**Campus Security Authentication System Design (University of Eswatini)**

**Object-Oriented Design (OOD) Overview**

This system uses Object-Oriented Design (OOD) principles as outlined in the Coad/Nicola model, including the four components: Problem Domain Component (PDC), Human Interaction Component (HIC), Data Management Component (DMC), and Task Management Component (TMC). The system is built in JavaScript, supports RFID card-based student authentication, and includes real-time monitoring and alerts.

**Component Relationships Overview (Cross-Component UML)**

+----------------+ +---------------------+ +--------------------+

| Student |<>------>| ScannerUI |<------->| AuthenticationMgr |

+----------------+ +---------------------+ +--------------------+

| | |

v v v

+----------------+ +---------------------+ +--------------------+

| RFIDCard | | StatusDisplay |<--------| LogStore |

+----------------+ +---------------------+ +--------------------+

| |

v v

+----------------+ +--------------------+

| AccessPoint |----------------------------------------->| AlertManager |

+----------------+ +--------------------+

Legend:

* <> denotes association (e.g. Student owns RFIDCard).
* --> denotes a message call or data flow.
* <--> denotes bidirectional interaction.

This high-level UML shows how the PDC (Student, RFIDCard, AccessPoint), HIC (ScannerUI, StatusDisplay), DMC (LogStore), and TMC (AuthenticationManager, AlertManager) interconnect.

1. **Problem Domain Component (PDC)**

Class relationships are as follows: each Student has-a RFIDCard; the Reader in an AccessPoint passes cardIDs to AuthenticationManager; AuthenticationManager writes to AccessLog and may call AlertManager. The AuthenticationManager and AccessLog interact with the Data Management component (below) to look up credentials and store events. In practice, RFID cards are used as conventional access tokens for such systems (biometric systems are an alternative but RFID is widely deployed), so we model the card and user entities accordingly.

**Classes and Relationships**

* **Student**
  + Attributes: studentID, name, cardID
  + Services: requestAccess(), reportIssue()
* **RFIDCard**
  + Attributes: cardID, status
  + Services: activate(), deactivate()
* **AccessPoint**
  + Attributes: doorID, location
  + Services: scanCard(), unlockDoor(), lockDoor()
* **AuthenticationManager**
  + Services: verifyCard(), processScan()
* **AccessLog**
  + Attributes: timestamp, studentID, doorID, accessGranted
  + Services: createEntry()
* **AlertManager**
  + Services: checkAlerts(), generateAlert()

**UML Class Diagram (PDC)**

**Diagram Relationships:**

* **AuthenticationManager** interacts with **CredentialStore** and **LogStore** in the DMC.
* **AccessLog** entries are created by **AuthenticationManager** and later analyzed by **AlertManager**.
* **Student** and **RFIDCard** data is used by the ScannerUI (HIC) during authentication.

+--------------------+ +------------------+

| Student |1 1| RFIDCard |

|--------------------|<------|------------------|

| - studentID | | - cardID |

| - name | | - status |

| - cardID | +------------------+

|--------------------|

| +requestAccess() |

| +reportIssue() |

+--------------------+

+---------------------------+

| AccessPoint |

|---------------------------|

| - doorID |

| - location |

|---------------------------|

| +scanCard() |

| +unlockDoor() |

| +lockDoor() |

+---------------------------+

+---------------------------+

| AuthenticationManager |

|---------------------------|

| +verifyCard() |

| +processScan() |

+---------------------------+

+---------------------------+

| AccessLog |

|---------------------------|

| - timestamp |

| - studentID |

| - doorID |

| - accessGranted |

|---------------------------|

| +createEntry() |

+---------------------------+

+---------------------------+

| AlertManager |

|---------------------------|

| +checkAlerts() |

| +generateAlert() |

+---------------------------+

1. **Human Interaction Component (HIC)**

The user interface must handle RFID scanning, display access status, and show real-time alerts. We assume a JavaScript-based UI (e.g. web front-end or embedded display) with the following elements:

* **ScannerUI** – A front-end class running on or near each door. It provides a prompt (e.g. LED blink or touchscreen message) telling the student to scan the RFID card, and then immediately displays “Access Granted” or “Access Denied” based on AuthenticationManager’s result. This could be implemented with existing JS UI components; for example, instantiating widgets for status indicators as suggested by standard GUI libraries[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20User%20Interface%20Component).
* **StatusDisplay** – A component (e.g. LCD or web panel) that shows live information, such as the last few access attempts or alert icons. After each scan, the ScannerUI updates this display with the student’s name and access outcome. If an unauthorized attempt occurs, it also shows an alert message (e.g. red warning).
* **AdminDashboard** – A JavaScript web application for security staff. It has pages or widgets for “Recent Access Logs” and “Active Alerts”. The dashboard continuously polls or subscribes to the Data Management component to retrieve new log entries and highlight any alerts. Menu or dashboard UI elements (tables, notification banners) can be built using common JS frameworks (React, Vue, or plain HTML/JS) without custom drawing logic.

In implementation, UI classes follow a Model-View-Controller (MVC) pattern: the **Model** for the admin UI is the data store (credential and log data), the **View** is the HTML/CSS display in the browser, and the **Controller** is JS code handling events (button clicks, periodic refresh). This separates presentation from business logic. For student interaction, the ScannerUI is essentially a controller/view combo that subscribes to scan events and updates the display immediately. Usage scenarios (student scans card, or admin reviews logs) guide the screen flows, as recommended for UI design

**Interfaces and Displays**

* **ScannerUI** – Prompts card scan, displays access result.
* **StatusDisplay** – Shows live access attempt results.
* **AdminDashboard** – Shows logs and alerts for admin monitoring.

**UML Diagram (HIC)**

**Diagram Relationships:**

* **ScannerUI** receives results from **AuthenticationManager** (PDC).
* **StatusDisplay** and **AdminDashboard** are updated with data from **LogStore** and **AlertStore** (DMC).

+---------------------+

| ScannerUI |

|---------------------|

| +promptScan() |

| +displayResult() |

+---------------------+

|

v

+---------------------+

| StatusDisplay |

|---------------------|

| +updateStatus() |

+---------------------+

|

v

+---------------------+

| AdminDashboard |

|---------------------|

| +viewLogs() |

| +viewAlerts() |

+---------------------+

1. **Data Management Component (DMC)**

The DMC provides persistent storage and retrieval of credentials and access logs. We use a layered data model to isolate storage details (e.g. files or databases) from the domain logic[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20Data%20Management%20Component). Key classes/modules include:

* **Database** – A storage layer (e.g. JSON files, local SQLite DB, or IndexedDB) that holds tables for *Users* (student credentials) and *AccessLogs*. This is the “Model” in MVC. For example, a SQL table Students(studentID, name, cardID) and Logs(timestamp, studentID, doorID, status).
* **CredentialStore (UserCredentialManager)** – A JS class providing methods like findStudentByCard(cardID), addStudent(student), updateStudent(), etc. It encapsulates database CRUD operations on the Users table
* **LogStore (AccessLogManager)** – A JS class with methods recordAccess(logEntry) and queryLogs(filter). It appends new entries to the AccessLogs table and can retrieve logs for reporting or monitoring.
* **AlertStore** – (Optional) A table or in-memory list of active alerts, with methods like addAlert(alert) and getActiveAlerts().

The system initializes with an offline list of valid student cards (loaded into the CredentialStore) since no external university DB is integrated. Whenever AuthenticationManager needs to verify a card, it calls CredentialStore.findStudentByCard(). When a scan is processed, AuthenticationManager calls LogStore.recordAccess(...) to save the event[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20Data%20Management%20Component). This layered design (separating data structure from higher-level objects) follows best practices in OOD[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20Data%20Management%20Component). In effect, the DMC acts as the persistent model layer: the database (e.g. MySQL, SQLite, or even a JSON file) provides a common data store for all components, and the JS data-manager classes interact with it.

**Storage Management**

* **CredentialStore** – Manages student and RFID credentials.
* **LogStore** – Manages logs of access attempts.
* **AlertStore** – Tracks active alerts.

**UML Diagram (DMC)**

The TMC coordinates real-time processes (RFID scans, monitoring, alerting). In a JavaScript implementation this is event-driven rather than multi-threaded, but we design conceptual tasks/objects to manage concurrency[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20Task%20Management%20Component). Key tasks/services include:

* **ScanTask (AuthenticationTask)** – Triggered by the Reader when a card is scanned. This high-priority task calls AuthenticationManager.verifyCard(). It runs immediately upon each scan event, ensuring low latency. According to [32], high-priority tasks (like access control) must operate even under load[openmeans.com](https://openmeans.com/articles/education/22-computer-science/8776-object-oriented-analysis-and-design.html#:~:text=The%20Task%20Management%20Component). The ScanTask receives the raw cardID, sends it to AuthManager, and then invokes the StatusDisplay to show the result.
*  **LoggingTask** – Runs asynchronously after the scan is processed. Once AuthenticationManager decides grant/deny, it invokes LogStore.recordAccess(...). Because writing to storage can be slower, this is done without delaying the ScanTask response to the user. The LoggingTask ensures data is committed to DMC.
*  **AlertTask** – Continuously monitors access patterns. For example, it can subscribe to new log entries or use a timer to check for three successive denials by the same student, then create an alert. This task publishes alerts to the HIC (AdminDashboard notifications) and may trigger external alarms. In JS, this could be implemented with callbacks or an observer: whenever LogStore.recordAccess() is called, it emits an event that AlertManager listens to, consistent with an event-driven design.
* **Scheduler/Coordinator** – A lightweight controller (could be part of AuthenticationManager) that handles event registration. In Node.js or browser JS, this is naturally the event loop: scan events invoke callbacks, while setTimeout or setInterval could simulate periodic monitoring if needed. The TMC ensures tasks run in logical order: for instance, ScanTask → LoggingTask → AlertTask.

Sequence/message interactions: when a student scans an RFID card, the Reader object emits an event onCardScanned(cardID). The ScanTask (or Reader event handler) catches this and calls AuthenticationManager.verifyCard(cardID). The AuthenticationManager in turn calls CredentialStore.findStudentByCard(cardID) and decides access. It then calls LogStore.recordAccess({timestamp, studentID, doorID, status}). Once logged, an event accessLogged(logEntry) is emitted, which AlertManager subscribes to. If logEntry.status indicates a failure or unusual pattern, AlertManager’s checkAlerts(logEntry) may generate an alert message. Finally, ScannerUI is updated to reflect the decision (via a message like ScannerUI.displayStatus(granted/denied)). In this way the components communicate via method calls and events (messages) to fulfill each access attempt.

By layering components and defining clear class responsibilities (as above), the design meets the requirements. The PDC classes model the real-world entities, the HIC provides student and admin interfaces, the DMC handles all storage, and the TMC orchestrates real-time behaviour. This OOD design (using JavaScript classes/modules) ensures robustness and maintainability

**Diagram Relationships:**

* **CredentialStore** is queried by **AuthenticationManager** (PDC).
* **LogStore** records events from **AuthenticationManager** and sends notifications to **AlertManager** (TMC).
* **AlertStore** feeds data into **AdminDashboard** (HIC).

+-----------------------+

| CredentialStore |

|-----------------------|

| +findStudentByCard() |

| +addStudent() |

+-----------------------+

+-----------------------+

| LogStore |

|-----------------------|

| +recordAccess() |

| +queryLogs() |

+-----------------------+

+-----------------------+

| AlertStore |

|-----------------------|

| +addAlert() |

| +getActiveAlerts() |

+-----------------------+

**4. Task Management Component (TMC)**

**Event-Driven Services**

* **ScanTask** – Triggered on RFID scan.
* **LoggingTask** – Records the event post-authentication.
* **AlertTask** – Monitors logs for abnormal patterns.

**UML Sequence Diagram (Scan Event)**

**Diagram Relationships:**

* Demonstrates how all four components (PDC, HIC, DMC, TMC) interact during a scan.
* Connects: Student & RFIDCard (PDC) → AccessPoint & AuthenticationManager (PDC) → CredentialStore & LogStore (DMC) → AlertManager & AlertStore (TMC) → ScannerUI (HIC).

Student --> RFIDCard : scans()

RFIDCard --> AccessPoint : sendCardID()

AccessPoint --> AuthenticationManager : verifyCard()

AuthenticationManager --> CredentialStore : findStudentByCard()

AuthenticationManager --> LogStore : recordAccess()

LogStore --> AlertManager : notify(logEntry)

AlertManager --> AlertStore : addAlert() (if triggered)

AuthenticationManager --> ScannerUI : displayResult()

**Conclusion**

This OOD model for a campus security authentication system leverages clear object boundaries and well-defined interactions between components. By organizing the system with the four OOD components, we ensure scalability, modularity, and support for real-time operations using RFID and JavaScript-based implementations.