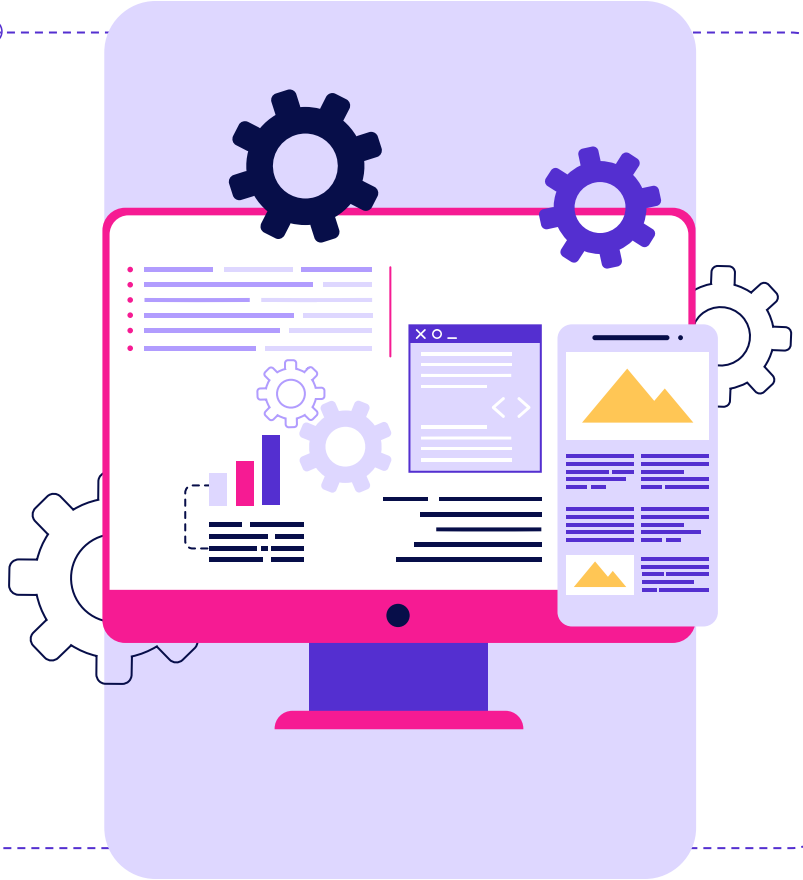
Project 3
Team 23



Team

Members

- Karan Bhatt
- Vedashree Ranade
- Madhusree Bera
- Yash Maheshwari
- Piyush Rana

Mentor

- Ankith

Team Contribution

Name	Contribution
Vedashree Ranade (2022201073)	-API's: /courses/filter, /reviews/average_ratings, /get_reviews/{course_id}, /track_progress, /add_to_progress, /remove_from_progress -Sub-Systems: Course Exploration, Course Review, Learning Management -Monolithic Architecture Subsystem -Features: Get filtered list of courses, Get average ratings of all courses, Get all reviews of a course, Progress Tracking
Karan Bhatt (2022202003)	-API's: /course, /course/{course_id}, /add_review, /enroll_student, /get_chapter_progress, /download_certificate -Subsystems: Course Exploration, Course Review, Learning Management -Architecture Analysis -Features: Get all courses list and a particular course information, Add reviews, Enroll Student, Get particular chapter progress, Get certificate for a course after 100% completion
Piyush Rana (2022202012)	-API's: /signup, /login, /logout, /validate, /protected -Sub-Systems: Authentication, Payments, Web Application (UI) -Integration of the subsystems -Features: User Signup, Login, Logout, Validations, Courses Page (Front End)
Madhusree Bera (2022202007)	-APIs: /register_service, /register_service_health_monitor, /register_service_load_balancer, /deregister_service/{service_id}, /get_service, /balance_load, /start_health_monitor -Sub-Systems: Service Registry, Load Balancer, Health Monitor, Web Application (UI) -Features: Email Notification feature, Health Monitor feature, Logging feature
Yash Maheshwari (2022201074)	-API's: /payment, /add_course, /delete_course -Sub-Systems: Course Management, Payments -Diagrams -Features: Add and Delete courses, Manage Payments

Task 1
Requirements and Subsystems

Functional Requirements

- User Registration and Authentication
- Course Management
- Course Exploration
- Track learning progress
- Notifications and Communication
- Payment Processing
- Course Review and Rating Submission
- Average Rating Calculation
- Review Editing and Deletion
- Review Verification
- Subscription Plan
- Device Limitation Setting

Non-Functional Requirements (1)

- Performance
  - ensure that users experience quick responses (latency < 1s)
  - detect fault if any service is down and notify admin
- Security
  - protect sensitive user data by encrypting information such as passwords, payment details, and personal information
  - secure authentication mechanisms and role-based access control
- Availability
  - 99.99% uptime over any given 12-month period
  - Load balancing mechanisms shall be employed to distribute incoming traffic evenly

Non-Functional Requirements (2)

- Scalability
  - handle increased workload by upgrading hardware (vertical scaling)
  - scale out across multiple servers or cloud instances (horizontal scaling)
- Usability
  - intuitive user interface that allows users to easily navigate
  - responsive layouts and adaptive designs
- Maintainability
  - modular and reusable components to facilitate easier maintenance, updates, and enhancements
  - well-structured code following coding standards making it easier to understand, debug, and extend.

Subsystems Overview

01 Course Management Subsystem

Key Functionalities:

- Course Creation
- Course Organization with Categories
- Content Management
- Course Editing and Updating
- Course Deletion

Subsystems Overview

02 Course Exploration Subsystem

Key Functionalities:

- Course Listing
- Search and Filter
- Course Details
- Enroll Courses
- Course Deletion

Subsystems Overview

03 Learning Management Subsystem

Key Functionalities:

- Enrollment Management
- Content Access
- Discussion Forum
- Progress Tracking
- Certificate Issuance

Subsystems Overview

04 Course Review Subsystem

Key Functionalities:

- Review Submission
- Review Listing
- Average Rating
- Review Management for Course Creators
- Verification for Reviews

Subsystems Overview

05 Payment Subsystem

Key Functionalities:

- Payment Processing
- Flexible Payment Options
- Secure Transactions

Subsystems Overview

06 Subscription Subsystem

Key Functionalities:

- Subscription Tiers
- Subscription Management
- Subscription Periods

Subsystems Overview

07 User Management Subsystem

Key Functionalities:

- User Registration
- User Login and Session Management
- Authorization and Role-based Access Control (RBAC)
- Devices Logged In Management

Subsystems Overview

08 Administration Subsystem

Key Functionalities:

- User Management
- Content Management
- Platform Configuration
- Analytics and Reporting

Subsystems Overview

09 Partner Integration Subsystem

Key Functionalities:

- Partner Portal
- Communication and Data Exchange

Subsystems Overview

10 Web Application Subsystem

Key Functionalities:

- Intuitive User Interface
- Responsive Design
- Accessibility

Task 2
Architecture Framework

Stakeholder Identification

01 Learners

02 Course creators

03 Admins

04 Platform Partners

05 Payment Gateways

06 Developers

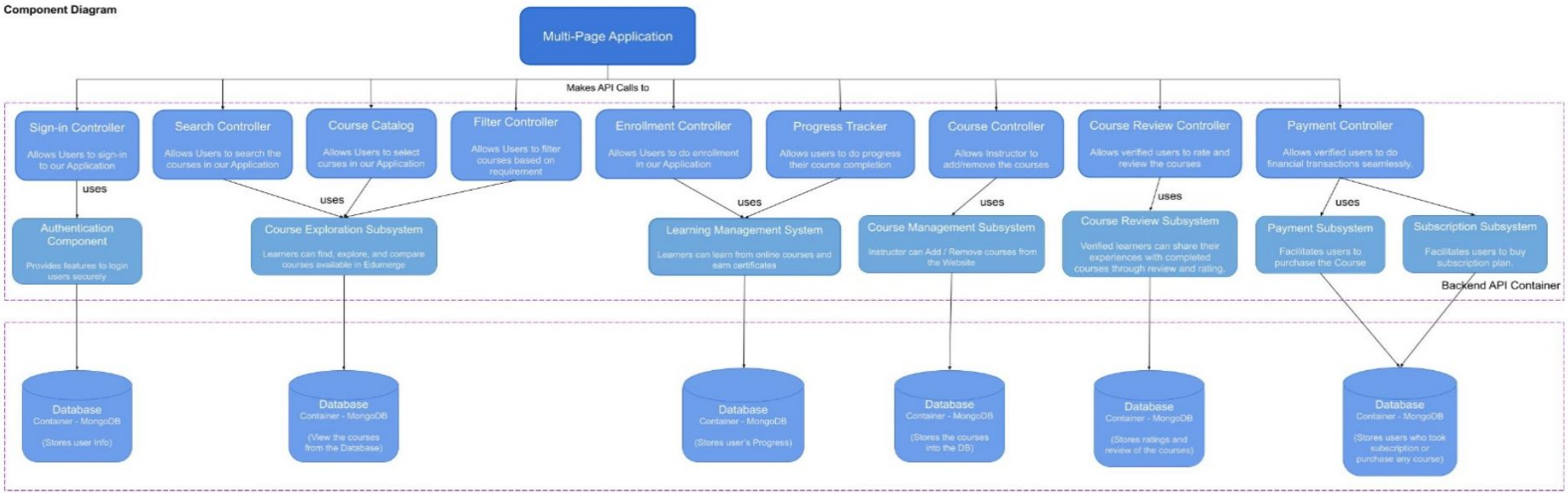
07 Investors

08 Regulators

Major Design Decisions:

ADR-1	**Selection of Microservices Architecture**
ADR-2	**Document Database for Course Content**
ADR-3	**Authentication and Authorization Strategy**
ADR-4	**Technology Stack for UI and Backend**
ADR-5	**Use of API Gateway**
ADR-6	**Use of CDN for Content Delivery**

Architecture Component Diagram



Task 3
Architectural Tactics and Patterns

Architectural Tactics

01 Architectural Tactics for High Availability in EduMerge

- Fault Detection Tactics:
  - Ping/Echo & Heartbeat
- Fault Recovery Tactics:
  - Active Redundancy
  - Passive Redundancy
- Fault Prevention Tactics:
  - Process Monitor

Architectural Tactics

02 Architectural Tactics for Performance in EduMerge

- Reduce Resource Demand
- Increase Resource Availability
- Resource Scheduling
- Caching

Architectural Tactics

03 Architectural Tactics for Security in EduMerge

- Authentication
- Authorization
- Data Encryption
- Input Validation
- Limit Access
- Regular Security Updates

Architectural Tactics

04 Architectural Tactics for Modifiability in EduMerge

- Generalize Modules/Modularity
- Localize Changes
- Anticipate Expected Changes
- Limit Possible Options

Architectural Tactics

05 Architectural Tactics for Usability in EduMerge

- Design Time
- Support User Initiative

Implementation Patterns

01 Adapter Pattern for Partner Platform Course Integration

Rationale:

Partner platforms might have different ways of representing courses, user information, and enrollment data. The adapter pattern isolates platform-specific details and provides a consistent interface for EduMerge to interact with any partner platform seamlessly. This promotes loose coupling and simplifies the integration process for new partners.

Implementation Patterns

02 Factory Pattern for Course Creation

Rationale:

The factory pattern centralizes the logic for course creation, making it easier to manage and maintain. It allows for adding new course types in the future without modifying existing code that interacts with the factory. This promotes code reusability and simplifies course management for administrators.

Implementation Patterns

03 Factory Pattern for User Creation

Rationale:

The factory pattern centralizes user creation logic, making it easier to manage and maintain. You can add new user roles in the future by simply adding new creator methods to the factory without modifying existing code. This promotes loose coupling and simplifies the user creation process.

Implementation Patterns

04 Strategy Pattern for Payment

Rationale:

The strategy pattern provides flexibility in handling payments. EduMerge can integrate with new payment gateways easily by adding new concrete strategies without modifying the core payment processing logic. This allows for future expansion and caters to a wider user base with diverse payment preferences.

Implementation Patterns

05 Criteria Pattern for Filtering the courses

Rationale:

The criteria pattern enables powerful and expressive course search functionalities. Users can filter courses based on specific combinations of attributes, leading to a more personalized and efficient learning experience. This pattern also simplifies the implementation and maintenance of search functionalities within EduMerge.

Implementation Patterns

06 Observer Pattern for Notifications

Rationale:

This pattern promotes loose coupling and simplifies development. Observers don't need to know how subjects work, and subjects don't need to manage individual user notification preferences. It also offers flexibility for users to control their notification streams and enhances scalability as EduMerge adds new features and notification types.

Task 4
Prototype Implementation and Analysis

Architectural Tactics in Prototype (1)

Availability tactics:

- Fault Detection
    1. Heartbeat
    2. Exception

Performance Tactics:

- Resource Management
    1. Introduce concurrency
    2. Make multiple copies and use load balancer to distribute load

Security Tactics:

- Resisting Attacks
    1. Authenticate users
    2. Authorize users

Architectural Tactics in Prototype (2)

Testability Tactics:

- Internal Monitoring
    1. Built-in-monitors for instrumentation (observability)

Usability Tactics:

- Design time
    1. Separate UI from the rest of system

Microservices vs. Monolithic Trade-offs

Development: Microservices require to design, build, and maintain multiple independent services. This involves managing separate codebases, APIs, and potentially different programming languages or frameworks for each service. Compared to a monolithic codebase, this requires more coordination during development.

Deployment: Deploying changes in a microservices architecture involves managing deployments for multiple services. This requires robust CI/CD pipelines to automate testing and deployment processes across all services. Additionally, communication and coordination between development teams working on different services become more crucial to ensure smooth deployments without introducing regressions.

Microservices vs. Monolithic Trade-offs

Operational Management: Monitoring and troubleshooting issues in a distributed system with multiple services can be more complex. We need to monitor the health and performance of individual services as well as how they interact with each other. Additionally, distributed tracing tools might be required to track requests across different services and pinpoint the root cause of problems.

Communication Overhead: In a monolithic application, components can directly access data and functionalities within the same codebase. In microservices, communication between services happens through APIs, which can introduce some network overhead compared to in-memory function calls. This overhead can be mitigated through techniques like API Gateway, Caching, Message Queues

Microservices vs. Monolithic Trade-offs

Conclusion: While microservices introduce additional complexity, the benefits for EduMerge are significant.

- **Increased Complexity:** Microservices introduce challenges, but the benefits outweigh them.
- **Scalability:** Scale specific services to meet EduMerge's growing demands.
- **Faster Releases:** Deploy features quicker to improve user experience.
- **Fault Tolerance:** Enhance platform stability by isolating service failures.
- **Management Strategy:** Proper development practices, CI/CD pipelines, and optimization techniques are key to success.

Links

Github: https://github.com/kbbhatt04/se_project_3

Report: project3_23

THANKS!

Questions are welcome!