Capstone Courseware, LLC
1 Washburn Place
Brookline, MA 02446
877-227-2477
capstonecourseware.com

HelpDesk - Design

In this exercise you will first design and then build a domain model to support various requirements of a help-desk application. This document describes the expected development process, requirements, and available resources.

Requirements

1. The help desk answers requests for technical assistance by creating "tickets" and managing these tickets to track the flow of work to resolve each stated problem.
2. Tickets capture information including the request originator, description, priority, status, history, and keyword tags. Possible priorities are low, medium, high, and urgent.
3. A tickets has a status that is one of created, assigned, or resolved.
4. Each event in the history of a ticket has a time stamp (to minute precision), possibly a new ticket status, and a text note. When tickets are created there is a stock note "Created ticket."
5. Tickets are assigned to technicians, on a rotating basis but with the aim of keeping an even load; so if a given technician has resolved several tickets and thus has fewer "active" tickets than others, they should be assigned the next ticket when it's created. Assignment of a ticket generates an event in its history, with a message of the form "Assigned to technician [technician ID], [technician name].".
6. Tickets can be resolved, with the caller providing a text note explaining the resolution. This generates a final event in the history, with the provided text being the event's note.
7. The caller can add notes to a ticket without changing the ticket's status: these notes become additional events in the ticket's history.
8. The caller can add any number of unique keyword tags to a ticket. Tags are compared in a case-insensitive manner and generally represented in all lower case.

9. The system must be able to find a single ticket by its ID.
10. The system must be able to find all tickets, all tickets in a given status, or all tickets not in a given status (such as “all unresolved tickets”).
11. The system must be able to find all tickets assigned to a specific technician.
12. The system must be able to report the tags associated with a given ticket, and be able to report all tickets that have a certain tag, or any of a given set of tags.
13. In all cases where multiple tickets are stored or reported, they should be sorted in descending order of priority – urgent first, low priority last – and then by their IDs in ascending order.
14. The system must be able to report the average time, in minutes, that it takes to resolve a ticket. This is calculated as the mean number of minutes between the creation and resolution events in the history of each resolved ticket; unresolved tickets are not considered.
15. The system must be able to report the average time to resolve tickets, grouped by technician.
16. The system must be able to return all tickets that include a given search string in their descriptions or event notes.

The following extended requirements should be considered as part of your initial design, but you might not need to delve into the details and inner workings as closely for now, and you might expand your design after implementing the above requirements.

17. When the originator of a ticket that's been resolved finds that the problem has not actually been resolved, they can re-open the ticket. This actually generates a new ticket that holds a reference to the "prior ticket." Note that the prior ticket may itself be a reopened ticket, so that there can be a chain of tickets if a ticket is resolved, reopened, resolved again, reopened again, etc.
18. A re-opened ticket automatically has the same originator as the prior ticket.
19. A re-opened ticket reports its history as its own history and the history of its prior ticket(s), in chronological order.
20. A re-opened ticket is effectively tagged with its own tags and any tags associated with its prior ticket(s).
21. A re-opened ticket can be found by a search string that is contained in its own description or event notes, or the description or event notes of its prior ticket(s).
22. To improve tagging and querying by tag, the caller can pre-define synonymous tags, essentially saying that tag X should be considered the same as tag Y. For example "remote access" and "RDP" may be synonyms for "remoting". If a caller adds the tag X, it should be translated to Y and stored as Y. If a caller queries for tag X, it should be translated to Y and queried as if the caller asked for Y.
23. The caller can also pre-define a preferred capitalization for a tag. Tags continue to be compared in a case-insensitive manner, but instead of assuming that we prefer all lower case, the preferred capitalization will be stored and reported. For example the caller may pre-define "VM" or "GitHub" to be used instead of "vm" or "github".

Design Process

You will begin by reading over the above requirements, analyzing the problem, and designing a solution in moderate detail: identify classes, fields and their types, constructors and their parameter types, and methods and their parameter and return types (but you don't need to write out every getter or setter), show relationships between classes, and be able to explain how objects will interact in order to support a given requirement (verbally, even if you haven't written out all of these interactions).

You'll review your design with the instructor, take any feedback, and possibly revise your design for further review. Expect to spend as much as 25% of your available time solely on analysis and design (and this should pay off in the form of a smoother and more rapid implementation).

When you have an approved design, you'll be given some starting code, and a second document that will guide your development process through implementation of the requirements, in phases.

Resources

To further inform your design, an Excel spreadsheet is available that holds a test data set: you can find it right next to this document as **TestData.xlsx**. Take note that some columns – in **bold, blue font** – are not provided data but rather values that your system is expected to generate or compute: things like ticket ID or assigned technician.

There are three sheets:

1. The **Technicians** sheet provides details of a help desk's staff of technicians.

2. The **Tickets** sheet is the most involved and shows a series of simulated events in the help desk's work flow. Each row depicts an event at a specific date and time (all between November 1-3, 2021, so only day, hour, and minute are shown), related to a specific ticket by ID (note that this is a value you will generate), with a specific event type and further data relevant to that type of event. The **Ticket counts** columns show a running tally of active tickets for each of the four technicians, which helps to explain why each new ticket is assigned as it is: for example why ticket 5 is assigned to Dineh even though it might be considered to be Andree's turn, because Dineh has by that time already resolved ticket 4.

3. The **Time to** resolve sheet shows an analysis of time to resolve tickets. This is all data that you will derive and compute in your own implementation; it's presented here for reference and comparison.

Examples and Clarifications

Regarding requirements #3, #4, #6, and #7: the history for ticket 3 would have four events, all on November 1, 2021. (Notice that adding tags to the ticket does not add events to its history; tags are managed separately.)

> At 8:37 the ticket is put in created status, with the note "Created ticket."
>
> At 8:37 the ticket is put in assigned status, with the note "Assigned to technician A05589, Andree."
>
> At 8:39 the ticket doesn't change status but a note is added: "Requested approval from manager."
>
> At 10:20 the ticket is put in resolved status, with the note "Received approval; added permission."

Regarding requirement #5: the first four tickets are assigned to Andree, Boris, Caelem, and Dineh, in sequence. Ticket 5 is assigned to Dineh – not to Andree, as it would be if we were always rotating around the group. This is because, by the time that ticket is created, Dineh has resolved ticket 4 and so has fewer active tickets than the other technicians.

Regarding requirements #8 and #12: both tickets 3 and 6 would turn up in a search for tickets tagged with "permissions" – or in a search for tickets tagged with "Permissions".

Regarding requirement #10: having simulated all of the events (except for the last few that are specifically about re-opening tickets), there would be 7 resolved tickets, and 7 not-resolved tickets.

Regarding requirement #11: at that same point, there would be 5 tickets assigned to Andree, and 3 to each other technician. Note that the system reports all assigned tickets – not just active ones.

Regarding requirement #13: the 14 tickets in the system would be reported in this order: 7, 11, 1, 2, 3, 8, 9, 10, 14, 3, 4, 6, 12, 13.

Regarding requirements #14 and #15, average times in minutes are as shown on the **Time to resolve** spreadsheet, including 1547 minutes as the average for the help desk as a whole, and specific numbers as shown for individual technicians.

Regarding requirement #16, the system should find ticket 5 on a search for "corrupt" because this text is contained in the ticket's description; it should find ticket 1 on a search for "browser" because this text is contained in the notes on that ticket.