Survey Corps

Ashutosh Shrivastava	Abhik Sarkar	Pratham Sharma	Samudra Borkakoti
24205743	24214927	24220984	24207626

Project Title: Expanse - Distributed Online Learning Platform

Repository Link: [Expanse GitLab Repository URL]

Synopsis

Expanse is an **online learning platform** designed to make education accessible, interactive, and personalized for students worldwide. Inspired by the spirit of exploration, it fosters a collaborative environment where students and instructors can share knowledge through discussion forums, quizzes, and personalized learning paths that adapt to individual progress. Additionally, Expanse incorporates real-time learning analytics to help students track their growth, identify areas for improvement, and optimize their learning experience.

The application operates in the **Educational Technology (Ed-Tech)** domain, focusing on improving teaching, learning, and assessment processes through scalable, cloud-based digital tools.

Key features of the platform:

- Facilitate a course management system for students and educators.
- Enable collaborative learning through course-specific discussion forums.
- Provide real-time quizzes with automated assessments and instant scores.
- Ensure secure user authentication, authorization, and role-based access control.
- Provide an intuitive, visually engaging and dynamic user interface for enhanced engagement.
- Responsive design principles to ensure a seamless experience across devices.
- Enable efficient file storage and retrieval for multimedia learning resources.

Technology Stack

1. **Frontend: Next.js + TypeScript**
   - **Purpose:** Build an interactive and performant user interface with SSR capabilities.
   - **Why Chosen:** High performance, efficient routing, and SEO benefits.

2. **Styling: TailwindCSS**
   - **Purpose:** Ensure responsive and consistent UI design.
   - **Why Chosen:** Simplified styling and utility-first approach.

3. **Backend: FastAPI (Python)**
   - **Purpose:** Handle API requests, business logic, and data processing.
   - **Why Chosen:** Fast, asynchronous capabilities and built-in validation.

4. **Inter-service Communication: gRPC**
   - **Purpose:** Enable high-performance communication between microservices.
   - **Why Chosen:** Lightweight, efficient, and reduces latency.

5. **Database:**
   - **PostgreSQL:** Store structured data like user details, quizzes, course and topic metadata, and discussions.
   - **MongoDB:** Store unstructured data like course materials. (eg. video, pdf, photo etc.)

6. **Authentication: NextAuth.js + Google OAuth**
   - **Purpose:** Provide secure user authentication and session handling.
   - **Why Chosen:** Easy integration with Next.js and robust OAuth2 support.

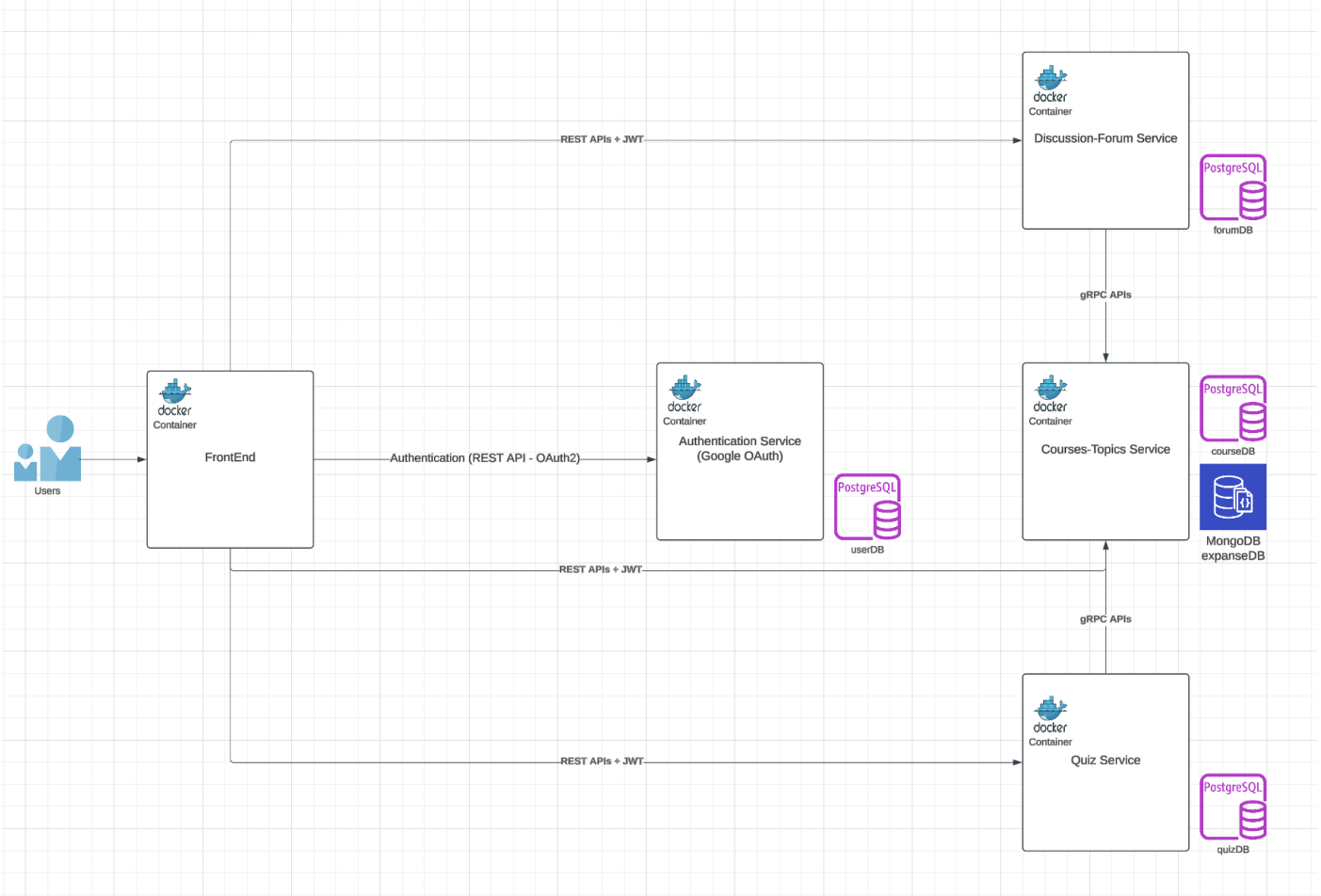
7. **API Documentation: Swagger**
   - **Purpose:** Provide auto-generated API documentation for backend services.
   - **Why Chosen:** Enhances clarity and ease of integration for developers.

8. **Containerization: Docker**
   - **Purpose:** Streamline local development and deployment processes.
   - **Why Chosen:** Ensures consistency across different environments.

9. **Deployment: Kubernetes (Frontend + Backend)**
   - **Purpose:** Host and scale application services.
   - **Why Chosen:** Simplified container orchestration, backend scalability and load balancing with Kubernetes.

System Overview

System Architecture Diagram:



System Components:

1. **Frontend (Next.js + TailwindCSS):**
   - Leverages server-side rendering (SSR) for SEO optimization and faster initial load times.
   - Supports dark mode and theme customization to improve user accessibility.
   - Handles routing, dynamic rendering, and API integration.
   - Implements responsive design principles to ensure a seamless experience across devices.
   - Uses code splitting and lazy loading to improve page load times and performance.
   - Includes accessibility best practices to ensure compliance with WCAG standards.

2. **Backend (FastAPI + gRPC):**
   - Manages API endpoints, authentication, and data processing.
   - Supports gRPC-based microservice communication.
   - Provides high-performance asynchronous communication to ensure scalability under heavy loads.
   - Supports horizontal scaling by containerizing services with Docker and orchestrating them via Kubernetes.

3. **Course and Topic Service:**
   - Provide a course management system for students and educators.
   - Utilizes PostgreSQL for structured data and MongoDB for unstructured file storage.
   - Enables seamless file upload and retrieval with MongoDB GridFS for efficient binary storage.
   - Supports hierarchical organization of topics for structured course navigation.

4. **Discussion Forum Service:**
   - Handles course-specific discussion threads and comments.
   - Ensures role-based access control for thread moderation.

5. **Quiz Service:**
   - Manages quiz creation, attempts, and auto-grading.
   - Ensures immediate feedback and score calculations.

6. **Authentication Service (NextAuth.js + Google OAuth):**
   - Implements user login and session management.
   - Allows seamless integration and authentication with Google.
   - Issues JWT tokens for secure communication.
   - Allows role-based authentication to differentiate access levels for students and instructors.

7. **Inter-service Communication (gRPC):**
   - Quiz Service and Discussion Forum Service utilize gRPC to communicate with Course-Topics Service for seamless data exchange and functionality.
   - Functions called
     a. **checkEnrollment:** Course Service checks whether the user is enrolled in the course or not. Based on the response, further logic on Quiz Service and Discussion Service will be executed.
     b. **courseValidity:** Course Service checks if a course is still active and accessible. Based on the response, further logic on Quiz Service and Discussion Service will be executed.
     c. **courseName:** Course Service fetches the name of the course based on the unique identifier provided.

8. **Database (PostgreSQL + MongoDB):**
   - PostgreSQL: Handles relational data.
     a. PostgreSQL contains all the metadata related to Users, Courses, Topics, Quiz and Discussion.

- MongoDB: Stores unstructured data like files and metadata.
    a. MongoDB stores the unstructured data related to Topic Content which might be a video, a photo or a pdf document.

9. **API Documentation (Swagger):**
    - Documents all backend endpoints, making integration seamless for developers. FastAPI automatically takes care of it. It generates the API documentation on its own.
        a. http://localhost:8080/docs : API Documentation for Courses Topics Service.
        b. http://localhost:8081/docs : API Documentation for Discussion Forum Service.
        c. http://localhost:8082/docs : API Documentation for Quiz Service.

10. **Deployment, Scaling and Load-Balancing (Kubernetes):**
    - **Containerization**: All microservices are containerized using Docker for consistency across environments.
    - **Kubernetes Orchestration**: Deployed on a Kubernetes cluster for automated deployment, scaling, and service management.
    - **Horizontal Pod Autoscaling (HPA)**: Dynamically scales pods based on resource usage to handle variable loads effectively.
    - **Load Balancing**: Kubernetes services distribute traffic evenly across pods for high availability and performance.
    - **Resource Allocation**: Defined resource limits ensure fair and efficient utilization of cluster resources.
    - **Fault Tolerance**: Kubernetes monitors and automatically restarts failing pods to maintain reliability.

Scalability and Fault Tolerance:

- **Microservices Architecture:** Independent scaling of services based on traffic.
- **Load Balancing:** Ensures even traffic distribution.
- **Token-Based Authentication:** Stateless authentication using JWT.
- **Containerization**: Uses Docker to package services, ensuring consistent deployment across environments.
- **Orchestration**: Employs Kubernetes for automated service scaling, load balancing, and failover management.

Contributions

Ashutosh Shrivastava (24205743)**:**

- System Design of the Project.
- Developed Frontend for Authentication and User Login Flow
- Developed Backend For Authentication Service.
- Implemented Google OAuth to have login with Google.
- Developed Frontend For Courses and Topics Service.
- Developed Frontend for Quiz Service
- Developed Frontend for Discussion Service.
- Developed authentication and token validation flows.
- Implemented Kubernetes Horizontal Pod Autoscaling (HPA) for Authentication Service and the entire frontend of the project to dynamically handle load balancing and scaling.
- Dockerised frontend and Authentication Service.
- Contributed in the final report.

Abhik Sarkar (24214927)**:**

- System Design of the Project.
- Set up the entire backend structure.
- Integrated gRPC communication between services.
- Integrated NoSQL database - MongoDB to store unstructured data (videos, photo, pdf etc.) in Courses and Topics Service. Dockerized backend services. (Course, Topic, Quiz, Discussion)
- Implemented Kubernetes Horizontal Pod Autoscaling (HPA) across all the services in the project to dynamically handle load balancing and scaling.
- Developed the Quiz Service.
- Developed the Courses and Topics Service.
- Contributed in development, debugging and fast tracking of Discussion Service.
- Contributed in the final report.

Pratham Sharma (24220984)**:**

- Set up the initial Quiz Database.
- Contributed in the development of the Quiz Service
- Contributed in the final report.

Samudra Borkakoti (24207626)**:**

- Designed the system and data models.
- Managed database schema in PostgreSQL.
- Developed the Discussion service.
- Contributed in the final report.

Reflections

Key Challenges and Solutions:

1. **Inter-service Communication Bottlenecks:**
   - **Issue:** Microservices need to communicate with each other for essential validations like course validity and enrollment. This communication needs to be reliable and fast.
   - **Solution:** Used **gRPC** for fast and reliable communication. It is efficient and facilitates low-latency communication.
2. **Scalability and High Traffic Handling**
   - **Issue:** It is important to ensure the application scales dynamically to accommodate fluctuating user loads.
   - **Solution:** Used **Kubernetes Horizontal Pod Autoscaler (HPA)** to automatically scale pods on CPU usage. Kept the CPU utilization threshold at 40%. Also we set the maximum number of replicas to be 20 and minimum number of replicas to be 2.
3. **Fault Tolerance and High Availability**
   - **Issue:** Minimizing downtime in case of service and pod failures.
   - **Solution: Kubernetes'** self-healing capabilities automatically restarts failing pods and load balancing ensures continuous availability of services.
4. **Token Validation Across Services:**

- **Issue:** Ensuring secure and seamless communication between the microservices and the frontend. Ensuring that users do not have access to quizzes and discussion-forums for the courses in which they are not enrolled.
- **Solution: Google OAuth2 with JWT** was implemented for stateless authentication. JWT token is passed along with each request, from which the userID. Tokens are validated in every backend service to ensure only authenticated users can access resources.

What Would We Do Differently?

1. **Implementing Monitoring and Observability Tools**
   a. **Current Situation:** There are no real time monitoring metrics or logs.
   b. **Improvement:** We can introduce tools like Prometheus for metrics collection and Grafana for visualization. It would make tracking resource utilization easier and help us detect bottlenecks.
2. **Add Caching Mechanisms**
   a. **Current Situation:** Frequently requested topic media, if accessed repeatedly might result in latency and increase load on the DB.
   b. **Improvement:** We can implement Redis, as a caching layer for highly requested content. We can serve content directly from in-memory cache making retrieval much faster. This will also enhance scalability, by offloading traffic to the caching layer.
3. **A more optimized Unstructured Data Storage**
   a. **Current Situation:** MongoDB GridFS has limitations, when it comes to scalability and performance, particularly for large files or high-volume traffic scenarios.
   b. **Improvement:** We can replace GridFS with Amazon S3. It provides better scalability, improved performance. In addition to that it is highly durable.

Learnings from Technologies:

- **Next.js:** Server-side rendering, Static Site Generation and API integration are seamless.
- **FastAPI:** Async capabilities enhance backend performance. Pydantic schemas make building robust REST APIs straightforward. API Documentation is an added benefit.
- **MongoDB:** MongoDB's schema-less design and GridFS were vital in managing unstructured data like videos, photos and pdfs.
- **PostgreSQL:** It is very reliable. Its powerful indexing and support for relational data made it ideal for our use-case when it came to structured data management.
- **Kubernetes:** It provides a robust platform for container orchestration. It helped us ensure scalability, high availability and automated failover. The HPA helped us effectively manage varying loads.
- **gRPC:** Enabled efficient inter-service communication with low-latency. The protobuf-based schema allowed for easy versioning and clear API definitions.

Limitations:

- **Real-time update**s (e.g., live discussions) have not yet been implemented.
- Using MongoDB GridFS posed limitations in scalability and performance, especially for high-volume traffic and large media files.
- A caching solution like Redis could have significantly increased retrieval times for highly requested media.

Benefits:

- Highly scalable architecture. Expanse scales dynamically with user demand, maintaining performance during high traffic. Made possible using Kubernetes HPA.

- Intuitive and user-friendly interface. Next.JS enabled server-side rendering and API integration, enhancing the overall performance and responsiveness of the frontend.
- Seamless inter-service communication using gRPC.
- FastAPI's asynchronous capabilities allowed us to handle multiple I/O operations efficiently, significantly improving API performance.
- The microservices-based architecture promoted modularity, making it easier to maintain, update and scale individual components without affecting the entire system.
- Secure and efficient API communication. Streamlined user authentication using Google Oauth eliminated the need for complex custom authentication mechanisms.

Next Steps / Future Improvements

- **Real-time Updates:** Implement **WebSocket** for live interactions in course materials and discussion forums.
- **Improved Notifications:** Enable real-time alerts for updates, replies, and quiz scores.
- **Enhance Discussion Forum:** Support **hierarchical comment structures** and implements **voting mechanisms** for threads and comments.
- **Multimedia Uploads:** Support **additional file formats** like DOCX, Excel, and ZIP, allowing users to upload, view, and download diverse types of learning materials.
- Integration with additional identity providers, such as **Microsoft** or **GitHub** OAuth.
- **Advanced Analytics:** Introduce dashboards for instructors to monitor **student progress**, analyze **engagement metrics**, and identify **performance trends** effectively.