MyDesk Ticketing System

Problem Statement

Due to the lack of customisation in the 3rd party ticketing system, our organisation is planning to create one. Before we have the complete solution, we would like to do an MVP project and validate.

Following stories are in the scope of the MVP:

- A ticket has type and description. A ticket can either have a manual action, where the employee calls the customer to resolve it, or an auto-action where the system takes a predefined action and auto resolves the ticket. Auto actions are inferred from ticket type. Actions are pre-implemented, i.e. a ticket type will have an associated action-type which is already implemented in the code.
- A user can create a ticket with type and description max 500 chars and a unique ticket id is returned to the customer.
- If type is '**check-wallet-balance**' then send SMS balance to the customer and mark ticket auto resolved
- If type is '**change-language**' then call the customer with IVR and allow the customer to change and mark ticket auto resolved
- If the type is **'others'**, then the ticketing system should queue the ticket to the customer care team.

No ticket is auto-assigned. Employees should always request the ticket-system to assign a ticket to them, same for the Supervisor. Currently, there are no priority queues in the ticketing system, tickets are served as the First-Requested-First-Served basis.

To resolve the assigned ticket the Employee calls the customer and notes down the resolution provided (mandatory) and marks the ticket as resolved and Supervisor verifies the resolved ticket and marks the ticket as resolution-verified.

There is no need to verify auto-resolved tickets as it is taken care of automatically by the system and chances of getting wrong is very less.

Both Employee and Supervisor can get assigned to a ticket by calling ticket-system's assign-ticket action. For the employee, the ticket is assigned from the open-tickets bucket and for the Supervisor ticket is assigned from the verify-resolved-ticket bucket. At any point in time, anyone can check the status of the ticket by the ticket number.
Also, at any point in time, one should be able to query the number of tickets, unresolved tickets, resolved tickets.

We interact with the system via a simple set of commands which produce a specific output.

Please take a look at the example below, which includes all the commands you need to support they're self-explanatory. The system should allow input in two ways.

Just to clarify, the same codebase should support both modes of input - we don't want two distinct submissions.

1. It should provide us with an interactive command prompt based shell where commands can be typed in
2. It should accept a filename as a parameter at the command prompt and read the commands from that file

Input

$> **create-ticket** <msg-type> <msg-desc>

<ticket-number>

$> **assign-ticket** <emp-name> // same command is used for employee and supervisor but tickets are assigned from the respective bucket (open tickets bucket or verification tickets bucket)

Ticket-<ticket-number> -> <emp-name>

$> **resolve-ticket** <emp-name> <resolution-comment>

Ticket-<ticket-number> resolved by <emp-name> with comment <resolution-comment>

$> **verify-ticket-resolution** <supervisor-name>

Ticket-<ticket-number> resolution verified by supervisor <supervisor>

$> **status**

XX - OPEN TICKETS
XX - ASSIGNED TICKETS
XX - CLOSED TICKETS
XX - TOTAL TICKETS

$> **status** <ticket-number>

Ticket-<ticket-number> status: <status> comment: <resolution-comment> resolved-by: <emp-name> verified-by: <supervisor-name>

$> **exit**

Exits the program

Implementation notes

Although this is an MVP, we want you to design a solution such that it is easily extensible to adapt any new change requests. Pre-create employees Sam, Tom and a supervisor, Harry. Mock the implementation wherever is necessary (SMS sending, IVR calling).
Design the system to be interactive or take input from the file if the ticket number generation logic is already defined.

Interactive mode example

```
> create-ticket check-wallet-balance #1st command
> 1

> status 1 #2nd command
> Ticket-1 status: auto-resolved comment: sent automatic SMS to customer
resolved-by: nil verified-by: nil

> create-ticket change-language #3rd command
> 2

> status 2 #4th command
> Ticket-2 status: auto-resolved comment: automatic IVR call made to the
customer resolved-by: nil verified-by: nil

> create-ticket others Need more details on transaction #5th command
> 3

> assign-ticket tom #6th command
> Ticket-3 -> tom

> resolve-ticket tom Resolved transaction issue #abc #7th command
> Ticket-3 resolved by tom with comment Resolved transaction issue #abc

> status 3 #8th command
> Ticket-3 status: resolved comment: Resolved transaction issue #abc
resolved-by: tom verified-by: nil


> status verified-by: nil #9th command
```

```
> 01 - OPEN TICKETS
01 - ASSIGNED TICKETS
02 - CLOSED TICKETS
03 - TOTAL TICKETS

> verify-ticket-resolution Harry #10th command
> Error! Harry has no ticket assigned to him. #an example to handle error,
feel free to add wherever is needed

> assign-ticket Harry #11th command
> Ticket-3 -> Harry

> verify-ticket-resolution Harry #12th command
> Ticket-3 resolution verified by supervisor Harry

> status 3 #13th command
> Ticket-3 status: resolved comment: Resolved transaction issue #abc
resolved-by: tom verified-by: Harry


> status #14th command
> 00 - OPEN TICKETS
00 - ASSIGNED TICKETS
03 - CLOSED TICKETS
03 - TOTAL TICKETS
```

File mode example

input.txt (assuming ticket number starts from 1)

```
create-ticket check-wallet-balance
status 1
create-ticket change-language
status 2
create-ticket others Need more details on transaction #abc
assign-ticket tom
resolve-ticket tom Resolved transaction issue #abc
status 3status
verify-ticket-resolution Harry // report error as shown below in the sample
status
verify-ticket-resolution Harry // report error as shown below in the sample
assign-ticket Harry
```

```
verify-ticket-resolution Harry
status 3
status
```

Expected output same as the consolidated output of interactive example.

Rules

1. We are really, really interested in your object-oriented or functional design skills, so please craft the most beautiful code you can.
2. We're also interested in understanding how you make assumptions when building software. If a particular workflow or boundary condition is not defined in the problem statement, what you do is your choice.
3. You have to solve the problem in an object-oriented or functional language without using any external libraries to the core language except for a testing library for TDD. Your solution must build+run on Linux. If you don't have access to a Linux dev machine, you can easily set one up using Docker.
4. Please use Git for version control. We expect you to send us a standard zip or tarball of your source code when you're done that includes Git metadata (the .git folder) in the tarball so we can look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus.
5. Please do not make either your solution or this problem statement publicly available by, for example, using GitHub or bitbucket or by posting this problem to a blog or forum.