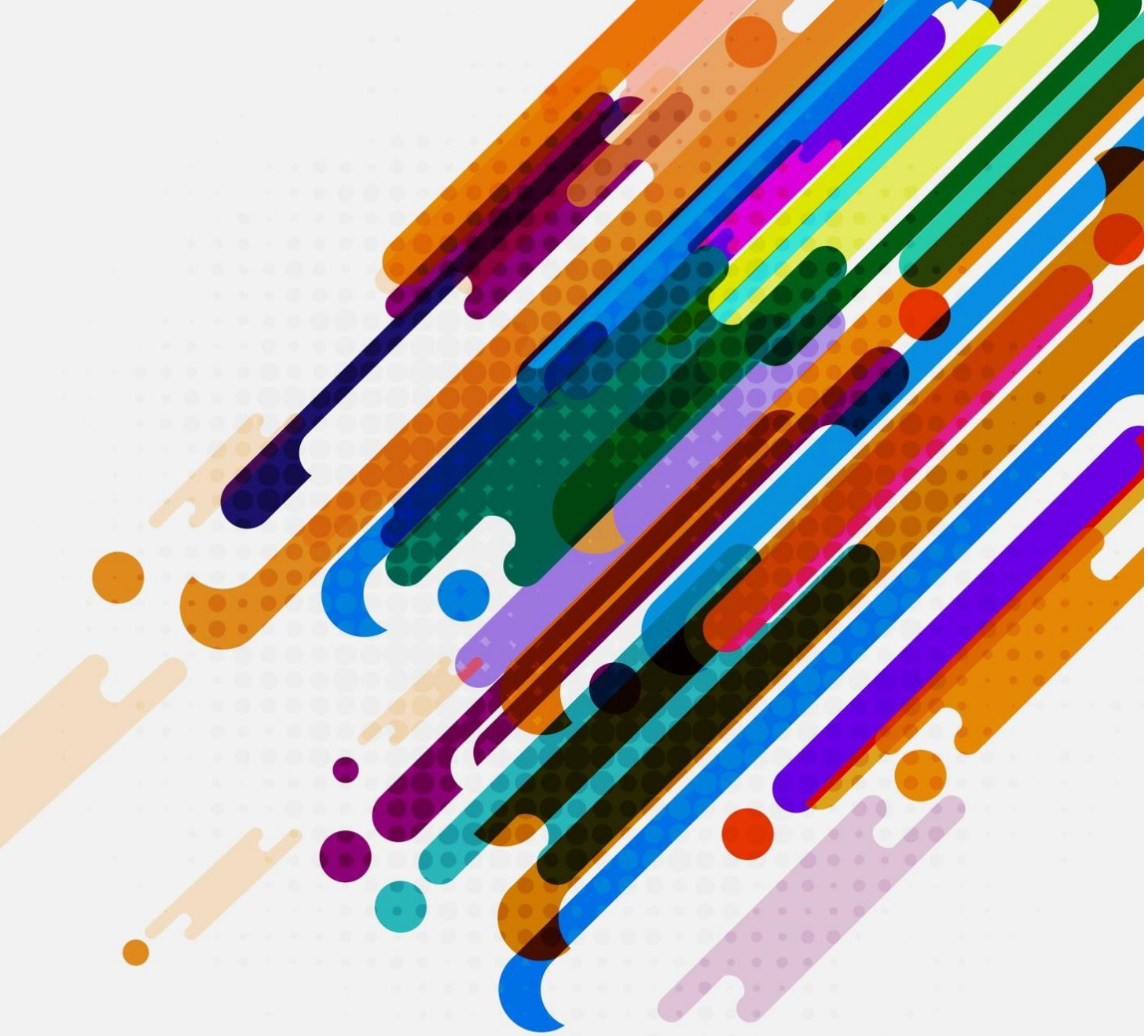


Programowanie aplikacji internetowych MVC

PROJEKT – ZALICZENIE ĆWICZEŃ

shoppingApp

- ShoppingApp to projekt sklepu internetowego, który oferuje symulacje zakupu samochodu. Aplikacja umożliwia użytkownikom przeglądanie dostępnych modeli pojazdów, dodawanie ich do koszyka oraz symulowanie finalizacji zakupu. W obecnej wersji brak jest jeszcze integracji z płatnościami. Aplikacja bangla także na telefonie. Wystarczy odpalić aplikację i wszystkie urządzenia w sieci będą mogły użyć aplikacji.
- Wrzucam tutaj starą wersję projektu, która nie przeszła jeszcze refaktoryzacji, ale działa i spełnia podstawowe funkcje. Jednocześnie rozwijam nową wersję, która jest zbudowana zgodnie z nowymi standardami oraz z podziałem na odpowiednie logiki, i zmianą z ThymeLeafa na Reacta w celu odseparowania części aplikacji. Nowa wersja jest jeszcze w fazie rozwoju, więc nie wrzucam jej, ponieważ nie jest w pełni funkcjonalna.

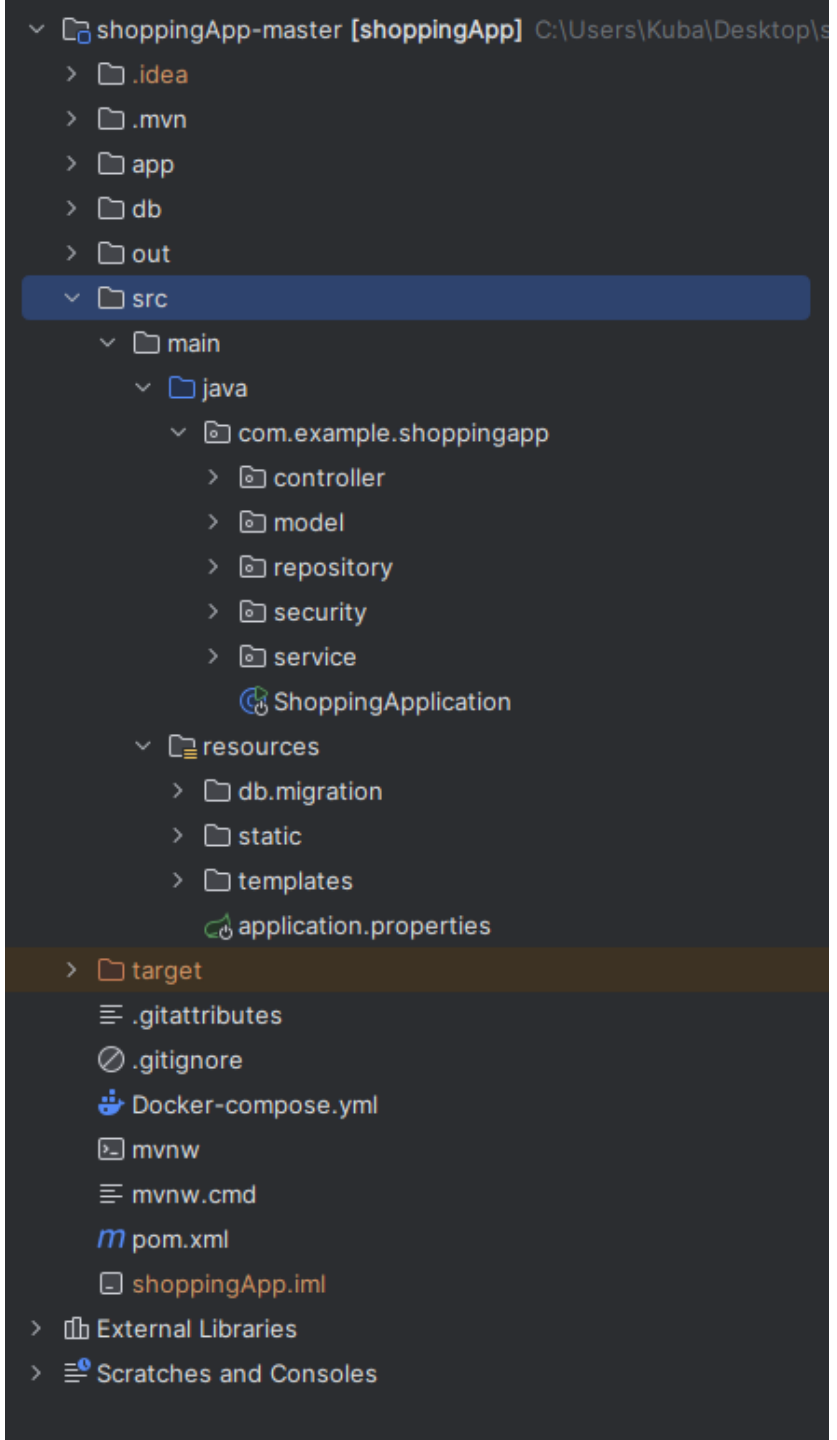
Opis

Aplikacja napisana jest w 4 językach:

- Java(60%) - wykorzystałem do tego framework Spring'a, wraz z jego komponentami (security, boot, MVC).
- Html(18%) - czysty html z thymeLeafem w celu wyświetlenia widoku użytkownikowi.
- Css(20%) - ...
- -mySQL z JS (2%) - prosta baza danych z paroma kluczami obcymi i relacjami, a do tego skrypt JS przekazujący adres strony.

Struktura projektu

- Struktura odpowiada typowemu projektowi Spring. Wyróżniamy:
- Controller
- Model
- Repository
- Security
- Service

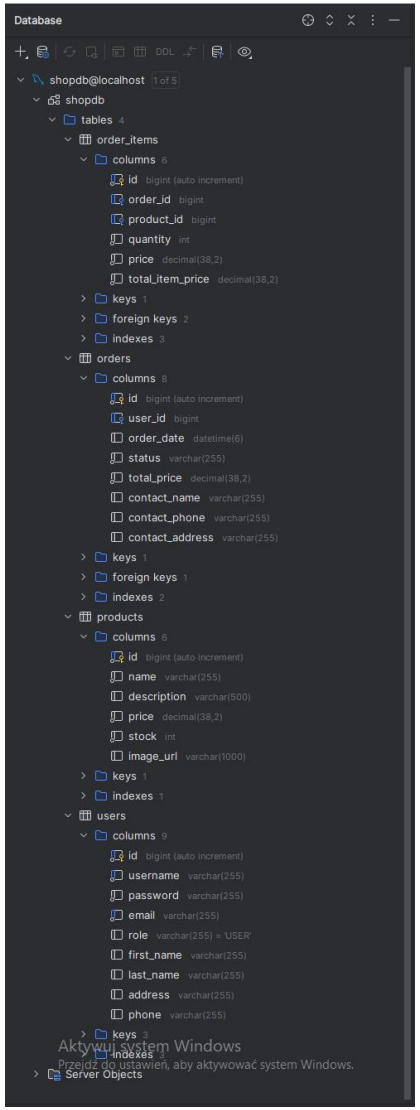


Controller

- Kontrolery do obsługi żądań HTTP (POST,GET,PUT,UPDATE). Niestety jak już wspominałem nie zdążyłem zrobić refaktoryzacji, przez co logika kontrolerów przejęła logikę serwisów. Nie powoduje to błędów, ale oczy bolą jak się na to patrzy.

Model

- Postanowiłem wykorzystać encje to stworzenia bazy danych, całkiem dobre narzędzie, ale nie oferuje checków na bazie(np. Cena nieujemna). Kontroluje tylko przepływ danych z apki do bazy, ale nie oferuje zabezpieczenia przed włożeniem nieprawidłowych danych bezpośrednio do bazy. Wykorzystałem także flyway w celu migracji, ponieważ baza się rozrastała stopniowo.



Repository

- Standardowe repozytoria rozszerzone o JPA. Dzięki temu nie musimy się komunikować z bazą ręcznie i wpisywać komend SQL. Bardzo popularna i prosta mechanika oferująca potężne narzędzia do komunikacji z bazą danych.

Security

- Tutaj wypadałoby się pochylić nad klasami:
- CustomUserDetails
- CustomUserDetailsService
- SecurityConfig

SecurityConfig

- Dzięki oznaczeniu @Component, możemy wykorzystać klasę, aby dodać @Beany, które będą odpowiedzialne za zarządzaniem bezpieczeństwem aplikacji i spring je z automatu zaaplikuje podczas startu. W skrócie securityFilterChain zapewnia dostęp do stron (zalogowanych/niezalogowanych) użytkowników, zapewnia weryfikacje przez formularz logowania, obsługuje wylogowanie. Dodatkowo skorzystałem z Argona w celu zahashowania hasła do bazy danych. Hasło będzie się składalo z 96-100 znaków co warto uwzględnić w wielkości tabeli w bazie danych. Hasło te jest niemożliwe do odhashowania dla zwyklego usera. Algorytm podczas rejestracji hashuje haslo, a podczas logowania porównuje surowe haslo z parametrami hashowania i wypluwa true or false.

CustomUserDetails

- Klasa implementujaca interfejs UserDetails. Dzięki temu mamy dostep do metod Springa, a najwazniejsze z nich to:
- getAuthorities – zwraca role (np..USER_ROLE, albo USER_ADMIN itp) - w moim przypadku wywala null. W nowej wersji dodałem kolumne w bazie i metoda szuka czy w bazie jest user/admin i nadaje odpowiednie uprawnienia zalogowanemy userowi.

- Możliwość zablokowania konta, oraz wygaśnięcia.

CustomUserDetailsService

- To bardzo ważna klasa w kontekscie uwierzytelniania Springa. Zwraca caly obiekt zalogowanego użytkownika. Ale nie tylko jego dane(rekordy w bazie danych), ale także informacje o jego rolach itp. Będzie potem to przydatne do sprawdzania czy uzytkownik jest zalogowany (porownując czy jest obiektem instancji UserDetails, albo przy koszyku czy innych funkcjach.

Resources

- Db.migration - Jest to folder w którym po wykonaniu migracji Flyway znajduje się kod SQL dodający do istniejącej struktury bazy danych potrzebny element - może to być klucz, tabela, kolumna itp. Spring przy uruchomieniu aplikacji sprawdza ten folder i stosuje zmiany.
- Static - Tutaj znajduja się frontendowe elementy tj. CSS, zdjęcia, czy skrypty JS.
- Templates - Zawiera nasze widoki stron, pliki z rozszerzeniami głównie .html, zdarzają się także .jsp

Application.properties

- Jest to główny plik konfiguracyjny w Springu, jeśli używamy Mavena, a nie Gradle. W skrócie konfigurujemy tutaj dostep do bazy danych, elementy pokazujace się w konsoli aplikacji, decydujemy o użycie flyway, ogarniamy czas trwania sesji czy porty na których chcemy, żeby nasza apka i baza śmigały. Ze względu na to, że port 8080 bywa zajęty, apka stoi na porcie 9091. Baza na 3308, ze względu na to, że na linuxie na 3306 często hula server SQL.

```
server.servlet.session.timeout=1800
server.port=9091
server.error.whitelabel.enabled=false
server.error.path=/error

spring.datasource.url=jdbc:mysql://localhost:3308/shopdb
spring.datasource.username=kuba
spring.datasource.password=kubaru1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver



spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true




spring.flyway.enabled=false
spring.flyway.url=jdbc:mysql://localhost:3308/shopdb
spring.flyway.user=kuba
spring.flyway.password=kubaru1234
spring.flyway.schemas=shopdb
spring.flyway.validate-on-migrate=false
```

Target

- Folder w którym po użyciu maven (clean, compile) utworzy plik, w moim przypadku JAR (WAR też jest możliwe). Dzięki temu mamy "obraz" naszej aplikacji (nie bazy danych!). Co możemy z nim zrobić? Serwery mogą działać na tych plikach, a w moim przypadku użyłem go do zrobienia kontenera Docker.

- Aplikacja będzie działać na każdym systemie operacyjnym, ze względu na użycie Docker'a w celu zapobiegnięcia problemu "u mnie nie działa". Polecam gorąco pobranie Docker Desktop na Windows, lub pobranie przez terminal pakietów Docker'a na Linuxa.
- Jeśli jednak wystąpiły jakieś problemy z Docker'em można uruchomić aplikacje w inny sposób, ale będzie to wymagało instalacji mySQL i utworzenia bazy z dump'a.

Implementacja projektu

1. Pobrać mój projekt z Git'a (https://github.com/jbialek99/shoppingApp.git), polecam pobrać projekt za pomocą zip, a potem wypakować go w dogodnym miejscu.

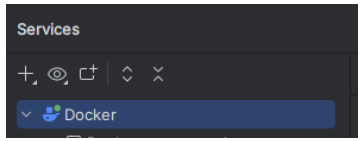
2. Otworzyć Intellij (może to być inna apka do programowania, ale kroki będą dotyczyły Intellij) i jakikolwiek projekt. W lewym górnym rogu wybieramy File -> New -> Project from existing sources -> wybieramy wczesniej wypakowany projekt -> Zaznaczamy import form external model, zaznaczamy Maven. Po tych krokach projekt powinien się zaimportować bez większych przeszkód. Mogą wystąpić ostrzeżenia w pliku pom(plik do którego w Maven dołączamy dependency) Jeśli będą to ostrzeżenia dotyczące przestarzalych bibliotek, proszę to zignorowac ;)

Opcja I

1. Pobrać i zainstalować Docker.

- A) Windows (preferowany wybór)– Docker Desktop, można go pobrać tutaj (https://www.docker.com/pricing/). Wybieramy plan za 0zł, logujemy się i pobieramy tylko Docker Desktop.
- B) Linux: Tutaj musimy pobrać pakiet przez terminal. Brak wersji graficznej nie boli, ale mogą wystąpić problemy z prawami lub dostępami do plików. Troszkę roboty trzeba czasami ogarnąć, więc polecam wersje na Winde.

Teraz przejść do mojego projektu w Intellij i nacisnąć alt+8. Wyświetli się okno na dole z sewisami. Powinien tam się znaleźć Docker, który jeśli będzie poprawnie zainstalowany, to będzie przy nim zielona kropka.

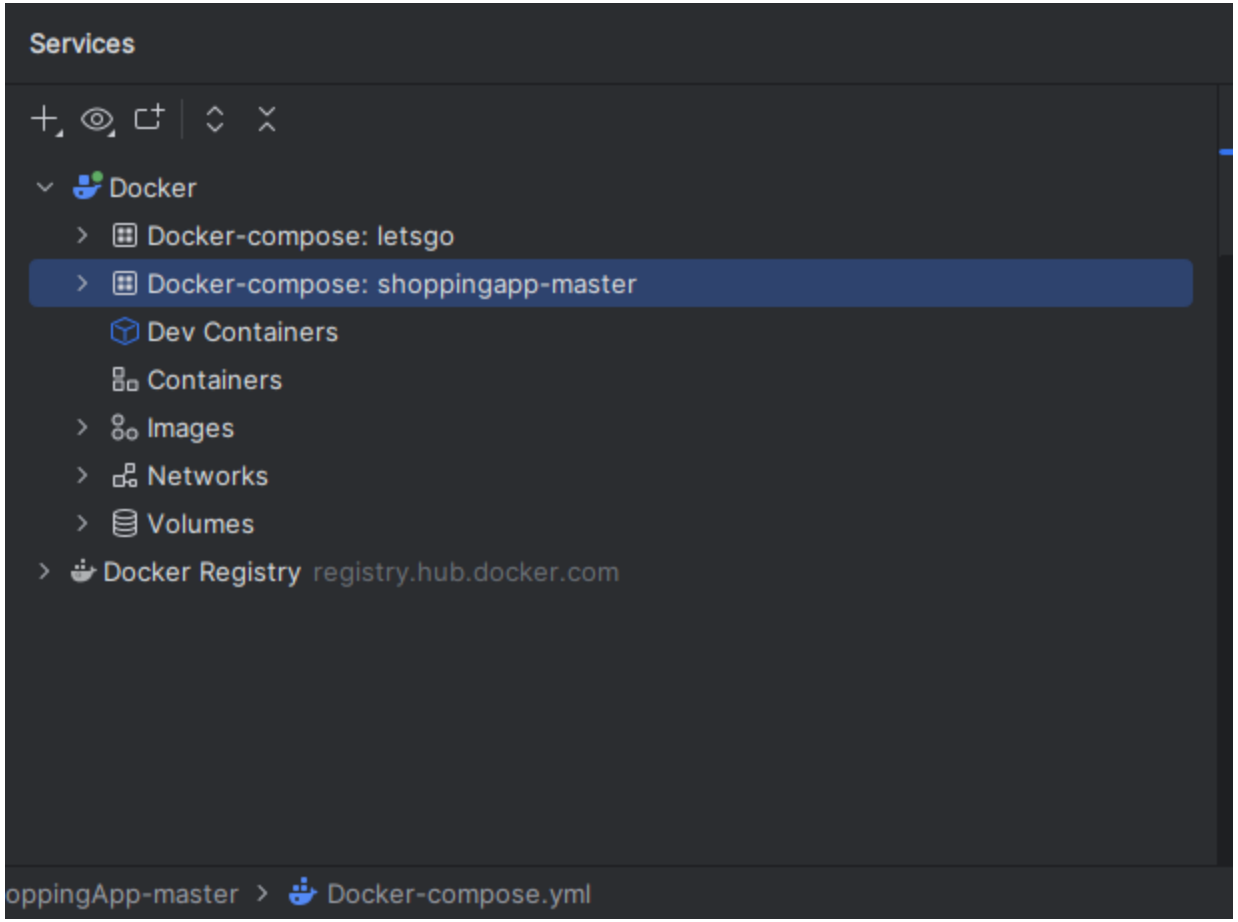


Teraz czas na kontenery :)

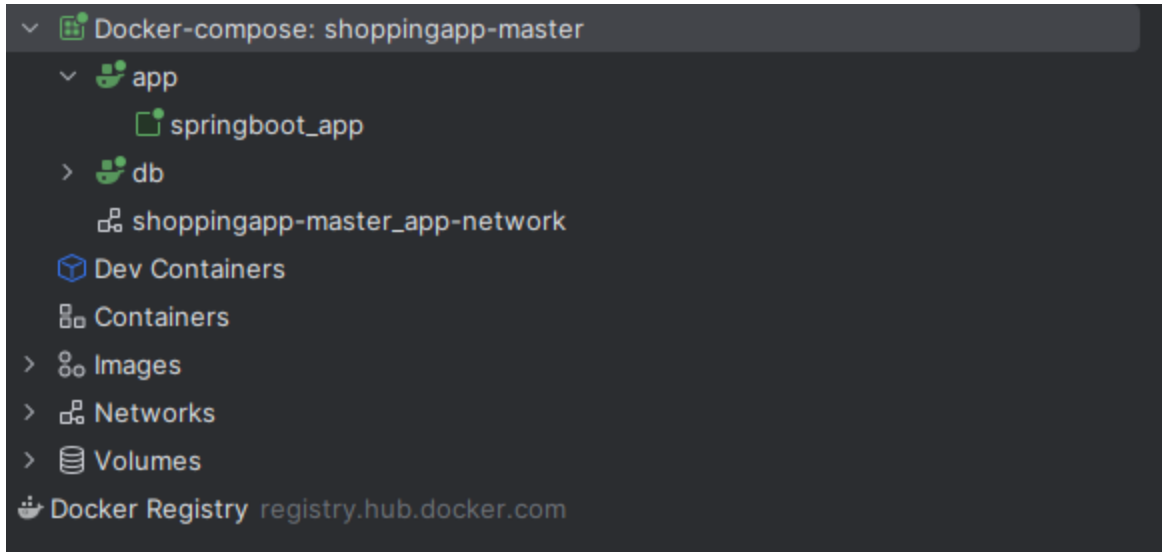
- W projekcie są 2 foldery (app i db) :
- Tutaj nic nie trzeba robić, wszystkim się już zająłem :) DockerFile to pliki konfiguracyjne dockera, które stworzyłem i dodatkowo shopapp.jar (obraz apki) i dump.sql(dump bazy danych)
- I plik docker-compose.yml, który będziemy musieli uruchomić. Plik ten zawiera opis w jaki mają powstać kontenery. W moim przypadku odseparowałem kontener db od app. WAŻNE odpalając aplikacje przez kontener application.properties nie będzie uwzględnione! Tworzymy nową konfiguracje aplikacji, która będzie w kontenerze.

```
3  services:
```

Proszę wejść w plik docker-compose.yml i kliknąc 2 strzałki przy sevices. Spowoduje to utworzenie kontenerów z bazą danych i z aplikacją. Po zakończonym etapie powinien nam się ukazać taki widok po naciśnięciu alt+8 (bez letsgo)



- Jeśli przy aplikacji nie pojawiły się zielone lampki proszę nacisnac prawym na Docker-compose: shoppingapp-master -> Start. Jeśli nadal nie zapaliły się zielone lamki (głównie problem może występować przy springboot_app) proszę kliknac 2x na element bez zielonej lampki i zobaczyć logi. Główne problemy jakie można spotkać to zajęte już porty albo błąd komunikacji z bazą danych.



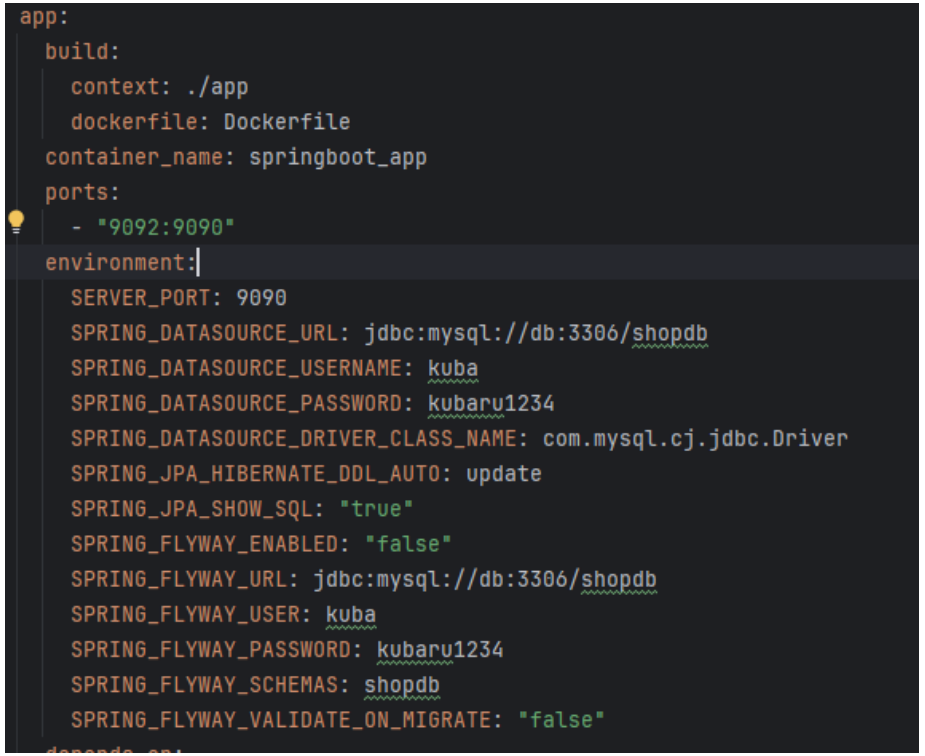
Zmiana portów w pliku docker-compose.yml

Jeśli z logów wynika, że problem występuje z "port already in used" z bazą danych proszę zmienic 3308 na dowolny wolny port.

```yaml
db:
  build:
    context: ./db
    dockerfile: Dockerfile
  container_name: newdbcont
  ports:
    - "3308:3306"
  environment:
    MYSQL_DATABASE: shopdb
    MYSQL_USER: kuba
    MYSQL_PASSWORD: kubaru1234
    MYSQL_ROOT_PASSWORD: root
  volumes:
    - db-data:/var/lib/mysql
  networks:
    - app-network
```

Zmiana portów w pliku docker-compose.yml

Jeśli problem napotyka kontener odpowiedzialny za aplikacje to proszę zmienić 9092 na wolny port.

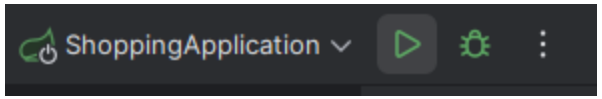


Opcja II

```
spring.datasource.url=jdbc:mysql://localhost:3308/shopdb
spring.datasource.username=kuba
spring.datasource.password=kubaru1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.flyway.enabled=false
spring.flyway.url=jdbc:mysql://localhost:3308/shopdb
spring.flyway.user=kuba
spring.flyway.password=kubaru1234
spring.flyway.schemas=shopdb
spring.flyway.validate-on-migrate=false
```

- Musimy pobrać jakiś server SQL, według gustu, ważne, aby wspierał mySQL (polecam Tomcata, bo instrukcja będzie się o nim opierała).
- Teraz majac mySQL tworzymy konto SQL. Po utworzeniu konta musimy stworzyć bazę danych o nazwie shopdb.
- Teraz w pliku application.properties trzeba zmienić dane użytkownika bazy na własne ustawione przy tworzeniu konta SQL, lub dodać konto SQL o podanych w pliku danych.
- Jeśli zostanie wyświetlony błąd podczas startu aplikacji JDBC link failure oznacza to, ze albo nie mamy odpowiednich uprawnień (złe dane w pliku application.properties) do kontaktu z baza danych, albo port na którym działa baza, lub jej nazwa jest błędnie podana. Możliwe są także inne błędy ze względu na inne dane techniczne sprzętu, więc serdecznie polecam Dockera, który umożliwi standaryzacje.



Jeśli postanowiliśmy skorzystać z opcji drugiej musimy jeszcze skonfigurować Tomcat'a, aby działał lokalnie

- Teraz proszę odpalić aplikacje, aby zobaczyć czy nie ma żadnych błędów.
- Jeśli wszystko przebiegło pomyślnie proszę znaleźć folder tomcata i plik server.xml (zazwyczaj w Windowsie znajduje się tutaj: TOMCAT_HOME/conf/server.xml a w linuxie to zalezy od sposobu instalacji i wielu innych czynników, ale zazwyczaj tutaj: /etc/tomcat/ , /var/lib/tomcat/ , /usr/share/tomcat/ , /var/log/tomcat/)
- Znajdź ten fragment:
- <Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443" />
- I zmień na:
- <Connector address="0.0.0.0" port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8443" />
- **To sprawi, że Tomcat będzie nasłuchiwał na wszystkich interfejsach, a nie tylko lokalnie.**

Koniec konfiguracji – czas na test

- Jeśli wszystko poszło zgodnie z planem proszę otworzyć swoją przeglądarkę i wpisać http://localhost:9092/home (9092 – jest to port, na którym działa aplikacja, jeśli zmieniliśmy port w pliku docker-compose.yml lub application.properties, jeśli nie zdecydowaliśmy się na opcje z Dockerem to proszę zmienić też w linku).

- Teraz końcowy etap testu - proszę sprawdzić swój adres ip (komputera na którym będzie działać aplikacja. Teraz proszę wpisać w przeglądarce na dowolnym urządzeniu podłączonym do tej samej sieci adres url, który będzie się skladal z adresu ip "servera", portu na którym działa aplikacja. Przykład:

Mój adres ip komputera z działającą aplikacją: 192.168.1.10

Skonfigurowany port : 9092

Pełny URL: 192.168.1.10:9092/