

# DEVOPS FALL 2024 PROJECT featuring Alexandre Fenayrou and Nicolas Thorrez

---

In this report, we will go over all the features of our app and how to deploy it and such.

## 1: The app

What does it do?

Our app is a chat app. As simple as it may seem, we created an app that lets a user chat in real time in a global chat-room with other users.

One can :

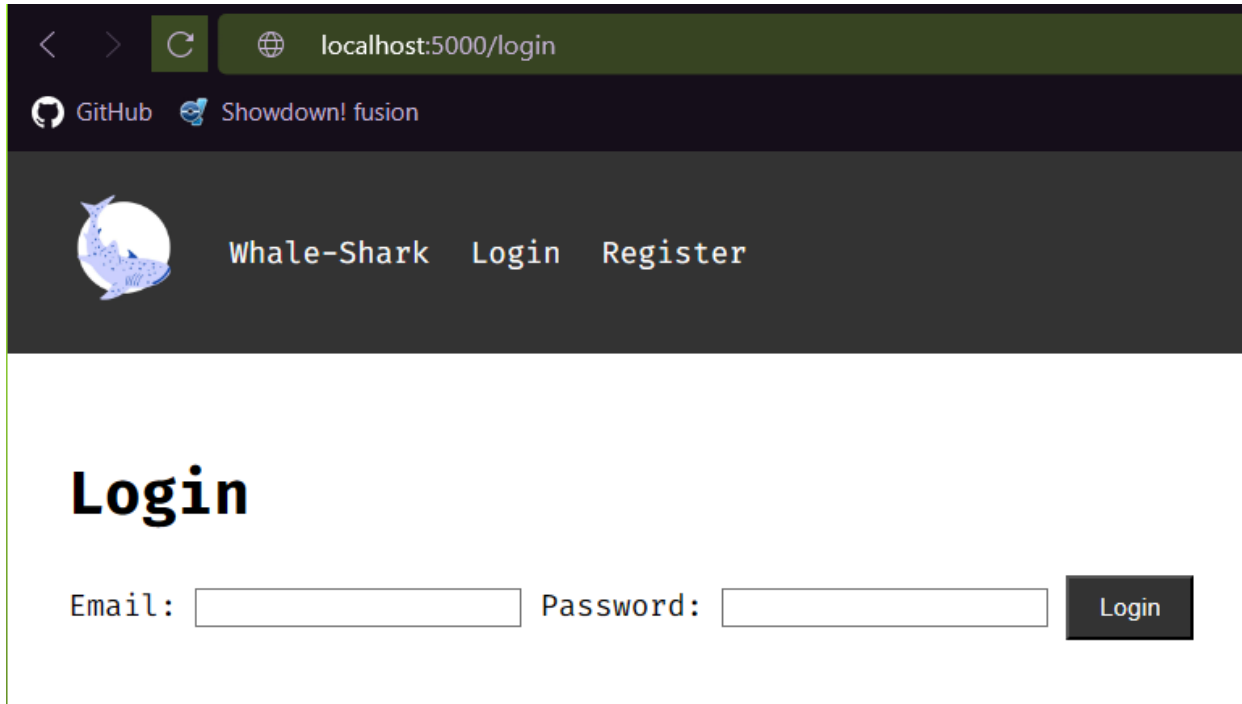
- Register using an email adress, username and password (note that both email and username can't be re-used by others who want to sign in) :



### Register

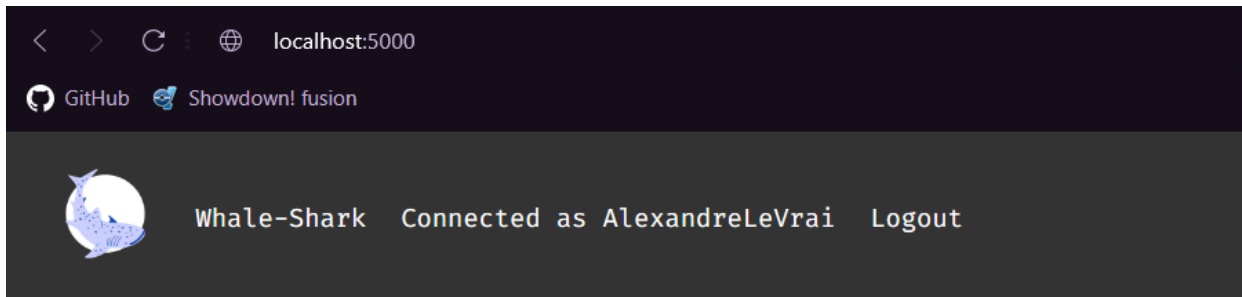
Username:  Email:  Password:

- Login using email and password :

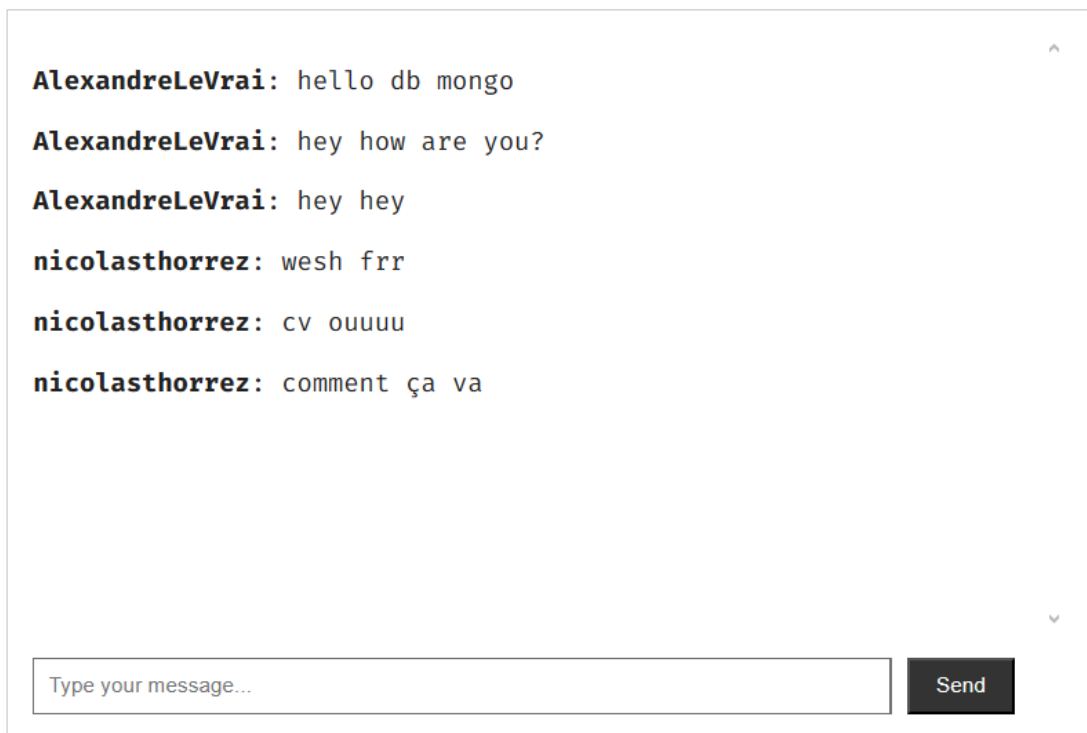


The screenshot shows a web browser at localhost:5000/login. The header includes a GitHub icon and the text 'Showdown! fusion'. Below the header is a navigation bar with the Whale-Shark logo, the text 'Whale-Shark', and links for 'Login' and 'Register'. The main content area has a large 'Login' heading. Below it are input fields for 'Email:' and 'Password:', followed by a 'Login' button.

- Chat in the chat-room ! :



The screenshot shows the chat room interface. The header is the same as the login page. The navigation bar now shows 'Whale-Shark', 'Connected as AlexandreLeVrai', and a 'Logout' link. The chat area is empty, and the input field at the bottom contains the placeholder text 'Type your message...'. A 'Send' button is located to the right of the input field.



The screenshot shows the chat room with several messages. The messages are as follows:

- AlexandreLeVrai: hello db mongo
- AlexandreLeVrai: hey how are you?
- AlexandreLeVrai: hey hey
- nicolasthorrez: wesh fr
- nicolasthorrez: cv ouuuu
- nicolasthorrez: comment ça va

The input field at the bottom contains the placeholder text 'Type your message...' and a 'Send' button.

Note that texts can't be deleted nor can accounts either.

## How was it made?

The app is made using python flask for back-end and html css as front-end. It is very likely that you are already familiar with this technology but we weren't. As we did not have the Web Technologies class this semester we learned from scratch what routes are and how they work. Notice that we also used a new database model, mongodb, which we have seen in a different class. The main issues we encountered during this project were:

- The Socket : how does it work and why does it not work sometimes.
- Windows/Linux compatibilities : As we don't use any Linux/Unix operating system on our machines it has been a hustle to do things the such as Ansible and Istio.

We are using [wait-for-it.sh](#), a powerful shell tool that helps us time the interaction between MongoDB and the app.

The MongoDB database is made of 2 collections: messages and users : mongodb.png


## How to deploy the app?

Well you can run the app:

- locally as long as you have python3 and MongoDB on your computer, it should work like a charm!
- On Docker using docker compose up in the directory where the file docker-compose.yml is.
- On Vagrant using the vagrant file, we might not have ansible ready but we can host the app on vagrant.
- On Kubernetes, using the k8s deployments we can run the application in multiple pods.


Once running you can access it through: [Localhost:5000](#)

## 2: CI/CD

We created some github actions that help us test our app everytime we push. These are simple unit tests and functionality tests that you can see in the [tests](#) folder, they are run through the [github/workflows/test.yml](#) : githubactions.png

## 3: Virtual Env + IaC

### Virtual Environment using Vagrant

Through our Vagrantfile, our application can be run on vagrant and exposed to the Localhost:5000 : vagrant.png

### IaC using Ansible

Even though we tried our very best to make ansible work on windows, it is definitely not made for it and we stand defeated by it. We though have some Ansible playbook.yml and Inventory.ini.

## 4 and 5: Docker image and docker-compose

### Docker image using Dockerfile

The Dockerfile is fairly simple, as we only get the components from the [/app](#) Directory plus [wait-for-it.sh](#).

## docker compose up

"docker compose up" is probably the line we ran the most during all the project. In the docker-compose.yml we've implemented volumes as to make our testing easier. Our deployment is made of two containers:

- One for the app
- One for MongoDB



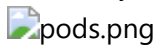
(note that volumes can too be removed using -v while removing. (I mean you probably know it right?))


## 6: Container Orchestration using Kubernetes

To use kubernetes we used Minikube, as learned in class.

Our [/k8s](#) file is the place where our yml files for deployments, connection with db, services and persistent volumes are stored.

Once they are added to kube, we have 3 pods in total: 2 for the deployment and 1 for the database :



Once our pods are running, we've tested their good working by exposing them to a port of our computer and using them in local : 

We *HIGLY* recommend you :

- log into docker and push the image of this Dockerfile in your account
- change line 19 of [deployment.yml](#) where you can find alexandrefenayrou/flask-chat-app:latest to put your image

if you find anything sketchy in the repo about the k8s file, just know that Nicolas Thorrez did it on Alexandre Fenayrou's machine and pushed in there. To get the blame updated, commits were done to revert and then repush.

## 7: Service mesh using Istio

Well this is a complete bummer since we did not succeed the lab in the first place, we do think that it's quite the harsh task to do on windows. We unfortunately have to skip this task.