The code is written in **JavaScript** and is using the **Express.js** framework, which is a web application framework for **Node.js**.

Regarding the security and design issues:

1. **Insecure Direct Object References (IDOR)**:

   - The code retrieves a user based on the `id` provided in the URL without first checking if the currently logged-in user has the necessary permissions to access that user's details. An attacker might be able to fetch details of other users by simply changing the `id` in the URL.

2. **Improper Control Flow**:

   - If the currently logged-in user does not have 'admin' permissions, the code sends a 'Permission denied' error but doesn't return or halt further execution. This means the code will continue to run, and the user details will be sent in the response anyway. This is a major flaw as it allows non-admin users to access data they shouldn't be able to.

3. **Potential Memory Leak**:

   - The code caches user data in the `users` object to avoid frequent database queries. However, this approach can lead to memory usage issues if there are a large number of users because all user data will be stored in memory.

4. **Lack of Input Validation and Sanitization**:

   - The `userId` is taken directly from the URL and used to query the database without any validation or sanitization. This can expose the application to potential vulnerabilities depending on the `fetchUserFromDatabase` implementation.

5. **Ambiguous Function Implementation**:

   - The `loggedInUser()` function is called to check the permissions of the current user, but its implementation isn't provided in the code snippet. Depending on its implementation, there might be additional security concerns.

6. **Lack of Authentication**:

   - There's no evident mechanism ensuring the user is authenticated before accessing the endpoint. Even though there's a check for 'admin' permissions, the authentication process itself isn't shown.

7. **Potential Denial of Service (DoS)**:

- If an attacker sends a large number of requests with random or non-existent `userId`s, it might cause the server to make unnecessary database queries, leading to a potential DoS condition.

Recommendations:

1. **Check Permissions Earlier**:
   - Before querying the database or accessing the cache, verify the user's permissions to ensure they're allowed to fetch the requested user's details.

2. **Properly Handle Control Flow**:
   - After sending a response, ensure you return from the function to prevent further code execution.

3. **Implement Input Validation**:
   - Validate the `userId` against expected patterns or values before processing.

4. **Implement Proper Authentication**:
   - Ensure only authenticated users can access user details.

5. **Reconsider Caching Strategy**:
   - If caching is necessary, consider using a dedicated caching solution like Redis instead of storing data in memory.

By addressing these concerns, the code's security can be significantly improved.