Software Requirements Specification

for

UniLabBooker

Version 1.0

Prepared by Alex Kachur, Cyrille Yannis Sonfack Ngoufack & Rami Alshanwar

Centennial College

2025-08-08

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-05-08	Initial draft: populated Sections 1–2.4 and 2.7-3.3.	0.1
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-05-15	Added Appendix C: Stakeholder Register.	0.2
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-05-22	Added Appendix D: Interview Q&A transcripts.	0.3
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-05-29	Added Appendices A: Glossary and Appendices E: Functional and Non-Functional Requirements	0.4
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-05-30	Finalized SRS Part A: formatting polish, updated TOC, integrated all appen submission.	1.0
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-06-18	Added Use-Case Table; Formal UC + Swimlane + UCD; Appendix B models (domain summary, CRCs, detailed class diagram); B6 Gen-AI code skeleton.	2.0
Alex Kachur; Cyrille Yannis Sonfack Ngoufack; Rami Alshanwar	2025-08-08	Added all part C deliverables and finalized final submission	3.0

1. Introduction

1.1 Purpose

UniLabBooker v1.0 is a web and mobile portal for booking lab equipment. It solves the problem of over and underutilized resources. Students, faculty, and technicians use it to view calendars, make and change reservations, and get usage reports. This document covers all features and interfaces in the first release. Future analytics dashboards and third-party integrations are out of scope.

1.2 Document Conventions

This SRS uses Times New Roman 12 pt for body text. Heading levels follow the IEEE style. Requirements are labeled FR01, FR02 for functions and NFR01, NFR02 for qualities. Mandatory requirements start with “The system shall…”. Priorities appear as (Expected), (Normal), or (Exciting) after each title. All acronyms are spelled out on first use and defined in Appendix A. UML 2.5 diagrams appear as embedded images with Visio sources provided separately. Sub requirements inherit the parent’s priority unless noted otherwise.

1.3 Intended Audience and Reading Suggestions

Developers read Sections 3.1–3.3 for interface details.
Testers use Sections 4–5 (when available) to plan test cases.
Project managers focus on Sections 1–2 for scope and stakeholder roles.
Lab technicians review Sections 1.4, 2.2, and 3.2 for workflows and hardware interfaces.
Students and faculty read Sections 2.2, 2.3, and 3.1 to learn booking processes.
Documentation writers refer to the glossary in Appendix A and follow the conventions in 1.2.

1.4 Product Scope

UniLabBooker is a standalone system that maximizes lab equipment use and avoids double bookings. It displays a unified calendar of all resources. It automates booking requests, approvals, modifications, and cancellations. It limits usage by quotas per user role. It generates weekly and monthly reports for capacity planning. This portal improves transparency and reduces idle time.

1.5 References

- University Identity Management Service documentation — see TBD-01 in Appendix F.
- Internet Calendaring and Scheduling Core Object Specification (iCalendar)
  Internet Engineering Task Force. (2009). Internet calendaring and scheduling core object specification (iCalendar) (RFC 5545). https://www.rfc-editor.org/info/rfc5545
- The OAuth 2.0 Authorization Framework
  Internet Engineering Task Force. (2012). The OAuth 2.0 authorization framework (RFC 6749). https://www.rfc-editor.org/info/rfc6749

- PostgreSQL 16 Documentation
  The PostgreSQL Global Development Group. (2023). *PostgreSQL 16 documentation*. https://www.postgresql.org/docs/16/

2. Overall Description

2.1 Product Perspective

UniLabBooker is a new, self-contained system. It uses the university's OAuth 2.0 SSO for authentication. It sends emails through the campus SMTP server. It stores data in the existing PostgreSQL database.

2.2 Product Functions

The system shows equipment availability in a calendar view. It lets users create, modify, and cancel bookings. It enforces per user and per group quotas. It generates usage and capacity planning reports. It sends in app and email notifications. It relies on SSO for role-based access.

2.3 User Classes and Characteristics

Students book equipment for coursework or research and need a simple interface. Faculty have higher quotas and can approve student requests. Lab technicians handle booking approvals, maintenance, and status monitoring. IT administrators manage users, roles, and system settings via an admin console.

2.4 Operating Environment

The server runs Ubuntu 20.04 LTS on 4 vCPUs and 16 GB RAM. The web server is Apache Tomcat 9. The database is PostgreSQL 16. Clients use Chrome 90+, Firefox 88+, or Safari 14+ on desktop. Mobile users access a hybrid container on iOS 14+ or Android 10+. The stack uses Java 11 (Spring Boot) and React 17 with RESTful JSON APIs.

2.5 Design and Implementation Constraints

• Must use University OAuth 2.0 SSO for authentication and role mapping (no local passwords). • Must deploy on Ubuntu LTS with PostgreSQL 16, Java 11 (Spring Boot), and React 17. • Must integrate with the institutional SMTP relay for outbound email. • Must adhere to campus security and privacy policies and audit requirements.

2.6 User Documentation

Provide in-app help tooltips and a Quick Start guide (PDF) for Students/Faculty, plus admin console documentation for IT Administrators.

2.7 Assumptions and Dependencies

The university maintains the SSO and SMTP services. Campus network availability is 24/7. Users have valid university credentials. Time is synced via campus NTP servers for booking accuracy. Future releases may integrate equipment status APIs.

3. External Interface Requirements

3.1 User Interfaces

Users access the web portal via Chrome, Firefox, or Safari. The mobile app runs in an Ionic/Capacitor container on iOS and Android. IT administrators use a secured React-based admin console.

3.2 Hardware Interfaces

The system supports USB-connected RFID/card readers for lab entry. It optionally supports USB barcode scanners for check-in/out. Future versions may integrate with network-connected equipment controllers. (admin)

3.3 Software Interfaces

The OAuth 2.0 SSO handles authentication and roles. The SMTP server sends booking confirmations and alerts. The PostgreSQL database stores users, equipment, bookings, and usage logs. The system can export .ics files following the iCalendar RFC 5545 standard.

3.4 Communications Interfaces

HTTP/HTTPS (TLS 1.3) for all client–server traffic; REST/JSON APIs between web/mobile and server; SMTP with STARTTLS for notifications; optional iCalendar (.ics) exports for booking events; time synchronization via NTP per campus standard.

3.5 Use-Case Table

Subsystem: Booking

Use Case Name	Related Requirement ID(s)	Actor(s)	Brief Description
View-Calendar	FR01	Student/Faculty (human)	Displays the equipment calendar view showing all existing reservations and available time slots.
Create-Reservation	FR02, FR03, FR06	Student/Faculty (human)	Creates a new booking request and saves it as Pending Approval; if the desired slot is full, offers option to join the waitlist.
Modify-Reservation	FR05	Student/Faculty (human)	Allows the user to modify date/time or lab resource for an existing reservation within policy (subject to approval if applicable).
Cancel-Reservation	FR05	Student/Faculty (human)	Allows the user to cancel an existing reservation at least 2 hours before start; releases the time slot and triggers notifications.
Review-Booking-Request	FR04	Lab Technician; Faculty Member	Reviewer approves or rejects a Pending Approval request; on approval, reservation becomes Confirmed and notifications are sent; on rejection, request is closed and next waitlisted user is promoted if applicable.

Use Case Name	Related Requirement ID(s)	Actor(s)	Brief Description
Join-Waitlist	FR06	Student; Faculty Member	Adds the requester to a FIFO waitlist when a desired time slot is fully booked.
Process-Waitlist	FR06	Lab Technician	Promotes the next user on the FIFO waitlist and holds the slot for 15 minutes; if not accepted in time, auto-promote the next user.

Subsystem: Administration

Use Case Name	Related Requirement ID(s)	Actor(s)	Brief Description
Generate-Usage-Report	FR08	Department Chair	Produces weekly/monthly utilization reports by equipment and user group.
Manage-Quotas	FR07	IT Administrator	Defines and enforces per-user and per-group weekly booking quotas.
Manage-Role-Mappings	FR10	IT Administrator	Assigns application roles to SSO identities; supports deactivating local access; no local user creation or password management (OAuth SSO owns authentication).

Abstract (no actors)

Use Case Name	Related Requirement ID(s)	Actor(s)	Brief Description
Login-User	FR10	— (abstract)	Authenticates a user and establishes role-based permissions.
Logout-User	FR10	— (abstract)	Ends an authenticated session and revokes access tokens.

4. System Features

The Detailed functional requirements are listed in Appendix E.

4.1 Create-Reservation Use Case

Use Case ID: UC-03
Use Case Name: Create-Reservation

Goal in Context

Enable a student or faculty member to reserve lab equipment for a specific time slot, with automatic waitlist handling if the slot is full.

Scope

UniLabBooker system

Level

User goal (typical end-user scenario)

Primary Actor

Student; Faculty Member

Supporting Systems

• ReservationService (handles booking logic)
• CalendarService (refreshes calendar after booking)
• WaitlistService (manages waitlist entries)
• EmailService (external, sends notifications: request received, approvals/rejections, and waitlist updates)

Stakeholders and Interests

- **Student/Faculty:** Wants a straightforward way to book lab time without conflicts, receive immediate acknowledgement and timely approval decision, and fall back to waitlist if needed.
- **Lab Technician:** Needs to ensure no double-bookings and that waitlisted requests can be processed when slots free up.
- **IT Administrator:** Must enforce per-user quotas, maintain system reliability, and guarantee secure authentication.

Preconditions

1. Actor is logged in with a valid session (FR10).

2. The system's database of equipment and available time slots is up-to-date (implied by FR01/FR08 background data).

Trigger

Actor clicks "New Reservation" on the calendar interface.

Main Success Scenario (Basic Flow)

1. [FR02] Actor selects “New Reservation.”
   - ReservationService displays the blank Reservation Form (date, start time, end time, lab resource dropdown).
2. [FR02] Actor enters the desired date, start time, end time, and selects a lab resource.
3. [FR02/FR03] Actor clicks “Submit.”
4. [FR03] System validates the requested time range against existing reservations.
   - If there is no conflict, proceed.
5. [FR07] System verifies the actor’s weekly quota has not been exceeded.
   - If under quota, proceed.
6. [FR03] System checks that the selected lab resource is available (not under maintenance or blocked).
   - If available, proceed.
7. [FR03] System saves the new reservation request with status **Pending Approval** and forwards it for review (FR04).
8) System refreshes the calendar and overlays a “Pending Approval” visual marker on the selected slot. The underlying TimeSlot.status remains AVAILABLE until the reservation is confirmed.
9. [FR09] System sends a “Request received — pending approval” notification (email and/or in-app).
10. Actor sees an on-screen confirmation message.

Extensions (Alternate Flows / Exceptions)

3a. [Time Slot Already Taken]
   1. During validation (Step 4), the system detects a conflicting reservation (FR03).
   2. System displays “Time slot unavailable. Would you like to join the waitlist?” on the Reservation Form.
   3. Actor may choose to:
      a. Select a different slot (return to Step 2).
      b. Click “Join Waitlist.”
         - [FR06] System adds actor to the waitlist for that time slot.
         - [FR09] System invokes EmailService to send “Added to Waitlist” notification.
         - **Postcondition (3a):** Actor is waitlisted (FR06) and notified (FR09).
   4. Flow ends.
5a. [Quota Exceeded]
   1. During quota check (Step 5), actor has already used up weekly quota (FR07).
   2. System displays “Quota exceeded. Cannot create a new reservation.”
3. Actor may either cancel or modify an existing reservation (cancellation permitted **≥ 2 hours** before start) to free up quota, then retry.
   4. Flow ends.
6b. [Lab Resource Unavailable]
   1. During availability check (Step 6), system finds the lab resource is blocked for maintenance (FR03).
   2. System displays “Selected lab is unavailable (maintenance). Choose another lab.”
   3. Actor returns to Step 2.
   4. Flow continues.
8a. [Email Failure]
1. After saving (Step 7) and calendar refresh (Step 8), the notification service fails to deliver (e.g., SMTP error) (FR09).
   2. ReservationService logs the email error in the system log and marks the reservation as “pending notification.”

3. Actor still sees an on-screen **pending request** message; the booking remains recorded.
4. IT Administrator is notified of the email queue failure via monitoring alerts (outside this use case).
5. Flow ends with postconditions as in the main scenario (request saved as **Pending Approval**, calendar updated).

Postconditions

- A new reservation **request** exists for the selected date/time/lab resource with status **Pending Approval** (FR03/FR04).
- Calendar view reflects the **pending** booking (FR01).
- A “request received — pending approval” notification is sent (or queued) to the actor (FR09).
- If actor opted for the waitlist, a waitlist entry is created (FR06) and a notification is sent (FR09).

Special Requirements (Non-Functional)

1. **NFR01**: Performance — **p95 ≤ 3s, p99 ≤ 5s at 200 concurrent users** for form display, validation, save, and calendar refresh.
2. **NFR03**: Security — All data transmissions (login, form submission, email API calls) must use **TLS 1.3**; authentication via OAuth 2.0 SSO; MFA required for admin operations.
3. **Input Validation**: Form fields (date/time) must enforce valid lab hours per FR02; server-side checks must match calendar policies (FR03).
4. **Audit Logging**: Every reservation action (create, modify, cancel, waitlist) must generate an audit entry with user ID, timestamp, and action type; logs retained for **≥ 1 year**.

4.2 Swimlane Activity Diagram

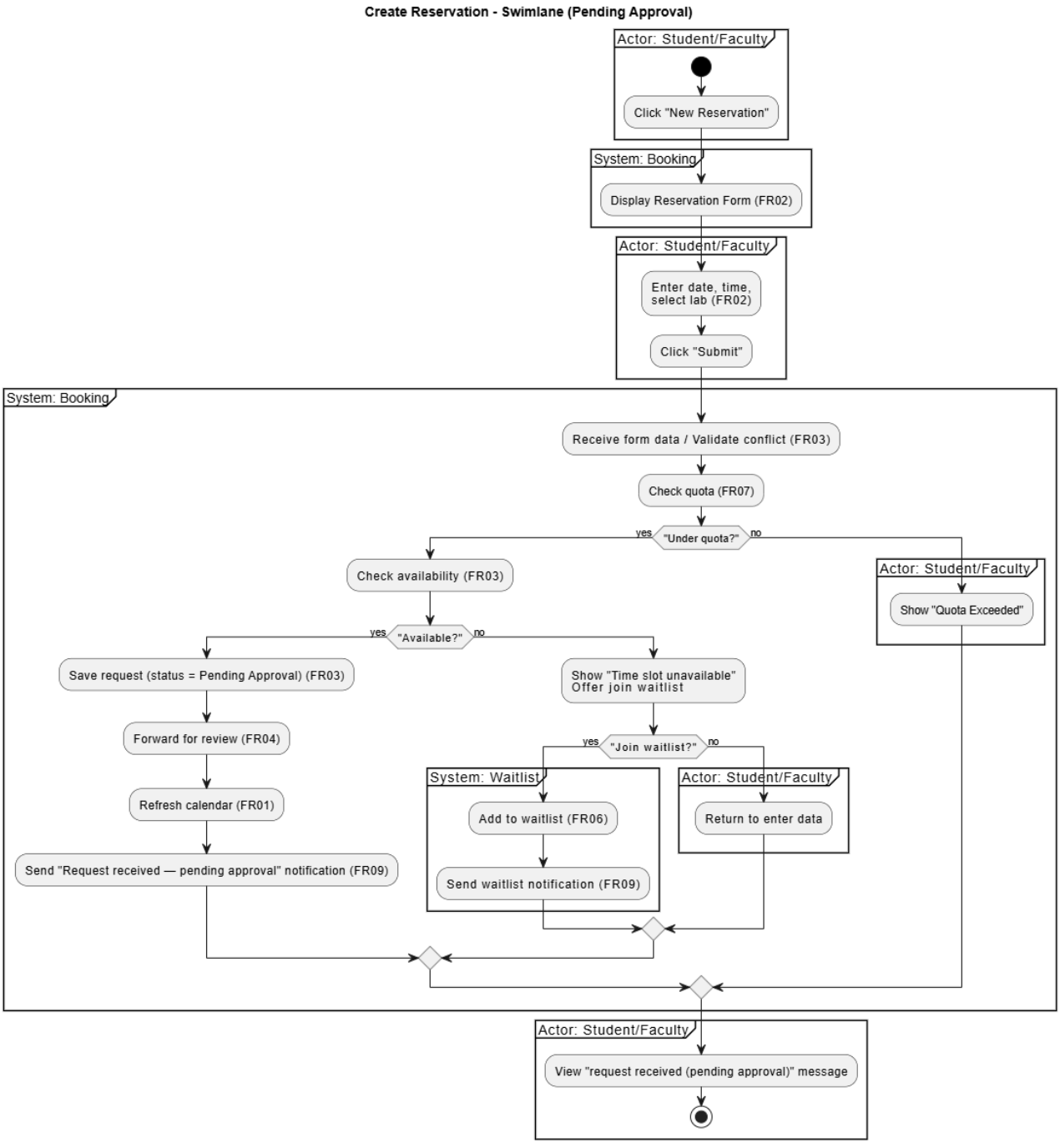

Figure 1 Swimlane for Create-Reservation

4.3 Use-Case Diagram

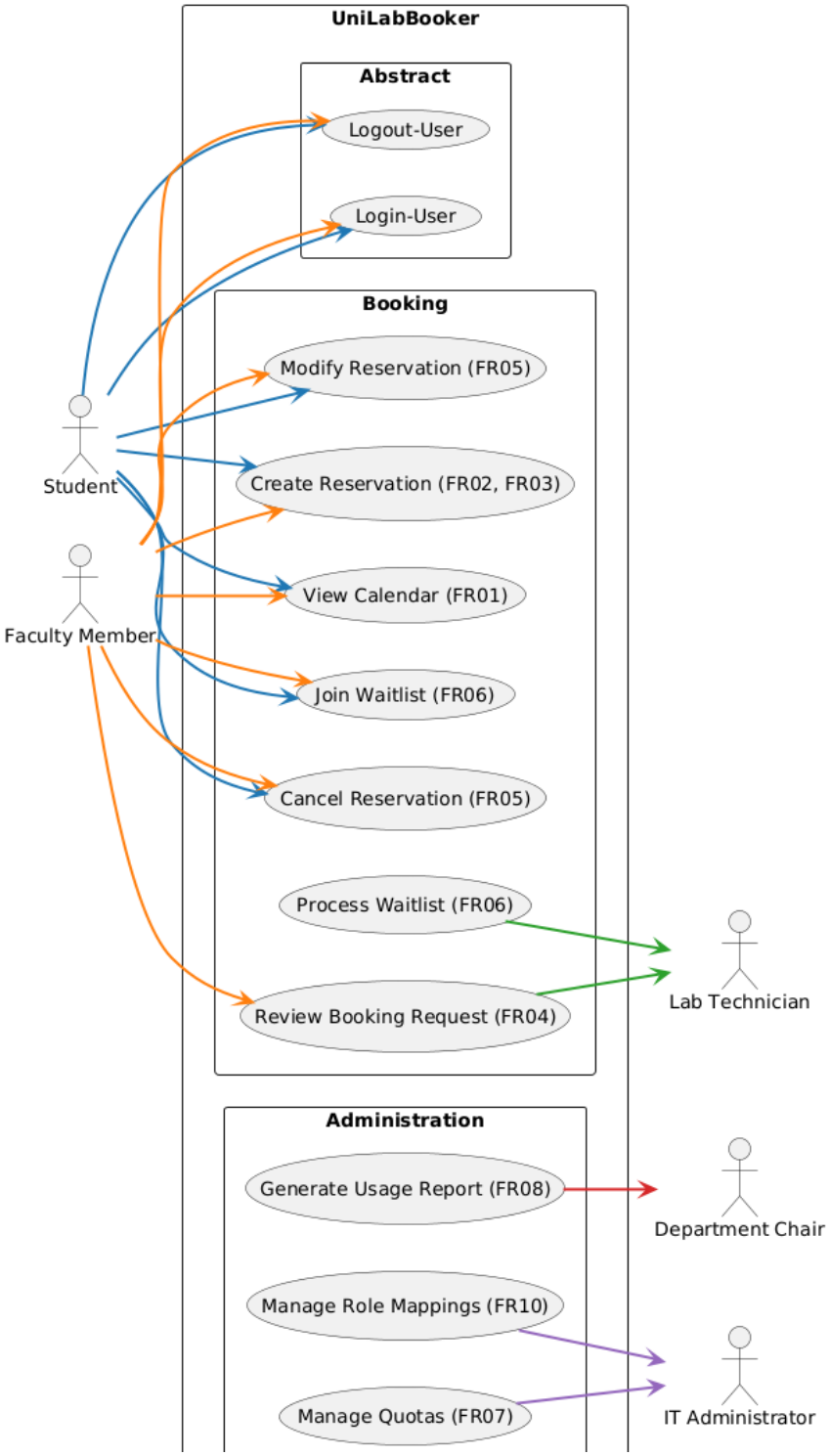

Figure 2 Use Case Diagram

5. Other Nonfunctional Requirements

The Detailed Non-functional requirements are listed in Appendix E.

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

5.5 Business Rules

<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

Term / Acronym	Definition
SRS	Software Requirements Specification: the official document that captures all functional and non-functional requirements of UniLabBooker v1.0.
API	Application Programming Interface: a set of routines, protocols, and tools for building software and applications.
DBMS	Database Management System: software for creating, querying, updating, and managing databases (e.g., PostgreSQL 16).
OAuth 2.0	An open standard for access delegation, used here to allow secure “Login with University” Single Sign-On (SSO) without exposing user passwords.
iCalendar	Internet standard (RFC 5545) for calendaring data exchange, used for import/export of booking events.
SSO	Single Sign-On: an authentication process that allows a user to access multiple applications with one set of login credentials.
UML	Unified Modeling Language: a standardized modeling language used to specify, visualize, and document software system artifacts.
SMTP	Simple Mail Transfer Protocol: the Internet standard for email transmission, used here for notification emails.
RBAC	Role-Based Access Control: access control model where permissions are assigned to roles, and users acquire permissions by their roles.
FR / FR01	Functional Requirement: a statement of a specific behavior or function of the system (e.g. FR01 “Display calendar view”).
NFR / NFR01	Non-Functional Requirement: a constraint on the system’s operation (e.g. NFR01

	"System shall respond within 3 s under 200 concurrent users").
TLS	Transport Layer Security: cryptographic protocol to secure communications over a network.
RFC	Request for Comments: a memorandum published by the IETF describing methods, behaviors, research, or innovations applicable to Internet technologies.
JSON	JavaScript Object Notation: lightweight data-interchange format used in API payloads and configuration files.
QoS	Quality of Service: overall performance of a system, particularly the guarantee of certain performance metrics (latency, uptime, throughput).
UI	User Interface: the means by which the user interacts with the system (web pages, dialogs, forms).

Appendix B: Analysis Models

1. Domain Class Summary & First-Cut Diagram

- **User** — system person authenticated via SSO. Attributes: userId, name, email, ssoSubject. Relationships: 1..* Reservations (requests), 1..* WaitlistEntries, 1..* Notifications (recipient), 1..* RoleMappings, 1..* QuotaUsages.
- **Role** — application role (Student, FacultyMember, LabTechnician, DepartmentChair, ITAdministrator).
- **RoleMapping** — SSO group→role mapping for a user. Attributes: ssoGroupId. Relationships: many-to-1 User, many-to-1 Role.
- **Lab** — lab container. Attributes: labId, name, location. Relationships: 1..* TimeSlots.
- **TimeSlot** — reservable time interval. Attributes: slotId, startTime, endTime, status: SlotStatus. Relationships: 0..* WaitlistEntries; 1 Reservation when reserved.
- **Reservation** — booking of one TimeSlot. Attributes: reservationId, createdAt, approvedAt?, cancelledAt?, status: ReservationStatus. Relationships: 1 User (requester), 1 TimeSlot (reserves), 0..1 Approval, 0..* Notifications.
- **Approval** — decision for a Reservation. Attributes: decision: Decision, decidedAt, comment. Relationships: 1 Reservation, 1 User (decidedBy).
- **WaitlistEntry** — user queued for a TimeSlot. Attributes: entryId, joinTime. Relationships: 1 User, 1 TimeSlot, 0..* WaitlistOffers. Constraint: ≤1 active entry per (User, TimeSlot).

- **WaitlistOffer** — offer to claim a slot. Attributes: offerId, createdAt, expiresAt (+15m), acceptedAt?, status: OfferStatus. Relationships: 1 WaitlistEntry, 0..* Notifications.
- **QuotaPolicy** — per-user cap policy. Attributes: period ('week'), maxActive. Relationships: 1..* QuotaUsages.
- **QuotaUsage** — user's usage in a period. Attributes: periodStart, activeCount, lastEvaluatedAt. Relationships: 1 User, 1 QuotaPolicy.
- **Notification** — message sent to a user. Attributes: type: NotificationType, channel: Channel, status: DeliveryStatus, sentAt. Relationships: 1 User (recipient), 0..1 Reservation, 0..1 WaitlistOffer.

Enumerations

SlotStatus: AVAILABLE, RESERVED, UNAVAILABLE
ReservationStatus: PendingApproval, Confirmed, Rejected, Cancelled
Decision: Approved, Rejected
OfferStatus: Sent, Accepted, Declined, Expired
NotificationType: RequestReceived, Approved, Rejected, WaitlistOffer, Reminder, Cancellation
Channel: Email, InApp
DeliveryStatus: Queued, Sent, Failed

Policy Notes / Invariants

• Cancel allowed only if now ≤ startTime − 2 hours.
• Waitlist offers hold for 15 minutes; on expiry/decline, slot passes to the next entry (FIFO).
• When Reservation.status = Confirmed ⇒ TimeSlot.status = RESERVED.
• When Reservation.status ∈ {Cancelled, Rejected} ⇒ TimeSlot.status = AVAILABLE.
• RoleMapping uniqueness: (user, role) or (user, ssoGroupId).
• QuotaUsage uniqueness: (user, periodStart).
• A user may have at most one ACTIVE WaitlistEntry per TimeSlot.

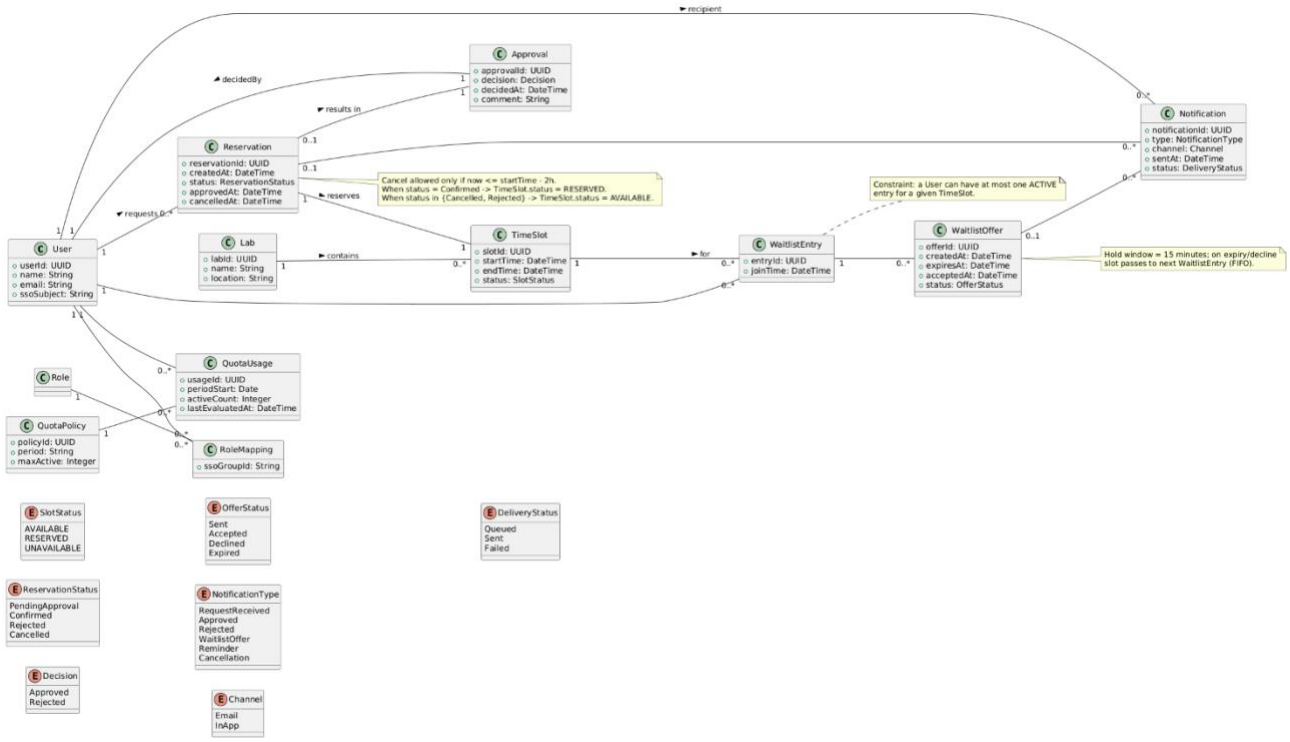

Figure 3 Class Diagram

2. CRC Cards

User	
Super Classes: None	
Sub Classes: None	
Description:	SSO-authenticated person; no local passwords. Holds identity and receives notifications.
Attributes:	
Name	Description
userId	Unique identifier for a user
name	Full name of the user
email	User's email address
ssoSubject	OIDC subject identifier from SSO
Responsibilities:	
Name	Collaborator
linkRoleMapping(ssoGroupId, role)	RoleMapping, Role
viewQuotaUsage(period)	QuotaUsage
receiveNotification(notification)	Notification

Role	
Super Classes: None	
Sub Classes: None	
Description:	Application role assigned via SSO mapping (e.g., Student, FacultyMember, LabTechnician, DepartmentChair, ITAdministrator).
Attributes:	
Name	Description
name	Role name
Responsibilities:	
Name	Collaborator
isAuthorized(user, action, resource)	RoleMapping

RoleMapping	
Super Classes: None	
Sub Classes: None	
Description:	Maps a user's SSO group to an application role; governs permissions.
Attributes:	
Name	Description
ssoGroupId	External SSO group identifier
Responsibilities:	
Name	Collaborator
assign(user, role, ssoGroupId)	User, Role
revoke(user, role)	User, Role
ensureUnique(user, role)	User, Role

Approval	
Super Classes: None	
Sub Classes: None	
Description:	Decision record for a Reservation (FR04).
Attributes:	
Name	Description
decision	Approved | Rejected
decidedAt	Decision timestamp
comment	Optional reviewer note
Responsibilities:	
Name	Collaborator
applyDecision(reservation, decision)	Reservation
setDecidedBy(user)	User

Notification	
Super Classes: None	
Sub Classes: None	
Description:	Message sent to a user for reservation lifecycle and waitlist offers (FR09).
Attributes:	
Name	Description
type	RequestReceived | Approved | Rejected | WaitlistOffer | Reminder | Cancellation
channel	Email | InApp
status	Queued | Sent | Failed
sentAt	Dispatch timestamp
Responsibilities:	
Name	Collaborator
deliverTo(user)	User
relateToReservation(reservation)	Reservation
relateToOffer(offer)	WaitlistOffer

Lab	
Super Classes: None	
Sub Classes: None	
Description:	A lab that contains many reservable time slots.
Attributes:	
Name	Description
labId	Unique identifier
name	Lab display name
location	Location or building/room
Responsibilities:	
Name	Collaborator
getTimeSlots()	TimeSlot
publishSchedule()	TimeSlot

TimeSlot	
Super Classes: None	
Sub Classes: None	
Description:	Reservable interval for a specific lab.
Attributes:	
Name	Description
slotId	Unique identifier
startTime	Start datetime
endTime	End datetime
status	AVAILABLE | RESERVED | UNAVAILABLE
Responsibilities:	
Name	Collaborator
reserveFor(reservation)	Reservation
release()	Reservation
maintainAvailability()	Reservation

Reservation	
Super Classes: None	
Sub Classes: None	
Description:	Booking request for one TimeSlot. FR03 saves as PendingApproval; FR04 decides.
Attributes:	
Name	Description
reservationId	Unique identifier
createdAt	Creation timestamp
approvedAt?	When approved
cancelledAt?	When cancelled
status	PendingApproval | Confirmed | Rejected | Cancelled
Responsibilities:	
Name	Collaborator
enforceCancelRule(now)	TimeSlot
linkApproval(approval)	Approval
attachNotification(notification)	Notification

WaitlistEntry	
Super Classes: None	
Sub Classes: None	
Description:	A user's queue entry for a TimeSlot (FR06).
Attributes:	
Name	Description
entryId	Unique identifier
joinTime	When entry was created
Responsibilities:	
Name	Collaborator
createOffer()	WaitlistOffer
leaveWaitlist()	Notification
notifyPromotion()	Notification

WaitlistOffer	
Super Classes: None	
Sub Classes: None	
Description:	Offer to claim a freed slot; expires after 15 minutes.
Attributes:	
Name	Description
offerId	Unique identifier
createdAt	Creation timestamp
expiresAt	Expiry timestamp (createdAt + 15m)
acceptedAt?	When accepted
status	Sent | Accepted | Declined | Expired
Responsibilities:	
Name	Collaborator
accept()	Reservation, Notification
expire()	Notification
decline()	Notification

QuotaPolicy	
Super Classes: None	
Sub Classes: None	
Description:	Defines per-user caps over a period (FR07).
Attributes:	
Name	Description
policyId	Unique identifier
period	e.g., 'week'
maxActive	Max active reservations per user in period
Responsibilities:	
Name	Collaborator
isAllowed(user, atTime)	QuotaUsage
applyPolicy(user)	QuotaUsage

QuotaUsage	
Super Classes: None	
Sub Classes: None	
Description:	Tracks a user's usage within a policy period.
Attributes:	
Name	Description
usageId	Unique identifier
periodStart	Period start date
activeCount	Active reservations count
lastEvaluatedAt	Last evaluation timestamp
Responsibilities:	
Name	Collaborator
incrementOn(reservation)	Reservation
decrementOn(reservation)	Reservation
withinLimit(policy)	QuotaPolicy

3. Detailed Class Diagram

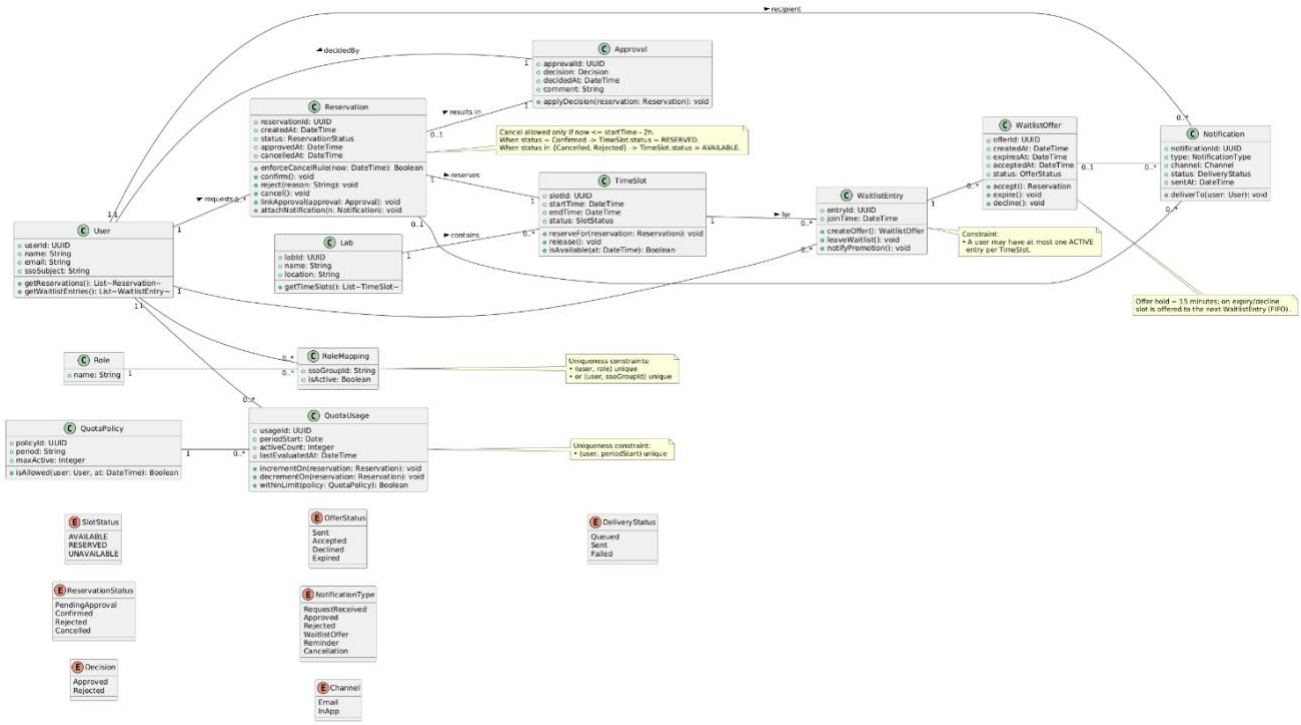

Figure 4 Updated Class Diagram

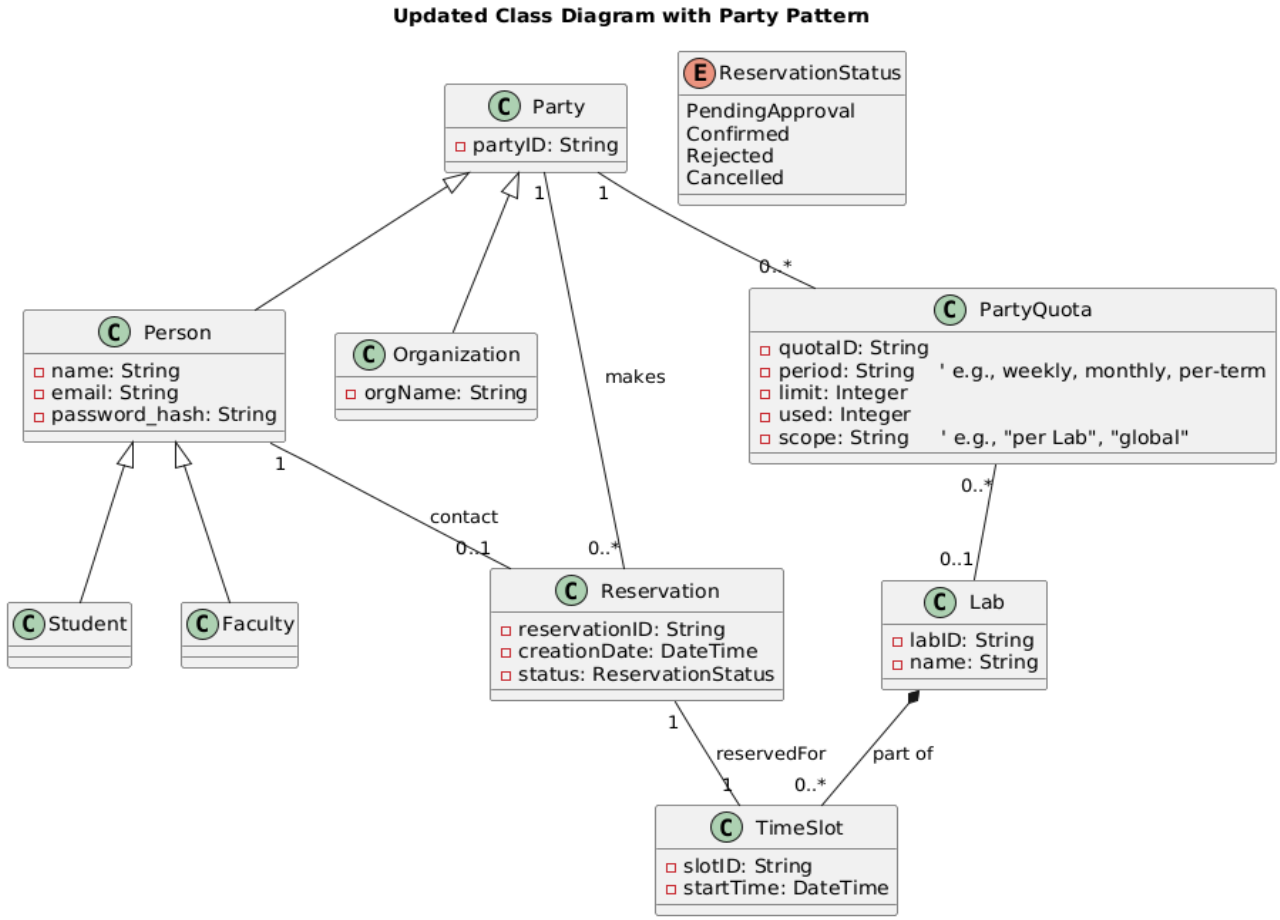

Figure 5 Party pattern class diagram

The Party Analysis Pattern provides significant value to the UniLabBooker project by elegantly solving the core requirement for "per-group" quota management, as stated in FR07. The original class model could only handle individual users, but by introducing an abstract Party superclass, the system can now treat both a Person (like a student or faculty member) and an Organization (like a research team or course section) as a single, unified entity. This greatly simplifies the design for key functions like making reservations and enforcing quotas, as the logic no longer needs to differentiate between an individual and a group. Ultimately, this pattern avoids code duplication and makes the application more flexible and scalable for future needs.

4. Gen AI Class Diagram (Code)

```
# UniLabBooker domain model (B-6, aligned with Part A/B/CRCs)
from __future__ import annotations
from dataclasses import dataclass, field
from datetime import datetime, timedelta, date
```

```python
from enum import Enum
from typing import List, Optional


# ===== Enums =====
class SlotStatus(Enum):
    AVAILABLE = "AVAILABLE"
    RESERVED = "RESERVED"
    UNAVAILABLE = "UNAVAILABLE"


class ReservationStatus(Enum):
    PendingApproval = "PendingApproval"
    Confirmed = "Confirmed"
    Rejected = "Rejected"
    Cancelled = "Cancelled"


class Decision(Enum):
    Approved = "Approved"
    Rejected = "Rejected"


class OfferStatus(Enum):
    Sent = "Sent"
    Accepted = "Accepted"
    Declined = "Declined"
    Expired = "Expired"


class NotificationType(Enum):
    RequestReceived = "RequestReceived"
    Approved = "Approved"
    Rejected = "Rejected"
    WaitlistOffer = "WaitlistOffer"
    Reminder = "Reminder"
    Cancellation = "Cancellation"


class Channel(Enum):
    Email = "Email"
    InApp = "InApp"


class DeliveryStatus(Enum):
    Queued = "Queued"
```

```
    Sent = "Sent"
    Failed = "Failed"


# ===== Core Identity (SSO) =====
@dataclass
class User:
    user_id: str
    name: str
    email: str
    sso_subject: str


@dataclass
class Role:
    name: str  # Student, FacultyMember, LabTechnician, DepartmentChair,
ITAdministrator


@dataclass
class RoleMapping:
    user: User
    role: Role
    sso_group_id: str
    is_active: bool = True
    # Constraint: unique (user, role) OR unique (user, sso_group_id)


# ===== Lab & Slots =====
@dataclass
class Lab:
    lab_id: str
    name: str
    location: str
    def get_time_slots(self) -> List["TimeSlot"]:
        raise NotImplementedError


@dataclass
class TimeSlot:
    slot_id: str
    start_time: datetime
    end_time: datetime
    status: SlotStatus = SlotStatus.AVAILABLE
    reservation: Optional["Reservation"] = None
```

```
    def reserve_for(self, reservation: "Reservation") -> None:
        if self.status != SlotStatus.AVAILABLE:
            raise ValueError("Slot not available")
        self.reservation = reservation
        self.status = SlotStatus.RESERVED

    def release(self) -> None:
        self.reservation = None
        self.status = SlotStatus.AVAILABLE

    def is_available(self, at: datetime) -> bool:
        return self.status == SlotStatus.AVAILABLE and self.start_time > at


# ===== Reservations & Approvals =====
@dataclass
class Reservation:
    reservation_id: str
    user: User
    time_slot: TimeSlot
    created_at: datetime
    status: ReservationStatus = ReservationStatus.PendingApproval
    approved_at: Optional[datetime] = None
    cancelled_at: Optional[datetime] = None

    def enforce_cancel_rule(self, now: datetime) -> bool:
        # Cancel allowed only if now <= start_time - 2 hours
        return now <= (self.time_slot.start_time - timedelta(hours=2))

    def confirm(self) -> None:
        self.status = ReservationStatus.Confirmed
        self.approved_at = datetime.utcnow()
        self.time_slot.reserve_for(self)

    def reject(self, reason: str = "") -> None:
        self.status = ReservationStatus.Rejected
        self.approved_at = None
        # Slot becomes AVAILABLE by invariant (if it had been held)

    def cancel(self, now: Optional[datetime] = None) -> None:
        now = now or datetime.utcnow()
        if not self.enforce_cancel_rule(now):
            raise ValueError("Cancellation window elapsed")
        self.status = ReservationStatus.Cancelled
        self.cancelled_at = now
```

```
        self.time_slot.release()

    def link_approval(self, approval: "Approval") -> None:
        approval.apply_decision(self)


@dataclass
class Approval:
    approval_id: str
    decided_by: User
    decision: Decision
    decided_at: datetime
    comment: str = ""

    def apply_decision(self, reservation: Reservation) -> None:
        if self.decision is Decision.Approved:
            reservation.confirm()
        else:
            reservation.reject(self.comment)


# ===== Waitlist =====
@dataclass
class WaitlistEntry:
    entry_id: str
    user: User
    time_slot: TimeSlot
    join_time: datetime
    # Constraint: user may have at most ONE ACTIVE entry per (user, time_slot)

    def create_offer(self, now: Optional[datetime] = None) -> "WaitlistOffer":
        now = now or datetime.utcnow()
        return WaitlistOffer(
            offer_id=f"offer-{self.entry_id}-{int(now.timestamp())}",
            entry=self,
            created_at=now,
            expires_at=now + timedelta(minutes=15),
            status=OfferStatus.Sent,
        )

    def leave_waitlist(self) -> None:
        # remove from queue in application service
        pass

    def notify_promotion(self) -> None:
```

```
        pass


@dataclass
class WaitlistOffer:
    offer_id: str
    entry: WaitlistEntry
    created_at: datetime
    expires_at: datetime
    status: OfferStatus = OfferStatus.Sent
    accepted_at: Optional[datetime] = None

    def accept(self) -> Reservation:
        self.status = OfferStatus.Accepted
        self.accepted_at = datetime.utcnow()
        # Create reservation for the entry's user/slot
        res = Reservation(
            reservation_id=f"res-{self.offer_id}",
            user=self.entry.user,
            time_slot=self.entry.time_slot,
            created_at=datetime.utcnow(),
            status=ReservationStatus.PendingApproval,
        )
        return res

    def expire(self) -> None:
        self.status = OfferStatus.Expired

    def decline(self) -> None:
        self.status = OfferStatus.Declined


# ===== Quotas =====
@dataclass
class QuotaPolicy:
    policy_id: str
    period: str  # e.g., "week"
    max_active: int

    def is_allowed(self, user: User, at: datetime) -> bool:
        # evaluate against QuotaUsage
        raise NotImplementedError


@dataclass
```

```python
class QuotaUsage:
    usage_id: str
    user: User
    policy: QuotaPolicy
    period_start: date
    active_count: int
    last_evaluated_at: datetime
    # Constraint: unique (user, period_start)

    def increment_on(self, reservation: Reservation) -> None:
        self.active_count += 1
        self.last_evaluated_at = datetime.utcnow()

    def decrement_on(self, reservation: Reservation) -> None:
        self.active_count = max(0, self.active_count - 1)
        self.last_evaluated_at = datetime.utcnow()

    def within_limit(self, policy: QuotaPolicy) -> bool:
        return self.active_count <= policy.max_active


# ===== Notifications =====
@dataclass
class Notification:
    notification_id: str
    type: NotificationType
    channel: Channel
    status: DeliveryStatus
    sent_at: Optional[datetime]
    recipient: User
    reservation: Optional[Reservation] = None
    waitlist_offer: Optional[WaitlistOffer] = None

    def deliver_to(self, user: User) -> None:
        self.recipient = user
        # delivery handled by infra layer; record status here
```

5. State-Transition Diagrams

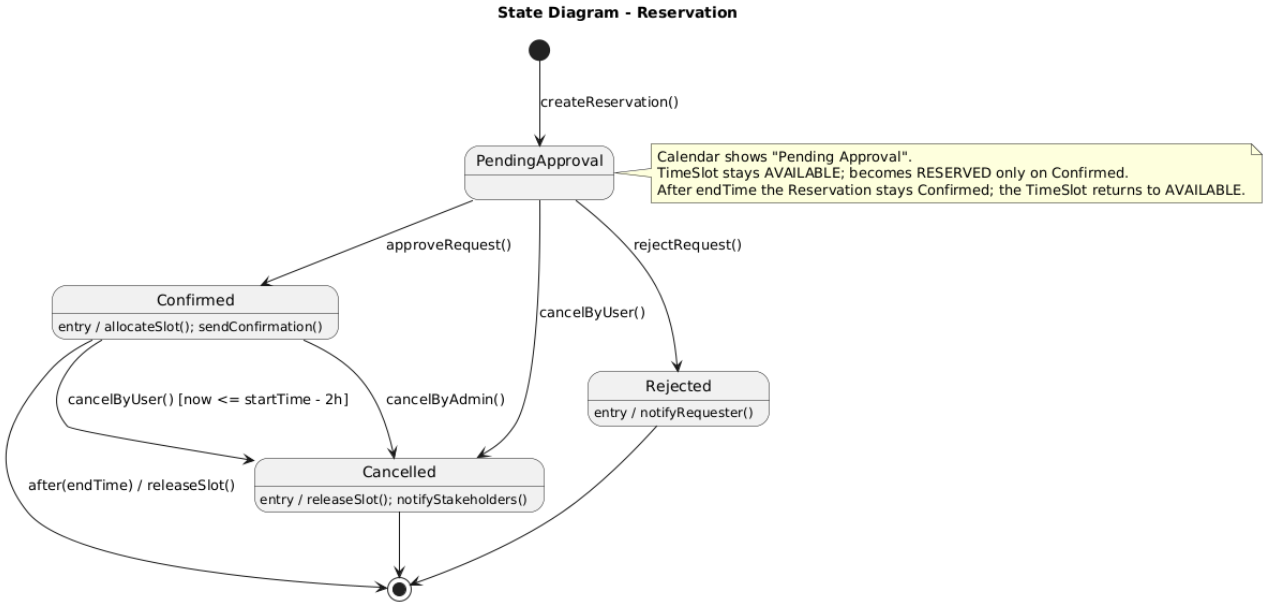

Figure 6 Reservation state diagram

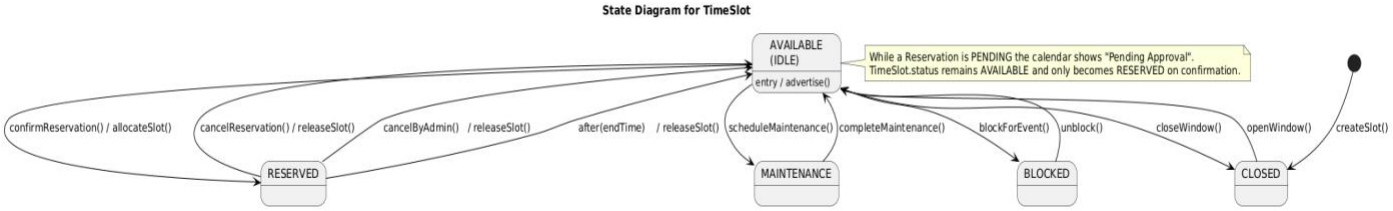

Figure 7 TimeSlot state diagram

6. Sequence Diagrams

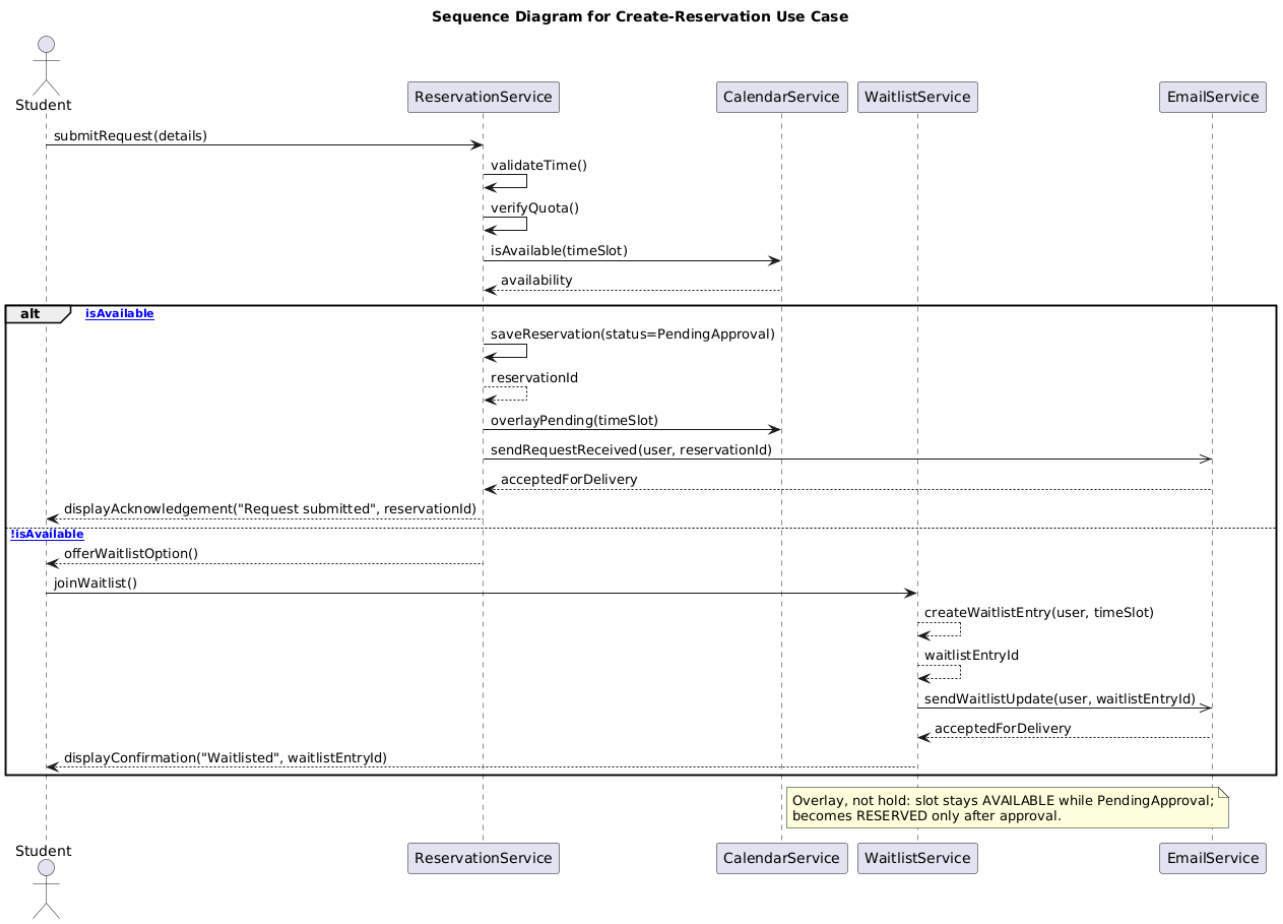

Figure 8 Create-Reservation sequence diagram

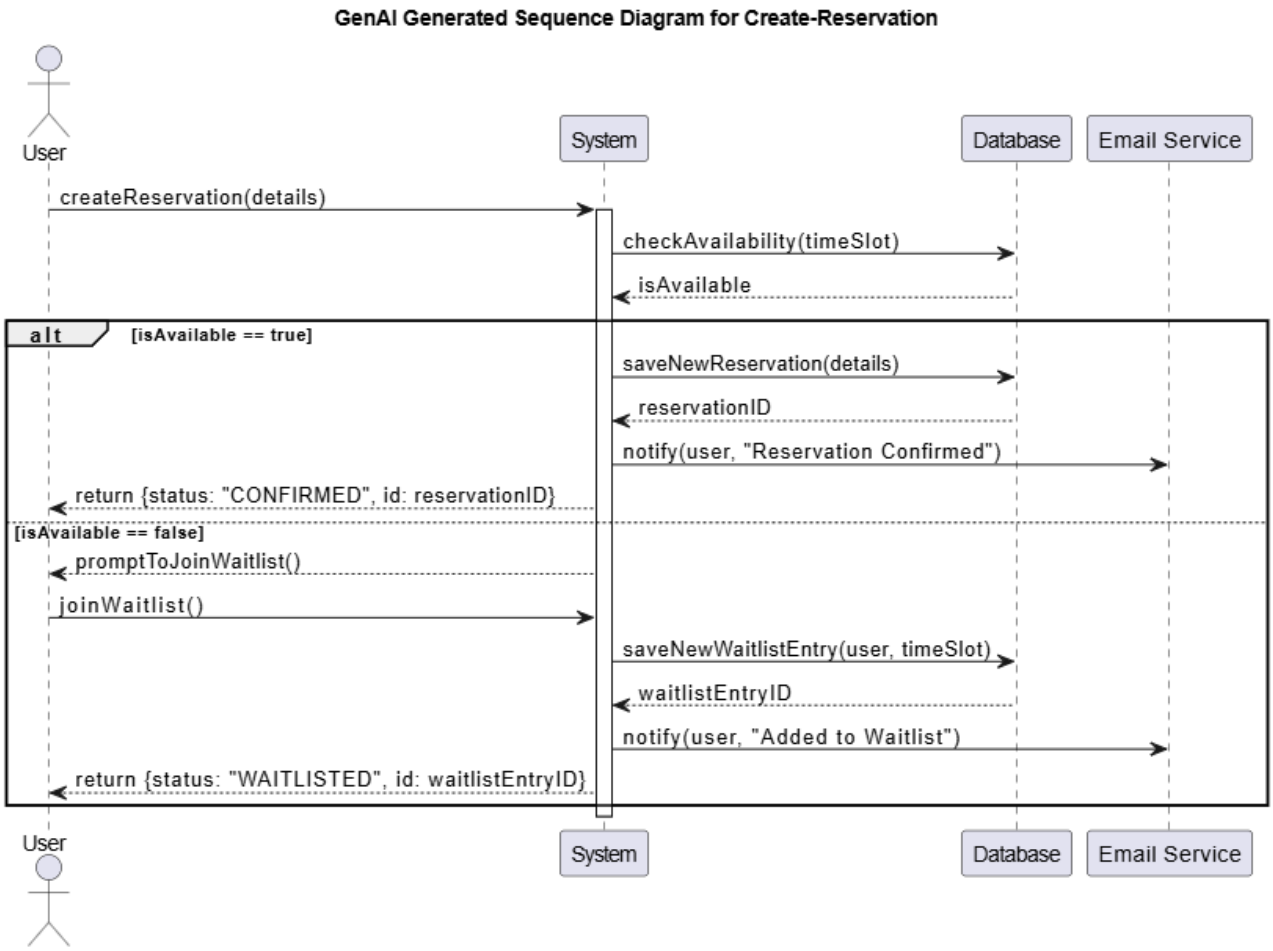

Figure 9 GenAI sequence diagram

Appendix C: Stakeholder Register

Stakeholder Name	Position	Internal/External	Contact	Category	Interest Level
John Smith	Student	Internal	john.smith@gmail.com	Operational	High
Mathew Nava	Faculty Member	Internal	mathew.nava@com.ca	Operational	High
Nath Bob	Lab Technician	Internal	nath.bob@comp.ca	Operational	High
Sara Von	Department Chair	Internal	sara.von@xcorp.com	Executive	High
Tom Harris	IT Security Officer	Internal	tom.harris@comp.ca	Executive	Low
Olivia Bennett	IT Administrator	Internal	olivia.bennett@comp.ca	Operational	Low

Appendix D: Interview Questions and Answers

Round 1 – John Smith (Student)

Question	Stakeholder Position	Answer
Which lab device do you book most often?	Student	I reserve the microscope several times a week.
How would you like to make a booking?	Student	I would use a web or mobile app with an interactive calendar.
What information must you see before you confirm a booking?	Student	I need date, time slots, and any usage fees.
How do you track your remaining usage quota today?	Student	I log my hours manually in a notebook.
How would you like to receive reminders for your bookings?	Student	I prefer an email reminder one hour beforehand.

Round 2 – Mathew Nava (Faculty Member)

Question	Stakeholder Position	Answer
Which equipment do you schedule for your classes?	Faculty Member	I book the 3D printer every Monday for my design labs.
How far in advance do you plan your bookings?	Faculty Member	I schedule labs at least two weeks ahead.
What approval process do you expect for student requests?	Faculty Member	I want a simple approve/reject button with notification.
Which reports help you adjust your course lab assignments?	Faculty Member	I review weekly usage summaries to spot peak times.
How should the system handle last-minute cancellations?	Faculty Member	It should free the slot immediately and notify waitlisted users.

Round 3 – Nath Bob (Lab Technician)

Question	Stakeholder Position	Answer
How do you review and approve booking requests now?	Lab Technician	I check email and update a booking spreadsheet.
What details do you need before approving maintenance work?	Lab Technician	I need the equipment ID and its recent usage history.
How would you like to handle equipment downtime in the future?	Lab Technician	I'd like the system to auto-block downtime slots, notify affected users, and suggest alternatives.
What alerts would help you identify equipment faults sooner?	Lab Technician	I need real-time error notifications from the device.
Which logs do you review each morning for your work?	Lab Technician	I look at the daily booking log and the maintenance log.

Round 4 – Sara Von (Department Chair)

Question	Stakeholder Position	Answer
How do you measure overall lab-resource utilization?	Department Chair	I compare monthly usage reports against department research needs.
Which high-level reports do you need to see?	Department Chair	I need monthly and quarterly summary reports.
How should bookings align with departmental priorities?	Department Chair	The system should give priority to critical research projects.
What data helps you justify new equipment purchases?	Department Chair	I need usage trends and peak demand metrics.
How often do you require summaries of lab usage?	Department Chair	I review usage at the end of each semester.

Round 5 – Tom Harris (IT Security Officer)

Question	Stakeholder Position	Answer
Which authentication protocols are mandatory?	IT Security Officer	We must support OAuth 2.0 and SAML for single-sign-on.
What encryption standards must we follow for stored data?	IT Security Officer	Data at rest must use AES-256 encryption.
How should security incidents be handled by the system?	IT Security Officer	It must log incidents immediately and alert the security team.
What access controls are required for admin functions?	IT Security Officer	Admin features must require multi-factor authentication.
Which compliance audits must the system support?	IT Security Officer	It must support annual internal and external security audits.

Round 6 – Olivia Bennett (IT Administrator)

Question	Stakeholder Position	Answer
What user roles must the system support?	IT Administrator	It needs roles for students, faculty, technicians, and admins.
What uptime level do you require?	IT Administrator	We need 99.9 percent availability.
Which audit logs must be retained, and for how long?	IT Administrator	All user and system events must be logged for one year.
How should backup and recovery be scheduled?	IT Administrator	We need daily full backups and hourly incremental backups.
What system configuration settings must you manage via the console?	IT Administrator	I must set usage quotas, manage user roles, and view health metrics.

Appendix E: Functional and Non-Functional Requirements

10 Functional Requirements:

ID	Title	Description	Priority	Requester
FR01	Equipment Calendar View	The system shall display a real-time calendar of all lab equipment availability, allowing users to see free and booked slots at a glance.	Expected	Student
FR02	Interactive Booking Interface	The system shall provide web and mobile booking screens with an interactive calendar picker for selecting equipment and time slots.	Expected	Student
FR03	Booking Request Submission	The system shall let users submit new booking requests by entering equipment, date, start/end time, and justification; requests are saved with status = Pending Approval and forwarded to Lab Technicians/Faculty for review (FR04).	Expected	Student

FR04	Approval & Rejection Workflow	The system shall allow lab technicians to approve or reject booking requests, enter comments, and trigger notifications on decision.	Exciting	Lab Technician, Faculty Member
FR05	Booking Modification & Cancellation	The system shall allow users to modify or cancel existing bookings within policy—cancellation permitted up to a configurable cutoff before start (default: 2 hours)—and immediately release slots for others.	Expected	Faculty Member
FR06	Waitlist Management	The system shall maintain a FIFO waitlist for fully booked slots and, when a slot opens, notify the next user and hold it for an acceptance window (default: 15 minutes) before auto-promoting the next user.	Exciting	Student
FR07	Quota Enforcement	The system shall enforce per-user and per-group time-quota rules, blocking requests that exceed weekly or monthly usage limits.	Expected	IT Administrator
FR08	Usage Reporting	The system shall generate weekly and monthly utilization reports by equipment and user group, and make them available to department chairs.	Expected	Department Chair
FR09	Notification Service	The system shall send email and in-	Expected	Student

		app alerts for booking confirmations, reminders (default: 1 hour before start), approvals, rejections, cancellations, and scheduled downtime.		
FR10	Role-Based Access Control	The system shall integrate with the university's OAuth 2.0 SSO to authenticate users and enforce role-based permissions for each user class.	Expected	IT Security Officer, IT Administrator

6 Non-Functional Requirements:

ID	Title	Description	Priority	Originator
NFR01	Performance	Performance: p95 user action ≤ 3 seconds and p99 ≤ 5 seconds at 200 concurrent users.	Expected	IT Administrator
NFR02	Availability	Availability: ≥99.9% per calendar month, excluding up to 2 hours of planned maintenance announced at least 48 hours in advance.	Expected	IT Administrator, Department Chair
NFR03	Security	Security: All in-flight data via TLS 1.3; authentication via OAuth 2.0 SSO; MFA required for admin functions; AES-256 encryption at rest; annual	Normal	IT Security Officer

		internal/external security audits. The application does not store or validate user passwords.		
NFR04	Scalability	Scalability: Support zero-downtime horizontal scaling to 500 concurrent users with ≤10% performance degradation after scaling.	Expected	IT Administrator
NFR05	Usability	New users shall be able to complete a standard equipment booking in under 3 minutes without formal training.	Normal	Student, Faculty Member
NFR06	Backup & Recovery	Backup & Recovery: Full backups daily and incremental backups hourly; RPO ≤ 1 hour; RTO ≤ 4 hours; quarterly restore tests; retain audit and booking logs for at least 1 year.	Expected	IT Administrator

Appendix F: To Be Determined List

TBD-01: University Identity Management (SSO) internal specification and integration guide (owner: Central IT).
TBD-02: Communications details — SMTP relay host/port/auth, allowed sender domains, and bounce handling.
TBD-03: User documentation inventory — PDFs, in-app help, and admin console manual; final formats and owners.
TBD-04: Policy constants — cancellation cutoff (default 2 hours), waitlist acceptance window (default 15 minutes), auto-approval rules.