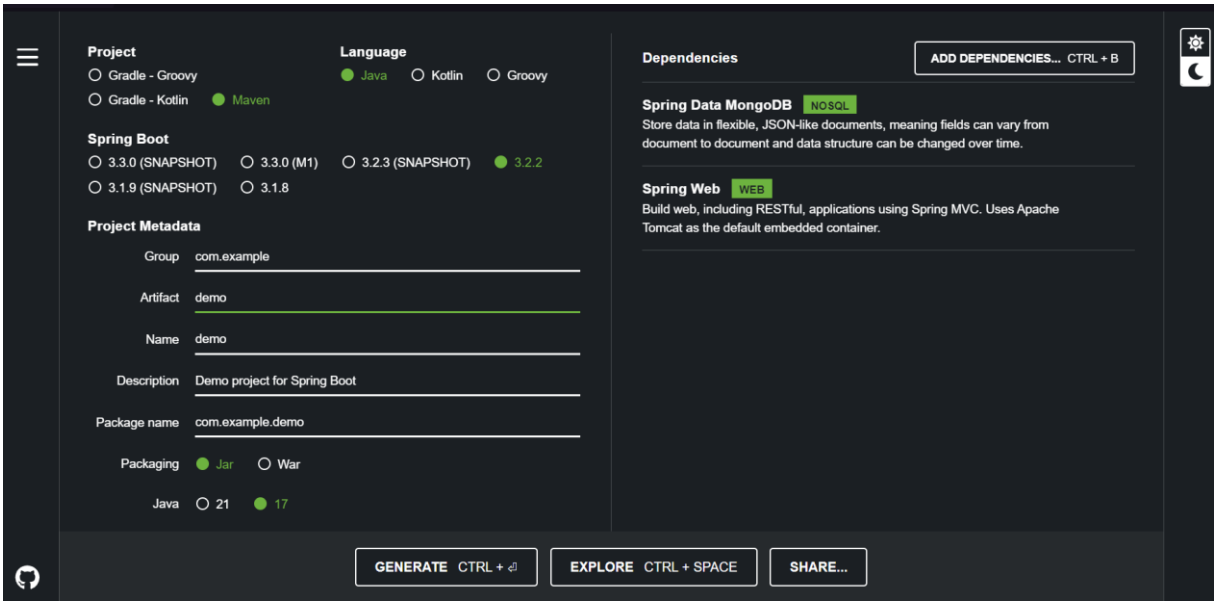
TD Docker / Spring Boot / MongoDB

Docker

Installer docker : https://docs.docker.com/desktop/install/windows-install/

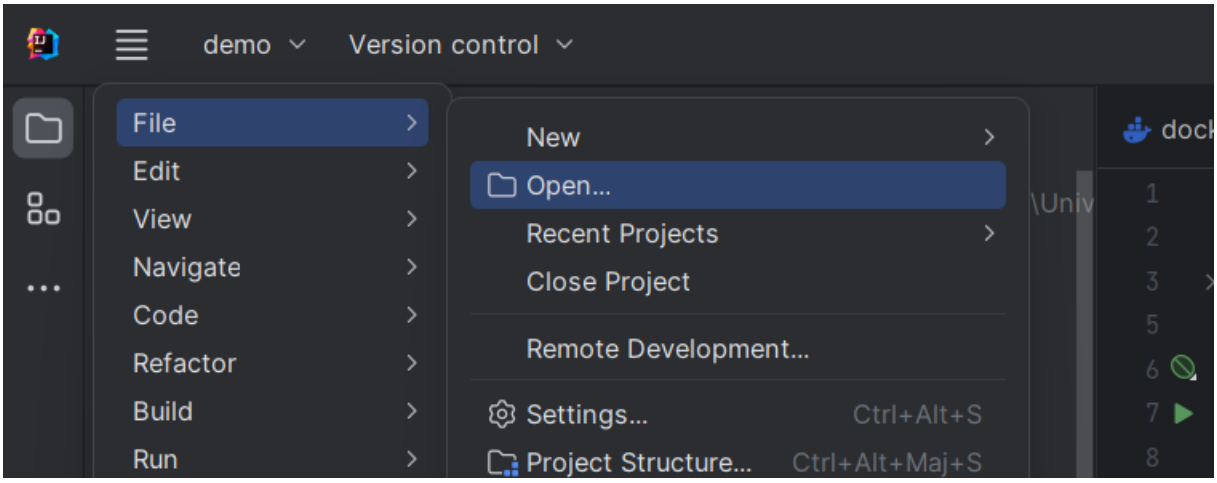
Générer un projet Java Spring Boot

Générer projet Java avec Spring Web et Spring Data MongoDB sur Spring Initializr, avec les dépendances *Spring Data MongoDB* et *Spring Web.*



IntelliJ IDEA

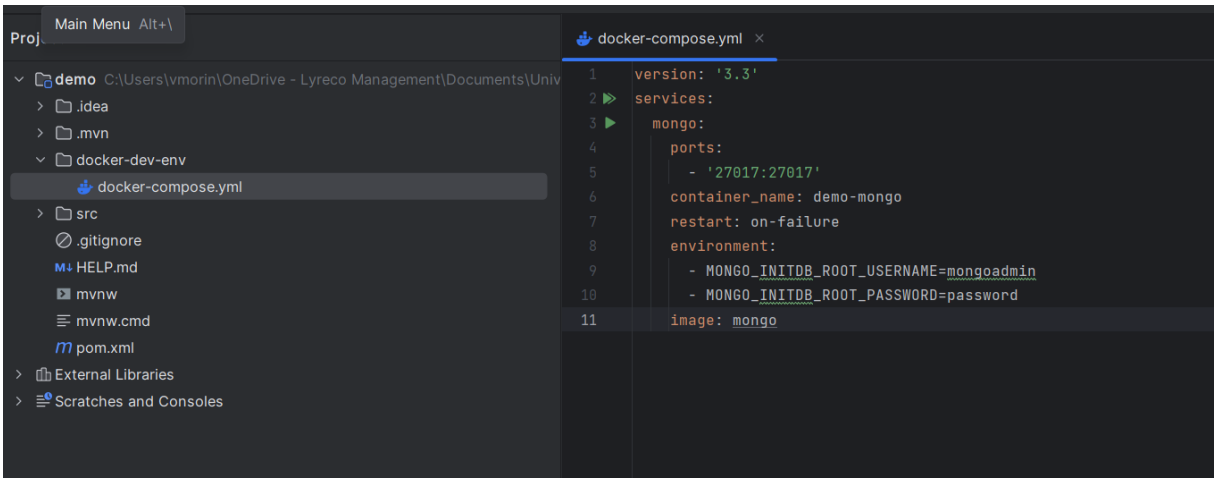
Installer IntelliJ https://www.jetbrains.com/idea/ puis importer le projet généré précédemment :



Si besoin installez Maven pour pouvoir installer et build le projet.

Installer Mongo sur Docker

Dans le projet créer un dossier *docker-dev-env*, et y créer un fichier *docker-compose.yml* :

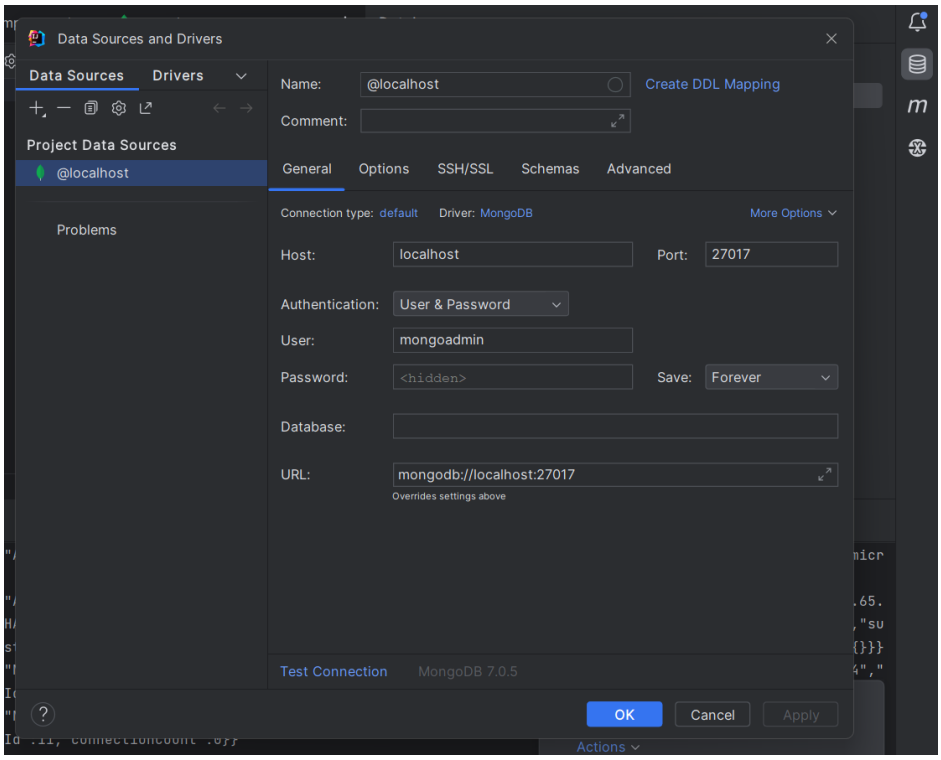



```
version:                                                    '3.3'
services:
  mongo:
    ports:
      -                                                 '27017:27017'
    container_name:                                        demo-mongo
    restart:                                               on-failure
    environment:
      -                                MONGO_INITDB_ROOT_USERNAME=mongoadmin
      -                                  MONGO_INITDB_ROOT_PASSWORD=password
    image: mongo
```

Accéder au dossier *docker-dev-env* via le terminal et lancer la commande :

```
docker-compose up
```

Checker la connexion à la mongo via IntelliJ :



Si tout fonctionne, plus qu'à mettre en place l'application Java pour se connecter à la MongoDB !

L'objectif sera de créer une application avec Spring Boot dont le nom sera « product-api » qui permettra de :

- Créer un produit dont les champs seront :
  - reference (code produit représenté par un int)
  - nom (« papier A4 » par exemple représenté par une String dont la taille max sera 25)
  - description (« exceptionnel papier de qualité supérieure ! » représenté par une String dont la taille minimum sera 25)
  - prix (5€ représenté par un int qui ne peut pas être 0)
- Récupérer un produit en fonction de son id
- Récupérer les produits dont le prix est < à une certaine somme
- Supprimer un produit en fonction de son id

Cette application sera testable grâce à Postman ou Swagger si vous l'installez.

Une fois cette application créée, il faudra la « Dockeriser », c'est-à-dire la build avec Docker et la déployer grâce au docker-compose.

Exemple de Dockerfile (celui expliqué en cours) :

```
FROM maven:3.9.6-amazoncorretto-21 as build
RUN mkdir -p /demo
WORKDIR /demo
COPY pom.xml /demo
COPY src /demo/src
RUN mvn -f pom.xml clean package

FROM amazoncorretto:21.0.2-alpine3.19
COPY --from=build /demo/target/*.jar app.jar
EXPOSE 8080/tcp
ENTRYPOINT ["java", "-Dspring.profiles.active=docker", "-jar","/app.jar"]
```

Grâce à une configuration adaptée au *spring profile* Docker, on réussira à se connecter au sein de l'environnement Docker à la mongo :

```
spring:
  data:
    mongodb:
      host: mongo-demo
      port: 27017
      username: mongoadmin
      password: password
      authentication-database: admin
      database: demodb
```

Il faudra donc adapter le docker-compose.yml en suivant le schéma suivant :

```yaml
services:
  mongo-demo:
    image: mongo
    container_name: mongo-demo
    environment:
      - MONGO_INITDB_ROOT_USERNAME=mongoadmin
      - MONGO_INITDB_ROOT_PASSWORD=password
    volumes:
      - mongo-demo:/data/db
    expose:
      - 27017
    ports:
      - '27017:27017'
    restart: on-failure
    healthcheck:
      test:
        [
          "CMD",
          "mongosh",
          "--quiet",
          "127.0.0.1/admin",
          "--eval",
          "'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)'",
        ]
      interval: 5s
      timeout: 5s
      retries: 5
      start_period: 10s

  api-demo:
    build: ../
    container_name:  api-demo
    ports:
      - "8080:8080"
    links:
      - mongo-demo
    depends_on:
      mongo-demo:
        condition: service_healthy

volumes:
  mongo-demo:
```