VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
INFORMATION TECHNOLOGIES STUDY PROGRAM

Problem-Based Project

Graffiti Library

Done by:
Morta Marija Matelytė
Živilė Nebutovaitė
Kernius Survila
Dominykas Švilpa

Supervisor:
dr. Linas Bukauskas

Vilnius
2023

Preface

This project was done during the 3rd semester of the study programme *Information Technologies* in the specialization of Innovative studies. We chose the topic *Graffiti Library* suggested during the project market on the first lecture of the subject ‘‘Problem-Based Project’’. The topic sounded very interesting as it deals with both technology and art, and we wanted to explore how technology can be used to preserve and promote graffiti as a form of art.

January 25, 2023

Morta Marija Matelytė

Živilė Nebutovaitė

Kernius Survila

Dominykas Švilpa

Contents

Terminology

Unit testing - a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.

Integration testing - a level of software testing where individual units / components are combined and tested as a group.

E2E testing - a software testing method that validates entire software from starting to the end along with its integration with external interfaces.

Manual testing - the process of manually testing software for defects.

Exif (Exchangeable Image Format) - a standard that defines specific information related to an image or other media captured by a digital camera.

Haversine's formula - a formula that determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

Nearest neighbor algorithm - a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

piexif - a JavaScript Exif library, that enables exif data reading, editing and removal.

Abstract

The goal of this project was dedicated to transferring street art onto an online platform accessible to anyone in the world. The graffiti library is used to display images on a map based on the sites the photographs were taken, giving users the ability to discover street art, and acquire inspiration for their own work. The application also opens the possibility for user interaction with liking and commenting functionalities under the posted images. Related projects were reviewed and our own ideas and explications were presented. The steps that were taken to achieve the goal of this project include creating a database model, according to which, a database schema was created. Several algorithms were incorporated into our project. An image quality assessment algorithm performs calculations for the orderliness of images in a carousel. Each assessed image obtains a score used for comparison when calculating image orderliness. The project also includes a nearest neighbour algorithm that helps with detecting graffiti spots at a certain distance. For user protection, a functionality was implemented where metadata is stripped from the uploaded images. The prototype of the website provides all the functionalities mentioned above with a minimal user interface and graphical design.

Keywords: Graffiti, Image, Map, Website, Library.

Santrauka

Šio projekto tikslas buvo perkelti gatvės meną į internetinę platformą, kuria galėtų naudotis bet kuris pasaulio gyventojas. „Graffinity" svetainėje galima rasti nuotraukas žemėlapyje su jame išdėliotais žymekliais vietose, kuriose yra atitinkami grafičiai. Šis funkcionalumas suteikia naudotojams galimybę susipažinti su gatvės menu, bei pasisemti įkvėpimo savo darbams. Svetainė taip pat suteikia interaktyvias galimybes, kaip „mėgti" paspaudimas prie įrašų ir komentavimas. Buvo apžvelgti panašūs projektai ir pateiktos mūsų pačių idėjos bei komentarai. Žingsniai, kurių buvo imtasi siekiant šio projekto tikslo, apima duomenų bazės modelio sukūrimą, pagal kurį buvo sukurta duomenų bazės schema. Į šį projektą buvo įtraukti keli algoritmai. Nuotraukų kokybės vertinimo algoritmas atlieka karuselėje esančių nuotraukų vertinimą iškeldamas aukštesnį taškų skaičių gavusias nuotraukas į karuselės priekį. Į projektą taip pat įtrauktas „artimiausio kaimyno algoritmas", padedantis aptikti netoli vienas nuo kito esančius grafičius. Siekiant apsaugoti naudotoją, sukurta funkcija, kuri iš įkeltų nuotraukų pašalina tam tikrus metaduomenis. Svetainės prototipas suteikia visas minėtas funkcijas su minimalia grafine naudotojo sąsaja ir grafiniu dizainu.

Raktiniai žodžiai: Grafiti, nuotrauka, žemėlapis, interneto svetainė, galerija.

Introduction

Graffiti is an art form that has been around for centuries, and it continues to evolve and inspire people worldwide. Many cities are full of exceptional art that can be found on the streets. This art gets pushed into the shadows and does not get the recognition it deserves. Therefore, Graffinity aims to document and showcase the diverse and creative expressions of graffiti artists from Vilnius. The website features a collection of images and information on graffiti art, its history, and the artists behind it.

This website allows users to explore and discover different styles, techniques, and meanings behind graffiti art. From the bold and colorful graffiti found on the main streets of cities to the more subtle and thought-provoking pieces found in more unexpected places, the website offers a glimpse into the world of graffiti and the people who create it.

The system provides new ways to explore street art and makes sure of a high-quality experience by examining every post, affirming information about it and selecting the best photos.

This graffiti library will push graffiti art out of the shadows, create new communities with interest in street art and expand the graffiti culture in the future.

1 System architecture

1.1 An overview of the system

In order to build an understanding of how the system will be implemented, an overview of the Graffinity web application will be provided and discussed in this portion of the document. To display this information, a deployment diagram will showcase how some aspects of the website work in correlation with others along with some additional information about the system's architecture.

1.2 UML deployment diagram

The deployment diagram shows what artifacts our team's deliverable will consist of, and where they will be deployed/published.

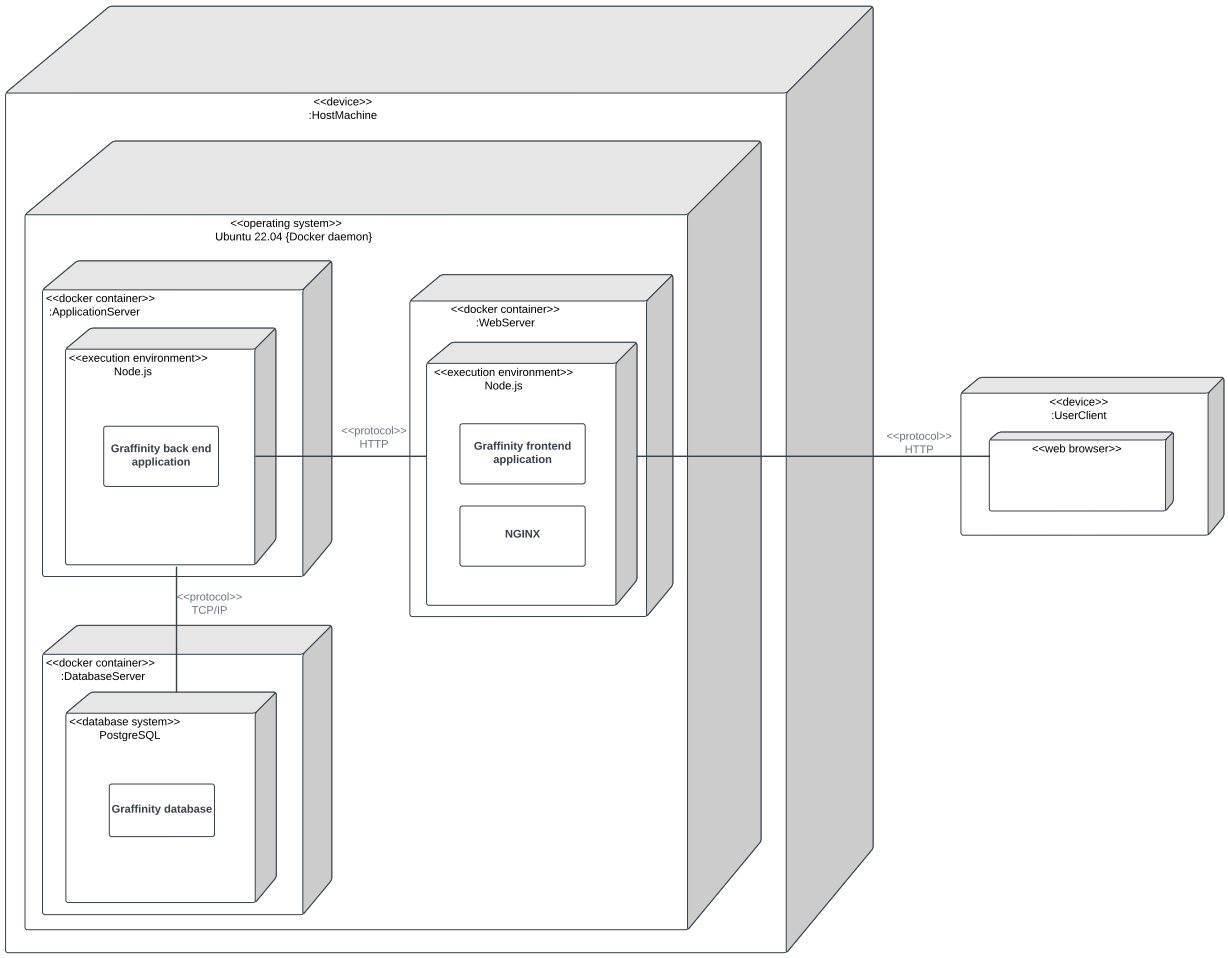


Figure 1. : Deployment Diagram

1.3 Authentication

Two different authentication methods will be used in the system. One of them is Google account authentication, and the other is using the email address and password that the user provided while registering.

1.4 Reverse proxy

In a network infrastructure, a reverse proxy is a sort of server that is placed between a client and a server. Client requests are received, forwarded to the proper server, and the server's response is then returned to the client.
Reverse proxy offers an extra layer of abstraction and control to make sure that network data between clients and servers flow without interruption.

1.5 Front-end

For front-end development, the library of the javascript programming language, React will be used. It is one of the most popular technologies for creating User Interfaces today, for its simplicity, speed, performance and other great features.

1.6 Back-end

Node.js will be applied for developing the back-end development which will be stored in a docker container. As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. It gives the opportunity for Javascript to be used both in back-end and front-end development.

1.7 Communication protocols and guidelines

All requests to the server should enter using the safe and encrypted HTTPS communication protocol, this way ensuring protection from Man in the Middle (MITM) attacks. Accessing the server (Ubuntu 20) will only be possible through HTTP:80 and HTTPS:433 ports. Other system components will only be available via the Nginx reverse proxy, not directly.

1.8 Technologies and tools

- **TypeScript** is used not only to write and build the front end of the project but the back end as well. We chose TypeScript over its more popular counterpart JavaScript because TypeScript adds on existing JavaScript features that are useful for less experienced developers for example with TypeScript you can avoid hidden errors that are prevalent in JavaScript, for example, the classic undefined is not a function error.
- **React.js** library is used to help us build the front end of the project. We chose React to help us with state management and rendering that state to the DOM.
- **Material UI** is paired with TypeScript and React to help create web application components and customize them to our needs.
- **Node.js** helps us to build the back end of the project by allowing us to run scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.
- **PostgreSQL** is being used as our database management system.
- **Prisma** is used for the object-relational mapping between TypeScript and our database PostgreSQL.
- **Express.js** framework is being utilized for building APIs with Node.js.
- **Google Maps API** is utilized for building the map with custom markers and info windows to show where are the graffiti posted on the website.
- **Docker** is utilized for containerizing the back end of the application.
- **Jira and Confluence** are used for tracking time spent on the project, task and in what stage the task is currently, this tool helps us to keep our meeting notes and other notes in the same convenient place.
- **Git and GitHub** are being utilized for version control.
- **Visual Studio Code** is used for writing code.
- **Jest** is a testing framework that helps us to write and execute tests for code we have written.
- **Postman** is an API that we use for testing and iterating APIs that we implement.
- **Ubuntu** we pair it with **WSL** for easier use with Git and **NPM**.
- **NPM** is used for downloading code packages.
- **Nest.js** is a framework that helps us to build our server-side application.

2 Analysis of related work

2.1 Street Art Cities

One example of a similar project to ours is Street Art Cities. The project's vision seems to be very similar to ours: it helps you explore the city of your choice, by providing you with the locations of different graffiti.

This website includes an interactive map, with markers placed in the locations of the graffiti. The user can find a short description, with the artist's name, if it is known, as well as one or more pictures of the art piece. This functionality is very similar to our project's map and pop-up features but our team noticed several ways that can be used to improve this website.

The Street Art Cities does not provide a page where you can find all of the works of one artist, nor a 'graffitis near me' feature. Therefore, if a user wants to explore places near their location, they would need to research that themselves. When clicking the search button, something similar to an explore page appears, but the search is only available by graffiti name. There seems to be an option to sign in, but it is unclear how it works because after signing in, the website presents the user with a message, saying that their account is not part of any city. Additionally, if the user logs out, they are required to sign in the same way as they created an account: through an email that is sent to their inbox. Therefore the additional advantages of creating an account are unknown.

One feature this project has that our project is not planning to incorporate is a navigation feature. When pressing the "Navigate" button, the users get redirected to google maps with the walking route from their location to the wanted graffiti location loaded on the screen. Another feature of Street Art City, that will not appear in our project is becoming a "graffiti hunter". While in our project, users have the option to submit posts or add pictures to existing posts, in Street Art City, you can apply to be part of the team that scouts new graffiti spots.No additional features are found on this website.

There seems to be an option to sign in, but it is unclear how it works because after signing in, the website presents the user with a message, saying that their account is not part of any city. Additionally, if the user logs out, they are required to sign in the same way as they created an account: through an email that is sent to their inbox. Therefore the additional advantages of creating an account are unknown.

2.2 Street Artwork

Another example of a similar project is a website called Street artwork. It is quite similar to Street Art Cities, although it is less popular and seems to be a less developed version.
This website includes a map on the landing page with markers in the spots where graffiti can be found. Pop-up windows are also shown after pressing on the marker, although all the information that is found on the pop-up is the picture. After clicking on the picture the site redirects the user to a page with more information about that graffiti, such as location, a short description on how to find the exact location of the graffiti, artist information, the date when the photo was taken, etc. There is an option to view the picture in full size and at the bottom of the page the user is presented with 'neighbors' of the chosen graffiti that are other graffiti within 100 meters from the chosen graffiti. There is a page called 'artworks' where the user can explore all artworks posted on the website with the additional ability to filter them by country and city. After clicking on a chosen picture, the user is taken to the same full graffiti page mentioned above.
There is an option to explore artists in the same principle as the graffiti explore page mentioned above. After clicking on a chosen artist, the user is taken to a page with pictures of the artist's work, a short description, the artist's website, and Instagram, if applicable. There is also an option to see the artist's work displayed on a map. This is a feature that is not seen on Street At Cities nor on our project.
Another interesting feature that has not been seen in other projects is an events page. It lists upcoming street art events. Although, this feature is either unfinished, because there are no events to be found, or no events are planned in the upcoming year.
A blog page is also listed in the menu. Clicking on this column redirects the user to another page that is dedicated to street art blogging. Lastly, contrary to Street art Cities, the user does not need to sign up to be a 'graffiti hunter'. Any user is free to submit a post, even without creating an account, although that option is also available.
All in all, even though this website is not fully developed, it has quite a few functionalities that differ from Street Art Cities and were not planned to be included in our project.

3 Development cycle

The team's development cycle begins with discussions on tasks that have already been assigned to team members. As a team, we discuss the problems we encounter while working on our tasks and how to solve them. To move further into development, we discuss new ideas and problems that need to be solved and add them to the Kanban board on Jira. We choose relevant tasks for the current stage and move them to the *"Selected for Development"* column, while less important tasks are added to the *"Development Backlog"* column. The team then divides the tasks that will be worked on, and each team member creates a git branch for their task, with a representative name. Research and development for the task then begin, and when a solution is found, the team member opens a pull request for other team members to review. If the solution is deemed suitable, the code is then merged with the *Origin master"* branch.

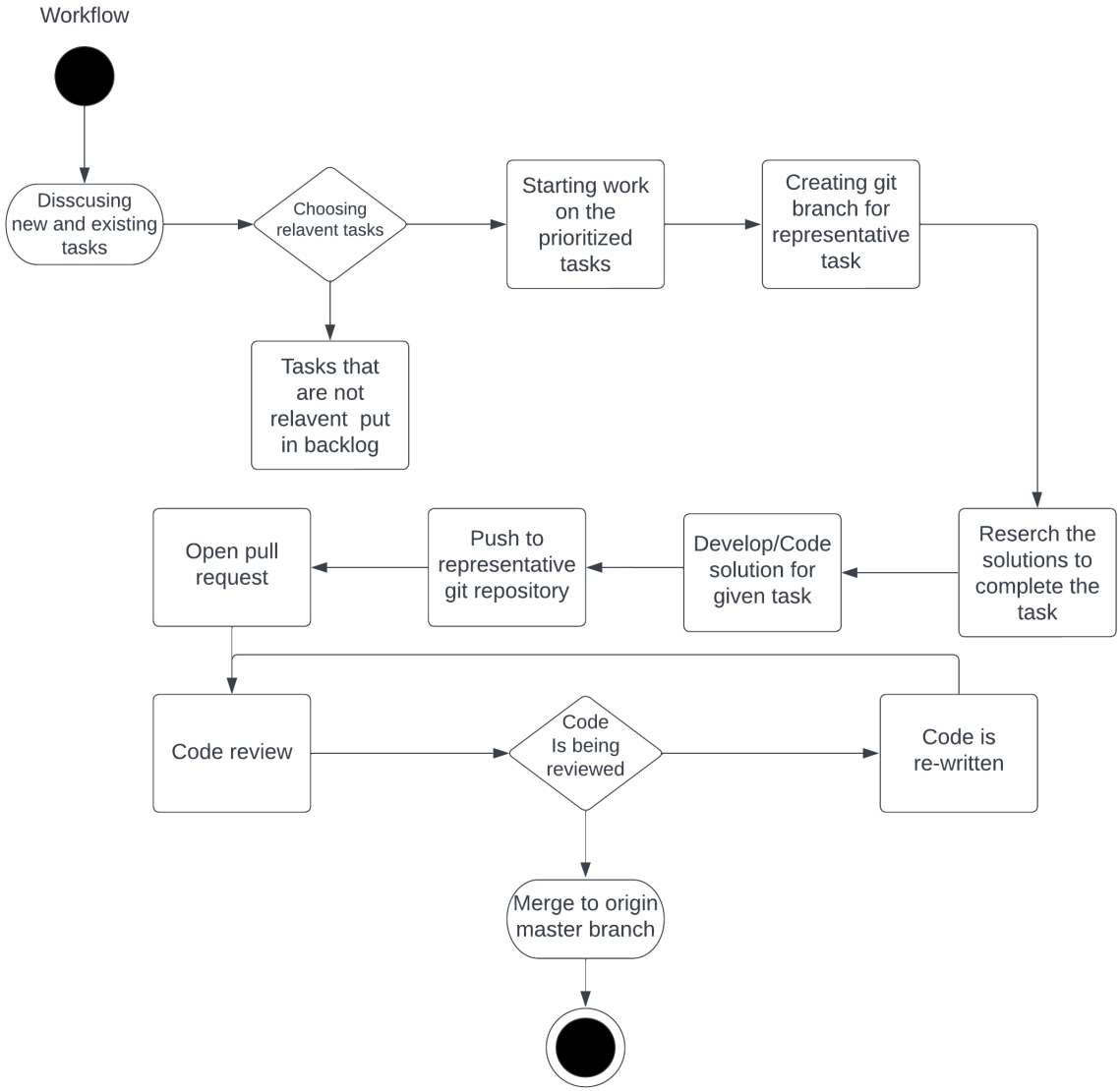


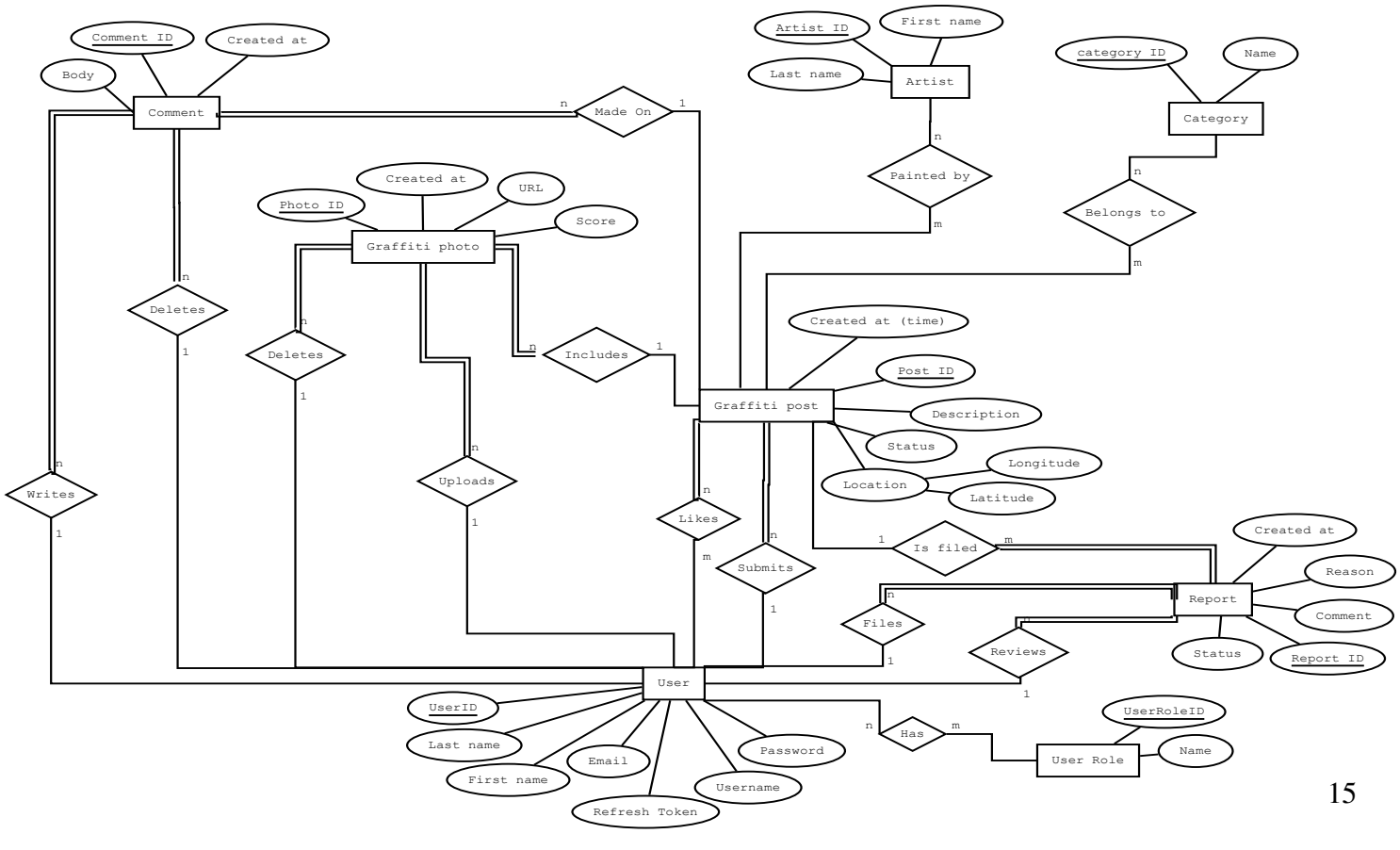
Figure 2. Work flow diagram

4 Database

4.1 ER diagram

A user can submit one or more graffiti posts. A user can like many graffiti posts and graffiti posts can be liked by many users. For a graffiti post to be submitted or liked, a user must exist. A user is described by a unique UserID, first and last name, email, username, password and refresh token. A graffiti post is described by a unique Post ID, the time it was created, a description, status and a location that consists of latitude and longitude. Many comments can be made on a graffiti post. A comment is described by a unique CommentID, the time it was created, and a body text, and cannot exist without a graffiti post. A user can also write and delete multiple comments (that were written by them). Comments cannot be written or deleted without a user. A user can also upload and delete many Graffiti photos that are included in a graffiti post. Graffiti photos cannot exist without a user that uploads or deletes them or without being included in a graffiti post. A graffiti photo is described by a unique Photo ID, the time it was uploaded, a URL and a picture score. Many Users can have many user roles that are described by a unique UserRoleID and a name. Reports cannot exist without a user that can file or review many of them. A report is described by a unique Report ID, status, the time it was created at, comment and reason for filing. Reports also must be filed for a graffiti post to exist. A graffiti post can include many artists that painted the graffiti if they are known and an artist can be the author of more than one graffiti. An artist is described by a unique Artist ID, a first and last name. A graffiti post can belong to many categories, which can also include many posts in them. Categories are described by a unique category ID and a name.

Figure 3. : ER model



4.2 RM

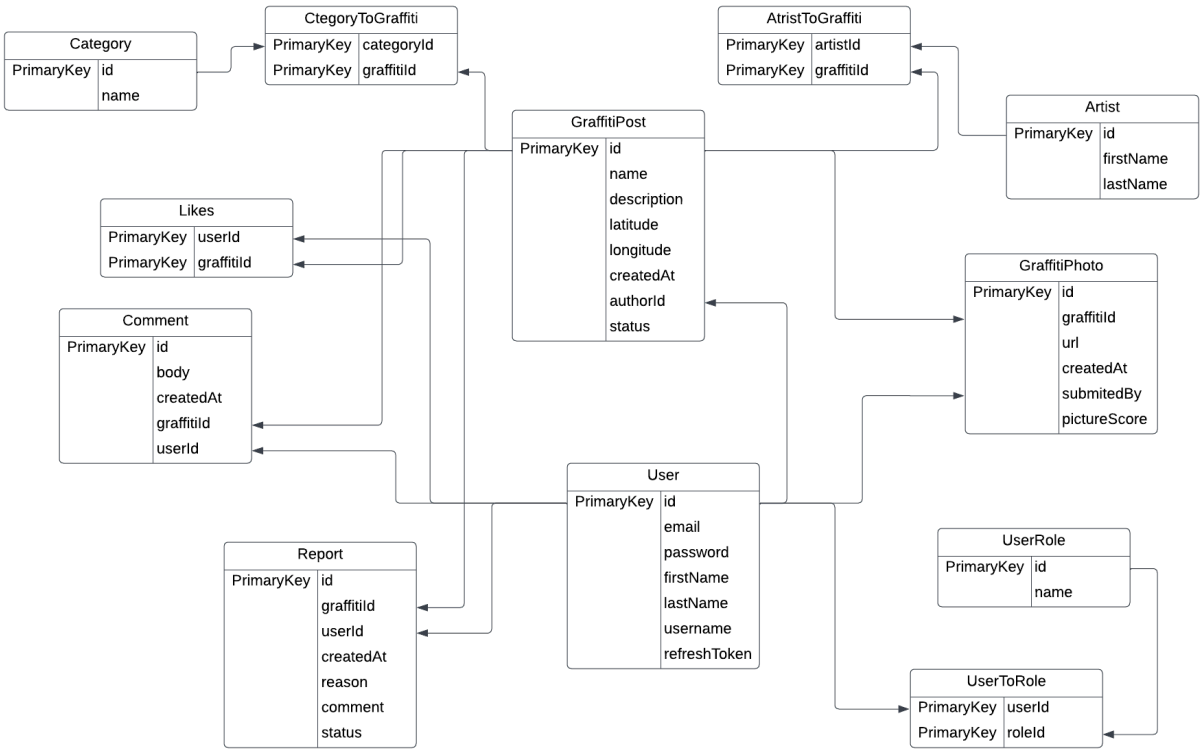


Figure 4. Relational Model

4.3 Object-Relational Mapping

The database will be created using the Prisma ORM. Prisma is a Typescript and Node.js Object-relational mapper (ORM) that allows faster and easier database management. It interacts between the back-end server and the database. The choice to use Prisma ORM (Object-Relational Mapping) instead of raw SQL is driven by several factors. One is the provision of a simple and intuitive API for interacting with the database, leading to more efficient and less error-prone development. Additionally, Prisma automatically generates a safe and efficient database schema based on the data model, eliminating the need for manual SQL migrations. Overall, using Prisma ORM greatly improves the development process by simplifying database management and reducing the potential for errors, while also providing a safer and more efficient database schema.

5 Algorithms

5.1 Nearest graffiti algorithm

The nearest graffiti algorithm is intended to help users add photos to an already existing graffiti when they try to submit it as a new graffiti post. The algorithm is loosely based on the nearest neighbor algorithm and uses Haversine's formula[1] to calculate the distance between locations.

- The **calculateDistance** function takes four parameters: latitude1, longitude1, latitude2 and longitude2, which represent the latitude and longitude of two coordinates. It uses the Haversine formula to calculate the distance, in kilometers, between the two coordinates.

```
FUNCTION calculateDistance(latitude1, longitude1, latitude2, longitude2)
grafiti = FETCH grafiti FROM DATABASE
latitudeDifference = toRadians(latitude2 - latitude1)
longitudeDifference = toRadians(longitude2 - longitude1)
a = sin(latitudeDifference / 2) * sin(latitudeDifference / 2) + cos(toRadians(latitude1)) *
cos(toRadians(latitude2)) * sin(longitudeDifference / 2) * sin(longitudeDifference / 2)
c = 2 * atan2(sqrt(a), sqrt(1 - a))
RETURN EarthRadiusKm * c
```

- The **toRadians** function is used to convert the input degrees to radians, which is necessary for the Haversine formula.

```
FUNCTION toRadians(degrees)
RETURN degrees * (PI / 180)
```

- The **findNearestNeighbor** function takes three parameters: graffitiList, which is an array of graffiti objects, and latitude and longitude. It then loops through the graffitiList, calculates the distance between the given coordinates and current graffiti, and if the distance is less than or equal to 1 km it pushes the graffiti object into nearestNeighbors.

```
FUNCTION findNearestNeighbor(graffitiList, latitude, longitude)
FOR i = 0 TO LENGTH OF graffitiList -1
graffiti = graffitiList[i]
distance = calculateDistance(graffiti.latitude, graffiti.longitude, latitude, longitude)
IF distance <= 1
APPEND graffiti TO nearestNeighbors
RETURN nearestNeighbors
```

5.2 Metadata removal

General Data Protection Regulation (later GDPR) prohibits posting data online that can be used to identify a person. Such data can be found in image metadata, such as geolocation, device information, etc. This project deals with a lot of picture data. Users have the ability to upload photos to the website, where they will be displayed for others users to see. To avoid crossing boundaries of the law, this project makes sure that no personally identifying information is accessible after uploading images. Therefore after an image is uploaded, its metadata, which is stored in the exif file format, is extracted and removed from the image. The metadata removal method works by this principle: it reads the file of the uploaded image, removes exif data using piexif and returns the image without the removed exif data.

6 Functional requirements

- Unauthorized user:
  - Can access the map feature and view the location of graffiti artworks
  - Can view and explore graffiti artworks in detail
  - Have the capability to register an account
- Authorized user:
  - Can log in to their account
  - Have access to the map feature and view the location of graffiti artwork
  - Can view and explore graffiti artworks in detail
  - Can submit new graffiti artwork for consideration
  - Can interact with existing posts by liking, commenting, unliking, and deleting their own comments
  - Can report posts that violate the website's guidelines
  - Have the capability to add photos to existing posts
- Admin user:
  - Have the ability to review and confirm or deny user-submitted graffiti artworks
  - Can edit the data associated with graffiti artworks
  - Have the ability to set appropriate categories for graffiti artwork
  - Can review user-reported posts and comments
  - Have access to all functionalities available to authorized users

- Use case diagram:

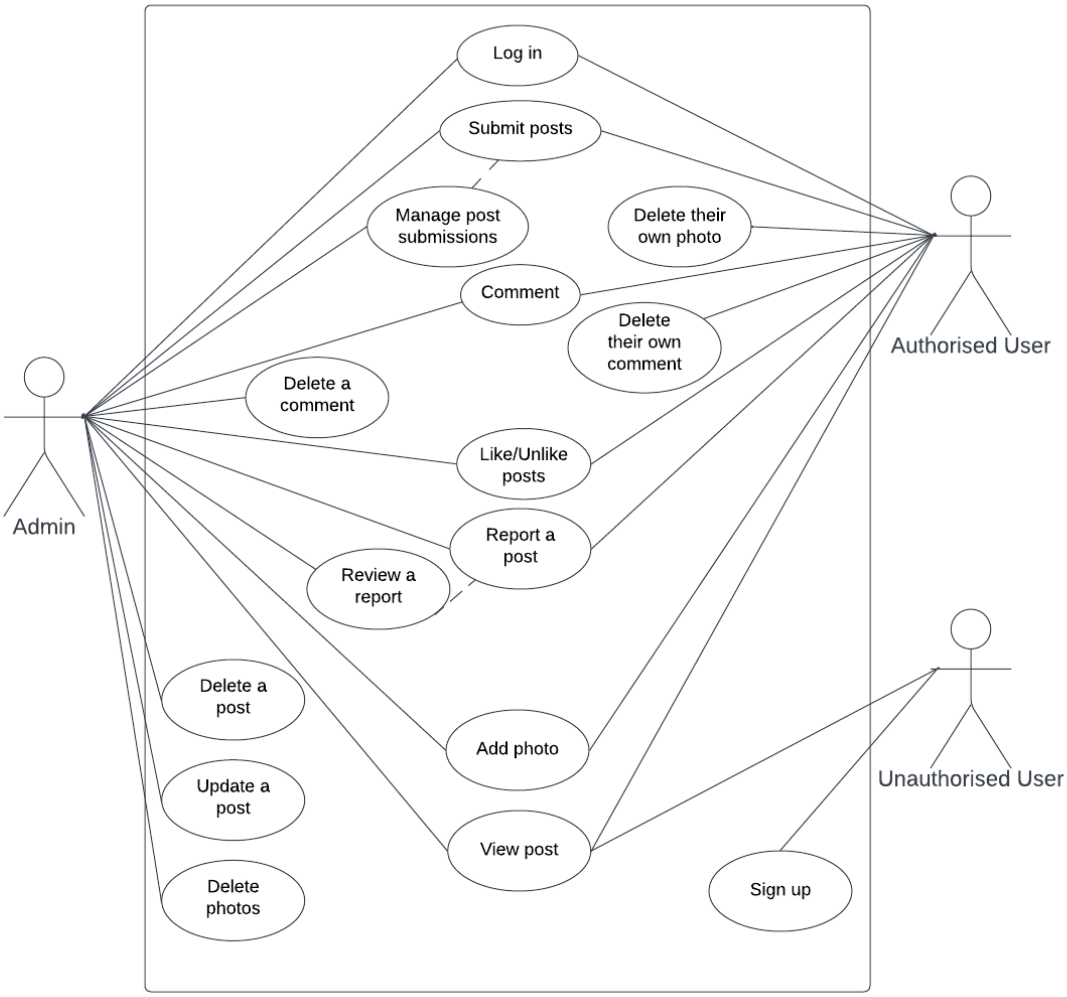


Figure 5. Use case diagram

7 Non-functional requirements

- Portability:
  - The main goal is to make the web application fit for use on the most popular computer screen sizes and various devices: from the smallest mobile phone to a tablet, from the smallest laptop displays to the most popular monitor sizes available. Website applications must achieve that without losing any functionality or performance.
- Performance:
  - The web application should load relatively quickly and handle some higher traffic at this stage of the project.
- Usability:
  - To ensure that the web application will be easy to navigate the layout of the pages have a clear and intuitive layout. The different pages and functionalities should be easily discoverable and understandable to every user.
- Security:
  - To ensure that the web application will not be vulnerable the metadata of the graffiti pictures will not be stored on the database as well as the geolocation of the user. In case of a data breach, no sensitive Information could be taken.
- Compatibility:
  - The web application is designed to work seamlessly across all commonly used web browsers, ensuring that users can access the full range of features and functionality regardless of their browser of choice. Additionally, the layout and design of the web application have been optimized to adapt to different screen sizes, ensuring that users can easily interact with the application on any device, including desktop computers, tablets, and smartphones, without losing access to any features.
- Scalability:
  - To ensure that the web application will be able to handle the increasing number of users and graffiti submissions This means that the web page should be designed and built in such a way that it can easily accommodate more users and more data without requiring significant changes to the underlying architecture.

8 Testing

8.1 Automated testing

Automated testing was used for some unit tests with the intention of increasing testing speed and efficiency, and minimizing manual work. The tool of choice for automated testing was Jest. Unit tests for different modules were written in service.specs.ts and controller.specs.ts files to make sure they are written correctly and work without any interference. Jest interacts with Prisma Client to fetch data from the database, if needed. It is important that tests do not interfere with the main database, therefore mocking was planned to be implemented to write automated tests. Due to the lack of time, this was not fully implemented.

8.2 Map Component testing

Test Type: Manual testing (map component testing)
Test case description: Test the resize capabilities of the map component.
Preconditions: The map component of the website loaded successfully.
Assumption: The user wants to check something on their desktop or wants to use multiple applications and have a compatible web browser.
Test scenario: The map component stays in the correct position when the user decides to change the size of their preferred browser and maximizes it again.
Expected result: The map component resizes when the browser window is made smaller and when the browser is maximized again the component resizes to its previous state and stays in the correct position.
Test steps:

1. Load the home page of the web application when the browser is maximized.
2. Resize the web browser and make it smaller.
3. Maximize the browser window.

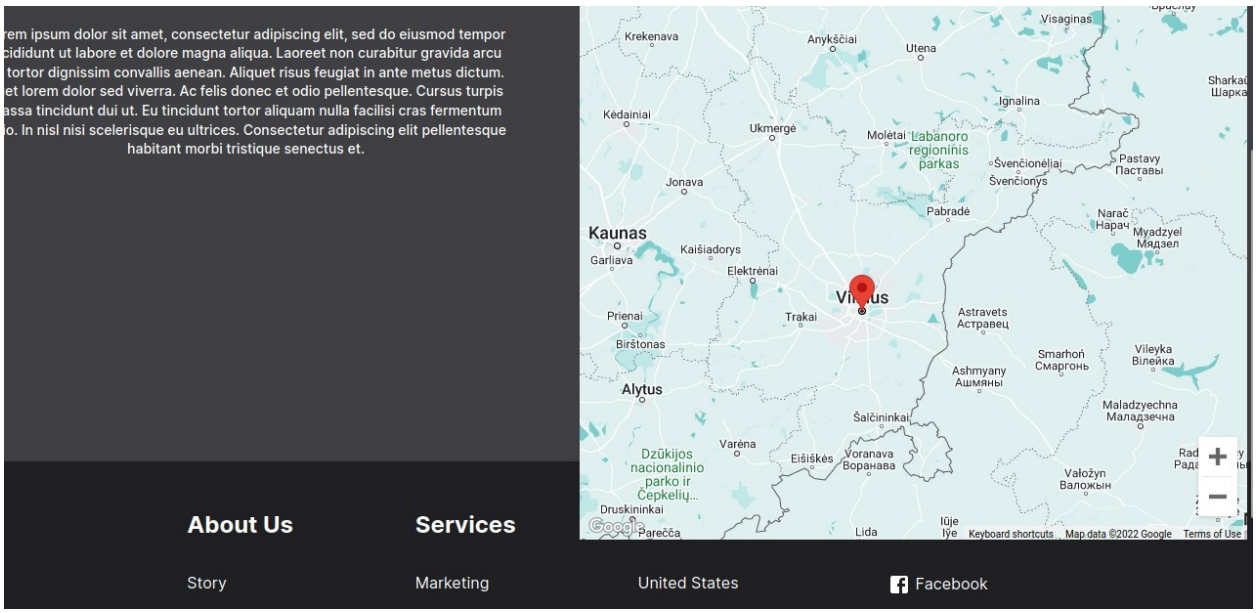


Figure 6. Map component after testing

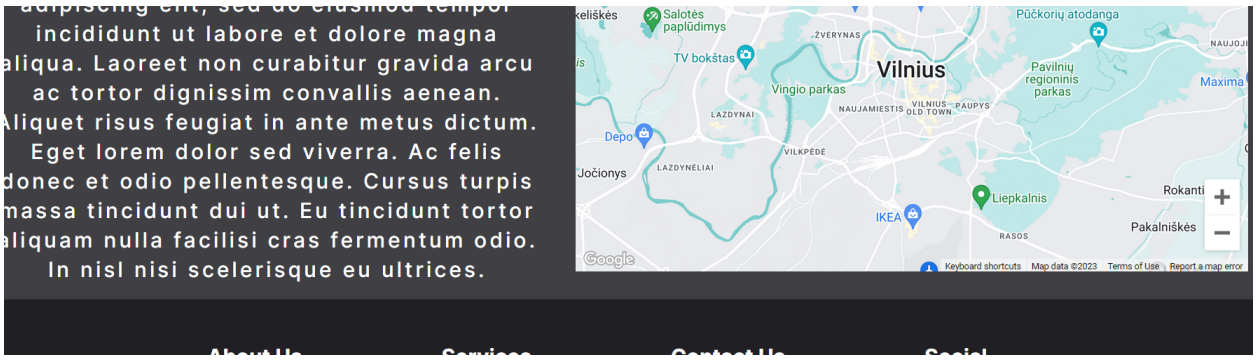


Figure 7. Map component after fixing the bug

8.3 Pan to users location testing

Test Type: Manual testing (button component, map component and permissions)
Test case description: Test if the button calls up the location permission window and the map component receives the data and pans to the user's location.
Assumption: The user has a compatible web browser.
Test scenario: In order for this feature to work, the user has to allow the web application to access the location data.
Expected results: User location has been taken and sent to the map component and the map pans to the general location of the user.
Test steps:

1. Load the home page of the web application.
2. Press the compass icon.
3. Press "Allow" on the prompt.
4. Wait for the map component to pan to your location.

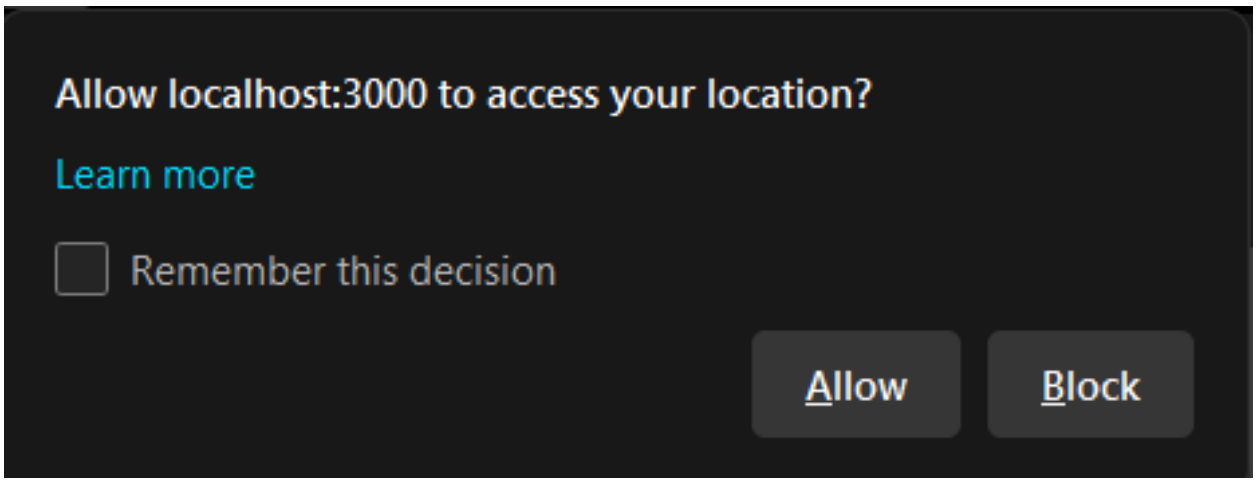


Figure 8. Location prompt after pressing the compass icon

8.4 Graffiti post submission

Test Type: Integration Testing (Graffiti post creation module, graffiti photo creation module)
Test case description: Test if a graffiti post is successfully created in the database.
Preconditions: A valid user exists, given data, is correct and formatted correctly. **Assumption:** Account with provided test data already exists in the database and can be used to log in, the user has an internet connection enabled, allowed the application to access needed information, and the API server is activated.
Test scenario: When the user enters the graffiti submission screen, they are presented with a template to create a new graffiti post. The template consists of a picture upload, selecting a location, description, name and artists' names, if applicable.
Test Data:

1. Name: "Graffiti 1"
2. Description: "The description of this graffiti. An explanation about what it depicts and other information known about it."
3. Location: "54.689581, 25.276015"
4. Date: 2023-01-10
5. authorId: 1
6. CategoryIds: [2]
7. artistIds: [1]

Expected result: The post is saved in the database, and no error message is thrown. Now the user can go back to the map and see a pin on the map with the uploaded graffiti.
Test steps:

1. Go to the Graffitinity website.
2. Move the post submission page.
3. Create a post with the same data that is provided in the Test Data set.
4. Click the "Submit" button and return to the landing page to see an added pin to the map.

8.5 Automated Graffiti Photo creation test

Test Type: Unit testing (Graffiti photo creation module)
Test case description: Successful photo uploading and saving in the database.
Preconditions: Necessary data exists in the TestDataFactory.ts file, test data is formatted correctly.
Test Data:

1. graffitiId: 2
2. url: https://graffinity-images.s3.eu-central-1.amazonaws.com/202211.jpg
3. addedAt: 2023-01-11
4. userId: 1
5. likeIds: [3]

Test scenario: The create module takes the given test data, creates a new object of graffiti photo details and returns that object.
Expected result: The photo object is successfully created, no errors are thrown, and all information about the object can be seen in Prisma Studio.

8.6 Successful registration process

Test Type: E2E Testing
Test case description: Test the Registration functionality in the application.
Preconditions: An account with provided test data must not exist in the database.
Assumption: The user has an internet connection enabled and established, allowed the application to access needed information and services, and the API server is activated.
Test scenario: The user provides the system with a username and email address that do not duplicate any usernames or emails that are already in the database and creates a password, therefore successfully registering to the system.
Test Data:

1. Username: registrationTest
2. Email: registrationTest@gmail.com
3. Password: password

Expected result: No errors should be thrown and with the registration being successful, the user should be redirected to the login page.
Test steps:

1. Go to the Graffinity website
2. Navigate to the registration page
3. Enter username, email, password and repeat the password from the Test Data.
4. Click the “Register” button.

8.7 Unsuccessful registration process

Test Type: E2E Testing
Test case description: Test the Registration functionality in the application.
Preconditions: An account with provided test data already exists in the database.
Assumption: The user has an internet connection enabled and established, allowing the application to access needed information and services, and the API server is activated.
Test scenario: The user provides the system with an already occupied username or email address, and can not create a duplicate account due to duplication issue.
Test Data:

1. Username: CameraWoman
2. Email: camera.woman@gmail.com
3. Password: hashedPassword

Expected result: An error message should be thrown informing the user that the account is already registered with the provided username or email address and registration should be denied.
Test steps:

1. Go to the Graffinity website
2. Navigate to the registration page
3. Enter username, email, password and repeat the password from the Test Data.
4. Click the “Register” button.

8.8 Login authentication

Test Type: E2E Testing
Test case description: Test the Login functionality in the website.
Preconditions: Account with provided test data already exists in the database and can be used to log in.
Assumption: The user has an internet connection enabled, allowed the application access needed information, and the API server is activated.
Test scenario: Authenticate the user to further usage of the application when provided with the correct username and password combination.
Test Data:

1. Username: CameraWoman
2. Password: hashedPassword

Expected result: No error is thrown, login is successful, and the user is redirected back to the landing page.

Test steps:

1. Go to the Graffinity website.
2. Go to the login page.
3. Enter the correct username and password combination, provided in the Test Data set.
4. Click the “Login” button.

9 Conclusions

Conclusions To sum up, the purpose of Graffinity is to promote street art globally. The website is accessible at no cost, and users can participate by sharing pictures and submitting posts, thus building an online community of graffiti art enthusiasts This is made possible with separate and secure user accounts for every user.Furthermore, Graffinity serves as a platform for artists to share their work online and connect with a community of street art enthusiasts, allowing them to showcase their work beyond the city. This project uses a picture scoring algorithm to determine the best quality pictures and order them accordingly. After a picture is uploaded, the algorithm assesses it and places the image into the array of images in the post according to its picture score. Moreover, the Graffinity website can be used not only to gather a street art community but also create new ways to explore the city for tourists and interested locals. The website features a map with all graffiti spots that are in the database displayed on it. Users can use the web application to navigate through the city and explore the street art in person.

9.1 Future work

Future development of Graffintiy is to expand and improve some functionalities of the website and improve the user interface by making it more appealing. In the future, the website is expected to have a “Graffitis near me” functionality, search possibility, an explore page, where graffiti can be filtered by category, added user roles and an improved user profile and authentication. Some of these functionalities can already be discovered in the backend of this project, but they are not implemented in the frontend, therefore are not attainable to the users.

9.2 Limitations at this point of the project

- Cannot select any artists while uploading a picture
- Cannot upload multiple images at once.
- For now, the graffiti full-view page and gallery page have locally stored photos.
- Sign up functionality is not fully implemented.

References

[1] Cecep Nurul Alam, Khaerul Manaf, Aldy Rialdy Atmadja, and Digital Khrisna Aurum. Implementation of haversine formula for counting event visitor in the radius based on android application. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1--6, 2016.

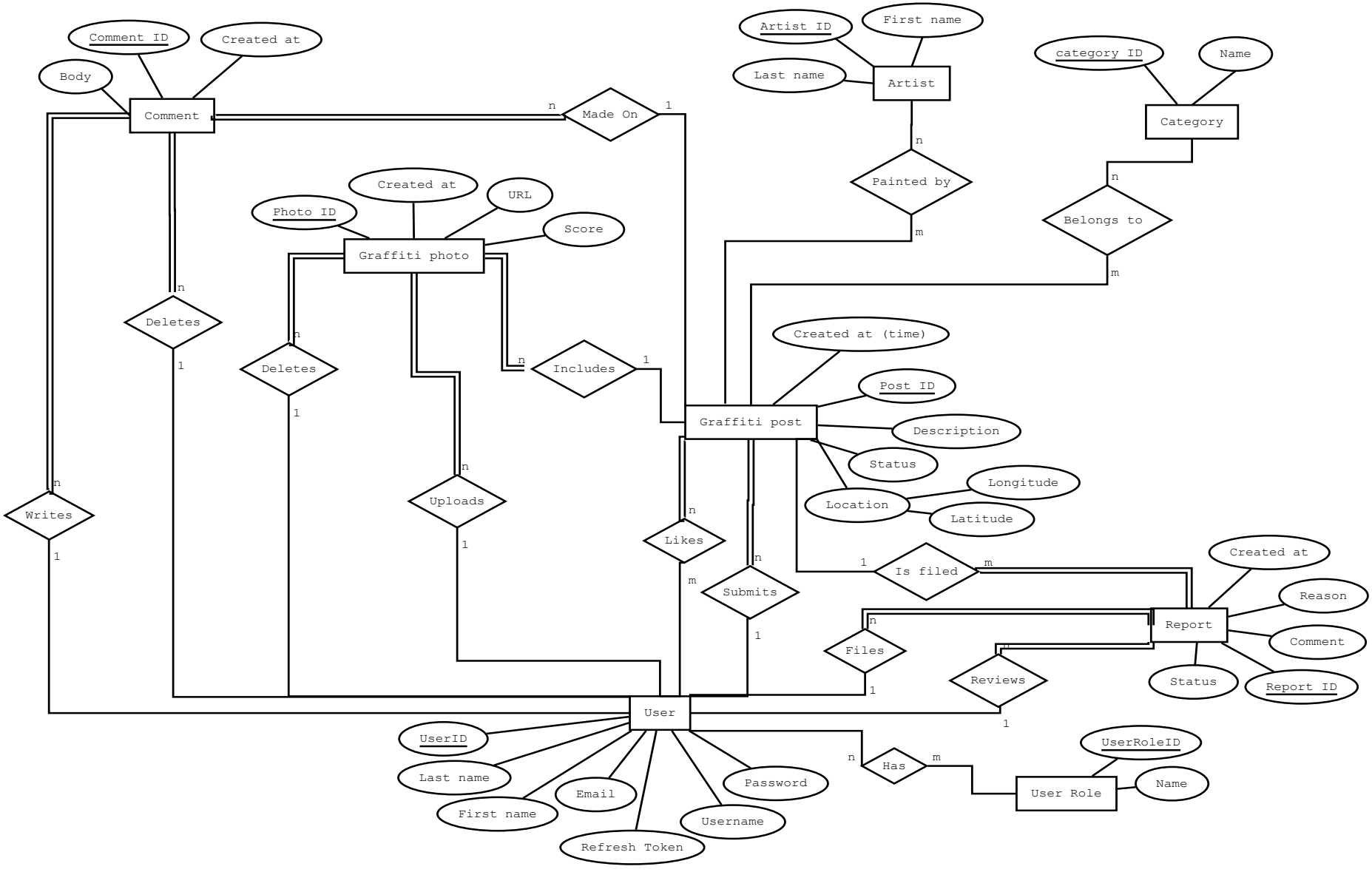
[2] Yuming Fang, Hanwei Zhu, Yan Zeng, Kede Ma, and Zhou Wang. Perceptual quality assessment of smartphone photography. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] Ali Lenjani, Chul Min Yeum, Shirley Dyke, and Ilias Bilionis. Automated building image extraction from 360° panoramas for postdisaster evaluation. *Computer-Aided Civil and Infrastructure Engineering*, 35(3):241--257, 2020.

[3] [2]

10 Annexes

- **Link to the website:** https://graffinity.art/
- The full view of the ER model is provided on the next page.


Comment ID
Created at
Body
Comment
Made On
Artist ID
First name
Last name
Artist
category ID
Name
Category
Painted by
Belongs to
Created at
URL
Photo ID
Score
Graffiti photo
Deletes
Deletes
Includes
Created at (time)
Post ID
Graffiti post
Description
Status
Longitude
Location
Latitude
Writes
Uploads
Likes
Submits
Is filed
Created at
Reason
Comment
Report
Status
Report ID
Files
Reviews
User
UserID
Last name
First name
Email
Password
Username
Refresh Token
Has
UserRoleID
Name
User Role