Phase 3: Online Course Registration System – System Design

1. Introduction

This document presents the comprehensive object-oriented design of the **Online Course Registration System**, derived directly from the Phase 2 Analysis Model. All design artifacts—refined class diagrams, package structure, state machines, and UI navigation flows—are provided with full alignment to the use cases, conceptual classes, and behavioral models previously defined.

Key Design Principles Applied

- Separation of Concerns via modular packaging
- Encapsulation through class interfaces
- Extensibility using design patterns
- Robustness via state modeling
- Maintainability through clear navigability

---

2. Design Approach and Justification

2.1 From Analysis to Design

Analysis Model	Design Model
Conceptual classes	Design classes with attributes & data types
Services	Stateless service classes
Registration behavior	Transactional entity with state machine
External systems	SISAdapter via facade pattern

Key Refinements

- Added access modifiers
- Added data types
- Added method signatures
- Centralized validation in RegistrationEngine

2.2 Architectural Style

Layered Architecture:

`Presentation → Application Logic → Domain → Integration`

2.3 Design Patterns Used

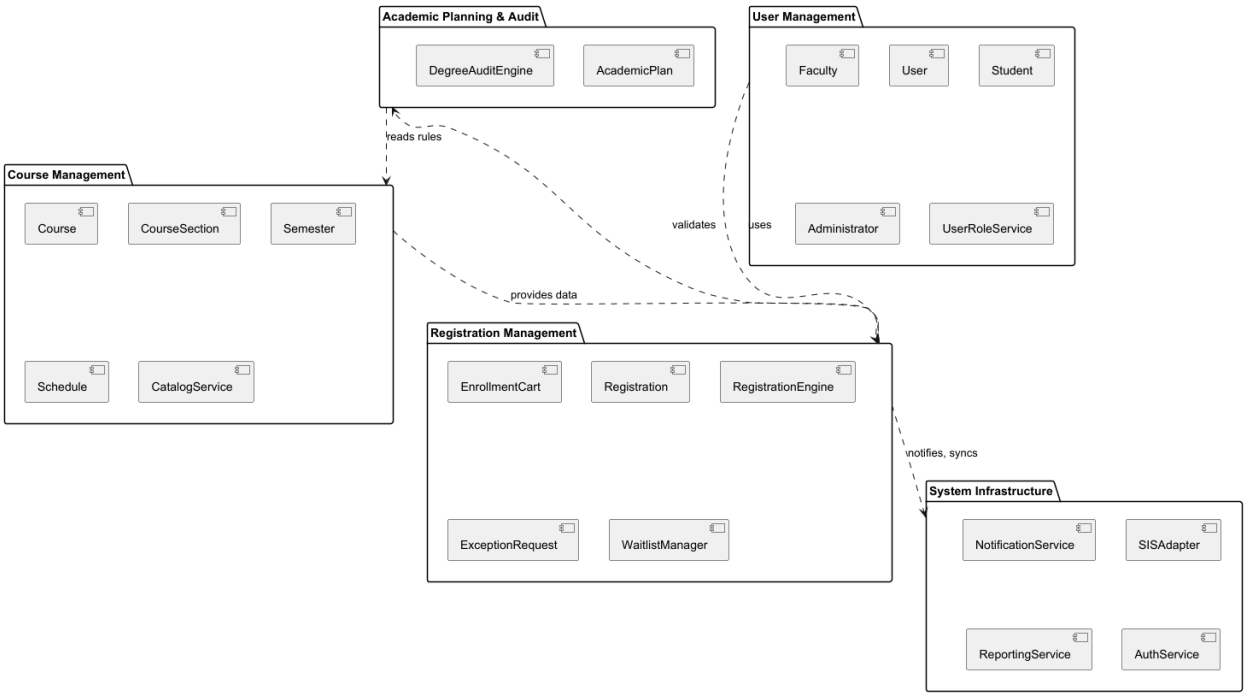
Pattern	Application	Reason
Factory	Creating registrations	Encapsulates creation
Observer	Notifications	Decoupling
Facade	SISAdapter	External integration
Strategy	Validation	Swappable rules
Singleton	Services	Single shared instance

---

3. Package/Component Diagram

```
@startuml
package "User Management" as UM {
 [User]
 [Student]
 [Faculty]
 [Administrator]
 [UserRoleService]
}

package "Course Management" as CM {
```

```
 [Course]
 [CourseSection]
 [Semester]
 [Schedule]
 [CatalogService]
}

package "Registration Management" as RM {
 [EnrollmentCart]
 [Registration]
 [RegistrationEngine]
 [ExceptionRequest]
 [WaitlistManager]
}

package "Academic Planning & Audit" as APA {
 [AcademicPlan]
 [DegreeAuditEngine]
}

package "System Infrastructure" as SI {
 [NotificationService]
 [SISAdapter]
 [ReportingService]
 [AuthService]
}

UM ..> RM : uses
CM ..> RM : provides data
RM ..> APA : validates
RM ..> SI : notifies, syncs
APA ..> CM : reads rules
@enduml
```



Package Responsibilities

Package	Responsibility
User Management	Authentication, roles
Course Management	Catalog, scheduling
Registration Management	Enrollment, waitlist, cart
Academic Planning & Audit	Plans, degree audits
System Infrastructure	Notifications, SIS sync, reporting

---

4. Refined Design Class Diagram

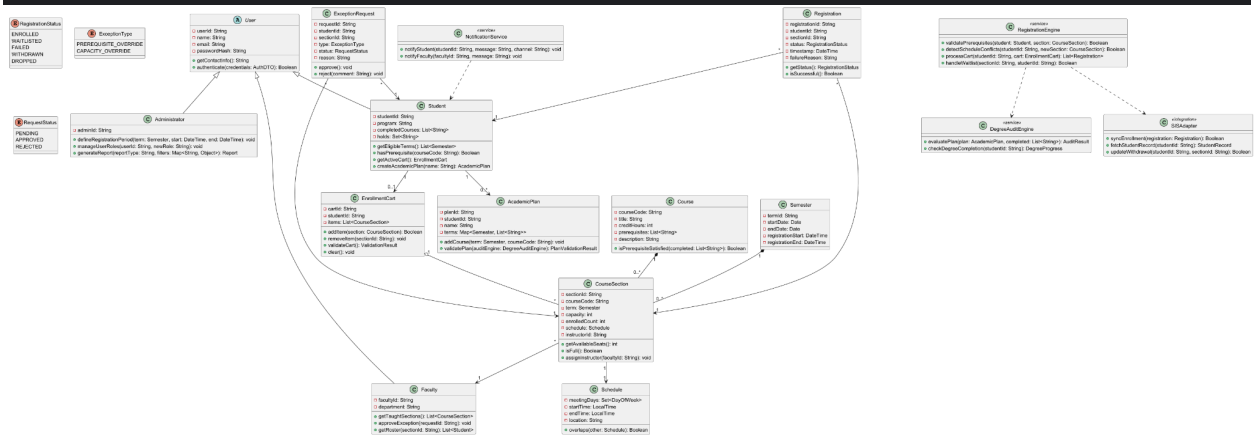
```
@startuml
' ================ USERS ====================
abstract class User {
 - userId: String
 - name: String
 - email: String
 - passwordHash: String
 + getContactInfo(): String
 + authenticate(credentials: AuthDTO): Boolean
}
```

```
class Student {
 - studentId: String
 - program: String
 - completedCourses: List<String>
 - holds: Set<String>
 + getEligibleTerms(): List<Semester>
 + hasPrerequisite(courseCode: String): Boolean
 + getActiveCart(): EnrollmentCart
 + createAcademicPlan(name: String): AcademicPlan
}

class Faculty {
 - facultyId: String
 - department: String
 + getTaughtSections(): List<CourseSection>
 + approveException(requestId: String): void
 + getRoster(sectionId: String): List<Student>
}

class Administrator {
 - adminId: String
 + defineRegistrationPeriod(term: Semester, start: DateTime, end: DateTime):
void
 + manageUserRoles(userId: String, newRole: String): void
 + generateReport(reportType: String, filters: Map<String, Object>): Report
}

User <|-- Student
User <|-- Faculty
User <|-- Administrator

' ================ COURSE ==================
class Course {
 - courseCode: String
 - title: String
 - creditHours: int
 - prerequisites: List<String>
 - description: String
 + isPrerequisiteSatisfied(completed: List<String>): Boolean
}

class CourseSection {
 - sectionId: String
 - courseCode: String
 - term: Semester
 - capacity: int
 - enrolledCount: int
 - schedule: Schedule
```

```
 - instructorId: String
 + getAvailableSeats(): int
 + isFull(): Boolean
 + assignInstructor(facultyId: String): void
}

class Semester {
 - termId: String
 - startDate: Date
 - endDate: Date
 - registrationStart: DateTime
 - registrationEnd: DateTime
}

class Schedule {
 - meetingDays: Set<DayOfWeek>
 - startTime: LocalTime
 - endTime: LocalTime
 - location: String
 + overlaps(other: Schedule): Boolean
}

Course "1" *-- "0..*" CourseSection
Semester "1" *-- "0..*" CourseSection
CourseSection "1" --> "1" Schedule
CourseSection "*" --> "1" Faculty

' ================ REGISTRATION ====================
class EnrollmentCart {
 - cartId: String
 - studentId: String
 - items: List<CourseSection>
 + addItem(section: CourseSection): Boolean
 + removeItem(sectionId: String): void
 + validateCart(): ValidationResult
 + clear(): void
}

class Registration {
 - registrationId: String
 - studentId: String
 - sectionId: String
 - status: RegistrationStatus
 - timestamp: DateTime
 - failureReason: String
 + getStatus(): RegistrationStatus
 + isSuccessful(): Boolean
}
```

```
enum RegistrationStatus {
 ENROLLED
 WAITLISTED
 FAILED
 WITHDRAWN
 DROPPED
}

Student "1" --> "0..1" EnrollmentCart
EnrollmentCart "1" o-- "*" CourseSection
Registration "*" --> "1" Student
Registration "*" --> "1" CourseSection

' ================ ACADEMIC PLANNING ====================
class AcademicPlan {
 - planId: String
 - studentId: String
 - name: String
 - terms: Map<Semester, List<String>>
 + addCourse(term: Semester, courseCode: String): void
 + validatePlan(auditEngine: DegreeAuditEngine): PlanValidationResult
}

class DegreeAuditEngine <<service>> {
 + evaluatePlan(plan: AcademicPlan, completed: List<String>): AuditResult
 + checkDegreeCompletion(studentId: String): DegreeProgress
}

Student "1" --> "0..*" AcademicPlan

' ================ EXCEPTION REQUESTS ====================
class ExceptionRequest {
 - requestId: String
 - studentId: String
 - sectionId: String
 - type: ExceptionType
 - status: RequestStatus
 - reason: String
 + approve(): void
 + reject(comment: String): void
}

enum ExceptionType {
 PREREQUISITE_OVERRIDE
 CAPACITY_OVERRIDE
}

enum RequestStatus {
 PENDING
```

```
 APPROVED
 REJECTED
}

ExceptionRequest "*" --> "1" Student
ExceptionRequest "*" --> "1" CourseSection

' ================ SERVICES ====================
class NotificationService <<service>> {
 + notifyStudent(studentId: String, message: String, channel: String): void
 + notifyFaculty(facultyId: String, message: String): void
}

class SISAdapter <<integration>> {
 + syncEnrollment(registration: Registration): Boolean
 + fetchStudentRecord(studentId: String): StudentRecord
 + updateWithdrawal(studentId: String, sectionId: String): Boolean
}

class RegistrationEngine <<service>> {
 + validatePrerequisites(student: Student, section: CourseSection): Boolean
 + detectScheduleConflicts(studentId: String, newSection: CourseSection):
Boolean
 + processCart(studentId: String, cart: EnrollmentCart): List<Registration>
 + handleWaitlist(sectionId: String, studentId: String): Boolean
}

RegistrationEngine ..> DegreeAuditEngine
RegistrationEngine ..> SISAdapter
NotificationService ..> Student
@enduml
```



Key Highlights

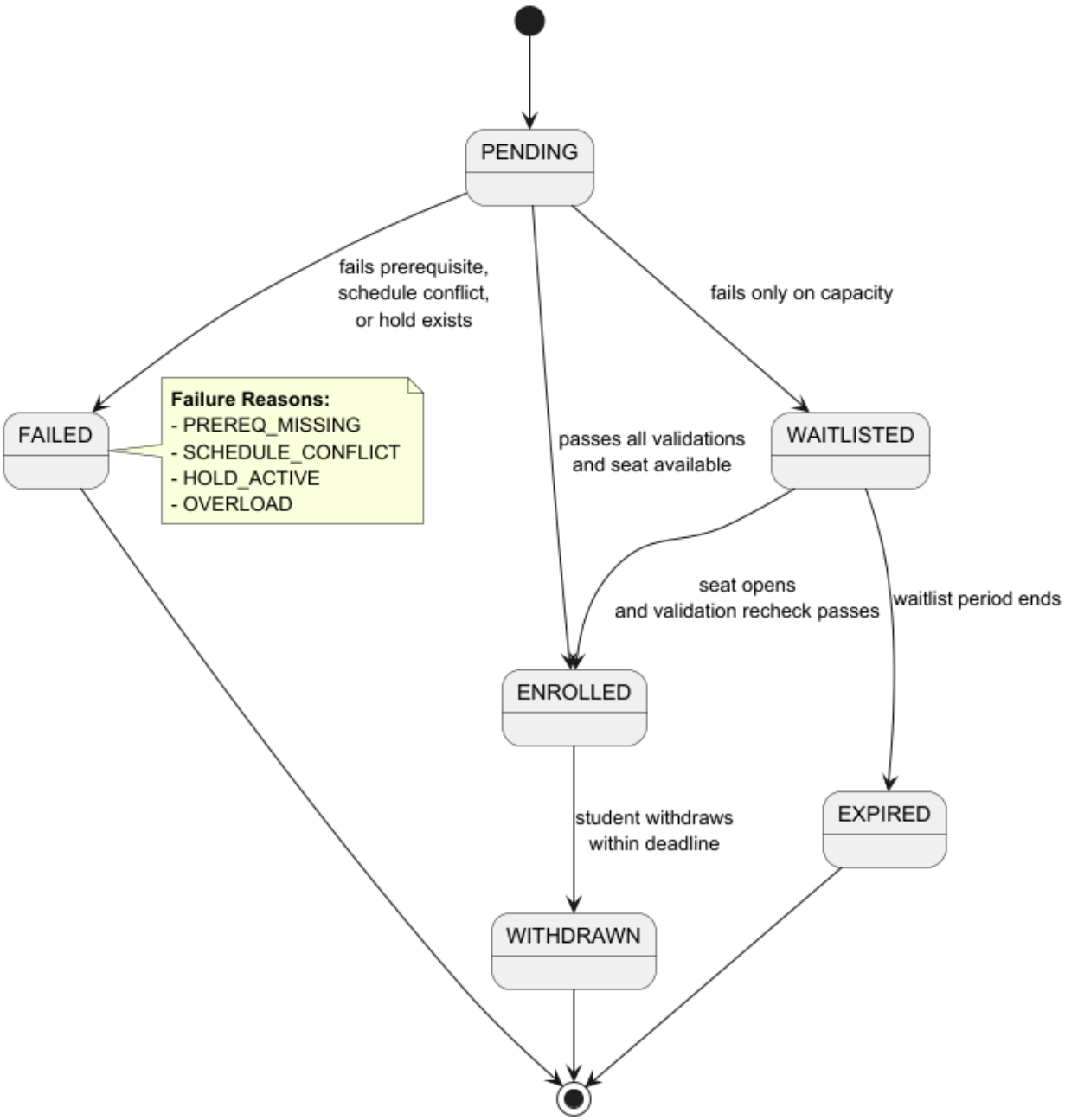
- Full multiplicities

- Navigability shown
- Real data types
- Access modifiers
- Polymorphism on users
- Services separated from domain
-

---

5. State Machine – Registration Entity

```
@startuml
[*] --> PENDING

state PENDING
state ENROLLED
state WAITLISTED
state FAILED
state WITHDRAWN
state EXPIRED

PENDING --> ENROLLED : passes all validations\nand seat available
PENDING --> WAITLISTED : fails only on capacity
PENDING --> FAILED : fails prerequisite,\nschedule conflict,\nor hold exists

ENROLLED --> WITHDRAWN : student withdraws\nwithin deadline

WAITLISTED --> ENROLLED : seat opens\nand validation recheck passes
WAITLISTED --> EXPIRED : waitlist period ends

FAILED --> [*]
WITHDRAWN --> [*]
EXPIRED --> [*]

note right of FAILED
 **Failure Reasons:**
 - PREREQ_MISSING
 - SCHEDULE_CONFLICT
 - HOLD_ACTIVE
 - OVERLOAD
end note
@enduml
```



State Transition Table

From	To	Trigger	Action
Pending	Enrolled	Valid	Notify + Sync
Pending	Waitlisted	Capacity issue	Add to queue
Pending	Failed	Validation fail	Log reason

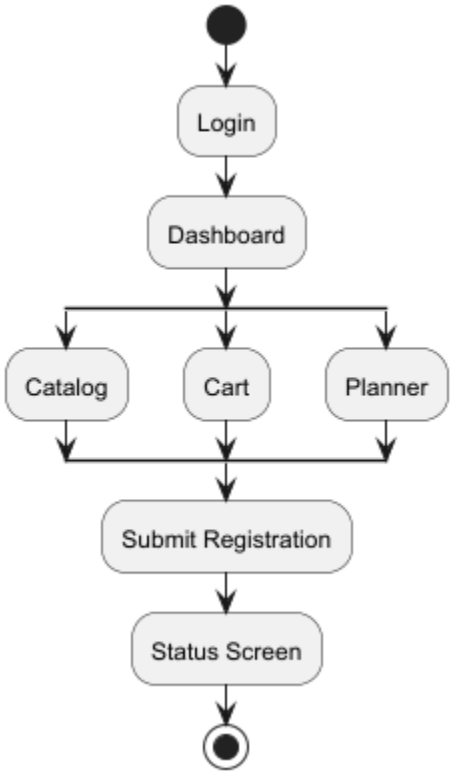
Enrolled	Withdrawn	Student withdraws	Update
Waitlisted	Enrolled	Seat opens	Revalidate

6. UI Mockups & Navigation Flow

Main Screens

- Dashboard
- Catalog Browser
- Enrollment Cart
- Academic Planner
- Faculty Exception Queue

Navigation



7. Summary of Design Decisions

Architectural Choices

- Layered architecture
- Service-based design
- State modeling
- Facade for SIS

Rationale

Decision	Reason
Centralizing validation	Consistent + testable
Registration entity	Audit trail
Notification service	Loose coupling
Package modularity	Maintainability

Extensibility

- Add validation rules easily
- Add notification channels
- Add report types
- Extend exception types

---

8. Traceability to Phase 2

Use Cases → Design Components

Use Case	Design Component

Register	RegistrationEngine
Browse catalog	CatalogService
Plan degree	AcademicPlan
Approve exception	Faculty + ExceptionRequest

Analysis Model → Design Model

- All conceptual classes refined
- Methods added
- Data types assigned
- Multiplicities fully defined

Conclusion

This document provides a fully detailed, implementation-ready object-oriented design for the Online Course Registration System. It is aligned with Phase 2 and ready for Phase 4 implementation planning.