CIAI Project Report

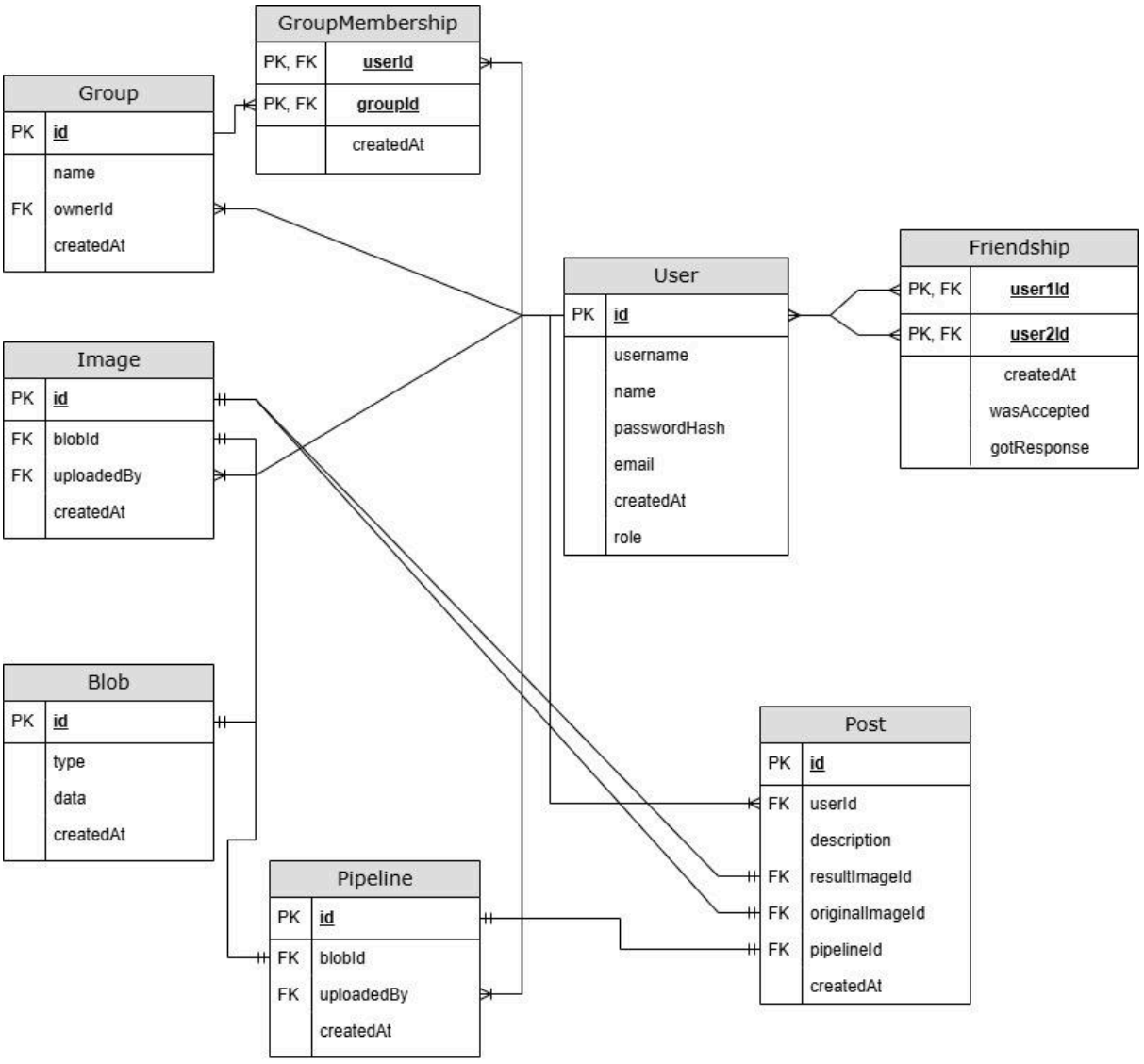
Guilherme Santana 60182

João Nini 60233

Pedro Cavaleiro 57974

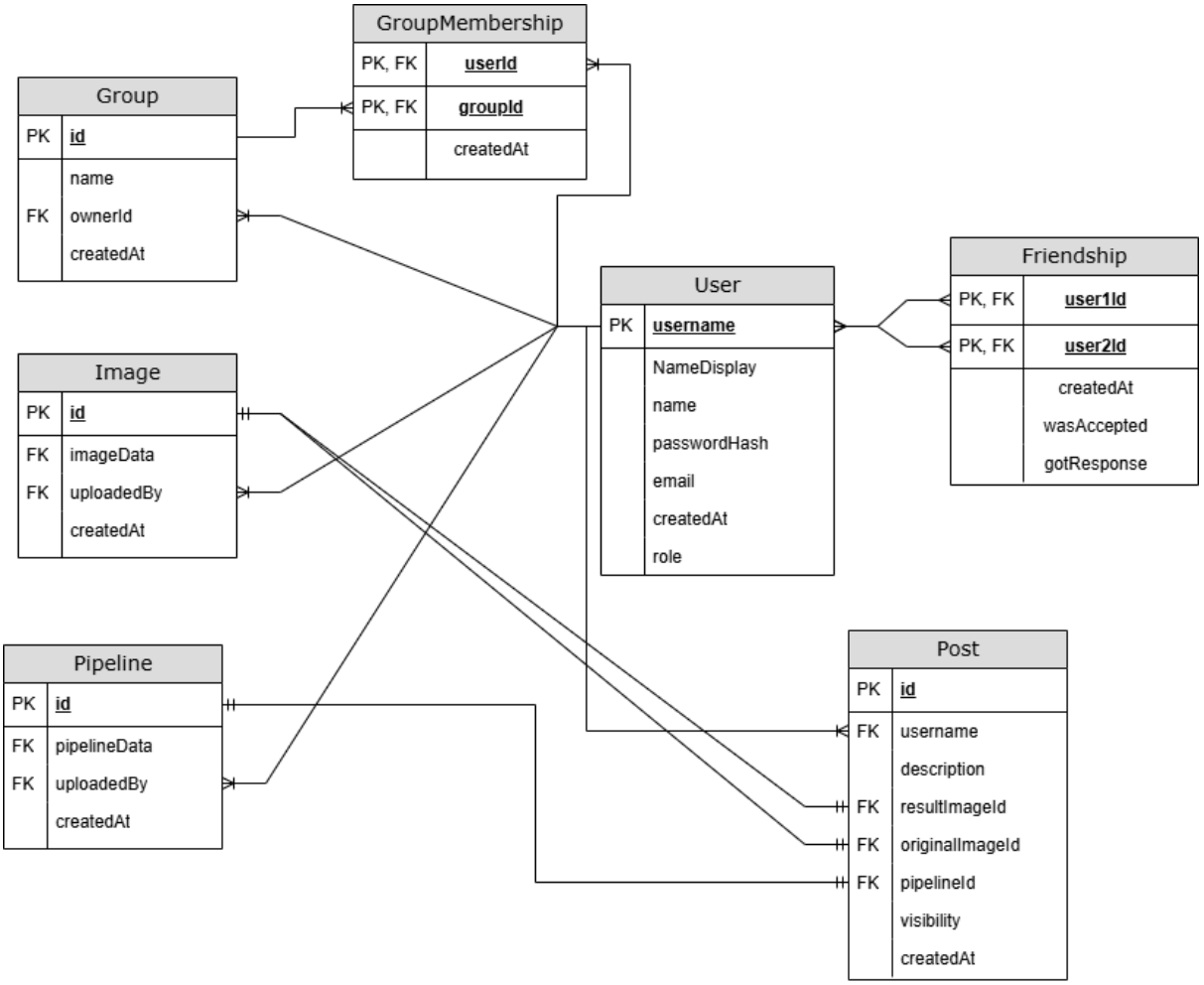
Changes

From the first delivery to this one we made some changes in the MicroServices Structure.



We deleted the blob service and in the image and pipeline service, after discovering that could have a column that represents a grant amount of data (`@Lob`) we stored the respective data in their respective service

System Architecture



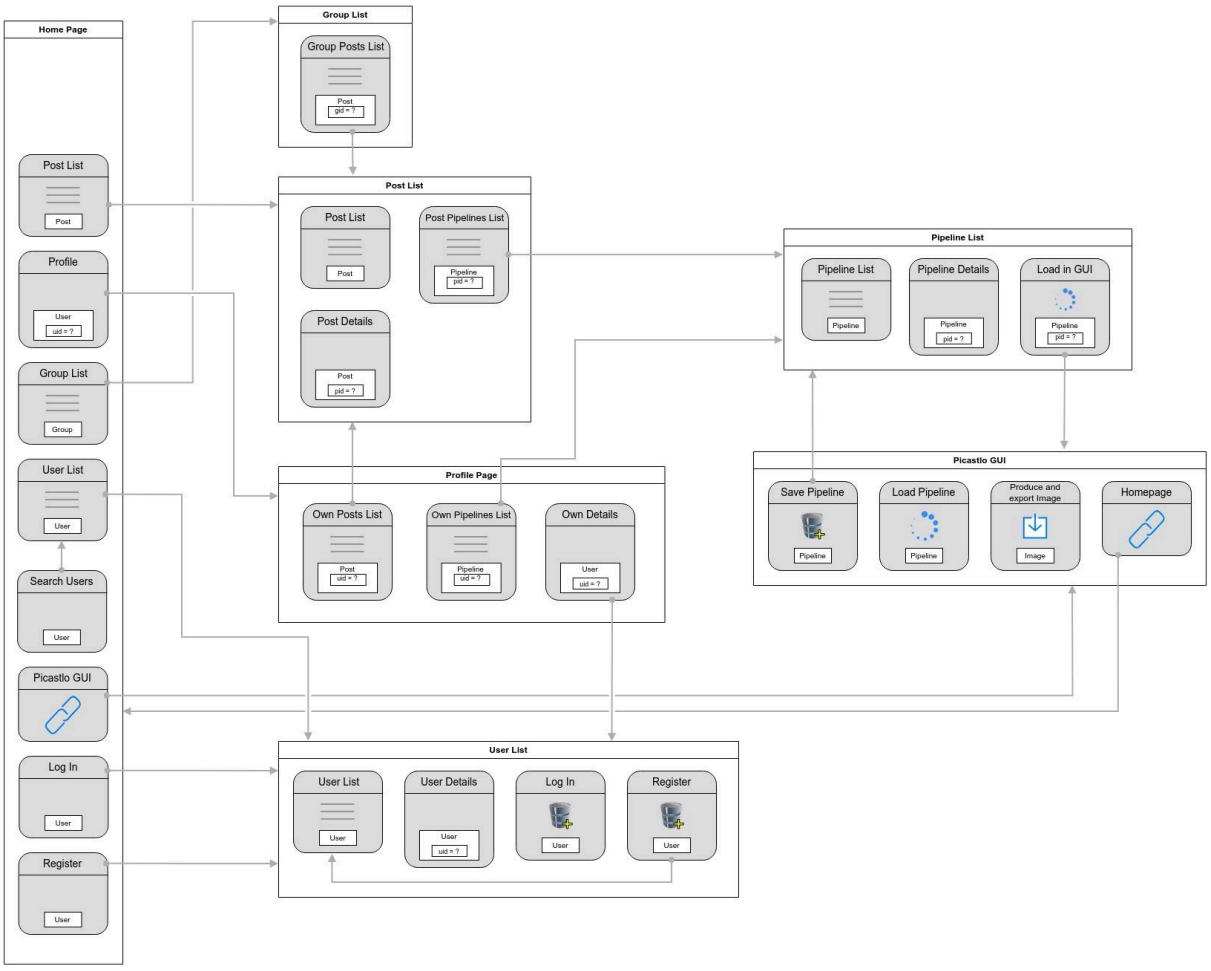
User stories

List the user stories that were completed

1. **Properly done**, the public posts can be seen by an unregistered user as he enters the application and checks the homepage.
2. **Properly done**, unregistered users can see the public posts in many different pages.
3. **Properly done**, we have a login page where the users can login into their accounts.
4. **Properly done**, a user can check a list of his created posts as well as his friends posts.
5. **Properly done**, a user can check as many posts in as many pages as he wishes.
6. **Properly done**, registered users can check the profile.
7. **Properly done**, users can select and check the posts of the groups they are in.
8. **Properly done**, the list of users can be checked.
9. **Properly done**, the users list page uses paging (and size).
10. **Properly done**, registered users can search for a user according to their username in the users list.
11. ***Functioning***, registered users can check the profiles of other registered users and see their posts and pipelines (the visible ones).
12. ***Functioning*** **for light transformations, the pipeline transformations can be loaded (although the image isn't loaded), because of the huge size of the pipelines generated by Picastlo (triangle transformation has 999999 rows) which would require major backend changes (uploading images block by block).**
13. **Properly done for light transformations**, registered users can load pipeline transformations onto Picastlo.
14. **Properly done**, users can produce and load (with Picastlo) the transformed images.
15. **Properly done**, users can leave the Picastlo GUI and navigate to other pages.
16. **Properly done**, registered users can save pipeline transformations onto their profile.

Description of the Application and Microservice

- A "UserService" for User and Friendship entities and its operations
- A "GroupService" for Group and GroupMembership entities and its operations.
- An "ImageService" for Image entities and its operations
- An "PipelineService" for Pipeline entities and its operations
- An "PostService" for Posts entities and its operations



APP API

Consultar AppAPI.pdf in the root of the repository

Security Roles

The user interact with the APi with a token. To have access to this token need to Authenticate the the username and is password. This token has the data to authenticate in the service and do the respective request. If the request is in other service rather than the userService (the app Service) a new token is generated, to communicate to that service. The new token have the necessary signature and information (like user friendships) to retrieve all the data required in the request.

An user have the following roles: Admin, Manager and User

Capabilities

All the services have capabilities accordingly with the role. This could differ between server, and between type of request (CRUD).

Image security Policies as an example:
picastlo-social-57974_60182_60233/ImageService/src/main/kotlin/com/example/imageservice/security/capabilities/SecurityPolicies.kt at Guilherme · IADI-FCT-NOVA/picastlo-social-57974_60182_60233

Assessment

System Architecture (Micro-services)	100%
Internal Architecture of Components (Layered Architecture, DAO, DTO, internal Services)	100%
Architecture of Client Application (React/Redux/Async/OpenAPI)	100%
REST API of Application	100%
REST API of Services	100%
Use of OpenAPI	100%
Seed Data Used	>50%
Secure Connection between Application and Services	100%
Completeness of Tests	<50%
Tests of security policies	>50%

Use of AI tools

Some of the design components (such as the login interface) were improved with ChatGPT in order to get a cleaner and more user friendly interface.

Used chatGPT and GitCopilot to generate: queries, Repeated code, like descriptions, DAOS, DTOS, for some error debugging and many others. The use of this AI tools help me save me some time for debugging and common code intra services.

Conclusions

We've learned how to implement an application through microservices, and how these microservices could be implemented as well as their respective security (Authentication / Authorization)

How we could save some time with the right frameworks to generate boilerplate code. The main example of this is Swagger/OpenAPI.
We used Swagger IO to generate the base server API and models and OpenAPI (with Docker) to generate the client code. Although it saved a huge amount of time, some of the code generated needed to be adapted because it was generated with errors.

Difficulties: Much time spent in the security of the services and the implementation/comprehension of the communication between services.

Negative aspects of the project: The project proved to be too long for a 3 member group, and the efficiency of the time spent developing the project could be greatly improved by having the User Stories earlier on to get a proper grasp of the application's requirements.