Lab 1

a. Read the following article: http://files.catwell.info/misc/mirror/2003-martin-fowler-who-needs-an-architect.pdf

b. Watch the following video: https://www.youtube.com/watch?v=DngAZyWMGR0

c. **Explain clearly why software architecture is important**

Software architecture is important because it defines the system's fundamental structure and key technical decisions early: components, responsibilities, interfaces, data flow, and quality tradeoffs.

It has a strong impact on non-functional requirements (performance, scalability, reliability, security, maintainability), team coordination, cost, delivery speed, and long-term evolvability.

If architecture is weak, projects usually fail later through high change cost, poor integration, and technical debt.

d. **Explain what the difference is between software architecture and software design**

Software architecture is the high-level blueprint of the system: major building blocks, integration style, deployment topology, technology constraints, and system-wide quality attributes.

Software design is lower-level and focuses on how each part is implemented: classes, methods, algorithms, data structures, and module internals.

Architecture answers "what are the main parts and how do they interact?" while *design* answers "how do we build each part correctly?"

e. **Explain what makes software architecture so difficult.**

Architecture is difficult because architects must make early decisions with incomplete information and many competing constraints.

They must balance tradeoffs (for example, performance vs maintainability, consistency vs availability, speed vs flexibility), align business goals with technical realities, and anticipate future change.

It is also socio-technical: success depends on people, teams, communication, governance, and organizational structure, not only technical diagrams.

f. **Explain clearly the main differences of software architecture in a traditional waterfall project and software architecture in an agile project.**

In a traditional waterfall project, architecture is mostly defined upfront, with heavy documentation and limited feedback until later phases. This can provide early structure but risks mismatch if requirements evolve.

In agile projects, architecture is more evolutionary: establish a strong baseline ("just enough architecture"), then refine continuously through iterations, spikes, testing, and frequent feedback.

So, waterfall architecture is typically plan-driven and fixed earlier; agile architecture is adaptive, incremental, and continuously validated.

g. Suppose you need to define the architecture for a large expensive system, and it is important that this system is future proof because this system will be used for at least 20 years. Explain how you can design a future proof system.

Use principles that maximize changeability and replaceability:

1. Define stable domain boundaries and clear module/service contracts (strong encapsulation, low coupling).
2. Use interface-first and contract versioning (backward compatibility strategy).
3. Prefer modular architecture (for example, layered + bounded contexts + event-driven integration where useful).
4. Separate business logic from frameworks and infrastructure to reduce lock-in.
5. Design for observability and operations from day one (logs, metrics, tracing, health checks).
6. Build for resilience and scaling (redundancy, graceful degradation, async processing where appropriate).
7. Automate quality gates (CI/CD, tests, architecture checks, security scanning).
8. Maintain architecture governance (ADRs, periodic architecture reviews, technical debt management).
9. Plan data evolution explicitly (schema migration/versioning strategy).
10. Keep technology choices conservative on core components and isolate parts likely to change.

h. List all the tasks you can think of that a software architect needs to do in a software development project.

1. Understand business goals, constraints, and quality attribute priorities.
2. Define architecture vision, principles, and target style/patterns.
3. Identify components/services and allocate responsibilities.
4. Define interfaces, integration patterns, and communication protocols.
5. Make technology/platform decisions with documented rationale (ADRs).
6. Design for quality attributes (performance, availability, security, scalability, maintainability).
7. Define data architecture, ownership, and consistency strategy.
8. Define deployment/runtime topology and environment strategy.
9. Assess risks and run tradeoff analysis/prototyping/spikes.
10. Collaborate with teams, coach developers, and align implementation with architecture.
11. Establish governance and standards (coding, APIs, observability, security).

12. Review architecture continuously and evolve it as requirements change.

i. For each of the following qualities, give at least 1 technique that you know to increase this quality:

1. **Performance**:
   - caching (application cache/CDN/query cache);
   - database indices.
2. **Availability**:
   - Use redundancy with failover (multiple instances across zones/regions + health checks);
   - horizontal scaling;
   - eventual consistency on data storage, prioritizing availability over consistency.
3. **Resilience**:
   - Dead letter queues to avoid data loss and failure
   - Rate limiter
   - Graceful degradation
   - Canary testing
   - Implement circuit breaker + retries with backoff and timeouts.
4. **Reusability**:
   - Microservices architecture with single responsibility and loose coupling
5. **Maintainability**:
   - Metrics and alarms;
   - CI/CD pipelines
   - enforce modular design with clear boundaries and strong automated tests (unit/integration).