Design Document: Refactor getUsers Function / API

Status	Submitted For Review
Issue Link	https://github.com/Real-Dev-Squad/website-backend/issues/2274
Date	22-01-2025
Document Owner	Prakhar Kumar (prakhar.meerut@gmail.com)
PRD	NA / Not Required

Reviewer	Date	Action
Tejas	29-01-2025	Comments Added
Anuj Chhikara	11-02-2025	Approved
Suvidh Kaushik	11-02-2025	Approved

Overview:

To improve the code quality of the **controllers/user.js** **getUsers** api / method. Link

- Involves improving cognitive complexity - which tells how hard a unit of code is to read and understand.
- Using smaller helper functions to (one for each of the 7 query parameters) for easier debugging and management.

Goals:

- Improve readability.

- Extracting query parameters at the top of the method to separate out the core logic and improve maintainability and promote better separation of concerns.
- Decompose into smaller functions based on query parameters for easier management and easier debugging.
- Reduce cognitive complexity.

Current Problems:

- Function is too big - it gets hard to read.
- Too many large if statements - cluttered code.

Solution Approach:

- Move nested code into separate helper functions, each helper function will handle a single query parameter - (following helper functions will be created - update after checking the code)
    - handleUserById - based on id in req.query.id
    - handleUserByProfileData - based on id in req.userData.id
        - Cannot use the same one as above because the response structure is the same.
    - handleUsersByDiscordId - req.query.discordId
    - handleDepartedUsers - req.query.departed
    - handleUsersByUnmergedPrs - req.query.filterBy=unmerged_prs
    - handleOverDueTasks - req.query.days
    - handleAllUsers - default case if no query param is passed.
- Using global try catch for method instead of try catches for every block of core logic for each query parameter. (Currently for each query parameter, we have a try-catch block which increases cognitive complexity). Example Below:

```
try {
      if (req.query.id) {
         const id = req.query.id;
         let result, user;
        try {
          result = await dataAccess.retrieveUsers({ id: id });
          user = result.user;
        } catch (error) {
          logger.error(`Error while fetching user: ${error}`);
          return res.boom.serverUnavailable(SOMETHING_WENT_WRONG);
        }
        if (!result.userExists) {
```

```
          return res.boom.notFound("User doesn't exist");
        }
        return res.json({
          message: "User returned successfully!",
          user,
       });
    }
} catch () {
     return res.boom.serverUnabailable()
}
```

Cognitive Complexity:

What this basically tells is how hard a unit of code is to read and to understand. A unit of code can be a function or a piece of code in question.

Basic rules for the task at hand:

Note: For more details about the rules, check this out - [Link]

1. Increment the count by 1 for any break in linear flow like if, else-if, else, loops etc.
2. Increment the count by additional +1 when nesting occurs.

```
if () {                 // +1 here
      if () {           // +1 here and +1 due to nesting.
      }
}
```

3. Each catch clause increments the count by 1.
4. A switch statement with any number of cases only adds 1 to the count.

```
switch(true) {          // complexity of 1, irrespective of multiple cases
      Case a:
      Case b:
      Case c:
}
```

Example from the code before the proposed change:

```
if (req.query.id) { // +1
    const id = req.query.id;
    let result, user;
    try {
      result = await dataAccess.retrieveUsers({ id: id });
      user = result.user;
    } catch (error) { // +2, (+1 nesting, +1 catch block)
      logger.error(`Error while fetching user: ${error}`);
      return res.boom.serverUnavailable(SOMETHING_WENT_WRONG);
    }
    if (!result.userExists) { // +2, (+1 nesting +1 if block)
      return res.boom.notFound("User doesn't exist");
    }
    return res.json({
      message: "User returned successfully!",
      user,
    });
  } // Total = 5 (ignoring other if blocks similar to above)
```

Example from the code after change:

Controller Refactor:

```
// contoller/user.js file
try {
   const { q,
    dev: devParam,
    query,
    id,
    profile,
    discordId,
    departed
   } = req.query;
   const dev = devParam === "true";
   // Reject query usage if no dev flag set.
   if (q && !dev) return res.boom.notFound("Route not found");
```

```
    // Handle user retrieval by ID - helper function
    if (id) return handleUserById(res, id);
} catch (error) {
    logger.error(`Error while fetching users: ${error}`);
    return res.boom.serverUnavailable(SOMETHING_WENT_WRONG);
 }
```

Corresponding helper function:

```
async function handleUserById(res, userId) {
 try {
    const result = await dataAccess.retrieveUsers({ id: userId });
    if (!result.userExists) return res.boom.notFound("User doesn't
exist");
    return res.json({ message: "User returned successfully!", user:
result.user });
 } catch (error) {
    logger.error(`Error while fetching user: ${error}`);
    return res.boom.serverUnavailable(SOMETHING_WENT_WRONG);
 }
}
```

Cognitive Complexity Calculation of updated method before refactoring:

- Cognitive Complexity - 35.
    - Try catch blocks - 5
    - If-else conditions - 16
    - Loops - 3
    - Logical operators - 3
    - Nesting adjustments - 8

Cognitive Complexity Calculation of updated method after refactoring:

- Cognitive Complexity - 18
    - Try catch blocks - 1
    - If-else conditions - 11
    - Logical operators - 3
    - Nesting adjustments - 3