

Universitatea „Politehnica” din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

*Sistem de management al unor gateway-uri pentru interfețe API*

## **Proiect de diplomă**

prezentat ca cerință parțială pentru obținerea titlului de  
*Inginer în domeniul Electronică și Telecomunicații*  
programul de studii de licență *Tehnologii și Sisteme de*  
*Telecomunicații*

Conducători științifici

*Conf. Dr. Ing. Alexandru VULPE*

*Alexandru GODRI*

Absolvent

*Delia Mihaela ANDONE*

*Anul 2022*



Universitatea "Politehnica" din București  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației  
Program de studiu **TST**.

**Anexa 1**

**TEMA PROIECTULUI DE DIPLOMĂ**  
a studentului **ANDONE S. Delia-Mihaela, 443C**

**1. Titlul temei:** Sistem de management al unor gateway-uri pentru interfețe API

**2. Descrierea temei și a contribuției personale a studentului (în afara părții de documentare):**

Proiectul își propune dezvoltarea și gestionarea unui gateway pentru interfețe API. Proiectul constă în mai multe etape. În prima etapă se vor analiza și alege două gateway-uri pentru interfețe API pentru integrare (Se vor considera gateway-uri existente precum Kong, Traefik, Apigee etc.) În următoarea etapă se va implementa un sistem simplu (create, read, update, delete) CRUD API. Acesta se va integra prin gateway-urile alese. În următoarea etapă se va implementa un sistem de punere în funcțiune prin două metode posibile: containere (docker compose) sau kubernetes (test local folosind minikube sau k3s). În ultima etapă se va implementa interfața grafică ce va permite următoarele funcționalități: login folosind SSO sau utilizatori locali; posibilitate deployment gateway într-un cluster de k8s (direct din interfață); posibilitate asociere API sau orice alt URL cu un API gateway deja pus în funcțiune.

**3. Discipline necesare pt. proiect:**

SDA, POO, BD

**4. Data înregistrării temei:** 2021-12-14 10:19:53

**Conducător(i) lucrare,**

Conf. Dr. Ing. Alexandru VULPE

Alexandru Godri, SIQSESS TECHNOLOGY S.R.L.

**Director departament,**

Cod Validare: **e8d29752fa**

**Student,**

ANDONE S. Delia-Mihaela

**Decan,**

Prof. dr. ing. Mihnea UDREA



## Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul “*Sistem de management al unor gateway-uri pentru interfețe API*”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică și Telecomunicații (ETC)* programul de studii *Tehnologii și Sisteme de Telecomunicații (TST)* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 24.06.2022

Absolvent Delia-Mihaela ANDONE



---



# Cuprins

<b>Introducere</b>	<b>13</b>
<b>Capitolul 1 - Aspecte teoretice</b>	<b>15</b>
1.1 Noțiuni generale:	15
1.2 Aprofundarea noțiunii de API Gateway	16
<b>Capitolul 2 - Analiza API Gateway-urilor alese</b>	<b>19</b>
2.1 Caracteristici Kong	19
2.2 Caracteristici TYK	19
2.3 Similitudini și diferențe între Kong și Tyk	20
<b>Capitolul 3 - Implementarea aplicației</b>	<b>21</b>
3.1 Entități principale	21
3.2 Arhitectura produsului	21
3.2.1 Arhitectura infrastructurii	21
3.2.2 Arhitectura software	23
3.2.3 Arhitectura de comunicare	24
3.2.4 Resursele din Docker	25
3.2.5 Resursele din Kubernetes	26
3.3 Detalierea scripturilor și a fișierelor de configurare	26
3.3.1 Dockerfile-ul API-ului de management	26
3.3.2 Scriptul executabil al API-ului de management	27
3.3.3 Fișierul de configurare a Kubectl-ului	28
3.3.4 Scriptul executabil al unei instanțe de Tyk	28
3.3.5 Fișierul de deployment al unei instanțe Kong	28
3.3.6 Fișierele frontend-ului	29
3.3.7 Fișierele API-urilor de test	29
3.4 Prezentarea generală a componentelor software dezvoltate	29
3.4.1 API-ul de management	29
3.4.2 Frontend-ul	31
3.4.3 API-urile de test	34
<b>Capitolul 4 - Prezentarea generală a arhitecturii rețelei</b>	<b>35</b>
<b>Capitolul 5 - Experiența utilizatorului</b>	<b>39</b>
5.1 Ghid de utilizare al aplicației	39
5.2 Specificațiile API-urilor REST de test (Swagger)	49
<b>Concluziile proiectului</b>	<b>53</b>
<b>Bibliografie</b>	<b>55</b>
<b>Anexa 1</b>	<b>57</b>





# Lista figurilor

Figura 1 Diagrama generală a unui API Gateway.....	17
Figura 3.1 Resursele folosite în cadrul proiectului.....	21
Figura 3.2 Arhitectura infrastructurii.....	22
Figura 3.3 Arhitectura software.....	23
Figura 3.4 Arhitectura de comunicare.....	24
Figura 3.5 Colecția de containere.....	25
Figura 3.6 Colecția de pod-uri.....	26
Figura 4.1 Arhitectura rețelei.....	35
Figura 4.2 Comunicare Docker/Kubernetes către rețeaua locală.....	36
Figura 4.3 Comunicarea între Docker și Kubernetes.....	37
Figura 5.1 Primul contact cu aplicația.....	39
Figura 5.2 Fluxul de autentificare.....	40
Figura 5.3 Fluxul de înregistrare.....	40
Figura 5.4 Pagina Home.....	41
Figura 5.5 Tabel de servicii din cadrul unei conexiuni de tip Kong.....	41
Figura 5.6 Formular de adăugare a unui serviciu din cadrul unei conexiuni de tip Kong.....	42
Figura 5.7 Formular de editare a unui serviciu din cadrul unei conexiuni de tip Kong.....	42
Figura 5.8 Fereastră de tip modal pentru ștergerea a unui serviciu din cadrul unei conexiuni de tip Kong.....	43
Figura 5.9 Formular de adăugare a unei rute din cadrul unei conexiuni de tip Kong.....	44
Figura 5.10 Formular de editare a unei rute din cadrul unei conexiuni de tip Kong.....	44
Figura 5.11 Fereastră de tip modal pentru ștergerea unei rute din cadrul unei conexiuni de tip Kong.....	44
Figura 5.12 Tabel de conexiuni.....	45
Figura 5.13 Formular de adăugare a unei conexiuni.....	45
Figura 5.14 Formular de editare a unei conexiuni.....	45
Figura 5.15 Fereastră de tip modal pentru ștergerea unei conexiuni.....	46
Figura 5.16 Conexiunea unei instanțe Kong.....	46
Figura 5.17 Rută asociată unui serviciu din cadrul unei conexiuni de tip Kong.....	47
Figura 5.18 Serviciu din cadrul unei conexiuni de tip Kong.....	47
Figura 5.19 Exemplu de răspuns al primului API de test.....	47
Figura 5.20 Conexiunea unei instanțe Tyk.....	48
Figura 5.21 Rută asociată unui serviciu din cadrul unei conexiuni de tip Tyk.....	48
Figura 5.22 Serviciu din cadrul unei conexiuni de tip Tyk.....	48
Figura 5.23 Exemplu de răspuns al celui de-al treilea API de test.....	48
Figura 5.24 Listarea endpoint-urilor și a metodelor suportate al primului API de test.....	49
Figura 5.25 Descrierea metodei PUT din cadrul primului API de test.....	50
Figura 5.26 Listarea endpoint-urilor și a metodelor suportate al celui de-al doilea API de test.....	50
Figura 5.27 Listarea endpoint-urilor și a metodelor suportate al celui de-al treilea API de test.....	51



# Lista acronimelor

API - Application Programming Interface = Interfață de programare a aplicației  
CRUD - Create, Read, Update, Delete = Creare, Citire, Editare, Ștergere  
DB - Database = Bază de date  
ENV - Environment = Mediu  
gRPC - General purpose remote procedure call = Apel general de procedură/funcție la distanță  
HTML - Hypertext Markup Language = Limbaj de marcare hipertext  
HTTP - Hypertext Transfer Protocol = Protocol de transfer hipertext  
HTTPS - Hypertext Transfer Protocol Secure = Protocol de transfer hipertext securizat  
IP - Internet Protocol = Protocol interrețea  
ID - Identifier = Identificator  
JSON - JavaScript Object Notation = Notarea obiectelor JavaScript  
REST - REpresentational State Transfer = Transfer de stare reprezentativ  
SOAP - Simple Object Access Protocol = Protocol de access folosind obiecte simple  
TCP - Transmission Control Protocol = Protocolul de control al transmisiei  
URL - Uniform Resource Locators = Localizator uniform de resurse  
YAML - Yet Another Markup Language = Încă un alt limbaj de marcare



# Introducere

Într-o lume în care Cloud-ul și aplicațiile din internet devin tot mai pregnante, apar foarte multe probleme de expunere a serviciilor în internet. De asemenea, faptul că există o multitudine de aplicații de tip container, ce rulează în medii virtualizate, cum ar fi: servicii Web (gmail, google, drive, etc.), API-uri, servicii virtualizate de networking, crește nivelul de complexitate și abstractizare a soluțiilor clasice privind expunerea directă a unui serviciu în internet.

Deoarece există amenințarea continuă a unor breșe de securitate, devine tot mai necesară utilizarea unei aplicații care guvernează și gestionează acest trafic.

Așadar, problema care reiese este necesitatea unei soluții performante și ușoare care să faciliteze configurarea endpoint-urilor expuse în internet.

Există o mulțime de soluții ce își propun să realizeze acest obiectiv, fiecare având avantaje, dezavantaje și costuri asociate. În cadrul cercetărilor mele nu am reușit să găsesc o soluție similară ce își propune să unifice funcționalitatea mai multor API Gateway-uri diferite.

Astfel, în această lucrare îmi propun să implementez o interfață Web împreună cu un API care să unifice una sau mai multe soluții de tipul API Gateway, astfel încât utilizatorul să aibe o experiență unică în utilizarea aplicației indiferent de tipul de API Gateway ales.

În acest sens, am ales să dezvolt în Python, folosind microframework-ul Flask [1] un API de management care se folosește de API-urile de administrare ale API Gateway-ului Kong [2], respectiv Tyk [3], cu ajutorul căruia se pot expune endpoint-uri ale unor aplicații de tip API simple (create, read, update, delete) care vor fi gestionate de sistem.

Întrucât am avut în vedere portabilitatea, implementarea lucrării a fost pusă în funcțiune cu ajutorul sistemului de orchestrare de containere, Docker [4].

Interfața grafică permite următoarele funcționalități: login folosind useri locali, posibilitate de punere în funcțiune a unui API Gateway prin deployment, într-un cluster de Kubernetes [5], iar în final, asocierea unui API de tip CRUD cu unul dintre cele două API Gateway-uri sau, în funcție de preferință, cu amândouă.

Pentru testarea funcționalității, am ales să creez trei API-uri simple, având tematicile: cărți, mașini, pokemoni.

Folosind acest produs, utilizatorul nu trebuie să treacă prin etapa de documentare detaliată a API Gateway-urilor, datorită faptului că punerea în funcțiune a acestora și asocierea lor cu API-uri de tip CRUD sunt deja abordate.

De asemenea, folosind interfața grafică, eforturile utilizatorului de asocia un API simplu cu un API Gateway sunt reduse, datorită simplității de utilizare. Astfel, pentru crearea unor servicii și rute, se completează doar un subset de câmpuri, iar după aceste operații avem ca rezultat definirea endpoint-urilor în produs. Totodată, este facilitată vizualizarea conexiunilor, serviciilor și a rutelor, cu posibilitatea de adăugare, editare sau ștergere a acestora.



# Capitolul 1 - Aspecte teoretice

## 1.1 Noțiuni generale:

În această secțiune voi expune principalele resurse și concepte utilizate în această lucrare. Voi detalia pentru început câteva concepte de bază urmând ca gradul complexității să crească odată cu gradul de abstractizare a noțiunilor expuse.

**CRUD**, acronimul provine de la create, read, update, delete. Acestea sunt operațiile ce vor fi folosite pe parcursul lucrării pentru a defini manipularea resurselor și a datelor. [6]

**API** sau application programming interface se referă la o interfață scrisă ce permite înțelegerea conceptelor într-un mod abstract fără a avea în vedere detaliile de implementare. În cazul acestei lucrări vom limita noțiunea la definirea unui set de metode/rute ce permit execuția unor acțiuni specifice din registrul CRUD. [7]

**API Gateway** se referă la un sistem prin care se controlează accesul la un API specific. De exemplu Produsul A oferă endpointurile a, b, c și d. Prin expunerea lor dintr-un API Gateway putem restricționa accesul astfel încât din exterior să fie vizibile doar a și c. Rolul unei astfel de soluții este de a controla accesul utilizatorilor finali la API-urile expuse. [8]

**REST** sau Representational State Transfer este un model de arhitectură pentru aplicații peste rețea. Principala caracteristică a acestei arhitecturi este lipsa stării. Astfel pentru a satisface condiția de idempotență când o metodă REST este apelată în cadrul unui API cu un set de date de intrare va furniza același rezultat indiferent de starea sistemului sau de trecerea timpului. [9]

**Virtualizarea** este procedeul prin care se abstractizează hardware-ul fizic folosind concepte și modele de tip software. [10]

**Containerul** este un super proces prin care se poate obține o formă de virtualizare cu un consum mult redus de resurse față de o mașină virtuală. [11]

**Docker** este o colecție software ce permite abstractizarea infrastructurii. Astfel obținem portabilitatea soluției.

**Kubernetes** este o colecție de aplicații software ce virtualizează infrastructura folosind mai multe concepte. Acesta fiind complex, având foarte multe componente și resurse voi enumera câteva dintre ele, care au fost folosite în cadrul deploymentului de Kubernetes: Deployment, Services, Job, Persistent Volume respectiv Persistent Volume Claim.

**Pod** este cea mai mică unitate ce poate fi configurată și lansată în Kubernetes. Acesta este format din unul sau mai multe containere având de obicei aceeași imagine. [12]

**Serverul** este un program sau colecție de programe ce rulează pe un suport fizic. [13] În această lucrare, voi folosi o infrastructură de tip virtual Docker pentru API-uri respectiv Kubernetes pentru API Gateways. Lucrarea a fost dezvoltată și va fi demonstrată folosind laptopul personal.

**YAML** este un limbaj descriptiv folosit pentru configurări și pentru structurarea datelor. [14] În cazul acestui proiect este folosit pentru a descrie resurse de tip Kubernetes.

**JSON** este un limbaj descriptiv folosit pentru configurări și pentru structurarea datelor. [15] În cazul acestui proiect este folosit pentru comunicarea între frontend, backend și alte servicii. De asemenea este folosit în Tyk pentru persistența resurselor, fiind salvate ca un fișier.

**Deployment** este o resursă de tip Kubernetes prin care, folosind limbajul YAML descriu pod-urile și imaginile folosite pentru a rula aplicația. [16]

**Services** este o resursă de tip Kubernetes prin care descriu conectivitatea la nivelul clusterului de Kubernetes și porturile expuse, folosind limbajul YAML. [17]

**Persistent Volume** este o resursă de tip Kubernetes prin intermediul căreia, folosind limbajul YAML descriu cerințele de stocare și path-ul pe care îl voi avea în container. [18]

**Persistent Volume Claim** este o resursă de tip Kubernetes prin intermediul căreia descriu tipul de volum dorit și permisiunile acestuia, folosind limbajul YAML.

**Job** este o resursă de tip Kubernetes, care creează unul sau mai multe pod-uri efemere, ce au ca scop execuția unei sarcini specifice, după care se va închide. Acesta va reîncerca execuția sarcinii până când aceasta va fi un succes. [19]

**Baza de date** reprezintă o colecție software formată din datele efective și sistemul de gestiune al bazei de date. [20]

**Frontend-ul** este o aplicație care este constituită dintr-o interfață grafică și este expusă direct utilizatorului. În majoritatea implementărilor, are o logică limitată și este responsabilă de afișarea și validarea datelor. [21]

**Backend-ul** este o aplicație care este responsabilă cu logica și manipularea datelor, stocarea cât și obținerea lor. De obicei are ca suport o aplicație de tip baze de date pentru a asigura persistența lor. [22]

**Python** este un limbaj de programare interpretat. [23] În cadrul lucrării este folosit pentru scrierea serviciilor simple de tip REST, dar și a API-ului de management și a frontendului.

**HTML** este un limbaj descriptiv folosit pentru a descrie paginile web. [24]

**Endpoint** este o interfață expusă de un backend ce are la bază o implementare REST. Exemplu: „adresa-unui-API:port/resursă”.

**Kernelul** sau nucleul sistemului de operare este responsabil pentru comunicarea cu hardware-ul în mod deosebit cu microprocesorul respectiv memoria și cu aplicațiile ce rulează la nivelul nucleului de operare (care sunt bazate pe nucleul sistemului de operare). [25]

**Apelul de sistem** este o funcție prin care o aplicație solicită un serviciu expus din nucleul sistemului de operare. [26]

**Swagger** este o bibliotecă disponibilă în mai multe limbaje care ajută la crearea unei documentații pentru API-uri. [27]

**Cloud** reprezintă un ansamblu de aplicații, servicii și sisteme distribuite care rulează într-o mulțime de datacenter conectate prin internet. [28]

**Microserviciu** este o aplicație Web cu un rol specific. Acesta este de mici dimensiuni și are un consum limitat de resurse. [29]

## 1.2 Aprofundarea noțiunii de API Gateway

API Gateway-ul reprezintă o unealtă situată între client și colecția de servicii din backend. Astfel, preia request-urile de la client și le redirectionează către serviciul potrivit din backend. Rolul acestuia este centralizarea mai multor API-uri.

Caracteristica de bază a unui astfel de sistem este de a controla accesul către endpoint-urile pe care le supraveghează.

La nivelul aplicației de management dezvoltată în cadrul acestei lucrări, am folosit noțiunea de servicii respectiv rute pentru a separa și defini logic conceptele cu care lucrez. Astfel un serviciu va reprezenta rădăcina căii de la care poate fi accesat API-ul pe care doresc să îl gestionez, iar ruta semnifică calea de acces către API-ul gestionat. Aceste modele au fost mapate cu Kong foarte ușor, datorită faptului că sunt conceptele de baza utilizate și în cadrul dezvoltării Kong. Pentru Tyk a fost



nevoie să folosesc resursa „apis” și să o împart în servicii și rute pentru a putea menține convenția deja aleasă.

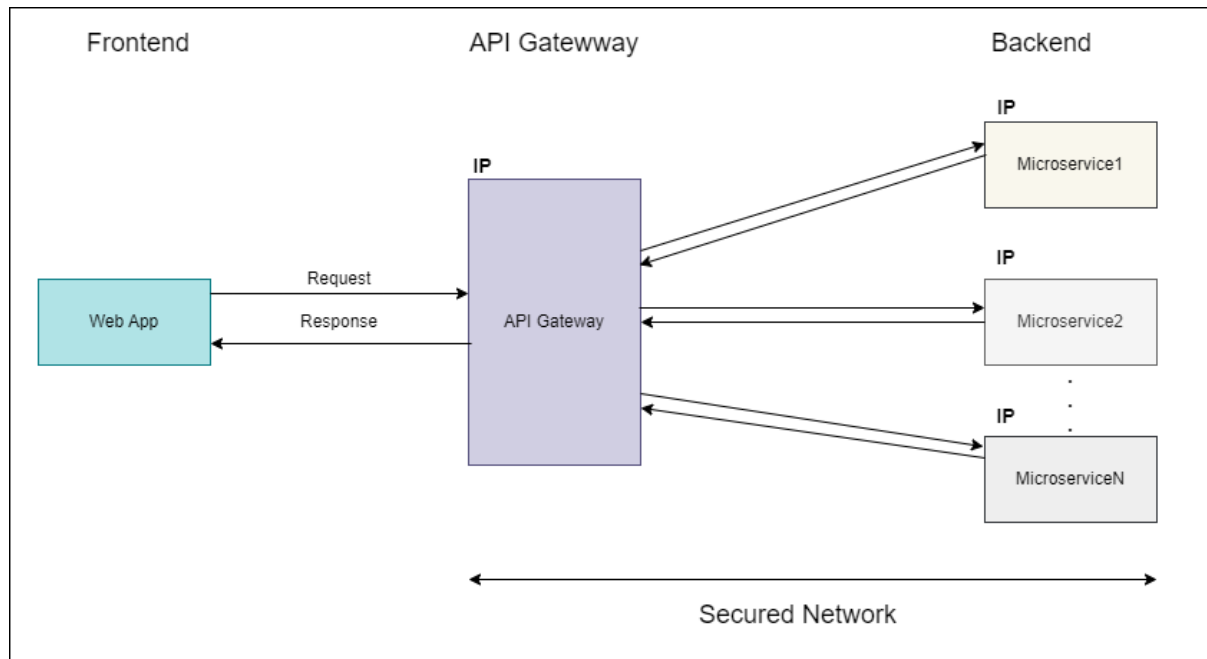


Figura 1 Diagrama generală a unui API Gateway

Conform figurii (Figura 1), utilizatorul va folosi Web App pentru a gestiona și configura din interfața grafică API Gateway-ul astfel încât să funcționeze conform nevoilor acestuia. Fiecare microserviciu de tip Backend are o adresă IP distinctă, același lucru este valabil și pentru API Gateway, ele aflându-se în aceeași rețea. Astfel utilizatorul sau serviciul ce consumă rutele puse la dispoziție printr-un API Gateway are nevoie să știe o singură adresă IP (cea a gateway-ului) restul fiind mascate și știute doar prin configurarea API Gateway-ului. Acest lucru facilitează dezvoltarea și conectarea aplicațiilor.

De asemenea, trebuie menționat că în cazul unei defecțiuni majore, API Gateway-ul este singurul punct de acces, devenind astfel un punct critic, iar oprirea acestuia va afecta orice alt sistem care depinde de el.

Alt aspect important este faptul că, odată introdus, API Gateway-ul va crește latența, în mod normal insesizabil dat fiind că există încă un hop prin care un request va trebui să treacă între serviciul ce consumă un API ce este gestionat de API Gateway.

Dintre caracteristicile unui API Gateway, amintesc:

- **autentificarea**, ce reprezintă procesul de identificare a utilizatorilor.
- **autorizarea**, prin care se definește ce resurse din sistem va putea accesa utilizatorul autentificat.
- **retry policy**; mecanismul de retry se asigură că aplicația face o altă încercare de a obține datele în cazul în care apare o eroare. [30]
- **circuit breaker** previne aplicația de a încerca în mod repetat să execute o operație care cel mai probabil va eșua. [31]
- **rate limiting**, prin care se aplică o limită în ceea ce privește numărul de request-uri permis.
- **load balancing** la nivelul căruia se ia decizia pe care dintre microserviciile din backend va fi transmis requestul.



## Capitolul 2 - Analiza API Gateway-urilor alese

În cadrul lucrării, cele două produse de tip API Gateway alese sunt Kong respectiv Tyk. Aceste soluții sunt disponibile gratuit în diverse forme de la executabil, imagine de docker cât și versiuni pentru cloud.

### 2.1 Caracteristici Kong

API Gateway-ul Kong dispune de o documentație bine realizată, în cadrul căreia este detaliat API-ul de administrare al acestuia [32], ce este responsabil de orice operație făcută asupra instanței de Kong, fiind apelat pentru realizarea acestora.

Dintre principalele resurse ale Kong-ului, în acest proiect au fost folosite „servicii” și „rute”. Astfel, după cum îi implica numele, un serviciu face referire la microserviciile din backend [33]. Rutele sunt entitățile asociate serviciilor, semnificând un set de reguli pentru a realiza corespondența pe request-urile clientului. [34] Se menționează faptul că un serviciu poate avea mai multe rute.

Principala sursă de documentare a proiectului în cadrul integrării Kong este documentația oficială privind API-ul de administrare. Așadar, accesând endpoint-ul „/services”, am posibilitatea folosirii metodelor „GET”, „POST”, „PUT”, „DELETE”, cu ajutorul cărora se pot implementa operațiile de tip CRUD. Același lucru este valabil și în cadrul resurselor de tipul „rute”, folosind endpoint-ul „/routes”.

Kong dispune de patru porturi prestabilite. În acest sens, se enumeră: portul 8000 care ascultă traficul HTTP primit de la client și îl redirecționează către microserviciile din backend, 8001 este portul implicit pe care API-ul de administrare ascultă apelurile de tip HTTP, portul 8443 ascultă traficul HTTPS (similar cu portul 8000) și 8444, care este portul implicit pe care API-ul de administrare ascultă apelurile de tip HTTPS.

Plugin-ul este o resursă suplimentară și opțională a unei aplicații ce extinde funcționalitățile acesteia. Pentru Kong, implică adăugarea caracteristicilor, precum: rate limiting, authentication, etc. API Gateway-ul Kong este disponibil în două variante: Open Source și Enterprise, cea din urmă având o licență asociată unui cost. Totodată, o instanță de Kong este ușor de instalat și configurat, instalarea fiind posibilă prin numeroase platforme, precum: Docker, Kubernetes, Ubuntu, CentOS, etc.

Câteva dintre avantajele folosirii unui astfel de API Gateway este consumul mic de resurse și asigurarea unei latențe scăzute.

### 2.2 Caracteristici TYK

Similar Kong-ului, Tyk dispune de un API de administrare [35], ce permite realizarea operațiilor de tip CRUD asupra resurselor sale. În cadrul proiectului, am folosit resursa de tipul „apis” ce este un obiect care descrie configurația unui microserviciu din backend.

Portul prestabilit pentru Tyk este 8080, pe care API-ul de administrare ascultă apelurile și care ascultă traficul primit de la client, redirecționându-l către microserviciile din backend.

În cazul în care se folosește o interfață grafică, resursele de tip „apis” sunt stocate într-o bază de date Mongo DB, altfel sunt stocate ca fișiere de tipul JSON în directorul „apps”, însă trebuie menționat faptul că Tyk oferă o licență de paisprezece zile de tip Trial (încercare) după care este plătită.

Asemănător Kong-ului, API Gateway-ul Tyk este disponibil în variantele: Open Source și Enterprise.

În ceea ce privește avantajele oferite, asigură o latență scăzută și posibilitatea de transformare a cererilor dintr-un protocol de comunicare în altul (de exemplu SOAP în JSON).

## **2.3 Similitudini și diferențe între Kong și Tyk**

O primă asemănare între cele două API Gateway-uri este reprezentată de faptul că sunt disponibile în două variante: Open Source și Enterprise. De asemenea, ambele asigură o latență scăzută și oferă suport pentru o gamă variată de protocoale.

Dintre diferențele dintre acestea, cea care a avut cel mai semnificativ impact în procesul de implementare al proiectului este legată de interfața grafică. Astfel, Kong oferă o interfață grafică „Konga” gratuită, timp în care, Tyk oferă o licență gratuită, pe durata a paisprezece zile, după care este plătită. Totodată, Kong oferă o documentație bine pusă la punct chiar și în varianta Open Source, fiind considerabil mai detaliată față de cea a Tyk-ului.

Kong oferă o multitudine de plugin-uri, în timp ce Tyk oferă posibilitatea de a le implementa.

Totodată, Kong oferă o validare mult mai completă și corectă a câmpurilor din cereri (request-uri).

Tyk oferă o simplitate a resurselor logice definite, astfel, resursa specifică „apis” înglobează entitățile „services” și „routes” din cadrul Kong-ului.

Spre deosebire de Kong, Tyk nu are nevoie de o bază de date pentru a rula, stocând informațiile referitoare la resurse într-un director sub forma unor fișiere.

În cadrul lucrării, nu sunt interesată de o comparație directă între cele două soluții dat fiind că amândouă vor fi înglobate în proiect.

# Capitolul 3 - Implementarea aplicației

## 3.1 Entități principale

În cadrul proiectului, se folosesc următoarele resurse: conexiuni, servicii și rute. Astfel, am definit conexiunile sub forma unor entități de tipul API Gateway, cu următoarele caracteristici: „ID”, „Name”, „Type” ( ce poate fi Kong sau Tyk) și „Admin API URL”, ce reprezintă URL-ul API-ului de administrare al API Gateway-ului respectiv, pe baza căruia creez celelalte resurse în cadrul conexiunii respective.

Așadar, o conexiune poate avea mai multe rute și mai multe servicii. Prin serviciu se înțelege rădăcina căii de la care poate fi accesat API-ul pe care doresc să îl gestionez. Această resursă este caracterizată de câmpurile: „ID”, „Name”, „Path”, ce semnifică rădăcina menționată, „Port”, ce reprezintă portul pe care rulează API-ul gestionat, în interiorul containerului și „Host”, fiind adresa IP sau denumirea containerului în cauză.

Ruta semnifică calea de acces către API-ul gestionat. Pentru aceasta, sunt definite câmpurile: „ID”, „Name”, „Paths”, „Service”, unde cel din urmă reprezintă numele serviciului care este corelat cu ruta respectivă.

Primul pas în procesul de asociere a unui API simplu cu un API Gateway în cadrul proiectului va fi crearea unei conexiuni. Pentru realizarea cu succes a asocierii, este necesară atât crearea unui serviciu cât și crearea unei rute corelate cu acesta.

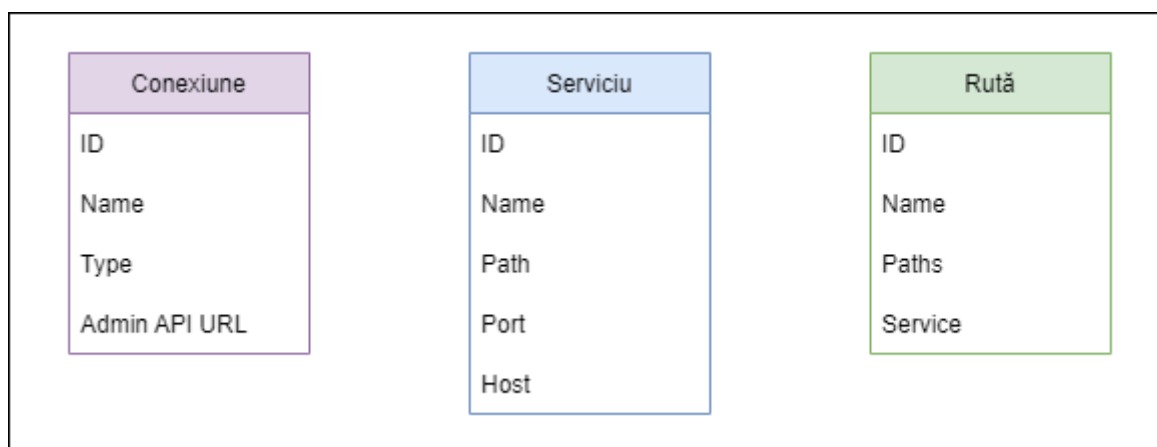


Figura 3.1 Resursele folosite în cadrul proiectului

## 3.2 Arhitectura produsului

### 3.2.1 Arhitectura infrastructurii

Docker reprezintă o unealtă de virtualizare, ce se folosește de Kernelul host-ului. Acesta se bazează pe imagini care conțin pachetul aplicației în cauză, împreună cu configurările și dependențele necesare. Kubernetes este o altă unealtă de virtualizare a infrastructurii, care este de o complexitate sporită față de Docker.

Produsul folosește o infrastructură virtuală hibrid. Astfel, majoritatea componentelor acestuia sunt în Docker, iar restul în Kubernetes.

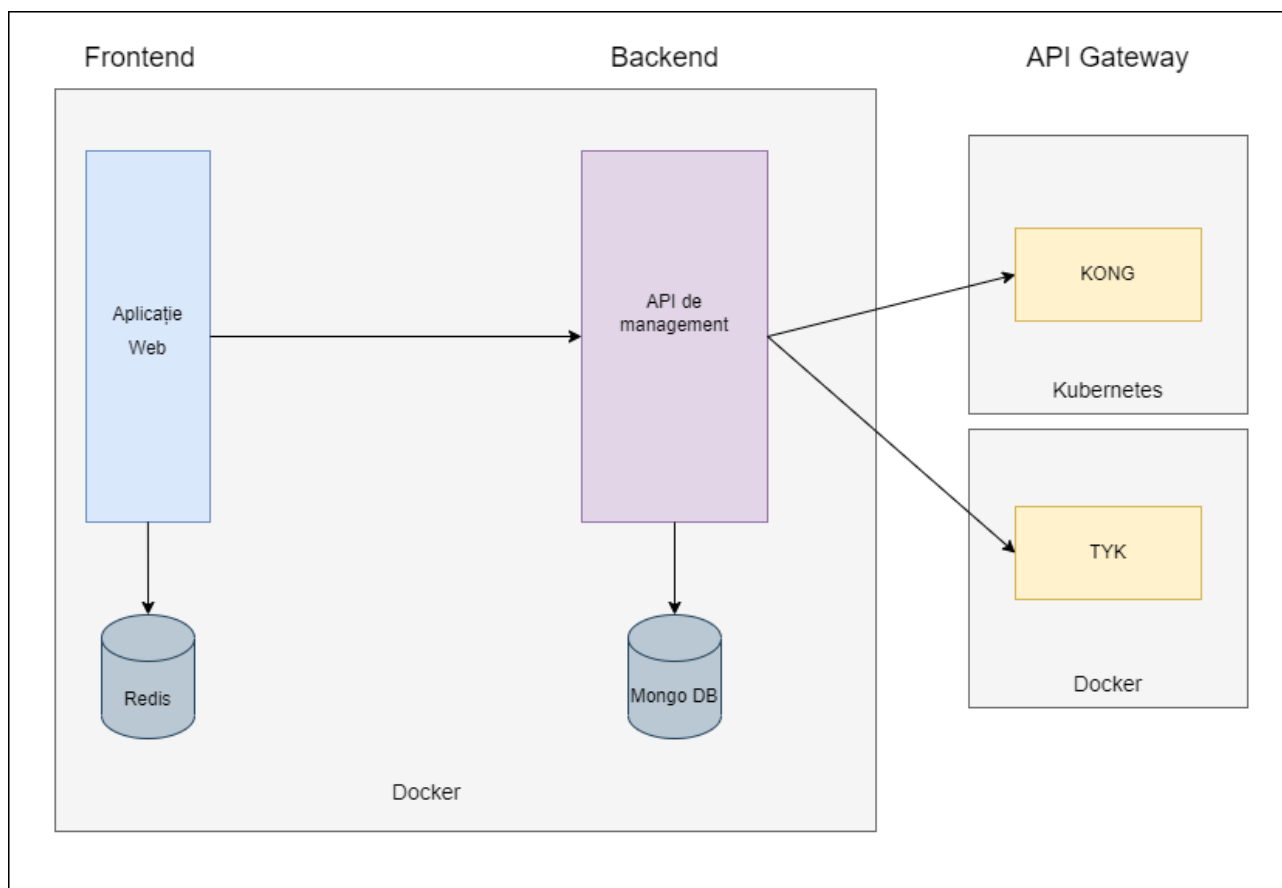


Figura 3.2 Arhitectura infrastructurii

Produsul este compus dintr-o multitudine de containere și pod-uri. În Docker, la nivelul containerelor se regăsesc: API-ul de management, aplicația web, bazele de date Redis și Mongo DB și API Gateway-urile de tip Tyk. În cazul containerelor care reprezintă bazele de date, am pornit de la imaginile oficiale, preluate din depozitul public al Docker-ului, Docker Hub: „redis”[36], respectiv „mongo”[37]. Atât pentru containerul care reprezintă API-ul de management cât și pentru cel care reprezintă aplicația Web, am creat imagini care au la bază varianta de Python 3.10.2. În cazul Tyk, se pornește de la imaginea „docker.tyk.io/tyk-gateway/tyk-gateway:latest”, folosind baza de date Redis, pentru procesul de cache, astfel încât să răspundă cererilor într-un mod rapid.

În ceea ce privește resursele din Kubernetes, au fost create pod-uri care pun în funcțiune API Gateway-urile Kong. Numărul acestora este dinamic, în funcție de dorința sau necesitatea utilizatorilor.

Pe de o parte, punerea în funcțiune a unui API Gateway de tip Kong, are la bază un fișier de tip yaml, are în componență un persistent volume, un persistent volume claim, două deployment-uri, două servicii și un job. Persistent volume alături de persistent volume claim asigură un volum persistent pentru aplicație, unul dintre deployment-uri este folosit pentru a porni baza de date Postgres, pe când celalalt lansează în execuție API Gateway-ul. În cadrul serviciilor, se descriu porturile expuse, necesare Kong-ului și Postgres-ului, iar job-ul porneste componenta efemeră necesară inițializării bazei de date.

### 3.2.2 Arhitectura software

Flask reprezintă un microframework (nu necesită biblioteci sau unelte particulare) scris în Python, folosit pentru a dezvolta aplicația Web și API-ul de management. Pe lângă Flask, pentru dezvoltarea produsului, am folosit Jinja[38], un template engine pentru Python, ce conține toate facilitățile necesare. Suplimentar, am utilizat HTML, care este un limbaj descriptiv folosit pentru a descrie paginile web.

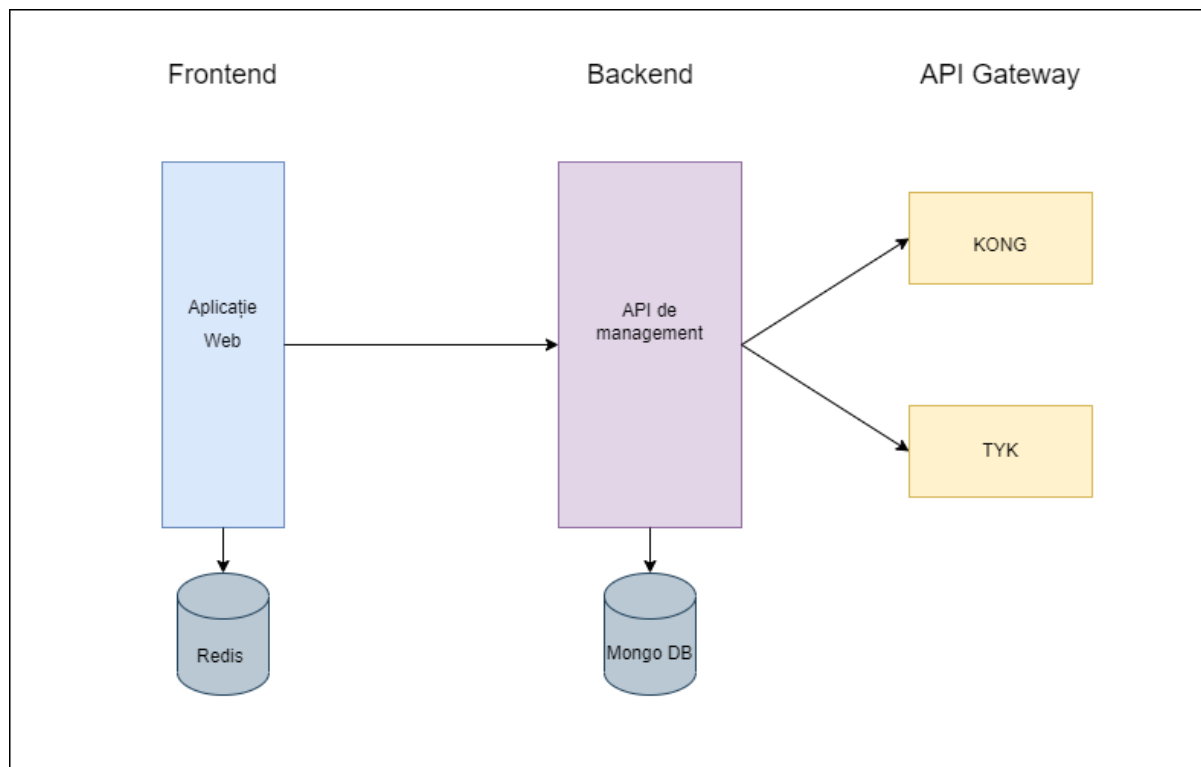


Figura 3.3 Arhitectura software

Una dintre principalele componente ale proiectului este API-ul de management, care reprezintă esența backend-ului, cu ajutorul căruia, sunt creați userii și conexiunile, folosindu-se de baza de date Mongo DB pentru a asigura persistența informațiilor. Totodată, prin intermediul acestuia, se apelează API-urile de administrare ale Kong-ului, respectiv ale Tyk-ului, pentru a crea servicii și rute. De asemenea, utilizând baza de date Mongo DB, sunt stocate informații cu privire la dependențele dintre: useri-conexiuni, conexiuni-servicii, conexiuni-rute. API-ul de management oferă atât posibilitatea de editare și ștergere a resurselor deja create, cât și posibilitatea de ștergere a dependențelor existente.

O altă componentă importantă este reprezentată de aplicația Web, având ca suport baza de date Redis, prin care se verifică starea de autentificare a utilizatorului și se asigură expunerea datelor pentru identificarea utilizatorului în tranziția dintre pagini. Paginile Web sunt definite în cadrul aplicației, respectând structura HTML și Python, astfel: fiecare pagină are în componența sa un fișier de tipul „pagină.html”, aflat în directorul „templates” și un fișier de tipul „pagină.py”, din directorul „routes”. În cazul fișierelor de tip HTML, structura paginilor este generată, folosind Jinja templates, iar în ceea ce privește fișierele de tip py, pentru a crea rute în cadrul acestui API, am folosit microframework-ul Flask.

În componența Kong-ului se regăsește baza de date Postgres, ce ajută la persistentă informațiilor referitoare la servicii și rute; suplimentar, există și o componentă efemeră, care inițializează baza de

date. Tyk-ul nu se folosește de o baza de date pentru a asigura persistența datelor, astfel, resursele create sunt stocate sub forma unor fișiere de tipul JSON, din folderul „apps”.

### 3.2.3 Arhitectura de comunicare

Docker Network reprezintă rețeaua virtuală utilizată de Docker engine pentru a asigura comunicarea între containere. Pentru a facilita comunicarea între cele două medii de virtualizare, am folosit Minikube Tunnel prin care creez o rută între host și adresa IP a clusterului de Kubernetes, care, de asemenea rulează pe host.

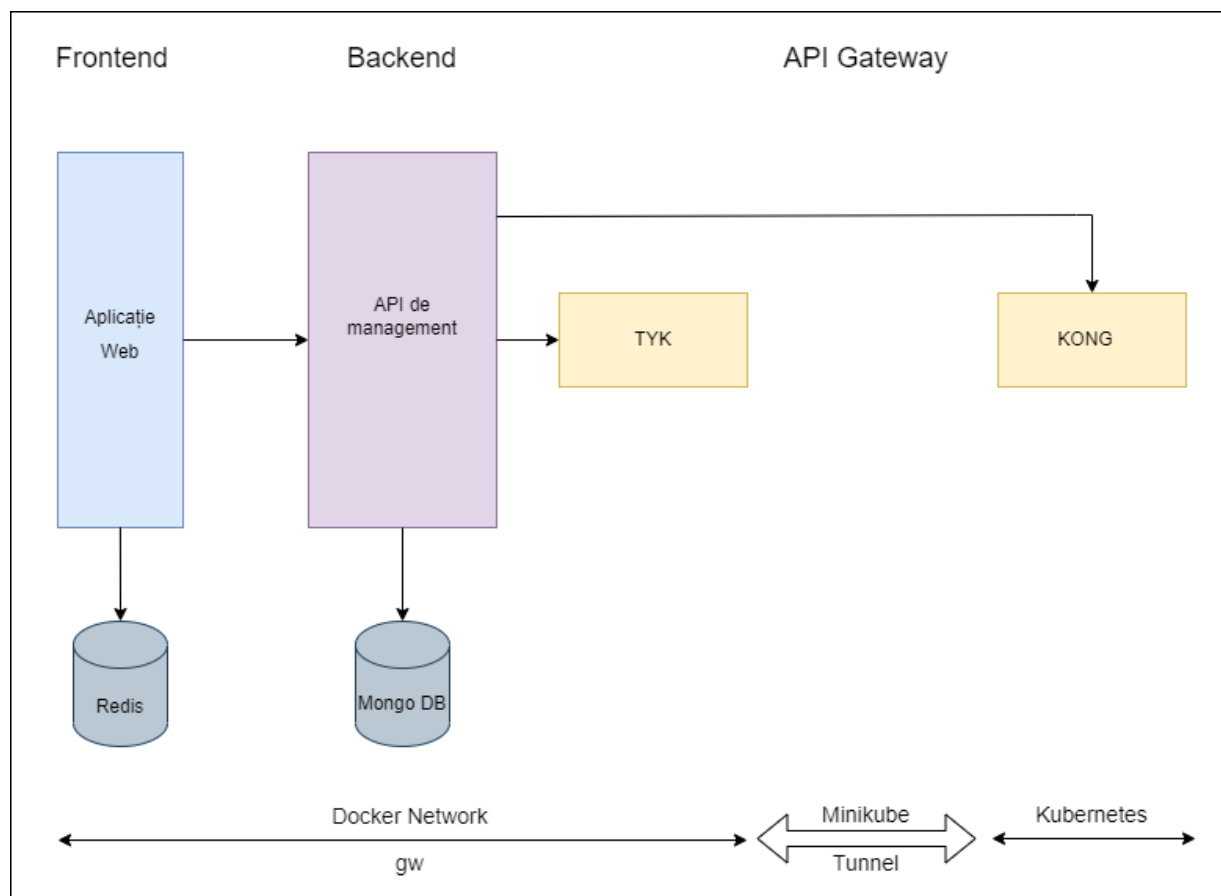


Figura 3.4 Arhitectura de comunicare

Conform figurii (Figura 3.4), componentele din Docker ce alcătuiesc backend-ul și frontend-ul, împreună cu bazele de date și API Gateway-urile de tip Tyk, se află în același Docker Network, denumit „gw”. Principalul avantaj fiind reprezentat de abilitatea de a se vedea unele pe altele pe baza hostname-ului, fără a mai fi necesară utilizarea adreselor IP.

Am avut în vedere crearea unui Docker Network, din cauza faptului că adresele IP pot fi dinamice (se pot modifica). Așadar, în cazul în care containerele ar fi fost pornite într-o altă ordine, acestora li s-ar fi asociat alte adrese IP, iar, pentru a putea utiliza produsul final, acest lucru ar fi presupus mai întâi o modificare la nivelul configurării fiecăruia.

Pentru a realiza comunicarea dintre Docker și Kubernetes este necesară pornirea unui serviciu de tunelare. Serviciul utilizat este Minikube Tunnel. Astfel, componentele din Docker le pot vedea pe cele din Kubernetes și vice versa.

Componentele din Kubernetes sunt expuse pe hostul local, iar modalitatea de diferențiere se realizează prin expunerea portului.



### 3.2.4 Resursele din Docker

Subsistemul Ubuntu pentru Windows este o extensie a sistemului de operare, ce permite rularea aproximativ nativă a aplicațiilor pentru Linux. În timp ce aplicațiile Linux se folosesc de biblioteci (apeluri de sistem) din familia Unix, subsistemul de Ubuntu extinde Kernelul de Windows, oferind multe funcții (apeluri de sistem), nativ integrate în Kernelul de Windows.

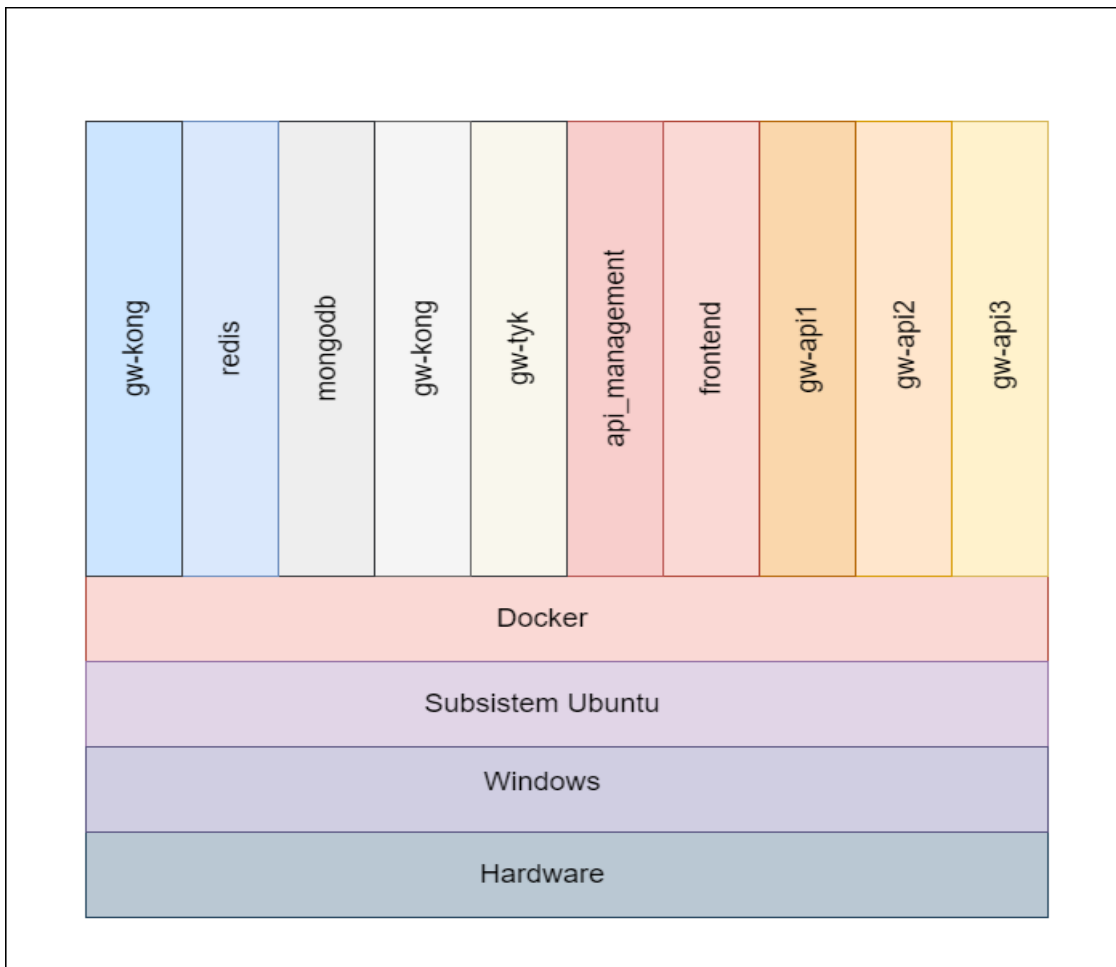


Figura 3.5 Colecția de containere

Componenta hardware este reprezentată de laptopul personal, pe care este instalat sistemul de operare Windows 10. Subsistemul Ubuntu este instalat în Windows 10, cu ajutorul căruia Docker-ul rulează containere.

În cadrul containerelor, voi enumera și prezenta pe scurt fiecare componentă (fiecare container) în parte. Astfel, în ceea ce privește bazele de date, pentru a avea un volum persistent în cadrul Kong-ului se folosește Postgres, integrată în containerul „gw-kong”. De asemenea, se folosește Mongo DB cu scopul de a stoca informații cu privire la utilizatori, conexiuni și dependențele dintre resurse, într-un container cu numele „mongodb” și Redis pentru a verifica starea de autentificare a userului și a expune datele acestuia în tranziția dintre paginile Web, cât și pentru a asigura procesul de cache în cadrul Tyk-ului, în containerul „redis”.

Cu privire la API Gateway-urile alese, se identifică containerele „gw-kong” și „gw-tyk”, care (în cazul Kong-ului, împreună cu componenta ce include baza de date postgres) pun în funcțiune cele două soluții.

Cat despre componentele principale ale proiectului, se regăsesc API-ul de management, integrat în containerul „api\_management” și componenta de frontend, integrată în containerul denumit „frontend”.

Pentru a mă asigura că produsul final îndeplinește cerințele inițiale, am creat trei API de tip CRUD, ce fac parte din containerele cu numele: „gw-api1”, „gw-api2”, „gw-api3”.

### 3.2.5 Resursele din Kubernetes

Suplimentar, față de figura (Figura 3.6) avem un nivel de abstractizare, introdus de Kubernetes, reprezentat de podurile create. Un pod reprezintă un grup de unu sau mai multe containere, ce împart același network și aceleași resurse de stocare.

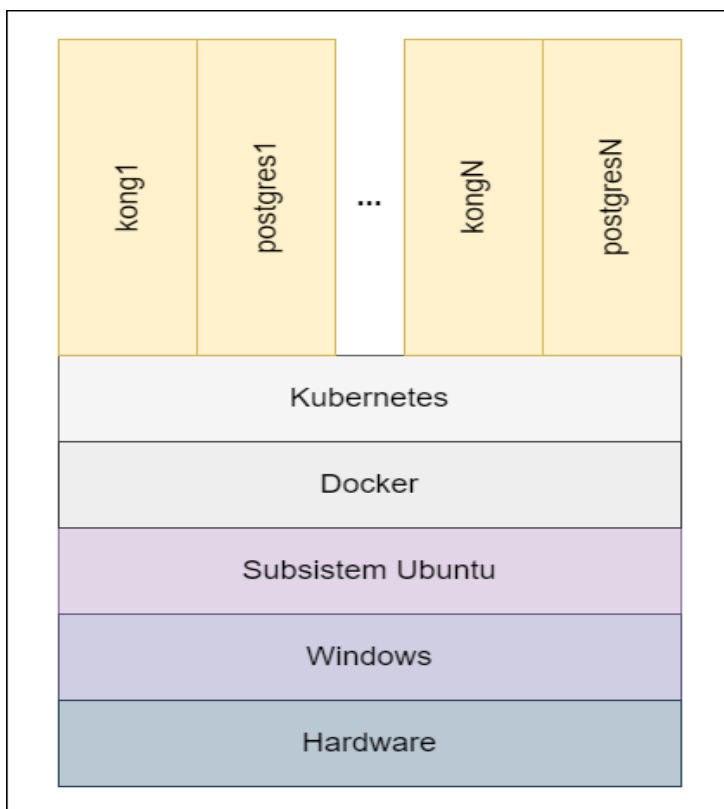


Figura 3.6 Colecția de pod-uri

În cadrul API Gateway-ului de tip Kong, funcționarea sa impune folosirea a două poduri: unul pentru aplicația în sine „kong” și al doi-lea pentru baza de date „postgres”.

## 3.3 Detalierea scripturilor și a fișierelor de configurare

Dockerfile reprezintă un fișier text ce conține comenzile necesare formării unei imagini.

Pentru fiecare componentă dezvoltată a aplicației ce rulează în Docker, există câte un fișier de tip Dockerfile, câte un script executabil, în cadrul căruia se construiește imaginea pe baza Dockerfile-ului respectiv și se lansează în execuție containerul.

### 3.3.1 Dockerfile-ul API-ului de management

În ceea ce privește API-ul de management, acesta este alcătuit dintr-un Dockerfile, două scripturi executabile: „run.sh”, care pornește aplicația și „run\_tyk.sh” care pune în funcțiune API Gateway-ul

de tip Tyk, împreună cu un fișier de configurare „tyk.standalone.conf”, pe langa două fișiere de tip YAML: „conf.yaml”, „kong.yaml”, și aplicația ce va rula în container „server.py”.

Dockerfile-ul are la bază varianta de Python 3.10.2, peste care se instalează cu ajutorul comenzii „RUN pip install numele-modulului” următoarele biblioteci: „flask”, care îmi permite să dezvolt aplicații Web de tip REST într-un mod facil, având ca principale caracteristici template engine și URL routing ( posibilitatea de a defini rute); „pymongo”, ce reprezintă o colecție de unelte necesare astfel încât să pot utiliza baza de date Mongo DB din Python; „Flask-PyMongo”, care unifică cele două biblioteci „flask” și „pymongo”; și modulul „requests” cu ajutorul căruia pot trimite HTTP request-uri din Python.

Prin intermediul comenzii: „RUN apt-get update” reîmprospătez pachetele Ubuntu din container, astfel, rulând „RUN apt-get install docker.io -y”, instalez unealta Docker cu scopul de a rula comenzi Docker în cadrul containerului, rezultând instanțe noi de Tyk.

De asemenea, stabilesc care va fi directorul implicit utilizat de container, prin intermediul comenzii „WORKDIR nume-director”, urmat de copierea în interiorul acestuia a conținutului întregului folder în care se află Dockerfile-ul în cauză.

Totodată, cu ajutorul comenzii curl, descarc executabilul Kubectl, în cadrul căreia folosesc o altă comandă curl pentru a obține ultima versiune stabilă a acestuia, iar „RUN chmod +x kubectl” îi atribuie fișierului Kubectl dreptul de a fi rulat. Ținând cont de caracteristicile unui Dockerfile, am folosit directiva „ENV” pentru a seta variabila de mediu „KUBECONFIG”, cu valoarea ce reprezintă calea către fișierul de configurare al Kubectl-ului folosind caracteristicile cluster-ului de Minikube.

Ultima linie a Dockerfile-ului conține directiva ENTRYPOINT, ce are două argumente: primul fiind interpretorul de Python „python3”, iar al doilea este reprezentat de fișierul lansat în rulare.

### 3.3.2 Scriptul executabil al API-ului de management

Prima acțiune din scriptul executabil „run.sh” este de a construi imaginea „api\_management” pe baza Dockerfile-ului descris anterior, urmată de ștergerea containerului cu același nume, pentru curățarea mediului, în cazul în care acesta exista în momentul lansării în execuție a scriptului. În caz contrar, pentru suprimarea erorii, folosesc comanda „true” pentru a putea trece la pasul următor în execuția scriptului. În mod asemănător, procedez și în momentul creării unei rețele Docker, denumită „gw”.

Folosesc comanda „docker run” pentru a crea containerul, pe baza imaginii construite. Dintre parametrii acestei comenzi amintesc: „-d” ce permite containerului să ruleze în background, „--net gw” prin care conectez containerul la rețeaua „gw”, prin „--link mongodb:mongodb” mă asigur ca host-ul respectiv este rezolvat la containerul dorit, „-p 5008:5000” realizează o mapare între portul extern 5008 și cel din interiorul containerului 5000, „--add-host kubernetes:192.168.65.2” translatează hostul la adresa IP respectivă. Adresă a mașinii pe care rulează sistemul Docker și Kubernetes, obținută, rulând comanda ping din interiorul containerului către adresa host.docker.internal.

Volumele montate ne permit accesul la certificatele: „certificate-authority”, „client-certificate” și „client-key” în interiorul containerului, pentru a putea fi folosite de Kubectl, pentru a se conecta la cluster-ul de Kubernetes.

### 3.3.3 Fișierul de configurare a Kubectl-ului

Asupra fișierului standard de configurare a Kubectl-ului am modificat următoarele câmpuri: „certificate-authority”, „client-certificate” și „client-key”, astfel încât noua cale să poată fi utilizată din interiorul containerului. Totodată, în cadrul câmpului „server” am introdus host-ul „kubernetes” menționat mai sus, deoarece certificatele sunt generate folosind acest hostname, făcând utilizarea prin adrese IP imposibilă; portul fiind atribuit dinamic, este necesară o modificare a acestuia la fiecare pornire a cluster-ului de Minikube. Toate aceste modificări sunt salvate în fișierul „conf.yaml”.

### 3.3.4 Scriptul executabil al unei instanțe de Tyk

Considerând fișierul „run\_tyk.sh”, asemănător cu comportamentul fișierului „run.sh” asociat API-ului de management, în primul rând se va realiza o verificare pentru unicitatea numelui containerului, astfel încât va fi șters în cazul în care nu se respectă această convenție. Prin comanda „docker run” creez containerul, pe baza imaginii „docker.tyk.io/tyk-gateway/tyk-gateway:latest”. Dintre parametrii acestei comenzi, se remarcă: „-p {{port1}}:8080” ce realizează o mapare între portul extern, ce va avea valoarea variabilei „port1”, preluată în cadrul execuției fișierului „server.py” (rezultată în urma adunării valorii prestabilite pentru port și a contorului ce ține evidența numărului de instanțe create de-a lungul proiectului, incrementat și stocat în baza de date Mongo DB, cu limitarea a 1000 de porturi posibile) și cel din interiorul containerului 8080, „--name gw-tyk{{port1}}” ce va reprezenta numele containerului, unicitatea acestuia fiind asigurată de introducerea valorii portului ca terminație a numelui.

Volumul

„/mnt/c/Users/andon/Desktop/licenta/api\_management/tyk.standalone.conf:/opt/tyk-gateway/tyk.conf” asigură accesul la fișierul de configurare „tyk.standalone.config” în cadrul containerului, iar volumul „/mnt/c/Users/andon/Desktop/licenta/api\_management/apps{{port1}}:/opt/tyk-gateway/apps” asigură o persistență a datelor în ceea ce privește resursele stocate în folderele denumite astfel: „apps{{port1}}” (pentru a menține unicitatea numelor lor).

### 3.3.5 Fișierul de deployment al unei instanțe Kong

Având în vedere fișierul „kong.yaml”, pornind de la o resursă găsită în cadrul unei documentații[39], am făcut modificările necesare aplicației mele. Astfel, am folosit resursele „Persistent Volume” și „Persistent Volume Claim” prin care am descris cerințele de stocare necesare Kong-ului, path-ul pe care îl voi avea în container și permisiunile volumului. Considerând utilitatea resurselor detaliate în subcapitolul precedent, fișierul mai are în componența sa și două deployment-uri, două servicii și un job. Scopul acestuia este de a lansa în execuție API Gateway-ul corespunzător.

Unicitatea fiecărei resurse din cadrul fișierului „kong.yaml” și a porturilor expuse în exterior, este asigurată printr-o modalitate asemănătoare celei descrise în cadrul respectării convențiilor în ceea ce privește API Gateway-urile de tip Tyk.

### 3.3.6 Fișierele frontend-ului

O altă componentă a produsului final este reprezentată de frontend, acesta fiind alcătuit dintr-un fișier de tipul Dockerfile, un script executabil „run.sh” și aplicația care va rula în container, ce va avea folderul „frontend” ca suport, unde se află fișierele de tipul HTML și py necesare. Fișierul de tip Dockerfile se aseamănă cu Dockerfile-ul API-ului de management, cu mențiunea că în cadrul instalării bibliotecilor, se folosește o singură comandă „RUN pip install -r frontend/requirements.txt”, fiecare modul regăsindu-se în fișierul requirements.txt din directorul „frontend”, generat automat cu ajutorul comenzii „pip3 freeze > requirements.txt”. Printre aceste biblioteci se regăsesc: Jinja2, redis și Flask-Session, ce reprezintă o extensie a Flask-ului, în cadrul căruia configurăm sesiunea astfel încât să fie de tipul Redis, folosind o instanță Redis integrată într-unul dintre containere.

### 3.3.7 Fișierele API-urilor de test

În cazul API-urilor de test, acestea au fost implementate asemănător. Așadar, fiecare are în componența lui câte un fișier de tipul Dockerfile, câte un script executabil și aplicația corespunzătoare „server.py” ce va rula în container.

Suplimentar, pentru acestea, în cadrul fișierului de tip Dockerfile, prin comanda „RUN pip install flasgger” am instalat o extensie a microframework-ului Flask, prin intermediul căreia am realizat o documentație pentru fiecare API de test în parte.

## 3.4 Prezentarea generală a componentelor software dezvoltate

### 3.4.1 API-ul de management

Backend-ul produsului final este reprezentat de API-ul de management. În cadrul acestuia, am definit rute ce manipulează din perspectivă CRUD următoarele resurse: utilizatori, conexiuni, servicii și rute. Astfel, dintre cele mai importante, amintesc rutele care au la bază operații prin care creez, editez, afișez sau șterg utilizatori și conexiuni, informațiile necesare fiind stocate în baza de date MongoDB.

Codul sursă al acestuia fiind disponibil în Anexa 1.

Așadar, pentru afișarea tuturor utilizatorilor există ruta „/users” care reprezintă metoda GET, ce se bazează pe funcția „get\_users()”, unde accesez colecția „users” din baza de date „users”, în care, cu ajutorul metodei „db.collection.find()”, găsesc fiecare document din aceasta, pe care îl adaug într-o listă „output”, inițial goală, listă care va servi drept răspuns al requestului.

Pentru Kong, în cazul operațiilor care implică servicii și rute, este necesară apelarea API-ului de administrare al Kong-ului. Un exemplu în acest sens, ar fi ruta „/connections/<connection\_id>/services”, care reprezintă metoda POST, ce are la bază funcția „create\_service(connection\_id)”, scopul ei fiind crearea unui serviciu pentru o anumită conexiune.

Astfel, accesez colecția „connections” a bazei de date MongoDB, în care fac o căutare după ID-ul conexiunii respective, pentru a obține câmpul „admin\_api\_url” al obiectului, care ulterior urmează a fi integrat în URL-ul pe baza căruia apelez API-ul de administrare al Kong-ului.

Se fac verificări la nivelul câmpurilor necesare creării unui serviciu, pentru a valida faptul că acestea nu sunt nule, astfel, dacă utilizatorul omite completarea unuia, de exemplu nu introduce niciun

port și execută acțiunea de creare a serviciului, API-ul va întoarce răspunsul, sub forma unui JSON de tipul:

```
{
    "response": "fail",
    "message": "Please provide a port!"
}.
```

În final, dacă fiecare câmp necesar a fost completat, se formează un dicționar de forma:

```
{
    "name": name,
    "path": path,
    "port": port,
    "host": host
}.
```

Unde name, path, port și host sunt valorile introduse de utilizator pentru a completa dicționarul, care va servi drept body metodei POST, cu ruta „<connection\_admin\_api\_url>/services/”.

Prin urmare, Kong-ul realizează o evaluare suplimentară a câmpurilor, iar dacă acestea se încadrează în normele specifice, serviciul va fi creat cu succes, returnând ca și răspuns un JSON ce conține mulțimea de câmpuri ce îl caracterizează. În caz contrar, răspunsul va indica greșeala prin câmpurile:

```
{
    "fields": {
        "path": "should start with: /"
    },
    "message": "schema violation (path: should start with: /)",
    "name": "schema violation"
}.
```

O limitare a API Gateway-ului Tyk este reprezentată de folosirea fișierelor în detrimentul unei baze de date. În cazul în care se dorește crearea unei resurse, având valoarea câmpului api\_id identică cu cea a unei alte resurse deja existente, se realizează suprascrierea acelei resurse. De aceea, în cazul metodei POST, am implementat o verificare suplimentară, prin care asigur faptul că se poate crea o resursă, doar dacă api\_id-ul este unic.

În ceea ce privește Tyk-ul, pentru a edita resursa de tipul apis, am definit ruta „/connections/<connection\_id>/tyk/apis/<tyk\_api\_id>” ce reprezintă metoda PUT, pe baza funcției „update\_tyk\_api(connection\_id, tyk\_api\_id)”.

Astfel, asemănător operației descrise mai sus, în cadrul Kong-ului, obțin câmpul „admin\_api\_url” al conexiunii. La nivelul câmpurilor ce trebuie completate, se realizează o serie de evaluări. Așadar, fiecare câmp necesar creării unui „api” va trebui completat, în caz contrar, răspunsul API-ului va fi de forma:

```
{
    "response": "fail",
    "message": "Please provide a name!"
}
```

Pentru ca request-ul să fie valid, câmpul api\_id din body va trebui să aibă aceeași valoare cu cea prezentă în ruta, în caz contrar, API-ul va trimite un răspuns de forma:

```
{
    "message": "Please don't modify the api_id!",
    "response": "fail"
}
```

De asemenea, este imperativ ca subcâmpul „listen\_path” al câmpului „proxy” să fie unic. În acest sens, se face o verificare la nivelul tuturor resurselor de tip „apis” ale acestei conexiuni. În cazul nerespectării acestei conventii, se afișează răspunsul:

```
{
  "message": "Please provide another listen_path!",
  "response": "fail"
}
```

În cazul în care, valorile câmpurilor respectă constrângerile enumerate mai sus, se crează un dicționar, care va fi trimis ca și body pentru metoda PUT, de forma:

```
{
  "name": name,
  "api_id": api_id,
  "proxy": {
    "listen_path": listen_path,
    "target_url": target_url,
    ...
  }
  ...
}
```

Unde, name, api\_id, listen\_path si target\_url sunt valorile introduse de utilizator.

Se menționează faptul că dicționarul trimis ca body, pe lângă câmpurile completate de utilizator, mai are în componența sa alte câmpuri necesare pentru funcționarea Tyk-ului.

Se apelează metoda PUT a API-ului de administrare al Tyk-ului, având ruta „/<connection\_admin\_api\_url>/tyk/apis/<tyk\_api\_id>”. În urma editării cu succes a resursei, se primește un răspuns de forma:

```
{
  "content": {
    "action": "modified",
    "key": "pokemons2",
    "status": "ok"
  },
  "response": 200
}
```

Din cauza limitării Tyk-ului menționată mai sus, este necesar ca după orice modificare executată în API-ul de tyk să fie apelat endpoint-ul de relod, pentru a face modificarea vizibilă în cadrul sistemului.

Pentru a descrie o metodă de tip DELETE din cadrul API-ului de management, detaliez ștergerea unei dependențe dintre o resursă de tipul conexiune și o resursă de tipul rută.

Astfel, operația se efectuează accesând ruta „/connection\_routes/<id>”, utilizând funcția „delete\_route\_dependency(id)”, în care, prin intermediul metodei „db.collection.deleteOne({'\_id': ObjectId(id)})” șterg documentul respectiv din colecție.

Răspunsul API-ului va fi de forma:

```
{
  "response" : "succes",
  "message" : "connection_route dependency deleted successfully"
}
```

### 3.4.2 Frontend-ul

O altă componentă esențială a produsului final este reprezentată de frontend. Pentru dezvoltarea acestuia, am folosit biblioteca Bootstrap [40], microframework-ul Flask și template engine-ul Jinja.

Bootstrap-ul reprezintă o bibliotecă pentru dezvoltarea frontend-ului, care facilitează crearea aplicațiilor Web. Caracteristicile acestei biblioteci sunt evidențiate la nivel vizual.

Fiecare pagină Web are în componența sa un fișier de tip HTML și un fișier de tip py. Astfel, dintre cele mai importante, se regăsește pagina de index, accesată prin URL-ul „<http://localhost:5004/>” care afișează numele aplicației și prin intermediul celor două butoane: „Login” și „Register” oferă utilizatorului posibilitatea de a se loga sau a se înregistra. Mesajele de eroare sunt afișate, redirecționând utilizatorul la această pagină.

În cadrul paginii „login”, accesată prin URL-ul „<http://localhost:5004/login>” obțin userul din sesiunea din Flask. De asemenea, o altă variabilă din cadrul sesiunii este variabila de tip boolean „logged\_in”, căreia i se atribuie valoarea False în cazul în care userul nu este autentificat. Valoarea acesteia este transmisă în fișierul HTML al acestei pagini, prin intermediul template engine-ului Jinja, astfel încât, pe baza acesteia se iau anumite decizii.

Așadar, în cazul în care utilizatorul nu este autentificat, acesta va fi întâmpinat de un formular de autentificare, folosind o rută auxiliară pentru a apela API-ul de management cu scopul de a verifica dacă utilizatorul este înregistrat.

În cazul în care acesta nu este înregistrat, va fi redirecționat la pagina de index cu mesajul de eroare „This user is not registered”. Altfel, populez sesiunea cu detaliile acestuia și îl redirecționez către pagina de Home.

După autentificare, în marginea dreaptă de sus a fiecărei pagini, utilizatorul este întâmpinat cu un mesaj de forma „Hello, user”, alături de un buton de delogare, iar în partea stângă există un Meniu ce conține principalele pagini ale aplicației.

Pentru vizualizarea resurselor, implementarea tabelelor, pornește de la un tutorial Bootstrap. [41]

Accesând ruta „`services-from/<connection_name>`” sunt afișate serviciile unei conexiuni. Astfel, se realizează un apel către API-ul de management, în urma căruia obțin ID-ul conexiunii pe baza numelui său. În continuare, se face o verificare bazată pe tipul de conexiune. Aceasta poate fi de tipul Kong sau Tyk.

În cazul în care conexiunea este de tipul Kong, fac un request către API-ul de management prin care obțin un dicționar ce conține toate serviciile conexiunii dorite. Ulterior, pentru a genera structura paginii Web, prin intermediul template engine-ului Jinja, transmit în fișierul HTML (fișier identic denumit cu fișierul de tipul py, în cadrul căruia a fost definită ruta respectivă) valorile variabilelor din sesiunea de Flask, numele conexiunii respective și dicționarul întors în urma ultimului request făcut către API-ul de management.

Dacă tipul conexiunii este Tyk, se realizează același set de pași ca în cazul Kong-ului, cu mențiunea că, se creează un dicționar nou pe baza celui întors de API-ul de management, în care se modifică numele câmpurilor astfel încât să corespundă cu ale Kong-ului, păstrând o abordare asemănătoare; acesta fiind în cele din urmă transmis fișierului HTML.

De asemenea, pentru operațiile de creare, editare și ștergere a serviciilor am folosit rute auxiliare, pentru a îi asocia fiecărei operații câte o funcție. Apelând ruta „`/create-service-for-connection/<connection_name>`” se apelează funcția „`do_create_service(connection_name)`”, în care, prin aceeași metodă descrisă mai sus obțin ID-ul conexiunii în cauză, urmând aceeași verificare pe baza tipului de conexiune.

Astfel, dacă tipul conexiunii este Kong, se creează un dicționar populat cu valorile introduse de utilizator, care va fi trimis ca body metodei POST către URL-ul API-ului de management „`http://api_management:5000/connections/<connection_id>/services`”. În cazul în care, serviciul a fost creat cu succes, pasul următor este reprezentat de crearea dependenței dintre



serviciul respectiv și conexiunea curentă, apelând metoda POST către „[http://api\\_management:5000/connection\\_services](http://api_management:5000/connection_services)”, având body-ul populat de câmpurile „service\_id” și „connection\_id”, în final, pagina actuală va fi reîncărcată. În caz contrar, dacă nu sunt respectate constrângerile impuse atât în API-ul de management cât și de Kong, utilizatorul va fi redirectionat către pagina de index, afișând mesajul de eroare.

Pentru tipul Tyk al conexiunii, se realizează același set de pași, cu mențiunea că dicționarul care servește drept body operației de creare a serviciului este populat cu câmpuri suplimentare, necesare funcționării Tyk-ului, experiența utilizatorului fiind aceeași. Din același motiv, am stabilit o convenție privind câmpurile „name” și „api\_id”, astfel încât ambele să fie populate cu valoarea introdusă de utilizator în cadrul câmpului „name”.

Pentru a crea serviciul, este apelată metoda POST către „[http://api\\_management:5000/connections/<connection\\_id>/tyk/apis](http://api_management:5000/connections/<connection_id>/tyk/apis)”, iar în cazul creării dependenței dintre serviciu și conexiune, se apelează URL-ul „[http://api\\_management:5000/connection\\_tyk\\_api](http://api_management:5000/connection_tyk_api)”.

În ceea ce privește operația de editare, accesând ruta „/update-service-for-connection/<connection\_name>”, se apelează funcția „do\_update\_service(connection\_name)”. Logica acesteia, este asemănătoare cu cea a funcției detaliate de mai sus „do\_create\_service(connection\_name)”, pentru Kong, se apelează metoda PUT a API-ului de management către URL-ul „[http://api\\_management:5000/connections/<connection\\_id>/services/<service\\_id>](http://api_management:5000/connections/<connection_id>/services/<service_id>)”, iar pentru Tyk:

„[http://api\\_management:5000/connections/<connection\\_id>/tyk/apis/<service\\_id>](http://api_management:5000/connections/<connection_id>/tyk/apis/<service_id>)”. Din cauza faptului că în Kong există două entități „services” și „routes” diferite, iar în Tyk există una singură ce le înglobează pe cele două, am stabilit o convenție prin care, la editarea unui serviciu, câmpurile care vor putea fi modificate sunt „port” și „host”, pentru a asigura utilizatorului o experiență unică.

Operația de ștergere se realizează accesând ruta „/delete-service-for-connection/<connection\_name>”, prin apelarea funcției „delete\_service(connection\_name)”, în care se obțin ID-ul conexiunii curente și al serviciului ce urmează a fi șters. Pe baza celui din urmă, obținând și ID-ul dependenței dintre cele două. În final, apelându-se metoda DELETE a API-ului de management, în cazul Kong-ului, către „[http://api\\_management:5000/connection\\_services/<service\\_id>](http://api_management:5000/connection_services/<service_id>)”, iar în cazul Tyk-ului către:

„[http://api\\_management:5000/connections/<connection\\_id>/tyk/apis/<tyk\\_api\\_id>](http://api_management:5000/connections/<connection_id>/tyk/apis/<tyk_api_id>)”.

Dependența dintre serviciul în cauză și conexiunea curentă va fi ștearsă după ce serviciul a fost șters cu succes, apelând, după caz metoda DELETE a API-ului de management, către „[http://api\\_management:5000/connection\\_services/<dependency\\_id>](http://api_management:5000/connection_services/<dependency_id>)”

sau către „[http://api\\_management:5000/connection\\_tyk\\_api/<dependency\\_id>](http://api_management:5000/connection_tyk_api/<dependency_id>)”.

O altă resursă esențială a proiectului este definită de conceptul „rută”. Pentru API Gateway-ul Kong, operațiile de tip CRUD asupra acestei resurse au o abordare identică cu implementarea pentru resursa „serviciu”.

În ceea ce privește Tyk, singurele operații posibile asupra rutelor sunt: afișare și editare, motivul fiind existența unei singure entități în cadrul Tyk-ului ce înglobează conceptele de rută și serviciu.

Așadar, o rută este creată/ștearsă atunci când serviciul respectivei rute este creat/șters, acest lucru reprezentând o limitare, deoarece, pentru Tyk, un serviciu poate avea atribuit o singură rută. Prin

urmare, limitări apar și în cazul operației de editare, astfel încât, utilizatorul poate modifica numai câmpul care face referire explicit la conceptul de rută, și anume, câmpul „paths”. În urma acestor limitări, am stabilit o convenție referitoare la numele rutelor, și anume, acesta să aibă valoarea câmpului „paths” introdusă de utilizator.

### 3.4.3 API-urile de test

Primul API simplu creat are ca temă cărțile. Numele containerului din care rulează este „gw-ap1”, realizându-se o mapare între portul extern 5001 și cel din interiorul containerului, 5000.

Tema celui de-al doilea API de test este reprezentată de mașini, numele containerului specific acestuia fiind „gw-ap2”, realizându-se maparea între portul extern 5002 și cel din interiorul containerului, 5000.

Pentru cel de-al treilea API, am avut în vedere tema „pokemoni”, numele containerului fiind „gw-api3”, realizându-se maparea între portul extern 5003 și 5000, din interiorul containerului.

Considerând că cele trei API-uri de test au fost implementate asemănător, voi detalia metodele definite numai pentru unul dintre acestea. În cazul acestora, nu am o modalitate de a persista datele, informațiile fiind ținute în memorie sub forma unei liste cu elemente de tip JSON.

Așadar, având în vedere API-ul cu tematica „pokemoni”, pentru a afișa toate resursele, se accesează ruta „/pokemons”, folosind metoda GET. Astfel se apelează funcția „get\_poke()”, care va returna din memorie lista cu pokemoni. Pentru metoda POST, se accesează aceeași rută, apelând funcția „create()” care stochează într-o variabilă body-ul metodei, acesta fiind de forma:

```
{
    "name" : "aron",
    "poke_id" : "1",
    "region" : "Johto",
    "trainer" : "Paul",
    "type" : "steel"
}
```

}. Ulterior, se parcurge lista ce stochează toate resursele, pentru a realiza o verificare la nivelul câmpului „poke\_id”, pentru a asigura unicitatea acestuia. În cazul în care se încearcă crearea unei noi resurse, având un ID deja existent, se returnează răspunsul:

```
{
    "error": "Please provide another poke_id"
}
```

}. În caz contrar, resursa va fi creată cu succes, având ca răspuns, body-ul trimis, aceasta fiind adăugată în lista inițială. Pentru metoda PUT, se accesează ruta „/pokemons/<string:id>”, care apelează funcția „update(id)”, în care fac o căutare pe baza ID-ului în lista cu resurse. Dacă nu există nicio potrivire, se returnează răspunsul:

```
{
    "error": "Not found"
}
```

, iar dacă există, se vor modifica valorile câmpurilor existente în body, în final, returnându-se resursa editată. Pentru metoda DELETE, se accesează aceeași rută ca în cazul anterior, apelând funcția „delete(id)”, ce realizează aceeași căutare pe baza ID-ului, înlăturând resursa cu ID-ul respectiv în cazul în care aceasta există, în final returnând răspunsul:

```
{
    "result": True
}
```

## Capitolul 4 - Prezentarea generală a arhitecturii rețelei

În cadrul produsului, există mai multe nivele de rețea folosite, pe care le voi enumera: localhost, sau rețeaua locală a laptopului, rețeaua Docker și rețeaua Kubernetes. În continuare voi prezenta arhitectura rețelei folosite, împreună cu particularitățile acesteia.

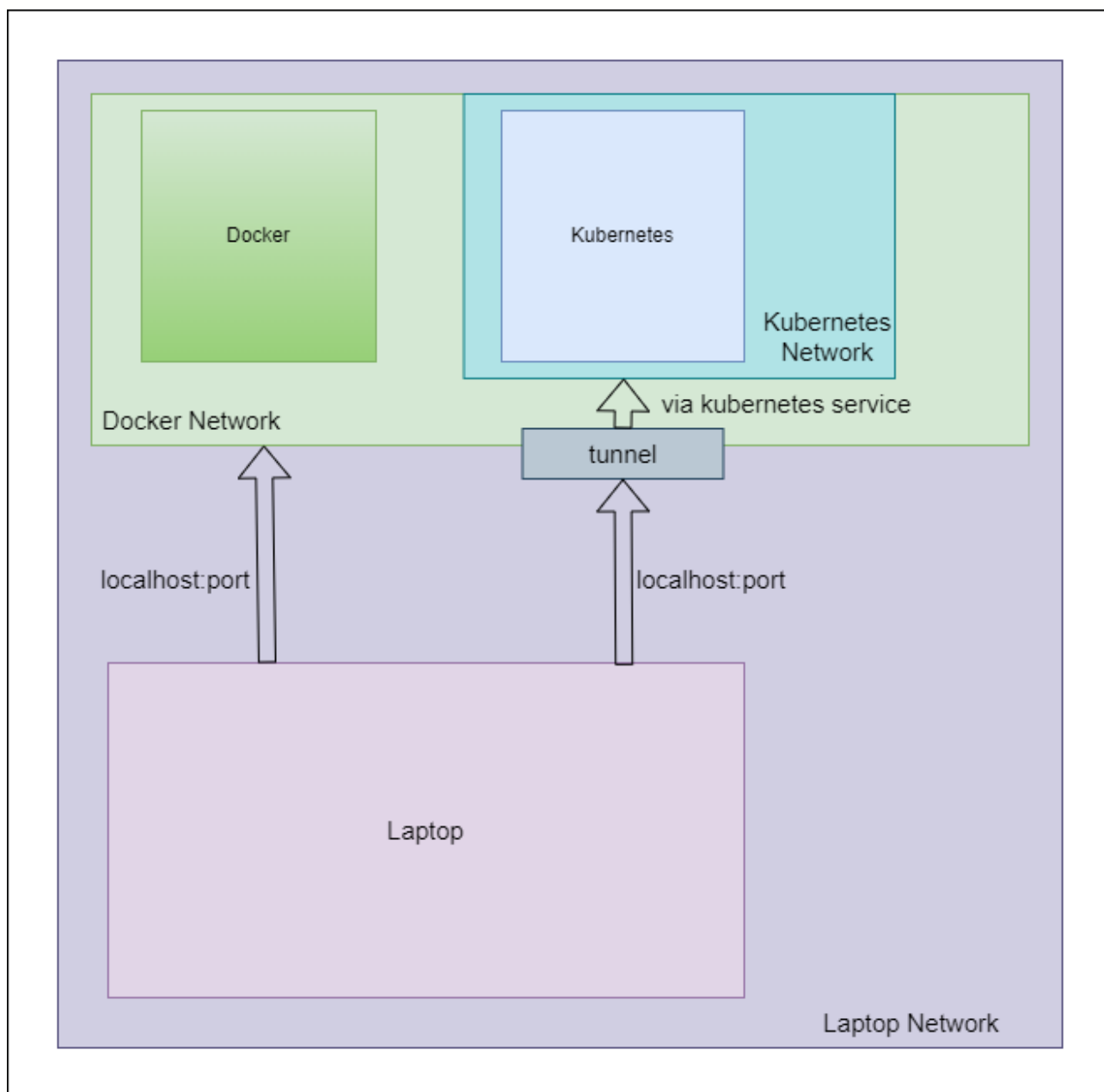


Figura 4.1 Arhitectura rețelei

În cadrul rețelei laptopului, există o rețea virtuală, Docker Network, separată de rețeaua locală, comunicarea dintre ele realizându-se printr-o punte virtuală. Astfel, când se efectuează un apel către o resursă ce aparține rețelei Docker, se va folosi adresa „localhost” urmată de portul expus al resursei. Cererea pleacă din rețeaua locală a laptopului, apoi prin intermediul puntii virtuale ajunge în rețeaua Docker, după care este translatată către containerul ce are portul expus respectiv, urmat

de o altă traducere de la acest port la cel intern unde rulează aplicația în interiorul containerului. Răspunsul va fi primit, urmând calea inversă descrisă.

De asemenea, în interiorul rețelei Docker, există o altă rețea virtuală Kubernetes Network. Comunicarea dintre rețeaua locală și aceasta se realizează printr-un tunel, care se ocupă de traducerea cererilor primite către serviciile din Kubernetes. Rețeaua Kubernetes este un nivel de abstractizare construit peste rețeaua Docker.

La efectuarea unui apel către o resursă din rețeaua Kubernetes, se folosește adresa „localhost”, urmată de portul expus al resursei. Traseul cererii pornește din rețeaua locală a laptopului, ajungând în rețeaua Kubernetes prin intermediul tunelului, fiind tradusă către serviciul care are portul expus respectiv. În serviciu, se face o traducere către portul pod-ului, în final, se face ultima traducere către portul aplicației ce rulează în interiorul containerului din pod. Răspunsul va fi primit, urmând traseul invers.

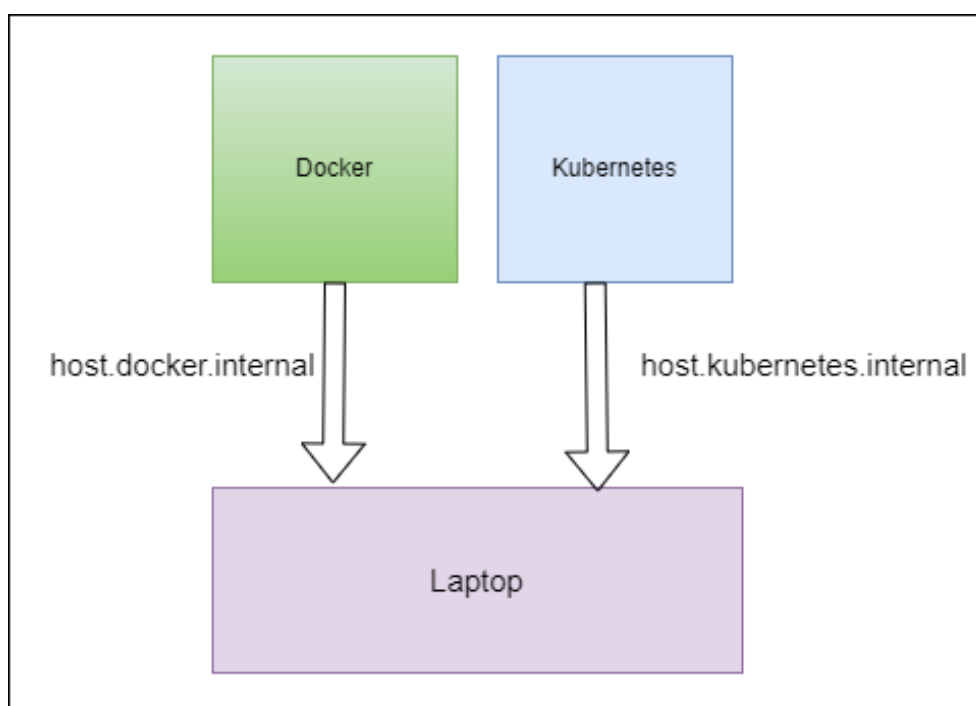


Figura 4.2 Comunicare Docker/Kubernetes către rețeaua locală

Conform figurii ( Figura 4.2), pentru a putea ajunge din Docker în rețeaua locală, se folosește adresa „host.docker.internal”, iar în cazul Kubernetes, se va folosi host-ul „host.kubernetes.internal”, acest aspect fiind important pentru comunicarea dintre aplicații, indiferent de rețeaua din care fac parte.

Atat „host.docker.internal” cât și „host.kubernetes.internal” reprezintă host-ul local al laptopului, din perspectiva rețelei Docker.

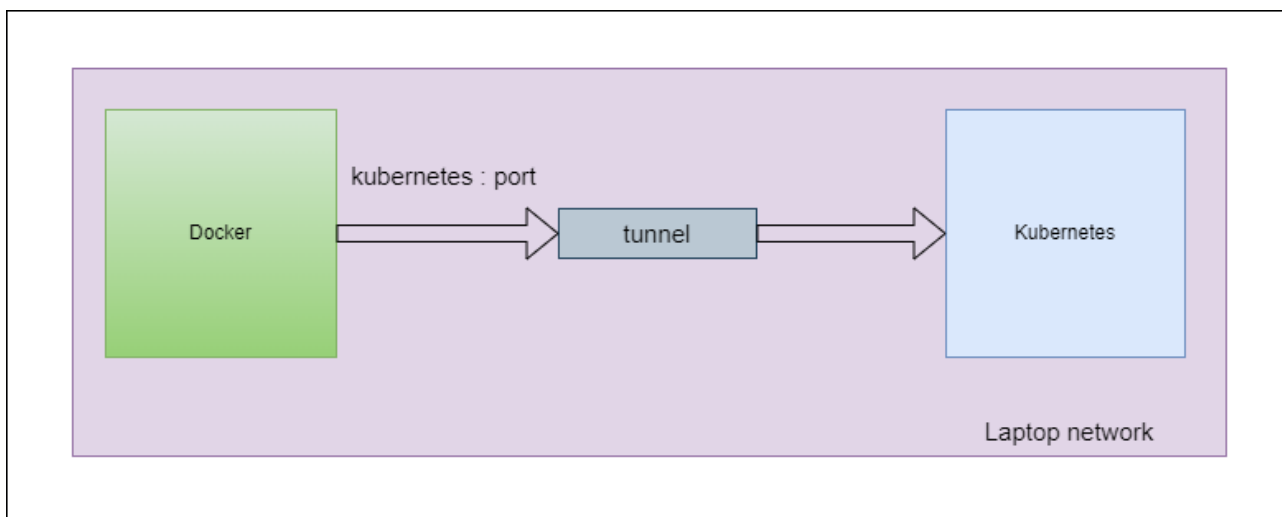


Figura 4.3 Comunicarea între Docker și Kubernetes

Comunicarea între cele două medii virtuale, se realizează prin intermediul tunelului. Voi folosi ca exemplu cazul API-ului de management, în cadrul căruia a fost adăugat hostul „kubernetes”, translatat la adresa IP „192.168.65.2”, ce reprezintă adresa host-ului local al laptopului din perspectiva rețelei Docker.



# Capitolul 5 - Experiența utilizatorului

## 5.1 Ghid de utilizare al aplicației

Pentru a folosi produsul final, utilizatorul va accesa link-ul „<http://localhost:5004/>”, ce reprezintă pagina de index în cadrul căreia este afișat numele aplicației, alături de două butoane „Login” și „Register”.

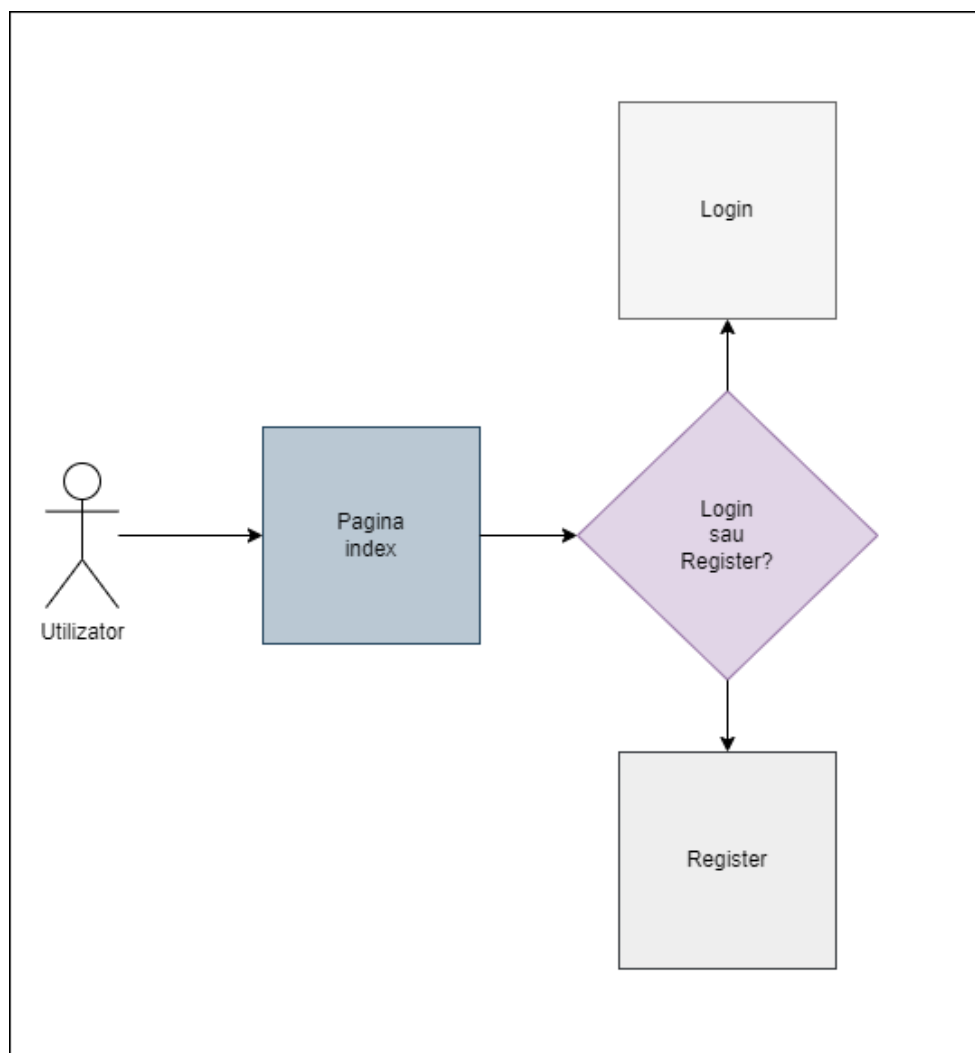


Figura 5.1 Primul contact cu aplicația

Dacă se dorește accesarea unui cont deja creat, se apăsă butonul de login, unde, după o redirectionare la „<http://localhost:5004/login>”, utilizatorul va fi întâmpinat de un formular de autentificare, în care va trebui să completeze câmpurile pentru numele de utilizator și parolă, la final se apasă butonul de submit, fapt care îl va redirectiona către pagina home, în cazul în care autentificarea se realizează cu succes, iar, în caz contrar, acesta va fi întors la pagina de index, unde va fi afișat un mesaj de atenționare, în funcție de tipul de eroare. Printre mesajele de atenționare, amintesc: „Wrong credentials!”, „Please provide both username and password!”, „This user is not registered.”.

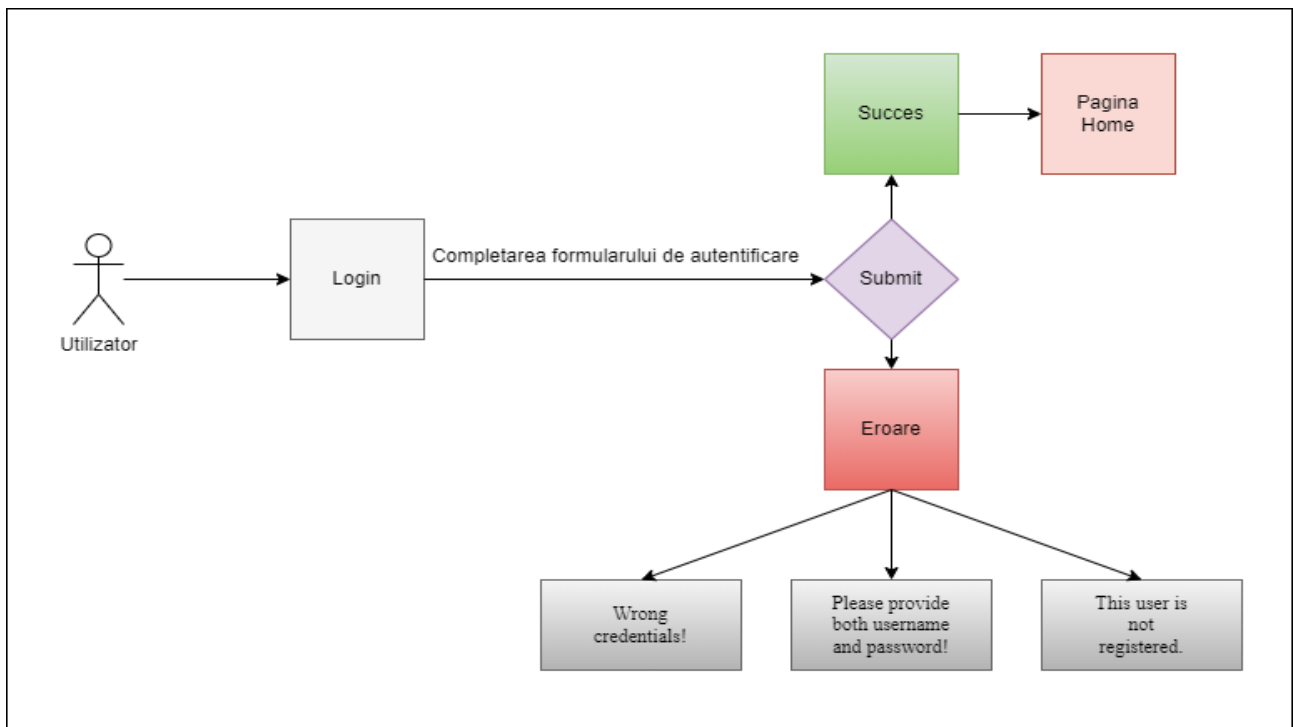


Figura 5.2 Fluxul de autentificare

Pentru a crea un nou utilizator, din pagina de index, se apasă butonul „Register”, ce realizează o redirectionare la [„http://localhost:5004/register”](http://localhost:5004/register), unde va trebui completat formularul de înregistrare, ce conține câmpurile: „Username”, „Password”, „Confirm password”. Dacă numele introdus nu se regăsește în baza de date ce stochează informațiile cu privire la utilizatori și cele două câmpuri ce fac referire la parolă sunt identice, se afișează mesajul „User created successfully. Please log in!”, apoi se apasă butonul de login și se reiau pașii descriși mai sus. În cazul unei nepotriviri între ultimele două câmpuri, se afișează mesajul de atenționare: „Password and password confirmation don't match”, iar dacă numele introdus se află deja în baza de date, se afișează: „Please provide another username”.

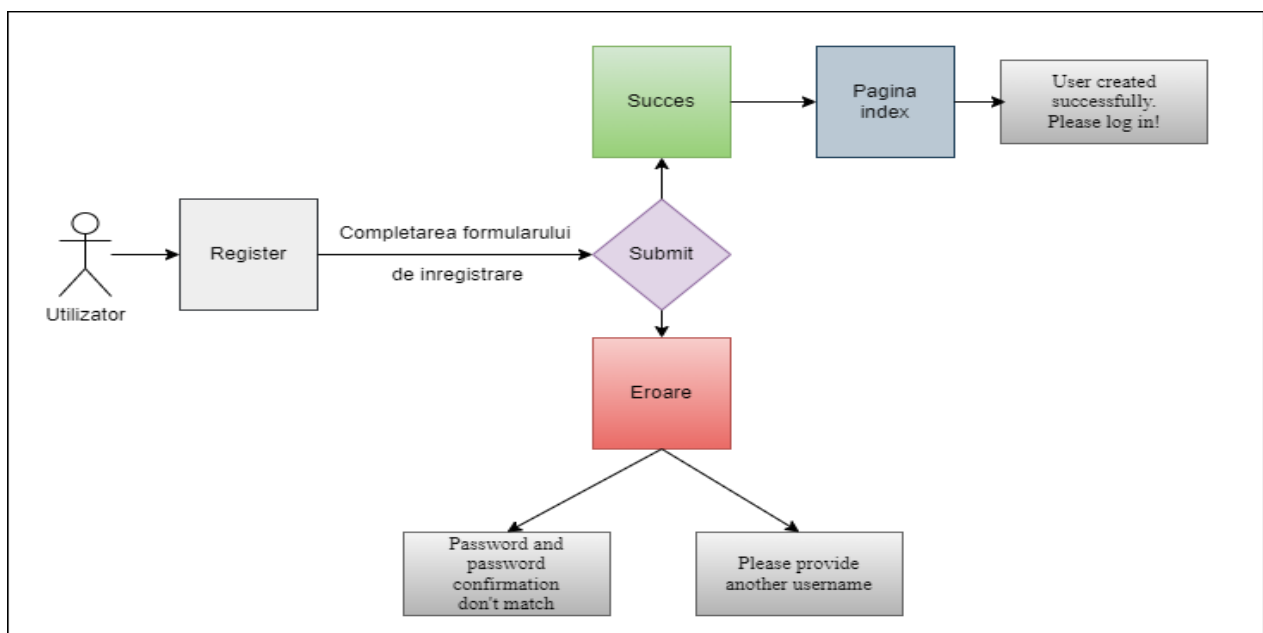


Figura 5.3 Fluxul de înregistrare



După autentificare, utilizatorul ajunge în pagina Home, având link-ul „<http://localhost:5004/home>”, unde, în partea dreaptă este afișat mesajul de întâmpinare, alături de butonul „Logout”, în timp ce în partea stângă a ecranului se află meniul ce conține principalele pagini Web ale aplicației, acestea fiind: „Services”, „Routes”, „Users”, „Connections”, „Deployments”; meniu ce poate fi accesat în cadrul fiecărei pagini din componența acestuia.

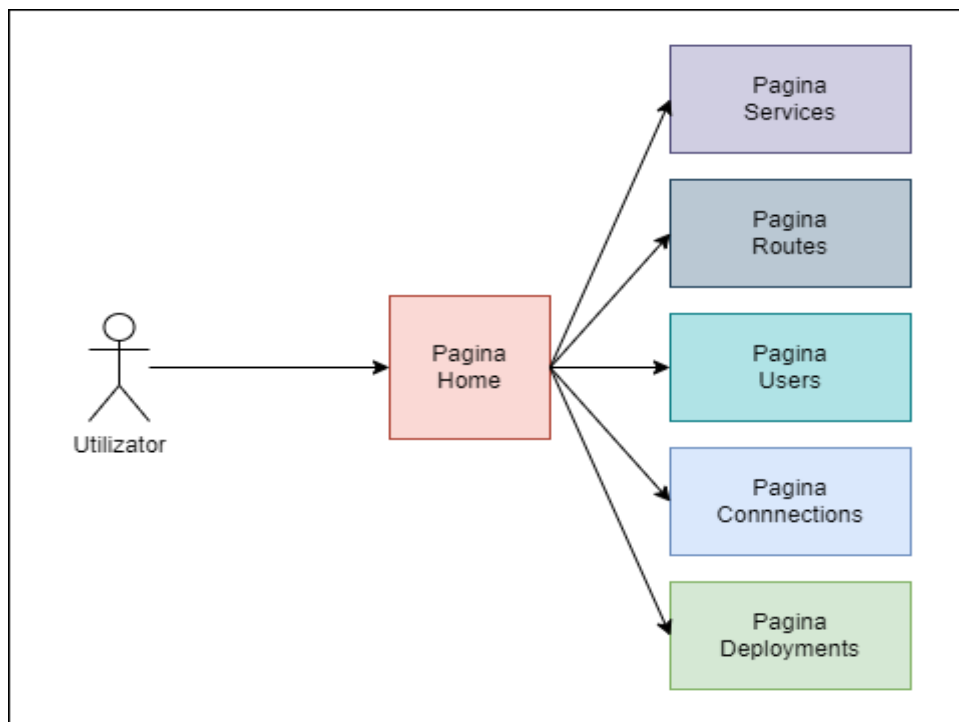


Figura 5.4 Pagina Home

În cazul în care, se alege „Services” din meniul, se realizează o redirecționare către pagina aflată la adresa „<http://localhost:5004/services-from-connection>”, în cadrul căreia se afișează mesajul de interogare: „For which connection would you like to display the services?”, dedesubtul căruia sunt listate numele conexiunilor utilizatorului respectiv, fiecare reprezentând un link către „[http://localhost:5004/services-from/<connection\\_name>](http://localhost:5004/services-from/<connection_name>)”, pagină unde sunt afișate toate serviciile conexiunii respective, sub forma unui tabel, ce prezintă câmpurile: „ID”, acesta fiind un identificator unic, generat, „Name”, care reprezintă numele serviciului, „Path”, ce semnifică rădăcina căii de la care poate fi accesat API-ul gestionat, „Port”, fiind portul pe care rulează API-ul pe care doresc să îl gestionez, în interiorul containerului, „Host”, ce poate fi completat cu adresa IP sau denumirea containerului respectiv și câmpul „Actions”, ce conține două butoane de tipul „Delete”, „Edit”, prin intermediul cărora pot șterge sau edita servicii. De asemenea, o altă acțiune posibilă este adăugarea unui nou serviciu, cu ajutorul butonului „Add new”, amplasat în partea dreaptă de sus a tabelului.







+ Add New					
ID	Name	Path	Port	Host	Actions
86209a1b-4911-4d71-bf73-ac6fecef55b0	api3-service	/	5000	172.19.0.7	 
b804f072-5da3-410a-8409-0a150554ce13	api2-service	/	5000	172.19.0.6	 
707599e8-7de6-4c7b-bf64-912de2aa56ac	api1-service	/	5000	172.19.0.5	 

Figura 5.5 Tabel de servicii din cadrul unei conexiuni de tip Kong

Acțiunea de creare a unui serviciu implică apariția unui formular ce conține câmpurile: „Name”, „Path” (în cazul Tyk-ului, are valoarea default „/”), „Port”, „Host”, ce se poate închide prin butonul „Close” plasat la baza formularului, sau prin apăsarea butonului „X” din dreapta, sus al acestuia, fără a se realiza acțiunea. După completarea formularului, pentru a crea serviciul, se apasă butonul „Add”, amplasat, de asemenea, la baza acestuia. Formularul dispune de un validator, ce verifică respectarea constrângerilor, astfel, în cazul nerespectării, utilizatorul va fi redirecționat la pagina index, unde se va afișa mesajul de eroare corespunzător; printre acestea, amintesc: „UNIQUE violation detected on '{name="api4-service"}'", „Please provide a path!", „Please make sure that the port is an integer!", „schema violation (path: should start with: /)”. Prin butonul „Back” al browser-ului, utilizatorul va putea reveni la formular, pentru a corecta câmpul în cauză. Referitor la crearea unui serviciu din cadrul unei conexiuni de tip Tyk, această acțiune implică și crearea rutei ce face referire la el.

De altfel, operația de editare a unui serviciu implică completarea unui formular, ce are câmpurile: „ID”, „Name”, „Path” (în cazul Tyk-ului, are valoarea default „/”), „Port”, „Host”. De menționat faptul că „ID”-ul este un câmp care nu poate fi modificat, suplimentar, pentru Tyk, nu se pot edita câmpurile „Name” și „Path”. Acțiunile de control ale formularului se manifestă asemănător cu cazul celor ale formularului creării unui serviciu, singura diferență fiind că, pentru a realiza operația, se apăsă butonul „Edit”. În cazul nerespectării constrângerilor asupra câmpurilor, modalitatea de a atenționa utilizatorul este aceeași ca în cazul creării unui serviciu, mesajele fiind identice.

Figura 5.6 Formular de adăugare a unui serviciu din cadrul unei conexiuni de tip Kong

Figura 5.7 Formular de editare a unui serviciu din cadrul unei conexiuni de tip Kong

În ceea ce privește operația de ștergere a unui serviciu, prin intermediul unei ferestre de tip modal, utilizatorul primește un mesaj de confirmare, de forma: „Do you want to delete the service with the following name”, urmat de valoarea câmpului „Name”, în urma căruia ia decizia de a continua acțiunea prin apăsarea butonului „Delete” sau de a renunța la aceasta prin modalitatea menționată mai sus. În cazul unei conexiuni de tip Kong, nu se poate șterge un serviciu dacă există

cel puțin o rută care face referire la el. În aceasta încercare, utilizatorul va fi întâmpinat de mesajul de eroare: „an existing 'routes' entity references this 'services' entity”. Pentru a șterge serviciul, se șterg mai întâi rutele care fac referire la el. Cât despre serviciile din cadrul unei conexiuni de tip Tyk, ștergerea unui serviciu implică ștergerea rutei cu care este corelat.

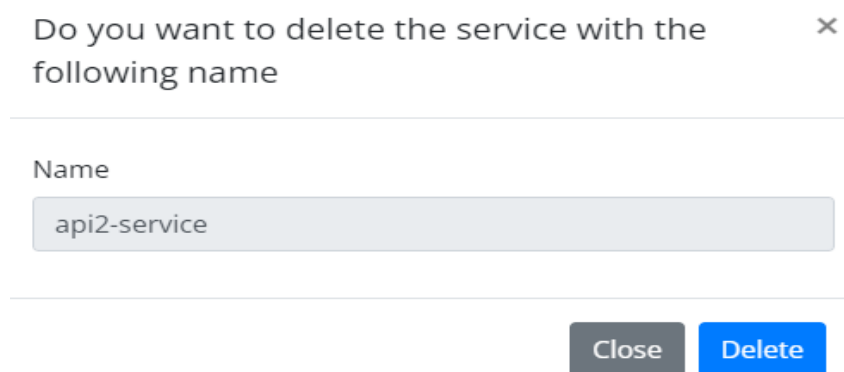


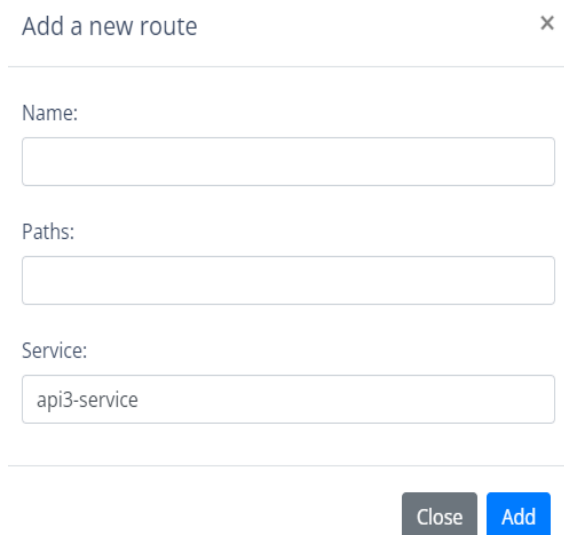
Figura 5.8 Fereastră de tip modal pentru ștergerea a unui serviciu din cadrul unei conexiuni de tip Kong

Opțiunea „Routes” din meniu va redirecționa utilizatorul la „<http://localhost:5004/routes-from-connection>”, unde este afișat mesajul de interogare: „For which connection would you like to display the routes?”, urmat de listarea numelor conexiunilor utilizatorului respectiv, fiecare dintre acestea fiind un link către pagina „[http://localhost:5004/routes-from/<connection\\_name>](http://localhost:5004/routes-from/<connection_name>)”, în cadrul căreia sunt afișate toate rutele conexiunii alese (în cazul în care conexiunea este de tipul Kong și pentru aceasta nu a fost creat niciun serviciu, se afișează mesajul „Please create a service first!”), dispuse sub forma unui tabel, ce conține câmpurile: „ID”, acesta fiind un identificator unic, generat (în cazul Tyk-ului, ia valoarea câmpului „Name”), „Name”, ce definește numele rutei, „Paths”, ce semnifică calea de acces către API-ul gestionat, „Service”, care reprezintă numele serviciului care este corelat cu ruta în cauză, „Actions”, care, pentru o conexiune de tip Kong, conține două butoane „Edit” și „Delete”, cu ajutorul cărora pot edita sau șterge rute, iar pentru o conexiune de tip Tyk, dispune numai de butonul „Edit”.

De asemenea, în cazul în care conexiunea este de tip Kong, există opțiunea de a crea o rută nouă, prin butonul „Add new”, accesând un formular, cu următoarele câmpuri: „Name”, „Paths”, „Service” (în cadrul căruia, prin intermediul unui dropdown, am posibilitatea de a alege valoarea dintr-o listă de servicii deja existente). Acțiunile de a efectua operația sau a renunța la aceasta, sunt identice cu cele din cazul creării unui serviciu. Mesajele de eroare pot fi de forma: „UNIQUE violation detected on '{name="api4-route"}'”, „schema violation (paths.1: should start with: /)”, în urma cărora se revine la formular prin butonul „Back” al browser-ului. În cazul unei conexiuni de tip Tyk, acțiunea de creare a unei rute nu există, rutele fiind create odată cu crearea serviciilor, lucru care reprezintă o limitare, astfel încât un serviciu din cadrul unei conexiuni de tip Tyk nu poate avea mai mult de o rută.

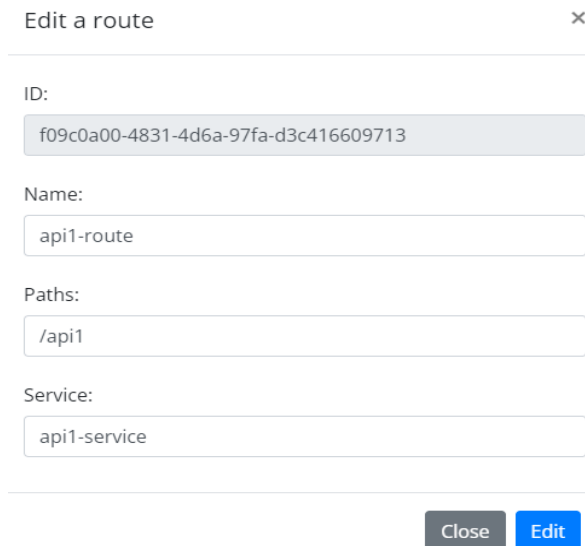
Totodată, pentru operația de editare a unei rute, se completează un formular ce conține următoarele câmpuri: „ID”, „Name”, „Paths”, „Service”. Se menționează faptul că „ID”-ul nu se poate modifica, iar în cazul unei conexiuni de tip Tyk, singurul câmp ce se poate edita este „Paths”. Acțiunile de control ale formularului sunt identice cu cele din cazul editării unui serviciu. Mesajele de eroare fiind aceleași în cazul conexiunii de tip Kong, însă la Tyk, deoarece se impune unicitatea valorii

câmpului „Paths” per conexiune, utilizatorul poate întâmpina mesajul de eroare: „Please provide another listen\_path!”.



A modal window titled "Add a new route" with a close button (X) in the top right corner. It contains three input fields: "Name:" (empty), "Paths:" (empty), and "Service:" (containing "api3-service"). At the bottom, there are two buttons: "Close" (grey) and "Add" (blue).

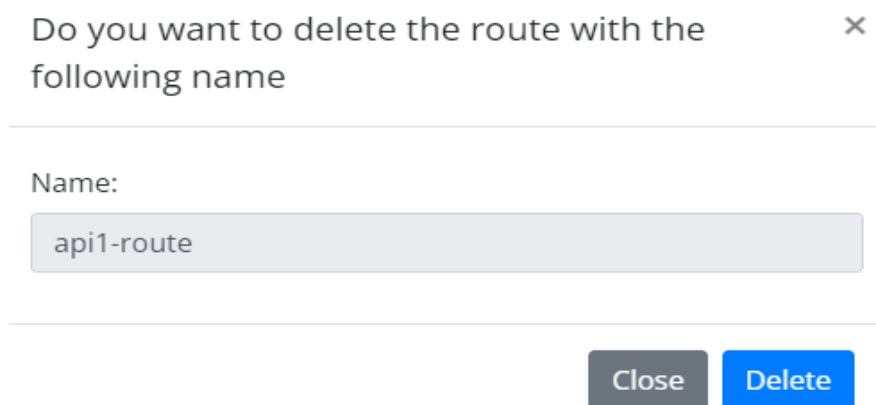
Figura 5.9 Formular de adăugare a unei rute din cadrul unei conexiuni de tip Kong



A modal window titled "Edit a route" with a close button (X) in the top right corner. It contains four input fields: "ID:" (containing "f09c0a00-4831-4d6a-97fa-d3c416609713"), "Name:" (containing "api1-route"), "Paths:" (containing "/api1"), and "Service:" (containing "api1-service"). At the bottom, there are two buttons: "Close" (grey) and "Edit" (blue).

Figura 5.10 Formular de editare a unei rute din cadrul unei conexiuni de tip Kong

Operația de ștergere a unei rute este posibilă numai în cadrul unei conexiuni de tip Kong (în cazul Tyk-ului, ruta va fi ștearsă odată cu ștergerea serviciului la care face referire). Astfel, prin intermediul unei ferestre de tip modal, se afișează un mesaj de confirmare, de forma: „Do you want to delete the route with the following name”, urmat de numele rutei ce urmează a fi ștearsă. Pentru a încheia operația, utilizatorul va apăsa butonul „Delete”, iar pentru a renunța la aceasta, va apăsa „Close” sau „X”.



A modal window titled "Do you want to delete the route with the following name" with a close button (X) in the top right corner. It contains one input field: "Name:" (containing "api1-route"). At the bottom, there are two buttons: "Close" (grey) and "Delete" (blue).

Figura 5.11 Fereastră de tip modal pentru ștergerea unei rute din cadrul unei conexiuni de tip Kong

Meniul aplicației conține și opțiunea „Users”, care, odată selectată va realiza o redirectionare către pagina „<http://localhost:5004/users>”, în care este afișat un tabel, ce conține informații referitoare la numele și ID-ul utilizatorilor.

O altă alegere a utilizatorului din meniul aplicației este reprezentată de „Connections”, fapt ce îl va redirectiona către „<http://localhost:5004/connections>”, unde îi este prezentat un tabel, ce conține informații despre conexiuni, având câmpurile: „ID”, acesta fiind un identificator unic, „Name”, ce definește numele conexiunii, „Type”, care poate fi de forma „kong” sau „tyk”, „Admin

API URL” ce reprezintă URL-ul API-ului de administrare a API Gateway-ului respectiv si „Actions” ce dispune de butoanele „Edit” și „Delete” care oferă posibilitatea de editare sau ștergere a conexiunilor. Prin intermediul butonului „Add new” pot fi create noi conexiuni.





+ Add New				
ID	Name	Type	Admin API URL	Actions
629781a465f4522e896ea4b2	forDawn	kong	http://gw-kong:8001	 
629f22ab7b28bbda915752d3	tykForDawn	tyk	http://gw-tyk:8080	 

Figura 5.12 Tabel de conexiuni

Pentru a crea o conexiune, se deschide un formular, cu câmpurile: „Name”, „Type” (prin intermediul unui dropdown, se pot alege valoarea „kong” sau „tyk”), „Admin API URL”. Mecanismul este același ca în cazurile creării unor servicii sau rute, mesajele de eroare fiind de forma: „Please provide another name”, „Please provide an admin\_api\_url!”.

În ceea ce privește editarea unei rute, formularul specific conține câmpurile: „ID” (acesta neputând fi modificat), „Name”, „Type” (prin intermediul unui dropdown, se poate alege valoarea „kong” sau „tyk”), „Admin API URL”. Acțiunile de control sunt identice cu cele din cazul editării unui serviciu sau a unei rute. Mesajele de eroare fiind identice cu cele menționate anterior.

Add a new connection

Name:

Type:

Admin\_api\_url:

Close Add

Figura 5.13 Formular de adăugare a unei conexiuni

Edit a connection

ID:

Name:

Type:

Admin\_api\_url:

Close Edit

Figura 5.14 Formular de editare a unei conexiuni

Acțiunea de ștergere a unei conexiuni, determină apariția unei ferestre de tip modal, ce afișează mesajul: „Do you want to delete the connection with the following ID?”, urmat de ID-ul conexiunii care va fi ștearsă. Se apasă butonul „Delete” pentru finalizarea operațiunii, și „Close” sau „X” pentru renunțarea la aceasta.

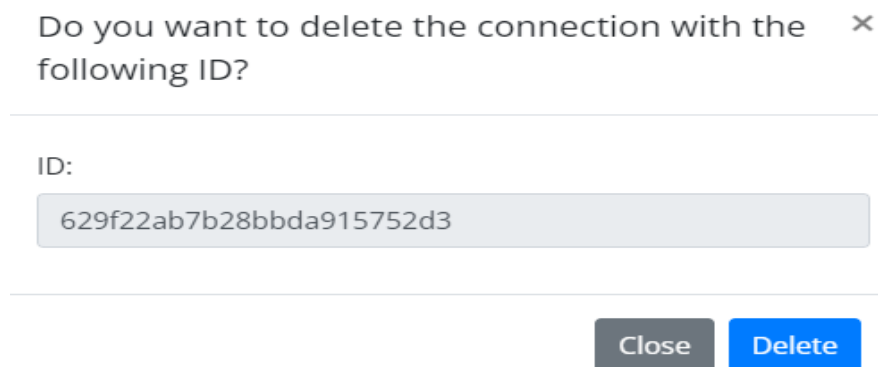


Figura 5.15 Fereastră de tip modal pentru ștergerea unei conexiuni

În cazul selectării opțiunii „Deployments” a meniului, utilizatorul ajunge la pagina „<http://localhost:5004/deployments>” unde este afișat mesajul de interogare „Which type of API Gateway would you like to deploy?” urmat de listarea a două opțiuni: „kong”, „tyk”. După realizarea alegerii, acesta este redirecționat la URL-ul „[http://localhost:5004/deployment-for/<connection\\_type>](http://localhost:5004/deployment-for/<connection_type>)”, unde este întâmpinat de mesajul de confirmare „Do you really want to deploy a <connection\_type> API Gateway?”, având posibilitatea de a continua acțiunea prin apăsarea butonului „Yes”, în cazul în care nu dorește acest lucru, apasă butonul „Back” al browser-ului sau orice altă opțiune din cadrul meniului. Dacă utilizatorul apasă butonul „Yes”, va apărea următorul mesaj: „A new <connection\_type> connection has been created.”, prin urmare, acesta va putea vizualiza conexiunea nou creată, accesând pagina „Connections”, având posibilitatea de a crea servicii/rute pentru aceasta.

Scopul final al utilizatorului fiind asocierea a unui API simplu cu un API Gateway, acesta trebuie să se asigure mai întâi că are cel puțin o rută creată în cadrul aplicației, cel puțin un serviciu pentru această și cel puțin o rută asociată serviciului.

Așadar, pasul următor este analizarea tabelului de conexiuni și identificarea portului din cadrul câmpului „Admin API URL”. În cazul Kong-ului, acesta fiind întrebuințat doar pentru ascultarea apelurilor de tip HTTP de către API-ul de administrare, prin urmare, pentru a accesa API-urile de test, se utilizează portul care ascultă traficul HTTP primit de la client și îl redirecționează către microserviciile din backend. În acest sens, în cazul creării unei conexiuni de tip Kong din pagina de Deployments, a fost stabilită o convenție, astfel încât, acest port să aibă o valoare egală cu valoarea portului prezent în câmpul „Admin API URL”, scăzând din aceasta valoarea 2000. Așadar, considerând figura (Figura 5.16), portul care ascultă traficul HTTP primit de la client va avea valoarea 41012.



+ Add New				
ID	Name	Type	Admin API URL	Actions
62b75c7952b751e9cd8648d0	kongNr12	kong	<a href="http://host.docker.internal:43012">http://host.docker.internal:43012</a>	 

Figura 5.16 Conexiunea unei instanțe Kong

În continuare, se verifică ruta asociată serviciului din cadrul conexiunii anterioare, având în vedere câmpul „Paths”, ce reprezintă calea de acces către API-ul gestionat.



<div>+ Add New</div>				
ID	Name	Paths	Service	Actions
7133ef32-234b-49d7-8017-8d8478c29a91	api1-route	/api1	api1-service	 

Figura 5.17 Rută asociată unui serviciu din cadrul unei conexiuni de tip Kong

Corelând aceste informații, putem compune URL-ul ce ajunge în endpoint-ul specificat în câmpul „Path” din tabelul de servicii, având în vedere serviciul de interes. Endpoint-ul aparținând API-ului al cărui port extern este precizat în același tabel, în cadrul câmpului „Port”. Așadar, URL-ul format va fi: „<http://localhost:41012/api1>”.

De exemplu, considerând figura (Figura 5.18), API-ul gestionat este cel al cărui port extern este 5001, iar endpointul în care acel URL va redirectiona utilizatorul este „/books” din componența API-ului respectiv.



<div>+ Add New</div>					
ID	Name	Path	Port	Host	Actions
4774c44b-8d2e-4f44-be2e-ed6c122977eb	api1-service	/books	5001	host.minikube.internal	 

Figura 5.18 Serviciu din cadrul unei conexiuni de tip Kong

În final, asocierea API-ului cu API-ul Gateway de tip Kong se va concretiza, accesând URL-ul menționat mai sus. Formatul paginii la care se ajunge în cele din urmă este similar cu conținutul figurii (Figura 5.19).

```
← → ↺ ⓘ localhost:41012/api1
{
  "Books": [
    {
      "author": "Jules Verne",
      "book_id": "0",
      "edition": "Pierre-Jules Hetzel",
      "price": "30.7",
      "title": "Two Years' Vacation",
      "year_written": "1888"
    },
    {
      "author": "Leo Tolstoy",
      "book_id": "1",
      "edition": "The Russian Messenger",
      "price": "25.5",
      "title": "Anna Karenina",
      "year_written": "1878"
    }
  ]
}
```

Figura 5.19 Exemplu de răspuns al primului API de test

În ceea ce privește Tyk-ul, portul din cadrul câmpului „Admin API URL” este întrebuințat atât pentru ascultarea apelurilor de către API-ul de administrare, cât și pentru ascultarea traficului primit de la client și redirectionarea către microserviciile din backend. Considerând figura (Figura 5.20), portul care ascultă traficul HTTP primit de la client are valoarea 44005.



+ Add New				
ID	Name	Type	Admin API URL	Actions
62b7694d7780e0049e192d2d	tykNr5	tyk	http://host.docker.internal:44005	 

Figura 5.20 Conexiunea unei instanțe Tyk

Se verifică ruta asociată serviciului din cadrul conexiunii anterioare, acest pas fiind identic cu cel din cazul unei conexiuni de tip Kong.


ID	Name	Paths	Service	Actions
api3-service	api3	/api3	api3-service	

Figura 5.21 Rută asociată unui serviciu din cadrul unei conexiuni de tip Tyk

URL-ul format pe baza acestor informații va fi: „<http://localhost:44005/api3>”.

Considerând figura (Figura 5.22), API-ul gestionat este cel al cărui port extern este 5003, iar endpointul în care acel URL va redirectiona utilizatorul este „/” din componența acestuia.



+ Add New					
ID	Name	Path	Port	Host	Actions
api3-service	api3-service	/	5003	host.docker.internal	 

Figura 5.22 Serviciu din cadrul unei conexiuni de tip Tyk

Formatul paginii la care se ajunge în cele din urmă este detaliat în conținutul figurii (Figura 5.23).

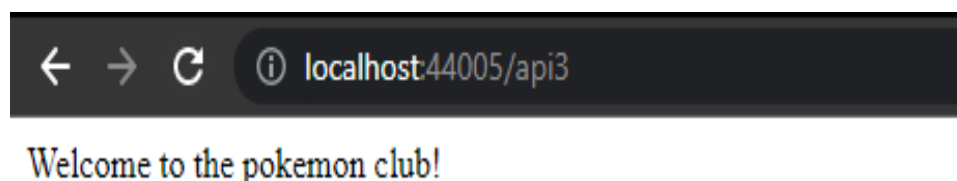


Figura 5.23 Exemplu de răspuns al celui de-al treilea API de test

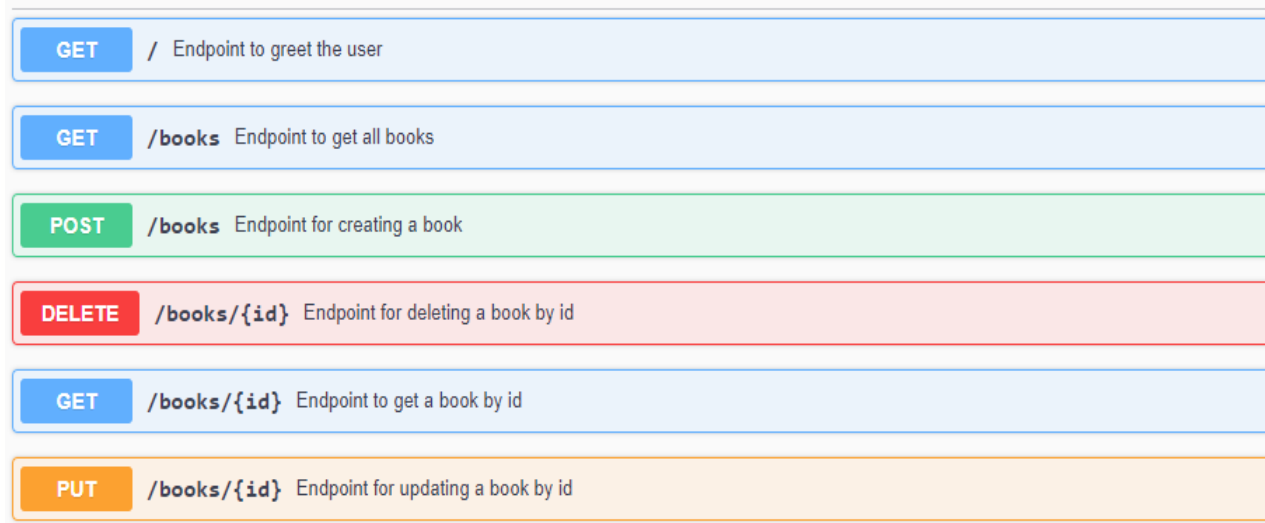


## 5.2 Specificațiile API-urilor REST de test (Swagger)

Swagger reprezintă un set de reguli, specificații și unelte pentru dezvoltarea și descrierea unor API-uri de tip REST, fiind open source. Principala contribuție a acestui framework în implementarea unui API este crearea unei documentații ușor de parcurs.

În cadrul proiectului, cea mai folosită componentă a Swagger-ului este interfața pentru utilizator, ce reprezintă o unealtă care ajută la generarea unei documentații de tipul API doc. Interfața este încorporată împreună cu aplicația pe care o descrie.

Am folosit Swagger pentru a realiza o documentație a API-urilor de test. În acest sens, pentru a ști ce endpointuri poate apela, utilizatorul poate accesa următoarele link-uri: „<http://localhost:5001/apidocs/>”, „<http://localhost:5002/apidocs/>”, „<http://localhost:5003/apidocs/>”. Accesând primul link, se obține documentația primului API de test. Astfel, după cum se observă în figura (Figura 5.24), în cadrul acesteia se pot vizualiza endpoint-urile ce îl compun, împreună cu operațiile suportate.



GET	/	Endpoint to greet the user
GET	/books	Endpoint to get all books
POST	/books	Endpoint for creating a book
DELETE	/books/{id}	Endpoint for deleting a book by id
GET	/books/{id}	Endpoint to get a book by id
PUT	/books/{id}	Endpoint for updating a book by id

Figura 5.24 Listarea endpoint-urilor și a metodelor suportate al primului API de test

Pentru mai multe detalii referitoare la operațiile suportate de endpoint-uri, utilizatorul va apăsa pe metoda dorită. Prin urmare, va apărea o desfășurare a metodei, așa cum se poate observa și în figura (Figura 5.25), unde se precizează dacă este necesară introducerea unui parametru, specificând numele acestuia. De asemenea, se oferă detalii asupra răspunsului primit în urma apelării endpoint-ului respectiv, prin metoda respectivă. Astfel, sunt precizate codul de răspuns, un exemplu de răspuns ce va fi primit de utilizator, împreună cu o scurtă descriere despre acesta, specificându-se și tipul acestuia.

**PUT**
**/books/{id}**
Endpoint for updating a book by id

Parameters
Try it out

Name	Description
<b>id</b> * required string (path)	<input type="text" value="id"/>

Responses
Response content type
application/json

Code	Description
200	Returns a dictionary with the updated object <div> Example Value Model </div> <pre> {   updated: {     "title": "Hamlet, Prince of Denmark",     "book_id": "7",     "author": "Shakespeare",     "year_written": 1603,     "edition": "Signet Classics",     "price": "7.95"   } } </pre>

Figura 5.25 Descrierea metodei PUT din cadrul primului API de test

Documentația celui de-al doilea API de test este disponibilă, accesand cel de-al doilea link menționat mai sus. Așadar, conform figurii (Figura 5.26) se obține listarea endpoint-urilor ce îl compun, împreună cu operațiile permise.

GET
/
Endpoint to greet the user

GET
/cars
Endpoint to get all cars

POST
/cars
Endpoint for creating a car

DELETE
/cars/{id}
Endpoint for deleting a car by id

GET
/cars/{id}
Endpoint to get a car by id

PUT
/cars/{id}
Endpoint for updating a car by id

Figura 5.26 Listarea endpoint-urilor și a metodelor suportate al celui de-al doilea API de test

În ceea ce privește documentația celui de-al treilea API de test, aceasta poate fi vizualizată, accesând cel de-al treilea link menționat anterior. Astfel, sunt înlanțuite endpoint-urile din componența acestuia, împreună cu operațiile suportate, conform figurii 5.27.

GET	/	Endpoint to greet the user
GET	/pokemons	Endpoint to get all pokemons
POST	/pokemons	Endpoint for creating a pokemon
DELETE	/pokemons/{id}	Endpoint for deleting a pokemon by id
GET	/pokemons/{id}	Endpoint to get a pokemon by id
PUT	/pokemons/{id}	Endpoint for updating a pokemon by id

Figura 5.27 Listarea endpoint-urilor și a metodelor suportate al celui de-al treilea API de test



## Concluziile proiectului

Proiectul a ajuns într-o stare de implementare ce oferă următoarele facilități: login, folosind useri locali, posibilitate de deployment unui API Gateway, în Kubernetes, în cazul în care acesta este de tipul Kong și în Docker, dacă este de tipul Tyk, fapt care va crea automat o conexiune cu tipul respectiv, crearea resurselor de tipul „servicii” și „rute” pentru acestea și vizualizarea, editarea sau ștergerea acestora. Astfel, garantându-se asocierea unui API simplu cu un API Gateway deja pus în funcțiune.

Folosind documentația expusă de cele două API Gateway Kong, Tyk am reușit să construiesc un API comun ce ține cont de particularitățile fiecărei soluții individuale. Însă, în cazul unei conexiuni de tip Tyk, există o limitare în ceea ce privește resursele de tip “rute”, astfel încât, acțiunile posibile asupra acestora sunt cele de vizualizare și editare, lucru cauzat de faptul că Tyk abstractizează resursele „serviciu” și „rută” în noțiunea de „apis”, timp în care, aceste resurse reprezintă două entități diferite în cadrul Kong-ului. Tot din acest motiv, o altă limitare în folosirea API Gateway-ului Tyk este reprezentată de atribuirea unei singure rute fiecărui serviciu.

De asemenea, au fost implementate constrângeri ce privesc unicitatea valorilor anumitor câmpuri asociate fiecărei resurse în parte, constrângeri ce nu sunt prezente în implementarea API-urilor de administrație ale API Gateway-urilor alese. În acest sens, am realizat un mecanism de validare, astfel încât, în situația în care aceste constrângeri sunt încălcate, utilizatorul va fi întâmpinat de un mesaj de eroare, în urma căruia i se aduce în vedere greșeala făcută.

Totodată, trebuie menționat că interfața grafică dezvoltată în cadrul acestei lucrări este benefică și din perspectiva costului asociat rulării interfeței grafice în cazul Tyk ce oferă o licență de paisprezece zile de tip Trial, după care este plătită.

Pentru a demonstra functionalitatea produsului, am implementat trei API-uri simple, pentru care am realizat o documentație folosind Swagger.

Consider că existența unei astfel de aplicații, reduce timpul de documentare în ceea ce privește utilizarea API Gateway-urilor alese, cât și efortul de a asocia un API simplu cu un API Gateway, acest lucru fiind posibil în urma completării unui subset de câmpuri. Un alt beneficiu este reprezentat de vizualizarea resurselor create, cu posibilitatea de adăugare, editare sau ștergere a acestora într-un mod facil.

Având în vedere continuarea proiectului, se pot extinde următoarele funcționalități: posibilitatea de deployment a altor tipuri de API Gateway-uri (fapt care implică documentarea cu privire la API-urile de administrare ale acestora, pentru a le apela în implementarea operațiilor CRUD pentru resursele de tip „servicii” și „rute”), adăugarea unei noi resurse, de tipul „plugin” pentru fiecare tip de conexiune în parte, care să expună functionalitatea din API Gateway-ul respectiv. În acest sens, se pot menționa: rate limiting, retry policy, circuit breaker, load balancing.

O altă îmbunătățire a produsului poate fi, adăugarea unui user cu drepturi de admin, dintre care se enumera: granularitatea drepturilor utilizatorilor, accesarea resurselor altor utilizatori și posibilitatea de a realiza operații pe acestea.



# Bibliografie

- [1] Microframework-ul Flask <https://flask.palletsprojects.com/en/2.1.x/>, accesat la data 17.02.2022
- [2] API Gateway-ul Kong <https://konghq.com/>, accesat la data 24.02.2022
- [3] API Gateway-ul Tyk <https://tyk.io/>, accesat la data 28.04.2022
- [4] Infrastructura virtuală Docker <https://www.docker.com/>, accesat la data 5.02.2022
- [5] Infrastructura virtuală Kubernetes <https://kubernetes.io/>, accesat la data 29.05.2022
- [6] CRUD Operations <https://www.educative.io/blog/crud-operations>, accesat la data 17.02.2022
- [7] Intro to APIs: History of APIs <https://blog.postman.com/intro-to-apis-history-of-apis/>, accesat la data 15.02.2022
- [8] API Gateway <https://www.nginx.com/learn/api-gateway/>, accesat la data 23.02.2022
- [9] REST: Advanced Research Topics and Practical Applications, Editura Springer, Cesare Pautasso, Erik Wilde, Rosa Alarcon, 2014
- [10] What is virtualization? <https://opensource.com/resources/virtualization>, accesat la data 2.02.2022
- [11] What is a container? <https://www.docker.com/resources/what-container/>, accesat la data 5.02.2022
- [12] Pods <https://kubernetes.io/docs/concepts/workloads/pods/>, accesat la data 29.05.2022
- [13] Server <https://www.britannica.com/technology/server>, accesat la data 4.02.2022
- [14] YAML <https://yaml.org/>, accesat la data 29.05.2022
- [15] JSON <https://www.json.org/json-en.html>, accesat la data 16.02.2022
- [16] Deployments <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>, accesat la data 29.05.2022
- [17] Service <https://kubernetes.io/docs/concepts/services-networking/service/>, accesat la data 29.05.2022
- [18] Persistent Volumea <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>, accesat la data 29.05.2022
- [19] Jobs <https://kubernetes.io/docs/concepts/workloads/controllers/job/>, accesat la data 29.05.2022
- [20] Ce este o baza de date <https://www.oracle.com/ro/database/what-is-database/>, accesat la data 14.02.2022
- [21] What is a frontend? <https://airfocus.com/glossary/what-is-a-front-end/>, accesat la data 4.04.2022
- [22] What is Backend Development? <https://www.guru99.com/what-is-backend-developer.html>, accesat la data 10.02.2022
- [23] Python <https://www.python.org/>, accesat la data 15.02.2022
- [24] HTML <https://html.com/>, accesat la data 4.04.2022
- [25] What is kernel? <https://www.techtarget.com/searchdatacenter/definition/kernel>, accesat la data 29.05.2022
- [26] System Calls in Operating System <https://www.javatpoint.com/system-calls-in-operating-system>, accesat la data 29.05.2022

- [27] Swagger <https://www.techtarget.com/searchapparchitecture/definition/Swagger>, accesat la data 16.02.2022
- [28] Cloud <https://www.oracle.com/ro/cloud/what-is-cloud-computing/>, accesat la data 26.05.2022
- [29] What are microservices? <https://microservices.io/>, accesat la data 26.05.2022
- [30] Retry policy <https://developers.liveperson.com/retry-policy-recommendations.html>, accesat la data 23.02.2022
- [31] Circuit Breaker pattern  
<https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker>, accesat la data 23.02.2022
- [32] Kong Admin API <https://docs.konghq.com/gateway/latest/admin-api/>, accesat la data 24.02.2022
- [33] Service Object <https://docs.konghq.com/gateway/latest/admin-api/#service-object>, accesat la data 24.02.2022
- [34] Route Object <https://docs.konghq.com/gateway/latest/admin-api/#route-object>, accesat la data 24.02.2022
- [35] Tyk Gateway API <https://tyk.io/docs/tyk-gateway-api/>, accesat la data 28.04.2022
- [36] Redis - Official Image [https://hub.docker.com/\\_/redis](https://hub.docker.com/_/redis), accesat la data 8.03.2022
- [37] Mongo - Official Image [https://hub.docker.com/\\_/mongo](https://hub.docker.com/_/mongo), accesat la data 29.02.2022
- [38] Jinja <https://palletsprojects.com/p/jinja/>, accesat la data 4.04.2022
- [39] Kong on Kubernetes  
<https://medium.com/engineering-applift/kong-in-kubernetes-1c4ef3c47dae>, accesat la data 3.06.2022
- [40] Bootstrap <https://getbootstrap.com/>, accesat la data 4.04.2022
- [41] Demo Bootstrap  
<https://www.tutorialrepublic.com/codelab.php?topic=bootstrap&file=table-with-add-and-delete-row-feature>, accesat la data 4.04.2022



# Anexa 1

## /api\_management/Dockerfile

```
FROM python:3.10.2

RUN pip install flask
RUN pip install pymongo
RUN pip install Flask-PyMongo
RUN pip install requests

RUN apt-get update
RUN apt-get install docker.io -y

WORKDIR /opt/app
COPY . /opt/app/

RUN curl -LO
"https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/li
nux/amd64/kubect1"
RUN chmod +x kubect1

ENV KUBECONFIG /opt/app/config.yaml

ENTRYPOINT ["python3", "./server.py"]
```

## /api\_management/kong.yaml

```
# Persistent Volume
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv-{{pg_port}}
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/postgres-pv-{{pg_port}}

---
# Persistent Volume Claim
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc-{{pg_port}}
  labels:
    type: local
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
  volumeName: postgres-pv-{{pg_port}}
```

```
---
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-kong-{{pg_port}}
spec:
  selector:
    matchLabels:
      app: postgres-container
  template:
    metadata:
      labels:
        app: postgres-container
    spec:
      containers:
        - name: postgres-container
          image: postgres:9.6
          env:
            - name: POSTGRES_USER
              value: "kong"

            - name: POSTGRES_DB
              value: "kong"

            - name: POSTGRES_PASSWORD
              value: "kong"

          ports:
            - containerPort: 5432
          volumeMounts:
            - mountPath:
/var/lib/postgresql/data
              name: postgres-volume-mount
          volumes:
            - name: postgres-volume-mount
              persistentVolumeClaim:
                claimName:
postgres-pvc-{{pg_port}}
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-kong-{{pg_port}}
spec:
  selector:
    app: postgres-container
  ports:
    - port: {{pg_port}}
      protocol: TCP
      targetPort: 5432
  type: LoadBalancer

---
# Migration Job
apiVersion: batch/v1
kind: Job
metadata:
  name: kong-migrations-{{pg_port}}
# namespace: {{ .namespace }}
spec:
```

```

template:
  metadata:
    name: kong-migrations-{{pg_port}}
    namespace: {{ .namespace }}
  spec:
    containers:
      - name: kong-migrations-{{pg_port}}
        image: kong
        env:
          - name: KONG_DATABASE
            value: postgres
          - name: KONG_PG_HOST
            value: postgres-kong-{{pg_port}}
          - name: KONG_PG_USER
            value: kong
          - name: KONG_PG_PASSWORD
            value: kong
          - name:
KONG_CASSANDRA_CONTACT_POINTS
            value: postgres-kong-{{pg_port}}
          - name: KONG_PG_PORT
            value: "{{pg_port}}"
            command: [ "/bin/sh", "-c", "kong
migrations bootstrap", "--vv" ]
            restartPolicy: Never
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kong-deployment-{{pg_port}}
  labels:
    app: kong
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kong
  template:
    metadata:
      labels:
        app: kong
    spec:
      containers:
        - name: kong
          image: kong
          ports:
            - containerPort: 8000
            - containerPort: 8443
            - containerPort: 8001
            - containerPort: 8444
          env:
            - name: KONG_DATABASE
              value: "postgres"
            - name: KONG_PG_HOST
              value:
"postgres-kong-{{pg_port}}"
            - name: KONG_PG_PASSWORD
              value: "kong"
            - name:
KONG_CASSANDRA_CONTACT_POINTS
              value:
"postgres-kong-{{pg_port}}"
            - name: KONG_PROXY_ACCESS_LOG
              value: "/dev/stdout"
            - name: KONG_ADMIN_ACCESS_LOG
              value: "/dev/stdout"
            - name: KONG_PROXY_ERROR_LOG
              value: "/dev/stderr"
            - name: KONG_ADMIN_ERROR_LOG
              value: "/dev/stderr"

```

```

      - name: KONG_ADMIN_LISTEN
        value: "0.0.0.0:8001,
0.0.0.0:8444 ssl"
      - name: KONG_PG_PORT
        value: "{{pg_port}}"
---
apiVersion: v1
kind: Service
metadata:
  name: kong-service-{{pg_port}}
spec:
  type: LoadBalancer
  selector:
    app: kong
  ports:
    - name: port1
      protocol: TCP
      port: {{port1}}
      targetPort: 8000
    - name: port2
      protocol: TCP
      port: {{port2}}
      targetPort: 8443
    - name: port3
      protocol: TCP
      port: {{port3}}
      targetPort: 8001
    - name: port4
      protocol: TCP
      port: {{port4}}
      targetPort: 8444

```

### /api\_management/run.sh

```

docker build -t api_management .
docker rm -f api_management || true
docker network create gw || true

```

```

docker run -d \
--net gw \
--link mongodb:mongodb \
--name api_management \
-p 5008:5000 \
-v
/mnt/c/Users/andon/AppData/Roaming/SPB_16.6/
.minikube/ca.crt:/opt/app/ca.crt \
-v
/mnt/c/Users/andon/AppData/Roaming/SPB_16.6/
.minikube/profiles/minikube/client.crt:/opt/
app/client.crt \
-v
/mnt/c/Users/andon/AppData/Roaming/SPB_16.6/
.minikube/profiles/minikube/client.key:/opt/
app/client.key \
-v /var/run/docker.sock:/var/run/docker.sock \
--add-host kubernetes:192.168.65.2 \
api_management

```

### /api\_management/run\_tyk.sh

```

docker rm -f gw-tyk{{port1}} || true

docker run -d \
  --name gw-tyk{{port1}} \
  --network gw \
  --link redis:redis \

```

```
-p {{port1}}:8080 \
-v
/mnt/c/Users/andon/Desktop/licenta/api_management/tyk.standalone.conf:/opt/tyk-gateway/tyk.conf \
-v
/mnt/c/Users/andon/Desktop/licenta/api_management/apps{{port1}}:/opt/tyk-gateway/apps \

docker.tyk.io/tyk-gateway/tyk-gateway:latest
```

### /api\_management/server.py

```
from operator import truth
from flask import *
from flask_pymongo import PyMongo
from bson.objectid import ObjectId
import requests
import time
import os
app = Flask(__name__)
app.config['MONGO_DBNAME'] = 'users'
app.config['MONGO_URI'] = "mongodb://mongodb:27017/users"

mongo = PyMongo(app)

#USERS
#get all users
@app.route('/users', methods = ['GET'])
def get_users():
    users = mongo.db.users          #the collection i have in my db
    output = []

    for query in users.find():      #find all the documents in the collection
        output.append({'_id': str(query['_id']), 'name': query['name'], 'password': query['password'], 'confirm_password': query['confirm_password']})

    return jsonify({'users': output})

#get a user by name
@app.route('/users/<string:name>', methods = ['GET'])
def get_user(name):
    users = mongo.db.users

    query = users.find_one({'name': name})
    if query:
        output = {'_id': str(query['_id']), 'name': query['name'], 'password': query['password'], 'confirm_password': query['confirm_password']}
    else:
        output = 'No results found'
    return jsonify({"user": output})

#verify if a user is registered
@app.route('/users/is_registered', methods = ['POST'])
def find_one():
    users = mongo.db.users
```

```
if ('name' in request.json) and (request.json['name'] != '') and ('password' in request.json) and (request.json['password'] != ''):
    _name = request.json['name']
    query = users.find_one({'name': _name})
    if query is not None:
        if ('password' in request.json) and (request.json['password'] != ''):
            if query['password'] == request.json['password']:
                return jsonify({"response": "success", "message": ''})
            else:
                return jsonify({"response": "fail", "message": "Wrong credentials!"})
        else:
            return jsonify({"response": "fail", "message": "Please provide both username and password!"})
    else:
        return jsonify({"response": "fail", "message": "This user is not registered."})
    else:
        return jsonify({"response": "fail", "message": "Please provide both username and password!"})

#create a user
@app.route('/create_user', methods = ['POST'])
def add_user():
    users = mongo.db.users

    if ('name' in request.json) and (request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail", "message": "Please provide both username and password!"})

    if ('password' in request.json) and (request.json['password'] != ''):
        password = request.json['password']
    else:
        return jsonify({"response": "fail", "message": "Please provide both username and password!"})

    if ('confirm_password' in request.json) and (request.json['confirm_password'] != ''):
        confirm_password = request.json['confirm_password']
    else:
        return jsonify({"response": "fail", "message": "Please confirm the password!"})

    query = users.find_one({'name': name})
    if query is not None:
        return jsonify({"response": "fail", "message": "Please provide another username"})

    if(password == confirm_password):
```

```

        if ('name' in request.json) and
        ('password' in request.json) and
        ('confirm_password' in request.json) and
        request.method == 'POST':
            users.insert_one({'name': name,
            'password': password, 'confirm_password':
            confirm_password})

            return jsonify({"response" :
            "success", "message": 'User created
            successfully. Please log in!'})
            else:
                return jsonify({"response": "fail",
            "message": "Password and password
            confirmation don't match"})

#update a user
@app.route('/users/<id>', methods = ['PUT'])
def update_user(id):
    users = mongo.db.users

    if 'name' in request.json:
        name = request.json['name']
        query = users.find_one({'name':
name})

        if query is not None:
            return jsonify({"response":
            "fail", "message": "Please provide another
            name"})
            else:
                users.update_one({'_id':
            ObjectId(id)}, {'$set': {'name': name}})

        if 'password' in request.json:
            password = request.json['password']
            users.update_one({'_id':
            ObjectId(id)}, {"$set": {"password":
            password}})

        if 'confirm_password' in request.json:
            confirm_password =
            request.json['confirm_password']
            users.update_one({'_id':
            ObjectId(id)}, {"$set": {"confirm_password":
            confirm_password}})

    return jsonify({"response" : "succes",
            "message" : "User updated successfully"})

#delete a user
@app.route('/users/<id>', methods =
['DELETE'])
def delete_user(id):
    mongo.db.users.delete_one({'_id':
            ObjectId(id)})
    return jsonify({"response" : "succes",
            "message" : "User deleted successfully"})

#CONNECTIONS
#function to get all connections from a
specific user
def get_connections_from_user(id):
    connections = mongo.db.connections

```

```

    user_connections =
    mongo.db.user_connections
    output = []

    user_connection_query =
    list(user_connections.find({'user_id': id}))
    print(user_connection_query, flush =
    True)

    for item in user_connection_query:
        print(str(item['connection_id']),
        flush = True)

        connection_query =
        connections.find_one({'_id':
        ObjectId(str(item['connection_id'])) })
        print(connection_query, flush =
        True)

        output.append({'_id':
        str(connection_query['_id']), 'name':
        connection_query['name'], 'type':
        connection_query['type'], 'admin_api_url':
        connection_query['admin_api_url']})

    return output

#function to get a connection by name for a
specific user
def get_connection_by_name(user_id, name):
    connections_from_user =
    get_connections_from_user(user_id)
    print("ALO", flush = True)
    print(name, flush = True)
    output = {}
    for item in connections_from_user:
        print(item, flush = True)
        if (item['name'] == name) :
            print(item['name'], flush =
            True)

            output = {'_id':
            str(item['_id']), 'name': item['name'],
            'type': item['type'], 'admin_api_url':
            item['admin_api_url']}
            return output

    return output

#function to get a connection by ID
def get_connection_by_ID(id):
    connections = mongo.db.connections
    output = {}

    query = connections.find_one({'_id':
            ObjectId(id)})
    if query:
        output = {'_id': str(query['_id']),
            'name': query['name'], 'type':
            query['type'], 'admin_api_url':
            query['admin_api_url']}

    return output

#get all connections
@app.route('/connections', methods =
['GET'])
def get_connections():
    connections = mongo.db.connections
    #the collection i have in my db
    output = []

```

```

        for query in connections.find():
#find all the documents in the collection
            output.append({'_id':
str(query['_id']), 'name': query['name'],
'type': query['type'], 'admin_api_url':
query['admin_api_url']})

        return jsonify({'connections': output})

#get a connection by id *
@app.route('/connections/<id>', methods =
['GET'])
def get_connection_by_id(id):
    output = get_connection_by_ID(id)
    return jsonify({"connection": output})

#get all connections from a specific user *
@app.route('/users/<user_id>/connections',
methods = ['GET'])
def get_user_connections(user_id):
    output =
get_connections_from_user(user_id)
    return jsonify({'connections': output})

#get a connection by name for a specific
user
@app.route('/users/<user_id>/connections/<na
me>', methods = ['GET'])
def get_connection(user_id, name):
    output = get_connection_by_name(user_id,
name)
    return jsonify({"connection": output})

#delete a connection *
@app.route('/connections/<id>', methods =
['DELETE'])
def delete_connection(id):
    mongo.db.connections.delete_one({'_id':
ObjectId(id)})
    return jsonify({"response" : "succes",
"message" : "Connection deleted
successfully"})

#get all user_connections dependencies
@app.route('/user_connections', methods =
['GET'])
def get_connections_dependency():
    connections = mongo.db.user_connections
#the collection i have in my db
    output = []

    for query in connections.find():
#find all the documents in the collection
        output.append({'_id':
str(query['_id']), 'connection_id':
query['connection_id'], 'user_id':
query['user_id'] })

    return jsonify({'connections': output})

#get user_connection dependency where id
connection is...
@app.route('/user_connections/<connection_id
>', methods = ['GET'])
def get_connection_user(connection_id):
    connections = mongo.db.user_connections
    output = []
    query =
connections.find_one({'connection_id':
connection_id})

```

```

        if query:
            output = {'_id': str(query['_id']),
'user_id': query['user_id'],
'connection_id': query['connection_id']}
            return jsonify({"dependency": output})

#create user-connection dependency
@app.route('/user_connections', methods =
['POST'])
def add_connection_dependency():
    connections = mongo.db.user_connections

    if ('connection_id' in request.json) and
(request.json['connection_id'] != ''):
        connection_id =
request.json['connection_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a
connection_id!"})

    if ('user_id' in request.json) and
(request.json['user_id'] != ''):
        user_id = request.json['user_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a user_id!"})

    if ('connection_id' in request.json) and
('user_id' in request.json) and
request.method == 'POST':

connections.insert_one({'connection_id':
connection_id, 'user_id': user_id})

    return jsonify({"response" : "success",
"message": 'User_connection dependency
created successfully!'})

#delete a user_connection dependency
@app.route('/user_connections/<id>', methods
= ['DELETE'])
def delete_connection_dependency(id):

mongo.db.user_connections.delete_one({'_id':
ObjectId(id)})
    return jsonify({"response" : "succes",
"message" : "User_connection dependency
deleted successfully"})

#create a connection for a specific user *
@app.route('/users/<user_id>/connections',
methods = ['POST'])
def add_connection(user_id):
    connections = mongo.db.connections

    if ('name' in request.json) and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('type' in request.json) and
(request.json['type'] != ''):
        type = request.json['type']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a type!"})

```

```

        if ('admin_api_url' in request.json) and
(request.json['admin_api_url'] != ''):
            admin_api_url =
request.json['admin_api_url']
        else:
            return jsonify({"response": "fail",
"message": "Please provide an
admin_api_url!"})

        print(user_id, flush = True )
        output =
get_connections_from_user(user_id)
        print(output, flush = True )

        for i in range(len(output)):
            #print(i, flush = True )
            #print(output[i], flush = True )
            if (output[i]['name'] == name):
                return jsonify({"response":
"fail", "message": "Please provide another
name"})

            if ('name' in request.json) and ('type'
in request.json) and ('admin_api_url' in
request.json) and request.method == 'POST':
                connection_id =
connections.insert_one({'name': name,
'type': type, 'admin_api_url':
admin_api_url}).inserted_id

            return jsonify({"response" : "success",
"message": 'Connection created
successfully!', 'new_connection_id':
str(connection_id)})

#update a connection for a specific user *
@app.route('/users/<user_id>/connections/<co
nnection_id>', methods = ['PUT'])
def update_connection(user_id,
connection_id):
    connections = mongo.db.connections

    if ('name' in request.json) and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('type' in request.json) and
(request.json['type'] != ''):
        type = request.json['type']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a type!"})

    if ('admin_api_url' in request.json) and
(request.json['admin_api_url'] != ''):
        admin_api_url =
request.json['admin_api_url']
    else:
        return jsonify({"response": "fail",
"message": "Please provide an
admin_api_url!"})

    if ('name' in request.json) and ('type'
in request.json) and ('admin_api_url' in
request.json) and request.method == 'PUT':

```

```

        output =
get_connections_from_user(user_id)
        for i in range(len(output)):
            if (output[i]['name'] == name)
and (output[i]['_id'] != connection_id):
                return jsonify({"response":
"fail", "message": "Please provide another
name"})
            else:

connections.update_one({'_id':
ObjectId(connection_id)}, {'$set': {'name':
name}})

                connections.update_one({'_id':
ObjectId(connection_id)}, {'$set': {'type':
type}})

                connections.update_one({'_id':
ObjectId(connection_id)}, {'$set':
{"admin_api_url": admin_api_url}})

                return jsonify({"response" : "success",
"message" : "Connection updated
successfully"})

#SERVICES
#function to get all servicies from a
specific connection
def
get_servicies_from_connection(connection_id)
:
    connection_services =
mongo.db.connection_services
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    output = []

    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )
    connection_service_query =
list(connection_services.find({'connection_i
d': connection_id}))
    print(connection_service_query, flush =
True)

    for item in connection_service_query:
        print(str(item['service_id']), flush
= True)

        res =
requests.get(connection_admin_api_url +
'/services/' + str(item['service_id']))
        dictFromServer = res.json()
        print(dictFromServer, flush = True )
        output.append(dictFromServer)

    return output

```

```

#get all services from a specific connection
*
@app.route('/connections/<connection_id>/ser
vices', methods = ['GET'])
def get_connection_services(connection_id):
    output =
get_servicies_from_connection(connection_id)
    return jsonify({'services': output})

#get all services ??
@app.route('/services', methods = ['GET'])
def get_services():
    res =
requests.get('http://gw-kong:8001/services/'
)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return jsonify({"content":
dictFromServer, "response": res.status_code
}))

#get a service by name *
@app.route('/connections/<connection_id>/ser
vices/<name>', methods = ['GET'])
def get_service_by_name(connection_id,
name):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )
    res =
requests.get(connection_admin_api_url +
'/services/' + name)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return jsonify({"content":
dictFromServer, "response": res.status_code
}))

#create a service for a specific connection*
@app.route('/connections/<connection_id>/ser
vices', methods = ['POST'])
def create_service(connection_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )

    if ('name' in request.json) and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('path' in request.json) and
(request.json['path'] != ''):
        path = request.json['path']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a path!"})

    if ('port' in request.json) and
(request.json['port'] != ''):

```

```

        port = request.json['port']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a port!"})

    if ('host' in request.json) and
(request.json['host'] != ''):
        host = request.json['host']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a host!"})

    if ('name' in request.json) and ('path'
in request.json) and ('port' in
request.json) and ('host' in request.json)
and request.method == 'POST':
        dictToSend = {"name": name, "path":
path, "port": port, "host": host}

        res =
requests.post(connection_admin_api_url +
'/services/', json = dictToSend)
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        return jsonify({"content":
dictFromServer, "response": res.status_code
}))

#update a service for a specific connection
*
@app.route('/connections/<connection_id>/ser
vices/<service_id>', methods = ['PUT'])
def update_service(connection_id,
service_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']

    if 'name' in request.json and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if 'path' in request.json and
(request.json['path'] != ''):
        path = request.json['path']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a path!"})

    if 'port' in request.json and
(request.json['port'] != ''):
        port = request.json['port']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a port!"})

    if 'host' in request.json and
(request.json['host'] != ''):
        host = request.json['host']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a host!"})

    if ( ('name' in request.json) or ('path'
in request.json) or ('port' in request.json)

```

```

or ('host' in request.json)) and
request.method == 'PUT':
    dictToSend = {"name": name, "path":
path, "port": port, "host": host}

    res =
requests.put(connection_admin_api_url +
'/services/' + service_id, json =
dictToSend)

    dictFromServer = res.json()
    print(dictFromServer, flush = True)

    return jsonify({"content":
dictFromServer, "response": res.status_code
})

#delete a service *
@app.route('/connections/<connection_id>/ser
vices/<name>', methods = ['DELETE'])
def delete_service(connection_id, name):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    res =
requests.delete(connection_admin_api_url +
'/services/' + name)

    if(res.status_code == 400) :
        dictFromServer = res.json()
        print(dictFromServer, flush = True )
        return jsonify({"content":
dictFromServer, "response": res.status_code
})
    else:
        return jsonify({"response":
res.status_code })

#get all connection_services dependencies
@app.route('/connection_services', methods =
['GET'])
def get_services_dependency():
    services = mongo.db.connection_services
#the collection i have in my db
    output = []

    for query in services.find():
#find all the documents in the collection
        output.append({'_id':
str(query['_id']), 'service_id':
query['service_id'], 'connection_id':
query['connection_id'] })

    return jsonify({'services': output})

#get connection_service dependency where id
service is...
@app.route('/connection_services/<service_id
>', methods = ['GET'])
def get_connection_service(service_id):
    connections =
mongo.db.connection_services

    query =
connections.find_one({'service_id':
service_id})
    if query:
        output = {'_id': str(query['_id']),
'service_id': query['service_id'],
'connection_id': query['connection_id']}
    else:

```

```

        output = 'No results found'
        return jsonify({"dependency": output})

#create connection-service dependency
@app.route('/connection_services', methods =
['POST'])
def add_service_dependency():
    services = mongo.db.connection_services

    if ('service_id' in request.json) and
(request.json['service_id'] != ''):
        service_id =
request.json['service_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a service_id!"})

    if ('connection_id' in request.json) and
(request.json['connection_id'] != ''):
        connection_id =
request.json['connection_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a
connection_id!"})

    if ('service_id' in request.json) and
('connection_id' in request.json) and
request.method == 'POST':
        connection_service_id =
services.insert_one({'service_id':
service_id, 'connection_id':
connection_id}).inserted_id #generates a
random id
        new_service =
services.find_one({'_id':
connection_service_id})
        #output = {'service_id':
new_service['service_id'], 'connection_id':
new_service['connection_id']}

        return jsonify({"response" :
"success", "message": 'connection_service
dependency created successfully!'})

#delete a connection_service dependency
@app.route('/connection_services/<id>',
methods = ['DELETE'])
def delete_service_dependency(id):

mongo.db.connection_services.delete_one({'_i
d': ObjectId(id)})
    return jsonify({"response" : "succes",
"message" : "connection_service dependency
deleted successfully"})

#ROUTES
#function to get all routes from a specific
connection
def
get_routes_from_connection(connection_id):

```



```

        connection_routes =
        mongo.db.connection_routes
        connection_admin_api_url =
        get_connection_by_ID(connection_id)['admin_a
        pi_url']
        output = []

        connection_route_query =
        list(connection_routes.find({'connection_id'
        : connection_id}))
        print(connection_route_query, flush =
        True)

        for item in connection_route_query:
            print(str(item['route_id']), flush =
            True)

            res =
            requests.get(connection_admin_api_url +
            '/routes/' + str(item['route_id']))
            dictFromServer = res.json()
            print(dictFromServer, flush = True )
            output.append(dictFromServer)

        return output

#function to get a route by id
def get_route_by_ID(connection_id,
route_id):
    connection_admin_api_url =
    get_connection_by_ID(connection_id)['admin_a
    pi_url']
    res =
    requests.get(connection_admin_api_url +
    '/routes/' + route_id)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return dictFromServer

#get all routes
@app.route('/routes', methods = ['GET'])
def get_routes():
    res =
    requests.get('http://gw-kong:8001/routes/')
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return jsonify({"content":
    dictFromServer, "response": res.status_code
    })

#get a route by name
@app.route('/connections/<connection_id>/rou
tes/<name>', methods = ['GET'])
def get_route_by_name(connection_id, name):
    connection_admin_api_url =
    get_connection_by_ID(connection_id)['admin_a
    pi_url']
    res =
    requests.get(connection_admin_api_url +
    '/routes/' + name)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return jsonify({"content":
    dictFromServer, "response": res.status_code
    })

#get all routes from a specific connection *
@app.route('/connections/<connection_id>/rou
tes', methods = ['GET'])
def get_connection_routes(connection_id):

```

```

        output =
        get_routes_from_connection(connection_id)

        return jsonify({'routes': output})

#create a route for a specific connection *
@app.route('/connections/<connection_id>/rou
tes', methods = ['POST'])
def create_route(connection_id):
    connection_admin_api_url =
    get_connection_by_ID(connection_id)['admin_a
    pi_url']
    routes =
    get_routes_from_connection(connection_id)

    if ('name' in request.json) and
    (request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
        "message": "Please provide a name!"})

    if ('paths' in request.json) and
    (request.json['paths'] != ''):
        if routes:
            for item in routes:
                if(request.json['paths'] !=
                item['paths']):
                    paths =
                    request.json['paths']
                else:
                    return
                    jsonify({"response": "fail", "message":
                    "Please provide another path!"})
            else:
                paths = request.json['paths']
        else:
            return jsonify({"response": "fail",
            "message": "Please provide at least a
            path!"})

    if ('service' in request.json) and
    (request.json['service'] != ''):
        service = request.json['service']
    else:
        return jsonify({"response": "fail",
        "message": "Please provide a service!"})

    if ('name' in request.json) and ('paths'
    in request.json) and ('service' in
    request.json) and request.method == 'POST':
        dictToSend = {"name": name, "paths":
        paths, "service": service}

        res =
        requests.post(connection_admin_api_url +
        '/routes/', json = dictToSend)
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        return jsonify({"content":
        dictFromServer, "response": res.status_code
        })

#update a route *
@app.route('/connections/<connection_id>/rou
tes/<route_id>', methods = ['PUT'])
def update_route(connection_id, route_id):

```

```

        connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
        routes =
get_routes_from_connection(connection_id)
        current_route =
get_route_by_ID(connection_id, route_id)

        if 'name' in request.json and
(request.json['name'] != ''):
            name = request.json['name']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a name!"})

        if ('paths' in request.json) and
(request.json['paths'] != ''):
            if routes:
                for item in routes:
                    if item != current_route:
                        if(request.json['paths']
!= item['paths']):
                            paths =
request.json['paths']
                        else:
                            return
jsonify({"response": "fail", "message":
"Please provide another path!"})
                        else:
                            paths =
request.json['paths']
                        else:
                            paths = request.json['paths']
                    else:
                        return jsonify({"response": "fail",
"message": "Please provide at least a
path!"})

        if 'service' in request.json and
(request.json['service'] != ''):
            service = request.json['service']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a service ID!"})

        if (('name' in request.json) or ('paths'
in request.json) or ('service' in
request.json)) and request.method == 'PUT':
            dictToSend = { "name": name,
"paths": paths, "service": service}

            res =
requests.put(connection_admin_api_url +
'/routes/' + route_id, json = dictToSend)
            dictFromServer = res.json()
            print(dictFromServer, flush = True )

            return jsonify({"content":
dictFromServer, "response": res.status_code
})

#delete a route *
@app.route('/connections/<connection_id>/rou
tes/<name>', methods = ['DELETE'])
def delete_route(connection_id, name):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(name, flush = True )

```

```

        res =
requests.delete(connection_admin_api_url +
'/routes/' + name)
        return jsonify({"response":
res.status_code})

#get all connection_routes dependencies
@app.route('/connection_routes', methods =
['GET'])
def get_routes_dependency():
    routes = mongo.db.connection_routes
#the collection i have in my db
    output = []

    for query in routes.find():          #find
all the documents in the collection
        output.append({'_id':
str(query['_id']), 'route_id':
query['route_id'], 'connection_id':
query['connection_id'] })

    return jsonify({'routes': output})

#create connection-route dependency
@app.route('/connection_routes', methods =
['POST'])
def add_route_dependency():
    routes = mongo.db.connection_routes

    if ('route_id' in request.json) and
(request.json['route_id'] != ''):
        route_id = request.json['route_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a route_id!"})

    if ('connection_id' in request.json) and
(request.json['connection_id'] != ''):
        connection_id =
request.json['connection_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a
connection_id!"})

    if ('route_id' in request.json) and
('connection_id' in request.json) and
request.method == 'POST':
        connection_route_id =
routes.insert_one({'route_id': route_id,
'connection_id': connection_id}).inserted_id
#generates a random id
        new_route = routes.find_one({'_id':
connection_route_id})
        #output = {'route_id':
new_route['route_id'], 'connection_id':
new_route['connection_id']}

        return jsonify({"response" :
"success", "message": 'connection_route
dependency created successfully!'})

#get connection_route dependency where id
route is...
@app.route('/connection_routes/<route_id>',
methods = ['GET'])
def get_connection_route(route_id):
    connections = mongo.db.connection_routes

```

```

        query =
connections.find_one({'route_id': route_id})
        if query:
            output = {'_id': str(query['_id']),
'route_id': query['route_id'],
'connection_id': query['connection_id']}
        else:
            output = 'No results found'
        return jsonify({"dependency": output})

#delete a connection_route dependency
@app.route('/connection_routes/<id>',
methods = ['DELETE'])
def delete_route_dependency(id):

mongo.db.connection_routes.delete_one({'_id'
: ObjectId(id)})
        return jsonify({"response" : "succes",
"message" : "connection_route dependency
deleted successfully"})

#TYK
#function to get all apis for a specific
connection
def get_tyk_apis(connection_id):
    connection_tyk_apis =
mongo.db.connection_tyk_apis
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    output = []
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )

    connection_tyk_api_query =
list(connection_tyk_apis.find({'connection_i
d': connection_id}))
    print(connection_tyk_api_query, flush =
True)

    if connection_tyk_api_query:
        for item in
connection_tyk_api_query:
            print(item['tyk_api_id'], flush
= True)

            res =
requests.get(connection_admin_api_url +
'/tyk/apis/' + item['tyk_api_id'],
headers={"x-tyk-authorization": "352d20ee67be
67f6340b4c0605b044b7"})
            dictFromServer = res.json()
            print(dictFromServer, flush =
True )

            output.append(dictFromServer)

    return output

#function to do reload
def reload(connection_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']

```

```

        print(connection_admin_api_url, flush =
True )
        print("ALO", flush = True )

        res =
requests.get(connection_admin_api_url +
'/tyk/reload',
headers={"x-tyk-authorization": "352d20ee67be
67f6340b4c0605b044b7"})
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        time.sleep(1.5)
        return dictFromServer

#function to get api by id
def get_api_by_id(connection_id,
tyk_api_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )

    res =
requests.get(connection_admin_api_url +
'/tyk/apis/' + tyk_api_id,
headers={"x-tyk-authorization": "352d20ee67be
67f6340b4c0605b044b7"})

    return res

#reload
@app.route('/connections/<connection_id>/tyk
/reload', methods = ['GET'])
def do_reload(connection_id):
    dictFromServer = reload(connection_id)

    return jsonify({'content':
dictFromServer})

#get all apis
@app.route('/tyk/apis', methods = ['GET'])
def get_all_tyk_apis():
    res =
requests.get('http://gw-tyk:8080//tyk/apis/'
,
headers={"x-tyk-authorization": "352d20ee67be
67f6340b4c0605b044b7"})
    dictFromServer = res.json()
    print(dictFromServer, flush = True )
    return jsonify({"content":
dictFromServer, "response": res.status_code
})

#get all apis for a specific connection
@app.route('/connections/<connection_id>/tyk
/apis', methods = ['GET'])
def
get_all_tyk_apis_per_connection(connection_i
d):
    dictFromServer =
get_tyk_apis(connection_id)
    return jsonify({'content':
dictFromServer})

#get api by id
@app.route('/connections/<connection_id>/tyk
/apis/<tyk_api_id>', methods = ['GET'])

```

```

def get_tyk_api_by_id(connection_id,
tyk_api_id):
    res = get_api_by_id(connection_id,
tyk_api_id)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    return jsonify({'content':
dictFromServer, "response":
res.status_code})

#create api
@app.route('/connections/<connection_id>/tyk
/apis', methods = ['POST'])
def create_tyk_api(connection_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )

    tyk_apis = get_tyk_apis(connection_id)
    print(tyk_apis, flush = True )

    print(request.json, flush = True )

    if ('name' in request.json) and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('api_id' in request.json) and
(request.json['api_id'] != ''):
        if tyk_apis:
            for item in tyk_apis:
                if(request.json['name'] !=
item['api_id']):
                    api_id =
request.json['name']
                else:
                    return
                    jsonify({"response": "fail", "message":
"Please provide another name!"})
            else:
                api_id = request.json['name']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('proxy' in request.json) and
(request.json['proxy'] != ''):
        proxy = request.json['proxy']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a proxy!"})

    if ('listen_path' in
request.json['proxy']) and
(request.json['proxy']['listen_path'] !=
''):
        if tyk_apis:
            for item in tyk_apis:
                if(request.json['proxy']['listen_path'] !=
item['proxy']['listen_path']):
                    listen_path =
request.json['proxy']['listen_path']

```

```

        else:
            return
            jsonify({"response": "fail", "message":
"Please provide another route_path!"})
        else:
            listen_path =
request.json['proxy']['listen_path']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a route_path!"})

        if ('target_url' in
request.json['proxy']) and
(request.json['proxy']['target_url'] != ''):
            target_url =
request.json['proxy']['target_url']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a target_url!"})

        if ('name' in request.json) and
('api_id' in request.json) and ('proxy' in
request.json) and ('listen_path' in
request.json['proxy']) and ('target_url' in
request.json['proxy']) and request.method ==
'POST':
            dictToSend = {
                "name": name,
                "api_id": api_id,
                "org_id": "default",
                "definition": {
                    "location": "header",
                    "key": "version"
                },
                "use_keyless": True,
                "version_data": {
                    "not_versioned": True,
                    "versions": {
                        "Default": {
                            "name": "Default"
                        }
                    }
                },
                "custom_middleware": {
                    "pre": [
                        {
                            "name":
"testJSVMData",
                            "path":
"./middleware/injectHeader.js",
                            "require_session":
False,
                            "raw_body_only":
False
                        }
                    ]
                },
                "driver": "otto",
                "proxy": {
                    "listen_path": listen_path,
                    "target_url": target_url,
                    "strip_listen_path": True
                }
            }

            res =
requests.post(connection_admin_api_url +
'/tyk/apis', json = dictToSend,

```

```

headers={"x-tyk-authorization":"352d20ee67be
67f6340b4c0605b044b7"})
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    reload(connection_id)

    return jsonify({'content':
dictFromServer, "response":
res.status_code})

#update api
@app.route('/connections/<connection_id>/tyk
/apis/<tyk_api_id>', methods = ['PUT'])
def update_tyk_api(connection_id,
tyk_api_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )
    print("request.json", flush = True )
    print(request.json, flush = True )

    tyk_apis = get_tyk_apis(connection_id)
    print(tyk_apis, flush = True )

    resp = get_api_by_id(connection_id,
tyk_api_id)
    current_api = resp.json()
    print(current_api, flush = True )

    if ('name' in request.json) and
(request.json['name'] != ''):
        name = request.json['name']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('api_id' in request.json) and
(request.json['api_id'] != ''):
        if(request.json['api_id'] ==
tyk_api_id):
            api_id = request.json['api_id']
        else:
            return jsonify({"response":
"fail", "message": "Please don't modify the
name!"})
    else:
        return jsonify({"response": "fail",
"message": "Please provide a name!"})

    if ('proxy' in request.json) and
(request.json['proxy'] != ''):
        proxy = request.json['proxy']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a proxy!"})

    if ('listen_path' in
request.json['proxy']) and
(request.json['proxy']['listen_path'] !=
''):
        if tyk_apis:
            for item in tyk_apis:
                print("item", flush = True )
                print(item, flush = True )
                if(item != current_api):

```

```

if(request.json['proxy']['listen_path'] !=
item['proxy']['listen_path']):
            listen_path =
request.json['proxy']['listen_path']
        else:
            return
            jsonify({"response": "fail", "message":
"Please provide another listen_path!"})
        else:
            listen_path =
request.json['proxy']['listen_path']
        else:
            listen_path =
request.json['proxy']['listen_path']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a listen_path!"})

        if ('target_url' in
request.json['proxy']) and
(request.json['proxy']['target_url'] != ''):
            target_url =
request.json['proxy']['target_url']
        else:
            return jsonify({"response": "fail",
"message": "Please provide a target_url!"})

        if ('name' in request.json) and
('api_id' in request.json) and ('proxy' in
request.json) and ('listen_path' in
request.json['proxy']) and ('target_url' in
request.json['proxy']) and request.method ==
'PUT':
            dictToSend = {
                "name": name,
                "api_id": api_id,
                "org_id": "default",
                "definition": {
                    "location": "header",
                    "key": "version"
                },
                "use_keyless": True,
                "version_data": {
                    "not_versioned": True,
                    "versions": {
                        "Default": {
                            "name": "Default"
                        }
                    }
                },
                "custom_middleware": {
                    "pre": [
                        {
                            "name":
"testJSVMData",
                            "path":
"./middleware/injectHeader.js",
                            "require_session":
False,
                            "raw_body_only":
False
                        }
                    ]
                },
                "driver": "otto",
                "proxy": {
                    "listen_path": listen_path,
                    "target_url": target_url,

```

```

        "strip_listen_path": True
    }
}

res =
requests.put(connection_admin_api_url +
'/tyk/apis/' + tyk_api_id, json =
dictToSend,
headers={"x-tyk-authorization":"352d20ee67be
67f6340b4c0605b044b7"})
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    reload(connection_id)

    return jsonify({'content':
dictFromServer, "response":
res.status_code})

#delete api
@app.route('/connections/<connection_id>/tyk
/apis/<tyk_api_id>', methods = ['DELETE'])
def delete_tyk_api_by_id(connection_id,
tyk_api_id):
    connection_admin_api_url =
get_connection_by_ID(connection_id)['admin_a
pi_url']
    print(connection_admin_api_url, flush =
True )
    print("ALO", flush = True )

    res =
requests.delete(connection_admin_api_url +
'/tyk/apis/' + tyk_api_id,
headers={"x-tyk-authorization":"352d20ee67be
67f6340b4c0605b044b7"})
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    reload(connection_id)

    return jsonify({'content':
dictFromServer, "response":
res.status_code})

#get all connection_tyk_apis dependencies
@app.route('/connection_tyk_apis', methods =
['GET'])
def get_tyk_apis_dependency():
    tyk_apis = mongo.db.connection_tyk_apis
#the collection i have in my db
    output = []

    for query in tyk_apis.find():
#find all the documents in the collection
        output.append({'_id':
str(query['_id']), 'tyk_api_id':
query['tyk_api_id'], 'connection_id':
query['connection_id'] })

    return jsonify({'tyk_apis': output})

#create connection-tyk_api dependency
@app.route('/connection_tyk_apis', methods =
['POST'])
def add_tyk_api_dependency():
    tyk_apis = mongo.db.connection_tyk_apis

    if ('tyk_api_id' in request.json) and
(request.json['tyk_api_id'] != ''):

```

```

        tyk_api_id =
request.json['tyk_api_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a tyk_api_id!"})

    if ('connection_id' in request.json) and
(request.json['connection_id'] != ''):
        connection_id =
request.json['connection_id']
    else:
        return jsonify({"response": "fail",
"message": "Please provide a
connection_id!"})

    if ('tyk_api_id' in request.json) and
('connection_id' in request.json) and
request.method == 'POST':
        connection_tyk_api_id =
tyk_apis.insert_one({'tyk_api_id':
tyk_api_id, 'connection_id':
connection_id}).inserted_id    #generates a
random id
        new_tyk_api =
tyk_apis.find_one({'_id':
connection_tyk_api_id})
        #output = {'tyk_api_id':
new_tyk_api['tyk_api_id'], 'connection_id':
new_tyk_api['connection_id']}

        return jsonify({"response" :
"success", "message": 'connection_tyk_api
dependency created successfully!'})

#get connection_tyk_api dependency where id
tyk_api is...
@app.route('/connection_tyk_apis/<tyk_api_id
>', methods = ['GET'])
def get_connection_tyk_api(tyk_api_id):
    connections =
mongo.db.connection_tyk_apis

    query =
connections.find_one({'tyk_api_id':
tyk_api_id})
    if query:
        output = {'_id': str(query['_id']),
'tyk_api_id': query['tyk_api_id'],
'connection_id': query['connection_id']}
    else:
        output = 'No results found'
    return jsonify({"dependency": output})

#delete a connection_tyk_api dependency
@app.route('/connection_tyk_apis/<id>',
methods = ['DELETE'])
def delete_tyk_api_dependency(id):

mongo.db.connection_tyk_apis.delete_one({'_i
d': ObjectId(id)})
    return jsonify({"response" : "succes",
"message" : "connection_tyk_api dependency
deleted successfully"})

```

```
#KUBERNETES/DOCKER DEPLOYMENT
```

```

@app.route('/kubernetes/<connection_type>',
methods = ['POST'])
def deployment(connection_type):
    file = ""
    if( connection_type == "kong"):
        file = "kong.yaml"

        with open(file) as kong_yaml:
#deschide fisierul in variabila kong_yaml de
tipul fisier object
            temp_kong = kong_yaml.read()
#string -> e o lista de linii, fiecare linie
e un string
            contor = mongo.db.contor
            contorKong =
contor.find_one({"_id":
ObjectId("62b217c06b7f9c0008198ac3"))
            print(contorKong, flush = True )
            contor_kong =
contorKong['contor_kong']
            print(contor_kong, flush = True
)

            if contor_kong == 1000:
                return jsonify({"response" :
"fail", "message" : "Out of limit!"})

print("\n\n-----\n\n",
flush = True )

            temp_kong =
temp_kong.replace("{}pg_port}", str(40000 +
contor_kong))

            temp_kong =
temp_kong.replace("{}port1}", str(41000 +
contor_kong) )

            temp_kong =
temp_kong.replace("{}port2}", str(42000 +
contor_kong) )

            temp_kong =
temp_kong.replace("{}port3}", str(43000 +
contor_kong) )

            temp_kong =
temp_kong.replace("{}port4}", str(44000 +
contor_kong) )

            print(temp_kong, flush = True )
            new_file = str(contor_kong)+file

            with open(new_file, "w") as
new_kong_yaml:

new_kong_yaml.write(temp_kong)

            os.system('./kubect1 create
namespace kong' + str(contor_kong))
            os.system('./kubect1 apply -f '
+ new_file + " -n kong" + str(contor_kong))

            port = 43000 + contor_kong

mongo.db.contor.update_one({'_id':
ObjectId("62b217c06b7f9c0008198ac3")),
{"$inc": {'contor_kong': 1}} ) # ->
increment the value

            return jsonify({"response" :
"succes", "message" : "", "port": str(port),
"contor": str(contor_kong)})

```

```

elif( connection_type == "tyk"):
    file = "run_tyk.sh"
    with open(file) as tyk_yaml:
#deschide fisierul in variabila tyk_yaml de
tipul fisier object
        temp_tyk = tyk_yaml.read()
#string
        contor = mongo.db.contor
        contorTyk =
contor.find_one({"_id":
ObjectId("62b217c06b7f9c0008198ac3"))
        print(contorTyk, flush = True )
        contor_tyk =
contorTyk['contor_tyk']
        print(contor_tyk, flush = True )
        if contor_tyk == 1000:
            return jsonify({"response" :
"fail", "message" : "Out of limit!"})

print("\n\n-----\n\n",
flush = True )

        temp_tyk =
temp_tyk.replace("{}port1}", str(44000 +
contor_tyk) )

        #temp_tyk =
temp_tyk.replace("{}redis_port}", str(58000
+ contor_tyk) )

        print(temp_tyk, flush = True )
        new_file = str(contor_tyk)+file

        with open(new_file, "w") as
new_tyk_yaml:
            new_tyk_yaml.write(temp_tyk)

            os.chmod(new_file, 0o0777) #il
face exec ii da toate drepturile
            os.system('./' + new_file)

            port = 44000 + contor_tyk

mongo.db.contor.update_one({'_id':
ObjectId("62b217c06b7f9c0008198ac3")),
{"$inc": {'contor_tyk': 1}} ) # -> increment
the value

            return jsonify({"response" :
"succes", "message" : "", "port": str(port),
"contor": str(contor_tyk)})

            return jsonify({"response" : "fail",
"message" : "Something went wrong!"})

if __name__ == '__main__':
    app.run(debug = False, host='0.0.0.0')

```

## /api1/Dockerfile

FROM python:3.10.2

RUN pip install flask  
RUN pip install flasgger

WORKDIR /opt/app  
COPY . /opt/app/

```
ENTRYPOINT ["python3", "./server.py"]
```

## /api1/run.sh

```
docker build -t gateway/api1 .
docker rm -f gw-api1 || true
docker network create gw || true
```

```
docker run -d \
--net gw \
--name gw-api1 \
-p 5001:5000 \
gateway/api1
```

## /api1/server.py

```
from flask import *
from flasgger import Swagger
```

```
app = Flask(__name__)
swagger = Swagger(app)
```

```
books = [
    {
        "title": "Two Years' Vacation",
        "book_id": "0",
        "author": "Jules Verne",
        "year_written": "1888",
        "edition": "Pierre-Jules Hetzel",
        "price": "30.7"
    },
    {
        "title": "Anna Karenina",
        "book_id": "1",
        "author": "Leo Tolstoy",
        "year_written": "1878",
        "edition": "The Russian Messenger",
        "price": "25.5"
    },
    {
        "title": "Tom Sawyer",
        "book_id": "2",
        "author": "Mark Twain",
        "year_written": "1876",
        "edition": "American Publishing
Company",
        "price": "37.75"
    },
    {
        "title": "Harry Potter",
        "book_id": "3",
        "author": "J.K. Rowling",
        "year_written": "2000",
        "edition": "Harcourt Brace",
        "price": "19.95"
    }
]
```

```
@app.route('/')
def home():
    """Endpoint to greet the user
    ---
    definitions:
        Message_to_greet:
            type: string
```

```

    responses:
        200:
            description: Returns a welcome
            message
            schema:
                $ref:
                '#/definitions/Message_to_greet'
            examples:
                application/json: |
                    Welcome to the book club!
    """
    return "Welcome to the book club!"

@app.route("/books", methods= ['GET'])
def get_book():
    """Endpoint to get all books
    ---
    definitions:
        Message_to_get_books:
            type: object
            properties:
                Books:
                    type: object
    responses:
        200:
            description: Returns a dictionary
            with all books
            schema:
                $ref:
                '#/definitions/Message_to_get_books'
            examples:
                application/json: |
                    {
                        Books: [
                            {
                                "title": "Hamlet,
                                Prince of Denmark",
                                "book_id": 7,
                                "author":
                                "Shakespeare",
                                "year_written":
                                1603,
                                "edition": "Signet
                                Classics",
                                "price": 7.95
                            },
                            {
                                "author":
                                "Tolstoy, Leo",
                                "book_id": 0,
                                "edition":
                                "Penguin",
                                "price": 12.7,
                                "title": "War and
                                Peace",
                                "year_written":
                                1865
                            }
                        ]
                    }
    """
    return jsonify({'Books' : books})

@app.route("/books/<string:id>", methods =
['GET'])
def get_book_by_id(id):
    """Endpoint to get a book by id
    ---
    parameters:
        - name: id
```



```

        in: path
        type: string
        required: true
definitions:
    Message_to_get_by_id:
        type: object
        properties:
            Book:
                type: object
responses:
    200:
        description: Returns a dictionary
with the specific book
        schema:
            $ref:
'#/definitions/Message_to_get_by_id'
        examples:
            application/json: |
                {
                    Book: {
                        "title": "Hamlet,
Prince of Denmark",
                        "book_id": "7",
                        "author":
"Shakespeare",
                        "year_written": 1603,
                        "edition": "Signet
Classics",
                        "price": "7.95"
                    }
                }
"""
    book = [book for book in books if
book['book_id'] == id]
    if len(book) == 0:
        return jsonify({'error': 'Not
found'})
    return jsonify({"Book" : book[0]})

@app.route("/books", methods = ['POST'])
def create():
    """Endpoint for creating a book
    ---
    definitions:
        Message_to_create:
            type: object
            properties:
                new:
                    type: object
    responses:
        200:
            description: Returns a dictionary
with the new object
            schema:
                $ref:
'#/definitions/Message_to_create'
            examples:
                application/json: |
                    {
                        new: {
                            "title": "Hamlet,
Prince of Denmark",
                            "book_id": "7",
                            "author":
"Shakespeare",
                            "year_written": 1603,
                            "edition": "Signet
Classics",
                            "price": "7.95"
                        }
                    }

```

```

        }
        """
        book = request.get_json()
        for item in books:
            if(item['book_id'] ==
book['book_id']):
                return jsonify({'error': 'Please
provide another book_id'})

        books.append(book)

        return jsonify({"New" : book})

@app.route("/books/<string:id>", methods =
['PUT'])
def update(id):
    """Endpoint for updating a book by id
    ---
    parameters:
        - name: id
          in: path
          type: string
          required: true
    definitions:
        Message_to_update:
            type: object
            properties:
                updated:
                    type: object
    responses:
        200:
            description: Returns a dictionary
with the updated object
            schema:
                $ref:
'#/definitions/Message_to_update'
            examples:
                application/json: |
                    {
                        updated: {
                            "title": "Hamlet,
Prince of Denmark",
                            "book_id": "7",
                            "author":
"Shakespeare",
                            "year_written":
1603,
                            "edition": "Signet
Classics",
                            "price": "7.95"
                        }
                    }
        """
        book = [book for book in books if
book['book_id'] == id]
        if len(book) == 0:
            return jsonify({'error': 'Not
found'})
        if 'title' in request.json and
request.json['title'] != "":
            book[0]['title'] =
request.json.get('title', book[0]['title'])
            if 'author' in request.json and
request.json['author'] != "":
                book[0]['author'] =
request.json.get('author',
book[0]['author'])
            if 'year_written' in request.json and
request.json['year_written'] != "":

```

```

        book[0]['year_written'] =
request.json.get('year_written',
book[0]['year_written'])
        if 'edition' in request.json and
request.json['edition'] != "":
            book[0]['edition'] =
request.json.get('edition',
book[0]['edition'])
            if 'price' in request.json and
request.json['price'] != "":
                book[0]['price'] =
request.json.get('price', book[0]['price'])

```

```

        return jsonify({'updated': book[0]})

```

```

@app.route("/books/<string:id>", methods =
['DELETE'])

```

```

def delete(id):
    """Endpoint for deleting a book by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_delete:
        type: object
        properties:
          result:
            type: boolean
    responses:
      200:
        description: A message to validate
the result of the delete operation
        schema:
          $ref:
            '#/definitions/Message_to_delete'
    examples:
      application/json: |
        {"result": True}
    """
    book = [book for book in books if
book['book_id'] == id]
    if len(book) == 0:
        return jsonify({'error': 'Not
found'})
    books.remove(book[0])
    return jsonify({'result' : True})

```

```

if __name__ == "__main__":
    app.run(debug = True, host='0.0.0.0')

```

## /api2/Dockerfile

```

FROM python:3.10.2

```

```

RUN pip install flask
RUN pip install flasgger

```

```

WORKDIR /opt/app
COPY . /opt/app/

```

```

ENTRYPOINT ["python3", "./server.py"]

```

## /api2/run.sh

```

docker build -t gateway/api2 .
docker rm -f gw-api2 || true
docker network create gw || true

```

```

docker run -d \
--net gw \
--name gw-api2 \
-p 5002:5000 \
gateway/api2

```

## /api2/server.py

```

from flask import *
from flasgger import Swagger

```

```

app = Flask(__name__)
swagger = Swagger(app)

```

```

cars = [
    {
        "manufacturer" : "Porsche",
        "car_id" : "0",
        "model" : "911",
        "price" : "135000",
        "colour" : "red",
        "owner" : "Lauren"
    },
    {
        "manufacturer" : "Nissan",
        "car_id" : "1",
        "model" : "GT-R",
        "price" : "80000",
        "colour" : "green",
        "owner" : "Loren"
    },
    {
        "manufacturer" : "BMW",
        "car_id" : "2",
        "model" : "M3",
        "price" : "60500",
        "colour" : "black",
        "owner" : "Xander"
    },
    {
        "manufacturer" : "Audi",
        "car_id" : "3",
        "model" : "S5",
        "price" : "53000",
        "colour" : "blue",
        "owner" : "Alex"
    },
    {
        "manufacturer" : "Audi",
        "car_id" : "4",
        "model" : "TT",
        "price" : "40000",
        "colour" : "black",
        "owner" : "Alexa"
    }
]

```

```

@app.route('/')
def home():
    """Endpoint to greet the user
    ---
    definitions:

```

```

        Message_to_greet:
            type: string
        responses:
            200:
                description: Returns a welcome
message
        schema:
            $ref:
'#/definitions/Message_to_greet'
        examples:
            application/json: |
                Welcome to the car club!
        """
        return "Welcome to the car club!"

@app.route("/cars", methods= ['GET'])
def get_car():
    """Endpoint to get all cars
    ---
    definitions:
        Message_to_get_cars:
            type: object
            properties:
                Cars:
                    type: object
            responses:
                200:
                    description: Returns a dictionary
with all cars
                    schema:
                        $ref:
'#/definitions/Message_to_get_cars'
                    examples:
                        application/json: |
                            {
                                Cars: [
                                    {
                                        "manufacturer" :
"Nissan",
                                        "car_id" : "1",
                                        "model" : "GT-R",
                                        "price" : "80000",
                                        "colour" : "black",
                                        "owner" : "Loren"
                                    },
                                    {
                                        "manufacturer" :
"BMW",
                                        "car_id" : "2",
                                        "model" : "X3",
                                        "price" : "50000",
                                        "colour" : "black",
                                        "owner" : "Dan"
                                    }
                                ]
                            }
                        """
        return jsonify({'Cars' : cars})

@app.route("/cars/<string:id>", methods =
['GET'])
def get_car_by_id(id):
    """Endpoint to get a car by id
    ---
    parameters:
        - name: id
          in: path
          type: string
          required: true
    definitions:

```

```

        Message_to_get_by_id:
            type: object
            properties:
                Car:
                    type: object
            responses:
                200:
                    description: Returns a dictionary
with the specific car
                    schema:
                        $ref:
'#/definitions/Message_to_get_by_id'
                    examples:
                        application/json: |
                            {
                                Car: {
                                    "manufacturer" :
"Nissan",
                                    "car_id" : "1",
                                    "model" : "GT-R",
                                    "price" : "80000",
                                    "colour" : "black",
                                    "owner" : "Loren"
                                }
                            }
                        """
        car = [car for car in cars if
car['car_id'] == id]
        if len(car) == 0:
            return jsonify({'error': 'Not
found'})
        return jsonify({"Car" : car[0]})

@app.route("/cars", methods = ['POST'])
def create():
    """Endpoint for creating a car
    ---
    definitions:
        Message_to_create:
            type: object
            properties:
                new:
                    type: object
            responses:
                200:
                    description: Returns a dictionary
with the new object
                    schema:
                        $ref:
'#/definitions/Message_to_create'
                    examples:
                        application/json: |
                            {
                                new: {
                                    "manufacturer" :
"Nissan",
                                    "car_id" : "1",
                                    "model" : "GT-R",
                                    "price" : "80000",
                                    "colour" : "black",
                                    "owner" : "Loren"
                                }
                            }
                        """
        car = request.get_json()
        for item in cars:
            if(item['car_id'] == car['car_id']):
                return jsonify({'error': 'Please
provide another car_id'})

```

```

cars.append(car)

return jsonify({"New" : car})

@app.route("/cars/<string:id>", methods =
['PUT'])
def update(id):
    """Endpoint for updating a car by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_update:
        type: object
        properties:
          updated:
            type: object
    responses:
      200:
        description: Returns a dictionary
        with the updated object
        schema:
          $ref:
            '#/definitions/Message_to_update'
        examples:
          application/json: |
            {
              updated: {
                "Nissan",
                "car_id" : 1,
                "model" : "GT-R",
                "price" : 80000,
                "colour" :
                "black",
                "owner" : "Loren"
              }
            }
    """
    car = [car for car in cars if
car['car_id'] == id]
    if len(car) == 0:
        return jsonify({'error': 'Not
found'})
    if 'manufacturer' in request.json and
request.json['manufacturer'] != "":
        car[0]['manufacturer'] =
request.json.get('manufacturer',
car[0]['manufacturer'])
    if 'model' in request.json and
request.json['model'] != "":
        car[0]['model'] =
request.json.get('model', car[0]['model'])
    if 'price' in request.json and
request.json['price'] != "":
        car[0]['price'] =
request.json.get('price', car[0]['price'])
    if 'colour' in request.json and
request.json['colour'] != "":
        car[0]['colour'] =
request.json.get('colour', car[0]['colour'])
    if 'owner' in request.json and
request.json['owner'] != "":
        car[0]['owner'] =
request.json.get('owner', car[0]['owner'])

    return jsonify({'updated': car[0]})

```

```

@app.route("/cars/<string:id>", methods =
['DELETE'])
def delete(id):
    """Endpoint for deleting a car by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_delete:
        type: object
        properties:
          result:
            type: boolean
    responses:
      200:
        description: A message to validate
        the result of the delete operation
        schema:
          $ref:
            '#/definitions/Message_to_delete'
        examples:
          application/json: |
            {"result": True}
    """
    car = [car for car in cars if
car['car_id'] == id]
    if len(car) == 0:
        return jsonify({'error': 'Not
found'})
    cars.remove(car[0])
    return jsonify({'result' : True})

if __name__ == "__main__":
    app.run(debug = True, host='0.0.0.0')

```

## /api3/Dockerfile

FROM python:3.10.2

RUN pip install flask

RUN pip install flasgger

WORKDIR /opt/app

COPY . /opt/app/

ENTRYPOINT ["python3", "./server.py"]

## /api3/run.sh

```

docker build -t gateway/api3 .
docker rm -f gw-api3 || true
docker network create gw || true

```

```

docker run -d \
--net gw \
--name gw-api3 \
-p 5003:5000 \
gateway/api3

```

### /api3/server.py

```
from flask import *
from flasgger import Swagger

app = Flask(__name__)
swagger = Swagger(app)

pokemons = [
    {
        "name" : "piplup",
        "poke_id" : "0",
        "type" : "water",
        "region" : "Sinnoh",
        "trainer" : "Dawn"
    },
    {
        "name" : "chimchar",
        "poke_id" : "1",
        "type" : "fire",
        "region" : "Sinnoh",
        "trainer" : "Paul"
    },
    {
        "name" : "pikachu",
        "poke_id" : "2",
        "type" : "electric",
        "region" : "Kanto",
        "trainer" : "Ash"
    },
    {
        "name" : "roselia",
        "poke_id" : "3",
        "type" : "grass",
        "region" : "Hoen",
        "trainer" : "Drew"
    }
]

@app.route('/')
def home():
    """Endpoint to greet the user
    ---
    definitions:
      Message_to_greet:
        type: string
    responses:
      200:
        description: Returns a welcome
        message
        schema:
          $ref:
            '#/definitions/Message_to_greet'
    examples:
      application/json: |
        Welcome to the pokemon club!
    """
    return "Welcome to the pokemon club!"

@app.route("/pokemons", methods= ['GET'])
def get_poke():
    """Endpoint to get all pokemons
    ---
    definitions:
      Message_to_get_pokemons:
        type: object
        properties:
          Pokemons:
            type: object
    responses:
```

```
200:
  description: Returns a dictionary
  with all pokemons
  schema:
    $ref:
      '#/definitions/Message_to_get_pokemons'
  examples:
    application/json: |
      {
        Pokemons: [
          {
            "name": "aron",
            "poke_id": "1",
            "region":
              "Johto",
            "trainer":
              "Paul",
            "type": "steel"
          },
          {
            "name":
              "piplup",
            "poke_id": "2",
            "region":
              "Sinnoh",
            "trainer":
              "Dawn",
            "type": "water"
          }
        ]
      }
    """
    return jsonify({'Pokemons' : pokemons})

@app.route("/pokemons/<string:id>", methods
= ['GET'])
def get_poke_by_id(id):
    """Endpoint to get a pokemon by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_get_by_id:
        type: object
        properties:
          Pokemon:
            type: object
    responses:
      200:
        description: Returns a dictionary
        with the specific pokemon
        schema:
          $ref:
            '#/definitions/Message_to_get_by_id'
    examples:
      application/json: |
        {
          Pokemon: {
            "name": "aron",
            "poke_id": "9",
            "region": "Johto",
            "trainer": "Paul",
            "type": "steel"
          }
        }
    """
```

```

    pokemon = [pokemon for pokemon in
pokemons if pokemon['poke_id'] == id]
    if len(pokemon) == 0:
        return jsonify({'error': 'Not
found'})
    return jsonify({"Pokemon" : pokemon[0]})

@app.route("/pokemons", methods = ['POST'])
def create():
    """Endpoint for creating a pokemon
    ---
    definitions:
      Message_to_create:
        type: object
        properties:
          new:
            type: object
    responses:
      200:
        description: Returns a dictionary
with the new object
        schema:
          $ref:
            '#/definitions/Message_to_create'
    examples:
      application/json: |
        {
          new: {
            "name": "aron",
            "poke_id": "9",
            "region": "Johto",
            "trainer": "Paul",
            "type": "steel"
          }
        }
    """
    pokemon = request.get_json()
    for item in pokemons:
        if(item['poke_id'] ==
pokemon['poke_id']):
            return jsonify({'error': 'Please
provide another poke_id'})

    pokemons.append(pokemon)

    return jsonify({"New" : pokemon})

@app.route("/pokemons/<string:id>", methods
= ['PUT'])
def update(id):
    """Endpoint for updating a pokemon by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_update:
        type: object
        properties:
          updated:
            type: object
    responses:
      200:
        description: Returns a dictionary
with the updated object
        schema:
          $ref:
            '#/definitions/Message_to_update'

```

```

examples:
  application/json: |
    {
      updated: {
        "name": "aron",
        "poke_id": "0",
        "region": "Johto",
        "trainer": "Paul",
        "type": "steel"
      }
    }
    """
    pokemon = [pokemon for pokemon in
pokemons if pokemon['poke_id'] == id]
    if len(pokemon) == 0:
        return jsonify({'error': 'Not
found'})
    if 'name' in request.json and
request.json['name'] != "":
        pokemon[0]['name'] =
request.json.get('name', pokemon[0]['name'])
    if 'region' in request.json and
request.json['region'] != "":
        pokemon[0]['region'] =
request.json.get('region',
pokemon[0]['region'])
    if 'trainer' in request.json and
request.json['trainer'] != "":
        pokemon[0]['trainer'] =
request.json.get('trainer',
pokemon[0]['trainer'])
    if 'type' in request.json and
request.json['type'] != "":
        pokemon[0]['type'] =
request.json.get('type', pokemon[0]['type'])

    return jsonify({'updated': pokemon[0]})

@app.route("/pokemons/<string:id>", methods
= ['DELETE'])
def delete(id):
    """Endpoint for deleting a pokemon by id
    ---
    parameters:
      - name: id
        in: path
        type: string
        required: true
    definitions:
      Message_to_delete:
        type: object
        properties:
          result:
            type: boolean
    responses:
      200:
        description: A message to validate
the result of the delete operation
        schema:
          $ref:
            '#/definitions/Message_to_delete'
    examples:
      application/json: |
        {"result": True}
    """
    pokemon = [pokemon for pokemon in
pokemons if pokemon['poke_id'] == id]
    if len(pokemon) == 0:
        return jsonify({'error': 'Not
found'})

```

```

    pokemons.remove(pokemon[0])
    return jsonify({'result' : True})

if __name__ == "__main__":
    app.run(debug = True, host='0.0.0.0')

```

### /db/run-db.sh

```

docker network create gw || true

docker rm -f mongoddb || true
docker run -d -p 27018:27017 \
  --name mongoddb \
  --network gw \
  -e "MONGO_INITDB_ROOT_USERNAME: mongo" \
  -e "MONGO_INITDB_ROOT_PASSWORD: mongo" \
  -v /my/own/datadir:/data/db \
  mongo

```

### /frontend/frontend/routes/\_init.py

```

from . import index
from . import login
from . import logout
from . import home
from . import register
from . import services
from . import routes
from . import users
from . import connections
from . import services_from_connection
from . import routes_from_connection
from . import deployments
from . import deployment_for_connection
from . import test

```

```

def start(app):
    index.start(app)
    login.start(app)
    logout.start(app)
    home.start(app)
    register.start(app)
    services.start(app)
    routes.start(app)
    users.start(app)
    connections.start(app)
    services_from_connection.start(app)
    routes_from_connection.start(app)
    deployments.start(app)
    deployment_for_connection.start(app)
    test.start(app)

```

### /frontend/frontend/routes/connections.py

```

import json, os, sys
import requests
from flask import render_template, request,
session, redirect
from flask_session import Session

```

```

def start(app):
    @app.route('/connections')
    def connections_route():
        session_data = {
            'user': session.get('user', ''),

```

```

            'logged_in': session.get('user', '')
        }

        res =
        requests.get('http://api_management:5000/users/' + session['user_id'] + '/connections')
        dictFromServer = res.json()
        print(dictFromServer, flush = True)
        return
        render_template('connections.html',
            session_data=session_data,
            dictFromServer=dictFromServer)

        @app.route('/create-connection',
            methods=['POST'])
        def do_create_connection():
            print(request.form['name'])
            print("We're on", flush = True)

            #create connection
            dictToSend = {"name":
                request.form['name'], "type":
                request.form['type'], "admin_api_url":
                request.form['admin_api_url']}
            res =
            requests.post('http://api_management:5000/users/' + session['user_id'] + '/connections',
                json = dictToSend)
            dictFromServer = res.json()
            print(dictFromServer, flush = True)

            if dictFromServer['response'] ==
            "success":
                #get connection id
                connection_id =
                dictFromServer['new_connection_id']

                #create the user_connection dependency
                dictToSend_dependency = {"user_id":
                    session['user_id'], "connection_id":
                    connection_id}
                res =
                requests.post('http://api_management:5000/user_connections', json =
                    dictToSend_dependency)
                dictFromServer_dependency = res.json()
                print(dictFromServer_dependency, flush
                    = True)

                if
                dictFromServer_dependency['response'] ==
                "success":
                    return redirect('/connections')
                else:
                    return redirect('/?message=' +
                        "Something went wrong!")

                elif dictFromServer['response'] ==
                "fail":
                    message = dictFromServer['message']
                    return redirect('/?message=' + message
                )
                else:
                    return redirect('/?message=' +
                        "Something went wrong!")

            @app.route('/update-connection',
                methods=['POST'])

```

```

def do_update_connection():
    print(request.form['id'])
    print("We're on update", flush = True )

    dictToSend = {"name":
request.form['name'], "type":
request.form['type'], "admin_api_url":
request.form['admin_api_url']}
    res =
requests.put('http://api_management:5000/users/' + session['user_id'] + '/connections/'
+ request.form['id'], json = dictToSend)
    dictFromServer = res.json()
    print(dictFromServer, flush = True)

    if dictFromServer['response'] ==
"success":
        return redirect('/connections')
    elif dictFromServer['response'] ==
"fail":
        message = dictFromServer['message']
        return redirect('/?message=' + message
)
    else:
        return redirect('/?message=' +
"Something went wrong!" )

@app.route('/delete-connection',
methods=['POST'])
def delete_connection():

    print("We're on delete", flush = True)
    connection_id = request.form['id']
    print(connection_id, flush = True)

    #get dependency id
    get_dependency_by_connectionsID =
requests.get('http://api_management:5000/user_connections/' + connection_id)
    dependency_id =
get_dependency_by_connectionsID.json()['dependency']['_id']
    print(connection_id, flush = True)

    #delete the route_connection dependency
    res =
requests.delete('http://api_management:5000/user_connections/' + dependency_id)
    dictFromServer_dependency = res.json()
    print(dictFromServer_dependency, flush =
True)

    #delete connections
    res =
requests.delete('http://api_management:5000/connections/' + connection_id)
    dictFromServer = res.json()
    print(dictFromServer, flush = True)

    return redirect('/connections')

```

## **/frontend/frontend/routes/deployment\_for\_connecti on.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session, redirect

```

```

from flask_session import Session

def start(app):

@app.route('/deployment-for/<connection_type>')
    def routes_deployment(connection_type):
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        return
render_template('deployment_for_connection.h
tml', session_data=session_data,
connection_type=connection_type,
created="No")

@app.route('/deployment/<connection_type>',
methods=['POST'])
    def deployment(connection_type):
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        res =
requests.post('http://api_management:5000/ku
bernetes/' + connection_type)
        response = res.json()
        print(response, flush = True )
        if response['response'] == "fail":
            return redirect('/?message=' +
"Something went wrong!" )
        port = response['port']
        print(port, flush = True )
        cnt = response['contor']
        print(cnt, flush = True )

        #create the connection
        dictToSend = {"name": connection_type +
"Nr" + cnt, "type": connection_type,
"admin_api_url":
"http://host.docker.internal:" + port}
        res =
requests.post('http://api_management:5000/us
ers/' + session['user_id'] + '/connections',
json = dictToSend)
        dictFromServer = res.json()
        print(dictFromServer, flush = True)

        if dictFromServer['response'] ==
"success":
            #get connection id
            connection_id =
dictFromServer['new_connection_id']

            #create the user_connection
            dependency
            dictToSend_dependency = {"user_id":
session['user_id'], "connection_id":
connection_id}
            res =
requests.post('http://api_management:5000/us
er_connections', json =
dictToSend_dependency)

```



```

        dictFromServer_dependency =
res.json()
        print(dictFromServer_dependency,
flush = True)

        if
dictFromServer_dependency['response'] ==
"success":
            return
render_template('deployment_for_connection.h
tml', session_data=session_data,
connection_type=connection_type,
created="Yes")
        else:
            return redirect('/?message=' +
"Something went wrong!")

        elif dictFromServer['response'] ==
"fail":
            message = dictFromServer['message']
            return redirect('/?message=' +
message )
        else:
            return redirect('/?message=' +
"Something went wrong!")

```

#### **/frontend/frontend/routes/deployments.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session, redirect
from flask_session import Session

def start(app):
    @app.route('/deployments')
    def routes_deployments():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        return
render_template('deployments.html',
session_data=session_data )

```

#### **/frontend/frontend/routes/home.py**

```

import json, os, sys
from flask import render_template, request,
session
from flask_session import Session

def start(app):
    @app.route('/home')
    def home_route():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        return render_template('home.html',
session_data=session_data)

```

#### **/frontend/frontend/routes/index.py**

```

import json, os, sys
from flask import render_template, request,
session
from flask_session import Session

def start(app):
    @app.route('/')
    def index_route():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        message = request.args.get('message')
        args = request.args
        message = args.get('message')

        if message is not None:
            return render_template('index.html',
session_data=session_data, message=message)
        else:
            return render_template('index.html',
session_data=session_data, message='')

```

#### **/frontend/frontend/routes/login.py**

```

import json, os, sys
import base64
import requests
from flask import render_template, request,
session, redirect
from flask_session import Session

def start(app):
    @app.route('/login')
    def login_route():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        #iau userul de pe sesiunea din Flask
        #creez variabila logged_in care
        #verifica ca userul exista, nu e gol(default
        #- '') TRUE - daca am user in sesiune, FALSE
        altfel

        return render_template('login.html',
session_data=session_data)

    @app.route('/do-login', methods=['POST'])
    def do_login_route():
        print(request.form['inputUser'])
        print(request.form['inputPass'])

        encoded_password =
base64.b64encode(request.form['inputPass']).e
ncode('ascii')
        encoded_password =
encoded_password.decode('ascii')

        #verify if the user is registered
        dictToSend = {"name":
request.form['inputUser'], "password":
encoded_password}

```

```

    res =
requests.post('http://api_management:5000/users/is_registered', json = dictToSend)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    message = dictFromServer['message']

    if dictFromServer['response'] == "success":
        session['user'] =
request.form['inputUser']

        get_user_by_name =
requests.get('http://api_management:5000/users/' + session['user'] )
        user_id =
get_user_by_name.json()['user']['_id']
        print(user_id, flush = True )
        session['user_id'] = user_id
        return redirect('/home')
    else:
        return redirect('/?message=' + message
)

```

#### **/frontend/frontend/routes/logout.py**

```

import json, os, sys
from flask import render_template, request,
session, redirect
from flask_session import Session

def start(app):
    @app.route('/logout')
    def logout_route():
        session['user'] = ''
        session['user_id'] = ''
        return redirect('/')

```

#### **/frontend/frontend/routes/register.py**

```

import json, os, sys
import requests
import base64
from flask import render_template, request,
session, redirect
from flask_session import Session

def start(app):
    @app.route('/register')
    def register_route():
        session_data = { #in sesiune
scriu userul
            'user': session.get('user', ''),
#iau userul de pe sesiunea din Flask
            'logged_in': session.get('user',
'') != '' #creez variabila logged_in care
verifica ca userul exista, nu e gol(default
- '') TRUE - daca am user in sesiune, FALSE
altfel
        }

        return
render_template('register.html',
session_data=session_data)

```

```

@app.route('/do-register',
methods=['POST'])
def do_register_route():
    print(request.form['inputUser'])
    print(request.form['inputPass'])

    encoded_password =
base64.b64encode(request.form['inputPass'].e
ncode('ascii'))
    encoded_password =
encoded_password.decode('ascii')

    encoded_confirm_password =
base64.b64encode(request.form['inputPassConf
irm'].encode('ascii'))
    encoded_confirm_password =
encoded_confirm_password.decode('ascii')

    dictToSend = {"name":
request.form['inputUser'], "password":
encoded_password, "confirm_password":
encoded_confirm_password}
    res =
requests.post('http://api_management:5000/cr
eate_user', json = dictToSend)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    message = dictFromServer['message']

    return redirect('/?message=' +
message )

```

#### **/frontend/frontend/routes/routes\_from\_connection.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session
from flask_session import Session

def start(app):
    @app.route('/routes-from-connection')
    def routes_from_connection():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
!= ''
        }

        res =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections')
        dictFromServer = res.json()
        print(dictFromServer, flush = True )
        return
render_template('routes_from_connection.html
', session_data=session_data,
dictFromServer=dictFromServer)

```

#### **/frontend/frontend/routes/routes.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session, redirect
from flask_session import Session

```

```

def start(app):

@app.route('/routes-from/<connection_name>')
def routes_route(connection_name):
    session_data = {
        'user': session.get('user', ''),
        'logged_in': session.get('user', '')
    }

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/users/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #get routes for a specific connection
        res =
requests.get('http://api_management:5000/connections/' + connection_id + '/routes')
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        #get services for a specific connection
        res_services =
requests.get('http://api_management:5000/connections/' + connection_id + '/services')
        dictFromServer_services =
res_services.json()
        print(dictFromServer_services, flush = True )

        return render_template('routes.html',
session_data=session_data,
dictFromServer=dictFromServer,
dictFromServer_services=dictFromServer_services,
connection_name=connection_name )

    if(connection_type == "tyk"):
        #get apis for a specific connection
        res =
requests.get('http://api_management:5000/connections/' + connection_id + '/tyk/apis')
        dict = res.json()
        output = []
        print("dict[content]", flush = True )
        print(dict['content'], flush = True )
        for item in dict['content']:
            print("item", flush = True )
            print(item, flush = True )
            paths = item["proxy"]["listen_path"]
            print(paths, flush = True )
            name = paths[1:]
            if (name[-1] == '/'):
                name = name[:-1]
            name = name.replace('/', '-')

            print(name, flush = True )

```

```

        dict_api = {"id": item["api_id"],
"name": name, "paths": paths, "service":
item["api_id"]}
        output.append(dict_api)
        dictFromServer = {"routes": output}

        print(dictFromServer, flush = True )

        return
render_template('routes_for_tyk.html',
session_data=session_data,
dictFromServer=dictFromServer,
connection_name=connection_name)

@app.route('/create-route-for-connection/<connection_name>', methods=['POST'])
def do_create_route(connection_name):
    print(request.form['name'])
    print(request.form['paths'])
    print("We're on", flush = True )
    print(connection_name, flush = True )

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/users/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #get service id
        get_service_by_name =
requests.get('http://api_management:5000/connections/' + connection_id + '/services/' +
request.form['service'])
        service_id =
get_service_by_name.json()['content']['id']
        print(service_id, flush = True )

        #str.split() => convert String to
array
        #json.loads() method => convert a
string into a dictionary # '{"id": ' +
service_id +'}'
        #create route
        dictToSend = {"name":
request.form['name'], "paths":
request.form['paths'].split(), "service":
json.loads('{"id": "' + service_id + '"')}
        print(dictToSend, flush = True )
        res =
requests.post('http://api_management:5000/connections/' + connection_id + '/routes',
json = dictToSend)
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        if dictFromServer['response'] == 201:
            #get route id

```

```

        route_id =
dictFromServer['content']['id']
        print(route_id, flush = True )

        #create the route_connection
dependency
        dictToSend_dependency = {"route_id":
route_id, "connection_id": connection_id}
        res =
requests.post('http://api_management:5000/co
nnection_routes', json =
dictToSend_dependency)
        dictFromServer_dependency =
res.json()
        print(dictFromServer_dependency,
flush = True )

        return redirect('/routes-from/' +
connection_name )

        if dictFromServer['response'] ==
"fail":
            message = dictFromServer['message']
            return redirect('/?message=' +
message )
            if dictFromServer['response'] == 409
or 400:
                message =
dictFromServer['content']['message']
                return redirect('/?message=' +
message )
            else:
                return redirect('/?message=' +
"Something went wrong!" )

        #if(connection_type == "tyk"):

@app.route('/update-route-for-connection/<co
nnection_name>', methods=['POST'])
def do_update_route(connection_name):
    print(request.form['id'])
    print("We're on update", flush = True )

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['
_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['
type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #get service id
        get_service_by_name =
requests.get('http://api_management:5000/con
nections/' + connection_id + '/services/' +
request.form['service'])
        service_id =
get_service_by_name.json()['content']['id']
        print(service_id, flush = True )

        #update the route

```

```

        dictToSend = {"name":
request.form['name'], "paths":
request.form['paths'].split(), "service":
json.loads('{ "id": "' + service_id + '" }')}
        res =
requests.put('http://api_management:5000/con
nections/' + connection_id + '/routes/' +
request.form['id'], json = dictToSend)
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        if dictFromServer['response'] == 200:
            return redirect('/routes-from/' +
connection_name )
            if dictFromServer['response'] ==
"fail":
                message = dictFromServer['message']
                return redirect('/?message=' +
message )
                if dictFromServer['response'] == 409
or 400:
                    message =
dictFromServer['content']['message']
                    return redirect('/?message=' +
message )
                else:
                    return redirect('/?message=' +
"Something went wrong!" )

        if(connection_type == "tyk"):
            #get api by id
            res =
requests.get('http://api_management:5000/con
nections/' + connection_id + '/tyk/apis/' +
request.form['id'])
            dict = res.json()
            print(dict, flush = True )
            if dict['response'] == 404:
                message = dict['content']['message']
                return redirect('/?message=' +
message )
            target_url =
dict['content']['proxy']['target_url']

            print(request.form['paths'], flush =
True )
            dictToSend = {
                "name": request.form['id'],
                "api_id": request.form['id'],
                "org_id": "default",
                "definition": {
                    "location": "header",
                    "key": "version"
                },
                "use_keyless": True,
                "version_data": {
                    "not_versioned": True,
                    "versions": {
                        "Default": {
                            "name": "Default"
                        }
                    }
                },
                "custom_middleware": {
                    "pre": [
                        {
                            "name":
"testJSVMData",
                            "path":
"./middleware/injectHeader.js",

```

```

        "require_session":
False,
        "raw_body_only":
False
    }
    ],
    },
    "driver": "otto",
    "proxy": {
        "listen_path":
request.form['paths'],
        "target_url": target_url,
        "strip_listen_path": True
    }
    }
    res =
requests.put('http://api_management:5000/con
nections/' + connection_id + '/tyk/apis/' +
request.form['id'], json = dictToSend)
    dictFromServer = res.json()
    print(dictFromServer, flush = True )

    if dictFromServer['response'] == 200:
        return redirect('/routes-from/' +
connection_name )
    if dictFromServer['response'] ==
"fail":
        message = dictFromServer['message']
        return redirect('/?message=' +
message )
    else:
        return redirect('/?message=' +
"Something went wrong!" )

@app.route('/delete-route-for-connection/<co
nnection_name>', methods=['POST'])
def delete_route(connection_name):

    print("We're on delete", flush = True )
    name = request.form['name']
    print(name, flush = True)

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['
_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['
type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #get route id
        get_route_by_name =
requests.get('http://api_management:5000/con
nections/' + connection_id + '/routes/' +
name)
        route_id =
get_route_by_name.json()['content']['id']
        print(route_id, flush = True )

        #get dependency id

```

```

        get_dependency_by_routeID =
requests.get('http://api_management:5000/con
nection_routes/' + route_id)
        dependency_id =
get_dependency_by_routeID.json()['dependency
']['_id']
        print(dependency_id, flush = True )

        #delete the route_connection
        dependency
        res =
requests.delete('http://api_management:5000/
connection_routes/' + dependency_id)
        dictFromServer_dependency = res.json()
        print(dictFromServer_dependency, flush
= True )

        #delete route
        res =
requests.delete('http://api_management:5000/
connections/' + connection_id + '/routes/' +
name)
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        return redirect('/routes-from/' +
connection_name )

```

#### **/frontend/frontend/routes/services\_from\_connection.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session
from flask_session import Session

def start(app):
    @app.route('/services-from-connection')
    def services_from_connection():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }
        res =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections')
        dictFromServer = res.json()
        print(dictFromServer, flush = True )
        return
render_template('services_from_connection.ht
ml', session_data=session_data,
dictFromServer=dictFromServer)

```

#### **/frontend/frontend/routes/services.py**

```

import json, os, sys
import requests
from flask import render_template, request,
session, redirect
from flask_session import Session

def start(app):

```

```

@app.route('/services-from/<connection_name>')
def services_route(connection_name):
    session_data = {
        'user': session.get('user', ''),
        'logged_in': session.get('user', '')
    }

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/users/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #get services for a specific
connection
        res =
requests.get('http://api_management:5000/connections/' + connection_id + '/services')
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        return
    render_template('services.html',
session_data=session_data,
dictFromServer=dictFromServer,
connection_name=connection_name)

    if(connection_type == "tyk"):
        #get apis for a specific connection
        res =
requests.get('http://api_management:5000/connections/' + connection_id + '/tyk/apis')
        dict = res.json()
        output = []
        print("dict[content]", flush = True )
        print(dict['content'], flush = True )

        if dict['content']:
            for item in dict['content']:
                print("item", flush = True )
                print(item, flush = True )
                target_url =
item["proxy"]["target_url"]
                print(target_url, flush = True )
                port =
target_url.split(":")[2][-1]
                print(port, flush = True )
                host =
target_url.split(":")[1][2:]
                print(host, flush = True )
                dict_api = {"id": item["api_id"],
"name": item["name"], "path": "/", "port":
port, "host": host}
                output.append(dict_api)

        dictFromServer = {"services": output}

        print(dictFromServer, flush = True )

```

```

        return
    render_template('services_for_tyk.html',
session_data=session_data,
dictFromServer=dictFromServer,
connection_name=connection_name)

@app.route('/create-service-for-connection/<
connection_name>', methods=['POST', 'GET'])
def do_create_service(connection_name):
    print(request.form['name'])
    print(request.form['path'])
    print("We're on", flush = True )
    print(connection_name, flush = True )

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/users/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['_id']
    print(connection_id, flush = True )
    connection_type =
get_connection_by_name.json()['connection']['type']
    print(connection_type, flush = True )

    if(connection_type == "kong"):
        #validate input params
        if request.form['port'] == '':
            return redirect('/?message=' +
        "Please provide a port!" )
        if request.form['port'].isnumeric() ==
False:
            return redirect('/?message=' +
        "Please make sure that the port is an
integer!" )

        #create service
        dictToSend = {"name":
request.form['name'], "path":
request.form['path'], "port":
int(request.form['port']), "host":
request.form['host']}
        res =
requests.post('http://api_management:5000/connections/' + connection_id + '/services',
json = dictToSend)
        if(res.status_code == 500):
            return redirect('/?message=' +
        "Please validate the Admin API URL for the
current connection!" )
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        if dictFromServer['response'] == 201:
            #get service id
            service_id =
dictFromServer['content']['id']
            print(service_id, flush = True )

            #create the service_connection
dependency

```

```

        dictToSend_dependency =
{"service_id": service_id, "connection_id":
connection_id}
        res =
requests.post('http://api_management:5000//c
onnection_services', json =
dictToSend_dependency)
        dictFromServer_dependency =
res.json()
        print(dictFromServer_dependency,
flush = True )

        return redirect('/services-from/' +
connection_name )

        if dictFromServer['response'] ==
"fail":
            message = dictFromServer['message']
            return redirect('/?message=' +
message )
            if dictFromServer['response'] == 409
or 400:
                message =
dictFromServer['content']['message']
                return redirect('/?message=' +
message )
            else:
                return redirect('/?message=' +
"Something went wrong!" )

        if(connection_type == "tyk"):
            #validate input params
            if request.form['host'] == '':
                return redirect('/?message=' +
"Please provide a host" )
            if request.form['port'] == '':
                return redirect('/?message=' +
"Please provide a port" )
            #create api
            dictToSend = {
                "name": request.form['name'],
                "api_id": request.form['name'],
                "org_id": "default",
                "definition": {
                    "location": "header",
                    "key": "version"
                },
                "use_keyless": True,
                "version_data": {
                    "not_versioned": True,
                    "versions": {
                        "Default": {
                            "name": "Default"
                        }
                    }
                },
                "custom_middleware": {
                    "pre": [
                        {
                            "name":
"testJSVMDData",
                            "path":
"./middleware/injectHeader.js",
                            "require_session":
False,
                            "raw_body_only":
False
                        }
                    ]
                },
            },

```

```

        "driver": "otto",
        "proxy": {
            "listen_path":
request.form['route_path'],
            "target_url": "http://" +
request.form['host'] + ":" +
request.form['port'] + "/",
            "strip_listen_path": True
        }
        res =
requests.post('http://api_management:5000/co
nnections/' + connection_id + '/tyk/apis',
json = dictToSend)
        if(res.status_code == 500):
            return redirect('/?message=' +
"Please validate the Admin API URL for the
current connection!" )
            dictFromServer = res.json()
            print(dictFromServer, flush = True )

            if dictFromServer['response'] == 200:
                #get tyk_api id
                tyk_api_id = request.form['name']
                print(tyk_api_id, flush = True )

                #create the tyk_api_connection
                dependency
                dictToSend_dependency =
{"tyk_api_id": tyk_api_id, "connection_id":
connection_id}
                res =
requests.post('http://api_management:5000//c
onnection_tyk_apis', json =
dictToSend_dependency)
                dictFromServer_dependency =
res.json()
                print(dictFromServer_dependency,
flush = True )

                return redirect('/services-from/' +
connection_name )

                if dictFromServer['response'] ==
"fail":
                    message = dictFromServer['message']
                    return redirect('/?message=' +
message )
                    else:
                        return redirect('/?message=' +
"Something went wrong!" )

@app.route('/update-service-for-connection/<
connection_name>', methods=['POST'])
def do_update_service(connection_name):
    print(request.form['id'])
    print("We're on update", flush = True )

    #get connection id
    get_connection_by_name =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections/'
+ connection_name )
    connection_id =
get_connection_by_name.json()['connection']['
_id']
    print(connection_id, flush = True )

```

```

        connection_type =
get_connection_by_name.json()['connection']['
type']
        print(connection_type, flush = True )

        if(connection_type == "kong"):
            #validate input params
            if request.form['port'] == '':
                return redirect('/?message=' +
                "Please provide a port!" )
            if request.form['port'].isnumeric() ==
False:
                return redirect('/?message=' +
                "Please make sure that the port is an
integer!" )

            #update service
            dictToSend = {"name":
request.form['name'], "path":
request.form['path'], "port":
int(request.form['port']), "host":
request.form['host']}
            res =
requests.put('http://api_management:5000/con
nections/' + connection_id + '/services/' +
request.form['id'], json = dictToSend)
            dictFromServer = res.json()
            print(dictFromServer, flush = True )

            if dictFromServer['response'] == 200:
                return redirect('/services-from/' +
connection_name )
            if dictFromServer['response'] ==
"fail":
                message = dictFromServer['message']
                return redirect('/?message=' +
message )
            if dictFromServer['response'] == 409
or 400:
                message =
dictFromServer['content']['message']
                return redirect('/?message=' +
message )
            else:
                return redirect('/?message=' +
                "Something went wrong!" )

            if(connection_type == "tyk"):
                print(request.form['port'], flush =
True )
                print(request.form['host'], flush =
True )
                print(request.form['id'], flush = True
)
            #get api by id
            res =
requests.get('http://api_management:5000/con
nections/' + connection_id + '/tyk/apis/' +
request.form['id'])
            dict = res.json()
            print(dict, flush = True )
            if dict['response'] == 404:
                message = dict['content']['message']
                return redirect('/?message=' +
message )
            paths =
dict['content']['proxy']['listen_path']
            print(paths, flush = True )
            dictToSend = {
                "name": request.form['id'],

```

```

                "api_id": request.form['id'],
                "org_id": "default",
                "definition": {
                    "location": "header",
                    "key": "version"
                },
                "use_keyless": True,
                "version_data": {
                    "not_versioned": True,
                    "versions": {
                        "Default": {
                            "name": "Default"
                        }
                    }
                },
                "custom_middleware": {
                    "pre": [
                        {
                            "name":
"testJSVMData",
                            "path":
"./middleware/injectHeader.js",
                            "require_session":
False,
                            "raw_body_only":
False
                        }
                    ],
                    "driver": "otto",
                    "proxy": {
                        "listen_path": paths,
                        "target_url": "http://" +
request.form['host'] + ":" +
request.form['port'] + "/",
                        "strip_listen_path": True
                    }
                }
            print(dictToSend, flush = True )
            res =
requests.put('http://api_management:5000/con
nections/' + connection_id + '/tyk/apis/' +
request.form['id'], json = dictToSend)
            dictFromServer = res.json()
            print(dictFromServer, flush = True )

            if dictFromServer['response'] == 200:
                return redirect('/services-from/' +
connection_name )
            if dictFromServer['response'] ==
"fail":
                message = dictFromServer['message']
                return redirect('/?message=' +
message )
            else:
                return redirect('/?message=' +
                "Something went wrong!" )

@app.route('/delete-service-for-connection/<
connection_name>', methods=['POST'])
def delete_service(connection_name):

    print("We're on delete", flush = True )
    print(request.form['name'])

    #get connection id

```



```

        get_connection_by_name =
requests.get('http://api_management:5000/use
rs/' + session['user_id'] + '/connections/'
+ connection_name )
        connection_id =
get_connection_by_name.json()['connection'][
'id']
        print(connection_id, flush = True )
        connection_type =
get_connection_by_name.json()['connection'][
'type']
        print(connection_type, flush = True )

        if(connection_type == "kong"):
            #get service id
            get_service_by_name =
requests.get('http://api_management:5000/con
nections/' + connection_id + '/services/' +
request.form['name'])
            service_id =
get_service_by_name.json()['content']['id']
            print(service_id, flush = True )

            #get dependency id
            get_dependency_by_serviceID =
requests.get('http://api_management:5000/con
nection_services/' + service_id)
            dependency_id =
get_dependency_by_serviceID.json()['dependen
cy']['id']
            print(dependency_id, flush = True )

            #delete service
            res =
requests.delete('http://api_management:5000/
connections/' + connection_id + '/services/'
+ request.form['name'])
            dictFromServer = res.json()
            print(dictFromServer, flush = True )

            if dictFromServer['response'] == 400:
                message =
dictFromServer['content']['message']
                return redirect('/?message=' +
message )

            #delete the service_connection
            dependency
            result =
requests.delete('http://api_management:5000/
connection_services/' + dependency_id)
            dictFromServer_dependency =
result.json()
            print(dictFromServer_dependency, flush
= True )

            return redirect('/services-from/' +
connection_name )

        if(connection_type == "tyk"):
            #get tyk_api id
            tyk_api_id = request.form['name']

            #get dependency id
            get_dependency_by_tyk_apiID =
requests.get('http://api_management:5000/con
nection_tyk_apis/' + tyk_api_id)
            dependency_id =
get_dependency_by_tyk_apiID.json()['dependen
cy']['id']

```

```

        print(dependency_id, flush = True )

        #delete tyk_api
        res =
requests.delete('http://api_management:5000/
connections/' + connection_id + '/tyk/apis/'
+ request.form['name'])
        dictFromServer = res.json()
        print(dictFromServer, flush = True )

        #delete the tyk_api_connection
        dependency
        result =
requests.delete('http://api_management:5000/
connection_tyk_apis/' + dependency_id)
        dictFromServer_dependency =
result.json()
        print(dictFromServer_dependency, flush
= True )
        return redirect('/services-from/' +
connection_name )

```

### /frontend/frontend/routes/users.py

```

import json, os, sys
import requests
from flask import render_template, request,
session
from flask_session import Session

def start(app):
    @app.route('/users')
    def users_route():
        session_data = {
            'user': session.get('user', ''),
            'logged_in': session.get('user', '')
        }

        res =
requests.get('http://api_management:5000/use
rs')
        dictFromServer = res.json()
        print(dictFromServer, flush = True )
        return render_template('users.html',
session_data=session_data,
dictFromServer=dictFromServer)

```

### /frontend/frontend/routes/connections.html

```

{% include 'header.html' %}

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css?famil
y=Roboto|Varela+Round|Open+Sans">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.5.0/css/bootstrap.min.css">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?fami
ly=Material+Icons">

```

```

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-awes
esome/4.7.0/css/font-awesome.min.css">
<script
src="https://code.jquery.com/jquery-3.5.1.mi
n.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.16.0/dist/umd/popper.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/js/bootstrap.min.js"></script>
<!-- Required meta tags -->
<meta charset="utf-8">
<meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap
@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH
/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then
Bootstrap JS -->
<script
src="https://code.jquery.com/jquery-3.3.1.sl
im.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQI
AqVgRVzpbzo5smXKp4YfRvH+8abtTElPi6jizo"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.14.7/dist/umd/popper.min.js"
integrity="sha384-U02eT0CpHqdSJQ6hJty5KVphtP
hzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@
4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoII
y6OrQ6VrjIEaFf/njGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>

<style>
  body {
    color: #404E67;
    background: #F5F7FA;
    font-family: 'Open Sans',
sans-serif;
  }

  .table-responsive {
    width: 1500px;
  }

  .table-wrapper {
    width: 1400px;
    margin: 30px auto;
    background: #fff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
  }

  .table-title {

```

```

padding-bottom: 10px;
margin: 0 0 10px;
}

.table-title h2 {
  margin: 6px 0 0;
  font-size: 22px;
}

.table-title .add-new {
  float: right;
  height: 30px;
  font-weight: bold;
  font-size: 12px;
  text-shadow: none;
  min-width: 100px;
  border-radius: 50px;
  line-height: 13px;
  margin-right: 4px;
}

table.table {
  table-layout: fixed;
}

table.table tr th,
table.table tr td {
  border-color: #e9e9e9;
}

table.table th i {
  font-size: 13px;
  margin: 0 5px;
  cursor: pointer;
}

table.table td button {
  cursor: pointer;
  display: inline-block;
  margin: 0 5px;
  min-width: 24px;
}

table.table td button.edit {
  color: #d1b150;
}

table.table td button.delete {
  color: #e46448;
}

table.table td i {
  font-size: 19px;
}

table.table td .add {
  display: none;
}
</style>

</head>

<body>
  <h2>Connections</h2>
  <div class="container-lg">
    <div class="table-responsive">
      <div class="table-wrapper">
        <div class="table-title">
          <div class="row">

```

```

        <div><button
type="button" class="btn btn-primary
btn-info add-new" data-toggle="modal"
data-target="#modal-add"><i class="fa
fa-plus"></i> Add New</button>
        <div
class="modal fade" id="modal-add"
tabindex="-1" role="dialog"
aria-labelledby="modal-add"
aria-hidden="true">
            <div
class="modal-dialog" role="document">
                <div
class="modal-content">
                    <div
class="modal-header">
                        <h5 class="modal-title" id="modal-add">Add a
new connection</h5>
                        <button type="button" class="close"
data-dismiss="modal" aria-label="Close">
                            <span aria-hidden="true">&times;</span>
                        </button>
                    </div>
                    <div
class="modal-body">
                        <form action="/create-connection"
method="POST" id="formaddnew">
                            <div class="form-group">
                                <label for="name"
class="col-form-label">Name:</label>
                                <input type="text" name="name"
class="form-control" id="name">
                            </div>
                            <div class="form-group">
                                <label for="type"
class="col-form-label">Type:</label>
                                <select name="type" class="form-control"
id="type">
                                    <option value="kong">kong</option>
                                    <option value="tyk">tyk</option>
                                </select>
                            </div>
                            <div class="form-group">
                                <label for="admin_api_url"
class="col-form-label">Admin_api_url:</label>
                                <input type="text" name="admin_api_url"
class="form-control" id="admin_api_url">

```

```

        </div>
    </form>
</div>
<div
class="modal-footer">
    <button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>
    <button type="submit" class="btn
btn-primary" form="formaddnew">Add</button>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
<table class="table
table-bordered">
    <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Type</th>
            <th>Admin API
URL</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for item in
dictFromServer.connections %}
            <tr>
                <td>{{item._id}}</td>
                <td>{{item.name}}</td>
                <td>{{item.type}}</td>
                <td>{{item.admin_api_url}}</td>
                <td>
                    <button
type="button" class="btn btn-primary
btn-info edit" data-toggle="modal"
data-target="#modal-edit{{loop.index}}"><i
class="material-icons">&#xE254;</i></button>
                    <div
class="modal fade"
id="modal-edit{{loop.index}}" tabindex="-1"
role="dialog"
aria-labelledby="modal-edit{{loop.index}}"
aria-hidden="true">
                        <div
class="modal-dialog" role="document">
                            <div
class="modal-content">
                                <div class="modal-header">
                                    <h5 class="modal-title"
id="modal-edit{{loop.index}}">Edit a
connection</h5>

```



```

class="form-control" id="id"
value={{item._id}} readonly>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>

<button type="submit" class="btn
btn-primary"
form="formdelete{{loop.index}}">Delete</butt
on>

</div>

</div>

</div>
</div>
</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</div>
</body>

</html>

{% include 'footer.html' %}

```

### /frontend/frontend/routes/deployment\_for\_connecti on.html

```

{% include 'header.html' %} {% if created ==
"No" %}
<h2>Do you really want to deploy a
{{connection_type}} API Gateway?</h2>

<form
action="/deployment/{{connection_type}}"
method="POST" id="formdeploy">
</form>
<button type="submit" class="btn
btn-primary"
form="formdeploy">Yes</button>{% else %}
<h2>API Gateway successfully deployed. A new
{{connection_type}} connection has been
created.</h2>
{% endif %} {% include 'footer.html' %}

```

### /frontend/frontend/routes/deployments.html

```

{% include 'header.html' %}
<h2>Which type of API Gateway would you like
to deploy?</h2>
{% for type in ["kong", "tyk"] %}
<a class="nav-link"
href="/deployment-for/{{type}}">{{type}}</a>
{% endfor %} {% include 'footer.html' %}

```

### /frontend/frontend/routes/footer.html

```

</div>
</div>
</div>

</body>
</html>

```

### /frontend/frontend/routes/header.html

```

<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

  <title>Manage API Gateway</title>

  <link rel="stylesheet"
href="/static/bootstrap/css/bootstrap.min.cs
s">
  <link rel="stylesheet"
href="/static/app.css">

  <script
src="/static/jquery-3.6.0.min.js"></script>
  <script
src="/static/bootstrap/js/bootstrap.bundle.m
in.js"></script>
  <script src="/static/app.js"></script>

  <style>
    .card {
      width: 170px;
    }
  </style>
</head>

<body>

  <nav class="navbar navbar-expand-lg
navbar-light bg-light">
    <div class="container-fluid">
      <a class="navbar-brand"
href="#">Manage API Gateway</a>
      <button class="navbar-toggler"
type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle
navigation">
        <span
class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse
navbar-collapse"
id="navbarSupportedContent">
        <ul class="navbar-nav
me-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link
active" aria-current="page"
href="/home">Home</a>
          </li>
        </ul>

```

```

        {% if session_data.logged_in
%}
        <span style="padding-right:
20px">Hello, {{ session_data.user }}</span>
        <a class="btn
btn-outline-primary"
href="/logout">Logout</a> {% else %}
        <meta http-equiv="refresh"
content="0; url=http://localhost:5000/" />
{% endif %}
    </div>
</div>
</nav>

    <div class="container"
style="margin-top: 20px; float:left;">
    <div class="row">
        <div class="col-2">

            <div class="card">
                <div
class="card-header">
                    Menu
                </div>
                <ul class="list-group
list-group-flush">
                    <li
class="list-group-item">
                        <a
class="nav-link"
href="/services-from-connection">Services</a>
                    >
                        </li>
                    <li
class="list-group-item">
                        <a
class="nav-link"
href="/routes-from-connection">Routes</a>
                        </li>
                    <li
class="list-group-item">
                        <a
class="nav-link" href="/users">Users</a>
                        </li>
                    <li
class="list-group-item">
                        <a
class="nav-link" href="/connections">Connections</a>
                        </li>
                    </li>
                    <li
class="list-group-item">
                        <a
class="nav-link" href="/deployments">Deployments</a>
                        </li>
                    </ul>
                </div>
            <div class="col-10">

```

### /frontend/frontend/routes/home.html

```

{% include 'header.html' %}
<!-- <h1>index</h1> -->
{% include 'footer.html' %}

```

### /frontend/frontend/routes/index.html

```

<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <title>Manage API Gateway</title>

    <link rel="stylesheet"
href="/static/bootstrap/css/bootstrap.min.cs
s">
    <link rel="stylesheet"
href="/static/app.css">

    <script
src="/static/jquery-3.6.0.min.js"></script>
    <script
src="/static/bootstrap/js/bootstrap.bundle.m
in.js"></script>
    <script src="/static/app.js"></script>
</head>

<body>

    <nav class="navbar navbar-expand-lg
navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand"
href="#">Manage API Gateway</a>
            <button class="navbar-toggler"
type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle
navigation">
                <span
class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse
navbar-collapse"
id="navbarSupportedContent">
                {% if session_data.logged_in
%}
                <ul class="navbar-nav
me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link
active" aria-current="page"
href="/home">Home</a>
                    </li>
                </ul>{% endif %} {% if
session_data.logged_in %}
                <span style="padding-right:
20px">Hello, {{ session_data.user }}</span>
                <a class="btn
btn-outline-primary"
href="/logout">Logout</a> {% else %}
                <a class="btn
btn-outline-primary" href="/login">Login</a>
                <a class="btn
btn-outline-primary"
href="/register">Register</a>{% endif %}

```

```

        </div>
    </div>
</nav>
<!-- This is a comment -->
{% if message != '' %}
<div class="alert alert-secondary"
role="alert">
    <p>{{ message }}</p>
</div>
{% endif %} {% include 'footer.html' %}

```

### /frontend/frontend/routes/login.html

```

<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <title>Manage API Gateway</title>

    <link rel="stylesheet"
href="/static/bootstrap/css/bootstrap.min.cs
s">
    <link rel="stylesheet"
href="/static/app.css">

    <script
src="/static/jquery-3.6.0.min.js"></script>
    <script
src="/static/bootstrap/js/bootstrap.bundle.m
in.js"></script>
    <script src="/static/app.js"></script>
</head>

<body>

    <nav class="navbar navbar-expand-lg
navbar-light bg-light">
        <div class="container-fluid">
            <a class="navbar-brand"
href="#">Manage API Gateway</a>
            <button class="navbar-toggler"
type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle
navigation">
                <span
class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse
navbar-collapse"
id="navbarSupportedContent">
                {% if session_data.logged_in
%}
                <ul class="navbar-nav
me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link
active" aria-current="page"
href="/">Home</a>
                    </li>
                </ul>{% endif %} {% if
session_data.logged_in %}

```

```

                <span style="padding-right:
20px">Hello, {{ session_data.user }}</span>
                <a class="btn
btn-outline-primary"
href="/logout">Logout</a> {% else %}
                <a class="btn
btn-outline-primary" href="/login">Login</a>
                <a class="btn
btn-outline-primary"
href="/register">Register</a>{% endif %}
            </div>
        </div>
</nav>

```

```

<form action="/do-login" method="POST">
    <div class="mb-3">
        <label for="inputUser">Username</label>
        <input type="text"
name="inputUser" class="form-control"
id="inputUser">
    </div>
    <div class="mb-3">
        <label for="inputPass">Password</label>
        <input type="password"
name="inputPass" class="form-control"
id="inputPass">
    </div>
    <button type="submit" class="btn
btn-primary">Submit</button>
</form>

{% include 'footer.html' %}

```

### /frontend/frontend/routes/register.html

```

<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <title>Manage API Gateway</title>

    <link rel="stylesheet"
href="/static/bootstrap/css/bootstrap.min.cs
s">
    <link rel="stylesheet"
href="/static/app.css">

    <script
src="/static/jquery-3.6.0.min.js"></script>
    <script
src="/static/bootstrap/js/bootstrap.bundle.m
in.js"></script>
    <script src="/static/app.js"></script>
</head>

<body>

    <nav class="navbar navbar-expand-lg
navbar-light bg-light">
        <div class="container-fluid">

```

```

        <a class="navbar-brand"
href="#">Manage API Gateway</a>
        <button class="navbar-toggler"
type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle
navigation">
        <span
class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse
navbar-collapse"
id="navbarSupportedContent">
            {% if session_data.logged_in
%}
            <ul class="navbar-nav
me-auto mb-2 mb-lg-0">
                <li class="nav-item">
                    <a class="nav-link
active" aria-current="page"
href="/">Home</a>
                </li>
            </ul>{% endif %} {% if
session_data.logged_in %}
                <span style="padding-right:
20px">Hello, {{ session_data.user }}</span>
                <a class="btn
btn-outline-primary"
href="/logout">Logout</a> {% else %}
                <a class="btn
btn-outline-primary" href="/login">Login</a>
                <a class="btn
btn-outline-primary"
href="/login">Register</a>{% endif %}
            </div>
        </nav>

        <form action="/do-register"
method="POST">
            <div class="mb-3">
                <label for="inputUser"
class="form-label">Username</label>
                <input type="text"
name="inputUser" class="form-control"
id="inputUser">
            </div>
            <div class="mb-3">
                <label for="inputPass"
class="form-label">Password</label>
                <input type="password"
name="inputPass" class="form-control"
id="inputPass">
            </div>
            <div class="mb-3">
                <label for="inputPassConfirm"
class="form-label">Confirm password</label>
                <input type="password"
name="inputPassConfirm" class="form-control"
id="inputPassConfirm">
            </div>
            <button type="submit" class="btn
btn-primary">Submit</button>
        </form>

        {% include 'footer.html' %}

```

## /frontend/frontend/routes/routes\_for\_tyk.html

```

{% include 'header.html' %}

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css?famil
y=Roboto|Varela+Round|Open+Sans">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/css/bootstrap.min.css">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?fami
ly=Material+Icons">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-aw
esome/4.7.0/css/font-awesome.min.css">
    <script
src="https://code.jquery.com/jquery-3.5.1.mi
n.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.16.0/dist/umd/popper.min.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/js/bootstrap.min.js"></script>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap
@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH
/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then
Bootstrap JS -->
    <script
src="https://code.jquery.com/jquery-3.3.1.sl
im.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQI
AqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.14.7/dist/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdsJQ6hJty5KVphtP
hzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@
4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoII
y6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>

    <style>
        body {
            color: #404E67;
            background: #F5F7FA;

```



```

        font-family: 'Open Sans',
sans-serif;
    }

    .table-responsive {
        width: 1500px;
    }

    .table-wrapper {
        width: 1400px;
        margin: 30px auto;
        background: #fff;
        padding: 20px;
        box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
    }

    .table-title {
        padding-bottom: 10px;
        margin: 0 0 10px;
    }

    .table-title h2 {
        margin: 6px 0 0;
        font-size: 22px;
    }

    .table-title .add-new {
        float: right;
        height: 30px;
        font-weight: bold;
        font-size: 12px;
        text-shadow: none;
        min-width: 100px;
        border-radius: 50px;
        line-height: 13px;
        margin-right: 4px;
    }

    table.table {
        table-layout: fixed;
    }

    table.table tr th,
    table.table tr td {
        border-color: #e9e9e9;
    }

    table.table th i {
        font-size: 13px;
        margin: 0 5px;
        cursor: pointer;
    }

    table.table td button {
        cursor: pointer;
        display: inline-block;
        margin: 0 5px;
        min-width: 24px;
    }

    table.table td button.edit {
        color: #d1b150;
    }

    table.table td button.delete {
        color: #e46448;
    }

    table.table td i {

```

```

        font-size: 19px;
    }

    table.table td .add {
        display: none;
    }
</style>

</head>

<body>
    <h2>Routes for {{connection_name}}
connection</h2>

    <div class="container-lg">
        <div class="table-responsive">
            <div class="table-wrapper">
                <div class="table-title">
                    <div class="row">

                        </div>
                    </div>
                    <table class="table
table-bordered">
                        <thead>
                            <tr>
                                <th>ID</th>
                                <th>Name</th>
                                <th>Paths</th>
                                <th>Service</th>
                                <th>Actions</th>
                            </tr>
                        </thead>
                        <tbody>
                            {% for item in
dictFromServer.routes %}
                                <tr>

                                    <td>{{item.id}}</td>

                                    <td>{{item.name}}</td>

                                    <td>{{item.paths}}</td>

                                    <td>{{item.service}}</td>

                                    <td>

                                        <button
type="button" class="btn btn-primary
btn-info edit" data-toggle="modal"
data-target="#modal-edit{{loop.index}}"><i
class="material-icons">&#xE254;</i></button>

                                        <div
class="modal fade"
id="modal-edit{{loop.index}}" tabindex="-1"
role="dialog"
aria-labelledby="modal-edit{{loop.index}}"
aria-hidden="true">

                                            <div
class="modal-dialog" role="document">

                                                <div
class="modal-content">

                                                    <div class="modal-header">

                                                        <h5 class="modal-title"
id="modal-edit{{loop.index}}">Edit a
route</h5>

```

```

<button type="button" class="close"
data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<form
action="/update-route-for-connection/{{conne
ction_name}}" method="POST"
id="formedit{{loop.index}}">

<div class="form-group">

<label for="id"
class="col-form-label">ID:</label>

<input type="text" name="id"
class="form-control" id="id"
value="{{item.id}}" readonly>

</div>

<div class="form-group">

<label for="name"
class="col-form-label">Name:</label>

<input type="text" name="name"
class="form-control" id="name"
value="{{item.name}}" readonly>

</div>

<div class="form-group">

<label for="paths"
class="col-form-label">Paths:</label>

<input type="text" name="paths"
class="form-control" id="paths"
value="{{item.paths}}">

</div>

<div class="form-group">

<label for="service"
class="col-form-label">Service:</label>

<input type="text" name="service"
class="form-control" id="service"
value="{{item.service}}" readonly>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>

```

```

<button type="submit" class="btn
btn-primary"
form="formedit{{loop.index}}">Edit</button>

</div>

</div>

</div>

</div>

</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</div>

</body>

</html>

{% include 'footer.html' %}

```

## /frontend/frontend/routes/routes\_from\_connection.html

```

{% include 'header.html' %}
<style>
    body {
        color: #404E67;
        background: #F5F7FA;
        font-family: 'Open Sans',
        sans-serif;
    }
</style>
<h2>For which connection would you like to
display the routes?</h2>
{% for item in dictFromServer.connections %}
<a class="nav-link"
href="/routes-from/{{item.name}}">{{item.nam
e}}</a> {% endfor %} {% include
'footer.html' %}

```

## /frontend/frontend/routes/routes.html

```

{% include 'header.html' %}

<head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css?famil
y=Roboto|Varela+Round|Open+Sans">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.5.0/css/bootstrap.min.css">
    <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?fami
ly=Material+Icons">

```

```

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-aw
esome/4.7.0/css/font-awesome.min.css">
<script
src="https://code.jquery.com/jquery-3.5.1.mi
n.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.16.0/dist/umd/popper.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/js/bootstrap.min.js"></script>
<!-- Required meta tags -->
<meta charset="utf-8">
<meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap
@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH
/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then
Bootstrap JS -->
<script
src="https://code.jquery.com/jquery-3.3.1.sl
im.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQI
AqVgRVzpbzo5smXKp4YfRvH+8abtTElPi6jizo"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.14.7/dist/umd/popper.min.js"
integrity="sha384-U02eT0CpHqdsSQ6hJty5KVphtP
hzWj9W0lclHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@
4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoII
y6OrQ6VrjIEaFf/njGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>

<style>
  body {
    color: #404E67;
    background: #F5F7FA;
    font-family: 'Open Sans',
sans-serif;
  }

  .table-responsive {
    width: 1500px;
  }

  .table-wrapper {
    width: 1400px;
    margin: 30px auto;
    background: #fff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
  }

  .table-title {

```

```

padding-bottom: 10px;
margin: 0 0 10px;
}

.table-title h2 {
  margin: 6px 0 0;
  font-size: 22px;
}

.table-title .add-new {
  float: right;
  height: 30px;
  font-weight: bold;
  font-size: 12px;
  text-shadow: none;
  min-width: 100px;
  border-radius: 50px;
  line-height: 13px;
  margin-right: 4px;
}

table.table {
  table-layout: fixed;
}

table.table tr th,
table.table tr td {
  border-color: #e9e9e9;
}

table.table th i {
  font-size: 13px;
  margin: 0 5px;
  cursor: pointer;
}

table.table td button {
  cursor: pointer;
  display: inline-block;
  margin: 0 5px;
  min-width: 24px;
}

table.table td button.edit {
  color: #d1b150;
}

table.table td button.delete {
  color: #e46448;
}

table.table td i {
  font-size: 19px;
}

table.table td .add {
  display: none;
}
</style>

</head>

<body>
  {% if
dictFromServer_services.services|length == 0
%}
    <h2>Please create a service first!</h2>
  {% else %}

```



```

<button
type="button" class="btn btn-primary
btn-info edit" data-toggle="modal"
data-target="#modal-edit{{loop.index}}"><i
class="material-icons">&#xE254;</i></button>
<div
class="modal fade"
id="modal-edit{{loop.index}}" tabindex="-1"
role="dialog"
aria-labelledby="modal-edit{{loop.index}}"
aria-hidden="true">
<div
class="modal-dialog" role="document">
<div
class="modal-content">
<div class="modal-header">
<h5 class="modal-title"
id="modal-edit{{loop.index}}">Edit a
route</h5>
<button type="button" class="close"
data-dismiss="modal" aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
</div>
<div class="modal-body">
<form
action="/update-route-for-connection/{{conne
ction_name}}" method="POST"
id="formedit{{loop.index}}">
<div class="form-group">
<label for="id"
class="col-form-label">ID:</label>
<input type="text" name="id"
class="form-control" id="id"
value="{{item.id}}" readonly>
</div>
<div class="form-group">
<label for="name"
class="col-form-label">Name:</label>
<input type="text" name="name"
class="form-control" id="name"
value="{{item.name}}">
</div>
<div class="form-group">
<label for="paths"
class="col-form-label">Paths:</label>
<input type="text" name="paths"
class="form-control" id="paths"
value={{item.paths[0]}}>
</div>

```

```

<div class="form-group">
<label for="service"
class="col-form-label">Service:</label>
<select name="service" class="form-control"
id="service">
{% for service in
dictFromServer_services.services %}
{% if item.service.id == service.id %}
<option selected="{{service.name}}"
value="{{service.name}}">{{service.name}}</o
ption>
{% else %}
<option
value="{{service.name}}">{{service.name}}</o
ption>
{% endif %}{% endfor %}
</select>
</div>
</form>
</div>
<div class="modal-footer">
<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>
<button type="submit" class="btn
btn-primary"
form="formedit{{loop.index}}">Edit</button>
</div>
</div>
</div>
<div>
<button
type="button" class="btn btn-primary
btn-info delete" data-toggle="modal"
data-target="#modal-delete{{loop.index}}"><i
class="material-icons">&#xE872;</i></button>
<div
class="modal fade"
id="modal-delete{{loop.index}}"
tabindex="-1" role="dialog"
aria-labelledby="modal-delete{{loop.index}}"
aria-hidden="true">
<div
class="modal-dialog" role="document">
<div
class="modal-content">
<div class="modal-header">
<h5 class="modal-title"

```



```

        font-family: 'Open Sans',
sans-serif;
    }

    .table-responsive {
        width: 1500px;
    }

    .table-wrapper {
        width: 1400px;
        margin: 30px auto;
        background: #fff;
        padding: 20px;
        box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
    }

    .table-title {
        padding-bottom: 10px;
        margin: 0 0 10px;
    }

    .table-title h2 {
        margin: 6px 0 0;
        font-size: 22px;
    }

    .table-title .add-new {
        float: right;
        height: 30px;
        font-weight: bold;
        font-size: 12px;
        text-shadow: none;
        min-width: 100px;
        border-radius: 50px;
        line-height: 13px;
        margin-right: 4px;
    }

    table.table {
        table-layout: fixed;
    }

    table.table tr th,
    table.table tr td {
        border-color: #e9e9e9;
    }

    table.table th i {
        font-size: 13px;
        margin: 0 5px;
        cursor: pointer;
    }

    table.table td button {
        cursor: pointer;
        display: inline-block;
        margin: 0 5px;
        min-width: 24px;
    }

    table.table td button.edit {
        color: #d1b150;
    }

    table.table td button.delete {
        color: #e46448;
    }
}
</style>
</head>

```

```

<body>
    <h2>Services for {{connection_name}}
connection</h2>
    <div class="container-lg">
        <div class="table-responsive">
            <div class="table-wrapper">
                <div class="table-title">
                    <div class="row">
                        <div><button
type="button" class="btn btn-primary
btn-info add-new" data-toggle="modal"
data-target="#modal-add"><i class="fa
fa-plus"></i> Add New</button>
                        <div
class="modal fade" id="modal-add"
tabindex="-1" role="dialog"
aria-labelledby="modal-add"
aria-hidden="true">
                            <div
class="modal-dialog" role="document">
                                <div
class="modal-content">
                                    <div
class="modal-header">
                                        <h5 class="modal-title" id="modal-add">Add a
new service</h5>
                                        <button type="button" class="close"
data-dismiss="modal" aria-label="Close">
                                            <span aria-hidden="true">&times;</span>
                                        </button>
                                    </div>
                                    <div class="modal-body">
                                        <form
action="/create-service-for-connection/{{con
nection_name}}" method="POST"
id="formaddnew">
                                            <div class="form-group">
                                                <label for="name"
class="col-form-label">Name:</label>
                                                <input type="text" name="name"
class="form-control" id="name">
                                            </div>
                                            <div class="form-group">
                                                <label for="path"
class="col-form-label">Path:</label>
                                                <input type="text" name="path"
class="form-control" id="path" value="/"
readonly>
                                            </div>
                                            <div class="form-group">

```





```

<label for="path"
class="col-form-label">Path:</label>

<input type="text" name="path"
class="form-control" id="path"
value="{{item.path}}" readonly>

</div>

<div class="form-group">

<label for="port"
class="col-form-label">Port:</label>

<input type="text" name="port"
class="form-control" id="port"
value="{{item.port}}">

</div>

<div class="form-group">

<label for="host"
class="col-form-label">Host:</label>

<input type="text" name="host"
class="form-control" id="host"
value="{{item.host}}">

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>

<button type="submit" class="btn
btn-primary"
form="formedit{{loop.index}}">Edit</button>

</div>

</div>

</div>

<button
type="button" class="btn btn-primary
btn-info delete" data-toggle="modal"
data-target="#modal-delete{{loop.index}}"><i
class="material-icons">&#xE872;</i></button>
<div
class="modal fade"
id="modal-delete{{loop.index}}"
tabindex="-1" role="dialog"
aria-labelledby="modal-delete{{loop.index}}"
aria-hidden="true">
<div
class="modal-dialog" role="document">
<div
class="modal-content">

<div class="modal-header">

```

```

<h5 class="modal-title"
id="modal-delete{{loop.index}}">Do you want
to delete the service with the following
name</h5>

<button type="button" class="close"
data-dismiss="modal" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

</div>

<div class="modal-body">

<form
action="/delete-service-for-connection/{{con
nection_name}}" method="POST"
id="formdelete{{loop.index}}">

<div class="form-group">

<label for="name"
class="col-form-label">Name</label>

<input type="text" name="name"
class="form-control" id="name"
value="{{item.name}}" readonly>

</div>

</form>

</div>

<div class="modal-footer">

<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>

<button type="submit" class="btn
btn-primary"
form="formdelete{{loop.index}}">Delete</butt
on>

</div>

</div>

</div>

</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>

</html>
{% include 'footer.html' %}

```

## /frontend/frontend/routes/services\_from\_connection.html

```
{% include 'header.html' %}
<style>
  body {
    color: #404E67;
    background: #F5F7FA;
    font-family: 'Open Sans',
sans-serif;
  }
</style>
<h2>For which connection would you like to
display the services?</h2>
{% for item in dictFromServer.connections %}
<a class="nav-link"
href="/services-from/{{item.name}}">{{item.n
ame}}</a> {% endfor %} {% include
'footer.html' %}
```

## /frontend/frontend/routes/services.html

```
{% include 'header.html' %}

<head>
  <meta charset="utf-8">
  <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
  <link rel="stylesheet"
href="https://fonts.googleapis.com/css?famil
y=Roboto|Varela+Round|Open+Sans">
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.5.0/css/bootstrap.min.css">
  <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?fami
ly=Material+Icons">
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-a
wesome/4.7.0/css/font-awesome.min.css">
  <script
src="https://code.jquery.com/jquery-3.5.1.mi
n.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.16.0/dist/umd/popper.min.js"></script>
  <script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/js/bootstrap.min.js"></script>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap
@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH
/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

  <!-- Optional JavaScript -->
  <!-- jQuery first, then Popper.js, then
Bootstrap JS -->
```

```
<script
src="https://code.jquery.com/jquery-3.3.1.sl
im.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQI
AqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.14.7/dist/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtP
hzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@
4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoII
y6OrQ6VrjIEaFf/nJGzIxFDs4x0xIM+B07jRM"
crossorigin="anonymous"></script>

<style>
  body {
    color: #404E67;
    background: #F5F7FA;
    font-family: 'Open Sans',
sans-serif;
  }

  .table-responsive {
    width: 1500px;
  }

  .table-wrapper {
    width: 1400px;
    margin: 30px auto;
    background: #fff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
  }

  .table-title {
    padding-bottom: 10px;
    margin: 0 0 10px;
  }

  .table-title h2 {
    margin: 6px 0 0;
    font-size: 22px;
  }

  .table-title .add-new {
    float: right;
    height: 30px;
    font-weight: bold;
    font-size: 12px;
    text-shadow: none;
    min-width: 100px;
    border-radius: 50px;
    line-height: 13px;
    margin-right: 4px;
  }

  table.table {
    table-layout: fixed;
  }

  table.table tr th,
  table.table tr td {
    border-color: #e9e9e9;
  }
```



```

        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Path</th>
            <th>Port</th>
            <th>Host</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>
        {% for item in
dictFromServer.services %}
        <tr>

            <td>{{item.id}}</td>

            <td>{{item.name}}</td>

            <td>{{item.path}}</td>

            <td>{{item.port}}</td>

            <td>{{item.host}}</td>

            <td>

                <button
type="button" class="btn btn-primary
btn-info edit" data-toggle="modal"
data-target="#modal-edit{{loop.index}}"><i
class="material-icons">&#xE254;</i></button>

                <div
class="modal fade"
id="modal-edit{{loop.index}}" tabindex="-1"
role="dialog"
aria-labelledby="modal-edit{{loop.index}}"
aria-hidden="true">

                    <div
class="modal-dialog" role="document">

                        <div
class="modal-content">

                            <div class="modal-header">

                                <h5 class="modal-title"
id="modal-edit{{loop.index}}">Edit a
service</h5>

                                <button type="button" class="close"
data-dismiss="modal" aria-label="Close">

                                <span aria-hidden="true">&times;</span>

                            </button>

                        </div>

                        <div class="modal-body">

                            <form
action="/update-service-for-connection/{{con
nection_name}}" method="POST"
id="formedit{{loop.index}}">

                            <div class="form-group">

                                <label for="id"
class="col-form-label">ID:</label>

                                <input type="text" name="id"

```

```

class="form-control" id="id"
value="{{item.id}}" readonly>

                            </div>

                            <div class="form-group">

                                <label for="name"
class="col-form-label">Name:</label>

                                <input type="text" name="name"
class="form-control" id="name"
value="{{item.name}}">

                            </div>

                            <div class="form-group">

                                <label for="path"
class="col-form-label">Path:</label>

                                <input type="text" name="path"
class="form-control" id="path"
value="{{item.path}}">

                            </div>

                            <div class="form-group">

                                <label for="port"
class="col-form-label">Port:</label>

                                <input type="text" name="port"
class="form-control" id="port"
value="{{item.port}}">

                            </div>

                            <div class="form-group">

                                <label for="host"
class="col-form-label">Host:</label>

                                <input type="text" name="host"
class="form-control" id="host"
value="{{item.host}}">

                            </div>

                        </form>

                    </div>

                </div>

                <div class="modal-footer">

                    <button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>

                    <button type="submit" class="btn
btn-primary"
form="formedit{{loop.index}}">Edit</button>

                </div>

            </td>

        </tr>
    </tbody>
</table>
</div>
</div>

```

```

<button
type="button" class="btn btn-primary
btn-info delete" data-toggle="modal"
data-target="#modal-delete{{loop.index}}"><i
class="material-icons">&#xE872;</i></button>
<div
class="modal fade"
id="modal-delete{{loop.index}}"
tabindex="-1" role="dialog"
aria-labelledby="modal-delete{{loop.index}}"
aria-hidden="true">
<div
class="modal-dialog" role="document">
<div
class="modal-content">
<div class="modal-header">
<h5 class="modal-title"
id="modal-delete{{loop.index}}">Do you want
to delete the service with the following
name</h5>
<button type="button" class="close"
data-dismiss="modal" aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
</div>
<div class="modal-body">
<form
action="/delete-service-for-connection/{{con
nection_name}}" method="POST"
id="formdelete{{loop.index}}">
<div class="form-group">
<label for="name"
class="col-form-label">Name</label>
<input type="text" name="name"
class="form-control" id="name"
value="{{item.name}}" readonly>
</div>
</form>
</div>
<div class="modal-footer">
<button type="button" class="btn
btn-secondary"
data-dismiss="modal">Close</button>
<button type="submit" class="btn
btn-primary"
form="formdelete{{loop.index}}">Delete</butt
on>
</div>
</div>
</div>
</div>

```

```

</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</body>
</html>
{% include 'footer.html' %}

```

## /frontend/frontend/routes/users.html

```

{% include 'header.html' %}
<head>
<meta charset="utf-8">
<meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?famil
y=Roboto|Varela+Round|Open+Sans">
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/boo
tstrap/4.5.0/css/bootstrap.min.css">
<link rel="stylesheet"
href="https://fonts.googleapis.com/icon?fami
ly=Material+Icons">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/font-a
wesome/4.7.0/css/font-awesome.min.css">
<script
src="https://code.jquery.com/jquery-3.5.1.mi
n.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.16.0/dist/umd/popper.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/boot
strap/4.5.0/js/bootstrap.min.js"></script>
<!-- Required meta tags -->
<meta charset="utf-8">
<meta name="viewport"
content="width=device-width,
initial-scale=1, shrink-to-fit=no">
<!-- Bootstrap CSS -->
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap
@4.3.1/dist/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH
/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then
Bootstrap JS -->
<script
src="https://code.jquery.com/jquery-3.3.1.sl
im.min.js"
integrity="sha384-q8i/X+965Dz00rT7abK41JStQI
AqVgRVzpbzo5smXKp4YfRvH+8abtTElPi6jizo"
crossorigin="anonymous"></script>

```

```

<script
src="https://cdn.jsdelivr.net/npm/popper.js@
1.14.7/dist/umd/popper.min.js"
integrity="sha384-U02eT0CpHqdSJQ6hJty5KVphtP
hzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@
4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoII
y6OrQ6VrjIEaFf/njGzIxFDs4x0xIM+B07jRM"
crossorigin="anonymous"></script>

<style>
  body {
    color: #404E67;
    background: #F5F7FA;
    font-family: 'Open Sans',
sans-serif;
  }

  .table-wrapper {
    width: 700px;
    margin: 30px auto;
    background: #fff;
    padding: 20px;
    box-shadow: 0 1px 1px rgba(0, 0,
0, .05);
  }

  .table-title {
    padding-bottom: 10px;
    margin: 0 0 10px;
  }

  .table-title h2 {
    margin: 6px 0 0;
    font-size: 22px;
  }

  table.table {
    table-layout: fixed;
  }

  table.table tr th,
  table.table tr td {
    border-color: #e9e9e9;
  }

  table.table th i {
    font-size: 13px;
    margin: 0 5px;
    cursor: pointer;
  }

  table.table td i {
    font-size: 19px;
  }
</style>
</head>

<body>
  <h2>Users</h2>
  <div class="container-lg">
    <div class="table-responsive">
      <div class="table-wrapper">
        <div class="table-title">
          <div class="row">

```

```

          <div
class="col-sm-8">

          </div>
        </div>
      </div>
      <table class="table
table-bordered">
        <thead>
          <tr>
            <th>Name</th>
            <th>ID</th>
          </tr>
        </thead>
        <tbody>
          {% for item in
dictFromServer.users %}
            <tr>

              <td>{{item.name}}</td>

              <td>{{item._id}}</td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
    </div>
  </div>
</body>

</html>

{% include 'footer.html' %}

```

### /frontend/frontend/server.py

```

import json, os, sys
import redis
from flask import Flask, session
from flask_session import Session

import routes

app = Flask(__name__)

app.config['SESSION_TYPE'] = 'redis'
app.config['SESSION_REDIS'] =
redis.from_url('redis://redis:6379')

app.config['SECRET_KEY'] =
'my-secret-awesome-key'
app.config['DEBUG'] = True

Session(app)

routes.start(app)

app.run(host='0.0.0.0')

```

### /frontend/Dockerfile

```

FROM python:3.10.2

WORKDIR /opt/app

```

```

COPY . /opt/app/
RUN ls -alh
RUN pip install -r frontend/requirements.txt
RUN pip install requests

ENTRYPOINT ["python3", "frontend/server.py"]

```

### **/frontend/run.sh**

```

docker build -t frontend .
docker rm -f frontend || true
docker network create gw || true

```

```

docker run -d \
--net gw \
--link redis:redis \
--name frontend \
-p 5004:5000 \
frontend

```

### **/gateway\_kong/migrate-db.sh**

```

docker network create gw || true

```

```

docker run --rm \
--net gw \
--link gw-kongdb:gw-kongdb \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=gw-kongdb" \
-e "KONG_PG_USER=kong" \
-e "KONG_PG_PASSWORD=kong" \
-e
"KONG_CASSANDRA_CONTACT_POINTS=gw-kongdb" \
kong kong migrations bootstrap

```

### **/gateway\_kong/run-db.sh**

```

docker network create gw || true

```

```

docker rm -f gw-kongdb || true
docker run -d --name gw-kongdb \
--net gw \
-e "POSTGRES_USER=kong" \
-e "POSTGRES_DB=kong" \
-e "POSTGRES_PASSWORD=kong" \
-p 5432:5432 \
-v
/tmp/postgres-data:/var/lib/postgresql/data
\
postgres:9.6

```

### **/gateway\_kong/run-kong.sh**

```

docker network create gw || true

```

```

docker rm -f gw-kong || true
docker run -d --name gw-kong \
--net gw \
--link gw-kongdb:gw-kongdb \
-e "KONG_DATABASE=postgres" \
-e "KONG_PG_HOST=gw-kongdb" \
-e "KONG_PG_PASSWORD=kong" \
-e
"KONG_CASSANDRA_CONTACT_POINTS=gw-kongdb" \
-e "KONG_PROXY_ACCESS_LOG=/dev/stdout" \
-e "KONG_ADMIN_ACCESS_LOG=/dev/stdout" \

```

```

-e "KONG_PROXY_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_ERROR_LOG=/dev/stderr" \
-e "KONG_ADMIN_LISTEN=0.0.0.0:8001,
0.0.0.0:8444 ssl" \
-p 8000:8000 \
-p 8443:8443 \
-p 8001:8001 \
-p 8444:8444 \
kong

```

### **/session\_db/run.sh**

```

docker rm -f redis || true
docker network create gw || true

```

```

docker run -td \
--net gw \
--name redis \
-p 6379:6379 \
redis

```

### **/tyk/run.sh**

```

docker rm -f gw-tyk || true
docker network create gw || true

```

```

docker run -d \
--name gw-tyk \
--network gw \
--link redis:redis \
-p 8080:8080 \
-v
$(pwd)/tyk.standalone.conf:/opt/tyk-gateway/
tyk.conf \
-v $(pwd)/apps:/opt/tyk-gateway/apps \

```

```

docker.tyk.io/tyk-gateway/tyk-gateway:latest

```

### **/do.sh**

```

cd gateway_kong
./run-db.sh
./migrate-db.sh
./run-kong.sh
./run-konga.sh
cd ..
cd api1
./run.sh
cd ..
cd api2
./run.sh
cd ..
cd api3
./run.sh
cd ..
cd db
./run-db.sh
./run-express.sh
cd ..
cd api_management
./run.sh
cd ..
cd session_db
./run.sh
cd ..
cd frontend

```

```
./run.sh  
cd ..  
cd tyk
```

```
./run.sh  
cd ..
```