# UC1: Reserve Tickets

## Level

user-goal

## Primary Actor

User

## Interests

* User:
  + Minimal efforts of reservation.

## Preconditions

* User logged in.
* Normally, User should also view the flights, or other *item categories* before they actually do the reservation.

## Postconditions

* User reserved the ticket (Unpaid)

## Main Success Scenario

1. User starts an reservation.
2. System creates a blank reservation and show it to User.
3. User does an action.
4. System responds to the action. *loop 3-5 until no more action needed*.
5. System presents a summary of the reservation to User.
6. User confirms.
7. System stores the reservation.
8. User pays within time limit (see *Special Requirements*).
9. System acknowledges the payment.

## Extensions

* At any time, when errors occur:
  + System records the error.
  + System informs User of the error.

1. …
2. …
3. The actions are:
   * *Add an item to the reservation*.
     1. User selects the item.
     2. System checks if the item is available.
        + Item available.
          1. System adds the item into the reservation list.
        + Item occupied or expired.
          1. System reports the anomaly.
   * *Remove an item from the reservation*.
     1. User selects an item.
     2. System checks if the item is in the list.
        + Item found.
          1. System removes the item from the list.
        + Item not found.
          1. System reports the anomaly.
   * *View the summary of the reservation*.
     1. System presents a summary page.
     2. User views as they like.
     3. User asks to go back.
     4. System returns to step 3.
4. …
5. …
6. User may decline the reservation. In this case:
   1. System discards all information about the reservation, returning all items in the list.
   2. Reservation aborted.
7. …
8. In the payment phase…
   * Cash payment.
     1. System prompts a payment page to User. The page should contain details of the cost.
     2. User confirms the payment.
     3. System updates the reservation as paid.
   * User may decline to pay, or may have not paid after the time limit. In this case:
     1. System does as in *Extension step 6*.

## Special Requirements

* User should pay within 5 minutes. After the limit, the reservation will be aborted.
* Since the project is only a simulation, we should let the payment succeed under any circumstances. I.e. the tickets are free!
* It’s okay to have little UI.

## Notes

In *Extensions*, situations such as “item not found in the reservation list” are considered anomaly, for in normal cases the user should only see what is valid. I.e. they can only issue *remove* commands on items in the list, and only issue *reserve* commands on items available.

## Open issues

Manipulation of reservations should always be verified with user info. For simplicity concerns, I omitted the requirement. # UC2: View Flights

## Level

user-goal

## Primary Actor

User

## Interests

* User:
  + view clear flights info
  + convenient filtering, querying and intelligent-suggestions.

## Preconditions

* User logged in.

## Postconditions

* User viewed the flights (items)

## Main Success Scenario

1. User queries about flights.
2. System processes the query.
3. System presents the result.

## Extensions

1. Users do the query…
   * *By departure place & destination*.
   * *They can also specify filters, sorting strategies, and so on*.
2. Some exceptions may occur. If so, System should report the error to the user and terminate the query.
   * *System can’t find the places*.
   * *No flights satisfying the requirements exist*.
3. …

## Open Issues

* Support queries of transferring flights. # UC3: Refund

## Level

user-goal

## Primary Actor

User

## Interests

* User:
  + Get back proper amount of money from refunded reservations.
  + Money goes back to its src.

## Preconditions

* User logged in.

## Postconditions

* Proper amount of money (see [technical specifications][t2]) is returned to the source.

## Main Success Scenario

1. User starts a refunding for a reservation.
2. [s2]User selects a reason for the refunding.
3. System checks the reason.
4. System performs the refunding.
5. System get informed of the refunding success.
6. System updates the state of the reservation.
7. System informs User of the success.
8. [s8]System closes the transaction.

## Extensions

1. User can enter with different approaches. （用户可以以不同的姿势进入这个用例 XD）
   * *through an independent entry*.
     1. System enlists all Users’ reservations that are able to be refunded.
     2. User chooses a reservation.
   * *through the interface of a reservation*.
     1. System goes directly into [step 2][s2] in the Main Success Scenario.
2. User can select the following reasons:
   * *flight overdue*.
     1. System checks if the flight is overdue.
   * *I don’t want it*.
     1. System checks if requirements are satisfied (see [technical specs][t1]).
3. System checks the reason.
   1. System finds that the reason is not valid. For the following scenarios:
      * *flight overdue*.
        1. System found that the flight is not overdue.
      * *I don’t want it*.
        1. See [technical specs][t1].
   2. System informs User of the failure and give reasons for the failures.
   3. System goes to [step 8][s8].
4. …
5. Refund failed.
   1. System informs User of the failure.
   2. System logs the anomaly.
   3. System terminates the refunding, the reservation stays effective.

## Technical Specifications

* [t1]Reservations that can be refunded must satisfy:
  1. has been paid, but not withdrawn.
  2. the flight is due no sooner than 2 hours later.
* [t2]For different refunding reasons, different portions (might exceed 100%) of the paid money will be returned. For example, if the plane were overdue more than 2 hours, User can get 120% percent of their refund. Note that the refunding strategies are likely to vary.
  1. *flight overdue*: 120%.
  2. *I don’t want it*: 80%.