Initial Pitch

text of a few paragraphs

1. **Objective**: A clear and concise explanation of what the project aims to achieve.
2. **Functionality**: A detailed list of the features the system will provide.
3. **Distributed Nature**: Highlight the distributed elements of the system (e.g., peer-to-peer architecture, decentralized communication).
4. **Target Users**: A description of the intended user base and their needs.
5. **Use Cases**: Real-world scenarios that demonstrate how the system will be used.
6. **Creativity and Challenge**: Ensure the proposal is innovative, beyond trivial applications, and presents technical or conceptual challenges.
7. **Preliminary High-Level Description**: This should avoid being overly detailed but sufficient to convey the essence of the project.

Learning Platform (Similar to Udemy)

Description:
An online platform where instructors can upload courses and students can access and complete them.

Technologies

- **Programming Language:**
  - **Python (Django or Flask)** for rapid development of APIs and backend logic.
  - **Node.js** for handling real-time features like chat or notifications.
  - **Java/Spring Boot** for robust microservices with extensive enterprise support.
- **Microservices Orchestration:**
  - **Docker** for containerizing each microservice, ensuring easy scaling and deployment.
  - **Kubernetes** for orchestration and management of microservices, allowing the system to scale horizontally as traffic increases.
- **Database:**
  - **PostgreSQL** for relational data (users, course metadata, assessments).
  - **Cassandra** for distributed data storage, ensuring scalability and performance across a large user base.
- **Authentication & Authorization:**
  - **OAuth2** or **JWT (JSON Web Tokens)** for secure, token-based authentication.
  - **Keycloak** as an identity and access management tool for handling role-based access control.
- **Notification**: Use a **Publish/Subscribe (Pub/Sub)** mechanism to deliver instant notifications when specific events occur.
  - **Tech Stack Recommendations:**

        - **Message Broker**: Use a message broker like **Redis (Pub/Sub)**, **RabbitMQ**, or **Apache Kafka** to handle real-time events.
        - **Database**: Store notification history in a database (e.g., **PostgreSQL**, **MongoDB**) for audit and re-delivery purposes.
        - **Notification Service**: Build a dedicated microservice that listens to events and sends notifications.

Frontend Technologies:

- **React.js or Vue.js** for building a responsive, interactive user interface.
- **Redux** (with React) or **Vuex** (with Vue) for state management.
- **Tailwind CSS** or **Bootstrap** for rapid UI development.
- Real-time notifications using technologies like **WebSockets** or **Server-Sent Events (SSE)**.
-

Video Streaming:

- **AWS S3** for scalable storage of video content.
- **AWS MediaConvert or FFmpeg** for video transcoding to various formats.
- **Content Delivery Networks (CDN)** like **Cloudflare** or **AWS CloudFront** for fast and reliable delivery of video content globally.

Communication:

- **WebSockets** for real-time features (e.g., chat, notifications, live interactions).
- **REST API or GraphQL** for communication between microservices and frontend.

Suggested Microservices

- **Course Management**:
    - Handles course creation, updates, and organization by instructors.
    - Manages course metadata, including titles, descriptions, categories, and prerequisites.
- **User Management**:
    - Manages registration, authentication, and profiles for instructors and students.
    - Differentiates roles and permissions (instructor, student).
    - It will be designed to interact with a centralized user database and will likely be backed by a **distributed database** like **Cassandra** or **PostgreSQL** with read replicas.
- **Microservice for Professor's Content Management**
    - Handles uploading, storing, and organizing various types of content (e.g., PDFs, Word documents, images, videos).

3.1 **Microservice for Student's Content Management**

- Students will be able to share their own notes and to see other students' announcements based on their preferences/courses . The functionalities are : create/modify/delete/upload/publish files.

- **Assessments**:
  - Provides functionality for creating quizzes and tests.
  - Provides functionality for students to submit answers and see results.

Optional:

- **Notification Management**
- **Forum**
  - Course-Specific Discussions: Each course will have its dedicated forum automatically created upon course setup. Students enrolled in the course and the instructor can participate in discussions relevant to the course material.
  - Topic-Based Threads: Users can start new threads based on specific topics or questions within the forum. Threads are tagged or categorized to make navigation easy (e.g., "Assignment Help," "Exam Preparation," "Concept Clarification").
  - Private and Group Discussions: Options for creating private discussion threads for group projects or study groups within the course. Restricted access based on roles or invitation.
  - Notifications: Users receive notifications for replies to their posts, new threads in subscribed forums, or instructor announcements.
  -
- **Certification Generation**:
  - Automates certificate issuance for students who complete courses.
  - Personalized certificates with course details, student names, and completion dates.
- **Payment Processing**:
  - Manages subscription plans, individual course purchases, and refunds.
  - Ensures secure payment handling via third-party integrations (e.g., Stripe, PayPal).

Proposal

This project aims to simplify and improve how students and professors manage courses, share materials, and conduct assessments. The platform will help professors and students, providing tools tailored to their specific needs.

Functionality

The primary goal of the project is to create an efficient and user-friendly platform for managing educational content, users, and assessments.
Professors can create and update courses, upload and organize teaching materials, and design quizzes.
On the other hand students, after submitting to courses, can access their resources, participate in quizzes and receive their assessments. Students can manage their own notes and decide to share them, browse and filter their colleagues' ones.

Main components

We designed the platform through a microservices architecture, dividing the system into smaller components, each handling a specific task like course management, user authentication, content sharing, or assessments.

- **Course management microservice**:
    - Handles course creation, updates, and organization by professors.
    - Manages course metadata, including titles, descriptions, categories, and prerequisites.
- **User management microservice**:
    - Manages registration, authentication, and profiles for professors and students.
    - Differentiates roles and permissions (professor, student).
- **Microservice for professor's content management**
    - Handles uploading, storing, and organizing various types of content (e.g., PDFs, Word documents, images).
- **Microservice for student's content management**
    - Students will be able to share their own notes and to see other students' publications based on their preferences. The functionalities involve uploading, modifying, deleting files.
- **Assessments microservice**:
    - Provides the professors the functionalities for creating tests.
    - Provides functionality for students to submit answers and see results.

Target users

- **Professors** play a central role in managing and delivering educational content, as well as assessing student progress.
- **Students** interact with the platform to access learning resources, participate in tests, and share notes.

Use cases

All users:

- **User registration and role selection:**
  A user registers in the application and selects its role (professor or student).
- **User login and logout:**
  A user logs into the system to access the functionalities.

Professors:

- **Create and manage courses:**
  Professors can create new courses, update course details and manage their structure.
- **Upload and organize course content:**
  Professors can upload and manage various types of content (e.g., PDFs, Word documents, videos) and organize them within their courses.
- **Create tests:**
  Professors can design and publish tests for their courses.

Students:

- **Upload notes:**
  Students can upload their own notes.
- **View notes from other students:**
  Students can access notes uploaded by other students based on their enrolled courses and preferences.
- **Take tests and view results:**
  Students can take tests provided by their professors and view the results.
- **Favourites notes**

Pratica

Colori Immagine
#ef6234
#3ee5f2

#f9ed6a
#475862
#273143

Requirements & Mockups

- Define user stories and document them in a spreadsheet.
- Create **LoFi mockups** using suggested tools (e.g., **Balsamiq**).
- Provide a textual description of each user story with non-functional requirements.

Effort Estimation

- Estimate software complexity, time, and effort using **Function Points** and **COCOMO** II.
- Document the estimation in spreadsheets and a booklet.

Development & SCRUM Process

- Define and document **SCRUM**-based sprints.
- Track progress using spreadsheets and burndown charts.

System Design & Architecture

- Document the software architecture and system design.
- Ensure modularity and separation of concerns in microservices.

Implementation & Deployment

- Develop the system using any technology/framework.
- Use Infrastructure as Code (IaC) for deployment (**Docker**, docker-compose).

Project Submission on GitHub
Repo structure:

```
├── input.txt
├── Student_doc.md
├── DataMetrics.json
├── source  # Source code and configuration files
├── booklets  # Documentation, estimation, sprints, slides
```

The repository should be named `<MATRICOLA>_<PROJECT>`