IDB Final Project Report

第 12 組: 交大十二舍空間小幫手

111550028 黃柏翔 111550061 邱冠崴 111550066 王裕昕
111550089 李宗諺 111550141 葉峻誠 (按學號排序)

Table of Contents

(1) An introduction of your application, including why you want to develop the application and the main functions of your application.

TL;DR

We built a public area borrowing system for Dormitory 12 at NYCU. This system allows users to conveniently reserve meeting rooms, audio-visual rooms, dancing rooms, and the kitchen with just a few clicks through our LINE BOT interface. We have also open-sourced our code, allowing anyone interested in building a similar system to utilize it.

Motivation:

As endorsed by our outstanding and distinguished principal, Dormitory 12 at NYCU offers an exceptional dormitory environment. The basement houses various public areas, providing students with substantial workloads a dedicated space to both work and unwind. However, the exceptional quality of environment is not paralleled by the space reservation system, which frequently prone to breakdowns and usability challenges. Occasionally, disputes among students arise regarding borrowing time limitations.

To address this issue, we, a group of students residing in the dormitory, propose a solution by leveraging LINE BOT and AWS service.

Functionalities

The system comprises four key functions: Registration, Reservation, Searching, and Cancellation. These functions are seamlessly implemented

through the integration of LINE BOT, AWS Lambda, and AWS RDS(Relational Database Service).

1. Registration: To ensure that borrowers are students, the registration process prompts new users to provide essential details such as student id and his or her real name within the LINE chat.
2. Reservation: Users can effortlessly make room or public area reservations and select preferred time slots through the LINE BOT interface with just a few clicks.
3. Search: The LINE BOT allows users to check the reservation status of the space, enhancing accessibility and convenience.
4. Cancellation: In instances where users need to cancel reservations, the system allows cancellation via the LINE BOT.

(2) Database design - describe the schema of all your tables in the database, including keys and index, if applicable (why you need the keys or why you think adding an index is or is not helpful).

E-R Model

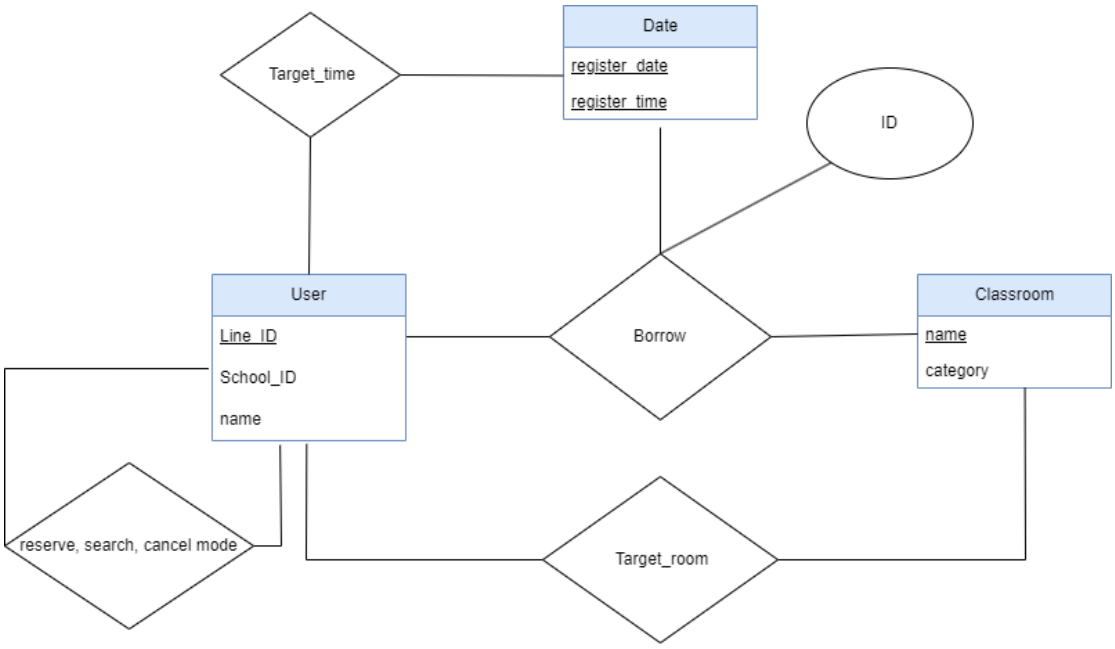

Fig. The E-R Model of our system.

Relational Schema

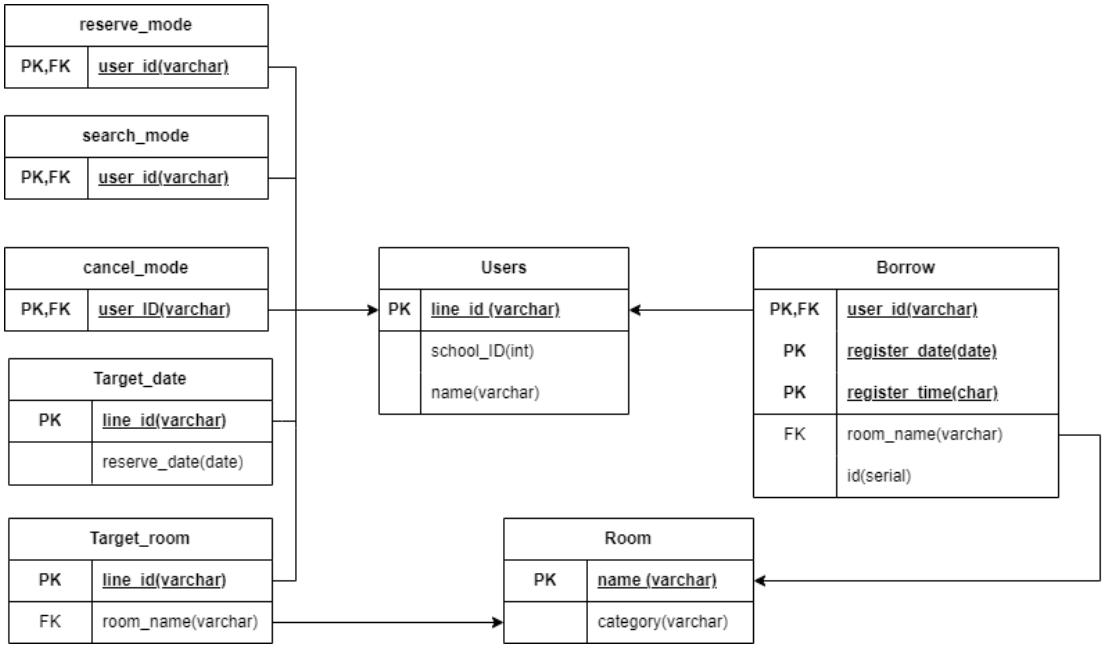

Fig. The relational schema of our system. Its functionality and structure is illustrated in detail below.

Table Functionality & Structure:

1. User Table (`users`):

- **Purpose:**
  - Stores information about users who interact with the system.
- **Schema:**
  - `line_id` (varchar, Primary Key): Line Messaging API user ID.
  - `school_id` (integer): User's student ID.
  - `name` (varchar): User's name.

2. Room Table (`room`):

- **Purpose:**
  - Contains information about different types or categories of rooms.
- **Schema:**
  - `name` (varchar, Primary Key): Room name.
  - `category` (varchar): Category or type of the room.

3. Borrow Table (`borrow`):

- **Purpose:**
  - Tracks the borrowing history of users for specific rooms.
- **Schema:**
  - `id` (serial): Auto-incremented unique identifier.
  - `user_id` (varchar, Foreign Key, Composite Primary Key): References `users(line_id)`.
  - `room_name` (varchar, Foreign Key): References `room(name)`.
  - `register_date` (date, Composite Primary Key): Date when the borrowing is registered.
  - `register_time` (char, Composite Primary Key): Specific time slot for borrowing.

4. Mode Tables (`reserve_mode`, `search_mode`, `cancel_mode`):

- **Purpose:**
  - Keep track of user modes (reserve, search, cancel).
- **Schema:**
  - `user_id` (varchar, Primary Key, Foreign Key): References `users(line_id)`.

5. Target Tables (`target_room`, `target_date`):

- **Purpose:**
  - Store temporary target room and date information for user interactions.
- **Schema:**
  - `line_id` (varchar, Primary Key, Foreign Key): References `users(line_id)`.
  - `room_name` (varchar) [for `target_room`].
  - `reserve_date` (date) [for `target_date`].

(3) Database design - describe the normal form of all your tables. If the tables are not in BCNF, please include the reason (performance trade-off, etc.).

1. User Table (`users`): 2nd Normal Form

- The primary key of the table is `{line_id}`. However, there is a functional dependency `{school_id} -> {name}`, which does not meet the requirements of 3NF.
- The decision to keep the `users(line_id, school_id, name)` table unified, rather than splitting it into `users(line_id, school_id)` and `students(school_id, name)`, is driven by the current emphasis on users registered via Line. The existing structure is more convenient for user registration and identity verification.

2. Room Table (`room`): BCNF

- No non-trivial functional dependencies, as the primary key `{name}` is a superkey.

3. Borrow Table (`borrow`): 2nd Normal Form

- The primary key of the table is `{user_id, register_date, register_time}`. However, there is a functional dependency `{id} -> {user_id, register_date, register_time, room_name}`, which does not meet the requirements of 3NF.
- The reason for this design choice is that, when users make reservations, the system doesn't have information about the reservation record ID since the ID is automatically generated upon inserting into the database. To make querying more straightforward, it's necessary to keep all relevant attributes in the table.

4. Mode Tables (`reserve_mode`, `search_mode`, `cancel_mode`): BCNF

- No non-trivial functional dependencies, as the primary key `{user_id}` is a superkey.

5. Target Tables (`target_room`, `target_date`): BCNF

- No non-trivial functional dependencies, as the primary keys (`{line_id}`) are superkeys.

(4) From the data sources to the database - describe the data source and the original format.

Since there are no names and a list of all borrowable areas in Dormitory 12, the data sources for our service are the investigation by the team members living in Dormitory 12. We manually compiled a list of all borrowable areas and created CSV file on our own.

(5) From the data sources to the database - describe the methods of importing the original data to your database and strategies for updating the data, if you have one.

To establish a table of borrowable rooms, we imported the mentioned CSV file directly into the database system.
Regarding the User list, after obtaining a comprehensive list of all students and those residing in dormitories, we can directly join the two tables. This enables us to retrieve the student ID and names of all students living in Dormitory 12, which can be directly imported into the “users” Table. However, the data remains inaccessible to us currently, and it can be further imported only when we gain access to the school database.
As for the strategy of updating the data, we will discuss it in three parts, which are about the list of public areas available for borrowing, the list of users (students), and updating the records of the database in response to the demand for reservation/cancellation request.
Regarding the list of available public areas, modifications to the table are necessary only for newly created areas or those under maintenance. Unfortunately, such situations are not automatable. On the contrary, the user list requires updating only once a year, and given its low frequency, this task can be accomplished manually. Lastly, for updating reservation records, we employ LINE BOT to interact with users and leverage AWS Lambda to commit changes to the database. The comprehensive logic is elaborated in detail in Problem (8).

(6) Application with database - explain why your application needs a database.

1. **To address the need for data storage.** The database is essential for storing and managing information related to reservation records, users, and more.
2. **To efficiently retrieve information.** When users want to check if a particular space is already reserved or retrieve their own reservation details, the use of database allows the system to respond in a fast and systematic manner.
3. **To maintain data integrity.** The database system provides a structured and organized way to store data, reducing the risk of errors or inconsistencies that might occur.
4. **To facilitate scalability.** As the number of users grows, a well-designed database can handle increased data volume and ensure that the application remains responsive and efficient.

(7) Application with database - includes the queries that are performed by your application, how your application performed these queries (connections between application and database), and what the cooperating functions for your application.

SQL Queries performed by AWS Lambda

Below, all queries performed by the Lambda function are listed along with a brief introduction to their respective usage.

CREATE TABLE

- **User Table** (Storing all user information)

```
CREATE TABLE users (line_id varchar PRIMARY KEY, school_id integer, name varchar);
```

- **Room Table** (Storing all room information)

```
CREATE TABLE room (name varchar PRIMARY KEY, category varchar);
```

- **Borrow Table** (Storing all borrowing records)

```
CREATE TABLE borrow (id serial, user_id varchar, room_name varchar, register_date date, register_time char, PRIMARY KEY (user_id, register_date, register_time), FOREIGN KEY (user_id) REFERENCES users(line_id), FOREIGN KEY (room_name) REFERENCES room(name));
```

- **Reserve Mode Table** (Storing users in reservation mode)

- `CREATE TABLE reserve_mode (user_id varchar PRIMARY KEY, FOREIGN KEY (user_id) REFERENCES users(line_id));`

- **Search Mode Table** (Storing users in searching mode)

- `CREATE TABLE search_mode (user_id varchar PRIMARY KEY, FOREIGN KEY (user_id) REFERENCES users(line_id));`

- **Cancel Mode Table** (Storing users in cancellation mode)

- `CREATE TABLE cancel_mode (user_id varchar PRIMARY KEY, FOREIGN KEY (user_id) REFERENCES users(line_id));`

- **Target Room Table** (Temporarily stores room information about currently-performed operations)

- `CREATE TABLE target_room (line_id varchar PRIMARY KEY, room_name varchar);`

- **Target Date Table** (Temporarily stores date information about currently-performed operations)

- `CREATE TABLE target_date (line_id varchar PRIMARY KEY, reserve_date date);`

INSERT RECORD

- **Insert User:** When a new user register for the system

- `INSERT INTO users (line_id, school_id, name) VALUES (%s, %s, %s);`

- **Insert Room:** The query should be performed if a new area is built

- `INSERT INTO room (name, category) VALUES (%s, %s);`

- **Borrow a Room:**

- `INSERT INTO borrow (user_id, room_name, register_date, register_time) VALUES (%s, %s, %s, %s);`

- **Switch to Reserve Mode:** Adding user_id to the table temporarily

- `INSERT INTO reserve_mode (user_id) VALUES (%s);`

- **Switch to Search Mode:** Adding user_id to the table temporarily

```
INSERT INTO search_mode (user_id) VALUES (%s);
```

- **Switch to Cancel Mode:** Adding user_id to the table temporarily

```
INSERT INTO cancel_mode (user_id) VALUES (%s);
```

- **Insert Target Room:** Adding targeting room_name to the table temporarily

```
INSERT INTO target_room (line_id, room_name) VALUES (%s, %s);
```

- **Insert Target Date:** Adding targeting date to the table temporarily

```
INSERT INTO target_date (line_id, reserve_date) VALUES (%s, %s);
```

RETRIEVE RECORD

- **Get Users:**

```
SELECT * FROM users WHERE line_id = %s;
```

- **Get User's Borrowed Records:**

```
SELECT * FROM borrow WHERE user_id = %s;
```

- **Check Room Availability:**

```
SELECT time_slot FROM (VALUES ('a'), ('b'), ('c'), ('d'), ('e'), ('f'), ('g'), ('h')) AS all_time_slots (time_slot) WHERE NOT EXISTS (SELECT 1 FROM borrow WHERE room_name = %s AND register_date = %s AND register_time = all_time_slots.time_slot);
```

- **Check Reserve Mode:**

```
SELECT * FROM reserve_mode WHERE user_id = %s;
```

- **Check Search Mode:**

```
SELECT * FROM search_mode WHERE user_id = %s;
```

- **Check Cancel Mode:**

```
SELECT * FROM cancel_mode WHERE user_id = %s;
```

- **Get Target Room:**

```
SELECT room_name FROM target_room WHERE line_id = %s;
```

- **Get Target Date:**

```
SELECT reserve_date FROM target_date WHERE line_id = %s;
```

DELETE RECORD

- **Cancel Reservation:**

```
DELETE FROM borrow WHERE id = %s AND user_id = %s;
```

- **Switch Off Reserve Mode:**

```
DELETE FROM reserve_mode WHERE user_id = %s;
```

- **Switch Off Search Mode:**

```
DELETE FROM search_mode WHERE user_id = %s;
```

- **Switch Off Cancel Mode:**

```
DELETE FROM cancel_mode WHERE user_id = %s;
```

- **Delete Target Room:**

```
DELETE FROM target_room WHERE line_id = %s;
```

- **Delete Target Date:**

```
DELETE FROM target_date WHERE line_id = %s;
```

How the queries are performed

At a high level, we leverage AWS Lambda and implement Python code to ascertain the current stage of execution. The queries are stored as strings in the Lambda function code and are executed through a helper function called `runSql(conn, QUERY, param)`, as illustrated below.

```
def runSql(conn, QUERY, param):
    res = "success"
    cur = conn.cursor()
```

```
    cur.execute(QUERY, param)
    if QUERY.split()[0] == "SELECT":
        res = cur.fetchall()
    conn.commit()
    cur.close()
    return res
```

(8) All the other details of your application that you want us to know.

Flowchart: how the system works

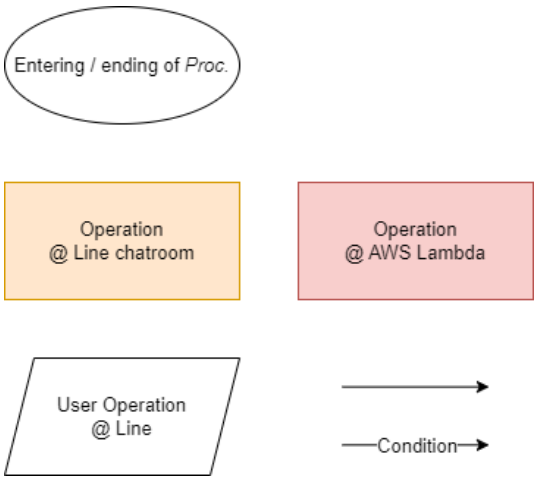

Fig. Symbols in the flowchart

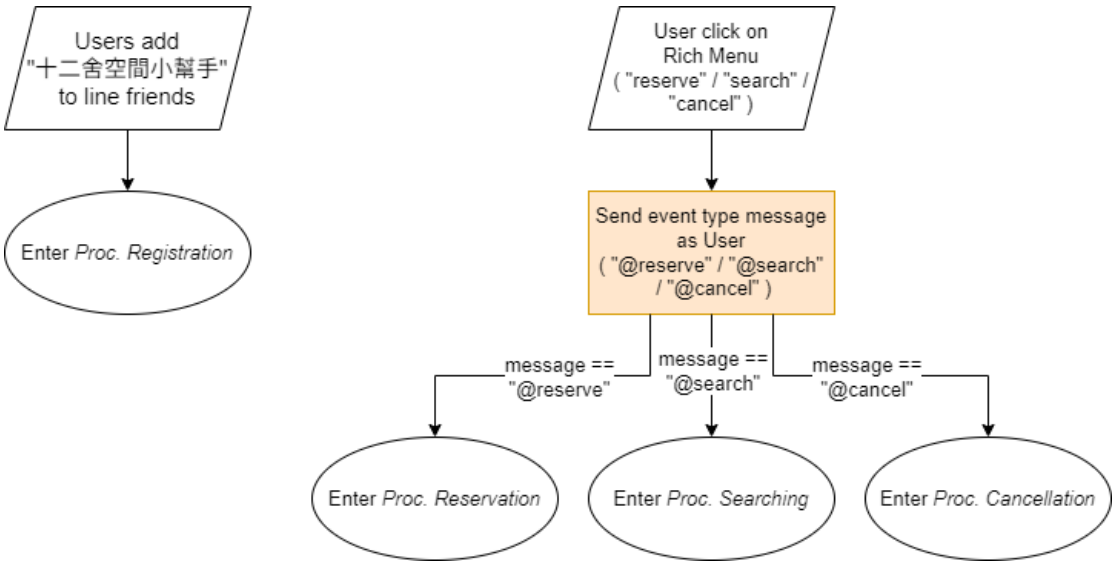

Fig. 4 main functions(procedures) and how to enter them

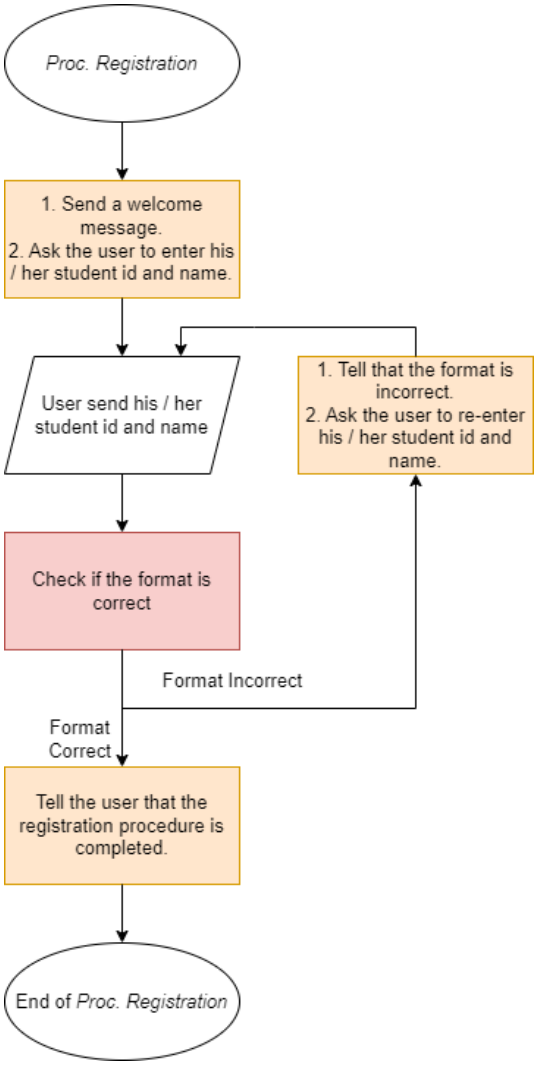

Fig. Flowchart of Procedure Registration

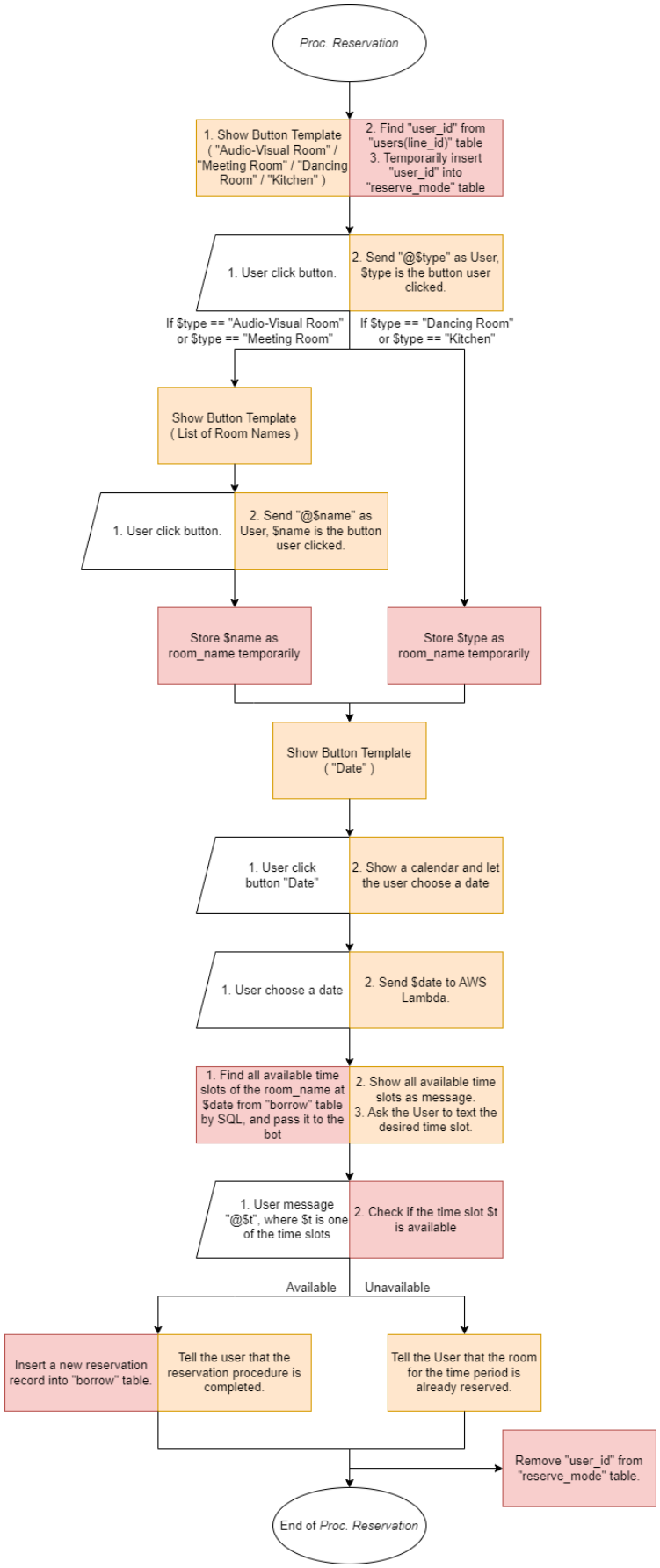

Fig. Flowchart of Procedure Reservation

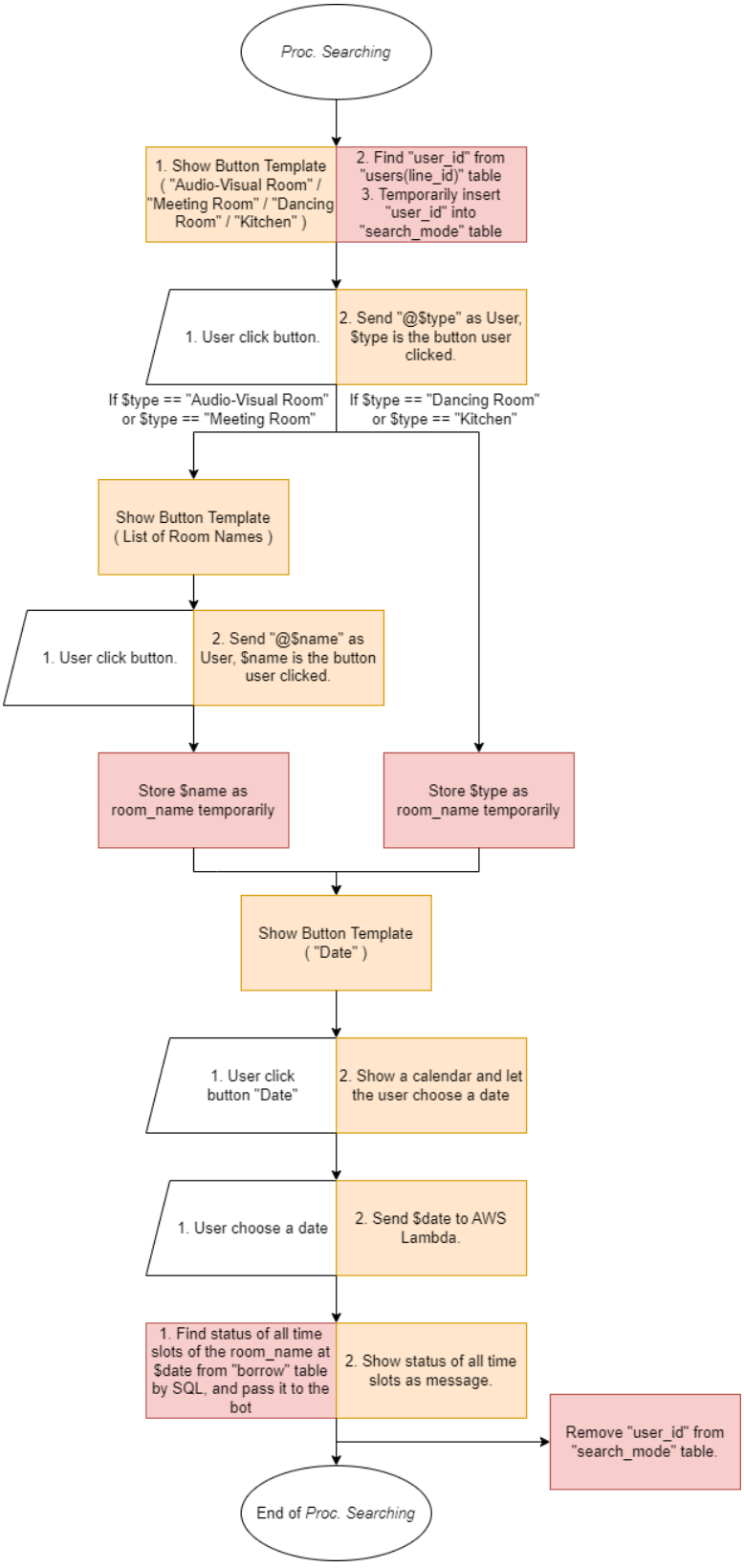

Fig. Flowchart of Procedure Searching

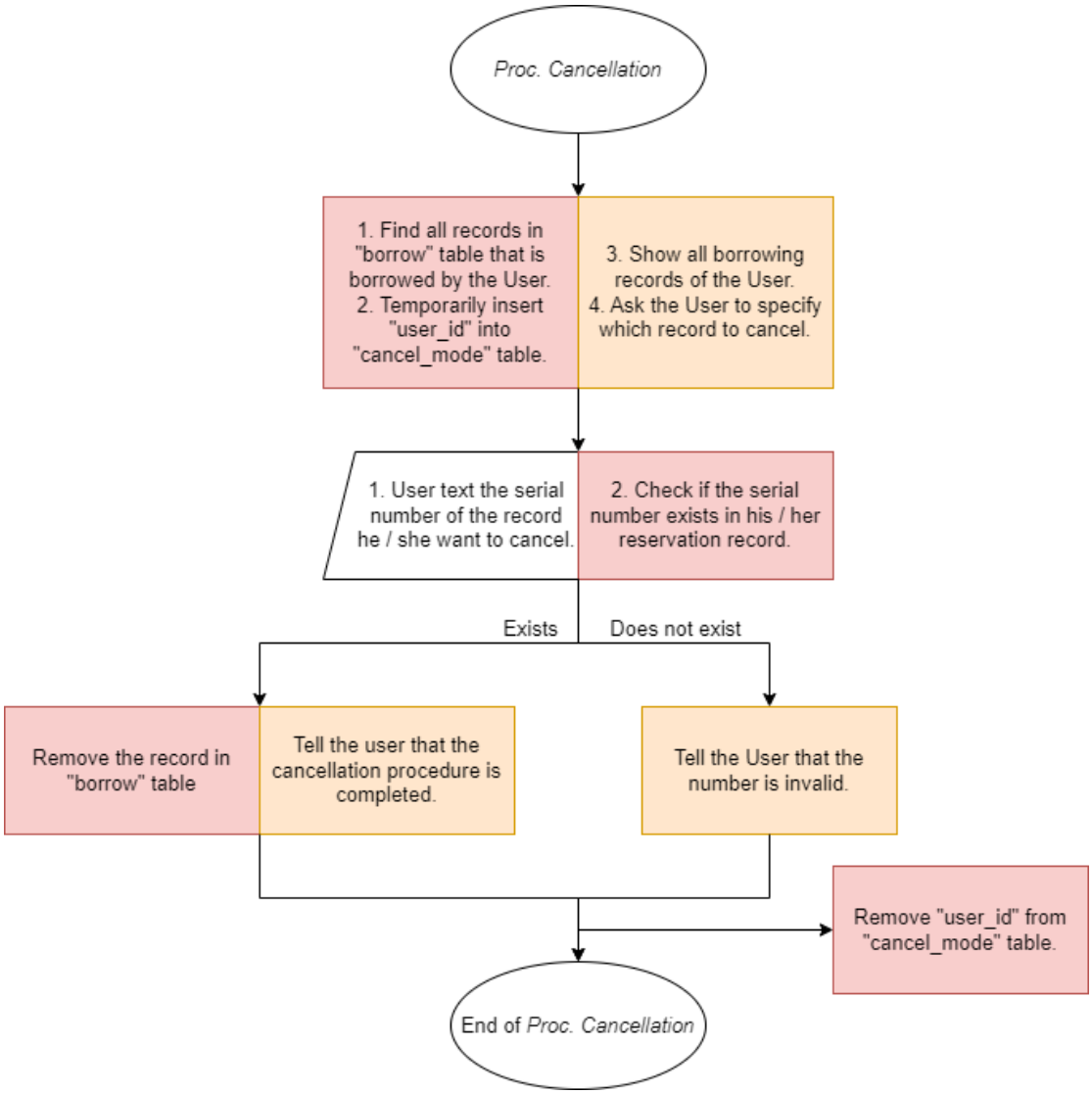

Fig. Flowchart of Procedure Cancellation

Demo Screenshots

Fig. New user are asked to provide his or her information. User need to provide information in the right format.

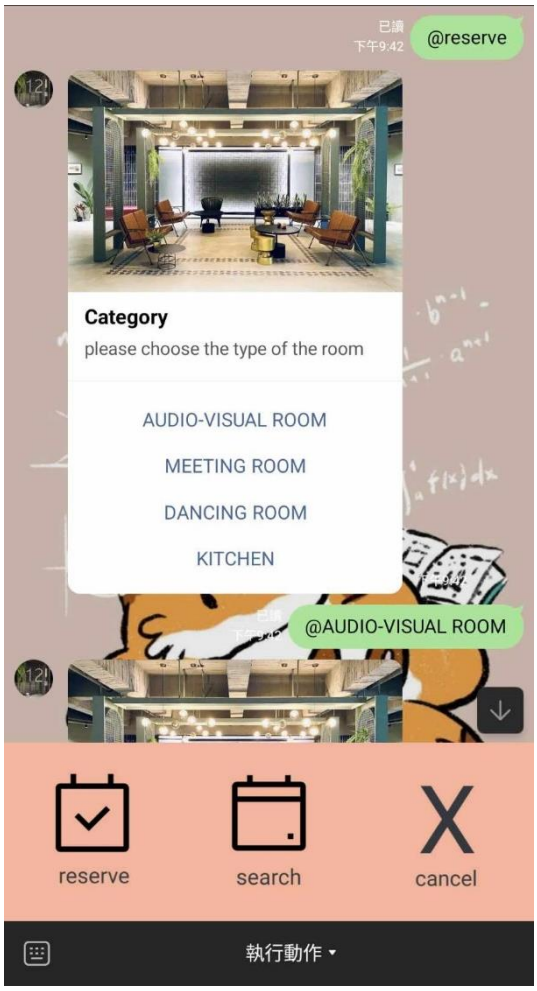

Fig. Rich menu is the three buttons(reserve, search, cancel) in the bottom of the screen. "@reserve" is sent if "reserve" is clicked, and a button template(Category) will be provided by LINE BOT.

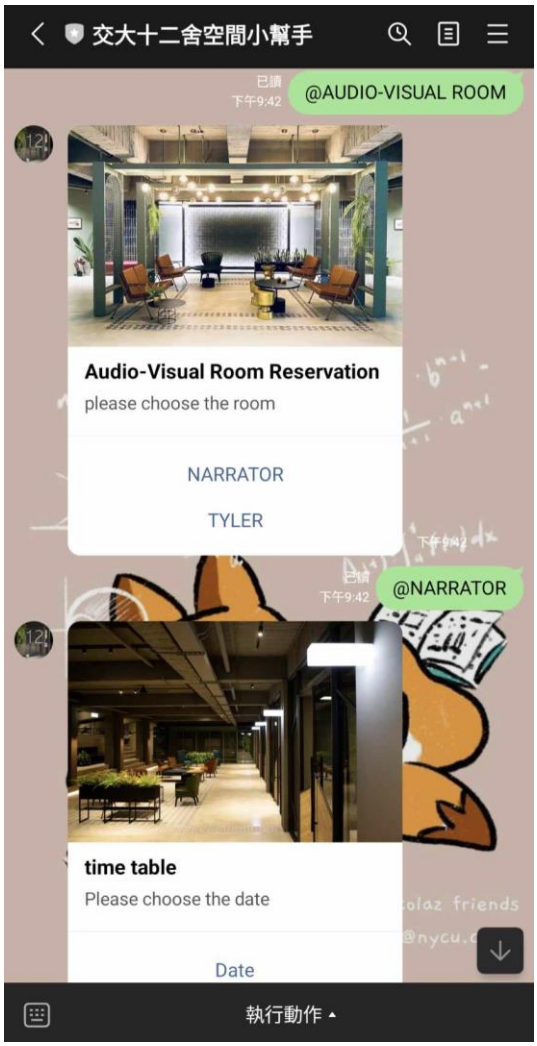

Fig. Another button template will be provided if there are multiple rooms in the selected category. The name of the room will be sent to the chat after clicking.

Fig. A calendar will be displayed if the user clicks on the "Date" button.

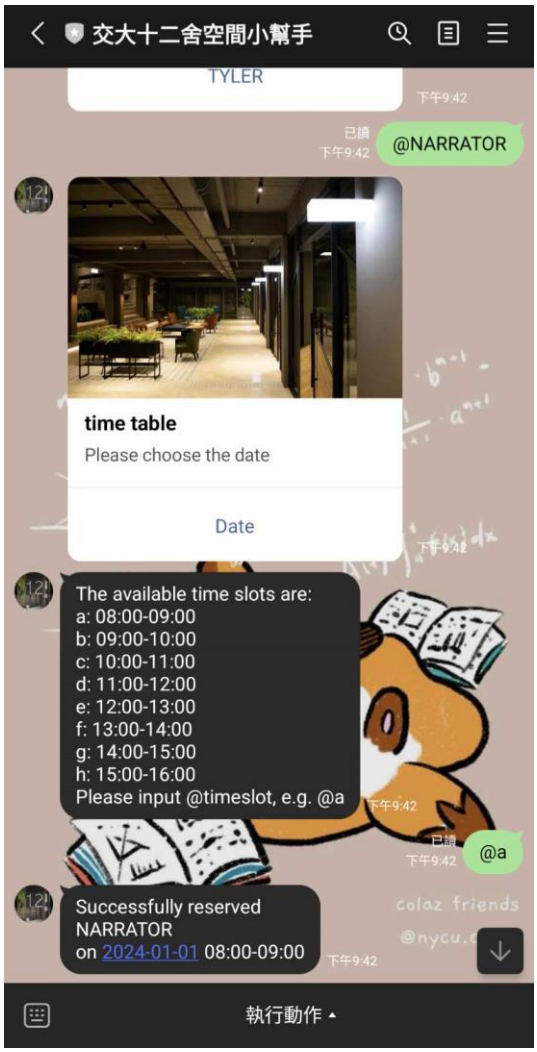

Fig. The borrowing status of the specified room for the whole day will be displayed. The last thing user need to do is to specify their desired time slot.

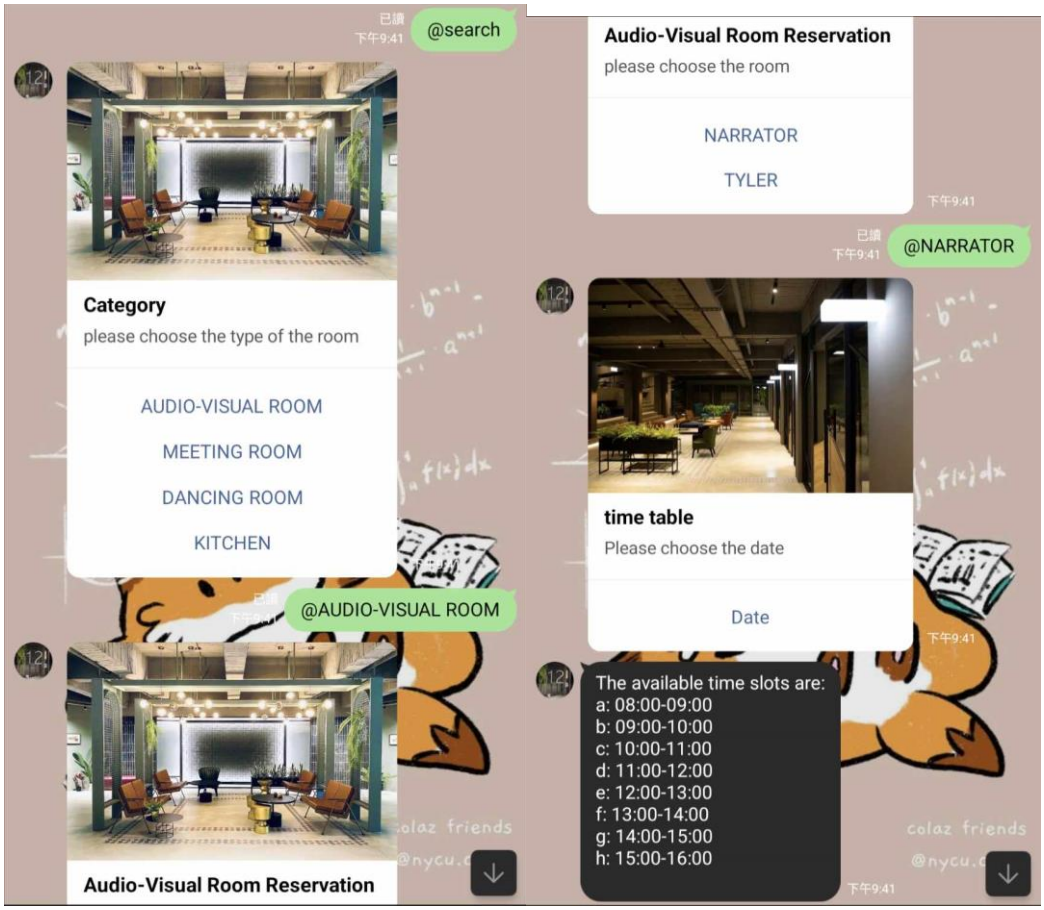

Fig. The interface is consistent in searching procedure.

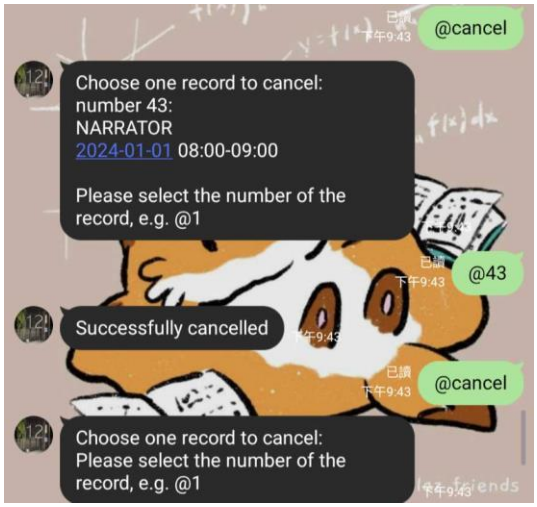

Fig. The cancellation is done by texting the serial number of the reservation.

Open-Sourced Code

Source code of the AWS Lambda function is included in this repo. Feel free to modify our system to suit your purpose!