**Issue Tracker**

**Mont Trance Inc.** is one of the leading imports and exports industry. They are operating across the globe. In order to facilitate the smooth running of their operations, they are in need of developing an application, through which any user can report any issue that they may encounter.

**Modules Required**

* Assignee
  + Fetch Assignee
  + Update Active Issue Count
* Issue
  + Report An Issue
  + Update Status
  + Show Issues
  + Delete Issues

**Artifacts Description**



**Tester**

This class acts like an interface between the users and the system. All necessary inputs can be supplied in this class, and the respective outputs can be displayed accordingly. We have provided objects with hard-coded values, according to the requirement. You may require changing the values, to check and verify different situations. The necessary code to call methods of Service class and displaying relevant output is already provided.

Note: Uncomment the commented code (if any) in the **IssueTester**, **AssigneeDAOImpl** and **IssueDAOImpl**classes, after the implementation of required model classes.

**IssueTester**

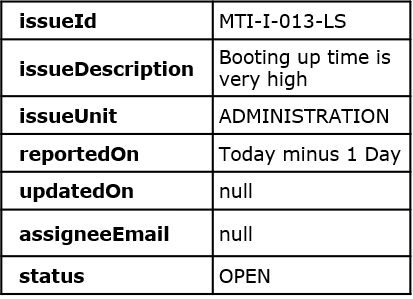
**Module 1: Report An Issue**

**Input:** Issue – issueId, description, issueUnit, reportedOn, status

**Description:** This method is used to validate the issue details and report it on the application

**Valid Input:**

For valid issue details,



the output should be,

03.PNG

**Invalid Inputs:**

For invalid issueId– "MTI-Issue-000-LS", the output should be,

04.PNG

For duplicate issueId– "MTI-I-005-MS", the output should be,

05.PNG

For invalid description – "5 payments have failed!", the output should be,

06.PNG

For invalid reportedOn – (any future date), the output should be,

07.PNG

For invalid status – "CLOSED", the output should be,

08.PNG

**Module 2: Update Status**

**Input:** Issue – issueId, status

**Description:** This method is used to update the status of an existing issue

**Valid Input:**

For valid issueId - "MTI-I-001-MS" and valid status – "RESOLVED", the output should be,

09.PNG

**Invalid Inputs:**

For invalid issueId– “MTI-I-111-MS”, the output should be,

10.PNG

For valid issueId - “MTI-I-001-MS” and invalid status – “IN PROGRESS”, the output should be,

11.PNG

For valid issueId - “MTI-I-001-MS” and incompatible status – “RECALLED”, the output should be,

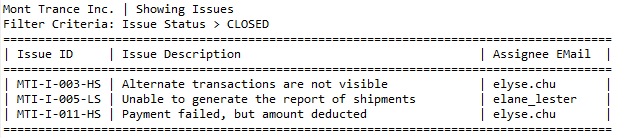
12.PNG

**Module 3: Show Issues**

**Input:** filterCriteria

**Description:** This method is used to generate a report of all the issues reported, filtered based on the given criteria

**Valid Input:** For a valid filterCriteria – key: “S” and value: “CLOSED”, the output should be,



**Invalid Input:**

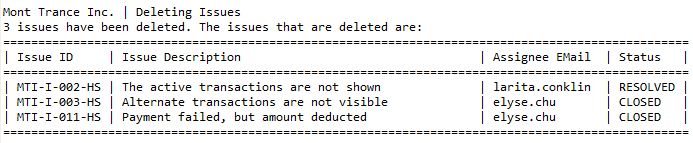
For an invalid filterCriteria – key: “A” and value: “jane\_andrews”, the output should be,

14.PNG

**Module 4: Delete Issues**

**Description:** This method is used to delete all the issues, which are resolved or closed before 2 weeks (14 days)

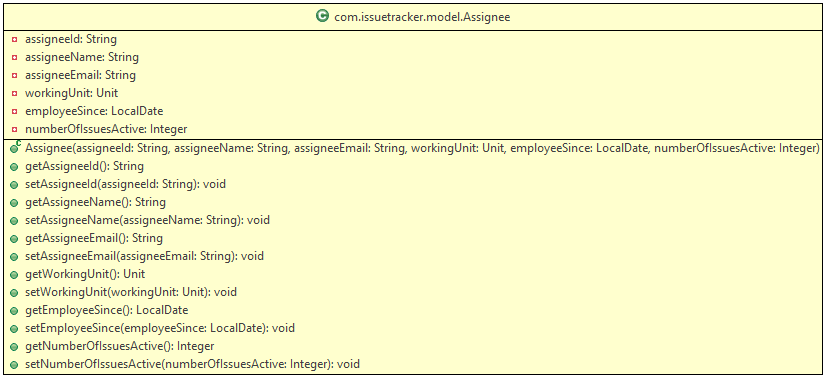
**Valid Execution:** The output should be,



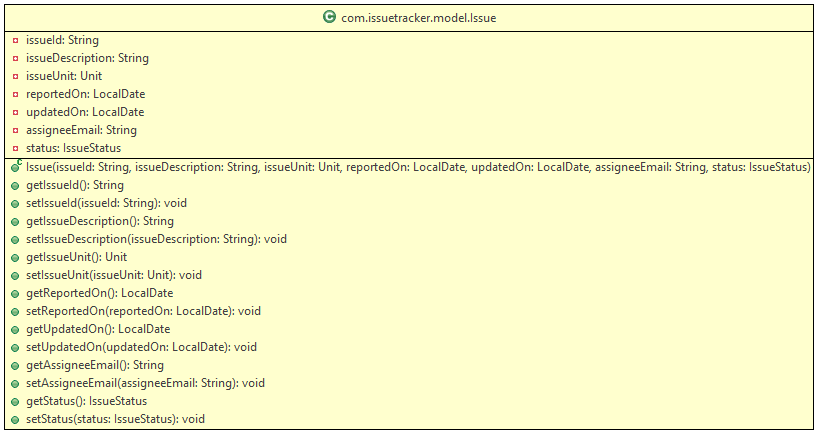
**Model Classes**

These classes help in storing related values of a single data entity. Objects of these classes transfers the values between the different layers of the Application. You are required to use them, whenever necessary.

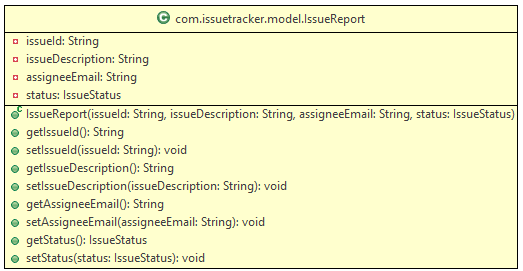
**Assignee**- (To be implemented)



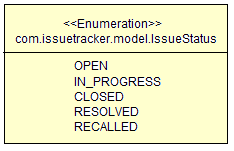
**Issue** - (Implemented)



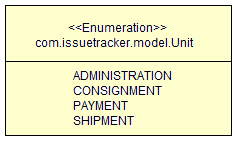
**IssueReport** - (To be implemented)



**IssueStatus** - (To be implemented)



**Unit** - (Implemented)



**Exceptions**

**IssueTrackerException** - (Implemented)

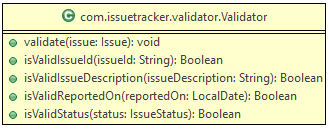
This is an User Defined Exception used to throw exception in this application

21.PNG

**IssueTrackerException(String message)**

This is constructor for **IssueTrackerException** class with message as a parameter.

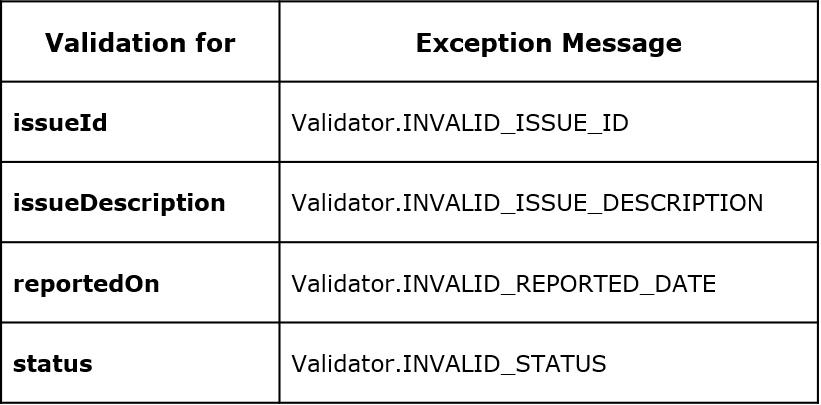
**Validator**



This class will be used to validate the **Issue**object received.

**validate(Issue issue)**

This method will validate an Issue object. It should call the respective validator methods (described below) to validate the values, and throw **IssueTrackerException** (if any) with the corresponding message codes, as shown in the table below. The **IssueTrackerException** has to be logged before being thrown.



**isValidIssueId(String issueId)**

This method validates the **issueId**

The **issueId**should be of the format:

<1st Part>-<2nd Part>-<3rd Part>

(In words, *<FirstPart><Hyphen><SecondPath><Hyphen><ThirdPath>*)

* The **issueId**cannot be null or empty
* The first part should be "MTI-I", representing Mont Trance Inc. - Issue
* The second part can be any 3-digit number, except "000"
* The third path be "LS", "MS" or "HS", representing Low, Medium or High Severity
* If all the conditions are satisfied, the method returns true; else, it should return false
* Valid Values:
  + MTI-I-001-MS
  + MTI-I-012-HS
  + MTI-I-004-LS
* Invalid Values:
  + MT-I-001-MS
  + MTI-I-012-HSS
  + MTI-I-0012-HS

**isValidIssueDescription(String issueDescription)**

This method validates the issueDescription

* The description cannot be null, empty or just spaces
* The description should contain only alphabets
* It can be of multiple words, each word separated by a single space
* The description should not contain leading or trailing spaces
* The description should be between 1 to 50 characters (both inclusive), in length
* If all the conditions are satisfied, the method returns **true**; else, it should return **false**
* Valid Values:
  + "The active transactions are not shown"
  + "Workstation is faulty"
* Invalid Values:
  + "1 active transactions are not shown"
  + "   Workstation is faulty"
  + "The active transactions are  "
  + "122367990"

**isValidReportedOn(LocalDate reportedOn)**

This method validates the reportedOn date

* The issue can be reported on any valid date, except from the future
* If all the conditions are satisfied, the method returns **true**; else, it should return **false**
* Valid Values:
  + Any past date
  + Current date
* Invalid Values:
  + null
  + Any future date

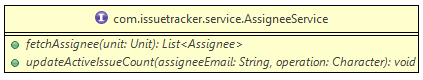
**isValidStatus(IssueStatus status)**

This method validates the status

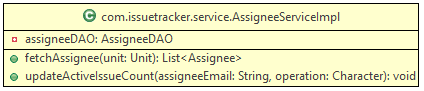
* The issue status has to be either OPEN or IN PROGRESS
* If all the conditions are satisfied, the method returns true; else, it should return false
* Valid Values:
  + OPEN
  + IN\_PROGRESS
* Invalid Values:
  + CLOSED
  + RESOLVED
  + RECALLED
  + null

**Service Layer**

**AssigneeService** - (Implemented)



**AssigneeServiceImpl**-(To be implemented)



Populate the attribute **assigneeDAO**with an object of **AssigneeDAOImpl**

**fetchAssignee(Unit unit)**

This method will receive an assignee unit. It should fetch and return all the assignees from that unit, who are available to handle a new issue

* Call the fetchAssignees method of AssigneeDAO by passing the received unit.
* From the list of assignees that is returned by the above method call, filter the assignees who is handling less than 3 (active) issues
* Return the filtered assignee list

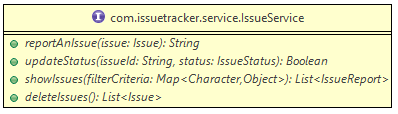
**updateActiveIssueCount(String assigneeEmail, Character operation)**

This method will receive an assignee email and an operation value. It should increment or decrement the number of issues being handled by the assignee.

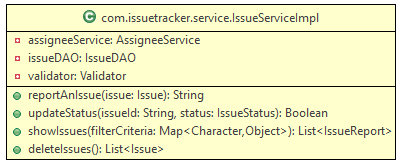
* Fetch the Assignee object by invoking the getAssigneeByEmail method of AssigneeDAO class
* If the operation value received is "I", increment the number of issues that are being handled by the assignee
* If the operation value received is "D", decrement the number of issues that are being handled by the assignee

Note: The operation value will always be either "I" or "D"

**IssueService**-(Implemented)



**IssueServiceImpl**- (To be implemented)



Populate the attributes **assigneeService** ,**issueDAO** and**validator**with an object of **AssigneeServiceImpl**,**IssueDAOImpl**and **Validator** respectively

**reportAnIssue(Issue issue)**

This method will receive an Issue object. It should check the validity of the given details, fetch an assignee to handle the issue and send the object for storage

* Validate the details in the received object using the validate method of the Validator class
* On successful validation, invoke the fetchAssignee method of the AssigneeService class by passing issue unit
* The fetchAssignee method will return a list of assignee who are available to handle the request. If the received list in not empty,
  + Pick the first assignee from the received list and assign him/her to the issue object
  + Call the updateActiveIssueCount method of AssigneeService class by passing the email address of the Assignee and the operation as "I", to increment the number of issues that are being handled by the chosen Assignee
* Invoke the reportAnIssue method of the IssueDAO, which in turn returns the issueId
* If the issueId received from IssueDAO is null, throw an **IssueTrackerException** with the message "**IssueService.DUPLICATE\_ISSUE\_ID"**. Else, return the issueId received
* Log only the IssueService exceptions. Re-throw all the exceptions

**updateStatus(String issueId, IssueStatus status)**

This method will receive an issue id and a status. It should update the status of the given issue.

* Invoke the getIssueById method of IssueDAO, which will return an issue object
* If the received issue object is null, throw an **IssueTrackerException** with the message **"IssueService.ISSUE\_NOT\_FOUND"**
* If the status to be updated is same as the existing status of the received issue object, throw an **IssueTrackerException** with the message **"IssueService.NO\_STATUS\_CHANGE"**
* If the status to be updated is RECALLED and if the existing status is not OPEN, throw an **IssueTrackerException** with the message **"IssueService.INCOMPATIBLE\_STATUS"**
* Else,
  + Invoke the updateStatus method of IssueDAO
  + If the updated status is not OPEN or IN PROGRESS, invoke updateActiveIssueCount method of AssigneeService by passing the assignee email and the operation as "D"
* Return true, if the issue is updated successfully
* Log only the IssueService exceptions.  Re-throw all the exceptions

**showIssues(Map<Character, Object> filterCriteria)**

This method will receive a filterCriteria, based on which a report of all the issues has to be generated

* Invoke the getIssueList method of IssueDAO to retrieve a list of Issue objects
* Filter the list based on the below guidelines
  + If the key in the received filterCriteria is “A”, filter the list so that it contains only the issues that are assigned to the received assigneeEmail (value for the key A, in filterCriteria)
  + If the key in the received filterCriteria is “S”, filter the list so that it contains only the issues whose status is equal to the received status (value for the key S, in filterCriteria)
* If the filtered list is empty, throw an **IssueTrackerException**with the message "**IssueService.NO\_ISSUES\_FOUND**". Else, return the filtered list
* Log only the IssueService exceptions.  Re-throw all the exceptions

Note: The filterCriteria will always contain only one key-value pair and the key will always be either "A" or "S"

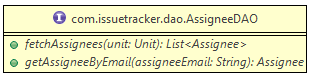
**deleteIssues()**

This method will delete all the issues that are either resolved or closed at least 2 weeks earlier.

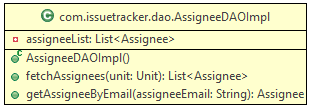
* Invoke the getIssueList method of IssueDAO to retrieve a list of Issue objects
* Populate the issues from the above list in a new list, if the issue status is RESOLVED or CLOSED and the issue is updated at least 14 days earlier.
* If the new list is empty, throw an **IssueTrackerException** with the message "**IssueService.NO\_ISSUES\_DELETED**". Else, remove the objects in the new list from the issueList of IssueDAO class and return the new list
* Log only the IssueService exceptions.  Re-throw all the exceptions

**DAO Layer**

**AssigneeDAO** -(Implemented)



**AssigneeDAOImpl** - (Implemented)



**AssigneeDAOImpl()**

This is constructor for AssigneeDAOImpl

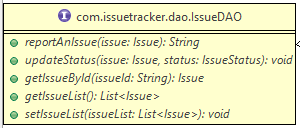
**fetchAssignees(Unit unit)**

This method returns a List<Assignee> populated with all the assignees from the given unit

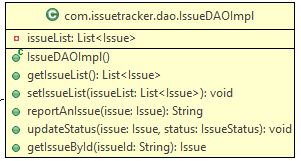
**getAssigneeByEmail(String assigneeEmail)**

This method returns a Assignee object populated with the details of the assignee, requested in the given email. If an assignee is not found for the given email, this method will return null.

**IssueDAO**- (Implemented)



**IssueDAOImpl**- (Implemented)



**IssueDAOImpl()**

This is constructor for IssueDAOImpl.

**getIssueList() & setIssueList(List<Issue> issueList)**

These are the Accessor and Mutator (Getter and Setter) methods for the issueList attribute of the IssueDAOImpl class

**reportAnIssue(Issue issue)**

This method adds the given issue to the issueList. If an issue is already existing with the given id, this method will return null; else, it will return the issueId.

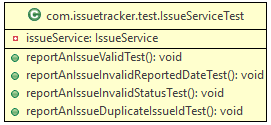
**updateStatus(Issue issue, IssueStatus status)**

This method updates the status for the given issue. If the updated staus is not OPEN or IN PROGRESS, it will also assign value to the updatedOn attribute of the issue object

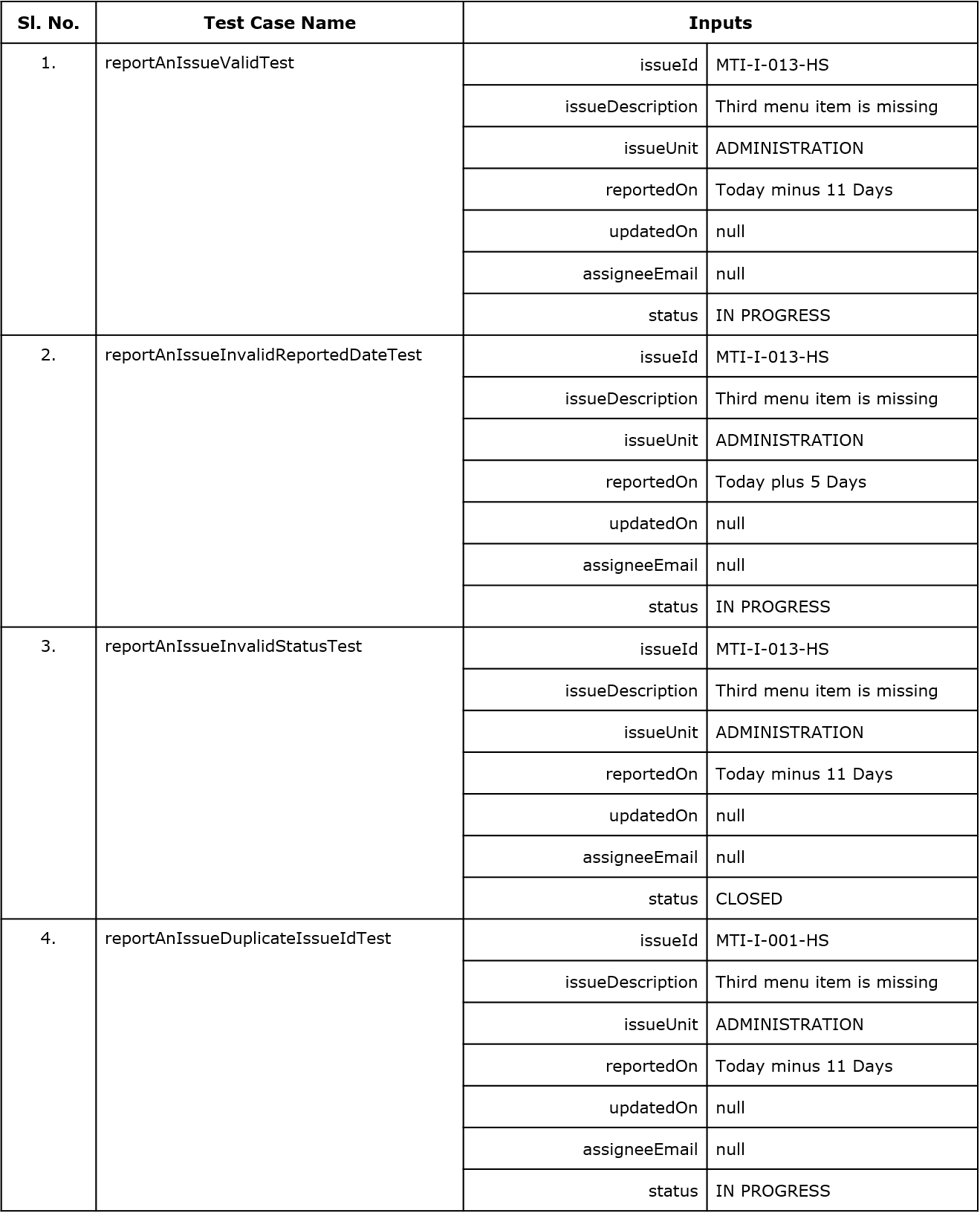
**getIssueById(String issueId)**

This method returns an Issue object populated with the details of the issue, requested in the given issueId. This method will return null, if no issue is found for the given id.

**JUnit Test Cases**



Implement the following test cases for testing the **reportAnIssue**method of **IssueService**



**Note**: Check the project using SonarLint to maintain the coding standards.