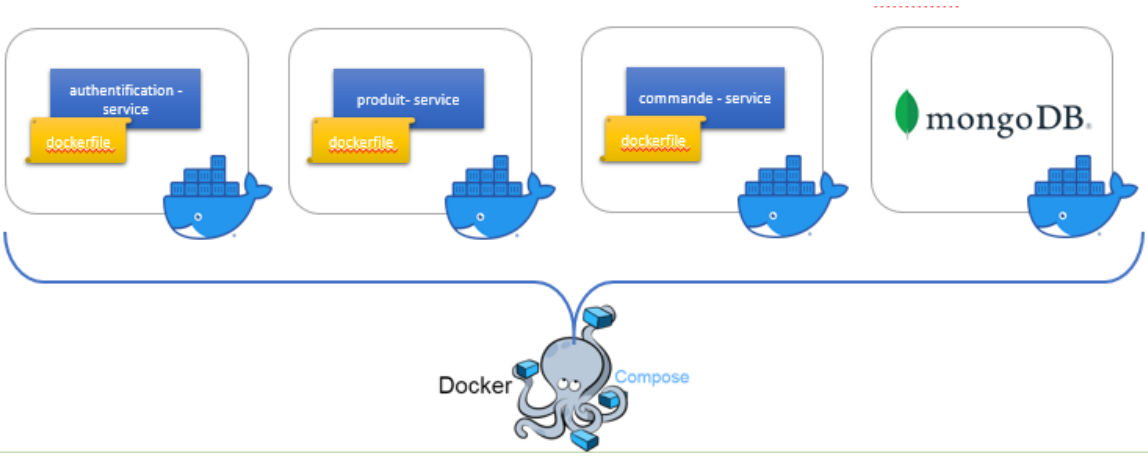
Filière: Développement Digitale option Web Full Stack	

Retournons à notre exemple de cours où on gérait trois services: **authentification**, **produit** et **commande**.

Chaque service aura son propre fichier **dockerfile** où on indiquera les instructions à suivre pour créer l'image docker correspondante.

En plus des trois conteneurs « service », on ajoutera un quatrième qui servira à conteneuriser le SGBD mongoDB.



Microservice Authentification

Dockefile :

```
FROM node:latest
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
CMD [ "npm", "run", "start" ]
```

.dockerignore :

```
node_modules
```

Microservice Produit

Dockerfile :

```
FROM node:latest
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
CMD [ "npm", "run", "start" ]
```

.dockerignore :

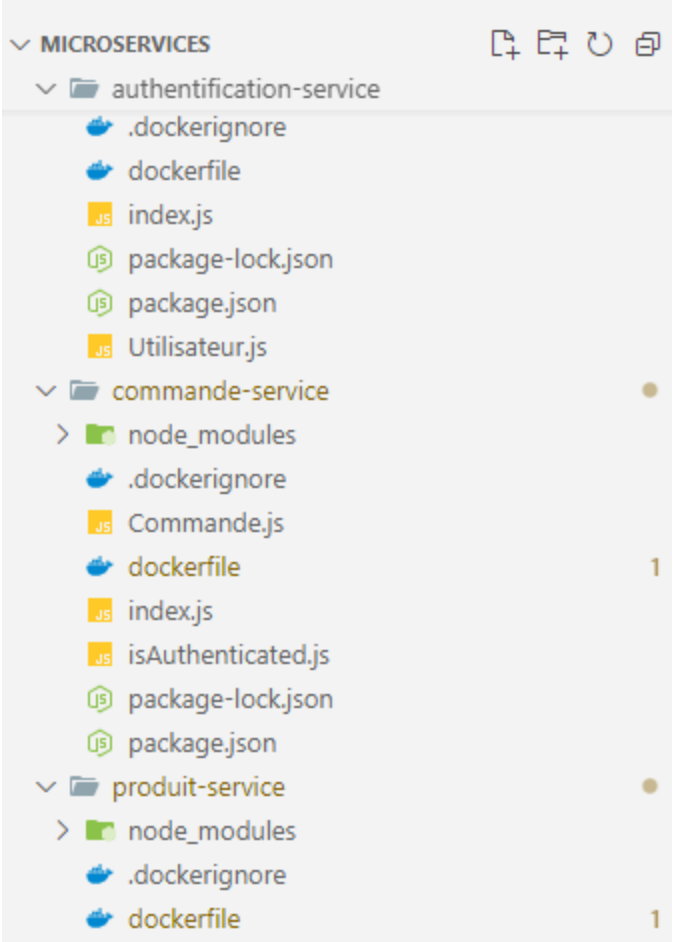
```
node_modules
```

Microservice Commande

```
FROM node:latest
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
CMD [ "npm", "run", "start" ]
```

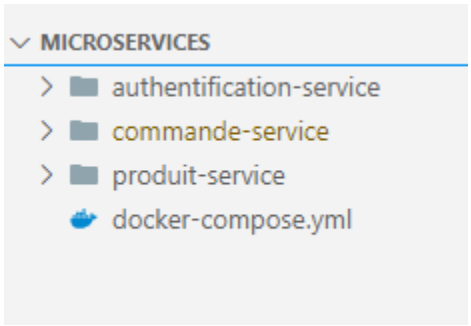
.dockerignore :

```
node_modules
```



Docker compose

Le contenu du fichier docker-compose.yml est présenté ci-après :



```yaml
version: '3.9'
services:
  db:
    container_name: db
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"

  produits:
    build: ./produit-service
    container_name: produit-service
    ports:
      - "5000:4000"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  authentification:
    build: ./authentification-service
    container_name: authentification-service
    ports:
      - "5002:4002"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  commande:
    build: ./commande-service
    container_name: commande-service
    ports:
      - "5001:4001"
    volumes:
      - /app/node_modules
    depends_on:
      - authentification
      - produits
      - db
```

Pour se connecter à la base de données du service « **db** », on doit modifier les chaines de connexion de tous les services de :

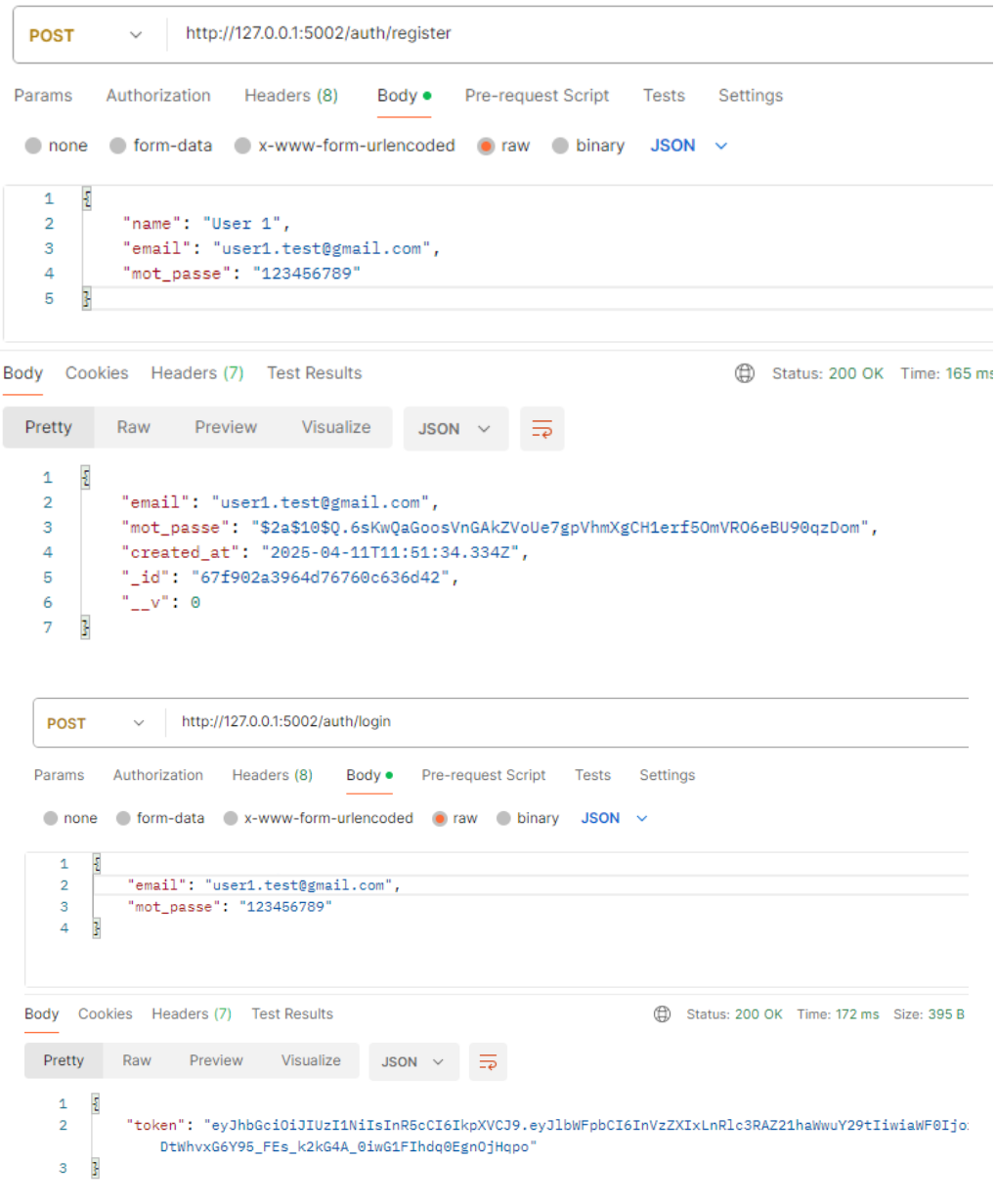
"mongodb://localhost/produit-service" à **"mongodb://db:27017/produit-service »**

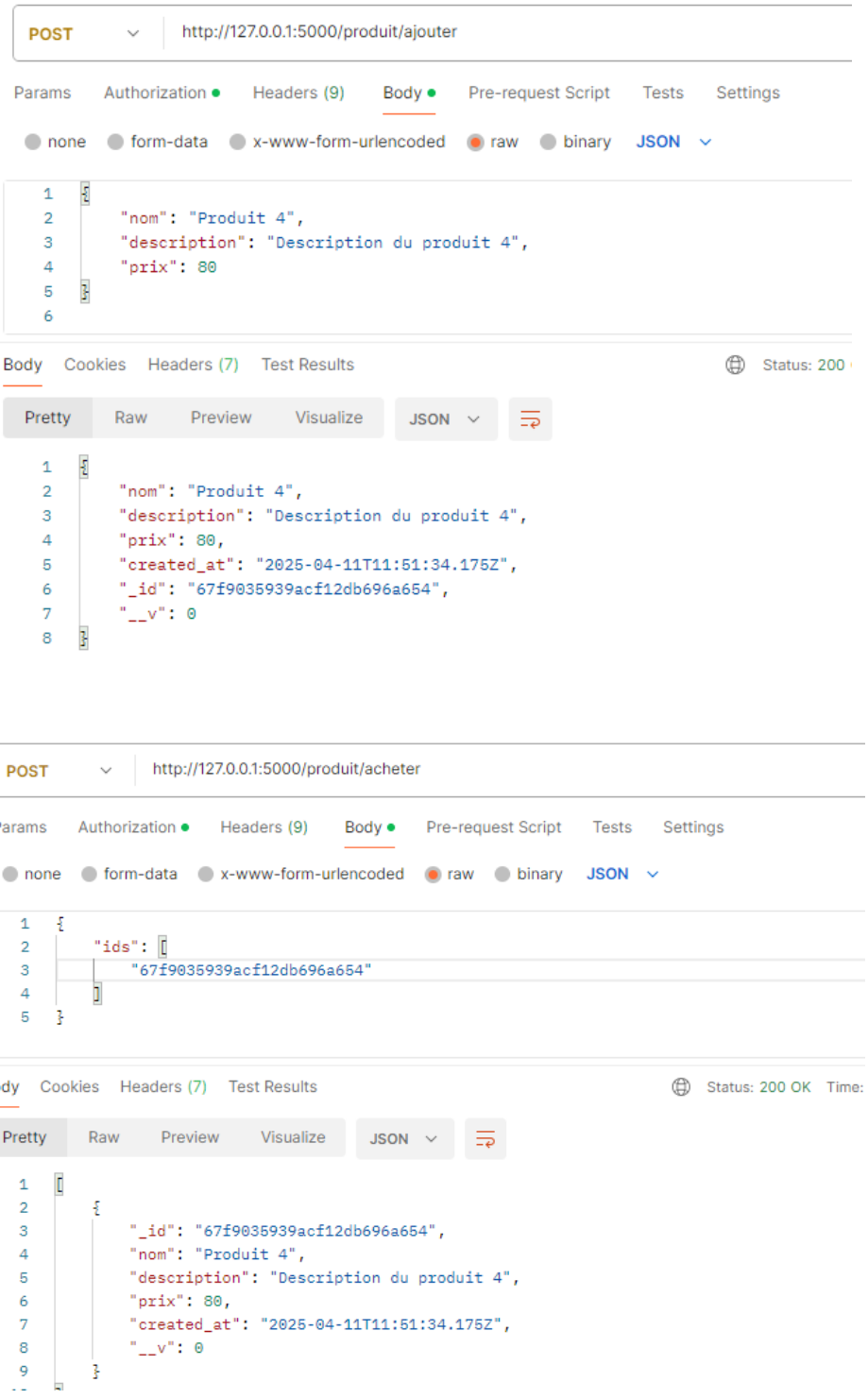
A l'aide des commandes : **docker-compose build** et **docker-compose up**, on peut construire les images et démarrer les conteneurs :

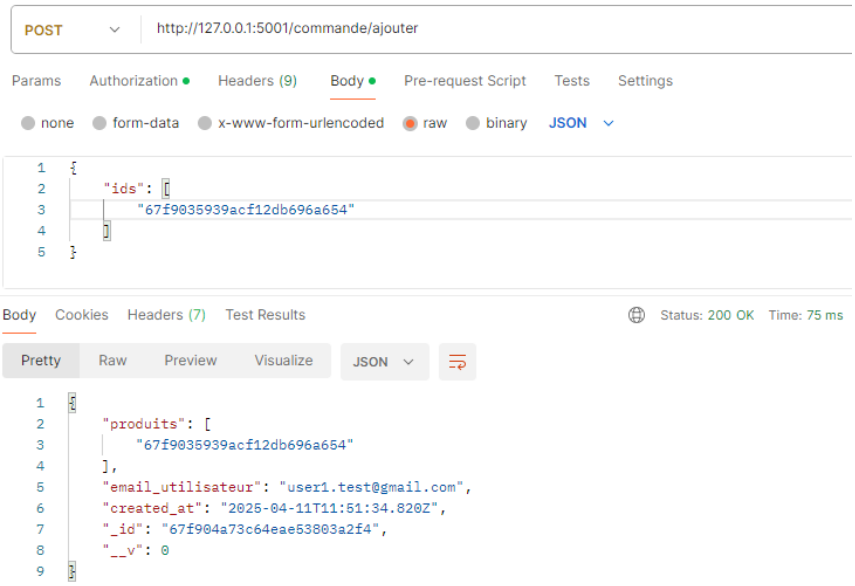
```
 => => exporting layers
 => => writing image sha256:bdd004f40c54b5374c9ea69b998daaaf785b2936635f945e86474639a56786a6
 => => naming to docker.io/library/microservices-produits
 => [authentification] exporting to image
 => => exporting layers
 => => writing image sha256:ea98c59f4ead4513b6cd5bf06e7b6c0e1be3bd02c48d9fbe901003b94863fb19
 => => naming to docker.io/library/microservices-authentification
 => [produits] resolving provenance for metadata file
 => [authentification] resolving provenance for metadata file
 => [commande internal] load build definition from dockerfile
 => => transferring dockerfile: 150B
 => [commande internal] load .dockerignore
 => => transferring context: 52B
 => [commande internal] load build context
 => => transferring context: 230B
 => CACHED [commande 3/5] COPY package*.json .
 => CACHED [commande 4/5] RUN npm install
 => CACHED [commande 5/5] COPY . .
 => [commande] exporting to image
 => => exporting layers
 => => writing image sha256:3b81e4f43e6d97859b310882c676a2735a18c3855af777b77a9e952314b29df6
 => => naming to docker.io/library/microservices-commande
 => [commande] resolving provenance for metadata file
[+] Building 3/3
 ✓ authentification  Built
 ✓ commande          Built
 ✓ produits          Built
```

```
[+] Running 5/5
 ✓ Network microservices_default       Created
 ✓ Container db                        Created
 ✓ Container produit-service           Created
 ✓ Container authentification-service  Created
 ✓ Container commande-service          Created
Attaching to authentification-service, commande-service, db, produit-service
db                     | {"t":{"$date":"2025-04-11T11:34:52.604+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   '
"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
commande-service       |
authentification-service |
produit-service        |
db                     | {"t":{"$date":"2025-04-11T11:34:52.625+00:00"},"s":"I",  "c":"CONTROL",  "id":5945603, '
"Multi threading initialized"}
commande-service       | > commande-service@1.0.0 start
authentification-service | > authentification-service@1.0.0 start
produit-service        | > produit-service@1.0.0 start
commande-service       | > nodemon index.js
```

☐	⌄	●	microservices	-	-	-	0.89%	2 minutes ag
☐		●	db	017301828a4e	mongo	27017:27017	0.89%	2 minutes ag
☐		●	authentification-service	bcf62723b8fe	microservic	5002:4002	0%	2 minutes ag
☐		●	produit-service	91eb04849735	microservic	5000:4000	0%	2 minutes ag
☐		●	commande-service	7265571cc242	microservic	5001:4001	0%	2 minutes ag


POST
http://127.0.0.1:5002/auth/register
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{
"name": "User 1",
"email": "user1.test@gmail.com",
"mot_passe": "123456789"
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 165 ms
Pretty
Raw
Preview
Visualize
JSON
{
"email": "user1.test@gmail.com",
"mot_passe": "$2a$10$Q.6sKwQaGoosVnGAkZVoUe7gpVhmXgCH1erf5OmVRO6eBU90qzDom",
"created_at": "2025-04-11T11:51:34.334Z",
"_id": "67f902a3964d76760c636d42",
"__v": 0
}
POST
http://127.0.0.1:5002/auth/login
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{
"email": "user1.test@gmail.com",
"mot_passe": "123456789"
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 172 ms
Size: 395 B
Pretty
Raw
Preview
Visualize
JSON
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InVzZXIxLnRlc3RAZ21haWwuY29tIiwiaWF0Ijo
DtWhvxG6Y95_FEs_k2kG4A_0iwG1FIhdq0EgnOjHqpo"
}


POST
http://127.0.0.1:5000/produit/ajouter
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{
"nom": "Produit 4",
"description": "Description du produit 4",
"prix": 80
}
Body
Cookies
Headers (7)
Test Results
Status: 200
Pretty
Raw
Preview
Visualize
JSON
{
"nom": "Produit 4",
"description": "Description du produit 4",
"prix": 80,
"created_at": "2025-04-11T11:51:34.175Z",
"_id": "67f9035939acf12db696a654",
"__v": 0
}
POST
http://127.0.0.1:5000/produit/acheter
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{
"ids": [
"67f9035939acf12db696a654"
]
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time:
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "67f9035939acf12db696a654",
"nom": "Produit 4",
"description": "Description du produit 4",
"prix": 80,
"created_at": "2025-04-11T11:51:34.175Z",
"__v": 0
}



Tapons la commande **mongosh** (Shell permettant la manipulation du SGBD MongoDB)

-La commande « **show databases** » permet d'afficher la liste des collections disponible:

```
test> show databases
admin                   40.00 KiB
authentification-service  40.00 KiB
commandes-service       40.00 KiB
config                 108.00 KiB
local                   72.00 KiB
produits-service        40.00 KiB
```