ChatGPT

System Requirements for Military Flight School Scheduling & Training Platform

Introduction

This document details the **system requirements** for a browser-based aviation scheduling and training platform tailored to military flight schools. The platform is organized into distinct modules, each addressing a key aspect of scheduling and training management. It is designed as a **web-based system** accessible via modern browsers on both PC and mobile devices (responsive design) [1] . Emphasis is placed on robust security (including biometric integration), modular architecture, and scalability to support large user bases and data volumes. The table below summarizes the core modules and their primary functions:

Module	Description
Scheduling	Plan, adjust, and publish flight and training schedules (calendar views, conflict checking, etc.).
Crew Management	Manage personnel data, qualifications, roles, and availability for training and missions.
Authorization	Mission approval workflow, including biometric identity verification for secure sign-offs.
Training Records	Record and track training syllabi, performance evaluations, and qualification progress for crew.
Notifications	Send alerts and notifications for schedule changes, training due dates, approvals, and other events.
Reporting	Generate reports and analytics (standard and custom) on scheduling, training progress, resource usage, etc.
Resource Management	Track and manage training resources (aircraft, simulators, classrooms), including availability and maintenance status.
System Administration	Configure system settings, manage user accounts/roles, and ensure security, audit, and integration with external systems.

Each module is broken down by **User Requirements** (functional and UI expectations), **Non-Functional Requirements** (performance, reliability, usability, etc.), and **Domain-Specific Requirements** (unique to military aviation training). The following sections provide an exhaustive list of requirements per module, along with development considerations for security, modularity, and scalability.

Scheduling Module

Overview: The Scheduling module handles the creation and management of flight schedules (and related training event schedules) for the flight school. It provides an intuitive calendar interface for

planners to assign missions, training flights, and simulators, while automatically enforcing operational rules and resource constraints. Users (schedulers, instructors, students) can view schedules in real-time and receive updates when changes occur.

User Requirements (Scheduling)

- **Intuitive Schedule Interface:** Provide a user-friendly scheduling interface (e.g. drag-and-drop calendar) to build daily, monthly, and annual schedules [2]. Users should be able to view schedules in multiple formats (e.g. master calendar, by location/unit, or timeline grid) with changes in one view reflected across all views [2].
- **Create & Modify Flights:** Allow schedulers to create new flight entries (missions or training sorties) and edit details such as date, time, route/leg, duration, and frequency of recurring events [3]. It should be easy to adjust flight details or reroute as needed to accommodate changes [4].
- **Real-Time Updates:** Ensure that all users (students, instructors, crew) can access the latest schedules in real-time from any device. When a schedule is updated (e.g. due to weather or maintenance issues), the changes should be instantly visible to all relevant personnel [5] [6].
- **Conflict Detection & Alerts:** Automatically detect scheduling conflicts or rule violations. For example, if two flights are scheduled for the same aircraft or a pilot is double-booked, the system should flag the conflict. Users should receive immediate alerts or warnings about conflicts and discrepancies [7] [8], allowing them to resolve issues before finalizing the schedule.
- **Rule-Based Scheduling:** Incorporate an engine that applies **scheduling rules** (crew rest requirements, maximum flight hours, training prerequisites, etc.) while building the schedule. The interface should visually highlight or prevent entries that violate these rules (e.g. if a pilot is scheduled beyond duty-hour limits or lacks a qualification for a mission) [9] [10]. Schedulers can define or adjust these rules via the UI as needed (within admin permissions).
- **Schedule Adjustment & Flexibility:** Enable schedulers to swiftly adjust the schedule for unforeseen changes [6]. Users should be able to cancel or delay missions, swap aircraft or instructors, and reassign crew on short notice (e.g. due to flight delays, weather, or mechanical issues) [6] [11]. The UI should make it easy to drag or edit entries and to notify affected users of changes (tie-in with Notifications module).
- **Visibility & Filtered Views:** Provide filter options and different views for users to focus on relevant information. For example, an instructor might view only their own sorties, a student sees their training lineup, and a scheduler sees a **global view of all assignments** [5]. Support a shift calendar or timeline view to show who is on duty, open slots, and crew availability [5]. This improves situational awareness for planning.

Non-Functional Requirements (Scheduling)

- **Performance & Scalability:** The scheduling module must handle a high volume of entries (potentially hundreds of flights and events per day) without performance degradation. Loading the schedule or applying filters should be fast (e.g. under 2 seconds for daily view). The system should scale to support multiple squadrons or training groups simultaneously – requiring efficient queries and possibly caching of schedule data. If deployed enterprise-wide, the architecture (e.g. microservices or cloud-based scaling) must allow additional servers/instances to handle increased load (horizontal scalability).
- **Reliability & Concurrency:** Ensure that updates to the schedule are handled reliably, especially if multiple schedulers or administrators edit concurrently. Implement record-locking or optimistic concurrency to prevent conflicts (e.g. two users editing the same flight). Changes should never result in a corrupt or partially saved schedule – use transactions to commit or roll

back changes atomically. The schedule data should be persistent and backed up regularly, as it's mission-critical.

- **Usability:** Use modern web design (HTML5/CSS3, responsive frameworks) for a **responsive interface** that works on large desktop screens as well as tablets in the field [1]. The drag-and-drop and interactive elements should degrade gracefully on touch devices (allow touch interactions for drag, or alternative input). Provide tool-tips, legends (for symbols or color codes for different mission types), and quick navigation (e.g. jump to date, search for flights by callsign) to enhance user experience. The interface should be consistent and intuitive so that schedulers and instructors can learn it quickly (minimal training).
- **Real-Time Sync:** The module should support real-time data sync (via web sockets or frequent polling) so that if one user updates the schedule, others see updates immediately without needing a page refresh. This ensures everyone operates on the current plan, critical for fast-changing training environments.
- **Integration & Modularity:** The Scheduling module must integrate with other modules (Crew, Resources, etc.) through well-defined APIs or services. For example, when building a schedule, it should pull data on crew qualifications and resource availability from the respective modules. The design should be modular so that the scheduling engine can be updated or replaced (e.g. to improve the auto-scheduling algorithm) without breaking other components. Loose coupling and adherence to a service-oriented architecture (or microservices) are expected to achieve this modularity.
- **Security (Scheduling Data):** Enforce access controls so that only authorized users can modify schedules (e.g. only scheduling officers or administrators can create/edit flights, instructors can only view or make limited changes for their sessions, students have read-only access to their schedule). All schedule data transmissions should be encrypted (TLS) and sensitive details (like mission objectives or classified locations) must be protected from unauthorized access. Audit logging should record any changes made to the schedule (who changed what and when) for accountability.

Domain-Specific Requirements (Scheduling)

- **Compliance with Flight Rules:** The scheduling system must comply with military flight operations rules and training syllabi constraints. For example, it should enforce crew rest and flight duty period limits as outlined in regulations (e.g. USAF's AFI 11-202 Vol 3 crew rest rules, or Navy NATOPS equivalents) [12]. If a scheduling action violates these rules, the system should prevent it or require a waiver from an authorized role. Custom military-specific rules (such as limiting student sorties per day, weather minimums for trainees, etc.) should be configurable and enforced automatically [9].
- **Seniority and Qualifications:** Incorporate military-specific considerations like crew seniority and qualification levels into scheduling. For instance, when assigning instructors or evaluators to flights, the system might prefer certain pairings (e.g. an instructor pilot must have an IP qualification and possibly a minimum rank for certain missions). The rules engine should support these domain constraints (e.g. student solo flights must be supervised by an instructor on the schedule, or certain missions require an instructor with examiner status).
- **Duty Hour Monitoring:** Provide a **duty hour monitor** or similar mechanism to track each aircrew's cumulative duty hours and flight hours per day/week/month [13]. This feature increases transparency of crew workload and ensures compliance with both safety standards and **currency requirements**. For example, it can show how many hours a pilot has remaining this week before hitting a limit [14]. If a proposed schedule would exceed a pilot's allowed hours, the system flags it.
- **Mission Briefing Integration:** The schedule entry for each mission should allow attaching mission briefs or profiles (e.g. sortie profile, training objectives). In a military context, each flight

often has a mission briefing document – the system could store or link to this, accessible to the crew. Additionally, if a mission requires a risk assessment or pre-mission checklist, the scheduling module should integrate with those forms (possibly via the Training Records or Authorization modules).

- **Operational Readiness Links:** The scheduling should connect with operational requirements – for example, if a certain training sortie is critical for a student's progression or a squadron's **mission-ready status**, the system might prioritize those in auto-scheduling (if auto-schedule algorithms are used). Also, cancelled missions might need to be flagged for reschedule to meet syllabus timelines. In essence, the scheduler should be aware of training throughput goals and include tools to ensure students meet their required flights on time.
- **Multi-Unit Coordination:** Military training often involves multiple units or squadrons (e.g. shared aircraft or joint training ops). The system should allow scheduling across units in a coordinated way. This might mean the schedule can be filtered or partitioned by unit, but still handle shared resources (e.g. a shared simulator building). It should also accommodate deployments or detachments (if a training flight goes to another base, schedules should reflect local times and resources accordingly).
- **Biometric Sign-Off Placeholder:** (Note: The actual biometric integration is covered under Authorization module below, but from a scheduling perspective, there might be a domain requirement that *prior to execution*, each sortie on the schedule must be authorized by a commander via biometric sign-off. The scheduling UI could indicate which missions are authorized vs pending authorization.)

Crew Management Module

Overview: The Crew Management module handles information about all personnel (student pilots, instructor pilots, maintenance crew if needed, etc.). It serves as a repository of **personnel data**, including contact information, rank, role, qualifications, certifications, experience hours, and availability. It closely interacts with Scheduling (to ensure only qualified and available crew are scheduled) and Training Records (to update qualifications).

User Requirements (Crew Management)

- **Personnel Profiles:** Provide a detailed profile for each crew member (student or instructor). Users with access (instructors, admins) should be able to view and update personal details like rank, service ID, contact info, and assigned unit. Crucially, profiles include each person's **qualifications and certifications** (e.g. aircraft type ratings, instructor qualifications, medical clearance status, ejection seat training, etc.) and their currency (when each qualification was last used or is due to expire) [9] [15].
- **Qualifications & Currency Tracking:** Allow training officers or administrators to input and update qualifications and automatically track currency/expiration. The system should clearly show if a crew member is **qualified** for a certain activity – for example, marking whether an instructor is current to fly at night or if a student has passed the prerequisites for solo flight. If a qualification expires or a required checkride is overdue, the profile should flag this (and the Notifications module should send alerts).
- **Availability & Scheduling Integration:** Enable crew members (or schedulers) to set availability statuses – e.g. marking when an instructor is on leave, or a student is unavailable due to other duties. The Scheduling module should respect these availability flags (e.g. do not schedule a person who is on leave or in training elsewhere). A crew calendar might show each person's assignments and free times. This could include a **leave/request system** where crew can request time off or swapping of assignments (with appropriate approval workflow).

- **Role-Based Views:** Adapt the UI based on user roles. For example, an **Instructor** can view the roster of students they oversee, including each student's current training progress and upcoming events. A **Student** can view their own profile and perhaps limited info like their grades or remaining syllabus requirements. A **Scheduler** or **Training Officer** can browse and search the entire roster, filter by qualifications (e.g. find all available IPs who can fly a certain aircraft), and assign crew to missions from within the scheduling interface [16].
- **Timesheets & Flight Hour Logs:** The system should maintain each pilot's accumulated flight hours (potentially by category: day/night, instrument, by aircraft type, etc.) and duty hours. Instructors or crew can view their log summary which gets updated as training flights are completed (this could be automatically updated from the Training Records module once flights are marked done). This helps in monitoring experience levels and is useful for qualifications (e.g. hitting a flight hour milestone). The crew profile could include a **timesheet** or flight log component [17].
- **Crew Documents:** Allow storage or linking of important documents per crew member. For instance, scans of certifications, medical exam forms, or orders assigning them to the training program. Users (with permissions) can upload and manage these documents in the profile. This ensures all necessary paperwork is centrally available (no lost physical records).

Non-Functional Requirements (Crew Management)

- **Data Consistency & Integrity:** The crew data is foundational – ensure strict data validation and integrity. For example, rank must be selected from valid values, dates (like qualification dates) must make sense, and certain fields should be mandatory (e.g. each crew must have an assigned squadron). The system should maintain historical records (when a qualification was earned or a role changed) to have an audit trail of a person's training history [15].
- **Performance:** Queries on crew data (such as filtering by qualification or searching names) should be optimized for quick results, even if the database contains thousands of personnel records. Indexing and efficient query design are required, especially since the Scheduling module will frequently query this data to verify qualifications and availability.
- **Security & Privacy:** Enforce role-based access control on personnel records. Personal Identifiable Information (PII) like contact and medical data must only be visible to authorized users (e.g. an instructor can see their students' records, but students cannot see others' details). Sensitive data should be encrypted at rest. All access to crew records should be logged (for security auditing). The system should also comply with any military privacy regulations regarding personnel data.
- **Usability:** The interface for crew management should allow quick updates and overviews. For instance, admin users might want to perform bulk actions (like marking a whole class of students as having completed a certain ground training event). A user-friendly filtering and bulk-edit capability can be provided. Additionally, the profile UI should be well-organized with sections/tabs (e.g. Personal Info, Qualifications, Training Progress, Documents) for clarity. This module should also be mobile-friendly so that, for example, an instructor on the flight line could quickly pull up a student's qualifications on a tablet or phone if needed.
- **Integration:** The Crew module must feed into other modules seamlessly. It should expose services or APIs that allow the Scheduling module to check a person's qualifications, or the Training Records module to update a person's record when they complete a training event. Modularity is important: this module could be swapped or updated (e.g. to integrate with an official military personnel system) with minimal changes to others, as long as it adheres to the agreed data interface.
- **Scalability:** As the platform may be used by multiple training squadrons or even across an entire training wing, the crew database could grow and see concurrent access from many users. The system should handle concurrent updates (e.g. two admins editing different people's profiles

simultaneously) without issues. It should also be scalable to handle additional units – potentially partitioning data by unit or providing multi-tenant support if the same instance serves different academies.

Domain-Specific Requirements (Crew Management)

- **Rank and Military Structure:** Include military-specific fields in profiles, such as rank, branch of service, and duty position (e.g. "Squadron Training Officer" or "Student Pilot Class 22-15"). The system should understand hierarchy and possibly use rank in certain logic (for example, only a senior officer or designated role can be the authorizer in the Authorization module). Display of names should often include rank (e.g. "Capt John Doe") as is standard in military environments.
- **Qualification Standards:** The system must accommodate the specific qualifications relevant to military aviation training. This includes tracking things like **Mission Qualification Training (MQT)** status, **Combat Mission Ready** status (if applicable for advanced training units), water survival training currency, physiological training (altitude chamber), and more. These are domain-specific requirements that a civilian system might not track. The system should allow administrators to define and update these qualification types and their valid durations or criteria.
- **Training Pipeline Integration:** A military flight school often has a set curriculum pipeline (e.g. phases of training). The Crew Management module should integrate with the Training Records module to reflect where each student is in the pipeline (e.g. "Phase 2 – Instruments" or "Awaiting Initial Flight Training"). This context is domain-specific and can be used to filter or group students (for instance, to easily see all students in a given phase, or all who are behind schedule).
- **Special Clearances and Medical Status:** Military flight training might involve tracking if a student has special clearances (e.g. security clearance level if needed for certain training missions or ranges) or current **medical status** (a pilot on DNIF – Duties Not Including Flying – should be marked as not available). The system should have fields to track if a pilot is medically cleared to fly or any temporary disqualifications. Only authorized staff can edit these fields, and scheduling logic should respect them (e.g. DNIF pilots cannot be scheduled).
- **Interfacing with Personnel Systems:** In a military environment, there may be existing personnel databases. A domain requirement might be to interface or **import data from official personnel systems** to avoid duplicate data entry. For instance, pulling basic info and rank from a central system via a secure feed. This ensures the platform stays in sync with official records.
- **Audit and Accountability:** Military operations require strict accountability. The Crew Management module should maintain an audit log of all changes to qualifications and availability (who made the change and when). This is important in case of investigations (e.g. if someone was scheduled without a proper qual, there needs to be a record of how the data was entered or missed). Additionally, when a person's status changes (e.g. a new qual earned), the system might require an electronic signature from a supervisor or examiner confirming it – reflecting the formal sign-off process in training.

Authorization Module

Overview: The Authorization module governs the **mission approval process**. In military flight training, certain personnel (e.g. a Squadron Supervisor or Operations Officer) must authorize each flight or training mission before it is executed. This module provides a workflow for approving or rejecting missions, with strong identity verification. A key feature is integration with a **biometric scanner** for mission authorization – meaning an approver (such as the commanding officer) must authenticate with a biometric credential (e.g. fingerprint or iris scan) to digitally sign off on a mission. This enhances security and accountability by ensuring the approval is given by the correct individual and is logged.

User Requirements (Authorization)

- **Mission Approval Workflow:** For each scheduled mission (from the Scheduling module), allow it to be submitted for authorization. Users with the appropriate role (e.g. Operations Supervisor, Instructor Pilot acting as supervisor for a student solo) should see a list of pending missions requiring approval. They can review details of the mission (crew, aircraft, weather, risk assessment, etc.) and either approve or deny the mission from the interface. The UI should make it clear which missions are authorized and which are pending.
- **Biometric Authentication for Approvers:** Integrate biometric scanning in the approval step. When an authorized approver clicks "Authorize" for a mission, the system should prompt for biometric verification (such as a fingerprint scan or a facial recognition via device camera, depending on hardware). The system must interface with a biometric scanner device connected to the user's computer or a mobile device's biometric capability. The scanned biometric data is used to verify the approver's identity before finalizing the approval. This provides a secure "digital signature" equivalent. **Biometric technologies are integral to modern defense operations**, offering secure and fast identity assurance for critical activities [18] . By requiring a biometric sign-off, the platform protects the mission authorization process with high confidence in the approver's identity [19] .
- **Digital Signature and Logging:** Once approved, the mission should be marked with an electronic signature (including approver's name, timestamp, and method of auth). This record should be visible in the schedule (e.g. a status icon "Approved" vs "Pending Approval") and in mission details. Users should be able to click to view who authorized it and when. All approvals/denials should be logged for later audit. If a mission is denied, the system should allow the approver to input a reason which is fed back to the scheduler or crew.
- **Pre-Mission Checklist Integration:** Often, authorizing a mission involves verifying certain prerequisites (weather check, risk assessment, briefing completed). The module should allow attachment or completion of a **digital checklist or risk assessment form** as part of the approval workflow. For example, an approver might have to confirm that the flight risk assessment was reviewed. If integrated with the Training Records module, these forms could be electronic and automatically included. The UI should present any required forms or data to the approver for review (e.g. a student's risk assessment score, or a link to the weather briefing).
- **Notifications of Authorization Status:** Users (especially the pilots/instructors on the mission) should receive notification when their mission is authorized or if it's been rejected (with reason). This can be via the Notifications module – e.g. an email or app notification saying "Mission X for 15:00 is authorized by Maj Smith" or "Mission Y was not authorized: weather below minima". In the UI, missions could highlight in green for authorized, red for rejected, etc., for quick reference.
- **Override / Emergency Mechanism:** Provide a mechanism for higher authorities or special scenarios to override the standard process if needed (for example, if the biometric device is down or in an emergency scramble, a mission might need quick authorization by a code or alternate method). This should be tracked and limited to certain roles, and later followed up (to avoid misuse). The UI for this could be a special "Emergency Authorize" button that requires a second confirmation or a one-time passcode from a commander.

Non-Functional Requirements (Authorization)

- **Security:** This module deals with critical security controls. It must enforce **strong authentication** – not only using biometrics but also ensuring the underlying system is secure. Biometric data should never be stored in raw form; if storing templates for verification, they must be encrypted and protected. Communication between the client device (browser) and the server during a biometric scan must be encrypted and safeguarded against interception. The module should likely comply with government security standards for handling biometric data

and identity verification. Regular security audits and certification (perhaps meeting DoD or NIST standards) will be required.

- **Reliability:** The authorization process should have near-zero downtime tolerance around mission launch times. Approvers often authorize missions shortly before takeoff; any downtime could delay training. The system should be available and responsive, especially during peak periods (e.g. early morning when many missions are authorized). If the biometric subsystem fails (hardware issue), the system should degrade gracefully (perhaps allow a CAC/PIN login or fallback method so training isn't halted).
- **Performance:** Biometric authentication should be quick – e.g. fingerprint scanning and verification should occur within a couple of seconds. The overall authorization transaction (loading a mission package for review and signing it) should be efficient, as any sluggishness can cause frustration for busy officers. The module's backend should handle many concurrent approval processes (if multiple approvers are working on different missions) without slowing down.
- **Audit and Accountability:** Non-functionally, the system must maintain an **immutable audit log** for all authorizations. This includes recording the biometric verification result, the approver's ID, timestamp, and mission ID. These logs should be tamper-proof (or tamper-evident) and retained for a long period (per military records retention policies). This is crucial for later investigations or just to have a clear chain-of-command record.
- **Compatibility:** The biometric integration should be flexible to different devices. On desktop, it may need to support common USB fingerprint scanners (the system should be tested with approved models). On mobile, it should leverage the device's built-in biometric (fingerprint sensor or face ID) via WebAuthn or similar modern web authentication standards if possible. The system should use standard APIs for biometrics to remain compatible with future hardware.
- **Modularity:** The Authorization module should be somewhat self-contained. For instance, if down the line the military wants to use a different method of authorization (say a new smart card system), the module should allow swapping out the biometric verification component without overhauling the entire platform. A clear interface or service for "verify identity of user X by method Y" would make it more modular. Also, if integrating with enterprise identity systems (like DoD Common Access Card), the design should accommodate that either as an alternative or supplement to biometrics.

Domain-Specific Requirements (Authorization)

- **Chain-of-Command Rules:** The system must implement the specific approval chain required by the military training environment. For example, student solo flights might require authorization by the Squadron Supervisor on duty, whereas routine dual-instructor training flights might be auto-authorized or authorized by a flight commander. The rules about *who* can authorize *what* should be configurable to match military protocols. Only users with designated roles (e.g. Operations Officer, Supervisor of Flying) can authorize missions, and the module should enforce those permissions strictly.
- **Integration with Flight Authorization Documents:** In many military units, a **Flight Authorization (FA)** or similar paper form is used, listing all flights and signatures of approval. The system should be able to generate an equivalent **digital Flight Authorization** report that lists all missions for the day and their authorization status, which can be archived or printed if needed (for legal flying orders). This might be a special report in the Reporting module, but it's driven by the Authorization data. It should meet whatever format the unit requires (which could be a domain-specific format, possibly even requiring an authorized signature on a hard copy in some cases).
- **Risk Management Integration:** Military flights often require a risk assessment (e.g. using a standardized risk management worksheet). The Authorization module should incorporate the

results of that risk assessment. For instance, if a mission is assessed as “high risk”, the system could require a higher-level approver’s biometric sign-off (like the Commanding Officer). This dynamic requirement – adjusting who must authorize based on risk level – should be supported. It’s a domain-specific logic reflecting safety protocols.

- **Biometric Data Handling Compliance:** The use of biometrics in a military context likely triggers compliance requirements (for example, compliance with DoD Biometrics standards or identity management policies). Domain-specific requirement: the system must align with the DoD’s biometric data handling guidelines, ensure any biometric info is stored in accordance with defense standards, and possibly integrate with existing defense biometric databases if applicable (for identity verification). For instance, the biometric scanner might need to be FBI-certified hardware [20] and use approved algorithms for fingerprint matching.
- **Multiple Authorization Levels:** There might be scenarios where multi-layer authorization is needed (e.g. a high-risk mission requires two officers to approve). The system should support **multi-factor or multi-person authorization** for such cases. This is specific to the domain where certain training missions or events (like weapons training flights or night solo flights for students) might need two sets of eyes. The module could require two distinct biometric sign-offs for those, and only mark the mission as fully authorized when both are completed.
- **Logging for Training Credits:** In a training context, the authorization can also serve as a trigger that a mission was officially flown under orders. The domain may require that only authorized missions count toward training requirements. Therefore, the Training Records module should only credit a sortie to a student’s record if it was properly authorized in this module. This interlock ensures no “unauthorized” training event slips through unrecorded.

Training Records Module

Overview: The Training Records module is responsible for tracking all training-related data for each student (and possibly instructors’ qualifications). It maintains the digital **gradebook** for the flight training program – logging every flight, simulator, and academic training event, the performance of the student, and the sign-offs by instructors. It also manages the training curriculum or syllabus definition, ensuring that each student progresses through required modules and that any changes to the curriculum are version-controlled. This module essentially replaces or augments the traditional paper training folders with a robust electronic system.

User Requirements (Training Records)

- **Electronic Grade Forms:** Provide electronic forms for instructors to grade or evaluate each training event (flight or simulator or academic) [21] . For example, after a flight, an instructor can fill out an evaluation form on a tablet/PC, rating the student’s performance on various maneuvers and tasks. These forms should be customizable per the curriculum (different forms for different phases or aircraft) and include sections for both grades and comments. The instructor should be able to sign the form digitally, and the student can also sign or acknowledge if required (possibly using CAC or a stylus signature).
- **Curriculum Management:** Allow admin-level users to define and manage the **training curriculum** (syllabus). The system should support setting up a hierarchy of training modules, lessons, and objectives (for example: Phase II has a series of flight lessons, each with specific maneuvers to be taught/graded). If the curriculum is updated (e.g. objectives change or a new block is added), the system should maintain version control – past records tied to old curriculum are kept, and new changes apply to future training [22] [23] . Updates to curriculum should automatically propagate to relevant forms or scheduling templates, avoiding duplicate data entry [22] [23] .

- **Training Event Logging:** For each student, the module logs each completed training event, along with the outcome (grade, pass/fail, etc.). Users (students and instructors) should be able to pull up a **training record report** for a student, showing what has been completed and what remains in the syllabus. The interface for instructors should make it easy to enter results for multiple students if needed (e.g. after a simulator session with several students).
- **Qualifications and Certifications:** When a student completes certain milestones (like first solo, instrument check, final checkride), the system should update their qualifications in the Crew Management module automatically. The training module should have business logic to mark when a student is "qualified" in a skill or ready for a certificate. It should also track academic requirements and automatically note when a student has completed all prerequisites for an advancement.
- **Performance Dashboard:** Provide a dashboard or overview for instructors and training officers to monitor student progress. This may include visual indicators (like a progress bar through the syllabus, or flags for students who are falling behind schedule or struggling with certain maneuvers). Instructors can quickly see the **proficiency trends** of a student – e.g. if a student has failed the last two flights in formation, that should be visible so they can adjust training. Similarly, students should have access to their own progress view, so they know where they stand and what's upcoming.
- **Archive and Retrieval:** The system should allow retrieval of historical training records quickly. If a student from a previous class needs their records or an instructor wants to review how a certain lesson was graded last year, the module should provide search and filter tools (by name, date, lesson type, etc.) to retrieve past records. All records should be **read-only archived** once a student graduates, but remain accessible for reference (with proper permissions).

Non-Functional Requirements (Training Records)

- **Data Integrity & Consistency:** Training records are official documents in a sense; thus, once an event's record is finalized (signed by instructor, and possibly acknowledged by student), it should be locked from casual editing. Any changes after finalization should require special permission or an addendum process (to maintain integrity). The system should ensure no records are lost and use strong consistency (transactions) when saving forms. If connectivity is an issue (say, an instructor is offline when filling a form), the system should queue or save locally and sync when possible to prevent data loss.
- **Offline Capability:** As training flights might occur in areas without network (or instructors may prefer to fill grading on the jet or in remote sims), the system should support offline data entry for training forms. For instance, a Progressive Web App or mobile app mode can allow an instructor to download the day's schedule and relevant grading forms, then later upload the results when back online [24] . This ensures flexibility and no dependency on constant connectivity during flights.
- **Performance:** When dealing with large datasets (years of records, or many students), the module should still perform well. Generating a student's training report or calculating class averages should happen in seconds. The database should be indexed by student, class, date, etc., to allow swift queries. Also, the system might need to produce complex queries (like "show all students who have not completed Lesson X") – those should be optimized or possibly run in the background if heavy.
- **Security & Access Control:** Training records contain evaluations and possibly sensitive observations about performance. Access must be restricted: e.g. a student can only see their own records (and perhaps a summary of their peers' progress but not the detailed grades), an instructor can see records of students they train, and a senior training officer can see everything. The module should authenticate and authorize each access. Additionally, because these records could be used in decisions (like eliminiation from training), they must be tamper-proof. All

changes or entries should be attributable to a specific user account with timestamp. Consider using digital signatures so that a record can be proven to be signed by a certain instructor.

- **Scalability & Storage:** This module will accumulate a *large* amount of data over time (daily evaluations for every student). The system must be scalable in terms of storage (potentially using cloud storage or partitioning by class year). Archival strategies might be needed (e.g. archive older classes' detailed data to a different storage or to static files, while retaining summary info in the main DB). The design should ensure that scaling out (more storage, or even distributing the module's database) can be done without affecting the rest of the system.
- **Integration:** The Training Records module should integrate with external analysis or **learning management systems** if required. Non-functionally, an API could be provided for exporting data for analysis (some military training programs might want to analyze performance data separately using data analysis tools). Internally, it must integrate tightly with Scheduling (to know which training events occurred) and Crew/Qualifications (to update statuses). The module's design should keep these interactions efficient – e.g. posting a grade might automatically update a qualification, which means a cross-module call that should happen quickly and robustly.

Domain-Specific Requirements (Training Records)

- **AQP and ISD Compliance:** The platform should support the **Advanced Qualification Program (AQP)** framework and **Instructional Systems Development (ISD)** methodology where applicable [25]. In military aviation training (especially in advanced or continuation training), AQP principles might be used. This means the system should allow definition of **Job Task Analyses (JTA)**, **Qualification Standards**, and other elements that tie training tasks to higher-level competencies [26]. For example, each training maneuver might be linked to a skill/knowledge/ability code; the system can store these linkages. This is a domain-specific way of structuring training data to ensure a data-driven approach to curriculum effectiveness.
- **Multiple Grading Scales and Criteria:** Military training may use complex grading systems (e.g. Qual/Nonqual, or numeric scores, or descriptive grades). The system must allow multiple grading scales to be defined and used for different lessons [27]. For instance, basic flights might be graded pass/fail, whereas advanced flights have numeric scores. The system should also handle predetermined comment dropdowns (common comments for certain maneuvers) to standardize evaluations [28]. This ensures consistency in how instructors grade and give feedback.
- **Qualification Expiry Alerts:** The system should automatically alert relevant stakeholders when a qualification or certification is due to expire or has lapsed [29]. For example, if instrument rating currency for a pilot is about to expire, the training officers get notified to schedule a refresher. These alerts tie into domain rules (e.g. instrument check must be done every 12 months). The Training Records module is where the last completion dates are recorded, so it should generate these domain-specific notifications (likely feeding the Notifications module).
- **Proficiency Data Analysis:** A domain-specific requirement is to collect and analyze proficiency data to improve training. The platform should enable **data-driven decision making** – for example, track how many repetitions of a maneuver each student needed before mastering it, and provide reports that might highlight if a syllabus change is needed [30] [31]. The system could anonymize and aggregate data to give insights, like "85% of students struggle in Lesson 5 – Formation Join-up, requiring an average of 1.5 extra sorties" – valuable info for commanders. This is facilitated by the records module keeping detailed scores and notes for every training event.
- **Certification of Training Forms:** At the end of training or at key milestones, the system should be able to produce official **"Certificate of Training"** documents or forms [21]. These might be required by the military to formally certify a pilot's completion of the program or a qualification. The requirement is that the system can populate and output these forms (with digital signatures)

so that the training can be formally recognized. This might also include integration with external systems that track pilot training across the military, meaning the module could send a data package or update to a central training registry once a student graduates.

- **Adapting to Military Processes:** The module must **adapt to military-specific processes rather than forcing the military to adapt to the software** [32] . This means it should be highly configurable to mirror exactly how the flight school runs its training. If a particular school has unique steps (like an additional check ride or a different sequence), the system should be able to accommodate that through configuration rather than custom code. Domain experts (admin users) should be able to tweak workflows (e.g. require an extra evaluator sign-off for certain flights) and the system should handle it, reflecting the rigid structure of military training programs but with flexibility to customize to each program's SOPs.

Notifications Module

Overview: The Notifications module sends out alerts and messages to users about important events in the system. Given the dynamic nature of flight training, timely notifications are critical – for example, when a schedule changes at the last minute, or when a student's qualification is about to expire. This module will likely provide both in-app notifications (banner, badges) and external notifications (emails, SMS, or push notifications to mobile) to ensure users stay informed.

User Requirements (Notifications)

- **Real-Time Alerts:** The system shall generate real-time alerts for key events, such as schedule changes, new assignments, or required actions. For instance, if a flight's time is changed or an instructor is re-assigned, all affected users (the crew, the schedulers) should receive an immediate notification in the system and via email/app notification. These alerts should be clearly visible (e.g. highlighted bell icon or pop-up) when the user logs in, and accessible via a notifications panel.
- **Email and Mobile Notifications:** Support configurable email and/or push notifications for users who are off the platform. A user should be able to opt in to receive an email or phone notification when, say, a **schedule is published or updated** for the next day, or when they receive a new message from an instructor. This ensures critical info reaches personnel who may not be constantly logged in. Mobile push notifications (if the platform has a mobile app wrapper or PWA) can alert on devices instantly – useful for urgent changes like a *"Flight tomorrow at 0800 has been cancelled"*.
- **Notification Preferences:** Provide users with control over their notification settings. A pilot might want all schedule changes and training results notifications, but perhaps not every general announcement. The system should allow users to choose which events they get notified about and through what channels (in-app only, email, etc.). However, some notifications might be forced for safety (e.g. an alert about a grounded aircraft affecting their flight).
- **Acknowledgment and Tracking:** For certain critical notifications, require acknowledgment. For example, if an important safety bulletin is issued or a schedule change within short notice, the system could ask the user to confirm they've seen it (e.g. click "Acknowledge" in the app). The module should track acknowledgments so that instructors or schedulers can see who has not yet viewed a critical update.
- **Announcements and Broadcasts:** Apart from automated alerts, the module should allow authorized users (like administrators or commanders) to send manual announcements to groups of users. For instance, a training wing commander could broadcast a message to all students about a policy change. The system should deliver these similarly (in-app and email) and possibly pin them on a dashboard if needed. Users might see a "Notifications/Announcements" section where these messages persist until read.

Non-Functional Requirements (Notifications)

- **Timeliness:** The notifications system must have minimal latency. When a triggering event occurs (schedule change, new record, etc.), the notification should be dispatched within seconds. This might involve using a message queue or real-time push service. The design should ensure that even under heavy load (many events) the notifications do not get backlogged significantly.
- **Reliability:** It is crucial that notifications are delivered reliably. The system should handle retries for email or push in case of failures. If the platform experiences downtime, it should queue notifications and send them when back up. Lost or undelivered notifications could result in missed flights or actions, which is unacceptable. Logging of notification deliveries is important so admins can audit that "yes, the cancellation notice was sent to Lt. Smith at 5:00 AM" for example.
- **Scalability:** The module should scale to potentially thousands of notifications, especially in larger training wings or if the platform is used across multiple bases. It should be built on a scalable service (e.g. using a cloud notification service or an SMTP server that can handle bulk emails). Sudden bursts (e.g. publishing a new day's schedule might trigger hundreds of notifications at once) need to be handled gracefully.
- **Integration:** The Notifications module works across all other modules and therefore should be modular and loosely coupled. Other components should be able to invoke it via an interface (for example, a Scheduling change calls a Notification service API with details). The notifications system should format the message appropriately per channel. It should also integrate with external systems if needed (for example, perhaps with a base-wide alert system or text message gateway if critical).
- **User Experience:** Non-functionally, the notifications should be designed to avoid overload. If a user gets too many trivial alerts, they might start ignoring them. Therefore, the system might implement some intelligent bundling (e.g. combine several minor schedule tweaks into one summary notification if they occur close in time). Additionally, the in-app notification UI should be clean, showing unread vs read, categorizing by type (schedule vs training vs system message) to help users prioritize.
- **Security and Privacy:** Some notifications might contain sensitive information (like details of a mission or personal performance feedback). The module should ensure that such content is only sent over secure channels and only to authorized recipients. For example, an email about a flight schedule should not include classified details. Also, if push notifications are used, consider that they might appear on a phone lock screen – so maybe only generic info in the push ("Schedule updated, please log in for details") to avoid leaking sensitive info.

Domain-Specific Requirements (Notifications)

- **Operational Alerts:** In a military training context, there may be domain-specific alert types. For example, **Weather or Ops alerts** – if the base declares a weather recall or some operational pause, the system should notify all crews immediately. This might interface with base systems (if the base operations center inputs a red flag, the platform broadcasts it). Ensuring all trainees and instructors get urgent operational notices is critical in the domain.
- **Training Alerts:** Send notifications related to training requirements unique to the military. For instance, an alert to a student and their instructor if the student is approaching a **training deadline** (e.g. "Student X has 2 weeks left to complete instrument check or will be eliminated per policy"). Or alerts for upcoming checkrides or evaluations that need special preparation. These ensure the training cadence is maintained and are specific to the structured nature of military syllabi.
- **Chain-of-Command Notifications:** The platform should notify the chain of command for certain events. For example, if a student fails a flight (gets an unsatisfactory grade), an automatic notification could be sent to the Squadron Commander or Flight Commander as per protocol.

This reflects the military practice of supervisory notification for failures or incidents. The requirements should outline which events trigger such escalated notifications (these can often be configured by policy in the system).

- **Integration with Duty Rosters:** If the unit has an **on-call duty** (like Supervisor of Flying schedule, or Duty Officer), the notifications system might need to integrate with that. For example, a notification of a severe issue (like an aircraft emergency in training) should also alert the current supervisor on duty. This might not be a core part of the scheduling system, but as a domain consideration it could interface with how the unit assigns duty personnel and ensure notifications reach them.
- **Acknowledgement and Compliance Tracking:** In a military environment, tracking compliance to notifications can be important. For instance, if a new safety bulletin is released, the system might not only notify everyone but also track who has read and acknowledged it (perhaps requiring them to click a confirmation). The requirement is that the system can produce a report of acknowledgment, fulfilling a domain need for accountability (e.g. "100% of instructors have acknowledged the new emergency procedure update").

Reporting Module

Overview: The Reporting module provides the ability to generate reports, analytics, and data exports from the platform. It collates data from scheduling, training records, resource usage, etc., to produce meaningful insights and official reports. Reports can range from daily flight schedules to individual training reports to high-level readiness metrics for commanders. The module should offer both pre-defined standard reports and flexible custom report queries.

User Requirements (Reporting)

- **Standard Reports:** Offer a suite of standard, commonly-used reports available at a click. Examples: **Daily Flight Schedule** (listing all missions, crew, times for the day – useful for an operations briefing), **Student Training Report** (summarizing a student's entire performance and achievements), **Instructor Currency Report** (listing which instructors are coming due for certain refresher training), and **Resource Utilization** (e.g. how many hours each aircraft flew in a month). These should be formatted clearly, printable, or exportable as PDF for briefing binders or archival.
- **Custom Report Builder:** Provide a user-friendly interface for authorized users (like analysts or training officers) to query the database and generate custom reports. This might be a form where they can select fields (e.g. "List of all flights between X and Y dates that were canceled and reason" or "All students who have not completed Lesson 10"). The interface should allow filtering by various criteria (date ranges, specific instructors, aircraft type, etc.) and the output can be viewed on screen or exported (CSV/Excel) for further analysis.
- **KPI Dashboard:** For key performance indicators (KPIs) like graduation rates, average hours per student, cancellation rates, etc., the module should provide a **dashboard view** with charts and graphs. Users (especially leadership) can view at a glance metrics such as sortie completion rate, student progress distribution, instructor workload, aircraft utilization percentage, etc. This might be an interactive dashboard updated in real-time.
- **Scheduling Analytics:** Include specific analytics for scheduling efficiency – e.g., reports on schedule conflicts over time, or how often reassignments happen due to maintenance, etc. Also, a **forecasting report** could be valuable: based on the current schedule and student pipeline, predict if the unit is on track to meet training output (for example, "projected graduations vs target"). Such reports help in resource planning and are tailored to the training environment.
- **Export & Data Sharing:** Users should be able to easily export reports in various formats (PDF for formal reports, Excel for data analysis, CSV or XML for interfacing with other systems) [33] [34] .

There might also be a need for **integrated data feeds** to other command-level systems [33] – for instance, automatically sending a nightly training summary to a headquarters database. A user with appropriate role should be able to set up scheduled exports or feeds (with security considerations) without needing a developer.

Non-Functional Requirements (Reporting)

- **Performance:** Generating reports, especially complex ones that aggregate a lot of data, should be optimized. The system may need to use pre-computed aggregates or a data warehouse approach for heavy analysis so as not to slow down the transactional part. For example, a report that analyzes years of training data should not lock up the live system – it could run on a replicated database. Aim for most standard reports to generate within a few seconds. For very large data sets, provide progress indicators or asynchronous generation with a notification when ready.
- **Accuracy and Consistency:** Reports must reflect accurate and up-to-date data. If the system is distributed, ensure that data synchronization issues do not cause reports to be generated on stale data. The calculations (like summing flight hours or counting qualifications) should be carefully implemented to avoid errors. It's important because decisions (and possibly official records) will be based on these reports. Consider versioning of reports if the underlying data changes while a report is being viewed (though likely reports are snapshot in time).
- **Security:** Access to certain reports should be restricted. For example, a student should not be able to run a report on other students' performance. The system should enforce report-level permissions – perhaps tie them to the same permissions as underlying data. Also, if reports contain sensitive data (personnel info, or perhaps classified mission details if any), the output must be protected. For instance, if exported to PDF, maybe apply watermarks or access control in the UI to who can see it. In some cases, the system might need to redact or omit classified info in generalized reports.
- **Formatting and Customization:** Reports should be well-formatted for readability since they may be presented in meetings or included in official documents. Non-functionally, this means investing in a good reporting engine that can handle pagination, company (or unit) branding (like logos, headers), and consistent styling. Users might want to customize the layout of certain reports (e.g. include a unit logo or adjust columns). The system should allow some level of template customization, possibly by the admin uploading a template or adjusting options.
- **Scalability:** If many users request reports simultaneously (e.g. end of week everyone pulling training reports), the system should scale or queue these tasks to handle the load. Offloading heavy report generation to background jobs can help. Also, if the platform eventually serves multiple schools, ensure that one school's large report doesn't impact another's performance (multi-tenancy isolation).
- **Audit:** For auditing purposes, track report generation of sensitive data. For example, if someone exports a list of all personnel, that might need to be logged (who ran it and when) because it could contain PII. This is a security non-functional aspect but important in a military context where data access is monitored.

Domain-Specific Requirements (Reporting)

- **Flight Authorization Log (Form):** As mentioned in Authorization, generate a **Flight Authorization form/report** daily that lists all scheduled flights, crew, risk level, and the authorizing official's name (with maybe a line for signature if printed). This report would mirror the traditional flight authorization paperwork in a digital format – a requirement often unique to military units. It should comply with whatever format the unit needs (which might be specified by an Air Force Instruction or Navy manual).

- **Training Progress Reports:** Military training programs often have periodic progress reviews for students. The system should produce a **Standard Progress Report** for each student that instructors and leadership can use in review boards. This might include the student's grades, strengths, weaknesses, and number of extra training rides if any. It should be formatted in a way that aligns with how training review boards expect information (possibly a standardized form in the training command).
- **Readiness & Resource Reports:** Because military training ultimately feeds into operational readiness, include reports that translate training data into readiness metrics. For instance, "Number of graduates produced this quarter vs. requirement" or "Instructor pilot manning and utilization rates". A domain-specific need could be a report for higher headquarters showing status of the training pipeline: how many are in training, washout rates, average time to complete, etc. Similarly, resource reports might tie into maintenance: e.g. average aircraft availability rate or cancellation causes (how many sorties canceled due to maintenance vs weather). These are specific to military oversight needs (ensuring the training program is healthy and resource constraints are managed).
- **Safety Reports:** Military aviation places high importance on safety. The platform should be able to support generating safety-related reports, such as trend analysis of **Training Incidents or Flight Discrepancies**. If the system logs minor incidents (like "student busted altitude on 3 flights"), one could generate a report aggregating such data. Or a report of all **hazard reports** submitted in training (if integrated with a safety system or if instructors can flag maneuvers as safety concerns). This is domain-specific as it ties into the Aviation Safety program.
- **Certification and Compliance:** Reports that demonstrate compliance with training and regulatory requirements. For example, a report that lists all pilots and whether they have completed their **annual evaluations**, required academic training (e.g. anti-G training, instrument refresher), etc. This helps ensure the unit is in compliance with regulations. It's specific to the domain because these requirements come from military directives (like annual testing, etc.). The system should be able to generate a compliance checklist report.
- **Multi-Level Aggregation:** The platform might be used at a squadron level, but domain needs might require data to roll up to a group or wing level. Reports should accommodate aggregation across multiple units if needed. For instance, a wing commander might want a report combining data from two squadrons. The requirement is that the system can, with proper access, aggregate data by higher organization, reflecting the military organizational structure. This could involve hierarchical reporting roles or export capabilities that allow higher HQ to merge inputs from multiple system instances.

Resource Management Module

Overview: The Resource Management module deals with physical resources required for training – primarily aircraft, but also simulators, flight equipment, and facilities (like classrooms or ranges). Its core functions include tracking resource availability, scheduling or deconflicting resources, and possibly interfacing with maintenance systems to know when an asset is down for maintenance. The goal is to optimize resource utilization and ensure training has the necessary resources available at the right times.

User Requirements (Resource Management)

- **Aircraft Status Tracking:** Provide a real-time status board of all training aircraft. Users (maintenance officers or schedulers) should be able to mark each aircraft as *available*, *down for maintenance*, or *other status* (e.g. spare, scheduled inspection). This status should integrate with scheduling – only available aircraft can be assigned to flights. If an aircraft goes down

unexpectedly (breaks), a maintenance user can update its status, and the system will flag any upcoming flights assigned to that tail number for rescheduling.

- **Simulator and Facility Scheduling:** Include the ability to schedule **simulators** and other facilities (classrooms, briefing rooms) as resources. This is similar to aircraft scheduling but for ground resources. Users scheduling an academic lesson should reserve a classroom; scheduling a sim event should reserve a sim bay. The system should prevent double-booking of the same resource and show utilization. Possibly provide a separate calendar view for each type of resource to see when it's booked.
- **Maintenance Integration:** If possible, integrate with the maintenance management system (or allow input of maintenance events). The module could ingest planned maintenance schedules (e.g. "Aircraft 123 in phase inspection from Jan 5-10") so that schedulers know those aircraft are unavailable [35] . Additionally, allow quick submission of a maintenance request from the system if an issue is reported (this could simply generate a notification to maintenance personnel). While full maintenance tracking might be outside scope, at least a **maintenance schedule interface** is needed.
- **Resource Allocation Optimization:** When scheduling, the system can assist in choosing optimal resources. For example, if multiple aircraft are available, the system might suggest using the one with the least hours flown that week to balance wear (or follow any business rule given by maintenance). Or for simulators, automatically assign the first free simulator of required type. This is more of a smart feature: users can override, but it streamlines the process and ensures even resource usage.
- **Inventory of Equipment:** Optionally, track inventory of key equipment used in training (like NVG goggles, parachutes, etc., if relevant). Users should be able to see how many of each item are available and maybe assign them to flights or personnel. This ensures, for instance, if a night flight is scheduled, that NVGs are available and not beyond inspection date. The UI for this might be simpler (just a list of equipment and counts), but it helps the unit plan if certain equipment is a limiting factor.

Non-Functional Requirements (Resource Management)

- **Reliability:** Resource status data must be reliable and up-to-date. Any changes (like an aircraft marked down) should be immediately reflected to avoid scheduling errors. The system might employ a push update mechanism for resource status (like the maintenance officer's update immediately pushes to all scheduler screens). In case of integration with external maintenance systems, reliability of that integration (and handling of failures or lag) is important so that stale data doesn't mislead schedulers.
- **Performance:** Queries and updates related to resources should be lightweight – typically there are far fewer resources than personnel, so this module is not extremely data-heavy. However, it might be accessed frequently by scheduling algorithms. Thus, updating an aircraft's status or fetching all available jets should be near-instant. If integrated with maintenance, possibly cache some maintenance data to reduce external calls latency.
- **Usability:** A clear UI for resource management is needed, as it's often used under time pressure (e.g. when something breaks, you quickly need to swap resources). A **visual status board** with color codes (green for available, red for down, yellow for pending/trouble) would be useful. Drag-and-drop could allow assigning a tail number to a mission. Also, make it easy to switch a mission's resource – e.g. if aircraft A is down, with two clicks assign aircraft B and notify crew. This user-centered design will help operations run smoothly.
- **Scalability:** Usually, the number of resources (aircraft, sims) is limited (dozens or maybe a hundred), so performance is fine. But if the system covers multiple bases, it should be able to partition resources by base or type. Also consider if the system might one day handle other

resource types (like scheduling instructor availability as a “resource” in a sense) – ensure the design is generic enough to extend to new resource types without massive overhaul.

- **Integration & Modularity:** The Resource Management module may integrate with external systems (maintenance software, inventory systems). Design it with a modular adapter pattern – e.g. one implementation might pull data from a maintenance API, another might allow manual entry. This way, whether or not the base has a computerized maintenance system, the platform can work (either through integration or manual updates). Keep it modular so that changing an external system’s API or adding a new resource type (like vehicles) can be done by adding a new component rather than rewriting core logic.
- **Security:** Access control is important here too – perhaps only maintenance or admin roles can change resource statuses, whereas schedulers can view and assign but not mark a plane as “fixed”. Also, the data of resource usage might be sensitive (some could infer operational tempo from it), so ensure that only authorized users (mostly internal) can see details like how often an aircraft flew, etc. Communication with external maintenance systems must be secure (authenticated and encrypted).

Domain-Specific Requirements (Resource Management)

- **Military Maintenance Workflows:** Military aircraft maintenance operates under specific rules. The system should accommodate these – for example, if an aircraft is awaiting parts, it might be down for an indefinite period; if it’s scheduled for a **phase inspection** after a certain number of hours, the system should ideally know that threshold is approaching (maybe alert maintenance at 10 hours out) [36] [37]. Domain requirement: incorporate at least basic tracking of aircraft flying hours and notify when maintenance is due (if not integrated with full maintenance system). This ensures training doesn’t accidentally overschedule an aircraft beyond its limits.
- **Resource Prioritization:** In a training wing, some aircraft might be prioritized for certain training or have different configurations. The system may need to store **tail-specific info** like “Aircraft 101 has an upgraded avionics suite used for instrument training” – so if scheduling an instrument sortie, prefer 101. Or conversely, “Aircraft 202 has a minor defect, can’t be used for formation” – and the system should prevent assigning it to formation sorties. These little domain-specific details about each resource should be capturable (notes or attributes for each tail number) and the scheduling logic should respect them.
- **Fuel and Sortie Turnaround:** Military training can have quick turnarounds (one aircraft doing multiple back-to-back sorties). The system should enforce a minimum ground time between sorties for the same aircraft (e.g. 1 hour to refuel and inspect). This is a domain rule to ensure maintenance has time to prep the aircraft. The scheduling module, aided by resource management, should block someone from scheduling the same jet on two sorties too close together.
- **Alternate Resource Planning:** In the military context, contingency planning is common. The system could allow designating **alternate resources**. For example, for each flight, an alternate aircraft (or alternate instructor) is identified in case of primary resource failure. The requirements might be that the system can list alternates and, if triggered (say primary aircraft breaks), quickly swap to alternate with minimal input. This aligns with military practice of always having a backup plan for missions.
- **Resource Utilization Reporting:** The resource module should feed the Reporting module with domain-specific metrics like **aircraft utilization rate**, **maintenance downtime**, etc. For example, it should be able to report mean time between failures or downtime for each aircraft, because military leadership will be interested in how effectively the fleet is being used [38]. This ties in with the readiness aspect – if resource availability is low, training throughput suffers. So domain-specific requirement is to capture data needed for those metrics (like hours flown per

aircraft, number of cancellations due to maintenance) to generate reports that ultimately affect maintenance and procurement decisions.

System Administration Module

Overview: The System Administration module allows administrative users to configure and maintain the platform. This includes managing user accounts, roles, permissions, and global settings (like system-wide preferences, rule sets, etc.). It also covers audit logs, data backups, and integration configuration. Essentially, this module is about the behind-the-scenes management to keep the platform secure, up-to-date, and tailored to the organization's needs.

User Requirements (System Administration)

- **User Account Management:** Enable admins to create, edit, and deactivate user accounts. They should assign roles to each user (e.g. Student, Instructor, Scheduler, Maintenance, Commander, Admin). The interface should allow bulk user import (perhaps from a CSV or via integration with an HR system) to onboard a new class of students quickly. It should also handle password resets or linking accounts to authentication methods (like CAC or SSO).
- **Role-Based Permission Configuration:** Provide a clear UI to configure what each role can do (RBAC – Role-Based Access Control). For example, define that an "Instructor" can fill training records but cannot approve flights, whereas an "Ops Supervisor" can approve flights, etc. This might be a matrix of permissions or a set of checkboxes per role. The system likely has default roles, but the admin module should allow customizing them or adding new roles to fit any unique positions [39].
- **System Configuration Settings:** Allow configuration of system-wide settings such as: what types of notifications are enabled by default, what are the rule parameters (e.g. maximum daily flight hours for crew, default duty day length), thresholds for alerts (like notify when 10 days to qualification expiry). The admin UI should make these **configurable parameters** easily viewable and editable, ideally with safe defaults and explanations. This lets the platform be tailored to each military organization's policies without code changes [40].
- **Audit Log Access:** Provide administrators with access to audit logs of user activities. There should be an interface to search or filter these logs (e.g. "show all actions by user X" or "show all changes to schedule on date Y"). This is helpful for troubleshooting and security audits. The admin should also be able to export these logs if needed for investigations.
- **Data Management:** Tools for data backup, restore, and archiving. For example, the admin module could allow an admin to archive a completed class (move their records to an archive state) which might improve system performance for active users. Also, an admin might trigger a manual backup or download of data (especially before system updates). This could include backing up configuration settings as well.
- **Integration & Modules Management:** If the system has integration points (like with maintenance system, or if modules can be toggled), the admin interface should let admins configure these. For instance, enter API keys/URLs for external systems, schedule data sync tasks, or enable/disable certain features. In a military setting, integration with authentication systems (LDAP/Active Directory or CAC infrastructure) would be configured here – e.g. uploading the trusted certificates, mapping roles from directory groups, etc.
- **Monitoring Dashboard:** It can be useful to have a system health dashboard in the admin module showing server status, usage statistics (# of active users, etc.), and any error alerts. This helps the admin proactively address issues or coordinate with IT support.

Non-Functional Requirements (System Administration)

- **Security & Access:** Only highly trusted personnel (perhaps one or two per squadron or base) should have the system admin role. This module must be extremely secure. It should enforce multi-factor authentication for admin access (e.g. admin accounts may require CAC login *plus* biometric or OTP). Every action here is sensitive, so audit logging for admin actions is critical. Consider implementing the principle of least privilege – possibly separating duties (one admin manages users, another manages system config, if needed).
- **Usability and Safety:** Admin actions can have wide impact. The UI should include safeguards (e.g. "Are you sure?" confirmations, especially for destructive actions like deleting a user or changing a global setting). Possibly require double-confirmation or even dual-admin approval for certain critical changes (like wiping data or altering an approval rule). Despite these safeguards, the interface should remain clear and not overly complicated, since having a confusing admin interface could lead to mistakes.
- **Audit and Compliance:** The system administration functions must comply with any audit requirements of the military. For example, if the system deals with classified or sensitive but unclassified data, the administration of it might be subject to inspections. The software should facilitate compliance by making logs easy to export and configuration easy to review (like a report of all users and their roles, or all system settings values). Compliance with standards (e.g. DoD Information Assurance guidelines) might require documentation; the system could assist by producing a config baseline report.
- **Maintainability:** The admin module should be built to accommodate updates to requirements. New roles, new settings, or integration changes will happen – designing it in a way that adding a new configurable parameter is straightforward (perhaps a generic settings table) will help maintainability. Also, the interface should ideally be self-documenting (with help text) so new admins can understand it without needing to read code.
- **Scalability:** Even though the number of admin users is small, the actions might affect large data sets (like archiving a whole class). Ensure that bulk operations (like user import or data archive) are done efficiently (maybe asynchronously with a progress indicator) so that the admin UI remains responsive. The module should handle scaling in terms of configuration as well – e.g. if we add dozens of new config toggles over time, the UI should remain organized (maybe categorized sections) to scale with feature growth.
- **Modularity:** The admin module ties into all others, but it should be treated as a separate component that can be secured and even deployed separately if needed (some systems have a separate admin portal). Its design should not be too intertwined with business logic of other modules – rather, it triggers changes by calling their interfaces or adjusting database entries. This modular approach means the core modules enforce rules, while admin just sets the parameters. It also means any failure or misuse in admin module should not directly break user-facing functions (e.g. if an admin enters a nonsense configuration, the system should validate and reject it, or at least fail gracefully).

Domain-Specific Requirements (System Administration)

- **CAC/PKI Authentication Integration:** In military environments, user authentication often uses the DoD Common Access Card (CAC) or other Personal Identity Verification (PIV) cards. A domain-specific requirement is that the system **supports CAC login** for users and administrators. This means integrating with DoD Public Key Infrastructure – likely configuring the web servers to require client certificates. The admin module would handle mapping the CAC's certificate info to a user account on first login, etc. Additionally, the system should support signing forms or approvals with CAC certificates if biometrics aren't used universally.
- **STIG Compliance and Hardening:** The system (including the admin module) must comply with DoD Security Technical Implementation Guides (STIGs) or similar cybersecurity requirements.

Domain specifically, the admin should be able to apply security configuration like password policies, session timeouts, and account lockout thresholds as dictated by these policies. For instance, the admin interface could expose settings for “inactive session timeout = 15 minutes” or “password complexity: 2 uppercase, 2 lowercase, 2 symbols” in line with DoD rules. This ensures the software can be accredited for use on DoD networks.

- **Multi-Tenancy and Data Segregation:** If the platform might serve multiple training units or even different branches, the admin module must allow segregation of data. Domain requirement could be that an Air Force training squadron’s data should never be visible to a Navy training squadron on the same platform. This might be accomplished by separate instances or a robust multi-tenant design with strict access controls. The admin module would then have to manage tenants (maybe an admin of admins to create new unit spaces). While not needed if each unit deploys its own system, the requirement should consider future scaling where a centralized system serves many.
- **Localization for Terminology:** Different military branches or countries may have different terms (one might say “Sortie”, another “Mission”, etc.). The system admin module might include the ability to customize certain terminology or data fields to match local doctrine. This is domain-specific in that it allows the software to be used in various military contexts without hardcoding one branch’s jargon. For example, an admin setting could let them rename the label “Flight Commander” to “Section Leader” if needed, or toggle modules that aren’t used in some context.
- **Backup and Disaster Recovery Procedures:** Military operations often require a continuity of operations plan. The system should support domain-specific backup protocols, like taking backups to a secure offline storage, or supporting an **offline mode** entirely if the network goes down (perhaps a read-only offline backup server that can be used). Admins should be able to initiate **data exports** that could be used to run the training program manually if needed (for example, exporting next week’s schedule to a file in case systems are offline). This requirement ensures the training mission can continue even in degraded IT scenarios, which is a military consideration.
- **Audit Preparation:** In the military, inspections (like Standardization and Evaluation audits, or IG inspections) may require showing evidence of training management. The system should have features to help with this, such as one-click generation of common audit data (e.g. list of all current students and their status, list of instructors and last eval dates, etc.). While this overlaps with reporting, it’s driven by administrative compliance needs. Ensuring the system can quickly provide whatever documentation an inspection team asks for (within the scope of the system’s data) is a valuable domain-specific capability.

Cross-Cutting Development Considerations

While each module has specific requirements, there are overarching **development-side considerations** that apply to the entire platform. These concern how the system is implemented with respect to security, modularity, and scalability, as well as maintainability and overall architecture. The following table summarizes these key cross-cutting requirements and their implications:

Aspect	Development Considerations and Requirements
Security	Implement end-to-end security: **Role-based access control** for all modules (granular permissions per role) [39] ; strong user authentication (integration with DoD CAC/PIV and biometrics for critical actions); encrypt data in transit (TLS 1.2/1.3) and at rest (database encryption for sensitive fields). Follow military cybersecurity standards (e.g. NIST 800-53 controls, DoD STIGs) in development and configuration. Conduct regular security audits and penetration testing. All user actions should be **auditable**, with tamper-evident logs stored securely. Ensure sensitive operations (like mission approvals, admin changes) have multi-factor authentication and logging. The system should also enforce **session security** (auto-logout on inactivity, protection against XSS/CSRF given it's web-based).
Modularity	Design the system in a modular fashion, ideally using a microservices or layered architecture. Each module (Scheduling, Training, etc.) should be a **separable component** with a well-defined API interface. This allows independent development, testing, and scaling of modules. For example, the scheduling engine could be one service, the training records another – communicating through secure APIs. Modules should be **loosely coupled**, meaning changes in one (say, adding a new field in Training Records) has minimal impact on others. The platform should be highly **configurable** to meet different organizational needs without code changes [40] , which is achieved via modular design (e.g. rule engines or config files driving behavior). This modularity not only aids scalability but also maintainability – updates or new features can be added to one module without rewriting the entire system. All modules integrate into a cohesive whole through an integration layer or message bus, ensuring **seamless data flow** [41] .
Scalability	Build the platform to **scale horizontally** to support growing numbers of users, data, and possibly multiple training sites. Use scalable technologies (e.g. clustering for the application server, load balancers, database replication) so that the system can handle peak loads (like all users checking schedules in the morning). Plan for potentially thousands of concurrent users (if scaled to multiple schools) and design with that capacity in mind (efficient database queries, use of caching for frequent reads like schedule display). The system should also scale in terms of data over years of operation – archiving strategies and database sharding or partitioning might be used to keep performance steady as records grow. If using cloud infrastructure, ensure it can auto-scale under load. Also, test the system with large datasets and user loads to identify bottlenecks early.

Aspect	Development Considerations and Requirements
Performance	Apart from scaling, ensure the application is optimized for quick response. **Front-end performance** matters for a good user experience – use asynchronous loading, minimize heavy scripts, and optimize code so that pages (especially data-heavy ones like calendars) load within a couple of seconds. On the **server side**, use appropriate indexing in databases, caching of common queries (like today's schedule), and possibly queue background tasks (such as report generation or sending bulk notifications) so they don't block interactive use. Set performance benchmarks for key actions (e.g., schedule view load under 2s, report generation under 10s for typical range) and test against them. Also consider the network environment – the app might be used on military networks with potential bandwidth restrictions, so efficient data transfer (sending only necessary data, possibly compressing) is important.
Maintainability	Write clean, well-documented code and use a modular structure to make maintenance easier. The system will likely evolve with new training requirements and tech updates, so using standard frameworks and clear separation of concerns (MVC or similar patterns) will facilitate future developers. Implement automated testing (unit and integration tests) for critical components like the scheduling logic and qualification rule checks, to catch regressions when changes are made. Provide an administrative UI that is largely self-service for configuration, so that many tweaks do not require developer intervention (e.g. new qualification types, rule changes can be done by admins). Logging and error handling should be robust – in case of issues, errors should be logged with enough detail to troubleshoot, and the system should fail gracefully (e.g. if one module is down, others still function in isolation as much as possible). Maintainability also includes providing **support for updates**: plan for data migration if the schema changes, and use version control and CI/CD pipelines to deploy updates with minimal downtime.
Compliance	Ensure the system complies with all relevant **military and aviation regulations** for software and training records. This includes IT security certifications (Risk Management Framework accreditation to get an Authority to Operate on military networks), as well as compliance with records management (e.g. how long training records must be kept) and possibly FAA or equivalent regulations if any civilian crossover (for example, if the training records double as logbook entries, they should meet logging requirements). Accessibility compliance (Section 508) is also important for federal systems – the web interface should be navigable via keyboard and screen readers for any users with disabilities. The development must keep these in mind, using best practices to ensure the platform can be certified for use in its environment.

Finally, it is worth noting that the platform's design and requirements aim to **support the mission of military flight training** by improving efficiency, transparency, and data-driven management. By adhering to the above requirements and considerations, developers will implement a robust system that enhances training operations while meeting the strict standards of military security and reliability. Each module works in concert with others to deliver a cohesive user experience – from planning a flight, getting it approved with biometric security, executing it and logging the outcome, to analyzing the results and resources for continuous improvement. This comprehensive approach will ensure the platform is both **user-friendly** for day-to-day use and **technically sound** for long-term sustainment and scalability in the demanding context of military aviation training.

Sources: The requirements and considerations above are informed by best practices and features from modern aviation scheduling and training systems. For instance, ProDIGIQ's military flight operations software emphasizes rule-based crew scheduling with real-time updates and conflict alerts [9] [42], as well as integrated views and reporting [2] [34]. Their training management (AQP) system highlights the importance of electronic forms, curriculum integration, and performance tracking [22] [21]. Biometric security's role in defense operations is underscored by HID Global, noting that biometrics provide secure identity assurance for critical missions [18]. These sources, alongside military-specific regulations and practices, guided the formulation of a system that is **feature-rich**, secure, and tailored to the unique demands of military flight training.

---

[1] [5] [6] [9] [12] [13] [14] [17] [30] [40] [41] Crew Scheduling Module - ProDIGIQ
https://www.prodigiq.com/military/products/flight-operations-system/crew-scheduling-module/

[2] [3] [4] [7] [8] [10] [11] [16] [33] [34] [42] Flight Scheduling System (MILOS) - ProDIGIQ
https://www.prodigiq.com/military/products/flight-scheduling-system/

[15] [21] [22] [23] [24] [25] [26] [27] [28] [29] [31] [32] [39] Advanced Qualification Program (SIROS) - ProDIGIQ
https://www.prodigiq.com/military/products/advanced-qualification-program/

[18] [19] [20] Military & Defense Biometric Technology & Software | HID
https://www.hidglobal.com/solutions/biometrics-defense

[35] [36] [37] [38] Asset Management System (PAROS) - ProDIGIQ
https://www.prodigiq.com/military/products/asset-management-system/