# CSE 222/505

## Term Project Final Report

Project Management And Planning System

Group-7


https://github.com/CSE222-Group7/ProjectManagementSystem/

# 1) Group Members

| Name | Student ID |
|---|---|
| HALİL İBRAHİM İLHAN | 200104004048 |
| BURCU SULTAN ORHAN | 1901042667 |
| ERAY ALÇİN | 1801042687 |
| ABDURRAHMAN BULUT | 1901042258 |
| YUSUF ALPEREN DÖNMEZ | 1801042085 |
| ERAY YAŞAR | 1901042635 |
| UĞUR TÜRKEL | 215008014013 |
| RIAN RYZHOV | 1901042801 |
| ARAS TOPUZ | 151044032 |
| SİNAN SARI | 1801042091 |

# 2) Problem Definition

People often have organizing problems while working in groups. They need a system to be more efficient in organizing their issues to achieve what they are working on. To achieve better consistency and quicken decision making the system needs a leader or in other words a project manager. Project managers should have an ability to distribute the issues to the group members to prevent misunderstanding and overdoing one issue by multiple members. Since some issues are more urgent than others, the project manager needs to be able to assign priority levels to issues. And since in a company or group there may be multiple projects there should be an overseer for all projects such as an admin. Admin should be able to create projects and issues in these projects.

Although the project issues can be distributed and drawn on desk or paper, for the project to be accessible from anywhere it must be implemented on the online platform. Tasks should have statuses that may be changed by people that were assigned to it or project manager for tracking project development. People also may want to know each other's progress on their assigned issues, so we must provide the solution for them to be able to do that. These solutions may be a calendar which shows a timeline of the entire project, a Kanban which can ease interaction with tasks and a Reports to record progress for each individual or team working on their task. And people working on the same issue must divide it into child-issues and need to communicate with each other to know each other's progress on assigned child-issues. Since some issues may be related to previous issues, some specific issues can't start before other related issues are completed. That's why the system should provide a comment section for each issue.

People may participate in multiple projects and that's why they need to have a Projects page to see all their projects they are working on.

In general the project management system is a must have for any company or group of people aiming to develop or create their own project. And the bigger the project and group is the bigger the need for such a system they have. Especially nowadays when everything gets faster with new technologies and you should keep up to that speed.

We tried to use the data structure and sort algorithms that were requested to be used in the project in certain places.

1. **List** - We used that structure to store assignees, comments and log history for each issue. Hence all these data won't be ordered and it won't be too large list is the most suitable data structure for this data.

2. **BST** - We used that structure to store Project members. Searching a member in the Binary search tree is easy and as compared to array and linked lists, insertion and deletion operations are faster in BST.

3. **Priority Queue** - We used that structure to sort issues by their priorities so people working with the system could easily understand which issues are most important at that time.

4. **Stack** - Backlog is the issue stack where all the issues are added at the beginning and distributed to the boards from there. Because of its similar structure, we implemented it using stack.

5. **HashMap** - With a key in a hashmap, it is quick and easy to find the information(values) corresponding to the key. For this reason, we used the hash map to keep user records-informations, find users and search among the users, as we can quickly access other information of users with their IDs as a key.

6. **SkipList** - We get faster results in searches in the average case with skip list. It allows us to display the data that will appear as a result of the search efficiently and quickly. For this reason, we used skip list while keeping the list of users which is assigned to the specific issues and used on searching operations for users in the issue part.

7. **Balanced Binary Search Tree** - We used balanced binary search tree instead of binary search tree on the board page/part for making searches more efficient and displaying the results more quicker in the worst case scenarios.

8. **Graph** - Since we do not want to move on to the next issues before other issues related to the specific issue(s) are completed, we used a graph structure in the issue section to show the links between the issues.

9. **Quicksort** - We used quicksort to sort the issues quickly and efficiently according to some properties such as createDate and dueDate.

We made a performance analysis of the methods in these data structures and the sorting algorithm, and we added both empirical and theoretical results in the performance analysis section.

# 3) Users Of The System

## Admin

- System’s unique user.
- Can create/delete users (project manager, project member, board member, guest), projects, issues.
- Can change projects state from verified to done.
- Can assign projects to a project manager.
- Can create and edit profiles.

## Project Manager

- Main users of specific projects.
- Can create/delete issues.
- Can change the project's state from in review to verified.
- Can assign issues to users.
- Can set priority and deadline of issues.

## Board Member

- Users who are assigned to a specific board.
- Can change the project's state from “selected for development” to “in progress”.
- Can change the project's state from in progress to in review.
- Can edit the issue title.
- Can add comments to issues.
- Can only edit/view their board on the project.

## Project Member

- Users who are assigned to a specific project.
- Can change the project's state from “selected for development” to “in progress”.
- Can change the project's state from in progress to in review.
- Can edit the issue title.
- Can add comments to issues.
- Can edit/view their board on the project.
- Can view other boards on the project.

## Guest

- Non-users of the system.
- Can view a specific board if permitted by the admin of the system.

# 4) Requirements

## 4.1) Functional Requirements

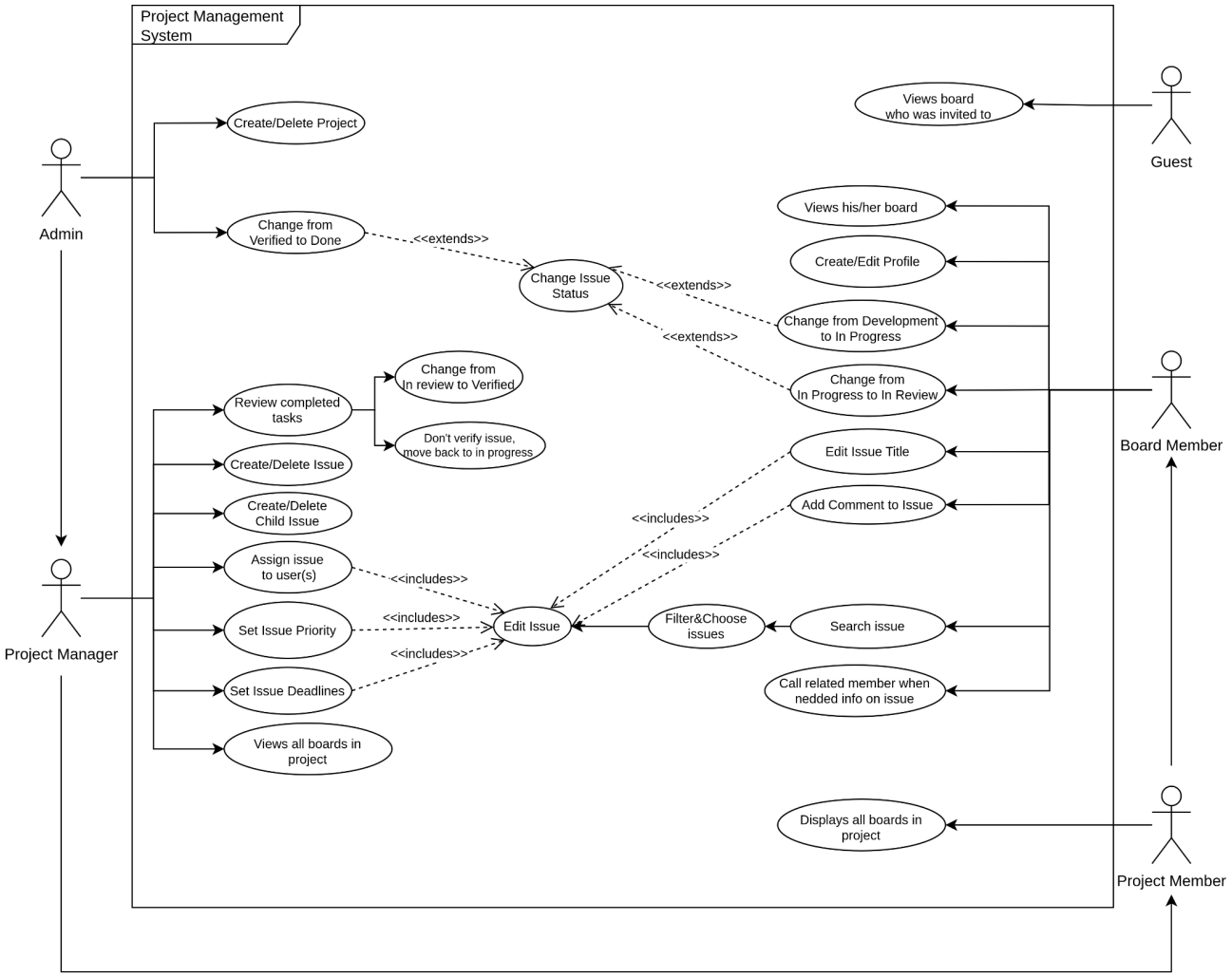
| Requirement ID | Description Of The Requirements |
|---|---|
| FR1 | Admin must be able to create projects. |
| FR2 | Admin must be able to create issues. |
| FR3 | Project manager must be able to assign Issues to users. |
| FR4 | Project must have a Name, Key (like abbreviation), Type and Lead (Company Name) . |
| FR5 | Issues must have an id (Key – Issue Number). |
| FR6 | Issues must have an assignee and reporter user. |
| FR7 | Issues must have a title, comment, history and description sections. |
| FR8 | Issues must have created and updated time. |
| FR9 | The comment section must have edit and delete options. |
| FR10 | Issues must have child-issues. |
| FR11 | Issues must have priority (highest, high, medium, low, lowest). |
| FR12 | Issues must have types like bug, task, story, epic (Epic issues, which represent high-level initiatives or bigger pieces of work in Jira. For software teams, an epic may represent a new feature they're developing. For IT service teams, epics may represent a major service change or upgrade.). |
| FR13 | System must have Projects section (projects list). |
| FR16 | Board section must have a Kanban Board, Backlog and Reports |
| FR17 | Kanban board must have the filters like show only my issues and recently updated. |
| FR20 | Kanban board must have selected for development, in progress, in review, need info, and verified and done section. |
| FR21 | Users must have a profile that must have about, contact, and teams sections. |
| FR22 | Users must be able to edit title, comment and status sections. |

| | |
|---|---|
| FR23 | Users must be able to change the task status from selected from development to in progress and assign them to project manager |
| FR24 | Users must be able to change the issue status from in progress to in review and need info, if user changes status to in review, it must be assigned to project manager, if it is changed to need info, issue must be assigned to the referred user who has the info. |
| FR25 | Project manager must be able to change the issue status from in review to “verified”. If it’s not verified, then the manager can move back the issue to “in progress”. |
| FR26 | Admin should give permission to guests. |
| FR27 | Project may have multiple kanban boards. |
| FR28 | Project member can see all of the kanban boards. |
| FR29 | Project member is also a board member of his/her board. But the project member cannot edit other than his/her board. |
| FR30 | Board member can see and edit only his/her board. (freelancer etc.) |
| FR31 | Guests can only view the board he was assigned to and can be assigned to issues.(Intern etc.) |

## 4.2) Non-Functional Requirements

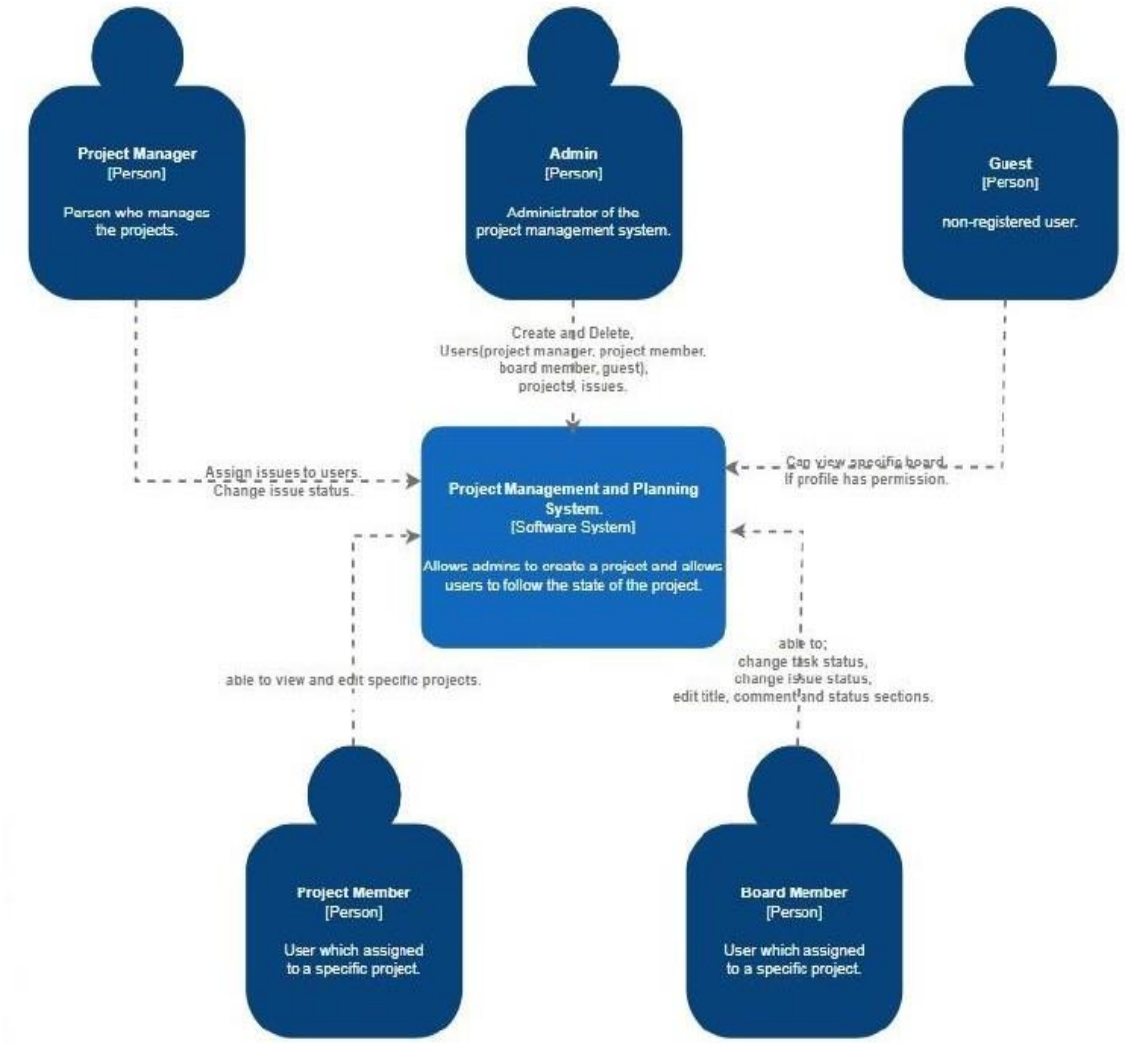
| Requirement ID | Description Of The Requirements |
|---|---|
| NFR1 | JavaSE13 or higher should be installed and running properly. |
| NFR2 | System can operate on any hardware. The user is not in charge of any leaks in security if user decides to change their hardware or their operating system. |
| NFR3 | System is user-friendly so a new user can learn easily. |
| NFR4 | Initial installation of the system should be done by professionals. |

# 5) Use-Case Diagrams

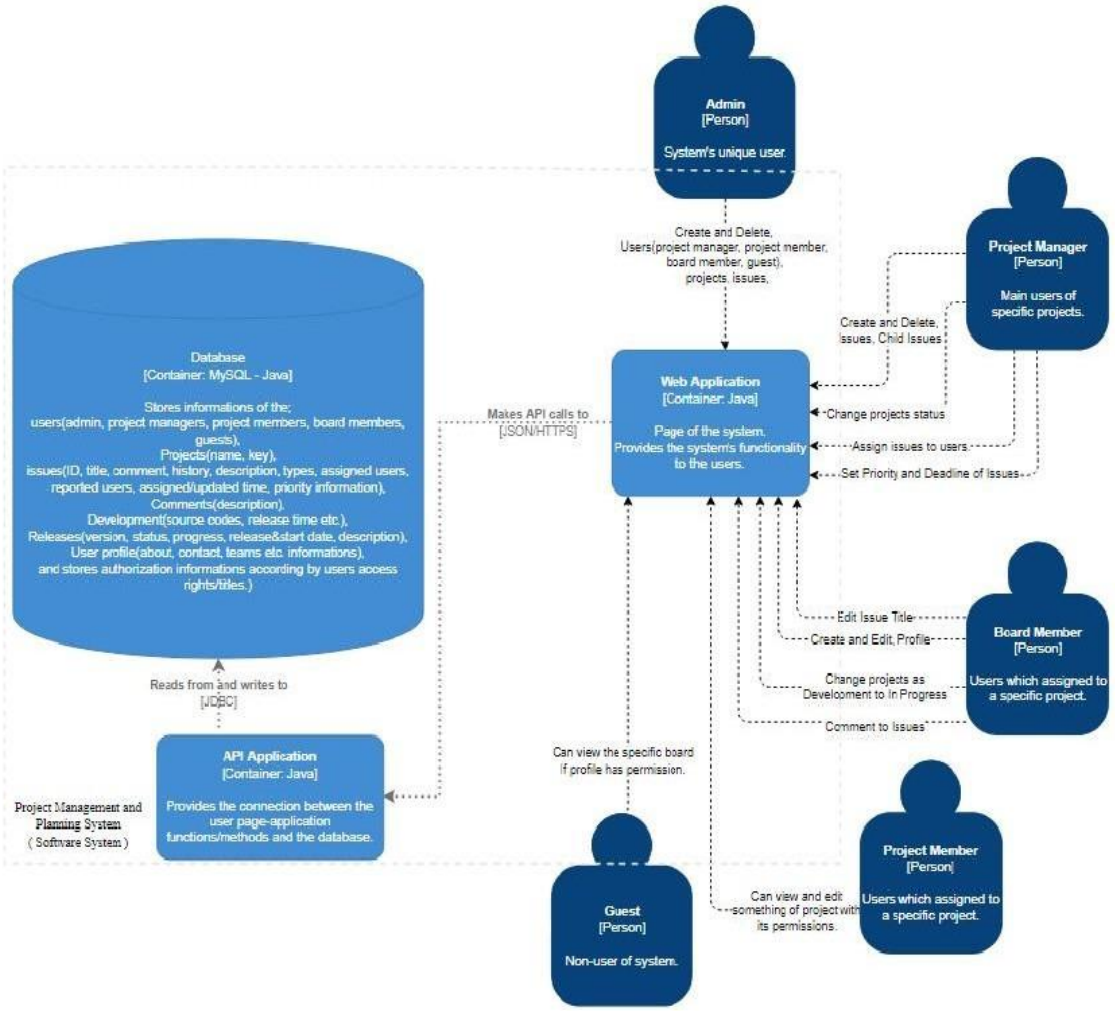


# 6) C4 Models

## I. System Context Diagram:

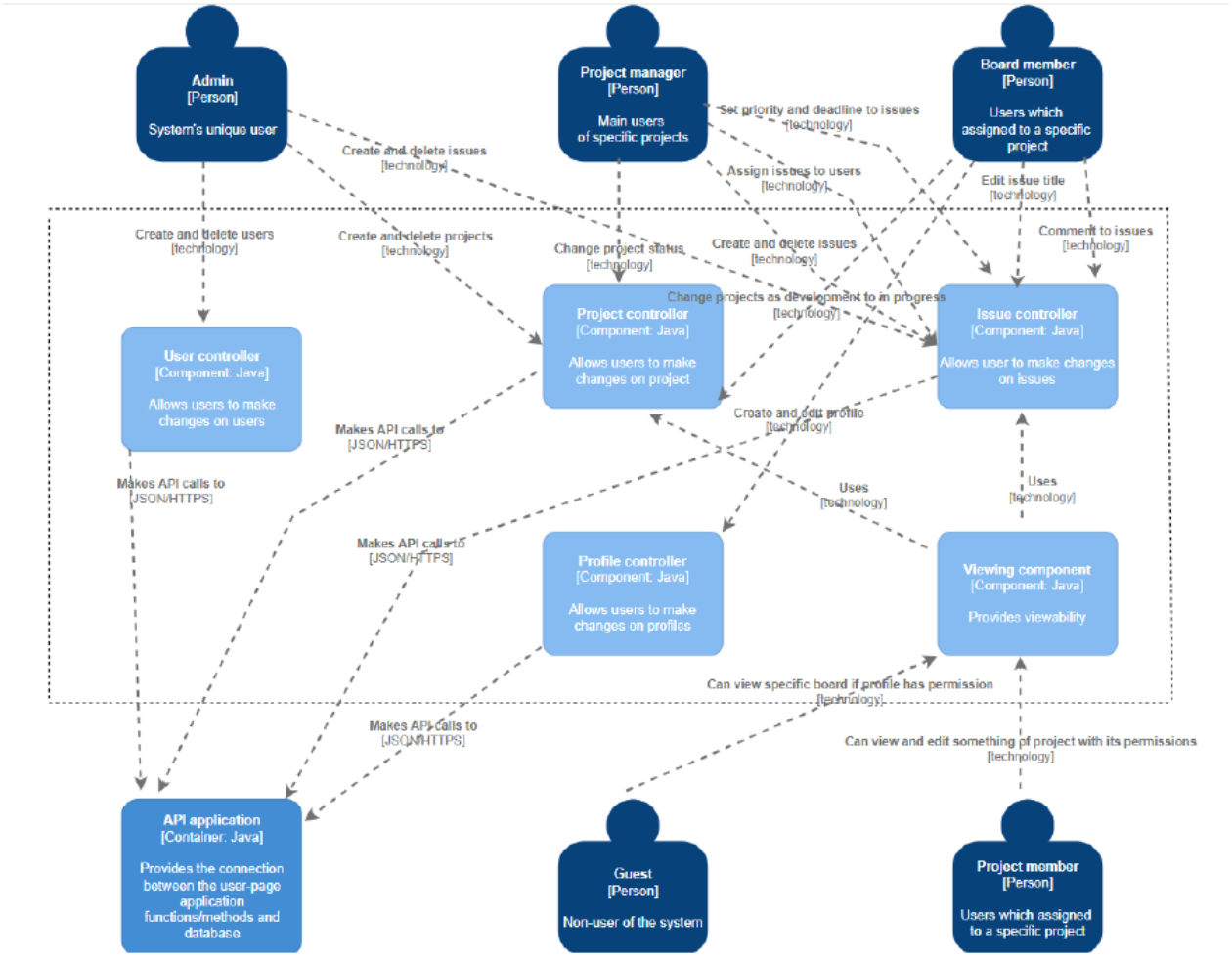


## II. Container Diagram:

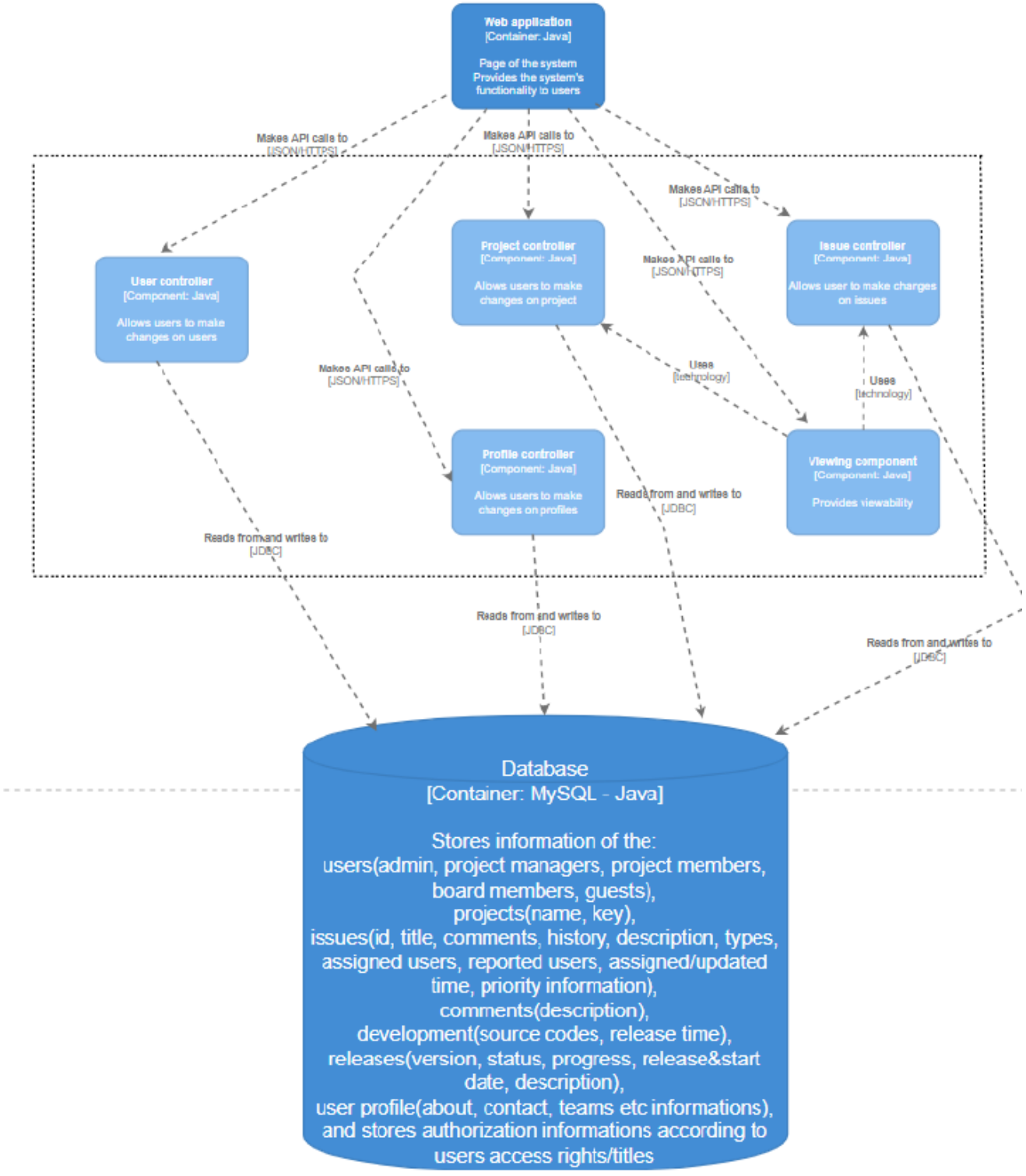


# III. Component Diagram

Component diagram for Project Management System – Web Application:

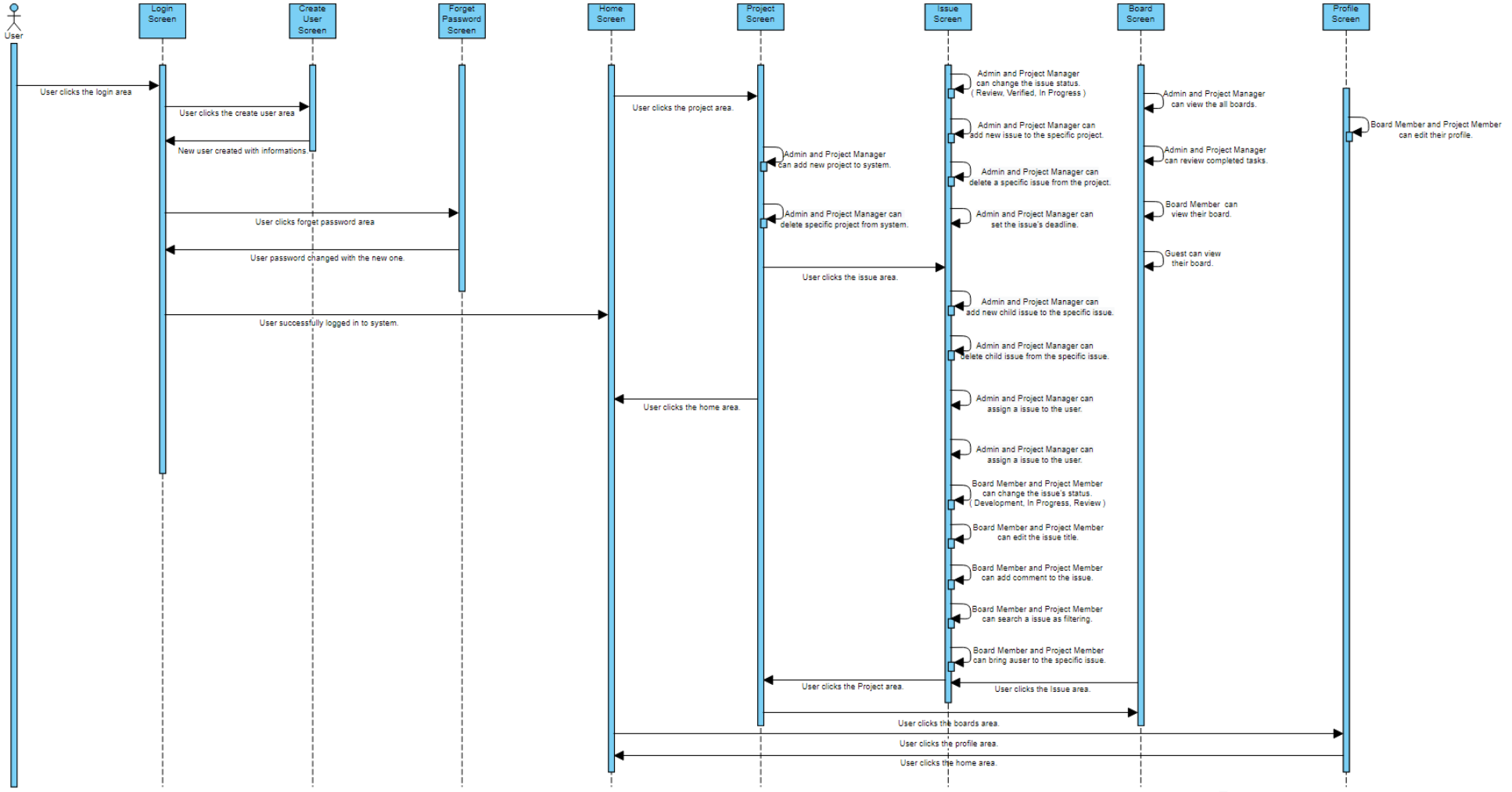


Component diagram for Project Management System – API Application:



# 7) Class Diagram

# 8) Sequence Diagram



# 9) The non-trivial implementation details.

**Passwords**

Passwords are salted and hashed with Bcrypt. Unique salts are generated for each user using Bcrypt's salt generation algorithm with 2^12 rounds of hashing. Java has built-in support for PBKDF2WithHmacSHA1/SHA256/SHA512 (Password-Based Key Derivation Function 2 With Hash-based message authentication code Secure Hashing Algorithm 1/256/512) which is quite secure but calculated relatively fast when compared to Bcrypt. To be safe for future brute force/rainbow table attacks, we are implementing Bcrypt as a 3rd party library which is slow to calculate making it secure for today's standards.

**Generating Issue IDs**

IDs should be unique for each issues, to generate an ID we follow the steps below:

- Create an empty string.
- Add the ıssues.
- Add the salted and hashed password of the user (Bcrypt).
- Then hash the final string using SHA256 (Secure Hashing Algorithm-256). It is important to note that if we were to use the original passwords while creating the ID. We would have introduced a weak point in the system. We are using SHA256 when hashing the ID which is weaker compared to Bcrypt and faster to compute, so attackers could target IDs to find the passwords. To prevent this from happening, we are using the salted and hashed version of the password of the issue, so even if an attacker finds the building blocks of the ID, he/she still needs to get through Bcrypt to find the original password. So we are trying to make the system secure by hashing the sensitive information with state-of-the-art algorithms but to make the system hacker-proof, we need to introduce another security measure to the system. Although we are hashing everything, we need to send this sensitive information to the database, through the internet. A man-in-the-middle attack can still cause great damage to the design of our system. To deal with this problem we are using Google's Firebase as our database. Firebase encrypts the data in transit using HTTPS (HyperText Transfer Protocol Secure). It uses ECDH (Elliptic Curve Diffie-Hellman) for key exchange and SHA256 with RSA (Rivest– Shamir–Adleman) for the signature algorithm which makes sure that the information is securely sent to the database.

In our System , Admin System is a unique user.It can create/delete users (project manager, project member, board member, guest), projects issues. Can change projects state from verified to done and assign projects to a project manager and create and edit profiles.

On the other hand Project Manager is a Main users of specific projects. They Can create/delete issues and change the project's state from in review to verified and assign issues to users and set priority and deadline of issues.Board Member is Users who are assigned to a specific board. They can change the project's state from “selected for development” to “in progress” and change the project's state from in progress to in review and edit the issue title and add comments to issues and only edit/view their board on the project.

On the other hand Project Member are users who are assigned to a specific project. They can change the project's state from “selected for development” to “in progress" and change the project's state from in progress to in review and edit the issue title and add comments to issues and edit/view their board on the project and Can view other boards on the project.

Guest are Non-users of the system they can view a specific board if permitted by the admin of the system.

# 10) Test cases

## Admin Action's Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | Add an User | Admin adds users to the system. | User added to the system. | Info. Message<br>'User added to the system successfully.' |
| 2. | Remove an User | Admin removes an user to the system. | User removed to the system. | Info. Message<br>'User removed from the system successfully.' |

## Admin/Project Manager Action's Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | Create Project | Admin/Project Manager adds a project to the system | Project added to the system. | Info. Message 'A project added to the system successfully.' |
| 2. | Create Issue | Admin/Project Manager adds an issue to the specified project. | Issue has been added to the specified project. | Info. Message 'An issue added to the specified project successfully.' |
| 3. | Delete Project | Admin/Project Manager removes a project to the system | Project removed to the system | Info. Message 'A project removed from the system successfully.' |
| 4. | Delete Issue | Admin/Project Manager removes an issue to the specified project. | Issue removed to the specified Project. | Info. Message 'An issue removed from the specified project successfully.' |

| | | | | |
|---|---|---|---|---|
| 5. | Change Issue Status | Admin/Project Manager changes from verified to done. | The issue is changed to done. | Info. Message<br>‘ The issue is set to done. ‘ |
| 6. | View All Boards | Admin/Project Manager displays all boards in the project. | All boards in the project are listed. | All boards are listed. |
| 7. | Review Completed Tasks | Admin/Project Manager displays review completed tasks in the system. | Review completed tasks is displayed. | All review completed tasks are listed. |
| 7.1 | Change Issue Status(Review to Verified) | Admin/Project Manager changes from review to verified. | The issue is changed to verified. | Info. Message<br>‘ The issue is set to verified. ‘ |
| 7.2 | Change Issue Status(In progress) | Admin/Project Manager changes from review to “in progress”. | The issue is changed to “in progress”. | The issue is changed to “in progress”. |
| 8. | Create Child Issue | Admin/Project Manager adds a child issue to the specified issue. | Child issue has been added to the specified issue. | Info. Message<br>‘A child issue added to the specified issue successfully.’ |
| 9. | Delete Child Issue | Admin/Project Manager removes a child issue to the specified issue. | A child issue removed to the specified issue. | Info. Message<br>‘A child issue removed from the specified issue successfully.’ |

| | | | | |
|---|---|---|---|---|
| 10. | Assign Issue to User | Admin/Project Manager assigns a issue to a user. | The user is assigned to the issue. | Info. Message<br><br>‘The user has been assigned to the issue successfully’ |
| 11. | Set Issue Priority | Admin/Project Manager sets issue priority. | The priority of the issue has been set. | Info. Message<br><br>‘ The priority of the issue has been set successfully. |
| 12. | Set Issue Deadlines | Admin/Project Manager sets issue deadlines. | The deadline of the issue has been set. | Info. Message<br><br>‘ The deadline of the issue has been set successfully. |

## Board Member/Project Member Action’s Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | Create Profile | Board Member/Project Member creates profile in system. | The profile is created. | Info. Message<br><br>‘The profile is created successfully.‘ |
| 2. | Edit Profile | Board Member/Project Member edits profile. | The profile information updated | Info. Message<br><br>‘The profile is updated successfully.‘ |
| 3. | Change Issue Status(Development to in progress) | Board Member/Project Member changes issue’s status from “development” to “in progress”. | The issue’s status is changed to “in progress”. | Info. Message<br><br>‘The issue’s status is set to in progress.‘ |

| | | | | |
|---|---|---|---|---|
| 4. | Change Issue Status(In progress to in review) | Board Member/Project Member changes issue's status from "in progress" to " review". | The issue's status is changed to "review". | Info. Message<br><br>'The issue's status is set to review.' |
| 6. | Edit Issue Title | Board Member/Project Member edits issue's title. | The issue's title updated. | Info. Message<br><br>'The issue's title is updated successfully.' |
| 7. | Add Comment to Issue | Project Member/Board Member adds a comment to<br><br>the issue | Comment has been added to the specified issue. | Info. Message<br><br>'A comment added to the specified issue successfully.' |
| 8. | Search Issue | Searches for the desired issue | The searched issue is listed. | If there is a searched issue, it is listed, otherwise, no issue is found. |
| 9. | Call related member on issue | Brings the member who is interested in that issue. | Displays the searched member. | Info. Message<br><br>'Show member information |

## Board Member Action's Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | View his/her board | Board member displays his/her board in Project. | His/her board in the project are displayed. | His/her board is listed. |

## Project Member Action's Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | Display all boards | Project member displays all boards in Project. | All boards in the project are listed. | All boards are listed. |

## Guest Action's Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1. | Display | The guest displays the board shared with her/him. | The shared board is displayed. | The board is displayed. |

## Issue Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1 | Compare issue to another issue: another issue object is given | Priority of both issues | returns 1 when higher, 0 when equal and -1 when lower priority | returns 1 when higher, 0 when equal and -1 when lower priority |
| 2.1 | add child issue: stores null issue as child issue | Issue: null | returns false | returns false |

| | | | | |
|---|---|---|---|---|
| 2.2 | add child issue: stores valid issue as child issue | Issue: that exists | adds child issue | adds child issue |
| 3.1 | Set due date: old date is given | Date: old date | returns false | returns false |
| 3.2 | Set due date: valid date is given | Date: future date | sets due date as given | sets due date as given |
| 4.1 | Add comment: empty line or null user are given | String: empty<br>User: null | returns false | returns false |
| 4.2 | Add comment: valid line and valid user are given | String: normal string<br>User: existing user | Adds comment with given string | Adds comment with given string |

## Issue List Test Cases

| Test No | Test Name | Action | Expected Result | Expected Feedback |
|---|---|---|---|---|
| 1.1 | Search in issue list: invalid id is given to search for issue | id: less or equal to 0 | couldnt find, returns null | couldnt find, returns null |
| 1.2 | Search in issue list: valid id is given to search for issue | id: that exists and bigger than 0 | returns Issue | returns Issue |
| 2.1 | Search in issue list: invalid label is given to search for issue | Label: "" or null | couldnt find, returns null | couldnt find, returns null |

| | | | | |
|---|---|---|---|---|
| 2.2 | Search in issue list: valid label is given to search for issue | Label: that exists | returns Issue | returns Issue |
| 3.1 | Add issue to issue list: null issue object is given | Issue: null | returns false | returns false |
| 3.2 | Add issue to issue list: valid issue object is given | Issue: newly created issue | adds issue and returns true | adds issue and returns true |
| 4.1 | Delete issue from issue list: null issue object is given | id: less or equal to 0 | returns false | returns false |
| 4.2 | Delete issue from issue list: valid issue object is given | id: that exists and bigger than 0 | deletes issue and returns true | deletes issue and returns true |

## Additional Test Cases and Results.

| Test No | Test case description | Test data | Expected result | Actual result |
|---|---|---|---|---|
| 1 | Login: valid username & valid password | username: testdata password: testdata | Successful Login | Successful Login |
| 2 | Login: invalid username & invalid | username: invaliduser password: | Show warning message for 3 trials. | Show warning message for 3 trials. |

| | | | | |
|---|---|---|---|---|
| | password | invalidpass | After third warning, exit from app | After third warning, exit from app |
| 3 | Login: valid username & invalid password | username: testdata password: invalidpass | Show warning message for 3 trials. After third warning, exit from app | Show warning message for 3 trials. After third warning, exit from app |
| 4 | Sorting by due dates: valid IssueList including randomly added issues is given will be sorted by due dates | IssueList:{date1,date2,date3,date4} due dates date3>date2>date4>date1 | IssueList sorted in ascending order {date3,date2,date4,date1} | Given issuelist didn't change, an arraylist of issues which consists of issues of given issuelist is sorted by due dates of issues and returned. |
| 5 | Sorting by create times: valid IssueList including randomly added issues is given will be sorted by create times | IssueList:{date1,date2,date3,date4} create times date3>date2>date4>date1 | IssueList sorted in ascending order {date3,date2,date4,date1} | Given issuelist didn't change, an arraylist of issues which consists of issues of given issuelist is sorted by create times of issues and returned. |
| 6 | Sorting by due dates: invalid IssueList is given (null) | IssueList : null | Nothing to sort, null should be returned. | Nothing to sort, returned null. |
| 7 | Sorting by create times: invalid IssueList is given (null) | IssueList : null | Nothing to sort, null should be returned. | Nothing to sort, returned null. |

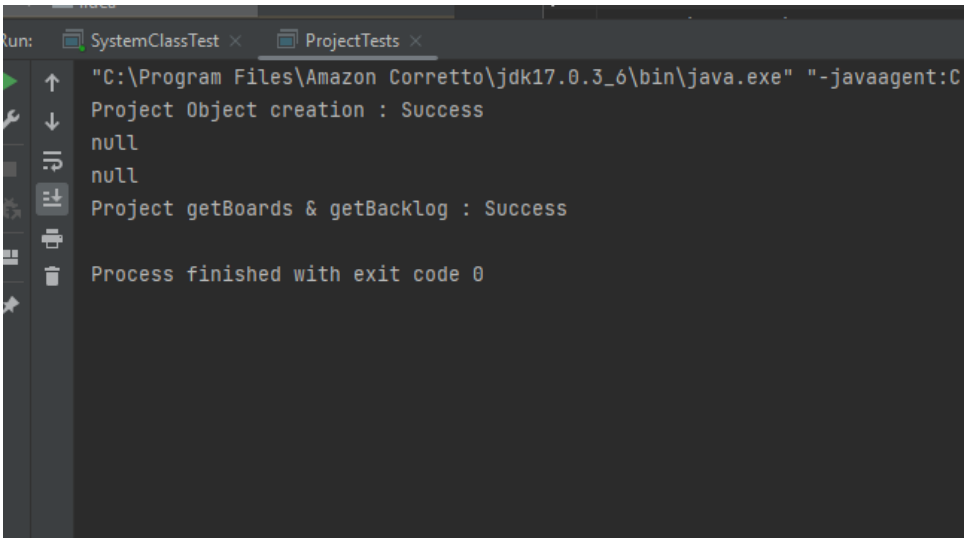
| | | | | |
|---|---|---|---|---|
| 8 | create issue: valid IssueList is given to add new issue | IssueList : valid issue list | Issue should be created succesfully and added into the given issue list | Issue is created succesfully and added into the given issue list |
| 9 | create issue: invalid IssueList is given to add new issue | IssueList : null | There is no valid issue list to add the issue, so creation shouldn't occur | There is no valid issue list to add the issue, creation didn't occur |
| 10 | Add member to board: valid members are given | Member: {0, "m1", "Member One", 505, "Software"} and two more similar member | Members added successfully. Also AVL tree balance acquired. | Members added and accessible. Tree balance is true. |
| 11 | Add issue to board: valid issues are given | Issue: {0, "Issue One", development, task} and two more similar issue | Issues added successfully. | Issues added and accessible. |
| 12 | Add issue to backlog: valid issues are given | Issue: {0, "Issue One", development, task} and two more similar issue | Issues added successfully to stack. | Issues added and accessible. |
| 13 | Remove issue from backlog: an existing issue given | Issue: {0, "Issue One", development, task} | Issue removed succesfully. | Issue removed and stack accessible. |
| 14 | Try to remove non-existing issue from backlog | Issue: {5, "Issue Six", development, task} | No items are removed. | Issue not found, returns false. |
| 15 | Remove last issue from backlog | Backlog object to manipulate | Last issue was removed successfully and LIFO action acquired. | The last added issue has been removed. |

## System Class Test

```
Project Object creation : Success
Username : gtu1
Password : sdf
Wrong username or password..
You have 2 entrie(s) left. Try again..
Username : gtu2
Password : gtu123456
Wrong username or password..
You have 1 entrie(s) left. Try again..
Username : gtu5
Password : gtu12345678
Login Sucesfully...
Admin
System Login : Success
Username : gtu1
Password : gtu123456
Login Sucesfully...
ProjectManager
System Login : Success
true
Add User & Add Project : Success

Process finished with exit code 0
```

## Project Class Test




## Admin Test Results

```
try{
    System.out.println(x: "Valid issue list given");
    admin.createIssue(issueList, id: 1, title: "title", Issue.Status.development, Issue.Type.task);
    if(issueList.getIssues().size() > 0){
        System.out.println(x: "Issue created succesfully.");
    }else{
        System.out.println(x: "Issue couldn't be created!");
    }
}catch(Exception e){
    System.err.println(x: "Issue couldn't be created!");
}
```

```
Valid issue list given
Issue created succesfully.
```

```
try{
    System.out.println(x: "Invalid issue list given");
    admin.createIssue(nullIssueList, id: 1, title: "title", Issue.Status.development, Issue.Type.task);
}catch(Exception e){
    System.err.println(x: "Issue couldn't be created!");
}
```

```
Invalid issue list given
Issue couldn't be added, try to add somewhere else.
```

```java
try{
    System.out.println(x: "Admin has valid system");
    admin.createProject(id: 1, key: "key", name: "name", type: "type", lead: null, members: null,
                        boards: null, new Backlog());
    if(system.getProjects().size() > 0){
        System.out.println(x: "Project created succesfully");
    }else{
        System.out.println(x: "Project couldn't be created!");
    }

}catch(Exception e){
    System.err.println(x: "Project couldn't be created!");
}
```

```
Admin has valid system
Project created succesfully
```

```java
try{
    System.out.println(x: "Admin does not have a system");
    admin_no_system.createProject(id: 1, key: "key", name: "name", type: "type", lead: null,
                                  members: null, boards: null, new Backlog());
}catch(Exception e){
    System.err.println(x: "Project couldn't be created!");
}
```

```
Admin does not have a system
Project cannot be created, invalid system!
```

```java
try{
    admin.createUser(usertype: "projectmanager", id: 1, username: "username",
    "fullname", contact: 1, teams: "teams", project: null, assignedBoards: null);
    System.out.println(x: "User created succesfully");
}catch(Exception e){
    System.err.println(x: "User couldn't be created!");
}
try{
    admin.createUser(id: 1, username: "username", fullname: "fullname", contact: 1, teams: "teams");
    System.out.println(x: "User created succesfully");
}catch(Exception e){
    System.err.println(x: "User couldn't be created!");
}
try{
    admin.createUser(id: 1, username: "username", fullname: "fullname", contact: 1, teams: "teams", invitedBoard: null)
    System.out.println(x: "User created succesfully");
}catch(Exception e){
    System.err.println(x: "User couldn't be created!");
}
```

```
User created succesfully
User created succesfully
User created succesfully
```

## Project Manager Test Results

```
Project project = new Project();
User user = new ProjectMember(id: 1, username: "username", fullname: "fullname", contact: 1, teams: "teams",
 project, assignedBoards: null);
Issue issue = new Issue(id: 1, title: "title", Issue.Status.development, Issue.Type.task);
ProjectManager projectManager = new ProjectManager(id: 1, username: "username", fullname: "fullname"
                              , contact: 1, teams: "teams", project, assignedBoards: null);
projectManager.assignUser(issue, user);
if(issue.getAssignees().contains(user)){
    System.out.println(x: "User assigned succesfully");
}else{
    System.err.println(x: "User couldn't be assigned!");
}
```

```
User assigned succesfully
```

```
System.out.println(x: "Invalid issue, error will occur");
projectManager.assignUser(issue: null, user);
```

```
Invalid issue, error will occur
User cannot be assigned because the issue doesn't exist.
```

```
System.out.println(x: "Invalid user, error will occur");
projectManager.assignUser(issue, user: null);
```

```
Invalid user, error will occur
User couldn't be added because user doesn't exist
```

```
System.out.println(x: "Invalid issue and user, error will occur");
projectManager.assignUser(issue: null, user: null);
```

```
Invalid issue and user, error will occur
User cannot be assigned because the issue doesn't exist.
```

## Project Member Test Results

```
    Backlog backlog = new Backlog(issues);
    Project project3 = null;
    Project project2 = new Project();
    Project project = new Project(id: 1, key: "key", name: "name", type: "type", lead: null, members: null,
                    boards: null, backlog);
    ProjectMember projectMember = new ProjectMember(id: 1, username: "username", fullname: "fullname",
                                contact: 1, teams: "teams",project,assignedBoards: null);
    projectMember.viewBacklog();
    ProjectMember projectMember2 = new ProjectMember(id: 1, username: "username", fullname: "fullname",
                                  contact: 1, teams: "teams", project2, assignedBoards: null);
    ProjectMember projectMember3 = new ProjectMember(id: 1, username: "username", fullname: "fullname",
                                contact: 1, teams: "teams", project3, assignedBoards: null);
```

```
{ issues='[1
description1Fri Dec 05 22:10:49 TRT 2025High, 2
description2Fri Dec 05 22:10:49 TRT 2025High, 3
description3Fri Dec 05 22:10:49 TRT 2025High, 41
description4Fri Dec 05 22:10:49 TRT 2025High]'}
```

```
try{
    projectMember2.viewBacklog();
}catch(Exception e ){
    System.err.println(x: "An error occured when displaying backlog");
}
```

```
There is no backlog to view.
```

```
try{
    projectMember.viewProject();
}catch(Exception e){
    System.err.println(x: "An error occured when displaying project");
}
```

```
Project name: name
Project id: 1
Project key: key
Project type: type
Project Manager of the project: null
```

```
try{
    projectMember3.viewProject();
}catch(Exception e){
    System.err.println(x: "An error occured when displaying project");
}
```

```
There is no project to view
```

## QuickSort Test Results

```java
IssueList issueList = new IssueList(issues,newTitle: "title");
ArrayList<Issue> sorted = QuickSort.sortByCreateDate(issueList);

System.out.println(x: "Sorted by create time");
for(int j = 0 ; j < sorted.size();j++){
    System.out.println(sorted.get(j));
}
```

```
Sorted by create time
0
descriptionThu Dec 05 22:01:07 TRT 2222High
1
descriptionSun Jan 05 22:01:07 TRT 2234High
2
descriptionSat Feb 03 22:01:07 TRT 2244High
3
descriptionMon Mar 05 22:01:07 TRT 2255High
4
descriptionThu Sep 27 22:01:07 TRT 2266High
```

```java
sorted = QuickSort.sortByDueDate(issueList);
System.out.println(x: "Sorted by due date");
for(int j = 0 ; j < sorted.size();j++){
    System.out.println(sorted.get(j));
}
```

```
Sorted by due date
0
descriptionThu Dec 05 22:01:07 TRT 2222High
1
descriptionSun Jan 05 22:01:07 TRT 2234High
2
descriptionSat Feb 03 22:01:07 TRT 2244High
3
descriptionMon Mar 05 22:01:07 TRT 2255High
4
descriptionThu Sep 27 22:01:07 TRT 2266High
```

```
System.out.println(x: "What happens if duplicate dates exist:");
for(int j = 0; j < 3;j++){
    issue = new Issue(i, "eray"+i, Issue.Status.development, Issue.Type.task);
    issue.setDueDate(date5);
    issue.setDescription(description: "description");
    issue.setPriority(Priority.High);
    issues.add(issue);
    i++;
}
sorted = QuickSort.sortByCreateDate(issueList);
System.out.println(x: "Sorted by create time");
for(int j = 0 ; j < sorted.size();j++){
    System.out.println(sorted.get(j));
}
```

```
What happens if duplicate dates exist:
Sorted by create time
0
descriptionThu Dec 05 22:01:07 TRT 2222High
1
descriptionSun Jan 05 22:01:07 TRT 2234High
2
descriptionSat Feb 03 22:01:07 TRT 2244High
3
descriptionMon Mar 05 22:01:07 TRT 2255High
4
descriptionThu Sep 27 22:01:07 TRT 2266High
6
descriptionThu Sep 27 22:01:07 TRT 2266High
7
descriptionThu Sep 27 22:01:07 TRT 2266High
5
descriptionThu Sep 27 22:01:07 TRT 2266High
```

```
sorted = QuickSort.sortByDueDate(issueList);
System.out.println(x: "Sorted by due date");
for(int j = 0 ; j < sorted.size();j++){
    System.out.println(sorted.get(j));
}
```

```
Sorted by due date
0
descriptionThu Dec 05 22:01:07 TRT 2222High
1
descriptionSun Jan 05 22:01:07 TRT 2234High
2
descriptionSat Feb 03 22:01:07 TRT 2244High
3
descriptionMon Mar 05 22:01:07 TRT 2255High
5
descriptionThu Sep 27 22:01:07 TRT 2266High
6
descriptionThu Sep 27 22:01:07 TRT 2266High
7
descriptionThu Sep 27 22:01:07 TRT 2266High
4
descriptionThu Sep 27 22:01:07 TRT 2266High
```

```
System.out.println(x: "What happens if null issuelist is given to be sorted");
IssueList issueList2 = null;
System.out.println(x: "Sorted by create time");
System.out.println(QuickSort.sortByCreateDate(issueList2));
System.out.println(x: "Sorted by due date");
System.out.println(QuickSort.sortByDueDate(issueList2));
```

```
What happens if null issuelist is given to be sorted
Sorted by create time
null
Sorted by due date
null
```

## Board and Backlog Class Test Results

```
// Test members
BoardMember m1 = new BoardMember( id: 0,  username: "m1",  fullname: "Member One",  contact: 505,  teams: "Software");
BoardMember m2 = new BoardMember( id: 1,  username: "m2",  fullname: "Member Two",  contact: 505,  teams: "Electronics");
BoardMember m3 = new BoardMember( id: 2,  username: "m3",  fullname: "Member Three",  contact: 505,  teams: "Mechanics");
```

```
// Test issues
Issue s1 = new Issue( id: 0,  title: "Issue One", Issue.Status.development, Issue.Type.task);
Issue s2 = new Issue( id: 1,  title: "Issue Two", Issue.Status.development, Issue.Type.story);
Issue s3 = new Issue( id: 2,  title: "Issue Three", Issue.Status.inProgress, Issue.Type.bug);
```

```java
boolean success = false;
try{
    board.addMember(m1);
    board.addMember(m2);
    board.addMember(m3);
    success = true;
}catch (Exception e){
    System.out.println("Error: " + e);
}
System.out.println("Add member success: " + success);
System.out.println(board.getMembers().toString());
```

```java
success = false;
try{
    board.addIssueList(issueList);
    board.addIssue(s1, index: 0);
    board.addIssue(s2, index: 0);
    board.addIssue(s3, index: 0);
    success = true;
}catch (Exception e){
    System.out.println("Error: " + e);
}
System.out.println("Add issue success: " + success);
```

```
*** Test Board ***
Add member success: true
0: User{id=2, username='m3', fullname='Member Three', contact=505, teams='Mechanics'}
  0: User{id=0, username='m1', fullname='Member One', contact=505, teams='Software'}
    null
    null
  0: User{id=1, username='m2', fullname='Member Two', contact=505, teams='Electronics'}
    null
    null

Add issue success:true
Issue title: Issue One, Issue Two, Issue Three,
```

```java
success = false;
try {
    backlog.addIssue(s1);
    backlog.addIssue(s2);
    backlog.addIssue(s3);
    success = true;
} catch (Exception e) {
    System.out.println("Error: " + e);
}
System.out.println("Add issue: " + success);
System.out.println(backlog.getIssues().toString()
        + "\n");

System.out.println("Get top issue: " +
        backlog.getTopIssue().toString() + "\n");
```

```java
success = false;
try {
    backlog.removeIssue(s1);
    backlog.removeLastIssue();
    success = true;
} catch (Exception e) {
    System.out.println("Error: " + e);
}
System.out.println("Remove first and last issue: "
        + success);
System.out.println(backlog.getIssues().toString());
```

```
*** Test Backlog ***
Add issue: true
[{0, Issue One}, {1, Issue Two}, {2, Issue Three}]

Get top issue: {2, Issue Three}

Remove first and last issue: true
[{1, Issue Two}]
```

**Issue Class Test Results**

```
Issue i1, i2;
i1 = new Issue( id: 1,  title: "i1", Issue.Status.inProgress, Issue.Type.epic);
i2 = new Issue( id: 2,  title: "i2", Issue.Status.inProgress, Issue.Type.epic);

try{
    System.out.println(i1.setTitle("title"));
    System.out.println("Issue getLogHistory: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue getLogHistory: Failed");
}

try{
    System.out.println(i1.setDescription("description"));
    System.out.println("Issue setDescription: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue setDescription: Failed");
}

try{
    i1.updateUpdateTime();
    System.out.println("Issue updateUpdateTime: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue updateUpdateTime: Failed");
}

try{
    System.out.println(i1.setDueDate(new Date( year: 2022, month: 06, date: 15)));
    System.out.println("Issue setDueDate: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue setDueDate: Failed");
}
```

```
try{
    i1.setPriority(Issue.Priority.High);
    System.out.println("Issue setPriority: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue setPriority: Failed");
}

try{
    i1.setStatus(Issue.Status.inProgress);
    System.out.println("Issue setStatus: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue setStatus: Failed");
}

try{
    i1.setType(Issue.Type.story);
    System.out.println("Issue setType: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue setType: Failed");
}

try{
    System.out.println(i1.addComment( line: "bla bla", new BoardMember( id: 1,  username: "c",  fullname: "b",  contact: 2,  teams: "a")));
    System.out.println("Issue addComment: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue addComment: Failed");
}
```

```
    System.out.println(i1.removeComment(i1.getLastComment()));
    System.out.println("Issue removeComment: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue removeComment: Failed");
}

try{
    System.out.println(i1.addAssignee( new BoardMember( id: 1,  username: "c",  fullname: "b",  contact: 2,  teams: "a")));
    System.out.println("Issue addAssignee: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue addAssignee: Failed");
}

try{
    System.out.println(i1.removeAssignee(new BoardMember( id: 1,  username: "c",  fullname: "b",  contact: 2,  teams: "a")));
    System.out.println("Issue removeAssignee: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue removeAssignee: Failed");
}

try{
    System.out.println(i1.toString());
    System.out.println("Issue toString: Success");
}catch (Exception e){
    System.err.println(e+"\n"+"Issue toString: Failed");
}

try {
    System.out.println(i1.compareTo(i2));
    System.out.println("Issue compare: Success");
} catch (Exception e) {
    System.err.println("Issue compare: Failed");
}
```

```
try {
    System.out.println(i1.getId());
    System.out.println("Issue getId: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getId: Failed");
}

try {
    System.out.println(i1.getTitle());
    System.out.println("Issue getTitle: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getTitle: Failed");
}

try {
    System.out.println(i1.getDescription());
    System.out.println("Issue getDescription: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getDescription: Failed");
}

try {
    System.out.println(i1.getCreateTime());
    System.out.println("Issue getCreateTime: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getCreateTime: Failed");
}

try {
    System.out.println(i1.getUpdateTime());
    System.out.println("Issue getUpdateTime: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getUpdateTime: Failed");
}
```

```
try {
    System.out.println(i1.getDueDate());
    System.out.println("Issue getDueDate: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getDueDate: Failed");
}

try {
    System.out.println(i1.getPriority());
    System.out.println("Issue getPriority: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getPriority: Failed");
}

try {
    System.out.println(i1.getStatus());
    System.out.println("Issue getStatus: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getStatus: Failed");
}

try {
    System.out.println(i1.getType());
    System.out.println("Issue getType: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getType: Failed");
}

try {
    System.out.println(i1.addChildIssue(i2));
    System.out.println("Issue addChildIssue: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue addChildIssue: Failed");
}
```

```java
try {
    System.out.println(i1.getChildIssue().getTitle());
    System.out.println("Issue getChildIssue: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getChildIssue: Failed");
}

try {
    System.out.println(i1.getLastComment().getData());
    System.out.println("Issue getLastComment: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getLastComment: Failed");
}

try {
    System.out.println(i1.getAssignees());
    System.out.println("Issue getAssignees: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getAssignees: Failed");
}

try {
    System.out.println(i1.getLogHistory());
    System.out.println("Issue getLogHistory: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue getLogHistory: Failed");
}
```

```
true
Issue getLogHistory: Success
true
Issue setDescription: Success
Issue updateUpdateTime: Success
true
Issue setDueDate: Success
Issue setPriority: Success
Issue setStatus: Success
Issue setType: Success
true
Issue addComment: Success
true
Issue removeComment: Success
true
Issue addAssignee: Success
true
Issue removeAssignee: Success
1
descriptionSat Jul 15 00:00:00 TRT 3922High
Issue toString: Success
0
Issue compare: Success
1
Issue getId: Success
title
Issue getTitle: Success
description
Issue getDescription: Success
Tue Jun 14 23:44:15 TRT 2022
Issue getCreateTime: Success
Tue Jun 14 23:44:15 TRT 2022
```

```
Issue getCreateTime: Success
Tue Jun 14 23:44:15 TRT 2022
Issue getUpdateTime: Success
Sat Jul 15 00:00:00 TRT 3922
Issue getDueDate: Success
High
Issue getPriority: Success
inProgress
Issue getStatus: Success
story
Issue getType: Success
true
Issue addChildIssue: Success
i2
Issue getChildIssue: Success

Issue getLastComment: Success
ProjectManagementSystem.SkipList@433c675d
Issue getAssignees: Success
[]
Issue getLogHistory: Success
```

```java
try {
    System.out.println(list.addIssue(new Issue( id: 1,  title: "1", Issue.Status.development, Issue.Type.epic)));
    System.out.println(list.addIssue(new Issue( id: 2,  title: "2", Issue.Status.development, Issue.Type.bug)));
    System.out.println(list.addIssue( i: null));
    System.out.println(list.addIssue(new Issue( id: 3,  title: "3", Issue.Status.inProgress, Issue.Type.story)));
    System.out.println("Issue List addIssue: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List addIssue : Failed");
}
list.getIssues().get(0).setDueDate(new Date( year: 2022,  month: 6,  date: 12));
list.getIssues().get(1).setDueDate(new Date( year: 2022,  month: 3,  date: 12));
list.getIssues().get(0).setPriority(Issue.Priority.High);
list.getIssues().get(1).setPriority(Issue.Priority.Low);

try {
    System.out.println(list.deleteIssue( id: -5));
    System.out.println(list.deleteIssue( id: 3));
    System.out.println("Issue List deleteIssue: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List deleteIssue : Failed");
}

try {
    System.out.println(list.getIssues().toString());
    System.out.println("Issue List getIssues: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List getIssues : Failed");
}
```

```
    System.out.println(list.getIssueQueue().toString());
    System.out.println("Issue List getIssueQueue: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List getIssueQueue : Failed");
}

try {
    System.out.println(list.getTitle());
    System.out.println("Issue List getTitle: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List getTitle : Failed");
}

try {
    System.out.println(list.setTitle("bla bla 2"));
    System.out.println("Issue List setTitle: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List setTitle : Failed");
}

try {
    System.out.println(list.searchInIssueList( title: "bla bla 2"));
    System.out.println(list.searchInIssueList( title: "2"));
    System.out.println("Issue List searchInIssueList: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List searchInIssueList : Failed");
}
try {
    System.out.println(list.searchInIssueList( id: 2));
    System.out.println(list.searchInIssueList( id: 13));
    System.out.println("Issue List searchInIssueList: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Issue List searchInIssueList : Failed");
```

```
Issue List addIssue: Success
false
true
Issue List deleteIssue: Success
[1
nullWed Jul 12 00:00:00 TRT 3922High, 2
nullWed Apr 12 00:00:00 TRT 3922Low]
Issue List getIssues: Success
[1
nullWed Jul 12 00:00:00 TRT 3922High, 2
nullWed Apr 12 00:00:00 TRT 3922Low]
Issue List getIssueQueue: Success
test
Issue List getTitle: Success
true
Issue List setTitle: Success
null
2
nullWed Apr 12 00:00:00 TRT 3922Low
Issue List searchInIssueList: Success
2
nullWed Apr 12 00:00:00 TRT 3922Low
null
Issue List searchInIssueList: Success
```

```java
try {
    System.out.println(com.getId());
    System.out.println("Comments getId: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments getId: Failed");
}

try {
    System.out.println(com.getOwner().toString());
    System.out.println("Comments getOwner: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments getOwner: Failed");
}

try {
    System.out.println(com.getData());
    System.out.println("Comments getData: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments getData: Failed");
}

try {
    System.out.println(com.getCreationDate().toString());
    System.out.println("Comments getCreationDate: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments getCreationDate: Faile
}

try {
    System.out.println(com.setId(12));
    System.out.println(com.setId(-5));
    System.out.println("Comments setId: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments setId: Failed");
}
```

```java
try {
    System.out.println(com.setOwner(new BoardMember( id: 12,  username: "bb",  fullname: "bb",  contact: 1,  teams: "a")));
    System.out.println(com.setOwner(null));
    System.out.println("Comments setOwner: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments setOwner: Failed");
}

try {
    System.out.println(com.setData("new line"));
    System.out.println(com.setData(""));
    System.out.println("Comments setData: Success");
} catch (Exception e) {
    System.err.println(e + "\n" + "Comments setData: Failed");
}
```

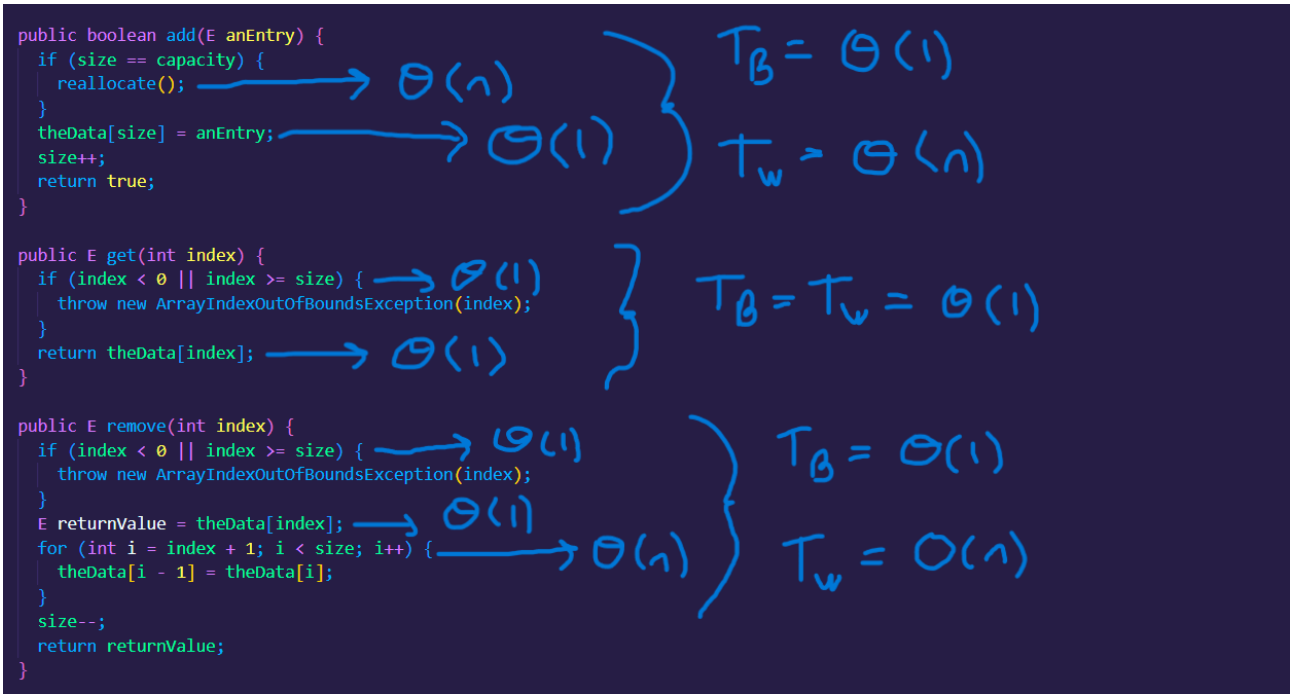
```
1
Comments getId: Success
User{id=12, username='bb', fullname='bb', contact=1, teams='a'}
Comments getOwner: Success
bla bla
Comments getData: Success
Tue Jun 14 23:44:15 TRT 2022
Comments getCreationDate: Success
true
false
Comments setId: Success
true
false
Comments setOwner: Success
true
false
Comments setData: Success
```

# 11) Performance analysis

**List**

List is a data structure we used to store assignees, comments and log history for each issue. Hence all those data won't be ordered and it won't be too large list is the most suitable data structure for this data.

| Operation | Best | Worst | Average |
|---|---|---|---|
| add(Adding issue) | Θ(1) | O(n) | Ω(n) |
| remove(Removing issue) | Θ(1) | O(n) | Ω(n) |
| get(Viewing issue) | Θ(1) | O(1) | Ω(1) |

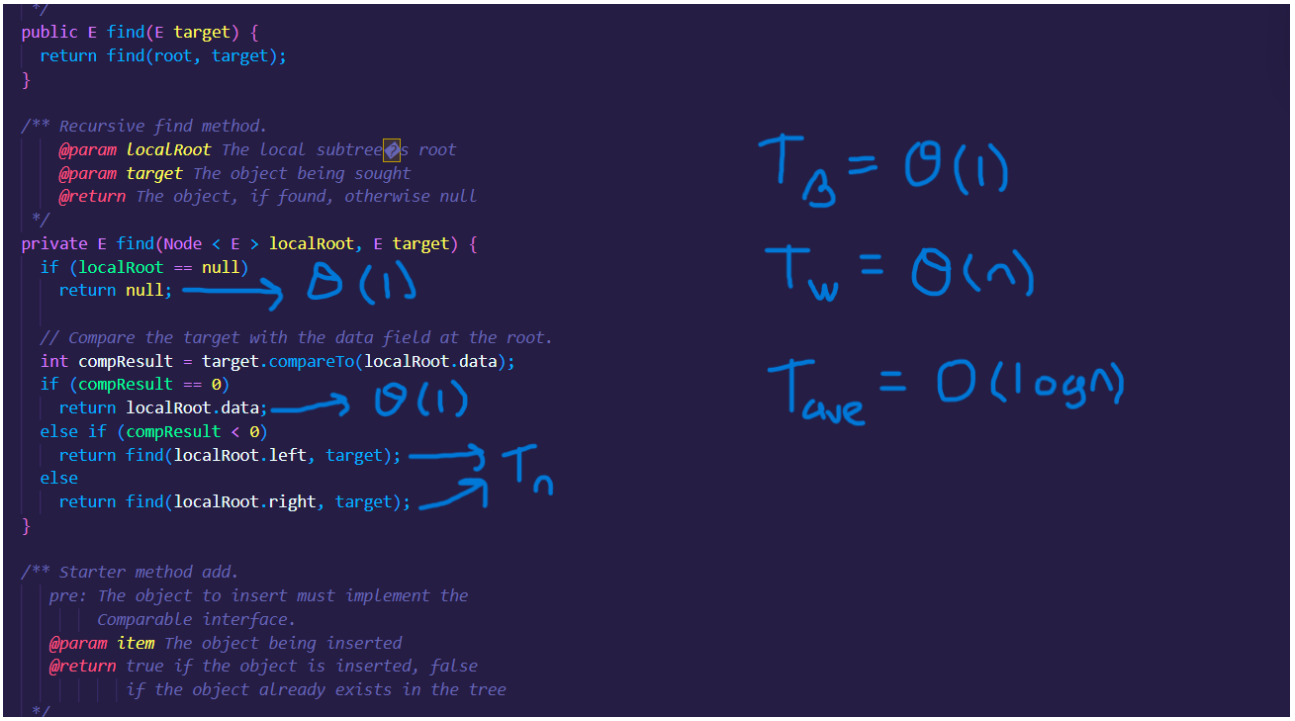


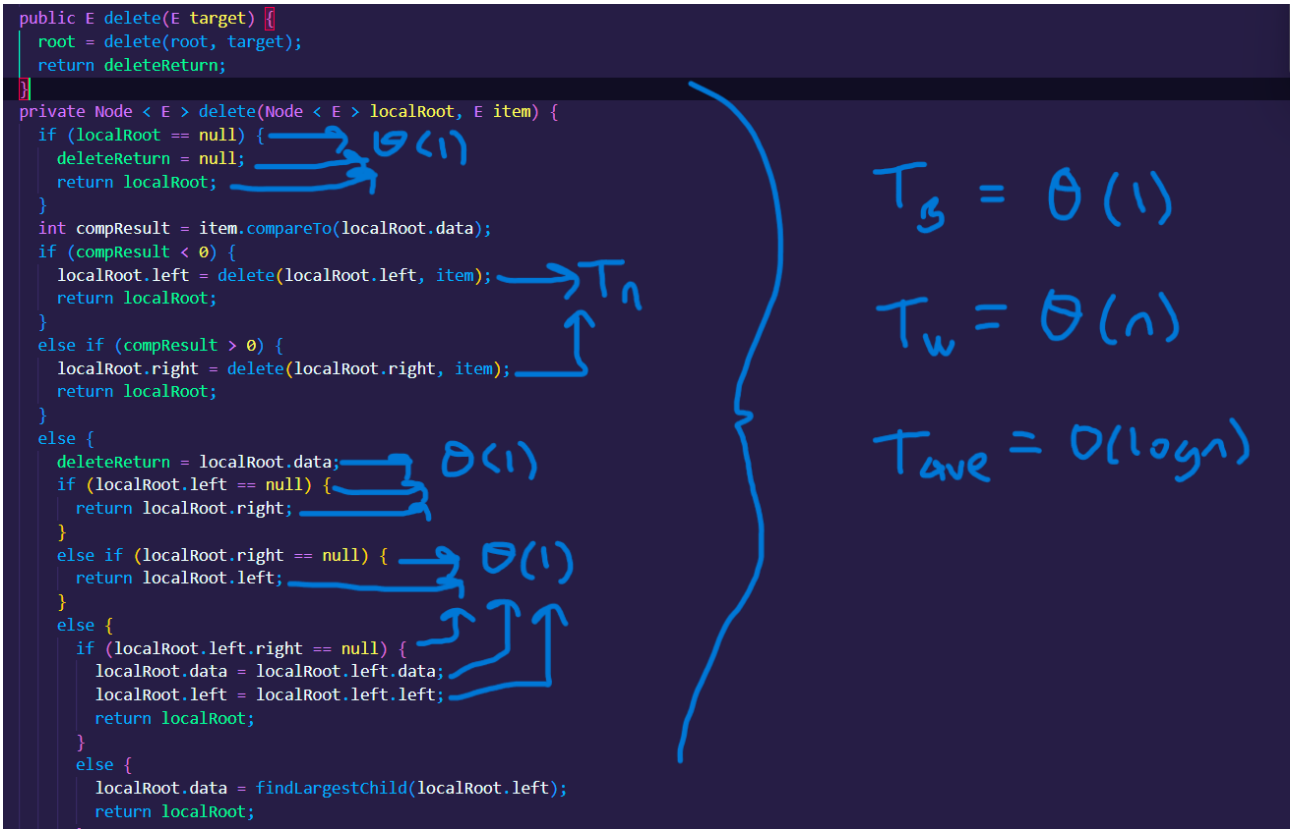

**Binary Search Tree**

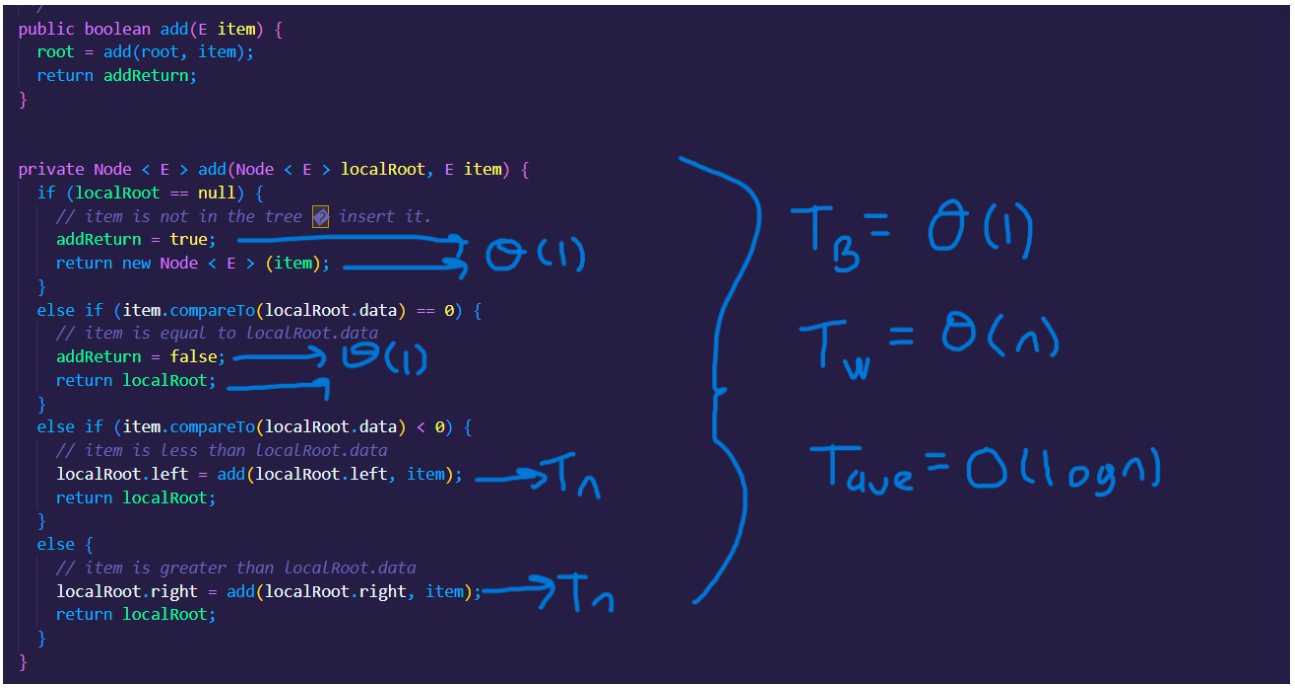
Binary search tree is used to store Project members. Searching a member in the Binary search tree is easy and as compared to array and linked lists, insertion and deletion operations are faster in BST.

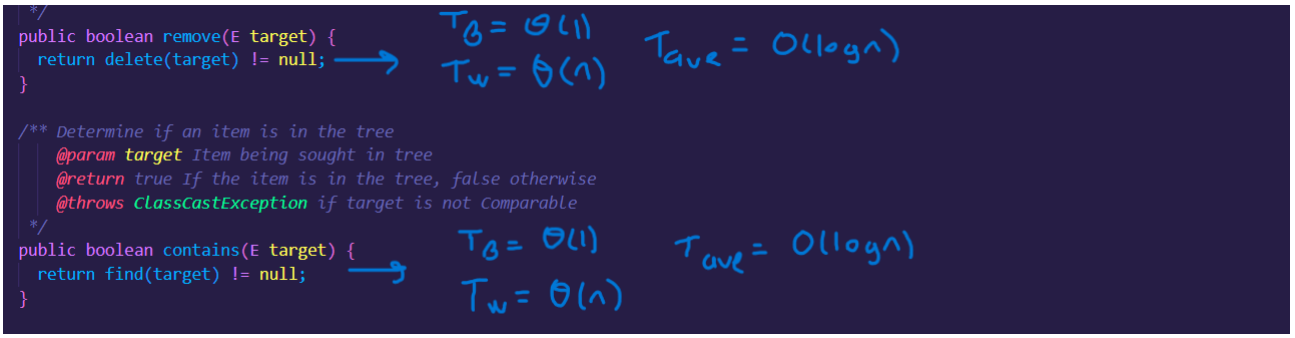
| Operation | Best | Worst | Average |
|---|---|---|---|
| boolean add(E item) | Θ(1) | O(n) | O(logn) |
| boolean contains(E target) | Θ(1) | O(n) | O(logn) |
| E find(E target) | Θ(1) | O(n) | O(logn) |
| E delete(E target) | Θ(1) | O(n) | O(logn) |

| | | | |
|---|---|---|---|
| boolean remove(E target) | Θ(1) | O(n) | O(logn) |






**Priority Queue**

We are using Priority Queue to sort issues by their priorities so people working with the system could easily understand which issues are most important at that time.

| Operation | Best | Worst | Average |
|---|---|---|---|
| Offer (Adding issue) | Θ(1) | O(n) | O(log(n)) |
| Poll (Removing issue) | Θ(1) | O(n) | O(log(n)) |
| Peek/top (Viewing issue) | Θ(1) | O(1) | Ω(1) |

```java
public boolean offer(E item) {
  theData.add(item);
  int child = theData.size() - 1;
  int parent = (child - 1) / 2; // Find child's parent.
  while (parent >= 0 && compare(theData.get(parent),
                                theData.get(child)) > 0) {
    swap(parent, child);
    child = parent;
    parent = (child - 1) / 2;
  }
  return true;
}
```

$\Theta(1)$ $\Theta(1)$ $O(n)$

$T_B = \Theta(n)$

$T_W = O(n)$

$T_{ave} = O(\log n)$

```java
public E poll() {
  if (isEmpty()) {
    return null;
  }
  E result = theData.get(0);
  if (theData.size() == 1) {
    theData.remove(0);
    return result;
  }
  theData.set(0, theData.remove(theData.size() - 1));

  int parent = 0;
  while (true) {
    int leftChild = 2 * parent + 1;
    if (leftChild >= theData.size()) {
      break; // Out of heap.
    }
    int rightChild = leftChild + 1;
    int minChild = leftChild;
    if (rightChild < theData.size()
        && compare(theData.get(leftChild),
                   theData.get(rightChild)) > 0) {
      minChild = rightChild;
    }
    if (compare(theData.get(parent),
                theData.get(minChild)) > 0) {
      swap(parent, minChild);
      parent = minChild;
    }
    else { // Heap property is restored.
      break;
    }
  }
  return result;
}
```

$\Theta(1)$ $\Theta(1)$ $O(n)$ $\Theta(1)$

$T_n = O(\log n)$

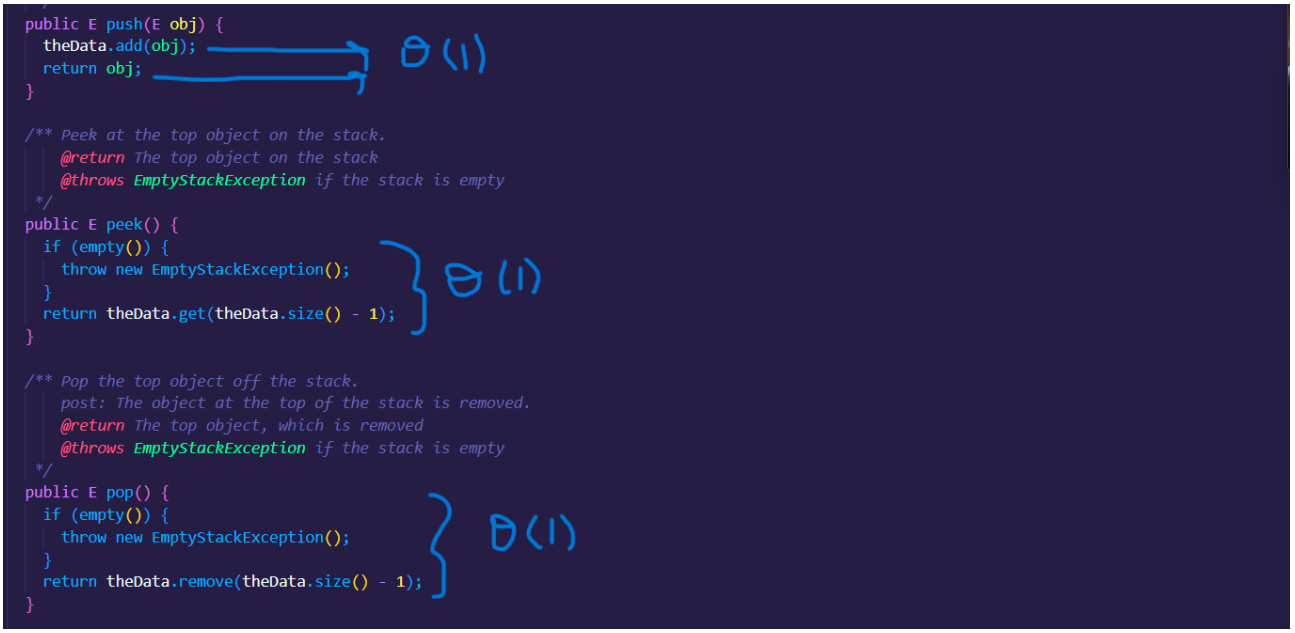
```java
public E peek() {
  return null;
}
```

$\Theta(n)$

**Stack**

Backlog is the issue stack where all the issues are added at the beginning and distributed to the boards from there. Because of its similar structure, we implemented it using stack. We performed performance analysis of operations such as adding, removing and viewing backlog issues.

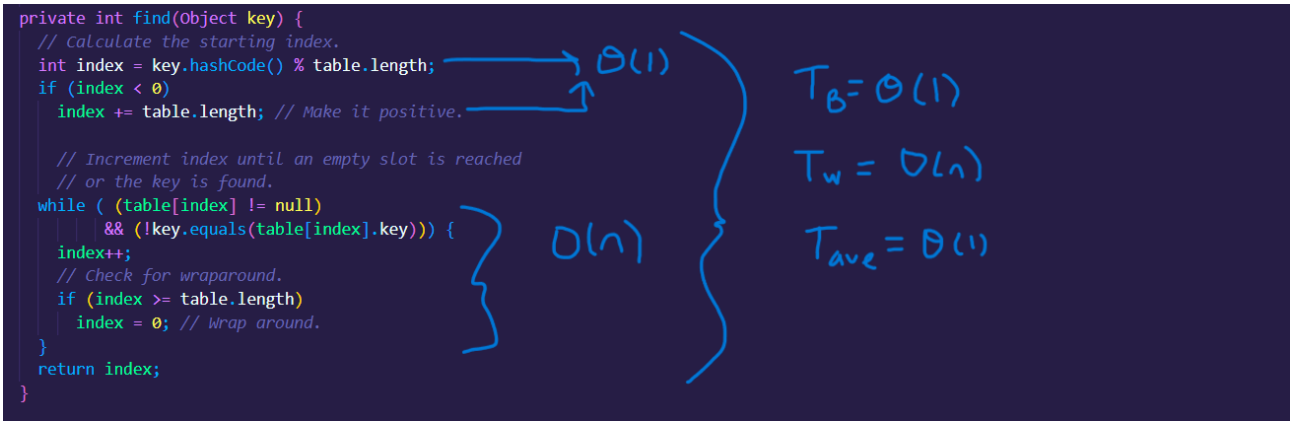
| Operation | Best | Worst | Average |
|---|---|---|---|
| Push (Adding issue) | Θ(1) | O(1) | Ω(1) |
| Pop (Removing issue) | Θ(1) | O(1) | Ω(1) |
| Peek (Viewing issue) | Θ(1) | O(1) | Ω(1) |

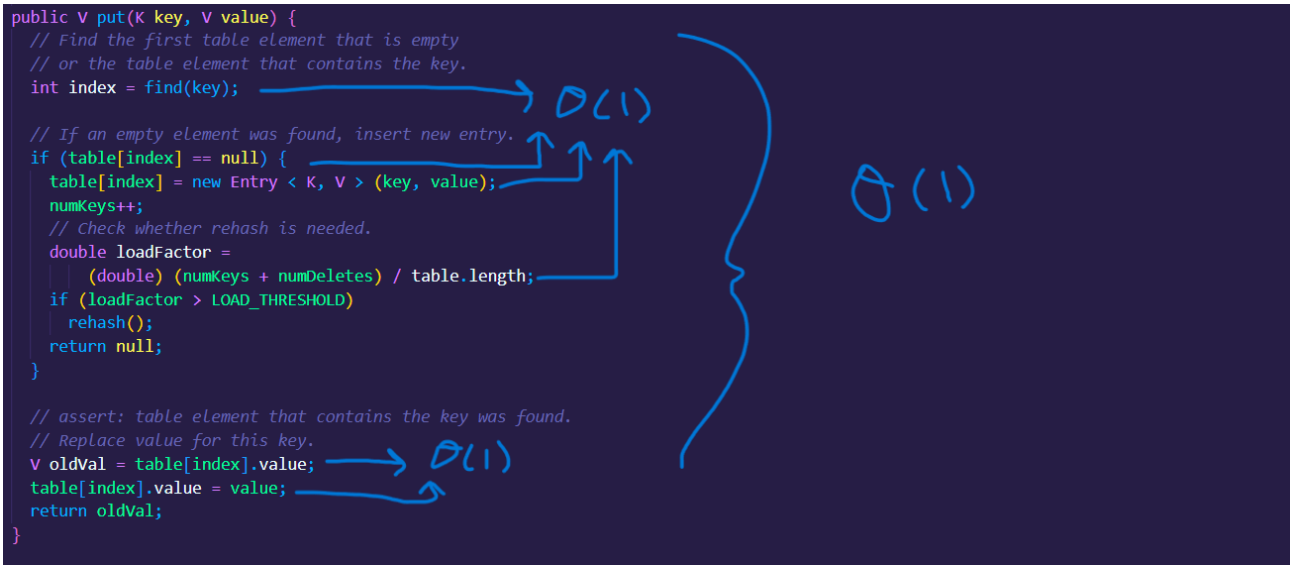


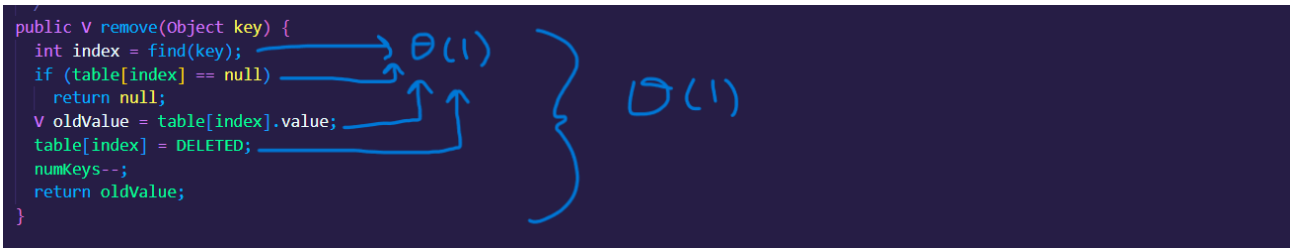
**HashMap**

With a key in a hash map, it is quick and easy to find the information(values) corresponding to the key. For this reason, we used the hashmap to keep user records-informations, find users and search among the users, as we can quickly access other information of users with their IDs as a key.

| Operation | Best | Worst | Average |
|---|---|---|---|
| Search (Search user) | Θ(1) | O(n) | Θ(1) |
| Insert (Add user) | Θ(1) | O(n) | Θ(1) |
| Delete (Delete user) | Θ(1) | O(n) | Θ(1) |






**Skip List**

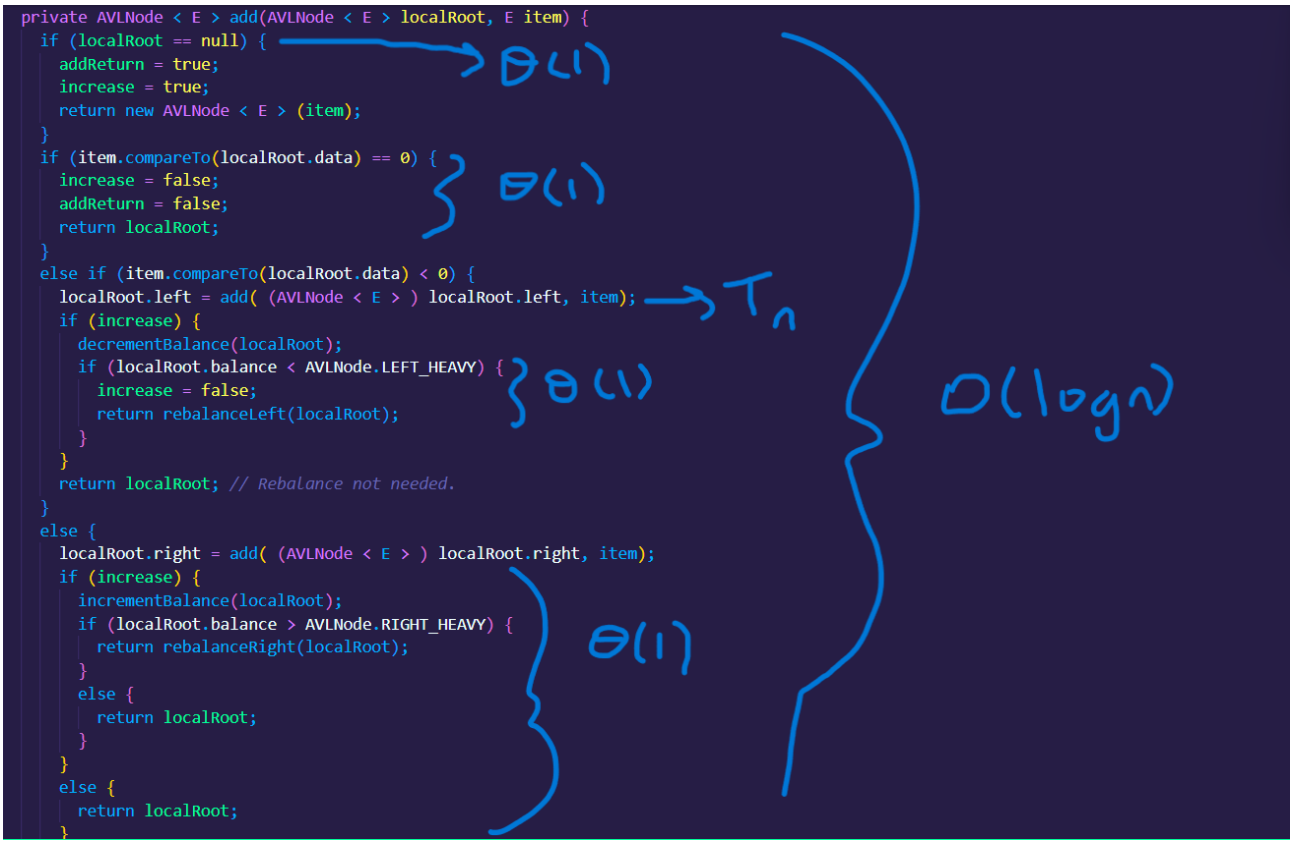
We get faster results in searches in the average case with skip list. It allows us to display the data that will appear as a result of the search efficiently and quickly. For this reason, we used skip list while keeping the list of users which is assigned to the specific issues and used on searching operations for users in the issue part.

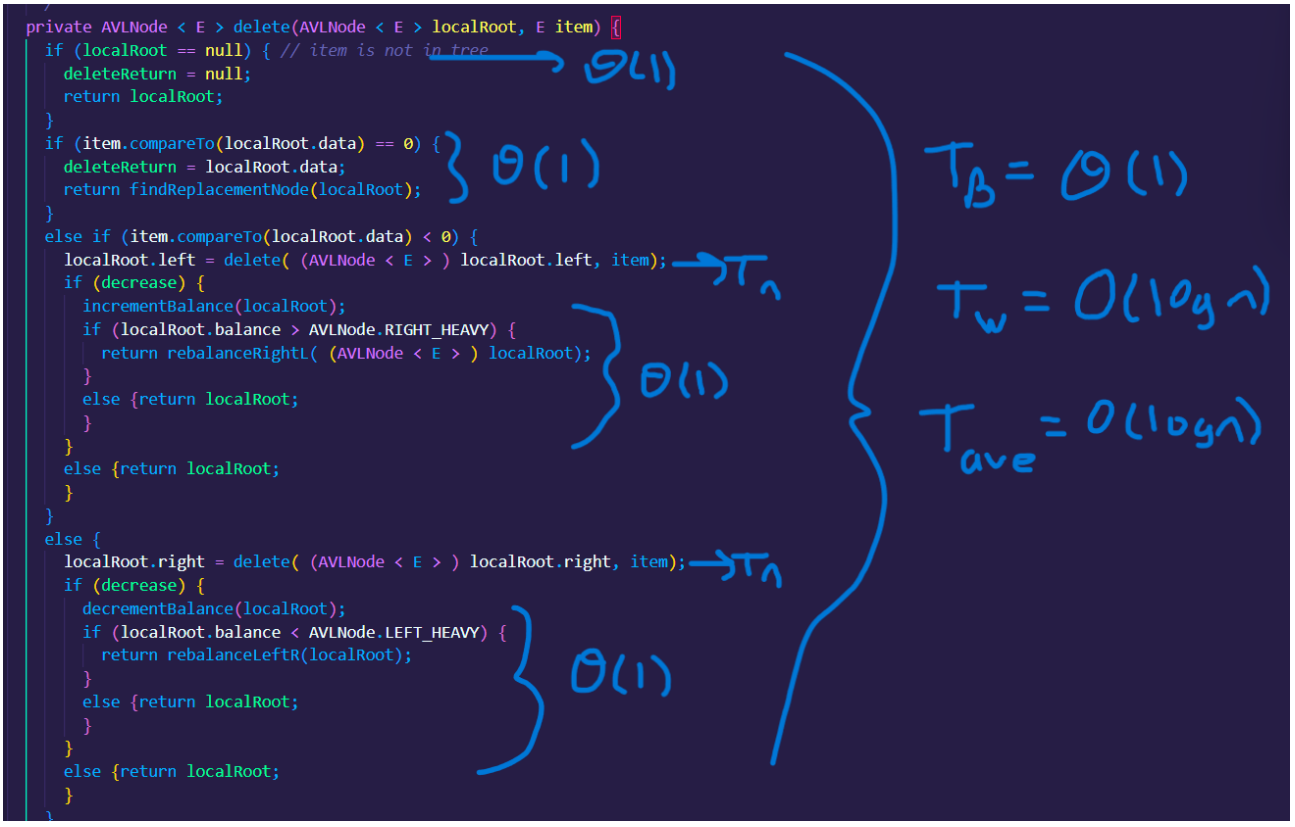
| Operation | Best | Worst | Average |
|---|---|---|---|
| Search (Search user) | Θ(1) | O(n) | Θ(logn) |
| Insertion (Add user) | Θ(1) | O(n) | Θ(logn) |
| Deletion (Delete user) | Θ(1) | O(n) | Θ(logn) |

**Balanced Binary Search Tree**

We used balanced binary search tree instead of binary search tree on the board page/part for making searches more efficient and displaying the results more quicker in the worst case scenarios.

| Operation | Best | Worst | Average |
|---|---|---|---|
| boolean add(E item) | Θ(1) | O(logn) | O(logn) |
| boolean contains(E target) | Θ(1) | O(logn) | O(logn) |
| E find(E target) | Θ(1) | O(logn) | O(logn) |
| E delete(E target) | Θ(1) | O(logn) | O(logn) |
| boolean remove(E target) | Θ(1) | O(logn) | O(logn) |


Θ(1)
Θ(1)
$T_n$
Θ(1)
Θ(1)
O(log n)


Θ(1)
Θ(1)
$T_n$
Θ(1)
$T_n$
Θ(1)
$T_B = \Theta(1)$
$T_w = O(\log n)$
$T_{ave} = O(\log n)$

**Graph**

Since we do not want to move on to the next issues before other issues related to the specific issue(s) are completed, we used a graph structure in the issue section to show the links between the issues.

| Operation | Best | Worst | Average |
|---|---|---|---|
| addVertex | O(\|V\|²) | O(\|V\|²) | O(\|V\|²) |
| addEdge | Θ(1) | Θ(1) | Θ(1) |
| removeVertex | O(\|V\|²) | O(\|V\|²) | O(\|V\|²) |
| removeEdge | Θ(1) | Θ(1) | Θ(1) |
| query(Check) | Θ(1) | Θ(1) | Θ(1) |

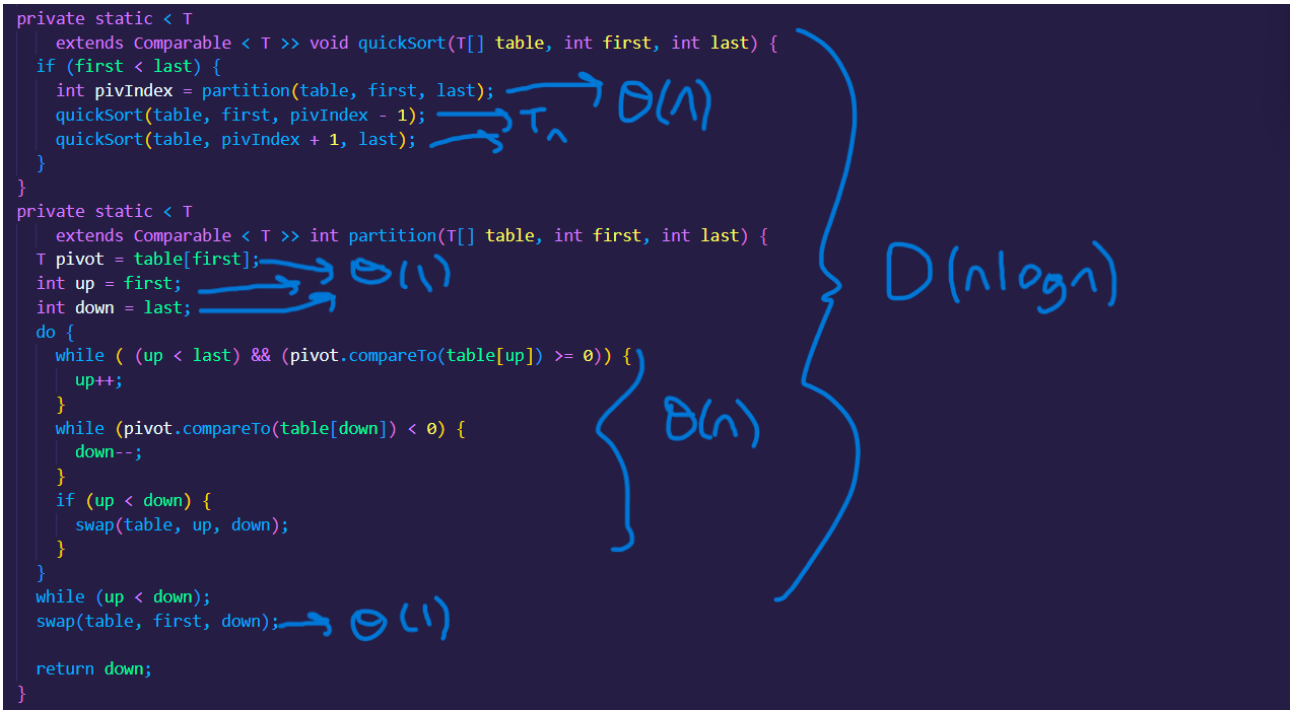
```
public boolean isEdge(int source, int dest) {
  return edges[source].contains(new Edge(source, dest));
}

/** Insert a new edge into the graph.
    @param edge The new edge
 */
public void insert(Edge edge) {
  edges[edge.getSource()].add(edge);
  if (!isDirected()) {
    edges[edge.getDest()].add(new Edge(edge.getDest(),
                                       edge.getSource(),
                                       edge.getWeight()));
  }
}
```

**Quicksort**

We used quicksort to sort the issues quickly and efficiently according to some properties such as createDate and dueDate.

| Operation | Best | Worst | Average |
|---|---|---|---|
| Search (Search user) | Θ(nlogn) | Θ(nlogn) | Θ(n²) |




**Analysis of Other Methods**

| Operation | Best | Worst | Average |
|---|---|---|---|
| Login(Member member) | Θ(1) | Θ(1) | Θ(1) |
| Profile menu() | Θ(1) | Θ(1) | Θ(1) |
| getBoard(int id) | O(1) | O(n) | O(n) |
| editBoard(Board board) | Θ(1) | Θ(1) | Θ(1) |

| | | | |
|---|---|---|---|
| getBacklog() | Θ(1) | Θ(1) | Θ(1) |

# 12) Revised Parts

**Removed:**

| | |
|---|---|
| FR14 | System must have Planning Section and Development Section. |

Development section was considered unneccesary because in our vision of the system it should be more general public oriented and not development oriented.

| | |
|---|---|
| FR15 | Planning section must have Board, Issues, and Components (like hardware, software, mechanics etc.). |

Includes description of Planning section which is the only section that our system has and repeats previous functional requirements.

| | |
|---|---|
| FR18 | Development section must have source code and releases sections. |

Includes description of Development section which was considered unnecessary.

| | |
|---|---|
| FR19 | Releases section must have version, status, progress, start date, release date and a description. |

Releases is a part of development section which was considered unnecessary

**Changed:**

| | |
|---|---|
| FR12 | Issues must have types like bug, task, sub-task, story, epic (Epic issues, which represent high-level initiatives or bigger pieces of work in Jira. For software teams, an epic may represent a new feature they're developing. For IT service teams, epics may represent a major service change or upgrade.). |

Sub-task was considered unnecessary because of an existence of child issues.

Problem definitions section changed with information about the data structures, sorting algorithm, where and why we used them in that specific part.

Performance analysis part changed. The HashMap, skip list, graph, balanced binary search tree data structures and quicksort sorting algorithm information (performance analysis as empirically and theoretically, where and why we used) added to that section. Theoretically results added as table, empirically results added as screenshots.

**Added:**

- Where the hash map structure is used and its performance analysis.
- Where the skip list structure  is used and its performance analysis.
- Where the graph structure  is used and its performance analysis.
- Where the balanced binary search tree structure is used and its performance analysis.
- Where the quicksort algorithm is used and its performance analysis.
- Additional test cases, also their descriptions, datas, expected results and actual results.
- We have added where we use the data structures and sorting algorithms that are required to be used in the project to the problem definitions section.