Software Requirements and Design Document

For

Group 1

Version 1.0

Authors:

Colby Leavitt

Dylan Connolly

Ivan Lepesii

Zack Lima

1. Overview (5 points)

LibMaster is a comprehensive reimagining of FSU's library study room reservation system, designed to enhance accessibility and user experience. The platform provides students with intuitive visualization of room availability through interactive library maps, advanced filtering capabilities, streamlined booking processes, and a comprehensive *timeline view* of each floor and its rooms.

Library administrators gain powerful tools to manage room statuses, track usage analytics, and maintain library resources. Using the integrated administrator interface, faculty and staff can easily view and supervise all libraries, floors, and rooms under their jurisdiction. By modernizing the reservation workflow and adding visual elements, *LibMaster* aims to eliminate the inefficiencies of the PHP-based *LibCal* system and create a more productive library experience for the FSU community.

2. Functional Requirements (10 points)

Student Functional Requirements

1. The system shall authenticate users with FSU credentials through either a confirmation to the student's e-mail, or via the system-wide Duo authentication provider from Cisco. (HIGH)
2. The system shall display available study rooms in an interactive floor-map layout. (HIGH)
3. The system shall allow filtering rooms by capacity, floor, library, and time availability. (HIGH)
4. The system shall display a user's upcoming reservations with room details and time information. (HIGH)
5. The system shall allow users to cancel their existing reservations. (HIGH)
6. The system shall provide a timeline visualization of current and future room availability on a per-library and per-library-floor basis. (HIGH)
7. The system shall enable search functionality for library's loanable items. (MEDIUM)
8. The system shall allow users to report issues with rooms or equipment. (MEDIUM)
9. The system shall save user booking preferences for expedited future reservations. (LOW)
10. The system shall send confirmation emails for successful reservations. (LOW)
11. The system shall notify users when their reservation has been cancelled by an administrator. (LOW)

Administrator Functional Requirements

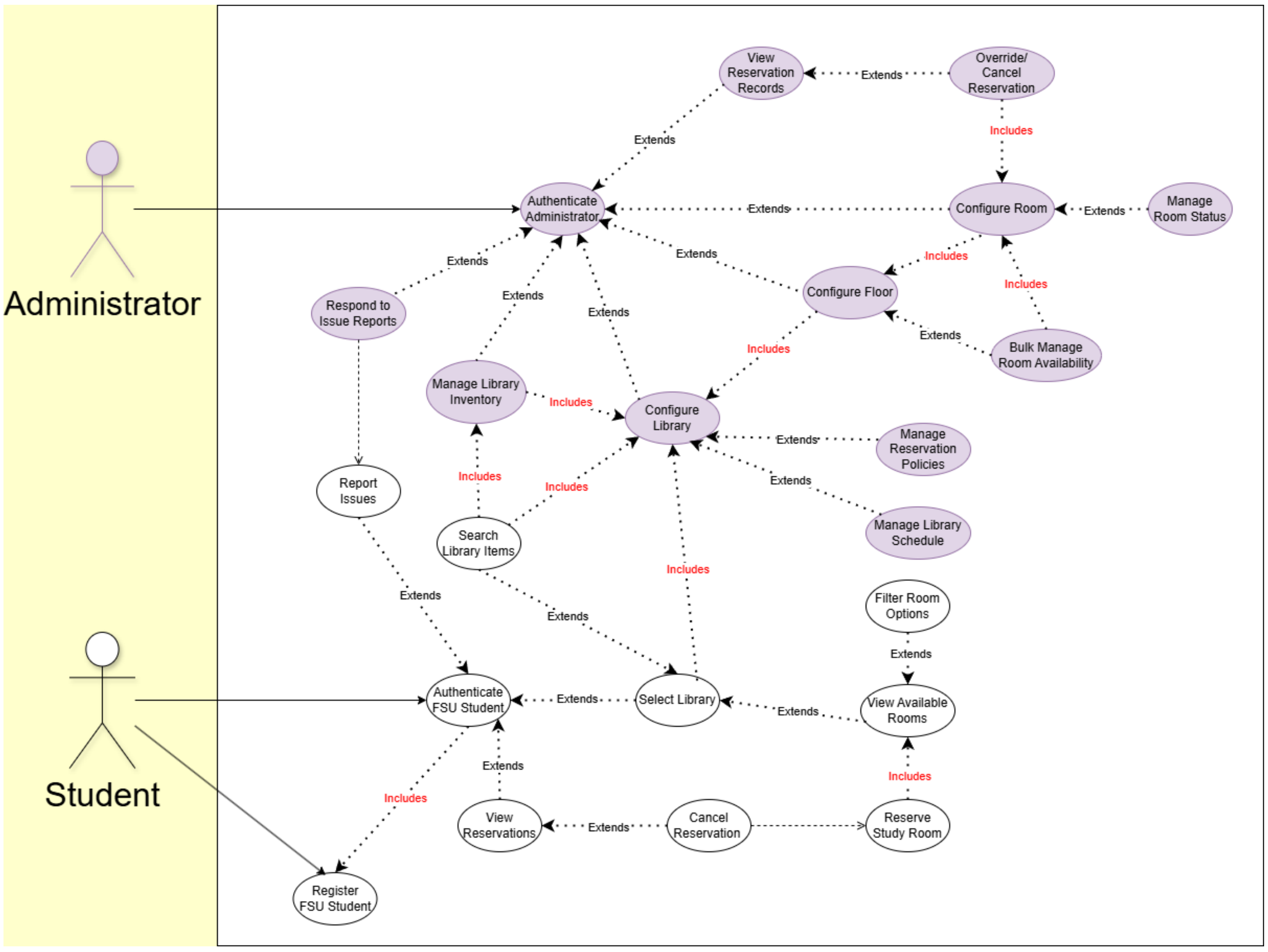
1. The system shall allow administrators to mark individual rooms as unavailable due to maintenance. (HIGH)
2. The system shall display past, current, and upcoming room reservation records with student information. (HIGH)
3. The system shall enable administrators to override or cancel existing bookings. (HIGH)
4. The system shall provide functionality to update multiple room statuses simultaneously. (MEDIUM)
5. The system shall enable management of library supply inventory. (MEDIUM)
6. The system shall allow tracking and responding to student-submitted issue reports. (MEDIUM)
7. The system shall provide adjustment capabilities for maximum reservation time limits, on a per-library, per-library-floor, and per-library-floor-room basis. (MEDIUM)
8. The system shall track equipment rental analytics and identify frequently unavailable items. (LOW)
9. The system shall provide booking analytics for space optimization and staffing decisions. (LOW)

3. Non-functional Requirements (10 points)

1. **Usability:** The system shall provide an intuitive interface requiring no training for students to make reservations.

2. **Performance:** The system shall load floor maps and room availability data in no more than three seconds.
3. **Responsiveness:** The system shall function properly on mobile devices, tablets, and desktop computers.
4. **Security:** The system shall protect user data and prevent unauthorized access to administrative functions.
5. **Reliability:** The system shall prevent double-booking of rooms for any overlapping time intervals, and shall gracefully resolve simultaneous identical room reservation requests.
6. **Accessibility:** The system shall comply with WCAG 2.1 AA standards for web accessibility.
7. **Maintainability:** The system shall employ a modular architecture to facilitate future enhancements and maintenance.
8. **Availability:** The system shall be available 24/7 with scheduled maintenance windows not exceeding 4 hours per month.
9. **Compatibility:** The system shall work correctly in modern web browsers (Chrome, Firefox, Safari, Edge, etc).
10. **Data Integrity:** The system shall maintain accurate reservation data even during concurrent booking attempts.

4. Use Case Diagram (10 points)



Actors:
Student – A regular user of the system.
Administrator – A user with elevated permissions.

Use Cases:
Authenticate (Both Student and Administrator)
Register FSU Student (Student)
Configure Library (Administrator)
Configure Floor (Admin)
Configure Room (Admin)
Manage Library Inventory (Administrator)
Override/Cancel Reservation (Admin)
View Libraries (Both User and Admin)
View Floors (Both User and Admin)
View Rooms (Both User and Admin)
Search Library Items (Both User and Admin)

Relationships:
Users and Admin can log in/log out.
Admin can add, remove, and update Libraries, Floors, Rooms, Materials, Reservations, and Users.
Users can view Libraries, Floors, Rooms, and Materials.
Users can Sign Up.
Admin can change their password.

5. Class Diagram and/or Sequence Diagrams (15 points)

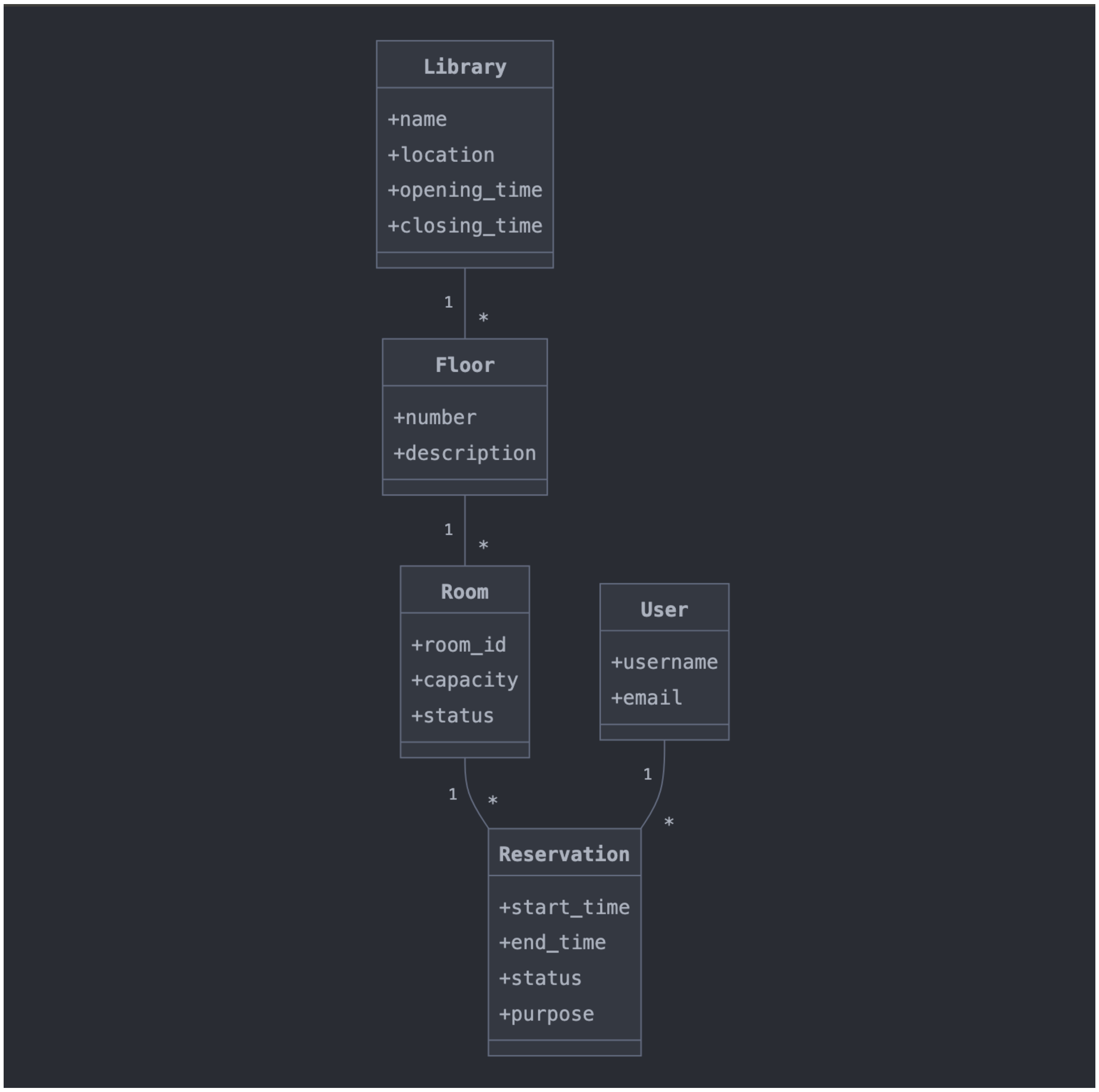
This section presents a high-level overview of the anticipated system architecture using a ***class diagram*** *and/or* ***sequence diagrams****.*

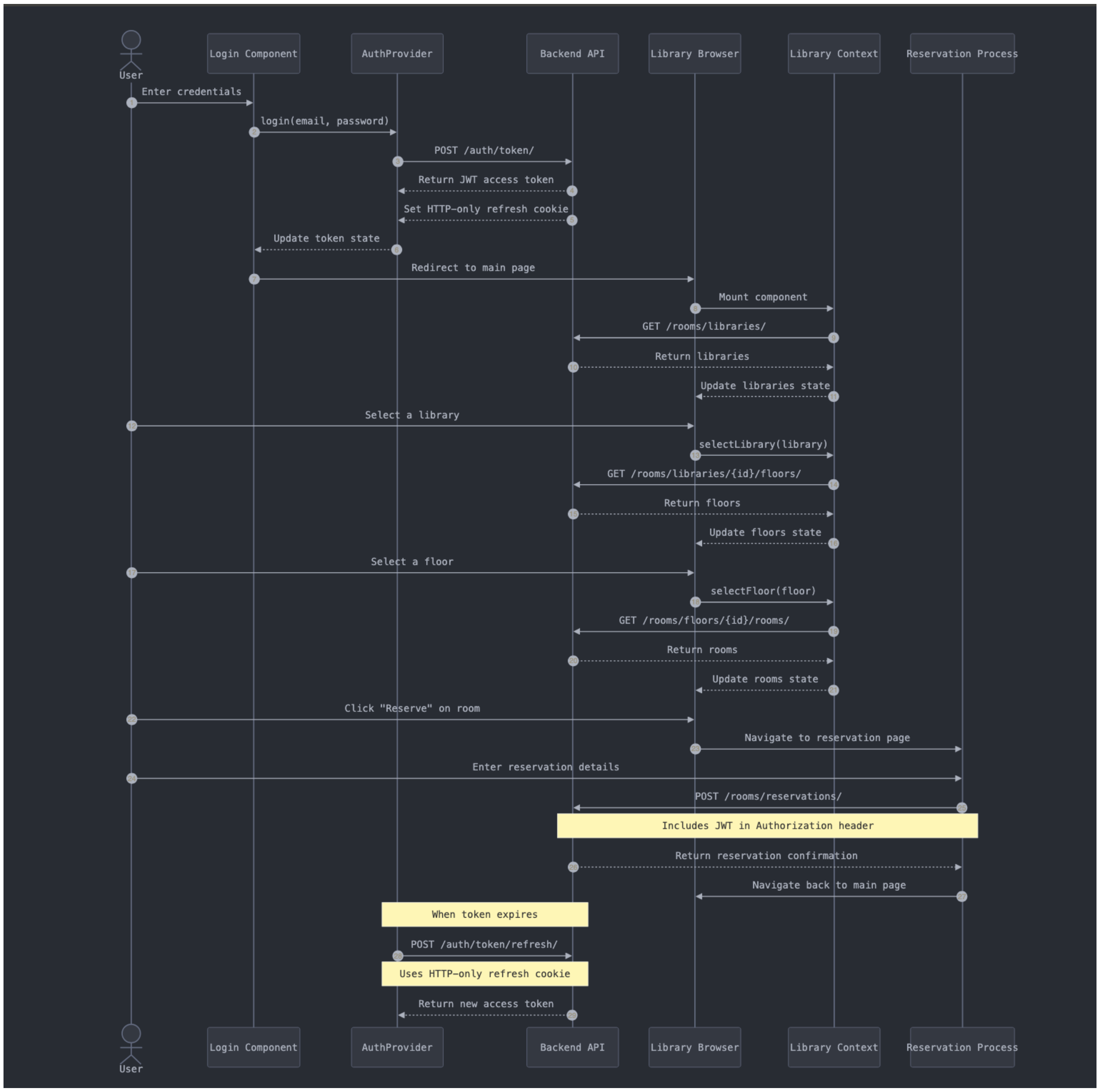
If the main ***paradigm*** *used in your project is* ***Object Oriented*** *(i.e., you have classes or something that acts similar to classes in your system), then draw the* ***Class Diagram of the entire system and Sequence Diagrams for the three (3) most important use cases in your system.***

If the main ***paradigm*** *in your system is* ***<u>not</u> Object Oriented*** *(i.e., you* ***<u>do not</u>*** *have classes or anything similar to classes in your system) then only draw* ***Sequence Diagrams*****,** ***but for <u>all</u> the use cases of your system.*** *In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the <u>functions</u> in the system involved in the action sequence.*

Class Diagrams *show the* ***fundamental objects/classes*** *that must be modeled with the system to satisfy its requirements and* ***the relationships*** *between them. Each class rectangle on the diagram* ***must also include the attributes and the methods of the class*** *(they can be refined between increments). All the* ***relationships between classes and their multiplicity*** *must be shown on the class diagram.*

A ***Sequence Diagram*** *simply depicts* ***interaction between objects*** *(or* ***functions -*** *in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*


Library
+name
+location
+opening_time
+closing_time
1
*
Floor
+number
+description
1
*
Room
+room_id
+capacity
+status
User
+username
+email
1
*
1
*
Reservation
+start_time
+end_time
+status
+purpose



6. Operating Environment (5 points)

For stakeholders contracting with the LibMaster application as system administrators: the software may be hosted from a Linux, MacOS, or Windows-based environment, on either ARM or x86-64 based instruction set architecture. Tests have been performed on Windows 11 Version 24H2, Ubuntu 24.04LTS, and MacOS Sequoia 15.3. It requires Node.js 22.13.5, Axios 1.7.9, Python 3.12.3, Django 5.1.6, PostgreSQL 17.4, React 18.2.0, TypeScript 5.7.2, PostCSS 8.4.35, TailwindCSS 3.4.1, and Vite 6.1.0 to be installed, in addition to the routing dependencies required for React. Aside from these core requirements, the system can run on any of the aforementioned environments with minimal configuration.

For stakeholders contracting with the LibMaster application as student users: The software is usable from any platform which supports a modern web browser. All core system functionalities are available through the LibMaster user portal regardless of the type of device used to access it, insofar that this device is compatible with most modern web-based application frameworks.

7. Assumptions and Dependencies (5 points)

Assumptions:

1. **Authentication Integration:** We assume the application will be able to successfully integrate with FSU's authentication system, either through email confirmation or via the university's Duo authentication provider from Cisco. If this integration proves challenging or impossible due to university policy, we may need to develop an alternative authentication approach.
2. **Access to Library Data**: We assume we will have some form of access to current library room data and scheduling information. If direct access to this data is restricted, we may need to implement manual data entry for each library's scheduling, and consider alternative data synchronization methods.
3. **Floor Plan Availability:** The floor-map visualization feature assumes we will have access to accurate floor plans for each library. If these floor plans are unavailable, incomplete, or outdated, we might need to create simplified representations or delay this feature implementation.
4. **Concurrent User Management:** We assume our database and application architecture will effectively manage scenarios where multiple users attempt to reserve the same room simultaneously. If our current conflict resolution approach proves to be insufficient in any way, additional locking mechanisms may be required.
5. **Browser and Device Compatibility**: We assume users will primarily access the system through modern web browsers on both desktops and Personal Mobile Devices (PMDs). If significant usage of legacy browsers is identified, additional compatibility work may be needed.
6. **Network Connectivity**: We assume users will have reliable internet connectivity when using the application. Limited offline functionality may need to be considered if this assumption proves to be incorrect.
7. **User Role Distinction**: We assume we can clearly differentiate between student users and administrative superusers with different permission levels. If the university's identity management system does not provide sufficient role information, additional user management features may be required.
8. **Mobile Experience**: We assume that a responsive web application will provide sufficiently satisfactory mobile functionality. If mobile usage demands more native functionality, developing a separate React Native application may become necessary.

Dependencies

1. **Frontend Technologies:** The project depends on React for web development, with the possibility of expanding to React Native for native mobile applications. We also rely on Tailwind CSS for styling.
2. **Backend Framework:** The system depends on Django for backend development, which affects the overall architecture and development approach. Significant changes to this choice would require substantial rework.

3. **Database System**: We depend on PostgreSQL for data persistence. Migration to a different database system would require changes to both the schema design and the application code.
4. **FSU Authentication Systems**: The application will ideally depend on FSU's existing authentication infrastructure. Changes or restrictions to these systems during development could impact our authentication implementation.
5. **Server Infrastructure**: The application depends on appropriate web server infrastructure for proper deployment. Changes in hosting requirements or limitations could affect our deployment strategy.
6. **Cross-Origin Resource Sharing (CORS):** As noted in the backend settings, the system depends on proper CORS configuration for communication between client and server components, particularly during development.
7. **Third-Party Libraries:** The project depends on several third-party libraries like Axios for HTTP requests, react-router for navigation, and JSON Web Token (JWT) authentication. Updates or deprecations to these libraries could necessitate codebase refactors.
8. **Data Migration:** If migrating from an existing reservation system, we depend on the availability and structure of current reservation data for potential import procedures. Using *LibCal* as an example, we would need a way to retrieve the metadata from their PHP-based application and store it on our own.
9. **Development Tools:** The project depends on development tools like TypeScript, ESLint, and Vite. Changes to these toolchains could affect the development process.
10. **JSON Web Token Infrastructure**: The application's authentication system depends on JWT implementation through Django REST framework SimpleJWT. Security updates or changes to this approach would require authentication mechanism revisions.