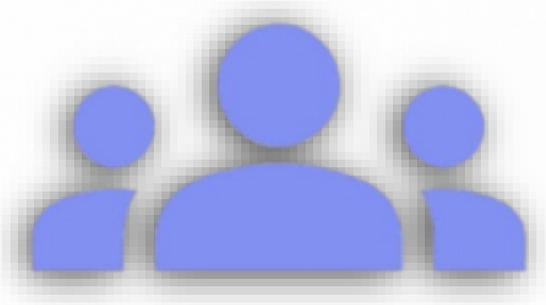
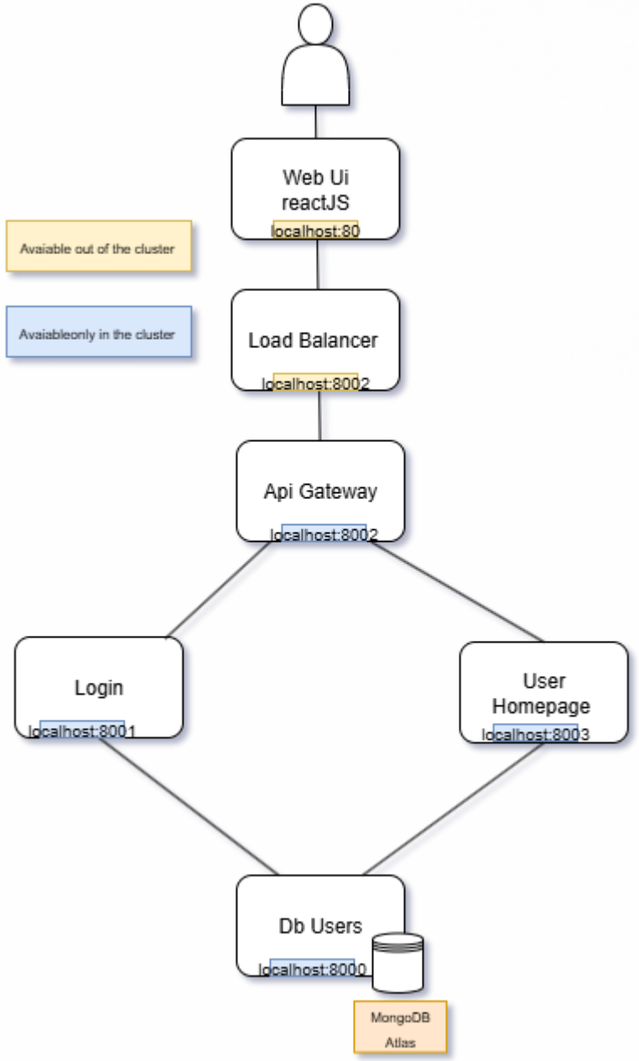
Let Me Know



Student: Francesco Astolfi



Kubernetes Microservices Architecture Demo

A comprehensive demonstration of a FastAPI-based microservices application deployed on Kubernetes, showcasing modern containerisation and orchestration practices.

Project Requirements & Configuration

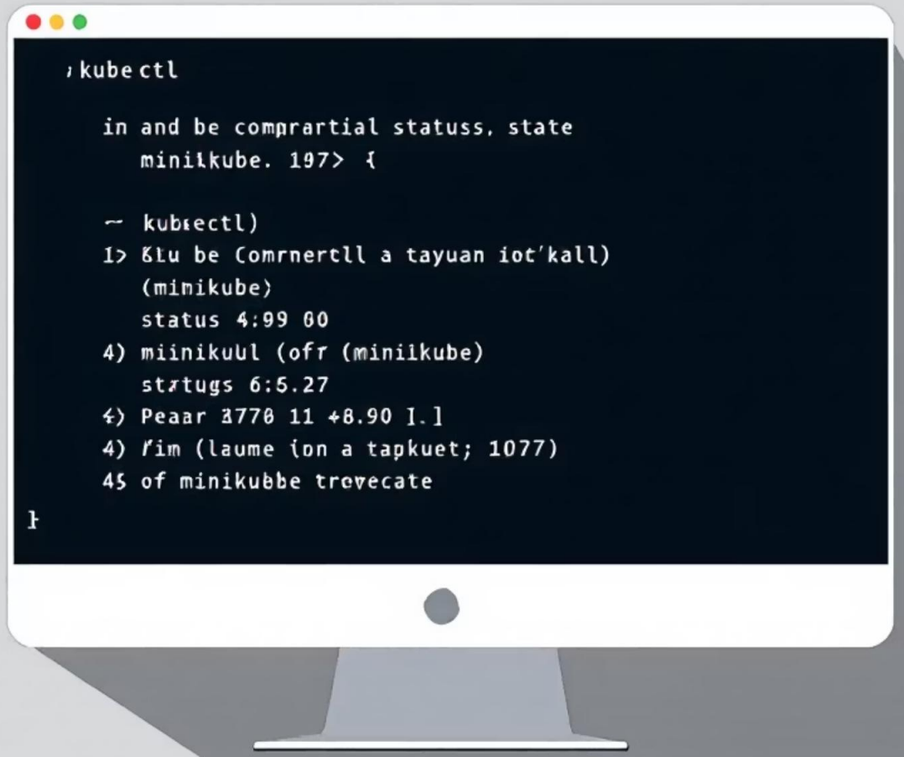
Kubernetes Setup

kubectl or minikube installed for container orchestration and cluster management. For the local deploy I used os Windows, DockerDesktop with Kubernetes enabled.

Repository Access

Clone the project repository containing all microservice components and configuration files:

https://github.com/FrancescoAstolfiDev/CloudComputingAssignment_v1.git



Backend
FastAPI Framework Benefits

Why FastAPI?

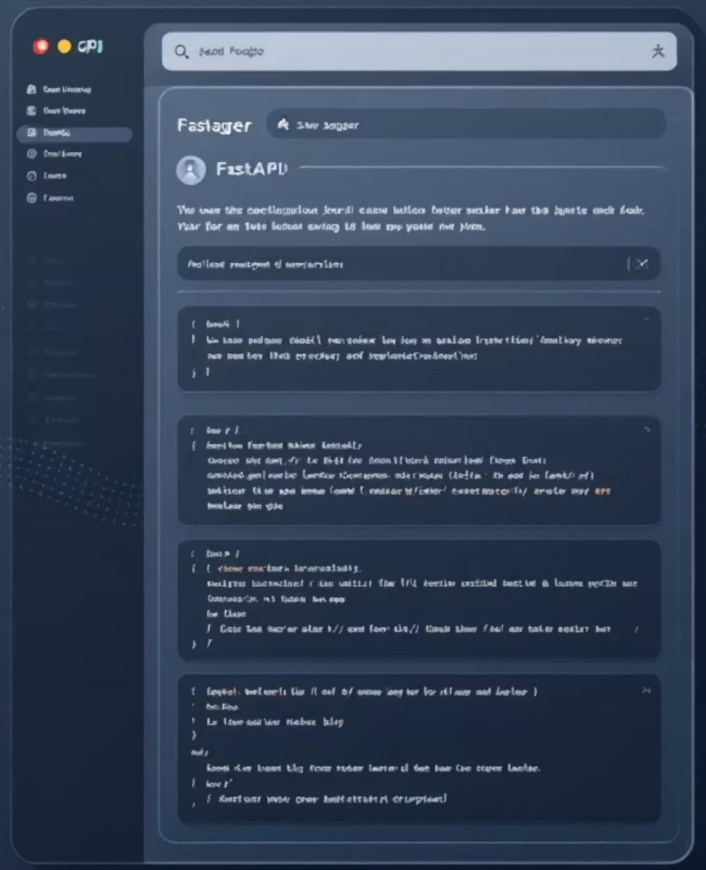
- *High performance with async support*
- *Automatic API documentation*
- *Type hints and validation*
- *Standards-based OpenAPI*

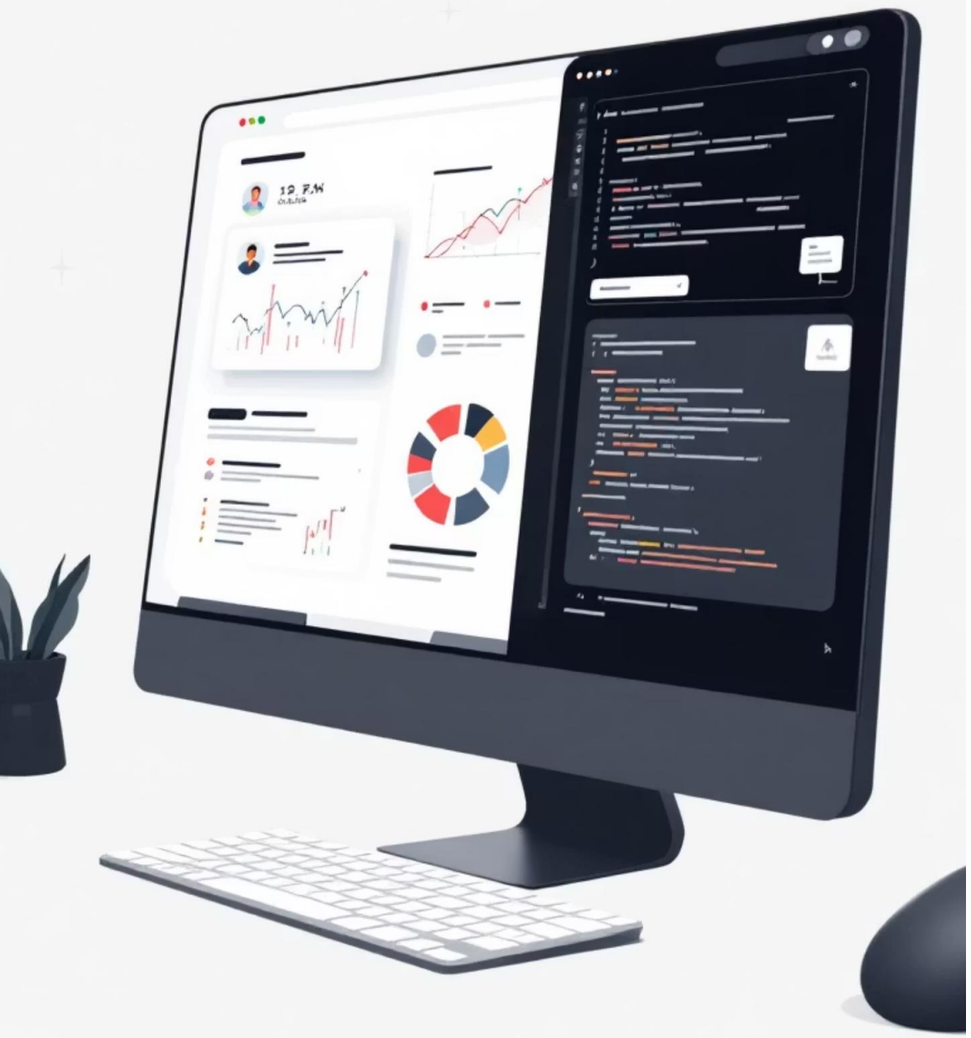
Uvicorn Integration

ASGI server providing lightning-fast performance with command:

`uvicorn main:app --reload`

Built-in documentation available at `/docs` *endpoint*





React Framework Benefits

Component-Based Architecture

Develop reusable UI components, fostering modularity and simplifying development for complex microfrontend structures.

Virtual DOM Efficiency

Experience faster updates and improved performance through React's efficient reconciliation process, leading to a smoother user experience.

Rich Ecosystem & Community

Leverage an extensive collection of libraries, tools, and strong community support for frontend development and problem-solving.

Declarative UI

Write more predictable and easier-to-debug code with React's declarative approach to building user interfaces.

Architectural Overview

01

Directory Structure

Organised service modules with clear separation of concerns and configuration files

02

Dockerfile Configuration

File for the provisioning for each service and optimizing the container size

03

YAML Manifests

Kubernetes deployment configurations defining pods, services, and ingress rules

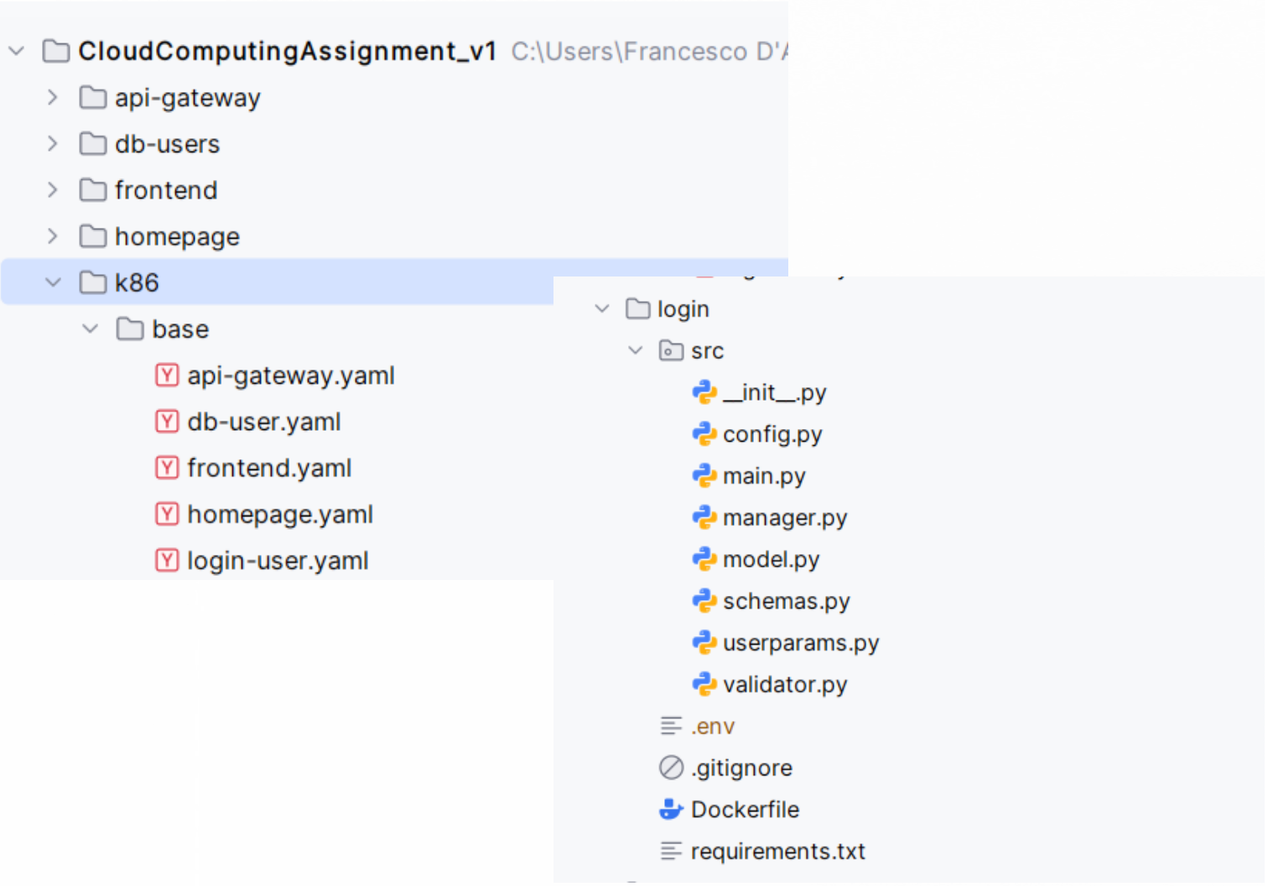
Architectural Overview

01

Directory Structure

Organised service modules with clear separation of concerns and configuration files.

Api and all the content to copy inside the microservice is inside the directory src.



Architectural Overview

02

Dockerfile Configuration

File for the provisioning for each service and optimizing the container size.

The docker files in the current application differs each other just for the door where Uvicorn is listening.

The only different dockerfile is in the for the frontend. One stage for provisioning the react frontend and the other node for listen the http request from the client.

1. **Docker Build**

   `docker build -t myapp:latest .`

2. **Image Push**

   `docker push username/myapp:latest`

3. **DockerHub Registry**

   Central image repository for Kubernetes deployment

Dockerfile

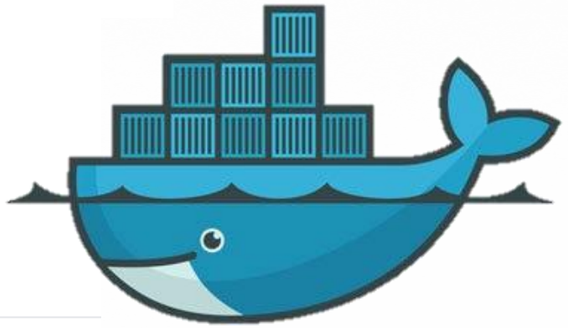
```python
FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install -r requirements.txt \
    && rm -rf /root/.cache/pip

COPY . .
EXPOSE 8001
CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port", "8001"]
```

login\Dockerfile db-users\Dockerfile

```python
FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install -r requirements.txt \
    && rm -rf /root/.cache/pip

COPY . .
EXPOSE 8000
CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Dockerfile

```python
# Stage 1: build
FROM node:20-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: serve
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

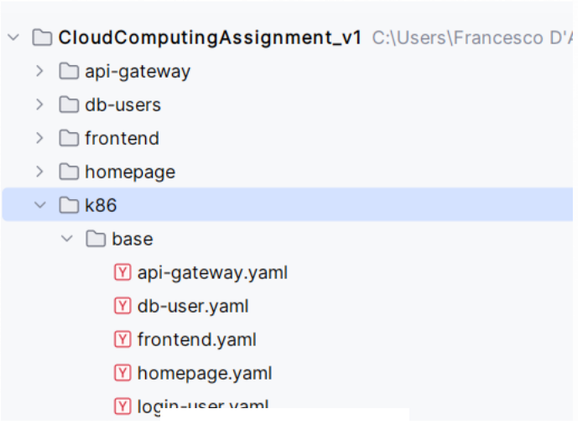


Architectural Overview

03

YAML Manifests

Kubernetes deployment configurations defining pods, services, and ingress rules. Each service has is own file of configuration .yaml.
They are in a directory instead of unique file for an easy maintain and integration.



ConfigMap

```yaml
# ConfigMap Login Service
apiVersion: v1
kind: ConfigMap
metadata:
  name: login-service-config
  labels:
    app: login-service
data:
  APP_NAME: letmeknow_login
  DB_ADDRESS: "http://db-users-service:8000/db_user"
```

Node Port

```yaml
---
# Service Login Service
apiVersion: v1
kind: Service
metadata:
  name: login-service
spec:
  selector:
    app: login-service
  ports:
    - port: 8001
      targetPort: 8001
  type: ClusterIP
```

Pods Content

```yaml
---
# Deployment Login Service
apiVersion: apps/v1
kind: Deployment
metadata:
  name: login-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: login-service
  template:
    metadata:
      labels:
        app: login-service
    spec:
      restartPolicy: Always
      initContainers:
        - name: wait-for-db
          image: busybox
          command:
            - sh
            - -c
            - |
              until nc -z db-users-service 8000; do
                echo "waiting for db..."
                sleep 2
              done
      containers:
        - name: login-service
          image: francescoastolfidev00/login-service:1.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8001
          env:
            - name: APP_NAME
              valueFrom:
                configMapKeyRef:
                  name: login-service-config
                  key: APP_NAME
            - name: DB_ADDRESS
              valueFrom:
                configMapKeyRef:
                  name: login-service-config
                  key: DB_ADDRESS
          livenessProbe:
            httpGet:
              path: /login/health
              port: 8001
            initialDelaySeconds: 5
            periodSeconds: 30
            timeoutSeconds: 2
            failureThreshold: 3
          readinessProbe:
            httpGet:
              path: /login/health
              port: 8001
            initialDelaySeconds: 5
            periodSeconds: 10
            timeoutSeconds: 2
            failureThreshold: 3
```

Application Structure

For a full overview of the onion pattern I invite u to see the source code in the Login Service

Main API Module

Central application entry point containing route definitions and middleware configuration

```python
import ...

app = FastAPI(title=settings.app_name)
# Root endpoint
@app.get("/")
async def root():
    return {"message": " Login Service"}
```

Pydantic Schemas

Type-safe request/response models ensuring data validation and serialisation

```python
class UserResponse(BaseModel):  7 usages
    user_id: str
    params: UserParams = Field(..., description="Param of the user")

    @field_validator('user_id')
    @classmethod
    def validate_user_id(cls, v):
        """Verify the len of the user_id 7 char len """
        if len(v) != 7:
            raise ValueError("user_id not valid ")
        return v

class UserCreate(BaseModel):  3 usages
    password: str
    email: EmailStr
```

Data Models

Database entity definitions with ORM mappings for persistent storage

```python
from pydantic import BaseModel, Field, EmailStr
from .userparams import UserParams

class UserInDB(BaseModel):  8 usages
    """Model that rapresent the data in the mongoDB"""
    user_id: str = Field(..., description="id of the user")
    email:EmailStr=Field(...,  description="email of the user")
    hashed_password: str = Field(..., description="crypted password")
    params: UserParams = Field(..., description="params of the user")
```

Validators & Managers

Business logic layer handling data validation and manage the core function.

Manager

```python
def uid_generator(self)-> str:  1 usage
    return ''.join(str(random.randint( a: 0,  b: 9)) for _ in range(7))

def create_user(self, user: UserInDB)->Optional[UserInDB]:  2 usages (2 dynamic)
    """create a new user in the database"""
    user_dict = user.model_dump(by_alias=True)
    hashed_password = self.pwd_context.hash(user.hashed_password)
    user_dict["hashed_password"] = hashed_password
    user_dict["user_id"] = self.uid_generator()
    user_dict["params"] = UserParams()
    return UserInDB(**user_dict)
```

Validator

```python
def password_validator(self,password):  2 usages
    # Password validation
    if len(password) < 8:
        raise ValueError("The password must be at least 8 characters long")
    if not re.search( pattern: r'[A-Z]', password):
        raise ValueError("The password must contain at least one uppercase letter")
    if not re.search( pattern: r'[a-z]', password):
        raise ValueError("The password must contain at least one lowercase letter")
    if not re.search( pattern: r'[0-9]', password):
        raise ValueError("The password must contain at least one number")
    if not re.search( pattern: r'[!@#$%^&*(),.?\":{}|<>]', password):
        raise ValueError("The password must contain at least one special character")
    return 0
```

Microservice Environment

1

Environment Variables

Service discovery through configurable endpoints and connection strings. When the service run in the pod the env is set by the configMap and secrets.

2

Settings Management

Centralised configuration using Pydantic settings for environment-specific values

Env file :

.env

```
APP_NAME=letmeknow_login
DB_ADDRESS="http://localhost:8000/db_user"
```

Config file :

config.py

```python
from typing import ClassVar
import os
from pydantic_settings import BaseSettings, SettingsConfigDict

# root path to the project directory
current_dir = os.path.dirname(os.path.abspath(__file__))
env_file_path = os.path.join(current_dir, "..", ".env")
env_file_path = os.path.abspath(env_file_path)


class Settings(BaseSettings):
    app_name: str
    db_address: str  # URL for the db

    # carica .env solo se non trovi già variabili d'ambiente
    model_config: ClassVar[SettingsConfigDict] = SettingsConfigDict(env_file=env_file_path)

# Settings() use the .env file if no ENV vars are set
# otherway use the ENV vars
settings = Settings()
```

Use of the variable :

```python
from .config import settings

app = FastAPI(title=settings.app_name)

async with httpx.AsyncClient() as client:
    resp = await client.get(settings.db_address, params={"identifier": request.identifier})
```

Backend Test Methods

PowerShell/Uvicorn

Direct development server launch with live documentation and API testing via Postman

Docker Run

Containerised deployment with port mapping and environment configuration

Kubernetes Cluster

Production-ready orchestration with ConfigMaps, Ingress controllers, and container management

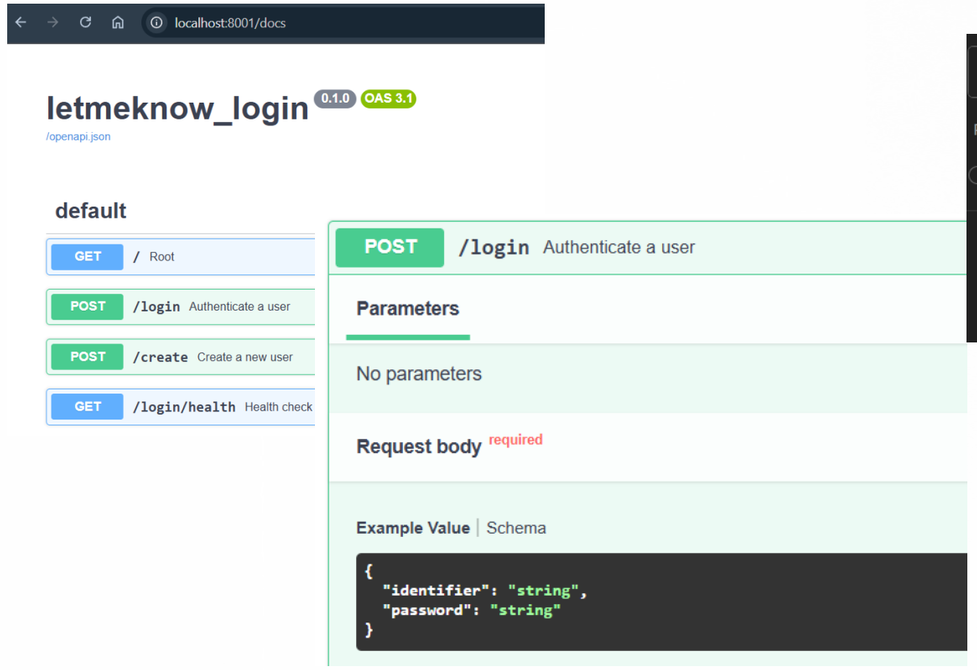
Backend Test Methods

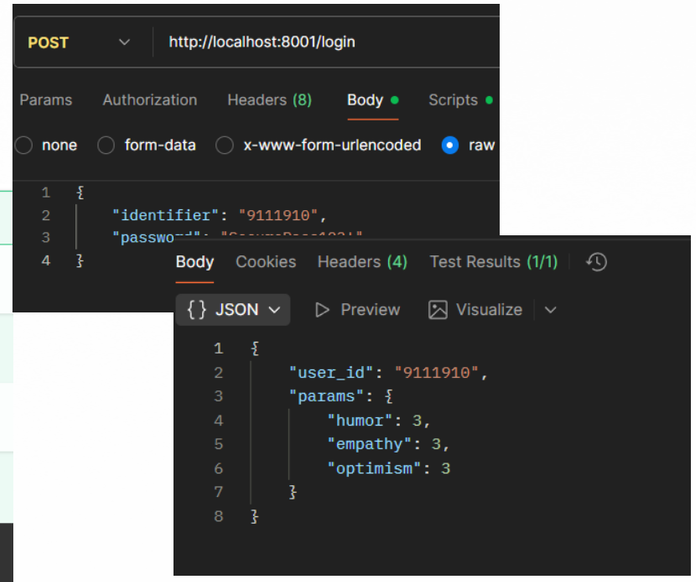
PowerShell/Uvicorn

Direct development server launch with live documentation and API testing via Postman.
No deploy needed but is an easy way for debug, begin complex when more service and call each other, in case of the login service-db must be run and login also.
The user must orchestrate and manage the door for the services.

```
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1\db-users> uvicorn src.main:app --host 0.0.0.0 --port 8000
Using .env file: C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1\db-users\.env
INFO:     Started server process [16024]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
✅ Connected with mongoDB!
INFO:     127.0.0.1:61925 - "GET /db_user?identifier=9111910 HTTP/1.1" 200 OK
```

```
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1\Login> uvicorn src.main:app --host 0.0.0.0 --port 8001
INFO:     Started server process [31160]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
prima della verifica delle password
True
out ottenuto dalla richiesta del db
{'user_id': '9111910', 'email': 'usertest1223@example.com', 'hashed_password': '$2b$12$MAGi3SMZP71lUYNVt5nVNeC4gkqeRY0saYH2X29Kt8NtxYIklKhGW', 'params': {'h
umor': 3, 'empathy': 3, 'optimism': 3}}
INFO:     127.0.0.1:61923 - "POST /login HTTP/1.1" 200 OK
```





Backend Test Methods

Docker Run

Containerised deployment with port mapping and environment configuration

That solution is not applicable with the current configuration, for make it work togheter the container must be in the same network each other! They need an orchestrator.

db-users-service

4bfae53e615d francescoastolfidev00/db-users-service:1.0

8000:8000

Logs | Inspect | Bind mounts | Exec | Files | Stats

```
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
Using .env file: /app/.env
INFO:     172.18.0.1:56848 - "GET /docs HTTP/1.1" 200 OK
INFO:     172.18.0.1:56848 - "GET /openapi.json HTTP/1.1" 200 OK
```

Containers / login-service

login-service

dc9f899db1a6 francescoastolfidev00/login-service:1.0

8001:8001

Logs | Inspect | Bind mounts | Exec | Files | Stats

```
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
INFO:     172.18.0.1:37328 - "GET /docs HTTP/1.1" 200 OK
INFO:     172.18.0.1:37328 - "GET /openapi.json HTTP/1.1" 200 OK
❌ Communication error to the DB service: ConnectError - All connection attempts failed
INFO:     172.18.0.1:36378 - "POST /login HTTP/1.1" 500 Internal Server Error
INFO:     172.18.0.1:37340 - "GET /docs HTTP/1.1" 200 OK
INFO:     172.18.0.1:37340 - "GET /openapi.json HTTP/1.1" 200 OK
```

Used Commands

1. Build the name space

```
kubectl create namespace letmeknow
```

2. Apply the manifest

```
kubectl apply -n letmeknow -f k86/base/
```

Get status of pods

```
kubectl get -n letmeknow pods
```

Get logs of a single pod

```
kubectl logs -n letmeknow api-gateway-service-576877f8b4-brs8v
```

3. CleanUp

```
kubectl delete namespace letmeknow
```

Kubernetes Cluster

Production-ready orchestration with ConfigMaps, Ingress controllers, and container management

Backend Test Methods

```
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1> kubectl create namespace letmeknow
Error from server (AlreadyExists): namespaces "letmeknow" already exists
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1> kubectl apply -n letmeknow -f k86/base/
configmap/api-gateway-config created
deployment.apps/api-gateway-service created
service/api-gateway-service created
secret/db-users-credentials created
configmap/db-users-config created
deployment.apps/db-users-service created
service/db-users-service created
Warning: resource namespaces/letmeknow is missing the kubectl.kubernetes.io/last-applied-configuration annotation which is required by kubectl apply. kubectl apply should
 only be used on resources created declaratively by either kubectl create --save-config or kubectl apply. The missing annotation will be patched automatically.
namespace/letmeknow configured
configmap/react-frontend-config created
deployment.apps/react-frontend created
service/react-frontend-lb created
configmap/homepage-service-config created
deployment.apps/homepage-service created
service/homepage-service created
configmap/login-service-config created
deployment.apps/login-service created
service/login-service created
```

```
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1> kubectl get -n letmeknow pods
NAME                                  READY   STATUS    RESTARTS   AGE
api-gateway-service-576877f8b4-4xkb2  1/1     Running   0          80s
db-users-service-67f9457984-q28p9     1/1     Running   0          80s
homepage-service-6d59ffcf86-gxfhf     1/1     Running   0          80s
login-service-c5c57995c-4lg9q         1/1     Running   0          79s
react-frontend-b6585cf7c-5v7wf        1/1     Running   0          80s
react-frontend-b6585cf7c-xvs8r        1/1     Running   0          80s
PS C:\Users\Francesco D'Amata\PycharmProjects\CloudComputingAssignment_v1>
```

k8s_login-service_login-service-c5c57995c-4lg9q_letmeknow_da835437-cf25-4ed3-959d-43879c48921b_0

6272405aa035 francescoastolfidev00/login-service:1.0

STATUS
Running

Logs | Inspect | Bind mounts | Exec | Files | Stats

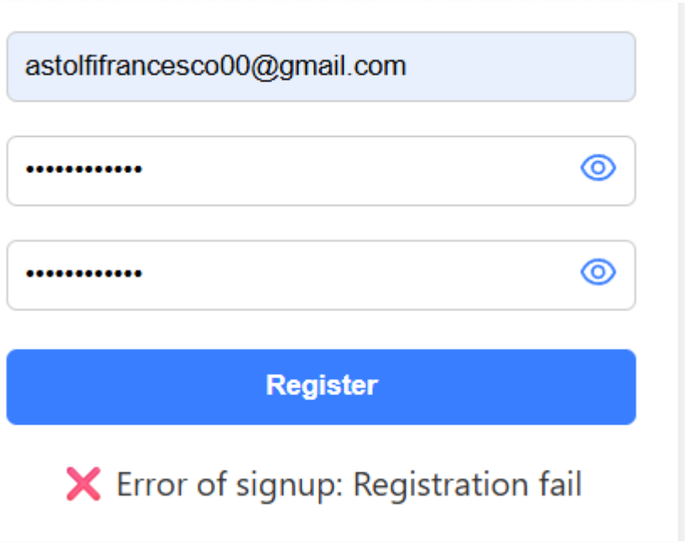
```
INFO:     10.1.0.1:34250 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:55208 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:43180 - "GET /login/health HTTP/1.1" 200 OK
(trapped) error reading bcrypt version
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/site-packages/passlib/handlers/bcrypt.py", line 620, in _load_backend_mixin
    version = _bcrypt.__about__.__version__
              ^^^^^^^^^^^^^^^^^
AttributeError: module 'bcrypt' has no attribute '__about__'
prima della verifica delle password
True
out ottenuto dalla richiesta del db
{'user_id': '1901850', 'email': 'userexample3@gmail.com', 'hashed_password': '$2b$12$ljuCK8lFw7ngSiHkcjTZwO3n0Lern1Qq72
3, 'empathy': 3, 'optimism': 3}}
INFO:     10.1.0.65:36660 - "POST /login HTTP/1.1" 200 OK
INFO:     10.1.0.1:55016 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:55028 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:51604 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:50132 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:60786 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:60792 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:56450 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:53616 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:46582 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:46598 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:48412 - "GET /login/health HTTP/1.1" 200 OK
```

Connect to Frontend

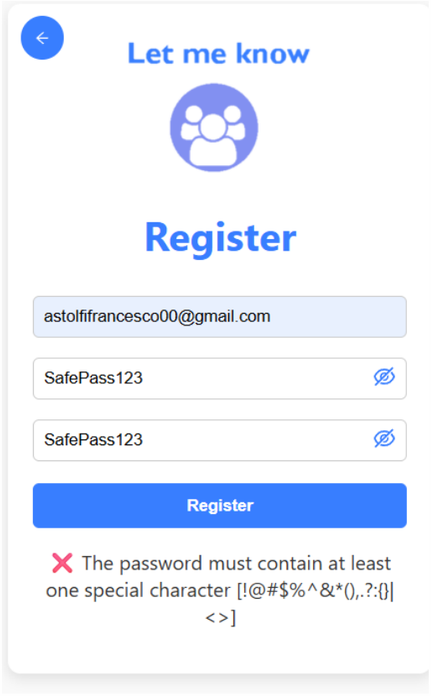
Use the connection to localhost:80 after the build of the manifest in the previous slides.

Registration Screen

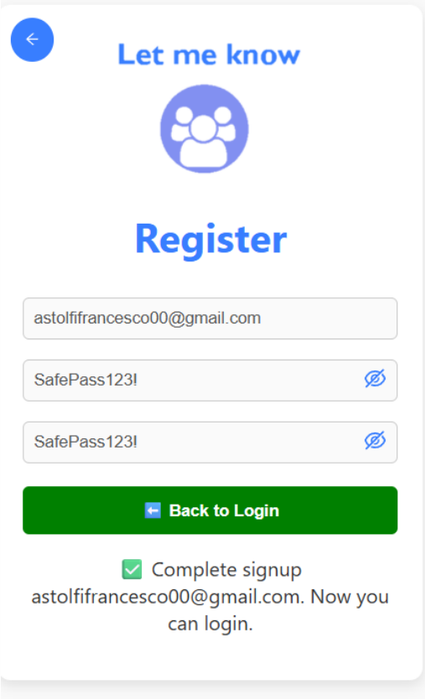
Fail in case of reuse email



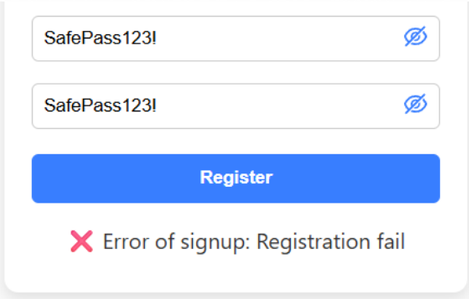
Creation of a User



Correct response User

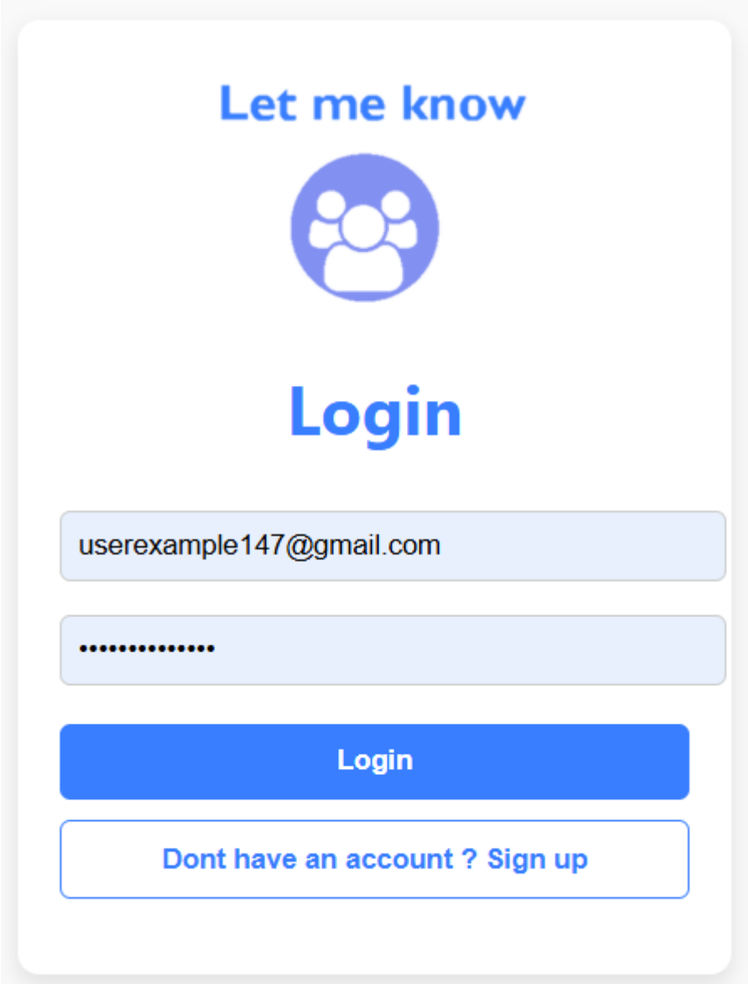


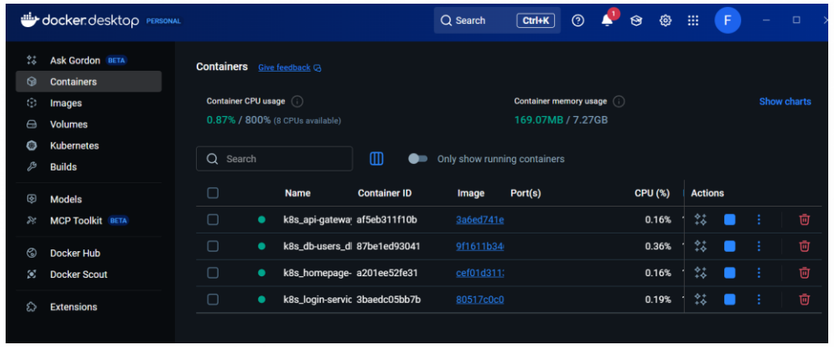
Check Password



Login Screen

Login of a User





Mongo DB- Entry :


```
_id: "7346726"
email : "astolfifrancesco00@gmail.com"
hashed_password : "$2b$12$NpwUclOPrVSmvMVZ9eXUy.HKLo1TJQinyzwNfPJTWTUQz89Nv.PWW"
▾ params : Object
    humor : 3
    empathy : 3
    optimism : 3
```


```
INFO:     192.168.65.3:35736 - "OPTIONS /login/user HTTP/1.1" 200 OK
INFO:     192.168.65.3:35736 - "POST /login/user HTTP/1.1" 200 OK
INFO:     10.1.0.1:38830 - "GET /APIgateway/health HTTP/1.1" 200 OK
INFO:     192.168.65.3:35736 - "GET /user?identifier=7346726 HTTP/1.1" 200 OK
```

Gateway

```
INFO:     10.1.0.1:50752 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:38012 - "GET /login/health HTTP/1.1" 200 OK
prima della verifica delle password
True
out ottenuto dalla richiesta del db
{'user_id': '7346726', 'email': 'astolfifrancesco00@gmail.com', 'hashed_password':
'$2b$12$NpwUclOPrVSmvMVZ9eXUy.HKLo1TJQinyzwNfPJTWTUQz89Nv.PWW', 'params': {'humor': 3, 'empathy': 3, 'optimism': 3}}
INFO:     10.1.0.18:53282 - "POST /login HTTP/1.1" 200 OK
INFO:     10.1.0.1:33534 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:33550 - "GET /login/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:40376 - "GET /login/health HTTP/1.1" 200 OK
```

Login Service

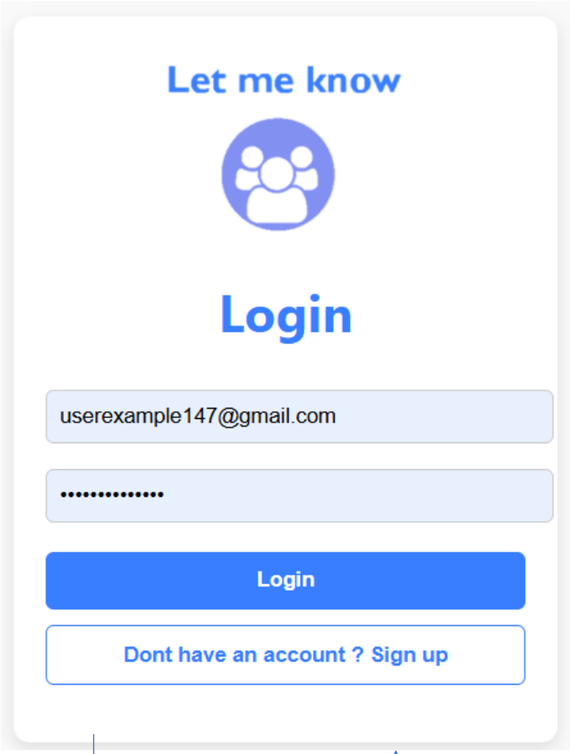
```
INFO:     10.1.0.1:39410 - "GET /db_user/health HTTP/1.1" 200 OK
INFO:     10.1.0.23:49556 - "GET /db_user?identifier=astolfifrancesco00%40gmail.com HTTP/1.1" 200 OK
INFO:     10.1.0.21:38694 - "GET /db_user?identifier=7346726 HTTP/1.1" 200 OK
INFO:     10.1.0.1:53564 - "GET /db_user/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:39586 - "GET /db_user/health HTTP/1.1" 200 OK
INFO:     10.1.0.1:46624 - "GET /db_user/health HTTP/1.1" 200 OK
```

Db Users Service

Homepage Screen

For this screen is used a session storage in Redux that enable the frontend to save the userid of the logged user and reuse it for the api of get on the homepage.

Login of a User

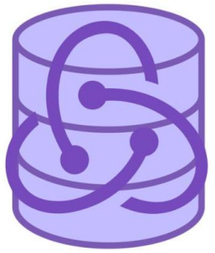


{email or uid, password }

200: uid

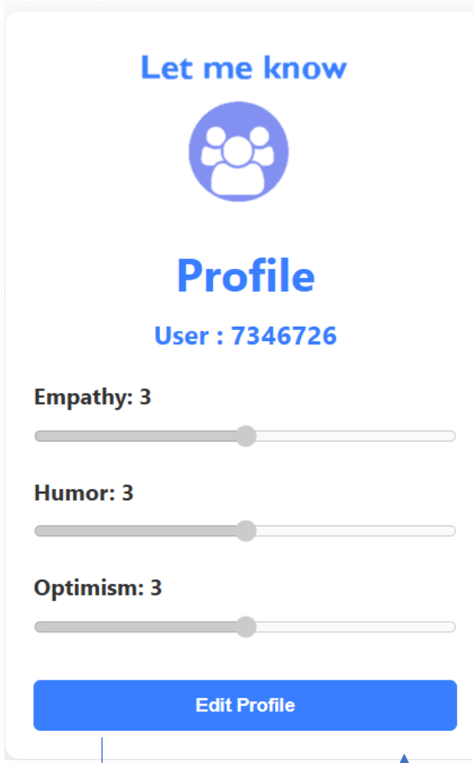
Post

Api Gateway



Store Userid

Homepage of a User



{userid}

{userid {params}}

Get

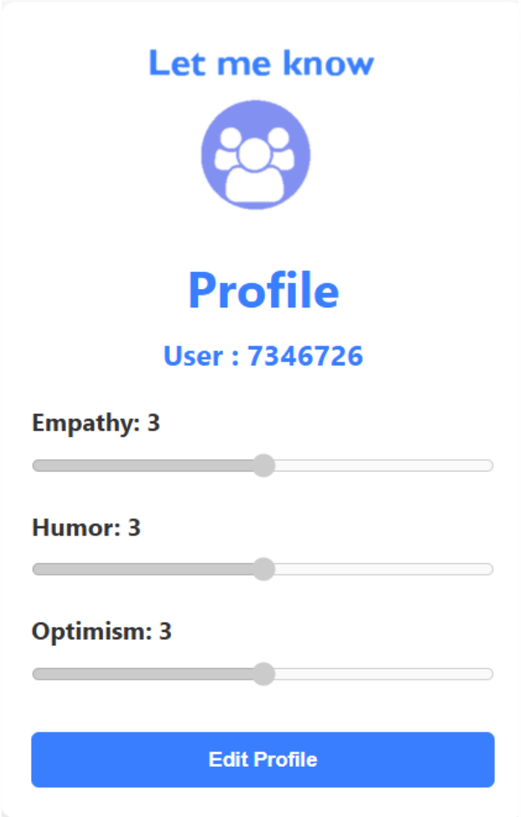
Api Gateway

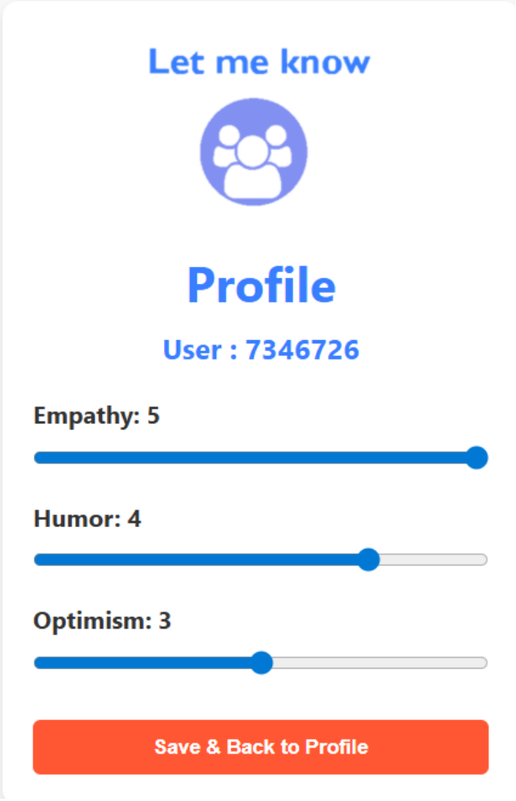
Edit Profile Screen

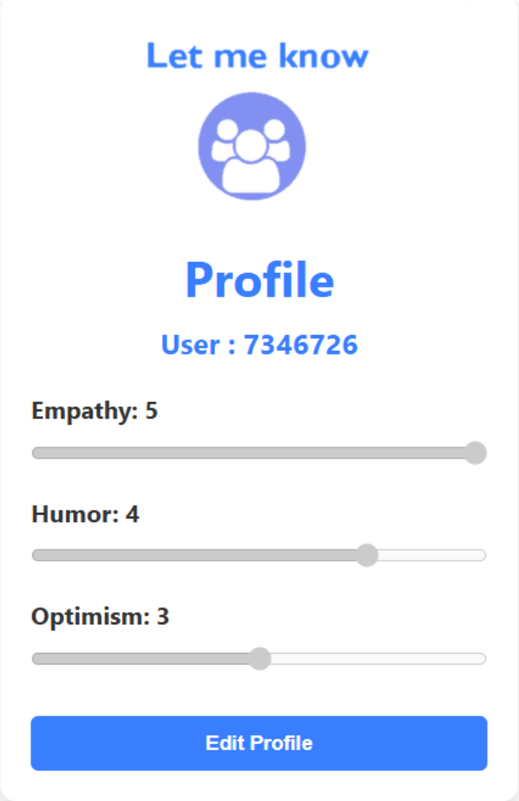
The data are stored in a persistent way in mongodb atlas.

Each edit on the profile will be consistent on the database.

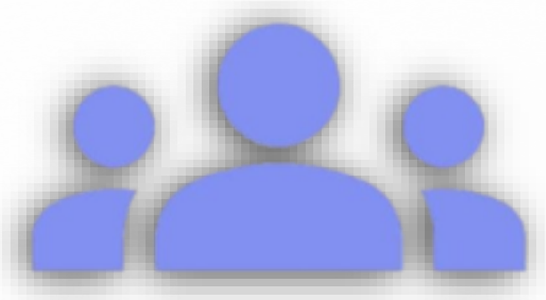

```
_id: "7346726"
email : "astolfifrancesco00@gmail.com"
hashed_password : "$2b$12$NpwUclOPrVSmvMVZ9eXUy.HKLo1TJQinyzwNfPJTWTUQz89Nv.PWW"
▾ params : Object
  humor : 4
  empathy : 5
  optimism : 3
```





Let Me Know



Thank you for
the attention !