

# Documentation Pissir Project

Le projet pissir est un projet microservice subdiviser en deux services:

- Le backend, subdiviser en deux modules :
  - Le module **API**, module concernant la gestion de la serre en elle même
  - Le module **IOT**, module concernant la gestion du **MQTT** Broker
- Le frontend, partie visuelle (UI) de l'application

les technologies utiliser pour la realisation du projet sont les suivantes :

- Java, V17 (Langage de programation du projet)
- Springboot, V3.0.5 (Framework Java pour l'implementation de l'API REST)
  - Spring JPA (Pour l'implementation des regles de connexion à la base de donnée)
  - Spring Web (Pour l'implementation des controlleurs http et endpoints)
  - Spring Validation (Pour la validation des données)
  - Spring Integration MQTT (Pour l'implementation du broker)
  - Spring Quartz (Pour l'implementation du CRON MQTT)
- Logback, V1.4.6 (Pour la gestion des logs)
- SL4J, V2.0.5 (Pour la gestion des logs)
- Lombok, V1.18.26 (Pour l'implementation des getter, setter et constructeur par annotation)
- Jupiter, V5.9.2 (Pour l'implementation des tests)
- AssertJ, V3.24.2 (Pour l'implementation des tests)
- H2Database, V2.1.214 (Pour la connexion à la base de donnée test)
- MYSQL-Connector-J, V8.0.32 (Pour la connexion à la base de donnée dev)
- Mapstruct 1.5.3 (Pour le mapping des entités en models et vice-versa)
- Gradle (Outils de gestion du build)
- JavaScript (Pour l'implementation du front)
- HTML (Pour l'implementation du front)
- CSS (Pour l'implementation du front)

## Architecture du projet:

L'architecture du projet respect un certain nombre de norme, decrit comme etant necessaire pour l'obtention d'une application saine.

## Backend architecture:

Le backend respect les normes ou principes de l'architecture MVC (Modele, Vue, Contrroleur) .

## Structure MVC du module API:

La constitution du module API sur le plan architectural est la suivante ;

- Le package **Configuration**:  
Contient toutes les ressources servant à la configuration du module plus precisement à la configuration des beans utilisés par le framework Springboot.  
Ici, nous trouverons une configuration pour le CORS afin de permettre à l'API de recevoir des appels http venant de l'exterieur (Autres IP)
- Le package **Controller**:  
Contient les ressources servant à l'implementation des endpoints (URL http).  
Nous avons opté pour une classification par entité, ainsi, nous retrouvons dans ce package plus ou moins de classe que d'entité présent dans l'API
- Le package **Entity**:  
Contient les entités de l'API. ces entités representent le models des informations sauvegardés dans la base de donnée. chaque données de l'entité est validé avant sauvegarde.
- Le package **Error**:  
Contient des resources representant les erreurs personalisées pouvant être retourné par le module.  
Nous avons opté une solution d'ensemble, une seul erreur pour le module customisable par status et message.

- Le package **Exception**:  
Contient des ressources représentant les exceptions personnalisé pouvant être levées par le module et un intercepteur d'exception qui se charge de convertir l'exception en erreur à retourner au client.
- Le package **Mapper**:  
Contient des ressources servant à mapper ou convertir les entités (information de la base de donnée) en models (Information retournée au client)
- Le package **Model**:  
Contient des ressources servant de models de données présenté au client comme réponse aux requêtes http
- Le package **Repository**:  
Contient des ressources servant d'implémentation des normes JPA pour une connexion normalisée à la base de données
- Le package **Service**:  
Contient des ressources servant d'implémentation de la logique fonctionnelle de l'API.  
Nous retrouverons dans ce package une partie importante de notre module.
- Le package **Utility**:  
Contient des ressources vue comme redondante dans notre module, regrouper en méthode static
- Le package **Validator**:  
Contient des ressources servant à l'implémentation de méthode de validation de données personnalisée

### **Structure MVC du module IOT:**

La constitution du module IOT sur le plan architectural est la suivante :

- Le package **Configuration**:  
Contient toutes les ressources servant à la configuration du module plus précisément à la configuration des beans utilisés par le framework Springboot.  
Ici, nous trouverons
  - une configuration pour le CORS afin de permettre à l'API de recevoir des appels http venant de l'extérieur (Autres IP)
  - Une configuration du MQTT broker
  - Une configuration du bean RestTemplate servant d'outils pour les appels d'API externe
- Le package **Controller**:  
Contient les ressources servant à l'implementation des endpoints (URL http).
- Le package **Model**:  
Contient des ressources servant de models de données présenté au client comme réponse aux requêtes http
- Le package **Service**:  
Contient des ressources servant d'implémentation de la logique fonctionnelle de l'API.  
Nous retrouverons dans ce package une partie importante de notre module.
- Le package **Publisher**:  
Contient des ressources servant à publier les informations des capteurs. ces publications sont faites par le biais d'un MQTT broker.

- Le package **Subscriber**:

Contient des resources servant à souscrire à un broker afin de recevoir les messages publier.

- Le package **Task**:

Contient des resources servant à runs des fonctions de publisher et suscriber à des intervalles de temps déterminés.

- Le package **Shared**:

Contient des resources servant à filtrer les types de sensore disponible pour le client appelant.

## **Frontend architecture:**

Le frontend ne possède aucune architecture, nous avons opté pour une simple UI en Html Css et JavaScript. les appels vers le backends sont effectuer avec la library JavaScript fetch.

Le dossier **Css** contient tous les fichiers de style de la vue, le dossier **Font** contient les polices utiliser sur la vue, le dossier **Image** contient tous les images utilisées sur l'interface, le dossier **Js** contient tous les scripts permetants le dynamisme de l'interface.

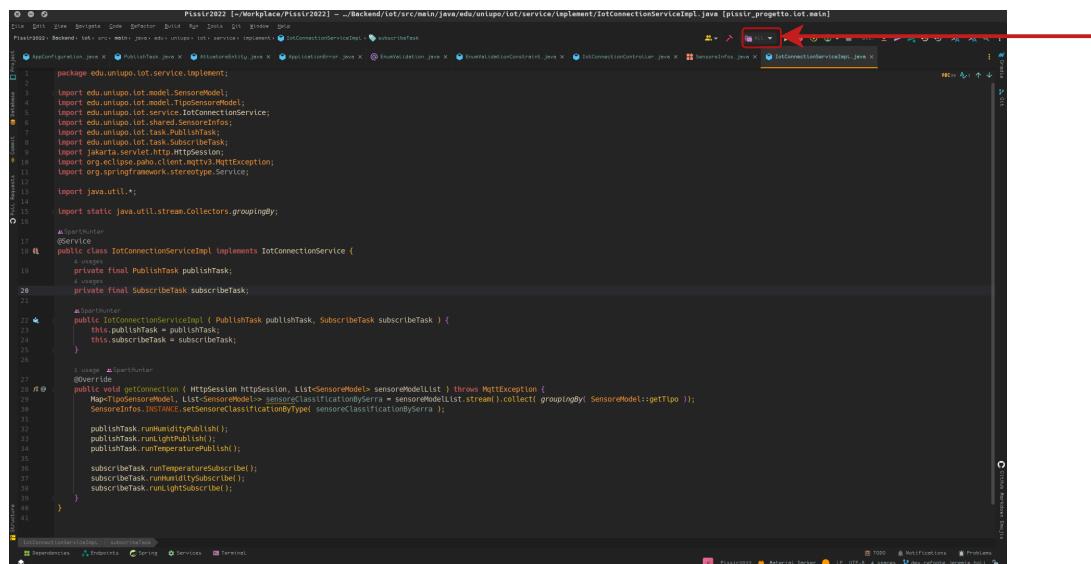
Le fichier index.html est le fichier principale de notre UI.

## **DEMARRAGE DU PROJET:**

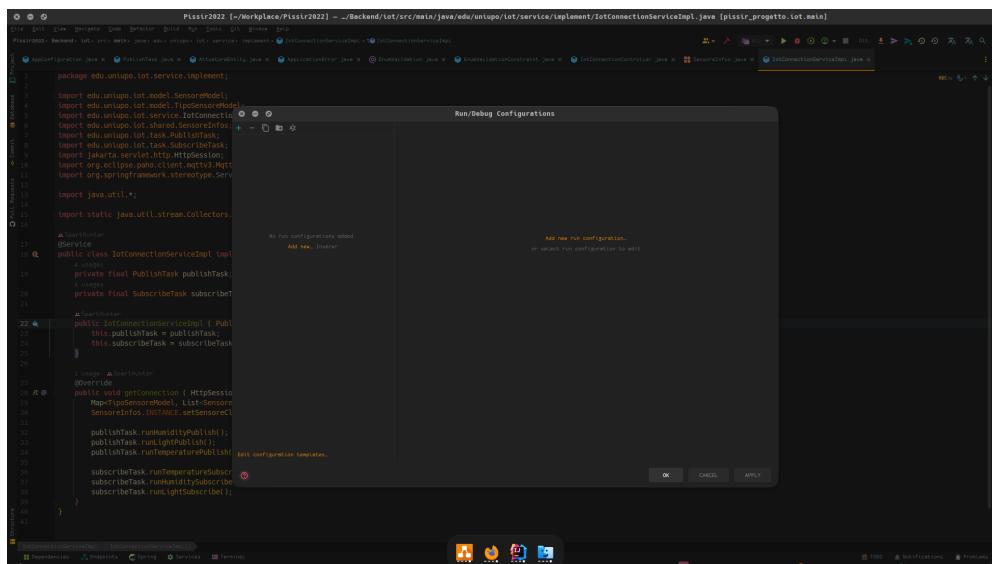
Le projet étant réalisé sur IntelliJ, nous allons dans cette documentation vous guider sur son démarrage avec cet IDE.

Pour run le projet :

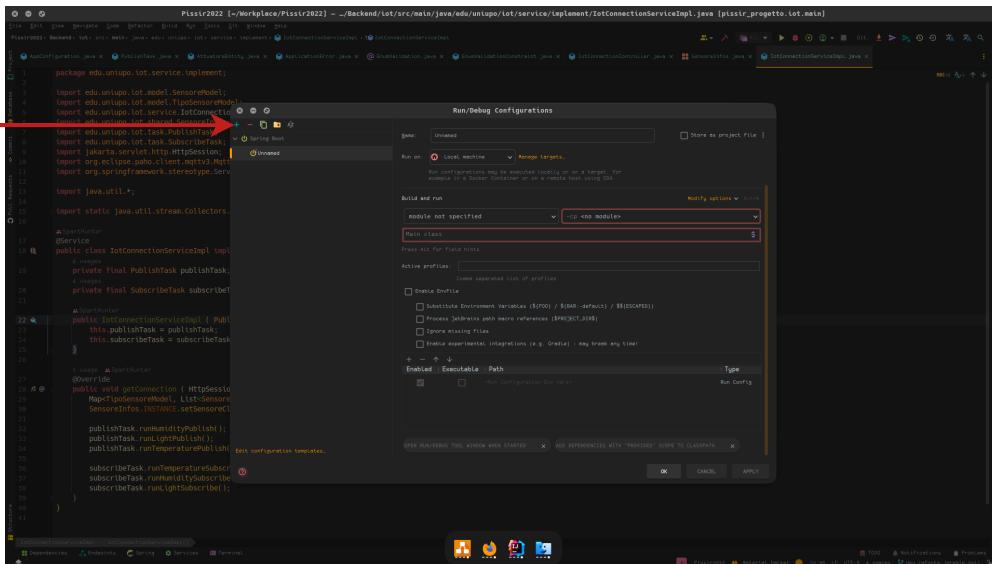
1/ ouvrir le projet avec l'IDE IntelliJ, aller dans les configuration de run et debug (Cadrant rouge sur l'image ci-joint)



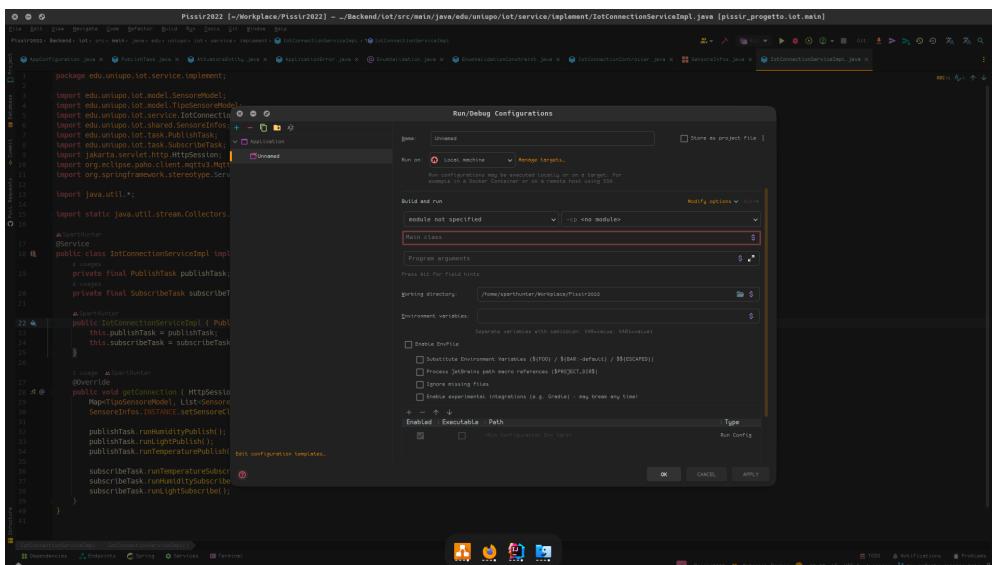
2/ Cliquer sur edit configuration, une nouvelle fenêtre s'ouvre.



3/ Cliquer sur le bouton et choisirer l'option springBoot si elle est presente sinon l'option application.

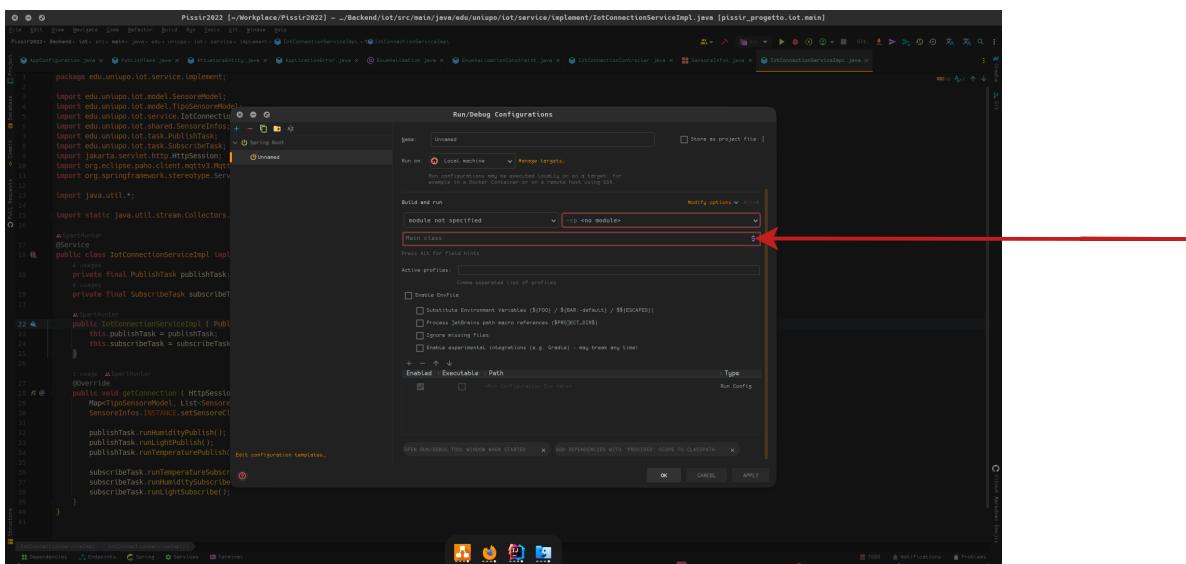


Option Springboot Vue



Option Application Vue

4/ Dans l'input avec le placeholder **Main Class**, cliquer sur le button dollar violet, et choisisser la class **PissirApplication**.



The screenshot shows the IntelliJ IDEA interface with the code editor open to a Java file named `IotConnectionServiceImpl.java`. A modal window titled "Run/Debug Configurations" is displayed. In the "Name" field, the text "IoTApplication" is typed. Below the name field, there is a dropdown menu set to "local machine". At the bottom of the modal, there are "OK", "CLOSE", and "APPLY" buttons.

5/ Dans l Name, tout en haut de la fenêtre, écrire le nom de la class sélectionnée précédemment.

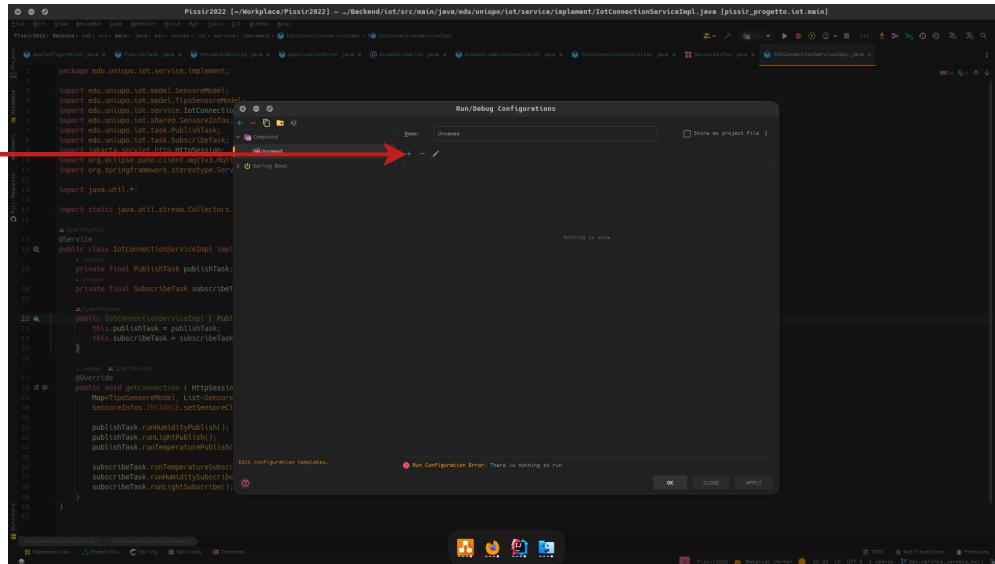
The screenshot shows the IntelliJ IDEA interface with the code editor open to the same `IotConnectionServiceImpl.java` file. The "Run/Debug Configurations" dialog is shown again, but this time the "Name" field contains "PisirApplication". The "local machine" option is still selected in the dropdown. The "OK", "CLOSE", and "APPLY" buttons are at the bottom.

6/ Dans l Active Profiles, écrire le mot dev.

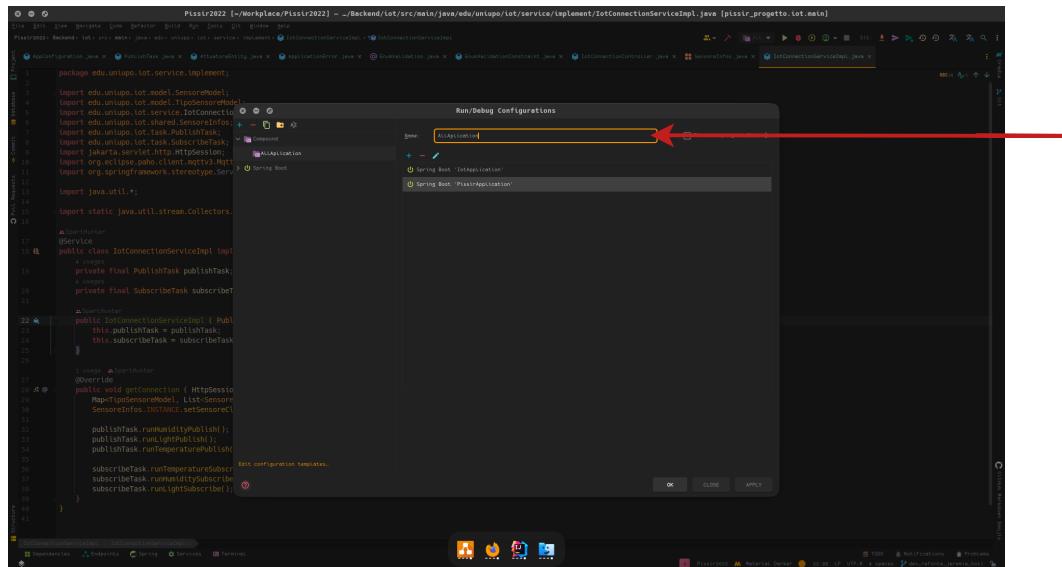
The screenshot shows the IntelliJ IDEA interface with the code editor open to the same `IotConnectionServiceImpl.java` file. The "Run/Debug Configurations" dialog is shown once more, with the "Name" field containing "PisirApplication". In the "Active profiles" field, the text "dev" is typed. The "local machine" option is selected. The "OK", "CLOSE", and "APPLY" buttons are at the bottom.

7/ Refaire l'action de l'étape numéro 3, et 4 Mais sélectionner la class `IotApplication` à l'étape 4, refaire aussi l'étape 5 Mais écrire le nom `IoTApplication`.

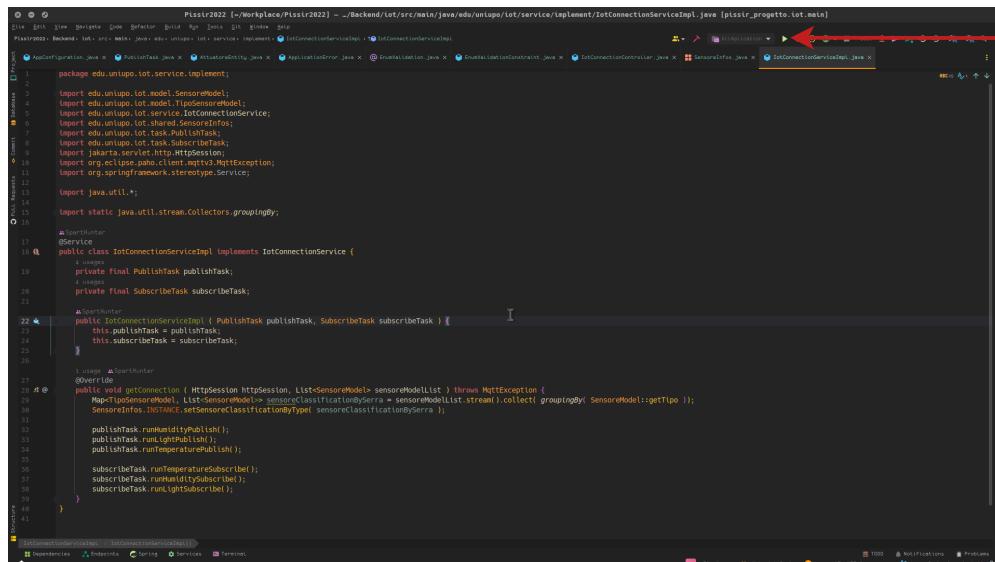
8/ Refaire l'etape numero 3, et selectionner l'option **Compound**, puis cliquer sur le **+** et selectionner les deux Class afficher.



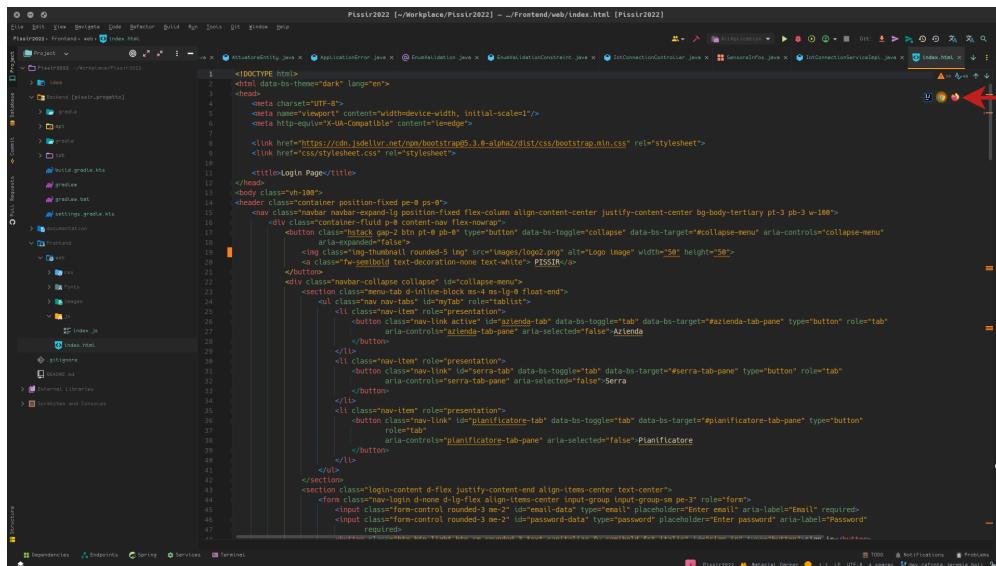
9/ Dans l'input **Name**, ecrire : **AllApplication.**



10/ Appuyer sur les buttons **Apply** et **Ok** pour fermer la fenetre. Puis sur le bouton vert pour demarrer le projet backend en entier.



11/ Ouvrir le fichier index.html et cliquer sur le bouton de votre navigateur preferer.



```
<!DOCTYPE html>
<html data-bs-theme="dark" lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta http-equiv="X-UA-Compatible" content="edge">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.8-alpha2/dist/css/bootstrap.min.css" rel="stylesheet">
<link href="css/styleSheet.css" rel="stylesheet">
<title>Login Page</title>
</head>
<body class="vh-100">
<header class="d-flex align-items-center justify-content-between position-fixed pe-0 ps-0">
<div class="nav-bar navbar-expand-lg position-fixed flex-column align-content-center justify-content-center bg-body-tertiary pt-3 pb-3 w-100">
<div class="container-fluid p-0 content-nav flex-wrap">
<button class="hstack-grow-1 px-0 py-0" type="button" data-bs-toggle="collapse" data-bs-target="#collapse-menu" aria-controls="collapse-menu">

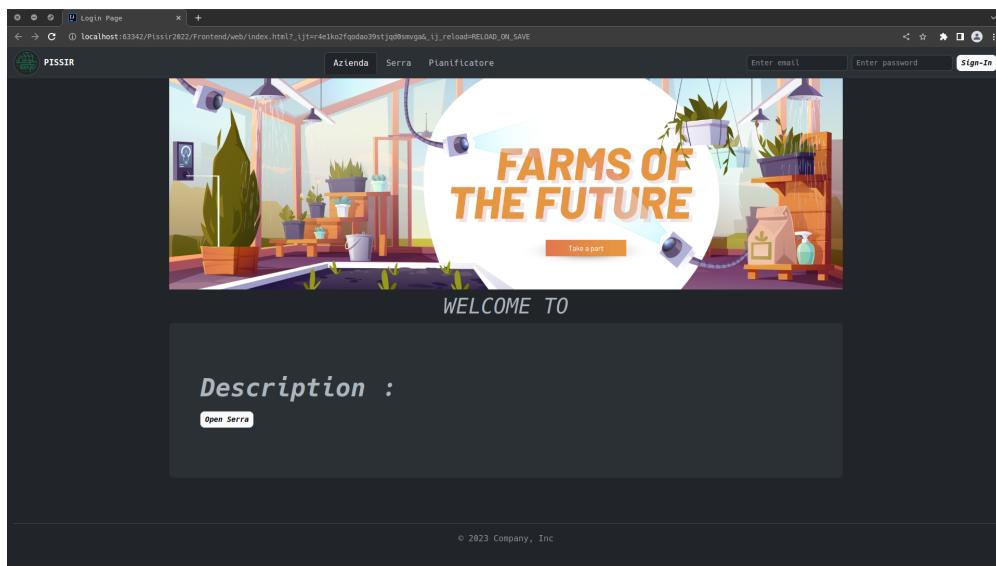
<a class="fw-semi bold text-decoration-none text-white" href="#">PISSIR</a>
</div>
<div class="nav-bar-collapse collapse" id="collapse-menu">
<div class="m-0 m-lg-0 float-end">
<ul class="list-group list-group-flush" role="tablist">
<li class="nav-item" role="presentation">
<button class="nav-link active" id="azienda-tab" data-bs-toggle="tab" data-bs-target="#azienda-tab-pane" type="button" role="tab" aria-controls="azienda-tab-pane" aria-selected="true">Azienda</button>
</li>
<li class="nav-item" role="presentation">
<button class="nav-link" id="serra-tab" data-bs-toggle="tab" data-bs-target="#serra-tab-pane" type="button" role="tab" aria-controls="serra-tab-pane" aria-selected="false">Serra</button>
</li>
<li class="nav-item" role="presentation">
<button class="nav-link" id="pianificatore-tab" data-bs-toggle="tab" data-bs-target="#pianificatore-tab-pane" type="button" role="tab" aria-controls="pianificatore-tab-pane" aria-selected="false">Pianificatore</button>
</li>






```

12/ Le navigateur dois alors s'ouvrir sur cette interface



13/ Entrer les identifiants ci-joint pour ce connecter et utiliser l'application.

- Agriculteur identifiant :
  - Email : prince@mail.com
  - Password : 5555
- Collaborateur identifiant :
  - Email : doe@mail.com
  - Password : 5555

