

**BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA**  
**FACULTY OF MATHEMATICS AND COMPUTER**  
**SCIENCE**  
**SPECIALIZATION [Computer Science in English]**

## **DIPLOMA THESIS**

### **Neighbourhood app**

**Supervisor**  
**[Emilia Pop]**

*Author*  
*[Dubău Horea-Filip]*

2023



---

## ABSTRACT

---

This paper investigates the utilization of graph databases and container technologies in social applications. It examines the benefits of graph databases for establishing connections and performing rapid queries in social networking scenarios. It also highlights the role of container technologies like Docker in enhancing scalability and portability in social applications. The paper presents the approach to designing a social networking app that utilizes Spring Data Neo4j as the backend, providing a standardized way to interact with the graph database. On the front-end side, the app is developed using Flutter, a versatile and multiplatform framework that ensures a seamless and consistent user experience across different devices and platforms. The implemented features within the social networking app encompass various functionalities aimed at fostering meaningful connections and interactions among users. These features include the ability to create friend dependencies, enabling users to establish and maintain connections with others on the platform. Users can also join groups based on shared interests, facilitating community-building and engagement. Additionally, the app allows users to create posts and comments, promoting user-generated content and facilitating conversations within the social network. This research contributes to advancing the understanding of graph databases and developing innovative social applications.

**This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.**

[Dubău Horea-Filip]

A handwritten signature in black ink, consisting of a stylized, cursive script that appears to be 'Dubău Horea-Filip'.

# Contents

<b>1</b>	<b>Inroduction</b>	<b>1</b>
<b>2</b>	<b>related work</b>	<b>4</b>
2.1	NextDoor . . . . .	4
2.2	Meetup . . . . .	6
2.3	Neighborland . . . . .	6
2.4	Ring . . . . .	7
<b>3</b>	<b>suporting tools and technologies</b>	<b>8</b>
3.1	Neo4j . . . . .	8
3.2	Spring Data . . . . .	10
3.3	Flutter . . . . .	11
3.4	Docker . . . . .	12
<b>4</b>	<b>Requirement Definition Specification (RDS)</b>	<b>16</b>
4.0.1	Requirements . . . . .	16
4.0.2	Definitions . . . . .	18
4.0.3	Specifications . . . . .	19
<b>5</b>	<b>The Solution</b>	<b>21</b>
5.0.1	Design and implementation . . . . .	21
5.0.2	Front end . . . . .	26
5.0.3	Back end . . . . .	29
5.0.4	Testing and validation . . . . .	32
<b>6</b>	<b>Conclusions</b>	<b>34</b>
6.1	Results . . . . .	34
6.2	future work . . . . .	35
6.3	conclusion . . . . .	36
	<b>Bibliography</b>	<b>39</b>

# Chapter 1

## Introduction

One of the primary goals of a neighbourhood app is to foster a sense of community among its residents. To achieve this, the app would need to provide a range of features that encourage engagement, collaboration, and communication between neighbours. A feature of the app would be the ability to organize and promote local events and activities, such as block parties, yard sales, or community cleanups. This could be done through a centralized calendar that residents can access and contribute to, ensuring that everyone is aware of upcoming events and opportunities to get involved. The app could also include a directory of local businesses and services, allowing residents to support and connect with their local economy. This would help to foster a sense of community pride and encourage residents to invest in their neighbourhood. To further encourage engagement and communication, the app could also feature a feedback mechanism where residents can provide feedback on local issues or suggest ideas for improvement. This feedback could be channelled to local government officials or neighbourhood associations, ensuring that residents' voices are heard and their concerns are addressed. [13]

Different methods, such as neighbourhood forums, community-based apps, and social networking apps, aim to unite individuals living in the same area, improving communication and collaboration for a better quality of life. Related work includes apps like Nextdoor, Meetup, Neighborland, and Ring, each with distinctive features and goals to boost engagement and build stronger, safer, and more connected communities.

When there is a need to establish connections between data and perform rapid queries on these connections, graph databases prove to be advantageous in scenarios such as social networking [2], recommendation engines, and fraud detection. A different way to represent social network data is by visualizing it as a graph. This approach is well-suited for converting data into a graph format. Neo4j is a database

specifically designed to cater to the needs of data analysis, graph storage, and intelligent applications. It enables direct access to data, efficient data management, and graph visualization capabilities for social media channels and digital marketing efforts on social platforms.

Based on my application's requirements, I have decided to use Spring Data Neo4j as the back-end technology. Spring Data Neo4j [9] is a powerful and efficient framework that provides a high level of abstraction and simplifies the process of working with Neo4j databases. It allows me to define repository interfaces that are automatically implemented by Spring, providing a standardized way to interact with the database. Additionally, Spring Data Neo4j integrates seamlessly with Spring, making it easy to incorporate into my existing Spring-based application. Overall, I believe that Spring Data Neo4j will provide a solid foundation for my back-end and help me build a robust and scalable application.

In the context of developing a neighbourhood app using Flutter, the user interface and design are key considerations. By focusing on intuitive navigation, visually appealing layouts, and responsive design, the app can provide a seamless and enjoyable user experience. Flutter's widget library enables developers to create visually pleasing interfaces, while the Navigator widget facilitates easy navigation between screens. Responsive design ensures the app adapts to different screen sizes, and incorporating animations and transitions adds interactivity. By prioritizing simplicity and clarity in design principles, the app can offer a user-friendly interface that enhances engagement and usability.

Privacy and security are crucial considerations when developing a neighbourhood app. To protect users' personal information and ensure the confidentiality of their interactions, it is important to implement security measures. This includes using encryption protocols, secure authentication mechanisms, and data access controls. By employing the BCryptPasswordEncoder for password encryption and employing proper authentication and authorization mechanisms, we can enhance the security of user credentials. Regularly updating software components, following secure coding practices, and using secure communication protocols further bolster privacy and security.

Testing for the Spring Data project involves using a Neo4j graph database, focusing on unit testing for Neo4j repositories to test specific functions and functionalities provided by Spring Data. Additionally, manual testing is employed for front-end validations within the application.

In conclusion, the neighbourhood app aims to foster community engagement and communication. Features such as event organization, local business directories, and posting mechanisms encourage active participation. Utilizing Spring Data Neo4j and Flutter ensures efficient data management and a seamless user experience. Emphasizing privacy and security through encryption and adherence to regulations instils trust. Thorough testing ensures the app's reliability, ultimately connecting residents and enhancing community bonds.

# Chapter 2

## related work

Various works are linked to or comparable to our planned neighbourhood app in that they focus on connecting people in the same area. These consist of neighbourhood forums, community-based apps, and social networking apps.

Neighbourhood forums are online discussion platforms where people living in the same area can communicate with each other. These forums allow residents to discuss local issues, share information, and organize events. Some examples of popular neighbourhood forums include Nextdoor, Neighborhood Link, and Front Porch Forum.

Community-based apps, on the other hand, are mobile applications designed to connect people who live in the same area. These apps usually offer features such as local news, event calendars, and business directories. Some examples of community-based apps include Patch, LocalCircles, and MyCoop.

Social networking apps, such as Facebook and Twitter, can also be used to connect people in the same area. These platforms allow users to create and join groups based on shared interests, including location-based groups. Users can share information, post updates, and organize events within these groups.

Overall, these various works serve a common goal of fostering a sense of community and connectedness among people who live in the same area. By providing platforms for communication and collaboration, these tools can help create a stronger sense of belonging and improve the overall quality of life for residents.

### 2.1 NextDoor

Nextdoor is a social app that is used to connect people who live in the same community or neighbourhood. It gives a platform for people to share news, events, announcements, and updates about their neighbourhood with the "Neighborhood News Feed." This lets users stay informed about what's happening in their area.



They can engage with their neighbours by commenting on and sharing posts. Nextdoor also can create and join groups based on interests, activities, or specific topics. For example, users can join groups about local gardening, book clubs, or crime prevention. This allows people to connect with easy to people of the same interest and concerns. Thus building stronger relationships within the community. Nextdoor also has a feature that lets users report and receive alerts. 2.1

These alerts can be about crime and safety issues in the vicinity. This helps users to be informed about safety risks in their area. They can take precautions to help themselves and their community to be safer. 2.2 Besides, users can recommend and review local businesses in their area. Such examples are restaurants, shops, and service providers. This feature is particularly useful for users who are new to the area. These recommendations can also come from people they trust. This app also has an option for selling/buying. This is a great way for users to declutter their homes or to find new items without having to leave their neighbourhood. Nextdoor allows users to create and promote events. Such examples are community gatherings, yard sales, or charity fundraisers. This helps users to connect with their neighbours and to build a stronger sense of community in their area.

Finally, users can send private messages to each other to discuss community issues, ask for advice, or coordinate events.

Overall, Nextdoor is made to help people connect with their neighbors and to build stronger communities.

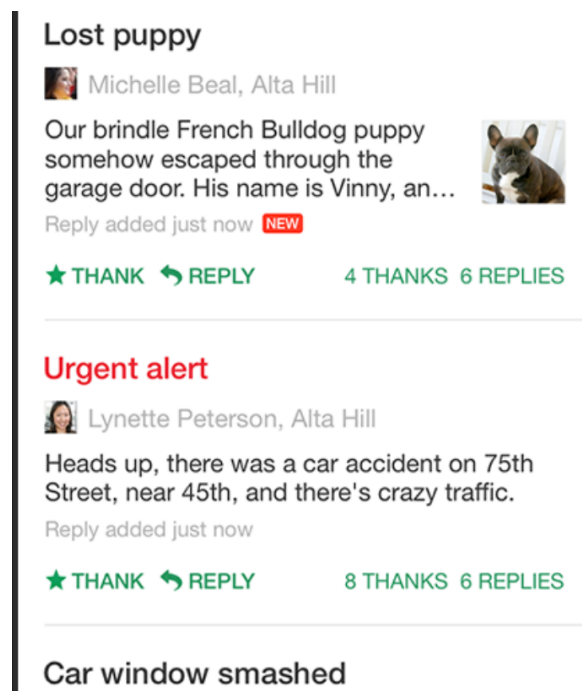


Figure 2.1: Reports in NextDoor

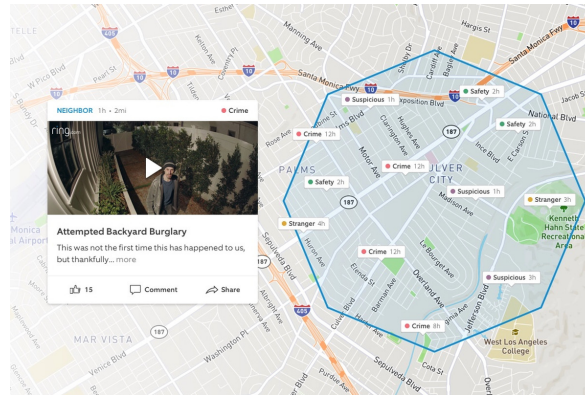


Figure 2.2: Reports in NextDoor

## 2.2 Meetup

Meetup is an app and website that lets you connect with people with matching interests and pursuits as you. You can use it to explore groups of like-minded individuals near you and join them. Once you find a group you like, you can RSVP to attend their events and meetups. With the app, it is easier than ever to find groups based on your location, interests, and schedule. So, if you're interested to discover new activities, making new friends, or networking with professionals, Meetup can help you achieve that.

Meetups it is used to find new exciting activities, have a chance at meeting new friends and associate with users who share your passions and hobbies. It is also a great way to network and connect with experts in your field. Therefore Meetup can be used to explore new job opportunities.

## 2.3 Neighborland

Neighborland is an app that encourages engagement and allows residents to transmit their ideas, concerns, and feedback to local government officials. The platform has functions for users to discuss and prioritize issues that count the most to them, and team up with other users to find a solution. The Neighborland platform is designed to be easily accessible to all users, regardless of their computer smarts. Citizens can share their ideas and feedback through the website or mobile app, and can also vote and comment on the ideas submitted by others.

Regional government administrators and community organizations can use Neighborland to get information from locals and engage with them on essential problems. The platform provides tools for officials to follow and respond to resident feedback, and to prioritize issues based on community input. Neighborland is currently used by a lot of united states cities, including New Orleans, Detroit, and the Knight Foun-

dation. The platform has been praised for its ability to increase civic engagement and empower residents to take an active role in shaping their communities.

## **2.4 Ring**

Another app that can be used to provide communication between neighbours is Ring. Rings' primary objective is to monitor and view live video streams from cameras. It has motion detection and sends alerts. It can also do 2-way communication for visitors. To help residents feel more secure the ring app also has a feature that allows the rewinding of the camera's recorded feed. It may so happen that the recorded video provides crucial information that helps the police in some cases. One of the many features of the Ring app is named "Neighbors" with which users can be informed and inform others of real-time events/crime/safety issues. They can post updated photos videos, and make posts about missing pets and local incidents. Other ring users in the area can interact with the posts themselves. The main objective of this feature is to keep the residents informed.

# Chapter 3

## supporting tools and technologies

### 3.1 Neo4j

Neo4j [6] is a graph database management system. It is appropriate for applications that demand complicated data connections and traversals since it is built to hold, manage, and query massive volumes of data in a graph structure. Neo4j stores data in nodes and relationships rather than tables, as do traditional relational databases. Cypher, a query language offered by Neo4j, is created primarily to work with graph data. Complex graph queries can be run using Cypher to access and modify database data. Numerous applications, such as social networking, recommendation systems, fraud detection, and network management, employ Neo4j. It is widely used to store and analyze intricate biological data in the domains of bioinformatics and genomics.

When there is a need to establish connections between data and perform rapid queries on these connections, graph databases prove to be advantageous in scenarios such as social networking, recommendation engines, and fraud detection. A different way to represent social network data is by visualizing it as a graph using a knowledge representation structure. This approach is well-suited for converting data into a graph format. To illustrate this process, the figure below 3.1 demonstrates how data can be restructured as a graph. The basic steps involve converting entities into nodes, which can represent different types such as people, organizations, and events. This deviates from the typical graph model where nodes are of one type. Edges in the graph correspond to relationships between entities, with directed edges representing relations such as "Knows (Bob, Fred)." If a relation is bidirectional, an undirected edge can be used between nodes. Finally, attributes of entities are converted into attributes on nodes, and attributes on edges are also possible in both models. This allows for additional information to be included, such as

evidence or the time of a relationship between two entities.

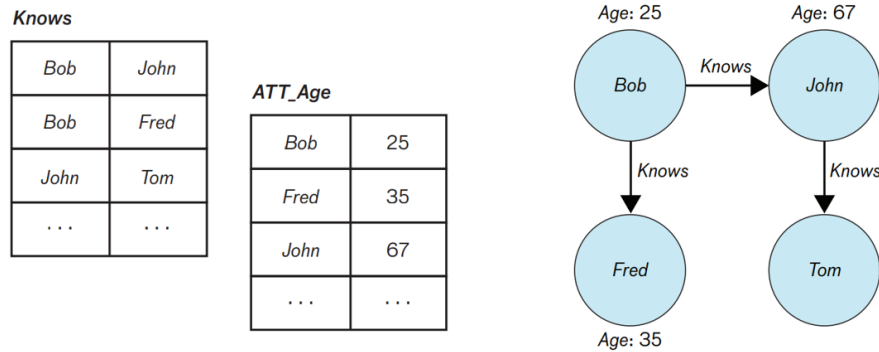


Figure 3.1: Representation of graph and relational relations [2]

Neo4j [8] is a database specifically designed for data analysis, graph storage, and intelligent applications. It offers direct access, data management, and graph visualization capabilities for social channels and digital marketing on social platforms. Meanwhile, relational databases are widely used in organizations for inventory, financial, and employee record-keeping, powering various business applications like billing systems and CRMs. Therefore, the relational database is a powerful information management tool for business. The nodes reflect entities like persons, categories, things, an item, or a piece of data. Relationships in Neo4j represent the connections between nodes and how they support graphical results. The model structure of nodes and relationships enables digital specialists to create various scenarios, from network devices to road systems, while enhancing the storage of content, supporting network management, and optimizing access and data management. Neo4j has well-defined features and characterizations that determine its overall performance and operational efficiency.

The relational database can provide a huge source of social networks. However, storing and representing the data of the social network the relational databases are not suitable. On the other hand, graph databases such as Neo4j are efficient in representing and storing such data. Graph databases provide an effective approach for extracting the social network from relational.[6] The approach provides an effective way to transform the relational database into a graph database. There is a research paper that presented a review analysis of how to handle big data for social networks such as Facebook, and Twitter. Modern society can benefit from the use of graph databases like NOSQL and Neo4j, which allow for the discovery of connections and relationships based on various activities. This technology can be

used to create unique and innovative social networks, as well as integrate existing social graphs into enterprise applications. Unlike traditional relational databases, social media networks inherently form graphs, eliminating the need for conversion. Graph databases are well-suited to handle complex and scalable data systems, making them capable of managing the growing connectivity of data and systems.

Graph databases have a wide range of applications in social networks, particularly as there is an increasing need for structured data. In social channels, there is often an abundance of redundant and overlapping data, which can be efficiently managed using either a relational or Neo4J database. By utilizing these databases, it becomes possible to effectively monitor data related to users, customers, accounts, and products in various formats and degrees. As a result, the scope of graph databases' application is broadened. The Neo4j database is designed to retain ACID transactions even after they have been completed, ensuring that they are stored consistently and reliably to support continuous business operations. This feature is particularly useful for organizations and enterprises that work on timed projects or projects involving social networks. In contrast, a relational database requires transactional support to store transactions, as it is not designed to handle such features. As a result, the Neo4j platform is more reliable and efficient for such businesses compared to the relational database.

## **3.2 Spring Data**

An open-source project called Spring [1] Data offers a collection of abstractions and assistant tools to make it more comfortable to create data access layers for Spring-based applications. It offers a uniform, high-level API that can be used with many kinds of databases and data stores. Thus it seeks to reduce the amount of boilerplate code developers must write. Developers may focus on business logic and application functionality instead of slight database interactions thanks to Spring Data.

One of Spring Data's key elements is support for a variety of database technologies, including SQL-based databases, NoSQL databases, and even file-based storage systems. This means that programmers can access an assortment of capabilities, like automatic pagination, sorting, and querying, by using a single API to deal with many data sources.

Spring Data supports declarative programming using annotations and interfaces, which is a significant benefit. Developers can define queries and associations be-

tween entities using annotations. Spring Data will automatically create the required database queries at runtime. As a result, it is easy to produce clear, readable code that is less inclined to errors and is simpler to maintain.

Spring Data Neo4j [9] is a module within the Spring Data family that provides integration with the Neo4j graph database. It offers an object-graph mapping framework that allows developers to interact with Neo4j using a domain model. This approach is particularly useful for building applications that rely heavily on graph data, as it simplifies the process of interacting with the database and eliminates the need for low-level data access code. With Spring Data Neo4j, developers can use a wide range of technologies and tools to build graph-based applications, including Spring Framework, Java Persistence API (JPA), and Java Persistence Query Language (JPQL). By offering support for Neo4j's Cypher query language, developers can conveniently query and manipulate graph data using a syntax they already know.

Spring Data Neo4j offers developers the ability to embed and use Neo4j instances independently. This feature allows for easy configuration and deployment of the instances, which in turn allows for scaling applications and handling large amounts of graph data. The tool also supports Spring Data's common CRUD operations, pagination, sorting, and querying, making it easier for developers to work with Neo4j. Moreover, using Spring Data Neo4j enables the integration of applications with other technologies and data sources. It can work with different databases, including SQL-based or NoSQL databases when combined with other Spring Data modules. This flexibility makes it an adaptable tool that can be used for a wide range of applications, such as social networking platforms, recommendation engines, and fraud detection systems.

In summary, Spring Data Neo4j is a versatile and robust tool that developers often use for creating graph-based applications. It's simple to use, can integrate with other Spring Data modules, and supports various technologies, making it a favourite among developers who work with graph data.

### **3.3 Flutter**

Flutter [5] is a framework for mobile applications. It is created by Google and is open source. This framework allows users to create native applications for web or mobile from the same code.

Flutter's main language is Dart, a programming language also created by Google. Dart has an assortment of customizable widgets for creating engaging and responsive user interfaces. These widgets can be used to manipulate layouts, user input, and content like photos, text, and videos. Flutter has a feature called hot reload [14] . This lets developers see changes to their code in real-time as they construct them, without the need to restart the app. This causes the development process faster and more efficient. Flutter also provides many tools for managing the app. Some examples are states, networking, and animation. There is also a group of users who contribute to making packages for integration in Flutter. In short, Flutter is a powerful and very favoured framework thanks to its ease of use, flexibility, and ability to be cross-platformed.

Flutter is based on a widget tree architecture where everything is a widget. Widgets are the building blocks of a Flutter app and are used to define the app's user interface. Widgets in Flutter can be either stateful or stateless. Stateful widgets have a mutable state that can change over time. Meanwhile, stateless widgets are immutable and cannot change after they are built. Flutter supports asynchronous programming [11] using the `async/await` syntax. This allows developers to write nonblocking code that runs in the background without freezing the UI. Flutter has builtin support for unit testing and integration testing, which makes it easy to test your app and ensure that it works as expected.

## **3.4 Docker**

Docker is a platform that is available for free and is designed to help developers and administrators. Its use is to create, distribute, and run applications in a convenient and efficient manner.

It makes use of a technology named container. This allows applications to be put together with their dependencies in a standardized structure. Docker delivers various tools such as Docker Engine. This allows for easy runtime and packaging, and Docker Hub, a cloud service that facilitates the sharing of applications.

One of the main advantages of Docker is that it reduces costs by replacing standard virtual machines with lightweight containers. Docker containers can be easily deployed and managed and run on the same hardware more efficiently than virtual machines[12]. Docker also offers speed, portability, scalability, rapid delivery, and density, making it a favoured choice among designers.



Regarding performance, Docker has been compared to virtual machines and other container technologies. Studies have shown that Docker performs well in terms of speed, portability, and resource utilization. It can run applications faster, use resources more efficiently, and provide a consistent and reliable environment for development, testing, and production systems.

Yet, Docker also has some disadvantages. It does not provide complete virtualization like virtual machines. It relies on the host operating system's kernel. Docker currently only supports 64-bit local machines. It may require additional tools for Windows and Mac machines.

According to Joy (2015), Docker provides several advantages in terms of usability. Applications built inside containers are portable and can be moved around as a single unit. This is done without affecting performance or the container itself. This is because of Docker's standardized container format. Software teams do not need to worry about each other's tasks. Developers only take care of applications inside the container, and administrators only take care of server deployments with containers. Docker containers can run in every Linux system and can also be deployed to various cloud environments, data centres, and physical servers. The scalability of Docker containers makes it easy to adjust the scale from one to thousands and back to one if it is not needed. Containers are small, allowing for rapid build times and reduced times in testing, development, and deployment. As Docker containers do not use a hypervisor, available resources can be used more efficiently, resulting in higher density and better performance[15].

Virtualization has been a well-established concept for quite some time now. It is particularly used in the realm of cloud computing. It gained prominence with the widespread acceptance of Infrastructure as a Service (IaaS). It was used as a fundamental approach for structuring systems, allocating resources, and facilitating multi-tenancy. Virtualized resources play a pivotal role in addressing challenges through the core principles of cloud computing. Additionally, Figure 3.2 provides a visual representation of the virtual machine architecture, depicting its underlying structure and components.

At the core of the virtual machine architecture lies the hypervisor. It is positioned between the host and guest operating systems. Acting as a virtual platform, the hypervisor enables the management of many operating systems on a single server. It serves as an intermediary layer between the operating system and the CPU. Virtualization can be classified into two main segments: Para-Virtualization and Full

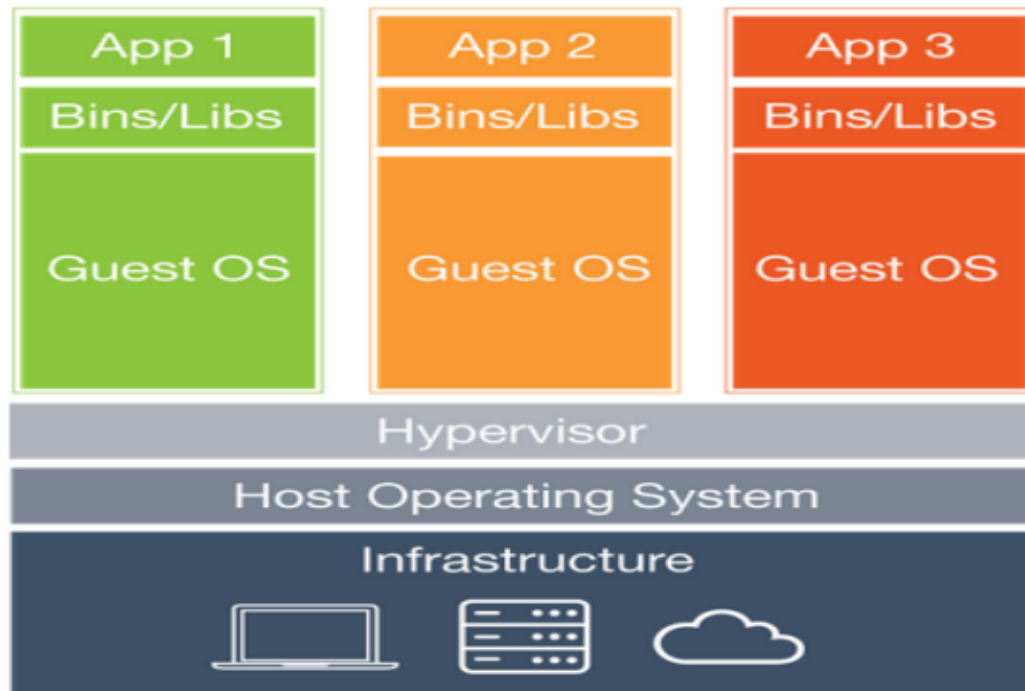


Figure 3.2: visual representation of the virtual machine architecture [15]

Virtualization. Figure 3.3 provides a summary of the Docker Container architecture, showcasing its components and their relationships.

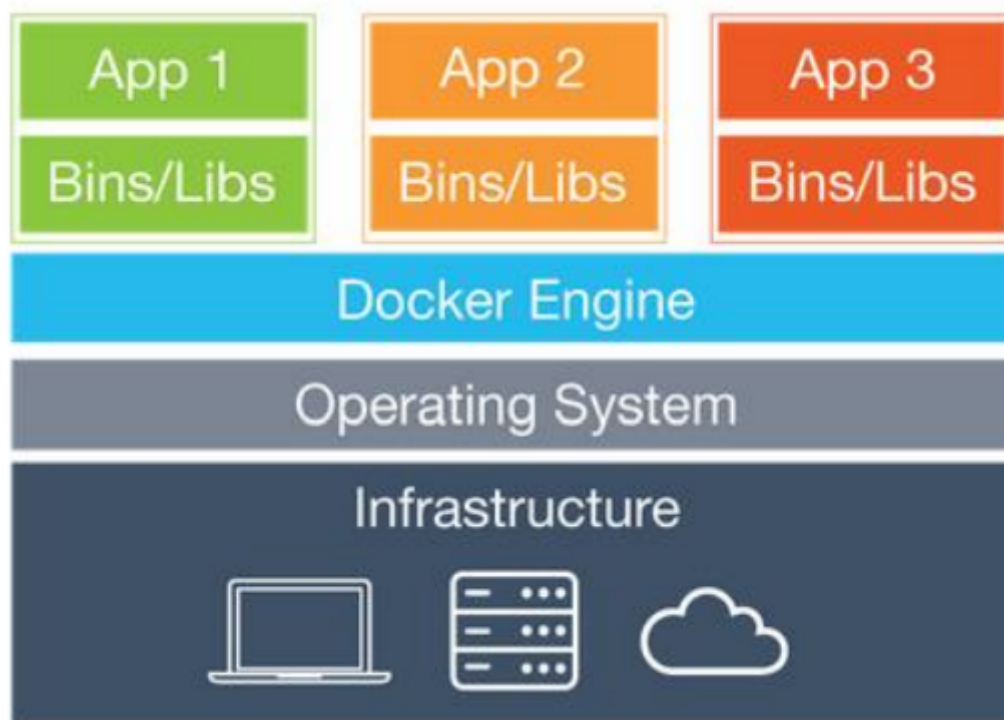


Figure 3.3: summary of the Docker Container architecture [15]

# Chapter 4

## Requirement Definition Specification (RDS)

### 4.0.1 Requirements

#### User Requirements

A neighbourhood app would have several requirements that we should keep in mind. One of the first requirements would be a secure method for logging in. Only secure authorised users should be able to have access to its features. [wu2004secure]

Another vital function of the app would be accessing the users' location. This will increase the user experience by giving personalized and relevant information. This is based on the given local area. Some examples are Nearby amenities, Real-time events, Safety alerts, and Community engagement. Push notifications would be another useful feature that the app could provide. The app should be able to inform users of suitable events and announcements. This should be done without entering the app for essential information.

Real-time messaging is another essential feature of any communication app. This enables the users to communicate instantly between friends, family or neighbours. This helps stay connected and up to date. This feature can furthermore promote more natural communication between users. This is because they can respond in real-time like real-life communication. This enables healthier connections between neighbours, as it enables them to communicate easier and more often. The app should have a social platform. This allows users to share their experiences, opinions and recommendations with others.

User profile customization is an essential feature of any social app. Letting users

personalize their profiles with their name, picture, interests, and bio can help them show who they are and find others who are like them. This customization feature can make users feel more invested in the app and more willing to engage with it. Furthermore, the app could also offer privacy settings. For example, users could control who can see their profile and what information is visible to other users. For example, users may choose to hide their profile picture or limit the visibility of their interests and bio. This could be visible only to their friends or selected groups.

### **Technical Requirements**

Scalability [4] is an important technical requirement for a neighbourhood app. This will ensure that the app can handle the growing number of users without affecting its performance.

Operating System Compatibility: To ensure that the app can be used by the largest number of possible users, it's essential to design it to work on the most common mobile operating systems, such as iOS and Android. By building the app using a single codebase, developers can avoid the need to create separate versions for each platform, which can be time-consuming and expensive. This approach allows developers to reach more users while also reducing development costs and time. In addition to iOS and Android, Flutter is also compatible with Windows, macOS, and Linux, which further expands its capabilities as a cross-platform development framework. With Flutter, developers can create desktop applications that can run on multiple operating systems, enabling them to reach even more users and expand their app's potential audience.

Storage capacity is an essential technical requirement for any app, including a neighbourhood app. It refers to the amount of data that the app can store on a device, server, or cloud storage. A neighbourhood app may need to store various types of data such as user profiles, images, videos, messages, and other user-generated content. Therefore, it's crucial to ensure that the app has adequate storage capacity to handle the amount of data generated by its users.

The storage capacity of a neighbourhood app depends on several factors such as the type and size of the data being stored, the app's usage patterns, and the number of users. If the app has a large user base and generates a substantial amount of data, it may need to use cloud storage to store data efficiently. Cloud storage solutions such as Amazon S3, Google Cloud Storage, or Microsoft Azure provide reliable, scalable, and cost-effective storage solutions for apps.

When designing a neighbourhood app, it's crucial to consider the storage capacity required for each type of data that the app will store. For example, user profiles and other user-generated content may require more storage capacity than app-generated data such as neighbourhood news or events. The app's architecture should be designed to optimize data storage and retrieval and to ensure efficient use of storage resources.

### **Design Requirements**

An effective neighbourhood app is designed with a focus on providing a visually attractive and intuitive user experience that captures the unique personality and identity of the community it serves. This involves careful consideration of various design elements such as colour schemes, typography, layout, user interface, images, graphics, branding, and accessibility. By incorporating these features, the app can deliver a user-friendly interface that invites users to explore and engage with its functionalities. User profile customization is a key feature of many online platforms and applications. This allows users to personalize their experience by adding their name, nickname, photo, and other information that they wish to share with others. By customizing their profile, users can create a unique identity that represents their personality and interests. This information can also help other users connect with them and create a sense of community within the platform. Additionally, user profile customization can provide users with a sense of ownership over their account and content, which can increase their overall engagement with the platform.

#### **4.0.2 Definitions**

A neighbourhood refers to a community of people who live in a particular geographic region and communicate shared interests, values, and experiences. A mobile application, or mobile app, is a software application designed to run on mobile devices such as smartphones and tablets. The operating system is the software that manages the hardware and software resources of a mobile device, allowing other software applications to run. The user interface (UI) is the visual and interactive components of a mobile app that enable users to interact with the app and access its features and services. Scalability refers to the ability of a mobile app to handle a large number of users and data and to accommodate future growth. User experience (UX) is the overall experience that a user has when using a mobile app, including usability, functionality, and aesthetics. The backend is the server-side component of a mobile app that manages data storage, processing, and communication with the front end (UI). An Application Programming Interface (API) [3] is a set of protocols,

routines, and tools used for building software applications and enabling communication between different software systems. Security measures are taken to protect the mobile app and its users from unauthorized access, data breaches, and other security threats.

Docker is a platform for creating, distributing, and running applications using containers. Containers are the standardized structure that packages applications and their dependencies. This allows them to run across different environments. Docker Engine is a tool that allows for easy runtime and packaging of Docker containers. Docker Hub is a cloud service that facilitates the sharing of Docker applications. A virtual machine [7] is a virtualized operating system that runs on top of a host operating system. Hypervisor is a software layer that enables the management of many virtual machines on a single physical machine.

### 4.0.3 Specifications

In order to provide the most useful and up-to-date information to users, a neighbourhood app may need to integrate with external APIs (application programming interfaces) that provide data or functionality not available within the app itself. For example, an app that provides information on local weather conditions might integrate with a weather API to obtain real-time data on temperature, precipitation, and other meteorological factors.

Regarding Operating system compatibility the application is designed to run on both IOS and Android. The app would also be available for computer users. This is because Flutter has the architecture to make running applications from the same code that runs on multiple platforms.

Another helpful element that could be included in the neighbourhood app is a message board or panel. This is where users can communicate with each other and share information. This would allow community members to post questions, start discussions, and engage with each other on a vast range of topics related to the neighbourhood.

A community events calendar is an essential part of a neighbourhood app. This feature will allow users to stay informed about approaching events and activities in their region. With this feature, users can access a vast list of events varying from block parties and yard sales to holiday celebrations and local concerts. The calendar will provide users with clear information about each event, including the date, time,

location, and a short description. This allows users to plan their schedules accordingly and ensure they don't miss out on any exciting moments in their community. Further, the events calendar should be able to be updated by members, making sure that the information provided is accurate and up-to-date. Overall, the community events calendar feature helps to foster a sense of connection and engagement among residents. This feature will be encouraging them to participate in local activities and forge stronger relations with their neighbours.



# Chapter 5

## The Solution

### 5.0.1 Design and implementation

Throughout the development of my Spring Data project with a Neo4j graph database, I followed a well-structured process that allowed me to ensure the project met all requirements and was free of any major bugs or issues.

To begin, I identified the requirements for my project and defined the entity relationships and data model using a visual modelling tool. This provided a foundation for the code implementation and a better understanding of the data structure and relationships within the application.

As part of my project implementation, I used Docker to manage the deployment of the database. A Docker Compose file was used with the necessary configuration to run Neo4j. The file specifies the Neo4j image to be used, sets the hostname, and provides the authentication credentials. Additionally, the file maps the container ports to the host machine ports, allowing access to the Neo4j database from the application.

With this Docker Compose file, I was able to easily spin up the database container and integrate it with my application. The back-end functionalities of my application are dependent on this database, and I made sure to have the container running whenever I was testing the application or deploying it to a production environment.

After selecting Neo4j as the graph database for my project, I designed the database schema to fit the data model, using the Cypher query language to create custom functions that were not provided by Spring Data. This approach allowed me to validate the custom functions' behaviour and ensure that they produced the expected results.

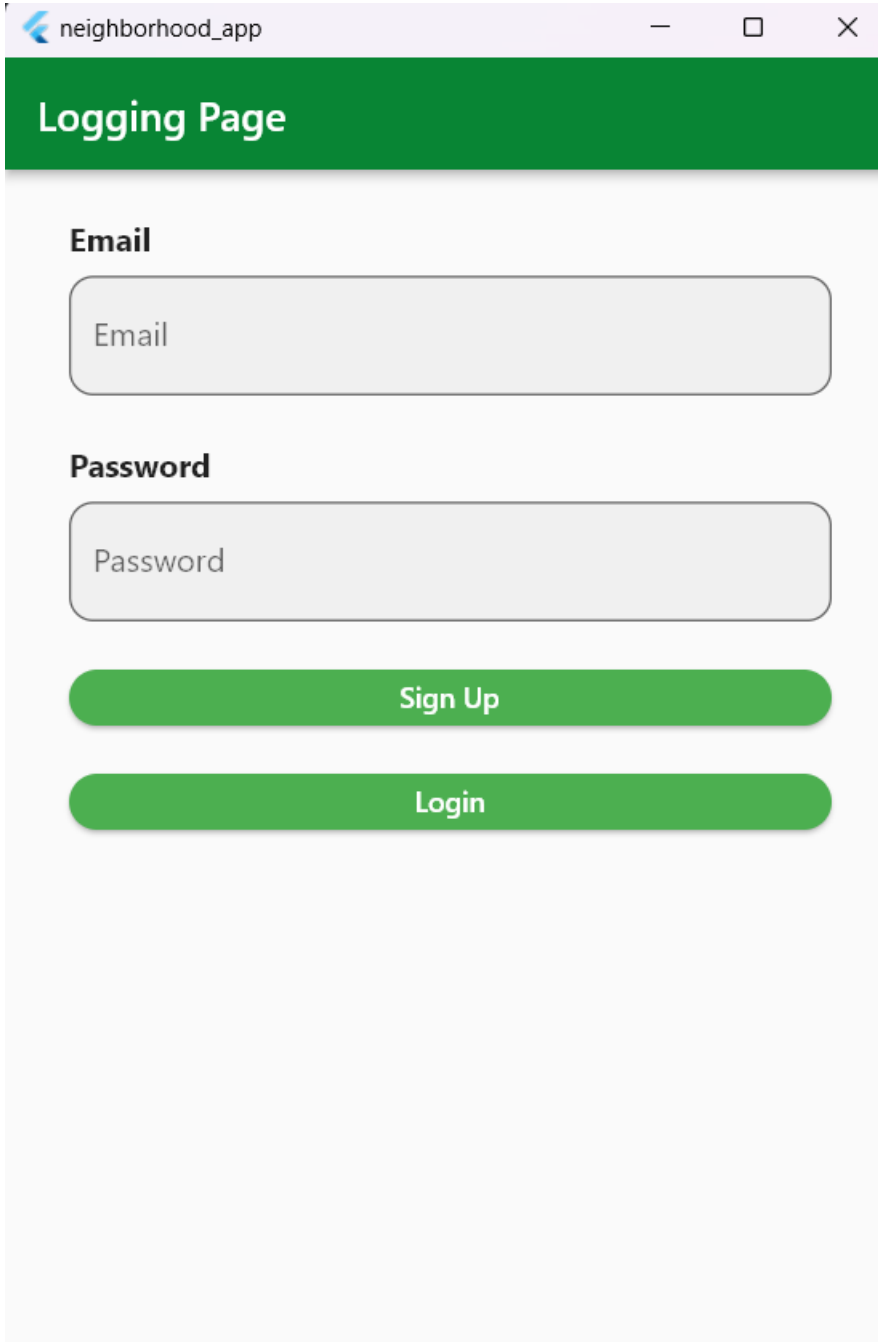
Next, I implemented the database schema and custom functions into my codebase using Spring Data Neo4j. To validate the functionality of the code, I created Unit Tests and identified and fixed any bugs or errors during testing.

In order to enable user authentication in my application, I first created the data model for user accounts, including necessary fields such as email and password. To implement the user authentication functionality, I decided to use Spring Security, a popular security framework for Spring-based applications. To start, I created login and sign-up pages for the users to access. I then developed the necessary back-end functionality to validate user credentials and manage user sessions. To ensure the security of user passwords in my project, a password encoder class that uses the BCryptPasswordEncoder from Spring Security was used. This encoder class hashes the user passwords before storing them in the Neo4j graph database, providing an extra layer of security to protect the user's sensitive data. By using this approach, we can be confident that the user passwords remain protected even if the database is compromised. Additionally, there are Unit Tests to validate the functionality of the password encoder class, ensuring that the passwords are properly hashed and can be correctly verified when needed.

### **mini-user manual for Function 1**

Here's a user manual for using the login and sign-in feature of the app. To get started, you need to launch the app either on your mobile device or computer. On the login page, you will be prompted to enter your email and password, and once you have entered your login details, click on the 'Login' button to submit your login details. If your login details are correct, you will be redirected to the app dashboard, where you can access all the features of the app. If the credentials are wrong a pop-up will info the issue. If you don't have an account, you can click on the 'Sign up' button on the home screen to create an account. On the sign-up page, you will be prompted to enter your name, email, and password. The password needs to be entered twice during the sign-up process for it to work properly. If it is not entered the same way both times, the sign-up will not be successful. Once you have entered your sign-up details, click on the 'Sign up' button to submit your details. If your sign-up details are correct, you will be redirected to the app dashboard, where you can access all the features of the app.

Creating the login and sign-up features for the front-end of an app involves multiple steps, starting with setting up the development environment by installing and configuring essential tools such as a code editor, Git, and a local server for testing. Then, a back-end API is created using a web framework, such as Spring Boot, that manages user authentication and stores user data in a database. Afterwards, two front-end pages are built with Flutter that includes input fields for email and password and buttons to submit forms. Subsequently, the front-end pages are linked



The image shows a web browser window with the title 'neighborhood\_app'. The page has a green header bar with the text 'Logging Page'. Below the header, there are two input fields: one labeled 'Email' and one labeled 'Password'. Both fields are light gray with rounded corners. Below the 'Password' field, there are two green buttons with white text: 'Sign Up' and 'Login'.

neighborhood\_app

## Logging Page

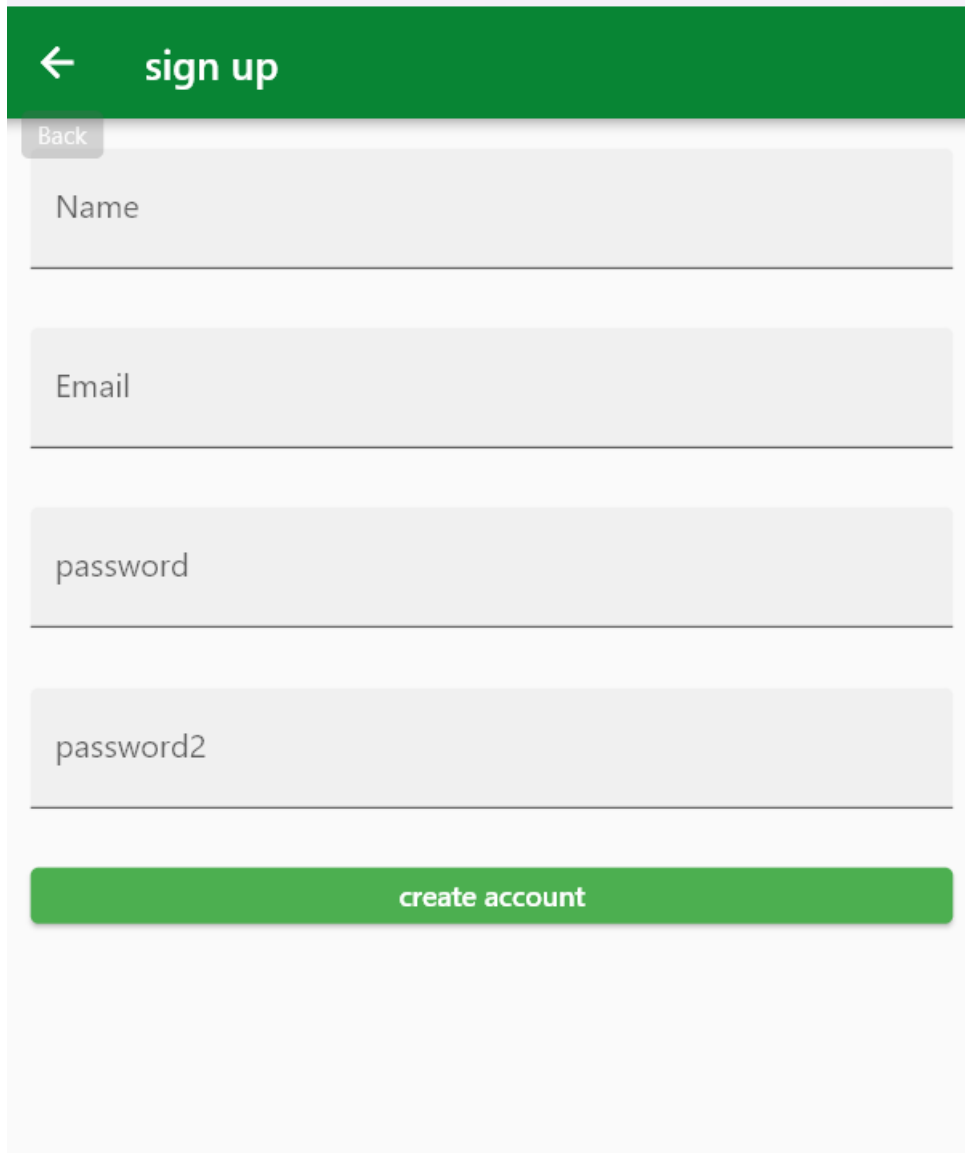
Email

Password

Sign Up

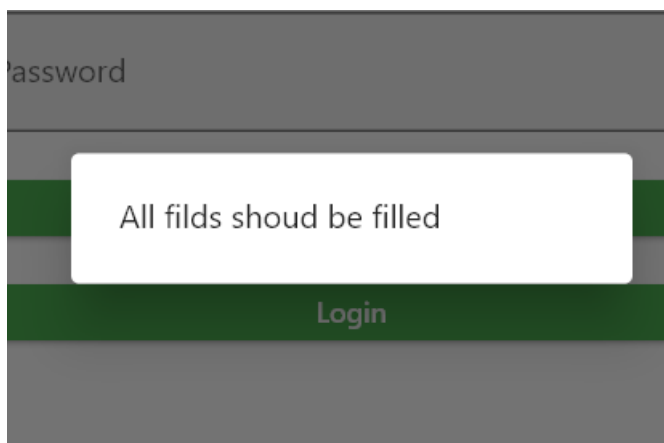
Login

Figure 5.1: Login page



The image shows a mobile app interface for a sign-up page. At the top is a green header bar with a white back arrow and the text "sign up". Below the header is a light gray background area containing four input fields: "Name", "Email", "password", and "password2". Each field has a thin gray border. Below the input fields is a green button with the text "create account".

Figure 5.2: Sign Up page



The image shows a mobile app interface for a login page. It features a dark gray background. At the top, there is a dark gray bar with the text "password" in white. Below this is a white rectangular box with the text "All filds shoud be filled" in gray. At the bottom, there is a green button with the text "Login" in white.

Figure 5.3: filds should be field

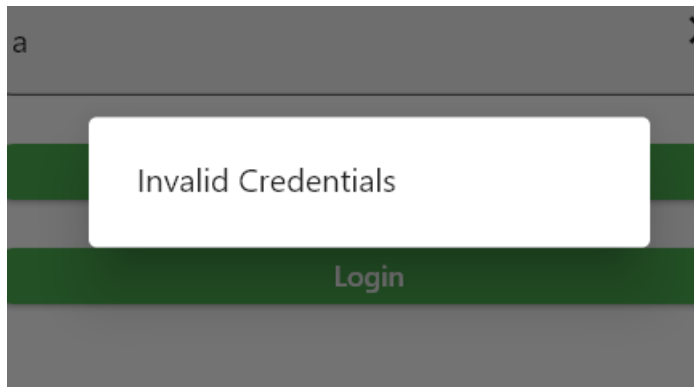


Figure 5.4: invalid data

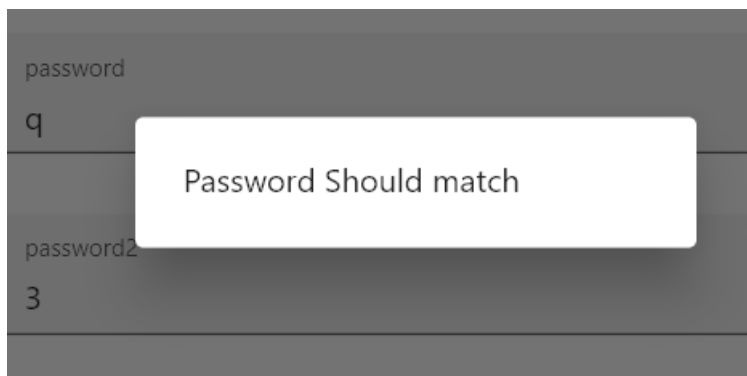


Figure 5.5: password mismatch

to the back-end API to enable the transmission of user data for authentication and storage purposes.

Overall, my process of designing and implementing my Spring Data project with a Neo4j graph database involved identifying requirements, designing the data model and database schema, implementing custom functions using Cypher, coding with Spring Data Neo4j, and testing and debugging the application. This iterative process allowed me to ensure that the project met all requirements and was free of any major bugs or issues.

### mini-user manual for Function 2

The post and comment feature in the neighbourhood app enhances community communication by providing users with the ability to share news, seek recommendations, and engage in conversations. To create a post, users can tap on the "Add Comment" button on the main screen, which takes them to a dedicated page. From there, they can select from a dropdown the appropriate group for their post and enter their comment in the provided text field 5.6. Additionally, users can engage in conversations by replying to specific posts. By tapping the message button from a post, they can open a new page to type and send their response as seen in figure

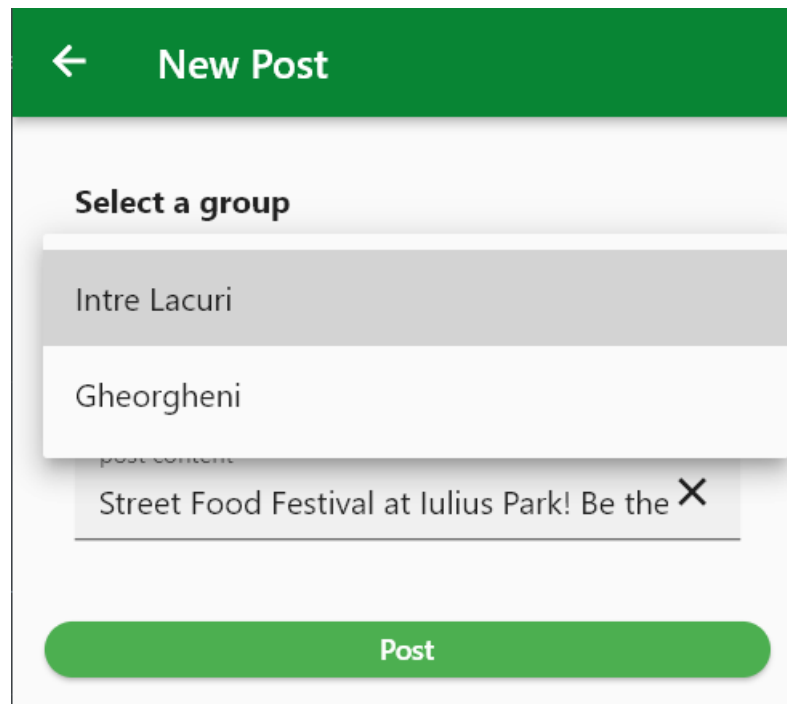


Figure 5.6: new post

5.8. This interactive functionality promotes open dialogue and facilitates meaningful interactions among community members. Furthermore, users can manage their comments by long-pressing on a comment from the main screen, allowing them to delete their comments if needed. This will open a dialogue for confirmation so no misclick will occur 5.7.

### 5.0.2 Front end

During the development process, I implemented several key features to create a comprehensive application. The primary objective of this application is to assist users, in finding and connecting with potential friends and groups. To achieve this, I incorporated various functionalities. Firstly, I designed a Group Membership Management feature that enables me to view my existing groups and explore potential groups that I can join. The user interface presents a detailed list of user groups, including the group name, and provides an option to remove myself from specific groups. Additionally, it displays potential groups, showcasing the group name, the number of common friends, and an option to join the group. Secondly, I incorporated a Friendship Management functionality that presents me with a list of potential friends based on mutual group affiliations. This feature allows me to view potential friends, including their names and the number of mutual groups we share. An "add friend" option is available for each potential friend, facilitating the establishment of connections. Furthermore, I integrated the application with a backend

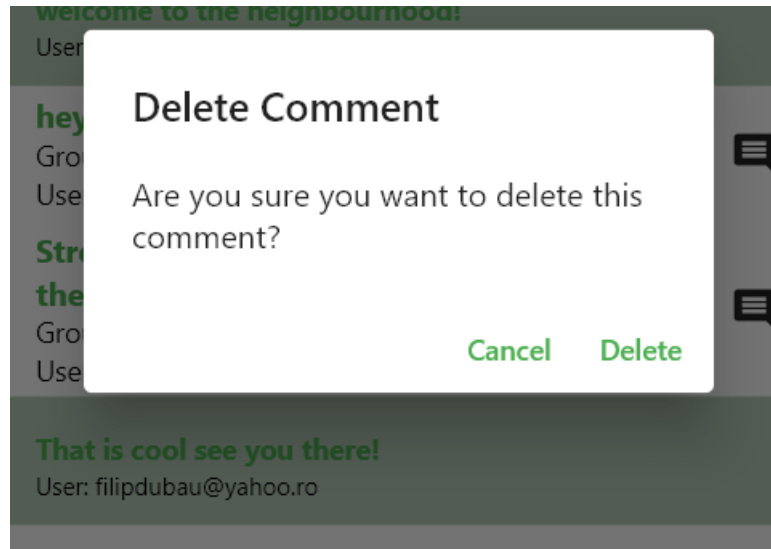


Figure 5.7: delete comment

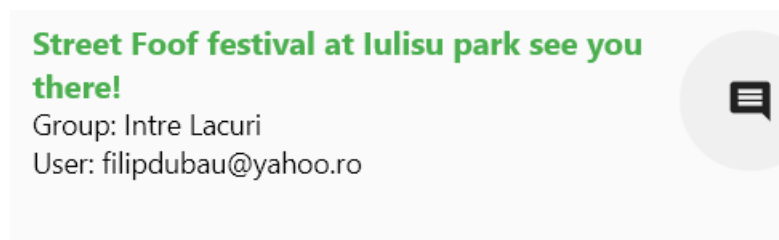


Figure 5.8: how to make a comment

API, enabling seamless communication to retrieve data related to groups, potential groups, potential friends, and user groups. I ensured that the application incorporates error handling mechanisms to gracefully manage unsuccessful API requests. In the event of an error, informative dialog boxes are displayed to provide me with appropriate feedback and guide me on the necessary steps to proceed effectively.

These features were implemented using the Flutter framework, which offers a rich set of tools and widgets for building cross-platform mobile applications. The UI elements were designed to provide a user-friendly experience, presenting information clearly and allowing for intuitive interactions.

During the development of the application, a few challenges and errors were encountered that required attention and resolution. One of the initial hurdles involved integrating the application with the backend API. Ensuring proper communication between the frontend and backend systems, handling different types of responses, and managing errors that arose during API requests required careful debugging and troubleshooting. Additionally, ensuring consistent and reliable data retrieval from the API endpoints required thorough testing and validation. Another challenge involved designing the user interface to effectively present the information related to groups and potential friends. Striking a balance between displaying sufficient details and maintaining a clean and intuitive design proved to be a delicate task. Iterative design adjustments were made to improve the overall user experience and ensure that the information was presented in a clear and visually appealing manner.

The project successfully achieved its goals by implementing the essential functionalities of friend requests and group interactions. The mobile application allowed users to connect with potential friends and extend their social circles. The server-side part ensured data integrity and responsiveness, enabling users to receive timely updates.

Throughout the development process, several modifications and improvements were made to the codebase in order to address various issues and enhance functionality. Another problem arose when integrating the comment fetching process into the `PostWidget` class. Initially, the API call was made in the `initState()` method, resulting in an error due to the asynchronous nature of the operation. To overcome this, the API call was moved to a separate method, `fetchComments()`, which was then invoked within the `initState()` method.

To improve the readability and distinguishability of the post and comment sections within the `PostWidget`, changes were made to the UI presentation. The post information was displayed within a `ListTile` using a bold font for the group name and a regular font for the post content. For the comments section, each comment was also presented as a `ListTile`, with the comment displayed in the primary colour and the user's email in a secondary colour. Furthermore, a global constant for the



API base URL was introduced to enable easier modification and maintainability of the codebase. This allowed for seamless updates to the API URL across multiple files and folders.

Also, a long-press gesture functionality was implemented within the 'PostWidget' class. By incorporating this feature, users can now execute a long press on comments, initiating a verification dialogue that lets them delete the specified comment.

To support the deletion of comments, I added a "deleteComment" method to the 'PostWidget' class. This method effectively sends a DELETE request to the server, thus removing a specific comment based on its unique identifier.

In terms of post-management, I implemented a filtering mechanism based on the associated groups within the 'PostListWidget' class. By extracting the available groups from the posts, users can now select or deselect specific groups using checkboxes. This filtering functionality ensures that only posts belonging to the selected groups are displayed.

To keep track of the selected groups, I maintained a list within the 'PostListWidget' class. This list is updated dynamically whenever a user checks or unchecks a group, providing real-time filtering of the displayed posts accordingly. Moreover, when no groups are selected, the system automatically shows all posts to ensure comprehensive visibility.

Lastly, I extended the versatility of the 'TextFieldWidgetController' class by introducing a new parameter, 'isVisible'. This parameter controls the visibility of the associated label within the text field. By toggling 'isVisible' between 'true' and 'false', the label can be displayed or hidden, offering customization options based on specific requirements.

### 5.0.3 Back end

Implementation involves using Spring Boot as the backend framework. The Spring Data Neo4j library facilitates the interaction with the Neo4j graph database. This enables the creation, retrieval, update, and deletion of friend relationships. The data model is designed to represent users as nodes and friendships as edges in the graph. The system provides endpoints to operations regarding users as friends. Some endpoints are adding friendships between users, and retrieving friends of a user. These operations are performed on graph-based queries. To make a new "Friend" relationship, in the backend, we have a function that verifies the existence of the relationship between users. If a "Friend" dependency does not exist, it creates a relationship between the users in the database. This dependency will be a 2-way relation having one from the first node to the second and vice versa. The function ensures that duplicate or conflicting friendships are not created.

Another important function is addressed at the retrieval of users belonging to a specific group in the database. Through the modification of the initial query, 'find-UsersInGroup', I was able to retrieve the desired results accurately.

During the implementation of query functions, I encountered dependency issues resulting in missing or null values for certain properties. To resolve this, I adjusted the query projections and ensured that only properties of the appropriate domain type were mapped or returned in the results. This modification successfully prevented warnings and null values in the retrieved data.

I then extended the functionality to find friends within a given group. The 'find-FriendsInGroup' query was designed to retrieve friends associated with the specified group. This function returned a list of 'Friend' objects containing their respective properties such as name, email, password, about, group list, and friends.

To address the requirement of identifying users in a group who are not friends with a particular user, we created the 'findNonFriendUsersInGroup' query. This query correctly identified users in the specified group who did not have a friendship connection with the provided user. The result was a list of 'Friend' objects containing their respective properties.

Throughout the discussion, query optimization was a key focus to enhance database performance. By utilizing Neo4j's query language (Cypher) and employing efficient filtering techniques such as relationship patterns, constraints, and the 'EXISTS' function, we were able to minimize the number of database operations and ensure optimal query execution. The 'Friend' model class was meticulously designed and constructed, encompassing crucial attributes such as 'id', 'name', 'email', 'password', and 'about'. Relationships between 'Friend' nodes were established through properties like 'friends' and 'groupList'. The class was equipped with appropriate getter and setter methods to facilitate data access and manipulation.

To facilitate the management of 'Group' nodes, the 'GroupRepo' interface was devised. This interface extended the 'Neo4jRepository' interface and performed as a repository for executing queries and performing operations on 'Group' data.

A significant feature successfully implemented in the application was the capability to handle friend requests efficiently. This functionality was realized through the creation of the 'addFriendshipByEmail' method, which facilitated the addition of friend requests between two individuals based on their email addresses. The method utilized the 'FriendRepository' to execute the necessary operations, leveraging the underlying database to establish a new relationship type between two nodes of type 'Friend'.

Upon invocation, the 'addFriendshipByEmail' method performed the following steps: Verification of Friend Existence: The method accessed the 'FriendRepository' to verify the existence of the specified friends by retrieving their corresponding

'Friend' objects from the database. **Addition of Friend Request:** Once the existence of both friends was confirmed, the method proceeded to add a friend request between them. This was achieved by creating a new relationship "FRIEND REQUEST" of a distinct type between the two 'Friend' nodes in the database, representing the friendship request. **Persistence of Data:** To ensure data consistency, the updated friend request information was persisted in the database. The 'FriendRepository' facilitated the storage of the modified 'Friend' objects, thereby reflecting the new friend request relationship in the underlying data store. These functionalities, executed through the utilization of the 'FriendRepository' and manipulation of relationships between 'Friend' nodes in the database, provided a robust framework for managing friend requests within the application.

The application allows users to create and store posts in the database. The 'Post' object consists of a 'User' (designating the author of the post), a 'Group' (indicating the group to which the post belongs), and the actual post content. The application utilizes the 'PostsRepo' repository, which extends the Neo4j repository, to perform database operations such as saving posts. The 'PostController' class handles the HTTP POST requests and maps them to the 'createPost' method, which receives a JSON request body containing the necessary information for creating a post. The implementation faced challenges in handling and mapping the complex object relationships within the graph database, as well as ensuring proper data validation and error handling during the process.

First, I needed to define the `CommentRepo` interface. This interface extends the `Neo4jRepository<Comment, Long>` provided by Spring Data Neo4j. Inside this interface, you can use Spring Data Neo4j annotations and queries to define the necessary methods for interacting with the comment data in the database. For example, you can use `@Query` annotations to define custom Cypher queries for creating, deleting, and retrieving comments. Ensure that the annotations and queries are correctly defined and mapped to the corresponding database entities and properties. Next, create the `CommentController` class, which will serve as the controller for handling REST API requests related to comments. Annotate this class with `@RestController` to indicate that it's a controller that handles REST API requests. Additionally, use `@RequestMapping("/api/comment")` to specify the base URL for the comment-related endpoints, ensuring that the correct URL mappings are defined for the desired functionalities.

For each functionality, such as creating a comment or retrieving all comments, I define a method in the `CommentController` class. Annotate each method with the appropriate HTTP method annotation (`@GetMapping`, `@PostMapping`, etc.) to specify the HTTP request method for accessing the functionality. Inside each method, implement the necessary logic by utilizing the `CommentRepo` interface to interact

with the database. Retrieve data, perform required operations, and handle any necessary validations or error handling. When handling the requests, make sure to return appropriate responses using the `ResponseEntity<T>` class. This class allows you to customize the HTTP response status code and body. Use `ResponseEntity.ok()` to return a 200 OK response along with the desired data when the operation is successful. If there are errors or invalid requests, use `ResponseEntity.badRequest()` to return a 400 Bad Request response with an appropriate error message.

### 5.0.4 Testing and validation

As I have been working on creating tests for my Spring Data project that uses a Neo4j graph database, I chose to focus on unit tests for my Neo4j repositories. Unit Testing [16] is a type of software testing where a section of code is isolated from the application and tested separately. The purpose is to validate each unit of the software code and make sure it behaves as expected. It is a WhiteBox testing technique performed by developers. The module tested is `BusinessLogicLayer` because it contains all the logic of the system. Yet, I faced some challenges with testing the Neo4j database and had to configure a test database instance with a unique URI and password. Additionally, I had to add a test dependency for Neo4j to my project and create a configuration file in my test resources. As I wrote my tests, I found that some of them were failing due to issues with my code or test configuration. To resolve these issues, I used debugging techniques to identify the root cause of the failures and made the necessary fixes. This iterative process has been effective in catching issues early and ensuring that my code works as expected.

In my testing process for my Neo4j graph database project, I placed a significant emphasis on testing the specific functions created within the project and the functionalities provided by Spring Data. These refer to the ones where I leverage my knowledge of Cypher, the Neo4j database language, to create custom functions not offered by Spring Data, further enhancing the functionality and value of the project. This approach allowed me to ensure that my custom functions behaved as expected and produced the desired results, ultimately contributing to the reliability and robustness of the application. Furthermore, Through this process, I gained a deeper understanding of the intricacies of the Neo4j database and developed expertise in testing custom functions to ensure optimal performance and functionality. Overall, my process for creating tests has involved researching, configuring my test environment, writing tests, debugging failures, and creating test data. This iterative process has been effective in catching issues early and ensuring that my code works as expected.

In terms of user interface (UI) testing, a thorough technique known as manual testing [10] has been utilized. This intricate method involves executing a series of actions while meticulously scrutinizing the corresponding outcomes against predetermined expectations. It is pertinent to note that the application in question features numerous front-end validations, such as precluding the user from submitting incomplete data and prohibiting the creation of duplicate user accounts with identical email addresses. Additionally, it also employs stringent password-matching measures for secure sign-in procedures.

# Chapter 6

## Conclusions

### 6.1 Results

The neighbourhood app incorporates a range of features that significantly contribute to community interaction, communication, and collaboration among residents. One key feature is the ability for users to make friends on the app. This is done in the make connections tab where a list of potential friends is shown based on common groups. Then they will have the ability to send friend requests and establish friendships within the app. The receiver of the request will have at the top of the make connections tab a list of friend requests. This feature enables individuals to connect with like-minded neighbours and build relationships based on shared interests and proximity. Another essential feature is the implementation of groups within the app. Users can create or join groups based on specific interests, and neighbourhoods. These groups serve as virtual communities where residents can engage in discussions and share information relevant to their shared interests or local area. The groups available will be sorted by the number of common friends. They will also have the ability to get out of their current groups. All these will be in a separate tab named Groups.

Furthermore, the app incorporates a post and comment system. Users can create posts to share news, seek recommendations, or initiate discussions on various topics. On the main screen is an add comment button that will take the user to a new page where they choose the corresponding group and the comment for making the post. The comment feature allows users to engage in conversations, provide feedback, or offer support. This can be done by pressing a message button in line with a specific comment. This feature enhances community communication by facilitating open dialogue and enabling residents to connect, exchange ideas, and contribute to the neighbourhood's collective knowledge. Comments can be deleted by long

pressing them from the main screen. Post on the other hand will be shown on the profile page. The profile page will be accessible from the main page and will show the name, email, an about section and a list of posts of the user. By pressing on the profile photo then the user will be able to modify the display name and the profile about section.

## **6.2 future work**

While the developed neighbourhood app shows promising results, there are several avenues for future work and improvement. Firstly, expanding the app's user base and conducting user surveys or interviews to gather more comprehensive feedback can provide insights into user needs and preferences.

In the future work of the neighbourhood app, there are several additional features that can enhance its functionality and user experience. One important feature to implement is direct chat functionality, enabling users to communicate privately and directly within the app. This feature promotes deeper connections and facilitates personalized interactions among residents.

Real-time notifications on new posts and comments are another crucial addition to the app. By implementing this feature, users will receive instant updates on relevant activities and discussions, keeping them engaged and informed about the latest community news.

Integrating an external API for weather maps is another valuable feature to consider. This integration will provide users real-time weather information specific to their location, offering a more personalized experience. Additionally, it can be leveraged to alert users about hazardous weather conditions like severe storms, heat waves, or potential animal attacks, ensuring their safety and well-being.

To improve the app's scalability and performance, hosting it on a dedicated server is recommended. Hosting the application on a reliable and secure server ensures stable access for users and allows for efficient management of data and resources.

Another idea I would like the app to have is the ability for users to organise events. By providing a platform where users can create and promote local events, the app fosters a sense of belonging and encourages participation. Whether it's organizing block parties, community clean-ups, or yard sales, the event organiza-

tion feature enables residents to connect with their neighbours, strengthen social ties, and build a vibrant community spirit. In the app, specific groups or neighbourhoods can have a separate tab dedicated to displaying upcoming events. This allows users within those groups to easily discover and engage with events that are relevant to their interests and location. Additionally, incorporating RSVP options such as "going," "interested," or "not going" further encourages participation as seeing their friends going will be a good motivation.

Continuing to gather user feedback and conducting regular usability testing can provide valuable insights for further improvements. Additionally, exploring opportunities for partnerships with local businesses, organizations, or government agencies can enrich the app's offerings and expand its impact within the community.

By incorporating these additional features and continually refining the app based on user feedback, the neighbourhood app can evolve into a comprehensive and indispensable tool for fostering community engagement and improving the quality of life for its residents.

Continual updates and improvements to the app based on user feedback and evolving community needs will contribute to its long-term success and effectiveness in fostering community engagement and connectivity.

## **6.3 conclusion**

The significance of the neighbourhood app in encouraging community engagement, communication, and collaboration among residents cannot be overdrawn. In today's world, communities often face challenges in maintaining strong social connections and active participation. The neighbourhood app serves as a powerful tool to address these challenges and create a sense of belonging and community pride. First and foremost, the app facilitates community engagement by providing a centralized platform where residents can come together, interact, and share their thoughts, ideas, and concerns. It breaks down the barriers of physical distance and allows individuals to connect with their neighbours, fostering a sense of unity and shared purpose. Through features such as user profiles, news feeds, and community forums, residents can engage in meaningful discussions, collaborate on local initiatives, and contribute to the overall well-being of their neighbourhood.

The developed features and technologies of the neighbourhood app have played a significant role in achieving the objectives of promoting community interaction



and improving the quality of life for residents. One of the primary objectives of the app was to promote denizens interaction. This objective has been met through various features such as user profiles, friend groups, and post and comment functionalities. By allowing users to create profiles and connect with their neighbours, the app has created a virtual space for residents to engage in conversations, share ideas, and build relationships. The user group feature further enhances this interaction by providing a platform for residents with common interests to connect and collaborate on specific topics or initiatives. Additionally, the ability to post and comment on the app's platform encourages lively discussions and enables residents to participate in community conversations actively. Moreover, the app's user-friendly interface and intuitive design have ensured ease of use for residents of all ages and technological backgrounds. This inclusivity has further enhanced community interaction, as individuals feel comfortable navigating the app and actively participating in its features. The app's responsiveness and real-time updates have also contributed to a seamless user experience, enabling residents to stay connected and engaged with the community at their convenience.

A significant aspect of the neighbourhood app is its ability to provide a safe space for introverted individuals who may find it challenging to make connections or participate in social events. The app creates an inclusive environment where introverts can connect with others by offering a user-friendly interface and a privacy-focused approach. This fosters meaningful online interactions and virtual relationships, bridging the gap between introverts and extroverts and promoting a diverse and connected community.

The integration of Neo4j in the app demonstrates the benefits of using a graph database for managing friendships and exploring social networks. The implementation successfully establishes friend relationships, maintains data consistency, and provides efficient traversal of friend connections. The use of Spring Boot and Flutter frameworks enables the development of a full-stack application with a user-friendly interface. The integration of Neo4j enhances the functionality and performance of the system, making it a reliable and effective tool for managing friendships and analyzing social networks.

In conclusion, the implementation of the neighbourhood app in real-world scenarios has practical implications that extend beyond the digital domain. By fostering community engagement, improving communication channels, and promoting collaboration, the app has the potential to alter neighbourhoods and bring residents together. Its functional benefits range from enhancing social connections and pro-

moting inclusivity to facilitating effective information sharing and increasing community participation. As communities continue to evolve, the neighbourhood app stands as a worthwhile tool in nurturing vibrant, connected, and resilient neighbourhoods.

# Bibliography

- [1] Jorge Acetozi and Jorge Acetozi. The spring framework. *Pro Java Clustering and Scalability: Building Real-Time Apps with Spring, Cassandra, Redis, WebSocket and RabbitMQ*, pages 47–53, 2017.
- [2] Soad Almadby. Comparative analysis of relational and graph databases for social networks. In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–4. IEEE, 2018.
- [3] Joshua Bloch. How to design a good api and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507, 2006.
- [4] André B Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203, 2000.
- [5] Shady Boukhary and Eduardo Colmenares. A clean approach to flutter development through the flutter clean architecture package. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1115–1120. IEEE, 2019.
- [6] Diogo Fernandes and Jorge Bernardino. Graph databases comparison: Allegro-graph, arangodb, infinitegraph, neo4j, and orientdb. In *Data*, pages 373–380, 2018.
- [7] Robert P Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, 1974.
- [8] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. Graph databases: Neo4j analysis. In *ICEIS (1)*, pages 351–356, 2017.
- [9] Michael Hunger and Oliver Gierke. *Good Relationships: The Spring Data Neo4j Guide Book*. C4Media, 2012.

- [10] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. How do testers do it? an exploratory study on manual testing practices. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 494–497. IEEE, 2009.
- [11] Marco L Napoli. *Beginning flutter: a hands on guide to app development*. John Wiley & Sons, 2019.
- [12] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):228, 2017.
- [13] Madeleine Renyi, Ramazan Gündogdu, Christophe Kunze, Petra Gaugisch, and Frank Teuteberg. The networked neighborhood. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–8. IEEE, 2018.
- [14] Aakanksha Tashildar, Nisha Shah, Rushabh Gala, Trishul Giri, and Pranali Chavhan. Application development using flutter. *International Research Journal of Modernization in Engineering Technology and Science*, 2(8):1262–1266, 2020.
- [15] Tuomas Vase. Advantages of docker. 2015.
- [16] Hong Zhu, Patrick AV Hall, and John HR May. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427, 1997.