Intelligent Reservation System

**Abstract:** In the wake of coronavirus pandemic, social distancing enforcement is a must. Establishments (e.g. shops, offices, clinics, education centres, etc.) must operate at reduced capacity to keep patrons adequately apart. Unfavourable lines/crowd may form at the entrance which poses transmission risk. Hence, establishment must accept reservations in order to optimize capacity while ensuring safety as well as saving time for patrons by reducing line size. This project proposes solution to this problem by utilizing rule-based reservation system, and search algorithm for reservation allocation. In addition, this project also attempts association analysis to discover new knowledge based on patron’s queries and reservations.

1. **Business Case**

A certain type of reservation is relevant to a certain establishment. For example, reservation to a restaurant is handled differently to reservation to a retail shop or return to office. Before a restaurant reservation slot is given to a patron who requests it, it is important to first ensure that there in enough table and seat fulfil the request. In contrast, for general entry (to shops/malls or office), it is necessary to ensure that any given time, the maximum number of patrons must not exceed the pre-set setting. Solution to this problem is referred to allocation system throughout this project.

In addition, in order to accept reservations (or bookings), establishment limitation in certain aspects must first be understood. This comprises (but not limited to), maximum capacity (number of patrons), operating hours (and days), maximum group size, booking for certain days in advance. This project uses rule-based system to qualify patron queries into confirmed reservation.

The project team manage to gather requirements from three establishments that fall under two business natures. First, Anjana Kitchin is a restaurant. Restaurant’s reservation is exclusive to another reservation within the same establishment. Meaning, if a patron requests for a table to seat 4 patrons, the reservation system must find a table that is able to seat at least 4 patrons. This table is final until the reservation is ended, two or more groups would not share table. Second, Animani Porchalai is jewellery shop, and TechSource System is office setting. These reservations are inclusive with one another within same establishment. Meaning, the patrons are not sub-grouped in tables or sub-location.

We propose that the reservation be handled by a software agent. The intelligent agent searches and slots patrons into their preferred slot, or to suggest alternative slots.

Agent interacts with patrons in real-time through a chatbot. Interaction may be bidirectional, as the knowledge base (KB) may be provided from establishment (e.g. informational KB or conditional KB; establishment may require patrons to show up with certain dress code, bring certain documents, admittance procedures that must be known beforehand, etc.) – hence, a solution. The solution may be tailored to suit applications in different areas (e.g. hospitality, retail, office, education, healthcare, etc.).

Every hit (a request: new/change/cancellation/query) could be registered. At the end of a period, a report will be produced. From the report, various analytical and intelligent approaches may be used to understand patron/market behaviour (Knowledge Discovery). Beyond pandemic, the solution may continue to provide business value as it may be configured to original capacity or even an expansion of business.

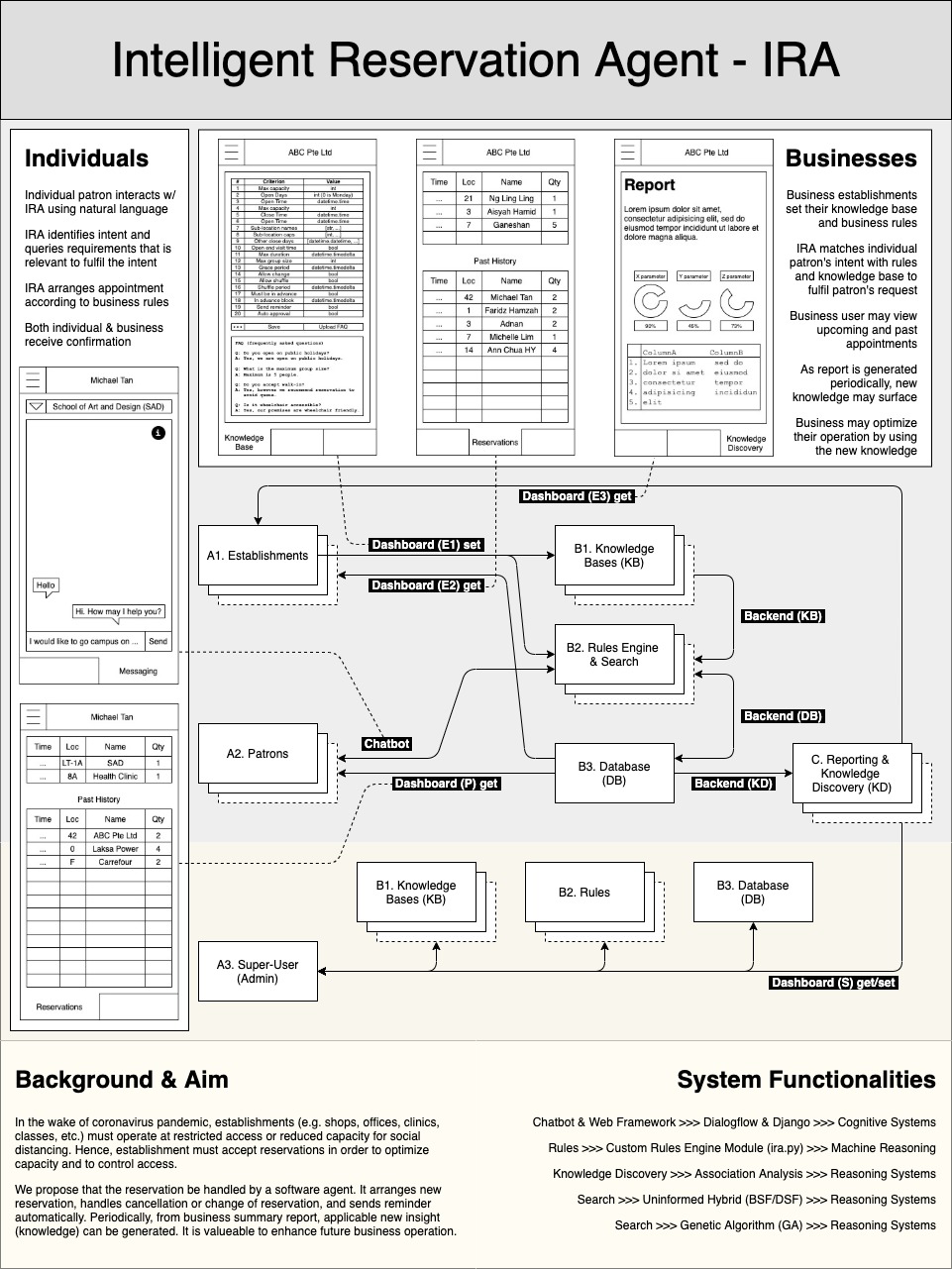
1. **Product Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **Solution Name** | **Intelligent Reservation Agent** |  |  |
|  |  |  |  |
| Contribution to Business Strategy | With Intelligent Reservation Agent, the Patron are able to: |  |  |
|  | 1. Book a reservation |  |  |
|  | 2. Change Reservation date to a new date |  |  |
|  | 3. Cancel Reservation |  |  |
|  | 4. Check Reservation |  |  |
|  | 5. Know the company details. These activities are carried out with guided approach by the app & Bot making the Patron life easier. |  |  |
|  | On the other hand, the establishment can track their customer’s visit through the analytics report generated based on reservations made. The Establishment will be able to cater to customers with Safety procedures especially in this Pandemic situation |  |  |
| Benefits | 1. Saving Labour cost |  |  |
|  | 2. Able to plan the staff shifts ahead and Attend to customers need in the given Timeframe. |  |  |
|  | 3. Better hospitality delivered to customers with peace. |  |  |
|  | 4. For Restaurant: Pre-booking enables to prepare the kitchen to cater to number of visitors |  |  |
|  | 5. For Jewellery Merchant: Attend the regular customers with their expected design. The purpose of customers may vary from purchase, sales or repair work. The Staffs can set up the table according to the customer needs. |  |  |
|  | 6. No queuing of customers can be experienced. |  |  |
|  | 7. Customers can also plan their day accordingly with the reserved timeline. |  |  |
| Timescales | 3 months |  |  |
|  |  |  |  |
| Costs | FOC |  |  |

**Team Members**

|  |  |  |
| --- | --- | --- |
| **Official Full Name** | **Work Items (Who Did What)** | **Email (Optional)** |
|  |  |  |
|  |  |  |
| Januwar Hadi | Rule-based Engine,  GA and Search,  Database Design,  Association Analysis (KD) | [januwar.hadi@gmail.com](mailto:januwar.hadi@gmail.com) |
|  |  |  |
| Rajamanickam Hamsamalini | Chatbot,  Market Research,  Proposal Presentation,  Django-Chatbot Integration,  Database Design | [chamsamalini@gmail.com](mailto:chamsamalini@gmail.com) |
|  |  |  |
| Kevin Chng Jun Yan | Django Backend,  Frontend,  Database Design | [kevinchng@hotmail.com](mailto:kevinchng@hotmail.com) |

1. **System Model**



**Figure 1. Mapped System Functionalities**

1. **System Development & Implementation**
2. **User Interface (UI)**
   1. Chatbot

Google Dialogflow has been trained to be able to detect intents (create, change, cancel and check reservation), and slot filling variables (number of patrons and time). These intents will be passed to rule-engine through **Agent** class.

* + 1. Mainly designed for patrons to interact to create, change, cancel and check reservations
    2. Patrons also are able to inquire general information as listed in establishment’s FAQ.
  1. Dashboard (A1/A2/A3 in figure 1)

Dashboard uses Django Web Framework for these functions:

* + 1. Fall back UI for patrons to create, change, cancel and check reservations.   
       This UI is form based, requiring information for **Agent** class.   
       This UI is available in event where patron chooses not to interact with chatbot.
    2. Patrons may check reservation history and upcoming reservations
    3. Establishment to create and update its knowledge base (KB) B1 (e.g. operating hours, maximum capacity, etc)
    4. Establishment to check reservations and history
    5. Establishment to view reservation and query summary

Dashboard has two view layouts. Patron’s layout is different from Establishment’s.

1. **Rule-Engine and Allocation System (B2)**

Custom IRA rule-engine with dual allocation systems (exclusive and inclusive).

Both rule-engine and allocation system are encompassed in one custom module (ira.py).

* 1. Rule-engine uses IF THEN ELSE components.   
     Rule-engine is arranged as series of sequence of IF THEN ELSE statements.
     1. Default (configuration file: config.ini)  
        e.g. IsOversizedGroup?OversizedGroup:  
        Example above is one rule component which mean: if IsOversizeGroup is equates to true, rule-engine will return OversizedGroup action message to patron (not success). Otherwise, (empty) execute next rule.
     2. In event of “not success”, IRA stores the state of conversation. This state is useful, in event when the patron responds, IRA *remembers* where it left off (as if it understands the context).
     3. Requirements gathered from sponsors enable us to generate default rules sequence (starter rules).
     4. Establishment (business user) may also create customizable rule sequence also known as rules builder.

As rule configuration is separated from the IT system.   
Refer to appendix: user manual for detailed explanation.

Business user may choose different rule and sequence them to their liking.

* + 1. Expandable rule library. It is easier for IT developer to expand rules (generate new rules) as each rule structure is similar.
    2. Rule returns message to patron for:
       1. Patron response needed
       2. Fail message (not success)
       3. Success message
  1. Allocation System  
     Reservation allocation system will attempt to slot patron request in reservation table. There are two cases:
     1. Exclusive (e.g. restaurant). This uses hybrid uninformed search algorithm to search depth and width or search space. Lower cost is assigned to first dimension (time slot in the same day), medium cost is for next dimension (time slot in different day), high cost for the final dimension (different week). Refer to appendix: user manual for illustration.
     2. Inclusive (e.g. general premise entry). This uses genetic algorithm to shuffle requests into slots based on fitness function.   
        Prior to executing the algorithm, patron requests are binned into slots. Refer to appendix: user manual for illustration.

1. **Resources and States Management (B3)**

Resources are information stored in database. Database uses multiple tables to store information. Total there are six tables as explained below:

* 1. Conversation/session is stored in database as **query table**. This is known as state. State is important to keep the context of rule. If patron query does not qualify a certain rule (fail message), IRA remembers so that it would not try to qualify from the beginning of rule sequence again, as explained in 2 above.
  2. Resources:
     1. KB essential to rules is also stored in database (e.g. operating hours) as **establishment table**.
     2. Patron identifier is stored in database as **patron table**.
     3. Lastly, reservations (with status e.g. confirmed, cancelled, etc.) are also stored in database as **reservation table**.
     4. Knowledge (D) is stored in database as **query** and **reservation report tables**. Refer to appendix: user manual (H) and (I) for example.

**Conclusion**

This project solution is limited to entirely exclusive and entire inclusive cases. However, our team discovered that there is a case where it is neither (e.g. classroom). It is still possible however to use solution in this project to cover classroom setting, as one classroom in inclusive within itself. It is to consider one classroom as an establishment.

Feedback from sponsors:   
Project aim has been reached. Patron now can:

* Book a reservation
* Cancel a reservation
* Change the reservation
* Check reservation and history

Users can:

* Register new establishments
* Automate scheduling
* View analytic report

Product is user-friendly. The requirements to do reservation can be fulfilled through:

* Chat Bot
* Web Form

1. **Appendix: Proposal**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | |  |
| Project Name | **Intelligent Reservation Agent** |  |  |
| Project Sponsor | Anjana Kitchen Pte Ltd | Project Team | Januwar Hadi |
|  | Thanaraj,  Director,  Animani Porchalai Pte ltd,  (+65 93809703) |  | Rajamanickam Hamsamalini |
|  |  |  | Kevin Chng Jun Yan |
| Date of Project Approval | Aug-17 | Last Revision Date | Oct-19 |
| Contribution to Business Strategy | With Intelligent Reservation Agent, the Patron is able to 1. Book a reservation 2. Change Reservation date to a new date 3. Cancel Reservation 4. Check Reservation 5. Know the company details These activities are carried out with guided approach by the app & Bot making the Patron life easier. |  |  |
|  | On the other hand, the establishment can track their customer’s visit through the analytics report generated based on reservations made. The Establishment will be able to cater to customers with Safety procedures especially in this Pandemic situation |  |  |
|  |  |  |  |
| Options Considered | For Restaurant |  |  |
|  | For Shops |  |  |
|  | For Training Center and Customer Meeting |  |  |
|  |  |  |  |
| Benefits | 1. Saving Labour cost 2. Able to plan the staff shifts ahead and Attend to customers need in the given Timeframe.  3. Better hospitality delivered to customers with peace. 4. For Restaurant: Pre-booking enables to prepare the kitchen to cater to number of visitors 5. For Jewellery Merchant: Attend the regular customers with their expected design. The purpose of customers may vary from purchase, sales or repair work. The Staffs can set up the table according to the customer needs. 5. No queuing of customers can be experienced. 6. Customers can also plan their day accordingly with the reserved timeline. |  |  |
| Future Scope | Purpose of reservation For Restaurant: Birthday, Anniversary, Team meeting For Jewellery Shop: Purchase, Sales, Repair work |  |  |
| Timescales | 3 month |  |  |
| Costs | FOC |  |  |

1. **Appendix: User Manual**

In order to run integrated system in local machine, you have to

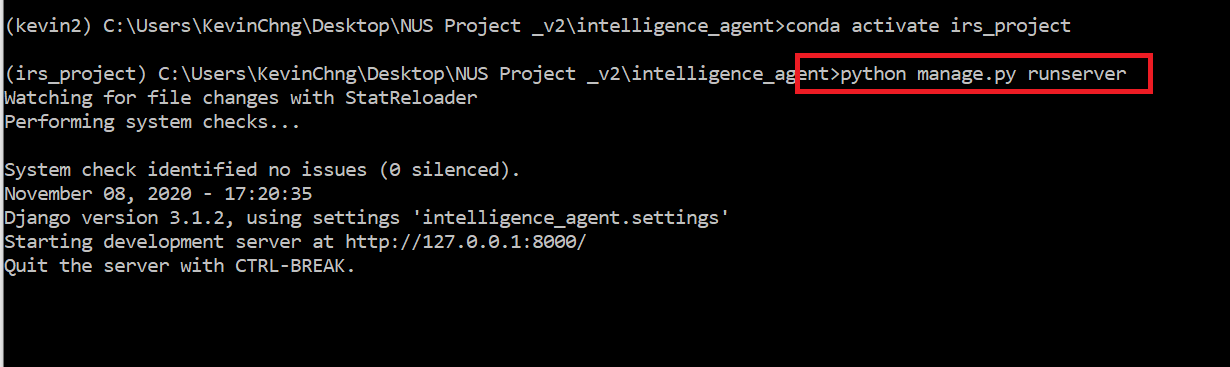
1. Setup "ngrok" for securing URL to your localhost server through any NAT or firewall.
2. Enable "Facebook Authentication", "Google Map Platform Authentication" and "Diaglowflow Google Authentication"
3. Install required packages for your backend django

please follow the guidance below to set up the above 3 prerequisite for running our integrated system in local machine

| **No** | **Area** | **Reference** |
| --- | --- | --- |
| 1 | Ngrok Activation | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/ngrok_activation.md) |
| 2 | Facebook Authentication | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/facebook_authentication.md) |
| 3 | Google Map Platform Authentication | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/google_map_platform_authentication.md) |
| 4 | Dialogflow – Google Authentication | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/dialglowflow_google_authentication.md) |
| 5 | Install Required Packages | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/install_django_package.md) |
| 6 | Configure Django Files | [Reference](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Documents/configure_django_file.md) |

Now you have completed all prerequisite, In your Anaconda prompt,

1. activate the virtual environment "conda activate irs\_project"
2. Go to the downloaded path of project code, "cd path\_of\_your\_project\_code"
3. run the django server, "python manage.py runserver"

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/django_run.png)

**Figure 2**

Other than running the integrated system in local host, you are also allowed to merely run our rule engine for you to understand the mechanism behind our scheduling process.

**Rules Engine & Reservation Allocation System:**

**Rules Engine**

Rules Engine is structured as to separate business logic from the IT system. Business logic and database configuration may be modified in **config.ini**.

Business logic is segmented by intent. Each intent has it own business logic rules sequence.

# =<Rule>?<Action-True>:<Action-False>

If a Rule executes True, <Action-True> will be returned, <Action-False> is otherwise.  
If an is absent, IRA will proceed to execute next rule in sequence. If there is an integer in place of <Action>, execution will jump to skip to as many times as the integer number.

Refer to section K for example.

IT user only needs to interact with Agent class to access the rules engine. Agent class accept four positional arguments: *session*, *intent*, *patron*, *establishment* *patron* and *establishment* information may be obtained from patron and establishment tables. *intent* must match section in **config.ini**.

Session (*session*) is arbitrary identifier that is unique for each conversation with **ira.py**. A conversation may have series of interactions with same session. Session stores state of conversation.

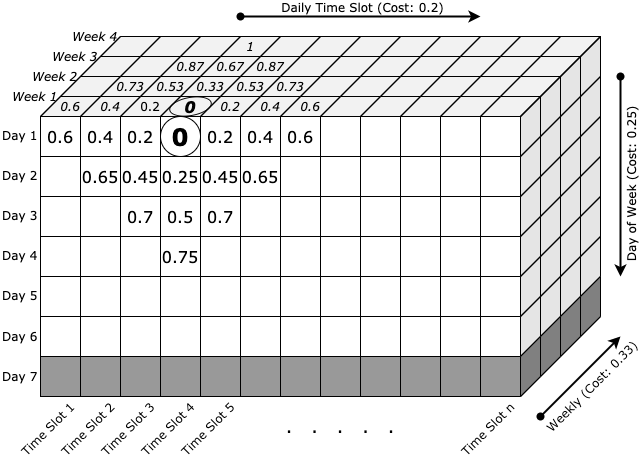
Keyword arguments *time\_in* and *n\_person* are to indicate what time and number of guests patron would like the reservation for.

Subsequent interaction requires contextual *selection* keyword argument.

#### Reservation Allocation System:

A. Searcher

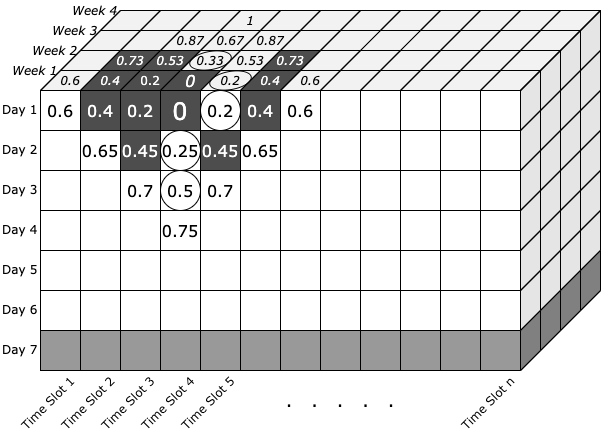
The Searcher class is used for exclusive scenarios (e.g. restaurants). It bins patron's time\_in request in a location in search space (the stacked cubes). The location is the small cube (time slot 4, day 1, week 1), as shown in picture in example below.

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/searcher1.png)

**Figure 3**

Distance along time slot axis costs less that day axis that costs less than week axis. This is in order to give preference (bias) to slots as compared to days as compared to weeks.

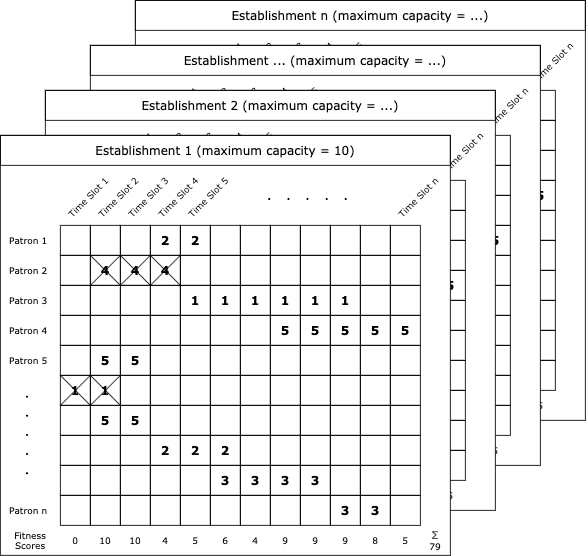
If time slot 4, day 1, week 1 (4/1/1) is available, the ideal situation is reached. Nevertheless, Searcher still captures all locations that is under cost threshold -- thresh, then sort it cost-wise from low to high. The sorted locations is what patrons receive as proposed time slots. It is when the rules action OfferSlots is invoked.

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/searcher2.png)

**Figure 4**

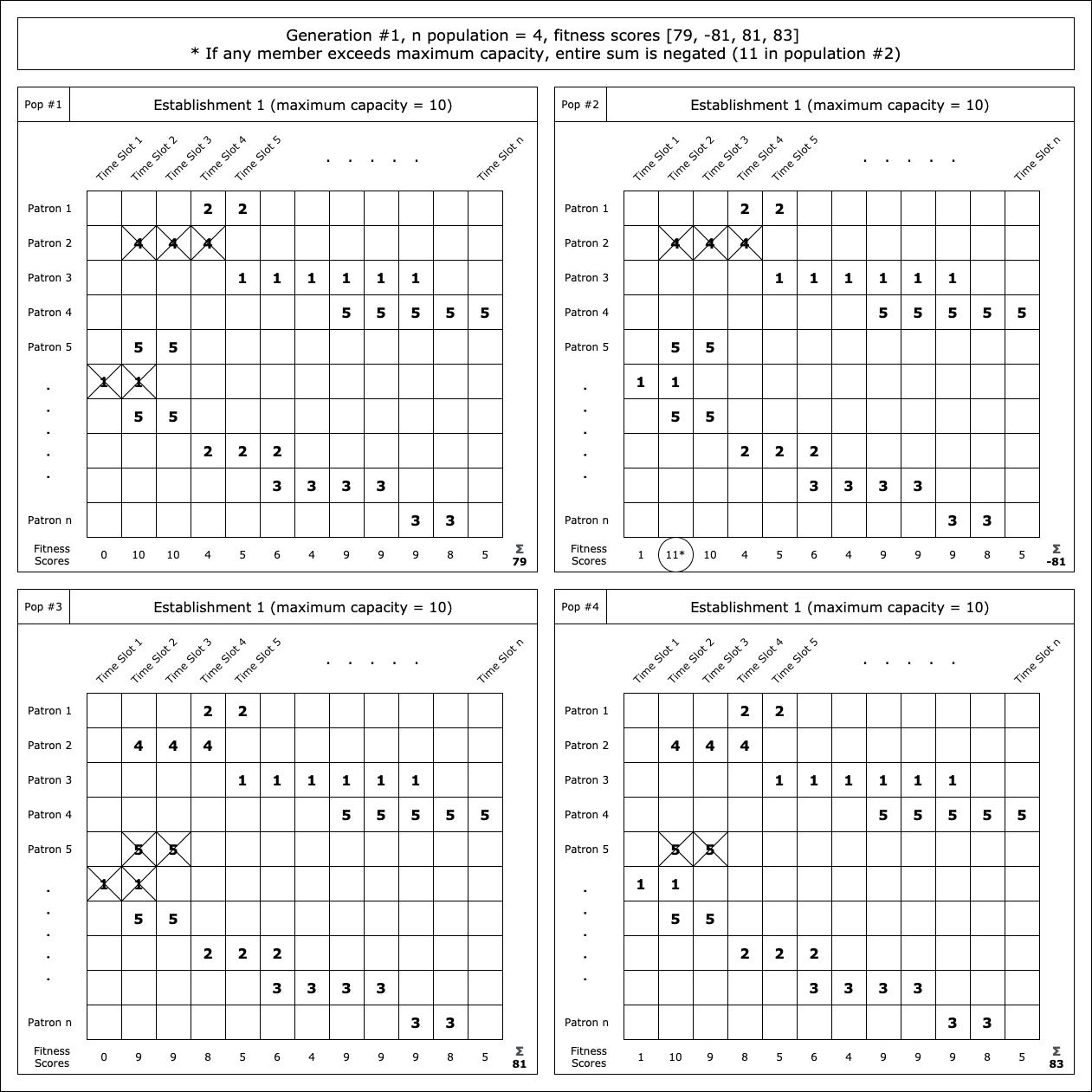
In case when, certain locations already filled up (ones that blacked out), patrons will receive time slot offer most relevant (close) to their request. In example picture above, Searcher will propose 5/1/1, 4/2/1, 4/2/2, 4/3/1, ... (from lowest to highest cost). This search space is exclusive for each establishment's subloc (sub-location). The bottom of the stacked cubes is greyed out to indicate that they are not open on that day.

B. Genetic Algorithm (GA)

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/genetic1.png)

**Figure 5**

GA is used for inclusive scenario (e.g. general entry to a premise). The search space for each establishment is in 2D grid. In order to avoid calculating all possible permutation, GA is used. Figure below depicts the usage of fitness function that sums up fitness score. The fitness score is flipped to negative if threshold (maximum capacity) is exceeded.

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/genetic2.png)

**Figure 6**

The actual implementation's n population is the total request on a particular day. Default batch size is 20. Default crossover and selection point are at half or batch size. Default crossover rate and mutation rate is one quarter. Maximum generation is 1000, however, early termination (convergence) is allowed. Convergence is defined to have been achieved if at least half of maximum generation have been executed, and none of its last fitness scores is negative, and 95% of batch member are identical.

#### Agent and HouseKeeping Classes

Rules and its attribute classes are contained in **ira.py**. Two classes that may be used are:

1. Agent class that handles all interactions with rules. After Agent object is created. Use check\_rules() function to run. The method will return a tuple, which its index 1 holds the response message.
2. HouseKeeping class that handles maintenance of resources (e.g. database entries)

Configuration file **config.ini** has link to database (and its tables) as well as rules builder. As a text file, It may be modified to:

1. Configure custom rules or rules order, and
2. Path to link to different database.

In addition, there is **util.py** that is useful for initialization and simulation.

Dependencies: Install packages as listed in **rules\_requirements.txt** (to preferably Anaconda virtual environment). Replace <env> with your desired virtual environment name.

$ conda create --name <env> --file rules\_requirements.txt -y

Once installation completed, activate the virtual environment to try out subsequent demonstration in either jupyter-lab (install separately), IDE or python shell.

$ conda activate <env>

Optional: To install jupyter-lab from virtual environment.

(<env>)$ conda install jupyterlab -y

Guide sections:

1. Section B, C, D cover reservation relevant to exclusive scenarios (e.g. restaurant, where group of patrons is allocated exclusive space for their own group).
2. Section D, E, F cover reservation relevant to inclusive situation (e.g. mall or other large communal space, where as many patrons may be admitted as long as within capacity threshold as any given time).
3. Section G, H, I: Reporting (Knowledge Discovery)
4. Section J: How to generate simulated data
5. Section K: Configuration and rules builder
6. Section L: Rules and actions library

Code for this demo is available in **test.ipynb**.

(<env>)$ jupyter-lab test.ipynb

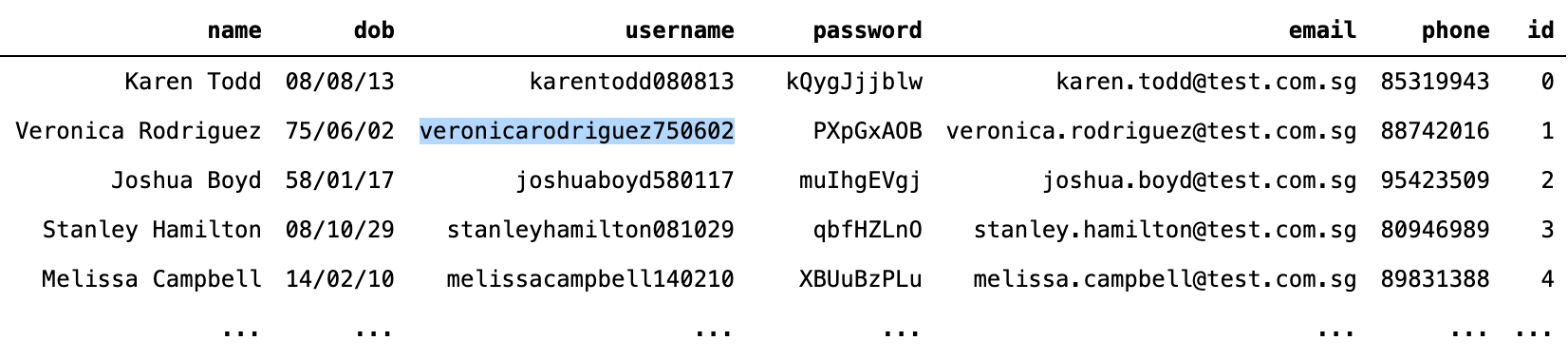
#### A. Initialization/Quickstart

In addition to **ira.py** module and configuration file **config.ini**, there must be a database file.

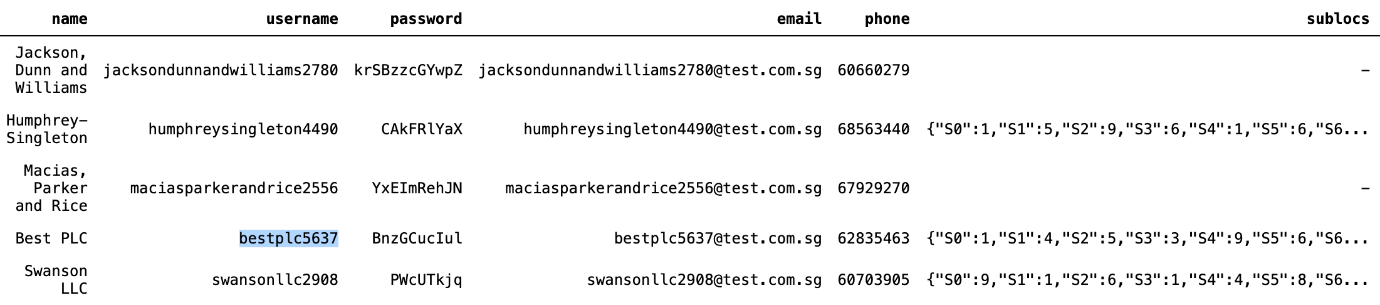
As a starter, there is **dummy.db** for the purpose of self-help demonstration and incremental integration to chatbot.

In order to do anything with a reservation, patron and establishment identifiers must first be obtained.

> util.load\_table('dummy\_pat', 'dummy.db')

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/patrons.png)

> util.load\_table('dummy\_est', 'dummy.db')

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/establishments.png)

Arguments dummy\_pat, dummy\_est and dummy.db are default patron table name and default database name.

Once patron and establishment are known, Agent can be invoked. Argument intent must match the section title in **config.ini**. IRA will execute according to rule sequence in that section.

#### B. Create New Reservation

Choose establishment whose sublocs is NOT empty (-). Sublocs is akin to restaurant table or lounge premise which groups are exclusive to one another.  
It is stored as json string in the database.

> from ira import Agent

>

> session = '001'

> intent = 'NewReservation'

> patron = 'veronicarodriguez750602'

> establishment = 'bestplc5637'

Session (session) is arbitrary identifier that is unique for each conversation with **ira.py**. A conversation may have series of interactions with same session. Keyword arguments time\_in and n\_person are to indicate what time and number of guests patron would like the reservation for.

**IMPORTANT:** Please ensure that the **time\_in** is a future time (preferably few days in future and within business hours as default rules sequence has such requirements). In any case where the response deviates from what shown in this document, refer to section K and L for explanation.

> time\_in = '20/11/11\_12:34'

> n\_person = '2'

> response = Agent(session, intent, patron, establishment, time\_in=time\_in, n\_person=n\_person).check\_rules()

> print(response[1])

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/new1.png)

As IRA offers options, patron chooses one as selection.

> selection = '20/11/11\_12:30'

> response = Agent(session, intent, patron, establishment, selection=selection).check\_rules()

> print(response[1])

[New 2](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/new2.png)

As IRA returns tuple, the confirmation message is at index 1.

#### C. Change Existing Reservation

Changing reservation requires matching intent as in **config.ini**. Default is ChangeReservation.

> session = '002'

> intent = 'ChangeReservation'

> patron = 'veronicarodriguez750602'

> establishment = 'bestplc5637'

>

> response = Agent(session, intent, patron, establishment).check\_rules()

> print(response[1])

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/change1.png)

Patrons would choose which existing reservation he/she would like to change. Option (1) is the one created from previous step (section B). Pass the selection as keyword argument selection.

> selection1 = '20/11/11\_12:30'

> response = Agent(session, intent, patron, establishment, selection=selection1).check\_rules()

> print(response[1])

[Change 2](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/change2.png)

IRA asks time and number of persons the reservation would need to be changed for. Pass them as respective keyword arguments.

> time\_in = '20/11/11\_13:00'

> n\_person = '5'

> response = Agent(session, intent, patron, establishment, time\_in=time\_in, n\_person=n\_person).check\_rules()

> print(response[1])

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/change3.png)

IRA responds with list of time slots, and patron to choose from the proposed list.

> selection2 = '20/11/11\_13:00'

> response = Agent(session, intent, patron, establishment, selection=selection2).check\_rules()

> print(response[1])

[Change 4](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/change4.png)

IRA responds with success message.

#### D. Cancel Existing Reservation

Cancelling reservation requires matching intent as in **config.ini**. Default is CancelReservation.

> session = '003'

> intent = 'CancelReservation'

> patron = 'veronicarodriguez750602'

> establishment = 'bestplc5637'

> response = Agent(session, intent, patron, establishment).check\_rules()

> print(response[1])

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/cancel1.png)

IRA responds with list of existing reservations. Patron is to choose which to cancel.

selection = '20/11/11\_13:00'

response = Agent(session, intent, patron, establishment, selection=selection).check\_rules()

print(response[1])

[Cancel 2](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/cancel2.png)

IRA confirms that reservation has been successfully cancelled.

#### E. Waitlist Reservation

Establishments without sublocs accept ballots at least one day in advance. In order to ballot, use establishment whose sublocs is empty (-). See establishment table in section A.

> session = '011'

> intent = 'NewReservation'

> patron = 'veronicarodriguez750602'

> establishment = 'maciasparkerandrice2556'

>

> time\_in = '20/11/12\_12:00'

> n\_person = '2'

> response = Agent(session, intent, patron, establishment, time\_in=time\_in, n\_person=n\_person).check\_rules()

> print(response[1])

[Ballot 1](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ballot1.png)

IRA responds with suggestion that fits into establishment's one time slot.

selection = '20/11/12\_12:00'

response = Agent(session, intent, patron, establishment, selection=selection).check\_rules()

print(response[1])

[Ballot 2](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ballot2.png)

As patron confirms with selection, IRA sends receipt message for acceptance of request. See section G for detail on how the request is confirmed or rejected.

Alternatively, patron may indicate time\_out for a request for block of multiple and continuous time slots instead of just one.

> time\_in = '20/11/12\_12:05'

> time\_out = '20/11/12\_15:20'

> n\_person = '5'

>

> response = Agent(session, intent, patron, establishment, time\_in=time\_in, time\_out=time\_out, n\_person=n\_person).check\_rules()

> print(response[1])

[Ballot 3](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ballot3.png)

> selection = '20/11/12\_12:30'

> response = Agent(session, intent, patron, establishment, selection=selection).check\_rules()

> print(response[1])

[Ballot 4](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ballot4.png)

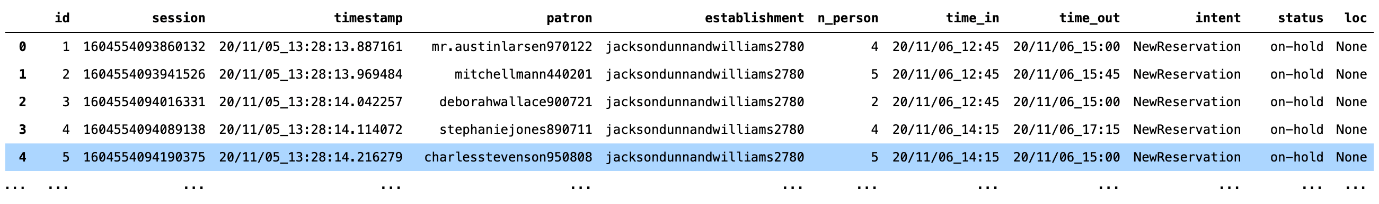
#### F. Waitlist Reservation Change and Cancellation

Default rule sequence does not allow change of waitlisted reservation. Cancellation, however, is allowed.

#### G. Waitlist Reservation Confirmation

Once a successful request is submitted, waitlisted request status is indicated as on-hold.

> util.load\_table('dummy\_rsv', 'dummy.db')

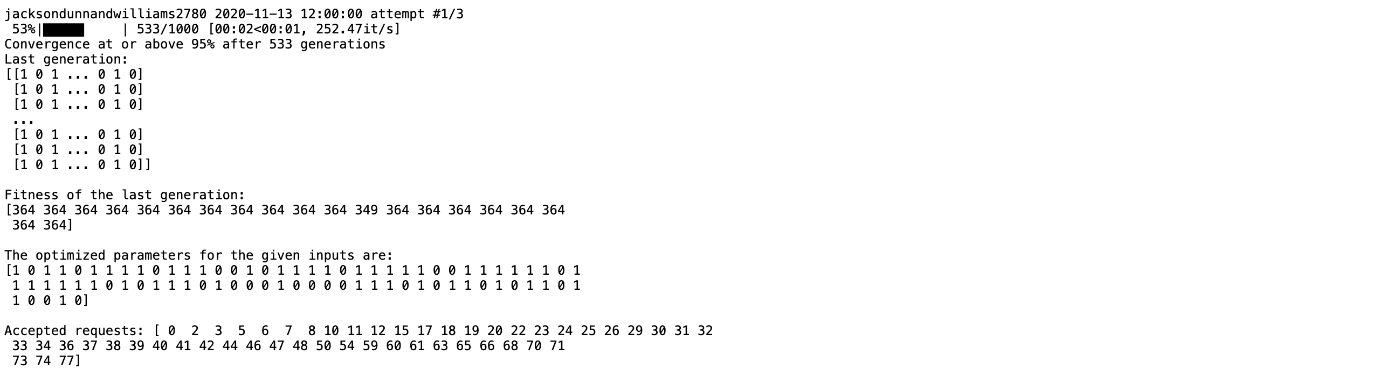
[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/on_hold.png)

In order to handle waitlisted request, a service must be subscribed to run periodically (daily). It is for an algorithm to assign each waitlisted request a confirmation or rejection. This is where HouseKeeping class is useful.

> from ira import HouseKeeping

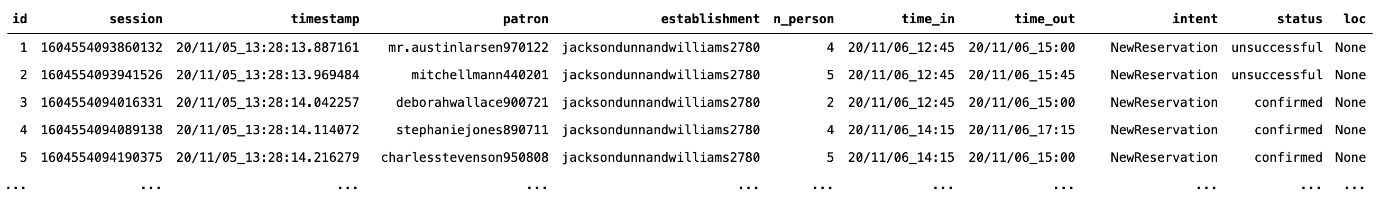
>

> HouseKeeping().genetic\_algorithm\_check(all\_days=True, plot=True)

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ga1.png) [](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ga2.png) [](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/ga3.png)

If all\_days set to True, the algorithm will run for all entries in the reservation table. If False, it will only run for the next business day. Default False. If plot is set to True, it will print the progress and plot fitness and other metrics. Default False. These metrics will be stored in reservation report database, default dummy\_rsv\_report. Refer to section H for more details.

> util.load\_table('dummy\_rsv', 'dummy.db')

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/unsuccessful.png)

If the algorithm finds solution, each request will be reassigned from on-hold to as either confirmed or unsuccessful. Otherwise, all reservation on that day will be assigned as walk-in.

#### H. Reservation Report

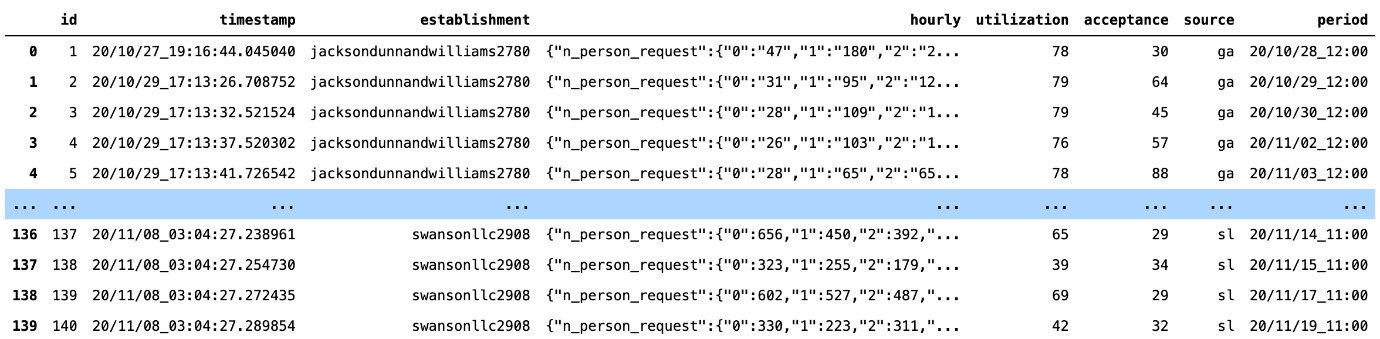
Running Reservation HouseKeeping().genetic\_algorithm\_check() will automatically generate reservation report for reservations whose status is on-hold -- the inclusive scenario. Report may also be generated for exclusive scenario.

> HouseKeeping().linear\_check(all\_days=True)

If all\_days set to True, the algorithm will run for all entries in the reservation table. If False, it will only run for the next business day. Default False.

The report is stored in reservation report table

> util.load\_table('dummy\_rsv\_report', 'dummy.db')

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/rsvrep1.png)

Source ga indicates that the report is generated by HouseKeeping().genetic\_algorithm\_check(), sl indicates HouseKeeping().linear\_check(). Entries in hourly column are hourly outlook, in json string format. Each may be tabulated as shown below.

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/rsvrep2.png)

Here the business user can do association analysis to discover new knowledge relating to reservations. It is apparent that the business least busy hours are near the opening and closing hours.

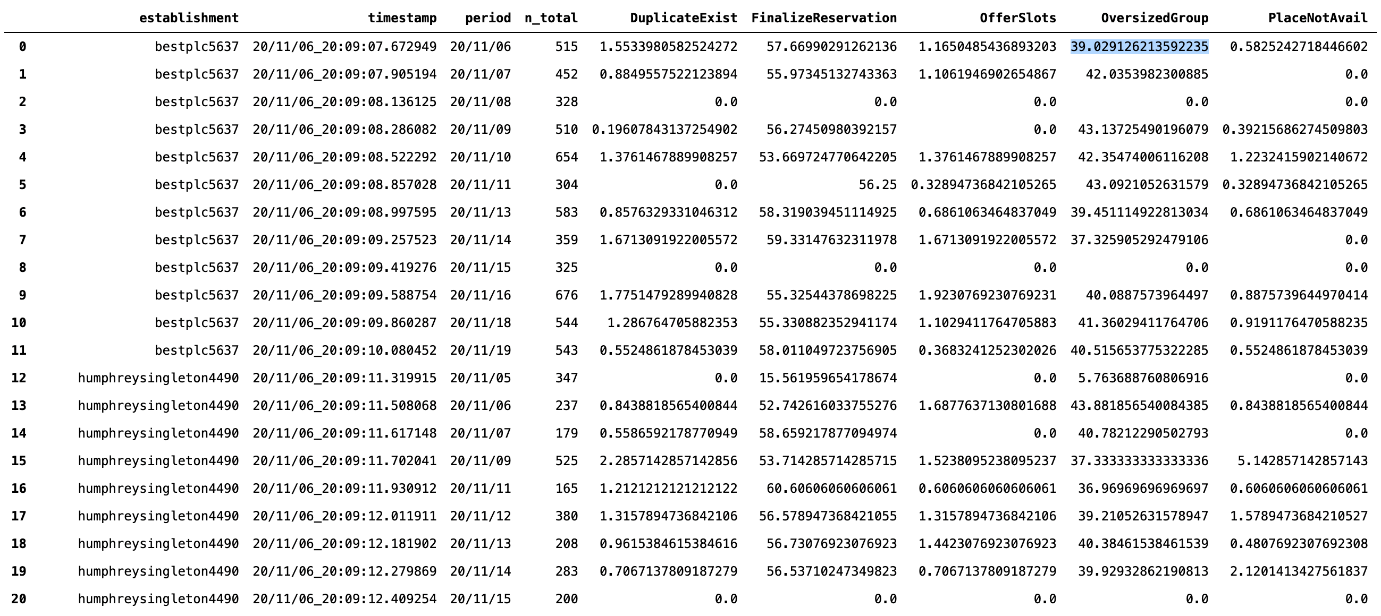
#### I. Query Report & Reservation Status Check

In order to gain insight of patron queries, run code below:

> HouseKeeping().summarize\_query(plot=True)

If plot set to True, it prints progress. Default False. Below is sample of what query report looks like.

> util.load\_table('dummy\_qry\_report', 'dummy.db')

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/qryrep.png)

n\_total indicates the number of queries an establishment receive in a day. Values in DuplicateExist, OversizedGroup, PlaceNotAvail, and so fort indicates percentage. The percentage out of n\_total that the patron query terminates at. The termination is last Action by IRA. For more details of Action, see section L.

For example, for establishment bestplc5637 reporting period 20/11/06, out of total 515 patron queries, 39.029126213592235 percents terminates at OversizedGroup. Meaning, on Nov 6 around 201 queries (~39%) are to reserve for a group larger than allowed group size.

Here the business user can do association analysis to discover new knowledge relating to queries. Here we see the figure around 40% for all establishment for OversizedGroup is because the simulated data randomly generates 1 to 9 figure as number of patron for all establishment. Hence, we can assume that the simulated data weakness is well reflected in this analysis.

Patrons may use CheckReservation intent to check for their reservation status.

> session = '021'

> intent = 'CheckReservation'

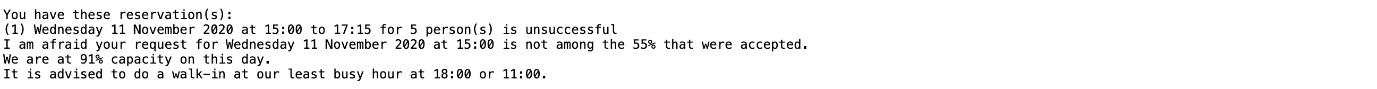
> patron = 'brittanymartinez120418'

> establishment = 'jacksondunnandwilliams2780'

>

> response = Agent(session, intent, patron, establishment).check\_rules()

> print(response[1])

[](https://github.com/KevinChngJY/IntelligentReservationSystem/blob/main/Miscellaneous/Images/check.png)

IRA would give an insight in case the waitlisted reservation eventually becomes unsuccessful. It advises patron the alternatives time for walk-in.

#### J. Simulation Data

To generate new database file run code below:

> import util

>

> util.init\_db()

This generates **dummy.db** database file, and six tables associated to rules engine.  
This also generates dummy patrons (default is 100 patrons) and establishments (default 5).

Default database, table names and number of entries may otherwise be overridden, shown below.

> util.init\_db(db='dummy.db', # database name

tn\_pat='dummy\_pat', # table name patron

tn\_est='dummy\_est', # table name establishment

tn\_qry='dummy\_qry', # table name query

tn\_rsv='dummy\_rsv', # table name reservation

tn\_qry\_report='dummy\_qry\_report', # table name query report

tn\_rsv\_report='dummy\_rsv\_report', # table name reservation report

n\_establishment=5, # number of dummy establishment, enter 0 if not needed

n\_patron=100, # number of dummy patron, enter 0 if not needed

)

To create simulated data, run code below from **util.py** module. This will populate query and reservation tables with random entries up to 3 days.

> util.create\_dummy\_appointments(n\_day=3, tn\_pat='dummy\_pat', tn\_est='dummy\_est')

The database may also be otherwise generated by Django or other web framework that has built-in database manager. As such, table column names and data type (all string) must be identical to what **dummy.db** uses.

Do take note that the database configuration must also match **config.ini** file (if default names are overridden) as it will actually invoke Agent to create entries in database. See section K (under DB) for more explanation.

#### K. Rules Builder

Configuration file **config.ini** has six segments. First two namely:

* DB: Link (relative path from **ira.py**) to database file, and how each table is named.
* Information: FAQ (or otherwise business information) for chatbot knowledge base. This may not be necessary as it would be handled primarily by Dialogflow.

The next four below enable a separation of business logic from system, for the benefit of business user to self-help with rules customization.

* NewReservation: Intent name and rule sequence for new reservation
* ChangeReservation: Intent name and rule sequence for change of existing reservation
* CancelReservation: Intent name and rule sequence for cancellation of existing reservation
* CheckReservation: Intent name and rule for checking of reservation history

Sample section of **config.ini**.

[CancelReservation]

1 = IsTimeout?Timeout:

2 = IsUpcomingExist?ListUpcoming:UpcomingNotExist

3 = SlotNotSelected?SelectionNotValid:

4 = IsValidSelection?CancelSuccess:InputNotValid

[CancelReservation] is segment (intent) name. Agent looks into this segment when CancelReservation string passed as intent.

For line 2 = IsUpcomingExist?ListUpcoming:UpcomingNotExist, number 2 is rule sequence number, rule sequence must start from 1 and be sequential thereafter. IsUpcomingExist is a Rule, which if equates to True, ListUpcoming Action will be returned, otherwise UpcomingNotExist is returned. In case where there is no Action is indicated after a Rule is executed, next in sequence will execute. Alternatively, an integer in place of an Action instructs the execution to jump as many as integer number of rule ahead.

List of rules and actions is explained in section L.

#### L. Rules and Action Library

List of Rules and Actions that business user may choose to build a sequence is explained below:

##### Rules:

1. WithinBookableWindow. Returns True if time\_in is:
   * Within establishment's operating hours (refer to establishment table open\_time and close\_time) and operating days (open\_days)
   * Greater or equal to days in advance (days\_in\_advance)
2. WithinCapacity. True if total number of persons from already confirmed reservations and n\_person at 'time\_in` do not exceed establishment's max\_cap.
3. WithinGroupSize. True if n\_person is equal or less than establishment's maximum allowable groupsize, max\_group\_size.
4. IsDuplicateExist. True if patron already have reservation at same time\_in with same establishment.
5. SlotsAvailable. True if search algorithm is successful to find solution. IRA uses Searcher class in **ira.py** to find list of proposed slots that is under cost threshold, thres. IRA looks for every establishment's sublocs (e.g. restaurant table) if any is available. List is sorted cost-wise, from low to high. Users do not need to invoke Searcher class manually. Success is stored in Searcher's is\_avail property, which is True if proposed list is not empty. Essentially, if search algorithm is able to propose slots based on patron time\_in, this returns True.
6. SlotAvailable. Similar to SlotsAvailable, except that this account for n\_person. In order to avoid assigning large allocation to small request (e.g. a table that can seat 5 to patron of 1), this rule restricts such allocation by factor of 2. As such, Searcher's is\_clashed property is flagged as False and this rule returns False as well.
7. SlotNotSelected. True if patrons do NOT pass one of proposed time slots as selection.
8. IsUpcomingExist. True if there is future reservation in its reservation table.
9. IsValidSelection. True if establishment have record of patrons time\_in in future time, in its reservation table.
10. IsTooSoon. True if patron's time\_in is less than establishment's days\_in\_advance.
11. IsInThePast. True if patron's time\_in is time in the past.
12. WithinOperatingHours. True if patron's time\_in is within establishment's operating hours (refer to establishment table open\_time and close\_time).
13. WithinOpenDays. True if patron's time\_in is within establishment's operating operating days (open\_days). Open days is indicated as series of sequential integers (e.g. 12345 is Monday to Friday)
14. IsTimeout. True if session has exceeded time out (default 1 minute). Conversation with Agent has ended due to inactivity.
15. IsArgsGiven. True if Agent is invoked with keyword arguments.
16. IsHistory. True if patron has reservation record with establishment.
17. IsBallot. True if establishment's reservation system is by ballot/waitlisting.
18. IsTOEarlierThanTI. True if Agent is passed time\_in that is later or equal to time\_out.

##### Actions:

Action returns string message as response.

1. NotImplemented\*: 'Not implemented'
2. OutsideBookableWindow: 'Your request is outside bookable window'
3. OverCapacity: 'I am afraid our full capacity has been reached'
4. OversizedGroup: 'Your request for n\_person number of people is too big. Only up to max\_group\_size people allowed'
5. DuplicateExist: 'You already have upcoming reservation slot(s): (1) ... (2) ... (5) ...'
6. PlaceNotAvail\*: 'I am afraid no slots available'
7. TooSoon: 'Please make reservation days\_in\_advance day(s) in advance'
8. UpcomingNotExist: 'You do not have future reservations'
9. SelectionNotValid: 'Please affirm if the proposed slot is okay' or 'Please choose from proposed slot(s)'
10. InputNotValid: 'Your input not valid'
11. InThePast: 'Your request time is in the past'
12. OutsideOperatingHours: 'Your request time is outside our operating hours'
13. OutsideOpenDays: 'I am afraid we are not open on that day'
14. Timeout: 'Your previous session has timed out. I am afraid you need to start over'
15. NoTimeInAndNPersonGiven: 'Please advise me with your desired reservation time and number of guests'
16. CancelSuccess\*: 'The reservation has been successfully cancelled'
17. Bye\*: 'Thanks for using Ira. Good bye'
18. OfferSlots: 'Available time slot(s): (1) ... (2) ... (5) ...'
19. FinalizeReservation\*: 'Reservation for n\_person on date at time has been confirmed"
20. OnHoldReservation\*: 'Your request for n\_person person(s) on ... to ... is now on-hold. Please status check after cufoff time close\_time.'
21. ListUpcoming: 'Please choose from the upcoming reservation slot(s): (1) ... (2) ... (5) ...'
22. InquireNewSlot: 'How many persons would need to be admitted at which date and time, please?'
23. ChangeReservation\*: 'Previous reservation on ... has been successfully cancelled, and new reservation for n\_person on ... has been confirmed'
24. ListHistory\*: 'You have these reservation(s):' (1) ... (n) ...'
25. ListEmpty\*: 'You do not have any reservation history'
26. InformBallotSlot: 'Time frame that fits your preferences is ... to ... . Please confirm if it's okay for you"
27. UnableToChange\*: 'I am afraid your reservation can only be cancelled. You may apply for the new one'
28. TOEarlierThanTI: 'Exit time should not be earlier than entry time'

\* Terminates session

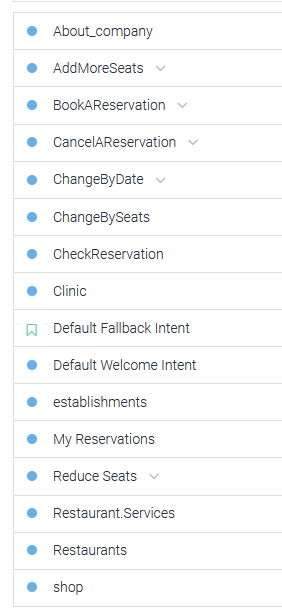
1. **Appendix: Individual Contribution**

Individual Project Report

|  |  |
| --- | --- |
| Your Name: | Rajamanickam Hamsamalini |
| Certificate: | Graduate Certificate in Intelligent Reasoning Systems |

# 1. Your personal contribution to the project.

1. Created an IRA Chatbot using Google Dialogflow with the following intents.



2. Knowledge Base with entities (Entity values for establishment were fetched from database using dialogflow\_betaV2 api). Integrating to Google cloud api to authenticate the entity updation using relevant credentials thereby enabling security mechanism

3. Django- Chatbot integration – The integration involved

Agent training for Intent identification, process the Patron requests through fulfilment to generate response. Response is generated by communicating with rules engine and database in Django.

4. Initial setup for database design for web app and Dialogflow to fetch the values.

2. What you have learnt from the project.

* Usage of Django-dialogflow
* Dialogflow fulfilment processing through webhook
* Agent Training
* Application of Rules based approach
* Analysis using Genetic algorithm & Heuristic algorithm
* Team coordination

# 3. How you can apply this in future work-related projects.

* Scheduling the Trainer slots for training using similar approach in application perspective
* From Knowledge perspective – identify the knowledge, skills and abilities of trainees through assessment structured based on MR and cognitive systems

Individual Project Report

|  |  |
| --- | --- |
| Your Name: | KEVIN CHNG JUN YAN |
| Certificate: | Graduate Certificate in Pattern Recognition Systems |

# Your personal contribution to the project.

* Build main-framework (Django) connected to rule-based engine, google dialogflow, facebook authentication, google map platform authentication, frontend (html+css+jacascript)
* Design form submission framework (other than diaglowflow), user is also allowed to submit their reservation request through form.
* Configure facebook authentication (user/patron) and google map platform authentication, Django-login authentication (business user)
* Design frontend (All dashboard – HTML, CSS, Javascript)

# 2. What you have learnt from the project.

* Scheduling Optimization Reasoning System : Genetic Algorithm
* Build full-stack of AI project : Cognitive System
* Data mining to discover knowledge from database.

# 3. How you can apply this in future work-related projects.

In my working environment, part of my job is to provide consultation to consumer. Although it is mainly on engineering problem, but optimization is one important skills set to optimize the design of engineering.

Individual Project Report

|  |  |
| --- | --- |
| Your Name: | Januwar Hadi |
| Certificate: | Graduate Certificate in Intelligent Reasoning Systems |

# 1. Your personal contribution to the project.

# Proposal Materials (poster and video)

# System Design/Architecture

# Rules Engine

# Allocation System

# Uninformed Search Algorithm (DSF/BSF)

# Genetic Algorithm

# 2. What you have learnt from the project.

# Different forms of knowledge representations

# Other than natural language, semantics, database and formula which we are already familiar with

# Tools readily available, to avoid reinventing wheels (e.g. KIE, Dialogflow, etc.)

# Working in team to divide and conquer problems as well as to brainstorm solutions

# 3. How you can apply this in future work-related projects.

# Reasoning techniques learnt will be very helpful to avoid brute force calculations

---

\*\*\*

End of Report