IADI PROJECT REPORT

DANIEL EUGÉNIO 59797, FRANCISCO COSTA 60220, and RAFAEL COSTA 60441

CONTENTS

1 System Architecture

Our system architecture is centered around a microservices design, emphasizing the interactions and seamless communication between these services.

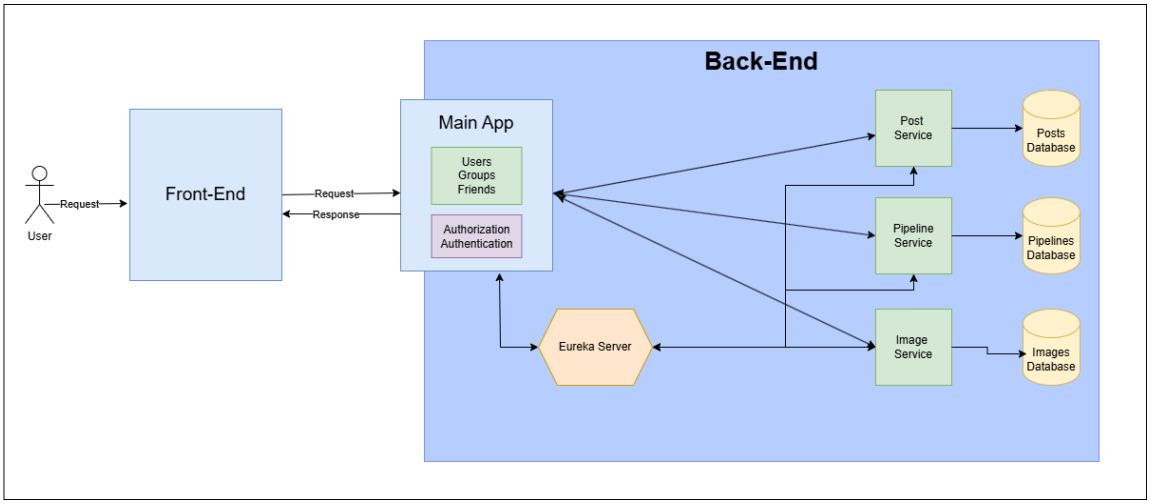

Fig. 1. System Diagram

2 User Stories

Completed User Stories:

(1) As an unregistered user, I want to access the picastlo social network homepage to see the first page of the public timeline of the system.
(2) As an unregistered user, I want to access the picastlo social network homepage to see a page of the public timeline of the system to see another page of public timeline of the system (next, previous, any number in between).
(3) As an unregistered user, I want to access the picastlo social network homepage to sign in to the system and see the homepage of the system.
(4) As a registered user, I want to access the picastlo social network homepage to see the homepage and see the first page of my own timeline with my posts, the public posts, my friends post, and the posts of the groups I am registered in.
(5) As a registered user, I want to access the picastlo social network homepage to see a page of my own timeline to see another page of my own timeline of the system (next, previous, any number in between).
(6) As a registered user, I want to access the picastlo social network homepage and select my profile to see my own posts and my own pipelines.
(7) As a registered user, I want to access the picastlo social network homepage and select a group that I am registered in to see the first page of the posts in that timeline.

(8) As a registered user, I want to access the picastlo social network homepage and select a page of the timeline of a group that I am registered in to see another page of the posts in that timeline (next, previous, any number in between).
(9) As a user, I want to access the picastlo social network homepage to see the first page of the list of users of the system.
(10) As a user, I want to access a page of the list of users of the system to see another page of the list of users of the system (next, previous, any number in between).
(11) As a user, I want to search the list of users by username and name.
(12) As a user, I want to select a user from the list of users and see their profile.
(13) As a user, I want to use the picastlo GUI to produce an image and export it.
(14) As a user, I want to select a pipeline from the user profile and load it in picastlo GUI.
(15) As a user, I want to exit the picastlo GUI and return to the homepage of the social network.
(16) As a user, I want to select a pipeline from a post and load it in picastlo GUI.
(17) As a registered user, I want to save a picastlo pipeline from the picastlo GUI to my profile.

3 Application and Microservices

The application is structured into a primary core application and three distinct microservices: **Images**, **Posts** and **Pipelines**. In the core application there are three resources: **Users**, **Groups** and **Friends**.

3.1 Users

The Users API provides functionality for user authentication and management. It includes endpoints for login, user creation and two user search options: retrieving all users or fetching a specific user by their username.

- **Create User** (POST /users) - Requires username and password and returns 200 if the user was created sucessfully.
- **Login** (POST /users/login) - Requires username and password and returns 200 if the user was created sucessfully.
- **Get All Users** (GET /users) - Returns a list of all usernames.
- **Get a User** (GET /users/username) - Requires an username and returns the user if any user exists with such username.

3.2 Groups

The Groups API offers functionality to create groups, add members to a group and retrieve group information in three ways: obtaining full details of a group, listing the members of a group and fetching all groups associated with a specific user.

- **Create Group** (POST /groups) - Requires a name and the username and returns 200 if the group was created sucessfully.
- **Add User To Group** (PUT /groups/id/username) - Requires the group ID and the username of the user to be added and returns the updated group information.
- **Get All User's Groups** (GET /groups/mygroups) - Returns a list of all groups that the logged-in user is a member of.

- **Get a Group** (GET /groups/id) - Requires an id and returns the group information if any group exists with such id.
- **Get a Group Members** (GET /groups/id/members) - Requires an id and returns the members of the group if any group exists with such id.

3.3 Friends

The Friends API provides friends management and a way to get the friend list of a user.

- **Add a New Friend** (POST /friends/username) - Requires a username and returns 200 if the user exists and was added to the friend list.
- **Remove a Friend** (DELETE /friends/username) - Requires a username and returns 200 if the user exists and was removed from the friend list.
- **Get a User's Friends** (GET /friends) - Returns a list of the friends of the logged-in user.

3.4 Posts

The Posts API allows for comprehensive post management, including publishing, updating, and deleting posts. It also provides multiple retrieval options: fetching a post by its ID, all public posts, all personal posts, all posts within a specific group, or an individual user's feed.

- **Publish a Post** (POST /post) - Requires a username, a image id, an optional pipeline id and a description and returns 200 if the post was created sucessfully.
- **Remove a Post** (DELETE /post/id) - Requires a post id and the updated information and returns the updated post information if it was sucessful.
- **Update a Post** (PUT /post/id) - Requires a post id and returns 200 if the post was deleted sucessfully.
- **Get a Post** (GET /post/id) - Requires a post id and returns the post if any post exists with such id.
- **Get Public Posts** (GET /post/public) - Returns a list of posts with Public visibility.
- **Get Personal Posts** (GET /post/personal) - Returns a list of the logged-in user's posts.
- **Get a Group Posts** (GET /post/group/id) - Requires a group id and returns a list of its' posts.
- **Get an User Feed** (GET /post/feed) - Returns the logged-in user's feed, which is a combination of public posts, friends' posts, and group posts.

For security, it was created five rules to produce capabilities:

(1) **canUpdateOrDeletePost** (@PreAuthorize) - Makes sure that only the owner of the post can update or delete the given post.
(2) **canPublish** (@PreAuthorize) - Makes sure that the post can be published with the given id.
(3) **canViewPost** (@PreAuthorize) - Verifies whether the user has the necessary permissions to view the specified post.
(4) **canViewMyPosts** & **canViewMultiplePosts** (@PostFilter) - Verifies whether the user can in fact view the given posts and only returns the ones that the user has access to.

3.5 Pipelines

The Pipelines API allows for pipeline management, including creating and updating its visibility. It also provides multiple retrieval options: getting a pipeline by its ID, a public pipeline by its ID and all pipelines of a specific user.

- **Create Pipeline** (POST /pipeline) - Requires a name and the username and returns 200 if the group was created sucessfully.
- **Update Pipeline Visibility** (PUT /pipeline/id) - Requires the pipeline ID and the updated visibility and returns the updated pipeline information.
- **Get All User's Pipelines** (GET /pipeline/all/username) - Requires an username and returns a list of all pipelines that the user has saved in its profile.
- **Get a Pipeline** (GET /pipeline/id) - Requires an id and returns the pipeline information, provided that a pipeline with the specified ID exists and the user has the necessary permissions to access it.
- **Get a Public Pipeline** (GET /pipeline/public/id) - Requires an id and returns the public pipeline information if any pipeline exists with such id.

For security, it was created three rules to produce capabilities:

(1) **canUpdatePipeline** (@PreAuthorize) - Makes sure that only the owner of the pipeline can update the given pipeline.
(2) **canReadPipeline** (@PreAuthorize) - Verifies whether the user has the necessary permissions to view the specified pipeline.
(3) **canReadMultiplePipelines** (@PostFilter) - Verifies whether the user can in fact view the given pipelines and only returns the ones that the user has access to.

3.6 Images

The Images API provides functionality for creating and deleting images, as well as retrieval options: fetching an image by its ID or all images posted by a specific user.

- **Create Image** (POST /images) - Requires a name and the username and returns 200 if the group was created sucessfully.
- **Delete Image** (PUT /images/id) - Requires the pipeline ID and the updated visibility and returns the updated pipeline information.
- **Get All User's Images** (GET /images/user/username) - Requires an username and returns a list of all images that the user has saved in its profile.
- **Get a Image** (GET /images/id) - Requires an id and returns the image information, provided that a image with the specified ID exists and the user has the necessary permissions to access it.
- **Get a Public Image** (GET /images/public/id) - Requires an id and returns the public image information if any image exists with such id.

For security, it was created three rules to produce capabilities:

(1) **canUpdateOrDeleteImage** (@PreAuthorize) - Makes sure that only the owner of the image can update or delete the given image.

(2) **canReadImage** (@PreAuthorize) - Verifies whether the user has the necessary permissions to view the specified image.
(3) **canReadMultipleImages** (@PostFilter) - Verifies whether the user can in fact view the given images and only returns the ones that the user has access to.

3.7 IFML Diagram

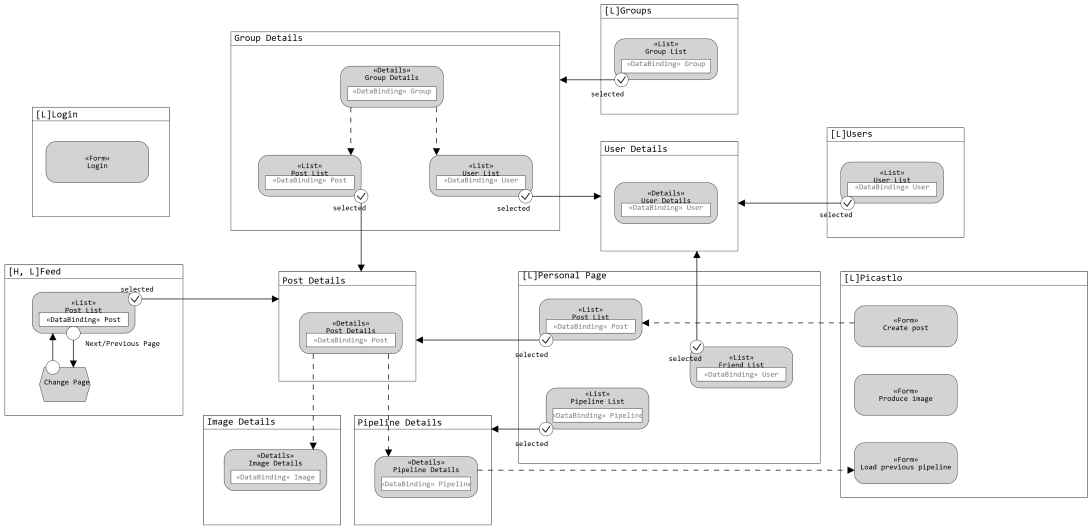

Fig. 2. IFML Diagram

4 Demo UI

Link will available soon

5 Assessement

6 Use of Third Party Tools

In the development of the project, we used ChatGPT as a helping tool but it was never relied upon to fully make any component since it was mainly used to fix bugs. It was extremely helpful in identifying and fixing bugs that we were unable to detect, but there were times when even AI couldn't assist in resolving certain issues. We also used React Material UI to help us build our front end, leveraging its available components to streamline development and ensure a consistent, modern user interface.

Topic	Fulfillment
System Architecture (Micro-services)	90%
Internal Architecture of Components (Layered Architecture, DAO, DTO, internal Services)	95%
Architecture of Client Application (React/Redux/Async/OpenAPI)	95%
REST API of Application	100%
REST API of Services	100%
Use of OpenAPI	100%
Seed Data Used	75%
Secure Connection between Application and Services	85%
Completeness of Tests	95%
Tests of security policies	>50%

7 Conclusion

This project provided valuable insights into building a complete application and adopting the best practices to use throughout the development process. However, it wasn't without its challenges. In the beginning, we faced difficulties in understanding the concept of microservices, their interactions with each other. As for the front-end part, we struggled with integrating the Picastlo GUI with our own website.

On the positive side, the project allowed us to deepen our knowledge of security in code and explore how separate components can seamlessly integrate to form a cohesive system. On the other hand, a significant drawback was the time consumption of the project which took us around 350 hours in total. Developing a good application required substantial effort, which was challenging to balance alongside other academic commitments.