ELABORATO FASE A

Bertozzi Pietro

E-mail istituzionale: pietro.bertozzi3@studio.unibo.it (mailto:pietro.bertozzi3@studio.unibo.it)
GitHub organizzation gruppo progetto finale sito web: https://ingegneria-sistemi-software-m.github.io (https://ingegneria-sistemi-software-m.github.io)
GitHub organizzation gruppo progetto finale: https://github.com/ingegneria-sistemi-software-m (https://github.com/ingegneria-sistemi-software-m)
GitHub repository personale: https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi (https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi)
Diario principale: https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi/tree/main/riflessioni (https://github.com/ingegneria-sistemi-software-m/ingegneria-sistemi-software-m-pietro-bertozzi/tree/main/riflessioni)

Quello che lo scrivente ha percepito sia la finalità dei contenuti discussi in questa prima fase, e quello che lo scrivente si aspetta sia lo scopo delle prossime fasi del corso

Il motto "Quello che non posso creare, non lo saprò mai capire" di Feynmann riflette pienamente la finalità di questa parte del percorso.
Il corso si distingue per il suo approccio pratico, interamente in laboratorio, dove seguire le lezioni non è mai un'attività puramente contemplativa.
L’obiettivo di abituarsi a un approccio pratico è già di per sé significativo, ma c'è di più.
I richiami all’ingegneria del software mostrano come l’attività in laboratorio non sia in antitesi con la teoria, ma piuttosto una sintesi di teoria e pratica.

La comprensione del problema, l'analisi e la progettazione, ma anche il testing, sono essenziali per costruire soluzioni funzionanti, e questo approccio completo permette di acquisire una visione più profonda dei contenuti trattati.
Per le prossime fasi del corso, mi aspetto di esplorare anche componenti hardware e credo che saremo chiamati a una maggiore autonomia operativa, affrontando sfide che richiederanno di integrare quanto appreso.
In generale, non ho le idee del tutto chiare su come si svilupperà il corso, ma in fondo mi va bene così.

I sistemi che lo scrivente è riuscito a realizzare e sperimentare, nell'ambito di quelli discussi nella prima fase del corso

Ho sperimentato con tutti i progetti forniti dal professore, eseguendoli per comprenderne il funzionamento.
Solo meno di metà di essi ho applicato modifiche per testare varianti e vedere l'effetto di piccoli cambiamenti.
Durante le prossime settimane, continuerò a sperimentare ulteriormente.

Le nuove abilità / competenze che lo scrivente pensa di avere appreso (o che sia possibile apprendere) dopo questa prima fase, sia in relazione alla parte 'pratica' del saper-fare (strumenti, tecnologie, etc.) sia con riferimento ad aspetti più 'concettuali' (metodologie, modelli, etc.)

Durante questa prima fase del corso, ho acquisito familiarità con diverse tecnologie come Spring Boot, MQTT, WebSocket, Docker e Gradle per la gestione delle dipendenze.
Dal punto di vista concettuale, molte delle idee trattate mi erano già familiari, grazie alla tesi in Ingegneria del Software T durante la triennale.
Inoltre, il corso di linguaggi e modelli computazionali mi ha aiutato a sviluppare una maggiore sensibilità verso concetti teorici avanzati, permettendomi di apprezzare appieno le riflessioni del professore.
Più di tutti mi ha interessato la metodologia Srum di sviluppo incrementale che ritengo illuminante e a cui, prima di questo corso, non ho mai dedicato il tempo che merita.

Personalmente mi ritengo più portato naturalmente per approfondire la teoria, piuttosto che essere efficiente nella pratica, ma spero di migliorare in questo nel corso dei prossimi mesi.

Quello che lo scrivente pensa sia il motivo per cui, come primo esempio di sistema software da sviluppare, è stato proposto, come caso di studio, il gioco della vita di Conway

Il gioco di Conway, Game of Life, è stato scelto come esempio principalmente per la sua semplicità e per la ricchezza di dinamiche che riesce a generare nonostante le sue regole intuitive.
Le regole di base del gioco sono facili da comprendere, intuitive, ma nonostante ciò, piccole modifiche nell'input, possono avere un impatto enorme sull'output e sulle evoluzioni successive.
Questo rende difficile testare il corretto funzinamento dell'applicazione "ad occhio" e incentiva ad affinare le capacità di testing automatizzato.
Inoltre, il concetto di turno o epoca, che definisce l'evoluzione del gioco, si presta bene a esperimenti con comunicazione asincrona; ogni "epoca" può essere vista come un momento in cui si verificano modifiche indipendenti e non necessariamente sincronizzate tra loro.
Un altro aspetto interessante di Conway è la relazione tra la griglia e le celle, che richiama un caso generale delle differenze tra composizione e aggregazione.
Questo approccio consente di riflettere sulle responsabilità all'interno di un sistema e prepara il terreno per l'ingresso nel mondo dei microservizi, dove le responsabilità sono chiaramente separate.
Un esperimento che potremmo svolgere, prevede che ogni host gestisca (visualizzi) solo una cella sul proprio monitor, esplorando ulteriormente la distribuzione delle responsabilità nel sistema.

Le motivazioni che lo studente ritiene siano alla base della scelta di utilizzare Java e framework (come SpringBoot) legati a Java per lo sviluppo del caso di studio e se ritiente queste scelte limitative oppure obsolete, indicando alternative che ritiene più moderne o più adatte

La scelta di utilizzare Java e framework come Spring Boot per lo sviluppo del caso di studio è stata motivata principalmente dal fatto che in ambito universitario siamo già abituati a lavorare con questo linguaggio e con tali tecnologie.
Optare per un cambio radicale di tecnologia avrebbe comportato il rischio di distrarre l'attenzione dai punti chiave del corso, che si concentrano anche su concetti architetturali e progettuali.
Non sarebbe stato ragionevole perdere tempo in quel modo, anche perchè Java con Spring Boot rimane una scelta solida per un caso di studio come il nostro, in quanto fornisce una struttura stabile, ben supportata e ampiamente utilizzata nel settore.
Sarebbe comunque stato possibile fare una scelta diversa. Da un breve ricerca online sembra che gli altri cantidati acceditati sarebbero potuti essere:

1. Python con il framework Django, che offre un'ottima velocità di sviluppo
2. Node.js con Express, che si adatta perfettamente a un modello asincrono, particolarmente adatto per applicazioni web moderne e comunicazione in tempo reale.

Gli aspetti di Ingegneria del software (key points) che il caso di studio ha permesso di richiamare o di introdurre nelle varie fasi ‘evolutive’ dei prototipi sviluppati durante la fase 1

Seguen un elenco dei punti chiave, spiegati brevemente:

1. Domain Driven Software Development: è essenziale identificare le entità che appartengono al dominio del sistema per poter creare un modello che ne rappresenti accuratamente gli aspetti fondamentali.
2. Single Responsibility Principle: applicato prima a funzioni e classi ma successivamente anche alle prime bozze di microvervizi.
3. Inversione delle Dipendenze: permette una soluzione tecnology indipendent. "IODev" e "WsIODev" sono esempi pratici.
4. Principio di Iron Man: utilizzando un framework no abbiamo dovuto cambiare l'applicazione preesistente, un po' come l'armatura che viene applicata al supereroe senza alterarlo.
5. Tecnologie Asincrone: L'uso di tecnologie asincrone, come WebSocket ed MQTT, ha permesso di ribaltare l'iniziativa dando il timone al server, come è naturale che sia.
6. M2M Interactions: all'interazione Machine-to-Machine segue naturlmente il concetto di microservizio, e suggerisce considerazioni interassanti riguardo al testing automatico.
7. Linguaggio come Astrazione: È stato evidente come il linguaggio di programmazione, e più in generale l'astrazione delle comunicazioni, ha influenzato il nostro modo di pensare e progettare soluzioni, spingendo verso una comprensione più profonda del dominio (In seguito maggiori dettagli).

Se lo scrivente ritiene oppure no, che il sistema software sviluppato al termine della prima fase sia un sistema distribuito basato su microservizi

Il sistema che stiamo sviluppando presenta già molte caratteristiche di un'architettura basata su microservizi, come la suddivisione in servizi distinti che comunicano tramite un broker (MQTT).
Questo approccio permette una distribuzione dei compiti, seppur iniziale, tra i vari componenti, che sono separati e si scambiano informazioni utilizzando i topic.
Tuttavia, ci sono alcune tematiche, come la scalabilità e la tolleranza ai guasti, che non sono state ancora adeguatamente affrontate.

Questi sono aspetti chiave nei sistemi distribuiti, come ho appreso durante il corso di Sistemi Operativi M.
La scalabilità permette ai servizi di crescere in modo indipendente, mentre la tolleranza ai guasti garantisce che, se uno dei microservizi fallisce, gli altri possano continuare a funzionare senza compromettere l'intero sistema.
Sebbene ci siano le basi per un'architettura a microservizi, l'approfondimento di questi aspetti è essenziale per parlare di un vero sistema distribuito, in grado di gestire in modo autonomo i propri errori e adattarsi alla crescita.

Il ruolo che lo scrivente ritiene abbia lo sviluppo di librerie ‘custom’ durante la realizzazione di un sistema software e perchè, per il caso di studio, sono state realizzate classi (relative al protocollo MQTT) inserite nella libreria unibo.basicomm23-1.0.jar

Lo sviluppo di librerie "custom" è fondamentale per colmare l'abstraction gap, ovvero la distanza tra le necessità del dominio e gli strumenti tecnologici disponibili.
Quando gli strumenti esistenti non soddisfano le esigenze specifiche, le librerie personalizzate offrono interfacce più adeguate, migliorando leggibilità e manutenibilità del sistema.
Inoltre, ridurre l'abstraction gap implica anche una technology independence, che consente di adattare facilmente il sistema a nuove esigenze senza essere vincolati a soluzioni troppo specifiche.
Credo che il professore abbia scelto MQTT proprio perché, essendo un protocollo a basso livello, è necessario astrarre ulteriormente per poter sviluppare con un linguaggio conforme al nostro modo di esseri umani, di esprimere la comunicazione
L’obiettivo è farci confrontare con le difficoltà che derivano dall’utilizzo di strumenti di basso livello, enfatizzando la problematica del linguaggio.

Opzionalmente: Il ruolo che lo scrivente ritiene abbia il concetto di linguaggio (di programmazione, ma non solo) più volte evocato dal docente come esempio di un salto evolutivo fondamentale dell’informatica rispetto al solo sviluppo di librerie

Il parallelismo tra linguaggi naturali e linguaggi di programmazione è perfettamente valido.
Un linguaggio, che sia di programmazione o naturale, non si limita a fornire una forma per esprimere idee, ma gioca un ruolo fondamentale nel processo stesso di creazione e formulazione delle idee.
Persone che conoscono linguaggi naturali diversi tendono a pensare in modo diverso proprio a causa della struttura e dei vocaboli a loro disposizione.
Imparare nuovi vocaboli non significa solo migliorare l’espressione, ma anche affinare la granularità con cui siamo in grado di cogliere le sfumature, e quindi di pensare criticamente.
Svulippare astrazioni, come possono essere le librerie, è quindi una procedura estremamente sofisticata: normalmente il linguaggio è il modo in cui pensiamo alle idee, ma per chiedersi in quale linguaggio è bene esprimere un'idee è necessaria un profonda familiarità con l'idea stessa. Questo evidenzia quanto sia importante, ma anche complesso, sviluppare un linguaggio che soddisfi le esigenze specifiche di un determinato contesto.