Table of Contents

Angular CI

TESTING LINKS

refer to page 2 for more info

test link without asciidoc

test link

test link with asciidoc

test link

test sublink in minus with asciidoc

test sublink in mayus before asciidoc

test sublink in minus before asciidoc

The Angular client-side of My Thai Star is going to have some specific needs for the CI-CD Pipeline to perform mandatory operations.

Pipeline

The Pipeline for the Angular client-side is going to be called **MyThaiStar_FRONTEND_BUILD** . It is located in the PL instance, under the MTS folder (as previously explained). It is going to follow a process flow like this one:

angular pipeline flow

Each of those steps are called *stages* in the Jenkins context.Let’s see what those steps mean in the context of the Angular application:

1. **Declarative: Checkout SCM**
2. **Declarative: Tool Install**
3. **Loading Custom Tools**
4. **Fresh Dependency Installation**
5. **Code Linting**
6. **Execute Angular tests**
7. **SonarQube code analysis**
8. **Build Application**
9. **Deliver application into Nexus**
10. Declarative: Post Actions

Adjustments

The Angular project Pipeline needed some "extra" features to complete all planned processes. Those features resulted in some additions to the project.

Pipeline Environment

In order to easily reuse the pipeline in other angular projects, all variables have been defined in the block environment. All variables have the default values that Production Line uses, so if you’re going to work in production line you won’t have to change anything. Example:

```
environment {
    // Script for build the application. Defined at package.json
    buildScript = 'build --configuration=docker'
    // Script for lint the application. Defined at package.json
    lintScript = 'lint'
    // Script for test the application. Defined at package.json
    testScript = 'test:ci'
    // Angular directory
    angularDir = 'angular'
    // SRC folder. It will be angularDir/srcDir
    srcDir = 'src'
    // Name of the custom tool for chrome stable
    chrome = 'Chrome-stable'

    // sonarQube
    // Name of the sonarQube tool
    sonarTool = 'SonarQube'
    // Name of the sonarQube environment
    sonarEnv = "SonarQube"

    // Nexus
    // Artifact groupId
    groupId = 'com.devonfw.mythaistar'
    // Nexus repository ID
    repositoryId = 'pl-nexus'
    // Nexus internal URL
    repositoryUrl = 'http://nexus3-core:8081/nexus3/repository/maven-snapshots'
    // Maven global settings configuration ID
    globalSettingsId = 'MavenSettings'
    // Maven tool id
    mavenInstallation = 'Maven3'
}
```

Description

- **buildScript** : script for build the applicaction. It must be defined at package.json.
- **lintScript** : Script for lint the application. Defined at package.json
- **testScript** : Script for test the application. Defined at package.json
- **angularDir** : Relative route to angular application. In My Thai Star this is the angular folder. The actual directory (.) is also allowed.
- **srcDir** : Directory where you store the soruce code. For angular applications the default value is `src`
- **chrome** : Since you need a browser to run your tests, we must provide one. This variable contains the name of the custom tool for google chrome.
- **sonarTool** : Name of the sonarQube scanner installation.
- **sonarEnv** : Name of the sonarQube environment. SonarQube is the default value for PL.
- **groupId** : Group id of the application. It will be used to storage the application in nexus3
- **repositoryId** : Id of the nexus3 repository. It must be defined at maven global config file.
- **repositoryUrl** : The url of the repository.
- **globalSettingsId** : The id of the global settings file.
- mavenInstallation: The name of the maven tool.

Angular design

Introduction

MyThaiStar client side has been built using latest frameworks, component libraries and designs:

Angular *4* as main front-end Framework. https://angular.io/

Angular/CLI *1.0.5* as Angular tool helper. https://github.com/angular/angular-cli

Covalent Teradata *1.0.0-beta4* as Angular native component library based on Material Design. https://teradata.github.io/covalent//

Angular/Material2 *1.0.0-beta5* used by Covalent Teradata. https://github.com/angular/material2

Note: this dependencies are evolving at this moment and if it is possible, we are updating it on the project.

Basic project structure

The project is using the basic project seed that Angular/CLI provides with â€œng new <project name>â€. Then the app folder has been organized as Angular recommends and goes as follows:

- app
  - components
    - sub-components
    - shared
    - component files
  - main app component
- assets folder
- environments folder
- rest of angular files

This structure can be shown in the following example image:

folder organization

Main Views and components

List of components that serve as a main view to navigate or components developed to make atomically a group of functionalities which given their nature, can be highly reusable through the app.

routes

Note: no-name-route corresponds to whatever URL the user introduced and does not exist, it redirects to HomeComponent.

Public area

AppComponent

Contains the components that are on top of all views, including:

Order sidenav

Sidenav where selected orders are displayed with their total price and some comments.

Navigation sidenav (only for mobile)

This sidenav proposal is to let user navigate through the app when the screen is too small to show the navigation buttons on the header.

Header

It contains the title, and some other basic functions regarding open and close sidenavs.

Footer (only for desktop)

At the end of the page that shows only when open on desktop.

HomeComponent

Main view that shows up when the app initializes.

MenuComponent

View where the users can view, filter and select the dishes (with their extras) they want to order it contains a component to each menu entry:

Menu-card

This component composes all the data of a dish in a card. Component made to display indeterminate number of dishes easily.

BookTableComponent

View to make book a table in a given data with a given number of assistants or create a reservation with a number of invitations via email.

Book-table-dialog

Dialog which opens as a result of fulfilling the booking form, it displays all the data of the booking attempt, if everything is correct, the user can send the information or cancel if something is wrong.

Invitation-dialog

Dialog which opens as a result of fulfilling the invitation form, it displays all the data of the booking with friends attempt, if everything is correct, the user can send the information or cancel if something is wrong.

UserArea

Group of dialogs with the proposal of giving some functionalities to the user, as login, register, change password or connect with Twitter.

Login-dialog

Dialog with a tab to navigate between login and register.

Password-dialog

Functionality reserved to already logged users, in this dialog the user can change freely their password.

Twitter-dialog

Dialog designed specifically to connect your user account with Twitter.

Waiter cockpit area

Restricted area to workers of the restaurant, here we can see all information about booked tables with the selected orders and the reservations with all the guests and their acceptance or decline of the event.

OrderCockpitComponent

Data table with all the booked tables and a filter to search them, to show more info about that table you can click on it and open a dialog.

Order-dialog

Complete display of data regarding the selected table and its orders.

ReservationCockpitComponent

Data table with all the reservations and a filter to search them, to show more info about that table you can click on it and open a dialog.

Reservation-dialog

Complete display of data regarding the selected table and its guests.

Email Management

As the application send emails to both guests and hosts, we choose an approach based on URLâ€™s where the email contain a button with an URL to a service in the app and a token, front-end read that token and depending on the URL, will redirect to one service or another. For example:

`http://localhost:4200/booking/cancel/CB_20170605_8fb5bc4c84a1c5049da1f6beb1968afc`

This URL will tell the app that is a cancelation of a booking with the token *CB_20170605_8fb5bc4c84a1c5049da1f6beb1968afc* . The app will process this information, send it to back-end with the correct headers, show the confirmation of the event and redirect to home page.

The main cases at the moment are:

Accept Invite

A guest accept an invitation sent by a host. It will receive another email to decline if it change its mind later on.

Reject Invite

A guest decline the invitation.

Cancel Reservation

A host cancel the reservation, everybody that has accepted or not already answered will receive an email notifying this event is canceled. Also all the orders related to this reservations will be removed.

Cancel Orders

When you have a reservation, you will be assigned to a token, with that token you can save your order in the restaurant. When sent, you will receive an email confirming the order and the possibility to remove it.

Services and directives

Services are where all the main logic between components of that view should be. This includes calling a remote server, composing objects, calculate prices, etc.

Directives are a single functionality that are related to a component.

As it can be seen in the basic structure, every view that has a minimum of logic or need to call a server has its own service located in the shared folder.

Also, services and directives can be created to compose a reusable piece of code that will be reused in some parts of the code:

Price-calculator-service

This service located in the shared folder of sidenav contains the basic logic to calculate the price of a single order (with all the possibilities) and to calculate the price of a full list of orders for a table. As this is used in the sidenav and in the waiter cockpit, it has been exported as a service to be imported where needed and easily testable.

Authentication

Authentication services serves as a validator of roles and login and, at the same time, stores the basic data regarding security and authentication.

Main task of this services is to provide visibility at app level of the current user information:

- Check if the user is logged or not.
- Check the permissions of the current user.
- Store the username and the JWT token.

SnackService

Service created to serve as a factory of Angular Material Snackbars, which are used commonly through the app. This service accepts some parameters to customize the snackBar and opens it with this parameters.

WindowService

For responsiveness reasons, the dialogs have to accept a width parameter to adjust to screen width and this information is given by Window object, as it is a good practice to have it in an isolated service, which also calculates the width percentage to apply on the dialogs.

Equal-validator-directive

This directive located in the shared folder of userArea is used in 2 fields to make sure they have the same value. This directive is used in confirm password fields in register and change password.

Mock Backend

To develop meanwhile a real back-end is being developed let us to make a more realistic application and to make easier the adaptation when the backend is able to be connected and called. Its structure is as following:

back end

Contains the three main groups of functionalities in the application. Every group is composed by:

- An **interface** with all the methods to implement.
- A **service** that implements that interface, the main task of this service is to choose between real backend and mock backend depending on an environment variable.
- **Mock backend service** which implements all the methods declared in the interface using mock data stored in a local file and mainly uses Lodash to operate the arrays.
- **Real backend service** works as Mock backend but in this case the methods call for server rest services through http.

Booking

The booking group of functionalities manages the calls to reserve a table with a given time and assistants or with guests, get reservations filtered, accept or decline invitations or cancel the reservation.

Orders

Management of the orders, including saving, filtering and cancel an order.

Dishes

The dishes group of functionalities manages the calls to get and filter dishes.

Login

Login manages the userArea logic: login, register and change password.

SAP Hana

Setting up MTSJ angular

update the following property in config file in my-thai-star\angular\src\app\core\config

```
enablePrediction: true,
```

Security

My Thai Star security is composed by two main security services:

Auth-guard

Front-end security approach, this service implements an interface called CanActivate that comes from angular/router module. CanActivate interface forces you to implement a canActivate() function which returns a Boolean. This service checks with the AuthService stored data if the user is logged and if he has enough permission to access the waiter cockpit. This prevents that a forbidden user could access to waiter cockpit just by editing the URL in the browser.

JWT

Jason Web Token consist in a token that is generated by the server when the user logs in, once provided, the token has to be included in an Authentication header on every Http call to the rest service, otherwise it will be forbidden. JWT also has an expiration date and a role checking, so if a user has not enough permissions or keeps logged for a long certain amount of time that exceeds this expiration date, the next time he calls for a service call, the server will return an error and forbid the call. You can log again to restore the token.

HttpClient

To implement this Authorization header management, an HttpClient service has been implemented. This services works as an envelope of Http, providing some more functionalities, likes a header management and an automatically management of a server token error in case the JWT has expired, corrupted or not permitted.