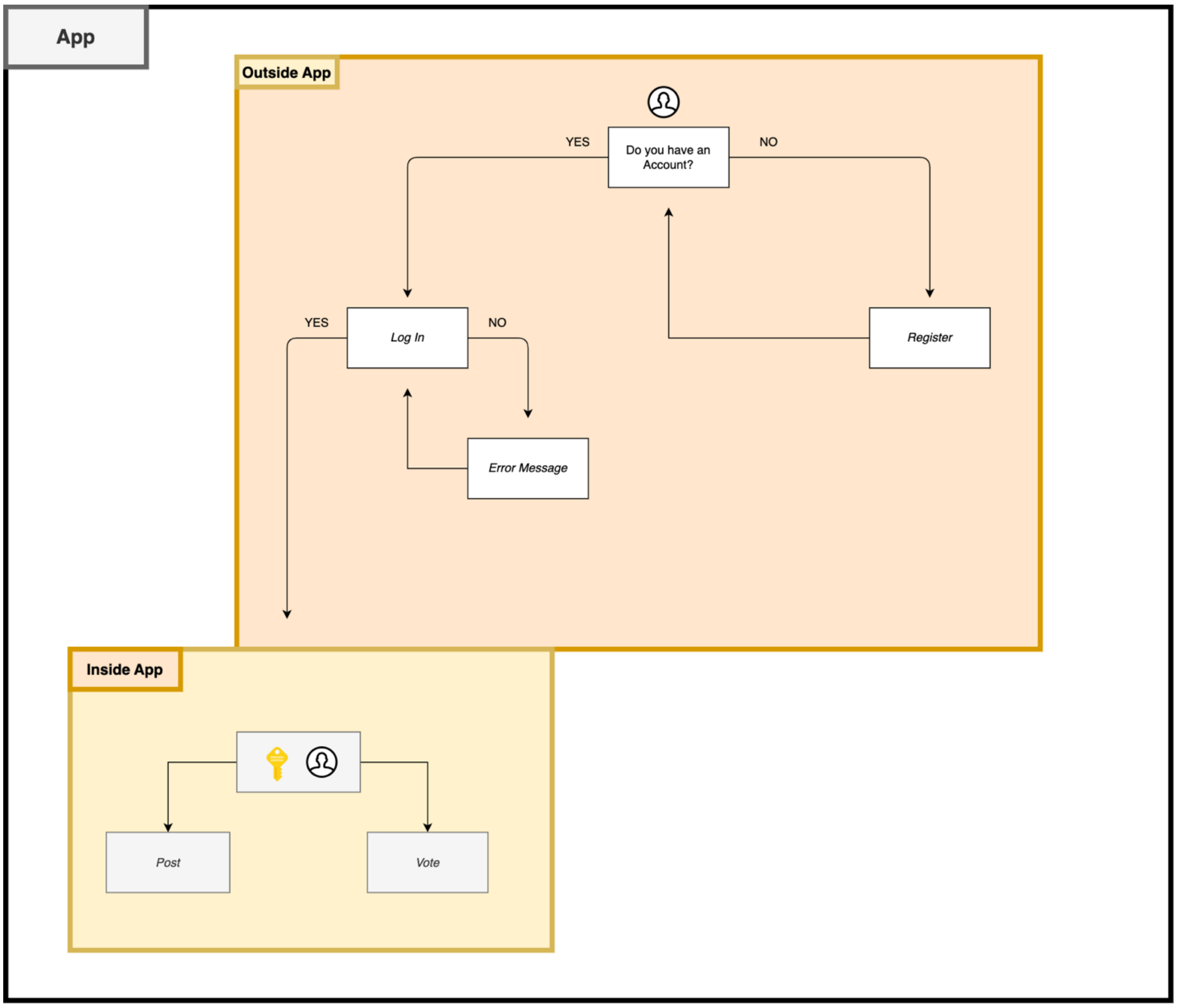
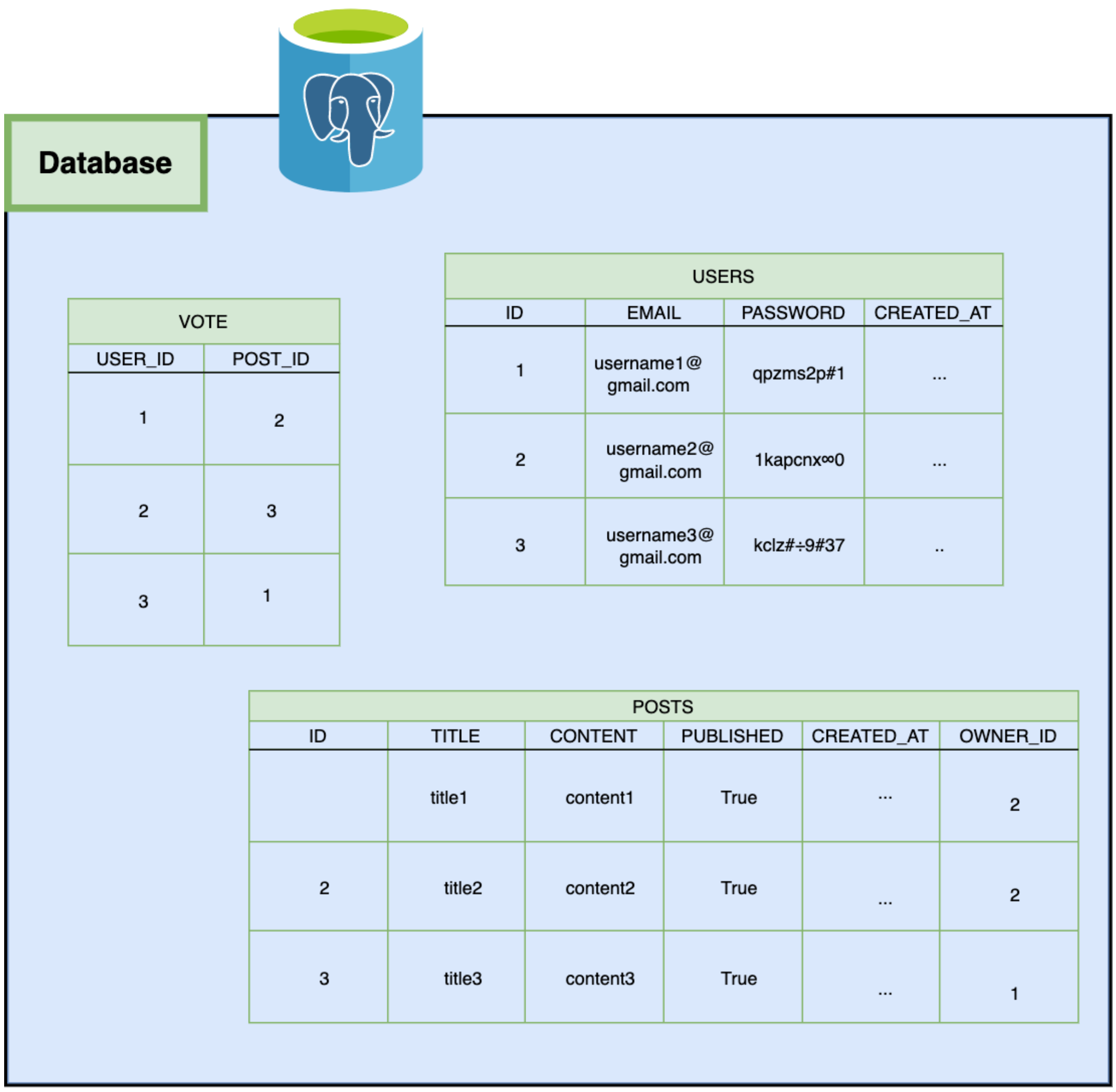
API Development with FastAPI

We will illustrate the general workflow of the API, in order to improve the clarity of the project.

There are two main services, the Database and the App.





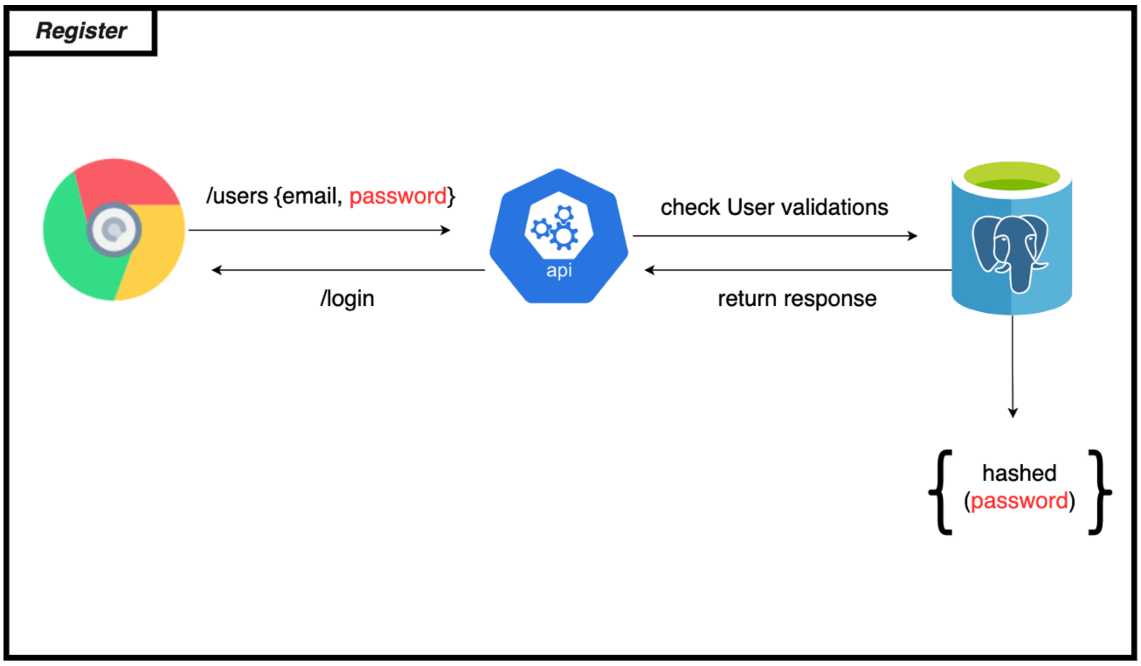
VOTE	
USER_ID	POST_ID
1	2
2	3
3	1

USERS			
ID	EMAIL	PASSWORD	CREATED_AT
1	username1@gmail.com	qpzms2p#1	...
2	username2@gmail.com	1kapcnx∞0	...
3	username3@gmail.com	kclz#÷9#37	..

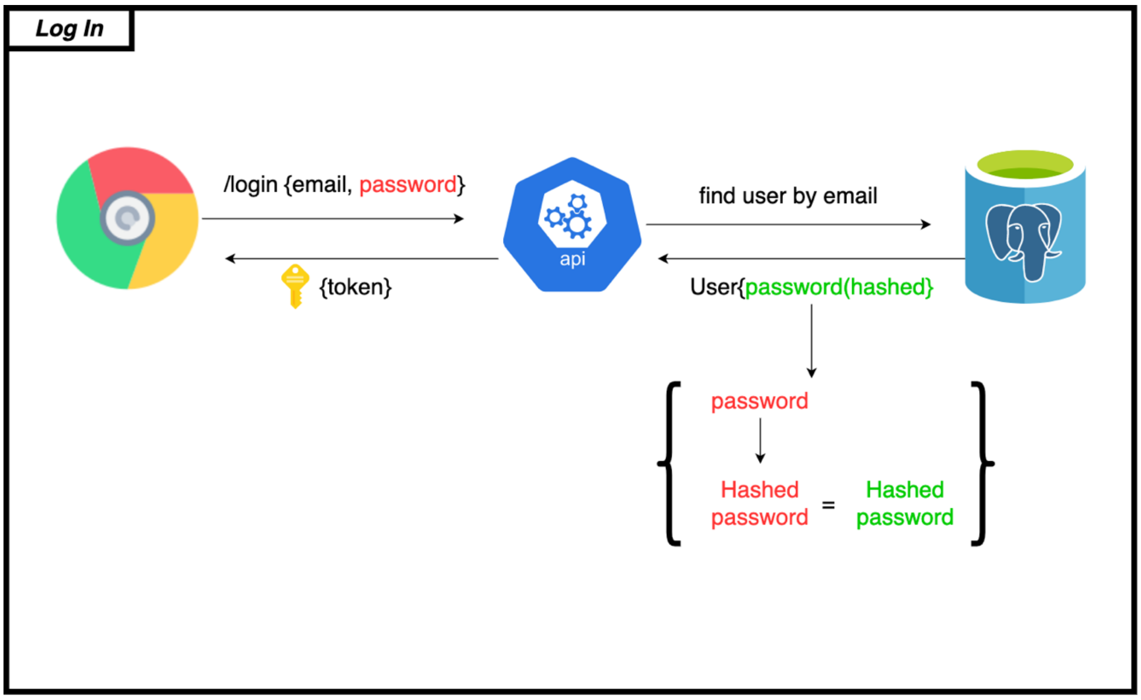
POSTS					
ID	TITLE	CONTENT	PUBLISHED	CREATED_AT	OWNER_ID
	title1	content1	True	...	2
2	title2	content2	True	...	2
3	title3	content3	True	...	1

As we can see on the App Diagram, we have several functionalities. More specifically, the User will be able to firstly register and then log In. Once logged in on the App, it will be able to do certain activities such as creating, updating or deleting a post and voting (like or dislike) them. Let´s start by briefly analyzing the User Registration, Log In and Authentication.

Just as an observation to avoid a possible confussion, on the App Diagram I separate the processes in two steps, “Outside” and “Inside” the App. This is not referencing the main FastAPI object commonly call as “App”, but what consumers tend to call as “App”, which is the application itself.



Firstly, the User must create an account. To do so, it will have to pass some validations, such as no repetition on the email selected, this being a valid one and so on... Once this criterias had been reached the person will have its account created and ready to be use whenever he wants. Our database will kept its email as username, the password (this is kept hashed because of security reasons) and finally a personal key identificator or “id”.



If you have already created your personal account you will be able to move on to this step. You load your account information, if it matches one with our database, it basically means that is your account, and it will enable you to browse around the app!.

As I mentioned previously, the passwords are saved hashed, so we have to hash it first before doing the comparison.

Lastly, every User logged in has an Access Token (in this case I chose a JWT Token). This will enable it to perform requests such as the ones mentioned before. Essentially, the objective is for security reasons, as a double authenticate we could say…, the token will last as much time as the one assigned it from the developer, in this case is 30 minutes.

Finally, now we are inside the Social Media app. Now is when the User will be able to perform all the requests related to posts and votes mentioned!. I want to clarify that the order does matter, by this I mean, you will not be able to log in if you have not created an account previously, or to put another example, you will not be able to delete a post which had never been created. Also, its respects the logic behavour of Users hierarchy as on any other ordinary "Social Media App". For instance, If I am logged in with my account, I would only be able to update or delete a personal post not someone else´s.

Well, If you are particularly interested on some of this features, I invite you to go and try them on the app. You can guide yourself from the names of the functions on the different pyhton files, inside the routers folder, they are pretty straightforward.

You can follow the instructions on the README.md file to dockerize the app and try it out in a matter of minutes. Hope this doc have helped you to get a clearer understanding of the project!