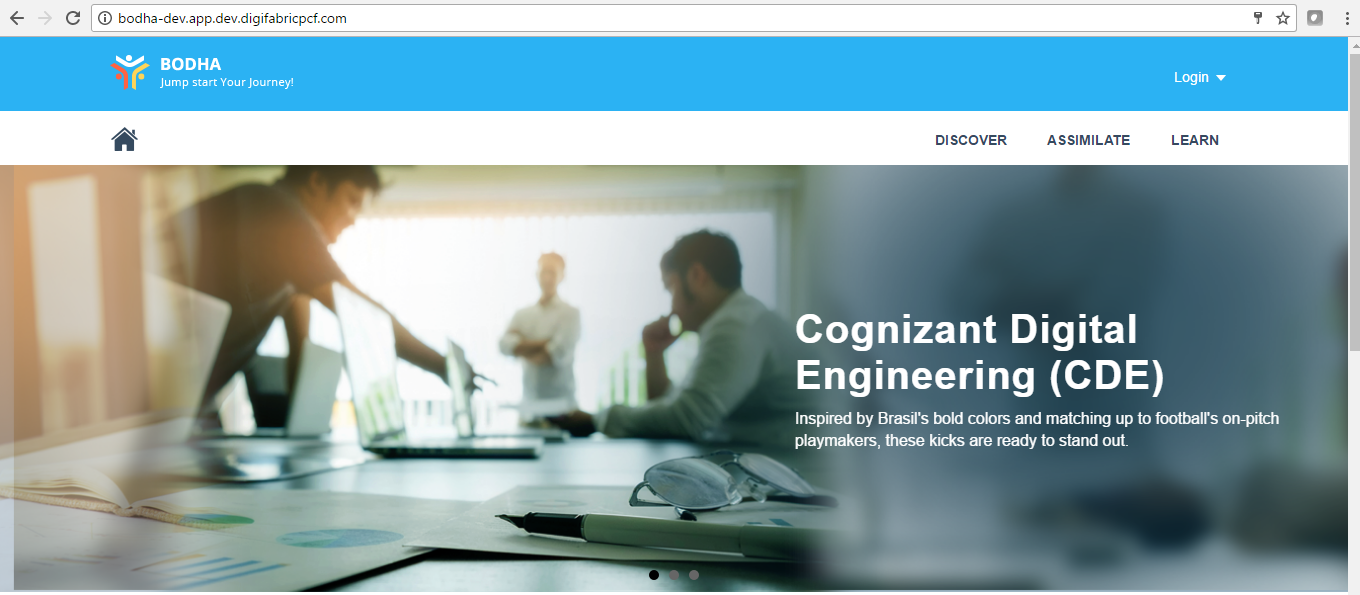
**About the Onboarding Application**

Onboarding application backend is written in Spring Boot and UI is in ReactJS and Database we have used MySQL.

In this application we have Admin as well as Non-Admin users, admin users will be able to add/modify admin level operations such as add/Edit/Delete **Roles**, add/Edit/Delete **Users**, add/Edit/Delete **Substage**, View **Stages** and add/Edit/Delete **Contents**. We have Stages namely **DISCOVER, ASSIMILATE and LEARN**.

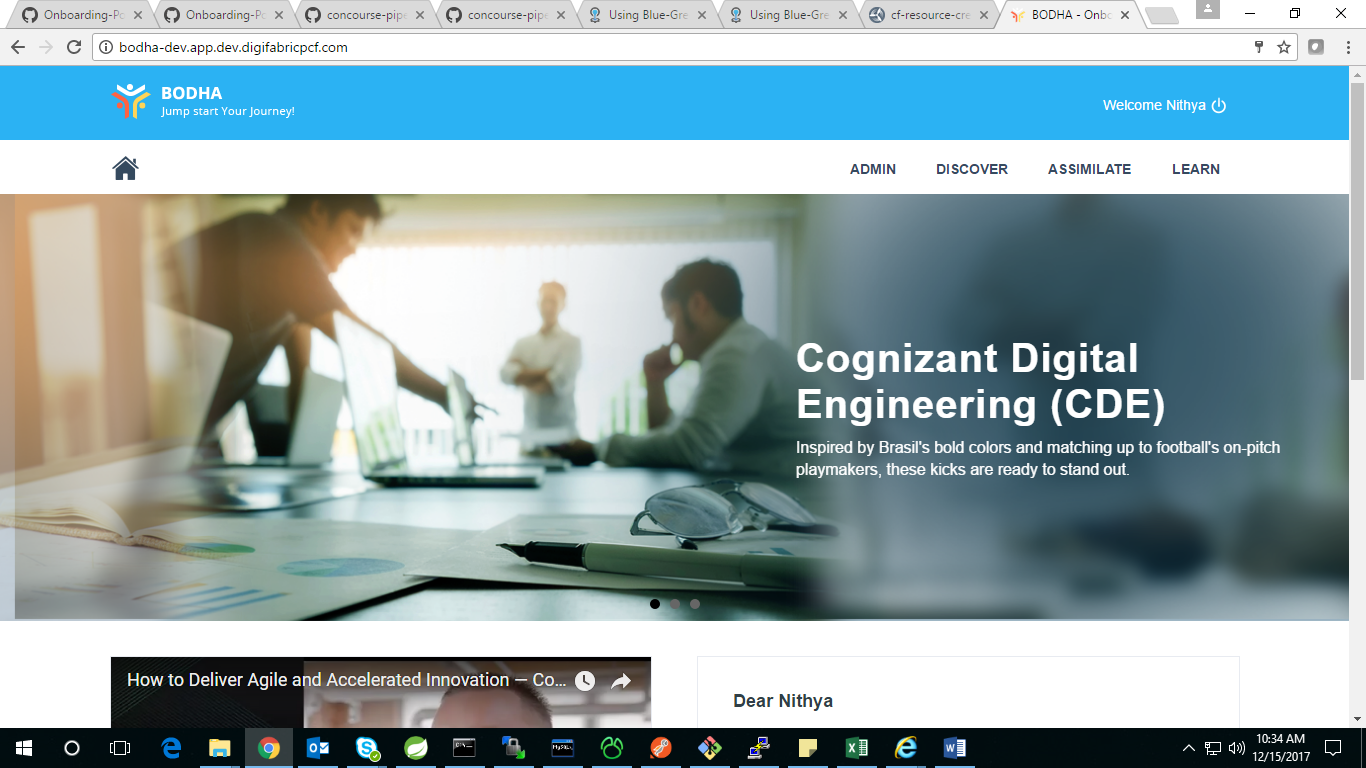
Content can be added to each Substage and assign which ever role admin wants to give access.

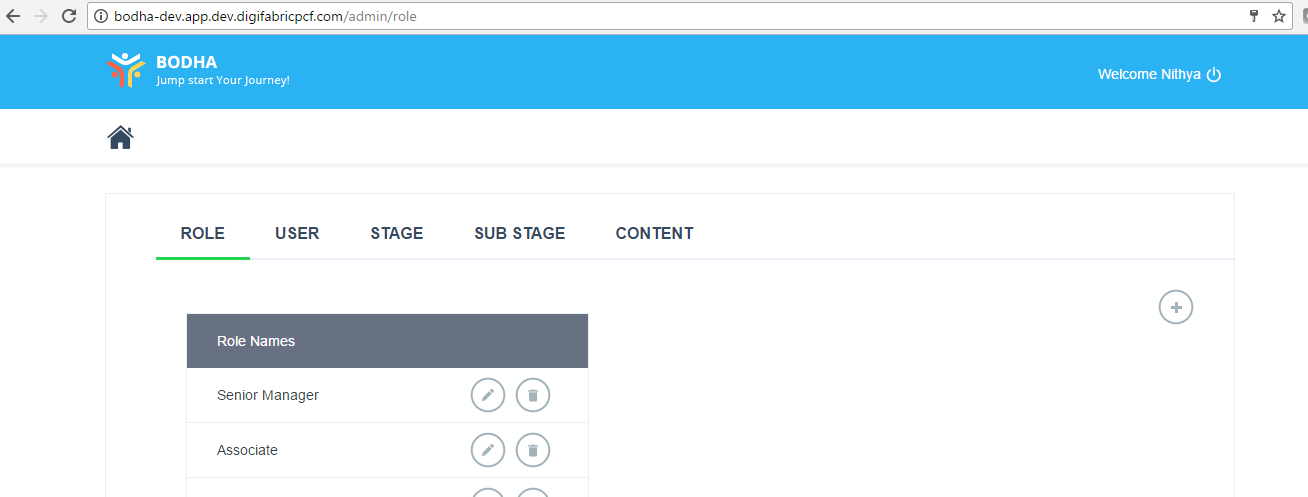
Non admin users can login and see the Stages, Substages and Contents which is given access to them by admin.



**Admin Page View:**

On clicking on Admin tab on the top it will list down all the admin privileges such as add/Edit/Delete **Roles**, add/Edit/Delete **Users**, add/Edit/Delete **Substage**, View **Stages** and add/Edit/Delete **Contents**





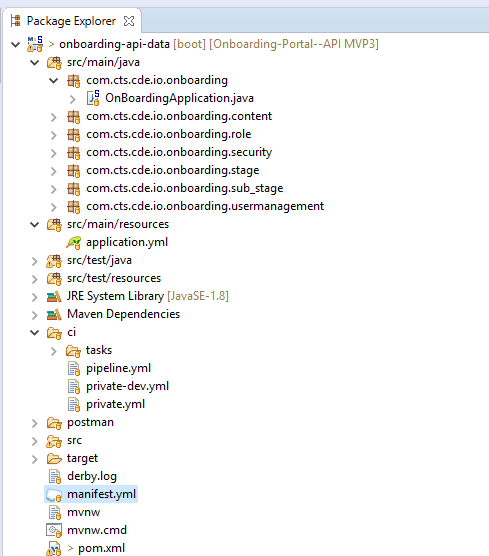
**Technical** **Stack**

|  |
| --- |
| Technology |
| Spring Boot |
| Java |
| MySQL |
| ReactJS, Redux |
| Maven |
| JUnit |
| Concourse |
| Fly |

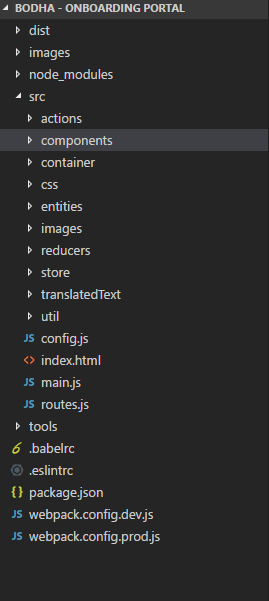
**Application Details**

1. **Project Structure:**

**API Application**

****

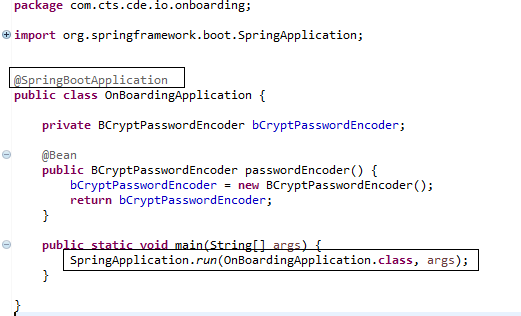
**UI Application**



1. **SpringBootApplication**

**Created SpringBootApplication and main class named as OnBoardingApplication.java**

Spring Boot provides a number of “Starters” that make easy to add jars to your classpath. Our application has already used spring-boot-starter-parent in the parent section of the POM. The spring-boot-starter-parent is a special starter that provides useful Maven defaults. It also provides a dependency-management.

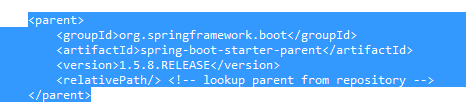
****

1. [**POM.xml**](file:///D:\OnBoarding\Onboarding-Portal--API\pom.xml)

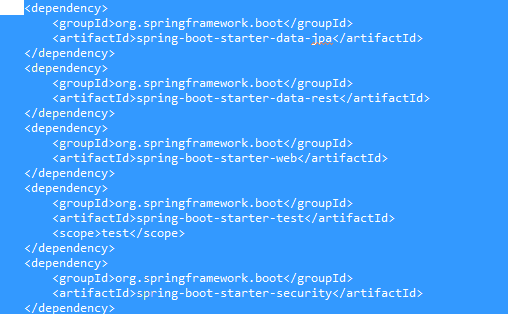
We need to start by creating a Maven pom.xml file. The pom.xml is the recipe that will be used to build your project

**Added spring-boot-starter-parent in pom.xml**

* Added Spring Boot related starters in the POM.
* Also Spring Test related starters are added.

****

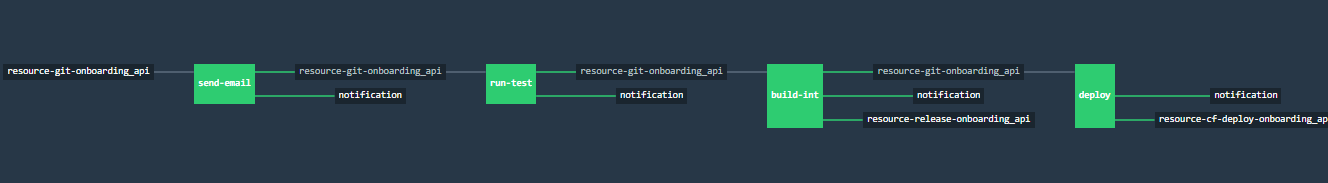
* **Add the dependencies**

****

1. **Create CI-CD Pipeline**

Concourse pipeline that builds, tests and deploys application to PCF.

|  |  |
| --- | --- |
| **Issue** | CI-CD pipeline doesn’t exists |
| **Impact** | Manual deployment every time |
| **Remediation** | Create CI-CD pipeline in pipeline.yml file along with private.yml  **Challenges and Solution:**   * Connecting Github from the Concourse. Found the way to generate private key and use it. * Creating the Release in github. * Created separate pipeline for DEV and PROD. * Added send mail, run test , build and deploy jobs. |

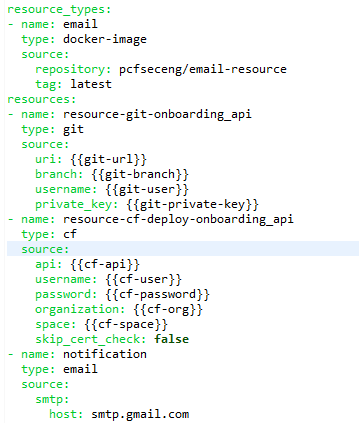
****

Above, the black boxes are [resources](https://concourse.ci/concepts.html#resources), and the colored boxes are [jobs](https://concourse.ci/concepts.html#jobs), whose color indicates the status of their most recent [build](https://concourse.ci/concepts.html#job-builds).

[**Pipeline.yml**](file:///D:\OnBoarding\Onboarding-Portal--API\ci\pipeline.yml)

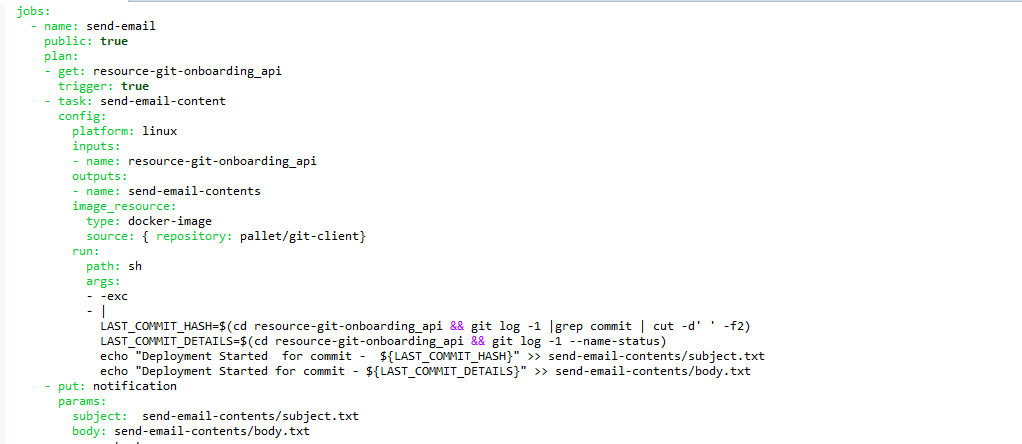
A pipeline is configured with two sections: [resources](https://concourse.ci/configuring-resources.html) and [jobs](https://concourse.ci/configuring-jobs.html).

Resources: [Resources](https://concourse.ci/concepts.html#resources) are the objects that are going to be used for jobs in the pipeline. They are listed under the resourceskey in the pipeline configuration.



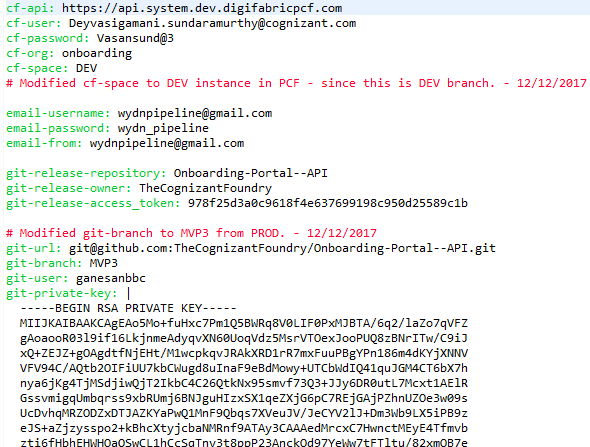
Jobs:

Jobs: [Jobs](https://concourse.ci/concepts.html#jobs) determine the actions of your pipeline, how resources progress through it, and how everything is visualized. They are listed under the jobs key in the pipeline configuration.



[**Private.yml**](file:///D:\OnBoarding\Onboarding-Portal--API\ci\private.yml)

Each pipeline is configured with secret PCF/Docker/GitHub **credentials.** Which we can configure using private.yml file.



To access github from the pipeline create private key from github and give it in the private.yml.

<https://help.github.com/articles/connecting-to-github-with-ssh/>

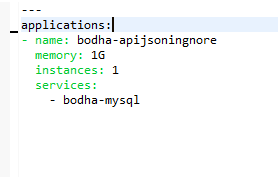
1. **Create manifest.yml**

By default, the cf push command deploys an application to cloud using a manifest.yml file in the current working directory. It can contain application name, No of instances, Memory required for the application, routes name and environmental variables and services needs to be bind with the application

**We have created** [**manifest.yml**](file:///D:\OnBoarding\Onboarding-Portal--API\manifest.yml) **with the following details:**

**API**

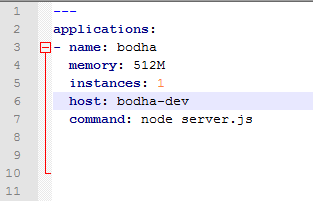
|  |  |
| --- | --- |
| **Issue** | CF Push related information not exists |
| **Impact** |  |
| **Remediation** | Create manifest .yml with the following information which contains basic need for CF Push.   * Application Name * Instances * Memory * Services |

****

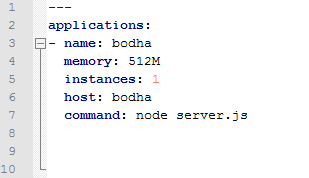
**UI**

|  |  |
| --- | --- |
| **Issue** | CF Push related information not exists |
| **Impact** |  |
| **Remediation** | Create manifest .yml with the following information which contains basic need for CF Push.   * Application Name * Instances * Memory * Host * command |

**Dev App**



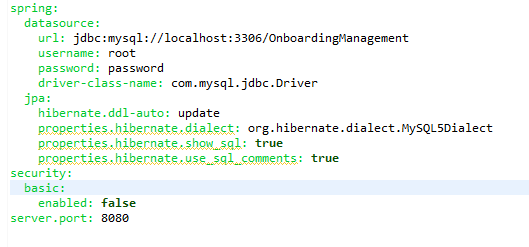
**Prod App**



1. **Create** [**application.yml**](file:///D:\OnBoarding\Onboarding-Portal--API\src\main\resources\application.yml)

Spring Boot allows you to externalize your configuration so you can work with the same application code in different environments. You can use properties files, YAML files, environment variables and command-line arguments to externalize configuration.

|  |  |
| --- | --- |
| **Issue** | Environment related application properties not found |
| **Impact** |  |
| **Remediation** | Create environment related application.yml file   * Specify DB URL, usr/pwd (Local : specify explicitly. For cloud : take it from VCAP) * The below environment specific files are created.   + application.yml |



1. **Test**

**Create Test for every classes and also done integration test.**

|  |  |
| --- | --- |
| **Issue** | Tests to be enhanced |
| **Impact** |  |
| **Remediation** | Create Test class and integration test and use application-test.yml |

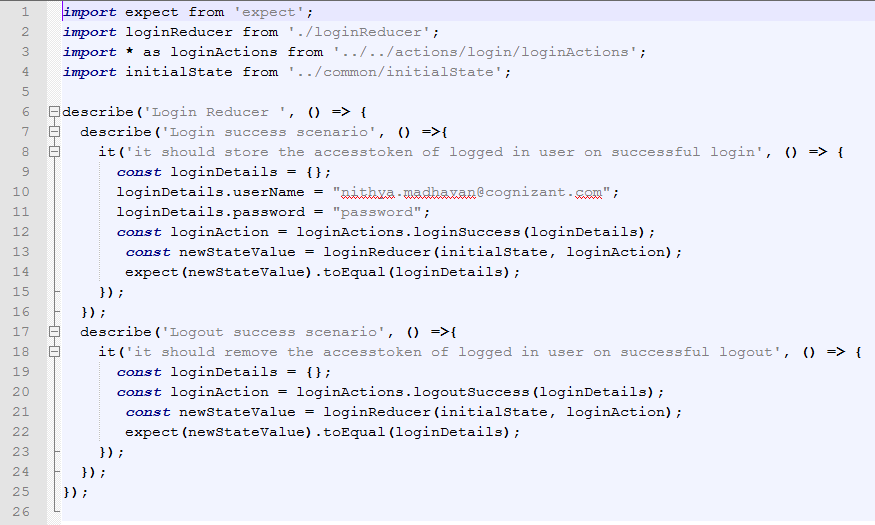
****

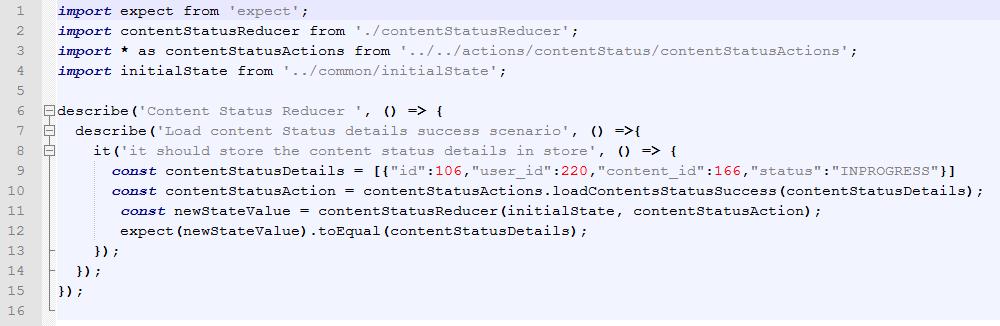
**Integration test class -** [OnBoardingApplicationIntegrationTest](file:///D:\OnBoarding\Onboarding-Portal--API\src\test\java\com\cts\cde\io\onboarding\OnBoardingApplicationIntegrationTest.java)**:**

****

**Created the test cases for UI**

Have created the test cases for each actions, action creators and reducers. Refer the package.json to run the test cases. Have created the test cases files a filename.test.js





1. **Run and build the application**
   1. **spring boot application:**

**Run:** mvn spring-boot:run

**Build:** mvn clean package

* 1. **spring boot application:**

**Run:** npm run start

**Build:** npm run build

**Test:** npm run test

1. **Creating pipeline for CI-CD:**

**To set the target:**

fly -t local login --concourse-url http://54.179.164.72:8080 --team-name pod-team-5

**To set the pipeline:**

fly -t local set-pipeline -p cf-resource-creation-dev --config pipeline - dev.yml -l private - dev.yml

1. **Application Dev and PROD Urls:**

**API URL**

**DEV API URL:** [**https://bodha-apijsoningnore.app.dev.digifabricpcf.com/**](https://bodha-apijsoningnore.app.dev.digifabricpcf.com/)

**PROD API URL :** [**https://bodhaapi.app.dev.digifabricpcf.com/**](https://bodhaapi.app.dev.digifabricpcf.com/)

**UI URL**

**DEV API URL:** [**http://bodha-dev.app.dev.digifabricpcf.com**](http://bodha-dev.app.dev.digifabricpcf.com)

**PROD API URL :** [**http://bodha.app.dev.digifabricpcf.com**](http://bodha.app.dev.digifabricpcf.com)