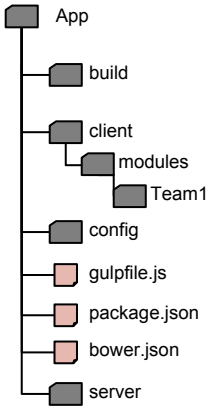


App Environments. Mapping to service bus Team/ Monolithic application Team



Index

Project Structure



Gulp

Gulp is the task automation software used to create the builds that will be used to deploy to the different environment. There are many tasks defined already that can be used and more to come.

Package.json

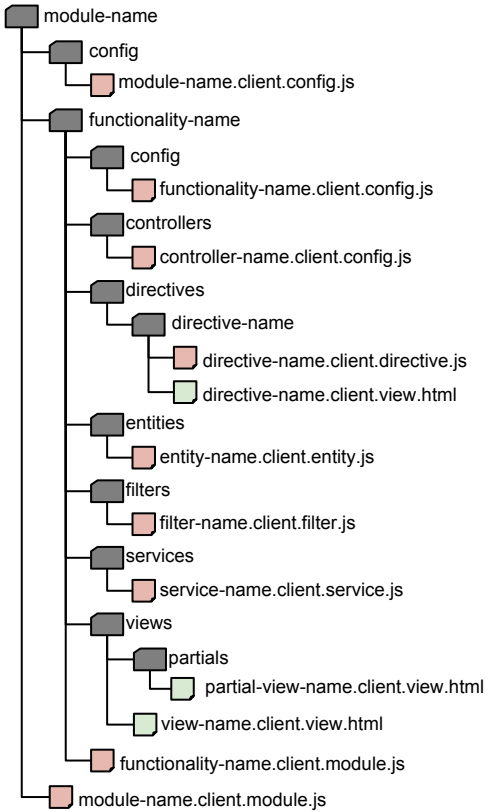
Package.json store all the **node dependencies** of the project, if you need to find what **version** of a **library** is being used, this is the place to look. If you need a new library, please request it to the Core App-Framework team.

Bower.json

Bower is the **library** management tool that is being used to manage **angular** libraries. If you need a new library, please request it to the Core App-Framework team.

Index

Module structure



Folder names

- **Lowercase** (dashboard).
- In case the directory name contains multiple words, use **lisp-case syntax** (manage-work).

Module structure

High-level divisions by module and functionality and lower-level divisions by component types.

- **Module level**: module name (flo).
- **Functionality level**: functionality name (dashboard).
- **Component level**: we can create nested folders inside every component. Most common components:
  - controllers
  - directives
  - entities (factories)
  - filters
  - services
  - views
- **Config folder**: we can place a config folder in the module level, in the functionality level or both. It's used to define:
  - **Routes and menu items.**
  - **Logging preferences** per environment.
  - Other configurations.
- **Unit tests**: The unit tests for a given component should be located in the directory where the component is. This way when you make changes to a given component finding its test is easy. The tests also act as documentation and show use cases.

Module structure. Other Folders

- ***core***: Core folder is the **application core**. This folder doesn't follow the guidelines and will be **manage by 'Core-App-Framework' team**. It contains:

    - **Generic functionality** (like user session).
    - **Generic directives, filters, controllers, etc.**
    - **Backend communication services** (complaints, notes, etc.).
    - **General services/entities**: they can be extended using wrappers.
    - **Special services**: logging, push, security, etc.

- ***tasks***: Task folder contains **all the actions and task used in Core**. It's not placed in core folder because will be **shared for different teams**.

Index

Environment configuration

- ***Where?***

  Part of the environment configuration is defined in ***/config/env* folder**, the other part is stored in Mongodb. All the configuration is accessible through the service: “/Team1/directives/configuration/”

- ***Config files***
  - There is **a file for each environment** in the **server**.

- ***Environment listing***
  - development
  - test
  - qa
  - beta
  - Production

Preproduction environment uses the production configuration so the test in preprod ensures that the application will work in production.

Environment configuration. Example

```
'use strict';

var backend = {
          url: 'http://service-bus-test1.company.com:5555/ws/',
          username: 'tester004',
          password: 'Qat3ster1'
};

module.exports = {
          db: 'mongodb://ualvtstrhel1.company.com/digitaldesktop-dev',
          //db: 'mongodb://localhost/digitaldesktop-dev',
          port: ((process.env.NODE_DEPLOY=='framework')? 4001 : 4000 ),
          app: {
                    title: 'Digital Desktop - Development Environment'
          },
          backend: backend,
          webServicesConfigurations: [
          name: 'AppUsersServices',{
                              name: 'AppUsersServices',
                              wsdl: backend.url +
'cb_SecurityAccess.v5_1.ws.producer:securityaccess?WSDL', // The url to WSDL
                              security: {
                                        username: backend.username,
                                        password: backend.password
                              }
                    },

...
```

Setting options

- **Database access**.
- Running **port**.
- Application **title**.
- **Service Bus backends**:
  - Environment url.
  - Security (user and password).

If a new wsdl is needed, notify the Core-App-Framework team to make sure it exists in all environments and request access for the application.

- **Service endpoints**:
  - Service name.
  - WSDL url.

Logging configuration

```
.module('ModManager')
.constant('ModConfiguration', {
        name: 'Mod',
        development: {
                debug: {console: true, http: false},
                info: {console: true, http: true},
                warn: {console: true, http: true},
                error: {console: true, http: true}
        },
        test: {
                debug: {console: true, http: false},
                info: {console: true, http: false},
                warn: {console: true, http: true},
                error: {console: true, http: true}
        },
        qa: {
                debug: {console: true, http: false},
                info: {console: true, http: false},
                warn: {console: true, http: true},
                error: {console: true, http: true}
        },
        production: {
                debug: {console: false, http: false},
                info: {console: false, http: false},
                warn: {console: false, http: true},
                error: {console: false, http: true}
        }
});
```

Where?

Logging configuration is defined in ***/config/module-name.client.config.js*** for each module.

What?

- ***console***: for **debug** purpose (not in production).
- ***http***: very **important info**, warns or errors. All requests/responses are automatically save on server side.

Setting options

- **Environment**: development, test, qa, production.
- **Level**: debug, info, warn, error.
- **Client transport**: console, http.

Index

Style guidelines. Naming convention

Naming Conventions

AngularJS elements

Element	Naming Style	Examples
Modules *	mainModule.lowerCamelCase	flo.dashboard, flo.manageWork
Controllers	Functionality + Controller	DashboardController
Directives	bbva + UpperCamelCase	bbvaTicketDetails
Filters	lowerCamelCase	jsonDate
Entities (factories)	UpperCamelCase	User
Constants	UpperCamelCase	ComponentsId
Services	UpperCamelCase + Service	ComplaintsService

***Note:** use the hierarchy to create module name to avoid namespace collision (mod.dashboard, users.dashboard, etc.)

Other elements

- **Routes**: lowerCamelCase.
- **Variables**: lowerCamelCase.
- **Constants** attributes: UPPERCASE.
- **Functions**: lowerCamelCase.
- **Push channels**: element id (groupCd, userCd, ticketNr, etc.)

Style guidelines. Code Comments

Comments

Use jsDoc syntax to document function names, description, params and returns.

```
/**
  * @name logError
  * @desc Logs errors
  * @param {String} msg Message to log
  * @returns {String}
  * @memberOf Factories.Logger
  */
 function logError(msg) {
     var loggedMsg = 'Error: ' + msg;
     $log.error(loggedMsg);
     return loggedMsg;
 };
```

Style guidelines. Syntax

Controllers

- Use **Controller as syntax**.

```
function CustomerController() {
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}

…

<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

Directives

- Use **scope instead of $scope** in your link function.
- **DOM manipulations** must be done only through directives.
- Create an **isolated scope** when you develop reusable components.
- Use directives as **attributes or elements instead of comments or classes**.
- Use **scope.$on('$destroy', fn)** for cleaning up.

Other rules

- Use:
  - **$timeout** instead of setTimeout
  - **$interval** instead of setInterval
  - **$window** instead of window
  - **$document** instead of document
  - **$http** instead of $.ajax
  - **$location** instead of window.location or $window.location
  - **$cookies** instead of document.cookie
- Use **promises ($q) instead of callbacks**.
- Use **services instead of $rootScope**.

Index

Security implementation

```
...
$stateProvider.
state('mod.dashboard', {
          url: '/dashboard',
          templateUrl: 'modules/mod/dashboard/views/dashboard.client.view.html',
          resolve: {
                    auth: ['$q', '$location', 'UserService',
                    function($q, $location, UserService) {
                              return UserService.hasAccess(['CORE-MOD-DASHBOARD-READ']).then(
                                        function(success) {},
                                        function(err) {
                                                  ...
                                                  return $q.reject(err);
                              });
                    }]
          },
          data: {
                    generalInfo: {
                              name: 'MOD Dashboard',
                              menu: {
                                        label: 'MOD Dashboard',
                                        iconClasses: 'glyphicon glyphicon-dashboard',
                                        url: '#!/dashboard',
                                        parent: 'none',
                                        order: 0
                              }
                    },
                    accessInfo: {
                              sam: {
                                        allow: ['COMPLIANCE_USR', 'CCU_BASIC_CAE'],
                                        deny: ['COLLECTIONS_USR', 'TBINSALESWKND']
                              },
                              ldap: {
                                        allow: [‘CORE-MOD-DASHBOARD-READ']
                              }
                    }
          }
...
```

Where?

Logging configuration is defined in ***/config/module-name.client.config.js*** for each module.

Setting options

For each route, **allow/deny access** using **SAM and LDAP**.

Schema

- **name**: route name.
- **url**: route url.
- **templateUrl**: route view.
- **resolve**: user service functions.
- **data**: json object to define the menu item:
  - **General info**: name, url, icon, parent, order, etc.
  - **Access info**:
    - **SAM info**:
      - Type: Allow/Deny
      - Array of SAM groups for this route.
    - **LDAP info**:
      - Type: Allow/Deny
      - Array of LDAP labels for this route.