Customer Issue Resolution System

PhonePe processes a vast number of transactions every day, wherein some transactions may fail (enter a FAILED state) or remain in a PENDING state due to various reasons such as bank or NPCI issues, or internal PhonePe errors. To handle such cases efficiently, a resolution system is needed, where customers can log their unsuccessful transactions and raise complaints against them.
The system must categorize customer issues into several types, such as payment-related, mutual fund-related, gold-related, or insurance-related. Different customer service agents will have their specific expertise based on the issue type, whom the system will assign the issues by marking them waiting in case all agents are busy.

The customer service agents can work on one issue at a time and update its status, and once it is resolved, the agent will receive another issue.

The solution should incorporate the following features:

- Customers can log a complaint against any unsuccessful transaction.
- Customer Service Agents can search for customer issues with issue ID or customer details (email).
- Agents can view their assigned issues and mark them resolved once they are resolved.
- System should assign the issue to agents based on an assigning strategy.
- System should allow the admin to onboard new agents.
- System should allow the admin to view the agent's work history.

Implementation requirements
Your solution should implement the following functions. Feel free to use the representation for objects you deem fit for the problem and the provided use cases. The functions are ordered in the decreasing order of importance (highest to lowest). We understand that you may not be able to complete the implementation for all the functions listed here. So try to implement them in the order in which they are declared down below.

```JavaScript
createIssue(transactionId, issueType, subject, description,
email)
addAgent(agentEmail, agentName ,List<issueType>)
assignIssue(issueId) // -> Issue can be assigned to the agents
based on different strategies. For now, assign to any one of the
free agents.
```

```
getIssues(filter) // -> issues against the provided filter
updateIssue(issueId, status, resolution)
resolveIssue(issueId, resolution)
viewAgentsWorkHistory() // -> a list of issue which agents worked
on
```

Example:

JavaScript

```
createIssue("T1", "Payment Related", "Payment Failed", "My
payment failed but money is debited", “testUser1@test.com”);
>>> Issue I1 created against transaction "T1"

createIssue("T2", "Mutual Fund Related", "Purchase Failed",
"Unable to purchase Mutual Fund", “testUser2@test.com”);
>>> Issue I2 created against transaction "T2"

createIssue("T3", "Payment Related", "Payment Failed", "My
payment failed but money is debited", , “testUser2@test.com”);
>>> Issue I3 created against transaction "T3"

addAgent(“agent1@test.com”, “Agent 1”, Arrays.asList("Payment
Related", "Gold Related"));
>>> Agent A1 created


addAgent(“agent2@test.com”, “Agent 2”, Arrays.asList("Mutual Fund
Related"));
>>> Agent A2 created


assignIssue("I1")
>>> Issue I1 assigned to agent A1

assignIssue("I2")
>>> Issue I2 assigned to agent A2
```

```
assignIssue("I3")
>>> Issue I3 added to waitlist of Agent A1



getIssue({"email": "testUser2@test.com"});
>>> I2 {"T2", "Mutual Fund Related", "Purchase Failed", "Unable
to purchase Mutual Fund", "testUser2@test.com", "Open"},
 I3 {"T3", "Payment Related", "Payment Failed", "My payment
failed but money is debited", , "testUser2@test.com", "Open"}

getIssue({"type": "Payment Related"});
>>> I1{"T1", "Payment Related", "Payment Failed", "My payment
failed but money is debited", "testUser1@test.com", "Open"},
 I3 {"T3", "Payment Related", "Payment Failed", "My payment
failed but money is debited", "testUser1@test.com", "Open"}

updateIssue("I3", "In Progress", "Waiting for payment
confirmation");
>>> I3 status updated to In Progress

resolveIssue("I3", "PaymentFailed debited amount will get
reversed");
>>> I3 issue marked resolved


viewAgentsWorkHistory()
>>> A1 -> {I1, I3},
    A2 -> {I2}
```

Things to keep in mind

- **Programming Language & Environment:** You can use the programming language of their choice to implement the solution. Free to use any local IDEs, such as IntelliJ, or the CodeSignal platform for writing code.
- **Scope of Implementation:** Use of REST APIs is not required; implementing equivalent functions for the functional requirements is sufficient.

- **Data Persistence:** Persistence should be achieved using in-memory data structures (e.g., Lists, Queues, Maps), and no connection to databases is needed.
- **Third-Party Libraries:** It is recommended to refrain from using third-party libraries like Guava or Hystrix for functional requirements.
- **Demonstrating Functionality:** There is no need to handle command-line inputs, inputs can be hardcoded within a driver class. A main() class or driver or test cases that simulates the functional requirements is sufficient.
- **Ambiguity in Problem Statement:** If there is any ambiguity in the problem statement, you can make reasonable assumptions and proceed with the solution. These assumptions should be clearly called out in comments within the code for them to be discussed in evaluation round.
- **Submission Guidelines:** You must zip your solution files and email them to the provided email address (Email address must be available in Interview Invite Link). This must be done even if you choose to code on Code signal Platform. Please ensure you submit your solution on time even if it is incomplete. A partial solution submitted on time is better than a complete solution submitted after time.

Evaluation Criteria

- Completeness of functional requirements
- Application of OO design principles
- Code efficiency
- Code readability and maintainability
- Testability
- Handling corner cases
- Language proficiency

Time Duration
You will have 1.5 hours to submit your solution to this problem statement.

- **[execution time limit] 4 seconds (js)**
- **[memory limit] 2g**