

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**  
**по дисциплине «Введение в нереляционные базы данных»**  
**Тема: Визуализатор и анализатор электронной почты в виде графа**

Студенты гр. 0383

Преподаватель

Бояркин Н.А.  
Самара Р.Д.  
Сергевнин Д.В.

Заславский М.М.

Санкт-Петербург

2023

## ЗАДАНИЕ

Студенты

Бояркин Н.А.

Самара Р.Д.

Сергевнин Д.В.

Группа 0383

Тема задания: Визуализатор и анализатор электронной почты в виде графа

Исходные данные:

Задача - сделать сервис, который позволяет визуализировать и анализировать электронную почту с помощью графов. Вершины - письма и авторы, ребра - хронология (цепочки писем), связи между авторами (работа). Необходимые (но не достаточные) фичи - визуализация в виде графа, настройки, аналитика (самые длинные цепочки, корреспонденты, сроки ответов ...), таблица поиска писем

Содержание пояснительной записки:

«Содержание»

«Введение»

«Сценарий использования»

«Модель данных»

«Разработанное приложения»

«Заключение»

«Приложение А. Документация по сборке и развертыванию приложения»

Предполагаемый объем пояснительной записки:

Не менее 28 страниц.

Дата выдачи задания: 27.09.2023

Дата сдачи реферата: 24.12.2023

Дата защиты реферата: 24.12.2023

Студенты		Бояркин Н.А. Самара Р.Д. Сергевнин Д.В.
Преподаватель		Заславский М.М.

## **АННОТАЦИЯ**

В рамках данного курса предполагалось разработать в команде приложение на одну из предложенных тем. Была выбрана тема: Визуализатор и анализатор электронной почты в виде графа. Для выполнения задания предлагается использовать СУБД Neo4j. Найти исходный код и всю дополнительную информацию можно по ссылке: <https://github.com/moevm/nosql2h23-graph-email>.

## **SUMMARY**

As part of this course, the team was supposed to develop an application on one of the proposed topics. The topic chosen was: Email graph visualizer and analyzer. To complete the task, it is proposed to use the Neo4j. You can find the source code and all additional information at the link: <https://github.com/moevm/nosql2h23-graph-email>.

## СОДЕРЖАНИЕ

Введение	6
1. Сценарии использования	7
1.1. Макет UI	7
1.2. Сценарии использования для задачи	7
2. Модель данных	11
2.1. Нереляционная модель данных	11
2.2. Аналог модели данных для SQL СУБД	18
2.3. Сравнение моделей	23
3. Разработанное приложение	25
3.1. Краткое описание	25
3.2. Используемые технологии	25
3.3. Снимки экрана приложения	25
Заключение	32
Список использованных источников	33
Приложение А. Документация по сборке и развертыванию приложения	34

## **ВВЕДЕНИЕ**

Цель проведенного исследования заключалась в изучении одного из типов нереляционных баз данных с последующим внедрением его в проект. Разработанное Android-приложение и сервер предоставляют функционал визуализации и анализа электронной почты в форме графа.

# 1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

## 1.1. Макет UI

Разработанный макет приложения доступен по ссылке:

<https://www.figma.com/file/Dm4kXpqoTVifqLebDLXhH8/NoSQL-App?type=design&mode=design&t=9AOs3AegrXKKyzER-0>

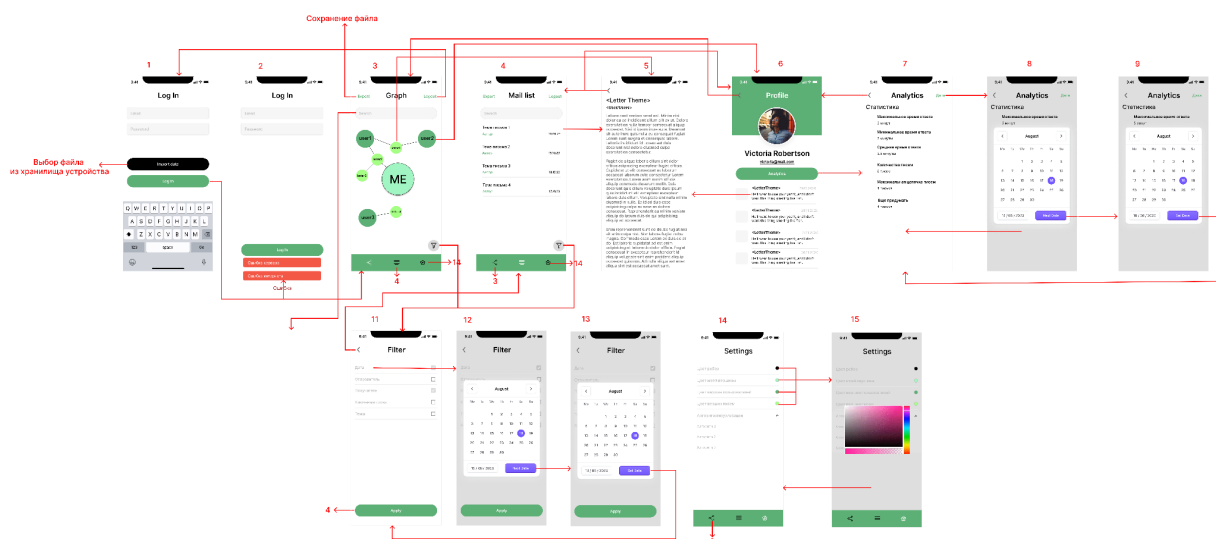


Рисунок 1. - Макет UI

## 1.2. Сценарии использования для задачи

### 1. Сценарий использования - "Вход в приложение":

**Действующее лицо:** Пользователь

**Основной сценарий:**

- 1) Пользователь открывает приложение.
- 2) Приложение запрашивает аутентификацию пользователя (логин и пароль).
- 3) Пользователь вводит свои учетные данные.
- 4) Приложение проверяет данные и входит в учетную запись пользователя.
- 5) Пользователь переходит к главному экрану №3 (переход к сценарию №2).

**Альтернативный сценарий:**

- 1) Пользователь может выбрать опцию "Импорт данных".
- 2) Происходит выбор файла из хранилища устройства
- 3) После завершения импорта, пользователь переходит к главному экрану №3 (переход к сценарию №2).

## 2. Сценарий использования - "Просмотр графа электронной почты":

**Действующее лицо:** Пользователь

**Основной сценарий:**

- 1) Пользователь открывает приложение.
- 2) Пользователь находится на главном экране №3, где приложение отображает граф электронной почты.
- 3) Пользователь может приближать, перемещать и исследовать граф.
- 4) На экране есть кнопки для перехода к настройкам отображения графа (экраны №14-15 и use case №5), отображение писем в виде списка (экран №4), поиску писем и отправителей (экран №16, №4 и use case №3), выходу из учетной записи пользователя (экран №1 и use case №1), фильтрации графа/списка (по дате, отправителю, получателю, по ключевым словам и темам (экран №11-13)), а также "Экспорт данных" (приложение генерирует машиночитаемый файл с данными и автоматически скачивается на устройство пользователя).
- 5) При нажатии на вершину контакта, пользователь переходит к профилю контакта (экран №6 и use case №4).
- 6) При нажатии на вершину письма, пользователь переходит к экрану с деталями письма (экран №5).

## 3. Сценарий использования - "Поиск и фильтрация писем по различным параметрам":

**Действующее лицо:** Пользователь

**Основной сценарий:**

- 1) Пользователь находится на главном экране №3.
- 2) При вводе текста в поле "Search", пользователь получает список писем, соответствующих запросу (экран №4) или измененный граф (экран №16), и может нажать на письмо для перехода к деталям (экран №5).



3) При нажатии на кнопку "Filter", пользователь переходит к экрану №11, где может выполнять поиск и фильтрацию писем по параметрам, таким как дата (экраны №12-13), отправитель, получатель, ключевые слова и темы.

4) После применения фильтра, пользователь возвращается к экрану с результатами поиска (экран №4 и №16).

#### 4. Сценарий использования - "Просмотр аналитики контакта":

**Действующее лицо:** Пользователь

**Основной сценарий:**

1) Пользователь находится на экране с профилем контакта (экран №6).

2) Пользователь может просматривать список тем писем с этим контактом.

3) При нажатии на кнопку "Analytics", пользователь переходит к экрану №7, где может просматривать различную статистику и аналитику связанную с этим контактом.

4) Пользователь может выбрать диапазон дат, за какой период он хочет увидеть аналитику (экраны №8-9).

5) При нажатии на письмо, пользователь переходит к деталям письма (экран №5).

6) Пользователь может вернуться к главному экрану приложения (экран №3).

#### 5. Сценарий использования - "Настройка отображения графа":

**Действующее лицо:** Пользователь

**Основной сценарий:**

1) Пользователь просматривает граф и находится на главном экране №3.

2) Пользователь нажимает на кнопку настроек и переходит к экрану №14.

3) Пользователь может настроить цвета, алгоритмы визуализации и другие параметры отображения графа.

4) Для выбора цвета вершин/рёбер появляется окно для выбора цвета (экран №15). Для применения выбранного цвета пользователь нажимает на любое место экрана, кроме палитры и попадает на экран №14.

5) После настройки, пользователь нажимает на кнопку для возвращения к главному экрану приложения и данные настройки применяются автоматически (экран №3).

6. Альтернативный сценарий:

- В случае любой ошибки, пользователь увидит всплывающее окно, информирующее об ошибке.

## 2. МОДЕЛЬ ДАННЫХ

### 2.1. Нереляционная модель данных

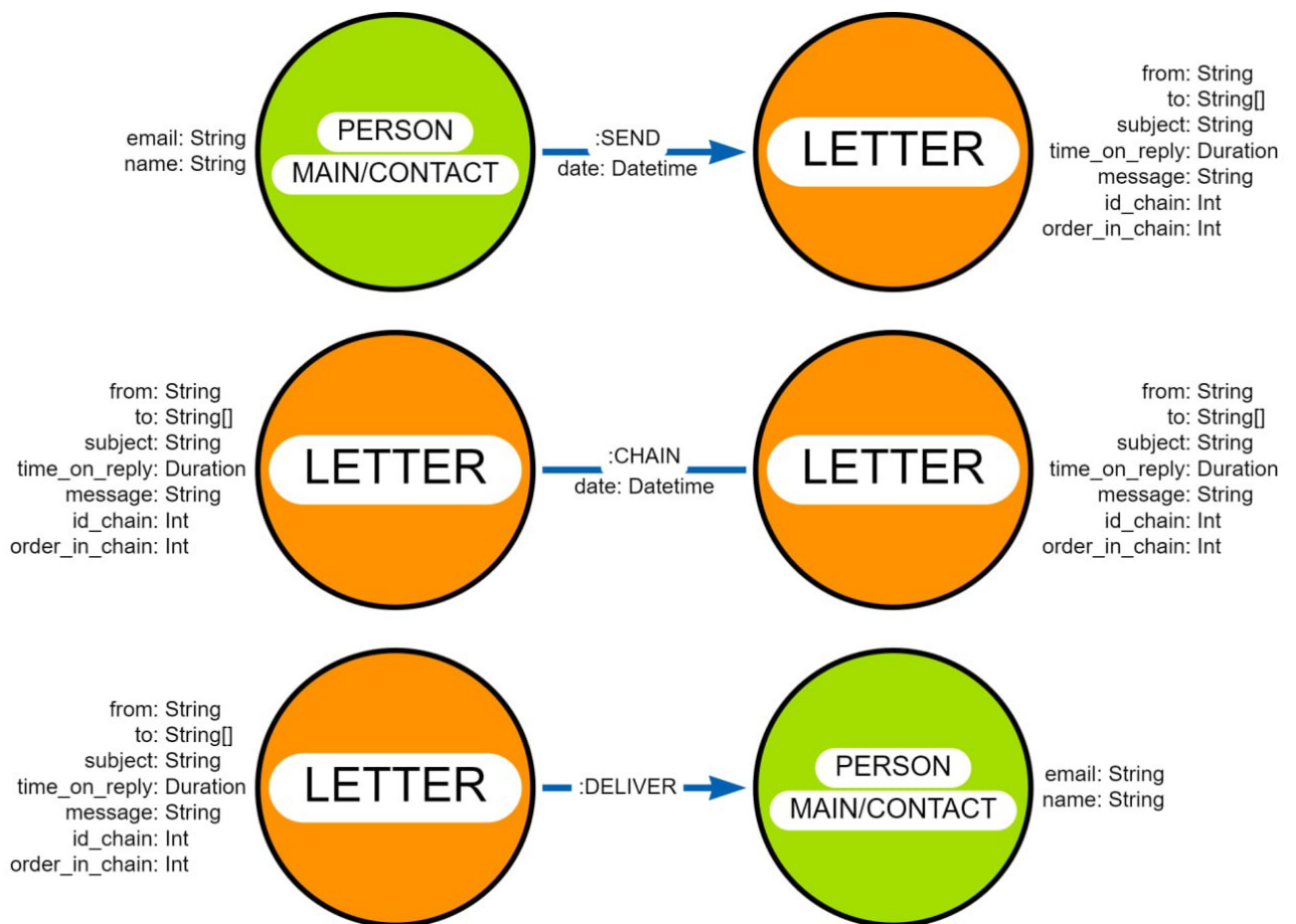


Рисунок 2. - Графическое представление нереляционной модели

#### 1. Описание назначений коллекций, типов данных и сущностей

Сущности:

Авторы ("PERSON"): Сущность, представляющая авторов писем.

Содержит атрибуты, такие как имя ("name"), адрес электронной почты ("email").

Используются метки, которые обозначают каким типом пользователя является вершина:

- метка "CONTACT" обозначает вершину контакта пользователя.
- метка "MAIN" обозначает вершину пользователя.

Письма ("LETTER"): Сущность, представляющая отдельные электронные письма. Содержит атрибуты, такие как тема письма ("subject"), текст письма ("message"), адрес электронной почты отправителя ("from"), адреса получателей

("to"), включая адреса, указанные в "cc" (копия), идентификатор цепочки писем ("id\_chain"), порядковый номер письма в цепочке ("order\_in\_chain"), а также время ответа на письмо ("time\_on\_reply").

Отношения:

1. "SEND": Это отношение связывает узел "PERSON" с узлом "LETTER" и представляет факт того, что человек отправил письмо. Это отношение содержит дату отправки (date).

2. "DELIVER": Это отношение связывает узел "LETTER" с узлом "PERSON" и представляет информацию о том, кому было доставлено письмо.

3. "CHAIN": Это отношение связывает один узел "LETTER" с другим и представляет цепочку связанных писем. Это отношение содержит дату ответа на предыдущие письмо (date).

Типы данных:

1. PERSON:

- name: String
- email: String

2. LETTER:

- from: String
- to: String[]
- subject: String
- time\_on\_reply: Duration
- id\_chain: Int
- order\_in\_chain: Int
- message: String

3. :SEND:

- date: Datetime

4. :CHAIN:

- date: Datetime

## 2. Оценка удельного объема информации, хранимой в модели

### 1. Оценка объема сущностей:

Обозначим:

- количество сущностей "PERSON" как  $nP$  (Предположим, ~20 контактов)
- количество сущностей "LETTER" как  $nL$  (Предположим, ~200 писем)
- количество связей как  $nR$  (Если в среднем с каждым контактом 10 писем,

то  $nR = 20$  (количество контактов с ":send") + 30 (количество контактов с ":deliver") + 20 (количество контактов) \* (среднее количество писем в цепочке с каждым контактом (9 писем) - 1) = 210.

Объем атрибута "from" как  $L\_from$  байт (~25 байт).

Объем атрибута "to" как  $L\_to$  байт (~25 байт \* 3 (количество получателей)).

Объем атрибута "id\_chain" как  $L\_id\_chain$  байт (4 байт).

Объем атрибута "order\_in\_chain" как  $L\_order\_in\_chain$  байт (4 байт).

Объем атрибута "subject" как  $L\_subject$  байт (~50 байт).

Объем атрибута "time\_on\_reply" как  $L\_time\_on\_reply$  байт (8 байт).

Объем атрибута "message" как  $L\_message$  байт (~500 байт).

Общий объем для хранения атрибутов в "LETTER" = ( $L\_from + L\_to + L\_id\_chain + L\_order\_in\_chain + L\_subject + L\_time\_on\_reply + L\_message$ ) \*  $nL$   
= 666 \*  $nL$  байт

Максимальная длина массива у атрибута "to" может составлять  $nP$ .

Объем атрибута "name" как  $P\_name$  байт (~25 байт).

Объем атрибута "email" как  $P\_email$  байт (~25 байт).

Общий объем для хранения атрибутов в "PERSON" = ( $P\_name + P\_email$ ) \*  $nP$  = 50 \*  $nP$  байт

### 2. Оценка объема связей:

Давайте обозначим:

Объем атрибута "date" в отношении ":SEND" как  $V\_SEND\_date$  байт (8 байт).

Объем атрибута "date" в отношении ":CHAIN" как  $V\_CHAIN\_date$  байт (8 байт).

Количество экземпляров отношения ":SEND" обозначим как  $nSEND$  (~20).

Количество экземпляров отношения ":DELIVER" обозначим как  $nDELIVER$  (~30).

Количество экземпляров отношения ":CHAIN" обозначим как  $nCHAIN$  (~180).

Теперь, формула для расчета общего объема для хранения атрибутов в этих связях будет следующей:

Общий объем для хранения атрибутов в ":SEND" =  $V\_SEND\_date * nSEND = 8 * nSEND$  байт

Общий объем для хранения атрибутов в ":DELIVER" = равен нулю, так как атрибута "date" нет.

Общий объем для хранения атрибутов в ":CHAIN" =  $V\_CHAIN\_date * nCHAIN = 8 * nCHAIN$  байт

Общий объем для хранения атрибутов связей = Общий объем для хранения атрибутов в ":SEND" + Общий объем для хранения атрибутов в ":CHAIN" =  $8 * nSEND + 8 * nCHAIN$  байт

### 3. Избыточность модели (отношение между фактическим объемом модели и “чистым” объемом данных).

Предположим, что для  $L$  писем и  $N$  авторов  $2/3$  всех писем будет со связью SEND, а оставшиеся со связью CHAIN. Также примерно оценим количество писем на одного автора:  $L \approx 10 * N$ . Тогда общий размер базы данных =  $666 * L + 50 * N + 2/3 * L * 8 + 1/3 * L * 8 = 674 * 10 * N + 50 * N = 6790 * N$

Чистый объем данных:  $50 * N + 666 * L = 50 * N + 666 * 10 * N = 6710 * N$

Избыточность модели: Общий размер базы данных / чистый объем данных =  $(6790 * N) / (6710 * N) = 1,012$

#### 4. Направление роста модели при увеличении количества объектов каждой сущности.

При увеличении числа объектов сущности "LETTER":

- возможное увеличение узлов "PERSON".
- увеличение числа отношений на 1 (":CHAIN") или 2 (":DELIVER" и ":SEND"), в зависимости от того, находится ли письмо в цепочке писем.

Увеличение числа объектов сущности "PERSON":

- увеличению числа связей вида ":DELIVER", ":SEND" и узла "LETTER" на 1, когда объект сущности "PERSON" является единственным получателем или отправителем.

- возможное увеличение числа связей вида ":DELIVER" на 1, когда объект сущности "PERSON" находится в списке получателей.

Совместное увеличение объектов сущности "PERSON" и "LETTER":

- Увеличивается число отношений ":SEND" и увеличивается число связей ":DELIVER" в зависимости от числа получателей.

#### 5. Запросы к модели, с помощью которых реализуются сценарии использования

##### 1. Фильтр по отправителю

```
WITH 'example@mail.com' AS SENDER
MATCH (p:PERSON)-[r]-(n:LETTER {from: SENDER})
WHERE r:SEND OR r:DELIVER
RETURN p, r, n;
```

##### 2. Фильтр по получателю/ям

```
WITH ["example1@mail.com", "example1@mail.com"] AS list_of_delivers
MATCH (p:PERSON)-[r]-(l:LETTER), (l)-[send_action:SEND]-(contacts:PERSON)
WHERE ALL(email in list_of_delivers WHERE email in l.to) AND (ANY(email in
list_of_delivers WHERE p.email = email) OR p:MAIN)
RETURN p, l, contacts;
```

### 3. Временной диапазон

```
WITH datetime("2023-10-20T04:20:00Z") AS date_start,  
datetime("2023-10-20T04:25:00Z") AS date_end  
MATCH (n1)-[r1]->(n2)  
WHERE date_start <= r1.date <= date_end  
RETURN n1, r1, n2
```

### 4. Поиск по темам

```
WITH toLower('example_subject') AS subject  
MATCH (n1)-[r1]->(n2)  
WHERE subject = toLower(n2.subject) OR subject = toLower(n1.subject)  
RETURN n1, n2
```

### 5. Получить макс. время ответа

```
WITH 'example@mail.com' AS contact  
MATCH (l1:LETTER {from: contact})  
RETURN max(l1.time_on_reply)
```

### 6. Получить мин. время ответа

```
WITH 'example@mail.com' AS contact  
MATCH (l1:LETTER {from: contact})  
RETURN min(l1.time_on_reply)
```

### 7. Получить ср. время ответа

```
WITH 'example@mail.com' AS contact  
MATCH (l1:LETTER {from: contact})  
RETURN avg(l1.time_on_reply)
```

### 8. Получить кол-во писем

```
WITH 'example@wall.street.com' AS contact  
MATCH (l1:LETTER {from: contact})  
RETURN count(l1)
```

### 9. Получить кол-во писем в цепочке с <id\_chain>

```
WITH 'example@mail.com' AS contact, example_id_chain AS id_chain_letter
```



```
MATCH (l1:LETTER {from: contact, id_chain: id_chain_letter})
RETURN count(l1)
```

## 10. Получить макс. цепочку

```
MATCH p=(:LETTER {order_in_chain: 0})-[:CHAIN*]->(:LETTER)
WITH MAX(length(p)) AS max_length
MATCH path=(:LETTER)-[:CHAIN*]->(:LETTER)
WHERE length(path) = max_length
RETURN path
```

## 11. Получить мин. цепочку:

```
MATCH path=(:LETTER {order_in_chain: 0})-[:CHAIN*]->(end_l:LETTER)
WITH end_l.id_chain AS chain_id, COUNT(nodes(path)) AS path_count,
COLLECT(path) AS path
RETURN path_count, chain_id;
```

Для получения пользовательской аналитики необходимо выполнить четыре запроса к базе данных, начиная с 5-го и заканчивая 11-м. В случае, если пользователь захочет провести анализ данных за определенный временной интервал, потребуется выполнить семь запросов.

## 2.2. Аналог модели данных для SQL СУБД

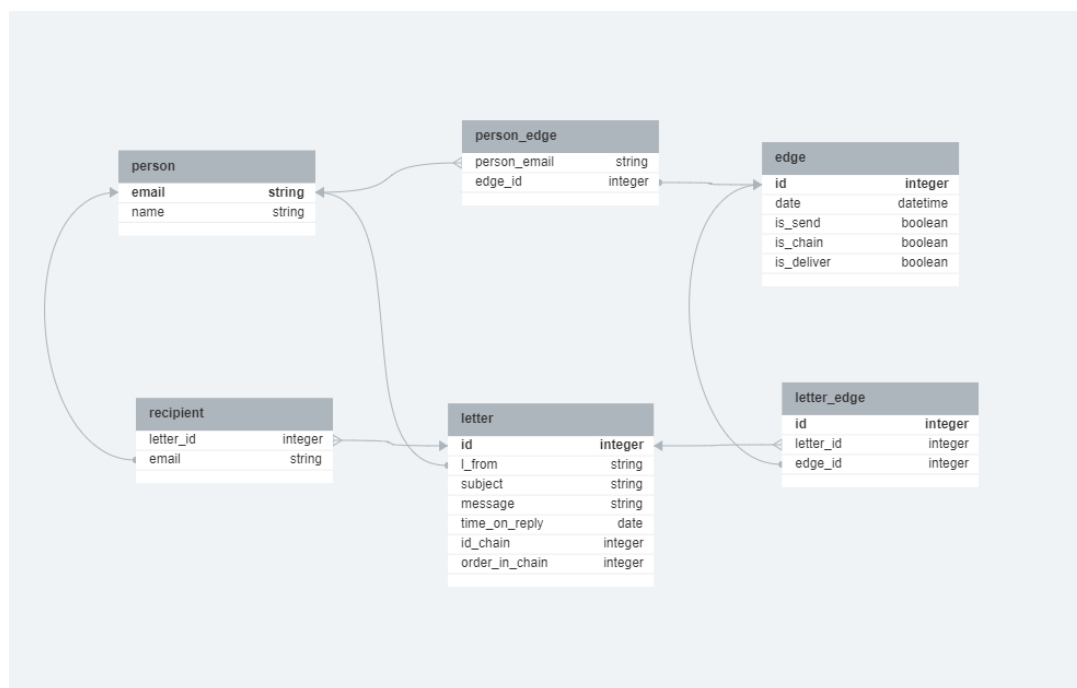


Рисунок 3. - Графическое представление реляционной модели

- ## 1. Описание назначений коллекций, типов данных и сущностей

**Сущности:**

Авторы ("PERSON"): Сущность, представляющая авторов писем.

Содержит атрибуты, такие как имя ("name"), адрес электронной почты ("email").

Письма ("LETTER"): Сущность, представляющая отдельные электронные письма. Содержит атрибуты, такие как тема письма ("subject"), текст письма ("message"), адрес электронной почты отправителя ("l\_from"), идентификатор цепочки писем ("id\_chain"), порядковый номер письма в цепочке ("order\_in\_chain"), а также время ответа на письмо ("time\_on\_reply").

Ребро ("edge"): Сущность, представляющая ребро между сущности автора и сущности письма. Содержит атрибуты, такие как дата отправки ("date"), булевы значения "is send", "is chain", "is deliver".

- ## 2. Оценка удельного объема информации, хранимой в модели

1. person (всего 50 байт)

- email: str (PK) (~25 байт)
- name: str (~25 байт)
- 2. person\_edge (всего 29 байтов)
  - email: str (FK to person.email) (many to one) (~25 байт)
  - edge\_id: int (FK to edge.id) (one to one) (~4 байта)
- 3. edge (всего 15 байт)
  - id: int (PK) (~4 байта)
  - date\_m: datetime (8 байт)
  - is\_send: boolean (~1 байт)
  - is\_chain: boolean (~1 байт)
  - is\_deliver: boolean (~1 байт)
- 4. letter\_edge (всего 12 байт)
  - id: int (PK) (~4 байта)
  - edge\_id: int (FK to edge.id) (one to one) (~4 байта)
  - letter\_id: int (FK to letter.id) (many to one) (~4 байта)
- 5. letter (всего 587 байт)
  - id: int (PK) (~4 байта)
  - l\_from: str (FK to person.email) (one to one) (~25 байт)
  - subject: string (~50 байт)
  - message: string (~500 байт)
  - id\_chain: int (4 байта)
  - order\_in\_chain (4 байта)
  - time\_on\_reply: datetime (8 байт)
- 6. recipient (всего 29 байт)
  - letter\_id: int (FK to letter.id) (many to one) (~4 байта)
  - email: string (FK to person.email) (one to one) (~25 байт)

#### 1. Оценка объема сущностей:

Авторы ("PERSON"): email: str (~25 байт) + name: str (~25 байт) = 50 байт

Письма ("LETTER"): id: int (~4 байта) + l\_from: str (~25 байт) + message: string (~500 байт) + time\_on\_reply: datetime (8 байт) + id\_chain: int (4 байта) + order\_in\_chain (4 байта) + subject: string (~50 байт) = 595 байт

Ребро ("edge"): id: int (~4 байта) + date\_m: datetime (8 байт) + is\_send: boolean (~1 байт) + is\_chain: boolean (~1 байт) + is\_deliver: boolean (~1 байт) = 15 байт

## 2. Оценка объема связей:

- person\_edge: email: str (~25 байт) + edge\_id: int (~4 байта) = 29 байт

- letter\_edge: edge\_id: int (~4 байта) + letter\_id: int (~4 байта) + id (~4 байта) = 12 байт

- recipient: letter\_id: int (~4 байта) + email: string (~25 байт) = 29 байт

## Количество ребер и связей:

Предположим, что имеем N авторов и L (10 \* N) писем, тогда общий размер базы данных =  $85 * L + 50 * N + 595 * L = 85 * 10 * N + 50 * N + 595 * 10 * N = 6850 * N$

## 3. Избыточность модели

Чистый объем данных:  $50 * N + 595 * L = 6000 * N$

Избыточность модели: Общий размер базы данных / чистый объем данных =  $(6850 * N) / (6000 * N) = 1,142$

## 4. Направление роста модели при увеличении количества объектов каждой сущности.

При увеличении количества объектов сущности авторов ("PERSON") размер модели увеличивается линейно.

При увеличении количества объектов сущности писем ("LETTER") размер модели увеличивается линейно.

## 5. Запросы к модели, с помощью которых реализуются сценарии использования

### 1. Фильтр по отправителю

```
SELECT person.*, letter.*, edge.*, recipient.*
FROM letter
JOIN person ON letter.l_from = person.email
LEFT JOIN recipient ON letter.id = recipient.letter_id
JOIN edge ON letter.id = edge.id
WHERE person.email = 'example@mail.com';
```

### 2. Фильтр по получателю/ям

```
SELECT person.*, letter.*, edge.*, recipient.*
FROM letter
JOIN recipient ON letter.id = recipient.letter_id
JOIN person ON recipient.email = person.email
JOIN edge ON letter.id = edge.id
WHERE person.email IN ('example1@mail.com', 'example2@mail.com');
```

### 3. Временной диапазон

```
SELECT person.*, letter.*, edge.*, recipient.*
FROM letter
JOIN recipient ON letter.id = recipient.letter_id
JOIN person ON recipient.email = person.email
JOIN edge ON letter.id = edge.id
WHERE edge.date_m BETWEEN date_start AND date_end;
```

### 4. Поиск по темам

```
SELECT person.*, letter.*, edge.*, recipient.*
FROM letter
JOIN recipient ON letter.id = recipient.letter_id
JOIN person ON recipient.email = person.email
JOIN edge ON letter.id = edge.id
WHERE letter.subject LIKE '%искомая_тема%';
```

### 5. Получить макс. время ответа

```
SELECT MAX(letter.time_on_reply) AS max_time_on_reply
FROM letter
JOIN edge ON letter.id = edge.id
WHERE edge.is_chain = TRUE AND 'example@mail.com' = letter.l_from;
```

### 6. Получить мин. время ответа

```
SELECT MIN(letter.time_on_reply) AS min_time_on_reply
FROM letter
JOIN edge ON letter.id = edge.id
WHERE edge.is_chain = TRUE AND 'example@mail.com' = letter.l_from;
```

### 7. Получить ср. время ответа

```
SELECT AVG(letter.time_on_reply) AS avg_time_on_reply
FROM letter
JOIN edge ON letter.id = edge.id
WHERE edge.is_chain = TRUE AND 'example@mail.com' = letter.l_from;
```

### 8. Получить кол-во писем

```
SELECT COUNT(*) AS total_letters
FROM letter;
WHERE 'example@mail.com' = letter.l_from;
```

### 9. Получить кол-во писем в цепочке с <id\_chain>

```
SELECT COUNT(*) AS chain_letters_count
FROM letter
WHERE id_chain = <id_chain>;
```

### 10. Получить макс цепочку

```
SELECT letter_id, MAX(path_length) AS max_length
FROM (
    SELECT letter_id, COUNT(*) AS path_length
    FROM letter_edge
    GROUP BY letter_id
) AS path_lengths
```

#### 11. Получить мин. цепочку:

```
SELECT letter_id, MIN(path_length) AS min_length
FROM (
    SELECT letter_id, COUNT(*) AS path_length
    FROM letter_edge
    GROUP BY letter_id
    HAVING COUNT(*) > 1
) AS path_lengths
```

Количество запросов для совершения юзкейсов в реляционной модели может увеличиваться с увеличением числа объектов в БД. Это связано с тем, что в реляционной модели связи между объектами хранятся в отдельных таблицах.

### 2.3. Сравнение моделей

#### 1. Удельный объем информации

Нереляционная модель имеет меньший удельный объем информации, чем реляционная модель. Это связано с тем, что в нереляционной модели связи между объектами хранятся в более сжатом виде, чем в реляционной модели.

#### 2. Количество запросов для совершения юзкейсов в зависимости от числа объектов в БД

Количество запросов для совершения юзкейсов в нереляционной модели не зависит от числа объектов в БД. В реляционной модели количество запросов может увеличиваться с увеличением числа объектов в БД.

#### 3. Вывод

Нереляционная модель является более предпочтительным выбором для хранения данных о почте. Она позволяет уменьшить избыточный объем информации и снизить число запросов для реализации сценариев использования.

Также нереляционная модель имеет следующие преимущества:

- Гибкость: нереляционная модель не требует строгого соблюдения схемы данных, что позволяет легко вносить изменения в структуру данных.
- Производительность: нереляционные базы данных могут быть более производительными, чем реляционные базы данных, при работе с определенными типами запросов.
- Легкость масштабирования: нереляционные базы данных можно легче масштабировать, чем реляционные базы данных.



### **3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ**

#### **3.1. Краткое описание**

Разработанное приложение выполняет следующие функции:

1. Визуализация цепочек писем в Android-приложении.
2. Возможность настройки через API приложения отображения графа, цветов, выбор алгоритмов визуализации графов и других параметров визуализации.
3. Проведение анализа графа электронной почты по API, включая:
  - сроки ответов: Модуль аналитики отслеживает время, затраченное на ответы на письма. Пользователи могут увидеть минимальный, средний и максимальный срок ответа для каждого контакта через API приложения.
4. Таблица поиска писем в Android-приложении:
  - Поиск и фильтрация писем по различным параметрам, таким как дата, отправитель, получатель, ключевые слова и темы.

#### **3.2. Используемые технологии**

БД: Neo4j (язык запросов Cypher)

Backend: Python (Flask)

Frontend: Android (Kotlin), HTML, CSS, JavaScript

#### **3.3. Снимки экрана приложения**

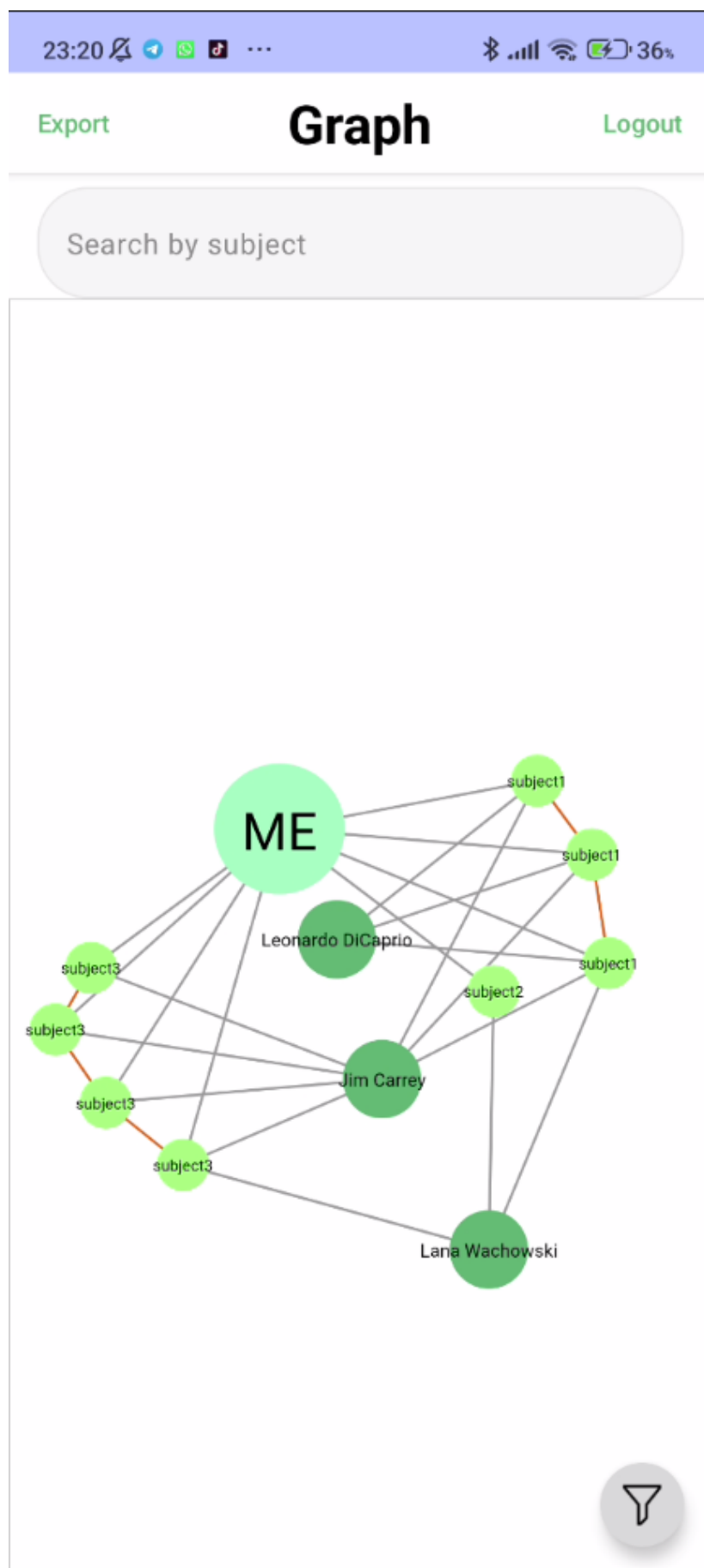


Рисунок 4. Все письма пользователя.

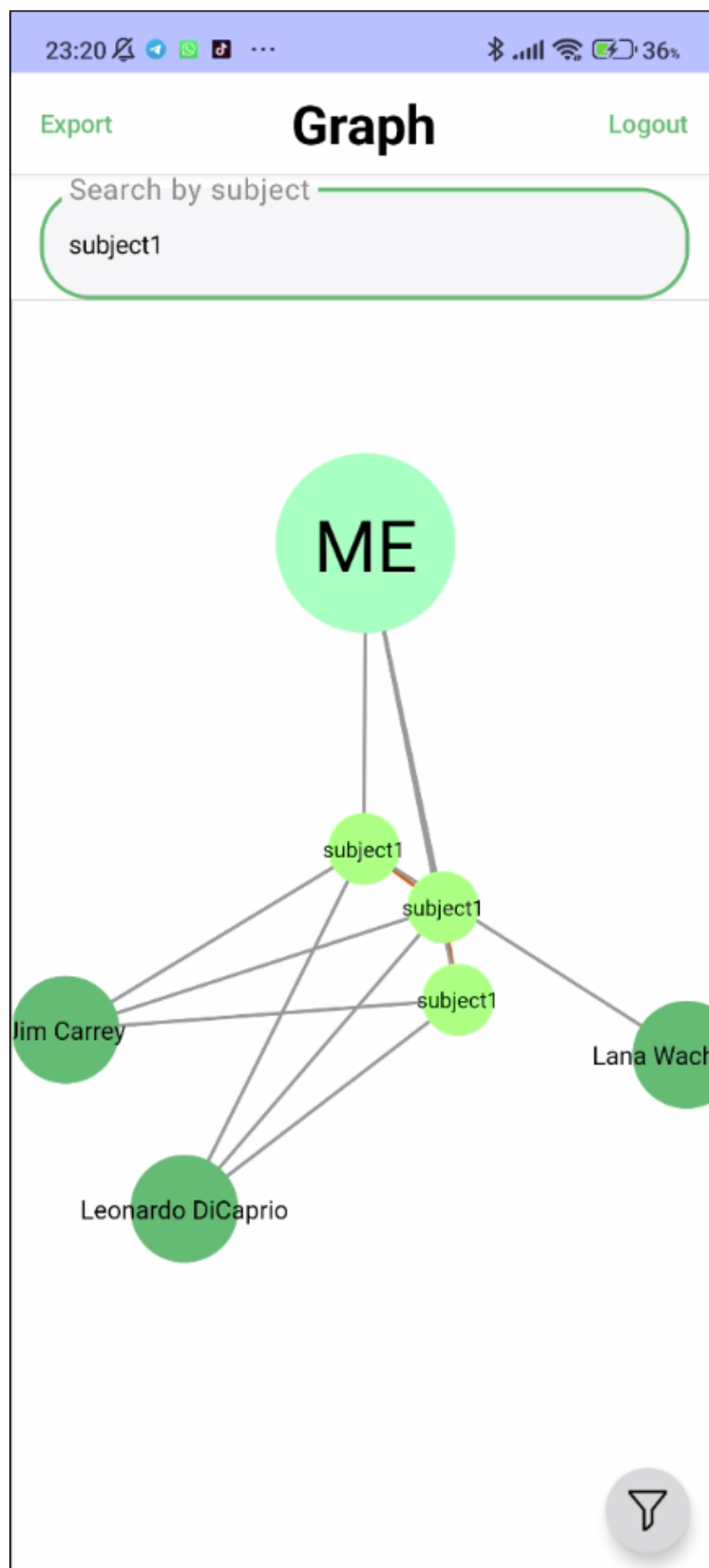












Рисунок 5. Визуализация писем по поиску.

23:20



 36%



Filter

Start date

End date

Sender

keanu@mail.com

Receiver

Subject

Apply

Рисунок 6. Фильтр по отправителю.

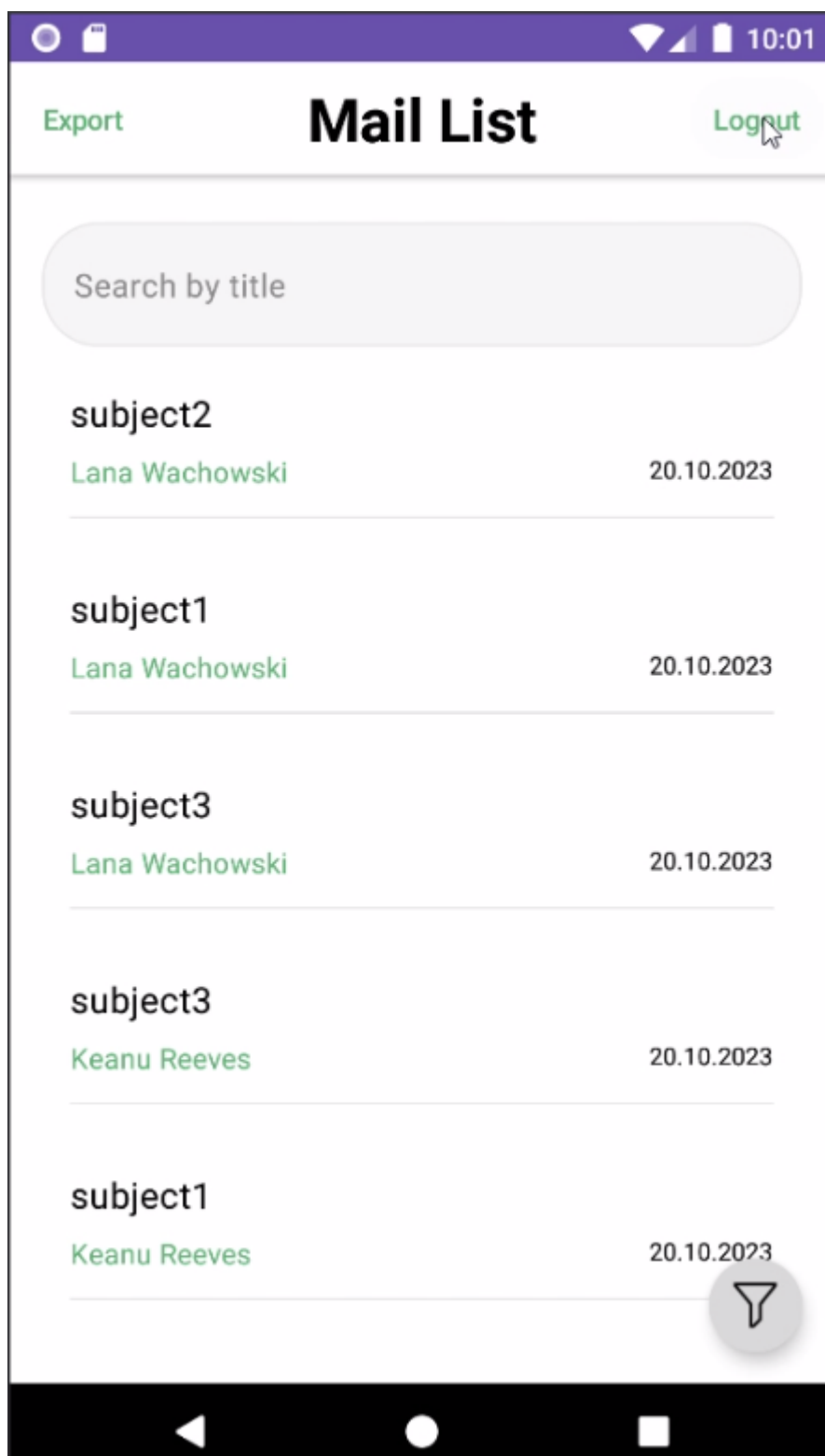


Рисунок 7. Список писем.

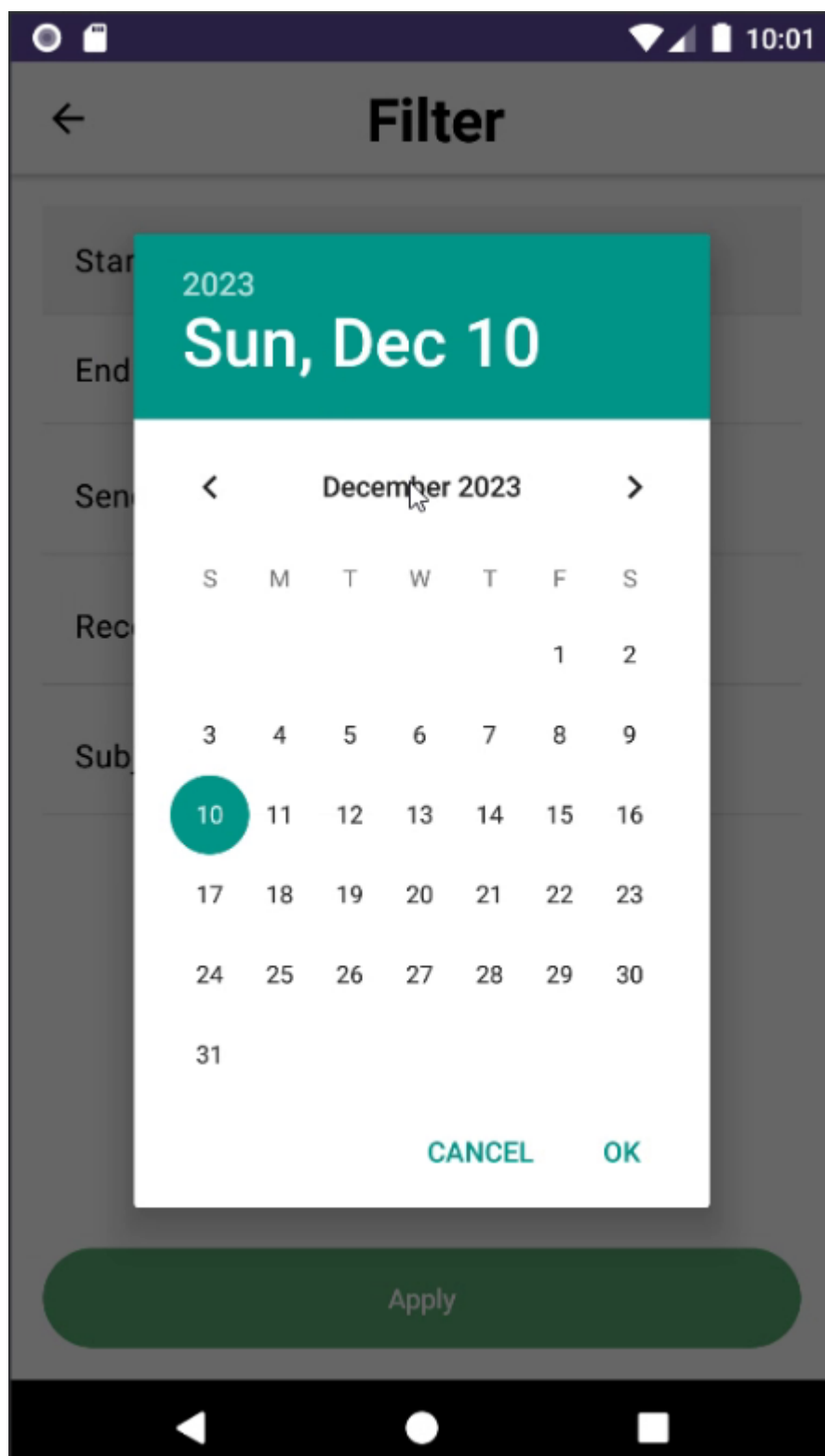


Рисунок 8. Фильтр письма по дате.

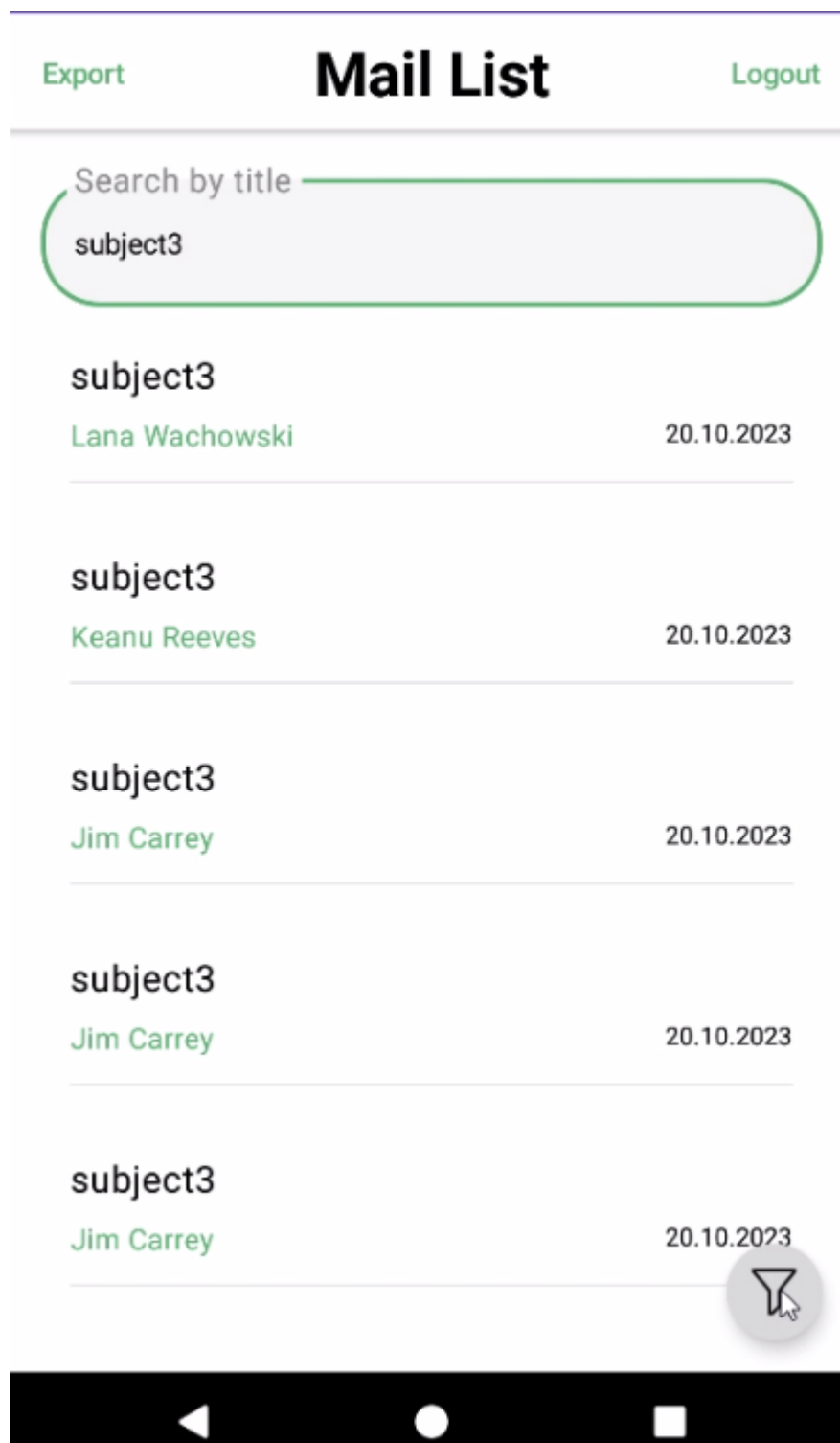


Рисунок 9. Результат поиска по теме.

## **ЗАКЛЮЧЕНИЕ**

Фильтрация и визуализация электронной почты в виде графа была успешно интегрирована в Android-приложение, однако доступ к аналитике и настройкам визуализации графа осуществляется только через API сервера. Проект использует технологии, такие как Python (Flask), Kotlin (Android), и Neo4j (Cypher). В дальнейшем развитии планируется интеграция с различными почтовыми сервисами, а также создание web-приложения.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий веб-приложения: [электронный ресурс]. URL: <https://github.com/moevm/nosql2h23-graph-email> (дата обращения 24.12.2023).
2. Документация по Neo4j: [электронный ресурс]. URL: <https://neo4j.com/docs/getting-started/> (дата обращения 24.12.2023).
3. Документация по Cypher: [электронный ресурс]. URL: <https://neo4j.com/docs/cypher-manual/> (дата обращения 24.12.2023).
4. Документация по Flask: [электронный ресурс]. URL: <https://flask.palletsprojects.com/en/latest/> (дата обращения 24.12.2023).
5. Документация по Kotlin: [электронный ресурс]. URL: <https://kotlinlang.org/docs/home.html> (дата обращения 24.12.2023).

## ПРИЛОЖЕНИЕ А

### ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

Сборка сервера с СУБД Neo4j:

`docker-compose up --build`

#### API:

*POST /api/graph/load\_json* - для загрузки JSON в приложение

*GET /api/graph/graph\_data* - для получения графа. Параметры `export` (используется для экспорта, принимает значения True/False), `only_letters` (используется для получения только писем, принимает значения True/False), `email_sender` (используется для фильтрации по отправителю), `subject` (используется для фильтрации по теме письма), `start_date` и `end_date` (используется для фильтрации по времени, принимает ISO формат, например, 2023-11-20T06:15:00)

*GET /api/analytics/get\_analytics* - для получения аналитики. Параметры `is_iso_format` (используется для возвращения значений с типом данных Duration в ISO формате, принимает значения True/False), `email_contact` (используется для получения аналитики по конкретному пользователю, обязательный параметр), `id_chain` (используется, чтобы указать id цепочки), `start_date` и `end_date` (используется для фильтрации по времени, принимает ISO формат, например, 2023-11-20T06:15:00)

*GET /api/analytics/get\_max\_chain* - Получение максимальной цепочки у главного пользователя. Возвращает только письма и связи между ними.

*GET /api/analytics/get\_min\_chain* - Получение мин. цепочку у главного пользователя. Возвращает JSON `chain_id_to_path_count`, где ключ `chain_id`, значение кол-во писем в цепочке, а `chain_id_min_path_count` - значение `chain_id`, который имеет мин. кол-во писем в цепочке. Имеет фильтр по временному диапазону.

*GET /api/graph/get\_letter\_data* - Получение данных письма по id. Параметр `letter_element_id` (Element\_ID письма)

*GET /api/graph/get\_person\_data* - Получение данных пользователя по email.

Параметр email\_person (email контакта)

*GET /api/graph/* - Визуализация графа. Параметры color\_contact\_vertices (используется для настройки цвета вершин контактов), color\_user\_vertex (используется для настройки цвета вершины пользователя), ,

color\_letter\_vertices (используется для настройки цвета вершин писем),

color\_edges\_sending (используется для настройки цвета ребер типа “SEND”),

color\_edges\_receiving (используется для настройки цвета ребер типа

“DELIVER”), color\_edges\_chain (используется для настройки цвета ребер типа “CHAIN”).