MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

MYCITY: A PARTICIPATORY SENSING APPLICATION FOR LOCAL COMMUNITIES

LICENSE THESIS

Graduate: **Andrei-Radu KOBLICICA**

Supervisor: **Conf.Dr.Ing. Lia-Anca HANGAN**

2023

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Andrei-Radu KOBLICICA**

MYCITY: A PARTICIPATORY SENSING APPLICATION FOR LOCAL COMMUNITIES

1. **Project proposal:**
   The purpose of this application is to improve the quality of life in a city community by allowing citizens to be more easily in contact with public institutions and service suppliers. The app would also allow the users to be more informed about things that are happening in the city, as well as its institutions and facilities.

2. **Project contents:** *Presentation page, advisor's evaluation, introduction, project objectives, bibliographic research, analysis and theoretical foundation, detailed design and implementation, testing and validation, user's manual, conclusions, bibliography, appendices.*

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**:

5. **Date of issue of the proposal:** October 21, 2022

6. **Date of delivery:** July 6, 2023

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declarație pe propria răspundere privind autenticitatea lucrării de licență

Subsemnatul(a) Andrei-Radu Koblicica, legitimat(ă) cu CI seria MM nr. 964256 CNP 5001103245042, autorul lucrării „MyCity: A participatory sensing application for local communities”, elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Calculatoare și Tehnologia Informației, din cadrul Universității Tehnice din Cluj-Napoca, sesiunea iulie 2023 a anului universitar 2022-2023, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

In cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data	Nume, Prenume
06.07.2023	Koblicica Andrei-Radu

Semnătura

Table of contents

Chapter 1. Introduction 1

Chapter 2. Project Objectives 3

2.1. Main objective 3

2.2. Functional requirements 4

2.3. Non-functional requirements 5

Chapter 3. Bibliographic Research 6

3.1. Participatory sensing 6

3.2. City service and ICT applications 7

3.3. Similar Applications 7

3.3.1. App descriptions 7

3.3.2. Comparative analysis 9

Chapter 4. Analysis and Theoretical Foundation 11

4.1. Problem and product position 11

4.2. User descriptions and needs 12

4.3. Features 13

4.4. Use cases 16

4.5. Conceptual architecture 37

Chapter 5. Detailed Design and Implementation 41

5.1. Design and implementation of the application server 42

5.2. Database structure 44

5.3. Design and implementation of the server for file uploading 45

5.4. Design and implementation of the web user interface 45

5.4.1. Routing and navigation 46

5.4.2. Components 47

5.4.3. Services 48

5.5. Design and implementation of the mobile application for citizens 49

5.5.1. Activities 50

5.5.2. Fragments 51

5.5.3. Navigation 51

5.5.4. Models 52

5.5.5. Adapters 52

5.5.6. APIs for server communication 52

5.5.7. Android resources 53
5.6. External Services 53
5.6.1. Google Cloud Storage 53
5.6.2. Google Maps, Location and Geocoding APIs 54
5.6.3. Firebase Cloud Messaging 54

Chapter 6. Testing and Validation 55

Chapter 7. User's Manual 59
7.1. System Requirements 59
7.2. Installation 59
7.3. Use Manual 60

Chapter 8. Conclusions 67

Bibliography 68

Appendix 1 – List of figures 69

Appendix 2 – List of tables 71

Chapter 1. Introduction

To define the context of the project, the first thing that needs to be addressed is the physical environment in which this project is relevant and designed to be used, which is the city of Cluj-Napoca.

For a more detailed exploration of the project context and a clearer domain specification, we need to analyze the problem statement. The main problem that this application attempts to solve is the difficulty of the maintenance and well-being of the infrastructure and facilities in Cluj-Napoca, such as the state of roads, parks and green spaces, street signs and illumination, parking spaces, water and sewage networks and so on. This maintenance should not only address the functioning of the mentioned city assets, but also their overall salubrity and state compared to more developed cities.

By promoting a way for the city authorities to be aware of the issues that inevitably crop up regarding the city's infrastructure and facilities, and by in turn allowing them to solve these issues faster, it leads to a better quality of life for its inhabitants. This improvement to the overall satisfaction of citizens should be one of the main priorities of the public institutions responsible for the functioning of the city, as it leads to more economic opportunities that facilitate the city's growth, a safer environment to live in, and a better general health, well-being, education and personal development of the people. All these factors also help promote the city on a social and cultural scale.

In this context, a problem that often appears is that of the communication between the city's inhabitants and its authorities. When encountering an issue that affects the people's quality of life, they sometimes don't know who is responsible for solving that issue, or who to contact in order for it to be addressed. This might lead to them ignoring that problem, hoping that someone else takes the initiative to report it to the authorities, or hoping that the authorities are aware of the problem's existence and are already working on a solution. They might complain on social media, or to other people that are also affected by that problem, and while this sometimes leads to something being done about it, it is not always the case. Even if this approach can and eventually does lead to the issue being addressed and taken care of, the time in which this process happens is almost always longer than if the citizens were provided a fast and simple way to report these issues as soon as they are encountered.

By promoting an interface through which people can register their complaints and know that they are actually received by the entities that they are addressed to, it not only leads to the chance of them to be addressed faster, but also to less frustration, conflict and speculation.

Solving this problem not only benefits the inhabitants of the city, but also the local institutions, because they can be aware of more things that need to be solved, and are provided a centralized way of receiving the complaints and reported issues, which allows them to more easily assign people to handle them, or share them with other institutions that need to take part in the solution.

Since any piece of infrastructure or any facility can be damaged and these things happen inevitably at some point, all inhabitants are bound to be affected by such problems. This is why, when defining the social environment of the project, all the people in Cluj-Napoca can be considered as part of the context, as they have a direct or indirect interest in the resolution of some of the problems.

So far, regarding the problem of inefficient communication between citizens and public institutions, we have mostly discussed about the communication from the citizens, who have a complaint or a problem to report, to the entities that are designated to address them. To fully place the project scope inside a context, the other direction of communication also needs to be considered.

In the context of providing people a way to send such reports, there should also be a way for institutions to provide feedback on the status of the complaints, namely if they have been viewed, acknowledged, if their resolution is in progress, or if they have been solved. Also, they should be able to ask for further clarifications regarding the issue if needed. Besides this, the people who have created that complaint should be aware of the entities that are involved in addressing it, and should also be able to provide additional information, or ask questions if they have any.

To further elaborate on the described problem and improve the interaction between the two parties, general communication from the institutions towards the city inhabitants has to exist. This is obviously happening to some extent, as most institutions, such as the city hall and municipality, already have official websites where they post news, press releases, announcements or any other relevant information, but their accessibility to the general population can be improved with the use of an application that focuses on aspects regarding the state of the city.

This way, the people could have a centralized location to check news and announcements made by all the institutions, and also provide a way for institutions to reach a larger part of the population, not just those who visit their website. Furthermore, a mobile application is a more suitable interface for institutions to send alerts or more critical information on aspects that are of more important or immediate effect.

Another way of improving the quality of life of the citizens of Cluj-Napoca is to allow them to more easily find the contact information of the institutions, such as their phone number, email address, or their official website and location in the city.

Regarding the domain of the project, the broader category to set it in is technology, namely software engineering, as the end product consists in a mobile application that attempts to solve the described problem by bridging the communication gap between the citizens of Cluj-Napoca and the public institutions and service providers. This allows for a more accessible and convenient use than some other alternatives that already exist. As for the institutions that would be using the product, a web application is more suitable, as it provides a more efficient way to track the reported issues, assign and share them, as well as post news and send alerts.

For a more specific description, the domain of the project could be classified as public service management, because one of the main purposes that the project is attempting to achieve is to allow the institutions belonging to the public sector for Cluj-Napoca to manage infrastructure and facilities more efficiently by integrating the population's complaints and feedback.

From the city's inhabitants' point of view, the domain could be seen as citizen engagement, as they can take part to help institutions solve the issues that appear by reporting them, but the application also serves as a platform for accessing information in multiple aspects regarding Cluj-Napoca. They would be able to view news and alerts provided by institutions, details about parks and facilities that are available in the city, as well as events that are happening, such as festivals, concerts, parties, exhibitions, shows, sports events, social or educational events, and also create their own for other people to see.

Chapter 2. Project Objectives

This chapter contains the objectives that the MyCity application is trying to achieve, from the point of view of the regular users, and that of the institutions.

The general objective of this application is to improve the quality of life in Cluj-Napoca by providing the citizens an interface for reporting city issues to public institutions, while also allowing them to be more informed about things that are happening in the city, and about its institutions and facilities.

The project is divided into two main components, each designed for a specific type of user:

- The Android mobile component, made for anyone that lives in Cluj-Napoca or wants to find out more about the city.
- The Web component, which is only accessible by employees of public institutions in order to view and solve the issues reported, add news and send notifications, and by the administrator of the application in order to manage all aspects of the application: users, institutions, facilities, events, issues, news and alerts.

2.1. Main objective

The main objective, as described by the context of the project, is to accomplish a better and more efficient maintenance of the city's facilities and infrastructure, as well as provide citizens with a way to report the public sector related issues they encounter so they can be solved. To achieve this, a cooperation between the regular users and the public institution employees is required.

- **Allow user to report issues**

Citizens can encounter problems that affect them or other people in the city, such as potholes or other road issues, damaged public or personal property, power, water and streetlight outages, park problems, fallen trees, abandoned vehicles, waste, pollution and many others. Some of them might be minor and not influence population in a very meaningful way, but some require immediate attention from the authorities that are responsible with solving them.

Providing the general population with a way to report them as soon as they are encountered allows the institutions to be aware of their existence faster, and take the appropriate action accordingly.

Some issues need to be solved by certain institutions, and others by different institutions. For example, damaged infrastructure or public property such as parks is handled by the city hall which assigns the resources necessary for handling them, while water outages or damaged pipes are solved by water companies.

When a user reports an issue, the system automatically assigns it to the institution responsible for solving it based on the selected type, allowing the user not to think about who to contact.

- **Allow institutions to solve issues**

Once an issue is assigned to an institution, all its employees are able to view its details, and one of them is able to assign it to themselves, which is when the status of the issue is set to “In progress”, notifying the user that it is taken care of. In case the reported issue is not real or justified, it can be deleted. When the issue it solved, the employee that assigned it marks it as “Solved”, and the successful status can be seen by the user.

- **Allow institutions to re-assign issues**

The system assigns the issues based on their type, but obviously, each issue is different and some might be incorrectly assigned. In case an institution receives an issue which cannot or should not be solved by them, they can manually re-assign it to another institution which has the authority or ability to do so.

- **Allow institutions to share issues**

Some issues might require the attention of multiple institutions, which introduces the requirement of a share feature, where the initially assigned institution can share the issue with multiple others, which will also be involved and able to view it. Once multiple institutions are involved, the issue is solved only when all involved institutions mark it as solved.

- **Allow involved parties to communicate**

Each reported issue has an associated chat section where the user who reported it and the institution employees who are involved in solving it can discuss in case there are some questions that need to be asked or further clarifications that need to be provided.

2.2. Functional requirements

- **Institution details**

Regular users can view all public institutions and service providers, in case they want to fiind their contact details such as email or phone number, access the institution’s official website or get directions to where it is located in the city.

- **News**

The institutions are able to post news or announcements about their activity, functions or objectives, which the users can view in the application.

- **Alerts**

Institutions can also send alerts which are received as notifications for users. These should usually involove information that is of high importance and should reach as many people as fast as possible.

- **Facilities**

The users can view a list of parks and facilities that exist in Cluj-Napoca and filter them based on type, in case they are looking for something specific: park, sports facility, mall, library, cinema. They can view details about the facility, such as its image, location, description and official website, and get map directions to them.

- **Events**

The application contains a list of events that are happening in Cluj-Napoca. Users can browse to fiind events they are interested in, and like with facilities, filter them based on type: concert, party, festival, show, sports, social, educational, business etc. They can also view the event details, such as the location, date and time, description, how many people are interested in it, and they can add it to favorites. Besides this, users can create their own events, which are added to the application once the administrator approves them.

- **Profile**

When users first download the app, they are able to create an account and log in. After being logged in, they can perform all the described operations and also view their profile, which contains their information and some statistics. They can change their profile details, log out or delete their account.

- **Manage data**

The system allows its administrator to perform create, read, update, delete operations on all entities in the context of the application. Institutions can also update their own details, create news and alerts, and create institution accounts that they assign to their employees in order for them to access the app.

- **Analytics**

The administrator has a dashboard section where they can view analytics about the entire application context, such as the number of users, institutions, facilities, news posted, alerts sent, issues reported, events created, as well as charts with issue and event distribution based on type and status.

2.3. Non-functional requirements

- **Security:** The application aims to provide security of personal data, such as passwords, issue details, chat messages through authentication and encryption
- **Performance:** The system should provide good response times and overall performance
- **Usability:** Both the mobile and web applications should have an interface that is simple to learn and use, which would allow users to perform tasks without encountering difficulties
- **Maintainability:** The application should be easy to maintain by the administrator, and the underlying implementation should allow for modifications if required
- **Compatibility:** The system should be compatible to most of the citizens' mobile devices and employees' computers

Chapter 3. Bibliographic Research

This chapter highlights some bibliographic research performed while developing this application, mainly focusing on and detailing the idea of participatory crowdsensing, which is the concept behind the issue reporting feature described previously. It also includes an overview of some applications which perform similar objectives, which this application aims to include and build upon. The final section provides a comparative analysis of the features provided by each application.

3.1. Participatory sensing

To understand this concept, we must first explore the idea of crowdsensing and its motivation.

As highlighted in chapter 1 of [1], smartphones have evolved into powerful tools which provide a large variety of sensors such as cameras, GPS, or accelertometers, which allow users' surrounding information to be gathered and processed easily.

This introduces the idea of crowdsensing, which in the context of this project is defined by the authors of [2] as the "measurement and collection of data through different kinds of sensing devices by a large mass of users, with the goal of sharing collected data with Smart City citizens through a common ICT platform for the sake of the community wellbeing."

The introduction section in article [3] classifies the crowdsensing concept in two types: participatory sensing and opportunisting sensing. Participatory sensing involves the sensor data related to large scale phenomena being provided through the active involvement of users, while the process of opportunisting sensing allows for minimal involvement of individuals by automatically collecting data without their explicit action. Given the nature of the problem this project is attempting to solve, participatory sensing is used.

This approach where citizens take initiative in the issue reporting process is motivated in chapter 2.1 of [4], which says that "humans are indeed the most versatile sensors. They can make judgments, where traditional sensing technology falls short. They are already embedded in spaces that are interesting to measure. They are good at understanding context, detecting anomalies, interpreting complex scenes, and prioritizing information transfer". This idea is also supported in chapter 2.1 of [5], which emphasizes that the sharing of sensed data is manually performed by users, who decide how, when, what and where this process happens.

The authors of [5] introduce the idea of a Participatory Sensor Network, which consists of a distributed system that allows data about personal daily experiences and other aspects regarding the city to be collected. With the use of such a PSN, the state of a city can be monitored with the help of citizens, who make their sensed data available or actively share information about certain contexts.

As mentioned in the introduction section, the maintenance and growth of the city can be improved with the use of such system, which institutions would have access to in order to process the gathered data.

Using the smartphones' sensory information raises the problem of privacy and protection of the users' personal data. This concept is explored in chapter 4 of [6], which highlights that this concern mostly affects opportunistic crowdsensing applications, in which the user doesn't actively provide the collected data. The author also explains that participatory sensing can also pose privacy issues, since elements such as location data need to be provided in order to enable the functioning of the application. [6] describes some ways of dealing with privacy issues, such as anonymization, encryption and data perturbation. In the case of the

MyCity application for Cluj-Napoca, partial anonymization is achieved, since institutions only have access to the individual's username, and the choice of providing more sensitive information such as a name, email and phone number along with the issue's details is in the hands of the user, since additional interaction can be performed in the application. Location information is necessary in this context, but in most cases it is related to something encountered in the city and doesn't raise the issue of providing a personal address.

3.2. City service and ICT applications

So far the crowdsensing idea used for issue reporting has been explored, but this application aims to provide more use for the inhabitants of Cluj-Napoca, allowing them to also access information regarding more general aspects of the city.

One such goal presented in chapter 2 is the possibility of receiving news and alerts. The authors of [7] explain how mobile consumption of news has increased in previous years compared to other types of consumption, and how news organizations have started to value the process of sending news alerts directly on users' devices as notifications, in order to achieve higher reach and build stronger relationships with audiences. This idea can be adopted in the context of news or alerts provided by the city's public institutions and service providers, who would be able to achieve these benefits by using a more mobile-friendly and centralized approach, as opposed to news being posted on their official website or on social media applications.

Article [8] explores the idea of Information Communication Technology applications and the citizens' acceptance to them in the context of smart cities. It motivates that smart cities are dependant on citizens being provided with services and information that is easily accessible, and that such applications offer increased opportunities in communication and information exchange. The study performed in [8] also shows that the citizens' intention to use such a city app is improved if its perceived usefulness or performance expectancy is high, and if the effort expectancy related to the difficulty of learning and adopting the technology is low. These were key factors to take into consideration when developing the MyCity application, which attemps to provide a simple and usable platform to offer citizens the previously mentioned functionalities that would improve their level of information and satisfaction regarding city services.

3.3. Similar applications

3.3.1. App descriptions

- **Mission Hills[9]**

This application is probably the most similar in functionality with the MyCity app for Cluj-Napoca. It is focused on the city of Mission Hills, Kansas and provides some common themes, such as the possibility of reporting issues by filling a form with details such as location, description, and a picture, but it also requires contact information such as a phone number and contact email in order to report issues.

It also offers descriptions and contact information regarding the city hall, its boards and committees, and members of its council. Apart from these, information about city services and utilities is present.

Similarly to the MyCity app, it allows the user to read news and view notifications, as well as fiind details, directions and brief descriptions about various parks and facilities in the city.

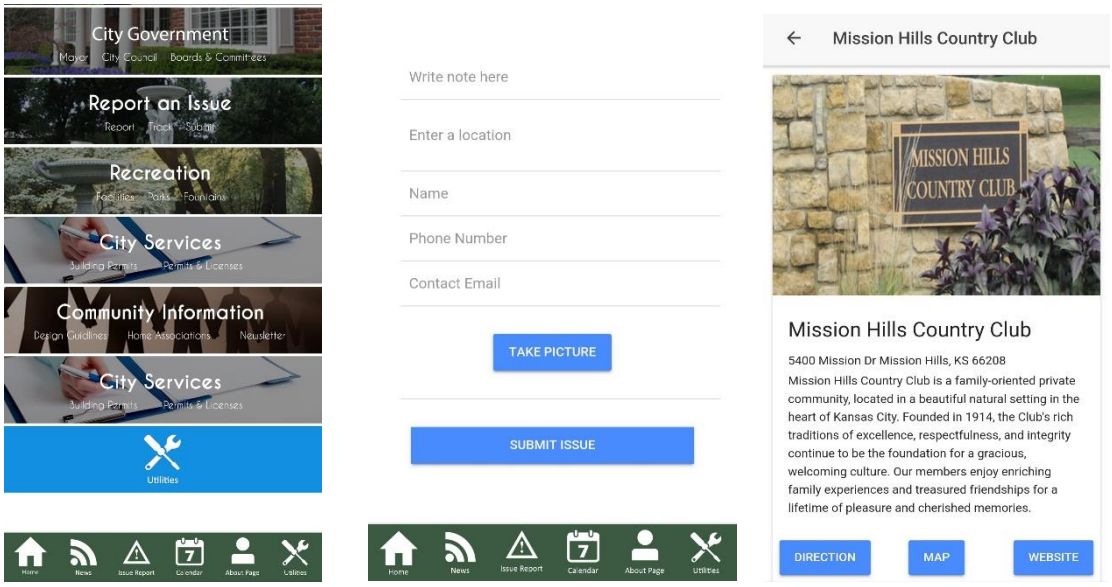


Figure 3.1. Mission Hills application

- **SeeClickFix[10]**

This application is designed for the city of Hoover, Alabama and also shares some common functionalities, such as issue reporting. It allows the user to select a location, then the type of issue they encountered, take a photo, and then fill out a form specifically designed for that issue type.

It also allows the users to view a map of with the issues reported by other poeple in the city, comment and share the issue. There is a way to see who the issue is assigned to and view its status. The application provides a way for users to receive notices, and also seems to have a section about events in the city.

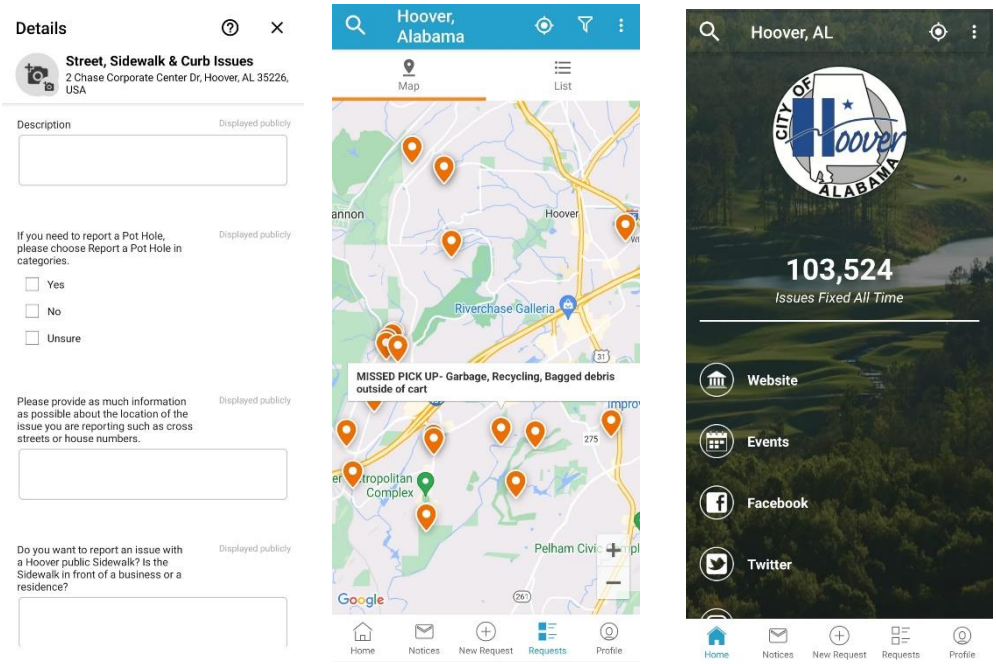


Figure 3.2. SeeClickFix application

- **My Cluj**[11]

This is an official application provided by Cluj-Napoca City Hall for this exact purpose of making complaints about the city infrastructure and facilities or reporting issues encountered in the city. It has the previously mentioned functionality, and since it is about Cluj-Napoca, it seems to already solve one of the goals this project is trying to achieve, but the main problem is that it is not availlable for download on newer versions of Android, so some users are forced to use the website version, which doesn't provide so much accessibility and flexibility.

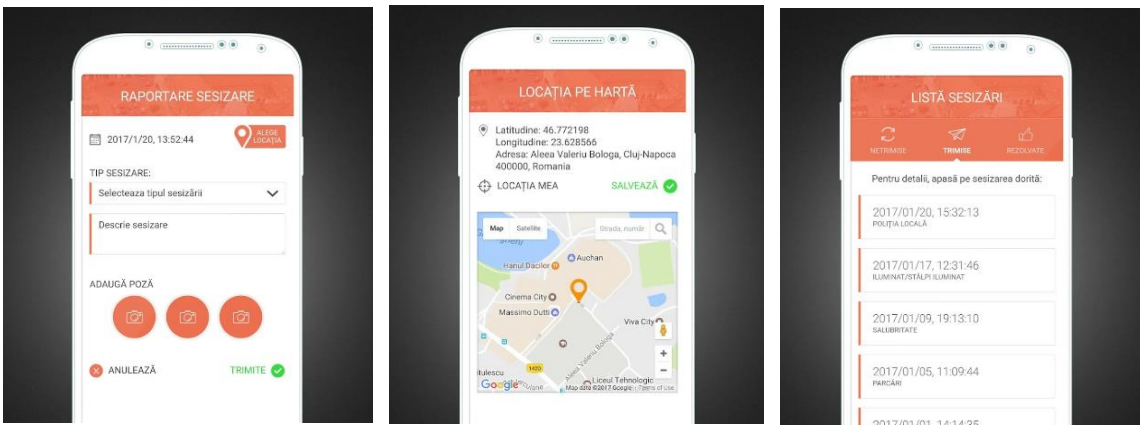


Figure 3.3. MyCluj application

3.3.2. Comparative analysis

This section aims to provide a visualization of how the features provided by the MyCity application for Cluj-Napoca compare to those existing in the previously mentioned applications. While some of these, like the MyCluj App, are focused on this single issue reporting functionality, our application aims to build on top of it and provide additional features that would be useful to people in the city, while simultaneously offering an interface for the institutions to view and handle them from the web application.

Table 3.1. Comparative analysis

Feature	Mission Hills	SeeClickFix	MyCluj	MyCity Cluj-Napoca
Report issue	Yes	Yes	Yes	Yes
Select issue type	Yes	Yes	Yes	Yes
Select location on map	Yes	Yes	Yes	Yes
Use current location	Yes	Yes	Yes	Yes
Take photo	Yes	Yes	Yes	Yes
Choose photo from gallery	Yes	Yes	Yes	Yes
View issue details and status	-	Yes		Yes
Chat section	-	-	-	Yes
View city map with all reported issues	-	Yes	Yes	-
View institution details and contact information	Yes	-	-	Yes
View facility details	Yes	-	-	Yes
Filter facilities	-	-	-	Yes

Read news	Yes	Yes	-	Yes
Receive alert notifications	Yes	Yes	Yes	Yes
Browse events	-	Yes	-	Yes
Filter events	-	-	-	Yes
Create events	-	-	-	Yes
Add events to favorites	-	-	-	Yes
Get directions	Yes	-	-	Yes
Access official websites	Yes	Yes	-	Yes
View statistics	-	-	-	Yes

Chapter 4. Analysis and Theoretical Foundation

This chapter will focus on the analysis process of developing MyCity by initially retouching on the problem statement and the user needs, then exploring the features that aim to satisfy these needs, and expanding on the features with the help of use case specifications and diagrams.

The second part of the analysis will include a conceptual architecture of the project, highlighting the main structural components, architectural decisions and patterns, and technologies used to achieve the desired structure and functionality.

4.1. Problem and product position

Table 4.1. Problem statement

The problem of	Maintenance of the infrastructure and facilities in Cluj-Napoca and difficult access to information caused by the lack of efficient interaction between inhabitants and public institutions
affects	Citizens in the community
the impact of which is	A decreased quality of life and population satisfaction
a successful solution would be	To allow issues that emerge to be reported and solved quickly To provide a more accessible way for people to know about important news and alerts regarding the city's functioning To allow citizens to be more informed about institutions, facilities and events that are happening Easy to use Easy to maintain

Table 4.2. Product position statement

For	Citizens and employees of public institutions in Cluj-Napoca
Who	Need a simpler way of interaction with regard to the events that occur in the city
MyCity	Is a mobile application that can be used by the citizens to improve their level of information, awareness and overall satisfaction
That	Allows users to quickly report issues that occur in a city, receive news and notifications from institutions, but also easily find events or facilities in the city
Unlike	Different websites or forms provided by institutions, which the user would have to search for
Our product	Allows for a quicker, closer and more convenient interaction process

4.2. User descriptions and needs

As mentioned in the introduction chapter, some users who have an interest in the behavior and features that this application has to offer have been identified. These are the citizens of Cluj-Napoca, who are interested in the state of the infrastructure, facilities and services they are provided, the institutions and institution employees, who are responsible for providing and managing these services, and the application administrator, whose interest is to maintain the proper functioning of the system and its entities.

Table 4.3. Summary of stakeholders

Name	Description	Responsibilities
Citizen	Person who lives in Cluj-Napoca	Responsible for using the application to report issues and see news, alerts, events, facilities or institution details
Institution	Public institution or service provider in a city	Responsible for maintaining the state of infrastructure, facilities and for providing public services
Institution Employee	Person who works for a public institution or service provider	Responsible for solving the issues reported by citizens, and providing them with news and updates regarding the functioning of their institution
Administrator	Person who has access to all the information encapsulated in the application	Responsible for managing the application so the information remains relevant, correct and up to date

Based on this stakeholder analysis, we can recognize the three types of users involved in the MyCity application for Cluj-Napoca: regular users (citizens), institution employees and the administrator/s. After gathering information about their interests for using the application and their responsibilities in doing so, some user needs can be identified.

In the context of this project, the most important need for the inhabitants of Cluj-Napoca is to be provided an interface for communication and information sharing in order to solution problems that they might encounter in their daily lives, problems that might impact their activities in a negative way. This need implies the necessity of an issue reporting feature, which would allow them to send the context and information related to that issue so that it can be handled properly. This also introduces the need to be able to view the state of resolution and maintain communication between the involved entities. This is not only satisfied by the possibility for them to interact by using the issue's chat section, but also through more direct approaches, by using the contact details associated with each institution.

Another need related to information and regarding the state of the city is the possibility to view news and alerts that are provided by institutions, so that they can keep track of important information about the institutions' actions, decisions and plans.

Besides the need to be constantly aware of official information that is shared, citizens also need to be informed about other aspects, such as the provided facilities that they can use, or the multitude of events that they can participate to or attend in the city.

Obviously, some information that is present on the app, such as the circumstances, details and contact information of the users who report issues, should be private and not accessible by everyone. Similarly, on the institutions' part, the tasks they are assigned and the in-app interactions with users should be protected, which introduces the need for data security by providing a log-in and authentication feature.

The previously mentioned need is also relevant for the administrator, who should be the only one able to access and perform operations on user accounts, approve events, and add facilities or institutions to the application. They also have the need to properly maintain the state and validity of the entities in order to provide a good experience.

Table 4.4. Summary of user needs

Need	Priority	Concerns	Current Solutions	Proposed Solution
Report, track and solve problems that occur in the city	0	Citizen, Institution Employee	Existing solutions like reporting to a general entity or looking for institution contact details	Platform that automatically assigns the issue to the related institution based on its type
View and provide news and alerts	0	Citizen, Institution Employee, Admin	Institutions using their own websites or social media platforms	Ability to provide alerts to citizens as notifications on their phone, have a centralized place for users to view news
View institution and facility details	1	Citizen, Admin	Existing solutions like city websites, google maps	Accessibility and usability improvements, ability to search and filter
View and add events that happen in the city	1	Citizen, Admin	Posts on social media applications, newspaper or news announcements	Accessibility and usability improvements, ability to search and filter
Security and data protection	0	Citizen, Institution Employee, Admin	Use of authentication, authorization and encryption	Use of authentication, authorization and encryption

4.3. Features

Based on the analysis of stakeholders, user types and each type's needs within the app, some main features can be identified, which highlight the main capabilities of the system and a basis for defining the use cases that can be performed when using the application. These features correspond to the goals defined in the project objectives chapter, and can be classified based on the user types mentioned previously.

4.3.1. Regular User features

- **Reporting issues**: The product enables citizens to easily report issues that they encounter in the city to the public institutions, so that they can be more quickly and efficiently solved. In order to report an issue, they need to fill out a form with details that help describe it, place it in a spatial and temporal context and help institutions understand it fully: the type, title, description, location and an image to provide visual information. At submission, the issue is automatically assigned to a public institution that has the authority and ability to solve it, taking the responsibility for correct assignment away from the user. Once they submit issues, users can easily view the details of their submission, the current status of the issue, and the name and status of all institutions involved in handling it. In case they have additional questions or institutions require more context information or details, the interaction can be performed in a dedicated chat section.

- **Viewing news**: This feature provides users a way to be up to date with things that occur in the city related to the actions of the public institutions. The news section allows users to preview news with details such as the title, image, date and institutions that posted them, and check out the full article, directly in the application, and also on the institution's main website, which can be easily accessed.

- **Receiving alerts**: In case some information that is of high priority or importance needs to reach the inhabitants of Cluj-Napoca faster, the alerts feature provides a way for this to happen by allowing users to receive notifications directly on their device, alerting them of something that has happened or is about to happen in a specified time frame. Sending direct notifications in case of such events assures that information spreads faster and to as many people as possible.

- **Viewing institution details**: This feature allows users to easily access the information they might be looking for regarding the public institutions and service providers in the city. In case they don't necessarily have an issue but want to get in contact with an institution, they can view its contact information, such as the email and phone number, get map directions to the institution's location, or access its official website.

- **Viewing facility details**: This feature is especially useful for people that are not living in Cluj-Napoca or are not entirely familiar with what the city has to offer. Even if this is not the case, anyone can benefit from the information provided: a list of parks, sports facilities, malls, libraries, cinemas etc., which can be filtered by type based on the user's needs. This allows them to discover information about the existing facilities or attractions, including their image, name, location, description, website, and event o get map directions to it.

- **Viewing events**: In case the user is looking for events that are happening or going to happen in the near future, this section allows them to easily discover some based on their preferences. They can filter them by type, view their location, time frame and description, and the number of people that are interested. If they are also interested in participating or want to save them for easier access, users can add them to favorites.

- **Creating events**: Users can not only browse through the events that the city has to offer, but they can also create their own events which will be displayed on the app. To do this, they need to complete a form with their event's details: the title, description, location, start and end dates and times, and upload a poster picture to help others visualize it. After submitting the event, it has to go through a process of approval performed by the app administrator, which makes sure that all events posted on the app are genuine and relevant. They can contact the users who submit them or delete them if necessary. Once they are approved by the admin, they are available for anyone to see.

4.3.2. Institution employee features

- **Solving issues**: Once users submit issues of specific types, institutions are assigned those issues so they can be aware of them and perform the necessary actions or allocate the necessary resources in order to handle them. Institutions employees can view all the details submitted by the users in a specific page, where they can also assign it to themselves, signaling to other employees that they are solving it. Once assigned, the employee can communicate with the user who submitted the issue for additional clarifications and updates. When finalized, they can mark it as solved. In the issues section, employees can filter all issues in order to more easily fiind the ones they have already assigned, or find ones that have been created and are not yet assigned by someone.

- **Reassigning issues**: This feature allows for issues that have been wrongly automatically assigned once submitted to reach the correct institution to handle them. In case an employee notices an issue that is not supposed to be solved by their institution, they can select another institution from a list and forward it to them. Generally, institutions should know based on the problem details provided by the user who they should reassign it to.

- **Sharing issues**: An institution might not have all the ncessary authorities, resources or capabilities to handle an issue by themselves, which introduces the need for a sharing feature. When reviewing the details submitted, employees can choose to share the issue to a number of other institutions which are registered on the application, in order for a collaboration that would reach a satisfactory issue resolution. Since multiple institutions need to play a part, the issue is only finalized when all involved institutions solve their own part of the problem.

- **Posting news**: Institutions likely already have a website where they post news or announcements about their involvement in the city, but this feature provides a way to also do it directly on the application, so users can have easier access to them directly from their phone. To add news to the app, they need to fill a form with the title, description and an image for visualization. Once posted, it is visible to all users on the app.

- **Sending alerts**: This feature allows institutions to send alerts directly to users, in case they need to be notified of any unexpected events that might affect them. They will be received as notifications, which the users can open from their notification bar to view.

- **Creating employee accounts**: Since the institution employee accounts shouldn't be available for anyone to create through a sign-up process, the application only allows institutions to create accounts for themselves in order to assign them to their employees. If there's an institution that's not currently using the application but wants to implement it in their system, they can send a request to the administrator which assigns them an initial account, from which they can create as many as they want to hand them to the employees.

- **Editing details**: This feature allows institutions to modify the contact details and information that is displayed to the regular users, in order to keep everything up to date in case of modifications.

- **Viewing analytics**: In the dashboard section of the institution accounts, there are multiple useful metrics which allow employees to see an overview of the activity on the app, such as the number of issues that were submitted to them, and their distribution based on status and type.

4.3.3. Administrator features

- **Manage data**: The application administrator can view, create, update and delete entities in order to maintain a correct state. They can perform operations on user accounts, institutions, facilties, submitted issues, events, news and alerts. They are also responsible for approving the events that are submitted by users.

- **Analytics**: Like in the case of institution accounts, the admin is provided a dashboard analytics feature, where information and charts about all aspects of the application are displayed.

4.4. Use cases

The use cases of the application are execution flows that users perform in order to complete tasks. They are related and usually correspond to features, so based on the previously described features we can identify the possible uses cases for each type of user in the application: regular users, institution employees and the administrator. Each type of user has multiple execution flows they can perform, based on their desired operation.

The first thing to do is identify the actors. In the case of this application, they correspond to the user types:

- **Citizen:** this is the regular user of the app, can use the features of the mobile application
- **Institution Employee:** these actors can use the features of the web application designed for institutions, once they have an account created
- **Administrator:** this actor also uses the web component to maintain the application

Each use case can have multiple different flows, which are called scenarios, based on the operations that the users perform, the choices they make, and the information they input. In most of the use cases of the application, there is one success scenario and an alternative failure scenario, in case the required steps are not performed correctly.

Besides the possible scenarios and the actors that initiate them, each use case has some preconditions and postconditions. Preconditions are facts that must be true in order for a use

case scenario to begin, while postconditions are facts that must be true upon the successful completion of the use case scenario.

In the next section, the use cases for the three types of actors will be mentioned and then highlighted using steps for the flows, and diagrams for better visualization.

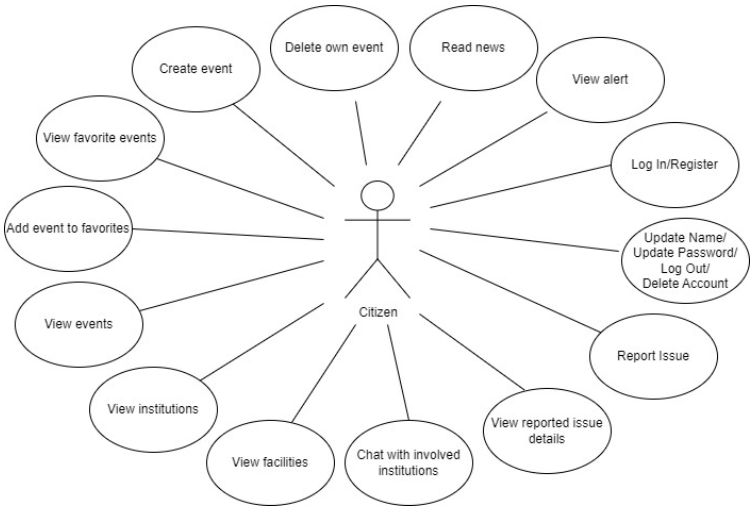


Figure 4.1. Citizen use cases

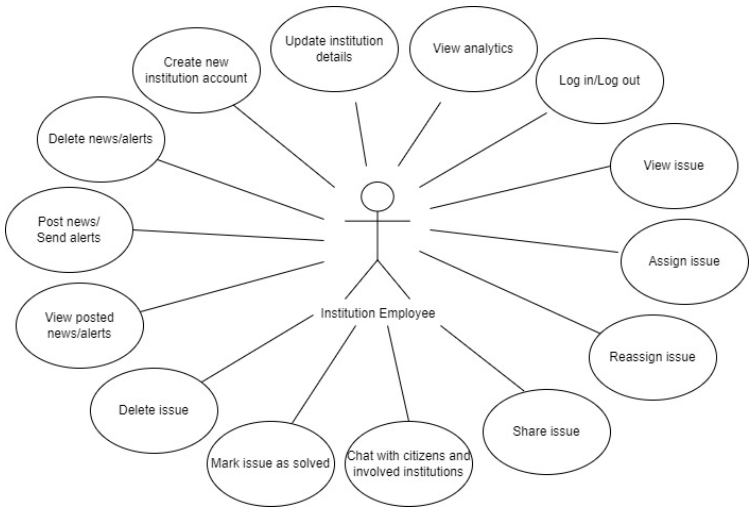


Figure 4.2. Institution employee use cases

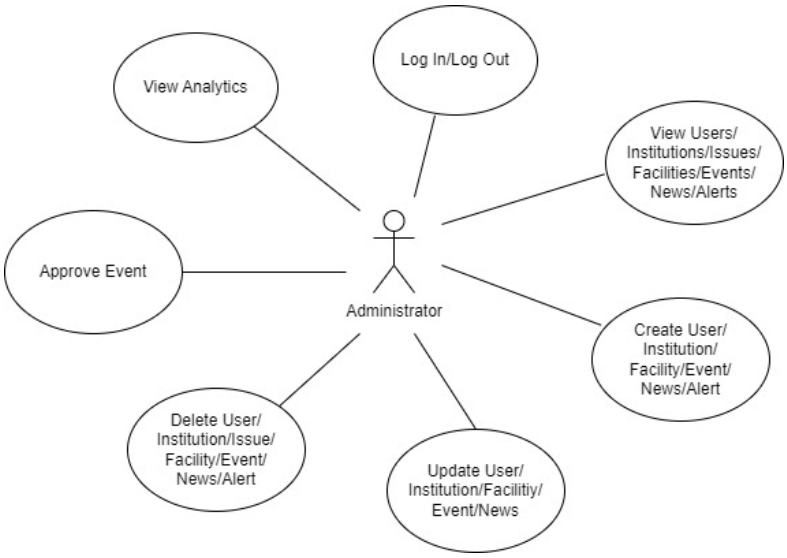


Figure 4.3. Administrator use cases

4.4.1. Use cases for all types of actors

Use case 1. Log in

1. User opens the application
2. System displays log in form
3. User enters their username, password
4. System validates authentication
5. System displays home page

Alternate flows:

5b. Authentication unsuccessful

1. System displays error message
2. Execution returns to step 3.

Preconditions: User is not logged in

Postconditions: User is logged in

Use case 2. Log out

1. User selects the log out
2. System displays log in page

Preconditions: User is logged in

Postconditions: User is not logged in

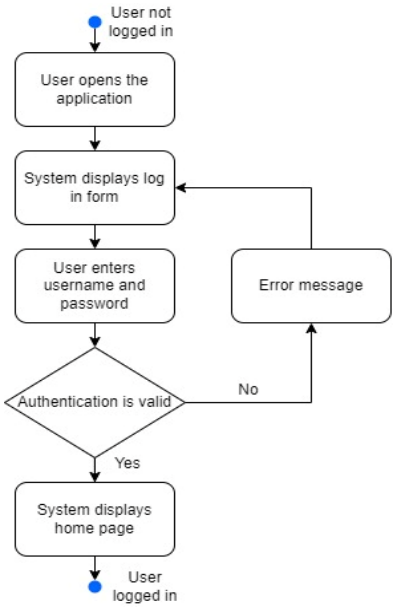


Figure 4.4. UC - Log In

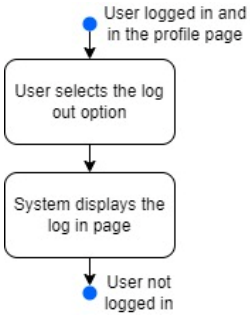


Figure 4.5. UC - Log Out

4.4.2. Use cases for citizen (regular user)

Use case 1. Register

1. User selects the register option
2. System displays a register form
3. User enters name, username, email, password and confirms password
4. System verifies the form is valid
5. System displays success message and log in possibility
6. User confirms message
7. System displays the log-in page with the username and password provided at register

Alertnate flows:

3a. Username already exists:

1. System displays an error message below the username field

3b. Email already exists or is not in the correct format:

1. System displays an error message below the email field

3c. Password is too short:

1. System displays an error message below the password field

3d. Passwords don't match:

1. System displays an error message below the confirm password field

4b. Invalid data:

1. System displays error message below the invalid fields
2. Execution returns to step 3.

Preconditions: User is not logged in an dis in the log in page

Postconditions: Account is registered and can be used for log in

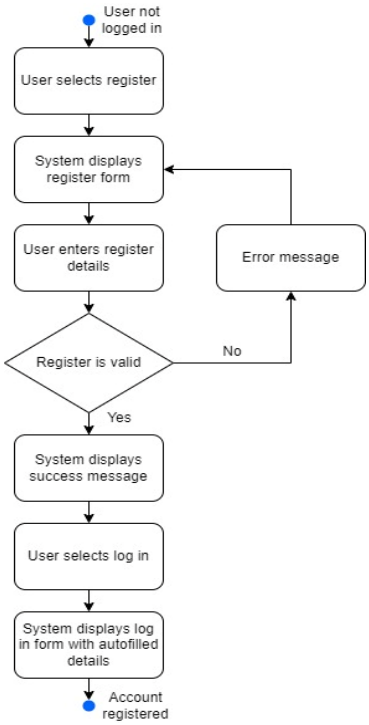


Figure 4.6. UC - Register

Use case 2. Report Issue

1. User selects report problem
2. System displays a form with fields for issue details
3. User fills the type, title, and description of the issue
4. User selects photo field
5. System asks whether user wants to take photo or choose photo
6. System opens camera
7. User takes photo and confirms
8. User clicks select location
9. System displays a map
10. User puts a marker on the map and confirms location
11. User submits form
12. System verifies the validity of data
13. System displays successful message

Aletrnate flows:

3a. Type not selected:

1. System displays an error message below the type field

3b. Title empty:

1. System displays an error message below the title field

3c. Description empty:

1. System displays an error message below the description field

5b. User selects choose photo:

1. System displays photo gallery
2. User chooses a photo

10b. User doesn't put a marker and doesn't have location on

1. System displays an error message below the location field

12b. Invalid data:

1. System displays error messages below the invalid fields
2. Execution returns to step 5.

Preconditions: User is logged in

Postconditions: Issue submitted, users can view its details or return to the issues page

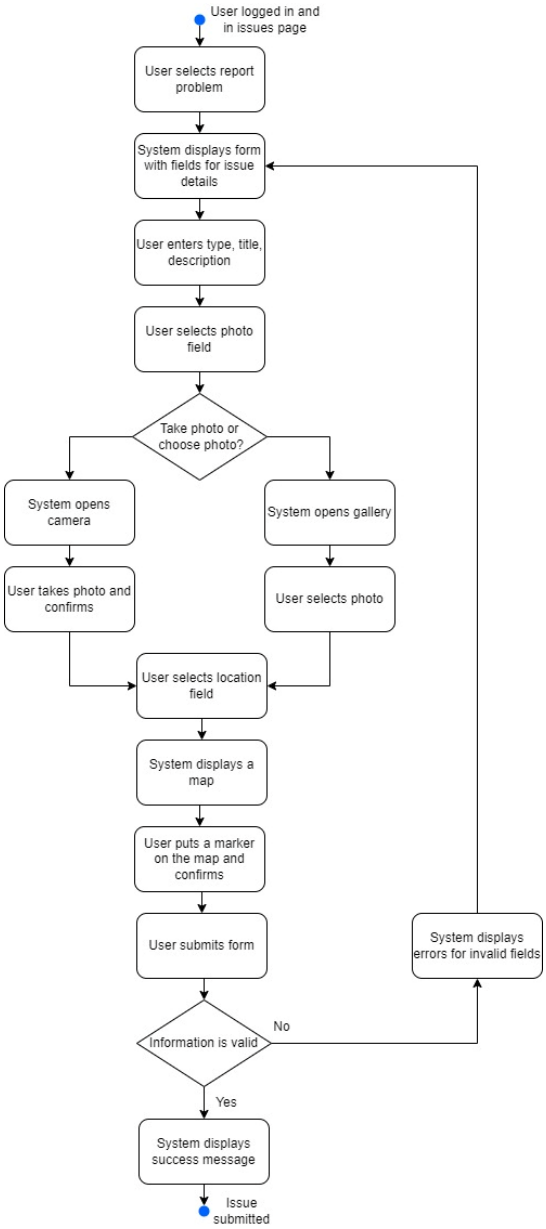


Figure 4.7. UC – Report an issue

Use case 3. View issue details and status and involved institutions

1. User selects one of the issues he reported
2. System displays issue details, status and involved institutions

Preconditions: User is logged in and in the issues page

Postconditions: None

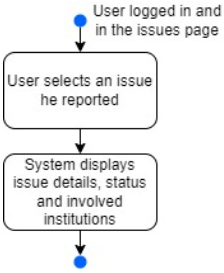


Figure 4.8. UC – View Issue Details

Use case 4. Chat with involved institutions

1. User selects the chat option from the options menu
2. System displays the chat section belonging to the issue
3. User types a message and sends it

Alternate flows:

1b. System displays a notification with a new message for an issue

1. User clicks the notification

Preconditions: User is logged in

Postconditions: Message is sent and visible by involved institutions

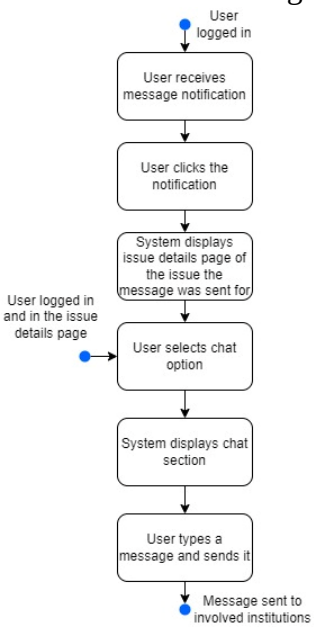


Figure 4.9. UC - Chat with institutions

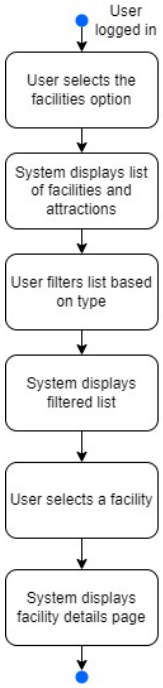


Figure 4.10. UC – View facility details

Use case 5. View facility details

1. User selects the facilities option from the home menu
2. System displays list of facilities in the city
3. User filters list based on type
4. System displays filtered list of facilities
5. User selects a facility from the list

6. System displays facility image, title, location, description

Preconditions: User is logged in

Postconditions: User can gen directions to the facility or access its website

Use case 6. View institution details

1. User selects the institutions option from the home menu
2. System displays list of institutions in the city
3. User selects an institution from the list
4. System display institution details: image, name, location, contact details

Preconditions: User is logged in

Postconditions: User can get directions to the institution or access its website

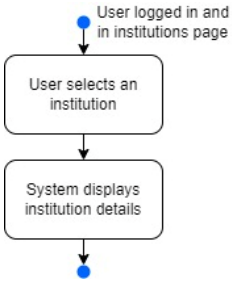


Figure 4.11. UC - View institution details

Use case 7. View event

1. User selects the events option from the home menu
2. System displays a list of events the user can choose from
3. User filters events based on type
4. System displays the filtered list of events
5. User selects an event he wants to see the details of
6. System displays event details: image, title, location, start date time and end date time, description

Preconditions: User is logged in

Postconditions: None

Use case 8. Add/remove event to/from favorites

1. User selects an event he wants to see the details of
2. System checks whether event is in favorites or not
3. System displays event details page with the appropriate meniu option
4. User selects add/remove to/from favorites
5. System displays confirmation message

Preconditions: User is logged in and in the events page

Postconditions: Event is added to favorites/ removed from favorites and the initial meniu option is toggled

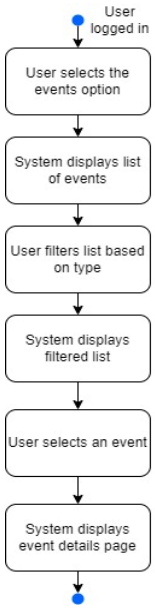


Figure 4.12. UC– View event details

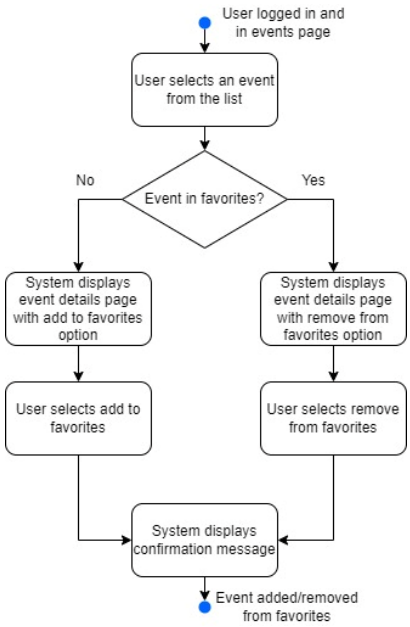


Figure 4.13. UC– Add/Remove from favorites

Use case 9. View favorite events

1. User selects My Events option from the options menu
2. System displays the list of events created by the user
3. User selects Favorites option from the options meniu
4. System displays list of events added to favorites

Preconditions: User is logged in an is in the My Events page

Postconditions: None

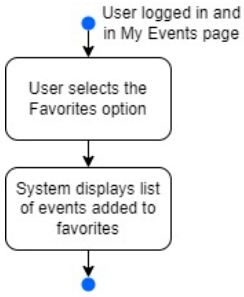


Figure 4.14. UC – View favorite events

Use case 10. Create event

1. User selects the create event option
2. System displays a form with details that need to be provided
3. User enters title, description, location, start date, start time, end date, end time
4. System selects upload photo
5. System asks whether to take photo or choose photo
6. User uploads a photo from gallery
7. User attempts to submit event
8. System validates the entered data
9. System displays a successful message

Alternate flows:

3a. User doesn’t input a title:

1. System displays an error message below the title field

3b. User doesn't input a location:

1. System displays an error message below the location field

3c. User doesn't input a description:

1. System displays an error message below the description field

3d. User doesn't input a start date:

1. System displays an error message below the start date field

3e. User doesn't input a start time:

1. System displays an error message below the start time field

3f. User also inputs an end date and time but they are before the start date time:

1. System displays an error message that end time cannot be before start time

4b. User takes photo instead of uploading from gallery

8b. The entered data is not valid:

1. System displays an error message
2. Execution returns to step 3.

Preconditions: User is logged in and in the My Events page

Postconditions: Event is created an dis visible for the admin to approve

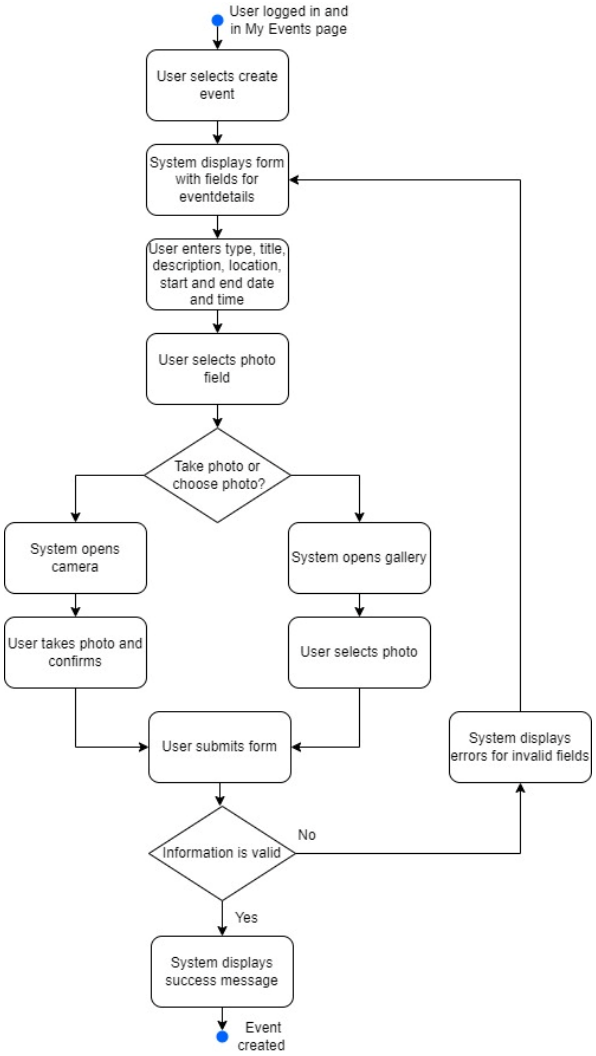


Figure 4.15. UC – Create event

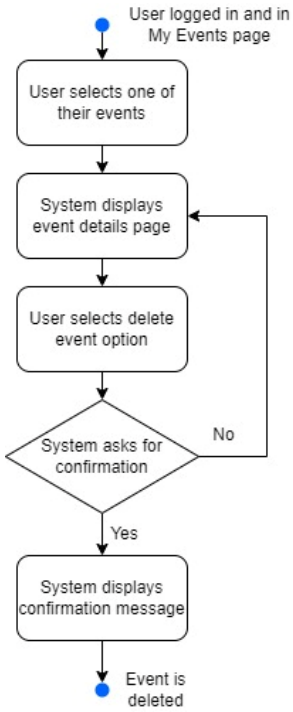


Figure 4.16. UC – Delete own event

Use case 11. Delete their event

1. User selects an event from their list of events

2. System displays page with event details
3. User selects the Delete option from the options menu
4. Systems asks for confirmation
5. System displays successful deletion message

Alernate flows:

4b. User cancells event deletion

1. Execution returns to step 2.

Preconditions: User is logged in and in the My Events section

Postconditions: Event is deleted and not available on the app anymore

Use case 12. Read news

1. User selects the news option from the home menu
2. System displays list with all nes posted by institutions
3. User selects news they want to read
4. System displays news page with the title, institution, date and the news article

Preconditions: User is logged in

Postconditions: User can also access the article on the official website of the institution

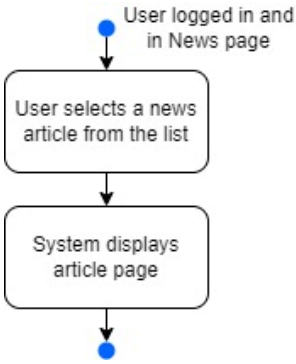


Figure 4.17. UC – Read news

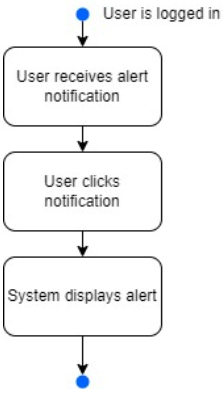


Figure 4.18. UC – View alert

Use case 13. View alert

1. User receives alert notification when the alert is sent
2. User clicks the notification
3. System displays list with alerts sent by institutions with the most recent one visible

Alternate flows:

2b. User selects the alerts option

Preconditions: User is logged in

Postconditions: None

Use case 14. Change name

1. User selects profile option
2. System displays profile section
3. User selects change name option
4. System displays form dialog with the user's name
5. User changes name and confirms operation
6. Systtem checks whether information is valid
7. System displays profile with updated name

Alternate flows:

5c. User cancells name change

1. Execution returns to step 2.

6b. Changed name is empty:

1. System displays error message below the name field

Preconditions: User is logged in and in the profile section

Postconditions: User's name is changed and modified version is displayed in profile

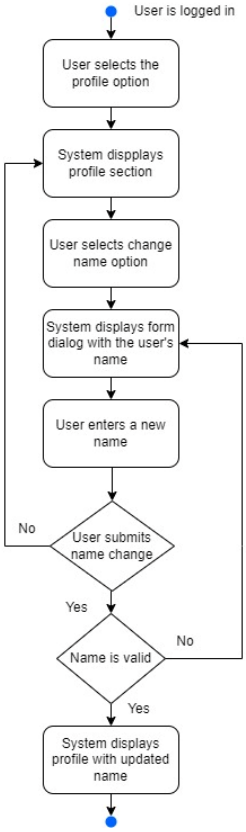


Figure 4.19. UC – Change name

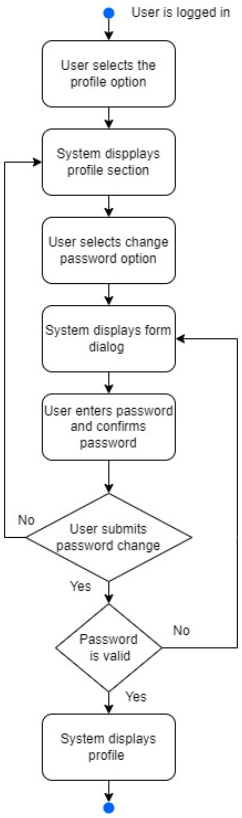


Figure 4.20. UC – Change password

Use case 15. Change password

1. User selects profile option
2. System displays profile section
3. User selects the change password option
4. System displays form dialog with change password fields
5. User enters new password and confirms password
6. System checks whether password is valid
7. System displays profile section

Alternate flows:

5b. User cancells password change

6b. New password is not long enough:

1. System displays error message below the password field
2. Execution returns to step 5

6c. Confirmed password doesn't match entered password

1. System displays error message below the confirm password field
2. Execution returns to step 5

Preconditions: User is logged in
Postconditions: User's password is updated

Use case 16. Delete account

1. User selects profile option
2. System displays profile section
3. User selects the delete account option
4. System asks for confirmation
5. Account is deleted and system displays the log in page

Alternate flows:

4b. User cancells delete account

1. Execution returns to step 2

Preconditions: User is logged in and in the profile section
Postconditions: Account is deleted

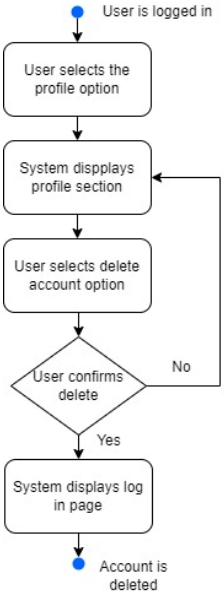


Figure 4.21. UC – Delete account

4.4.3. Use cases for institution employee

Use case 1. View issue

1. User selects the Issues option from the side meniu
2. System displays table with all issues submitted to institution
3. User filters issue by status or whether it was assigned by them or not
4. System displays filtered list in table
5. User selects the view option of an issue
6. System displays issue page with its details: details submitted by user, map location, chat section, share tab

Preconditions: Intitution employee is logged in
Postconditions: None

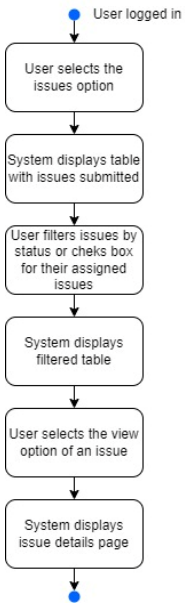


Figure 4.22. UC – View issue

Use case 2. Assign issue

1. User selects an issue from the issues table
2. System displays issue details
3. User selects assign option
4. System assigns user to solve the issue and enables chat section and solved option

Preconditions: User is logged in and in the Issues page, issue selecte dis not already assigned by other employees
Postconditions: Issue is displayed as assigned to other employees, institution employee name is displayed in the details, issue can be marked as solved ony by assigned employee

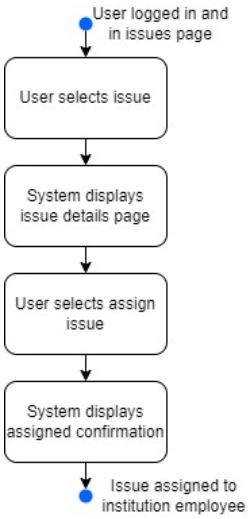


Figure 4.23. UC – Assign issue

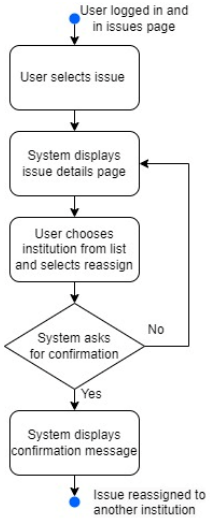


Figure 4.24. UC – Reassign issue

Use case 3. Reassign issue

1. User selects an issue from the issues table
2. System displays issue details

3. User chooses another institution and selects Reassign
4. System asks for confirmation
5. User reassigns issue
6. System displays successful message

Alternate flows:

4b. User cancells reassign:

1. System displays issue details page

Preconditions: User is logged in, issue is either not assigned by anyone or is assigned by the user who wants to reassign it

Postconditions: Issue is reassigned, visible on the selected institution's page and not visible anymore on the actor's institution's page

Use case 4. Share issue

1. User selects an issue from the issues table
2. System displays issue details
3. User chooses one or more institutions and selects Share
4. System asks for confirmation
5. System displays successful message

Alternate flows:

4b. User cancells share:

1. Execution returns to step 2.

Preconditions: User is logged in, issue is assigned by the user who wants to share it

Postconditions: Issue is shared, it is also visible on the selected institutions' page

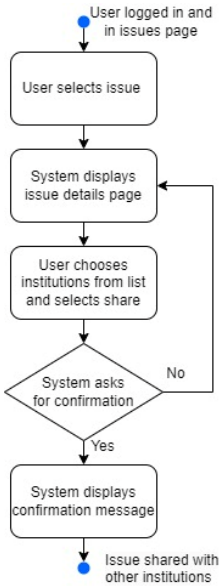


Figure 4.25. UC – Share issue

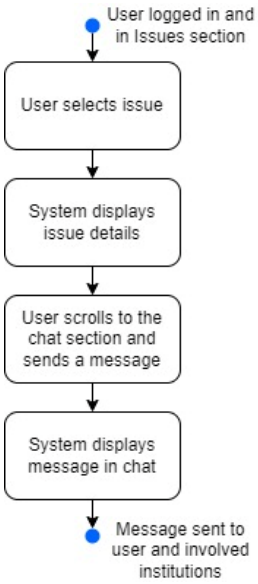


Figure 4.26. UC – Chat

Use case 5. Chat with citizen and involved institutions

1. User selects an issue from the issues table
2. System displays issue details page
3. User scrolls to the chat section and sends a messageSystem displays the message in the chat section

Preconditions: User is logged in and in the issues page

Postconditions: Message is sent and visible by citizen and involved institutions

Use case 6. Mark issue as solved

1. User selects an issue from the issues table
2. System displays issue details
3. User selects the Solved option
4. System displays confirmation and updates status

Preconditions: User is logged in, issue is assigned by the user who wants to mark it as solved

Postconditions: Issue is marked as solved for this current institution, overall issue status is marked as solved if only the employee's institution is involed. If more institutions are involved, issue is set as solved only when all involved institutions mark it as solved.

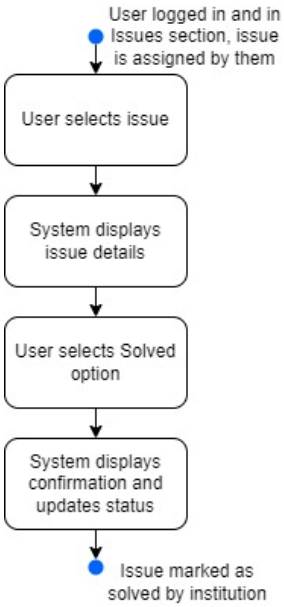


Figure 4.27. UC – Mark issue as solved

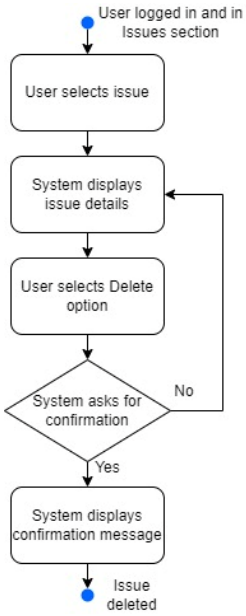


Figure 4.28. UC – Delete issue

Use case 7. Delete issue

1. User selects an issue from the issues table
2. System displays issue details
3. User selects the Delete option
4. System asks for confirmation
5. User confirms delete
6. System displays success message

Alternate flows:

1b. User initiates delete directly from the table

    1. Execution skips to step 4.

5b. User cancells delete

    1. Execution returns to step 2.

Preconditions: User is logged in

Postconditions: Issue is deleted from every user's account

Use case 8. View posted news or alerts

1. User selects the news or alerts option from the side menu
2. System displays news/alerts page
3. User selects view option of news/alert from table
4. System displays news/alert details

Preconditions: User is logged in
Postconditions: None

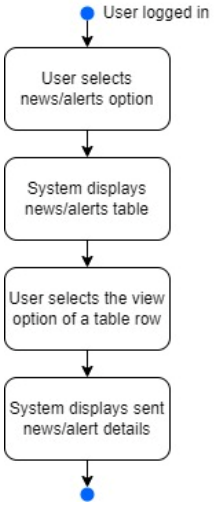


Figure 4.29. UC – View posted news or alerts

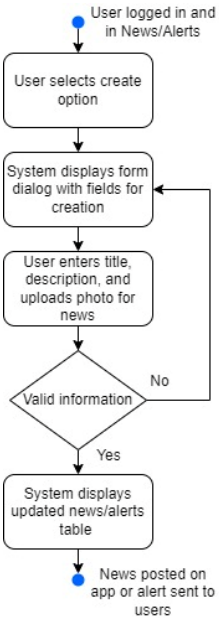


Figure 4.30. UC – Post news or send alerts

Use case 9. Post news or sends alerts

1. User selects the create option from news/alerts section
2. System displays form dialog
3. User completes form with title, description, and photo in case of news
4. System validates input data
5. System displays news/alerts table with added entry

Alternate flows:

3b. Title is empty:

1. System displays error message below the title field

3c. Description is empty:

1. System displays error message below the description field

5b. Invalid data:

1. System displays error messages below the invalid fields
2. Execution returns to step 5.

Preconditions: User is logged in
Postconditions: News/alerts posted by institution, they are visible to regular users. In case of alerts, regular users also receive a notification

Use case 10. Delete news/alerts

1. User selects the news or alerts option from the side meniu
2. System displays news/alerts page
3. User selects the delete option of a row in the table
4. System asks for confirmation
5. System displays updated table

Alternate flows:

4b. User cancells delete operation

1. Execution returns to step 2.

Preconditions: User logged in and in the news/alerts section
Postoconditions: News/alert is deleted from app

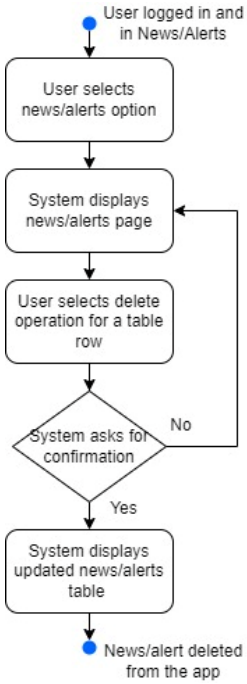


Figure 4.31. UC – Delete news or alerts

Use case 11. Create new institution account

1. User selects create account from the dashboard page
2. System displays form dialog
3. User enters name, username, email, password and confirms password
4. System validates entered data
5. System displays success message

Alternate flows:

3a. User enters invalid or already existing username:

1. System displays error message below the username field

3b. User enters invalid or already existing email:

1. System displays error message below the email field

3c. User enters password that is too short:

1. System displays error message below the password field

3d. Confirm password doesn't match password:

1. System displays error message below the confirm password field

4b. Provided data is invalid:

1. System displays error message below the invalid fields

Preconditions: User is logged in

Postconditions: New institution account is created and can be handed to new employe

Use case 12. Edit institution details

1. User selects the edit details from the dashboard section
2. System displays form dialog with institution data
3. User modifies address, contact information, website
4. System validates entered data
5. System displays successful message

Alternate flows:

3b. Modified field is invalid

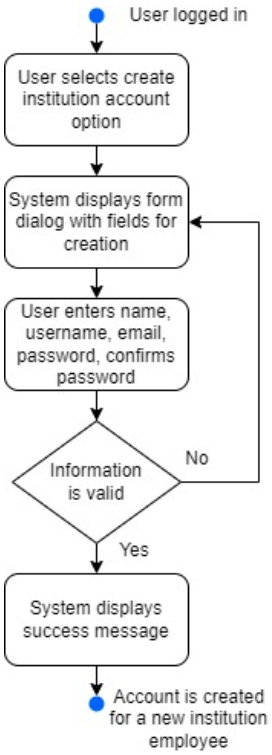
1. System displays error message below the invalid field

4b. Data provided is invalid on submit

1. System displays error message below the invalid fields

Preconditions: User is logged in

Postconditions: Institution details are modified and regular users can view updated details



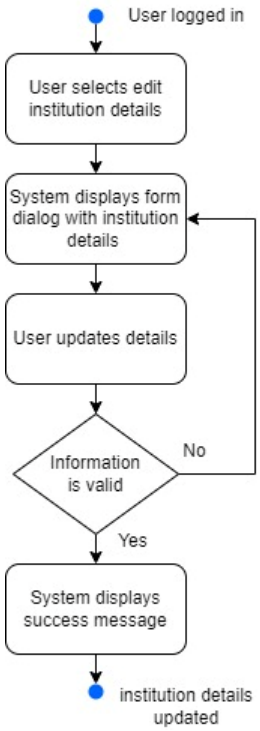


Figure 4.32. UC – Create employee account Figure 4.33. UC – Update institution details

Use case 13: View analytics

1. User selects the dashoard option from the side menu
2. System displays analytic data about the functioning of the institution

Preconditions: User is logged in

Postconditions: None

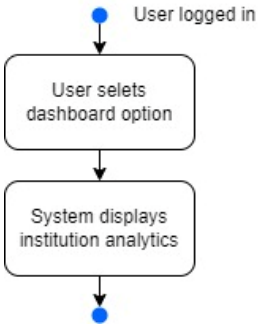


Figure 4.34. UC – View analytics

4.4.4. Use cases administrator

Use case 1. View user/institution/facility/event/issue/news/alert

1. User selects the corresponding entity option from the side menu
2. System displays page containing list/table of entities
3. User selects the the view option for one of the entities

4. System displays details of entity

Preconditions: User is logged in

Postconditions: None

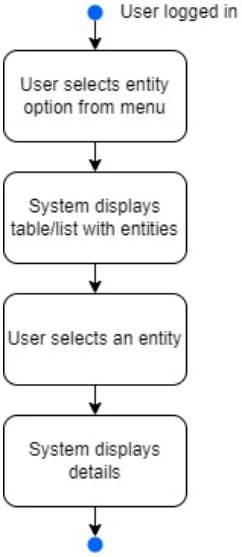


Figure 4.35. UC – View entity

Use case 2. Create user/institution/facility/event/news/alert

1. User selects the corresponding entity option from the side menu
2. System displays page containing list/table of entities
3. User selects the create option
4. System displays form with the necessary fields
5. User enters entity details for creation
6. User submits update operation
7. System validates entered data
8. System displays updated page with new entity added

Alternate flows:

4b. User enters invalid information

1. System displays error message under the invalid field

6b. User cancells update operation

1. Execution returns to step 2

7b. Submitted data is invalid

1. System displays error message under the invalid fields
2. Execution returns to step 4

Preconditions: User is logged in

Postconditions: New entity is created and visible

Use case 3. Update user/institution/facility/event/news

1. User selects the corresponding entity option from the side menu
2. System displays page containing list/table of entities
3. User selects the edit operation of the entity they want to modify
4. System displays form with the entity's details
5. User modifies the desired fields
6. User submits update operation
7. System validates modified data
8. System displays updated page with the modified entity

Alternate flows:

4b. User cancells update operation

2. Execution returns to step 2.

5b. Modified field is invalid

1. System displays error message below the invalid field

7b. Submitted data is invalid:

1. System displays error message below the invalid fields
2. Execution returns to step 5.

Preconditions: User is logged in

Postconditions: New entity is updated and visible

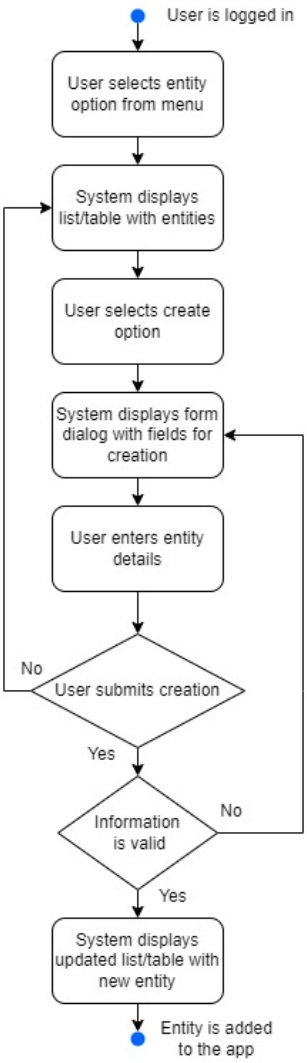


Figure 4.36. UC – Create entity

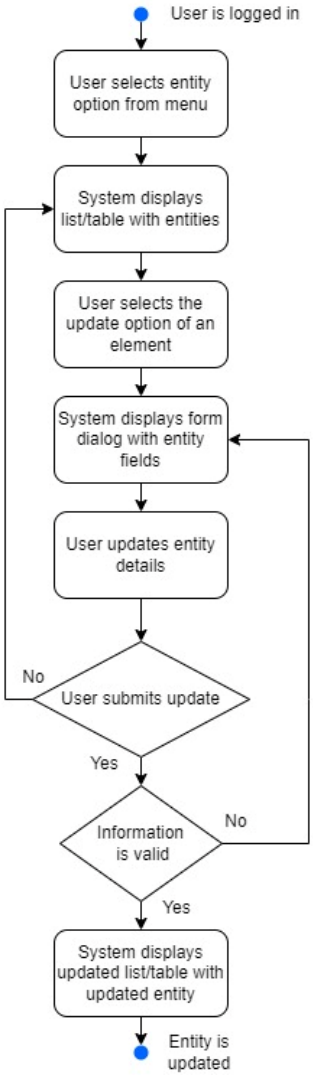


Figure 4.37. UC – Update entity

Use case 4. Delete user/institution.facility/event/issue/news/alert

1. User selects the corresponding entity option from the side menu
2. System displays page containing list of entities
3. User selects the delete operation of the entity they want to remove
4. Systems asks for confirmation
5. User confirms delete operation
6. System displays updated page with the deleted entity

Alternate flows:
5b. User cancells delete operation
1. Execution returns to step 2.

Preconditions: User is logged in
Postconditions: New entity is deleted

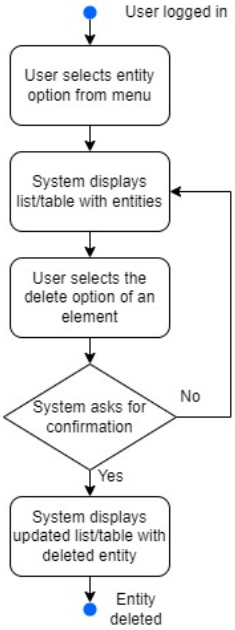


Figure 4.38. UC – Delete entity

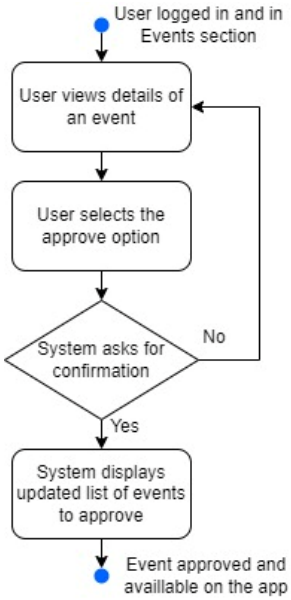


Figure 4.39. UC – Approve event

Use case 5. Approve user submitted event
1. User views details of desired event
2. User selects the approve option
3. System asks for confirmation
4. System displays updated page of created events

Alternate flows:
3b. User cancells approve operation
1. Execution returns to step 1

Preconditions: User is logged in and in the Events section
Postconditions: Event is approved, visible in the approved tab and to regular users of the app in the events section

Use case 6. View analytics
1. User selects the dashboard option from the side meniu
2. System displays statistics and charts related to the application's entities

Preconditions: User is logged in
Postconditions: None

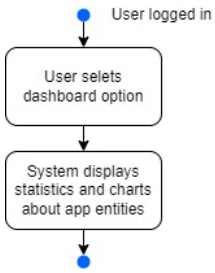


Figure 4.40. UC – View analytics

4.5. Conceptual Architecture

This subsection presents the architecture of the system from a conceptual point of view, highlighting the main components and the functions that each of them perform.

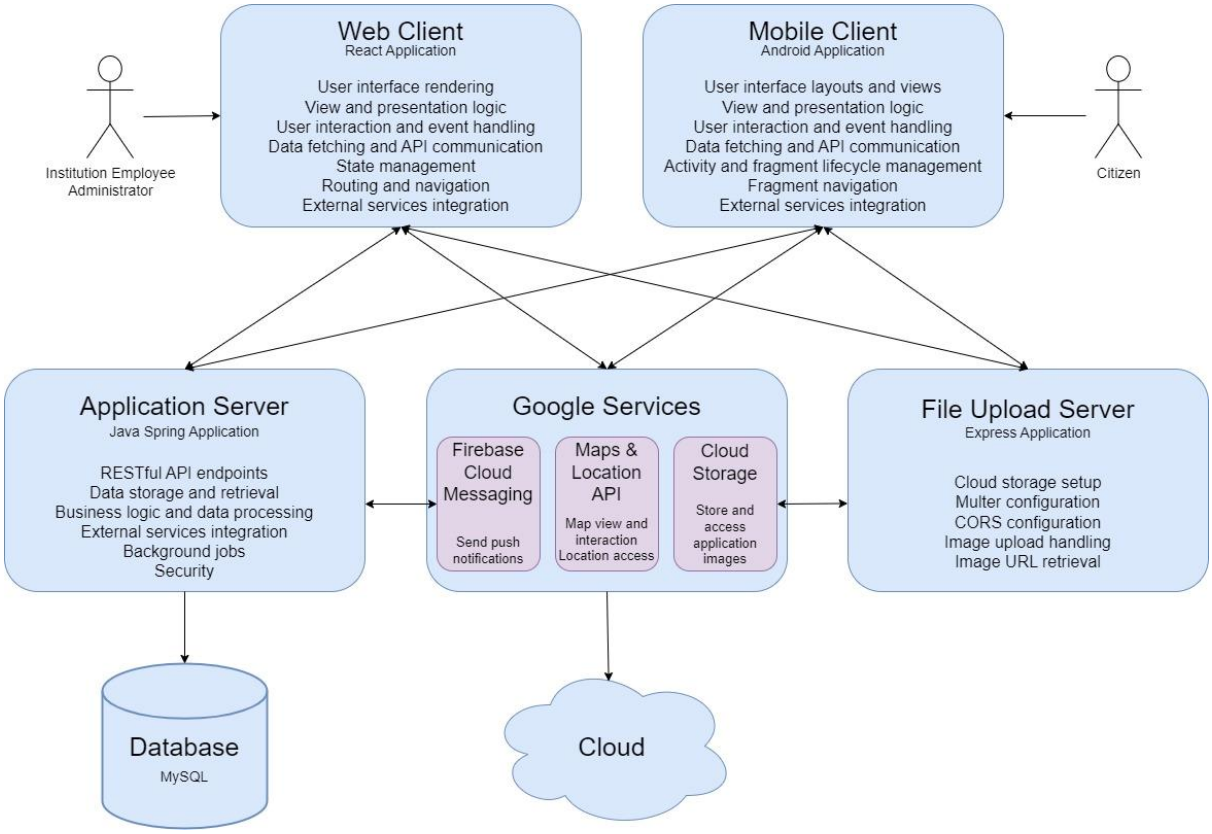


Figure 4.41. Conceptual Architecture

The application is based on a Client-Server architecture. This approach is suitable because it separates the client-side code responsible for rendering the user interface, implementing the presentation logic and handling user interaction, events and navigation, from the server-side code that deals with business logic and data storage, retrieval and processing.

By implementing such an architecture, it allows for higher maintainability and modularity, since the code in one part of the system will not be affected by changes in another part of the system. This approach is also useful because scaling and reusing can be done more efficiently, and it allows for implementation on different devices and platforms, communication between components being performed through protocols and serializtion of data.

In the context of MyCity, using the Client-Server architecture allows for a more convenient development based on the requirements for each type of user. The client application for the citizens of Cluj-Napoca can be implemented as a mobile platform for Android devices, while the requirements for institution employees and the administrator are more suitable for a web based implementation. Even in this scenario, both client types can efficiently use the application's server for data storage, retrieval, processing and for communication.

As it can be observed in figure 4.41., components are organized using the client-server structure, but there are multiple client types and servers to accomodate for the requirements of the project. Each client can communicate with each server in order to perform the necessary operations. There are two client applications, one designed for the regular users of the system, while the other for the administrator and institution employees. There are also two servers, the main one used for data operations and database interaction, while the other one is used for file uploading to cloud. Each component also interacts with different external services provided by Google, for easier implementation of various functionalities.

Communication between clients and servers is done through the HTTP protocol, which is based on sending text-based request-response type messages by using TCP/IP connections. It is a suitable approach because of its universality, simple to understand and use nature, and the supported request methods availlable.

The next section briefly details each component of the system, its function in the context of the problem, and its relation to the others.

Application Server:

This is the application's main server, as it is used for most operations that can be done by users, which are mostly related to the application's data: Create, Read, Update, Delete operations on users, institutions, events, facilities, issues, news and alerts, database storage and retrieval, business logic etc.

It uses Java and Spring Framework, which provide multiple benefits, such as Object Oriented Programming, suitable for the application's data structures, and Dependency Injection for maintaining a high coherence, low coupling and high maintainability and reusability of code.

File Upload Server:

This component is used for simpler functionality regarding image uploading to the application, as it provides a gateway between clients and the cloud storage service provided by Google.

As for technologies, it uses Express, which is a framework made for Node.js that provides speed, efficiency, and a lightweight and minimal structure which is appropriate for this operation.

Google services:

These are external services and APIs provided by Google, allowing for useful functionalities and features that are already implemented and can be integrated into the project.

- **Firebase Cloud Messaging**: this service is used for real-time delivery of push notifications to the application's regular users when an alert is sent by institutions or the user receives a new message related to one of the issues they submitted. It allows for simple integration and message sending with Spring, targetting the sent messages to topics which the client subscribes to when opening the application.
- **Cloud Storage**: this service is used for storing and accessing the application's images in the cloud, allowing for fast access via URL and more convenient mapping to the entities by storing only the URL's in the application database.
- **Maps, Location and Geocoding APIs:** the Maps API provides integration of Google Maps into the application for a rich map visualization and interaction features, such as moving, zooming, placing markers and accessing coordinates, which are useful for citizens when they are reporting issues and selecting the exact place where the issue is located. It also provides better visualization for

institution employees when handling the reported issues. Location API provides the access and display of the user's exact location if they provide location permissions and their location is on, while Geocoding API allows the coordinates of the marker placed by the user or of their current location to be transformed into a readable address line, which is also useful for institution employees.

Web Client:

This component represents the interface for visualization and interaction of the institution employees and the administrator with the application. It provides a simple nagivation and performing of operations, event handling and validation, allowing institution emmployees to easily filter, view issue details and collaborate with other institutions, as well as to post news, send alerts and edit their details. Similary, the administrator has quick access to management operations on all aspects of the application.

As for used technologies, the client is implemented in React, which is a JavaScript framework that offers component-based implementation and a virtual DOM (Documents Object Model) for rendering data. The user interface is created using Material UI, which is a component library for React designed by Google. Communication with the server is made using Axios.

Mobile Client:

The mobile client is developed for Cluj-Napoca's inhabitants in order for them to access the features that the MyCity application provides. It is made for Android devices using Android 8 or higher and offers a simple and usable interface for reporting issues and contacting institutions, reading news, receiving alerts and finding out more about the city's facilities and events.

The application is created using Google's Material Design and the Kotlin programming language. It provides easy integration of the mentioned external services and REST API communication through the Retrofit library.

Database:

The application's data is stored in a relational MySQL database used by the server with Hibernate, Entity Framework and JDBC.

Chapter 5. Detailed Design and Implementation

This chapter will contain a more detailed insight into the design and implementation process of the MyCity application. We will begin by taking a closer look at the architecture of the system, then analyze each component separately from an implementation point of view.

Each subsection corresponding to a component will focus on the structure and details about relevant classes and methods, and package/class diagrams.

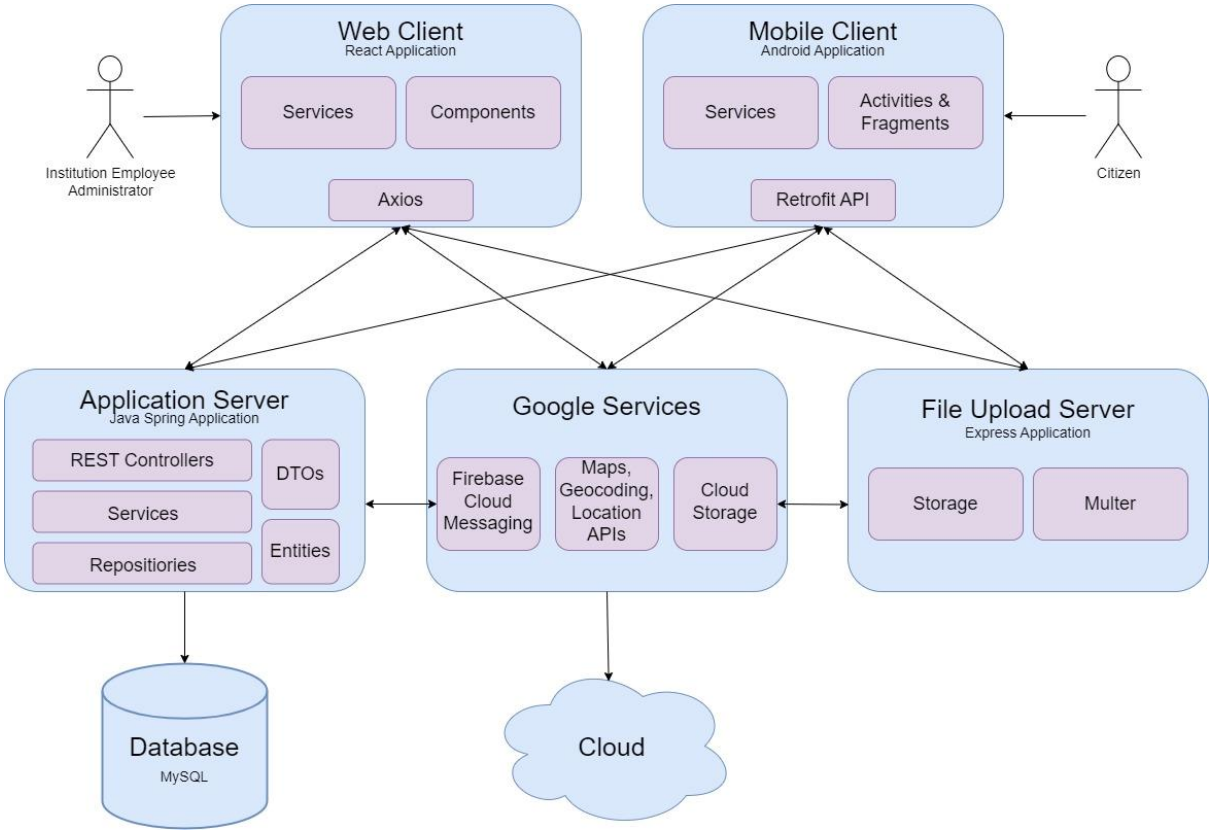


Figure 5.1. System Architecture

Figure 5.1 shows the previously discussed conceptual architecture of the system, highlighting the main subcomponents of each part of the project, such as controllers, services, repositories, entitues and DTOs for the application server. The clients also have the technology used for server communication displayed.

The arrows represent communication between components, which is mostly done between clients and servers through the HTTP protocol, but the application server can also send data to the web client through Web Sockets, in order to provide real-time display of chat messages. For message sending from the server to the mobile client, the Firebase Cloud Messaging is used.

5.1. Design and implementation of the application's server

The server provides:

- Data storage and retrieval to and from the MySQL database using JDBC, Hibernate and Entity framework.
- Data processing for Create, Read, Update, Delete operations, filtering and business logic functionality.
- Integration for external services, such as the Firebase Could Messaging, used for sending push notifications to the application's mobile users when certain events occur.
- RESTful API endpoints for communication with clients when HTTP requests that must be fulfilled by the server are sent.
- Background jobs, such as the one for automatically updating event statuses to „Finished" once once their end date passes.
- Security, authentication and encryption by using Spring Security with request filtering, JWT tokens, web configuration and password encryption.

The impementation of the server follows a layered architecture, where each layer only communicates with the one below it in order to provide a separation of concerns.

These are:

- Controller layer: These classes define the methods that are mapped to REST endpoints, so that requests to the server can be made using HTTP. Each controller contains an instance of the corresponding server, for example the InstitutionController calls methods of the InstitutionService and returns new response entities to the clients.
- Service layer: The services are responsible for business logic and data processing. Each service contains an instance of the corresponding repository, and other repositories if needed. For example, the InstitutionService uses the InstitutionRepository to access database operations.
- Repository Layer: The repositories perform storage and retreival of the entities from the database. Each repository is implemented as an interface extending the JpaRepository (Java Persistance API), used for its object relational mapping, simplified data acess and querying.

The application contains packages for each of the main entities: User, Institution, Issue, News, Alert, Facility, Event, and also packages for chat messages, security, and analytics displayed to the administrator and institution users.

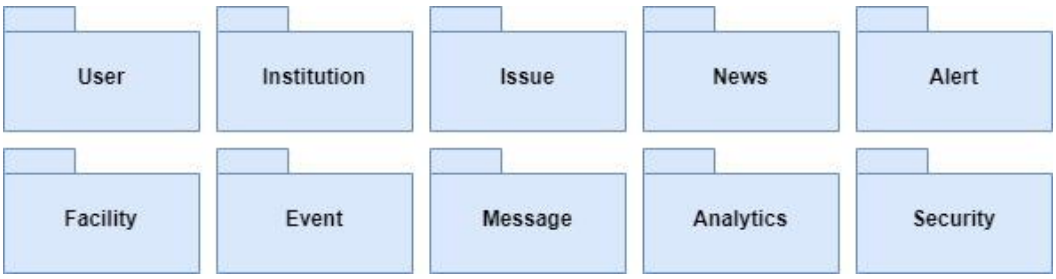


Figure 5.2. Application Server packages

Each package follows the previously mentioned layered architecture. For example, the implementation of the Issue package contains classes reporesenting each of the three layers: IssueController, IssueService and IssueRepository. Besides these, there is the entity Issue class that contains the fields specific to an issue and that is stored in the database using Entity framework. Each issue has a type and a status, both represented as enums in the diagram. The package also contains classes for holding statistics used for analytics, and the IssueBuilder used to map between Issue and IssueDTO/ReportedIssueDTO.

Figure 5.3. Class diagram of the Issue package

As for important methods from key classes, each could be considered equally important, as they take part in one or more app functionalities. Most of them are straighforward, as they call the corresponding method in the layer below.

Some more relevant methods are:

- **public IssueDTO create(ReportedIssueDTO reportedIssueDTO):** This method in the IssueService receives as parameter from the controller a DTO containing the issue details submitted by the mobile client, maps it to an issue Entity and saves it in the database. It also creates an institution involvement by using the IssueTypeAssignment class' static method that maps the issue type to the institution which is responsible for solving it, making it visible in the institution employee's web page. The method returns the saved issue by mapping it back to a DTO which will be sent in the response of the create request, so that the client can view its details in the mobile app.
- **@Scheduled(fixedRate=300000)**
  public void setFinishedEvents(): This is a background job in the EventService class that executes every 5 minutes and automatically updates the status of events to „Finished" if their end date is after the current date, allowing users to only be recommended upcoming events.
- **public Analytics getAdminAnalyticsData():** AnalyticsService method that Uses the repositories to count the number of entities of each type and get an issue and event distribution based on type and status in order to display the app statistics and charts in the admin's dashboard section.
- **private void sendNotification(AlertDTO alertDTO):** This method creates a Notification format and gets the FirebaseMessaging instance configured in the app in order to send a push notification containing the alert details to the users' phones, which they can click to view. The FirebaseMessaging instance is also used in the IssueService in order to send a notification when users receive a message regarding one of their submitted issues from one of the involved institutions.
- **@CrossOrigin**
  @PostMapping(SEND)
  public ResponseEntity<Void> sendMessage(@RequestBody MessageDTO messageDTO): This method in the MessageController uses a SimpMessagingTemplate to send messages to institution employees through a WebSocket connection, which allows the messages sent by citizens or other involved institutions to be seen in real-time in the issue details page. WebSocket messages are sent using topics through the STOMP protocol.
- **@PostMapping(LOG_IN)**
  public ResponseEntity<JwtResponse> authenticateUser(@Valid @RequestBody LoginRequest loginRequest): This method in the AuthenticationController is used to authenticate users by using Spring Security when they attempt to log in. It uses an Authentication object to validate the credentials, stores it in a SecurityContextHolder, and generates a JSON Web Token using JwtUtils. The token is part of the JwtResponse so that it can be included in the Authorization header when clients send requests that require the user to be logged into the application in order to perform.

5.2. Database structure

The application uses a MySQL relational database, which allows for structuring data based on the relationships between the project's entities. It makes use of the Java Persistance provided by Spring in order to automatically map objects adnotated with @Entity to database tables, and to establish their connection and create foreign keys by using @OneToMany, @ManyToOne or @ManyToMany. Database columns are creating by using @Column on an entity's field, and the primary key is selected with the @Id adnotation.

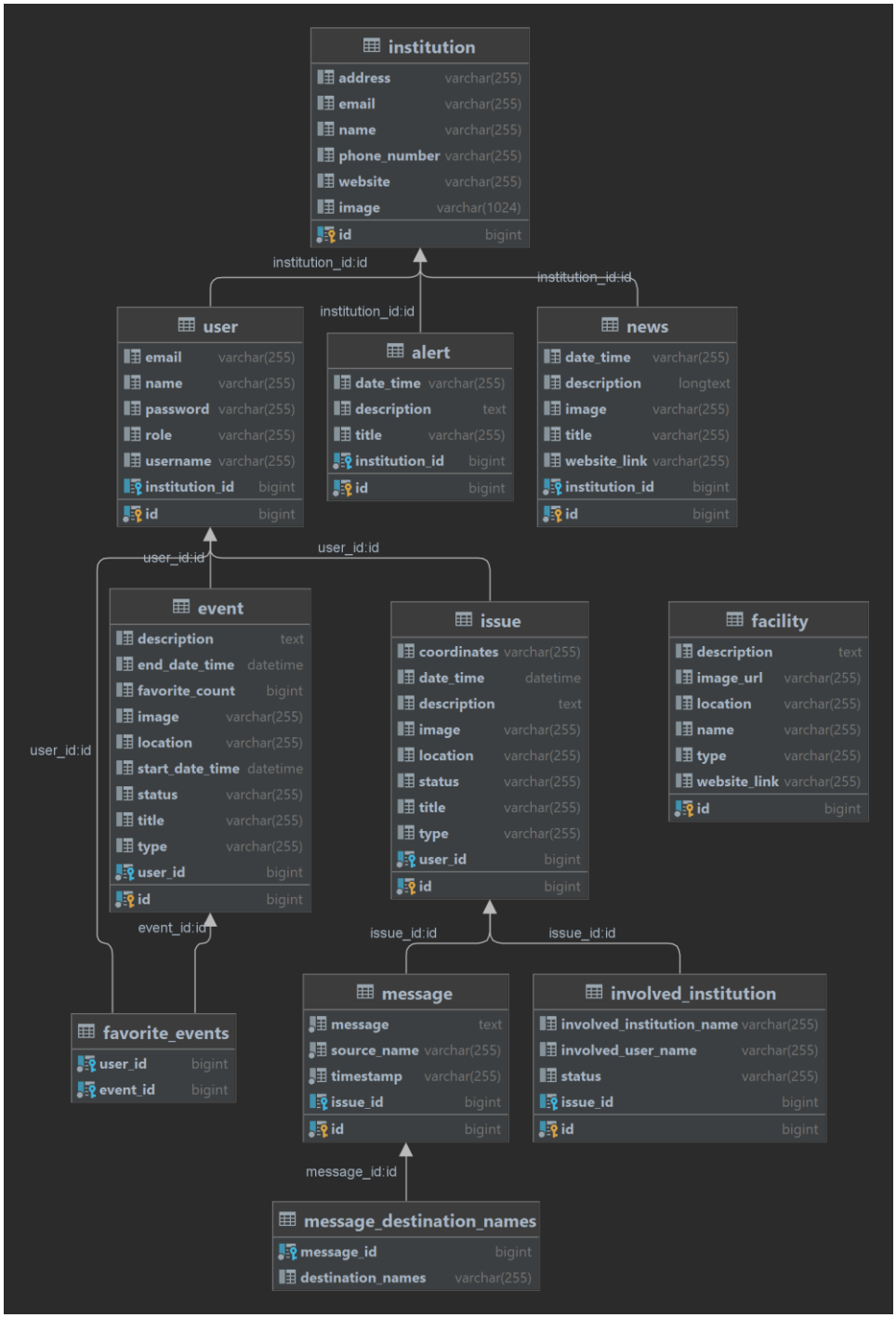


Figure 5.4. Database diagram

5.3. Design and implementation of the server for file uploading

This server performs:

- Setup of the Google Cloud storage bucket, multer and cors configuration
- Handling of requests made by clients that want to upload images to the application when reporting issues, creating events, or in case of the administrator, when performing operations on institutions, facilities, events, news.
- Retrieval of URL's for accessing the images from the cloud storage once they are successfuly uploaded.

The file upload server consists of a single JavaScript file app.js in a Node.JS Express project, which is configured to listen on port 5000 for requests from clients.

Regarding the implementation, initially, a multer is created, which is a Node.JS middleware that is used to handle the complex parsing of multipart/form-data for file uploading.

After this, a Storage instance is initialized by providing the JSON key file corresponding to a Google service account, and the ID of the cloud project where the project's bucket for image storage is located. A bucket instance is also created from the Storage object by providing the bucket name.

The server listens for requests, and when a post request is sent to the /upload endpoint, it checks whether the request body contains a file to upload, and in case it doesn't, it throws an error.

If a file is present in the request, the application creates a Blob in the bucket and a write stream in order to upload the image to storage. When the stream finishes and the upload is performed, a URL can be created in order to be able to directly access the image from the bucket. In case the upload is successful, this URL is sent back to the client who made the request, so it can be stored in the application's database.

5.4. Design and implementation of the web user interface

The web client handles:

- User interface rendering for the institution employees and the administrator of the application by using ReactJS
- View and presentation logic
- User interaction and event handling, providing the users with the functionalities described in chapter 4.
- Data fetching and API communication by using the Axios framework to send HTTP requests to the servers
- State management by using React Hooks
- Routing and navigation by defining a router and navigation paths for all pages
- Integration of external services, mainly the Google Maps API for displaying issue location to users
- User input validation, error handling and display

5.4.1. Routing and navigation

By using a Router, React allows developing websites in the form of a single page application, which allows the dynamic update of components in the view instead of loading entire pages every time navigation between views is made.

This is done by having an App component that includes all the separate views as components, and each component's underlying HTML and CSS is loaded at runtime.

Inside the App component, the React Router is defined, containing a list of routes corresponding to the pages in the application that users can nagivate to. Each route contains a path and an associated component that is rendered when the user accesses that path. It allows organizing the app in a hierarchical manner based on the defined paths.

In the context of the MyCity application, there are two types of users that can access the web client, each having different functionalities, so separate pages need to be created for them, corresponding to the "/admin" and "/institution" paths. When initially loading the application, the Log In component, associated with the base path "/", is rendered. When logging into the application, based on the role of the user that is authenticated, Admin or Institution Employee, the respective home page is displayed.

From there, each type of user can navigate to the desired section of the application by using the sidebar menu. Each element in the menu has an associated Route defined in the App Router, which renders the appropriate component when a menu item is selected.

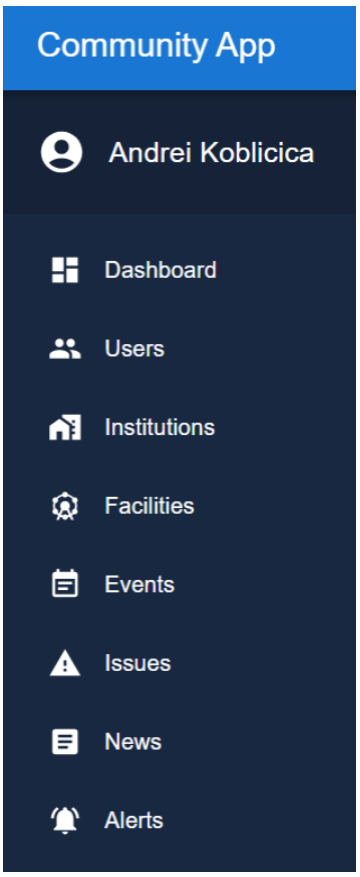


Figure 5.5. Administrator Menu

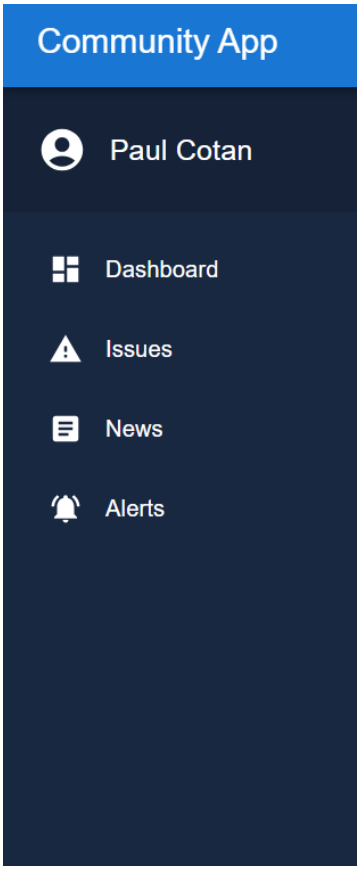


Figure 5.6. Institution Employee Menu

For example, if the Admin selects the Users option, navigation is performed by using the useNavigate React hook to the “admin/users” route defined, and the Users component is loaded. Figure 5.7 illustrates the navigation paths of the application.

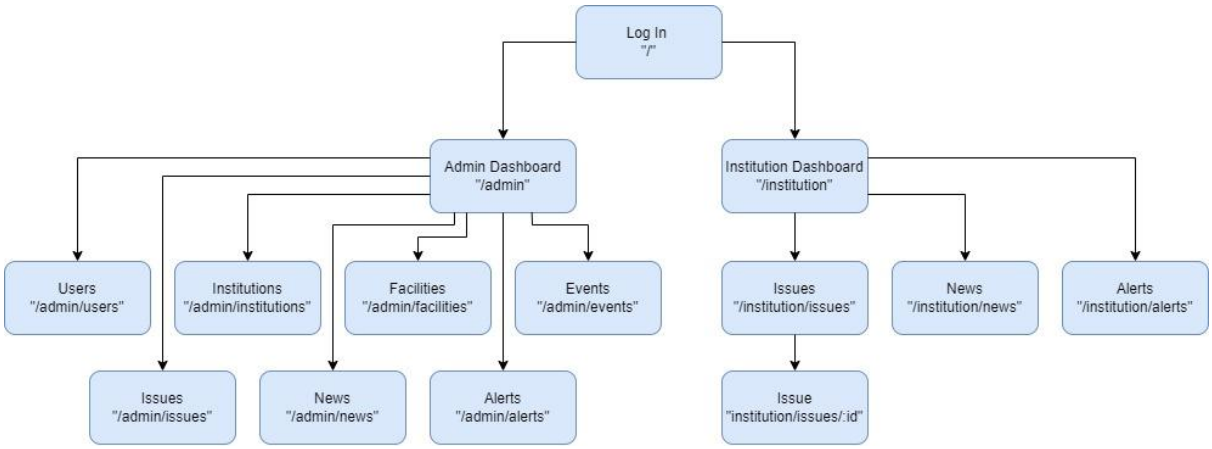


Figure 5.7. Web interface routes

Besides these routes, the top Appbar component is rendered for logged in users and the respective Sidebar component for logged in users based on their role.

5.4.2. Components

The application provides components for each route, plus the Appbar, and two Sidebar components. There is also a Map component used for displaying the issue location in the Issue details page and two card components, EventCard and FacilityCard, used for displaying event and facility details in the admin page.

Each of them is developed as a functional component, meaning they are represented as functions, which can receive props as parameters, can contain react hooks and other functions, and which return React elements in a HTML-like structure that define how the user interface looks. Each component is placed in a different JavaScript file, exported and can be used in different parts of the app.

Functional components are stateless, so in order to add state, the useState hook is used, which is an array containing two values, the current state and a function used to update it. For example, in the Issue page that displays the details of a reported issue to institution employees, it is used to hold the messages in the chat section: const [messages,setMessages]=useState([])

The [] represents the initial state, when loading the page it is initialized by using the setMessages function with the messages belonging to that issue, and each element from the messages list is mapped to a React element in the UI. When the institution employee sends a new message, it is added to the list using the setMessages function, which causes the re-render of the view elements the list is mapped to, displaying the new message.

The page is initialized upon access using the useEffect hook with no dependencies, allowing the data that needs to be displayed, in this case the issue that the user selects in order to view its details, to be requested from the server via the services and assigned to a state variable.

A component also contains multiple functions, such as for handling button click events. These are implemented as lambda functions and can be passed as parameter to the button's onClick prop.

The UI is defined in the return section of a component, in a hierarchy of Material UI elements, such as <Box> which is equilavent with a HTML , <DataGrid> in order to display the tables for admin and institution CRUD operations, <TextField> for user text input, <Button> for UI interaction, <Dialog> for pop up windows in which users can view, create and update details of app entities, <Typography> for displaying text, <Card> for displaying cards etc.

The map is displayed by loading a script with the google maps api key when the application is opened, and then using the <GoogleMap> component from React Google Maps API, and passing props such as the container style, the center coordinates, the zoom level and the marker coordinates.

The dashboard charts for displaying issue and event distribution by status and type are created using the React ChartJS 2 library, which has useful components such as Bar or Doughnut for the different types of charts displayed. The components need the labels and datasets passed as props, which are accessed by performing GET requests to the server to retrieve analytics data.

Web sockets are implemented and initialized with the appropriate URL and topics by using the SockJsClient component of the React Stomp library, which also provides useful handler functions such as onConnect, onDisconnect, onMessageReceived.

5.4.3. Application services

Services are used for performing HTTP requests to the servers in order to retrieve, add or modify data. There is a service for handling operations on each main entity of the application: users, issues, institutions, facilties, events, news, and alerts. Besides these, there is a service for authentication, file uploading, analytics and one for setting up HTTP.

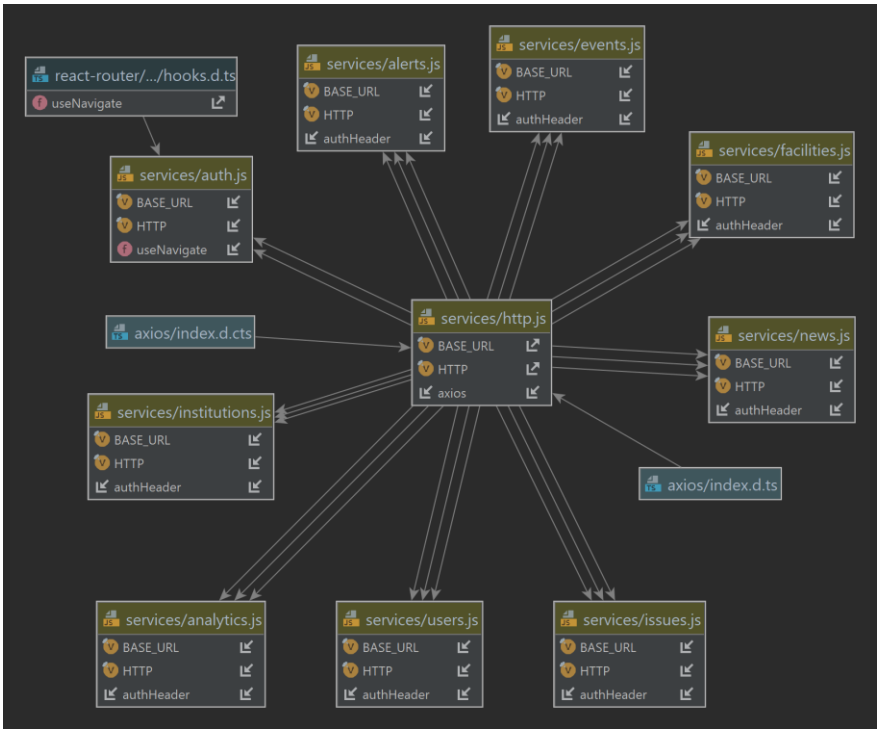


Figure 5.8. Web client services

- **http.js**: in this service, an instance of Axios is created in order to handle HTTP requests to the servers. It needs to be provided the base URL of the server. This is also where an authentication header is defined by parsing the user details which are set in session storage when logging in, and using the JWT token to define an Authorization Bearer header. This header is added to all the requests that need to be performed only when the user is logged in.
- **auth.js**: the auth service contains functions for log in and log out. On log in, a POST request is sent to the server and the jwt response with the user details is set to session storage.
- **fileUploader.js**: the file uploading service handles creating a new FormData with the selected file that needs to be stored in cloud and sending a POST request to the File Upload Server. The response message is parsed to access the URL needed for using the image.
- **issues.js:** this service is used for sending requests related to the reported issues with the necessary paths, path variables, request bodies and including the previously mentioned authHeader. The other services are similar in structure and purpose.

5.5. Design and implementation of the mobile application for citizens

The mobile client performs:

- User interface rendering of views and layouts for the regular users, consisting in the inhabitands of Cluj-Napoca or people who want to find out more about the city
- View and presentation logic
- User interaction and event handling, providing the citizens with the functionalities described in chapter 4.
- Data fetching and API communication by using the Retrofit libary to send HTTP requests to the servers
- Activity and fragment lifecycle management
- Navigation by using a navigation controller and navigation graph
- Integration of external services, Google Maps API for allowing users to select a location and display locations, Firebase Messaging for receiving push notifications
- User input validation, error handling and display

The project is structured based on the main components necessary for building an Android application. Probably the most important of those are Activities, which are screens that the user can see in the application. Fragments are similar, in the sense that they are components with their own behavior and lifecycle used within activities, allowing for UI composition. Models are data structures for representing and accessing the application data, adapters are used for binding data to UI list components, APIs are used for making server requests for data fetching and modification, while resources are assets like XML layouts and menus, images, icons, strings, colors and fonts.

Figure 5.9 highlights these components and the package structure of the mobile MyCity application.

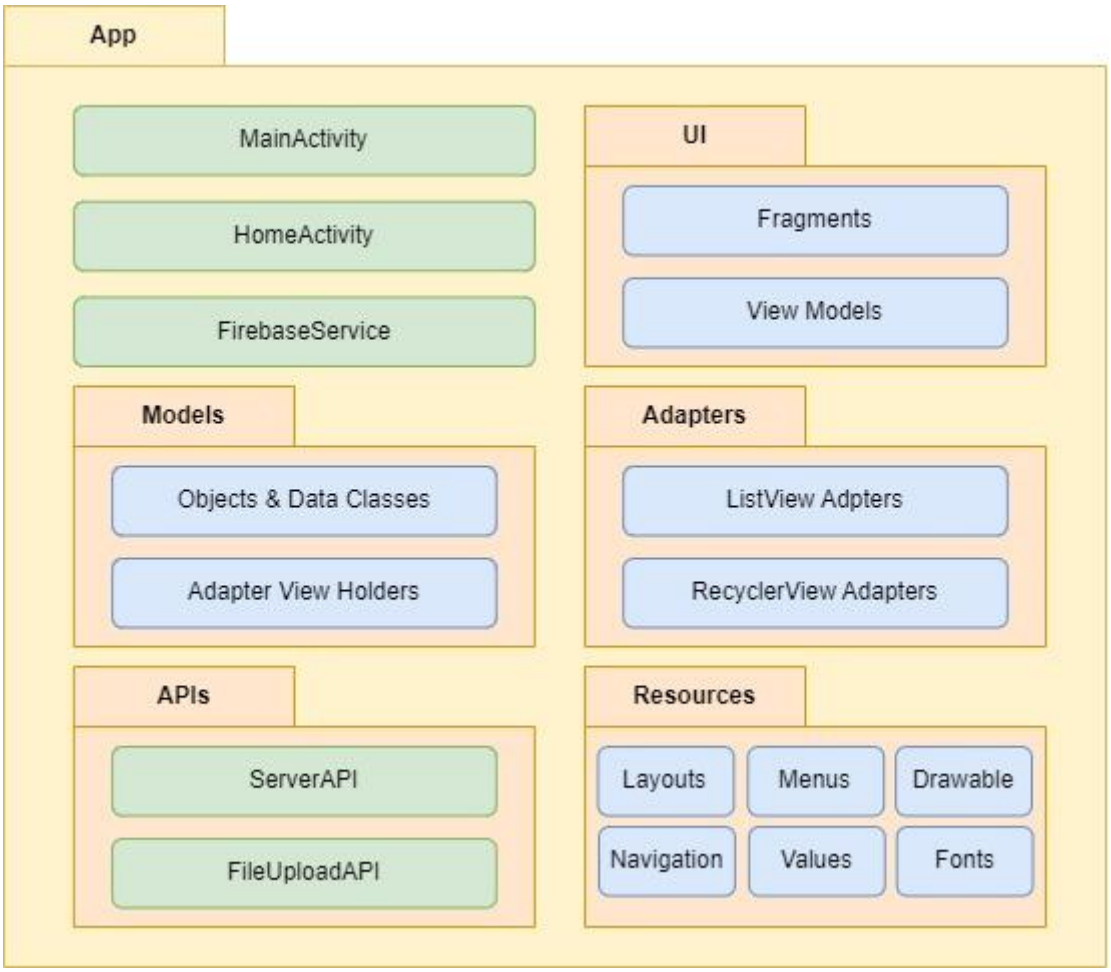


Figure 5.9. Mobile client structure

5.5.1. Activities

- **MainActivity**: this is the entry point to the application and displays the Log In fragment in case the user isn't already logged in or doesn't have an account. It is responsible for initializing the map fragment provided by Google Maps API so that it can be then used in the application. In its onCreate method, it also attempts to aceess the user details from Shared Freferences, in order to allow the application to directly open the Home Activity in case the user has previously logged in and is not logged out on their device, so that the authentication process doesn't need to be performed every time the app is opened.
- **HomeActivity**: this is the primary screen of the application, in which all the other fragments are hosted and navigation is performed between them based on what the user does. On creation, it configures the bottom navigation menu which is present throughout the application in order for quick navigation between the home section, the user profile and the alerts section. It is also responsible for displaying the home fragment containing a menu with the main app features, and for the initialization of the Firebase Messaging and its topics, so that notification intents provided by it can be handled.

5.5.2. Fragments

In the context of this application, fragments are mostly used for defining the application pages that are displayed above the bottom navigation menu. Besides the log in and register fragments, there are fragments for each option in the bottom navigation menu, for each main app option in the home menu – Issues, Facilities, Institutions, Events and News - and for each subsection corresponding to one of these options. Each of them has an associated XML layout and option menus.

Each fragment is resposible for accessing their corresponding layout binding and initializing the layout elements with application data fetched from the server or passed by the previous fragment as arguments. In case the fragment has an associated view model, it also has to be initialized with application data.

Apart from this, in the fragments' onCreateView methods, adapters are created and attached to list views, and event listeners are attached to interactive elements of the layout, such as buttons, menus, dropdowns and text fields. This allows user interaction to be properly handled and the necessary operations to be performed accordingly.

5.5.3. Navigation

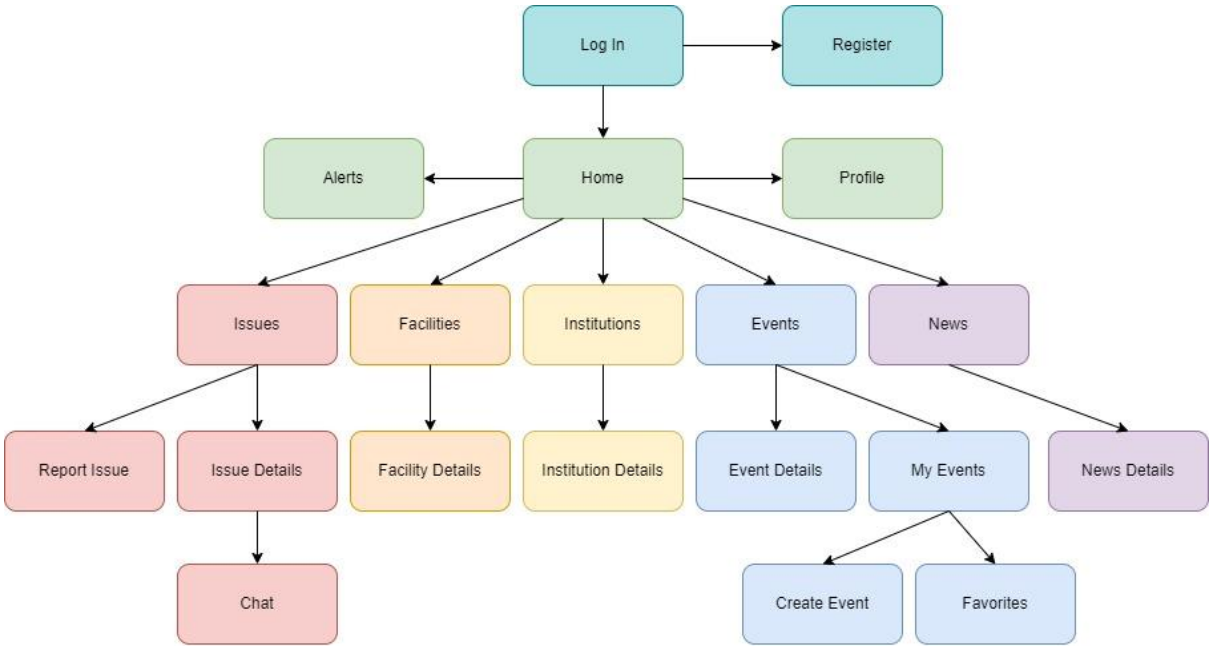


Figure 5.10. Mobile client pages and navigation

Figure 5.10 illustrates the pages that can be accessed within the application in a general tree hierarchy for simplicity, but the actual navigation is more like a graph. After a successful register, the application navigates back to the log in Fragment. The green fragments represent the pages that can be accessed by using the bottom navigation menu from anywhere in the app. The colored fragments below the Home fragment can be navigated to using the home menu, each of them then having multiple navigation options for accessing information, issue reporting, event creation and so on. In case of events, the details Fragment can also be reached from the user's own events page and from his favorites section. Similarly, the Issue Details page can be reached directly after submitting an issue and from anywhere in the app upon receiving a message notification from an institution employee.

Navigation is performed between fragments of the application by using the acitivity's nav controller, based on the mapping done between fragments and their layouts in the mobile navigation XML. When an event such as a button click is performed, Android allows the creation of a fragment which will be nagivated to and displayed, also allowing passing arguments between fragments using a Bundle. For example, when the user selects an event from the list, the corresponding event object is passed as argument to the EventDetails fragment which allows accessing its data within the fragment. For navigation, the nav controller needs to be passed the resouce id of the mobile navigation fragment definition, along with the optional data bundle, that is serialzed and then deserialized.

5.5.4. Models

The models are used for encapsulating the application data in useful structures, that can be collected and passed. In this context, they are of three types: objects, data classes and view holders.

- Objects: These are singleton classes that can be used anywhere in the application, like the User object, which is initialzed with data at log in or from shared preferences at application launch, and can be used when the user's id, name, username or jwt token are needed.
- Data classes: These represent the application entities, such as the Facility, Event, Institution, Issue and so on. They are also used for defining the request and response body structures for API calls, such as the AuthRequest and Response, or the MessageResponse.
- View Holders: These classes define the fields contained in a ListView or RecyclerView and are used by adapters for populating the lists with application data.

5.5.5. Adapters

The adapters are responsible for binding data, such as the list of facilities in the city that the user can view, to the UI's RecyclerView element that displays the facilities based on the view holder of a facility list element, that contains the facility image. name, type, and location.

They also handle events such as clicks on a specific element in the list, the updating of the elements stored in the list and their filtering based on user interaction. For example, when the user selects that he only wants to view the existing parks, the adapter filters the list based on the selected type to only display the requested elements.

5.5.6. APIs for server access

The application contains two API classes corresponding to each one of the two servers that can be accessed. They are defined as interfaces with method signatures that are mapped to the server's endpoints.

```
@POST("auth/login")
@Headers("Content-Type:application/json")
fun authenticate(@Body body: AuthRequest): Call<AuthResponse>
```

For example, the authenticate method signature called when the user clicks log in has an adnotation with the request type and endpoint, one for the the request headers, the AuthRequest body as parameter, and a Call<AuthResponse> as the return type, which is a

callback that contains an AuthResponse object and needs to implement the onResponse and onFailure methods after the authentication request is sent in the Log in fragment.

The interface also needs to define a companion object, which provides a createApi function that can be called anywhere a request needs to be performed. This function creates a ServerAPI instance by using the Retrofit builder and providing it the base URL of the server. Retrofit also needs to be provided a converter factory, which performs automatic bidirectional conversion between the objects and JSON strings used as content type for the requests.

5.5.7. Android resources

The resouces package contains multiple sub packages. The standout ones are:

- **Layouts**: These are the most important resources, as they define the layout structure used by fragments in XML files that contain Material Deisgn elements in a tree-like structure, such as the <LinearLayout>, <RecyclerView>, <TextView>, <ImageView>,<Button> etc. Each of them can be provided properties like width, height, ids for access from code, margins, text styles and so on.
- **Menus**: Menu layouts contain a list of items that are displayed when a menu is inflated and allow for operations to be performed based on the user selection.
- **Drawables**: This package contains images that need to be displayed in the application besides the ones that are accessed from cloud, and XML files for defining icons as vectors and configuring their style properties.
- **Navigation**: This is where the fragment navigation XML file is placed
- **Values**: These are XML files that contain application values, such as predefined strings, dimensions and colors that are used. It is also used to store the API key for Google Maps API.

Another important component of the Android application is the AndroidManifest.xml file, where the app's activities are defined, and meta data like the API keys is placed. It is also used for defining the Firebase Service, and creating a FileProvider for accessing the files on the users' devices, in case they want to upload a photo instead of taking one. Apart from these, it is also the place where the internet, network state, wi-fi state, fine location, coarse location, read external storage and camera permissions are declared.

5.6. External Services

5.6.1. Google Cloud Storage

Cloud storage is used for storing images that provide better visualization for institution, facility, news and event details, as well as images that are uploaded when regular users report issues or create their events. This service allows for a large storage capacity, reliability and availability, and an easy upload process and access by URL.

For the use Google Could Storage, a few steps were required: creating a new project in the Google Cloud Console, creating a bucket where files are stored, setting up a service account with create permissions, and providing the JSON keyfile associated with that account, as well as the project ID and bucket ID, to the server used for image uploading. Once images are uploaded to the cloud, they are stored in the application's database using the access URL and used by the clients.

5.6.2. Google Maps, Location and Geocoding APIs

These services provide map visualization and interaction, access to the user's location and coordinates based on their devices' GPS services, and automatic translation between addresses and coordinates.

In order for them to be used, the APIs need to be enabled for the project from the Google Cloud Console, which allows the creation of API keys that are provided to the mobile client in its manifests file, and to the web client as part of the load script. The GMS play-services-maps and play-services-location also need to be added as dependencies in the project's build.gradle file.

Location API is used in the IssueLocationFragment of the Android application by asking the user for location permissions, creating an instance of LocationServices and using its getFusedLocationProviderClient to define a callback function that is repeatedly executed at a fixed rate and that returns a LocationResult with their exact coordinates. These coordinates are then used to display the user's location on the map, and are passed to the location field of the issue report form in case the user confirms the location selection without placing a marker on the map.

Geocoding API allows the coordinates provided by the Location API or by the retrieval of Marker coordinates in the onMapClick function to be translated to an address which is then also included in the created issue object, so the issue address can also be viewed by institution employees besides the map view. It is done by creating a Geocoder instance, which executes an asynchronous task based on the provided coordinates and returns a list of suggested addresses.

5.6.3. Firebase Cloud Messaging

Firebase allows for sending push notifications from the application's server to the users' devices when an alert or message is sent by the administrator or institution employees. This is implemented in a couple of steps.

The first step requires creating a firebase project and associating it with the previously created google cloud project, then creating an admin service acount. In the server application, the firebase-admin dependency needs to be included in the pom.xml and the FirebaseInitializer needs to be configured with the JSON keyfile corresponding to the service account. Then, messages can be sent using the FirebaseMessaging instance by using the Message builder.

In the Android application, the FirebaseInitializer is created in the HomeActivity, topic subscription is performed, and a service class is defined. The FirebaseService handles the push notifications' creation based on the topic messages were received from the server. It is used for defining the notification layout, its display and priority, and the intent started when clicking it.

Chapter 6. Testing and Validation

This chapter focuses on the testing and validation of the functionalities provided by MyCity app for Cluj-Napoca.

Probably the most important component of the application is the mobile component, as it represents the end product that is made to be used by the citizens, and that encompases the largest portion of the project's functionalities. Since the application integrates external services and requires steps such as taking photos or selecting locations on a map, which would be more difficult to test automatically, a manual testing approach was chosen. Testing was performed in a functional manner, meaning that the aim was to validate whether the application functions correctly, according to the specified goals, requirements and use cases described.

In order to test the functioning of the application, test scenarios were created based on the use cases described in Chapter 4. Testing was performed for all the app's functionalities, but the next section will only incude testing the entire process of an issue being reported by a citizen and then solved by the institutions it is assigned to.

We assume a scenario where a citizen notices a company is performing repeated illegal dumping in an area of the city. He can use the application to report the issue and select the "Garbage Issue" type, which upon reporting is automatically assigned to the company designated for city sanitation, Supercom SA in case of Cluj-Napoca. They notice the issue also requires investigation of the company performing illegal dumping, so the issue is also shared with the local police department, who can view its details. The police can ask the user questions about the issue in the chat section, in order to get more context information. Once both involved instititutions do their part, they can mark the issue as solved, which the user will be able to see.

This scenario allows for testing multiple use cases, each with a couple of steps.

- Citizen reports the issue
  1. User opens the application and selects the Issues option from the menu
  2. User clicks the "+" button to report a new issue and the form section is displayed
  3. User selects the "Garbage Problem" type, enters a title, description, uploads a photo of the incident and selects the location on the map
  4. User submits the issue, which is assigned to Supercom SA, and is displayed in the user's issues list

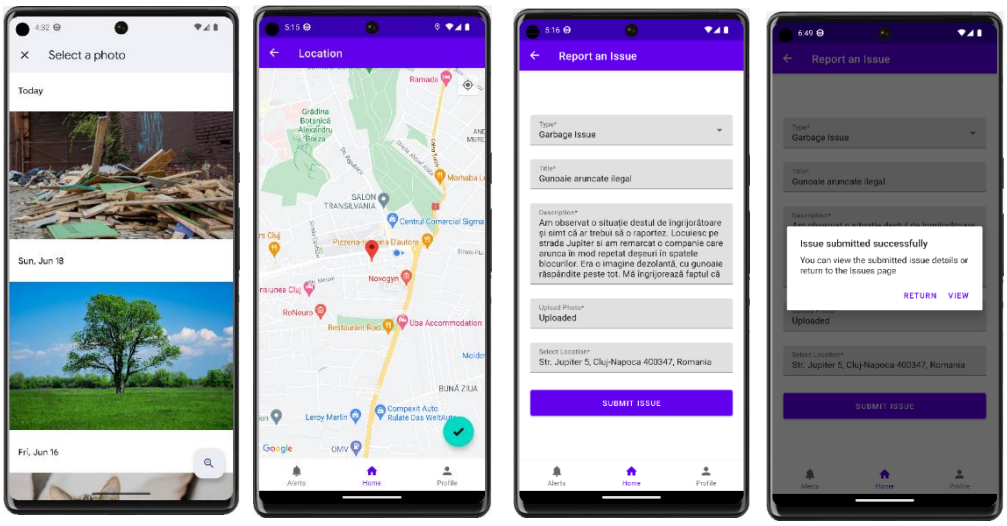


Figure 6.1. User reports issue

- Employee of Supercom SA views, assigns and shares and solves the issue
  1. User opens the application and navigates to the Issues table
  2. User notices a new submitted issue, which he views the details of and assigns to himself, and based on the provided description, shares it with the police department by using the share section
  3. After the issue is solved on their institution's part, user marks it as solved

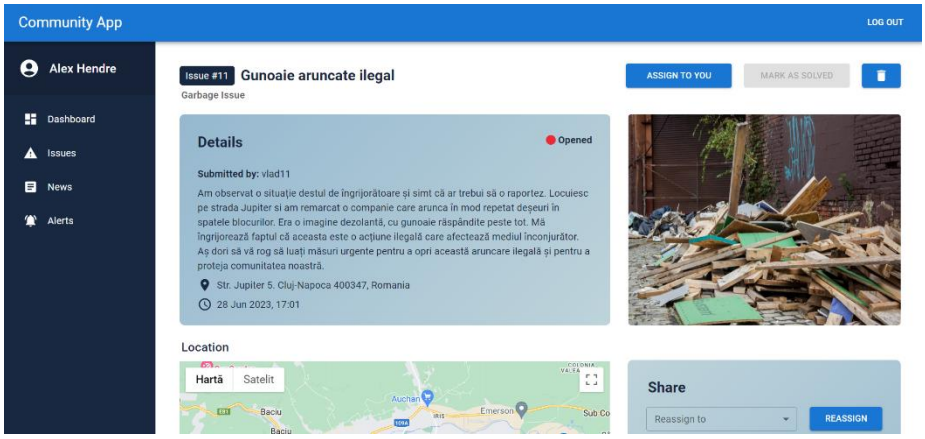


Figure 6.2. User views issue details

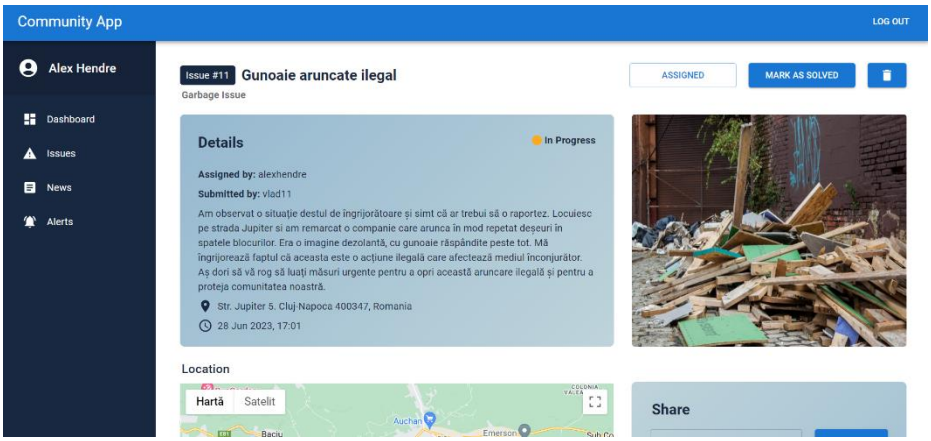


Figure 6.3. Details updated after user assigns it

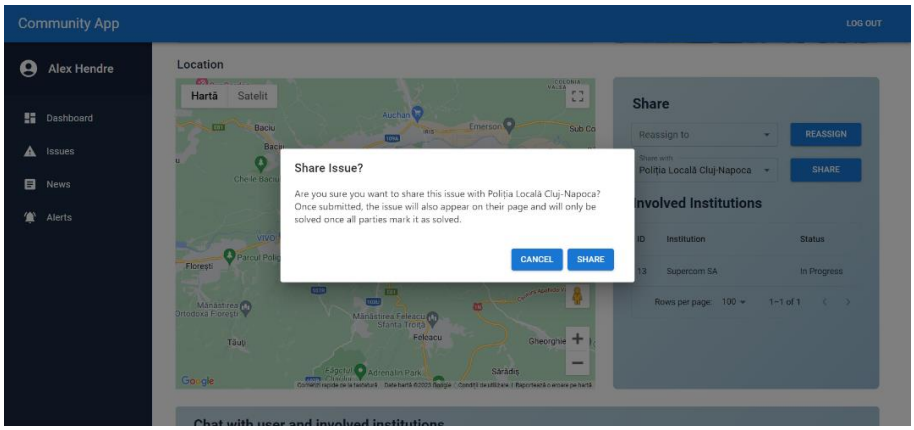


Figure 6.4. User shares issue with Politia Locala Cluj-Napoca

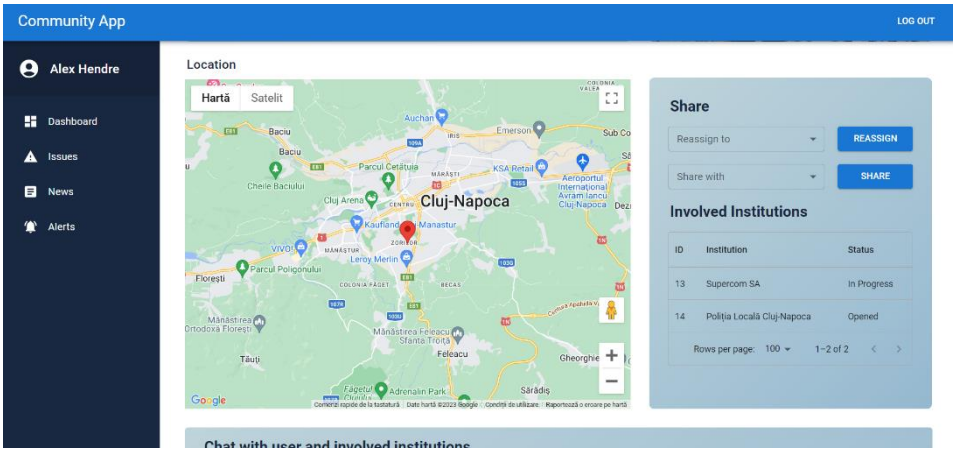


Figure 6.5. Issue location and involved institutions

- Employee of the police department views, assignes the issue, asks the user a question, and solves the issue
  1. User opens the application and navigates to the Issues table
  2. User notices the shared issue, which he views the details of and assigns to himself, and based on the provided description, asks the user additional details, which the user can provide
  3. When the issue is handled on the police's part, the user also marks it as solved

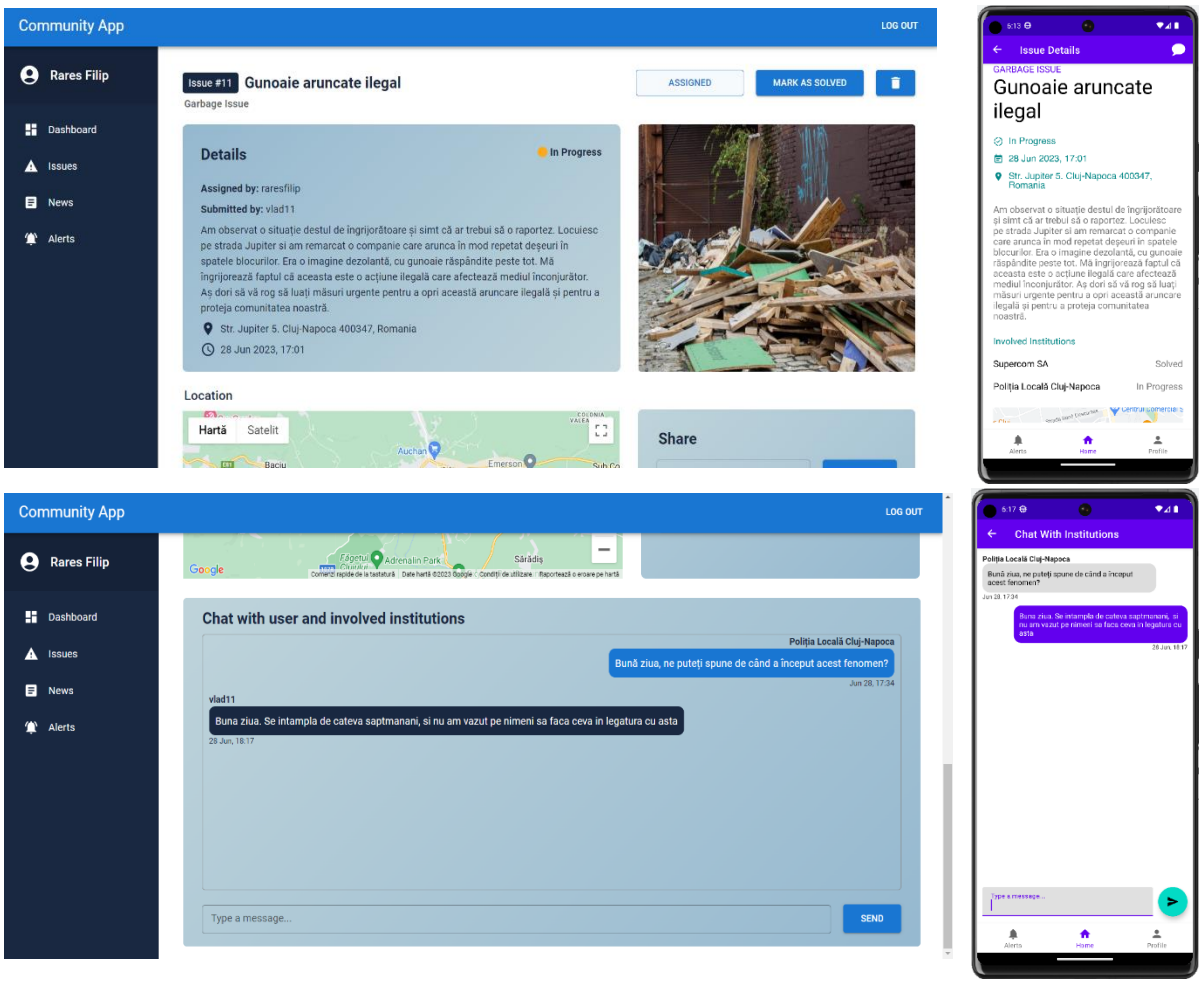


Figure 6.6. Issue details and chat on police employee page and user device

In figure 6.6., we can see the issue is also visible in the police employee page once the initial employee from Supercom SA shared the issue. The police employee also assigns the issue and then he can use the chat section to communicate with the citizen. He asks a question, the user receives a notification that when clicked displays the issue details page. They can view the involved institutions and the current status of the issue. When opening the chat section, they are able to view the message sent by the institution employee and answer to it.

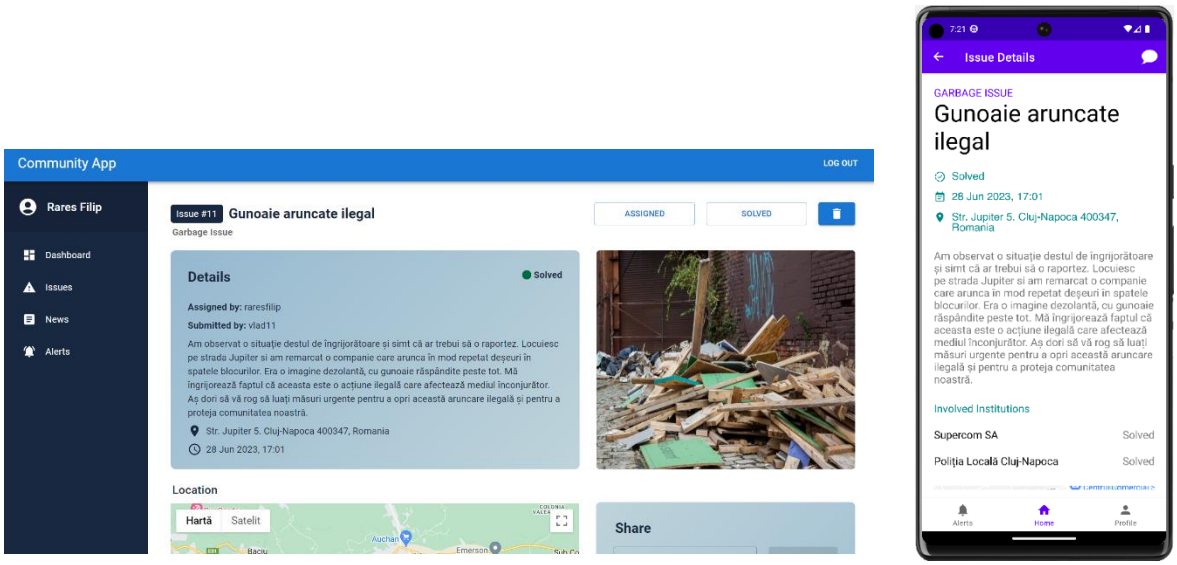


Figure 6.7. Issue displayed as solved

Once the police employee handles their part of the issue, they also mark it as solved, and since it was previously marked as solved by the institution employee from Supercom SA, the issue is solved overall. This is also visible on the user's device.

Testing based on use case scenarios was performed on all functionalities of the application. In all cases, the actual results matched the expected results.

Chapter 7. User's Manual

This chapter presents the hardware and software resources necessary for installing the application, as well as a user manual for using the application, from the point of view of the citizens, institution employees and administrator.

7.1. System Requirements

Servers:

- PC/Laptop with Windows
- IntelliJ IDEA or another IDE compatible with Java Spring and Express apps
- JavaSDK 17 installed
- MySql installed

Web Client:

- PC/Laptop with Windows
- IntelliJ IDEA or another IDE compatible with React apps
- A browser like Chrome, Firefox etc

Mobile Client:

- Android device with Android version 8.0 (Oreo) or higher
- Internet connection
- At least 150MB available storage space

7.2. Installation

Application Server:

- Download application directory and open it in IDE
- Modify application.properites database configurations of user and password
- Add a MySql datasource to the project and create a schema
- Run the server

File Upload Server:

- Download application directory and open it in IDE
- Run the server

Web Client:

- Download application directory and open it in IDE
- Run the client with "npm start" in terminal, which will open a web browser

Mobile Client:

- Download the application's APK file and install it on device
- Open the application

7.3. Use manual

Citizen:

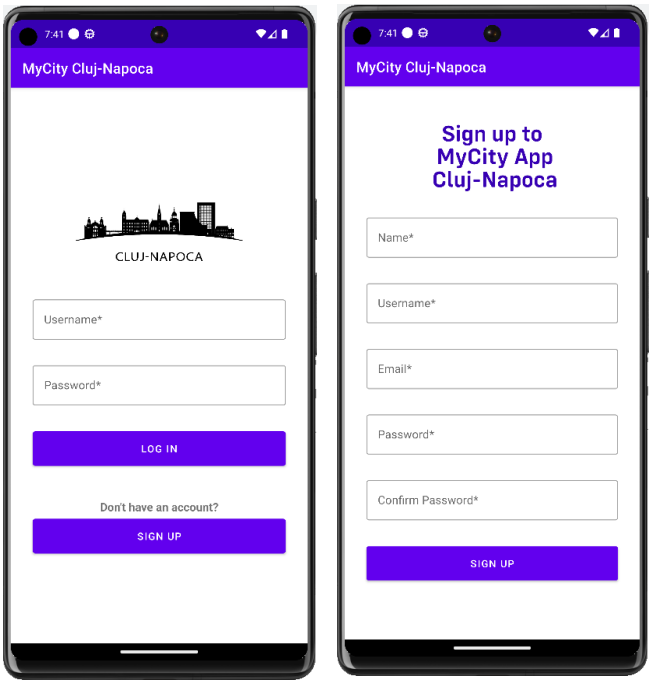


Figure 7.1. Log In & Sign Up pages

To log in, users need to enter their username and password and click the Log In button. If they don't have an account, they can click the Sign Up button which displays a new screen for account details. Once they sign up, a confirmation is displayed and they can log in.

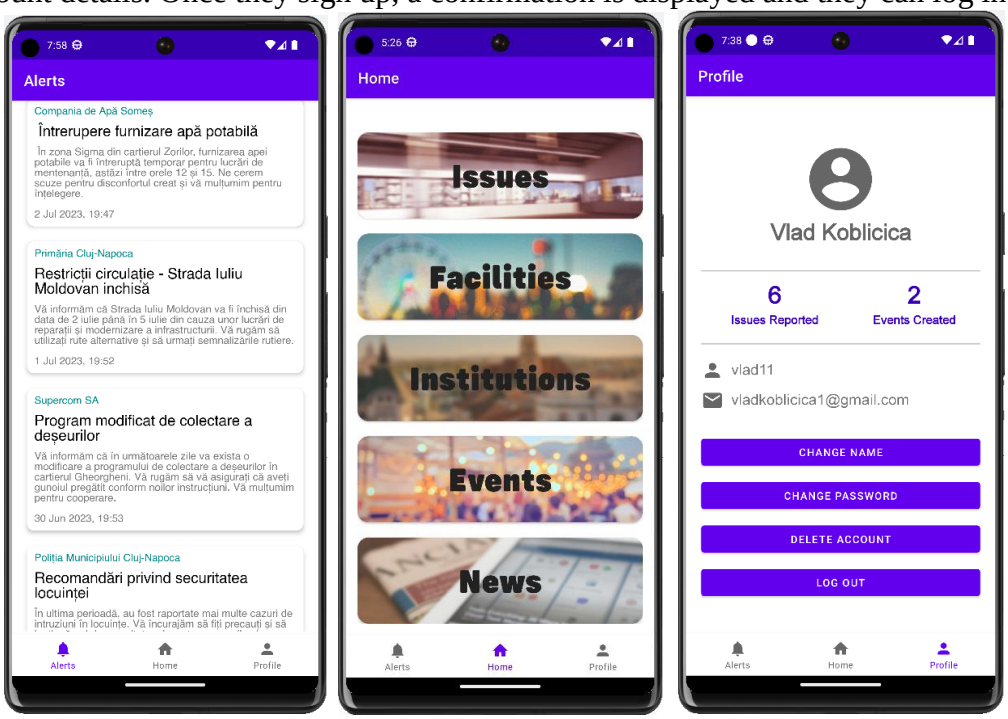


Figure 7.2. Home, Alerts and Profile pages

After logging in, the Home page is displayed and navigation can be done to any of the app options, or to Alerts or Profile. In the profile section, there are buttons for actions that can be performed.

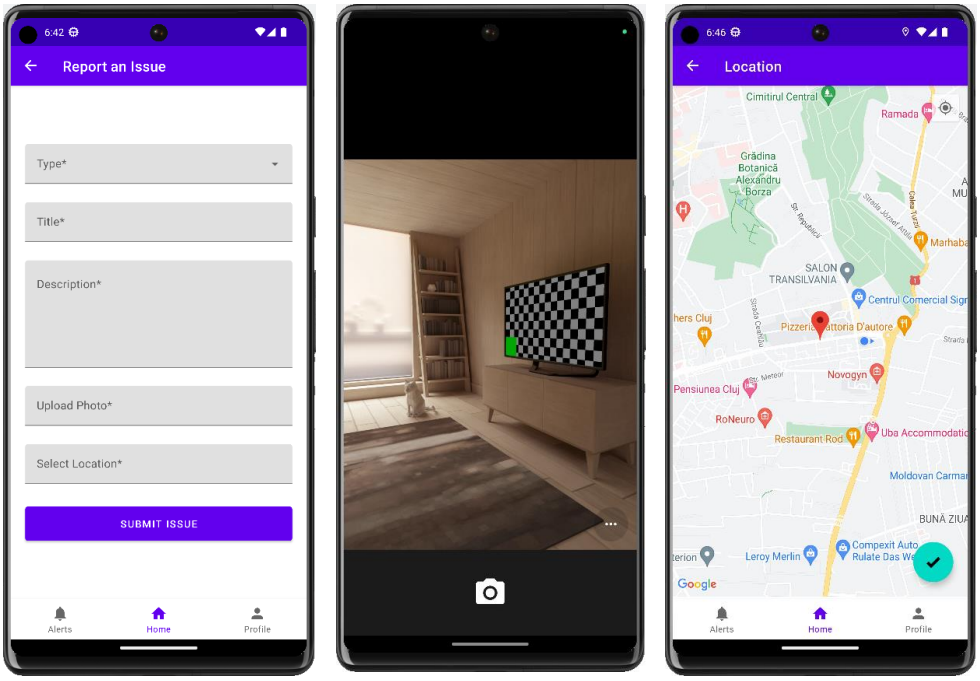


Figure 7.3. Report Issue page, Camera and Location page

To report an issue, users need to click the "+" button in the Issues page, and the report form is displayed. They need to enter a type, title, description. When clicking the photo field, they are given the possibility to take or choose a photo, which opens camera or gallery. To display the map, they need to click the Location field. In the map section, they can select confirm using their current location or put a pin on the map and confirm. After the form is filled, they need to submit the issue, and a success message is displayed.

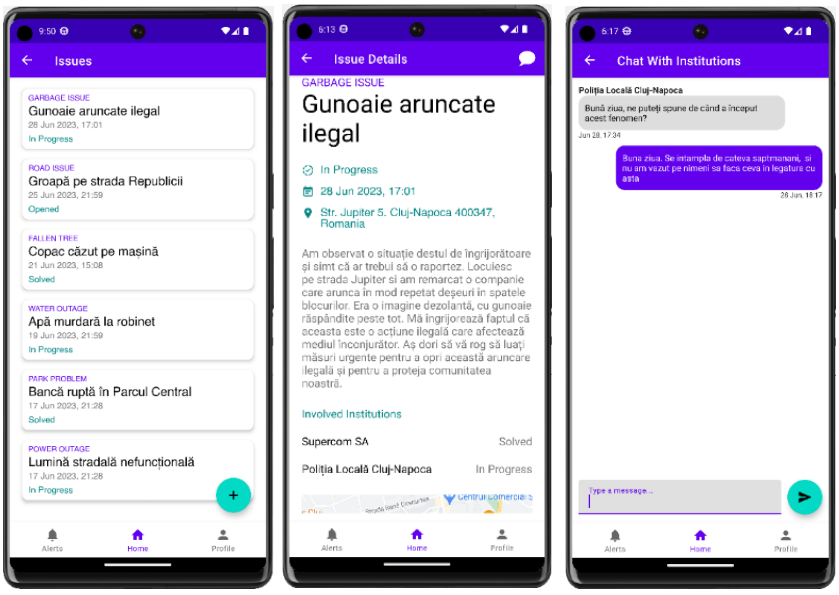


Figure 7.4. Issues, Issue Details, Chat pages

In the Issues page, users can view the status of the issues they submitted. By clicking on one, the Issue Details page is displayed, where they can review their submission and see what institutions are involved. Here, they can select the chat option and a chat section is displayed, where they can send messages to involved institutions and view received messages.

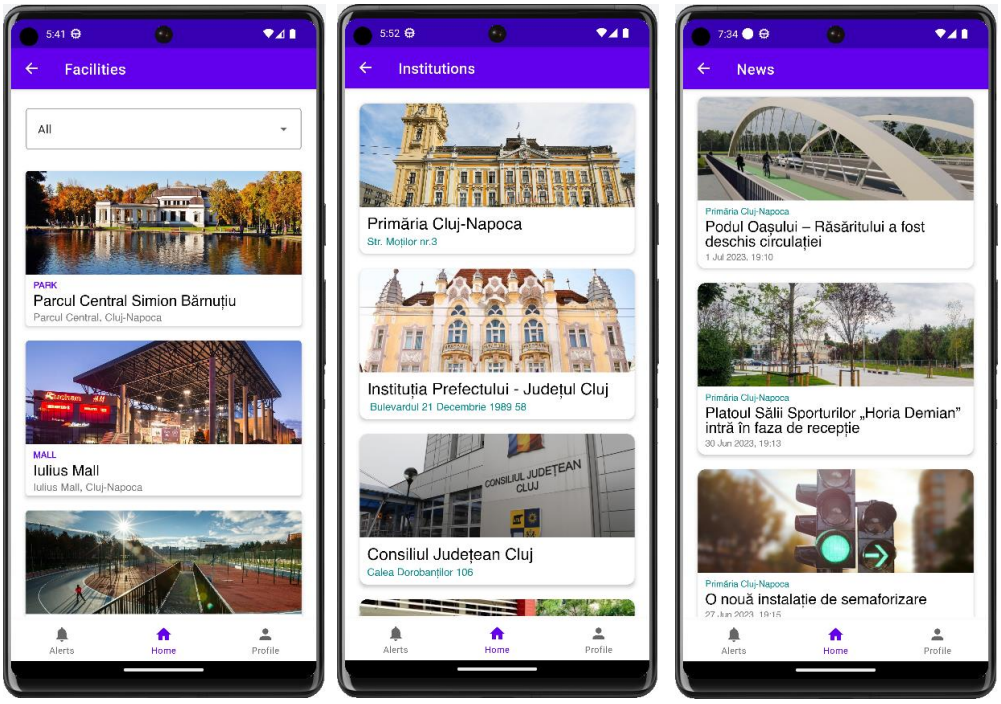


Figure 7.3. Facilities, Institutions, News pages

If the user selects Facilities, a list is displayed and they can scroll to view the city facilities or filter them by type. Similarly, if they select Institutions, the list of public institutions and service providers in the city is displayed. By selecting News, they can browse through the news posted by institutions.

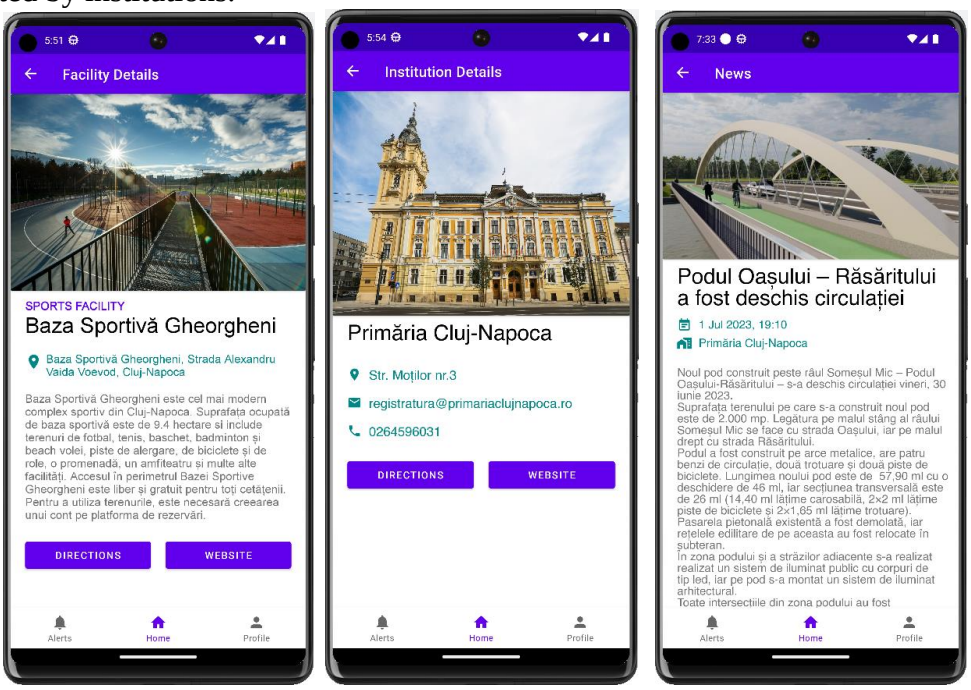


Figure 7.4. Facility Details, Institution Details, News Details pages

By clicking on a facility, a page with details is displayed, and they can get Google Maps directions or access its official website using the buttons. By clicking an institution, users can access the contact details, directions and the website. Similarly, by selecting an article from the news list, they can read it and open the posted article on the institution's website.

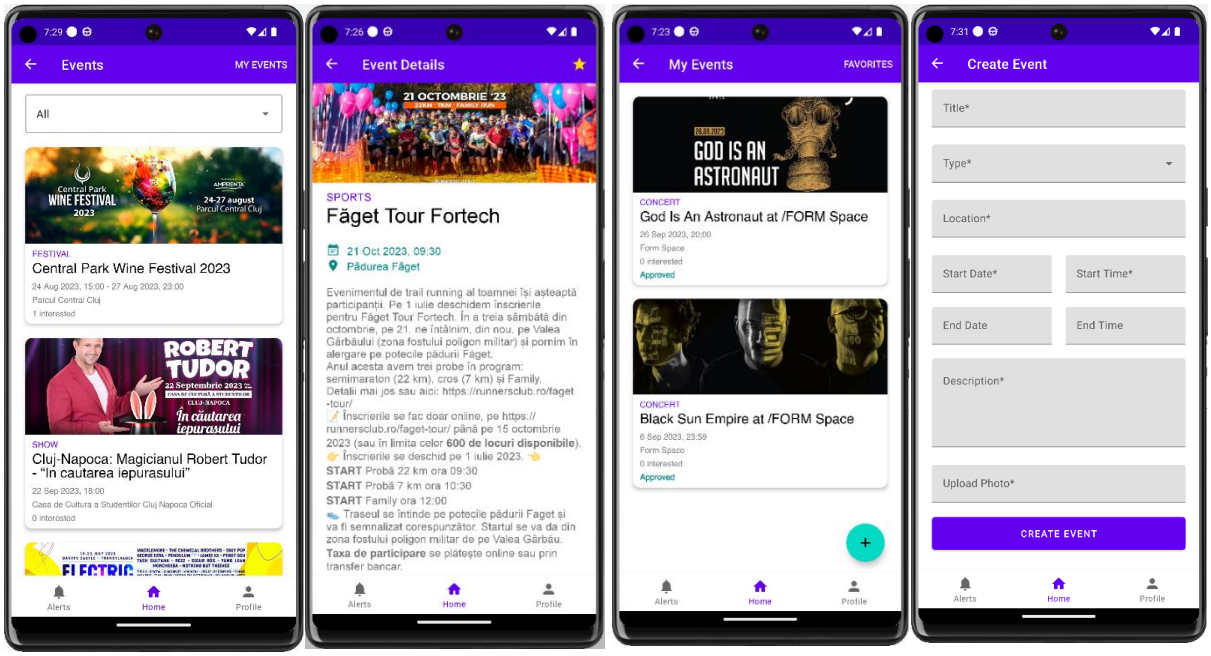


Figure 7.7. Events, Event Details, My Events and Create Event pages

If the user selects Events, they can view and filter a list of events happening in the city. By selecting one, the Event Details page is displayed, where they can view more information, and add/remove the event from favorites by clicking the star icon. If the user selects the My Events option in the Events page, a list of the user's created events is displayed, and an option to view their favorites or create an event by clicking "+". If they want to create an event, a form for details is displayed, which they can fill and submit.

Institution employee:

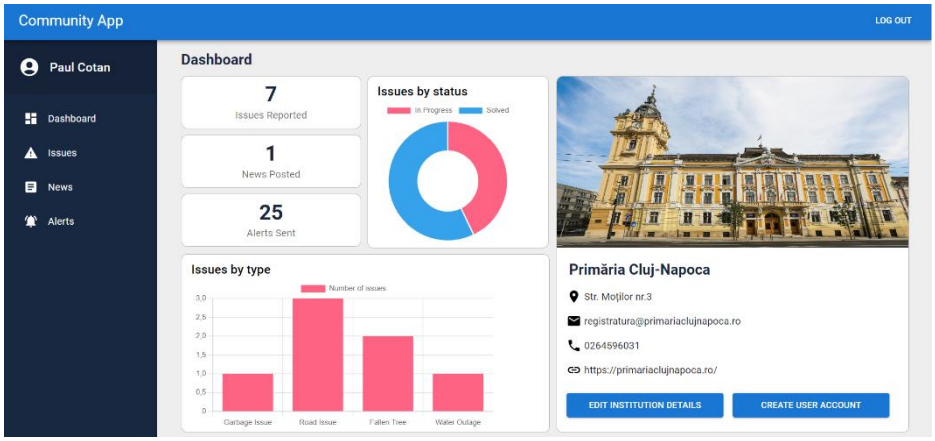


Figure 7.8. Institution dashboard

From the dashboard section, users can view statistics and charts regarding their institution's activity. Using the buttons, they can update the institution details or create new accounts for employees to use. From the top bar, they can log out, or they can use the sidebar menu to navigate to the issues, news or alerts section.

In the issues section, users can view issues submitted to their institution, use the dropdown to filter based on status, and the checkbox to display issues assigned by them. Each table row has a delete option and a view option, which displays the issue details page.

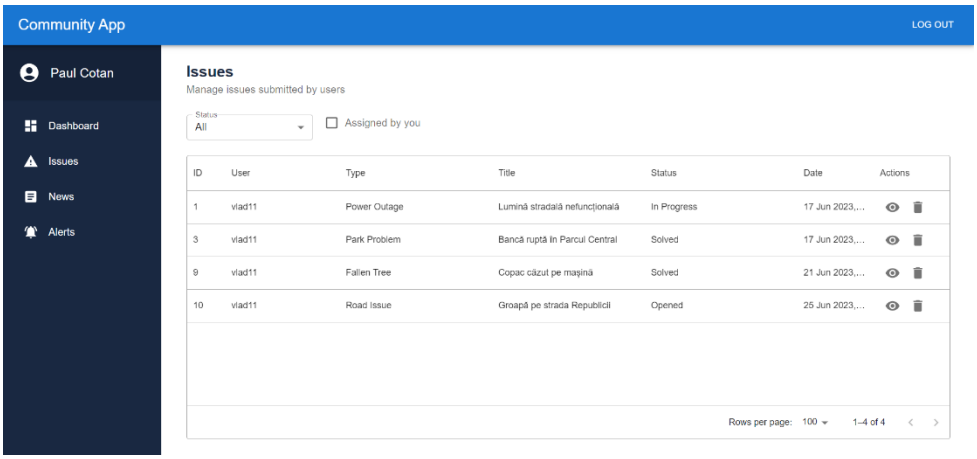


Figure 7.9. Issues page

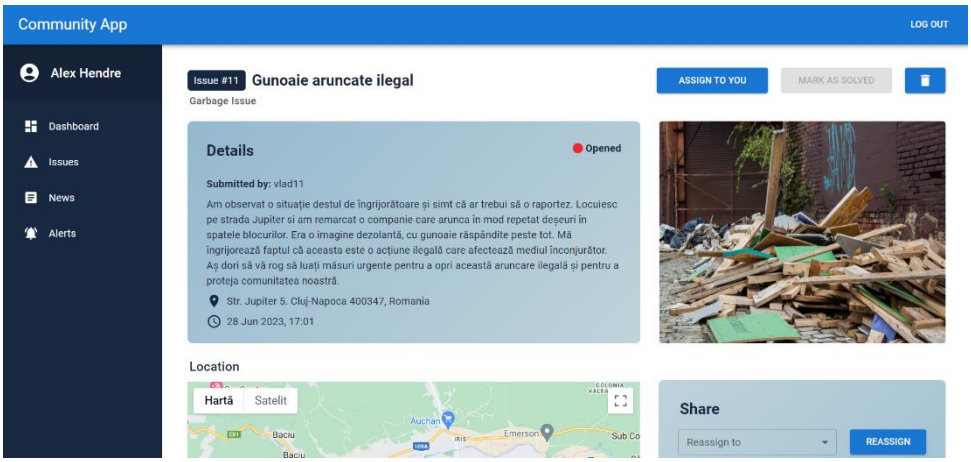


Figure 7.10. Issue Details page

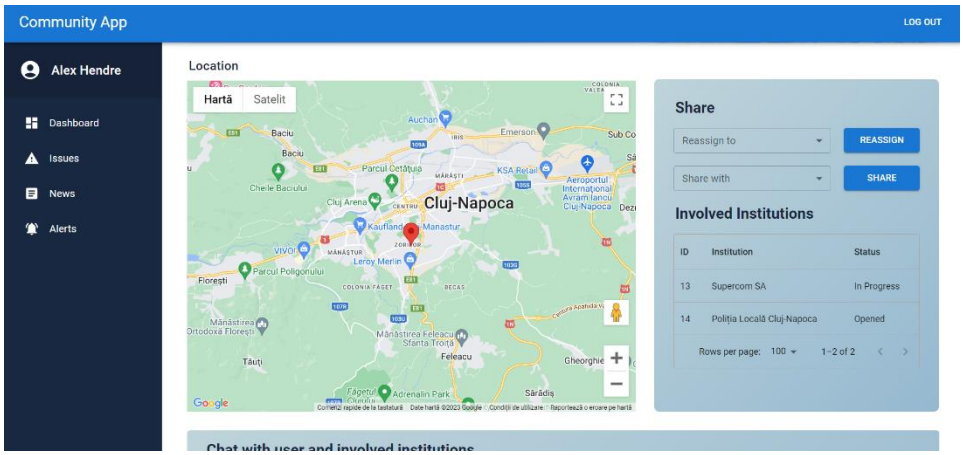


Figure 7.11. Issue Details page

In the issue page, users can view the details, image, and map location of the issue. To assign it to themselves, they need to click the Assign button, which enables the Solved button and allows the user to share the issue and use the chat section. The issue can be reassigned by selecting another institution and clicking the Reassign button. Share can be performed similarly, in the same section. To send a message, the user can simply use the text field and Send button.

If they want to view or delete previously posted news and alerts, the process is the same as with issues. To create news or alerts, they need to use the Create button which will open a create dialog.

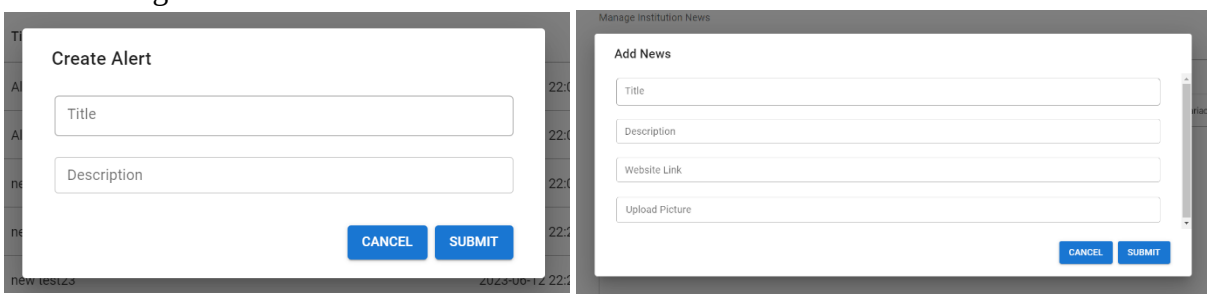


Figure 7.12. Create news and alerts

Administrator:

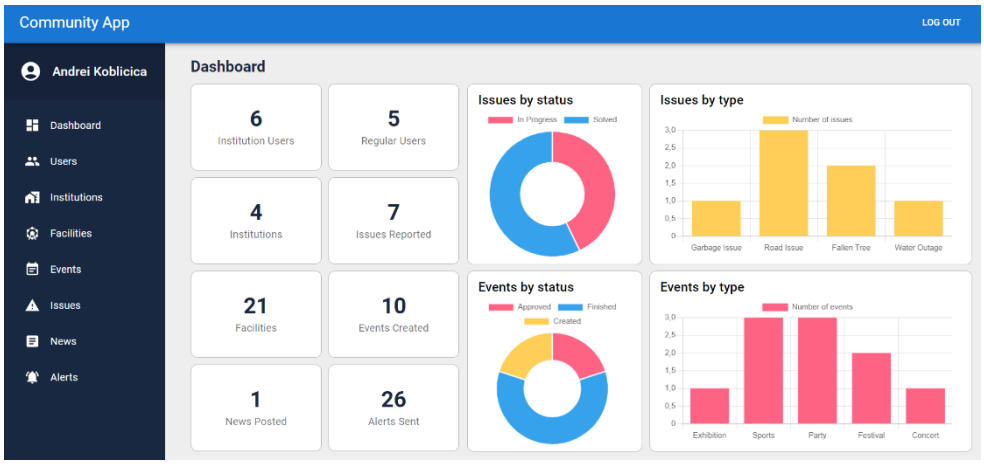


Figure 7.13. Admin dashboard

The administrator can view statistics and charts about the application's entities in the dashboard section. They can use the log out button or navigate to one of the pages in the side menu.

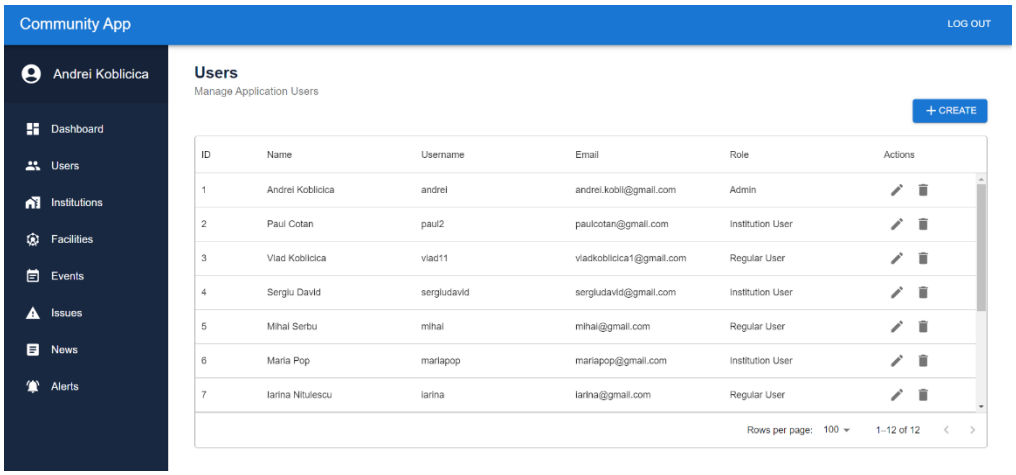


Figure 7.14. Users page

For users, institutions, issues, news and alerts, the admin can view data in a table, with each row having a delete option, along with an edit option, or view option in case of news and alerts. To create a new instance, he can use the Create button and complete the pop up form.

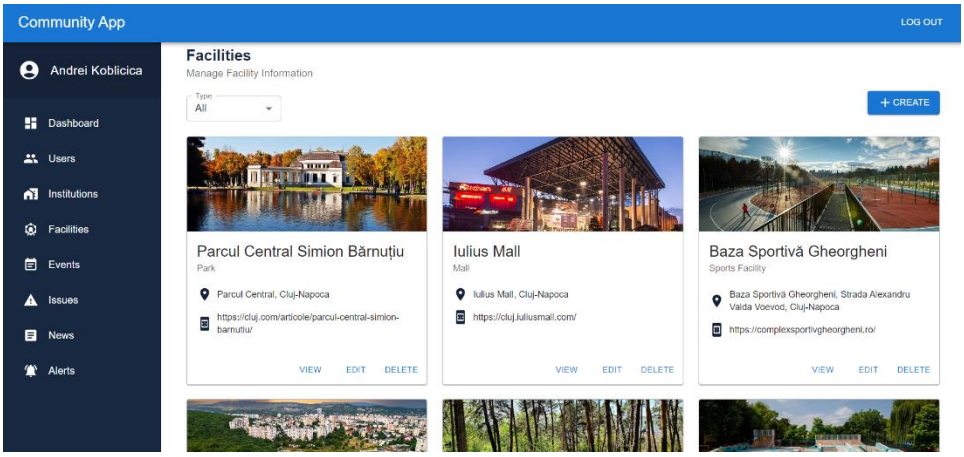


Figure 7.15. Facilities page

In the facilities page, the admin can filter them based by type using the dropdown menu, and each facility has options for view details, edit or delete.

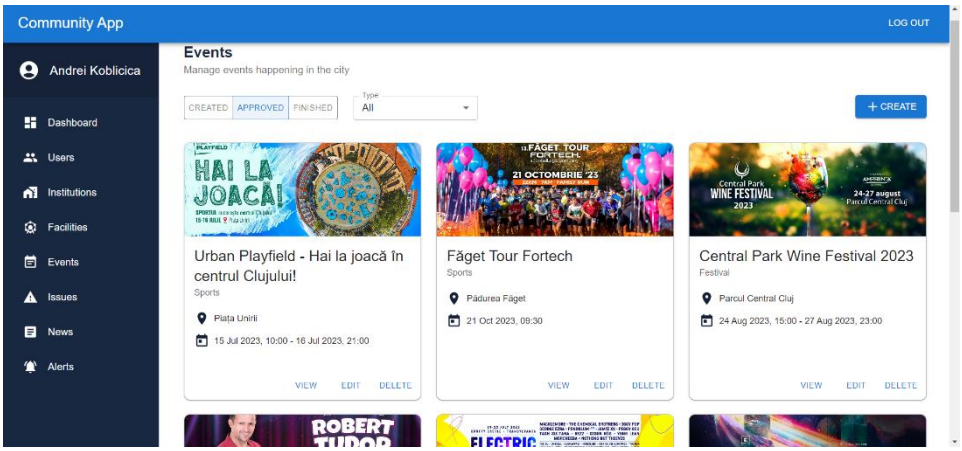


Figure 7.16. Events page

The events page is similar to the one for facilities. Additionaly, there is a toggle button where they can switch between created events that need approval, approved events, and finished events. To approve an event, they need to click the Approve button and confirm.

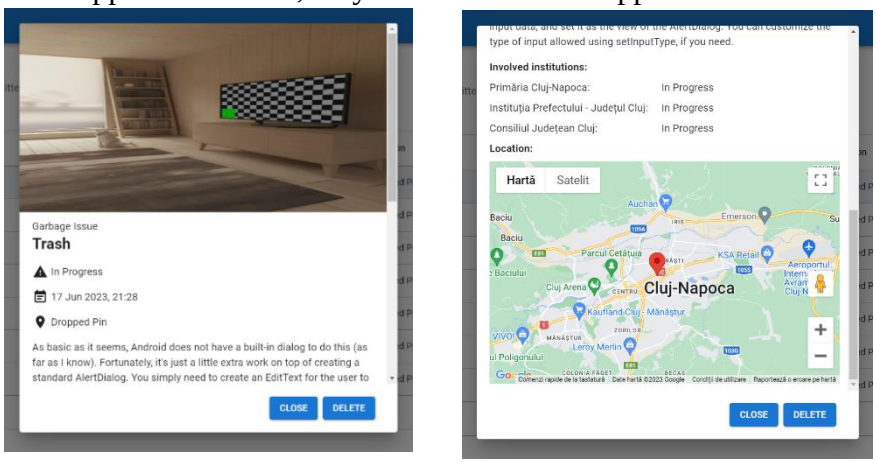


Figure 7.17. Issue details

Even though the admin cannot be involved in the issue solving process, they can still view the issue details in a dialog and delete it if necessary.

Chapter 8. Conclusions

This chapter will contain a summary of what this project managed to achieve, and a description of possible future improvements and further development.

8.1. Achievemnts summary

In conclusion, this project accomplished the objectives that were initially defined. It can be a very useful tool for the people in Cluj-Napoca, improving their overall knowledge about the things that are happening in the city, while also contributing to a better quality of life thanks to the proposed way of handling issues based on collaboration between the population and the public institutions and service providers. It managed to achieve some functionalities already present in other applications, but build upon them by including additional features dedicated to the city of Cluj-Napoca, while also attempting to offer a better overall user experience. To summarize, the application managed to provide:

- A fast and effective way for users to report problems encountered in the city
- A better way for institutions to handle the issues by allowing them to be aware of their existence sooner, and to easily visualize the underlying context and details
- A way for general communication to be improved by allowing direct interaction between citizens and institutions in the issue solving process, as well as through the accessibility of contact information and the possibility of posting and viewing news and alerts
- A means for the citizens to discover the facilities, parks, attractions and events that the city has to offer

8.2. Future development

This project can still be further improved by providing additional features, such as:

- Allowing users to register to the application by using their Google or Facebook accounts
- Adding an extra layer of security by implementing mail sending for confirmation when users create an account
- Providing institutions with a better overview of the issues reported to them by displaying the issues with markers on a map of Cluj-Napoca
- Allowing users to view a map with all the issues that have already been submitted, so that the chances of the same problem being reported multiple times is decreased
- Creating a section with frequently asked questions to help the users find information they might be looking for regarding the application
- Sending notifications to users when other events occur, such as when the administrator approves their event, or when the issue they submitted is marked as solved
- A better looking and more usable user interface
- Optimizing the performance of the application and providing better respose times
- Deploying the application and adding it to Google Play Store so that it can be used by the population

Bibliography

[1] Scholary Community Encyclopedia, "Crowd Sensing". Available: https://encyclopedia.pub/entry/1215#:~:text=Crowd%20sensing%20(also%20known%20as,fragmenting%20tasks%20into%20small%20pieces.

[2] P. Bellavista, G. Cardone, A. Corradi, L. Foschini and R. Iannielo, "Handbook of Research on Social, Economic, and Environmental Sustainability in the Development of Smart Cities", IGI Global, 2015.

[3] R.K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," IEEE communications Magazine, 49(11), pp.32-39, 2011.

[4] D. Wang, T, Abdelzaher and L.Kaplan, Social Sensing: Building Reliable Systems on Unreliable Data, Morgan Kaufmann, 2015.

[5] C. Dobre and F. Xhafa, Pervasive Computing: Next Generation Platforms for Intelligent Data Collection, Morgan Kaufmann, 2016.

[6] InfoSec Institute, "Crowdsensing: State of the Art and Privacy Aspects". Available: https://resources.infosecinstitute.com/topic/crowdsensing-state-art-privacy-aspects/.

[7] L. Mäkelä, M. Boedeker and N. Helander, "Value Emergence in the Usage of Mobile News Alerts", in Digital Journalism, Routledge, 2019.

[8] J. Hou, L. Arpan, Y. Wu, R. Feiock, E. Ozguven and R. Arghandeh, "The road toward smart cities: A study of citizens' acceptance of mobile applications for city services," in Energies, 13(10), 2496, MDPI, 2020.

[9] Mission Hills, 2020. Available: https://play.google.com/store/apps/details?id=com.hudld.missionhills1&hl=en&gl=US

[10] SeeClickFix, 2010. Available: https://play.google.com/store/apps/details?id=com.seeclickfix.ma.android&hl=en&gl=US

[11] MyCluj, Municipiul Cluj-Napoca, 2017. Available: https://play.google.com/store/apps/details?id=com.indsoft.mycluj&pli=1

Appendix 1 – List of figures

Figure 3.1. Mission Hills Application……..…….10
Figure 3.2. SeeClickFix Application……..……..….10
Figure 3.3. MyCluj Application…….……...11

Figure 4.1. Citizen use cases ……....19
Figure 4.2. Institution employee use cases…….……..….19
Figure 4.3. Administrator use cases…….……19
Figure 4.4. UC – Log in ……...…….20
Figure 4.5. UC – Log out ……..……...…….20
Figure 4.6. UC – Register …….……...…….21
Figure 4.7. UC – Report an issue …….….22
Figure 4.8. UC – View issue details ……..……...…….23
Figure 4.9. UC – Chat with institutions ……23
Figure 4.10. UC – View facility details …….……...23
Figure 4.11. UC – View institution details ……..…….24
Figure 4.12. UC – View event details …….……..25
Figure 4.13. UC – Add/remove from favorites …….25
Figure 4.14. UC – View favorite events ……..…….25
Figure 4.15. UC – Create event …….……..…….26
Figure 4.16. UC – Delete own event …….26
Figure 4.17. UC – Read news……..…….….27
Figure 4.18. UC – View alert …….……..…….27
Figure 4.19. UC – Change name ……..….28
Figure 4.20. UC – Change password ……..……..28
Figure 4.21. UC – Delete account…….29
Figure 4.22. UC – View issue ……...30
Figure 4.23. UC – Assign issue ……..……..30
Figure 4.24. UC – Reassign issue …….……..…….…….30
Figure 4.25. UC – Share issue ……..31
Figure 4.26. UC – Chat …….31
Figure 4.27. UC – Mark issue as solved …….…….…….32
Figure 4.28. UC – Delete issue ……..……...…....32
Figure 4.29. UC – View posted news or alerts ……..……...….33
Figure 4.30. UC – Post news or send alerts …….…….33
Figure 4.31. UC – Delete news or alerts ……..…….34
Figure 4.32. UC – Create employee account…….35
Figure 4.33. UC – Update institution details …….……..…….35
Figure 4.34. UC – View analytics …….35
Figure 4.35. UC – View entity……..……...…….36
Figure 4.36. UC – Create entity…….……...…….37
Figure 4.37. UC – Update entity ……..…….37
Figure 4.38. UC – Delete entity ……..…….….38
Figure 4.39. UC – Approve event ……..……..…….38
Figure 4.40. UC – View analytics (admin) ……..…….38
Figure 4.41. UC – Conceptual architecture ……..…...…….39

Figure 5.1. System architecture……...……..42
Figure 5.2. Application server packages……....……..43
Figure 5.3. Class diagram of the issue package……...……..44
Figure 5.4. Database diagram……...……46
Figure 5.5. Administrator menu……...……...……..48
Figure 5.6. Institution employee menu……..……...……...……..48
Figure 5.7. Web interface routes……...…...…...……..49
Figure 5.8. Web client services.……...……...……..50
Figure 5.9. Mobile client structure……...……...……..52
Figure 5.10. Mobile client pages and navigation……..……...……..53

Figure 6.1. Citizen reports issue....……...……...……..57
Figure 6.2. Employee views issue details…..…...……...……..58
Figure 6.3. Details updates after employee assigns issue.……..……..…...……..58
Figure 6.4. Employee shares issue with another institution...……...……...……..58
Figure 6.5. Issue location and involved institutions.……...……..59
Figure 6.6. Issue details and chat on institution employee page and user device….....59
Figure 6.7. Issue displayed as solved…….....60

Figure 7.1. Log in and sign up pages……...……...……..62
Figure 7.2. Home, alert and profile pages…..…...……...……..62
Figure 7.3. Report issue page, camera and location page.……..……..…...……..63
Figure 7.4. Issues, issue details and chat pages……....……...……...……..63
Figure 7.5. Facilities, institutions and news pages...……...……..64
Figure 7.6. Facility details, institution details and news details pages…….....64
Figure 7.7. Events, event details and create event pages…...…….....65
Figure 7.8. Institution dashboard…….……..……..…...……..65
Figure 7.9. Institution issues page……....……...……...……..66
Figure 7.10. Institution issue details page……...……...……..66
Figure 7.11. Institution issue details page……..…….....66
Figure 7.12. Create news and alerts……...…….....67
Figure 7.13. Administrator dashboard….……....……...……...……..67
Figure 7.14. Users page……...……...……..67
Figure 7.15. Facilities page……...…….....68
Figure 7.16. Events page……..…….…….....68
Figure 7.17. Issue details……..…….…….....68

Appendix 2 – List of tables

Table 3.1. Comparative analysis……...11

Table 4.1. Problem statement……13
Table 4.2. Product position statement statement……...……13
Table 4.3. Summary of stakeholders..……...……14
Table 4.3. Summary of user needs….……...……15