**Software Architecture Document**

**TafeSA Online Enrolment System**

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Author** |
| **04/03/2025** | **1.0** | **Mau** |

# Section A – Stakeholder Identification/Communication Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Stakeholder** | **Description** | **Communication Strategies (Digital vs Non Digital)** | **Comments** |
| Project Lead | Oversees project, validates requirements, and provides feedback. | Online meetings, email reports, project management tools (Jira, Trello) | Regular check-ins to review progress and validate system requirements. |
| Development Team | Analysts and developers responsible for implementing system requirements | Daily standups (virtual or in-office), GitHub repository updates, Slack communication. | Coordination is crucial to ensure UML models align with system specifications. |
| TAFE SA Rep | Client representatives ensuring system meets business and compliance needs. | Formal presentations, email updates, in-person review meetings. | Ensures legal, compliance, and business needs are met |
| End User (Students and Teachers) | Primary users of the system, interacting with enrolment and course selection. | User surveys, training sessions, online help desk, feedback forms | Feedback is essential for usability and functional improvements. |
| IT Works Management | |  | | --- | |  |  |  | | --- | | Company stakeholders overseeing project alignment with organizational goals. | | Periodic project reports, strategy meetings, compliance documentation. | Approval required at key milestones for project continuation. |

# Section B - Determine/Design the Business Model and Architecture

## Business Model and Architectural Impact

The TafeSA Course Enrolment Management System is designed to support an online self-service model that enables students to register for courses, view report cards, and manage schedules, while registrars and lecturers can perform administrative tasks such as closing registrations and choosing courses to teach. A decentralized access model, role-based interactions, and automated back-end operations such as billing and course validation are supported by this business model. In addition to handling concurrent users securely and efficiently, the system also must interface with external services like billing. In order to ensure scalability, security, and maintainability, this influences the architecture towards a multi-tiered, service-oriented architecture. In order to ensure business continuity, the architecture must support real-time validation, session management, robust error handling, and fallback mechanisms (e.g. when the billing system is unavailable).

## Object-Oriented Design Principles and MVC Influence

The system architecture is heavily influenced by object-oriented principles such as encapsulation, abstraction, inheritance, and polymorphism. These principles support modularity, reuse, and separation of concerns, which are crucial for maintaining complex systems. The system adopts the Model-View-Controller (MVC) design pattern, which clearly separates data (Model), user interface (View), and control logic (Controller). This separation facilitates team development, unit testing, and future scalability. For example, the "Register for Courses" use case encapsulates course and student data in Models, while Controllers handle the registration logic, and Views provide role-specific interfaces. MVC also simplifies integration with web technologies and third-party services, supporting a clean and extensible system design.

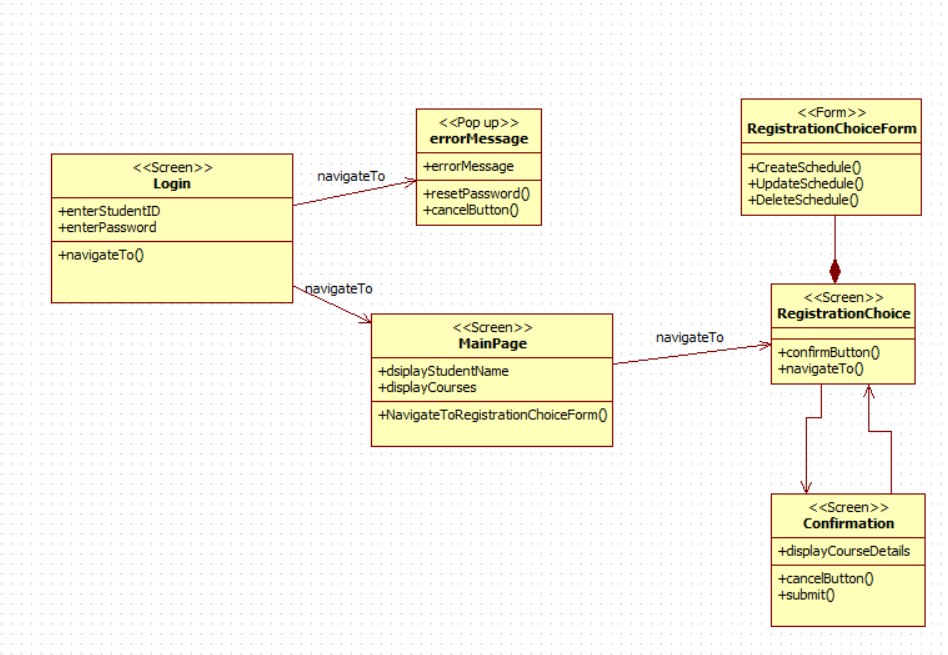
## Domain Model

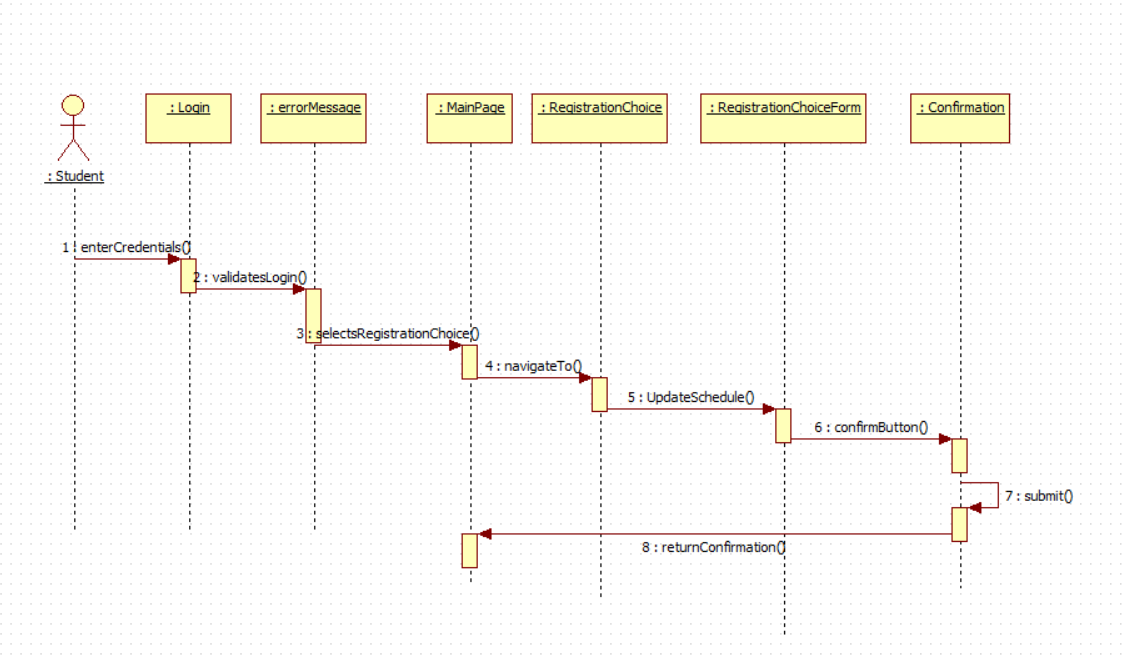
A diagram of a course

AI-generated content may be incorrect.

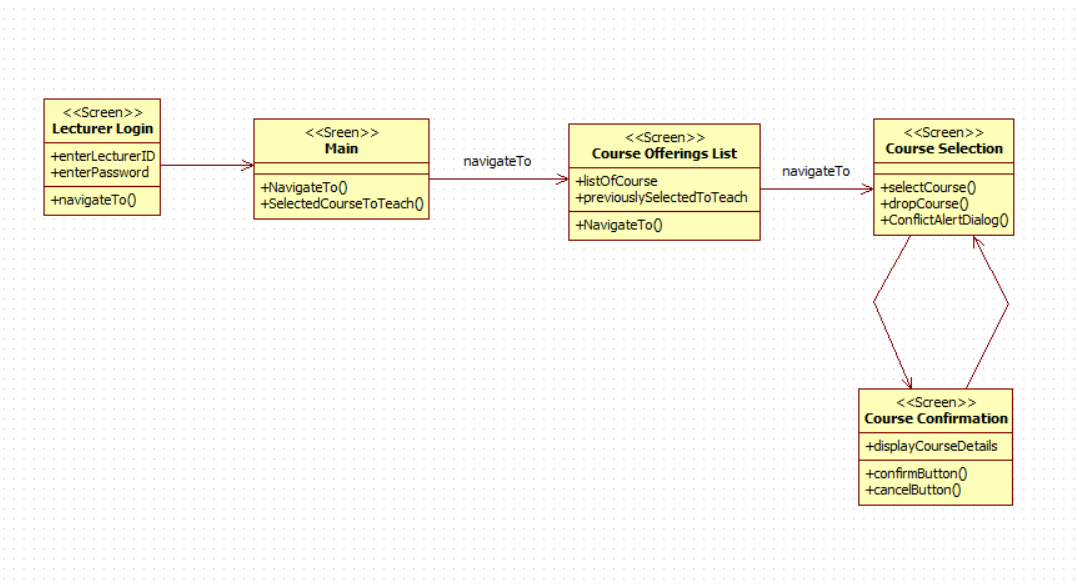
# Section C - Create the user experience model for above Use Cases

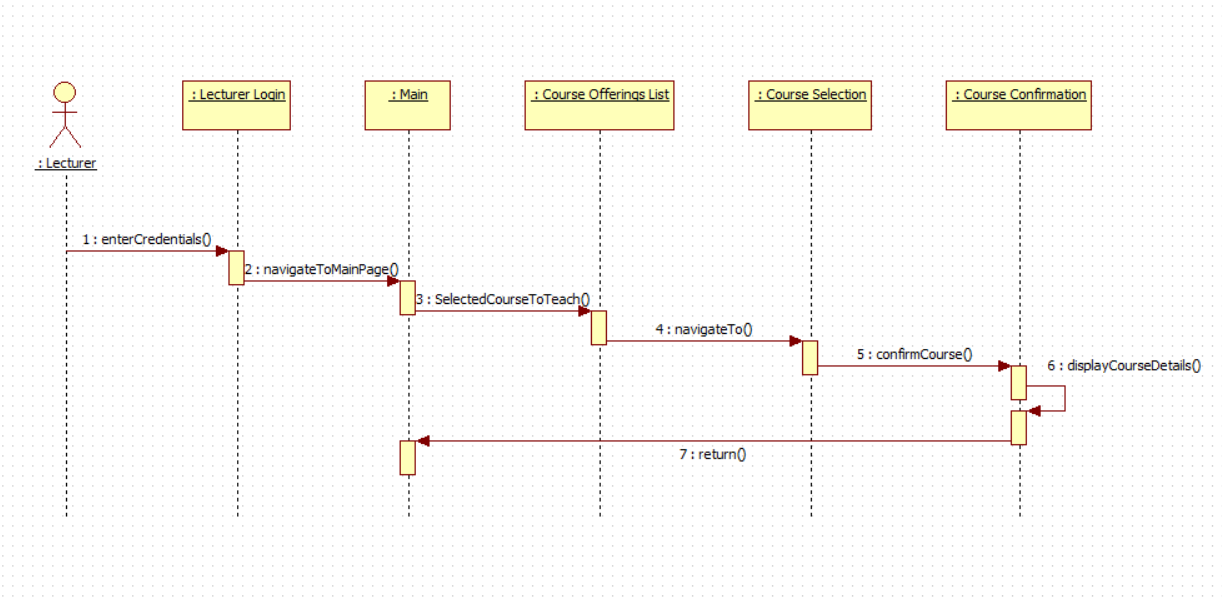
## Register For Courses UX Model

**

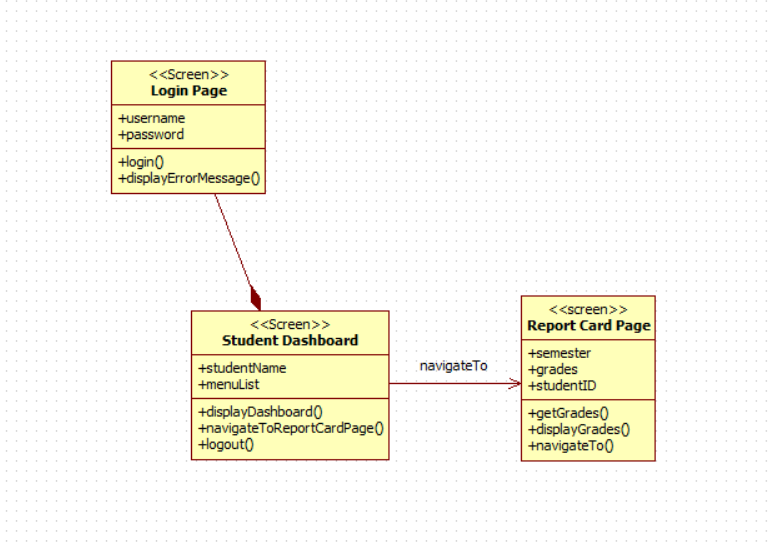
**

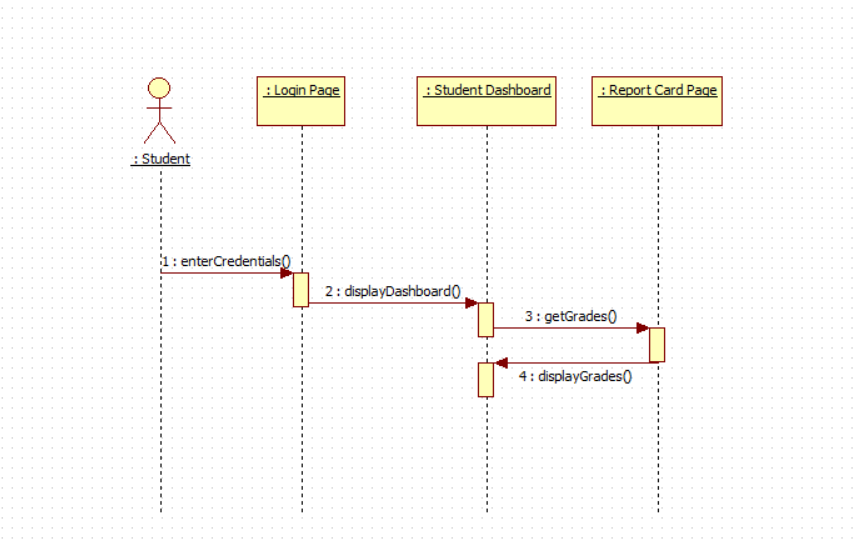
## Select Courses to Teach UX Model

**

**

## View Report Card UX Model



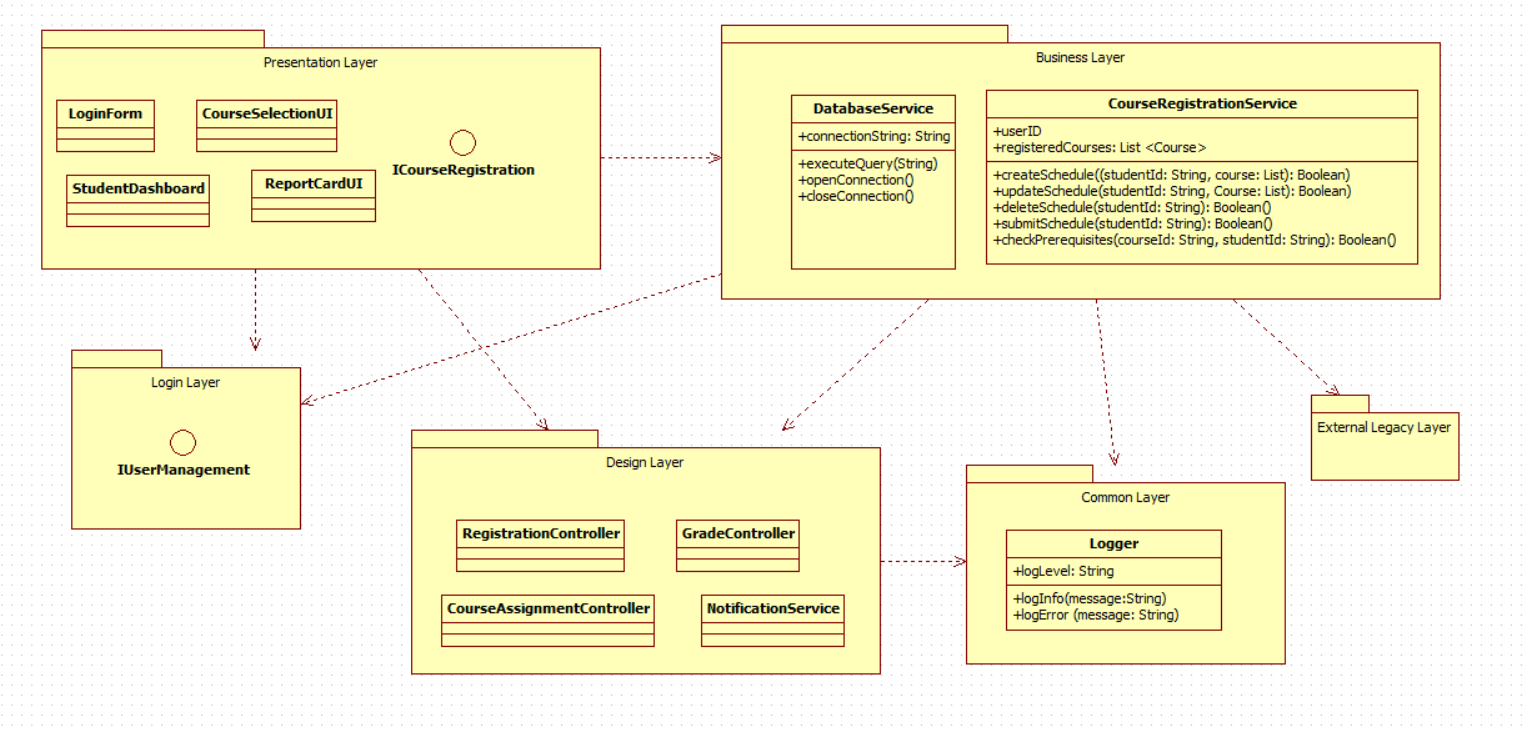


# Section D – Design and Implementation Mechanisms (Done by Individual)

|  |  |  |  |
| --- | --- | --- | --- |
| **Analysis Mechanisms** | **Design Mechanisms** | **Implementation Mechanism** | **Justification for choice and influence of Organizational procedures and standards** |
| **Persistence** | MongoDB | Use MongoDB, a NoSQL database, to store course catalogue and registration data. | Preferred for its schema flexibility and scalability, aligning with standards for distributed campuses. |
| **External System interaction** | Azure Blob Storage | Use Azure Blob Storage for storing static content like course materials. | Supports cost-effective, cloud-based file storage suitable for remote operations. |
| **Legacy Systems interaction** | REST APIs | Use REST APIs for integration with external systems like the course catalogue. | Follows distributed system standards for modular and interoperable services. |
| **Cultural Specific and Worldwide considerations (Onshore and Offshore partner campus in Vietnam) and Distribution** | OAuth 2.0 | Utilize OAuth 2.0 for identity authentication from external providers. | Meets security policies and reduces overhead for custom authentication management. |
| **System Parameter Management** | ETL Tools | Use ETL (Extract, Transform, Load) tools to migrate data from legacy databases. | Supports seamless transition to modern infrastructure while preserving data integrity. |
| **Authentication** | Middleware Integration | Introduce middleware to connect legacy components with new modules. | Facilitates phased upgrades compliant with rollout standards. |
| **Authorization** | Multi-Language Support | Implement i18n libraries for multilingual UI localization. | Addresses inclusivity goals across Vietnam and global campuses. |
| **Transaction Management** | Azure CDN | Use Azure CDN to distribute course content globally. | Ensures fast and reliable content access across offshore locations. |
| **Error Handling** | Configuration Files | Store server paths and thresholds in config files. | Follows practice of separating business logic from configuration. |
| **Concurrency** | Environment Variables | Use environment variables to manage sensitive parameters. | Protects credentials and adheres to best practices in data security. |
| **User Interface/User Experience** | Multi-Factor Authentication (MFA) | Implement MFA using email or SMS verification. | Adds a security layer consistent with organizational access policies. |
| **Security** | Single Sign-On (SSO) | Integrate Azure Active Directory for centralized login. | Enhances user convenience and supports centralized control. |

# Section E– Identify Design Elements and interconnecting Components

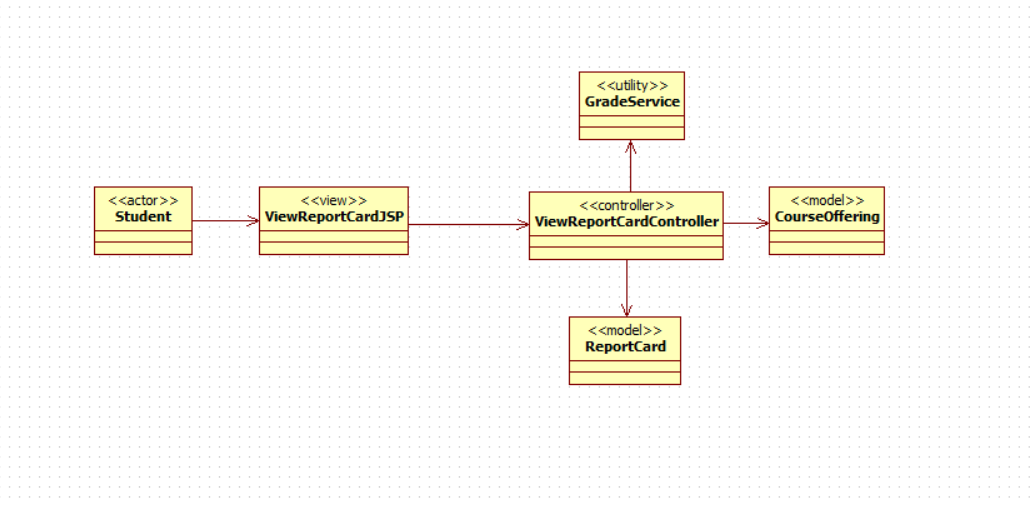
## Design/Component UML Model

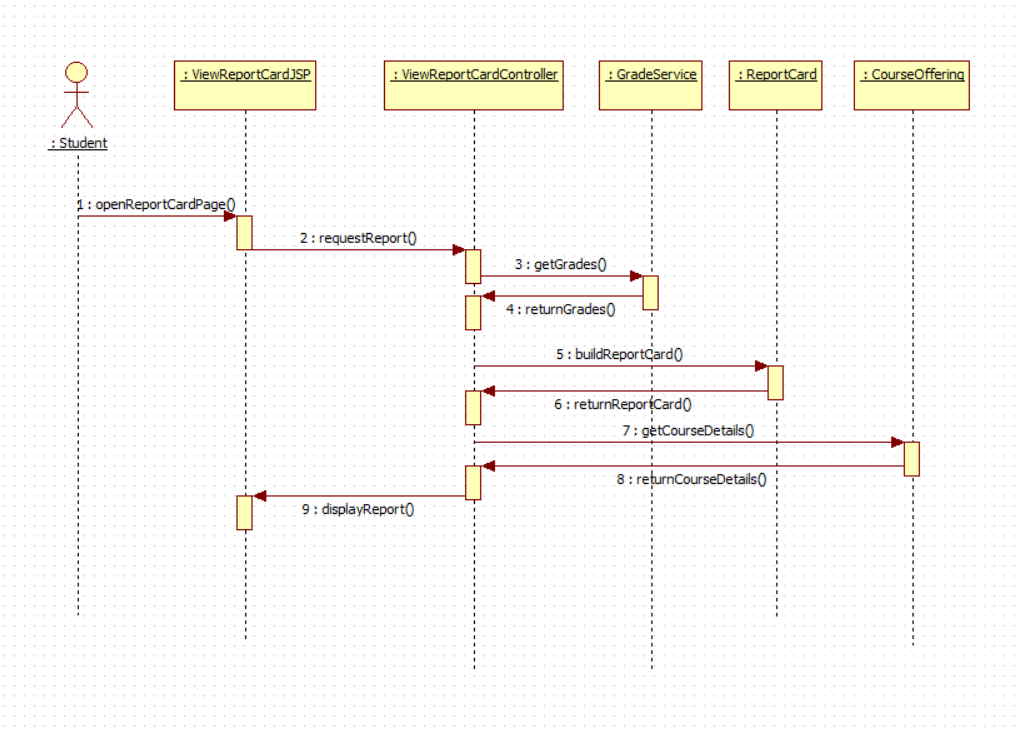
**

# Section F –Model the use case realization (Done by Individual)

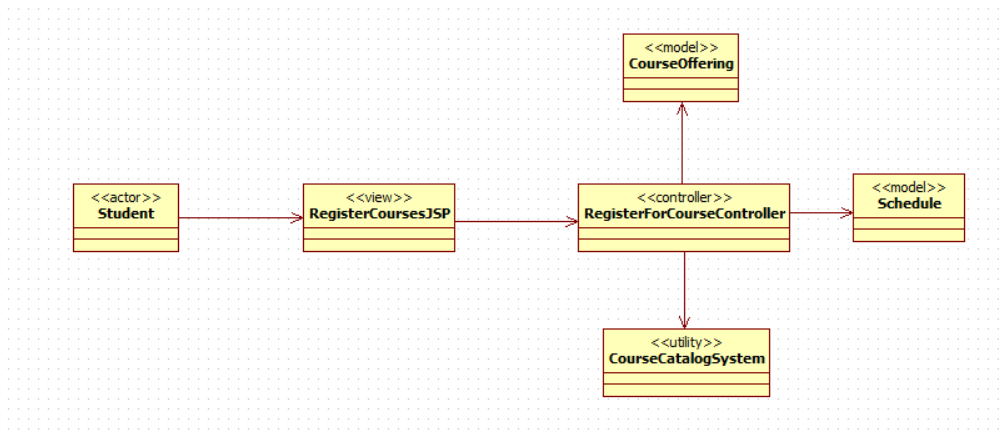
## Use Case Realizations UML Models

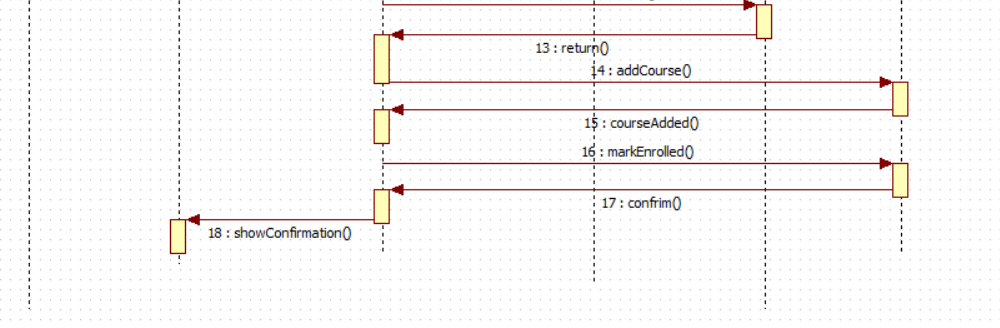
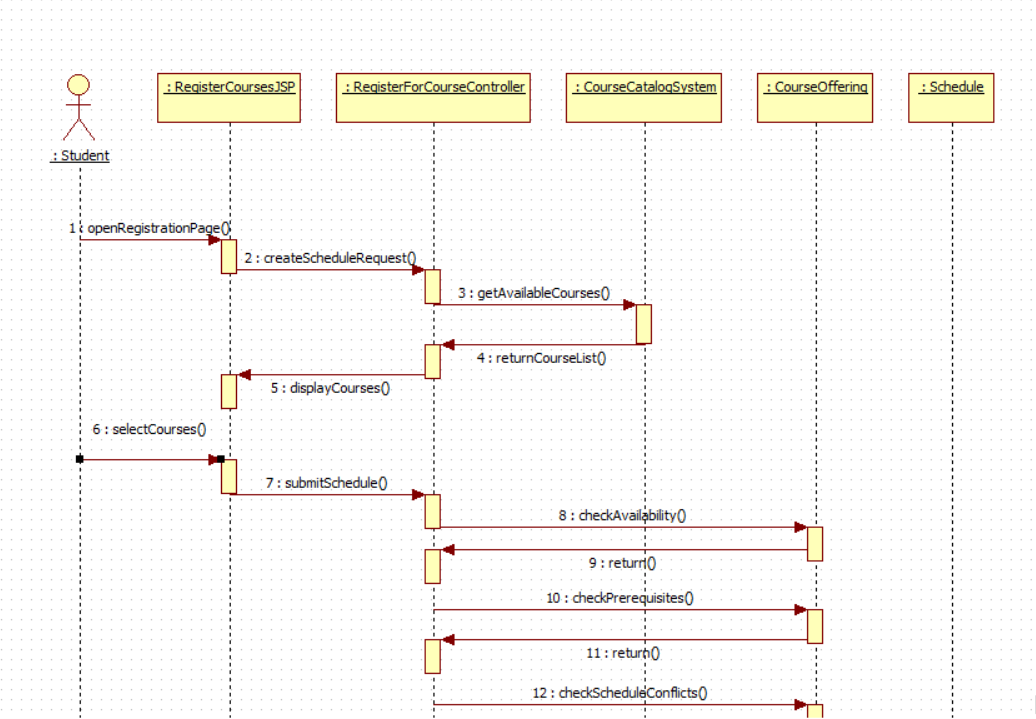
## VOPC ViewReportCard





## VOPC RegisterForCourses

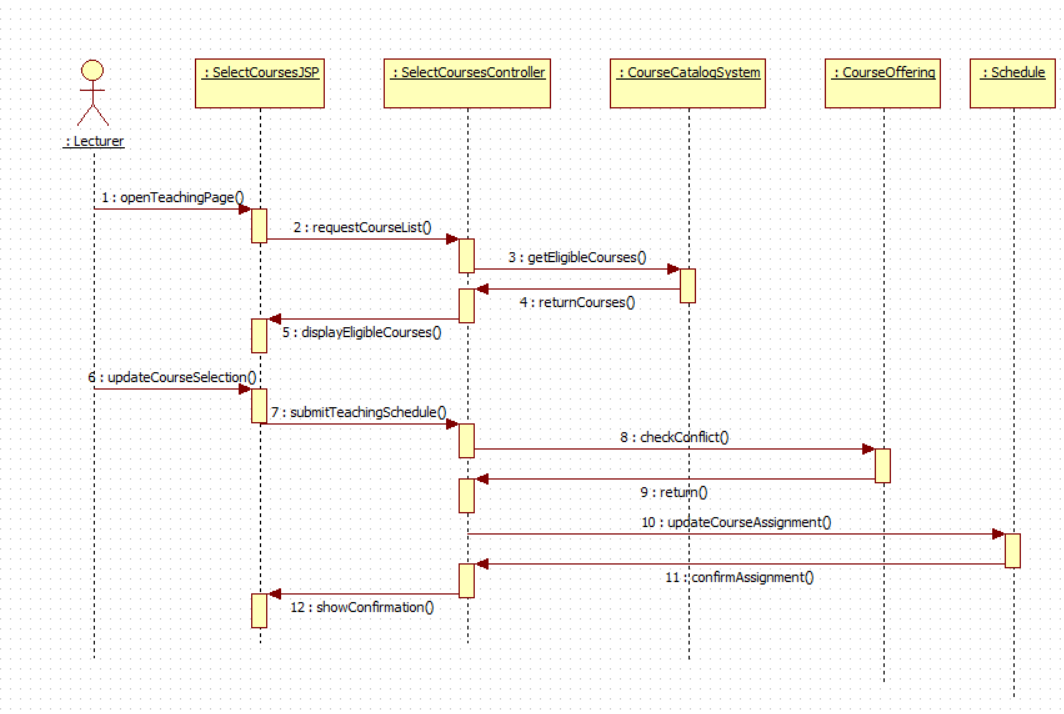




## VOPC Select Courses to Teach

A diagram of a computer program

AI-generated content may be incorrect.



# Section G – Class Design (Done by Individual)

*A diagram of a computer

AI-generated content may be incorrect.*

# Section H – Database Design

*A screenshot of a computer

AI-generated content may be incorrect.*

# Section I – Supplementary Specifications (Non-Functional Requirements)

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Implementation Strategy | How is it measured | Benchmark Value |
| Functionality | Ensure all core features (registration, report cards, schedules) are implemented and tested. | Functional testing against use cases. | All use cases tested and passed. |
| Usability | Design user-friendly interfaces with localization support for English and Vietnamese. | Conduct user testing with feedback surveys. | 90% positive feedback on usability. |
| Reliability | Implement fault tolerance and retry mechanisms for critical operations. | Monitor system logs for failures and downtime. | 99.9% uptime during testing. |
| Performance | Use caching, indexing, and optimized queries to reduce response times. | Load testing with tools like JMeter or LoadRunner. | Average response time ≤ 3 seconds. |
| Scalability | Deploy on a scalable cloud platform with load balancing. | Simulate concurrent users with increasing traffic. | Support up to 10,000 concurrent users. |
| Re-usability | Use modular design with reusable components in business logic and UI layers. | Track reused modules across different system components. | 50% of components reused across modules. |
| Testability | Automate unit and integration testing using testing frameworks (e.g., Selenium, NUnit). | Code coverage and defect density during automated testing. | 90% code coverage with ≤ 5 defects/module. |
| Security | Apply AES-256 encryption for sensitive data and role-based access control (RBAC). | Penetration testing and security audits. | No critical vulnerabilities detected. |
| Adaptability to different locations (Onshore and Offshore site in Vietnam) | Implement dynamic content for localization and timezone support. | User testing across regions and environments. | Compatible with onshore and offshore sites. |
| Copyright and Intellectual property protection | Use licensing tools and watermarks for digital resources. | Review against copyright policies and intellectual property compliance. | 100% compliance with organizational standards. |
| Pre-release Testing (Refer to Section 2.1.4 of the Organizational Standards Document) | Conduct alpha and beta testing phases per organizational standards. | Evaluate test results and bug reports. | ≤ 5 critical bugs reported before release. |

# Section J – Deployment

|  |  |  |
| --- | --- | --- |
| **Hardware Requirements** | **Software Requirements** | **Notes** |
| Client Machine (User PC) | Web Browser (e.g., Chrome, Edge, Firefox) | For students, lecturers, and registrars accessing the application. |
| Developer Workstation | Visual Studio / VS Code, StarUML, Postman | For coding, UML modelling, and API testing during development. |
| Application Server (Test) | Node.js / Java EE / .NET Core Runtime | Hosts the backend during testing; could be a local server or VM. |
| Application Server (Prod) | Azure App Service / AWS EC2 instance | Hosts production backend services. |
| Database Server (Test) | MongoDB (local or cloud instance) | Stores course catalogue, user data, registration data during testing. |
| Database Server (Prod) | Azure Cosmos DB / MongoDB Atlas | Cloud-based production database for scalability and reliability. |
| Authentication Server (Opt.) | OAuth 2.0 Identity Provider (Azure AD, Auth0) | Used if the system integrates with an external authentication provider. |
| Storage Server | Azure Blob Storage / AWS S3 | Stores course materials and large files. |

*A diagram of a computer

AI-generated content may be incorrect.*

# Section K – Analyse and Document the Impact of the new system

## Existing Processes Affected by the New System

**The introduction of the new course registration and management system will significantly change several current manual or outdated digital processes. The following existing processes will be directly impacted:**

|  |  |  |
| --- | --- | --- |
| **Existing Process** | | **Impact of the New System** |
| **Manual course registration** | **Replaced with an automated online registration system using secure login and course selection interface** | |
| **Paper-based schedule planning** | **Replaced with real-time online scheduling with conflict detection** | |
| **Grade reporting through admin** | **Automated digital report cards accessible directly by students via a secure portal** | |
| **Lecturer allocation via spreadsheets** | **Replaced by dynamic selection of courses to teach, integrated into the academic dashboard** | |
| **In-person student support for registration issues** | **Reduced through user-facing error messages and form validation in the JSP views** | |

## Three Reasons the New System Adds Business Value

|  |  |
| --- | --- |
| **Reason** | **Explanation** |
| **Increased Efficiency** | **Automating course registration, scheduling, and grading reduces administrative workload, allowing faster processing and fewer manual errors.** |
| **Improved User Experience** | **Students and lecturers have a centralised, user-friendly portal to manage their tasks, improving engagement and satisfaction.** |
| **Data Accuracy & Real-Time Access** | **Integration with a central database ensures that updates (like course availability or grade entries) are reflected instantly across the system.** |

## 3. Training Needs Analysis

**The new system introduces functionalities and interfaces that will require training for multiple stakeholders. The table below outlines the training gaps and the corresponding training requirements:**

|  |  |  |
| --- | --- | --- |
| **Stakeholder Group** | **Identified Training Gaps** | **Training Requirements** |
| **Students** | **Unfamiliarity with the self-registration process and report card access** | **Guided tutorials, onboarding emails, FAQ sections in the portal** |
| **Lecturers** | **Lack of experience using online course selection and grade entry** | **Live training sessions, video walkthroughs, role-based manuals** |
| **Administrative Staff** | **Limited exposure to back-end scheduling and database interactions** | **Hands-on workshops covering course catalogue updates, user management, and support ticket handling** |
| **IT Support Team** | **Need to troubleshoot new MVC-based system with security layers and validators** | **Technical training on system architecture, error logging, and middleware integration** |
| **Documentation Specialist** | **Understanding of interface features to create accurate user guides** | **System walk-throughs and document templates for process documentation** |

# Section L – Work Breakdown Structure (WBS), Gantt Charts and Project metrics/costs calculations

## Work Breakdown Structure (WBS)

**The Work Breakdown Structure (WBS) is a structured decomposition of the project into smaller, manageable tasks and sub-tasks. It outlines the activities to be completed and identifies the roles and resources required to carry them out.**

## Resource Roles & Costs:

Here are five typical roles and their associated assumptions for the project:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Role | Number of Personnel | Cost per Hour ($) | Usage Timeline (Hours) | Total Cost ($) |
| System Architect | 2 | 80 | 120 | 19200 |
| Analyst/Designer | 2 | 60 | 150 | 18000 |
| Tester | 3 | 50 | 200 | 30000 |
| Database Designer | 2 | 70 | 100 | 14000 |
| Documentation Specialist | 1 | 40 | 80 | 3200 |

### Work Breakdown Structure (WBS)

The project tasks are divided into clearly defined phases, each consisting of specific deliverables. This hierarchical WBS structure helps allocate resources, track progress, and manage dependencies effectively.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WBS Code | Task Description | Assigned Role(s) | Duration | Dependencies |
| 1 | Project Initiation & Planning | Project Manager, System Architect | 1 week | - |
| 1.1 | Define project scope and objectives | System Architect | 3 days | - |
| 1.2 | Identify required resources and timelines | Project Manager | 2 days | 1.1 |
| 2 | Requirements & Analysis | Analyst/Designer | 2 weeks | 1.2 |
| 2.1 | Gather functional & non-functional requirements | Analyst/Designer | 1 week | 2 |
| 2.2 | Document use cases and business rules | Analyst/Designer | 1 week | 2.1 |
| 3 | System Design & Modeling | System Architect, Database Designer | 3 weeks | 2.2 |
| 3.1 | Design class diagram and database schema | System Architect, Database Designer | 2 weeks | 3 |
| 3.2 | Create UI mockups and sequence diagrams | Analyst/Designer | 1 week | 3.1 |
| 4 | Implementation | Developer, Analyst/Designer | 4 weeks | 3.2 |
| 4.1 | Develop core modules and backend logic | Developer | 2 weeks | 4 |
| 4.2 | Implement JSP views and controllers | Developer | 2 weeks | 4.1 |
| 5 | Testing & Integration | Tester | 2 weeks | 4.2 |
| 5.1 | Unit and integration testing | Tester | 1 week | 5 |
| 5.2 | User acceptance testing | Tester, Analyst/Designer | 1 week | 5.1 |
| 6 | Documentation & Review | Documentation Specialist | 1 week | 5.2 |
| 6.1 | Create and finalise system documentation | Documentation Specialist | 3 days | 6 |
| 7 | Deployment & Post-Support | All Roles | 2 weeks | 6.1 |
| 7.1 | Deploy system to production environment | System Architect, Developer | 1 week | 7 |
| 7.2 | Monitor and fix post-deployment issues | Developer, Tester | 1 week | 7.1 |

## Gantt Chart

**A screenshot of a computer screen

AI-generated content may be incorrect.**

## Project Costing

## 

For tracking project costs, you will need to measure the following:  
• Actual Costs (AC): The actual costs incurred in performing project activities.  
• Cost Performance Index (CPI): This ratio (EV/AC) helps measure cost efficiency.  
• Cost Variance (CV): This difference (EV - AC) shows if the project is under or over budget.

The table below summarizes project costing for each major phase:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task | Planned Cost | Actual Cost | Variance | CPI | CV |
| Initiation & Planning | $5,000 | $5,200 | $200 | 0.96 | -$200 |
| Requirements & Analysis | $7,000 | $6,800 | -$200 | 1.03 | $200 |
| System Design & Modeling | $10,000 | $10,200 | $200 | 0.98 | -$200 |
| Implementation Phase | $15,000 | $14,900 | -$100 | 1.01 | $100 |
| Testing & QA | $6,000 | $6,100 | $100 | 0.98 | -$100 |
| Documentation & Review | $3,000 | $2,900 | -$100 | 1.03 | $100 |
| Deployment & Post-Support | $4,000 | $3,900 | -$100 | 1.03 | $100 |

**Section M – Verification/Validation and Signoff**

*Please tick-off Sections A-L completed*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Use Case** | **A** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **Project Lead Comments/Signoff** |
| View Report Card | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Mau Patel |
| Select Courses to Teach | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Checkmark outline | Mau Patel |

# Section N – Post Project Analysis

## 1. Software Development Methodologies

Below is a comparison of three software development methodologies commonly used in the industry:

|  |  |  |  |
| --- | --- | --- | --- |
| Methodology | Description | Advantages | Disadvantages |
| Waterfall | Linear and sequential approach with clearly defined phases. | Simple to manage; clear documentation and milestones. | Inflexible to change; late discovery of issues. |
| Agile | Iterative and incremental model focused on collaboration and rapid delivery. | Flexible; continuous feedback; faster releases. | Requires high user involvement; less predictability. |
| DevOps | Integrates development and operations to enhance automation and delivery. | Faster deployment; improved collaboration and CI/CD. | Needs strong infrastructure setup; cultural shift required. |

## 2. Recommended Methodology: Agile

For a system like the online course registration and academic management portal, Agile is the most suitable methodology. Agile supports iterative development and frequent feedback, which allows rapid delivery of components like registration, login, and reporting features.  
  
In this project, Agile would follow these SDLC phases:  
- Requirements gathered in user stories and prioritized.  
- Design and prototype screens (e.g., JSPs) iteratively.  
- Development occurs in sprints, allowing feedback.  
- Testing integrated early using automated unit tests.  
- Deployment and documentation reviewed every sprint.  
- Adjustments based on stakeholder feedback before final release.

## 3. Policy & Procedure Improvement Suggestions

• Technical Suggestions:  
1. Implement continuous integration and automated testing in the software pipeline.  
2. Enforce standard code review and security scanning practices.  
  
• Non-Technical Suggestions:  
1. Introduce Agile-based training for project stakeholders and developers.  
2. Enhance communication by using a centralized platform for updates and decisions.

## Impact on Supply Chain and Mitigation Strategies

|  |  |
| --- | --- |
| Risk | Analysis/Improvements/Mitigation Strategies |
| Hardware supply delay for scaling infrastructure | Use cloud infrastructure like Azure or AWS to avoid physical hardware dependency. |
| Software licensing conflicts or versioning issues | Adopt open-source tools where feasible; maintain version control policies. |
| Shortage of skilled Agile developers | Partner with training providers and recruit interns to build internal capability. |
| Dependency on single vendor for testing tools | Diversify tooling options and train staff on multiple platforms to reduce lock-in. |