curio.

Curio - Software Design

Supervisor

Jiankun Cai

Clients

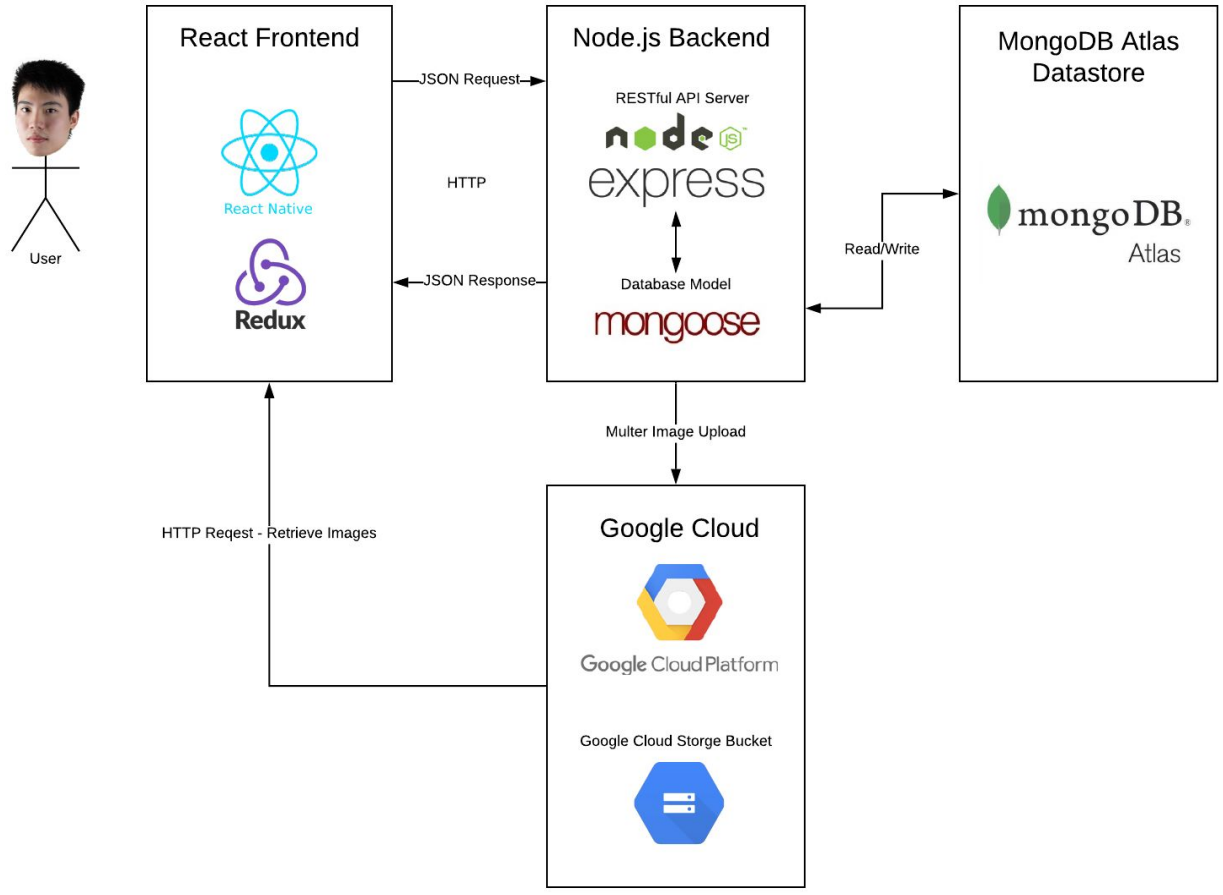
Tan Toi Vo
Linh Que Mach

Team Members

Christian Sugianto - Frontend Developer
Shaun Soong - Designer/Frontend Developer
Nick Ang - Designer/Frontend Developer
Andy Vo - Operations/Full-stack Developer

Table of Contents

1 - Architecture Design Diagram



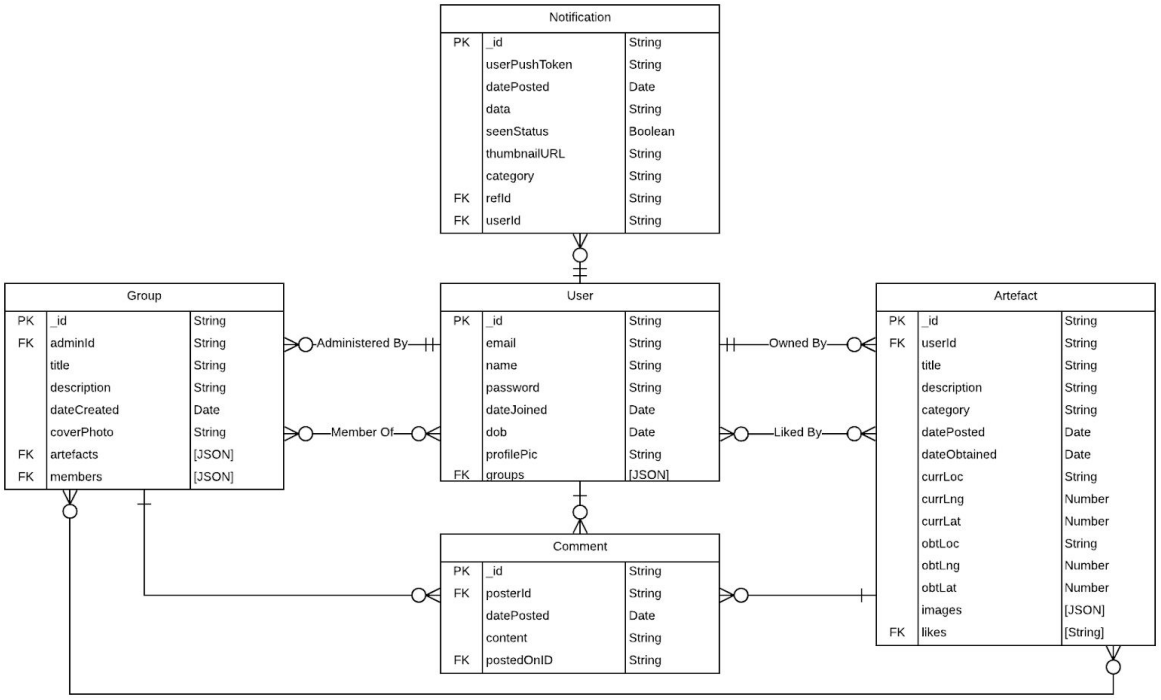
Description

- The front end app runs on React Native and Redux. The app will be primarily developed for the Android platform but we have kept in mind the ability to port to other platforms when developing the app.
- The backend API server runs on Node.js and Express.js with Mongoose used to model collection schemas.
- MongoDB will be used to store persistent data.
- Google Cloud Storage Bucket will be used to store binary data (blobs) such as images.

Justification

- React Native was chosen as the framework for the front end for server reasons. Firstly, it is widely used in industry so the team decided it was a useful technology for us to learn. Secondly, it makes it relatively easy to port the app to multiple platforms e.g. Android, iOS and Web. The client mentioned that it would be important for them to be able to share with users who don't use Android so the choice of using React Native support this user story.
- Redux was chosen since we wanted a highly interactive app that behaved largely like a single-page app, using redux stores instead of constantly reloading pages makes the experience a lot more fluid.
- Node.Js, Express.js and Mongoose were chosen as the tech stack for our API server as the whole team has had experience with this stack before so it would aid us in quickly getting the project off the ground and thus give us more time to learn and develop the front-end.
- MongoDB Atlas was chosen as the platform to host our database as it is free and feature rich e.g. allows easy building of indexes and optimisation
- Google Cloud Platform was chosen to host image due to its performance and features. AWS S3 was considered as they team has had experience with it but we found it unintuitive and GCS had features we wanted to utilise for this project.
- This tech stack also allows the team to code exclusively in JavaScript ES6 which allows for smoother transition between roles and makes sure the team is more able to support one-another with basic coding issues.

2 - Database Entity Relationship Diagram



Description

Data

User Model

- Stores basic information about the user including their emails and usernames which are both unique ids which can be used for logging in.
- Stores password in a hash and salted string.

Artefact Model

- Stores basic information about an artefact including potentially geolocation data which can be used to place the artefact on a map.

Group Model

- Stores a collection of members as well as a list of artefacts posted in the group.

Comment Model

- Stores comments data including when they were posted and who posted them.

Notification Model

- Stores the data required to deliver notifications to users.

Relationships

User - Group

- The User/Group relationship is the more complicated relationship in this database design. It is a many-to-many relationship which we must be able to access two ways. I.e. given a user we must be able to get all the groups they are a member of and given a group we must be able to get a list of all it's members. To accommodate for this both models contain an array listing primary keys to the other, each primary key is also attached to a date representing the date the user joined the group. This makes adding/removing users from group a complicated action but thankfully this interaction is don't relatively infrequently.

User - Artefact

- The User/Artefact relationship is much simpler. It is a one-to-many relationship were given a user we must be able to retrieve all their artefact and given an artefact we must be able to get its owner. All that's needed to accomplish this is a foreign key in artefact referencing its owner. If we index this key with a hash index, getting all the artefact a user owns is an O(1) operation.

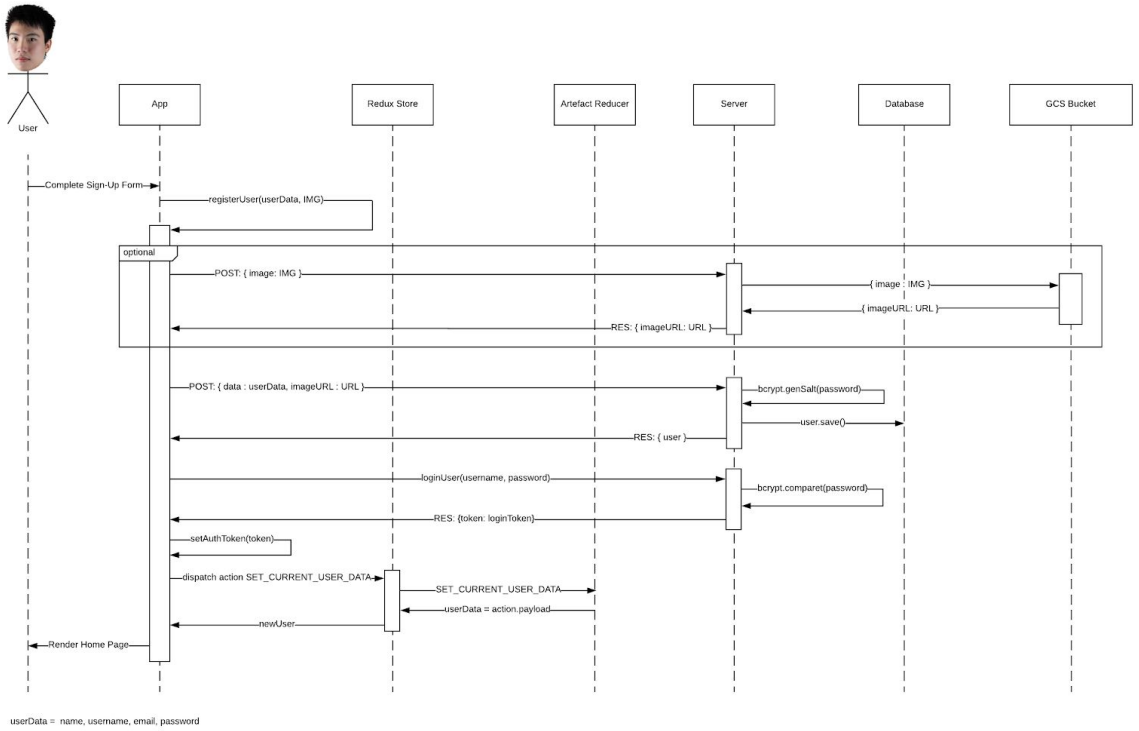
Comments - Artefact/User/Group

- The comment model is unique in that the same model is used for comment on all of artefacts, users and groups. The comment document simply contains a foreign key to the artefact/user/group it is posted on. This is possible as MongoDB is (almost[1]) guaranteed to generate unique keys even across different collections.

[1] While it is possible for a conflict to occur this is unlikely to happen in the lifetime of the universe.

3 - System Sequence Diagrams

a) Register a New Account and Login



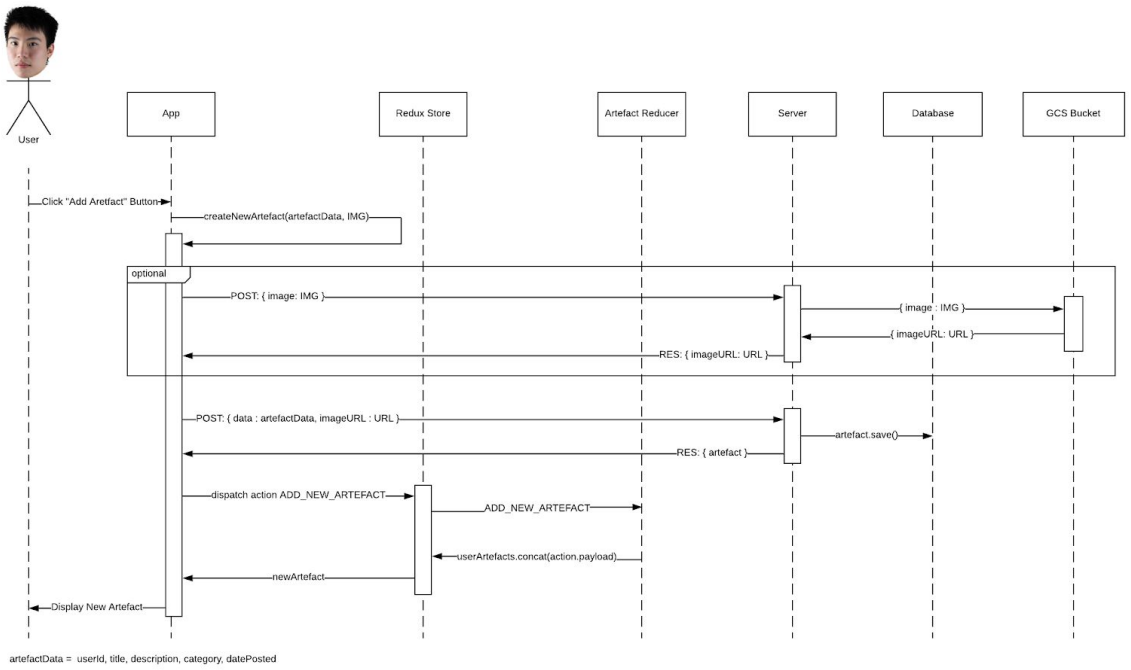
Description

Register a new account involves numerous actions on the frontend and backend. Once a new user is created the user is automatically logged in to the app saving users the effort of having to log in immediately after signing up.

1. Firstly the user fills out a form containing their name, username, email and password.
2. If the user opts to attach an image a POST request is sent to the backend server with this image as multipart form data.
    a. The backend upload this image to GCS
    b. GCS then returns the URL of where this image can be accessed.
    c. The backend then return this URL to the frontend where it is added to the rest of the user's data
3. Once all the data has been gathered a new POST request is sent to the backend where the user's password is hashed and salted.
4. The server then creates a new database document for the user and stores it in MongoDB Atlas and returns the complete User object to the frontend.
5. The frontend then automatically log the user in by sending a POST request to the backend with the email (or username) and password.

6. The server responds with an authentication token.
7. The frontend then dispatches both the authentication token and the User object to the Redux Store to be reduced and stored for quick access by the app.
8. Finally the logged in user is directed to the homepage of the app.

b) Posting a New Artefact

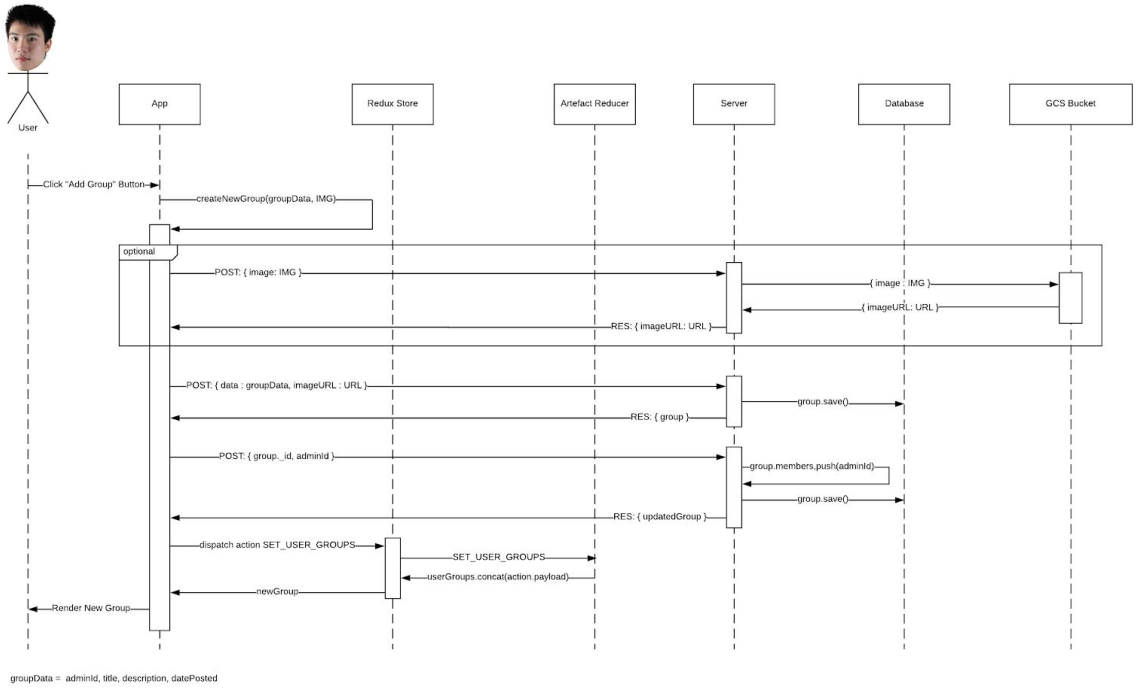


Description

Posting a new artefact involves several steps and varies depending on whether the user ops to upload an image with the artefact.

1. First the user fills out a form containing data about the artefact, including its title, description, and category. This data is then combined with the user's unique ID and the current date and time.
2. If the user opts to attach an image a POST request is sent to the backend server with this image as multipart form data.
    a. The backend upload this image to GCS
    b. GCS then returns the URL of where this image can be accessed.
    c. The backend then return this URL to the frontend where it is added to the rest of the artefact's data
3. Once all the data has been gathered a new POST request is sent to the backend which creates a new database document for the artefact and stores it in MongoDB Atlas and returns the complete Artefact object to the frontend.
4. The frontend then dispatches this object to the Redux Store to be reduced and stored for quick access by the app.
5. Finally the stored artefact can be accessed/rendered by the app.

c) Creating a New Group



Description

Creating a new group is very similar to adding a new artefact but with a few added steps.

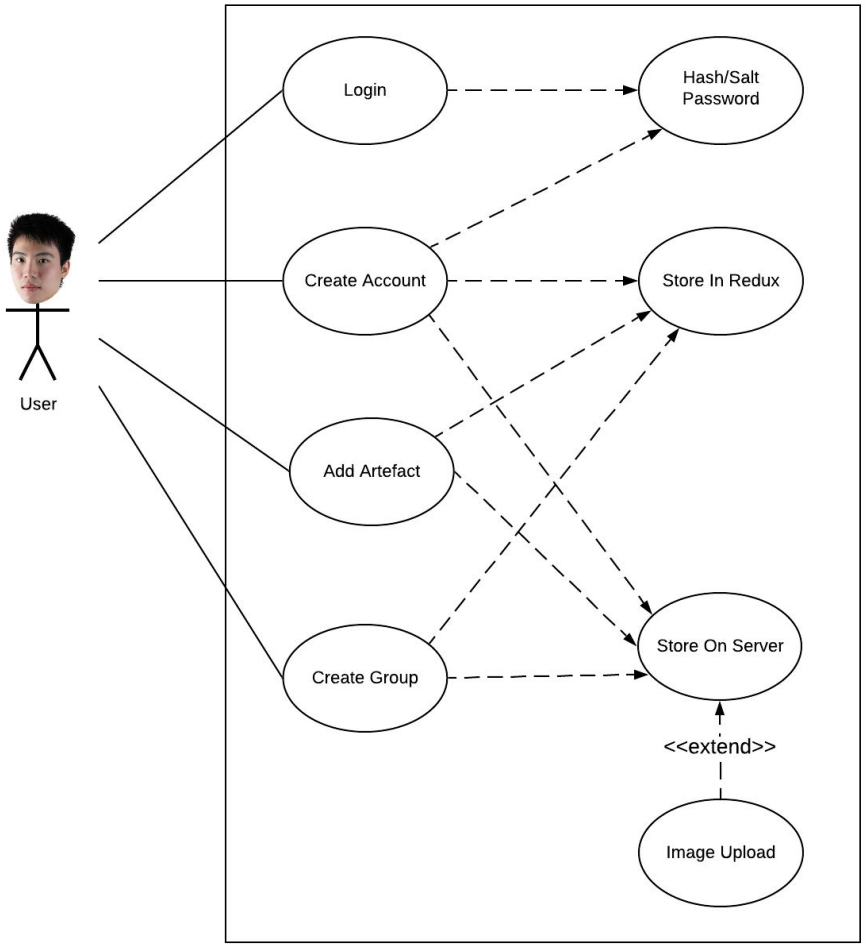
1. First the user fills out a form containing data about the group, including its title and description. This data is then combined with the user's unique ID which is to serve as the admin of the group and the current date and time.
2. If the user opts to attach a cover image a POST request is sent to the backend server with this image as multipart form data.
    a. The backend upload this image to GCS
    b. GCS then returns the URL of where this image can be accessed.
    c. The backend then return this URL to the frontend where it is added to the rest of the group's data
3. Once all the data has been gathered a new POST request is sent to the backend which creates a new database document for the group and stores it in MongoDB Atlas and returns the complete Group object to the frontend.

4. Once the frontend has the new group object yet another PUT request is sent to the back end to add the current user/admin as a member of the group.[2] The backend then replies with the complete group with the user added as a member.
5. The frontend then dispatches this object to the Redux Store to be reduced and stored for quick access by the app.
6. Finally the stored group can be accessed/rendered by the app.

[2] The reason the action of adding the admin as a member of the group is not completed in the backend is that the user needs to add a FK to reference groups they are a member of and the new group doesn't have an assigned ID until after it is created.

4 - Use Case Diagrams

a) Adding a User, Artefact or Group

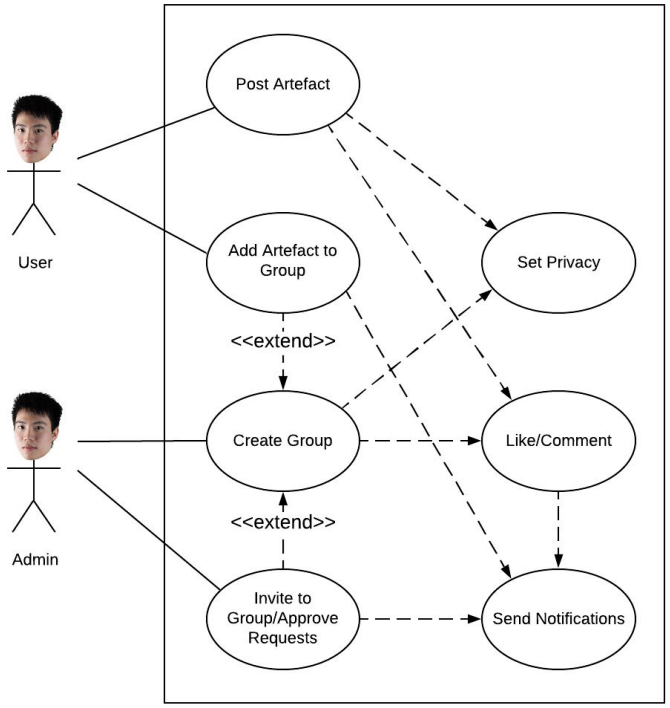


Unlabelled dotted line relationship are <<include>> relationships

Description

As seen in this use case diagram user have the option of adding images to their profiles, artefacts or groups. In the case of artefacts users have the option to upload and attach multiple images. All data is stored on both the database (via the API server) for remote access as well as locally for fast cached access. Passwords are salted and hashed to protect them in the case of a data leak.

b) Using the App



Unlabelled dotted line relationship are <<include>> relationships

Description

Normal users can post artefacts and add their artefacts to groups. Users can also set the privacy of their personal artefacts to determine whether they are visible on their public profile. Users who create groups become their defacto admins. Admins have the ability to set the privacy of groups as well as invite members to join groups or approve requests from users to join. Users can like/comment on either individual artefacts or groups as a whole. Notifications are sent out to artefact owners whenever someone has like/commented on their artefact. Notifications are sent out to all group members when a new artefact is added to the group. Notifications are sent to group admins whenever someone requests to join their group.

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK