LZSCC.330 Software Design Studio Project II (Networked Studio)

Assessment Description

1 Introduction

LZSCC.330 provides students with the opportunity to build a networked software system using modern architectural patterns and agile development practices. The focus is on building a real-world application using microservices architecture, event-driven systems, and continuous integration (CI) practices.
In this setting, you are expected to work continuously and self-driven on the given software development task and cooperate and communicate frequently with your group members. The development process adopts Scrum methodology with emphasis on disciplined software engineering practice, including sprint planning, daily standups, and retrospectives.

GREEN CATEGORY
Under this category, Gen AI is designed to be
a part of the assessment as specified by the module tutor
and required by the assessment.

Generative AI Policy: **GREEN**

GenAI Policy: You are **encouraged** to use generative AI tools for programming tasks. When you use them, please **record and declare your usage**. You must include a section in your reports describing how you used GenAI, with example prompts and responses. You are expected to understand all code you submit, regardless of how it was generated.

2 Task

This year's project is:

2.1 MARP-Guide: RAG Chatbot with Microservices and Event-Driven Architecture

2.1.1 Product Goal

Build a chat application that answers questions about Lancaster University's **Manual of Academic Regulations and Procedures (MARP)**. Answers must be derived from the processed MARP PDF documents, properly cited (with title, page number, and link), and presented in an understandable manner.
Target Audience: Students and staff who need quick access to university regulations.
Value: Reliable, quickly accessible information with sources—no hallucinations.

2.1.2 Framework and Constraints

You are asked to work on this project in groups of 3–4 persons.

- **Architecture:** Microservice architecture **and** Event-Driven Architecture (EDA). Implement decoupled services for ingestion, indexing/retrieval, and chat orchestration.
- **RAG (Retrieval-Augmented Generation):** Mandatory. Use hybrid or dense retrieval, but answers must **only** contain information supported by sources.

- **LLM:** Use **OpenRouter** `https://openrouter.ai/` (free models are permitted for this course).
  - → **No sensitive data** may be sent to external APIs. MARP documents are public, so this is acceptable.
- **Data Source:** Main page: `https://www.lancaster.ac.uk/academic-standards-and-quality/regulations-and-policies/manual-of-academic-regulations-and-procedures/`
  - Linked PDF documents form the knowledge base of your system.
- **Team Size:** 3–4 persons
- **Duration:** 10 weeks (90 hours per person)
- **Methodology:** Scrum (short sprints, reviews, retrospectives, visible artefacts)

2.1.3 Minimum Functionality (MVP)

A. **Ingestion:** Automatically discover and fetch MARP PDFs, storing text and metadata (title, URL, date, page numbers).

B. **Indexing:** Implement chunking and embeddings (choice of model is yours), preserve metadata.

C. **Retrieval:** Return top-k snippets reproducibly via API.

D. **RAG Answers:** Provide chat endpoint/UI that generates answers with **at least 2 citations** (title, page, link). *Note: Assessment 1 (first increment) requires at least 1 citation; final MVP requires at least 2 citations.*

E. **EDA Flow:** Key workflow steps must generate events (e.g., `DocumentDiscovered`, `DocumentExtracted`, `ChunksIndexed`, `AnswerGenerated`).

F. **Operations:** Local deployment via **Docker Compose**. Health checks for each service.

G. **Quality:** Logs for all services.

2.1.4 Technology Constraints and Implementation Requirements

To ensure adequate complexity and learning depth, the following constraints apply. If you are unsure, please ask the module convenor.
Prohibited Technologies:
The following frameworks and pre-built solutions, as well as functionally similar ones, are **not permitted**:

- **RAG Frameworks:** OpenWebUI, anything-llm, Haystack, LlamaIndex, ...
- **AI Agent Frameworks:** Langchain, LangGraph, Pydantic AI, CrewAI, AutoGen, ...
- **Pre-built Chat Applications:** Complete chat application templates or RAG tutorial projects that provide more than 50% of required functionality

Permitted Technologies:
The following frameworks and pre-built solutions, as well as functionally similar ones, are **permitted**:

- **HTTP/API Clients:** Standard libraries (requests, httpx, fetch) for direct OpenRouter API calls
- **UI Frameworks:** React, Vue, Svelte, Angular, and UI component libraries (Material-UI, Chakra UI, shadcn/ui)
- **Backend Frameworks:** FastAPI, Flask, Express.js, NestJS, and database ORMs (SQLAlchemy, Prisma, TypeORM)
- **Infrastructure:** Messaging systems (RabbitMQ, Kafka, Redis Pub/Sub), vector databases (Qdrant, Weaviate, ChromaDB, pgvector), SQL/NoSQL databases, Docker and Docker Compose

- **ML/NLP Libraries:** sentence-transformers (for embeddings), Hugging Face transformers, OpenAI Python SDK (only for OpenRouter-compatible calls), tiktoken (token counting), PyPDF2, pdfplumber (PDF parsing)

Core Implementation Requirements:
The following components **must be implemented from scratch** by your team:

A. **RAG Pipeline Core:**
   - Custom chunking strategy with justification
   - Custom retrieval orchestration logic (query $\rightarrow$ embedding $\rightarrow$ search $\rightarrow$ ranking)
   - Custom citation extraction and formatting logic
   - Custom prompt templates for RAG responses

B. **Event-Driven Architecture:**
   - At least 5 custom event types defined with schemas
   - Custom event handler implementations
   - Documented event flow through the system

C. **Microservices Architecture:**
   - At least 3 independent services with clear responsibilities
   - Custom-designed REST/GraphQL APIs with documentation
   - Inter-service communication (both synchronous and event-based)

Required Additional Features:
Each team must implement **exactly 2 additional features** beyond the MVP:

- **1$\times$ Feature from Tier 1 (Foundation Features)** — extends basic infrastructure
- **1$\times$ Feature from Tier 2 (Advanced Features)** — adds significant technical complexity

Teams select their features during Sprint Planning (Week 2) and must document the design in Assessment 1 and demonstrate implementation in Assessment 2.
Tier 1: Foundation Features

A. **User Authentication & Session Management:** Login/logout, session handling, user-specific chat isolation

B. **Chat History Persistence:** Store all chats in database, retrieve past conversations

C. **Bookmarks & Favourites:** Users can mark important messages, retrieve saved items

D. **Document Management Interface:** Admin interface for manual PDF upload, document status overview

E. **Basic Monitoring Dashboard:** Service health checks, indexing statistics, request counts

Tier 2: Advanced Features

A. **Hybrid Search:** Combine dense vector search with sparse retrieval (BM25), weighted fusion

B. **Query Rewriting & Expansion:** Automatic query reformulation, synonym expansion, LLM-based rewriting

C. **Re-Ranking Pipeline:** Two-stage retrieval (recall $\rightarrow$ precision) with separate re-ranking model

D. **Multi-Model Comparison:** Parallel answer generation with different LLMs, side-by-side UI comparison

E. **User Feedback & Quality Metrics:** Thumbs up/down, feedback tracking, confidence scoring

F. **Advanced Monitoring & Observability:** Distributed tracing, latency metrics per service, event flow visualisation

G. **Rate Limiting & Intelligent Caching:** Request limits per user/service, semantic caching for similar queries

H. **A/B Testing Infrastructure:** Test different retrieval strategies in parallel, automated metric collection

Networked System Design:
The application must demonstrate proper networked system architecture with multiple services communicating over network protocols. All inter-service communication must be explicitly designed and documented. No in-process communication between services is allowed—all services must communicate via network interfaces (HTTP REST APIs or message queues).

2.1.5 Non-Goals

- No personal advice or decision recommendations; **citations and links only**.
- No user login required **(unless chosen as Tier 1 additional feature)**.
- No completeness guarantee for all university documents—focus is the **MARP area**.
- No use of pre-built RAG frameworks or AI agent frameworks (see Technology Constraints).

2.1.6 Acceptance Criteria (Definition of Done)

- **Correctness & Sources:** $\geq 90\%$ of answers in the demo catalogue contain **correct** citations (title + page + link).
- **Fallback:** If no suitable source exists $\rightarrow$ explicitly state "not certain / no source found".
- **Reproducibility:** `docker compose up` brings the system up and running (README with commands).
- **Network Architecture:** All services communicate via network protocols (HTTP REST APIs or message queues); no in-process communication between services allowed.
- **Technology Compliance:** No prohibited frameworks used; core RAG pipeline, EDA, and microservices implemented from scratch as specified.
- **Additional Features:** Both required features (1$\times$ Tier 1, 1$\times$ Tier 2) documented in Assessment 1, fully implemented in Assessment 2.
- **Scrum Evidence:** Product goal, prioritised backlog, sprint artefacts, reviews/retrospectives documented.
- **OpenRouter Compliance:** No upload of sensitive data; MARP text snippets only in RAG context to LLM.

2.1.7 Example Milestones

Sprint 1 (Weeks 1–5):

- **M1 (Week 1):** Team formation; Scrum fundamentals; Sprint planning; feature selection (Tier 1 + Tier 2); repository setup + GitHub Projects configured
- **M2 (Week 2):** Microservices architecture designed; tech decisions documented; repository setup; ingestion service started
- **M3 (Week 3):** Ingestion service functional (discover + fetch MARP PDFs); Docker Compose setup; Event-Driven Architecture applied; first events (`DocumentDiscovered`, `DocumentExtracted`)

- **M4 (Week 4):** Indexing service (chunking + embeddings); retrieval service (`/search` endpoint); basic RAG (`/chat` endpoint with 1 citation); 3+ events total
- **M5 (Week 5):** System integration verified; Docker Compose fully functional; **Assessment 1 presentation**

Sprint 2 (Weeks 6–10):

- **M6 (Week 6):** Sprint 1 retrospective completed; Sprint 2 planning finished; additional features work started
- **M7 (Week 7):** Chat UI implemented; complete RAG (2 citations); 5+ events implemented; automated tests started; CI pipeline planned
- **M8 (Week 8):** Additional features (Tier 1 + Tier 2) functional; automated tests implemented; GitHub Actions CI pipeline operational
- **M9 (Week 9):** Test catalogue (10–15 questions) completed; acceptance criteria verified; final integration and bug fixes
- **M10 (Week 10):** Final system polish and documentation; **Assessment 2 presentation**

2.1.8 Suggested Initial Backlog

- **EPIC Ingestion:** Discover MARP links → store PDFs → `DocumentDiscovered`
- **EPIC Extraction:** PDF→Text+Metadata → `DocumentExtracted`
- **EPIC Index:** Chunking+Embedding → `ChunksIndexed`
- **EPIC Retrieval:** `/search` returns top-k snippets with pages/URLs
- **EPIC RAG & Chat:** `/chat` generates answer with ≥2 citations, `AnswerGenerated`
- **EPIC UX:** Web chat with sources panel & link-out
- **EPIC Quality:** Measurements, test catalogue, fallbacks

Acceptance (per story): measurable, demo-capable, checkable in DoD checklist.

2.1.9 Data Protection and Ethics

- **No** personal data or confidential content may be sent to OpenRouter.
- Answers must **always** include sources; communicate clearly when uncertain.
- Respect robots/policies of university site; process only publicly accessible PDFs.

3 Assessment Structure

The module follows a **two-increment Scrum approach** with assessments after each 5-week sprint. Each assessment comprises both a working system demonstration and individual reflection, representing 50% of your final module grade.

3.1 Assessment 1: First Increment — Core RAG Pipeline (50%)

Submission deadline: 6 November, 15:00 (upload to Moodle)
Presentation: 7 November
Focus: Functional core system with basic RAG capability, microservices architecture, event-driven design, and comprehensive documentation.
Required Functionality:

- **Ingestion Service:** MARP PDFs discovered, fetched, and parsed (text + metadata)
- **Indexing Service:** Chunking, embeddings, and vector database storage functional
- **Retrieval Service:** API endpoint `/search` returns top-k snippets with metadata (title, page, URL)
- **Basic RAG:** Chat endpoint `/chat` generates answers with **at least 1 citation**
- **Event-Driven Architecture:** Minimum 3 events implemented (e.g., `DocumentDiscovered`, `ChunksIndexed`, `RetrievalCompleted`)
- **Deployment:** `docker compose up` brings system online
- **Network Architecture:** All services communicate via network protocols (HTTP REST or message queue)

Submission Requirements:
Each student submits to Moodle:

- Repository snapshot (zipped, without build artefacts such as `node_modules/`, `.venv/`, `target/`)
- Individual reflective report (PDF, 2 pages maximum)
- Presentation slides (PDF or PowerPoint)

The repository must contain:

- **`README.md`** with:
  - High-level architecture overview (Mermaid diagrams recommended)
  - Setup and run instructions (`docker compose up` must work)
  - Technology stack overview
- **Complete source code** with `docker-compose.yml`
- **`/docs` folder** containing:
  - Detailed architecture diagrams (microservices + network communication patterns)
  - Event schemas for implemented events with example payloads
  - API specification for REST endpoints
  - Additional features design (documented design and implementation plan for chosen Tier 1 and Tier 2 features)
  - Scrum artefacts (backlog, sprint log, team retrospective)

Group Component (50% of Assessment 1 = 25% of overall module grade):
The team demonstrates a working system with the core RAG pipeline functional. Assessment focuses on system architecture, network design, event-driven implementation, and code quality.
Individual Component (50% of Assessment 1 = 25% of overall module grade):
Each student submits an individual reflection on their technical contributions, architectural decisions, and development process. The report should reference specific code contributions visible in Git history.

3.2 Assessment 2: Second Increment — Production-Ready System (50%)

Submission deadline: 11 December, 15:00 (upload to Moodle)
Presentation: 12 December
Focus: Complete MVP with all acceptance criteria met, both additional features implemented, automated testing, and CI pipeline operational.
Required Functionality:

- **Complete RAG:** Chat generates answers with **at least 2 citations** (as per MVP)
- **Full Event-Driven Architecture:** Minimum 5 events implemented with complete event flow documentation
- **Chat UI:** Web interface or API-based chat interaction
- **Additional Features Implemented:** Both chosen features (1× Tier 1 + 1× Tier 2) fully functional
- **Automated Tests:** Unit and integration tests (minimum 10–15 test cases), test catalogue with demo questions
- **CI Pipeline:** GitHub Actions running tests on push
- **Acceptance Criteria Met:** $\geq$90% correct citations in demo catalogue, proper fallback handling

Submission Requirements:
Each student submits to Moodle:

- Repository snapshot (zipped, without build artefacts such as `node_modules/`, `.venv/`, `target/`)
- Individual reflective report (PDF, 2 pages maximum)
- Presentation slides (PDF or PowerPoint)

The repository must contain:

- **`README.md`** with:
  - Updated high-level architecture overview (reflecting changes since Assessment 1)
  - Setup and run instructions (`docker compose up` must work)
  - Technology stack overview
- **Complete source code** with `docker-compose.yml`
- **`/docs` folder** containing:
  - Updated detailed architecture diagrams (changes since Assessment 1 highlighted)
  - Complete event schemas (5+ events) with event flow documentation
  - Updated API specification for all REST endpoints
  - Additional features documentation (design, implementation, and rationale for Tier 1 + Tier 2 features)
  - Test documentation (coverage report, test catalogue with 10–15 demo questions)
  - Scrum artefacts (updated backlog, sprint log, lessons learnt, final retrospective)
- **Complete test suite** (unit and integration tests)
- **CI configuration** (GitHub Actions workflow files)

Group Component (50% of Assessment 2 = 25% of overall module grade):
The team demonstrates the complete system with all MVP requirements met, additional features working, comprehensive tests, and CI pipeline operational. Assessment focuses on system completeness, quality, and production-readiness.
Individual Component (50% of Assessment 2 = 25% of overall module grade):
Each student submits an individual reflection on their technical contributions to the second increment, testing approach, lessons learned, and final system quality. The report should describe individual work on features, tests, or infrastructure.

3.3 Summary: Four Graded Components

This means there are **four graded components in total**, each worth 25% of your final grade:

- Assessment 1 Demo (25%) + Assessment 1 Report (25%) = 50%
- Assessment 2 Demo (25%) + Assessment 2 Report (25%) = 50%

3.4 Demonstration Format

Each team demonstration should allocate approximately **5 minutes per group member** (e.g., a team of 4 has 20 minutes presentation time), followed by up to 10 minutes of open discussion and reflection.
Presentation Structure:

- Present slides covering the architecture, implementation, and sprint progress
- Focus should be on **running software demonstration** and **code walkthrough**
- Each team member should demonstrate their individual contributions
- Show relevant code, architectural decisions, and technical implementation

Like in real-world software development, the module convenor and marker may interrupt the planned demo and ask questions to direct what is shown. After the demonstration and discussion, you should reflect on your progress and impediments.
Marking: The demonstration receives both a **group mark** (assessing overall system quality, architecture, and team coordination) and an **individual mark** (assessing each student's contribution, technical understanding, and presentation skills).

3.5 Individual Reflective Report Format

Each report should contain **2 pages maximum**, without a cover page.
The report is a **technical reflection** on your individual contribution to the project. It should describe **what you did, why you did it, and what you learned**. The focus is on your thought process, decision-making, and learning, not just listing features.
Required Content:

- **Requirements & Individual Contribution (approx. 0.5 pages):**
  – Which requirements/features did you implement in this sprint?
  – What was your individual contribution vs. collaborative work?
  – Who did you work with, and how did you divide responsibilities?
- **Architecture & Design Decisions (approx. 0.5 pages):**
  – What architectural decisions did you make or contribute to?
  – What alternatives did you consider, and why did you choose your approach?
  – How does your code fit into the overall system architecture?
- **Implementation & Technical Challenges (approx. 0.5 pages):**
  – What design patterns or code structures did you use, and why?
  – What technical challenges did you encounter, and how did you solve them?
  – What aspects of code quality did you focus on (readability, maintainability, performance)?
- **Testing, Tools & Process (approx. 0.25 pages):**
  – What testing strategy did you follow, and what tests did you write?

  - What tools did you use (IDEs, debuggers, build tools), and how did they help?
  - How did you use GenAI (if at all)? Include specific examples of prompts and what you learned.

- **Critical Reflection (approx. 0.25 pages):**
  - What impediments or challenges did you face (technical, team, process)?
  - What solutions did you try, and how effective were they?
  - What would you do differently next time? What did you learn?

How to Reference Your Code:
Throughout your report, reference specific code you wrote using file paths and line numbers. For example:

- "I implemented the chunking algorithm in `src/indexing/chunker.py:45-120`..."
- "The event handler for `DocumentDiscovered` is in `src/events/handlers.py:34`..."

This helps markers connect your reflection to your actual code contributions. However, remember that your code quality is assessed during the demonstration, not from the report.
GenAI Usage:
Since this module has a GREEN GenAI policy, you are **encouraged** to document your GenAI usage. Include:

- What tools you used (ChatGPT, GitHub Copilot, Claude, etc.)
- Example prompts you used
- What you learned from the AI's responses
- How you verified and adapted the AI-generated content

If you have many GenAI examples, you may add a brief appendix (max 1 additional page) with example prompts.
Collaborative Work:
When describing contributions you led, state who helped you. When describing contributions you assisted with, state who led. For example:

- "I led the implementation of the retrieval service (`src/retrieval/`), with Alice helping on the ranking algorithm."
- "I assisted Bob with the ingestion pipeline, specifically implementing the PDF metadata extraction."

The report should describe your **individual** contribution, but acknowledge collaboration honestly.

Assessment Schedule

Assessment	**Upload Deadline**	**Presentation**	**Weight (%)**
Assessment 1 (First Increment: Core RAG Pipeline)	6 November, 15:00	7 November	50 (Group: 25%, Individual: 25%)
Assessment 2 (Second Increment: Production-Ready System)	11 December, 15:00	12 December	50 (Group: 25%, Individual: 25%)

4 Support

Support will be provided throughout the module through regular contact hours for technical and non-technical questions.
Since this is a software development project in the third year, you are expected to work **self-driven** and **ask questions when necessary**. As is normal with projects, small delays can happen, so start work early and plan ahead.

5 Marking

5.1 Demonstration Marking

Each demonstration is worth **25% of your final module grade** (50 marks). This marking scheme applies to both Assessment 1 (7 November) and Assessment 2 (12 December) demonstrations.
Important: The demonstration assesses both the **working system** (group aspects) and **individual code contributions** (individual aspects). This includes the technical quality and functionality of the code you described in your reflective report.
The presentations will be marked as follows:

- **Timing (2 marks):** Slides and report draft submitted on time (1); each group member stays within time allocation ± 1 minute (1).
- **System Functionality (10 marks):** [Group]
  - Requirements clearly described and demonstrated (5)
  - Features work error-free during demonstration (5)
- **Architecture & Network Design (12 marks):** [Group]
  - Network architecture with service communication patterns described using appropriate diagrams (3)
  - Microservices decomposition justified (3)
  - Event-driven architecture properly implemented and explained (3)
  - Alternatives discussed with justification (3)
- **Code Quality & Implementation (12 marks):** [Individual]
  - Code is executable and properly integrated (`docker compose up` works) (3)
  - Code quality: appropriate structure, readable, well-documented (4)
  - Implementation choices align with described architecture (3)
  - Individual contributions clearly identifiable in codebase (2)
- **Testing & Automation (4 marks):** [Individual]
  - Tests implemented and automated (2)
  - Test quality and coverage appropriate (2)
- **Scrum Process (4 marks):** [Group] Sprint artefacts visible and appropriate (2); empirical process adaptation evident (2).
- **Presentation Skills (4 marks):** [Individual] Language correct and appropriate (2); presentation meets basic criteria (volume, speed, engagement) (2).
- **Technical Understanding & Reflection (6 marks):** [Individual]
  - Questions answered correctly, demonstrating deep technical understanding (3)
  - Reflection on progress, challenges, and learning is open and realistic (3)

Total: 50 marks per demonstration (= 25% of final module grade)
Note on Individual vs. Group Marking: Items marked [Individual] contribute to your individual demo mark based on your personal work and understanding. Items marked [Group] assess the team's collective work. The final demonstration mark combines both group and individual aspects.
After each demo, you will receive feedback based on this marking scheme. You can incorporate this feedback for the next assessment.

5.2 Individual Reflective Report Marking

Each individual report is worth **25% of your final module grade** (50 marks). This marking scheme applies to both Assessment 1 (7 November) and Assessment 2 (12 December) reports.
Important: This marking scheme assesses your **written reflection** on your technical work. Code quality, functionality, and technical implementation are assessed during the **demonstration** (see Demonstration Marking).
The reports will be marked as follows:

- **Requirements & Individual Contribution (12 marks):**
  - Requirements implemented clearly described with appropriate detail (6)
  - Individual contribution explicitly stated and distinguished from team work (3)
  - Collaboration with team members documented (who helped, who you helped) (3)
- **Architecture & Design Decisions (12 marks):**
  - Architectural decisions explained with clear rationale (4)
  - Alternative approaches considered and discussed (4)
  - Consistency between described architecture, requirements, and code references (4)
- **Implementation Approach & Technical Challenges (10 marks):**
  - Code design and quality considerations discussed (design patterns, readability, maintainability) (5)
  - Technical challenges encountered and solutions explained (3)
  - Alignment between implementation choices and architectural decisions (2)
- **Testing & Quality Assurance Strategy (8 marks):**
  - Testing strategy and approach clearly described (4)
  - Specific test cases, scenarios, or testing techniques explained (4)
- **Development Process & Tool Usage (5 marks):**
  - Use of development tools (IDEs, build automation, debugging tools) described (2)
  - GenAI usage documented with concrete examples (prompts, how used, what learned) (3)
- **Critical Reflection & Learning (8 marks):**
  - Impediments and challenges identified and analysed (3)
  - Solutions attempted or proposed, with evaluation of effectiveness (3)
  - Personal learning outcomes and insights (2)
- **Report Presentation Quality (5 marks):**
  - Logical structure, coherence, and appropriate use of signposting to code (3)
  - Grammar, spelling, and academic writing style (2)

Total: 50 marks per report (= 25% of final module grade)
Note on Code References: Your report should reference specific code files, classes, or functions you implemented (e.g., "implemented the `ChunkingService` in `src/indexing/chunking.py:45-120`"). This helps markers connect your reflection to your actual work, but the code itself is evaluated during the demonstration.
If tasks are incomplete by assessment deadlines, submit the report anyway and explain what parts are incomplete, what problems caused the delay, and how you attempted to resolve them. Circumstances will be considered during marking.
The report is marked **individually** based on your personal contribution and reflection.

6 Submission

6.1 Submission Deadlines

- **Assessment 1:** 6 November, 15:00 (upload to Moodle) — Presentation: 7 November
- **Assessment 2:** 11 December, 15:00 (upload to Moodle) — Presentation: 12 December

6.2 What to Submit

Each student submits the following to Moodle by the deadline:

A. **Repository snapshot** (zipped, without build artefacts)

B. **Individual reflective report** (PDF, 2 pages maximum)

C. **Presentation slides** (PDF or PowerPoint)

What must be in the repository:
The repository must contain everything needed to run and understand your system:

- `README.md` with high-level architecture (Mermaid diagrams recommended), setup instructions, and how to run
- Complete source code with `docker-compose.yml`
- `/docs` folder with detailed documentation (see specific requirements for each assessment)
- Tests and CI configuration (Assessment 2)

Important notes:

- The state of your repository's **main branch** at the assessment deadline will be used for marking
- **Git commit timestamps** count as submission evidence—ensure all team members have regular, meaningful commits throughout the project
- Work continuously via GitHub throughout the project period

6.3 Technical Requirements

- **Runnable code:** Your system must be executable using `docker compose up`. Include all necessary build automation (Maven, Gradle, npm, etc.).
- **No build artefacts:** Never include compilation results (`target/`, `build/`, `node_modules/`, `.venv/`, etc.) in your repository—these are generated automatically.
- **Complete documentation:** Your `README.md` must contain clear setup and run instructions. Code is only considered runnable if it can be executed following your README.
- **Team integration:** If your component requires other team members' components, include everything in the repository and document individual parts clearly.

For further questions or discussion, please contact the module convenor: m.grimmer@lancaster.ac.uk