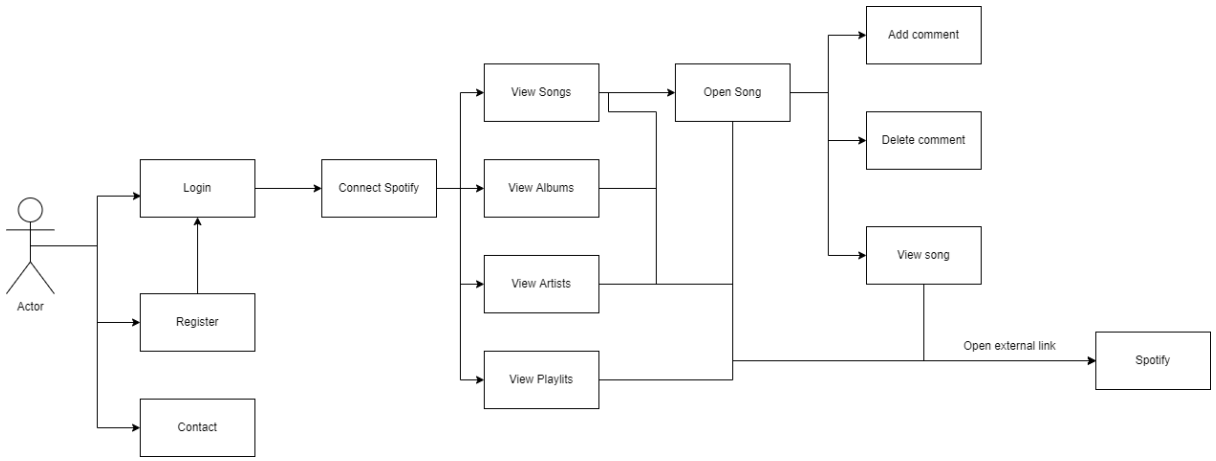
# SRApp

## Scopul aplicației

SRApp este o aplicație bazată pe microservicii care oferă modalități de vizualizarea și comentarea conținuturilor muzicale pe care utilizatorul le ascultă din aplicația Spotify. Sistemul permite utilizatorilor să își creeze un cont, să se conecteze la contul său de Spotify, să vizualizeze melodii, artiști, albume și playlisturi. Acesta poate să asculte un preview al melodiei, să afle anumite informații despre aceasta și să o poată comenta. De asemenea, acesta are și posibilitatea să-și șteargă propriul său comentariu.

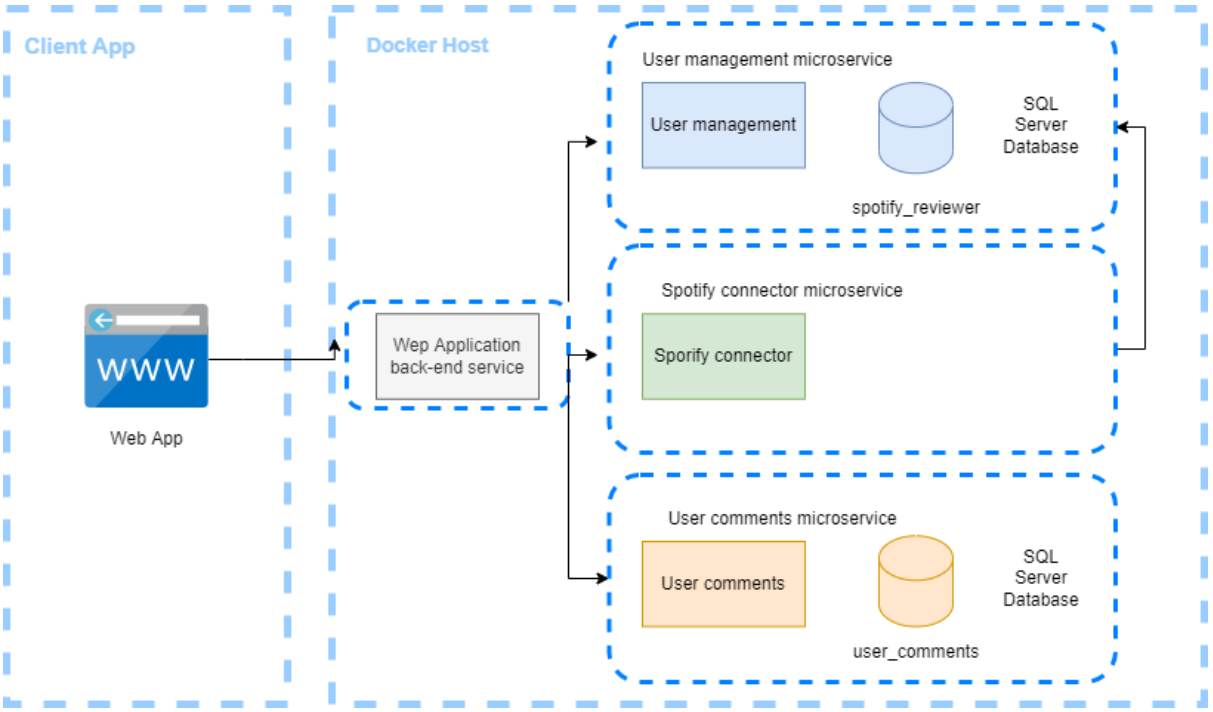
## Diagrama Use Case




În această diagramă regăsim pașii de urmat pe care îi are utilizatorul în fiecare moment din aplicație. Prima interacțiune a sa va fi cu pagina de **Login**, unde are posibilitatea să acceseze aplicația. Dacă acesta nu deține un cont are posibilitatea să ajungă pe pagina de **Register** unde își va crea un cont nou. Mai există și pagina de **Contact** unde se regăsesc informațiile despre dezvoltatorii aplicației și metoda de a îi contacta.

În momentul în care utilizatorul intră în cont, acesta va trebui să se conecteze cu contul său din aplicația Spotify. După ce a făcut acest lucru are posibilitatea să vizualizeze melodiile sale apreciate, albumele, artiștii și playlisturile care l-au încântat. Pentru fiecare categorie enumerată anterior, regăsim și o pagină dedicată acesteia. Categoria specială este cea de melodii. Pe pagina **Songs**, putem accesa orice melodie apreciată de noi. Fiecare melodie în parte are asignată o pagină dedicată ei. În această pagină putem avea informații despre denumirea melodiei, artiștiștii, albumul, popularitatea acestea, posibilitatea de ascultare a unui preview, adăugarea și ștergerea unui comentariu. Toate aceste pagini te pot redirecționa și la pagina din Spotify dedicată pentru ceea ce ai căutat.

## Arhitectura aplicației




SRApp este o aplicație bazată pe microservicii. Aceasta este formată din două părți. Partea din stânga se referă la aplicația client care este o aplicație web. Aplicația nu discută direct cu microserviciul, ci prima dată intră în contact cu **Web Application beck-end service**. Acesta are rolul de a discuta cu fiecare microserviciu, de a lua informațiile de la fiecare microserviciu, de a le face pachet și a le transmite **Aplicației client**. De exemplu, să presupunem că un utilizator logat dorește să vadă o melodie din aplicație. Aplicație web trimite un request, **Web App beck-end service** preia requestul, trimite un request către **User comments** și așteaptă răspunsul. După acest lucru, **Web App beck-end service** trimite răspunsul către **Web App**.

## Microservicii

1. **Web Application back-end service** - Acest serviciu se ocupă cu relația dintre Web App și celelalte microservicii. El servește modelele pentru a fi randate în aplicația web. Aceasta folosește atât requesturi sincrone, cât și asincrone.
2. **User management** - Acest serviciu deține baza lui de date. El se ocupă cu login, register și cu verificarea dacă userul este autentificat.
3. **Spotify connector** - Pe baza unui utilizator autentificat de autentifică în Spotify. El se mai ocupă cu regenerarea tokenului de acces. Acest token poate fi folosit mai departe pentru extragerea de date și metadate din Spotify. Refresh token este pasat la **User Management** care îl gestionează corespunzător.
4. **User comments** - Acest microserviciu se ocupă cu adăugarea și ștergerea unui comentariu la o anumită melodie.
   Toate microserviciile comunică prin REST API.

**Tehnologiile folosite sunt:**

Toate microserviciile împreună cu **Web App back-end** folosesc NodeJS. Pentru crearea tuturor API-urilor de tip REST se folosește Express. **User Managemen**t și **User Comments** folosesc MySQL pentru stocarea persistentă a datelor. Spotify connector folosește cele două metode “Request Access Token” și “Request a refreshed Access Token” recomandate în cerințele de dezvoltator din documentația Spotify (https://developer.spotify.com/documentation/general/guides/authorization/code-flow/).

**Web App beck-end** setează modelele care vor popula template-urile aplicației web implementate prin EJS. Aplicația Web folosește sesiuni pentru menținerea conectată a utilizatorui.

## Docker

Fiecare microserviciu este împachetat într-o imagine cu ajutorul Dockerfile.
Comanda folosită este: docker build -t <tag> <path_to_dockerfile>

Pot fi construite containere folosind aceste imagini în mai multe modalități:

- manual, cu docker run: docker run –name <instance_name> –network <container_network> –network-alias <name_to_be_resolved_in_dns_lookup> <image_name>
- folosind docker compose: acesta se folosește de un fișier declarativ yaml și este o modalitate facilă de a porni aplicațiile multi-container; comanda folosită este docker-compose up
- folosind docker stack în Swarm mode: folosește un fișier declarativ yaml similar cu docker compose, însă permite doar utilizarea imaginilor deja existente

Modul Docker Swarm permite scanarea serviciilor în timp real, actualizarea acestora folosind imagini noi, precum și fault-tolerance prin pornirea de noi containere în cazul unui crash astfel incat sa se pastreze numărul dorit de “replici”.
Pentru aplicația curentă, s-a ales folosirea de baze de date independente pentru fiecare serviciu ce necesită această funcționalitate, însă doar serviciul “user-comments” este prezent în 3 replici, restul fiind stateful, nepretându-se a fi folosite mai mult replici.

Setarile standard nu permit comunicarea pe rețea a containerelor între ele, fiind în izolare.
Pentru a permite conexiunea, o rețea comună a fost creata “micro-net”.
De asemenea, porturile serviciilor ce trebuie sa aib comunicare cu rețele externe au fost deschise.